```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt
import torch.optim as optim

torch.set_grad_enabled(True)
```

```
<torch.autograd.grad_mode.set_grad_enabled at 0x7f9ecc772828>
```

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


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=120)
    self.fc2 = 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)

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

    # (5) Hidden Linear layer
    t=self.fc2(t)
    t=F.relu(t)

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

    return t
```

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

```
  0%|          | 16384/26421880 [00:00<03:00, 146407.28it/s]Downloading http://fashio
26427392it [00:00, 76387966.68it/s]
Extracting ./data/FashionMNIST/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/
32768it [00:00, 454632.21it/s]
  2%|▏         | 98304/4422102 [00:00<00:04, 895225.86it/s]Downloading http://fashio
Extracting ./data/FashionMNIST/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./data/
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-id
4423680it [00:00, 21921201.88it/s]
8192it [00:00, 111627.18it/s]
Extracting ./data/FashionMNIST/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./data/F
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-id
Extracting ./data/FashionMNIST/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/F
Processing...
Done!
```

Training with a single batch

```
network = Network()

train_loader = torch.utils.data.DataLoader(train_set, batch_size=100)
optimizer = optim.Adam(network.parameters(), lr=0.01)


for epoch in range(5):

  total_loss = 0
  total_correct = 0

  for batch in train_loader:
    images, labels = batch

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

    optimizer.zero_grad()
    loss.backward() #Calculate gradients
    optimizer.step() #Update weights

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

  print('epoch:', epoch, 'total_correct:', total_correct, 'loss:', total_loss)
```

```
epoch: 0 total_correct: 46521 loss: 350.3630883693695
epoch: 1 total_correct: 51225 loss: 237.73708787560463
epoch: 2 total_correct: 52009 loss: 215.0774446427822
epoch: 3 total_correct: 52288 loss: 208.53166519105434
epoch: 4 total_correct: 52615 loss: 199.48026624321938
```

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

```
0.8769166666666667
```

Getting predictions for the complete training set, without gradients and backpropagation. The cod
and backprop but now that we want to just get the predictions we can turn the gradient tracking fe
during the 5th epoch are still available in the 'Parameters' of the netowork layers. So, by turning the
the previously generated weights to make new predictions. By turning the gradient tracking off the
be less.

```
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
```

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

```
train_preds.shape
```

```
torch.Size([60000, 10])
```

```
print(train_preds.requires_grad) # Tells if gradient tracking feature is enabled or not. H
```

```
True
```

```
train_preds.grad # Generates no output because there no backpropagation
```

```
train_preds.grad_fn # Since the gradient tracking feature is turned on initially(in the im
# tensor, even though the gradient tensor is empty
```

```
<CatBackward at 0x7f9ecc23b4e0>
```

There are 2 ways of turning off gradients 1) Turn it of globally(check the import cell)

2) Turn if locally with 'torch.no_grad()' function

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

```
train_preds.shape
```

```
torch.Size([60000, 10])
```

```
print(train_preds.requires_grad) # Tells if gradient tracking feature is enabled or not. H
```

```
False
```

```
train_preds.grad # Generates no output because there no backpropagation
```

```
train_preds.grad_fn # Generates no output because pytorch is not keeping track of the grad
```

```
preds_correct = get_num_correct(train_preds, train_set.targets)
print('Total correct predictions = ', preds_correct)
print('Accuracy = ', preds_correct/ len(train_set))
```

```
Total correct predictions =  52894
Accuracy =  0.8815666666666667
```

Building a confusion matrix manually(without using the function from sklearn)

```
train_preds.shape
```

```
torch.Size([60000, 10])
```

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

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

```
train_set.targets
```

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

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

```
torch.Size([60000])
```

```
train_set.targets.shape
```

```
torch.Size([60000])
```

```
stacked = torch.stack((train_set.targets, train_preds.argmax(dim=1)), dim=1)
```

```
stacked.shape
```

```
torch.Size([60000, 2])
```

```
stacked
```

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

```
cmat = torch.zeros(10, 10, dtype=torch.int64)
cmat
```

```
tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
for pair in stacked:
  tar, pr = pair.tolist()
  cmat[tar, pr] = cmat[tar, pr]+1
```

```
cmat
```

```
tensor([[5526,    1,   49,   68,    9,    0,  319,    0,   28,    0],
        [  30, 5851,    2,   74,    6,    2,   33,    0,    2,    0],
        [  61,    1, 4433,   62,  771,    0,  643,    0,   29,    0],
        [ 330,   15,    4, 5333,  156,    0,  160,    0,    2,    0],
        [  13,    2,  257,  210, 4852,    0,  652,    0,   14,    0],
        [   2,    0,    2,    5,    0, 5746,    1,  166,    7,   71],
        [1321,    2,  399,   89,  415,    0, 3747,    0,   27,    0],
        [   0,    0,    0,    0,    0,   31,    0, 5769,    5,  195],
        [  43,    4,   15,   22,   21,    2,   82,    7, 5804,    0],
        [   0,    0,    0,    1,    0,   16,    1,  147,    2, 5833]])
```

Creating confusion matrix using a built-in function

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(train_set.targets, train_preds.argmax(dim=1))
```

```
cm
```

```
array([[5526,    1,   49,   68,    9,    0,  319,    0,   28,    0],
       [  30, 5851,    2,   74,    6,    2,   33,    0,    2,    0],
       [  61,    1, 4433,   62,  771,    0,  643,    0,   29,    0],
       [ 330,   15,    4, 5333,  156,    0,  160,    0,    2,    0],
       [  13,    2,  257,  210, 4852,    0,  652,    0,   14,    0],
       [   2,    0,    2,    5,    0, 5746,    1,  166,    7,   71],
       [1321,    2,  399,   89,  415,    0, 3747,    0,   27,    0],
       [   0,    0,    0,    0,    0,   31,    0, 5769,    5,  195],
       [  43,    4,   15,   22,   21,    2,   82,    7, 5804,    0],
       [   0,    0,    0,    1,    0,   16,    1,  147,    2, 5833]])
```