Pytorch tutorial

January 28, 2018

1 Introduction a PyTorch

based on https://github.com/mila-udem/welcome_tutorials/tree/master/pytorch By Sandeep Subramanian

https://github.com/jcjohnson/pytorch-examples

In [1]:
```
import torch
import numpy as np
from torch.autograd import Variable

torch.from_numpy(np.array([1,2,3,4]))
```

Out[1]:
```
 1
 2
 3
 4
[torch.LongTensor of size 4]
```

In [2]:
```
torch.LongTensor([1,2,3,4])
torch.Tensor([1,2,3,4]).type(torch.LongTensor)
```

Out[2]:
```
 1
 2
 3
 4
[torch.LongTensor of size 4]
```

In [3]:
```
x = torch.Tensor([1,2,3,4])
```

In [4]:
```
x.sqrt()
```

Out[4]:
```
 1.0000
 1.4142
 1.7321
 2.0000
[torch.FloatTensor of size 4]
```

```
In [5]: x = x.view(-1, 1)

In [6]: torch.cat([x, x], 1)

Out[6]:
         1  1
         2  2
         3  3
         4  4
        [torch.FloatTensor of size 4x2]
```

1.1 Autograd

```
In [7]: x = Variable(x, requires_grad=True)
        x

Out[7]: Variable containing:
         1
         2
         3
         4
        [torch.FloatTensor of size 4x1]

In [8]: print(x.grad)

None


In [9]: y = x + 2
        z = (3*y*y)
        print(z)

Variable containing:
  27
  48
  75
 108
[torch.FloatTensor of size 4x1]



In [10]: z.mean().backward()

In [11]: print(x.grad)

Variable containing:
 4.5000
 6.0000
 7.5000
 9.0000
[torch.FloatTensor of size 4x1]
```

1.2 Optim and loss function

```
In [12]: optimizer = torch.optim.Adam([x], lr=0.01)

In [13]: optimizer.step()

In [14]: print(x)

Variable containing:
 0.9900
 1.9900
 2.9900
 3.9900
[torch.FloatTensor of size 4x1]



In [15]: loss_fn = torch.nn.MSELoss()
```

2 Fashion MNIST

based on MNIST tutorial: https://github.com/pytorch/examples/blob/master/mnist/main.py

```
In [16]: %matplotlib inline
         import matplotlib.pyplot as plt
         import torch
         from torch.autograd import Variable
         import torchvision
         import torch.nn.functional as F
         from fashion import FashionMNIST
         import torchvision.transforms as transforms
         from torch import nn
         from torch import optim

In [17]: train_data = FashionMNIST('../data', train=True, download=True,
                                   transform=transforms.Compose([
                                       transforms.ToTensor(),
                                       transforms.Normalize((0.1307,), (0.3081,))
                                   ]))

         valid_data = FashionMNIST('../data', train=True, download=True,
                                   transform=transforms.Compose([
                                       transforms.ToTensor(),
                                       transforms.Normalize((0.1307,), (0.3081,))
                                   ]))

In [18]: train_idx = np.random.choice(train_data.train_data.shape[0], 54000, replace=False)

In [19]: train_data.train_data = train_data.train_data[train_idx, :]
         train_data.train_labels = train_data.train_labels[torch.from_numpy(train_idx).type(to
```

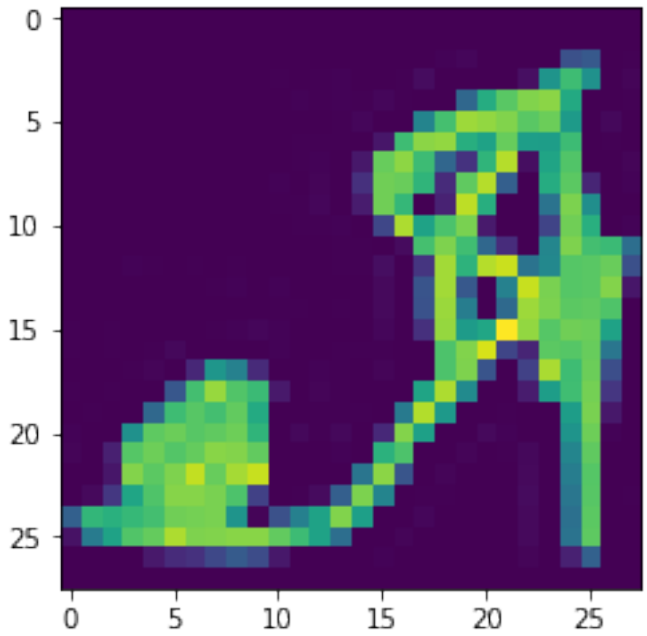
In [20]:
```
mask = np.ones(60000)
mask[train_idx] = 0
```

In [21]:
```
valid_data.train_data = valid_data.train_data[torch.from_numpy(np.argwhere(mask)), :]
valid_data.train_labels = valid_data.train_labels[torch.from_numpy(mask).type(torch.By
```

In [22]:
```
batch_size = 100
test_batch_size = 100

train_loader = torch.utils.data.DataLoader(train_data,
    batch_size=batch_size, shuffle=True)

valid_loader = torch.utils.data.DataLoader(valid_data,
    batch_size=batch_size, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    FashionMNIST('../data', train=False, transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ])),
    batch_size=test_batch_size, shuffle=True)
```

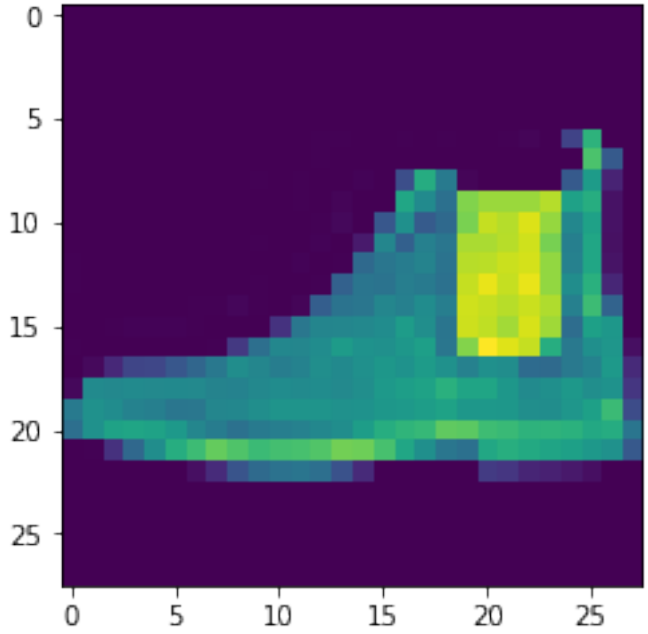
In [23]:
```
plt.imshow(train_loader.dataset.train_data[1].numpy())
```

Out[23]:
```
<matplotlib.image.AxesImage at 0x114594ef0>
```



In [24]:
```
plt.imshow(train_loader.dataset.train_data[10].numpy())
```

Out[24]: <matplotlib.image.AxesImage at 0x11474a7b8>




```
In [25]: class FcNetwork(nn.Module):
             def __init__(self):
                 super().__init__()
                 self.fc1 = nn.Linear(28*28, 512)
                 self.fc2 = nn.Linear(512, 10)

             def forward(self, image):
                 batch_size = image.size()[0]
                 x = image.view(batch_size, -1)
                 x = F.sigmoid(self.fc1(x))
                 x = F.log_softmax(self.fc2(x), dim=0)
                 return x

In [26]: model = FcNetwork()
         optimizer = optim.SGD(model.parameters(), lr=0.01)

In [27]: def train(epoch):
             model.train()
             for batch_idx, (data, target) in enumerate(train_loader):
                 data, target = Variable(data), Variable(target)
                 optimizer.zero_grad()
                 output = model(data)
                 loss = F.nll_loss(output, target)
                 loss.backward()
```

```
                optimizer.step()


        def test(loader, name):
            model.eval()
            test_loss = 0
            correct = 0
            for data, target in loader:
                data, target = Variable(data, volatile=True), Variable(target)
                output = model(data)
                test_loss += F.nll_loss(output, target, size_average=False).data[0] # sum up
                pred = output.data.max(1, keepdim=True)[1] # get the index of the max log-pro
                correct += pred.eq(target.data.view_as(pred)).cpu().sum()

            test_loss /= len(loader.dataset)
            print('\n' + name + ' set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.form
                test_loss, correct, len(loader.dataset),
                100. * correct / len(loader.dataset)))

        epochs = 5
        for epoch in range(1, epochs + 1):
            train(epoch)
            test(valid_loader, 'valid')

        test(test_loader, 'test')
```

```
valid set: Average loss: 3.3339, Accuracy: 4418/6000 (74%)


valid set: Average loss: 3.1786, Accuracy: 4540/6000 (76%)


valid set: Average loss: 3.1068, Accuracy: 4657/6000 (78%)


valid set: Average loss: 3.0620, Accuracy: 4758/6000 (79%)


valid set: Average loss: 3.0288, Accuracy: 4786/6000 (80%)


test set: Average loss: 3.0480, Accuracy: 7844/10000 (78%)
```

In [ ]:

In [ ]:

In [ ]: