# DianAI培训1 - 作业讲解

贝贝组 黄涛

2018.07.26

# Learning Pytorch

- Official Site
- Pytorch Tutorials
- Pytorch docs
- Pytorch Examples

# Lectures & Solutions

https://github.com/hunto/DianAICourse_Summer

# 1. Define your network

A network is a subclass of `torch.nn.Model`

```
class FCNet(torch.nn.Module):
    # <-- class func. below -->
```

## 1.1 Complete Init Function

```
def __init__(self, input_size, hidden_size, output_size):
    super(FCNet, self).__init__()  # initialize father class
    self.input_size = input_size
    # then define your network layers, exp:
    self.fc1 = torch.nn.Linear(in_features=input_size,
                               out_features=hidden_size)
    self.relu = torch.nn.ReLU()
    self.fc2 = nn.Linear(in_features=hidden_size,
                         out_features=output_size)
```

## 1.2 Complete foward function

```
def forward(self, x):
    x = x.view(-1, self.input_size)
    out = self.fc1(x)
    out = self.relu(out)
    out = self.fc2(out)
    return out
```

**NOTE: You don't need to define backward function, Pytorch can automatically cal gradients and update weights.**

# 2. Process data

`torch.utils.data.Dataloader`

```
train_loader = DataLoader(
    datasets.MNIST(root='../data/MNIST', train=True,
       download=True,
       transform=transforms.Compose([transforms.ToTensor()])),
       batch_size=batch_size,
       shuffle=True
)

test_loader = DataLoader(
    datasets.MNIST(root='../data/MNIST', train=False,
       transform=transforms.Compose([transforms.ToTensor()])),
       batch_size=batch_size
)
```

## 3. Loss function & Optimizer

```
# define network
model = FCNet(input_vec_length, cell_num, num_classes)

if use_cuda:
    model = model.cuda()

# define loss function
ce_loss = torch.nn.CrossEntropyLoss()

# define optimizer
optimizer = optim.SGD(model.parameters(), lr=1e-3)
```

# 4. Train & Evaluation

```
# start train
train_step = 0
for eopch in range(1, 101):
    for data, target in train_loader:
    train_step += 1
    if use_cuda:
        data = data.cuda()
    acc, loss = train(model, data, target,
                      ce_loss, optimizer)
    if train_step % 100 == 0:
        print('Train set: Step: {}, Loss: {:.4f}, Accuracy: {:.
    if train_step % 1000 == 0:
        acc, loss = test(model, test_loader, ce_loss)
        print('\nTest set: Step: {}, Loss: {:.4f}, Accuracy: {:
```

# 5. Train function

```
def train(model, data, target, loss_func, optimizer):
    # initial optimizer
    optimizer.zero_grad()
    # net work will do forward computation defined in net's [fo
    output = model(data)
    # get predictions from outputs, the highest score's index i
    predictions = output.max(1, keepdim=True)[1]
    # cal correct predictions num
    correct = predictions.eq(target.view_as(predictions))
                                .sum().item()
    # cal accuracy
    acc = correct / len(target)
    # use loss func to cal loss
    loss = loss_func(output, target)
    # backward will back propagate loss
    loss.backward()
    # this will update all weights use the loss we just back pr
    optimizer.step()
    return acc, loss
```

# Run

```
Train set: Step: 100, Loss: 2.2471, Accuracy: 0.28
Train set: Step: 200, Loss: 2.2365, Accuracy: 0.33
Train set: Step: 300, Loss: 2.1982, Accuracy: 0.59
Train set: Step: 400, Loss: 2.1723, Accuracy: 0.61
Train set: Step: 500, Loss: 2.0980, Accuracy: 0.75
Train set: Step: 600, Loss: 2.1116, Accuracy: 0.66
Train set: Step: 700, Loss: 2.0654, Accuracy: 0.66
Train set: Step: 800, Loss: 2.0381, Accuracy: 0.70
Train set: Step: 900, Loss: 1.9948, Accuracy: 0.69
Train set: Step: 1000, Loss: 1.9796, Accuracy: 0.75

Test set: Step: 1000, Loss: 1.9691, Accuracy: 0.72
```