Tutorial: Training a Classifier

Friday, July 17, 2020 4:32 PM

Data

You can use *standard* *python* *packages* that load data into a ***numpy array***

- Then you can **convert** this array into a torch.*Tensor

- *Python Packages*:
- For **images** => can use *Pillow* and *OpenCV*
- For **audio** => can use scipy and librosa
- For **text** => can use:
  - either ***raw*** *Python or Cython* ***based loading***
  - or *NLTK* and *SpaCy*

For **vision**, use the **package** => torchvision

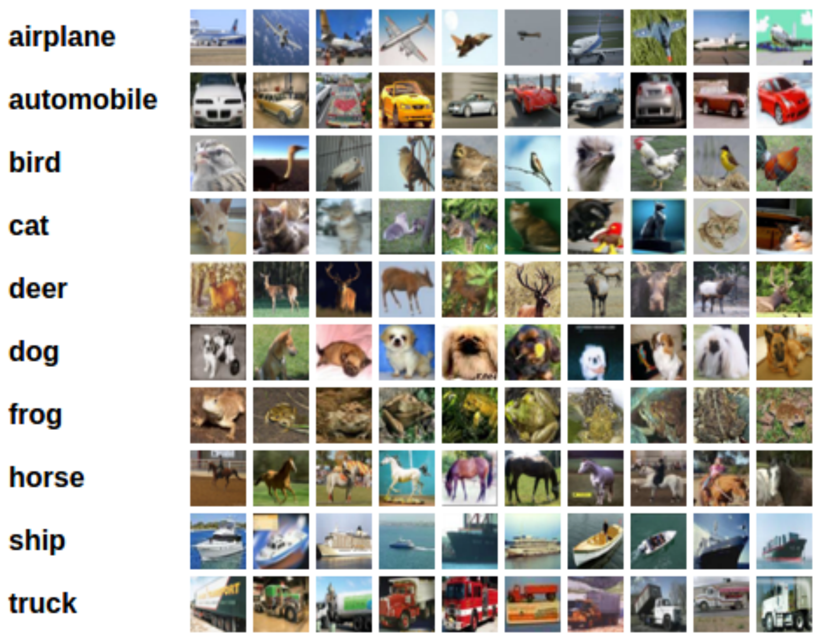
- It has data loaders for common datasets such as ***Imagenet***, ***CIFAR10***, ***MNIST***, etc. and data transformers for images,visualizations
  - torchvision.datasets and torch.utils.data.DataLoader

------ ------ ------ ------ ------ ------ ------ ------ ------ ------ ------

For this tutorial

Tutorial Link:
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py

Uses the ***CIFAR10*** dataset:
- It has the **classes**: ‘airplane’, ‘automobile’, ‘bird’, ‘cat’, ‘deer’, ‘dog’, ‘frog’, ‘horse’, ‘ship’, ‘truck’.
- The ***images*** *in* ***CIFAR-10*** are of **size** 3x32x32
  - *Example:*
    - **3-channel color** images of **32**x**32** ***pixels*** in **size**.



Steps for 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 *Convolutional Neural Network* (***CNN***)

3. **Define** a ***loss*** function

4. **Train** the **network** on the *training* **data**

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

------ ------ ------ ------ ------ ------ ------ ------ ------ ------ ------

1. Loading and normalizing CIFAR10

The **output** of *torchvision datasets* are **PILImage** images of **range [0, 1]**.

- Need to **transform them to Tensors** of normalized **range [-1, 1]**.

STEPS:

a. **Create** the **transform function**.
   - This ***creates tensors*** and ***normalizes*** the data.

b. **Get** the **training** set **data** from torchvision.datasets.
   - Set argument *train=True*
   - Set transform function to transform argument

c. **Pass** the **training set** to the **DataLoader** using torch.utils.data.DataLoader.
   - This adds the *batch size* and *num_workers*, etc.

d. **Repeat** *step b and c* for **test set**.

e. **Create** the **list of classes** for the dataset as a *python tuple.*

------ ------ ------ ------ ------ ------ ------ ------ ------ ------ ------

2. Define a Convolutional Neural Network

- *Example:*
  - **Takes 3 channels images**

```
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

USE:

- **Loss function** => ***Classification Cross-Entropy loss***
- **Optimizer** => ***SGD with momentum***
    - *Updates* the *weights*
    - *Need to* ***set a learning rate*** *(<u>example</u>:* `lr=0.001`*)*
    - *Need to* ***set momentum*** *(<u>example</u>:* `momentum=0.9`*)*
        - ???

------ ------ ------ ------ ------ ------ ------ ------ ------ ------ ------

4. Train the Network

Model **loops** over the data iterator, and **feeds the inputs to the network** and **optimize**.

- *Each loop* is called an **epoch**

- **GOAL** is to **lower the loss**.
    - By **mini batches**. (*<u>example</u>:* loss for every 2000 samples)
- Then **save** the **trained model**.
    - *More details on saving PyTorch models:*
        - https://pytorch.org/docs/stable/notes/serialization.html
    - *<u>Example:</u>*

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

------ ------ ------ ------ ------ ------ ------ ------ ------ ------ ------

5. Test the network on the test data

Need to **check if the network** has **learnt** anything at all.

A. By **predicting the class label** that the neural network **outputs**,

B. and by **checking it against the ground-truth** (*test dataset*)

- If the prediction is correct, we ***add the sample to the list of correct predictions***.

STEPS:

1. Check and **view a few images** and **corresponding labels** from the *test dataset*.

2. **Load** back **saved model.**

- *Note:*
    - **Saving** and **re-loading the model** is only necessary is its going be open elsewhere:

```
net = Net()
net.load_state_dict(torch.load(PATH))
```

3. **Run testing images** through model.

4. **Get** the **index** of the **highest *energy***:

    - The **outputs are *energies*** for the 10 **classes**.
    - The higher the energy for a class, the more the network thinks that the image is of the particular class.

5. **Look at how** the network performs on the **whole dataset** by checking the *accuracy*.

6. **Check** the performance (***accurancy***) of every class.

Training on GPU

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

- 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")
# Assuming that we are on a CUDA machine, this should print a CUDA device:
print(device)
```

➢ **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 CUDA tensors*:

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

- You **will have to send the inputs** and **targets at every step** to the GPU too:

```
inputs, labels = data[0].to(device), data[1].to(device)
```

- To **increase the speedup** on the model:
    - Try increasing the width of the network.
        - By changing **argument 2** of the **first** nn.Conv2d,
        - and **argument 1** of the **second** nn.Conv2d ,
        - they **need to be the same number**!

------ ------ ------ ------ ------ ------ ------ ------ ------ ------ ------

Training on multiple GPUs

For **more** even more speedup **using all of your GPUs**, please check out:

- doc: data_parallel_tutorial