变分自动编码器

在 pytorch 实现中，我们可以将我们的 loss 定义为下面的函数，由均方误差和 KL divergence 求和得到一个总的 loss

```
def loss_function(recon_x, x, mu, logvar):
    """
    recon_x: generating images
    x: origin images
    mu: latent mean
    logvar: latent log variance
    """
    MSE = reconstruction_function(recon_x, x)
    # loss = 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD_element = mu.pow(2).add_(logvar.exp()).mul_(-1).add_(1).add_(logvar)
    KLD = torch.sum(KLD_element).mul_(-0.5)
    # KL divergence
    return MSE + KLD
```

下面我们用 mnist 数据集来简单说明一下变分自动编码器

```
import os

import torch
from torch.autograd import Variable
import torch.nn.functional as F
from torch import nn
from torch.utils.data import DataLoader

from torchvision.datasets import MNIST
from torchvision import transforms as tfs
from torchvision.utils import save_image
```

```
im_tfs = tfs.Compose([
    tfs.ToTensor(),
    tfs.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5]) # 标准化
])

train_set = MNIST('./mnist', transform=im_tfs)
train_data = DataLoader(train_set, batch_size=128, shuffle=True)
```

```
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(784, 400)
        self.fc21 = nn.Linear(400, 20) # mean
        self.fc22 = nn.Linear(400, 20) # var
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 784)

    def encode(self, x):
        h1 = F.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

    def reparametrize(self, mu, logvar):
        std = logvar.mul(0.5).exp_()
        eps = torch.FloatTensor(std.size()).normal_()
        if torch.cuda.is_available():
            eps = Variable(eps.cuda())
        else:
            eps = Variable(eps)
        return eps.mul(std).add_(mu)

    def decode(self, z):
        h3 = F.relu(self.fc3(z))
        return F.tanh(self.fc4(h3))

    def forward(self, x):
        mu, logvar = self.encode(x) # 编码
        z = self.reparametrize(mu, logvar) # 重新参数化成正态分布
        return self.decode(z), mu, logvar # 解码，同时输出均值方差
```

```
net = VAE() # 实例化网络
if torch.cuda.is_available():
    net = net.cuda()
```

```
x, _ = train_set[0]
x = x.view(x.shape[0], -1)
if torch.cuda.is_available():
    x = x.cuda()
x = Variable(x)
_, mu, var = net(x)
```

```
print(mu)
```

```
Variable containing:

Columns 0 to 9
-0.0307 -0.1439 -0.0435  0.3472  0.0368 -0.0339  0.0274 -0.5608  0.0280  0.2742

Columns 10 to 19
-0.6221 -0.0894 -0.0933  0.4241  0.1611  0.3267  0.5755 -0.0237  0.2714 -0.2806
[torch.cuda.FloatTensor of size 1x20 (GPU 0)]
```

可以看到，对于输入，网络可以输出隐含变量的均值和方差，这里的均值方差还没有训练

下面开始训练

```
reconstruction_function = nn.MSELoss(size_average=False)

def loss_function(recon_x, x, mu, logvar):
    """
    recon_x: generating images
    x: origin images
    mu: latent mean
    logvar: latent log variance
    """
    MSE = reconstruction_function(recon_x, x)
    # loss = 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD_element = mu.pow(2).add_(logvar.exp()).mul_(-1).add_(1).add_(logvar)
    KLD = torch.sum(KLD_element).mul_(-0.5)
    # KL divergence
    return MSE + KLD

optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)

def to_img(x):
    '''
    定义一个函数将最后的结果转换回图片
    '''
    x = 0.5 * (x + 1.)
    x = x.clamp(0, 1)
    x = x.view(x.shape[0], 1, 28, 28)
    return x
```

```
for e in range(100):
```

```
    for im, _ in train_data:
        im = im.view(im.shape[0], -1)
        im = Variable(im)
        if torch.cuda.is_available():
            im = im.cuda()

        optimizer.zero_grad()
        recon_im, mu, logvar = net(im)
        loss = loss_function(recon_im, im, mu, logvar) / im.shape[0] # 将 loss 平均
        loss.backward()
        optimizer.step()

    if (e + 1) % 20 == 0:
        print('epoch: {}, Loss: {:.4f}'.format(e + 1, loss.data[0]))
        save = to_img(recon_im.cpu().data)
        if not os.path.exists('./vae_img'):
            os.mkdir('./vae_img')
        save_image(save, './vae_img/image_{}.png'.format(e + 1))
```

```
epoch: 20, Loss: 61.5803
epoch: 40, Loss: 62.9573
epoch: 60, Loss: 63.4285
epoch: 80, Loss: 64.7138
epoch: 100, Loss: 63.3343
```

可以看看使用变分自动编码器得到的结果，可以发现效果比一般的编码器要好很多

我们可以输出其中的均值看看

```
x, _ = train_set[0]
x = x.view(x.shape[0], -1)
if torch.cuda.is_available():
    x = x.cuda()
x = Variable(x)
_, mu, _ = net(x)
```

```
print(mu)
```

```
Variable containing:

Columns 0 to 9
 0.3861  0.5561  1.1995 -1.6773  0.9867  0.1244 -0.3443 -1.6658  1.3332  1.1606

Columns 10 to 19
 0.6898  0.3042  2.1044 -2.4588  0.0504  0.9743  1.1136  0.7872 -0.0777  1.6101
[torch.cuda.FloatTensor of size 1x20 (GPU 0)]
```

变分自动编码器虽然比一般的自动编码器效果要好，而且也限制了其输出的编码 (code) 的概率分布，但是它仍然是通过直接计算生成图片和原始图片的均方误差来生成 loss，这个方式并不好，在下一章生成对抗网络中，我们会讲一讲这种方式计算 loss 的局限性，然后会介绍一种新的训练办法，就是通过生成对抗的训练方式来训练网络而不是直接比较两张图片的每个像素点的均方误差