IE 345 - K “Introduction to Deep Learning: Fundamentals Concepts”

Prof. Yuzo

Resolution of the "Hello World" of Deep Learning with the MNIST dataset.

pg. 80 - 89

In [1]:

```
#Libraries

import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.autograd import Variable
```

In [2]:

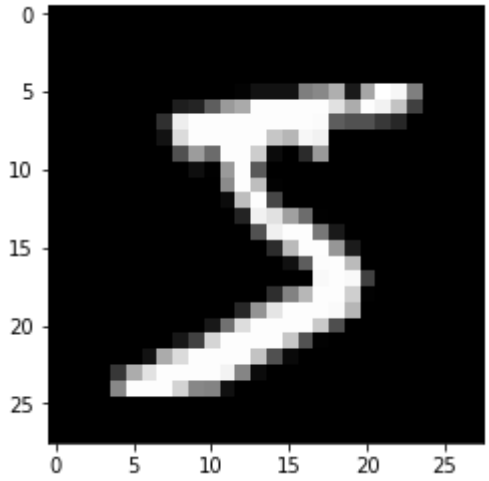
```
# Hyperparameters

input_size = 784
hidden_size =500
num_classes =10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
```

In [3]:

```
#MNIST dataset for training and test set

train_dataset = dsets.MNIST(root = './data',
                            train = True,
                            transform = transforms.ToTensor(),
                            download = True)

test_dataset = dsets.MNIST(root = './data',
                           train = False,
                          transform = transforms.ToTensor())
```

In [4]:

```
# Using Matplotlib

sample_image = torch.Tensor.numpy(train_dataset[0][0][0])
plt.imshow(np.reshape(sample_image, [28, 28]), cmap = 'gray')
plt.show()
```



In [5]:

```
#Load MNIST using data loader
train_loader = torch.utils.data.DataLoader(dataset = train_dataset,
                                           batch_size = batch_size,
                                           shuffle = True)

test_loader = torch.utils.data.DataLoader(dataset = test_dataset,
                                          batch_size = batch_size,
                                          shuffle = False)
```

In [6]:

```
# Define Artificial neural network

class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

In [7]:

```
# Initialize Neural Network Class
net = Net(input_size, hidden_size, num_classes)

# Create loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr = learning_rate)
```

In [8]:

```
# Train the model for num_epochs

for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = Variable(images.view(-1, 28*28))
        labels = Variable(labels)
        optimizer.zero_grad() # zero the gradient buffer
        outputs = net(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if(i+1) % 100 == 0:
            print('Epoch [%d/%d], Step[%d/%d], Loss: %.4f'
                  %(epoch+1, num_epochs, i+1,
                    len(train_dataset)//batch_size, loss.item()))
```

```
Epoch [1/5], Step[100/600], Loss: 0.2994
Epoch [1/5], Step[200/600], Loss: 0.2067
Epoch [1/5], Step[300/600], Loss: 0.2743
Epoch [1/5], Step[400/600], Loss: 0.2538
Epoch [1/5], Step[500/600], Loss: 0.1228
Epoch [1/5], Step[600/600], Loss: 0.2062
Epoch [2/5], Step[100/600], Loss: 0.0531
Epoch [2/5], Step[200/600], Loss: 0.0290
Epoch [2/5], Step[300/600], Loss: 0.2119
Epoch [2/5], Step[400/600], Loss: 0.1275
Epoch [2/5], Step[500/600], Loss: 0.0692
Epoch [2/5], Step[600/600], Loss: 0.1148
Epoch [3/5], Step[100/600], Loss: 0.0868
Epoch [3/5], Step[200/600], Loss: 0.0452
Epoch [3/5], Step[300/600], Loss: 0.0370
Epoch [3/5], Step[400/600], Loss: 0.1808
Epoch [3/5], Step[500/600], Loss: 0.0227
Epoch [3/5], Step[600/600], Loss: 0.0366
Epoch [4/5], Step[100/600], Loss: 0.1595
Epoch [4/5], Step[200/600], Loss: 0.0799
Epoch [4/5], Step[300/600], Loss: 0.0479
Epoch [4/5], Step[400/600], Loss: 0.0152
Epoch [4/5], Step[500/600], Loss: 0.0373
Epoch [4/5], Step[600/600], Loss: 0.0553
Epoch [5/5], Step[100/600], Loss: 0.0275
Epoch [5/5], Step[200/600], Loss: 0.1470
Epoch [5/5], Step[300/600], Loss: 0.0361
Epoch [5/5], Step[400/600], Loss: 0.0107
Epoch [5/5], Step[500/600], Loss: 0.1437
Epoch [5/5], Step[600/600], Loss: 0.0509
```

In [9]:

```
#Test the model

correct = 0
total = 0
for images, labels in test_loader:
    images = Variable(images.view(-1, 28*28))
    outputs = net(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted.cpu() == labels).sum()

print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct / tota
l))
```

```
Accuracy of the network on the 10000 test images: 97 %
```

NOTE: Installation of Pytorch

If you follow our step by step tutorial of installation, follow the next code for install Pytorch:

1. Open the Anaconda Prompt and activate the environment which you created with the code:
    - *activate tensorflow*
2. After activating your environment, type in the environment:
    - *conda install pytorch-cpu torchvision-cpu -c pytorch*

Instllation, tutorials, and examples with Pytorch visit: https://pytorch.org/get-started/locally/ (https://pytorch.org/get-started/locally/)

Pablo David Minango Negrete

$pablodavid218@gmail.com$