Artificial Intelligence - Model Implementation Report with Pytorch

Jiyeong Oh, Dpt.DataScience

Hanyang University

Pytoch is a Python-based open-source machine learning library, well known for its usefulness in implementing machine learning and deep learning models. This report covers the process of designing multiple models and tuning hyperparameters, starting with loading datasets using the library, to broaden the understanding of the model itself and the factors that affect its performance.

Load MNIST Data – Data Split

Database used in this report is MNIST, which includes handwritten digit images with 784 attributes (28*28) and 10 labels. To apply the training and testing process with the given data, function "data_split" which splits the data into train/validation/test set with the same digit proportion, was created.

```
def data_split(attr, tar)
```

The function performs split taking two arguments: attribute and target(label).

First, with the target data, a dictionary with answer label as key and its indices as values, are being created. Also, in order to prevent data from not being mixed in the process of training/test split, the values under each key are shuffled here.

```
y_dict = dict(zip([i for i in tar.unique()], [list(tar[tar.values==i].index) for i in tar.unique()]))
y_dict = dict(zip([i for i in tar.unique()], [list(np.random.choice(y_dict[i], len(y_dict[i]), replace = Fa
lse)) for i in y_dict.keys()])) # Shuffling
```

With the shuffled dictionary, dataset indices are trimmed at a consistent rate for each key: 70% for train set, 15% for validation set, and 15% for test set.

```
train_tmp = [y_dict[i][:int(round(len(y_dict[i])*0.7, 1))] for i in sorted(y_dict.keys())]
val_tmp = [y_dict[i][int(round(len(y_dict[i])*0.7,-1)):int(round(len(y_dict[i])*0.85,-
1))] for i in sorted(y_dict.keys())]
test_tmp = [y_dict[i][int(round(len(y_dict[i])*0.85,-1)):] for i in sorted(y_dict.keys())]
```

With indices for train/validation/test saved in different list, value indexing was applied to split the whole dataset into three. Finally, converting the fetched data into Pytorch Tensor type, and Normalization for X was applied here also.

```
X_train, y_train =
torch.tensor(attr.loc[train_idx].to_numpy())/255, torch.tensor(tar.loc[train_idx].to_numpy())
...(omitted)
return X_train, y_train, X_val, y_val, X_test, y_test
```

Binary Classification via soft-margin SVM

(a) Classifier Building & Train/Val/Test Code Description

First, Class "SoftMargin_SVM" with network from input to output layer, was built. Then, using the created class as model, function "softMarginSvm" which trains data with model, calculates validation accuracy for each epoch, then spits out the final test accuracy was built.

```
def softMarginSvm
```
As SVM is a binary classifier between 1 and -1, the function converts the input dataset label (2, 3) into -1 and 1.

```
if y_two_three.iloc[i] == 2:

        y_two_three.iloc[i] = -1

else:

      y_two_three.iloc[i] = 1
```
Then the function trains the model, adding up cost for each batch. Here, hinge loss was defined and used for calculating the cost.

```
def hinge_loss(y_actual, y_pred):

    return torch.clamp(1-y_pred*y_actual, min=0)
```
Validation accuracy was also calculated with each epoch. However, parameters are not being updated in validation accuracy calculating process.

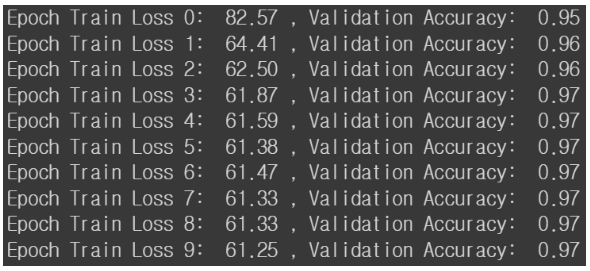
After all the epoch iterations are over, the final test accuracy calculation with test set, is being applied.

(b) Training for 10 epochs with batch size 64

Several training trials with different hyperparameters were implemented to improving the final test accuracy; The default trial was training for 10 epochs with batch size 64.

```
epochs_num = 10

batch_size = 64
```
Training and checking validation accuracy along epochs, it was confirmed that the train loss decreased every epoch, and the validation accuracy starts from 0.95 and ends with 0.97.



```
Epoch Train Loss 0:  82.57 , Validation Accuracy:  0.95
Epoch Train Loss 1:  64.41 , Validation Accuracy:  0.96
Epoch Train Loss 2:  62.50 , Validation Accuracy:  0.96
Epoch Train Loss 3:  61.87 , Validation Accuracy:  0.97
Epoch Train Loss 4:  61.59 , Validation Accuracy:  0.97
Epoch Train Loss 5:  61.38 , Validation Accuracy:  0.97
Epoch Train Loss 6:  61.47 , Validation Accuracy:  0.97
Epoch Train Loss 7:  61.33 , Validation Accuracy:  0.97
Epoch Train Loss 8:  61.33 , Validation Accuracy:  0.97
Epoch Train Loss 9:  61.25 , Validation Accuracy:  0.97
```

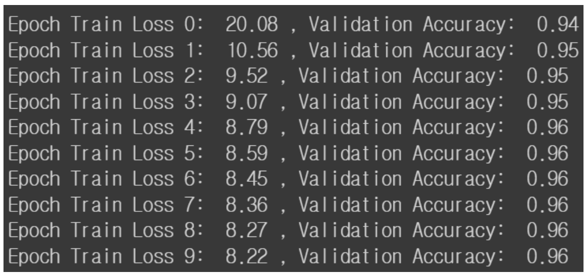
(c) Iteration Number Tuning for Generalization

As the number of iterations affects the generalization of model, batch_size was tuned with both larger size and smaller size to change the iteration number.

The larger batch size, showed worse generalization. As the number of iterations decreases because of the enlarged batch size, the validation set showed lower accuracy. For example, batch_size with 256 showed the validation accuracy of below:

```
epochs_num = 10

batch_size = 256
```

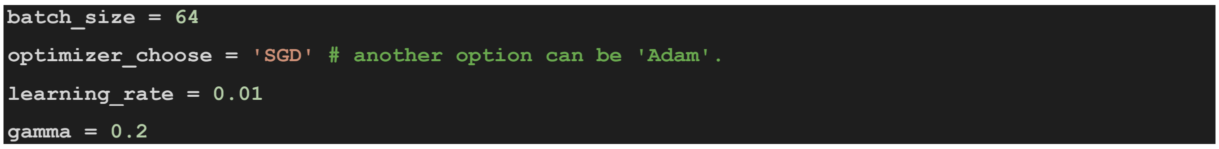


```
Epoch Train Loss 0:  20.08 , Validation Accuracy:  0.94
Epoch Train Loss 1:  10.56 , Validation Accuracy:  0.95
Epoch Train Loss 2:  9.52 , Validation Accuracy:  0.95
Epoch Train Loss 3:  9.07 , Validation Accuracy:  0.95
Epoch Train Loss 4:  8.79 , Validation Accuracy:  0.96
Epoch Train Loss 5:  8.59 , Validation Accuracy:  0.96
Epoch Train Loss 6:  8.45 , Validation Accuracy:  0.96
Epoch Train Loss 7:  8.36 , Validation Accuracy:  0.96
Epoch Train Loss 8:  8.27 , Validation Accuracy:  0.96
Epoch Train Loss 9:  8.22 , Validation Accuracy:  0.96
```

Comparing the accuracy with the one with batch size 64, it was confirmed that the validation accuracy has decreased, which indicates overfitting. Similarly, when batch size was even more enlarged, the Acc decreased more.

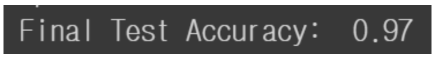
On the other hand, increased number of iterations caused by smaller batch size showed improved generalization to some extent. With the batch size 16 and 8, it was confirmed that the validation accuracy at initial few epochs rose a little. However, since the validation accuracy was no longer improved in a large unit compared to the large reduction in the batch size, it was determined that the existing batch size of 64 was appropriate.

(d) Final test Accuracy



```
batch_size = 64
optimizer_choose = 'SGD' # another option can be 'Adam'.
learning_rate = 0.01
gamma = 0.2
```

With the hyperparameters set above, the final test accuracy ended up with 0.97.

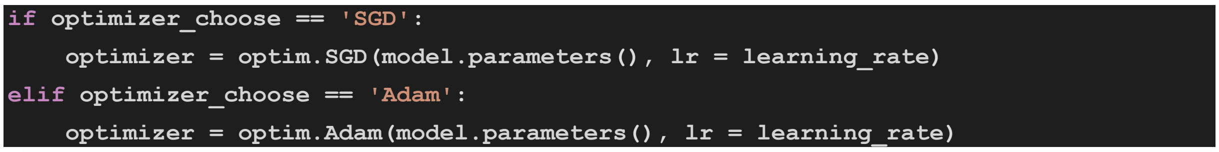


```
Final Test Accuracy:  0.97
```

(e, f) Hyperparameter Tuning

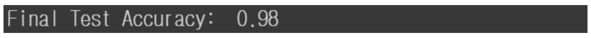
Beyond 0.97, the model was built to allow multiple hyperparameters to be changed to increase the test accuracy, and several combinations was tried to find out which parameter values produce high test accuracy.

Optimizer – Adam & SGD



```
if optimizer_choose == 'SGD':
    optimizer = optim.SGD(model.parameters(), lr = learning_rate)
elif optimizer_choose == 'Adam':
    optimizer = optim.Adam(model.parameters(), lr = learning_rate)
```

The model is designed so that the type of optimizer can be selected in the model training process. In the previous runs, the optimizer selected was SGD. Switching the optimizer to Adam, which does not requires much tuning as the SGD requires, showed higher accuracy in test set.



```
Final Test Accuracy:  0.98
```

Learning Rate

Learning rate for optimizer was also tuned with various degree. The default learning rate was set to 0.01, and smaller learning rate and larger rate were applied to see whether it improves the performance of model or not.

Smaller learning rate (0.001) did not improve the performance of the model.

```
Final Test Accuracy:  0.98
```

Mush smaller learning rate like 0.0001, started to show worse performance score, which implies that too small learning rate can be an obstacle of good performance on test accuracy.

Too large learning rate(0.1) showed worse performance below:

```
Final Test Accuracy:  0.95
```

Final test accuracy decreased comparing with other combinations of hyperparameters, so it was concluded that learning rate 0.1 is also too large for this model.

Gamma

Gamma, which controls how much slack is allowed in soft-margin SVM, was also tuned.

```
loss = gamma*torch.norm(list(model.parameters())[0])/2 + torch.mean(hinge_loss(answ
er, prediction))
```

While default value was 0.2, too large amount of gamma showed decreased test accuracy.

```
Final Test Accuracy:  0.95
```

On the other hand, decreased amount of gamma showed improved performance:

```
Epoch Train Loss 0:  18.22 , Validation Accuracy:  0.97
Epoch Train Loss 1:  12.30 , Validation Accuracy:  0.97
Epoch Train Loss 2:  11.41 , Validation Accuracy:  0.97
Epoch Train Loss 3:  10.75 , Validation Accuracy:  0.97
Epoch Train Loss 4:  10.91 , Validation Accuracy:  0.97
Epoch Train Loss 5:  10.83 , Validation Accuracy:  0.97
Epoch Train Loss 6:  10.38 , Validation Accuracy:  0.97
Epoch Train Loss 7:  10.25 , Validation Accuracy:  0.97
Epoch Train Loss 8:  10.70 , Validation Accuracy:  0.97
Epoch Train Loss 9:  10.12 , Validation Accuracy:  0.97
Final Test Accuracy:  0.98
```

Concluding that there are absolutely no good-sized hyperparameters and proper tuning is needed to improve model performance, the best combination of performance is determined as follows.

```
epochs_num = 10
batch_size = 64
optimizer_choose = 'Adam'
learning_rate = 0.01
gamma = 0.001
```

Binary Classification via MLP

(a) Classifier Building & Train/Val/Test Code Description

First, model class "MLP_binary" was initialized. Two hidden layers with 256 units were built.

```
self.module1=nn.Linear(input_dim,256,bias=True
self.module2=nn.Linear(256,256,bias=True
self.module3=nn.Linear(256,output_dim,bias=True
```

Also sigmoid function as activation function was applied at the end of forward process, as long as the goal of this classifier is to identify binary digits.

```
y_pred = torch.sigmoid(x)
```

Based on this model class, function “binaryMLP” was built and was ran to be train and predict.

Similar with SVD, the data was converted into 0 and 1 as activation function is sigmoid.

Loss function, was set to cross entropy to make the model follow the distribution of correct answer.

```
criterion = nn.BCELoss()
...
loss = criterion(prediction, answer)
```

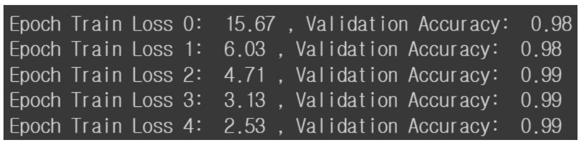
Validation accuracy along epoch, and the final test accuracy calculation with test set, is being applied, as usual.

(b) Training for 5 epochs with batch size 64

The first training/validation trial was for 5 epochs with batch size 64.

```
epochs_num = 5
batch_size = 64
optimizer_choose = 'Adam'
learning_rate = 0.01
```

Training and checking validation accuracy along epochs, the validation accuracy starts from 0.98 and ends with 0.99.

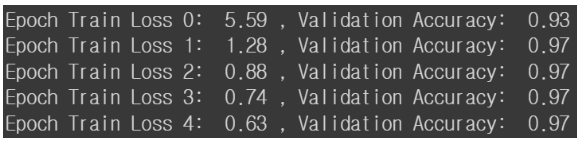


```
Epoch Train Loss 0:  15.67 , Validation Accuracy:  0.98
Epoch Train Loss 1:  6.03 , Validation Accuracy:  0.98
Epoch Train Loss 2:  4.71 , Validation Accuracy:  0.99
Epoch Train Loss 3:  3.13 , Validation Accuracy:  0.99
Epoch Train Loss 4:  2.53 , Validation Accuracy:  0.99
```

(c) Iteration Number Tuning for Generalization

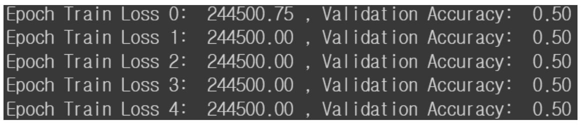
Tuning the number of iterations was also done. Both extremely large or small size of batch leaded the worse performance score.

Batch_size 1024



```
Epoch Train Loss 0:  5.59 , Validation Accuracy:  0.93
Epoch Train Loss 1:  1.28 , Validation Accuracy:  0.97
Epoch Train Loss 2:  0.88 , Validation Accuracy:  0.97
Epoch Train Loss 3:  0.74 , Validation Accuracy:  0.97
Epoch Train Loss 4:  0.63 , Validation Accuracy:  0.97
```

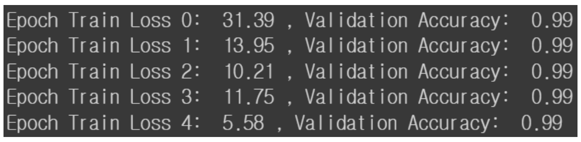
Batch_size 2



```
Epoch Train Loss 0:  244500.75 , Validation Accuracy:  0.50
Epoch Train Loss 1:  244500.00 , Validation Accuracy:  0.50
Epoch Train Loss 2:  244500.00 , Validation Accuracy:  0.50
Epoch Train Loss 3:  244500.00 , Validation Accuracy:  0.50
Epoch Train Loss 4:  244500.00 , Validation Accuracy:  0.50
```

Considering the speed of iterations and the degree of generalization, it was judged that the batch size of about 32 was the most appropriate.

Batch_size 32



```
Epoch Train Loss 0:  31.39 , Validation Accuracy:  0.99
Epoch Train Loss 1:  13.95 , Validation Accuracy:  0.99
Epoch Train Loss 2:  10.21 , Validation Accuracy:  0.99
Epoch Train Loss 3:  11.75 , Validation Accuracy:  0.99
Epoch Train Loss 4:  5.58 , Validation Accuracy:  0.99
```

(d) Final test Accuracy

```
epochs_num = 5
batch_size = 32
optimizer_choose = 'Adam'
learning_rate = 0.01
```

With the hyperparameters set above, the final test accuracy ended up with 0.99.

Multiclass Classification via MLP

(a) Classifier Building & Train/Val/Test Code Description

For multiclass MLP, exactly same model class as the one for binary class was use. Difference is that sigmoid function was not used, and softmax was used as activation function. As cross entropy in Pytorch applies softmax instead, the softmax code was not used in model class defining process.

The other process such as training, validation, testing is also almost same with binary model. The only difference is that output dimension is 10, not 2.

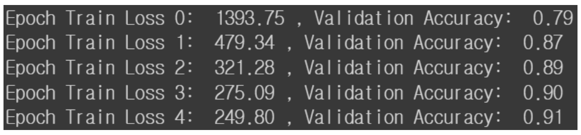
```
model = MLP_multi(X_train.shape[1], 10)
```

Therefore, the loss function was changed into cross entropy from binary cross entropy, which was used in previous problem.

```
criterion = nn.CrossEntropyLoss()
```

(b) Training for 5 epochs with batch size 64

Training and checking validation accuracy along 10 epochs with batch size of 64, the validation accuracy started from 0.79, kept increased and ended with 0.91.



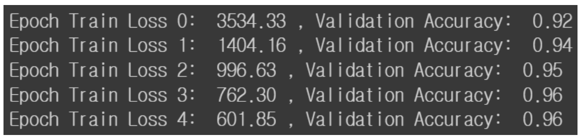
```
Epoch Train Loss 0:  1393.75 , Validation Accuracy:  0.79
Epoch Train Loss 1:  479.34 , Validation Accuracy:  0.87
Epoch Train Loss 2:  321.28 , Validation Accuracy:  0.89
Epoch Train Loss 3:  275.09 , Validation Accuracy:  0.90
Epoch Train Loss 4:  249.80 , Validation Accuracy:  0.91
```

(c) Iteration Number Tuning for Generalization

Starting with the batch size of 64, the iteration number has been tuned to find the optimal number of iterations for good generalization. Enlarged size of batch size (shorted number of iterations) showed worse generalization; Batch size 128 never showed over 0.89 validation accuracy in every epoch.

On the other hand, more iterations caused by smaller batch size showed improved generalization. Validation accuracy with batch size 32 ended up with 0.93, and that with batch size 16 even ended up with 0.95, which is a lot of improvement comparing with the initial trial.

Considering the time consumption, batch size 8 was selected as iterations criterion for good generalization. Below is its validation accuracy for each epoch.

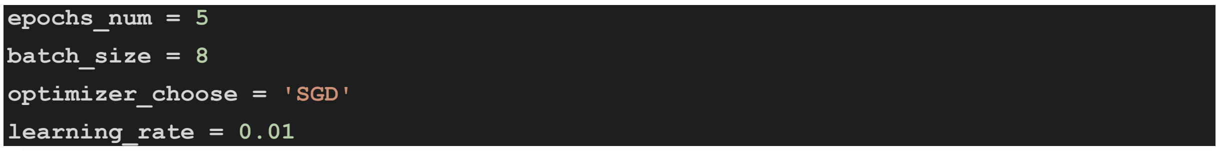


```
Epoch Train Loss 0:  3534.33 , Validation Accuracy:  0.92
Epoch Train Loss 1:  1404.16 , Validation Accuracy:  0.94
Epoch Train Loss 2:  996.63 , Validation Accuracy:  0.95
Epoch Train Loss 3:  762.30 , Validation Accuracy:  0.96
Epoch Train Loss 4:  601.85 , Validation Accuracy:  0.96
```

(d) Final test Accuracy

Therefore, with the hyperparameters set below, the final test accuracy ended up with 0.96.

(e, f) Hyperparameter Tuning



```
epochs_num = 5
batch_size = 8
optimizer_choose = 'SGD'
learning_rate = 0.01
```

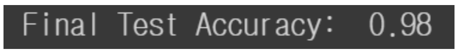
In addition to changing the number of iterations, determining hyperparameter combination which improves the model performance by adjusting the optimizer and learning rate, was applied. The default combination was model with batch size 8, with SGD optimizer and learning rate of 0.01, and the final test accuracy was about 0.96.

Optimizer – Adam & SGD

Switching the optimizer to Adam did not show higher accuracy in both validation set and test set; It showed 0.93 of accuracy in test set. However, with the enlarged batch size of 64, Adam showed a test accuracy comparable to 0.96 which was the best result when using SGD as an optimizer.

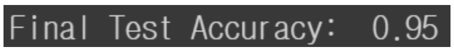
Learning Rate

Tuned learning rate improved the performance score even better. Adam optimizer with the learning rate of 0.001, showed much improved performance of 0.98, which is even better than SGD.



```
Final Test Accuracy:  0.98
```

On the other hand, too small amount of learning rate (0.0001) leaded the model not to be trained well. The test accuracy ended up with 0.95.



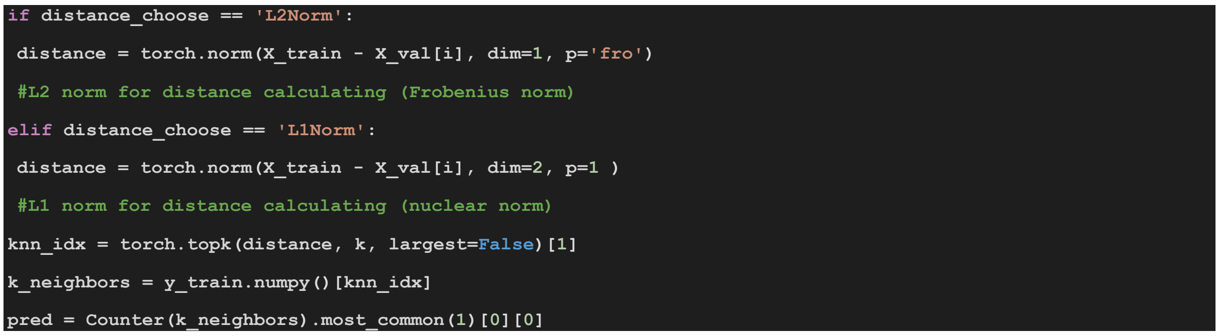
```
Final Test Accuracy:  0.95
```

This suggests that finding and tuning an appropriate learning rate as a hyperparameter is necessary for model learning.

Multiclass Classification via k-NN

(a) Classifier Building & Train/Val/Test Code Description

As k-NN does not require any training process as long as the dataset itself is the training, validation accuracy checking and final testing process was applied in the code. The model was simply constructed with its two hyperparameters: 'k' (the number of neighbors) and 'distance_choose' (method of calculating distance).



```
if distance_choose == 'L2Norm':
 distance = torch.norm(X_train - X_val[i], dim=1, p='fro')
 #L2 norm for distance calculating (Frobenius norm)
elif distance_choose == 'L1Norm':
 distance = torch.norm(X_train - X_val[i], dim=2, p=1 )
 #L1 norm for distance calculating (nuclear norm)
knn_idx = torch.topk(distance, k, largest=False)[1]
k_neighbors = y_train.numpy()[knn_idx]
pred = Counter(k_neighbors).most_common(1)[0][0]
```

(b) Final test Accuracy

With the hyperparameters set below, the final test accuracy ended up with 0.96.

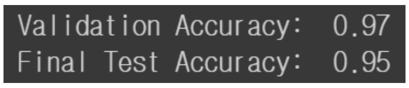
```
k = 5
distance_choose = 'L2Norm'
```

(c, d) Hyperparameter Tuning

Tuning for two hyperparameters were implemented here: measurement type of distance and the number of neighbors.

Distance Type: L2 norm & L1 norm

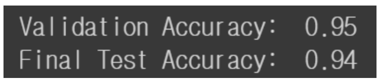
The default distance measuring method was set to L2 norm, and its validation accuracy and test accuracy was 0.97 and 0.96, respectively. Manipulating the measure to L1 norm so that the distance is not calculated by squaring up (L2 norm), final test accuracy has decreased by 0.95.



```
Validation Accuracy:  0.97
Final Test Accuracy:  0.95
```

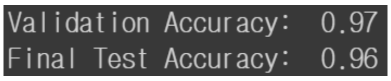
K (Number of Neighbors)

The number of k has been tuned also. At first, k=50 was tried to check whether the model is underfitting. Running the model, it was confirmed that both validation set accuracy and test accuracy had decreased by 0.95 and 0.94, respectively.



```
Validation Accuracy:  0.95
Final Test Accuracy:  0.94
```

Another trial with k=10 was also applied. As value 10 did not make significant different on classification with k=5, the performance score was similar with the very initial combination of hyperparameters (k=5, L2 norm)



```
Validation Accuracy:  0.97
Final Test Accuracy:  0.96
```