## 模型的持久化，

1. 使用tensorflow中saver=tf.train.Saver()函数创建保存器，使用saver.save()函数可以保存图模型和变量取值，方便在断电或者内存不够时恢复模型。避免因为意外原因没有得到任何结果（任何一次训练都是宝贵的）。

## 使用合适的评价函数来评价模型，

1. 对于一个模型来说，首先是训练数据和测试数据其次就是评价指标，如果没有明确的评价指标，就像一个无头苍蝇，不知道下一步要调节的是什么，模型在训练集和测试集上出了什么错误。

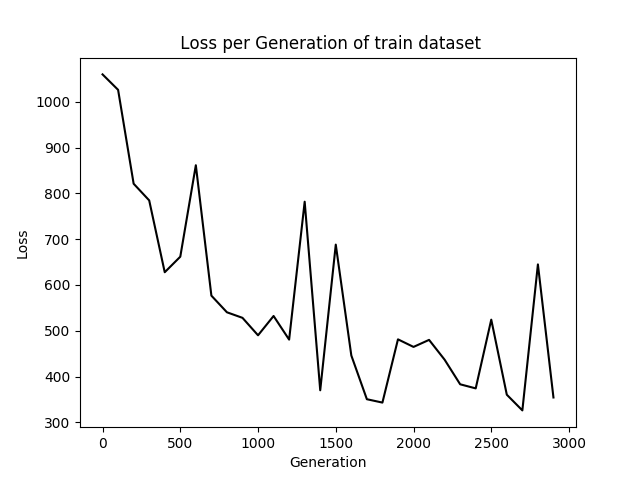
2.1回归模型可以使用：R2系数决定系数：总体平方和SStot,残差平方和SSerr R2=1-(SSerrr/SStot)表示模型对于数据的拟合度，此值越大越好，最好的情况是此值为1.

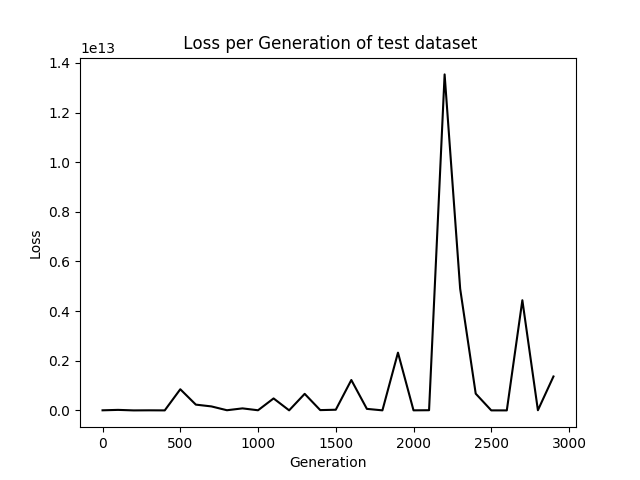
2.2 相对分析误差RPD=(stdev/remse)此值只在测试集上测定，stdev表示测定值标准差，rmse表示测试集上均方根误差。RPD值越大越好。

2.3对于多元素的预测问题，使用多个神经元表示输出，要在不同的元素上分别在训练集和测试集上使用评价指标（因为数据的分布和数据的范围是不同的，但是对于多任务学习其loss值是多种任务loss值的加和，我们不能知道导致loss值大的原因是来自于模型对哪一种预测元素的偏差，只有在多种任务上分别使用测试指标，才能判出模型对于不同任务的准确程度。）

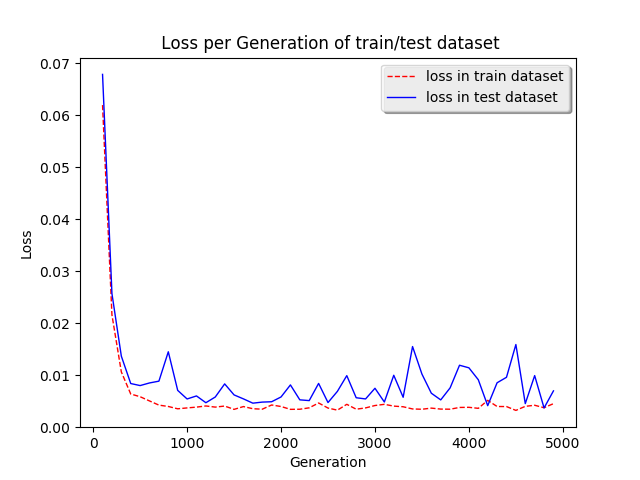
## 数据的预处理，

数据的不同维度一般来说很可能来自不同的数量级，比如目标屈服强度数据范围为200-600，伸长率数据范围为20-60，而硼元素的数据范围一般在0.0005-0.0025之间，如果使用原始数据进行预测则会出现模型不拟合的后果，此时就需要上使用数据归一化处理（max-min归一化，0归一化等等）将数据各维度缩放到一个数据范围内，通常为[-1,1]和[0,1].

****

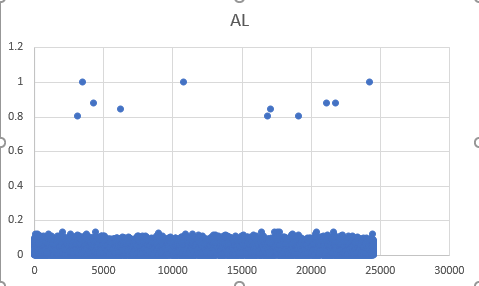
****

以上为不使用归一化的后果。（注意数量级）

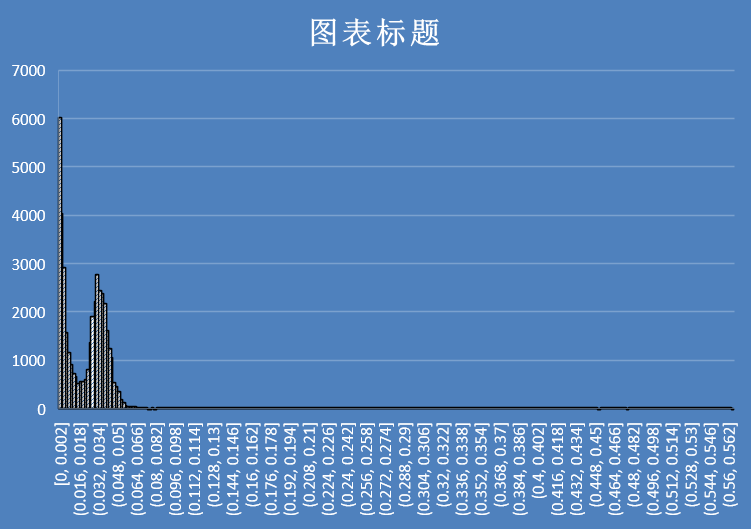


上图是使用数据归一化后的结果

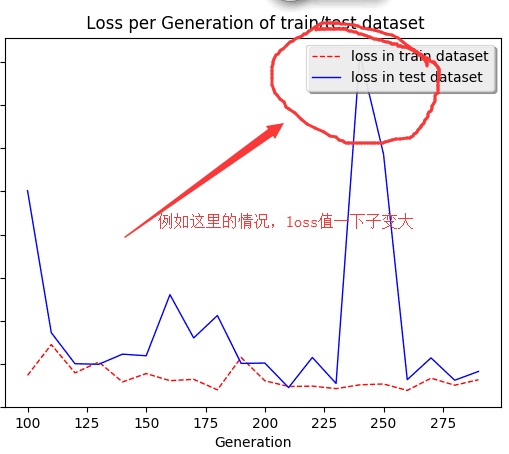
1. 数据一定要保证干净，数据在处理之前要清洗，去除错误的和整体分布明显不一致的数据。（可以使用excel查看csv文件中数据的总体分布）



在归一化的数据后我们发现AL的值一般是分布在0-0.066之间，25000个数据中出现3个值为1的值肯定是数据采集时本身有错误。



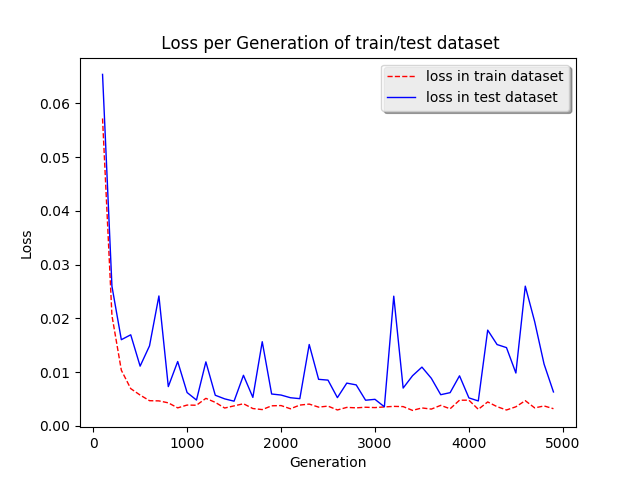
如果这种数据出现在测试集或者验证集中而训练集中没有这种数据很可能出现

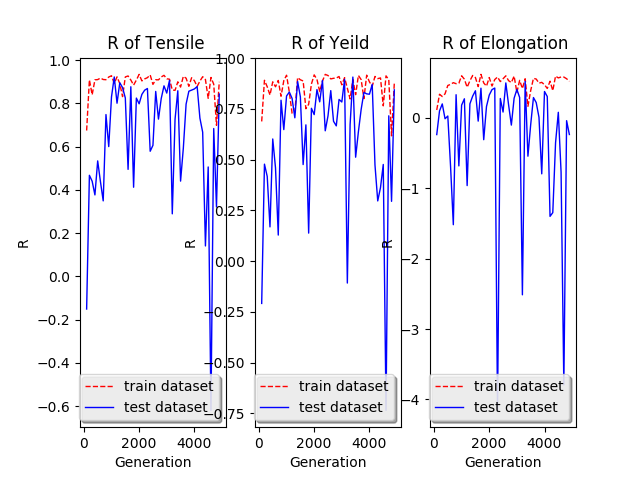


Loss值突然一下爆炸的情况

## 使用图表展示训练指标函数

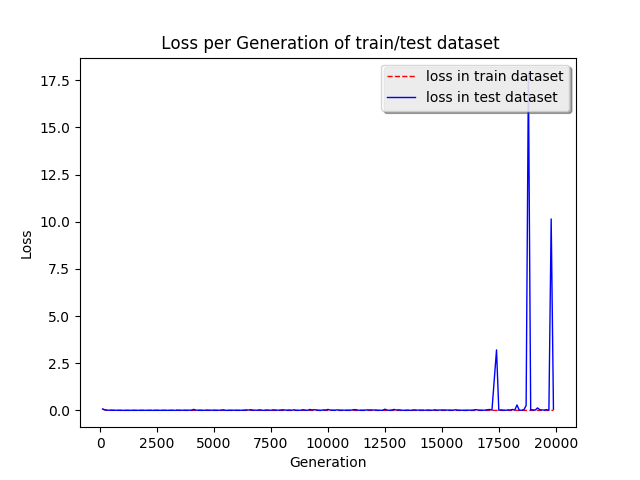
单纯的使用每100或1000次迭代后输出难以看清评价函数的本质特征（毕竟人肉眼观察是十分有限的）。我们应该使用图形的方式展示训练指标函数的变化。



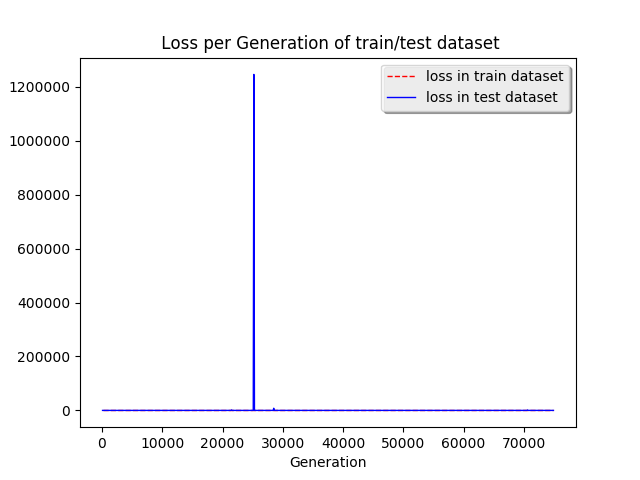


1：可以通过判断训练集和测试集上的误差来判断是否过拟合，或者欠拟合。如果训练集上已经表现不好，则欠拟合，如果训练集表现不错但是测试集上表现太差就过拟合（一般使用L2正则化，Dropout来避免，BN算法也有一定的正则化作用）

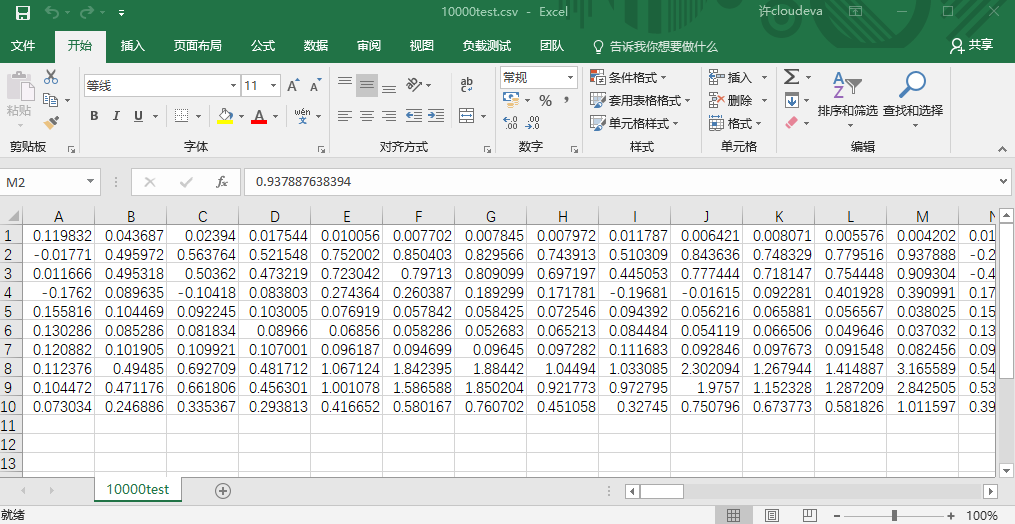
2.：方便提前终止训练，因为一般不太复杂的问题，不用迭代很多次，一个不太复杂的模型迭代训练长时间会发生过拟合，因为它会越来越向训练集趋近（即时此时我们加入正则化但是适当的正则化参数是合理的，通常为0.00001-0.0001，如果继续加大一个不太复杂的模型的正则化参数很容易导致模型欠拟合。防止过拟合适可而止，不要因为测试集表现有差距就无限制的加大正则化力度，很容易导致欠拟合）



1. 画图不要等到迭代完成后在调用画图函数，应该在程序运**行时每隔一定时间**调用画图函数
2. 避免程序到后期出现bug或者断电等意外而失去了实验结果（每一次训练都是宝贵的）
3. 避免程序到后期输出爆炸而失去原来结果的对比度。（因为数据的数量级相差太大了，原来微小的差异被覆盖掉了）



## 训练得到的结果要实时保存（最好5000次迭代保存一次）



例如：上面使我们某次迭代后保存的数据，每一行的值都是一个测试指标而列值则是迭代次数，通过比较分析，我们能够很方便的确认模型的迭代情况。（例如我们经过观察，发现模型在第M列中的2,3行的值是最大的，这表示模型在1300次迭代时有了最好的结果）

## 紧跟开源框架的步伐

Tensorflow是一个开源的框架，其所有代码在github上都有开源，其新版本都有很棒的功能，例如在1.0版本以前，其并没有使用tf.layers.batch\_normalization的函数，使用batch\_normalization需要使用低阶函数tf.nn.batch\_normalization自行定义高级功能。随着其版本的迭代，其功能越来越完善，使用方法越来越便捷，这就需要我们不断的学习与跟进其API。

## 一定要使用Batch normalization

Batch normalization 对于神经网络相当于一个万金油的存在，其能够对每层神经层的输出进行批处理标准化，大大提高训练的速度，精准度，也能为模型带来少量的正则化。注意BN的正确使用方法，BN函数只在训练集时使用，但是在测试集上不使用。

## 调节参数

### 学习率对学习的影响



可以从图中看到低的learn\_rate曲线会趋近于线性（蓝线），其收敛很慢在前1000次批次几乎不收敛

而过高的learn\_rate曲线会趋近于指数（绿线）。在  
高的学习率会使得曲线迅速下降，但是可能会在一个不好的结果下收敛。过高的学习率甚至会使得结果往差的方向发展（黄线）。比较合理的学习率应该体现出入红色线所示的形状。