E12 Backpropagation

Suixin Ou

School of Computer Science
Sun Yat-sen University

December 28, 2021

Background

The Horse Colic Dataset

- The UCI dataset (http://archive.ics.uci.edu/ml/index.php) is the most widely used dataset for machine learning. If you are interested in other datasets in other areas, you can refer to https://www.zhihu.com/question/63383992/answer/222718972.
- Colic in horses is defined as abdominal pain, but it is a clinical symptom rather than a diagnosis. Colic surgery is usually an expensive procedure as it is major abdominal surgery, often with intensive aftercare. Among domesticated horses, colic is the leading cause of premature death. So we want to predict whether a horse with colic will live or die in the Horse Colic Dataset.

Task

Description

- Dataset statistics

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:**	179592

- Domain information

horse-colic.names - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
1: surgery?
      1 = Yes, it had surgery
      2 = It was treated without surgery

2: Age
      1 = Adult horse
      2 = Young (< 6 months)

3: Hospital Number
      - numeric id
      - the case number assigned to the horse
        (may not be unique if the horse is treated > 1 time)

4: rectal temperature
      - linear
      - in degrees celsius.
      - An elevated temp may occur due to infection.
      - temperature may be reduced when the animal is in late shock
      - normal temp is 37.8
      - this parameter will usually change as the problem progresses
           eg. may start out normal, then become elevated because of
               the lesion, passing back through the normal range as the
               horse goes into shock
```

Solution

Read the file "horse-colic.data" or "horse-colic.test"

```python
def loadData(filename):

    dataSet = []
    with open(filename) as fr:
        for i, line in enumerate(fr.readlines()):
            cur_line = []
            now_line = line.strip("\n").strip(" ").split(" ")
            for j in range(len(now_line)):
                if now_line[j] == "?":
                    now_line[j] = -1
            result_line = list(map(float, now_line[23]))
            now_line = list(map(float, now_line[:22]))
            cur_line.append(now_line)
            cur_line.append(result_line)
            dataSet.append(cur_line)
    return dataSet
```

Solution

Initialize parameters

```python
class NeuralNetwork:
    LEARNING_RATE = 0.5

    def __init__(⋯
    ): ⋯

    def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights):
        index = 0
        for h in range(len(self.hidden_layer.neurons)):
            for i in range(self.num_inputs):
                if not hidden_layer_weights:
                    self.hidden_layer.neurons[h].weights.append(random.random())
                else:
                    self.hidden_layer.neurons[h].weights.append(
                        hidden_layer_weights[index]
                    )
                index += 1

    def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(
        self, output_layer_weights
    ):
        index = 0
        for o in range(len(self.output_layer.neurons)):
            for h in range(len(self.hidden_layer.neurons)):
                if not output_layer_weights:
                    self.output_layer.neurons[o].weights.append(random.random())
                else:
                    self.output_layer.neurons[o].weights.append(
                        output_layer_weights[index]
```

Solution

Visualization

```
    def plotInfo(self, training_sets):
        x = list(range(0, self.epochs))
        plt.figure(1)

        plt.xlabel("epochs")
        plt.ylabel("loss")
        loss_epochs = []
        for i in range(self.epochs):
            for k in range(len(training_sets)):
                inputs, outputs = training_sets[k]
                self.train(inputs, outputs)
            loss = self.calculate_total_error(training_sets)
            loss_epochs.append(loss / len(training_sets))
        plt.plot(x, loss_epochs, color="red", linewidth=0.5)
        # plt.show()

        plt.figure(2)
        plt.xlabel("epochs")
        plt.ylabel("accuracy")

```

Solution

Training, testing and visualization framework

```python
def main():
    train_data = loadData("horse-colic.data")
    test_data = loadData("horse-colic.test")

    nn = NeuralNetwork(len(train_data[0][0]), 5, len(train_data[0][1]))
    for i in range(100):
        training_inputs, training_outputs = random.choice(train_data)
        nn.train(training_inputs, training_outputs)

    # print(nn.inspect())
    accuracy = 0
    for i in range(len(test_data)):
        training_inputs, training_outputs = test_data[i]
        nn.train(training_inputs, training_outputs)
        neuron_output = nn.feed_forward(training_inputs)
        if abs(neuron_output[0] - training_outputs[0]) < 0.01:
            accuracy += 1
    accuracy_rate = accuracy / len(test_data)
    print("the accuracy rate is", accuracy_rate)

    nn.plotInfo(test_data)
```

Solution

Forward calculation.

```python
class Neuron:
    def __init__(self, bias):...

    def calculate_output(self, inputs):...

    def calculate_total_net_input(self):...

    # 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):...

    # 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/∂yⱼ) and
    # the derivative of the output with respect to the total net input (dyⱼ/dzⱼ) we can calculate
    # the partial derivative of the error with respect to the total net input.
    # This value is also known as the delta (δ) [1]
    # δ = ∂E/∂zⱼ = ∂E/∂yⱼ * dyⱼ/dzⱼ
    #
    def calculate_pd_error_wrt_total_net_input(self, target_output):
        return (
            self.calculate_pd_error_wrt_output(target_output)
            * self.calculate_pd_total_net_input_wrt_input()
        )

    # The error for each neuron is calculated by the Mean Square Error method:
    def calculate_error(self, target_output):
        return 0.5 * (target_output - self.output) ** 2
```

Solution

Please Finish the Backward Propagation.

```python
    # 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/∂zⱼ

        # 2. Hidden neuron deltas
        # We need to calculate the derivative of the error with respect to the output of each hidden layer neuron
        # dE/dyⱼ = Σ ∂E/∂zⱼ * ∂z/∂yⱼ = Σ ∂E/∂zⱼ * wᵢⱼ
        # ∂E/∂zⱼ = dE/dyⱼ * ∂zⱼ/∂
        # Your Code Here

        # 3. Update output neuron weights
        # ∂Eⱼ/∂wᵢⱼ = ∂E/∂zⱼ * ∂zⱼ/∂wᵢⱼ
        # Δw = α * ∂Eⱼ/∂wᵢ
        # Your Code Here

        # 4. Update hidden neuron weights
        # ∂Eⱼ/∂wᵢ = ∂E/∂zⱼ * ∂zⱼ/∂wᵢ
        # Δw = α * ∂Eⱼ/∂wᵢ
        # Your Code Here
```

Submission

Submission

pack your report `E12_YourNumber.pdf` and source code into zip file `E12_YourNumber.zip`, then send it to `ai_course2021@163.com`.

Optional exercise: Deep Learning

Convolutional Network and Cifar-10 Dataset

- So far we have worked with deep fully-connected networks and backward propagation. Fully-connected networks are a good testbed for experimentation because they are very computationally efficient, but in practice state-of-the-art results in Cifar-10 Dataset use convolutional networks instead.
- You can implement several layer types that are used in convolutional networks, then use these layers to train a convolutional network on the CIFAR-10 dataset.
- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The target is to predict the class for unseen examples.
- I suggest you to refer to https://github.com/maxis42/CS231n/blob/master/assignment2/Con

The End