短信验证码接口


90平米怎么装修

图灵赠书——程序员11月书单 【力荐】写给想成为前端工程师的同学们！ 异步赠书:kafka/TensorFlow机器学习 每周荐书：OpenCV、自然语言、SpringBoot2

LSTM神经网络的详细推导及C++实现

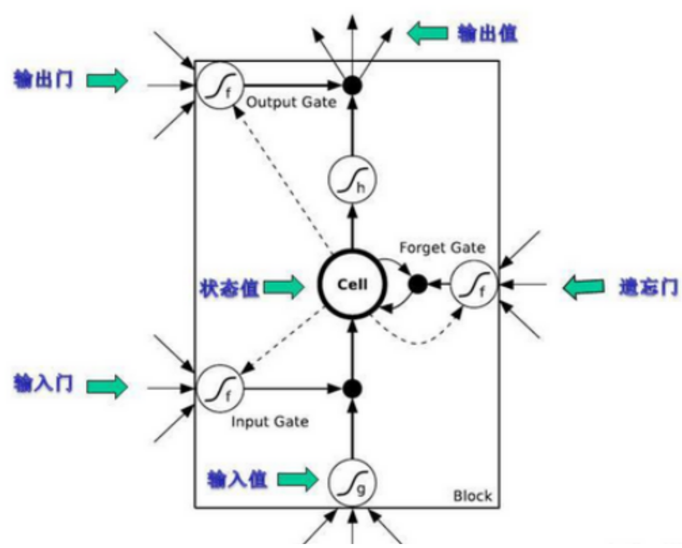
2016-10-12 23:53 6428人阅读 评论(15)

分类：

机器学习笔记（38）

版权声明：本文为博主原创文章，转载请注明出处。

LSTM隐层神经元结构：



LSTM隐层神经元详细结构：

访问： 287748次
积分： 10366
等级： BLOG 7
排名： 第1783名
原创： 775篇 转载： 43篇
译文： 4篇 评论： 113条

文章搜索

文章分类

关闭
开发一个app多少钱
全透明手机价格
好手机排行榜
ubuntu

Python (12)
STL (20)
Qt (24)
VC (18)
OpenGL (12)
优化 (11)
实践 (19)
字符串 (12)
30天自制OS笔记 (11)
机器学习笔记 (39)
鸟哥的Linux私房菜笔记 (12)
大数据 (7)
图形学 (20)
游戏 (21)
COM技术内幕笔记 (8)
Arcgis (12)
openCV (8)
视觉 (3)
ros机器人操作系统 (2)

文章存档

2017年11月 (6)
2017年10月 (35)
2017年09月 (36)
2017年08月 (17)
2017年07月 (20)

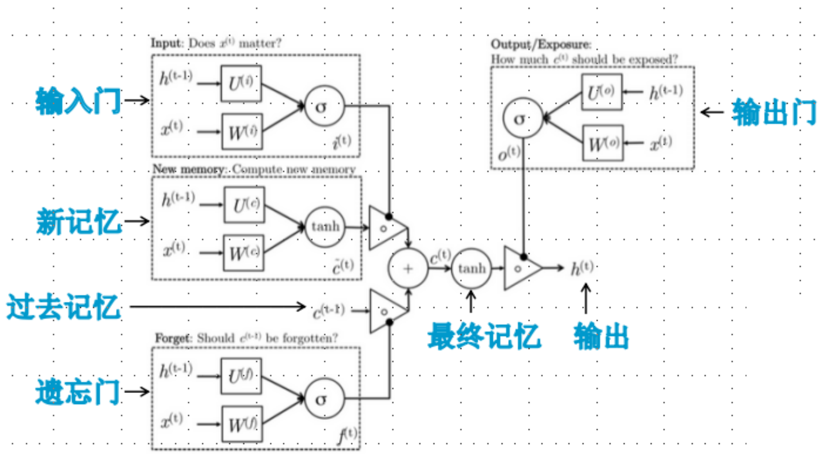
展开

阅读排行

Qt---自带的数据库QSQLITE (11242)
Qt---为窗体添加滚动条：QScr... (7647)
LSTM神经网络的详细推导及C... (6407)
Qt---多种方式读写二进制文件 (6294)
Qt---基于TCP聊天室 (4982)
vs---vs2015 + qt5.7环境配置 (4740)
Qt---Label显示图片，自动设... (4650)
遗传算法---实数编码方式 (3913)
stata---门限回归 (3551)
RNN递归神经网络的详细推导... (3391)

评论排行

基于启发式的分割算法 (15)
LSTM神经网络的详细推导及C... (15)
非单位时间任务安排问题 (11)
RNN递归神经网络的详细推导... (10)
Qt---基于TCP聊天室 (8)
租用游艇问题 (4)



输入门：

$$I_t^0 = W^{(i)}x_t + U^{(i)}h_{t-1} \tag{1}$$

$$I_t^1 = \sigma(I_t^0) \tag{2}$$

遗忘门：

$$F_t^0 = W^{(f)}x_t + U^{(f)}h_{t-1} \tag{3}$$

$$F_t^1 = \sigma(F_t^0) \tag{4}$$

输出门：

$$O_t^0 = W^{(o)}x_t + U^{(o)}h_{t-1} \tag{5}$$

$$O_t^1 = \sigma(O_t^0) \tag{6}$$

输入值，也就是新“记忆”：

$$G_t^0 = W^{(g)}x_t + U^{(g)}h_{t-1} \tag{7}$$

$$G_t^1 = \sigma(G_t^0) \tag{8}$$

关闭

状态值，也就是“最终记忆”：

$$S_t = F_t^1 * S_{t-1} + I_t^1 * G_t \tag{9}$$

输出值：

$$h_t = O_t^1 * \tanh(S_t) \tag{10}$$

输出层：

$$y_t^0 = W^{(out)} h_t \tag{11}$$

$$y_t^1 = \sigma(y_t^0) \tag{12}$$

标准差：

$$e_t = \frac{1}{2}(y_d - y_t^1)^2 \tag{13}$$

反向传播：

$$\frac{\partial e_t}{\partial y_t^0} = (y_t^1 - y_d)\sigma^{'}(y_t^0) \tag{14}$$ 由(13)(12)得到

$$\frac{\partial e_t}{\partial h_t} = \frac{\partial e_t}{\partial y_t^0}\frac{\partial y_t^0}{\partial h_t} + \frac{\partial e_{t+1}}{\partial I_{t+1}^0}\frac{\partial I_{t+1}^0}{\partial h_t} + \frac{\partial e_{t+1}}{\partial F_{t+1}^0}\frac{\partial F_{t+1}^0}{\partial h_t} + \frac{\partial e_{t+1}}{\partial O_{t+1}^0}\frac{\partial O_{t+1}^0}{\partial h_t} + \frac{\partial e_{t+1}}{\partial G_{t+1}^0}\frac{\partial G_{t+1}^0}{\partial h_t}$$

$$= \frac{\partial e_t}{\partial y_t^0} W^{(out)} + \frac{\partial e_{t+1}}{\partial I_{t+1}^0} U^{(i)} + \frac{\partial e_{t+1}}{\partial F_{t+1}^0} U^{(f)} + \frac{\partial e_{t+1}}{\partial O_{t+1}^0} U^{(o)} + \frac{\partial e_{t+1}}{\partial G_{t+1}^0} U^{(g)} \tag{15}$$ 由(14)(11)(1)(3)(5)(7)得到

$$\frac{\partial e_t}{\partial O_t^0} = \frac{\partial e_t}{\partial h_t}\frac{\partial h_t}{\partial O_t^0} = \frac{\partial e_t}{\partial h_t}\tanh(S_t)\sigma^{'}(O_t^0) \tag{16}$$ 由(15)(10)(6)得到

$$\frac{\partial e_t}{\partial S_t} = \frac{\partial e_t}{\partial h_t}\frac{\partial h_t}{\partial S_t} + \frac{\partial e_{t+1}}{\partial S_{t+!}}\frac{\partial S_{t+1}}{\partial S_t} = \frac{\partial e_t}{\partial h_t} O_t^1 \tanh^{'}(S_t) + \frac{\partial e_{t+1}}{\partial S_{t+1}} F_{t+1}^1 \tag{17}$$ 由(15)(10)得到

$$\frac{\partial e_t}{\partial F_t^0} = \frac{\partial e_t}{\partial S_t}\frac{\partial S_t}{\partial F_t^0} = \frac{\partial e_t}{\partial S_t} S_{t-1}\sigma^{'}(F_t^0) \tag{18}$$ 由(17)(9)(4)得到

$$\frac{\partial e_t}{\partial I_t^0} = \frac{\partial e_t}{\partial S_t}\frac{\partial S_t}{\partial I_t^0} = \frac{\partial e_t}{\partial S_t} G_t^1\sigma^{'}(I_t^0) \tag{19}$$ 由(17)(9)(2)得到

$$\frac{\partial e_t}{\partial G_t^0} = \frac{\partial e_t}{\partial S_t}\frac{\partial S_t}{\partial G_t^0} = \frac{\partial e_t}{\partial S_t} I_t^1\sigma^{'}(G_t^0) \tag{20}$$ 由(17)(9)(8)得到

关闭

RL强化学习 C++实现 (3)
VC6.0 编译Json生成lib文件 (3)
Arcgis---使用sql数据库中的数... (3)
DBN深度信念网 C++实现 (3)

推荐文章

* 【GitChat】每日精选20171109——双 11 大前端工程师读书清单
* 改做人工智能之前，90%的人都没能给自己定位
* 想入行 AI，别让那些技术培训坑了你...
* 大型技术组织 DevOps 转型经验总结
* 两款敏捷工具，治好你碎片化交付硬伤
* 低学历又如何？这样的程序员照样可以逆袭

最新评论

leetcode---jump-game---贪心
TyiTguoQ ：不正确 5 0 0 0 4 0 1 试试

基于启发式的分割算法
将明丶 ：@qq_29944221:是我理解错了吗？

基于启发式的分割算法
将明丶 ：@u012319493:可是这个Delta*Eta没看懂啊

LSTM神经网络的详细推导及C++实现
zhouhongyue1976 ：double dtanh2(double y){return 1.0 - y * y;}//190 ...

基于启发式的分割算法
步悠然 ：@qq_29944221:是的，具体步骤已补充

RNN递归神经网络的详细推导及C++实现
步悠然 ：@qq_34385798:自动学习加法

基于启发式的分割算法
将明丶 ：您好，请教一下 double Eta = 4.19 * log(len) - 11.54; /...

世界名画陈列馆问题
左金都御史 ：大佬，5 7 ，你输出的图明显没有覆盖完。。。5 6 也不对。。

RNN递归神经网络的详细推导及C++实现
qq_34385798 ：所以这个实现了什么功能呢？

Arcgis---使用sql数据库中的数据刷新图层
技术小牛牛 ：哦哦，我这边试试，谢谢您啦

$$\Delta W^{(out)} = \eta \frac{\partial e_t}{\partial W^{(out)}} = \eta \frac{\partial e_t}{\partial y_t^0} \frac{\partial y_t^0}{\partial W^{(out)}} = \eta \frac{\partial e_t}{\partial y_t^0} h_t \qquad (21) \qquad \text{由(14)(11)得到}$$

$$\Delta W^{(i)} = \eta \frac{\partial e_t}{\partial W^{(i)}} = \eta \frac{\partial e_t}{\partial I_t^0} \frac{\partial I_t^0}{\partial W^{(i)}} = \eta \frac{\partial e_t}{\partial I_t^0} x_t \qquad (22) \qquad \text{由(19)(1)得到}$$

$$\Delta U^{(i)} = \eta \frac{\partial e_t}{\partial U^{(i)}} = \eta \frac{\partial e_t}{\partial I_t^0} \frac{\partial I_t^0}{\partial U^{(i)}} = \eta \frac{\partial e_t}{\partial I_t^0} h_{t-1} \qquad (23) \qquad \text{由(19)(1)得到}$$

$$\Delta W^{(f)} = \eta \frac{\partial e_t}{\partial W^{(f)}} = \eta \frac{\partial e_t}{\partial F_t^0} \frac{\partial F_t^0}{\partial W^{(f)}} = \eta \frac{\partial e_t}{\partial F_t^0} x_t \qquad (24) \qquad \text{由(18)(3)得到}$$

$$\Delta U^{(f)} = \eta \frac{\partial e_t}{\partial U^{(f)}} = \eta \frac{\partial e_t}{\partial F_t^0} \frac{\partial F_t^0}{\partial U^{(f)}} = \eta \frac{\partial e_t}{\partial F_t^0} h_{t-1} \qquad (25) \qquad \text{由(18)(3)得到}$$

$$\Delta W^{(o)} = \eta \frac{\partial e_t}{\partial W^{(o)}} = \eta \frac{\partial e_t}{\partial O_t^0} \frac{\partial O_t^0}{\partial W^{(o)}} = \eta \frac{\partial e_t}{\partial O_t^0} x_t \qquad (26) \qquad \text{由(16)(5)得到}$$

$$\Delta U^{(o)} = \eta \frac{\partial e_t}{\partial U^{(o)}} = \eta \frac{\partial e_t}{\partial O_t^0} \frac{\partial O_t^0}{\partial U^{(o)}} = \eta \frac{\partial e_t}{\partial O_t^0} h_{t-1} \qquad (27) \qquad \text{由(18)(3)得到}$$

$$\Delta W^{(g)} = \eta \frac{\partial e_t}{\partial W^{(g)}} = \eta \frac{\partial e_t}{\partial G_t^0} \frac{\partial G_t^0}{\partial W^{(g)}} = \eta \frac{\partial e_t}{\partial G_t^0} x_t \qquad (28) \qquad \text{由(16)(5)得到}$$

$$\Delta U^{(g)} = \eta \frac{\partial e_t}{\partial U^{(g)}} = \eta \frac{\partial e_t}{\partial G_t^0} \frac{\partial G_t^0}{\partial U^{(g)}} = \eta \frac{\partial e_t}{\partial G_t^0} h_{t-1} \qquad (29) \qquad \text{由(18)(3)得到}$$

```
//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"
#include "vector"
#include "assert.h"
using namespace std;

#define innode  2       //输入结点数，将输入2个加数
#define hidenode  26    //隐藏结点数，存储“携带位”
#define outnode  1      //输出结点数，将输出一个预测数字
#define alpha  0.1      //学习速率
#define binary_dim 8    //二进制数的最大长度

#define randval(high) ( (double)rand() / RAND_MAX * high )
#define uniform_plus_minus_one ( (double)( 2.0 * rand() ) / ((double)RAND_MAX + 1.0) - 1.0 )  //均匀随机分


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

//激活函数
double sigmoid(double x)
{
```

关闭

```
    return 1.0 / (1.0 + exp(-x));
}

//激活函数的导数，y为激活函数值
double dsigmoid(double y)
{
    return y * (1.0 - y);
}

//tanh的导数，y为tanh值
double dtanh(double y)
{
    y = tanh(y);
    return 1.0 - y * y;
}

//将一个10进制整数转换为2进制数
void int2binary(int n, int *arr)
{
    int i = 0;
    while(n)
    {
        arr[i++] = n % 2;
        n /= 2;
    }
    while(i < binary_dim)
        arr[i++] = 0;
}

class RNN
{
public:
    RNN();
    virtual ~RNN();
    void train();

public:
    double W_I[innode][hidenode];     //连接输入与隐含层单元中输入门的权值矩阵
    double U_I[hidenode][hidenode];   //连接上一隐层输出与本隐含层单元中输入门的权值矩阵
    double W_F[innode][hidenode];     //连接输入与隐含层单元中遗忘门的权值矩阵
    double U_F[hidenode][hidenode];   //连接上一隐含层与本隐含层单元中遗忘门的权值矩阵
    double W_O[innode][hidenode];     //连接输入与隐含层单元中遗忘门的权值矩阵
    double U_O[hidenode][hidenode];   //连接上一隐含层与现在时刻的隐含层的权值矩阵
    double W_G[innode][hidenode];     //用于产生新记忆的
    double U_G[hidenode][hidenode];   //用于产生新记忆的权值矩阵
    double W_out[hidenode][outnode];  //连接隐层与输出层的权值矩阵

    double *x;             //layer 0 输出值，由输入向量直接设定
    //double *layer_1;     //layer 1 输出值
    double *y;             //layer 2 输出值
};

void winit(double w[], int n) //权值初始化
{
```

关闭

```
    for(int i=0; i<n; i++)
        w[i] = uniform_plus_minus_one;  //均匀随机分布
}

RNN::RNN()
{
    x = new double[innode];
    y = new double[outnode];
    winit((double*)W_I, innode * hidenode);
    winit((double*)U_I, hidenode * hidenode);
    winit((double*)W_F, innode * hidenode);
    winit((double*)U_F, hidenode * hidenode);
    winit((double*)W_O, innode * hidenode);
    winit((double*)U_O, hidenode * hidenode);
    winit((double*)W_G, innode * hidenode);
    winit((double*)U_G, hidenode * hidenode);
    winit((double*)W_out, hidenode * outnode);
}

RNN::~RNN()
{
    delete x;
    delete y;
}

void RNN::train()
{
    int epoch, i, j, k, m, p;
    vector<double*> I_vector;      //输入门
    vector<double*> F_vector;      //遗忘门
    vector<double*> O_vector;      //输出门
    vector<double*> G_vector;      //新记忆
    vector<double*> S_vector;      //状态值
    vector<double*> h_vector;      //输出值
    vector<double> y_delta;        //保存误差关于输出层的偏导

    for(epoch=0; epoch<11000; epoch++)  //训练次数
    {
        double e = 0.0;  //误差

        int predict[binary_dim];             //保存每次生成的预测值
        memset(predict, 0, sizeof(predict));

        int a_int = (int)randval(largest_number/2.0);  //随机生
        int a[binary_dim];
        int2binary(a_int, a);                //转为二进制数

        int b_int = (int)randval(largest_number/2.0);  //随机生成另一个加数 b
        int b[binary_dim];
        int2binary(b_int, b);                //转为二进制数

        int c_int = a_int + b_int;           //真实的和 c
        int c[binary_dim];
        int2binary(c_int, c);                //转为二进制数
```

关闭

```
    //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
    double *S = new double[hidenode];     //状态值
    double *h = new double[hidenode];     //输出值

    for(i=0; i<hidenode; i++)
    {
        S[i] = 0;
        h[i] = 0;
    }
    S_vector.push_back(S);
    h_vector.push_back(h);

    //正向传播
    for(p=0; p<binary_dim; p++)           //循环遍历二进制数组，从最低位开始
    {
        x[0] = a[p];
        x[1] = b[p];
        double t = (double)c[p];          //实际值
        double *in_gate = new double[hidenode];     //输入门
        double *out_gate = new double[hidenode];    //输出门
        double *forget_gate = new double[hidenode]; //遗忘门
        double *g_gate = new double[hidenode];      //新记忆
        double *state = new double[hidenode];       //状态值
        double *h = new double[hidenode];           //隐层输出值

        for(j=0; j<hidenode; j++)
        {
            //输入层转播到隐层
            double inGate = 0.0;
            double outGate = 0.0;
            double forgetGate = 0.0;
            double gGate = 0.0;
            double s = 0.0;

            for(m=0; m<innode; m++)
            {
                inGate += x[m] * W_I[m][j];
                outGate += x[m] * W_O[m][j];
                forgetGate += x[m] * W_F[m][j];
                gGate += x[m] * W_G[m][j];
            }

            double *h_pre = h_vector.back();
            double *state_pre = S_vector.back();
            for(m=0; m<hidenode; m++)
            {
                inGate += h_pre[m] * U_I[m][j];
                outGate += h_pre[m] * U_O[m][j];
                forgetGate += h_pre[m] * U_F[m][j];
                gGate += h_pre[m] * U_G[m][j];
            }

            in_gate[j] = sigmoid(inGate);
```

关闭

```
            out_gate[j] = sigmoid(outGate);
            forget_gate[j] = sigmoid(forgetGate);
            g_gate[j] = sigmoid(gGate);

            double s_pre = state_pre[j];
            state[j] = forget_gate[j] * s_pre + g_gate[j] * in_gate[j];
            h[j] = in_gate[j] * tanh(state[j]);
        }


        for(k=0; k<outnode; k++)
        {
            //隐藏层传播到输出层
            double out = 0.0;
            for(j=0; j<hidenode; j++)
                out += h[j] * W_out[j][k];
            y[k] = sigmoid(out);           //输出层各单元输出
        }

        predict[p] = (int)floor(y[0] + 0.5);   //记录预测值

        //保存隐藏层，以便下次计算
        I_vector.push_back(in_gate);
        F_vector.push_back(forget_gate);
        O_vector.push_back(out_gate);
        S_vector.push_back(state);
        G_vector.push_back(g_gate);
        h_vector.push_back(h);

        //保存标准误差关于输出层的偏导
        y_delta.push_back( (t - y[0]) * dsigmoid(y[0]) );
        e += fabs(t - y[0]);          //误差
    }

    //误差反向传播

    //隐含层偏差，通过当前之后一个时间点的隐含层误差和当前输出层的误差计算
    double h_delta[hidenode];
    double *O_delta = new double[hidenode];
    double *I_delta = new double[hidenode];
    double *F_delta = new double[hidenode];
    double *G_delta = new double[hidenode];
    double *state_delta = new double[hidenode];
    //当前时间之后的一个隐藏层误差
    double *O_future_delta = new double[hidenode];
    double *I_future_delta = new double[hidenode];
    double *F_future_delta = new double[hidenode];
    double *G_future_delta = new double[hidenode];
    double *state_future_delta = new double[hidenode];
    double *forget_gate_future = new double[hidenode];
    for(j=0; j<hidenode; j++)
    {
        O_future_delta[j] = 0;
        I_future_delta[j] = 0;
```

关闭

```
        F_future_delta[j] = 0;
        G_future_delta[j] = 0;
        state_future_delta[j] = 0;
        forget_gate_future[j] = 0;
    }
    for(p=binary_dim-1; p>=0 ; p--)
    {
        x[0] = a[p];
        x[1] = b[p];

        //当前隐藏层
        double *in_gate = I_vector[p];     //输入门
        double *out_gate = O_vector[p];    //输出门
        double *forget_gate = F_vector[p]; //遗忘门
        double *g_gate = G_vector[p];      //新记忆
        double *state = S_vector[p+1];     //状态值
        double *h = h_vector[p+1];         //隐层输出值

        //前一个隐藏层
        double *h_pre = h_vector[p];
        double *state_pre = S_vector[p];

        for(k=0; k<outnode; k++)  //对于网络中每个输出单元，更新权值
        {
            //更新隐含层和输出层之间的连接权
            for(j=0; j<hidenode; j++)
                W_out[j][k] += alpha * y_delta[p] * h[j];
        }

        //对于网络中每个隐藏单元，计算误差项，并更新权值
        for(j=0; j<hidenode; j++)
        {
            h_delta[j] = 0.0;
            for(k=0; k<outnode; k++)
            {
                h_delta[j] += y_delta[p] * W_out[j][k];
            }
            for(k=0; k<hidenode; k++)
            {
                h_delta[j] += I_future_delta[k] * U_I[j][k];
                h_delta[j] += F_future_delta[k] * U_F[j][k];
                h_delta[j] += O_future_delta[k] * U_O[j][k];
                h_delta[j] += G_future_delta[k] * U_G[j][k];
            }

            O_delta[j] = 0.0;
            I_delta[j] = 0.0;
            F_delta[j] = 0.0;
            G_delta[j] = 0.0;
            state_delta[j] = 0.0;

            //隐含层的校正误差
            O_delta[j] = h_delta[j] * tanh(state[j]) * dsigmoid(out_gate[j]);
            state_delta[j] = h_delta[j] * out_gate[j] * dtanh(state[j]) +
```

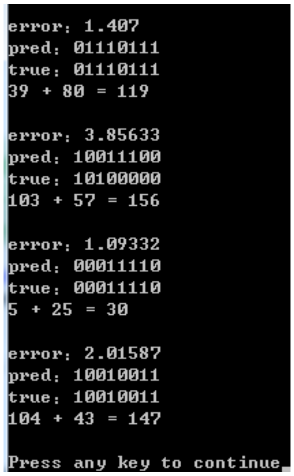
关闭

```
                    state_future_delta[j] * forget_gate_future[j];
            F_delta[j] = state_delta[j] * state_pre[j] * dsigmoid(forget_gate[j]);
            I_delta[j] = state_delta[j] * g_gate[j] * dsigmoid(in_gate[j]);
            G_delta[j] = state_delta[j] * in_gate[j] * dsigmoid(g_gate[j]);

            //更新前一个隐含层和现在隐含层之间的权值
            for(k=0; k<hidenode; k++)
            {
                U_I[k][j] += alpha * I_delta[j] * h_pre[k];
                U_F[k][j] += alpha * F_delta[j] * h_pre[k];
                U_O[k][j] += alpha * O_delta[j] * h_pre[k];
                U_G[k][j] += alpha * G_delta[j] * h_pre[k];
            }

            //更新输入层和隐含层之间的连接权
            for(k=0; k<innode; k++)
            {
                W_I[k][j] += alpha * I_delta[j] * x[k];
                W_F[k][j] += alpha * F_delta[j] * x[k];
                W_O[k][j] += alpha * O_delta[j] * x[k];
                W_G[k][j] += alpha * G_delta[j] * x[k];
            }

        }

        if(p == binary_dim-1)
        {
            delete  O_future_delta;
            delete  F_future_delta;
            delete  I_future_delta;
            delete  G_future_delta;
            delete  state_future_delta;
            delete  forget_gate_future;
        }

        O_future_delta = O_delta;
        F_future_delta = F_delta;
        I_future_delta = I_delta;
        G_future_delta = G_delta;
        state_future_delta = state_delta;
        forget_gate_future = forget_gate;
    }
    delete  O_future_delta;
    delete  F_future_delta;
    delete  I_future_delta;
    delete  G_future_delta;
    delete  state_future_delta;

    if(epoch % 1000 == 0)
    {
        cout << "error : " << e << endl;
        cout << "pred : " ;
        for(k=binary_dim-1; k>=0; k--)
            cout << predict[k];
```

关闭

```
            cout << endl;

            cout << "true : " ;
            for(k=binary_dim-1; k>=0; k--)
                cout << c[k];
            cout << endl;

            int out = 0;
            for(k=binary_dim-1; k>=0; k--)
                out += predict[k] * pow(2, k);
            cout << a_int << " + " << b_int << " = " << out << endl << endl;
        }

        for(i=0; i<I_vector.size(); i++)
            delete I_vector[i];
        for(i=0; i<F_vector.size(); i++)
            delete F_vector[i];
        for(i=0; i<O_vector.size(); i++)
            delete O_vector[i];
        for(i=0; i<G_vector.size(); i++)
            delete G_vector[i];
        for(i=0; i<S_vector.size(); i++)
            delete S_vector[i];
        for(i=0; i<h_vector.size(); i++)
            delete h_vector[i];

        I_vector.clear();
        F_vector.clear();
        O_vector.clear();
        G_vector.clear();
        S_vector.clear();
        h_vector.clear();
        y_delta.clear();
    }
}


int main()
{
    srand(time(NULL));
    RNN rnn;
    rnn.train();
    return 0;
}
```

关闭



```
error: 1.407
pred: 01110111
true: 01110111
39 + 80 = 119

error: 3.85633
pred: 10011100
true: 10100000
103 + 57 = 156

error: 1.09332
pred: 00011110
true: 00011110
5 + 25 = 30

error: 2.01587
pred: 10010011
true: 10010011
104 + 43 = 147

Press any key to continue
```

参考：

http://lib.csdn.net/article/deeplearning/45380

http://www.open-open.com/lib/view/open1440843534638.html

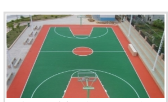
顶 3 踩 0

- 上一篇 leetcode---Reorder List
- 下一篇 C#---distinct

相关文章推荐

- 长短记忆型递归神经网络LSTM
- MVVM在美团点评酒旅移动端的最佳实践--王禹华
- 人人都能用Python写出LSTM-RNN的代码！[你的...
- C语言大型软件设计的面向对象--宋宝华
- 简单理解LSTM神经网络
- Retrofit 从入门封装到源码解析
- 深入理解LSTM神经网络
- 跳过Java开发的各种坑
- RNN以及LSTM的介绍和公式梳理
- Spring Boot 2小时入门基础教程
- 理解长短期记忆(LSTM) 神经网络
- Shell脚本编程
- 基于LSTM的神经网络语言模型的实现
- RNN递归神经网络的详细推导及C++实现
- 深度学习BP算法的推导（附加RNN LSTM的推导说
- 循环神经网络（RNN）

关闭


文件管理系统

塑胶篮球场造价

拼接屏


建站

erp软件排名

查看评论

qq_39951243 10楼 2017-08-23 17:17发表

楼主大神我想问一下我用这套程序为什么结果和楼主的不一样？？而且运行完窗口自己就关了

iccff 9楼 2017-08-01 17:45发表

作者你好，看了你的lstm实现加法器，很受益，在此有一个问题，需要请教作者，我在实现乘法器的时候，可以用训练好的加法器去直接实现乘法器，我想和作者交流，有没有什么办法直接训练乘法器，谢谢作者，我的qq：835097453，希望能和作者交流。

汪洋之舟---seaboat 8楼 2017-07-31

你好，误差是不是应该是整个样本的误差，这里用的是某个时刻的误差。

xiaohe_nh 7楼 2017-06-25 14:26发表

公式8是错了吧，应该是tanh而不是sigmoid

zhouhongyue1976 Re: 2017-11-06 2

```
double dtanh2(double y)
{
return 1.0 - y * y;
}
//190 //g_gate[j] = sigmoid(gGate);
g_gate[j] = tanh(gGate);

//299 //G_delta[j] = state_delta[j] * in_gate[j] * dsigmoid(g
_gate[j]);
G_delta[j] = state_delta[j] * in_gate[j] * dtanh2(g_gate[j]);
```

lamjevence 6楼 2017-05-11 20:44发表

67行注释真的是遗忘门吗？看你的代码更像输出门

CDownLoad_ZXL 5楼 2016-10-30 16:20发表

作者你好，我觉得你的代码中有两个小问题，
第一个是在前向传播的最后一行，193行 h[j] = in_gate[j] * tanh(state[j]);，
我觉得因该是h[j] = out_gate[j] * tanh(state[j]);

第二个是在反向传播过程中，第294~295行，其中的 dtanh(state[j])是错误的，由于你的dtanh的定义是需要传入一个tanh(state[j]),而不能直接传入state[j]。

望交流！

关闭

步悠然 Re: 2016-11-03 11:49发表

回复CDownLoad_ZXL：是的，我当初想当然的觉得处理起来跟dsigmoid一样，现在发现，是不一样的，因为disgmoid算的时候，里面的参数本身就已经是sigmoid值了，而dtanh不是，谢谢你的指正。
修改如下：

```
//tanh的导数，y为tanh值
double dtanh(double y)
{
```

```
y = tanh(y);
return 1.0 - y * y;
}
```

桃之夭妖 4楼 2016-10-22 15:11发表

你好厉害啊，刚看了你写的独立任务最优调度问题，怎么做到的啊，向你学习

DreamLife-Root 3楼 2016-10-17 11:29发表

完全看不懂+1111，先膜拜在收藏

拟态 2楼 2016-10-17

good

baidu_36228476 1楼 2016-10-15 2

完全看不懂。。。。大神啊！先膜拜再说！！！

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved

关闭