School of Data and Computer Science, Sun Yat-sen University

E13 BP Algorithm (C++/Python)

18340013 Conghao Chen

December 10, 2020

Contents

1 Horse Colic Data Set

The description of the horse colic data set (`http://archive.ics.uci.edu/ml/datasets/Horse+Colic`) is as follows:

Data Set Characteristics:	Multivariate	**Number of Instances:**	368	**Area:**	Life
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	27	**Date Donated**	1989-08-06
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	108569

We aim at trying to predict if a horse with colic will live or die.

Note that we should deal with missing values in the data! Here are some options:

- Use the feature' s mean value from all the available data.
- Fill in the unknown with a special value like -1.
- Ignore the instance.
- Use a mean value from similar items.
- Use another machine learning algorithm to predict the value.

2 Reference Materials

1. Stanford: **CS231n: Convolutional Neural Networks for Visual Recognition** by Fei-Fei Li,etc.
   - Course website: `http://cs231n.stanford.edu/2017/syllabus.html`
   - Video website: `https://www.bilibili.com/video/av17204303/?p=9&tdsourcetag=s_pctim_aiomsg`
2. **Machine Learning** by Hung-yi Lee
   - Course website: `http://speech.ee.ntu.edu.tw/~tlkagk/index.html`
   - Video website: `https://www.bilibili.com/video/av9770302/from=search`
3. A Simple neural network code template

```
# -*- coding: utf-8 -*
import random
import math

# Shorthand:
# "pd_" as a variable prefix means "partial derivative"
# "d_" as a variable prefix means "derivative"
# "_wrt_" is shorthand for "with respect to"
```

```
# "w_ho" and "w_ih" are the index of weights from hidden to output layer neurons
    and input to hidden layer neurons respectively

class NeuralNetwork:
    LEARNING_RATE = 0.5
    def __init__(self, num_inputs, num_hidden, num_outputs, hidden_layer_weights =
        None, hidden_layer_bias = None, output_layer_weights = None,
        output_layer_bias = None):
    #Your Code Here

    def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights
        ):
    #Your Code Here

    def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(self,
        output_layer_weights):
    #Your Code Here

    def inspect(self):
        print('------')
        print('* Inputs: {}'.format(self.num_inputs))
        print('------')
        print('Hidden Layer')
        self.hidden_layer.inspect()
        print('------')
        print('* Output Layer')
        self.output_layer.inspect()
        print('------')

    def feed_forward(self, inputs):
        #Your Code Here

    # Uses online learning, ie updating the weights after each training case
    def train(self, training_inputs, training_outputs):
        self.feed_forward(training_inputs)

        # 1. Output neuron deltas
        #Your Code Here
        #  E/ z

        # 2. Hidden neuron deltas
```

```
        # We need to calculate the derivative of the error with respect to the output of each hidden layer neuron
        # dE/dy = Σ E/ z * z/ y = Σ E/ z * w
        # E/ z = dE/dy * z /
        #Your Code Here

        # 3. Update output neuron weights
        # E / w = E/ z * z / w
        # Δw = * E / w
        #Your Code Here

        # 4. Update hidden neuron weights
        # E / w = E/ z * z / w
        # Δw = * E / w
        #Your Code Here

    def calculate_total_error(self, training_sets):
        #Your Code Here
        return total_error

class NeuronLayer:
    def __init__(self, num_neurons, bias):

        # Every neuron in a layer shares the same bias
        self.bias = bias if bias else random.random()

        self.neurons = []
        for i in range(num_neurons):
            self.neurons.append(Neuron(self.bias))

    def inspect(self):
        print('Neurons:', len(self.neurons))
        for n in range(len(self.neurons)):
            print(' Neuron', n)
            for w in range(len(self.neurons[n].weights)):
                print('  Weight:', self.neurons[n].weights[w])
            print('  Bias:', self.bias)

    def feed_forward(self, inputs):
        outputs = []
        for neuron in self.neurons:
```

```
            outputs.append(neuron.calculate_output(inputs))
        return outputs

    def get_outputs(self):
        outputs = []
        for neuron in self.neurons:
            outputs.append(neuron.output)
        return outputs

class Neuron:
    def __init__(self, bias):
        self.bias = bias
        self.weights = []

    def calculate_output(self, inputs):
    #Your Code Here

    def calculate_total_net_input(self):
    #Your Code Here

    # Apply the logistic function to squash the output of the neuron
    # The result is sometimes referred to as 'net' [2] or 'net' [1]
    def squash(self, total_net_input):
    #Your Code Here

    # Determine how much the neuron's total input has to change to move closer to the expected output
    #
    # Now that we have the partial derivative of the error with respect to the output ( E/ y ) and
    # the derivative of the output with respect to the total net input (dy/dz ) we can calculate
    # the partial derivative of the error with respect to the total net input.
    # This value is also known as the delta ( ) [1]
    #   = E/ z = E/ y * dy/dz
    #
    def calculate_pd_error_wrt_total_net_input(self, target_output):
    #Your Code Here

    # The error for each neuron is calculated by the Mean Square Error method:
    def calculate_error(self, target_output):
```

```
    #Your Code Here

    # The partial derivate of the error with respect to actual output then is calculated by:
    # = 2 * 0.5 * (target output - actual output) ^ (2 - 1) * -1
    # = -(target output - actual output)
    #
    # The Wikipedia article on backpropagation [1] simplifies to the following, but most other learning material does not [2]
    # = actual output - target output
    #
    # Alternative, you can use (target - output), but then need to add it during backpropagation [3]
    #
    # Note that the actual output of the output neuron is often written as y  and target output as t  so:
    # =  E/ y  = -(t  - y )
    def calculate_pd_error_wrt_output(self, target_output):
    #Your Code Here

    # The total net input into the neuron is squashed using logistic function to calculate the neuron's output:
    # y  =   = 1 / (1 + e^(-z ))
    # Note that where   represents the output of the neurons in whatever layer we're looking at and   represents the layer below it
    #
    # The derivative (not partial derivative since there is only one variable) of the output then is:
    # dy /dz  = y  * (1 - y )
    def calculate_pd_total_net_input_wrt_input(self):
    #Your Code Here

    # The total net input is the weighted sum of all the inputs to the neuron and their respective weights:
    # = z  = net  = x w  + x w  ...
    #
    # The partial derivative of the total net input with respective to a given weight (with everything else held constant) then is:
    # =  z / w  = some constant + 1 * x w ^(1-0) + some constant ... = x
    def calculate_pd_total_net_input_wrt_weight(self, index):
    #Your Code Here
```

```
# An example:

nn = NeuralNetwork(2, 2, 2, hidden_layer_weights=[0.15, 0.2, 0.25, 0.3],
    hidden_layer_bias=0.35, output_layer_weights=[0.4, 0.45, 0.5, 0.55],
    output_layer_bias=0.6)
for i in range(10000):
    nn.train([0.05, 0.1], [0.01, 0.99])
    print(i, round(nn.calculate_total_error([[[0.05, 0.1], [0.01, 0.99]]]), 9))
```

3 Tasks

- Given the training set `horse-colic.data` and the testing set `horse-colic.test`, implement the BP algorithm and establish a neural network to predict if horses with colic will live or die. In addition, you should calculate the accuracy rate.
- Please submit a file named `E14_YourNumber.pdf` and send it to `ai_2020@foxmail.com`
- Draw the training loss and accuracy curves.
- (optional) You can try different structure of neural network and compare their accuracy and the time they cost.

4 Codes

该神经网络的结构为：输入层神经元个数为属性的列数（预处理过后），隐藏层神经元个数为 8（不断调参得到的结果，在 8 附近均可），输出层神经元个数为 3（类数）。
在实现过程中添加了一些之前在实验室写神经网络用到的、课上没有讲过的内容：

- 参数初始化时使用了 `Kaiming` 分布；
- 设置 `batch`，使用了 `SGD` 方法；
- 权重衰减（weight decay）与学习率衰减（learning rate decay）；

此外，由于近 30% 的数据缺失，因此在预处理数据时用到了标签编码和独热编码的相关内容。`nn.py` 为神经网络（并没有使用文档给的框架）；`bp.py` 为数据预处理以及训练过程。

nn.py

```
import numpy as np

class fullparam(object):

```

```
    def __init__(self, in_features, out_features,bias):
        self.in_features = in_features
        self.out_features = out_features
        self.weight = np.random.normal(0,np.sqrt(2/in_features),(out_features,
            in_features))  # kaiming
        if bias:
            self.bias = np.random.rand(out_features)
        else:
            self.bias = None

    def forward(self, inputs):
        if type(self.bias) != type(None):
            return np.dot(inputs, self.weight.T) + self.bias
        else:
            return np.dot(inputs, self.weight.T)

    def __call__(self,x):
        return self.forward(x)

class Network(object):

    def __init__(self,in_features,hidden_features,out_features,learning_rate=0.01):
        self.w_ih = fullparam(in_features,hidden_features,True)
        self.w_ho = fullparam(hidden_features,out_features,True)
        self.learning_rate = learning_rate
        self.memory = {}  # used for store results
        self.train_flag = True

    def train(self):
        self.train_flag = True

    def end(self):
        self.train_flag = False

    def sigmoid(self,x):
        return 1 / (1 + np.exp(-x))

    def d_sigmoid(self,x):
        return self.sigmoid(x) * (1 - self.sigmoid(x))

    def MSE(self,y_hat,y):
```

```python
        return (np.linalg.norm(y_hat - y)) * (np.linalg.norm(y_hat - y))   # mean square
            error

    def forwardpass(self,x):
        # training
        if self.train_flag:
            self.memory["a0"] = np.copy(x)
            x = self.w_ih(x)       # between input and hidden
            self.memory["z1"] = np.copy(x)
            x = self.sigmoid(x)
            self.memory["a1"] = np.copy(x)
            x = self.w_ho(x)      # between hidden and out
            self.memory["z2"] = np.copy(x)
            x = self.sigmoid(x)
        # train end
        else:
            x = self.w_ih(x)
            x = self.sigmoid(x)
            x = self.w_ho(x)
            x = self.sigmoid(x)
        return x

    def backpass(self,y_hat,y,lamb=0):
        batchsize = y.shape[0]
        delta = [0] * 3
        # out layer delta calculate
        delta[2] = (y_hat - y) * self.d_sigmoid(self.memory["z2"])
        # hidden layer delta calculate
        delta[1] = np.dot(delta[2],self.w_ho.weight) * self.d_sigmoid(self.memory["z1"])
        grad_W = [0] * 2
        # N * out_features * hidden_features
        grad_W[1] = np.einsum("ij,ik->ijk",delta[2],self.memory["a1"])
        # N * hidden_features * in_features
        grad_W[0] = np.einsum("ij,ik->ijk",delta[1],self.memory["a0"])
        # every col's mean value
        grad_W[1] = grad_W[1].mean(axis=0)
        grad_W[0] = grad_W[0].mean(axis=0)
        # weight update
        self.w_ho.weight -= self.learning_rate * (grad_W[1] + lamb * self.w_ho.weight /
            batchsize)
        self.w_ih.weight -= self.learning_rate * (grad_W[0] + lamb * self.w_ih.weight /
```

```
            batchsize)
```

bp.py

```
import nn
import pickle
import os, sys, time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 0: continuous,1: nominal,2: categorical/discrete
attrlist = {"surgery": 1,
 "Age": 2,
 "Hospital Number": 1,
 "rectal temperature": 0,
 "pulse": 0,
 "respiratory rate": 0,
 "temperature of extremities": 2,
 "peripheral pulse": 2,
 "mucous membranes": 1,
 "capillary refill time": 2,
 "pain": 1,
 "peristalsis": 2,
 "abdominal distension": 1,
 "nasogastric tube": 1,
 "nasogastric reflux": 2,
 "nasogastric reflux PH": 0,
 "rectal examination": 2,
 "abdomen": 1,
 "packed cell volume": 0,
 "total protein": 0,
 "abdominocentesis appearance": 1,
 "abdomcentesis total protein": 0,
 "outcome": 1,
 "surgical lesion": 1,
 "type of lesion 1": 1,
 "type of lesion 2": 1,
 "type of lesion 3": 1,
 "cp_data": 1}

train_data = pd.read_csv("horse-colic.data",names=attrlist.keys(),index_col=False,
```

```
    delim_whitespace=True)
test_data = pd.read_csv("horse-colic.test",names=attrlist.keys(),index_col=False,
    delim_whitespace=True)


def preprocessing(data):
    removelist = ["type of lesion 2", "type of lesion 3","Hospital Number","nasogastric
        reflux PH","abdomcentesis total protein"]
    attributes = []
    for a in data.columns.values:
        indata = attrlist.get(a,None)
        if indata == None:    # newly append
            attributes.append(a)
        elif indata == 0 and a not in removelist: # continuous
            attributes.append(a)
        else: # discrete, no need to append
            pass
    df = data[attributes]
    return df


def fulldata(data):
    """
    Fill the missing data
    For continuous: fill them with mean values
    For discrete: fill them with the mode values
    """
    for a in data.columns.values:
        if a in ["type of lesion 1", "Hospital Number"]: # remove
            continue
        if data[a].dtype != np.int64:   # missing data
            have_data = data[data[a] != "?"][a]
            if attrlist[a]:   # discrete
                data.loc[data[a] == "?",a] = have_data.value_counts().idxmax() # mode
                    values
                if a != "outcome" and attrlist[a] != 2:
                    # generate one-hot encoding
                    data[a] = pd.Categorical(data[a])
                    dummies = pd.get_dummies(data[a],prefix="{}_category".format(a))
                    data = pd.concat([data,dummies],axis=1)
            else:    # continuous
```

```
                data.loc[data[a] == "?",a] = np.mean(have_data.astype(np.float))  # mean values
        elif attrlist[a] == 1:
            # generate one-hot encoding
            data[a] = pd.Categorical(data[a])
            dummies = pd.get_dummies(data[a],prefix="{}_category".format(a))
            data = pd.concat([data,dummies],axis=1)
    return data.astype(np.float)


data = pd.concat([train_data,test_data],axis=0)
data = fulldata(data)
label = data["outcome"].astype(np.float)
train_label, test_label = label[:len(train_data)], label[len(train_data):]
train_label = [[1,0,0] if label == 1 else ([0,1,0] if label == 2 else [0,0,1]) for label in train_label]
# train_data, test_data = data[:len(train_data)], data[len(train_data):]
# train_data = preprocessing(train_data)
# test_data = preprocessing(test_data)
data = preprocessing(data)
train_data, test_data = data[:len(train_data)], data[len(train_data):]
# print(train_data.columns)
# print(len(train_data.columns))


def minibatch(data,label,batchsize=16):
    num_batches = len(data) // batchsize
    for i in range(0,num_batches,batchsize):
        yield data[i:i+batchsize].to_numpy(), np.array(label[i:i+batchsize])

def train(net,max_iter=70000):
    losslist, acclist = [], []
    losses = []
    for i in range(max_iter):
        net.train()
        batches = minibatch(train_data,train_label,16) # generator
        for x, y in batches:
            y_hat = net.forwardpass(x)
            loss = net.MSE(y_hat, y)
            losses.append(loss)
            # net.backpass(y_hat,y)   # without weight decay
```

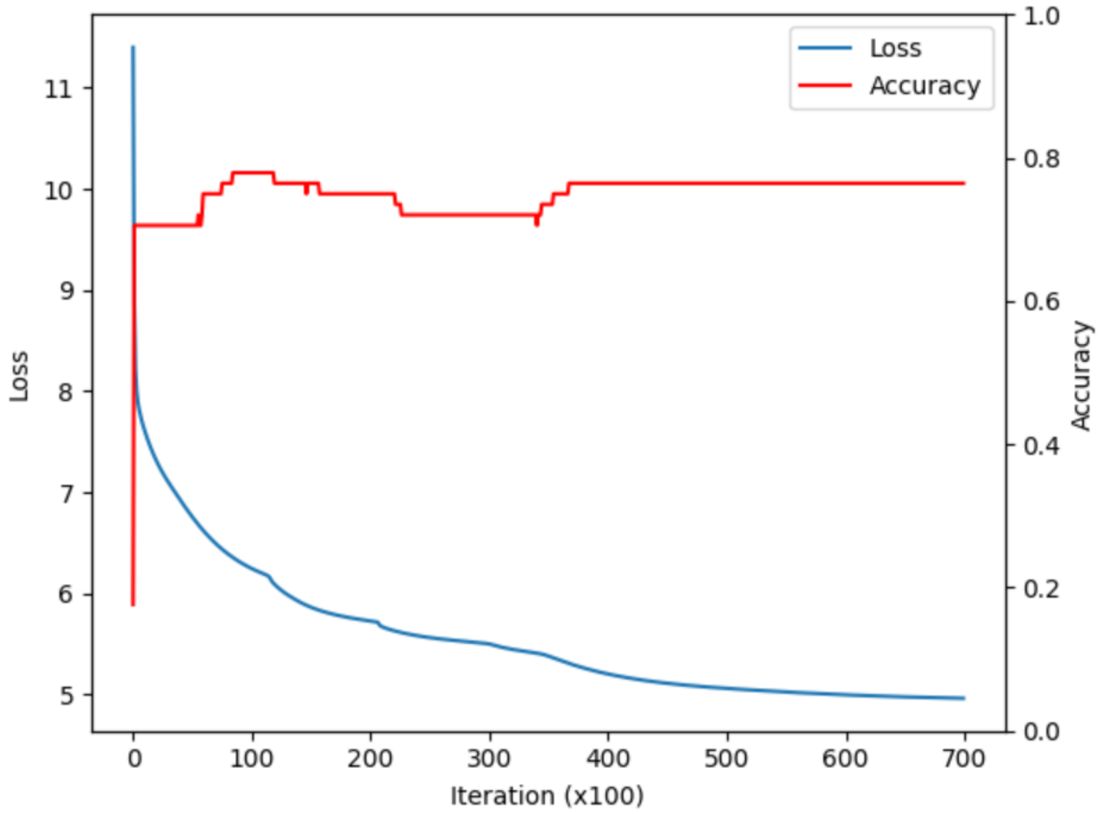
```
            net.backpass(y_hat,y,0.1) # weight decay

        # update learning rate and record parameters
        if (i+1) % 100 == 0:
            avg_loss = np.array(losses).mean()
            losslist.append(avg_loss)
            losses = []
            acc = test(net,test_data,test_label)
            acclist.append(acc)
            if (i+1) % 1000==0:
                net.learning_rate *= 0.99
    return losslist, acclist

def test(net,test_X,test_Y,flag=True,print_flag=False):
    count = 0
    for j, x in test_X.iterrows():
        net.end()
        y_hat = net.forwardpass(x.to_numpy().reshape(1,-1))
        predicted = np.argmax(y_hat) + 1
        y = test_Y[j]
        if print_flag:
            print(y_hat,predicted,y)
        if flag:
            if predicted == y:
                count += 1
        else:
            if [1 if t + 1 == predicted else 0 for t in range(3)] == y:
                count += 1
    return (count / len(test_X))


# print(len(test_data.columns.values))
# print(len(train_data.columns.values))
start_time=time.time()
net = nn.Network(len(test_data.columns.values),8,3,0.01)
losslist, acclist = train(net,70000)
print(acclist[-1])
end_time=time.time()
print("Time:{}s".format(end_time-start_time))


```

```python
# plot
fig = plt.figure()
ax = fig.add_subplot(111)
lns1 = ax.plot(losslist,label="Loss")
ax2 = ax.twinx()
lns2 = ax2.plot(acclist,"-r",label="Accuracy")
lns = lns1 + lns2
labs = [l.get_label() for l in lns]
ax.legend(lns,labs,loc=0)
ax.set_xlabel("Iteration (x100)")
ax.set_ylabel("Loss")
ax2.set_ylabel("Accuracy")
ax2.set_ylim(0,1)
plt.savefig(r"fig/iteration.pdf",format="pdf",dpi=200)
plt.show()
```

5 Results

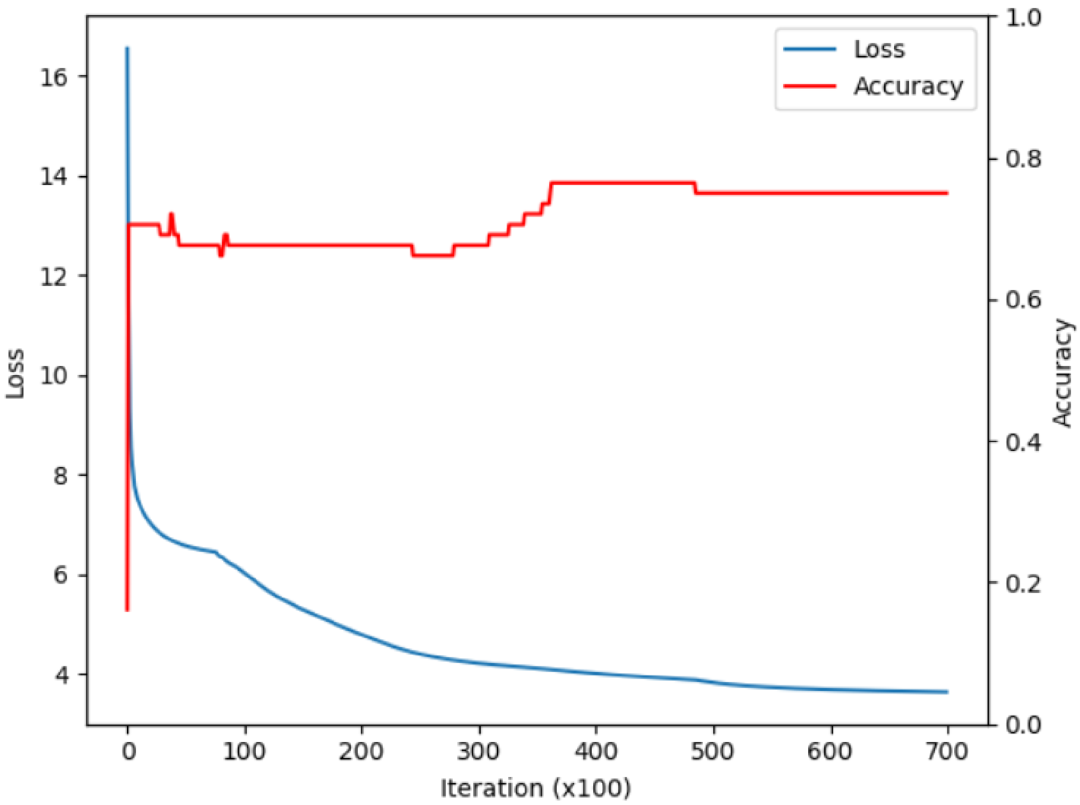
结果如下：

```
C:\Users\czh\.conda\envs\Pycharm\python.exe
0.7647058823529411
Time:110.73993492126465s
```



修改隐藏层神经元数量（不是隐藏层层数，层数始终为 1，没有使用 DNN），由原来的 8 变为 10，结果如下：

```
C:\Users\czh\.conda\envs\Pycharm\python.exe
0.75
Time:97.2220995426178s
```


Loss
Accuracy
Iteration (x100)
0
100
200
300
400
500
600
700
4
6
8
10
12
14
16
0.0
0.2
0.4
0.6
0.8
1.0