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

E14 BP Algorithm (C++/Python)

17341203 Yixin Zhang

December 14, 2019

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
```

```python

# 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_201901@foxmail.com`

4 Codes and Results

将前馈和反向传播过程向量化后就很好写代码了，使用矩阵与向量乘法，可以避免用循环逐个更新每个参数。最终，我的准确率为 75%。

前馈过程：

$$a^l = g(W^l a^{l-1} + b^l)$$

反向传播：

$$\delta^L = \frac{\partial J}{\partial z^L} \circ g'(z^L)$$
$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \circ g'(z^l)$$
$$\frac{\partial J}{\partial W^l_{jk}} = a^{l-1}_k \delta^l_j$$
$$\frac{\partial J}{\partial b^l_j} = \delta^l_j$$

以下的页面是从 Jupyter Notebook 导出的，但排版不是很美观。为了更好地阅读体验，也可以点击以下链接：https://down.jeddd.com/temp/BackPropagation.html。

[1]:
```
import numpy as np
import pandas as pd
from scipy.stats import zscore
```

[2]:
```
def sigmoid(z):
    """The sigmoid function."""
    return 1.0 / (1.0 + np.exp(-z))

def sigmoid_d(z):
    """Derivative of the sigmoid function."""
    s = sigmoid(z)
    return s * (1 - s)
```

[3]:
```
class NeuralNetwork:
    def __init__(self, input_dim, hidden_dim, out_dim, g=sigmoid,␣
 ↪g_d=sigmoid_d):
        self.W_ih = 0.1 * np.random.rand(hidden_dim, input_dim)  # 输入层到隐含
层的权重矩阵
        self.b_ih = 0.1 * np.random.rand(hidden_dim)             # 输入层到隐含
层的偏置
        self.W_ho = 0.1 * np.random.rand(out_dim, hidden_dim)    # 隐含层到输出
层的权重矩阵
        self.b_ho = 0.1 * np.random.rand(out_dim)                # 隐含层到输出
层的偏置

        self.g = g       # 激活函数
        self.g_d = g_d   # 激活函数的梯度

    def feedForward(self, x):
        """ 输入 x, 前馈产生输出。 """
        self.x = x                                      # 输入
        self.in_h = self.W_ih @ self.x + self.b_ih      # 隐含层输入
        self.out_h = self.g(self.in_h)                  # 隐含层输出
        self.in_o = self.W_ho @ self.out_h + self.b_ho  # 输出层输入
        self.out_o = self.g(self.in_o)                  # 输出层输出, 即网络最终
输出
        return self.out_o

    def backPropagate(self, target):
        """ 反向传播并产生各层敏感度。 """
        self.delta_o = (self.out_o - target) * self.g_d(self.in_o)         # 输
出层敏感度
        self.delta_h = (self.W_ho.T @ self.delta_o) * self.g_d(self.in_h)  # 隐
含层敏感度

    def update(self, rate):
        """ 更新各个参数。 """
```

```python
        self.W_ho -= rate * (np.mat(self.delta_o).T @ np.mat(self.out_h))
        self.b_ho -= rate * self.delta_o
        self.W_ih -= rate * (np.mat(self.delta_h).T @ np.mat(self.x))
        self.b_ih -= rate * self.delta_h

    def predict(self, x):
        """ 对输入特征 x 进行预测，返回预测结果的下标。"""
        in_h = self.W_ih @ x + self.b_ih
        out_h = self.g(in_h)
        in_o = self.W_ho @ out_h + self.b_ho
        out_o = self.g(in_o)
        return out_o.argmax()  # 返回结果中最大值的下标
```

---

[4]:
```python
data_df = pd.read_csv('dataset/horse-colic-data.csv')
data_df['outcome'] = data_df.pop('outcome')
for column in data_df.columns:
    if column == 'outcome' or data_df[column].var() == 0:
        continue
#     data_df[column] = zscore(data_df[column])
    data_df[column] = (data_df[column] - data_df[column].min()) /␣
↪(data_df[column].max() - data_df[column].min())  # 归一化
# data_df

test_df = pd.read_csv('dataset/horse-colic-test.csv')
test_df['outcome'] = test_df.pop('outcome')
for column in test_df.columns:
    if column == 'outcome' or test_df[column].var() == 0:
        continue
#     test_df[column] = zscore(test_df[column])
    test_df[column] = (test_df[column] - test_df[column].min()) /␣
↪(test_df[column].max() - test_df[column].min())  # 归一化
# test_df
```

[5]:
```python
EPOCHS = 400
RATE = 0.01  # 学习率
nn = NeuralNetwork(35, 12, 3)

for epoch in range(EPOCHS):
    for i in range(len(data_df)):
        sample = data_df.iloc[i].to_numpy()
        x = sample[:-1]
        target = np.zeros(3)
        target[int(sample[-1])-1] = 1

        output = nn.feedForward(x)
        nn.backPropagate(target)
```

```
        nn.update(RATE)

    data_df = data_df.sample(frac=1)  # 打乱样本顺序
    RATE *= 0.99  # 减少学习率
```

[6]:

```
count = 0
for i in range(len(test_df)):
    sample = test_df.iloc[i].to_numpy()
    x, target = sample[:-1], sample[-1]
    if nn.predict(x) + 1 == target:
        count += 1
print('{} / {} = {:.2%}'.format(count, len(test_df), count/len(test_df)))
```

```
51 / 68 = 75.00%
```

[ ]:

```

```