```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#!tipset 3 pm_peak_psrt="Peak/Off-Peak,Full,Off-Peak" --save="data/" --name="v1"
```

```
# Set device
device = torch.device("cuda:0")
def plot_grad(inp, ax, title):
    inp = inp.cpu().detach().numpy().T
    im = ax.imshow(inp, cmap='jet')
    ax.set_title(title)
    return im
def plot_grad_flow(named_parameters, fig, ax, filename):
    '''Plots the gradients flowing through different layers in the net during training.
    Usage: Call this function after loss.backward()
    model.parameters() should be passed as argument
    '''
    ave_grads = []
    max_grads = []
    layers = []
    for n, p in named_parameters:
        if(p.requires_grad and "bias" not in n):
            layers.append(n)
            ave_grads.append(p.grad.abs().mean())
            max_grads.append(p.grad.abs().max())
    print("Max grads: ", max_grads)
    print("Ave grads: ", ave_grads)
    ax.plot(ave_grads, alpha=0.3, color="b")
    ax.plot(max_grads, alpha=0.3, color="r")
    ax.hlines(0, 0, len(ave_grads), +1, lw=2, color="k" )
    ax.xticks(range(len(ave_grads)), layers, rotation="vertical")
    ax.xlim(left=0, right=len(ave_grads))
    ax.ylim(bottom = -0.001, top=0.02) # zoom in on the lower gradient regions
    ax.xlabel("Layers")
    ax.ylabel("average gradient")
    ax.title("Gradient flow")
    ax.grid(True)
    ax.legend([Line2D([0], [0], color="c"), Line2D([0], [0], color="b"), Line2D([0], [0], color="k")], ['max-gradient', 'mean-gradient', 'zero-gradient'])
    fig.savefig(filename, bbox_inches='tight')
```

```
%matplotlib inline
fig, ax = plt.subplots(1, 1)
ax.set_xlabel('Epochs')
ax.set_ylabel('Loss')
ax.set_ylim(0.0, 1.0)
ax.set_xlim(0, 10)
ax.grid(True)
ax.legend(['Train', 'Test'])
plt_loss = ax.plot([], [], 'r', [], [], 'b')
```

```
%matplotlib inline
fig, ax = plt.subplots(1, 1)
ax.set_xlabel('Epochs')
ax.set_ylabel('Accuracy')
ax.set_ylim(0.0, 1.0)
ax.set_xlim(0, 10)
ax.grid(True)
ax.legend(['Train', 'Test'])
plt_acc = ax.plot([], [], 'r', [], [], 'b')
```

```
class Net_v1(nn.Module):
    def __init__(self):
        super(Net_v1, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.bn1 = nn.BatchNorm2d(6)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.bn2 = nn.BatchNorm2d(16)
        self.conv3 = nn.Conv2d(16, 32, 5)
        self.bn3 = nn.BatchNorm2d(32)
        self.conv4 = nn.Conv2d(32, 64, 5)
        self.bn4 = nn.BatchNorm2d(64)
        self.conv5 = nn.Conv2d(64, 128, 5)
        self.bn5 = nn.BatchNorm2d(128)
        self.conv6 = nn.Conv2d(128, 256, 5)
        self.bn6 = nn.BatchNorm2d(256)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(256 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        x = F.max_pool2d(F.relu(self.bn1(self.conv1(x))), (2, 2))
        x = F.max_pool2d(F.relu(self.bn2(self.conv2(x))), (2, 2))
        x = F.max_pool2d(F.relu(self.bn3(self.conv3(x))), (2, 2))
        x = F.max_pool2d(F.relu(self.bn4(self.conv4(x))), (2, 2))
        x = F.max_pool2d(F.relu(self.bn5(self.conv5(x))), (2, 2))
        x = F.max_pool2d(F.relu(self.bn6(self.conv6(x))), (2, 2))
        x = x.view(-1, 256 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
net = Net_v1().to(device)
```

```
criterion = nn.CrossEntropyLoss()
learning_rate = 0.01
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
```

```
# Loader & Loss function
# Training & Evaluation functions
criterion = nn.MSELoss()

#Training variables
epochs = 10
batch_size = 128

train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                           shuffle=True, num_workers=2)
test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                          shuffle=False, num_workers=2)

# Lists for visualization
train_losses = []
test_losses = []
train_correct = []
test_correct = []

for epoch in range(epochs):
    trn_corr = 0
    tst_corr = 0

    # Run the training batches
    for b, (X_train, y_train) in enumerate(train_loader):
        b+=1
        # Apply the model
        y_pred = net(X_train)
        loss = criterion(y_pred, y_train)

        # Tally the number of correct predictions
        predicted = torch.max(y_pred.data, 1)[1]
        batch_corr = (predicted == y_train).sum()
        trn_corr += batch_corr

        # Update parameters
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Print interim results
        if b%600 == 0:
            print(f'epoch: {epoch:2}  batch: {b:4}  loss: {loss.item():10.8f}  \
            accuracy: {trn_corr.item()/(b*batch_size):7.3f%}')

    # Update train loss & accuracy for the epoch
    train_losses.append(loss)
    train_correct.append(trn_corr)

    # Run the testing batches
    for b, (X_test, y_test) in enumerate(test_loader):
        y_val = net(X_test)

        # Tally the number of correct predictions
        predicted = torch.max(y_val.data, 1)[1]
        batch_corr = (predicted == y_test).sum()
        tst_corr += batch_corr

    # Update test loss & accuracy for the epoch
    loss = criterion(y_val, y_test)
    test_losses.append(loss)
    test_correct.append(tst_corr)
```

```
CuDNNBatchNormBackward0(tensor([0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 
```