udacity_mnist

September 16, 2018

In [75]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

from collections import OrderedDict

import numpy as np
import time

import torch
from torch import nn
from torch import optim
import torch.nn.functional as F

import helper

import matplotlib.pyplot as plt
```

In [76]:
```
from torchvision import datasets, transforms

# Define a transform to normalize the data
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                               ])
# Download and load the training data
trainset = datasets.MNIST('MNIST_data/', download=False, train=True, transform=transf
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
```

In [77]:
```
print(len(trainset))
```

```
60000
```

In [78]:
```
# Hyperparameters for our network
input_size = 784
hidden_sizes = [128, 64]
output_size = 10

# Build a feed-forward network
```

```
        model = nn.Sequential(OrderedDict([
                              ('fc1', nn.Linear(input_size, hidden_sizes[0])),
                              ('relu1', nn.ReLU()),
                              ('fc2', nn.Linear(hidden_sizes[0], hidden_sizes[1])),
                              ('relu2', nn.ReLU()),
                              ('logits', nn.Linear(hidden_sizes[1], output_size))]))
```

In [79]:
```
# class net(nn.Module):
#     def __init__(self):
#         super().__init__()
#         self.fc1 = nn.Linear(784,128)
#         self.fc2 = nn.Linear(128,64)
#         self.fc3 = nn.Linear(64,10)
#     def forward(self,x):
#         x = self.fc1(x)
#         x = F.relu(x)
#         x = self.fc2(x)
#         x = F.relu(x)
#         x = self.fc3(x)
#         out = F.softmax(x,dim=1)
#         return out
# model=net()
```

In [80]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.003)
```

In [81]:
```
epochs = 3
print_every = 40
steps = 0
for e in range(epochs):
    running_loss = 0
    for images, labels in iter(trainloader):
        steps += 1
        # Flatten MNIST images into a 784 long vector
        images.resize_(images.size()[0], 784)

        optimizer.zero_grad()

        # Forward and backward passes
        output = model.forward(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

        if steps % print_every == 0:
            print("Epoch: {}/{}... ".format(e+1, epochs),
```

```
                      "Loss: {:.4f}".format(running_loss/print_every))

                running_loss = 0
```

```
Epoch: 1/3...  Loss: 2.2980
Epoch: 1/3...  Loss: 2.2735
Epoch: 1/3...  Loss: 2.2506
Epoch: 1/3...  Loss: 2.2263
Epoch: 1/3...  Loss: 2.1986
Epoch: 1/3...  Loss: 2.1715
Epoch: 1/3...  Loss: 2.1425
Epoch: 1/3...  Loss: 2.1163
Epoch: 1/3...  Loss: 2.0907
Epoch: 1/3...  Loss: 2.0302
Epoch: 1/3...  Loss: 1.9894
Epoch: 1/3...  Loss: 1.9494
Epoch: 1/3...  Loss: 1.8836
Epoch: 1/3...  Loss: 1.8537
Epoch: 1/3...  Loss: 1.7815
Epoch: 1/3...  Loss: 1.7160
Epoch: 1/3...  Loss: 1.6712
Epoch: 1/3...  Loss: 1.5999
Epoch: 1/3...  Loss: 1.5396
Epoch: 1/3...  Loss: 1.4908
Epoch: 1/3...  Loss: 1.3939
Epoch: 1/3...  Loss: 1.3424
Epoch: 1/3...  Loss: 1.2688
Epoch: 2/3...  Loss: 0.6709
Epoch: 2/3...  Loss: 1.2037
Epoch: 2/3...  Loss: 1.1494
Epoch: 2/3...  Loss: 1.0792
Epoch: 2/3...  Loss: 1.0238
Epoch: 2/3...  Loss: 1.0283
Epoch: 2/3...  Loss: 0.9714
Epoch: 2/3...  Loss: 0.9519
Epoch: 2/3...  Loss: 0.8932
Epoch: 2/3...  Loss: 0.8527
Epoch: 2/3...  Loss: 0.8513
Epoch: 2/3...  Loss: 0.8306
Epoch: 2/3...  Loss: 0.7917
Epoch: 2/3...  Loss: 0.7989
Epoch: 2/3...  Loss: 0.7422
Epoch: 2/3...  Loss: 0.7502
Epoch: 2/3...  Loss: 0.7211
Epoch: 2/3...  Loss: 0.6943
Epoch: 2/3...  Loss: 0.6783
Epoch: 2/3...  Loss: 0.6856
Epoch: 2/3...  Loss: 0.6496
```

```
Epoch: 2/3...  Loss: 0.6488
Epoch: 2/3...  Loss: 0.6571
Epoch: 3/3...  Loss: 0.0650
Epoch: 3/3...  Loss: 0.6259
Epoch: 3/3...  Loss: 0.5910
Epoch: 3/3...  Loss: 0.6067
Epoch: 3/3...  Loss: 0.5553
Epoch: 3/3...  Loss: 0.5934
Epoch: 3/3...  Loss: 0.5802
Epoch: 3/3...  Loss: 0.5537
Epoch: 3/3...  Loss: 0.5575
Epoch: 3/3...  Loss: 0.5469
Epoch: 3/3...  Loss: 0.5377
Epoch: 3/3...  Loss: 0.5699
Epoch: 3/3...  Loss: 0.5013
Epoch: 3/3...  Loss: 0.5393
Epoch: 3/3...  Loss: 0.5283
Epoch: 3/3...  Loss: 0.5126
Epoch: 3/3...  Loss: 0.5260
Epoch: 3/3...  Loss: 0.4764
Epoch: 3/3...  Loss: 0.4812
Epoch: 3/3...  Loss: 0.5122
Epoch: 3/3...  Loss: 0.4650
Epoch: 3/3...  Loss: 0.4785
Epoch: 3/3...  Loss: 0.4754
Epoch: 3/3...  Loss: 0.4897
```

In [83]:
```
images, labels = next(iter(trainloader))

img = images[0].view(1, 784)
# Turn off gradients to speed up this part
with torch.no_grad():
    logits = model.forward(img)

# Output of the network are logits, need to take softmax for probabilities
ps = F.softmax(logits, dim=1)

plt.subplot(1,2,1)
plt.imshow(img.view(1,28,28).numpy().squeeze(),cmap='Greys_r')

plt.subplot(1,2,2)
plt.barh(range(10),ps.data.numpy().squeeze())
plt.xlim(0,1)
plt.ylim(-0.5,9.5)
plt.yticks(range(10))
plt.title('Prediction')
for x,y in enumerate(ps.data.numpy().squeeze()):
```

```
plt.text(y + 0.01, x-0.1, '{:.3f}'.format(y))
```