```
%matplotlib inline
```

Training a Classifier

This is it. You have seen how to define neural networks, compute loss and make updates to the weigh

Now you might be thinking,

What about data?

Generally, when you have to deal with image, text, audio or video data, you can use standard python pa
Then you can convert this array into a `torch.*Tensor`.

- For images, packages such as Pillow, OpenCV are useful
- For audio, packages such as scipy and librosa
- For text, either raw Python or Cython based loading, or NLTK and SpaCy are useful

Specifically for vision, we have created a package called `torchvision`, that has data loaders for com
CIFAR10, MNIST, etc. and data transformers for images, viz., `torchvision.datasets` and `torch.ut`

This provides a huge convenience and avoids writing boilerplate code.

For this tutorial, we will use the CIFAR10 dataset. It has the classes: ‘airplane’, ‘automobile’, ‘bird’, ‘cat’,
The images in CIFAR-10 are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels in size.

.. figure:: /_static/img/cifar10.png :alt: cifar10

cifar10

Training an image classifier

We will do the following steps in order:

1. Load and normalizing the CIFAR10 training and test datasets using `torchvision`
2. Define a Convolution Neural Network
3. Define a loss function
4. Train the network on the training data
5. Test the network on the test data
6. Loading and normalizing CIFAR10 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Using `torchvision`, it’s extremely easy to load CIFAR10.

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

The output of torchvision datasets are PILImage images of range [0, 1]. We transform them to Tensor

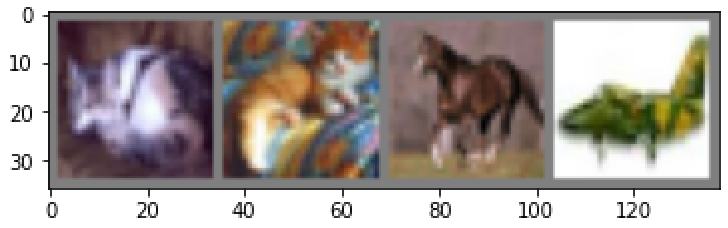
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/ci
99%                                169369600/170498071 [00:12<00:00, 16804263.89it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Let us show some of the training images, for fun.

```
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)))


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

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

```
  cat   cat horse plane
```



2. Define a Convolution Neural Network ^^ Copy the neural network from the N
   take 3-channel images (instead of 1-channel images as it was defined).

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


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

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

3. Define a Loss function and optimizer ^^ Let's use a Classification Cross-En

```
import torch.optim as optim

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

4. Train the network ^^^^^^^^^^^^^^^^^^^^

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

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        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: 2.208
[1,  4000] loss: 1.854
[1,  6000] loss: 1.669
[1,  8000] loss: 1.563
[1, 10000] loss: 1.507
[1, 12000] loss: 1.433
[2,  2000] loss: 1.384
[2,  4000] loss: 1.349
[2,  6000] loss: 1.329
[2,  8000] loss: 1.316
[2, 10000] loss: 1.302
[2, 12000] loss: 1.259
Finished Training
```

5. Test the network on the test data ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

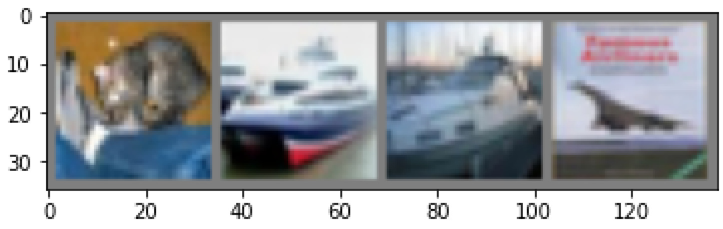
We have trained the network for 2 passes over the training dataset. But we need to check if the netwo

We will check this by predicting the class label that the neural network outputs, and checking it agains correct, we add the sample to the list of correct predictions.

Okay, first step. Let us display an image from the test set to get familiar.

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

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

```
GroundTruth:    cat  ship  ship plane
```



Okay, now let us see what the neural network thinks these examples above are:

```
outputs = net(images)
```

The outputs are energies for the 10 classes. Higher the energy for a class, the more the network think
So, let's get the index of the highest energy:

```
_, predicted = torch.max(outputs, 1)

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

```
Predicted:    cat  ship plane plane
```

The results seem pretty good.

Let us look at how the network performs on the whole dataset.

```
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 %
```

That looks waaay better than chance, which is 10% accuracy (randomly picking a class out of 10 clas
something.

Hmmm, what are the classes that performed well, and the classes that did not perform 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 : 72 %
Accuracy of   car : 62 %
Accuracy of  bird : 26 %
Accuracy of   cat : 25 %
Accuracy of  deer : 63 %
Accuracy of   dog : 45 %
Accuracy of  frog : 68 %
Accuracy of horse : 74 %
Accuracy of  ship : 62 %
Accuracy of truck : 50 %
```

Okay, so what next?

How do we run these neural networks on the GPU?

Training on GPU

Just like how you transfer a Tensor on to the GPU, you transfer the neural net onto the GPU.

Let's first define our device as the first visible cuda device if we have CUDA available:

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

# Assume that we are on a CUDA machine, then this should print a CUDA device:

print(device)
```

```
cpu
```

The rest of this section assumes that `device` is a CUDA device.

Then these methods will recursively go over all modules and convert their parameters and buffers to

.. code:: python

```
 net.to(device)
```

Remember that you will have to send the inputs and targets at every step to the GPU too:

.. code:: python

```
    inputs, labels = inputs.to(device), labels.to(device)
```

Why dont I notice MASSIVE speedup compared to CPU? Because your network is realllly small.

Exercise: Try increasing the width of your network (argument 2 of the first `nn.Conv2d`, and argument
to be the same number), see what kind of speedup you get.

Goals achieved:

- Understanding PyTorch's Tensor library and neural networks at a high level.
- Train a small neural network to classify images

Training on multiple GPUs

If you want to see even more MASSIVE speedup using all of your GPUs, please check out :doc: `data_`

Where do I go next?

- :doc: `Train neural nets to play video games </intermediate/reinforcement_q_lear`
- `Train a state-of-the-art ResNet network on imagenet`_
- `Train a face generator using Generative Adversarial Networks`_
- `Train a word-level language model using Recurrent LSTM networks`_
- `More examples`_
- `More tutorials`_
- `Discuss PyTorch on the Forums`_
- `Chat with other users on Slack`_