MNIST Handwritten Data Modeling

mz2840

Description

The MNIST database of handwritten digits is one of the most commonly used dataset for training various image processing systems and machine learning algorithms. It has a training set of 60,000 examples, and a test set of 10,000 examples. MNIST is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. The original black and white (bilevel) images from MNIST were size normalized. We developed several classification methods including KNN, AdaBoost, SVM and some deep learning model like Convolutional Neural Network to do pattern recognition and predict what that digit is. And cross-entropy error and miss-classification error to improve models and judge which model is better.

Dependencies

Python distribution

- Anaconda Python 3.6 or Miniconda Python 3.6

General

- numpy: conda install numpy or pip install numpy
- scipy: conda install scipy or pip install scipy
- sklearn: conda install scikit-learn [basic learning algorithms] or pip install install scikit-learn
- matplotlib: conda install matplotlib [visualization] or pip install matplotlib

Deep learning

- pytorch>=0.4.0: conda install pytorch -c pytorch
- tensorflow>=1.2: conda install tensorflow-gpu -c anaconda
- keras: pip install keras
- cuda: torch.cuda.is_available()

Libraries and setting

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torchvision
import warnings
warnings.filterwarnings("ignore")
```

```
import torch
import torch.nn as nn
import torchvision
import torch.utils.data as Data
import torch.nn.functional as F
import torchvision.transforms as transforms
```

```
# initialize the parameters
input_size = 784
hidden_size = 100
num_classes = 10
num_epochs = 150
batch_size = 100
learning_rate = .1
```

Data preparation

The dimension of the X_train is (60000, 28, 28), and the dimension of the X_test is (10000,28,28). For the before deep learning part, we normalize the X_train and X_test by applying to *normalize* function in the *sklearn.preprocessing* package. Then we use *one-hot* embedding to deal with labels by using the *OneHotEncoder* function in the sklearn.preprocessing package.

For the deep learning part, I used Pytorch to train model later and transformed it to the torch tensors data and then applied *torch.utils.data.TensorDataset* to load the data.

```
DOWNLOAD_MNIST = True
train_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=True , # this is training data
    transform=torchvision.transforms.ToTensor (),
    download=DOWNLOAD_MNIST ,
    )
```

```
test_data = torchvision.datasets.MNIST(root='./ mnist/',
                                       transform=torchvision.transforms.ToTensor (),
                                       train=False,download=DOWNLOAD_MNIST,)
```

```
# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_data,batch_size = batch_size,shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_data,batch_size = batch_size,shuffle=False)
```

Model executing

KNN method

After applying *GridSearchCV* function in the *sklearn.model_selection* package and *KNeighborsClassifier* function. And the output is when n_neighbors = 5, the model has the highest accuracy, the accuracy is 96.68%.

AdaBoost method

Here the weak classifier is C4.5 decision tree. When we set the max_depth = 10, and n_estimators = 70, it can tree the dataset very well, and the error after applying Adaboost is 1-0.9643 = 3.569%.

SVM method

We apply the SVM model on reshaped normalized X and raw label in the dataset. From the result of GridSearchCV output, when gamma = 0.01, C = 10, the model has the highest accuracy, the accuracy is 97.42%.

Neural Network method

I firstly add 3 layers in the sequential and applied *GridSearchCV* to search the best parameters. And the result is that the accuracy is 97.96%. And I would like to increase accuracy by introducing the Convolutional neural network model, it generates a better result, which the accuracy is 98.89%. I will talk more about CNN model later.

A Single Layer Neural Network

Initially I trained a single layer neural network model with 100 hidden layers. And I also set hyparameters: batch size equals 100 and learning rate equals 0.1. Here is the basic neural network

I built. The single neural network only has one hidden layer which contains 100 units, and the activation function is the ReLU function. And the optimizer technique I used is stochastic gradient descent. And I used *torch.manual_seed* to set five different seeds.

```
torch.manual_seed(1)    # reproducible, set different seeds
class NeuralNet(nn.Module):
     def __init__(self, input_size, hidden_size, num_classes):
            super(NeuralNet, self).__init__()
            self.input_size = input_size
            self.hidden = nn.Linear(input_size, hidden_size)
            self.relu = nn.ReLU()
            self.out = nn.Linear(hidden_size, num_classes)

     def forward(self, x):
        # nonlinear activation
        x = F.relu(self.hidden(x))
        x = self.out(x)
        return x

net = NeuralNet(input_size, hidden_size, num_classes)
```

Then I calculated and visualized the average training cross-entropy error and the average validation cross-entropy error by applying *torch.nn.CrossEntropyLoss()* I also calculated and visualized the miss-classification error of the training set and the testing set.

CNN with two 2-D convolutional layers

Then I developed a two 2-D convolutional layers of CNN model to train the dataset. Here is the CNN model I built. And the optimizer technique I used is stochastic gradient descent. And I used *torch.manual_seed* to set five different seeds.

```
torch.manual_seed(1)

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 10, kernel_size=5),
            nn.MaxPool2d(2),
            nn.ReLU(),
            nn.Conv2d(10, 20, kernel_size=5),
            nn.Dropout(),
            nn.MaxPool2d(2),
            nn.ReLU(),
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(320, 50),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(50, 10),
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(-1, 320)
        x = self.fc_layers(x)
        return x

CNN = CNN()
CNN.cuda()
```

Then I calculated and visualized the average training cross-entropy error and the average validation cross-entropy error by applying *torch.nn.CrossEntropyLoss()* I also calculated and visualized the miss-classification error of the training set and the testing set.

Favorite Deep Learning Architecture

Here I developed a three layers 2-D convolutional layers of CNN model to train the dataset, and I added Batch Normalization to standardize the inputs to a layer for each batch and Dropout rate prevent overfitting.

```
class Net(nn.Module):
  def __init__(self):
    super(Net,self).__init__()
    self.conv1=nn.Conv2d(1,32,3,1)
    self.conv1_bn=nn.BatchNorm2d(32)

    self.conv2=nn.Conv2d(32,64,3,1)
    self.conv2_bn=nn.BatchNorm2d(64)

    self.conv3=nn.Conv2d(64,128,3,1)
    self.conv3_bn=nn.BatchNorm2d(128)

    self.dropout1=nn.Dropout(0.25)

    self.fc1=nn.Linear(15488,128)
    self.fc1_bn=nn.BatchNorm1d(128)

    self.fc2=nn.Linear(128,64)
    self.fc3=nn.Linear(64,10)
  def forward(self,x):
    x=self.conv1(x)
    x=F.relu(self.conv1_bn(x))

    x=self.conv2(x)
    x=F.relu(self.conv2_bn(x))

    x=self.conv3(x)
    x=F.relu(self.conv3_bn(x))

    x=F.max_pool2d(x,2)
    x=self.dropout1(x)

    x=torch.flatten(x,1)

    x=self.fc1(x)
    x=F.relu(self.fc1_bn(x))
```

Then I calculated and visualized the average training cross-entropy error and the average validation cross-entropy error by applying *torch.nn.CrossEntropyLoss()* I also calculated and visualized the miss-classification error of the training set and the testing set.

More about Deep Learning

Data preparation

rain.txt, val.txt and test.txt. In particular, train.txt contains 20,000 lines and val.txt and test.txt contains 5000 lines in the same format. Each line contains 1569 coordinates, with the first 784 real-valued numbers correspond to the 784 pixel values for the first digit, next 784 real valued numbers correspond to the pixel values for the second digit.

After loading the data, and visualizing some examples, it is essential to know that the pixels were scanned out in row-major order. And from the three examples we plot, the last coordinate of each line is the sum of the 2 digits.

Since txt file is not suitable to use Pytorch to train model later, here we need to process it and transform it to the torch tensors data and then applied *torch.utils.data.TensorDataset* to load the data.

```
train_x_reshape = np.array(train.iloc[:,:1568]).reshape(20000,28,56)
test_x_reshape = np.array(test.iloc[:,:1568]).reshape(5000,28,56)
val_x_reshape = np.array(val.iloc[:,:1568]).reshape(5000,28,56)
```

```
train_y_reshape = np.array(train.iloc[:,1568])
test_y_reshape = np.array(test.iloc[:,1568])
val_y_reshape = np.array(val.iloc[:,1568])
```

```
train_x_tensor = torch.tensor(train_x_reshape)
test_x_tensor = torch.tensor(test_x_reshape)
val_x_tensor = torch.tensor(val_x_reshape)
```

```
train_y_tensor = torch.tensor(train_y_reshape)
test_y_tensor = torch.tensor(test_y_reshape)
val_y_tensor = torch.tensor(val_y_reshape)
```

```
train_dataset = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
test_dataset = torch.utils.data.TensorDataset(test_x_tensor, test_y_tensor)
val_dataset = torch.utils.data.TensorDataset(val_x_tensor, val_y_tensor)
```

```
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)
val_loader = torch.utils.data.DataLoader(dataset = val_dataset,batch_size = batch_size,shuffle=False)
```

Model executing

Model 1: A three layers 2–D convolutional layers of CNN

Here I developed a three layers 2-D convolutional layers of CNN model to train the dataset, and I added Batch Normalization to standardize the inputs to a layer for each batch.

```
torch.manual_seed(1)
class cnn(torch.nn.Module):
    def __init__(self):
      super(cnn, self).__init__()
      self.conv1 = nn.Sequential(
        nn.Conv2d(1, 36, 3, 1,1),
        nn.ReLU(),
        nn.MaxPool2d(2)
        )
      self.conv2 = nn.Sequential(
        nn.Conv2d(36, 72, 3, 1,1),
        nn.ReLU(),
        nn.MaxPool2d(2)
        )
      self.conv3 = nn.Sequential(
        nn.Conv2d(72, 144, 3, 1,1),
        nn.ReLU(),
        nn.Flatten()
        )
      self.l1 = nn.Linear(144*7*14, 512)
      self.l2 = nn.Linear(512, 19)
    def forward(self, X):
      X = self.conv1(X)
      X = self.conv2(X)
      X = self.conv3(X)
      X = nn.functional.relu(self.l1(X))
      X = self.l2(X)
      return X
net = cnn()
net.cuda()
```

Model 2: A three 2-D convolutional layers of CNN model

with BatchNorm and Dropout

Here I developed three layers of 2-D convolutional layers of the CNN model to train the dataset. What is different from the first model is that I added three Batch Normalization to standardize the inputs to a layer for each batch.

```
class Net(nn.Module):
  def __init__(self):
    super(Net,self).__init__()
    self.conv1=nn.Conv2d(1,32,3,1)
    self.conv1_bn=nn.BatchNorm2d(32)

    self.conv2=nn.Conv2d(32,64,3,1)
    self.conv2_bn=nn.BatchNorm2d(64)

    self.conv3=nn.Conv2d(64,128,3,1)
    self.conv3_bn=nn.BatchNorm2d(128)

    self.dropout1=nn.Dropout(0.25)

    self.fc1=nn.Linear(35200,100)
    self.fc1_bn=nn.BatchNorm1d(100)

    self.fc2=nn.Linear(100,64)
    self.fc3=nn.Linear(64,19)
  def forward(self,x):
    x=self.conv1(x)
    x=F.relu(self.conv1_bn(x))

    x=self.conv2(x)
    x=F.relu(self.conv2_bn(x))

    x=self.conv3(x)
    x=F.relu(self.conv3_bn(x))

    x=F.max_pool2d(x,2)
    x=self.dropout1(x)

    x=torch.flatten(x,1)

    x=self.fc1(x)
    x=F.relu(self.fc1_bn(x))
    x=self.fc2(x)
    x=self.fc3(x)
    output=F.log_softmax(x,dim=1)
    return output
net = Net()
net.cuda()
```

After training the model in the proceeded dataset, and visualize the average training cross-entropy error and the average validation cross-entropy error as well as miss-classification error of training set and validation set. After comparing, here I choose **the second model**. Then I used the second model to train on the test.txt, and the generalization miss-classification error is 0.025. The generalization error is slightly larger than the error of the original MNIST data (Neural network:0.02, CNN: approximately 0.005).

Authors

Contributors names and contact info
Name: Mengjie Zhang
Email: mz2840@columbia.edu

License

This project is licensed under the MNIST License - see the LICENSE.md file for details.

Acknowledgments

Inspiration, code snippets, etc.