Problem 3

In [ ]:
```
import torch.nn as nn
from torch.utils.data import DataLoader
import torch
import torchvision
import torchvision.transforms as transforms
```

In [ ]:
```
# Instantiate model with BN and load trained parameters
class smallNetTrain(nn.Module) :
    # CIFAR-10 data is 32*32 images with 3 RGB channels
    def __init__(self, input_dim=3*32*32) :
        super().__init__()

        self.conv1 = nn.Sequential(
                            nn.Conv2d(3, 16, kernel_size=3, padding=1),
                            nn.BatchNorm2d(16),
                            nn.ReLU()
                            )
        self.conv2 = nn.Sequential(
                            nn.Conv2d(16, 16, kernel_size=3, padding=1),
                            nn.BatchNorm2d(16),
                            nn.ReLU()
                            )
        self.fc1 = nn.Sequential(
                            nn.Linear(16*32*32, 32*32),
                            nn.BatchNorm1d(32*32),
                            nn.ReLU()
                            )
        self.fc2 = nn.Sequential(
                            nn.Linear(32*32, 10),
                            nn.ReLU()
                            )
    def forward(self, x) :
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.float().view(-1, 16*32*32)
        x = self.fc1(x)
        x = self.fc2(x)

        return x

model = smallNetTrain()
model.load_state_dict(torch.load("./smallNetSaved",map_location=torch.device('cpu')))
```

Out[ ]:
```
<All keys matched successfully>
```

In [ ]:
```
class smallNetTest(nn.Module) :
    # CIFAR-10 data is 32*32 images with 3 RGB channels
    def __init__(self, input_dim=3*32*32) :
        super().__init__()

        self.conv1 = nn.Sequential(
                            nn.Conv2d(3, 16, kernel_size=3, padding=1),
                            nn.ReLU()
                            )
        self.conv2 = nn.Sequential(
                            nn.Conv2d(16, 16, kernel_size=3, padding=1),
                            nn.ReLU()
```

```
                            )
        self.fc1 = nn.Sequential(
                            nn.Linear(16*32*32, 32*32),
                            nn.ReLU()
                            )
        self.fc2 = nn.Sequential(
                            nn.Linear(32*32, 10),
                            nn.ReLU()
                            )
    def forward(self, x) :
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.float().view(-1, 16*32*32)
        x = self.fc1(x)
        x = self.fc2(x)

        return x

model_test = smallNetTest()
```

Function combine_conv_and_batch merges convolutional network and batchnorm, and combine_lin_and_batch merges linear network and batchnorm.

In [ ]:
```
conv1_bn_beta, conv1_bn_gamma = model.conv1[1].bias, model.conv1[1].weight
conv1_bn_mean, conv1_bn_var = model.conv1[1].running_mean, model.conv1[1].running_va
conv2_bn_beta, conv2_bn_gamma = model.conv2[1].bias, model.conv2[1].weight
conv2_bn_mean, conv2_bn_var = model.conv2[1].running_mean, model.conv2[1].running_va
fc1_bn_beta, fc1_bn_gamma = model.fc1[1].bias, model.fc1[1].weight
fc1_bn_mean, fc1_bn_var = model.fc1[1].running_mean, model.fc1[1].running_var
eps = 1e-05

# merging function
def combine_conv_and_batch(conv : nn.Conv2d, batch : nn.BatchNorm2d):
    conv_w = conv.weight.clone().view(conv.out_channels,-1)
    batch_w = torch.diag(batch.weight / torch.sqrt(batch.running_var+batch.eps))

    # conv_b = conv.bias.clone() -> conv.bias
    batch_b = batch.bias - batch.weight * batch.running_mean / torch.sqrt(batch.run

    return (batch_w@conv_w).view(conv.weight.size()), batch_w@conv.bias + batch_b

def combine_lin_and_batch(lin : nn.Linear, batch : nn.BatchNorm2d):
    # lin_w = lin.weight.clone().view(lin.out_features,-1) -> lin.weight
    batch_w = torch.diag(batch.weight / torch.sqrt(batch.running_var+batch.eps))

    # conv_b = conv.bias.clone() -> conv.bias
    batch_b = batch.bias - batch.weight * batch.running_mean / torch.sqrt(batch.run

    return batch_w@lin.weight, batch_w@lin.bias + batch_b

# Initialize the following parameters
w, b = combine_conv_and_batch(model.conv1[0], model.conv1[1])
model_test.conv1[0].weight.data = w
model_test.conv1[0].bias.data = b

w, b = combine_conv_and_batch(model.conv2[0], model.conv2[1])
model_test.conv2[0].weight.data = w
model_test.conv2[0].bias.data = b

w, b = combine_lin_and_batch(model.fc1[0], model.fc1[1])
model_test.fc1[0].weight.data = w
model_test.fc1[0].bias.data = b
```

```
model_test.fc2[0].weight.data = model.fc2[0].weight.data
model_test.fc2[0].bias.data = model.fc2[0].bias.data
```

In [ ]:
```
model.eval()
# model_test.eval()  # not necessary since model_test has no BN or dropout

test_dataset = torchvision.datasets.CIFAR10(root='./cifar_10data/',
                                train=False,
                                transform=transforms.ToTensor(), download = True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=100, shu

diff = []
with torch.no_grad():
    for images, _ in test_loader:
        diff.append(torch.norm(model(images) - model_test(images))**2)

print(max(diff))
```

```
Files already downloaded and verified
tensor(6.5861e-09)
```

The result is less than 10^{-8}, as we desired.