CAP5415
Computer Vision

Yogesh S Rawat
yogesh@ucf.edu
HEC-241
PyTorch Tutorial - I

Lecture 8
Deep learning libraries

- Torch (Lua):
  - http://torch.ch/

- PyTorch (Python)
  - http://pytorch.org/

- TensorFlow (Python and C++):
  - https://www.tensorflow.org/

- Theano (Python)
  - http://deeplearning.net/software/theano/

- Keras
  - https://keras.io/
PyTorch Tensor

• Similar to NumPy arrays
• They can also be used on a GPU
  • Faster computation
• Random matrix

```python
import torch
x = torch.rand(2, 3)
y = torch.rand(3, 3)
print x
print y
```
PyTorch Tensor

• Similar to NumPy arrays
• They can also be used on a GPU
  • Faster computation
• All zeros
• Directly from data
• Size of a tensor

import torch
x = torch.zeros(5, 3)
x = torch.tensor([5.5, 3])
print x.size()
Operations

- Adding tensors
- Indexing

```python
x = torch.randn(4, 4)
y = torch.randn(4, 4)
print(torch.add(x, y))
print(x[:, 1])
```
Operations

• Resizing
  • If you want to resize/reshape tensor

```python
x = torch.randn(4, 4)
y = x.view(16)
z = x.view(-1, 8)

print(x.size(), y.size(), z.size())
```

**Output:**
- `torch.Size([4, 4])`
- `torch.Size([16])`
- `torch.Size([2, 8])`
Pop Quiz

Send private message.

Message **to all** will not be considered!
Reshaping tensor

```python
x = torch.randn(1, 4, 32, 24)

y = x.view(8, 2, -1, 3, 8)

print(y.size())
```

*Output shape? 30 seconds!*
Reshaping tensor

```python
x = torch.randn(1, 4, 32, 24)

y = x.view(8, 2, -1, 3, 8)

print(y.size())
```

*Output shape: (8, 2, 8, 3, 8)*
Torch tensor vs NumPy array

• NumPy array
  • CPU
• Torch tensor
  • GPU

```python
a = torch.ones(5)
tensor([1., 1., 1., 1., 1.])

b = a.numpy()

a = numpy.ones(5)
b = torch.from_numpy(a)
```
Matrix Multiplication in PyTorch

```python
import torch

mat1=torch.randn(2,3)
mat2=torch.randn(3,3)
res=torch.mm(mat1,mat2)

print(res.size())

Output:
(2L, 3L)
```
Batch Matrix Multiplication in PyTorch

```python
import torch

batch1 = torch.randn(10, 3, 4)
batch2 = torch.randn(10, 4, 5)
res = torch.bmm(batch1, batch2)

print(res.size())
```

Output:
(10L, 3L, 5L)
Many Tensor operations in PyTorch...

torch.mm
• Matrix multiplication
torch.bmm
• Batch matrix multiplication
torch.cat
• Tensor Concatenation
torch.squeeze/torch.unsqueeze
• Change Tensor dimensions
...

Check documentation at http://pytorch.org/docs/master/torch.html#tensors
Computational Graphs

import torch

x = torch.ones(2,2)
y = torch.ones(2,1)
w = torch.randn(2,1,requires_grad=True)
b = torch.randn(1,requires_grad=True)
Computational Graphs

\[ p = \text{torch.sigmoid}(\text{torch.mm}(x, w) + b) \]  
# prediction

\[ \text{loss} = -y \times \text{torch.log}(p) - (1-y) \times \text{torch.log}(1-p) \]  
# cross-entropy loss

\[ \text{cost} = \text{loss.mean()} \]  
# the cost to minimize
Automatic Gradient Computation

\[ p = \text{torch.sigmoid}(\text{torch.mm}(x, w) + b) \]
\[ \text{loss} = -y * \text{torch.log}(p) - (1-y) * \text{torch.log}(1-p) \]
\[ \text{cost} = \text{loss}.\text{mean}() \]

\[ \text{cost}.\text{backward}() \]

print w.grad
print b.grad
Training procedure

• Define the neural network
• Iterate over a dataset of inputs
• Process input through the network
• Compute the loss
• Propagate gradients back into the network’s parameters
• Update the weights of the network
Training procedure

• Define the neural network
  • Iterate over a dataset of inputs
  • Process input through the network
  • Compute the loss
  • Propagate gradients back into the network’s parameters
  • Update the weights of the network
Build Neural Networks using PyTorch

Neural networks can be constructed using the torch.nn package.

Forward
• An nn.Module contains layers, and
• A method forward(input) that returns the output
• You can use any of the Tensor operations in the forward function

Backward
• nn depends on autograd
• You just have to define the forward function
Define a Network Class

```python
import torch
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # create layers

    def forward(self, x):
        # define feed-forward function

You don’t need to define a backward function!
```
CNN for MNIST: A Full Example

Define a CNN Network

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
Define a CNN Network

def __init__(self):
    super(Net, self).__init__()
    # 1 input image channel, 6 output channels, 3x3 square convolution kernel
    self.conv1 = nn.Conv2d(1, 6, 3)
    self.conv2 = nn.Conv2d(6, 16, 3)
    # an affine operation: y = Wx + b
    self.fc1 = nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)
Define a CNN Network

def forward(self, x):
    # Max pooling over a (2, 2) window
    x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
    # If the size is a square you can only specify a single number
    x = F.max_pool2d(F.relu(self.conv2(x)), 2)
    x = x.view(-1, self.num_flat_features(x))
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)

    return x
Define a CNN Network

```python
def num_flat_features(self, x):
    size = x.size()[1:]  # all dimensions except the batch dimension
    num_features = 1
    for s in size:
        num_features *= s

    return num_features
```
Training procedure

- Define the neural network
- Iterate over a dataset of inputs
  - Process input through the network
  - Compute the loss
  - Propagate gradients back into the network’s parameters
  - Update the weights of the network
Data

• For images
  • Pillow, OpenCV are useful

• For audio
  • Scipy and librosa

• For text
  • NLTK and SpaCy are useful

• Load data into memory as NumPy array
  • Then convert to tensor for GPU
Loading data - torchvision

• Torchvision
  • it’s extremely easy to load existing datasets.

import torchvision
import torchvision.transforms as transforms
Loading data - torchvision

```python
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)
```
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))])

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)
Questions?

Sources for this lecture include materials from pytorch.org