Machine Learning Lab 9

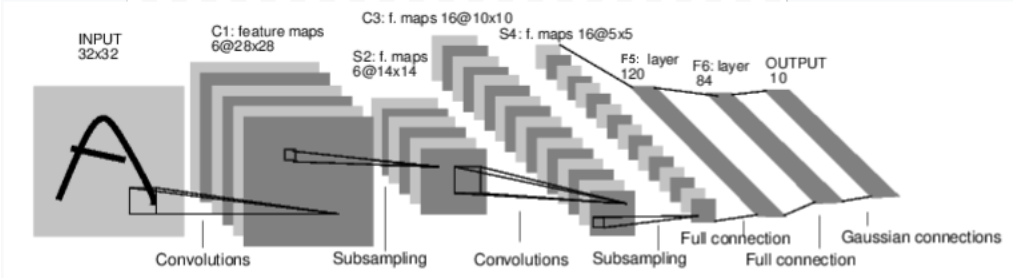
Pytorch and CNN

In this lab, you will use CNN with Pytorch to solve the problem of handwritten digit recognition. As for the dataset, we still use MNIST which we used in our previous labs. You will install Pytorch on your computer and learn how to develop your first deep learning based program. There is a Pytorch tutorial put together with this lab. You should study it first before finishing the following tasks.

1. Use Anaconda to install Pytorch package. To do so, you can open Anaconda Navigator and under the environment you use and search for Pytorch. Select Pytorch for installation. Note some of you may use pip for installation. In this case you should follow the pytorch official documentation to do the installation (see https://pytorch.org/get-started/locally/). You should select the build, your OS, Python version, package (pip) and CUDA version (none if Mac or windows laptop without Nvidia GPU). Then the command for pip will be generated for you to use. You should be able to run the following code if Pytorch is properly installed in your computer:

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

2. We will build LeNet[1], which is a simple CNN designed for MNIST, with Pytorch. The architecture is shown as follows:



LeNet contains 2 convolution layers, each of which is followed by a ReLU and

[1] http://yann.lecun.com/exdb/lenet/index.html

pooling layer, and 3 fully connected layers. To make your life easier, you can use the following code for the architecture:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable

class Net(nn.Module):
    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)
        # 6*6 from image dimension
        self.fc1 = nn.Linear(16 * 6 * 6, 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))
        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):
        # all dimensions except the batch dimension
```

```
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

Read the code carefully and try to understand the code by compare it with the figure above.

3. LeNet actually requires input to be 32x32 where images from MNIST are 28x28. Also, Pytorch uses default data type as float32 instead of float64. Furthermore, the above Pytorch class requires the input data to be from class `Variable` and the shape of Nx1x32x32, where N is number of images. So before training your model, you should firstly convert numpy array you always use, which is in the shape of 1000x784 to the above mentioned dimension, to the format required by Pytorch. For example, you can do the following:

```
X_small = X_small.reshape((-1, 1, 28,28))
X_small = np.pad(X_small, ([0,0], [0,0], [2,2], [2,2]),
mode='constant')
X_training =
Variable(torch.from_numpy(X_small.astype('float32')))
y_training = Variable(torch.from_numpy(y_small.astype('int')))
```

Note that `np.pad` will pad 0s around the original image to 32x32.

4. Write your own code to train the model (see the additional materials for the lab for how to do so). You are suggested to use `CrossEntropyLoss` from `torch.nn` and `SGD` from `torch.optim` for loss calculation and optimization (see documentation for these classes) as follows:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.25)
```

After the training, you should test the model and look at the performance (accuracy, etc) similar to what you did in previous labs. Try to tune the model by changing number of epochs and learning rate for better performance.