利用torch.nn构建神经网络

构建的神经网络拓扑图如下：

定义网络如下：

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)

        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        x = x.size()[1:]
        num = 1;
        for i in x:
            num = num * i

        return num


net = Net()
print(net)
```

注意：在模型中必须要定义forward函数，backward函数 会被 Autograd 自动创建。可以在forward 函数中使用任何针对Tensor的操作。

定义loss如下，采用均方差

```
net = Net()

# 输入的是4维向量（样本数，通道数，高，宽）
input = torch.randn(1, 1, 32, 32)
out = net(input)
target = torch.randn(10)
target = target.view(1, -1)
criterion = nn.MSELoss()
loss = criterion(out, target)
print(loss)
```

tensor(1.0299, grad_fn=)

调用loss.backward()获得反向传播的误差

```
net.zero_grad()
print('conv1.bias.grad before backward')
print(net.conv1.bias.grad)

loss.backward()

print('conv1.bias.grad after backward')
print(net.conv1.bias.grad)
```

conv1.bias.grad before backward None conv1.bias.grad after backward tensor([-0.0214, -0.0100, -0.0087, -0.0241, -0.0247, -0.0268])

更新权重

```
import torch.optim as optim

# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```