In [1]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

import torchvision
import torchvision.transforms as transforms
import torch.optim as optim

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

torch.set_printoptions(linewidth = 120)
torch.set_grad_enabled(True)
```

Out[1]:

```
<torch.autograd.grad_mode.set_grad_enabled at 0x297f952f5c0>
```

In [2]:

```
def get_num_correct(preds, labels):
    return preds.argmax(dim = 1).eq(labels).sum().item()
```

In [3]:

```
class Network(nn.Module):
    def __init__(self):
        super(Network, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 6, kernel_size = 5)
        self.conv2 = nn.Conv2d(in_channels = 6, out_channels = 12, kernel_size = 5)

        self.fc1 = nn.Linear(in_features = 12 * 4 * 4, out_features = 240)
        self.fc2 = nn.Linear(in_features = 240, out_features = 120)
        self.fc3 = nn.Linear(in_features = 120, out_features = 60)
        self.out = nn.Linear(in_features = 60, out_features = 10)

    def forward(self,t):
        #1. Input layer
        t = t

        #2. hidden conv layer
        t = self.conv1(t)
        t = F.relu(t)
        t = F.max_pool2d(t, kernel_size = 2,stride = 2)

        #3. hidden conv layer
        t = self.conv2(t)
        t = F.relu(t)
        t = F.max_pool2d(t, kernel_size = 2,stride = 2)

        #5. Hidden linear layer
        t = t.reshape(-1, 12 * 4 * 4)
        t = self.fc1(t)
        t = F.relu(t)

        #6. Hidden linear layer
        t = self.fc2(t)
        t = F.relu(t)

        #7. Hidden linear layer
        t = self.fc3(t)
        t = F.relu(t)

        #8. Output layer
        t = self.out(t)

        return t
```

In [4]:

```
train_set = torchvision.datasets.FashionMNIST(
            root = './data/FashionMNIST',
            train = True,
            download = True,
            transform = transforms.Compose([transforms.ToTensor()])
            )
```

In [5]:

```
network = Network()

train_loader = torch.utils.data.DataLoader(train_set, batch_size = 100)
optimizer = optim.Adam(network.parameters(), lr = 0.001)

for epoch in range(10):
    total_loss = 0
    total_correct = 0

    for batch in train_loader:
        images, labels = batch

        preds = network(images)
        loss = F.cross_entropy(preds, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        total_correct += get_num_correct(preds, labels)

    print("epoch:", epoch, "correct:", total_correct, "loss:", total_loss)
```

```
epoch: 0 correct: 40902 loss: 493.6738988161087
epoch: 1 correct: 47688 loss: 317.38948878645897
epoch: 2 correct: 49843 loss: 271.6878546476364
epoch: 3 correct: 51116 loss: 240.6783087849617
epoch: 4 correct: 51895 loss: 219.22745741903782
epoch: 5 correct: 52468 loss: 203.74074424803257
epoch: 6 correct: 52893 loss: 191.55619211494923
epoch: 7 correct: 53271 loss: 181.39917167276144
epoch: 8 correct: 53583 loss: 173.77001784741879
epoch: 9 correct: 53829 loss: 166.76734860241413
```

In [6]:

```
total_correct / len(train_set)
```

Out[6]:

```
0.89715
```

In [7]:

```
#CONFUSION MATRIX
#Getting predictions for the entire training set
@torch.no_grad()
def get_all_preds(model, loader):
    all_preds = torch.tensor([])
    for batch in loader:
        images, labels = batch
        preds = model(images)
        all_preds = torch.cat((all_preds, preds), dim = 0)
    return all_preds
```

In [8]:

```
prediction_loader = torch.utils.data.DataLoader(train_set, batch_size = 10000)
train_preds = get_all_preds(network, prediction_loader)
```

In [9]:

```
preds_correct = get_num_correct(train_preds, train_set.targets)
print('total_correct:', preds_correct)
print('accuracy:', preds_correct/len(train_set))
```

```
total_correct: 54014
accuracy: 0.9002333333333333
```

In [10]:

```
train_set.targets
```

Out[10]:

```
tensor([9, 0, 0,  ..., 3, 0, 5])
```

In [11]:

```
train_preds.argmax(dim = 1)
```

Out[11]:

```
tensor([9, 0, 0,  ..., 3, 0, 5])
```

In [12]:

```
#BUILDING CONFUSION MATRIX
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(train_set.targets, train_preds.argmax(dim = 1))
print(cm)
cm.shape
```

```
[[5217    6   79  100   19    0  527    0   52    0]
 [  10 5910    2   60    5    1    8    0    4    0]
 [  51    1 4818   72  661    0  381    0   16    0]
 [ 169   44    9 5543  133    1   96    0    5    0]
 [  14    7  233  221 5074    0  435    0   16    0]
 [   0    1    0    0    0 5723    0  190    4   82]
 [ 750   11  381  143  441    0 4222    0   51    1]
 [   0    0    0    0    0   23    0 5892    2   83]
 [  21    5    3   15   16    6   40    6 5888    0]
 [   0    1    0    0    0    8    0  258    6 5727]]
```

Out[12]:

```
(10, 10)
```