PyTorch Tutorial

Rajbir Kataria

09/17/2018

Overview

- Tensors
- Optimizers
- nn module
- Examples! Examples! Examples!
  - https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/01-basics/pytorch_basics/main.py

PyTorch Tensors

- Similar to numpy's ndarrays
  - Can also be used on a GPU
  - Lost of operations supported like '+' and '-'

```
# import pytorch
import torch

# define a tensor
torch.FloatTensor([2])
```

```
 2
[torch.FloatTensor of size 1]
```

Optim module

- Module that implements various optimization algorithms used for building neural networks
  - Most common methods are already supported

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

nn module

- Easy to define computational graphs and take gradients
  - nn package defines a set of modules that can be used to make a neural network

```
import torch


# define model
model = torch.nn.Sequential(
 torch.nn.Linear(input_num_units, hidden_num_units),
 torch.nn.ReLU(),
 torch.nn.Linear(hidden_num_units, output_num_units),
)
loss_fn = torch.nn.CrossEntropyLoss()
```

Loading data from numpy

```
# Create a numpy array.
x = np.array([[1, 2], [3, 4]])

# Convert the numpy array to a torch tensor.
y = torch.from_numpy(x)

# Convert the torch tensor to a numpy array.
z = y.numpy()
```

Autograd example

```
# Create tensors of shape (10, 3) and (10, 2).
x = torch.randn(10, 3)
y = torch.randn(10, 2)

# Build a fully connected layer.
linear = nn.Linear(3, 2)
print ('w: ', linear.weight)
print ('b: ', linear.bias)

# Build loss function and optimizer.
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(linear.parameters(), lr=0.01)

# Forward pass.
pred = linear(x)

# Compute loss.
loss = criterion(pred, y)
print('loss: ', loss.item())

# Backward pass.
loss.backward()

# Print out the gradients.
print ('dL/dw: ', linear.weight.grad)
print ('dL/db: ', linear.bias.grad)

# 1-step gradient descent.
optimizer.step()
```

Input pipeline

```
# Download and construct CIFAR-10 dataset.
train_dataset = torchvision.datasets.CIFAR10(root='../../data/',
                                             train=True,
                                             transform=transforms.ToTensor(),
                                             download=True)

# Fetch one data pair (read data from disk).
image, label = train_dataset[0]

# Data loader (this provides queues and threads in a very simple way).
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=64,
                                           shuffle=True)

# When iteration starts, queue and thread start to load data from files.
data_iter = iter(train_loader)

# Mini-batch images and labels.
images, labels = data_iter.next()

# Actual usage of the data loader is as below.
for images, labels in train_loader:
    # Training code should be written here.
    pass
```

Input pipeline for custom dataset

```
# You should your build your custom dataset as below.
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self):
        # 1. Initialize file paths or a list of file names.
        pass

    def __getitem__(self, index):
        # 1. Read one data from file (e.g. using numpy.fromfile, PIL.Image.open).
        # 2. Preprocess the data (e.g. torchvision.Transform).
        # 3. Return a data pair (e.g. image and label).
        pass

    def __len__(self):
        # You should change 0 to the total size of your dataset.
        return 0

# You can then use the prebuilt data loader.
custom_dataset = CustomDataset()
train_loader = torch.utils.data.DataLoader(dataset=custom_dataset,
                                           batch_size=64,
                                           shuffle=True)
```

GPU/CPU interaction

```
# Specify device to use
device = torch.device("cuda" if use_cuda else "cpu")

# Use the specific device
model = Net().to(device)

# Put data on the device and pass it to the model
for batch_idx, (data, target) in enumerate(train_loader):
    data, target = data.to(device), target.to(device)
    optimizer.zero_grad()
    output = model(data)

    # Convert output to CPU and print
    print (output.data.cpu().numpy())
```

Pretrained model

```
# Download and load the pretrained ResNet-18.
resnet = torchvision.models.resnet18(pretrained=True)

# If you want to finetune only the top layer of the model, set as below.
for param in resnet.parameters():
    param.requires_grad = False

# Replace the top layer for finetuning.
resnet.fc = nn.Linear(resnet.fc.in_features, 100)  # 100 is an example.

# Forward pass.
images = torch.randn(64, 3, 224, 224)
outputs = resnet(images)
print (outputs.size())     # (64, 100)
```

Save and load the model

```
# Save and load the entire model.
torch.save(resnet, 'model.ckpt')
model = torch.load('model.ckpt')

# Save and load only the model parameters (recommended).
torch.save(resnet.state_dict(), 'params.ckpt')
resnet.load_state_dict(torch.load('params.ckpt'))
```

References

- All examples from:
  - https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/01-basics/pytorch_basics/main.py
- Other resources:
  - https://www.analyticsvidhya.com/blog/2018/02/pytorch-tutorial/