In [105]:
```
#how to use google colab: https://pytorch.org/tutorials/beginner/colab.html
#pytorch tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
import torch
import torch.nn.functional as F
from torchvision import datasets, transforms
import numpy as np
```

In [211]:
```
# define the network structure
class fc_net(torch.nn.Module):
  def __init__(self, num_in=784, num_out=10):
    super(fc_net, self).__init__()
    self.h1 = torch.nn.Linear(in_features=num_in, out_features=256)
    self.h2 = torch.nn.Linear(in_features=256, out_features=128)
    self.h3 = torch.nn.Linear(in_features=128, out_features=num_out)
  def forward(self, inputs):
    a1 = F.relu(self.h1(inputs))
    #print('Lets see a1')
    #print(a1[0, 0:10])
    a2 = F.relu(self.h2(a1))
    a3 = F.softmax(self.h3(a2),dim=-1)
    return a3
```

In [212]:
```
# use data_loader to load_in data
train_data = datasets.MNIST('./', train=True, download=True, transform=transforms.Compose([tran
                                (0.1307,), (0.3081,))]))
train_loader = torch.utils.data.DataLoader(train_data, batch_size=10, shuffle=True)

val_data = datasets.MNIST('./', train=False, download=True, transform=transforms.Compose([trans
                                (0.1307,), (0.3081,))]))
val_loader = torch.utils.data.DataLoader(val_data, batch_size=10, shuffle=True)


cur_x, cur_y = next(iter(train_loader))
print(cur_x.size())
print(cur_y.size())
```

```
torch.Size([10, 1, 28, 28])
torch.Size([10])
```

In [234]:
```
cur_x = torch.reshape(cur_x, (10, 28*28))
model = fc_net(num_in=28*28, num_out=10)
loss = torch.nn.CrossEntropyLoss()    # loss function
optimizer = torch.optim.SGD(model.parameters(), lr=.08)  #stochastic gradient descent optimizer
preds = model.forward(cur_x) # prediction
cur_loss = loss(preds, cur_y)  # loss
optimizer.zero_grad()    #set the gradients to zero
cur_loss.backward()     #collect a new set of gradients , model learns
optimizer.step()        #optimize weights
new_preds = model.forward(cur_x)
print(preds[0])
print(new_preds[0])
```

```
tensor([0.1118, 0.1095, 0.0897, 0.0917, 0.0983, 0.1171, 0.0963, 0.0967, 0.0985,
        0.0903], grad_fn=<SelectBackward>)
tensor([0.1121, 0.1104, 0.0897, 0.0919, 0.0988, 0.1163, 0.0965, 0.0964, 0.0979,
        0.0900], grad_fn=<SelectBackward>)
```

In [235]:
```
# convert between numpy and tensor
pred_array = preds[0].detach().numpy()
print(pred_array)
pred_tensor = torch.from_numpy(pred_array).float()
print(pred_tensor)
```

```
[0.11180003 0.10950512 0.08971722 0.09169915 0.09831317 0.11706373
 0.09633358 0.09672844 0.09852324 0.09031633]
tensor([0.1118, 0.1095, 0.0897, 0.0917, 0.0983, 0.1171, 0.0963, 0.0967, 0.0985,
        0.0903])
```

In [236]:
```
for epoch in range(8):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs = torch.reshape(inputs, (10, 28*28))

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model.forward(inputs) # prediction
        cur_loss = loss(outputs, labels)  # loss
        cur_loss.backward()
        optimizer.step()

        # print statistics
        running_loss += cur_loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 1.663
[1,  4000] loss: 1.546
[1,  6000] loss: 1.527
[2,  2000] loss: 1.516
[2,  4000] loss: 1.510
[2,  6000] loss: 1.507
[3,  2000] loss: 1.498
[3,  4000] loss: 1.500
[3,  6000] loss: 1.495
[4,  2000] loss: 1.491
[4,  4000] loss: 1.491
[4,  6000] loss: 1.491
[5,  2000] loss: 1.487
[5,  4000] loss: 1.489
[5,  6000] loss: 1.488
[6,  2000] loss: 1.484
[6,  4000] loss: 1.486
[6,  6000] loss: 1.487
[7,  2000] loss: 1.485
[7,  4000] loss: 1.482
[7,  6000] loss: 1.484
[8,  2000] loss: 1.481
[8,  4000] loss: 1.481
[8,  6000] loss: 1.481
Finished Training
```

In [237]:
```
correct_count, all_count = 0, 0
for images,labels in val_loader:
  for i in range(len(labels)):
    img = images[i].view(1, 784)
    with torch.no_grad():
        logps = model(img)


    ps = torch.exp(logps)
    probab = list(ps.numpy()[0])
    pred_label = probab.index(max(probab))

    true_label = labels.numpy()[i]

    if(true_label == pred_label):
      correct_count += 1
    all_count += 1

print( correct_count)
print(all_count)

print("Number Of Images Tested =", all_count)
print("\nModel Accuracy =", (correct_count*1./all_count))
```

```
9705
10000
('Number Of Images Tested =', 10000)
('\nModel Accuracy =', 0.9705)
```

In [ ]:

In [ ]:

In [ ]: