1. 基于目前的网络主要可调参数。n\_hidden: 隐藏层尺寸，越大参数越多； n\_sample: 总sample数；n\_chain\_per\_rank: 每个rank上chain的数目，单进程只有一个rank；n\_sample\_per\_chain：每个chain上有多少个sample，这个是基于总sample数和总chain数算出来的；：regularization 项，在计算梯度的时候假定loss函数里面除了能量还有这一项（类似于机器学习中的regularization 项），但是在计算能量的时候并没有加上这项，如果设置，等价于不要这一项。Mover：可选BitFlip(), BondSwap(charge=n), 或者FermiBondSwap(charge=(n,n))，理论上来说这三种选择的采样效率是由低到高的, BondSwap(charge=n)，这里面的n=2\*电子数-总比特数，FermiBondSwap(charge=(n,n))，这里n = (2\*电子数-总比特数)/2。最后learn\_rate和diag\_shift原则上也是可调的，不过根据经验目前都设置为0.01是比较稳定的。梯度优化算法目前用的是ADAM，也可以任意换成Flux.jl里面提供的其他优化器。

2. 目前使用的网络我称为FCN，基本上就是一层Dense加一个nonlinear activation 函数（tanh）。可以直接看源代码里面的定义。理论上来说可以换任意的神经网络，自己定义并搭建一下就行了，可以参考我的源代码rbm.jl里面的网络搭建。新网络的主要接口函数是，从一个比特串映射到一个振幅，或者从一个batch比特串映射到一列振幅。还需要用Flux.@functor 新网络来标记下以便Flux可以自动找到新网络里面的参数。从目前的经验来看，神经网络态应该尽量选用平滑的函数，而不是relu这种不平滑函数。这是因为神经网络态通常都不是表达能力的问题，而是采样空间太大很难从随机生成的比特串通过MC走到relevant的采样空间（相反，MPS通常不存在收敛问题，而是表达能力的问题）。**构建一个合适的神经网络，以及给一个合适的MC的初始态，可能是我们这个项目成功的关键！**

3. Mover 也可以自定义，mover.jl里面定义了三种mover。原则上来说满足ergodicity的mover都可以，但不同的mover采样效率可能有差别。但似乎没有一个更高效的办法来定义新的mover了。

4. h2.jl里面是串行程序，parallel\_h2.jl里面是并行程序。julia h2.jl 运行串行程序，julia -p 5 parallel\_h2.jl 运行一个5进程(还有一个master进程，所以这样实际上是6进程)的并行程序。

5. 一种可能的加速收敛的方案：MC采样的时候，初始比特串不随机生成，而采用HF平均场解？

6. 大分子时，mol\_hamiltonian.jl里面的coupled\_states函数需要进一步优化，目前这个函数是计算瓶颈。但我们目前主要重复ChooCarleo2020里面的结果，暂时应该不需要优化这个函数。