In [1]:

```python
import os

import torch
import torchvision
from torch import nn
from torch.autograd import Variable
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision.datasets import MNIST
from torchvision.utils import save_image
```

In [2]:

```python
if not os.path.exists('./mlp_img'):
    os.mkdir('./mlp_img')
```

In [3]:

```python
def to_img(x):
    x = 0.5 * (x + 1)
    x = x.clamp(0, 1)
    x = x.view(x.size(0), 1, 28, 28)
    return x
```

In [4]:

```python
num_epochs = 100
batch_size = 128
learning_rate = 1e-3
```

In [6]:

```
img_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])

dataset = MNIST('./data', download=True,transform=img_transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIS
T/raw/train-images-idx3-ubyte.gz


Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIS
T/raw/train-labels-idx1-ubyte.gz


Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/
raw/t10k-images-idx3-ubyte.gz


Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/
raw/t10k-labels-idx1-ubyte.gz


Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

In [7]:

```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True), nn.Linear(64, 12), nn.ReLU(True), nn.Linear(12, 3))
        self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(True),
            nn.Linear(12, 64),
            nn.ReLU(True),
            nn.Linear(64, 128),
            nn.ReLU(True), nn.Linear(128, 28 * 28), nn.Tanh())

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

In [11]:

```
model = autoencoder().cuda()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(
    model.parameters(), lr=learning_rate, weight_decay=1e-5)
```

In [12]:

```
for epoch in range(num_epochs):
    for data in dataloader:
        img, _ = data
        img = img.view(img.size(0), -1)
        img = Variable(img).cuda()
        # ===================forward=====================
        output = model(img)
        loss = criterion(output, img)
        # ===================backward====================
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    # ===================log========================
    print('epoch [{}/{}], loss:{:.4f}'
          .format(epoch + 1, num_epochs, loss.item()))
    if epoch % 10 == 0:
        pic = to_img(output.cpu().data)
        save_image(pic, './mlp_img/image_{}.png'.format(epoch))

torch.save(model.state_dict(), './sim_autoencoder.pth')
```

```
epoch [1/100], loss:0.1782
epoch [2/100], loss:0.1633
epoch [3/100], loss:0.1622
epoch [4/100], loss:0.1551
epoch [5/100], loss:0.1587
epoch [6/100], loss:0.1650
epoch [7/100], loss:0.1488
epoch [8/100], loss:0.1379
epoch [9/100], loss:0.1314
epoch [10/100], loss:0.1431
epoch [11/100], loss:0.1380
epoch [12/100], loss:0.1537
epoch [13/100], loss:0.1230
epoch [14/100], loss:0.1375
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-12-88e1cc92a78c> in <module>()
      1 for epoch in range(num_epochs):
----> 2     for data in dataloader:
      3         img, _ = data
      4         img = img.view(img.size(0), -1)
      5         img = Variable(img).cuda()

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py in __next__(self)
    519             if self._sampler_iter is None:
    520                 self._reset()
--> 521             data = self._next_data()
    522             self._num_yielded += 1
    523             if self._dataset_kind == _DatasetKind.Iterable and \

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py in _next_data(sel
f)
    559     def _next_data(self):
    560         index = self._next_index()  # may raise StopIteration
--> 561         data = self._dataset_fetcher.fetch(index)  # may raise StopIteration
    562         if self._pin_memory:
    563             data = _utils.pin_memory.pin_memory(data)

/usr/local/lib/python3.7/dist-packages/torch/utils/data/_utils/fetch.py in fetch(self,
 possibly_batched_index)
     47     def fetch(self, possibly_batched_index):
     48         if self.auto_collation:
---> 49             data = [self.dataset[idx] for idx in possibly_batched_index]
     50         else:
     51             data = self.dataset[possibly_batched_index]

/usr/local/lib/python3.7/dist-packages/torch/utils/data/_utils/fetch.py in <listcomp>(.
0)
     47     def fetch(self, possibly_batched_index):
     48         if self.auto_collation:
---> 49             data = [self.dataset[idx] for idx in possibly_batched_index]
     50         else:
     51             data = self.dataset[possibly_batched_index]

/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py in __getitem__(sel
f, index)
    132 
    133         if self.transform is not None:
--> 134             img = self.transform(img)
    135 
    136         if self.target_transform is not None:

/usr/local/lib/python3.7/dist-packages/torchvision/transforms/transforms.py in __call__
(self, img)
     59     def __call__(self, img):
     60         for t in self.transforms:
---> 61             img = t(img)
     62         return img
     63 

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in _call_impl(self, *
input, **kwargs)
   1100         if not (self._backward_hooks or self._forward_hooks or self._forward_pr
e_hooks or _global_backward_hooks
   1101                 or _global_forward_hooks or _global_forward_pre_hooks):
```

```
-> 1102                 return forward_call(*input, **kwargs)
   1103             # Do not call functions when jit is used
   1104             full_backward_hooks, non_full_backward_hooks = [], []

/usr/local/lib/python3.7/dist-packages/torchvision/transforms/transforms.py in forward
(self, tensor)
    224                 Tensor: Normalized Tensor image.
    225             """
--> 226             return F.normalize(tensor, self.mean, self.std, self.inplace)
    227 
    228         def __repr__(self):

/usr/local/lib/python3.7/dist-packages/torchvision/transforms/functional.py in normaliz
e(tensor, mean, std, inplace)
    348             mean = mean.view(-1, 1, 1)
    349         if std.ndim == 1:
--> 350             std = std.view(-1, 1, 1)
    351         tensor.sub_(mean).div_(std)
    352         return tensor

KeyboardInterrupt:
```

Conv Autoencoder

In [15]:

```python
if not os.path.exists('./dc_img'):
    os.mkdir('./dc_img')
```

In [22]:

```python
class Conv_autoencoder(nn.Module):
    def __init__(self):
        super(Conv_autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, 3, stride=3, padding=1),  # b, 16, 10, 10
            nn.ReLU(True),
            nn.MaxPool2d(2, stride=2),  # b, 16, 5, 5
            nn.Conv2d(16, 8, 3, stride=2, padding=1),  # b, 8, 3, 3
            nn.ReLU(True),
            nn.MaxPool2d(2, stride=1)  # b, 8, 2, 2
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(8, 16, 3, stride=2),  # b, 16, 5, 5
            nn.ReLU(True),
            nn.ConvTranspose2d(16, 8, 5, stride=3, padding=1),  # b, 8, 15, 15
            nn.ReLU(True),
            nn.ConvTranspose2d(8, 1, 2, stride=2, padding=1),  # b, 1, 28, 28
            nn.Tanh()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

In [40]:

```python
model = Conv_autoencoder().cuda()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate,
                             weight_decay=1e-5)
```

In [42]:

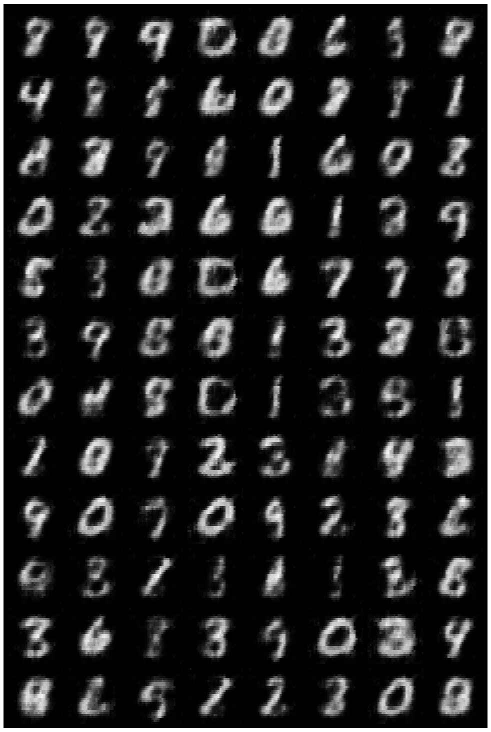
```
from PIL import Image

for epoch in range(num_epochs):
    total_loss = 0
    for data in dataloader:
        img, _ = data
        img = Variable(img).cuda()
        # ===================forward=====================
        output = model(img)
        loss = criterion(output, img)
        # ===================backward====================
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.data
    # ===================log========================
    print('epoch [{}/{}], loss:{:.4f}'
          .format(epoch+1, num_epochs, total_loss))
    if epoch % 10 == 0:
        pic = to_img(output.cpu().data)

        save_image(pic, './dc_img/image_{}.png'.format(epoch))


torch.save(model.state_dict(), './conv_autoencoder.pth')
```

```
epoch [1/100], loss:83.8589
```

In [44]:

```
img
```

Out[44]:



In [38]:

```
print('./dc_img/image_'+str(epoch)+'.png')
```

```
./dc_img/image_0.png
```