yunjey / **pytorch-tutorial**

Branch: master ▾ **pytorch-tutorial** / tutorials / 02-intermediate / convolutional_neural_network / **main.py** Find file Copy path


yunjey Update tutorials for pytorch 0.4.0 78c6afe on May 10, 2018

1 contributor

100 lines (82 sloc) 3.28 KB Raw Blame History

```
1  import torch
2  import torch.nn as nn
3  import torchvision
4  import torchvision.transforms as transforms
5
6
7  # Device configuration
8  device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
9
10 # Hyper parameters
11 num_epochs = 5
12 num_classes = 10
13 batch_size = 100
14 learning_rate = 0.001
15
16 # MNIST dataset
17 train_dataset = torchvision.datasets.MNIST(root='../../data/',
18                                             train=True,
19                                             transform=transforms.ToTensor(),
20                                             download=True)
21
22 test_dataset = torchvision.datasets.MNIST(root='../../data/',
23                                           train=False,
24                                           transform=transforms.ToTensor())
25
26 # Data loader
27 train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
28                                            batch_size=batch_size,
29                                            shuffle=True)
30
31 test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
32                                           batch_size=batch_size,
33                                           shuffle=False)
34
35 # Convolutional neural network (two convolutional layers)
36 class ConvNet(nn.Module):
37     def __init__(self, num_classes=10):
38         super(ConvNet, self).__init__()
39         self.layer1 = nn.Sequential(
```

```
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7*7*32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out

model = ConvNet(num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))

# Test the model
model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
```

```
92          outputs = model(images)
93          _, predicted = torch.max(outputs.data, 1)
94          total += labels.size(0)
95          correct += (predicted == labels).sum().item()
96
97      print('Test Accuracy of the model on the 10000 test images: {} %'.format(100 * correct / total))
98
99  # Save the model checkpoint
100 torch.save(model.state_dict(), 'model.ckpt')
```