Demo of Convolutional Neural Network on CIFAR10

David I. Inouye

Demo of Convolutional Neural Network in PyTorch on CIFAR10

Adapted from PyTorch tutorial from (skipping details):
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Load data (skipping details see tutorial for details)

```python
import torch
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True,
    download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=4,
    shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(
    root='./data', train=False,
    download=True, transform=transform)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=4,
    shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Show some images from CIFAR10

```python
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```


```
horse horse truck  frog
```

Define a Convolutional Neural Network

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

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # nn.Conv2d(in_channels, out_channels, kernel_size)
        self.conv1 = nn.Conv2d(3, 6, 5)
        # nn.MaxPool2d(kernel_size, stride)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # nn.Linear(in_features, out_features)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Input is (N, 3, 32, 32)
        x = F.relu(self.conv1(x)) # (N, 6, 28, 28)
        x = self.pool(x)  # (N, 6, 14, 14)
        x = F.relu(self.conv2(x)) # (N, 16, 10, 10)
        x = self.pool(x)  # (N, 16, 5, 5)
        x = x.view(-1, 16 * 5 * 5) # (N, 400)
        x = F.relu(self.fc1(x))  # (N, 120)
        x = F.relu(self.fc2(x))  # (N, 84)
        x = self.fc3(x)  # (N, 10)
        return x
```

Parameters of layers

- `torch.nn.Conv2d` and similar functions produce object that automatically registers its parameters inside the `torch.nn.Module`
- Thus, when calling `model.parameters()`, it will include these parameters
- Note that simple ReLU and maxpool functions do not have parameters

```
# Remember convolution weight has size (out_channels, in_chan
total_num_params = 0
for name, p in net.named_parameters():
    total_num_params += p.numel() # Number of elements
    print(name, ',', p.size(), type(p))
print(f'Total number of parameters: {total_num_params}')
```

```
conv1.weight , torch.Size([6, 3, 5, 5]) <class
'torch.nn.parameter.Parameter'>
conv1.bias , torch.Size([6]) <class
'torch.nn.parameter.Parameter'>
conv2.weight , torch.Size([16, 6, 5, 5]) <class
'torch.nn.parameter.Parameter'>
conv2.bias , torch.Size([16]) <class
'torch.nn.parameter.Parameter'>
fc1.weight , torch.Size([120, 400]) <class
'torch.nn.parameter.Parameter'>
fc1.bias , torch.Size([120]) <class
'torch.nn.parameter.Parameter'>
fc2.weight , torch.Size([84, 120]) <class
'torch.nn.parameter.Parameter'>
fc2.bias , torch.Size([84]) <class
'torch.nn.parameter.Parameter'>
fc3.weight , torch.Size([10, 84]) <class
```

Define a Loss function and optimizer

Let's use a Classification Cross-Entropy loss and SGD with momentum.

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(
    net.parameters(), lr=0.001, momentum=0.9)
```

Train the network

This is when things start to get interesting. We simply have to loop over our data iterator, and feed the inputs to the network and optimize.

```python
for epoch in range(2):  # Loop over dataset
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:  # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
print('Finished Training')
```

```
[1,  2000] loss: 2.238
[1,  4000] loss: 1.945
[1,  6000] loss: 1.730
[1,  8000] loss: 1.614
[1, 10000] loss: 1.555
[1, 12000] loss: 1.493
[2,  2000] loss: 1.432
[2,  4000] loss: 1.396
[2,  6000] loss: 1.377
[2,  8000] loss: 1.344
[2, 10000] loss: 1.323
[2, 12000] loss: 1.290
Finished Training
```

Saving and loading trained model

- Note: If you want to restart training, you also need to save the optimizer states.

```
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)

# Load the model from saved path
net = Net()
net.load_state_dict(torch.load(PATH))
```

```
<All keys matched successfully>
```

See here <https://pytorch.org/docs/stable/notes/serialization.html>_ for more details on saving PyTorch models.

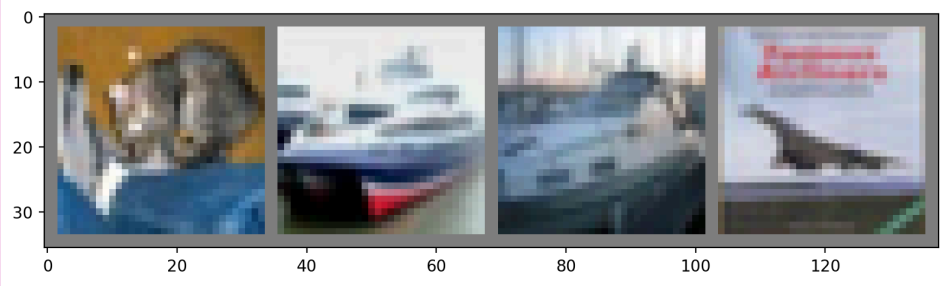
Check the network on the test data

```
dataiter = iter(testloader)
images, labels = next(dataiter)

# Show images with ground truth
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] fo

# Predict outputs
outputs = net(images)
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```




```
GroundTruth:    cat  ship  ship plane
Predicted:    cat   car   car  ship
```

Evaluate on test dataset

```python
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(
    'Accuracy of the network on the 10000 test images: %d %%'
    % (100 * correct / total)
)
```

```
Accuracy of the network on the 10000 test images: 55 %
```

- Note that 10% is random chance.

Let's evaluate per-class accuracies to see which ones performed well

```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 70 %
Accuracy of   car : 71 %
Accuracy of  bird : 27 %
Accuracy of   cat : 27 %
Accuracy of  deer : 53 %
Accuracy of   dog : 52 %
Accuracy of  frog : 79 %
Accuracy of horse : 57 %
Accuracy of  ship : 56 %
Accuracy of truck : 55 %
```

What about running on GPUs?

```
device = torch.device("cuda:0" if torch.cuda.is_available() e

# Assuming that we are on a CUDA machine, this should print a
print(device) # Would print GPU if available.

# This would move all model parameters to device`
net.to(device)

# You would also have to move inputs and target to device
data = next(iter(trainloader))
inputs, labels = data[0].to(device), data[1].to(device)
```

```
cpu
```