In [31]:
```
import torch
import random
import numpy as np

# set random seed
seed = 66
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
np.random.seed(seed)
random.seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

from torchvision.datasets import mnist # import dataset

from torch import nn
from torch.autograd import Variable
```

In [32]:
```
# Convert the data
def data_tf(x):
    x = np.array(x, dtype='float32') / 255
    x = (x - 0.5) / 0.5
    x = x.reshape((-1,))
    x = torch.from_numpy(x)
    return x

train_set = mnist.MNIST('./data', train=True, transform=data_tf,
test_set = mnist.MNIST('./data', train=False, transform=data_tf,
```

In [33]:
```
# print the shape of data
a, a_label = train_set[0]
print(a.shape)
print(a_label)
```

```
torch.Size([784])
5
```

In [34]:
```
# Set the data size of a batch
from torch.utils.data import DataLoader
train_data = DataLoader(train_set, batch_size=64, shuffle=True)
test_data = DataLoader(test_set, batch_size=128, shuffle=False)
a, a_label = next(iter(train_data))
print(a.shape)
print(a_label.shape)
```

```
torch.Size([64, 784])
torch.Size([64])
```

In [35]:
```
# Four-Layer network
net = nn.Sequential(
    nn.Linear(784, 400),
    nn.ReLU(),  # Activation function: ReLU
    nn.Linear(400, 200),
    nn.ReLU(),
    nn.Linear(200, 100),
    nn.ReLU(),
    nn.Linear(100, 10)
)
net
```

Out[35]:
```
Sequential(
  (0): Linear(in_features=784, out_features=400, bias=True)
  (1): ReLU()
  (2): Linear(in_features=400, out_features=200, bias=True)
  (3): ReLU()
  (4): Linear(in_features=200, out_features=100, bias=True)
  (5): ReLU()
  (6): Linear(in_features=100, out_features=10, bias=True)
)
```

In [36]:
```
# Define Loss function
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), 1e-1) # learning r
```

In [37]:
```
# Training with GPU
import torch
from torch.autograd import Variable
device = torch.device("cuda" if torch.cuda.is_available() else '
net = net.to(device)
device
```

Out[37]:
```
device(type='cuda')
```

In [38]:
```
import matplotlib.pyplot as plt
# Ensure inline plotting in the notebook
%matplotlib inline
# Train
losses = []
acces = []
eval_losses = []
eval_acces = []

for e in range(25):
    train_loss = 0
    train_acc = 0
    net.train()
    for im, label in train_data:
```

```
        # GPU
        im = im.to(device)
        label = label.to(device)
        im = Variable(im)
        label = Variable(label)
        # Train the net
        out = net(im)
        loss = criterion(out, label)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        # Calculate the accuracy
        _, pred = out.max(1)
        num_correct = (pred == label).sum().item()
        acc = num_correct / im.shape[0]
        train_acc += acc

    losses.append(train_loss / len(train_data))
    acces.append(train_acc / len(train_data))

    # test
    eval_loss = 0
    eval_acc = 0
    net.eval()
    with torch.no_grad():
        for im, label in test_data:
            # GPU
            im = im.to(device)
            label = label.to(device)
            im = Variable(im)
            label = Variable(label)
            # Test the net
            out = net(im)
            loss = criterion(out, label)
            eval_loss += loss.item()
            _, pred = out.max(1)
            num_correct = (pred == label).sum().item()
            acc = num_correct / im.shape[0]
            eval_acc += acc

    eval_losses.append(eval_loss / len(test_data))
    eval_acces.append(eval_acc / len(test_data))

    print('epoch: {}, Train Loss: {:.6f}, Train Acc: {:.6f}, Eva
          .format(e, train_loss / len(train_data), train_acc / l
                  eval_loss / len(test_data), eval_acc / len(tes
print("Accuracy{:.2f}%".format(100*(eval_acc / len(test_data))))
```

```
# plot
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.title('train loss')
plt.plot(np.arange(len(losses)), losses)
plt.title('train Acc')
plt.subplot(1, 2, 2)
plt.plot(np.arange(len(eval_acces)), eval_acces)
```

```
epoch: 0, Train Loss: 0.507897, Train Acc: 0.838153, Eval Loss:
0.232668, Eval Acc: 0.924941
epoch: 1, Train Loss: 0.178281, Train Acc: 0.945279, Eval Loss:
0.125707, Eval Acc: 0.961234
epoch: 2, Train Loss: 0.120802, Train Acc: 0.962187, Eval Loss:
0.099292, Eval Acc: 0.969244
epoch: 3, Train Loss: 0.096452, Train Acc: 0.969316, Eval Loss:
0.122085, Eval Acc: 0.960245
epoch: 4, Train Loss: 0.075509, Train Acc: 0.976296, Eval Loss:
0.095585, Eval Acc: 0.969244
epoch: 5, Train Loss: 0.063368, Train Acc: 0.980211, Eval Loss:
0.079110, Eval Acc: 0.974782
epoch: 6, Train Loss: 0.054704, Train Acc: 0.982459, Eval Loss:
0.073092, Eval Acc: 0.976859
epoch: 7, Train Loss: 0.045404, Train Acc: 0.985391, Eval Loss:
0.083736, Eval Acc: 0.972903
epoch: 8, Train Loss: 0.040190, Train Acc: 0.987040, Eval Loss:
0.064448, Eval Acc: 0.980123
epoch: 9, Train Loss: 0.033675, Train Acc: 0.988656, Eval Loss:
0.077416, Eval Acc: 0.975672
epoch: 10, Train Loss: 0.028912, Train Acc: 0.990538, Eval Loss:
0.070495, Eval Acc: 0.978540
epoch: 11, Train Loss: 0.027194, Train Acc: 0.991321, Eval Loss:
0.157035, Eval Acc: 0.953125
epoch: 12, Train Loss: 0.021699, Train Acc: 0.993087, Eval Loss:
0.097316, Eval Acc: 0.974486
epoch: 13, Train Loss: 0.021499, Train Acc: 0.992654, Eval Loss:
0.076297, Eval Acc: 0.979134
epoch: 14, Train Loss: 0.016154, Train Acc: 0.994786, Eval Loss:
0.070035, Eval Acc: 0.981112
epoch: 15, Train Loss: 0.016631, Train Acc: 0.994570, Eval Loss:
0.074968, Eval Acc: 0.978540
epoch: 16, Train Loss: 0.011775, Train Acc: 0.996585, Eval Loss:
0.070516, Eval Acc: 0.980123
epoch: 17, Train Loss: 0.010250, Train Acc: 0.996802, Eval Loss:
0.068274, Eval Acc: 0.982100
epoch: 18, Train Loss: 0.009988, Train Acc: 0.996885, Eval Loss:
0.083006, Eval Acc: 0.980024
epoch: 19, Train Loss: 0.011779, Train Acc: 0.996302, Eval Loss:
0.080636, Eval Acc: 0.981013
epoch: 20, Train Loss: 0.007749, Train Acc: 0.997435, Eval Loss:
0.085839, Eval Acc: 0.978738
epoch: 21, Train Loss: 0.013789, Train Acc: 0.995902, Eval Loss:
0.072715, Eval Acc: 0.982793
epoch: 22, Train Loss: 0.006529, Train Acc: 0.998018, Eval Loss:
0.072597, Eval Acc: 0.983386
epoch: 23, Train Loss: 0.004952, Train Acc: 0.998484, Eval Loss:
0.077594, Eval Acc: 0.984078
epoch: 24, Train Loss: 0.002080, Train Acc: 0.999584, Eval Loss:
```

```
0.072328, Eval Acc: 0.984869
Accuracy98.49%
```

Out[38]: [<matplotlib.lines.Line2D at 0x199ef8f6a00>]