10_With_Hyperparametere_ANN_on_GPU_fashion_mnist_PyTorch

July 14, 2025

1 ANN with PyTorch - Fashion MNIST

2 Import libraries

[32]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from sklearn.model_selection import train_test_split
```

[30]:
```
# Set random seed for reproducibility
torch.manual_seed(42)
```

[30]:
```
<torch._C.Generator at 0x7c096adf0f30>
```

[31]:
```
# Check for GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Using device: cuda
```

3 import dataset

[7]:
```
!unzip /content/fashion_mnist_dataset.zip
```

```
Archive:  /content/fashion_mnist_dataset.zip
  inflating: fashion-mnist_test.csv
  inflating: fashion-mnist_train.csv
  inflating: t10k-images-idx3-ubyte
  inflating: t10k-labels-idx1-ubyte
  inflating: train-images-idx3-ubyte
  inflating: train-labels-idx1-ubyte
```

[33]:
```
# import dataset
df = pd.read_csv('/content/fashion-mnist_train.csv')
df.head()
```

[33]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	\
0	2	0	0	0	0	0	0	0	0	
1	9	0	0	0	0	0	0	0	0	
2	6	0	0	0	0	0	0	0	5	
3	0	0	0	0	1	2	0	0	0	
4	3	0	0	0	0	0	0	0	0	

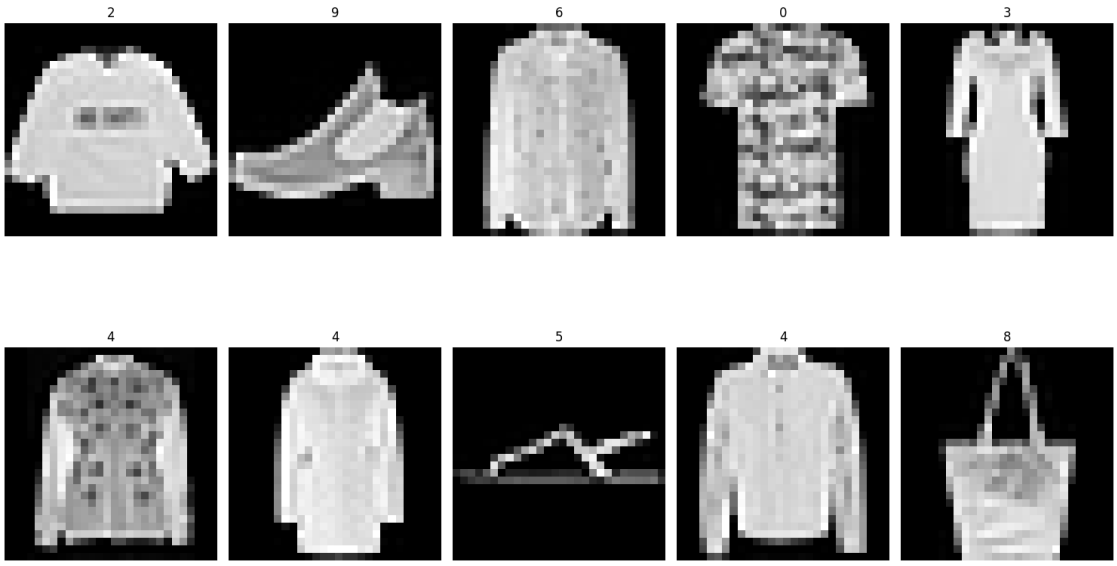
	pixel9	…	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	\
0	0	…	0	0	0	0	0	0	
1	0	…	0	0	0	0	0	0	
2	0	…	0	0	0	30	43	0	
3	0	…	3	0	0	0	0	1	
4	0	…	0	0	0	0	0	0	

	pixel781	pixel782	pixel783	pixel784
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

[5 rows x 785 columns]

[34]:
```
# let's plot first 10 images with 5x5 grid
fig, ax = plt.subplots(2, 5, figsize=(15, 10))
for i in range(10):
    ax[i//5, i%5].imshow(df.iloc[i, 1:].values.reshape(28, 28), cmap='gray')
    ax[i//5, i%5].axis('off')
    ax[i//5, i%5].set_title(df.iloc[i, 0])

plt.tight_layout()
plt.show()
```



3.1 Split the data

[35]:
```
# split the data into features and labels
X = df.iloc[:, 1:].values
y = df.iloc[:, 0].values
X.shape, y.shape
```

[35]:
```
((60000, 784), (60000,))
```

[36]:
```
# split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[36]:
```
((48000, 784), (12000, 784), (48000,), (12000,))
```

3.2 Scale the data

[37]:
```
# scaling the data
X_train = X_train / 255.0
X_test = X_test / 255.0
```

4 create dataloader

[38]:
```
# create CustomDataset class
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
```

```
        self.y = torch.tensor(y, dtype=torch.long)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, index):
        x = self.X[index]
        y = self.y[index]
        return x, y
```

[39]:
```
# create dataloader
train_dataset = CustomDataset(X_train, y_train)
test_dataset = CustomDataset(X_test, y_test)
```

5 Model building

[47]:
```
class MyNN(nn.Module):

    def __init__(self, input_dim, output_dim, num_hidden_layers,␣
 ↪neurons_per_layer, dropout_rate):
        super().__init__()
        layers = []
        for i in range(num_hidden_layers):

            layers.append(nn.Linear(input_dim, neurons_per_layer))
            layers.append(nn.BatchNorm1d(neurons_per_layer))
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(dropout_rate))
            input_dim = neurons_per_layer

        layers.append(nn.Linear(neurons_per_layer, output_dim))

        self.model = nn.Sequential(*layers)

    def forward(self, x):
        x = self.model(x)
        return x
```

6 Objective function

[51]:
```
# Objective function
def objective(trial):
  # next hyperparameter values from the search space
  num_hidden_layers = trial.suggest_int('num_hidden_layers', 1, 5)
  neurons_per_layer = trial.suggest_int('neurons_per_layer', 8, 128, step=8)
```

```
 epochs = trial.suggest_int('epochs', 10, 100, step=5)
 lr = trial.suggest_float('lr', 1e-5, 1e-1, log=True)
 dropout_rate = trial.suggest_float('dropout_rate', 0.1, 0.5, step=0.1)
 batch_size = trial.suggest_categorical('batch_size', [32, 64, 128])
 optimizer_name = trial.suggest_categorical('optimizer', ['Adam', 'SGD',␣
↪'RMSprop'])
 weight_decay = trial.suggest_float('weight_decay', 1e-5, 1e-3, log=True)

 # data loader
 train_loader = torch.utils.data.DataLoader(train_dataset,␣
↪batch_size=batch_size, shuffle=True, pin_memory=True)
 test_loader  = torch.utils.data.DataLoader(test_dataset,␣
↪batch_size=batch_size, shuffle=False, pin_memory=True)

 # model init
 input_dim = X_train.shape[1]
 output_dim = 10
 model = MyNN(input_dim, output_dim,num_hidden_layers, neurons_per_layer,␣
↪dropout_rate)
 model.to(device)

 # optimizer selection
 criterion = nn.CrossEntropyLoss()

 if optimizer_name == 'Adam':
   optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)

 elif optimizer_name == 'SGD':
   optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9,␣
↪weight_decay=weight_decay)

 elif optimizer_name == 'RMSprop':
   optimizer = optim.RMSprop(model.parameters(), lr=lr,␣
↪weight_decay=weight_decay)

 # training loop
 for epoch in range(epochs):

     for batch_features, batch_labels in train_loader:
         # move data to gpu
         batch_features = batch_features.to(device)
         batch_labels = batch_labels.to(device)

         # forward pass
         outputs = model(batch_features)
```

```
            # calculate loss
            loss = criterion(outputs, batch_labels)

            # zero gradients
            optimizer.zero_grad()

            # backward pass
            loss.backward()

            # update weights
            optimizer.step()

    # evaluation
    model.eval()

    with torch.no_grad():
      correct = 0
      total = 0
      for images, labels in test_loader:
          images = images.to(device)
          labels = labels.to(device)
          outputs = model(images)
          _, predicted = torch.max(outputs.data, 1)
          total += labels.size(0)
          correct += (predicted == labels).sum().item()

      accuracy = correct / total

    return accuracy
```

[52]:
```
# lets create study
import optuna
study = optuna.create_study(direction='maximize')
```

```
[I 2025-07-08 08:33:25,359] A new study created in memory with name: no-
name-d67083ee-7fe1-408f-945e-21864b5f1b97
```

[53]:
```
study.optimize(objective, n_trials=20)
```

```
[I 2025-07-08 08:36:05,974] Trial 0 finished with value: 0.8814166666666666 and
parameters: {'num_hidden_layers': 3, 'neurons_per_layer': 72, 'epochs': 70,
'lr': 0.0016976313223769004, 'dropout_rate': 0.4, 'batch_size': 64, 'optimizer':
'Adam', 'weight_decay': 5.4260526198393654e-05}. Best is trial 0 with value:
0.8814166666666666.
[I 2025-07-08 08:37:10,028] Trial 1 finished with value: 0.87925 and parameters:
{'num_hidden_layers': 5, 'neurons_per_layer': 64, 'epochs': 40, 'lr':
0.001687930783405996, 'dropout_rate': 0.2, 'batch_size': 128, 'optimizer':
'Adam', 'weight_decay': 0.00014998256990705265}. Best is trial 0 with value:
```

```
0.8814166666666666.
[I 2025-07-08 08:42:56,455] Trial 2 finished with value: 0.8005833333333333 and
parameters: {'num_hidden_layers': 2, 'neurons_per_layer': 112, 'epochs': 100,
'lr': 0.007960225498043969, 'dropout_rate': 0.5, 'batch_size': 32, 'optimizer':
'RMSprop', 'weight_decay': 0.00046980805192913753}. Best is trial 0 with value:
0.8814166666666666.
[I 2025-07-08 08:45:05,165] Trial 3 finished with value: 0.86425 and parameters:
{'num_hidden_layers': 5, 'neurons_per_layer': 72, 'epochs': 85, 'lr':
0.004344539222441111, 'dropout_rate': 0.5, 'batch_size': 128, 'optimizer':
'SGD', 'weight_decay': 0.0005744733676755499}. Best is trial 0 with value:
0.8814166666666666.
[I 2025-07-08 08:46:17,479] Trial 4 finished with value: 0.8590833333333333 and
parameters: {'num_hidden_layers': 3, 'neurons_per_layer': 64, 'epochs': 60,
'lr': 0.0009264008464073184, 'dropout_rate': 0.5, 'batch_size': 128,
'optimizer': 'SGD', 'weight_decay': 3.8621196692052926e-05}. Best is trial 0
with value: 0.8814166666666666.
[I 2025-07-08 08:47:42,236] Trial 5 finished with value: 0.8810833333333333 and
parameters: {'num_hidden_layers': 2, 'neurons_per_layer': 120, 'epochs': 80,
'lr': 0.0010222542982193145, 'dropout_rate': 0.5, 'batch_size': 128,
'optimizer': 'SGD', 'weight_decay': 0.00018070933916449335}. Best is trial 0
with value: 0.8814166666666666.
[I 2025-07-08 08:52:01,570] Trial 6 finished with value: 0.8765 and parameters:
{'num_hidden_layers': 1, 'neurons_per_layer': 128, 'epochs': 100, 'lr':
0.009017944278052047, 'dropout_rate': 0.1, 'batch_size': 32, 'optimizer': 'SGD',
'weight_decay': 0.00020421724797268796}. Best is trial 0 with value:
0.8814166666666666.
[I 2025-07-08 08:54:35,361] Trial 7 finished with value: 0.85775 and parameters:
{'num_hidden_layers': 1, 'neurons_per_layer': 32, 'epochs': 60, 'lr':
0.008589890737706248, 'dropout_rate': 0.4, 'batch_size': 32, 'optimizer': 'SGD',
'weight_decay': 0.00030725793580119805}. Best is trial 0 with value:
0.8814166666666666.
[I 2025-07-08 08:56:51,854] Trial 8 finished with value: 0.8886666666666667 and
parameters: {'num_hidden_layers': 5, 'neurons_per_layer': 80, 'epochs': 90,
'lr': 0.006949968609931943, 'dropout_rate': 0.30000000000000004, 'batch_size':
128, 'optimizer': 'SGD', 'weight_decay': 6.677334308862196e-05}. Best is trial 8
with value: 0.8886666666666667.
[I 2025-07-08 08:57:05,253] Trial 9 finished with value: 0.8750833333333333 and
parameters: {'num_hidden_layers': 1, 'neurons_per_layer': 48, 'epochs': 15,
'lr': 0.014994162103390576, 'dropout_rate': 0.2, 'batch_size': 128, 'optimizer':
'SGD', 'weight_decay': 6.916027568905143e-05}. Best is trial 8 with value:
0.8886666666666667.
```

[54]:
```
study.best_value, study.best_params
```

[54]:
```
(0.8886666666666667,
 {'num_hidden_layers': 5,
  'neurons_per_layer': 80,
```

```
  'epochs': 90,
  'lr': 0.006949968609931943,
  'dropout_rate': 0.30000000000000004,
  'batch_size': 128,
  'optimizer': 'SGD',
  'weight_decay': 6.677334308862196e-05})
```

[56]:
```
# Best trial se model banaen
best_trial = study.best_trial
best_params = best_trial.params
# model init
input_dim = X_train.shape[1]
output_dim = 10
# Model ko train karein best hyperparameters ke saath
model = MyNN(input_dim, output_dim, best_params['num_hidden_layers'],␣
 ↪best_params['neurons_per_layer'], best_params['dropout_rate'])
# ... (training code jaisa pehle diya)

# Model ko save karein
torch.save(model.state_dict(), 'best_model.pth')
```