SNN仿真框架-C代码

按照之前的思路进行设计，以Matlab的那份程序作为辅助参考。

对于C代码而言，类的控制和封装，数据的具体处理与格式更容易控制，并且，到时候和芯片的具体软件实现更加贴近。从仿真效率而言，CPU串行运算效率较高，并行运算则需要进一步考虑有关的GPU加速方案，第一阶段暂时不考虑。

程序仿真方案

每个Frame时间起始，得到仿真数据，该Frame结束后，对仿真结果进行分析，给出最终分析结果。但该Frame时间内，以sample时间Ts为基本仿真步长，进行sampleBased的仿真。对于ANN系统而言，则都是FrameBased。

主要处理模块有：

Spike\_input\_generator：根据原始输入，得到输出的脉冲序列组，相当于一个Analog->Digital的转换过程，输入参数有：Xin，In\_Size，In\_len，通常，In\_len=FrameTime/Ts，或者考虑到系统延时，尾部加上若干的0值。

Neuron\_NN\_pro: 计算Neuron的各种变化、状态等值；基本上是sampleBased，以Ts为基本仿真步长进行计算。从仿真角度，以及描述网络的便利性，是以各个Layer为大的数据处理块进行仿真的。假设，网络结构Ln层，每层用Layers(i)表示，Layers(i)中含有NeuN(i)个神经元；神经元模型用NeuronModel表示。最终输出在NeuronUnit(Ln-1)中，该输出在一个Frame时间内的所有输出存储为Xou。

Neuron\_out\_pro：根据Xou统计计算You，相当于Digital->Analog的转换。若是为了节省Xou的统计部分（因为有in\_Len\*NeuN(Ln-1)个数据），可以考虑每个Ts就运行一次，节省有关的存储开销。

Judge\_pro：这个是对You进行softmax等判决运算了，计算error，CostFunction等。

Main函数中：

得到测试数据和期望的结果；

得到仿真所需的SNN

For each test\_data, expected\_data：

Neuron\_Sim\_Process(test\_data, snn, &res\_data); // 得到处理结果

Judge\_process(res\_data, expected\_data, &judge\_res); // 所有要处理的比对结果存入中judge\_res中

// 最后，可能有再进一步的统计分析

Analyze\_process(judge\_res, fin\_res); // judge\_res是一个结构体数组啥的，最终结果存入fin\_res

Neuron\_Sim\_Process(test\_data, snn, &res\_data); // 得到处理结果

//对test\_data进行计算，另外将有关的调试信息保存下来

Snn.reset(); // snn清空内部的运算数据 // 显示处理，避免遗漏

Neuron\_simu\_one(test\_data,snn, &res\_data, ); // 系统层面，改到SNN自己内部处理?涉及到外面数据分析

Debug\_analyze(res\_data);

核心仿真：Neuron\_simu\_one(test\_data, snn, & res\_data) // 含有外面数据分析，不在SNN内部处理

spikeSeq = Spike\_input\_generator(test\_data); // 根据输入得到spike序列

for tidx in FrameTime/Ts: // 一个Frame时间内

inX = spikeSeq(tidx); // each spike

snn.Neuron\_NN\_pro(tidx, inX, &outX);

Neuron\_out\_pro(outX, &outY);

Get\_one\_neu\_res(tidx, outY, Res\_data)； // 提取所需的信息保存到res\_data

// other process

Snn.Neuron\_NN\_pro(tidx, inX, &snn, &outX) // snn对inX进行处理，得到此次处理结果，相当于forward

// 若snn本身有反馈信息，是在snn内部做处理，对接口不做变动

InX2 = data\_trans(inX); // 转化下数据方便后续处理

For Lidx = 1: snn.Ln-1 // snn的每层进行有关的处理

Snn.layers[Lidx].ModProcess(inX2, tmp\_out); // 处理

inX2 = tmp\_out;

NN\_out\_pro\_oneTs(snn, &outX);

Snn.layers[Lidx]. ModProcess // 这个是对一个Layer的处理方式，对Layer定义成一个类

根据layers的类型和有关参数进行处理，目前，可以直接处理的类是：FCN,CNN和Pooling\_Ave，关于Pooling\_Max，需要进一步研究，对于复合类，需要进一步研究其处理方法。对于复合类，其内部是一个复合的NN，由各种Layers组合完成，对于并行的，最终将并行结果再按等权相加，对于cast合并在一起的，将数据组合起来而已；另外，还有一个特殊的处理，就是展平处理，这个在输入做还是输出做呢？在输入做。注意的是，展平的时候，一般的做法是Map\*Co，Map为X\*Y，先按照先Y后X再Co的方式进行的。但是，pytorch中，是先X后Y在Co，对不上？在C代码中，先确定好数组为[Co][X][Y]就好，那么，在做卷积时，一个Map中为[X][Y]，不是对应图中的[X][Y]？因为，图中的(x,y)对应的是MAP中的[Y][X]，是的，Matlab中对于图像的处理就是Y\*X。C代码中，是按照自然顺序[Y][X]来进行，即row,col的顺序，某个（x,y）对应的？说明pytorch中的kernel的含义是[yi][xi]而不是[xi][yi]，需要弄清楚这个含义。

整个过程分成2个大部分，一个是SNN的高层仿真，另一个是SNN的底层仿真。

SNN的高层仿真：

从算法层面来进行SNN系统的高层仿真，步骤为：

建立SNN的系统仿真框架；

根据ANN转SNN的方法，得到SNN仿真所用的权重等参数；

针对具体的ANN的网络进行SNN的高层仿真；

确定好SNN的系统架构，主要是仿真LIF模型。

SNN的底层仿真：

结合芯片架构进行SNN系统仿真，3个方面：

加速核Neuron Core的行为仿真；

对加速核Neuron Core进行调度的系统仿真；

系统中加入NoC功能后的系统仿真；

从项目角度来说，根据网络的复杂程度和网络类型进行有层次的仿真。

XOR：验证SNN系统的基本功能

LeNet：先采用MNIST数据集进行验证

随后是demo中所要求的各种仿真，有：

阿拉伯数字识别，cifar-10识别，MobileNetV1；语音识别；在线学习；

1. demo场景是否合理  
2. 算法选择及评估计划  
3. 算法验证，架构验证方案及计划  
4. 细化算法在SNN芯片上的部署

算法选择与评估计划：

根据场景选择已有的类似场景的ANN算法，根据算法的性能和复杂度进行综合评估；具体计划为：

调研类似场景的ANN算法和所需要的数据集；

根据文献中各算法的性能和复杂度，确定2~4种算法方案；

根据数据集和需评估的算法方案进行ANN算法仿真，进行综合评估；

若需进行SNN的相关demo，则需再进行SNN的系统仿真；

如此，根据最终的算法复杂度和性能情况进行最终的算法确认。

主要工作量：ANN算法仿真的优化（权重定点化）和SNN算法仿真，涉及到需要进一步的优化结构和参数。

3. 算法验证，架构验证方案及计划

ANN算法验证：根据具体的仿真工具，先在电脑上进行性能仿真，最终在FPGA上进行系统级别（SOC）整体仿真；

SNN系统验证，整个过程分成2个大部分，一个是SNN的高层算法仿真，另一个是SNN的底层架构仿真，其中：

SNN的高层算法仿真：

从算法层面来进行SNN系统的高层仿真，步骤为：

建立SNN的系统仿真框架；

根据ANN转SNN的方法，得到SNN仿真所用的权重等参数；

针对具体的ANN的网络进行SNN的高层仿真；

确定好SNN的系统架构，主要是仿真LIF模型。

SNN的底层架构仿真：

结合芯片架构进行SNN系统仿真，3个方面：

加速核Neuron Core的行为仿真；

对加速核Neuron Core进行调度的系统仿真；

系统中加入NoC功能后的系统仿真；

4. 细化算法在SNN芯片上的部署

根据SNN系统仿真的情况，依次进行最终的算法部署与验证。