PyTorch

It is my management about PyTorch.

Chieh

Step

1. Setting up data
2. Create a model
3. Create an optimizer
4. Assign the model to GPU
5. Training
6. Making predictions
7. Saving Models

1.Setting up data

We can use the built-in dataset of `torchvision.datasets.ImageFolder` to quickly set up some dataloaders of downloaded images.

Besides, we also need to pre-process our data (images) by `transforms` before we use them.

DataLoaders

```
import torchvision
from torchvision import transforms

data_path = "./data/"
data = torchvision.datasets.ImageFolder(root=train_data_path, \
        transform=img_transforms)

batch_size=64
data_loader = torch.utils.data.DataLoader(data, \
        batch_size=batch_size)
```

Transforms

Basically, the parameters depend on your dataset.
For example, I set up the transforms for every image:

- Resize to 64x64
- Convert to tensor
- Normalize using ImageNet mean & std

We can use this function `transforms.Compose` to wrap them.

```
img_transforms = transforms.Compose([
    transforms.Resize((64,64)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225] )
    ])
```

2.Create a model

```
class NetworkName(nn.Module):

    def __init__(self):
        super(NetworkName, self).__init__()
        self.fc1 = nn.Linear(1000, 100)
        self.fc2 = nn.Linear(100, 10)

    def forward(self, x):
        x = x.view(-1, 1000)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

NetworkName = NetworkName()
```

You can also use `nn.Sequential` to wrap them.

```
class NetworkName(nn.Module):
    def __init__(self):
        super(NetworkName, self).__init__()
        self.classifier = nn.Sequential(
        nn.Linear(1000, 100),
        nn.ReLU(),
        nn.Linear(100,10))

    def forward(self,x):
        x = x.view(-1, 1000)
        x = self.classifier(x)
        return x
```

3.Create an optimizer

Here, we use `Adam` as our optimizer with a learning rate of 0.001.

```
optimizer = optim.Adam(NetworkName.parameters(), lr=0.001)
```

4.Assign the model to GPU

Assign the model to the GPU if available.

```
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")

NetworkName.to(device)
```

5.Training

Trains the model:

- Copying batches to the GPU if required.
- Calculating losses.
- Optimizing the network.
- Perform validation for each epoch.

```
epochs=5
for epoch in range(epochs):
    training_loss = 0.0
    valid_loss = 0.0
    NetworkName.train()
    for batch in train_data_loader:
        optimizer.zero_grad()
        inputs, targets = batch
        inputs = inputs.to(device)
        targets = targets.to(device)
        output = NetworkName(inputs)
        loss = torch.nn.CrossEntropyLoss(output, targets)
        loss.backward()
        optimizer.step()
        training_loss += loss.data.item() * inputs.size(0)
    training_loss /= len(train_data_loader.dataset)
```

```
NetworkName.eval()
num_correct = 0
num_examples = 0
for batch in val_data_loader:
    inputs, targets = batch
    inputs = inputs.to(device)
    output = NetworkName(inputs)
    targets = targets.to(device)
    loss = torch.nn.CrossEntropyLoss(output,targets)
    valid_loss += loss.data.item() * inputs.size(0)
    correct = torch.eq(torch.max(F.softmax(output, dim=1),
        dim=1)[1], targets)
    num_correct += torch.sum(correct).item()
    num_examples += correct.shape[0]
valid_loss /= len(val_data_loader.dataset)

print('Epoch: {}, Training Loss: {:.2f}, Validation Loss:\
{:.2f}, accuracy = {:.2f}'.format(epoch, training_loss,
valid_loss, num_correct / num_examples))
```

6.Making predictions

Labels are in alphanumeric order. For example, cat will be 0, fish will be 1. We'll need to transform the image and also make sure that the resulting tensor is copied to the appropriate device before applying our model to it.

```
labels = ['cat','fish']

img = Image.open("./image.JPG")
img = img_transforms(img).to(device)

prediction = F.softmax(NetworkName(img), dim=1)
prediction = prediction.argmax()
print(labels[prediction])
```

7.Saving Models

1. Save the entire model using `save`

```
torch.save(simplenet, "./NetworkName")
NetworkName = torch.load("./NetworkName")
```

2. Save the parameters using `state_dict` which is normally preferable, as it allows you to reuse parameters even if the model's structure changes (or apply parameters from one model to another).

```
torch.save(NetworkName.state_dict(), "./NetworkName")
NetworkName = NetworkName()
NetworkName_state_dict = torch.load("./simplenet")
NetworkName.load_state_dict(NetworkName_state_dict)
```

Reference

- beginners-pytorch-deep-learning
- PyTorch

In this slide, the material was sourced from Chapter2.