问题求解与实践工程报告

廖坚钧 **518021910467**

数据输入

所有数据都存在DataStruct.h中。

- **struct fitter**

> 用于存放各个参数的均值和方差，用于后面数据的标准化。

```
// zero-mean normalization
struct fitter {
    double *mu, mu_res;  //mu[NUM_E_1], mu_res;   // 样本均值
    double *sigma, sigma_res;  //sigma[NUM_E_1], sigma_res;   // 样本方差
    ~fitter();
};
```

- **struct interval**

> 用于存放输入数据中各个参数的最小值和最大值，以及参数之间的协方差。

```
struct interval {
    double *tmin, *tmax; //tmin[PRE_NUM], tmax[PRE_NUM];
    double *pov; //pov[PRE_NUM_RES*(PRE_NUM_RES - 1) / 2];
    /*
        0<=i<=n-1
        (i, j)  ->   [i(n-1) - (i-1)i/2 + j - i - 1]
        (0,1), (0,2), ..., (0, n-1)
        (1,2), (1,3), ..., (1, n-1)
    */
    ~interval();
};
```

- **struct data**

> 通过从一行文本提取数据并存放参数和结果

```
struct data {
    double *e_1;//e_1[NUM_E_1];
    double res;  // diameter
    data(const std::string & line, size_t res_index);
    data(const std::string & line, size_t &cnt, fitter & ft, size_t res_index);
    data(data && rhs);
    ~data();
};
```

- **struct trainres**

存放每个神经层的权重和每次迭代后权重的梯度

```
struct trainres {
    double **W_1; //W_1[NUM_E_2][NUM_E_1];
    double **W_2; //W_2[NUM_E_3][NUM_E_2];
    double *W_3; //W_3[NUM_E_3];
    double *b_1, *b_2, b_3; //b_1[NUM_E_2], b_2[NUM_E_3], b_3;
    double **dW_1; //dW_1[NUM_E_2][NUM_E_1];
    double **dW_2; //dW_2[NUM_E_3][NUM_E_2];
    double *dW_3; //dW_3[NUM_E_3];
    double *db_1, *db_2, db_3; //db_1[NUM_E_2], db_2[NUM_E_3], db_3;
    trainres();
    ~trainres();
};
```

- **struct Axisdata**

存放绘制数据用的坐标点数据

```
struct Axisdata {
    const char * title;
    const char * X_title, *Y_title;
    double x_min, x_max;
    double y_min, y_max;
    int X_div_num;
    int Y_div_num;
    double *Y;
    Axisdata(const char * title, const char * Xtitle, const char * Ytitle,
        double x_min, double x_max, double y_min, double y_max, int xdn, int ydn, double *y)
        : title(title), X_title(Xtitle), Y_title(Ytitle), x_min(x_min), x_max(x_max),
        y_min(y_min), y_max(y_max), X_div_num(xdn), Y_div_num(ydn), Y(y) {}

};
```

- 使用

DataTrainer类使用std::vector<data> 存放数据组。 std::map<std::string, size_t> name_list用于标记每个参数名在double *data::e_1对应的索引。

```
class FileReader {
public:
    explicit FileReader(const std::string & str, const std::string & res_str,
    void read(std::vector<data> &, std::vector<data> & datas2, fitter & ft, si
    ~FileReader();
private:
    int hashcode(const std::string &str);
    void del(size_t res_index, size_t iter_cur_num);
    size_t num_of_partitions;
    std::string FILE_RD;
    std::set<std::string> FILE_NAMES;
    std::map<std::string, size_t> name_list; // 名字与索引的map,用于下面数据结构
    size_t *isnan_cnt;
    std::string res_str;
};
```

```
class DataTrainer {
public:
    friend void read(FileReader *, DataTrainer *);
    void traindatas(int epoch, int batch_size, double alpha, double lambda);
    void show();
private:
    void transform(std::vector<data> & vec, interval & itv);
    double getR_square(std::vector<data> & vec, double mean_Y);
    std::vector<data> train_datas;
    std::vector<data> test_datas;
    fitter ft;
    interval train_itv, test_itv;
    trainres tr;
    size_t res_index;
    std::map<std::string, size_t> name_list;

    TR::Axis::AxisWindow * AW;
};
```

```
class  AxisWindow : public Fl_Window {
public:
    AxisWindow(int x, int y, int w, int h, const std::string & ti
    void init();
    virtual ~AxisWindow() { delete[] density; delete[] stitle; }
    void nextG() {
        graph_cur = (graph_cur + 1) % 3;
        if (graph_cur == 1) nextPt->set_visible();
        else nextPt->clear_visible();
        if (graph_cur == 0) this->resize(xxx, yyy, width, height
        else this->resize(xxx, yyy, width, height);
    }
    void nextP() { parameter_cur = (parameter_cur + 1) % pre_num_
protected:
    void draw();
private:
    void drawAxis(int x, int y, int w, int h, const TR::Axisdata
    std::vector<TR::Axisdata> Axsds;
    const std::vector<TR::data> & ref_datas;
    const TR::interval & ref_itv;
    const std::map<std::string, size_t> & name_list;
    size_t parameter_cur;
    size_t res_index;
    int xxx, yyy;
    int width, height;
    double **density; //density[PRE_NUM][AXIS_DIV];
    Fl_Button *nextBt;
    Fl_Button *nextPt;
    int graph_cur;
    std::string *parameter_name;
    std::string *stitle; //stitle[PRE_NUM];
};
```

脏数据发现和处理

- 脏数据定义

  - 数据中带有非('0'~'9','.')的字符都不是可计算的值。将值大多都是不可计算值的参数称为无效参数，例如name参数。
  - 任何值为0或空的数据都需要剔除。

- 脏数据处理

  - 处理代码在FileReader类的构造函数中。
  - 避免数据过大，进行批量处理。处理结果分成多个partition（这些partition是DataTrainer类读取的对象，即剔除脏数据后的可用文件。在DataTrainer读取partition前，FileReader会发给DataTrainer一份包含所有partition文件名的set。）。
  - 读入原始数据时，把结果值diameter符合脏数据定义的那条数据剔除，即不写到临时文件中。这样每条数据的结果值diameter都是数字。
  - 置零一个用于计数的数组cnt，遍历读取的数据，每条数据检查每个参数，若参数i符合上述脏数据定义，则执行命令++cnt[i]。
  - 任何cnt[i]超过diameter的cnt的参数i，即为无效参数，将该参数剔除（此剔除是假剔除，即在写到partition的时候无视该参数即可）。
  - 剔除无效参数后，剩下的每条数据若存在一个有效参数值为无效值，则忽略该条数据，否则，写到partition。这些partition都是可用数据。

```
GM 0.999558
H 0
IR 1
UB 0.95746
a 0
ad 0
albedo 0.000795967
class 1
condition_code 0.99969
data_arc 0
diameter 4.42204e-05
e 0
extent 1
i 0
ma 0
moid 0
n 0
n_obs_used 0
name 1
neo 1
om 0
per 0
per_y 0
pha 1
q 0
rot_per 0.690369
spec_B 1
spec_T 1
w 0
```

  - 结果如下：

```
读取表头，存下表名
读取原始数据（批量读取），每批数据{
    一行行读取数据，若diameter为空，则跳过
```

```
    否则写到临时文件中，写入前统计每个参数的cnt
}

读入临时文件，先读表头
一行行读取{
    若该行某个参数是坏的，则跳过。
    遍历每个参数，若该参数的cnt小于或等于diameter的cnt，就写到partition里。
}
```

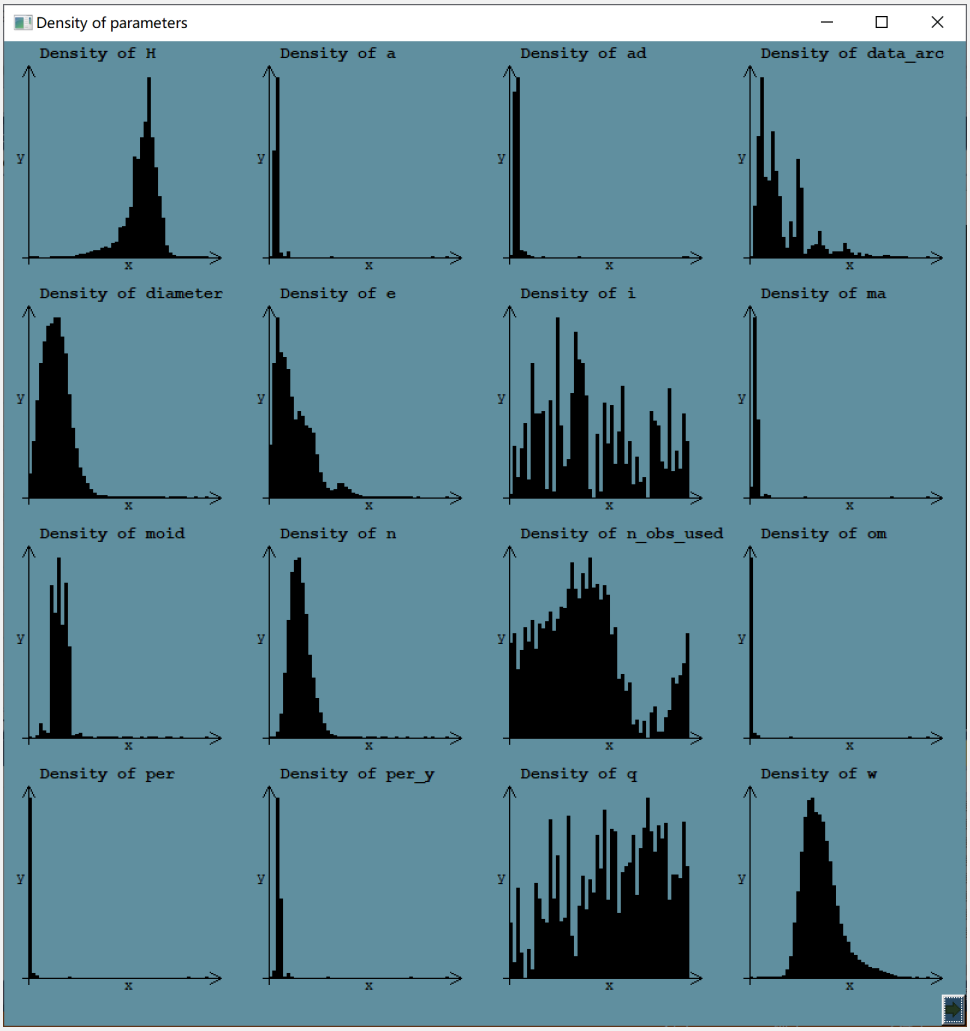
统计与可视化

- 统计方面

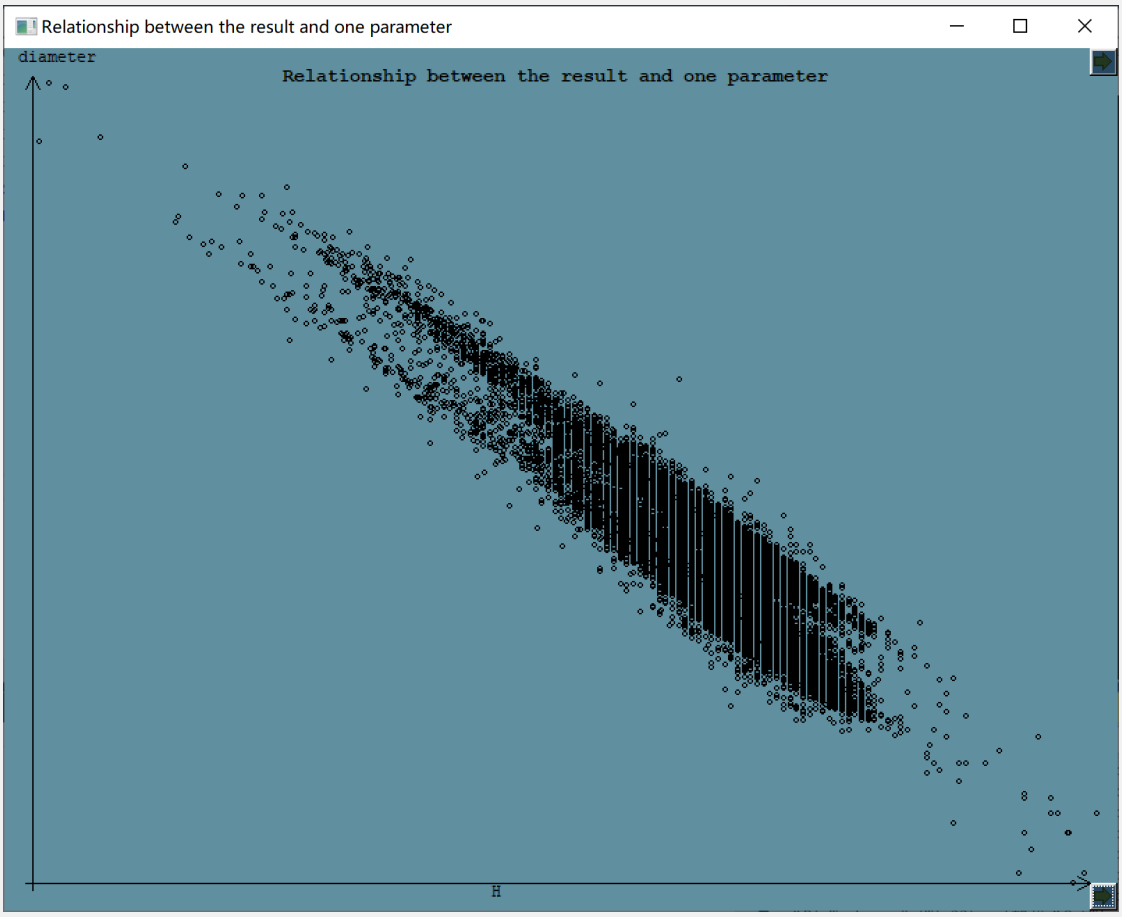
  - 各个参数的密度分布
  - 每个参数和结果值diameter的散点图
  - 参数之间的协方差相关性

- **FLTK**界面可视化设计

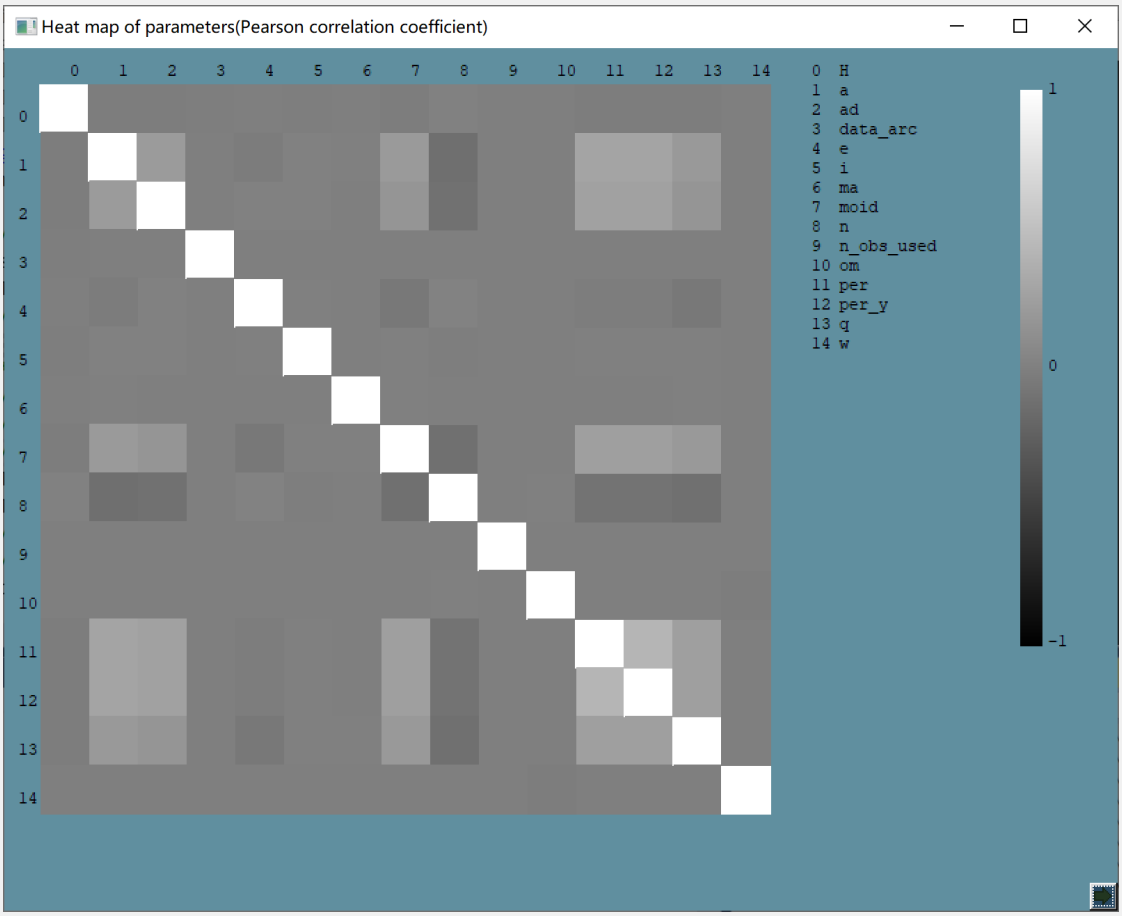
  - 声明：屏幕分辨率需要超过800*820
  - 背景色是我精心挑选的美丽的浅蓝色，可以缓解视觉疲劳，再配上黑色的文字和图案，让数据可视化变得清晰。
  - 第一个界面是各个参数的密度分布，右下角有个按钮，是用来切换到下一张图的（这个按钮会一直保留且作用不变）。



- 第二个界面是每个参数和结果只`diameter`的散点图，右上角有个按钮可以切换不同的参数。



- 第三个界面是个参数的热力图，热力值代表每个参数之间的相关性，使用黑白颜色是为了清晰的看出参数之间的相关性。

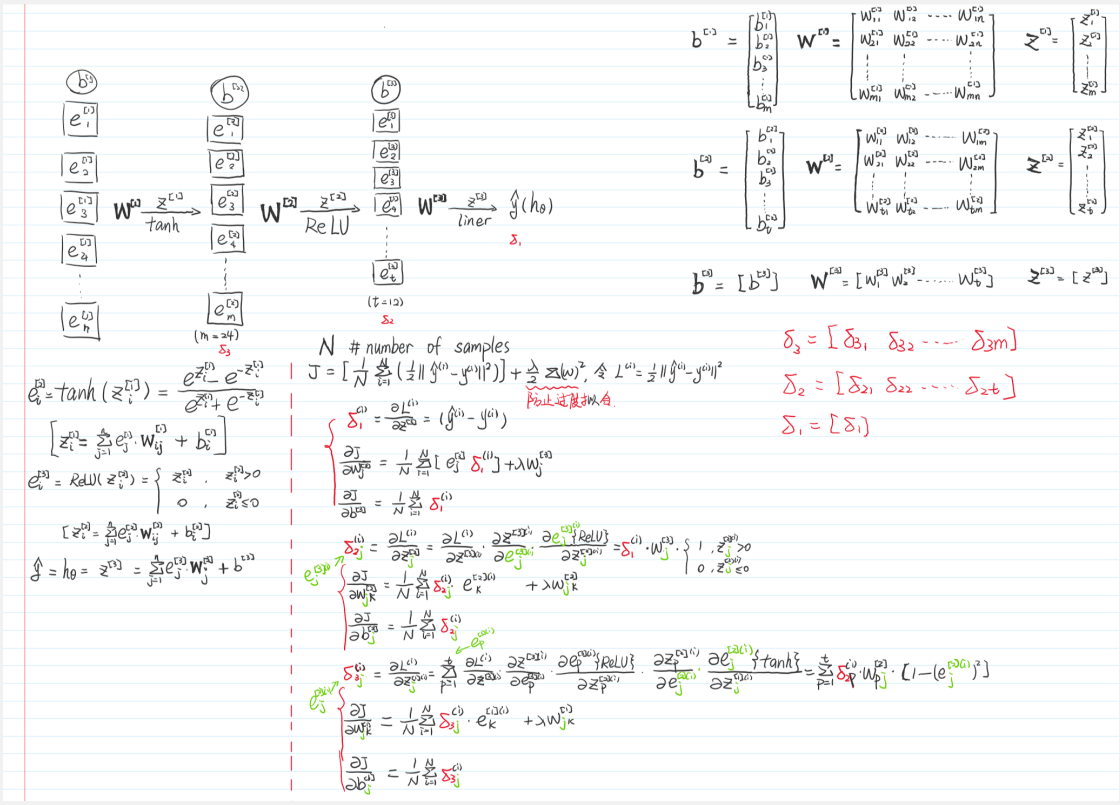


- (请原谅我的审美)

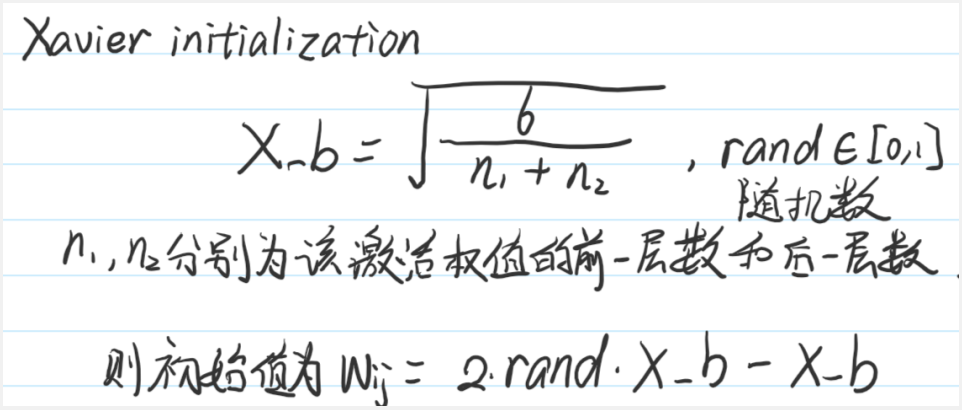
趋势预测

- 预测方法

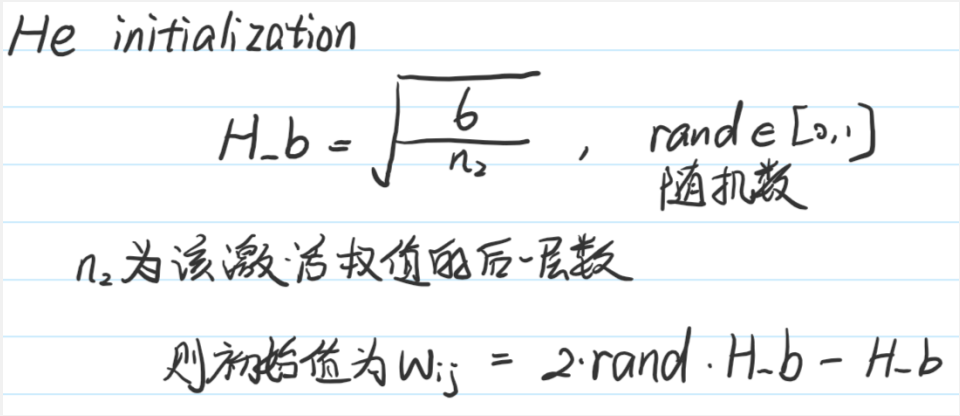
  - 我选择的数据是根据小行星的一些参数来估计小行星的直径`diameter`。
  - 训练过程由`DataTrainer`类执行，在其`traindatas`成员函数中（代码过长）。
  - 因此通过拟合来达到预测效果，在这里，我采用神经网络的方法来拟合。
  - 下图是这个工程中神经网络的推导公式。



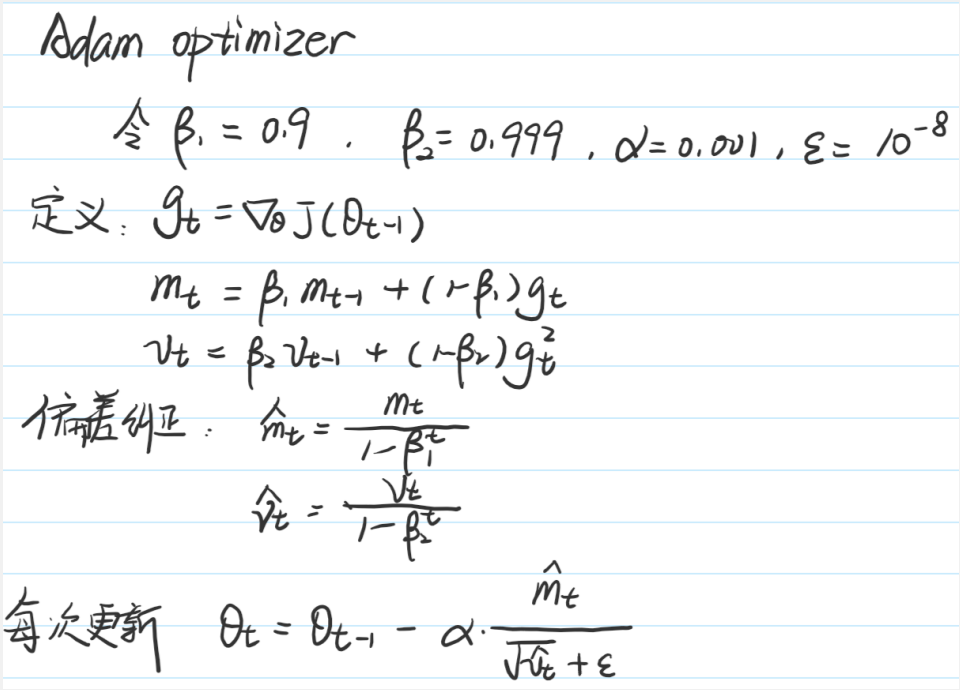
  - 第一层使用tanh激活函数，第二层使用ReLU激活函数，第三层线性拟合
  - 采用方差来计算误差，进而反向传播，改变权重。
  - 为了优化该神经网络，分别从以下几个方面改进:
  - ■ 初始化
  - ■ ■ 使用`Xavier initialization`对`tanh`激活函数权重初始化



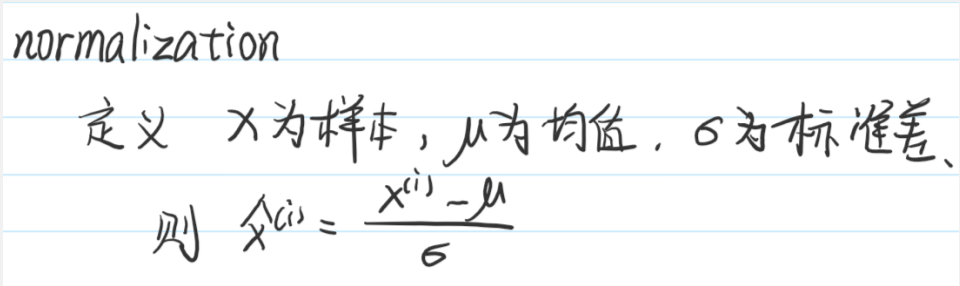
- 使用He initialization对ReLU激活函数权重初始化



- 使用随机值对线性拟合函数权重初始化
- 迭代优化器
- 使用Adam optimizer对迭代进行优化，让其保留过去迭代的记忆，缓冲某次迭代误差过大的影响。



- 标准化
- 对参数进行标准化，使其服从正态分布，这样会提高训练的效率。

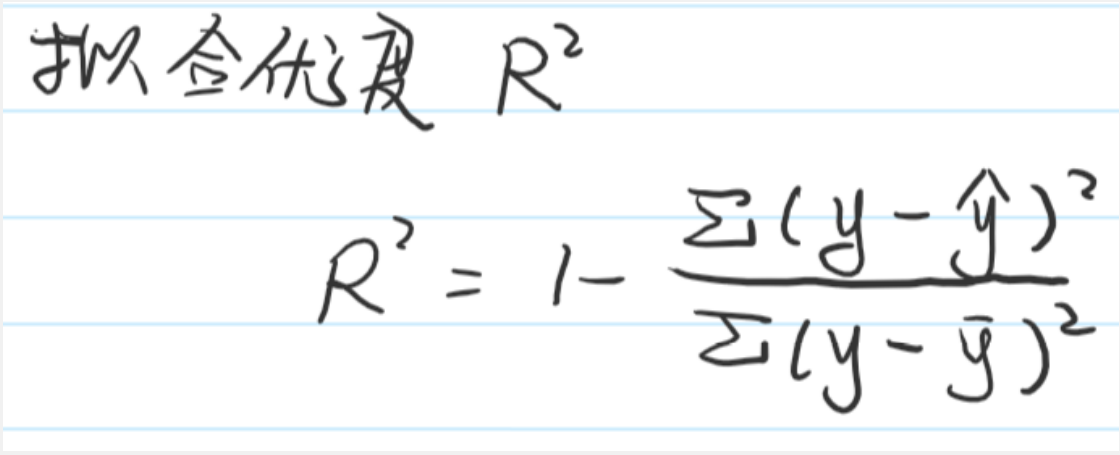


- 正则化
- 在代价函数中增加惩罚因子lambda，防止过拟合。

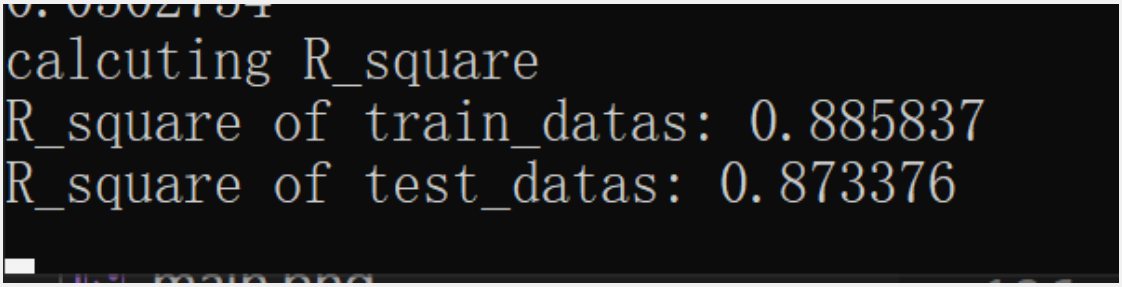
- 评价标准

- 为了评价数据拟合是否欠拟合或过拟合，将数据随机切割为80%的训练集和20%的测试集。训练集训练后的模型应用于测试集，并评价其误差。

- 由于最后一层为线性拟合，故采用拟合优度R^2来评价拟合结果。



- 实现函数是DataTrainer的R_square函数。



运行流程

```
int main() {

    // 用于预处理数据
    cout << "输入文件路径: ";
    string filename;// = "Asteroid_Updated.csv";
    getline(cin, filename);
    cout << "正在预处理数据。。。";
    TR::FileReader *fr = new TR::FileReader(filename, "diameter", 30000, 1);
    /*
        for test
    */
    //std::string filename = "test.csv";
    //std::string filename = "Asteroid_train.csv";
    //TR::FileReader *fr = new TR::FileReader(filename);
    TR::DataTrainer *dt = new TR::DataTrainer();
    TR::read(fr, dt);
    // datas : use 38 MB memories
    // epoch = 100, batch_size = 256, alpha = 0.005, lambda = 0
    dt->traindatas(100, 256, 0.005, 0);

    dt->show();
```

项目是以一般性为目的写的，所以可以很轻松地套在其他预测数据集上，只需要给定文件名和目标参数名。

- 构造一个**FileReader**实例

> - FileReader读入原始数据，并进行分partition处理，每次处理统计脏数据，然后将检验通过的数据写到相应partition。

- 构造一个**DataTrainer**实例并读取

> - 构造一个DataTrainer实例，使用上述初始化方法对各个层的训练权重初始化。
> - TR::read函数负责将FileReader从partitions读取的数据存放到DataTrainer，并统计参数均值，方差，区间和协方差等，然后对参数进行正态标准化。

- 训练模型

> - 初始化Adam optimizer变量m和v。
> - 开始随机梯度训练。
> - 训练结束后计算训练集和测试集的方差

- 显示

> - 构造数据统计视图窗口