The Experiment Report of Deep Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Junteng Pang

Supervisor:
Mingkui Tan

Student ID：201720145174

Grade:
Graduate

December 22, 2017

Handwritten number recognition based on shallow neural network

Abstract— neural network can be used to solve many problems. This experiment aims at recognizing handwritten number based on shallow neural network, which can help us understand the neural network better and have the chance to use it to solve concrete problems.

I. Introduction

In this experiment, we use MNIST database to train and test a neural network. Neural network learns (progressively improve performance on) tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any a priori knowledge about cats, e.g., that they have fur, tails, whiskers and cat-like faces. Instead, they evolve their own set of relevant characteristics from the learning material that they process.

Through this experiment, we can understand the process of building and using neural networks in deep learning, learn how to use deep learning framework pytorch and get started, experience neural network training and testing process and have further understanding in convolution, pooling, ReLu, fully connected and other network layer.

II. Methods and Theory

MNIST

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23 percent. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support vector machine to get an error rate of 0.8 percent. An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits.

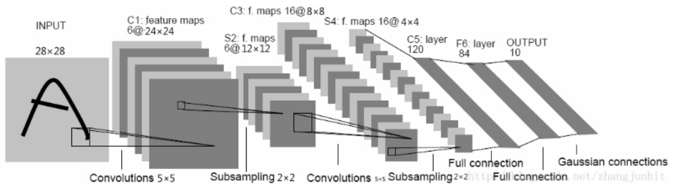
Convolutional Neural Network

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. In this article we will discuss the architecture of a CNN and the back propagation algorithm to compute the gradient with respect to the parameters of the model in order to use gradient based optimization. See the respective tutorials on convolution and pooling for more details on those specific operations.

A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a m x m x r image where mm is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3. The convolutional layer will have k filters (or kernels) of size n x n x q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size m−n+1. Each map is then subsampled typically with mean or max pooling over p x p contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map. The figure below illustrates a full layer in a CNN consisting of convolutional and subsampling sublayers. Units of the same color have tied weights.

III. Experiment

a. In the model.py write class LeNet5 (this class inherits from nn.Module, at least complete the init and forward two methods), the network structure is as follows:



```
import torch
import torch.nn as nn
import torch.nn.functional as F
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1   = nn.Linear(256, 120)
        self.fc2   = nn.Linear(120, 84)
        self.fc3   = nn.Linear(84, 10)
    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 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):
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
    return num_features
```

b. training and verification of the neural network in train.py

Load data using torch.utils.data.DataLoader, torchvision.datasets.MNIST, and torchvision.transforms.

```
import torch
import torch.nn as nn
from torch.autograd import Variable
from torchvision import datasets, transforms
import torch.nn.init as nnInit
from model import LeNet5

kwargs = {'num_workers': 4, 'pin_memory': True} if torch.cuda.is_available() else {}

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
        transform=transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081, ))
        ])),
    batch_size=64, shuffle=True,**kwargs)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data/', train=False,
        transform=transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081, ))
        ])),
batch_size=64, shuffle=False,**kwargs)
```

Instantiate torch.optim.SGD to get the optimizer and Instantiate LeNet5 to get the neural network.

```
# create model instance
model = LeNet5()
# initialize weights and bias
for m in model.modules():
    if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
        nnInit.xavier_normal(m.weight)
        if m.bias is not None:
            m.bias.data.zero_()

if torch.cuda.is_available():
    model.cuda()

optimizer = torch.optim.SGD(model.parameters(),
    lr=0.1,
    momentum=0.9,
    weight_decay=1e-4,
nesterov=True)
```

Use torch.nn.CrossEntropyLoss to calculate the loss of predict target and ground truth target.

```
if torch.cuda.is_available():
    criterion = nn.CrossEntropyLoss().cuda()
else:
    criterion = nn.CrossEntropyLoss()
```

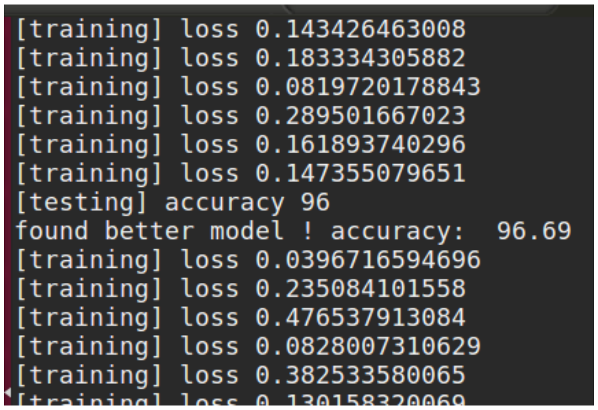
Calculate the gradient using loss.backward and optimize with optimizer.step. Calculation of recognition accuracy. Repeat steps until all data is placed on the network. The number of selected training epoch, repeat the training process. Save the best model as a .pkl file.

```
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader) :
        if torch.cuda.is_available():
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print'[training] loss', loss.data[0]

def test(epoch):
    model.eval ()
    correct = 0
    for batch_idx, (data, target) in enumerate(test_loader) :
        if torch.cuda.is_available():
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        output = model(data)
        pred = output.data.max(1)[1]
        correct += pred.eq(target.data).cpu().sum()
    print'[testing] accuracy', 100*correct/len(test_loader.dataset)
    return model,correct
```

```
if __name__ == "__main__":
    maxCorrect = 0
    for epoch in range(1, 6):
        train(epoch)
        model,correct = test(epoch)
        if correct >= maxCorrect :
            bestModel = model
            maxCorrect = correct
            print'found better model ! accuracy: ',correct/100.0
torch.save(bestModel,'model.pkl')
```

Result:



```
[training] loss 0.143426463008
[training] loss 0.183334305882
[training] loss 0.0819720178843
[training] loss 0.289501667023
[training] loss 0.161893740296
[training] loss 0.147355079651
[testing] accuracy 96
found better model ! accuracy:  96.69
[training] loss 0.0396716594696
[training] loss 0.235084101558
[training] loss 0.476537913084
[training] loss 0.0828007310629
[training] loss 0.382533580065
[training] loss 0.130158320069
```

IV. CONCLUSION

In this experiment, a shallow neural network LeNet5 is used to recognize handwritten number and it has a good performance. I understand the process of building and using neural networks in deep learning while building the LeNet5. Also, I learn how to use deep learning framework pytorch and experience neural network training and testing process and have further understanding in convolution, pooling, ReLu, fully connected and other network layer.