






 諦聽-

 3   37  1 

RNN遞歸神經網路的詳細推導及C++實現

原创 諦聽- 於2016-10-11 11:31:50 發布 閱讀量1.1w 收藏37 按讚數3

版權

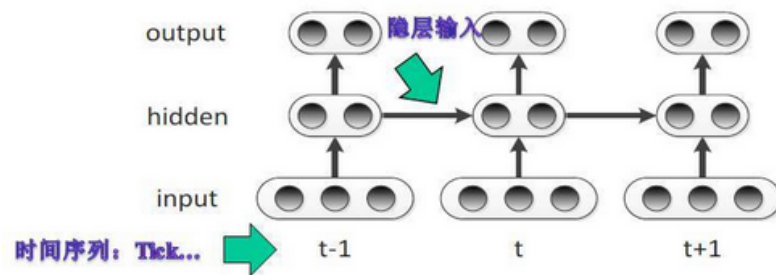
分類專欄：[機器學習筆記](#)



機器學習筆記 專欄收錄該內容

0 訂閱 42 篇文章

[訂閱專欄](#)



諦聽-

[關注](#)

3



37

1



正向传播：

$$h_t^0 = W^{(ih)}x + W^{(hk)}h_{t-1}^1 \quad (1)$$

$$h_t^1 = \sigma(h_t^0) \quad (2) \quad \text{隐层}$$

$$y_t^0 = W^{(ho)}h_t^1 \quad (3)$$

$$y_t^1 = \sigma(y_t^0) \quad (4) \quad \text{输出层}$$

标准误差：

$$e_t = \frac{1}{2}(y_d - y_t)^2 \quad (5) \quad y_d \text{ 为真实值}$$

误差反向传播：

$$\frac{\partial e_t}{\partial y_t^0} = \frac{\partial e_t}{\partial y_t^1} \frac{\partial y_t^1}{\partial y_t^0} = (y_t - y_d) \sigma'(y_t^0) \quad (6) \quad \text{由(5)(4)得到}$$

$$\frac{\partial e_t}{\partial h_t^0} = \left(\frac{\partial e_t}{\partial y_t^0} \frac{\partial y_t^0}{\partial h_t^1} + \frac{\partial e_{t+1}}{\partial h_{t+1}^0} \frac{\partial h_{t+1}^0}{\partial h_t^1} \right) \frac{\partial h_t^1}{\partial h_t^0} = \left(\frac{\partial e_t}{\partial y_t^0} W^{(ho)} + \frac{\partial e_{t+1}}{\partial h_{t+1}^0} W^{(hk)} \right) \sigma'(h_t^0) \quad (7) \quad \text{由(6)(3)(1)得到}$$

$$\Delta W^{(ho)} = \eta \frac{\partial e_t}{\partial W^{(ho)}} = \eta \frac{\partial e_t}{\partial y_t^0} \frac{\partial y_t^0}{\partial W^{(ho)}} = \eta \frac{\partial e_t}{\partial y_t^0} h_t^1 \quad (8) \quad \text{由(7)(3)得到}$$

$$\Delta W^{(hk)} = \eta \frac{\partial e_t}{\partial W^{(hk)}} = \eta \frac{\partial e_t}{\partial h_t^0} \frac{\partial h_t^0}{\partial W^{(hk)}} = \eta \frac{\partial e_t}{\partial h_t^0} h_{t-1}^1 \quad (9) \quad \text{由(7)(1)得到}$$

$$\Delta W^{(ih)} = \eta \frac{\partial e_t}{\partial W^{(ih)}} = \eta \frac{\partial e_t}{\partial h_t^0} \frac{\partial h_t^0}{\partial W^{(ih)}} = \eta \frac{\partial e_t}{\partial h_t^0} x \quad (10) \quad \text{由(7)(1)得到}$$

//让程序自己学会是否需要进位，从而学会加法

```
#include "iostream"
#include "math.h"
#include "stdlib.h"
#include "time.h"
```

1



諦聽-

關注

👍 3



☆ 37

💬 1



```
2
3
4 #include "vector"
5 #include "assert.h"
6 using namespace std;
7
8 #define innode 2      //输入结点数, 将输入2个加数
9 #define hidenode 16   //隐藏结点数, 存储“携带位”
10 #define outnode 1     //输出结点数, 将输出一个预测数字
11 #define alpha 0.1     //学习速率
12 #define binary_dim 8  //二进制数的最大长度
13
14 #define randval(high) ( (double)rand() / RAND_MAX * high )
15 #define uniform_plus_minus_one ( (double)( 2.0 * rand() ) / ((double)RAND_MAX + 1.0) - 1.0 ) //均匀随机分布
16
17
18 int largest_number = ( pow(2, binary_dim) ); //跟二进制最大长度对应的可以表示的最大十进制数
19
20 //激活函数
21 double sigmoid(double x)
22 {
23     return 1.0 / (1.0 + exp(-x));
24 }
25
26 //激活函数的导数, y为激活函数值
27 double dsigmoid(double y)
28 {
29     return y * (1 - y);
30 }
31 //将一个10进制整数转换为2进制数
32 void int2binary(int n, int *arr)
33 {
34     int i = 0;
35     while(n)
36     {
37         arr[i++] = n % 2;
38         n /= 2;
39     }
40     while(i < binary_dim)
41         arr[i++] = 0;
42 }
```



諦聽-

關注

3



37

1



```

43 }
44
45 class RNN
46 {
47 public:
48     RNN();
49     virtual ~RNN();
50     void train();
51
52 public:
53     double w[innode][hiddenode];    //连接输入层与隐藏层的权值矩阵
54     double w1[hiddenode][outnode];  //连接隐藏层与输出层的权值矩阵
55     double wh[hiddenode][hiddenode]; //连接前一时刻的隐含层与现在时刻的隐含层的权值矩阵
56
57     double *layer_0;    //layer 0 输出值, 由输入向量直接设定
58     //double *layer_1;    //layer 1 输出值
59     double *layer_2;    //layer 2 输出值
60 };
61
62 void winit(double w[], int n) //权值初始化
63 {
64     for(int i=0; i<n; i++)
65         w[i] = uniform_plus_minus_one; //均匀随机分布
66 }
67
68 RNN::RNN()
69 {
70     layer_0 = new double[innode];
71     layer_2 = new double[outnode];
72     winit((double*)w, innode * hiddenode);
73     winit((double*)w1, hiddenode * outnode);
74     winit((double*)wh, hiddenode * hiddenode);
75 }
76
77 RNN::~~RNN()
78 {
79     delete layer_0;
80     delete layer_2;
81 }
82
83

```



諦聽-

關注



3



37



1



```
83
84
85 void RNN::train()
86 {
87     int epoch, i, j, k, m, p;
88     vector<double*> layer_1_vector;    //保存隐藏层
89     vector<double> layer_2_delta;    //保存误差关于Layer 2 输出值的偏导
90
91     for(epoch=0; epoch<10000; epoch++) //训练次数
92     {
93         double e = 0.0; //误差
94         for(i=0; i<layer_1_vector.size(); i++)
95             delete layer_1_vector[i];
96         layer_1_vector.clear();
97         layer_2_delta.clear();
98
99         int d[binary_dim];    //保存每次生成的预测值
100         memset(d, 0, sizeof(d));
101
102         int a_int = (int)randval(largest_number/2.0); //随机生成一个加数 a
103         int a[binary_dim];
104         int2binary(a_int, a);    //转为二进制数
105
106         int b_int = (int)randval(largest_number/2.0); //随机生成另一个加数 b
107         int b[binary_dim];
108         int2binary(b_int, b);    //转为二进制数
109
110         int c_int = a_int + b_int;    //真实的和 c
111         int c[binary_dim];
112         int2binary(c_int, c);    //转为二进制数
113
114         double *layer_1 = new double[hiddenode];
115         for(i=0; i<hiddenode; i++)    //在0时刻是没有之前的隐含层的，所以初始化一个全为0的
116             layer_1[i] = 0;
117         layer_1_vector.push_back(layer_1);
118
119         //正向传播
120         for(p=0; p<binary_dim; p++)    //循环遍历二进制数组，从最低位开始
121         {
122             layer_0[0] = a[p];
123
```



諦聽-

關注

👍 3



★ 37

💬 1



```

124     layer_0[1] = b[p];
125     double y = (double)c[p];           //实际值
126     layer_1 = new double[hiddenode];   //当前隐含层
127
128     for(j=0; j<hiddenode; j++)
129     {
130         //输入层传播到隐含层
131         double o1 = 0.0;
132         for(m=0; m<innode; m++)
133             o1 += layer_0[m] * w[m][j];
134
135         //之前的隐含层传播到现在的隐含层
136         double *layer_1_pre = layer_1_vector.back();
137         for(m=0; m<hiddenode; m++)
138             o1 += layer_1_pre[m] * wh[m][j];
139
140         layer_1[j] = sigmoid(o1);       //隐藏层各单元输出
141     }
142
143     for(k=0; k<outnode; k++)
144     {
145         //隐藏层传播到输出层
146         double o2 = 0.0;
147         for(j=0; j<hiddenode; j++)
148             o2 += layer_1[j] * w1[j][k];
149         layer_2[k] = sigmoid(o2);       //输出层各单元输出
150     }
151
152     d[p] = (int)floor(layer_2[0] + 0.5); //记录预测值
153     layer_1_vector.push_back(layer_1);   //保存隐藏层, 以便下次计算
154
155     //保存标准误差关于输出层的偏导
156     layer_2_delta.push_back( (y - layer_2[0]) * dsigmoid(layer_2[0]) );
157     e += fabs(y - layer_2[0]);           //误差
158 }
159
160 //误差反向传播
161
162 //隐含层偏差, 通过当前之后一个时间点
163 double *layer_1_delta = new double[hiddenode];
164

```



諦聽-

關注



3



37



1



```

165
166 double *layer_1_future_delta = new double[hiddenode]; //当前时间之后的一个隐藏层误差
167 for(j=0; j<hiddenode; j++)
168     layer_1_future_delta[j] = 0;
169 for(p=hiddenode-1; p>=0; p--)
170 {
171     layer_0[0] = a[p];
172     layer_0[1] = b[p];
173
174     layer_1 = layer_1_vector[p+1]; //当前隐藏层
175     double *layer_1_pre = layer_1_vector[p]; //前一个隐藏层
176
177     for(k=0; k<outnode; k++) //对于网络中每个输出单元, 更新权值
178     {
179         //更新隐含层和输出层之间的连接权
180         for(j=0; j<hiddenode; j++)
181             w1[j][k] += alpha * layer_2_delta[p] * layer_1[j];
182     }
183
184     for(j=0; j<hiddenode; j++) //对于网络中每个隐藏单元, 计算误差项, 并更新权值
185     {
186         layer_1_delta[j] = 0.0;
187         for(k=0; k<outnode; k++)
188             layer_1_delta[j] += layer_2_delta[p] * w1[j][k];
189         for(k=0; k<hiddenode; k++)
190             layer_1_delta[j] += layer_1_future_delta[k] * wh[j][k];
191
192         //隐含层的校正误差
193         layer_1_delta[j] = layer_1_delta[j] * dsigmoid(layer_1[j]);
194
195         //更新输入层和隐含层之间的连接权
196         for(k=0; k<innode; k++)
197             w[k][j] += alpha * layer_1_delta[j] * layer_0[k];
198
199         //更新前一个隐含层和现在隐含层之间的权值
200         for(k=0; k<hiddenode; k++)
201             wh[k][j] += alpha * layer_1_delta[j] * layer_1_pre[k];
202     }
203 }
204

```



諦聽-

關注



3



37



1




```
205         if(p == binary_dim - 1)
206             delete layer_1_future_delta;
207         layer_1_future_delta = layer_1_delta;
208     }
209     delete layer_1_future_delta;
210
211     if(epoch % 1000 == 0)
212     {
213         cout << "error: " << e << endl;
214         cout << "pred: " ;
215         for(k=binary_dim-1; k>=0; k--)
216             cout << d[k];
217         cout << endl;
218
219         cout << "true: " ;
220         for(k=binary_dim-1; k>=0; k--)
221             cout << c[k];
222         cout << endl;
223
224         int out = 0;
225         for(k=binary_dim-1; k>=0; k--)
226             out += d[k] * pow(2, k);
227         cout << a_int << " + " << b_int << " = " << out << endl << endl;
228     }
229 }
230 }
231
232
233 int main()
234 {
235     srand(time(NULL));
236     RNN rnn;
237     rnn.train();
238     return 0;
239 }
```



諦聽-

關注



3



37



1



```
error: 4.30389
pred: 01100000
true: 10000100
111 + 21 = 96

error: 3.20899
pred: 01111001
true: 01111101
5 + 120 = 121

error: 2.06103
pred: 10101110
true: 10101110
51 + 123 = 174

error: 0.380599
pred: 00100101
true: 00100101
32 + 5 = 37
```

參考:

<http://blog.csdn.net/zzukun/article/details/49968129>

<http://www.cnblogs.com/wb-DarkHorse/archive/2012/12/12/2815393.html>

C++實作的神經網路

09-14

雖然用C++實現神經網路會比較麻煩，不過如果注意到以下下的trick，在用C++實現神經網路時會舒服很多，程式碼也會非常簡潔，核心程式碼在百行左右也不是問題。C++沒有內建的向量、矩陣函式...

文字辨識網路C RNN

bblingbbling的博客 1932

CRNN文字识别网络简介网络结构CNN层LSTM层CTC Loss代码实现 简介 CRNN，全称Convolutional Recurrent Neural Network，卷积循环神经网络。它是一种基于图像的序列识别网络，可以对不...

1 条评论



dxk_093812 热评 博主，你好，我想请问第155行和第190行代码中dsigmoid()函数的参数有没有写错啊，有点不懂，好像和公式中的不一样。

写评论

RNN_递归神经网络

5-1

此外,RNN可以表现出许多不同的形式:一对一(文本生成)、多对一(顺序图像分类)、一对多(图像描述)和多对多(机器翻译)。 1.3 RNN模型 递归神经网络,带有一个指向自身的环,用来表示它可以传递当...

RNN递归神经网络_n_neurons

4-20

#按照规定,outputs是最后一层的输出,即为[batch_size,step,n_neurons]n_neurons是神经元的个数 #按照规定,final_state是每一层的最后一个step的输出,其实本程序只是用了一个隐藏层,因为是LSTM...



諦聽

關注

👍 3



★ 37

💬 1



机器学习概念、步骤、分类和实践 **最新发布**

Python老吕的博客 688

机器学习是指机器通过统计学算法，对大量历史数据进行学习，进而利用生成的经验模型指导业务。它是一门多领域交叉学科，专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或...

RNN模型（大白话+公式推导）

Null 2228

标准神经网络不适合处理带时间序列的网络场景（标准神经网络不适合处理带时间序列的网络场景）

深度学习——神经网络之RNN循环(递归)神经网络_递归循环神经网络-CSDN...

5-1

神经网络之RNN循环(递归)神经网络) 1、什么是循环神经网络 我们之前学习了全连接神经网络DNN,还学了卷积神经网络CNN,为什么还需要RNN递归神经网络?有以下两种原因: BP神经网络和CNN...

深入理解递归神经网络(RNN)_递归神经网络 学习能力

4-27

递归神经网络(Recurrent Neural Networks,简称RNN)是深度学习领域中一种强大的神经网络结构,它在自然语言处理、语音识别、时间序列预测等任务中表现出色。本博客将带你深入了解RNN的工作...

基于C++的 RNN/lstm神经网络算法（不调用外源库）

修行之路 445

基于C++的 RNN/lstm神经网络算法（不调用外源库）

『OCR_Recognition』CRNN

libo-coder 4563

文章目录前言一、CRNN1.1 CRNN 介绍1.2 CRNN 网络结构1.2.1 CNN1.2.2 Map-to-Sequence1.2.3 RNN1.2.4 CTC Loss1.2.4.1 序列合并机制1.2.4.2 训练阶段参考链接 前言 现今基于深度学习的端...

递归神经网络RNN总结_传统递归神经网络(rnn)主要问题

4-27

2、RNN局限性问题 I am Chinese, I Love China 递归神经网络参数太多,信息量冗余(因为最后的预测可能只 依赖它最近的词,但我们输入的时候是所有的词,所以信息量冗余)、梯度消失或者爆炸。

递归神经网络(RNN)简介

4-15

先别急着能理解RNN,我们来点轻松的,先介绍这样的序列化网络结构包含的参数记号: 网络某一时刻的输入xtxt,和之前介绍的多层感知器的输入一样,xtxt是一个nn维向量,不同的是递归网络的输入将是...

递归神经网络(超详细|附训练代码)

qq_73462282的博客 1648

另外两个词『Germany』和『France』因为表示的都是地点，它们的向量与上面两句话的向量的距离，就比另外两个表示时间的词『Monday』和『Tuesday』的向量的距离近得多。比如，在左边的...

C++元编程——计算链和RNN

Dr_Jack的博客 550

反向传播时候有个计算链，误差传播时也是反向走过各个计算链，所以这个计算链的概念很重要。下面是这个计算链的实现cal_chain.hpp： 下面看一看用这个计算链实现RNN： 这个用法当然不是R...

rnn循环神经网络基本原理

5-1

3.5 反向传播求参推导 4. 传统RNN的优缺点 5. rnn简单代码实现 5.1 创建数据相关代码 5.2 完整代码 5.3 效果展示 学习视频链接 RNN循环神经网络 RNN、pytorch、人工智能、神经网络、深度学习 ...

循环神经网络(RNN)_c语言实现rnn

4-25

1. 关于RNN的神经网络 1.1 结构图 其中,xt是t时刻的输入值,ht是t时刻的隐藏层的值,yt是t时刻的输出层的值。U,V,W分别是输入,隐藏层,输出的权重矩阵。从图上的参数(U,V,W)的下标不变可以看出,R...

C++元编程——双向RNN

Dr_Jack的博客 152

可以看到，训练只要训练6000次结果就稳定了，但是问题在于出来的结果和期望结果有一定的差距。可以看出输出结果是各个输入数据的均值，这个就非常尴尬了。我觉得这和我的训练方式不当可...

Pytorch实现RNN预测模型并使用C++相应的ONNX模

Pytorch实现RNN预测模型并使用C++相应的ONNX模型推理。



諦聽-

關注

3

37

1



LSTM神经网络的详细推导及C++实现 热门推荐	新博客：https://aping-dev.com/ 2万+
LSTM隐层神经元结构：LSTM隐层神经元详细结构： //让程序自己学会是否需要进位，从而学会加法#include "iostream" #include "math.h" #include "stdlib.h" #include "time.h" #include "vector" #inc...	
simple-rnn:具有Boost的C ++中的递归神经网络	05-21
简单 具有Boost的C ++中的递归神经网络 这是Alex Graves令人难以置信的多维rnnlib的部分简化版本。 它使用较小的功能集，并降低了复杂性，可以再次使用一维数据。 有关其实现的更多详细信...	
frnn:C ++快速递归神经网络库	05-16
快速神经网络 概述 frnn是一个用c ++编写的快速循环神经网络库，它使用CUDA和OpenMP进行并行化。 它支持RNN的所有可能的配置。 将为CPU-GPU和纯CPU系统提供支持，并将根据其功能确...	
LSTM C++源代码	01-30
https://blog.csdn.net/u012465304/article/details/82656803 我的博客上的LSTM例程，用c++写的，由于找我私聊要的人太多了，就上传到这个上面，大家可自行下载	
RNN与LSTM源代码	12-18
消费者请注意，本资源是分别用RNN(循环神经网络)和LSTM(长短记忆网络)编写的MATLAB的案例，内部RNN.m和LSTM.m文件程序可以直接运行，内部已包含所需功能函数，如过不能直接运行请...	
RNN实现源码	04-29
RNN的实现源码	
如何用PyTorch实现递归神经网络？	02-25
这些模型大多数将语言视为单调的单词或字符序列，并使用一种称为循环神经网络（recurrentneuralnetwork/RNN）的模型来处理该序列。但是许多语言学家认为语言最好被理解为具有树形结构的层...	
RNN代码_recurrentnetwork_RNN_RNN神经网络_递归神经网络_	10-01
RNN递归神经网络，应用于自然语言处理等大数据处理领域	
RNN-LSTM卷积神经网络Matlab实现.zip	09-07
RNN卷积神经网络，LSTM，使用matlab实现，简单的数据拟合	
递归神经网络(RNN)	05-01
遞歸神經網路（Recurrent Neural Network，RNN）是一類經典的神經網路模型，它專門用於處理序列資料。相對於傳統的前饋神經網絡，RNN的最大特徵在於它能夠保持一定的記憶狀態，而這個記...	

「相關推薦」對你有幫助麼？



非常沒幫助



沒幫助



一般



有幫助



非常有幫助

關於我們 招賢納士 商務合作 尋求報道 400-660-0108 kefu@csdn.net 網路客服 工作時間 8:30-22:00

公安備案號11010502030143 京ICP備19004658號 京網文〔2020〕1039-165號 經營性網站備案資訊 北京網路違法與不良資訊檢舉中心

家長監護 網路110警



諦聽-

關注



3



37



1





諦聽-

部落格等級 碼齡11年

1183
原創

533
讚

2232
收藏

718
粉絲

關注

私訊



搜博主文章



熱門文章

python---將多條曲線畫在一幅圖 48329

如何在Excel裡將兩個不同橫座標的圖做在一張圖上 39804

Qt---多種方式讀寫二進位檔案 39802

Qt---自備的資料庫QSQLITE 38943

Qt---為視窗新增捲軸：QScrollArea
37247

分類專欄



k8s

1篇



劍指offer

8篇



編譯原理之美

4篇



諦聽-

關注



3






37



1



	演算法引論	6篇
	遞迴與分治	40篇
	動態規劃	57篇



最新評論

- 非單位時間任務安排問題
2301_79799934: 鑑定為依托謝特
- OpenGL---載入obj模型
LM_2233: 麻煩問一下主程式35行報錯誤是為什麼，顯示const char的型別實參char ...
- 三維圖形的幾何變換
weixin_73949247: 樓主享有太廟。
- C#---最速下降+牛頓迭代求二元非線性方...
挖坑小助手: 特定的方程式是指什麼呢，以Fx(x)為例，似乎是指求導方程一的分子: ...
- 快速排序-- C 語言
CSDN-Ada助手: 哇, 你的文章品質真不錯, 值得學習! 不過這麼高品質的文章, 還值 ...

您願意向朋友推薦「部落格詳情頁」嗎?



強烈不推薦 不推薦 一般般 推薦 強烈推薦

最新文章

- dfs & bfs
- kubeadm init 逾時
- 旋轉數組的最小數字

2021年 13篇 2019年 144篇

 諦聽-

關注

 3   37  1 

2018年 250篇

2017年 250篇

2016年 388篇

2015年 213篇



NT\$89


NT\$65


NT\$950


NT\$30


 詠聽-

關注

 3



 37

 1

