SRV理论

构象转移理论

考虑由构象组成的态空间，我们有稳态分布，时刻*t*系统在态空间上的概率分布为。假设时刻后系统的概率分布

我们假设当比较大时系统是马尔可夫的，那么转移矩阵便是与时间无关的

对于稳态分布，对转移矩阵做出细致平衡约束(自然导出还是额外约束?)

我们可以知道稳态分布是转移矩阵本征值为1的本征矢

本征值的求解

我们关心的是转移矩阵的本征值和本征矢，尤其是本征值最大的几个本征矢(除了1)。如何求解高维空间中特定的本征值和本征矢？我们假设系统或者说转移矩阵有本征值为的本征矢

我们在与正交的子空间中任取向量，我们可以把用表示

使用转移矩阵作用上

于是我们可以做内积

所以我们要寻找的实际上就是转移矩阵作用后内积最大的子空间的向量.

寻找子空间中内积最大的向量

那么现在的问题是如何在高维空间中寻找内积尽可能大使得(8)中等号成立的向量？

一个向量靠近待定向量

噪声项，改造内积算符

现在假设我们有内积为的向量，我们如何得到向量使得其内积.寻找这样的操作实际上是把向量的频谱往低频移动.有一种方法是让高频逐渐蒸发掉，这样低频占比便会增加.如何向中没有的低频成分移动?进行随机杂交有可能会引入低频成分.所以总的思路就是引入更低频成分，保留低频成分，蒸发掉高频成分.便在这三个思路上做文章.

蒸发掉高频成分

使用类似于虚时演化的方法让高频成分自然衰减掉.使用转移矩阵连续作用上

所以使用转移矩阵连续作用上就可以蒸发掉高频成分，但要注意归一化保留的长度.使用转移矩阵的不同变换形式有不同的蒸发效果，哪种形式的的蒸发效果最好?

引入更低频成分

通过随机出噪声项与杂交有可能引入更低频成分.如何把这个思路数学化?

是否与正交最好，以及系数与如何确定?我们假设有频谱分解

其中是包含的成分，是与正交的成分.如果噪声项是完全随机的，那么包含各个频率成分以及各个频率成分的幅度的概率都是完全一致的.但是我们希望尽可能多地包含更低频成分，少包含已有频率成分，最好不包含高频成分.正交化可以剔除掉已有频率成分，所以也应该对做蒸发操作.或者有什么产生噪声的办法使得产生的噪声与频率挂钩，产生低频成分多高频成分少的噪声而不是产生白噪声?

产生低频噪声

如何产生低频噪声?

多个向量靠近待定向量

多个固定向量中的最低频成分

假设我们有个向量，对这个向量做频谱分解

如何找到的最低频成分?蒸发操作是一个方法，下面我们暂时把注意力集中到广义本征值问题.

我们先计算几个关键量

做内积

我们把这两个内积矩阵元扩展写成矩阵的形式

这里每一个求和的矩阵只与在第个本征矢上的投影有关，我们把这个矩阵记成，那么这两个矩阵(16)(17)便可以分别写成和.我们现在指出这两个矩阵满足这样的关系(即广义本征值问题)

要看出这一点，只需指出(如何具体得到?).所以这里的思路便是由这个向量计算出矩阵(16)(17)，然后求解广义本征值问题(18)便可以得到低频成分.

蒸发操作得到最低频成分

这是可以思考的重点.

非线性降维结合广义本征值问题

神经网络的作用

神经网络的输入是时刻和时刻系统的状态，也就是和，但是并没有直接使用这两者做内积而是把它们通过神经网络进行了变换，为什么要进行这样的变换?进行非线性降维.所以SRV的思路便是首先通过非线性降维把高维空间的坐标映射到低维空间，然后通过求解广义本征值问题解出本征值，并通过梯度下降法更新非线性降维的方向使得这样的降维尽可能映射到本征值最大的低频空间.

非线性降维

我们对时刻系统的状态进行非线性降维得到的具有怎样的性质?是否可能含有中没有的本征矢方向?我们首先看

线性降维

如果是一种线性变换也就是线性降维的话，那么得到的是的线性组合(?)也是的线性组合，也就是说线性降维是无法得到新的频率成分的(需要进一步考虑!).

线性降维为什么可以看做是线性组合?

假设的维度是，线性降维后的维度是.那么多若干个进行降维操作是彼此独立的，只是使用的降维矩阵相同都是维度的.相当与对每一个的维进行同样的次组合得到低维矢量.

单层神经网络

而如果是非线性变换，我们首先只考虑一层神经网络的情况，也即假设

想法总结

1. 通过在神经网络中加入蒸发操作增大低频分量的比例.
2. 引入随机项看看能否引入低频分量.实际上都是想帮助神经网络更准确地降维到最低频子空间.

RNN

代码与理论的学习

deep.f90

数据准备，rnn\_initialize()，rnn\_training()，rnn\_output().

rnn.f90

神经网络的结构

(4,10,2,10,4).输入输出均为4个结点，有4个隐藏层，最中间是瓶口只有2个结点可以看作是降维.最后一层还加了一个额外的线性层，其权重矩阵大小是4\*4.

initialize

最后一层线性层的初始化(权重/偏置及相应的导数，输出)为0，RNN层的初始化(偏置，前一层到本层的权重，前一个时刻到本时刻的权重，偏置)进行了正交化操作.

前向传播

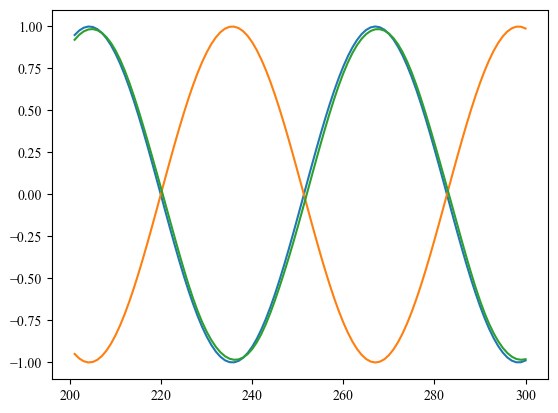
RNN层

反向传播

19页的几个公式.线性层.

实际例子的测试

简谐振动



修改激活函数

如果我们修改激活函数为Relu(x).

RNN自身有一些问题.尤其是长程依赖性，会产生梯度消失的问题.

在现有RNN代码的基础上实现SRV

思路

神经网络模块的基本架构是有的，缺少的是对高维空间进行本征值求解、利用本征值构造SRV的损失函数、对含有本征值求解过程进行导数计算下降梯度、更加效率和专业的训练方法.

关键的是我们首先要选择一个确定的模型/问题来保证我们的SRV是正确的.在这里我们先选择1D 4-well potential模型.然后我们可以多研究几个其它的确定性模型来进一步验证我们的SRV代码.

SRV的实现

主要有以下几个方面:

1. 在现有RNN的基础上实现孪生子神经网络.
2. 广义本征值问题求解模块.
3. 计算损失函数与改变梯度计算方式.
4. 对网络结构和训练方式的改进和优化.

利用SRV的非线性降维进行增强采样

本征值对参数的导数

孪生神经网络的作用.C00,C01,C10,C11矩阵是什么?

前向过程

我们先来理清由神经网络输出到到本征值之间的关系与步骤:

1. 由神经网络的输出得到矩阵和.
2. 对做Cholesky分解得到矩阵.
3. 计算的逆矩阵.
4. 利用相关矩阵对做变换得到.
5. 对进行对角化得到本征值.

反向过程

我们反过来理清每一步都是如何进行自动微分的(每一步分开实现并单独验证):

1. 计算本征值对矩阵的偏导.矩阵的对角化可以写成

可以写成更为紧凑的形式

于是

1. 计算矩阵对矩阵的偏导.我们先写出两者之间的关系

于是

1. 计算矩阵对矩阵的偏导.我们先来看矩阵是怎样影响矩阵的:以及.我们需要解决两个子问题:Cholesky分解与矩阵求逆的自动微分如何计算.我们把这步拆成若干小步进行计算:
2. 由对的偏导计算对的偏导.那么
3. 由对的偏导计算对的偏导.
4. 由对的偏导计算对的偏导.对由计算的代码过程进行逆推.对于一个3\*3的矩阵

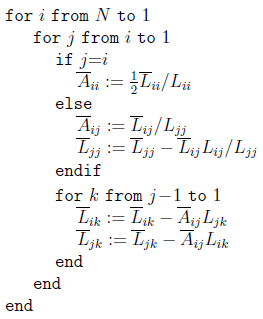
我们可以显式地写出的计算结果

我们可以看到几乎会影响所有的，而只会影响.于是我们可以从右下角开始向左向上计算，对于我们有

两边取微分

于是反向导数

具体的计算过程由



给出(An extended collection of matrix derivative results.).

1. 计算矩阵和矩阵对参数的偏导和.尤其需要注意对孪生神经网络的处理.需要计算对神经网络输出的反向导数然后再传播到.
2. 矩阵和矩阵都对有贡献.由输出我们可以计算

两边做微分

1. 由通过神经网络反向传播到.使用mini-batch的技术.

矩阵对矩阵的导数

对矩阵乘法，那么

1d-4well的求解

现在的问题是本征矢求解不对。将其细分为以下几步：

1. 为什么计算出的本征矢完全没有负数？因为偏置的初始化问题。偏置似乎都被学习成了零。
2. 训练效果不好，输出基本无变化，本征矢就是单调平滑曲线。
3. 学习率太慢?似乎不是，改大还是一样。增大学习率只是增大了振幅，网络并没有学到内部结构。
4. 梯度计算不对?验证下tanh的计算有没有问题，没有问题。
5. 训练方式的问题?

影响的因素有初始化，激活函数，最后一层线性层，网络结构，训练方式。总过应该有多少个参数?如果网络结构是[1,100,100,3]，那么权重有1\*100+100\*100+100\*3=10400个参数，偏置有100+100+3=203个参数；那么这样总共应该有10603个参数。但是参考的python代码却有11003个多出来400个参数。更加奇怪的是，将python训练后的10603个参数导出使用fortran计算却不能计算出正确的本征矢。需要验证是否使用这10603个参数就能计算出本征矢?

训练结果的验证

直接使用python进行验证。python输出与fortran完全一致，这说明并不是fortran神经网络的代码有问题，而是python神经网络前向传播的过程有未知的东西!那么这个未知的东西是什么呢?是剩余的那400个参数构成的辅助层还是其余的未知操作?通过对比得到剩余的那400个参数实际上是batch\_normalization的辅助层技术。

发现如果注释掉batch normalization技术，虽然结果变化较大，但是直接的前向传播依然完全得不到相似的本征矢。这说明除了这些未使用的技术层之外，还有未知的东西!这可能是Lambda层.

Python代码中的各个层

Python训练后似乎对模型进行了修改.必须细心地验证每样东西的作用.

1. Batch normalization层.对计算的本征矢有优化作用.
2. Gaussian noise层.有一些优化作用.
3. 如果不对训练好的hde.encoder进行再编码(加入Lambda层等)，那么结果与直接前向传播类似。说明再编码过程很可能就是关键。

对再编码(\_encoder->encoder)的过程进行细致的研究，发现加入的Lambda层使用了eigenvec/norm等信息，应该是关键作用。验证得出关键便是Lambda层的作用，我们将Lambda层的具体计算整理出来

所以缺少的东西主要是在训练之后使用全部的训练数据进行前向传播得到输出数据，然后使用输出数据计算均值和C/Q矩阵，对角化得到本征矢和本征值.在这个步骤中，主要是本征矢起了作用，均值只是增加了一个平移量.

Tensorflow中的batch normalization技术

Python中使用的batch normalization技术似乎是在两个隐藏层中各使用了一次。

Fortran中的具体实现

Python代码训练一次epoch即可得到不错的结果。不同训练本征矢可能不同，但是本征值是一样的，说明训练到的始终在一个子空间，这是合理的。而Fortran代码训练不到这样的空间，问题表现在

1. 本征值与python背景训练的结果是一样的，说明网络什么也没有学到.
2. 偏置似乎总喜欢学习到零值.

所以现在的核心问题是如何学习到正确的本征值.入手点是网络最后一层的偏置梯度为何为零?手误，输入变量错了。本征矢需要按本征值重排?加入了本征矢按本征值重排的操作，但是改变依然不大。

现在的问题是

1. 为何python代码没有最大的本征值1.
2. Fortran代码的学习实际上只是在原点附近振荡，并没有朝某一个方向学习.

我们把注意力集中到问题1，查看是python代码中的哪些操作使得其避过了本征值1.

中心归零似乎是一个必要的操作?还有可能是训练方式的区别，即随机选取数据进行训练而不是选取连续的一段数据进行训练?

修改fortran的训练方式，需要将原来的神经拆开成两个分别记录各自的中间变量，但是两个网络共享参数.修改了神经网络的变量存储之后第一个问题解决了，但是第二个问题改善依然不大.继续修改喂入数据的方式，并不适用连续的一段数据而是间隔大一些(也没有完全随机)，现在损失函数与本征值的计算看起来是正常的，但是仍然没能输出正常的本征矢?究竟是真得没有学习到还是输出不对?

Fortran中的具体实现

暂时认为是输出不对的原因，接下来我们把注意力转移集中到对输出的处理.影响本征矢输出的有哪些东西呢?网络参数和最后需要的变换矩阵.

1. 网络参数(权重和偏置)等训练好了以后便固定了.
2. 变换矩阵的作用像是将网络的直接输出投影到本征方向上.是在训练好之后计算的，所以具体结果依赖于网络参数.最终发现原来是变换矩阵使用错了，使用的应该是原来的矩阵而非!

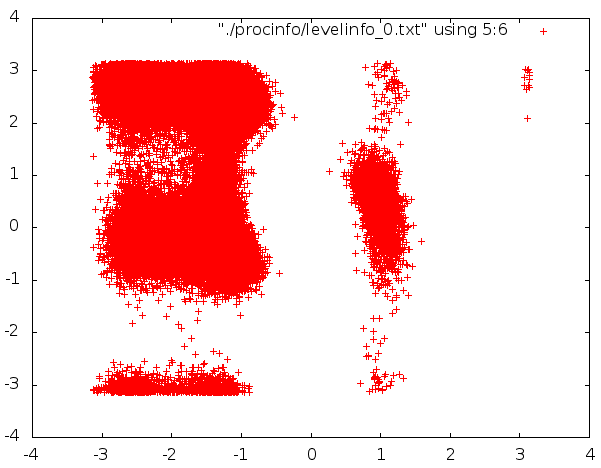
调试工作的经验总结

这一个多星期的调试工作告诉我们，需要将问题总结在纸面上并细心分析、对问题要有整体把握并对改善的方向进行准确判断.

ala的求解

数据的产生与输入

我们使用自己的显式溶剂模型产生数据.进行了两次模拟，第一次模拟没有采样到中间那个小岛，第二次模拟基本上需要的都采样到了.



Python代码的结果

损失函数大致到-1.06.本征值是[0.97597936, 0.35273443, 0.00652353].

现在的问题

现在处于一种上不能上下不能下的尴尬境地.程序整体没问题能运行，但是结果一般般.单点的梯度精度只有0.005左右，尝试增加输出距离的位数也只是减小到0.002左右.那么这个问题究竟是代码哪里有bug哪里精度确实只能到这样?按理ala与1d\_4well的区别只在于输出的节点增加到45个，精度应该也能到，但是整个算法也不需要改变啊，难道是计算逆矩阵的梯度时候的简化出现了问题?而训练的结果本征值基本可以[0.96147284, 0.37392729, 0.01233058].损失函数也在-1.06但是本征矢则依赖输入数据，大小/上下/左右上个本征矢一般有2个训练到，另一个可能就是重复似乎尤其是左右的重复?

现在就属于盲人摸象的状况，说差吧本征值没问题本征矢也总能学到两个，但表现总不是很好.只能一点一点想办法使之成阶梯式地改善.

1. 在每一轮epoch对序号重排，从而使mini-batch真正具有随机性.使用Knuth shuffle洗牌算法进行重排.
2. 研究矩阵对角化的反向梯度传播，是否需要修改?文献中完整的公式是



但是对于我们的情况，似乎因为损失函数只与相应的本征值D有关而与本征矢U无关，所以第二项为零，这样就可以简化成27)的形式.

1. 将python代码修改的部分修改回去，确保每次结果正常，这样才有一个较好的标准.影响python结果的正确性/稳定性的似乎是C0/C1的计算.C0/C1各自对应相加取平均似乎是为了确保是一个对称阵.修改这一点后单点误差已经可以很高(基本符合理论值)到.

现在的方向

现在的情况是改善了不少地方似乎已经没什么问题，但是最终结果还是不理想.这暗示我们现在处于一个什么样的状态呢?我猜想大致是已经经过这个问题的拐点，一般的难度大致已经解决，再进行一部分的工作就能解决这个问题了.那么现在的工作方向是什么呢?

1. 确认python的参数.神经网络按照[45,100,100,3]的节点数目确实共有15003个参数.最后的Lambda层有什么作用?有Lambda层也就是说还是进行了变换.
2. 输出的问题.最终结果不理想很有可能是输出的问题.加上Lambda层并对输出进行修改之后最终结果基本吻合了!
3. 需要对代码进行整合.现在基本问题已经解决了，现在需要花几天时间对代码进行整合.

代码的整合

首先确定代码整合的标准:

1. 能够通过三个测试样例.并且都有的变量放全局，各自的变量算自己.以第三个样例为主，确保第三个样例正确的情况下让另外两个往第三个靠拢.第三个样例的变量基本整合完毕.
2. 整合变量的表示，节点/batch/steps.简化删去中间性质或者不需要的变量.
3. 整合函数.损失函数的指定，相同功能的统一实现.尤其是统一激活函数、损失函数、训练过程和预测过程.激活函数与损失函数的接口基本成型，rnn的需要整体写好之后再调试sin样例最后再修改.接下来需要写训练过程的接口然后调试三个样例最后再进一步整合和修改.

步骤按先变量再功能替换再修改进行.

进行增强采样

增强采样的基本思路是先进行一段时间的模拟得到数据训练好神经网络.然后在正式模拟的过程中将某个时刻的结构通过神经网络算出主分量，然后反向修改辅助势从而进行增强采样.我们选择1l2y进行每一步的具体测试:

1. 通过sim进行模拟得到数据.尤其是153对*Cα*-*Cα*原子距离.仍然使用之前八月份文章中1500ns的gb数据，总共有7500个数据点.
2. 使用这些原子距离对srv模块进行网络训练.
3. 现在基本的运行框子打好了.需要调试运行成功与初步看到损失函数的下降.现在的问题是:lmat逐渐变小，linvmat逐渐变大，主要是outputs0/1逐渐减小.进一步发现是因为更新参数导致的，如果不更新则对所有的输入数据是一样的.这说明参数更新的步骤有问题，会是学习率太大导致的?

输出为什么会消失?是更新之后的参数有问题.减小输入层的节点数缩小范围.如果输入层改成45个就不会出现这个问题.90也正常.135似乎也正常.大概在148-153之间会出现输出消失的情况.修改随机种子，问题似乎解决了.可能是有些随机会有产生输出消失的问题.

1. 现在损失函数没有明显问题之后，我们需要展示计算出来的本征矢看是否有明显问题.
2. 八月份通过PCA计算的本征值: 0.973,0.178,0.099.现在使用srv计算出的本征值是: 0.965,0.569,0.086.
3. 我们对于153个输入节点，每回只给一个节点值: (1,0,0,…),(0,1,0,…).看三个输出值在这些节点上的权重.

这部分工作现在的问题是数值不稳定+结果不理想(本征值会变，本征矢无明显指向).

1. 重新进行显含水模拟并利用训练好的神经网络进行增强采样.
2. 将训练好的网络参数(权重加偏置储存下来).
3. 重新建立网络结构并读取网络参数开始预测.
4. 实现增强采样模拟中flooding step时，由计算得到的结构计算原子对距离，由原子对距离前向传播输出，由输出反向更新辅助势。
5. 需要明确辅助势是定义在那个坐标下的?笛卡尔坐标、输入坐标还是输出坐标.
6. 反向更新的方式需要对输入坐标求导.

实际的操作需要获取/读入一个结构->计算输入坐标->输入坐标->更新辅助势，关键点在于辅助势的表示与更新.

反向导数的计算

SRV的作用是将笛卡尔坐标降维到属性坐标.而增强采样的作用是学习属性坐标下的辅助势.辅助势会对原子产生额外的力

第一个因子由属性坐标构造辅助势的具体形式确定(统计分布+插值?)，第二个因子由神经网络的结构和参数决定，下面我们关心第二个因子的计算.我们显式地写出单层隐藏层的计算公式

相应的导数

依次前向传播回去，只与权重有关，与偏置无关.

实现了反向导数的计算之后，现在真正开始利用训练好的神经网络进行增强采样的工作.

1. 将srv嵌入colvar.f90并构建显含水增强采样的实例.程序运行的逻辑基本走通，从flood.f90->colvar.f90->srv.f90.开始着手前向传播过程，需要先计算原子对距离.flood.f90中有四次colvar\_fsainfo，第一次三个副本都会前向传播，中间两次似乎没有影响，最后一次只有最后一个偏倚副本会反向传播.需要在反向传播后stop不然会报错，可能其它地方还有一些问题?
2. 验证实例的结果并开始调试.

现在正向传播和反向传播已经可以形成闭环且运行没有问题.接下来我们需要找个方法Check一下有没有问题.

我们以10000步时候的结构为例.增强采样的逻辑链条是原子坐标->网络输入->集合坐标->偏倚势->偏倚势对集合坐标的力->偏倚势对网络输入的力->偏倚势对原子坐标的力.所以真正的因果关系是原子结构->原子力的修正.于是一个很自然的验证方法便是我们修改某个原子的坐标然后观察原子力的变化.现在将网络输出到偏倚势的部分直接设为1，那么从原子坐标到网络输出的部分是已经通过了.

我们开始测试增强采样对1l2y的实际效果.需要做三个独立的模拟:不进行增强采样，使用PCA进行增强采样，使用SRV进行增强采样.

现在的问题

真正使用训练好的SRV网络进行增强采样的效果并不好.如何改进?查看有关使用SRV与增强采样的文献，问题与代码的分析.

问题与代码的分析

训练效果不好可能来源于网络训练不好、采样代码有问题或者采样模拟设置有问题.用类似于文献中的对Trp-cage的本征矢分析验证/排除网络的问题，采样代码是我觉得最可能出现问题的地方(需要一边修改一边重新做模拟)，模拟设置我觉得只要和另两个一致应该问题不大.所以我觉得现在的方向大致是修改采样代码并同时做模拟，并查看网络的问题，一点一点改进.

有些地方有明显的问题，从这些明显的问题入手。levelinfo\_0.txt中srv输出有问题，surfacedata\_2.txt中出现NaN的情况.

小插曲:帮助使用openmm输出信息调试了好几个小时。以后注意:openmm在server和node3上各有一套，server上是conda安装、可以和parmed搭配使用来测试;node3上是源码编译需要make,make install,make PythonInstall，与parmed搭配有点问题(版本?)但是可以修改源码.

对于现在增强采样的问题，首先是为什么集合坐标输出值几乎只取两个值，需要修改集合坐标取值范围.现在修改范围重新进行模拟之后发现集合坐标输出值并非只取两个值而是基本只集中于两三个值(相差了百千倍)，必须进行短时间模拟的细致测试.现在发现*Cα*-*Cα*原子距离的计算与读入没有问题，那么问题就是为什么读入不同的距离数据，却得到可以说是完全一样的输出?从网络隐藏层的输出来看，产生这种情况的原因在于输入层的数据到第一个隐藏层之后便饱满了，有大量的数据使得第一个隐藏层神经元处于极值状态(1,-1)而未被激活.我们必须对输入数据做归一化操作.

归一化层的使用

归一化的操作应该包含在神经网络中而非包含在数据的产生和输入中.我们先尝试最简单的最大-最小归一化

这样的话反向梯度

需要注意的是，在输入层之后引入归一化层并不会对网络参数梯度的反向传播带来任何影响.

现在的方向

现在基本的功能和操作都已经实现了，代码运行也看不出来有什么明显的问题，我们需要确定现在工作的方向。方向无非两个部分，一是继续对1l2y做srv相关的大量测试，二是对现有的代码进行整合和优化。这两个方面的工作应该同时进行但是偏向于前者。