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
1  import torch
2  import torchvision
3  import torchvision.transforms as transforms
4
5  transform = transforms.Compose(
6      [transforms.ToTensor(),
7       transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
8
9  trainset = torchvision.datasets.CIFAR10(
10     root='./data', train=True,
11     download=True, transform=transform)
12 trainloader = torch.utils.data.DataLoader(
13     trainset, batch_size=4,
14     shuffle=True, num_workers=2)
15
16 testset = torchvision.datasets.CIFAR10(
17     root='./data', train=False,
18     download=True, transform=transform)
19 testloader = torch.utils.data.DataLoader(
20     testset, batch_size=4,
21     shuffle=False, num_workers=2)
22
23 classes = ('plane', 'car', 'bird', 'cat',
24            '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)))
```
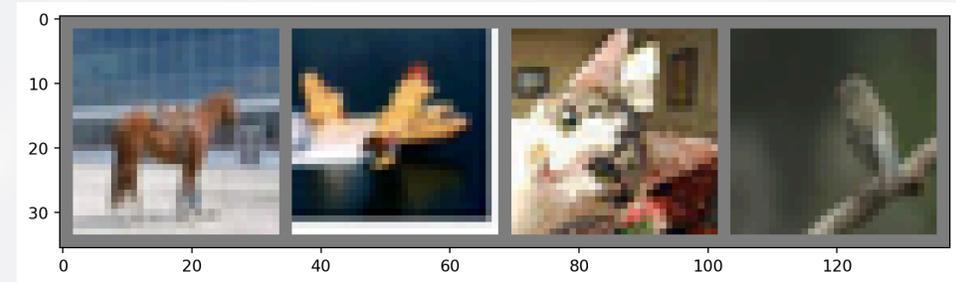

horse plane   cat bird

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

```python
1  # Remember convolution weight has size (out_channels, in_chan
2  total_num_params = 0
3  for name, p in net.named_parameters():
4      total_num_params += p.numel() # Number of elements
5      print(name, ',', p.size(), type(p))
6  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.

```python
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
import os
PATH = './cifar_net.pth'
if os.path.isfile(PATH):
    net = Net()
    net.load_state_dict(torch.load(PATH))
    print('Loaded saved model, skipping training')
else:
    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-bat
                print('[%d, %5d] loss: % 3f' %
```

```
Loaded saved model, skipping training
```

Saving and loading trained model

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

```
1  torch.save(net.state_dict(), PATH)
2
3  # Load the model from saved path
4  net = Net()
5  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

```python
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)))
```
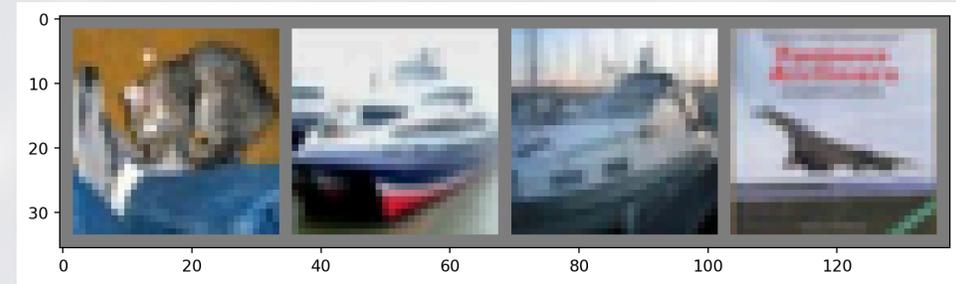


```
GroundTruth:     cat  ship  ship plane
Predicted:     frog plane  ship  ship
```

Evaluate on test dataset

```python
1  correct = 0
2  total = 0
3  with torch.no_grad():
4      for data in testloader:
5          images, labels = data
6          outputs = net(images)
7          _, predicted = torch.max(outputs.data, 1)
8          total += labels.size(0)
9          correct += (predicted == labels).sum().item()
10
11 print(
12     'Accuracy of the network on the 10000 test images: %d %%'
13     % (100 * correct / total)
14 )
```

```
Accuracy of the network on the 10000 test images: 53 %
```

- Note that 10% is random chance.

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

```python
1  class_correct = list(0. for i in range(10))
2  class_total = list(0. for i in range(10))
3  with torch.no_grad():
4      for data in testloader:
5          images, labels = data
6          outputs = net(images)
7          _, predicted = torch.max(outputs, 1)
8          c = (predicted == labels).squeeze()
9          for i in range(4):
10             label = labels[i]
11             class_correct[label] += c[i].item()
12             class_total[label] += 1
13
14
15 for i in range(10):
16     print('Accuracy of %5s : %2d %%' % (
17         classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 56 %
Accuracy of   car : 76 %
Accuracy of  bird : 63 %
Accuracy of   cat : 34 %
Accuracy of  deer : 13 %
Accuracy of   dog : 43 %
Accuracy of  frog : 63 %
Accuracy of horse : 56 %
Accuracy of  ship : 67 %
Accuracy of truck : 54 %
```

What about running on GPUs?

```
 1  device = torch.device("cuda:0" if torch.cuda.is_available() e
 2
 3  # Assuming that we are on a CUDA machine, this should print a
 4  print(device) # Would print GPU if available.
 5
 6  # This would move all model parameters to device`
 7  net.to(device)
 8
 9  # You would also have to move inputs and target to device
10  data = next(iter(trainloader))
11  inputs, labels = data[0].to(device), data[1].to(device)
```

```
cpu
```