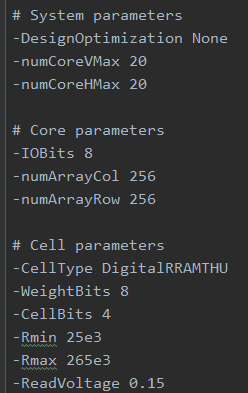
XPE仿真器学习笔记

硬件参数配置文件：

sim\_examples/simconfig文件 用来设置硬件参数



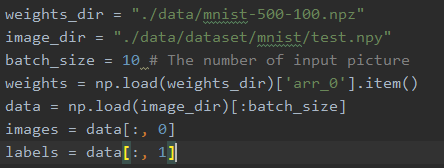
# 一、MLP测试文件：

sim\_examples/inference\_mlp\_loadweight.py

该文件包含了MLP实例，显示了MLP的算法评估和硬件评估。

仿真过程：

## 读取数据：



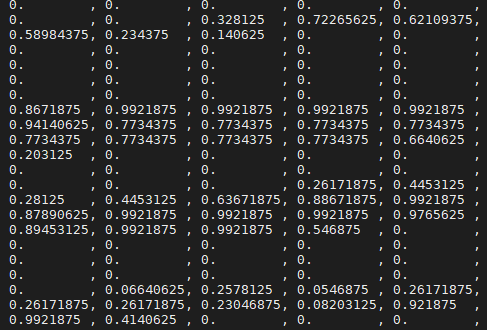
这是一个分类网络：数据集包含图像和标签

网络参数和数据参数目录：

sim\_examples /data/mnist-500-100.npz



sim\_examples /data/dataset/mnist/test.npy



文件类型为 npz和npy类型

修改网络模型时，对应的数据集改为相同类型；

## 从配置文件中读取硬件配置：

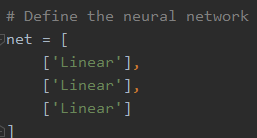


目录：sim\_examples/simconfig 得到的params是一个类，成员变量是具体参数，

具体声明在simulator/simulator/nnsim/parameterinput.py文件中

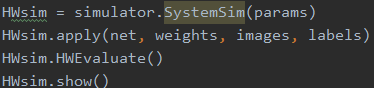


## 定义网络层数和类型：

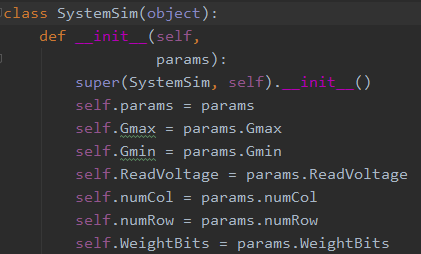


定义神经网络：Linear为全连接层 Conv2d 为卷积层

## 仿真过程



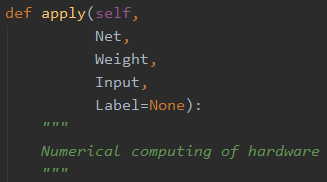
HWsim得到一个类 具体定义在simulator/simulator/nnsim/systemsim.py 文件中



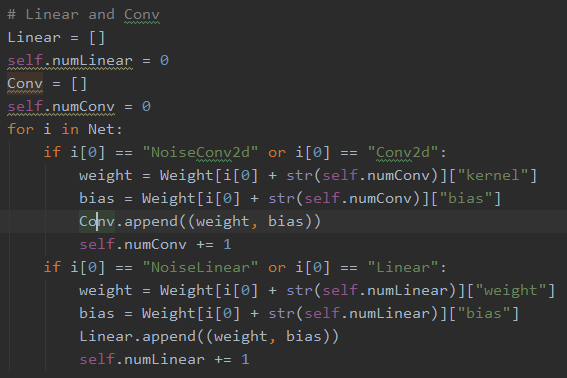
使用配置文件中读取的参数创建类的实例HWsim

### 一、调用SystemSim类的apply方法

将网络结构、权重参数、测试图片和对应标签数据传入；



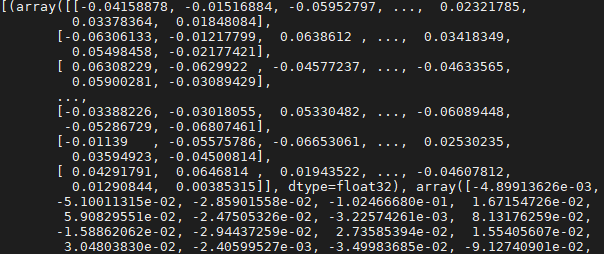
声明两个列表Linear Conv用来存放 全连接层、卷积层的权重和偏置



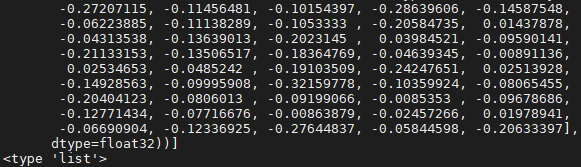
数据格式：得到 对应层类型和层数的weight和bias参数列表项。

Linear是一个列表，键值对为 （weight，bias）键和值都是数组，包含一层的权重和偏置

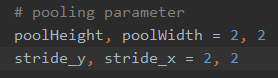




Conv2d也是一个列表

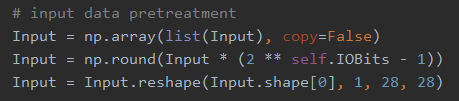


手动修改池化参数

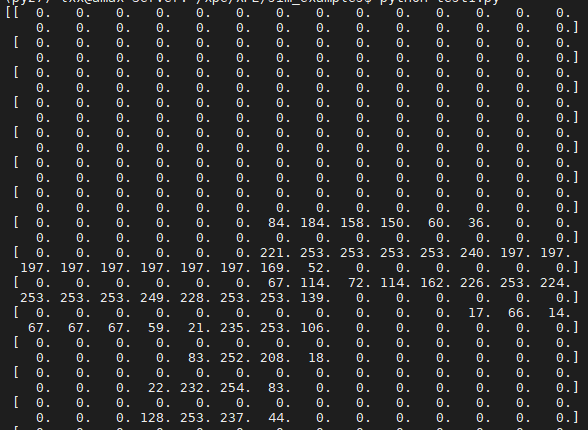


输入数据预处理

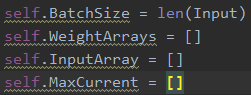
输入数据是归一化以后的浮点类型小数，需要先还原成0-255灰度值的，先把元组转换为列表修改，最后规整为 10 \* 1 \* 28 \* 28 将来修改网络结构需要修改对应大小。



查看Input[0][0]的数据：依旧是float类型



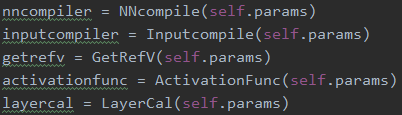
BatchSize 大小设置为10



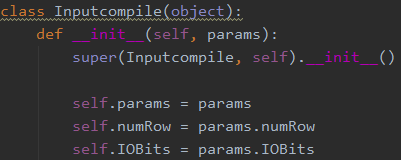
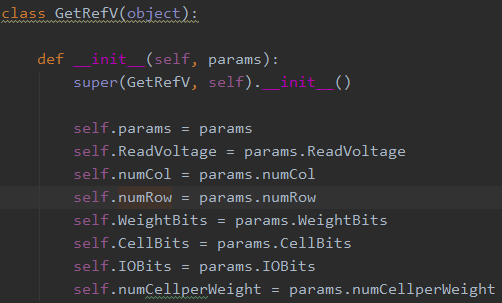
将参数赋予内置的函数执行：

函数定义在 XPE/simulator/simulator/nnsim/module/nncompile.py、inputcompile.py、

Getrefv.py等对应名称文件中

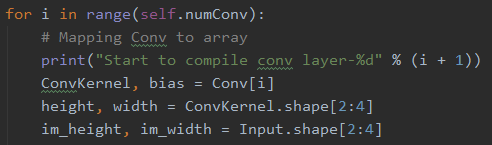


函数的具体实现代码如下：

 略。

每个类都拥有一个apply方法。

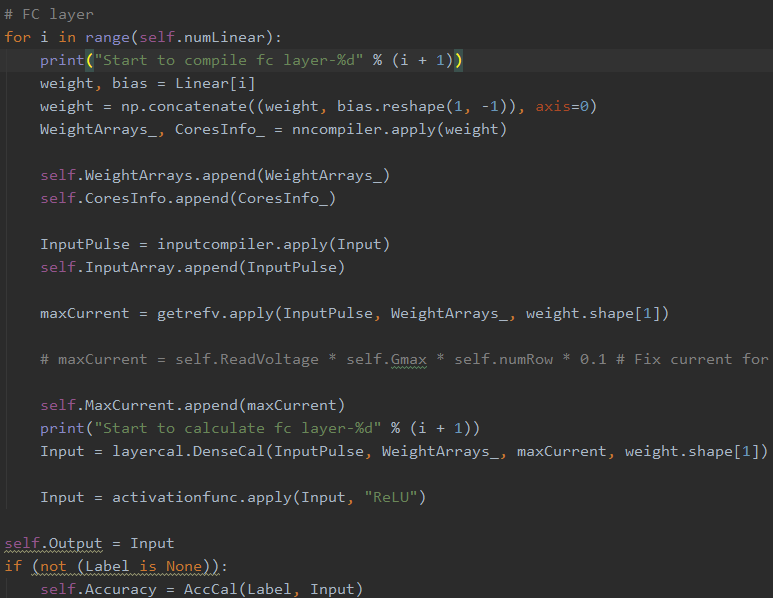
下面是一个基于网络中Conv层数的循环，由于MLP只有3个FC层，这个循环略过。（cnn的example跑不通，这部分代码存在bug）



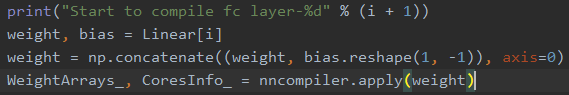
原来(10 \* 1 \* 28 \* 28)的输入被整型成(10 \* 784) 形状



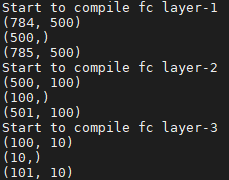
下面是针对全连接层的处理：



首先将每一次的参数从列表Linear取出，将weight和bias拼接为一个数组，数组作为参数调用nncomplier的apply方法。



各层数组拼接前后得到的数据大小：

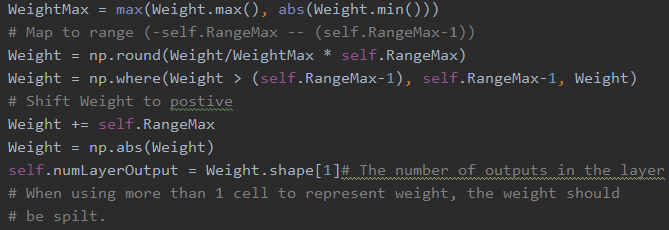


最后调用/home/lxx/xpe/XPE/simulator/simulator/nnsim/module/目录下的nncompile.py

#### 1. nncompile函数调用过程：

函数功能：将权重映射到数组。

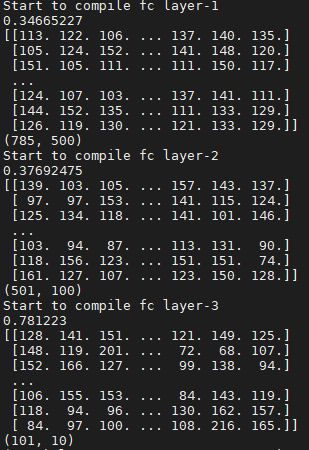
需要参数：weight：一层（float）的weight和bias



前三行代码将参数weight（其实是weight+bias）映射到（-128，127）范围内。

第四行将数据范围修改为（0，255） numLayerOutput等于weight数组的第二维度。

下图为各层WeightMax值和量化完成的Weight值和维度：



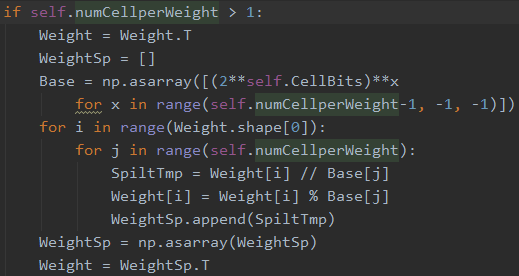
如果使用超过一个核来表示weight（参数numCellperWeight表示），则应将weight切片。

该参数在/home/lxx/xpe/XPE/simulator/simulator/nnsim/目录下的parameterinput.py中定义

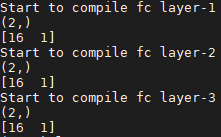


公式中两个参数在配置文件simconfig中设置，默认 8/4=2

具体切片操作如下：



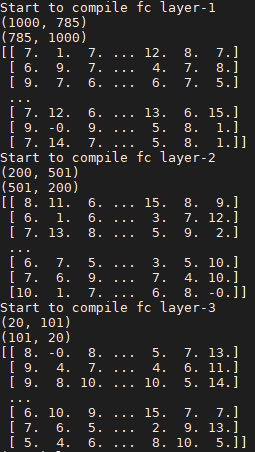
得到的Base的维度和内容如下：



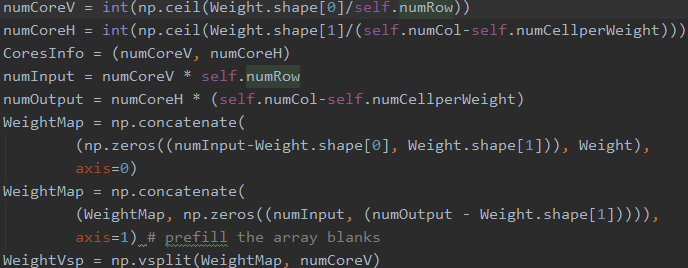
之后使用一个双重循环将Weight数据按Base位切割 默认核数量2 核的位宽4

所有8位数据被切割为前四位和后四位。每次整除完取余，数据每4位切割开。

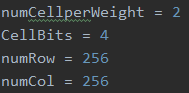
得到数据格式如下：



把参数编译入数组：



参数值由配置文件定义

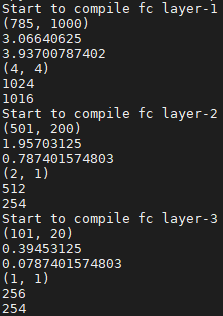


NumCoreV等于参数行数除以预设的numRow数

NumCoreH 等于参数列树 除以 预设numCol数减每个权重占用核的数目

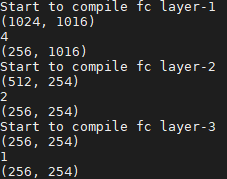
猜测是用于行有效位的判断

最后得到各个参数值如下：numCoreV numCoreH CoresInfo numInput numOutput



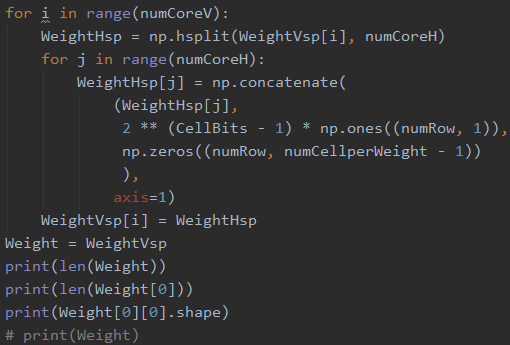
之后将参数不够的位数补零 使维度与阵列对应。

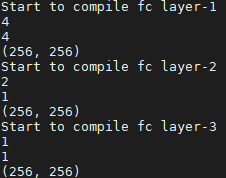
再按片数按行切割为列表 填充后维度 和切片数如下：



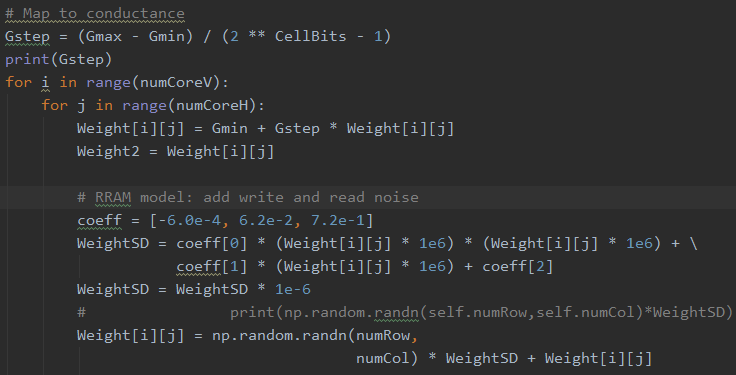
将1016列分割成四份，每份末尾填充一列8 一列0

将Weight重新赋值成嵌套列表 每一维度大小如下：





Weight现在是一个四行四列的列表 每个单元大小256\*256

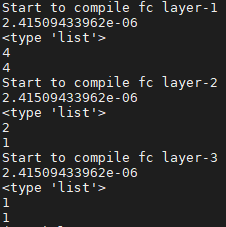


这一代码块映射到电导，Gmax Gmin都是配置文件simconfig中预设的。

下面的coeff是Reram模型的计算公式，最后得到更新的Weight，

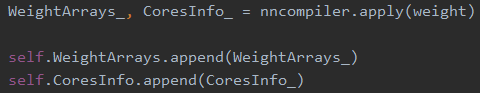
其中定义的Weight2没有被使用，这行代码可以删除。

下图是各层的Gstep Weight的两个维度



函数调用完成，返回Weight和CoresInfo（因为在循环中，每一层返回一次，下面调用其他函数时还会用到。）

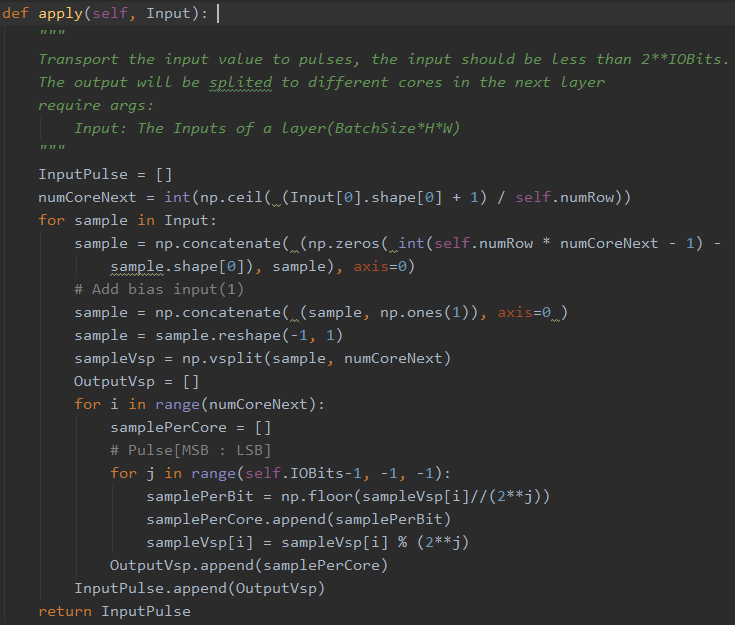
#### 2. 返回上一层函数：systemSim.py的apply()方法



循环执行完会将三层fc的Weight 和CoresInfo加入到这两个类的变量中，

Weight是4\*4\*256\*256 的列表 CoresInfo是3\*2的列表，表示三层的纵向和横向块的个数。

#### 3．inputcompile.py函数调用过程

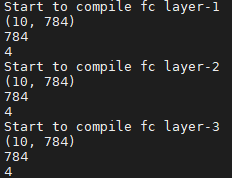


将输入值传输为脉冲，输入应小于2 ^ IOBits。这里是256

输出将在下一层拆分为不同的内核

需要参数：输入：图层的输入（BatchSize \* H \* W）

得到的numCoreNext 和Input维度如下



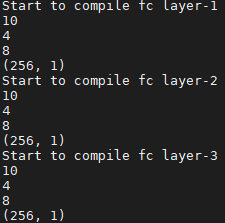
之后sample被整流成1023大小，最后再加一行1 组成1024\*1的二维矩阵

sampleVsp得到sample被切分成（256\*1）大小元素的列表

下面一个双重循环将4\*256\*1的列表中存储的每一位取出来，存入InputPulse

InputPulse的维度为10\*4\*8\*（256\*1）

10是batchsize的大小，因为有10张图片、4是输入被存在的核的个数。8是把数据按位切割开得到的每一位的大小（0或1）



#### 4 . getrefv.py函数调用过程

在层的核心之间找到最大电流作为AD参考电流

需要参数：

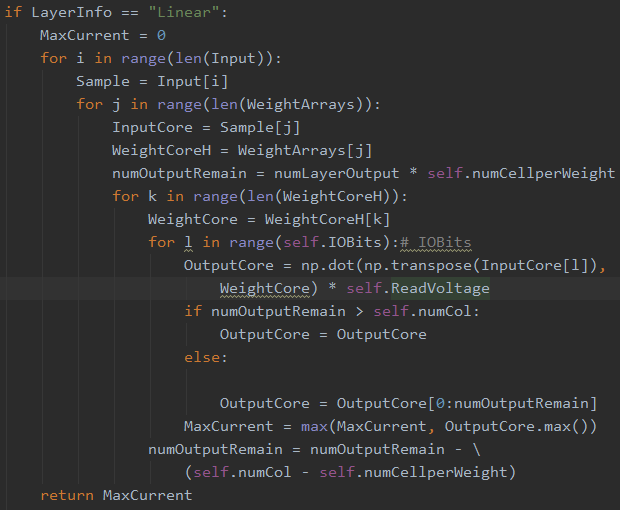
输入：图层的输入

WeightArrays：图层的编译权重

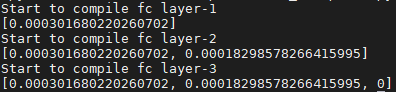
numLayerOutput：图层的输出大小

LayerInfo：“线性”或“转化”

ReadVoltage在simconfig文件中配置，默认值0.15



执行完成后得到各层的最大电流



不知道为什么第三层的最大电流是0，

#### 5 . layercal.py函数调用过程

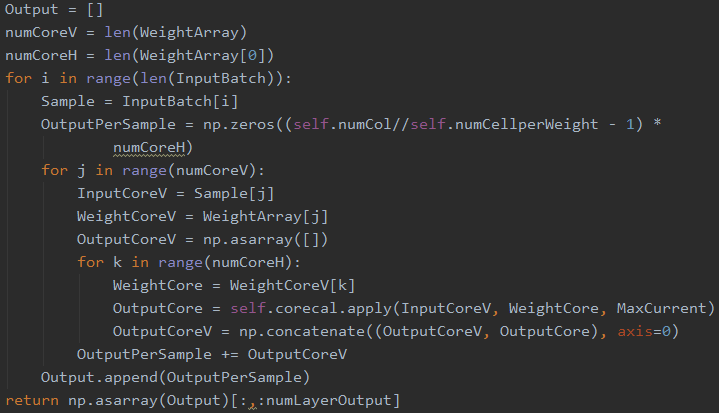
计算密集层中的输出

需要参数：

InputBatch：图层的输入

WeightArray：图层的已编译权重

MaxCurrent：层的参考电流



##### corecal.py函数调用过程

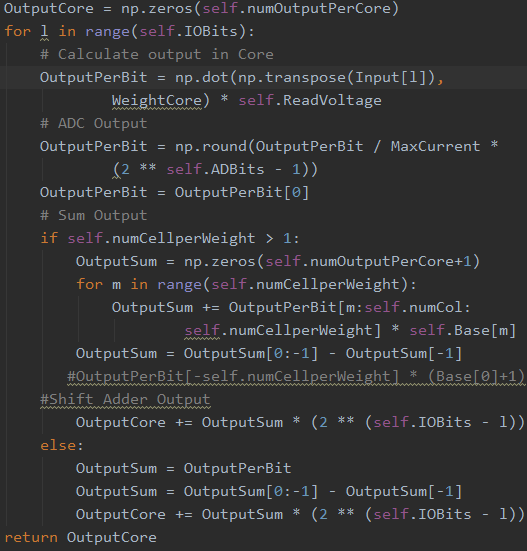
计算核心输出

需要参数：

输入：核心的输入

WeightCore：核心的已编译权重

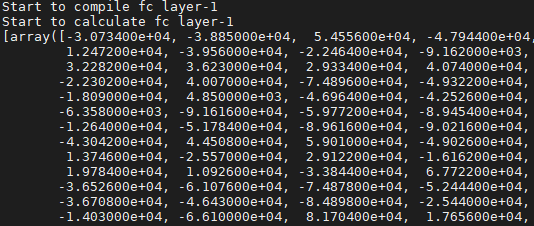
MaxCurrent：层的参考电流

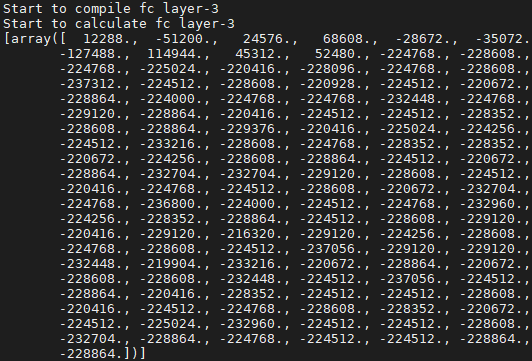


通过公式计算核心输出，公式选择与核心数相关，

循环结束得到的中间结果OutputCore会返回上层layercal的循环中，

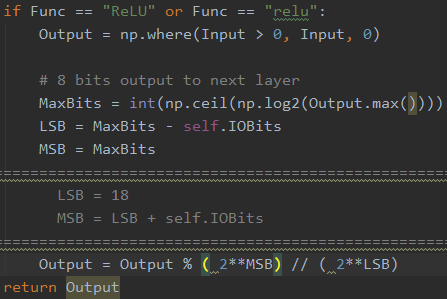
拼接并加入到最终结果Output





第三层的输出远大于前两层，

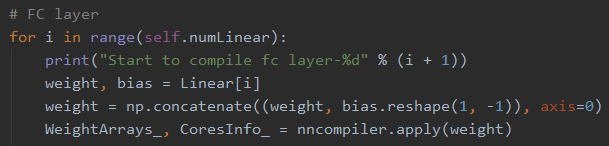
#### 6 . activationfunc.py函数调用过程



只有relu的激活，输出赋值给Input作为下一层输入，传入下一层。

#### 7. 循环调用

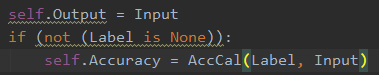
上层更新过的Input，执行下一次循环时，作为参数送入1-6中重复计算过程



最后一次循环结束得到最终结果，存入Output，

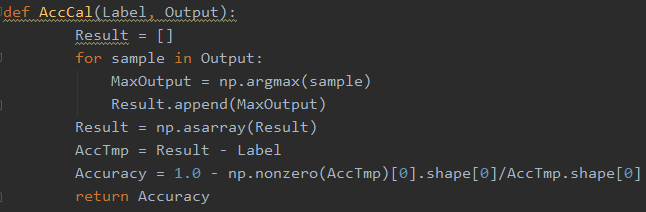
#### 8. 验证分类准确性

最后验证MLP网络对mnist数据集的分类准确性Accuracy



AccCal函数定义于/home/lxx/xpe/XPE/simulator/simulator/nnsim/目录下

Sysfunctional.py文件中，



MaxOutput保存每一维度最大值，因为一个batchSize10张图片，Output大小为10

MaxOutput对应预测数字，存入Result数组

Result数组减去Label数组，预测正确的元素值为0

1.0减去非零元素占比即为预测准确性

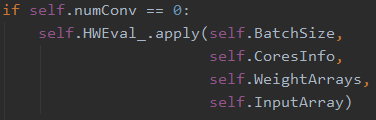
下图是计算值与实际值，还有预测准确度，100%



仿真结束。

## 评估硬件性能

### 调用SyetemSim类的HWEvaluate方法



MLP的numConv为0，所以调用hweval的apply方法

#### 1. hweval.py函数调用过程

评估硬件性能。

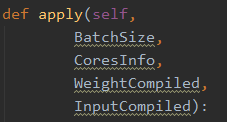
需要参数：

BatchSize：批处理的大小

CoresInfo：分裂核的信息

WeightCompiled：编译后的权重（numLayrs \* numCoreV \* numCoreH \* numRow \* numCol）

InputCompiled：编译的Inputpulse（numLayers \* BatchSize \* numCoreV \* numBits \* numRow \* 1）

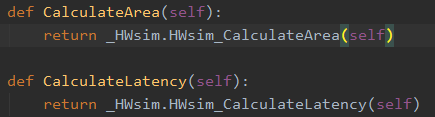


通过仿真得到的BatchSize CoresInfo\_1 WeightArrays\_1 InputArray 数据传入函数中，

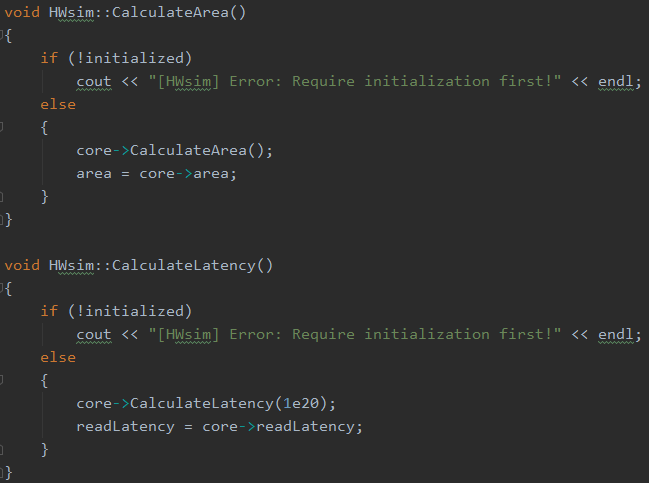
计算面积和时延



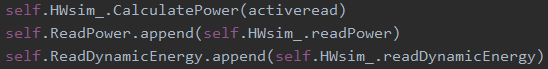
对应函数包装在/home/lxx/xpe/XPE/simulator/simulator/hardwaresim/目录下的HWsim.py



实际实现在相同目录的HWsim.cpp文件中，C++编写的底层功能模块



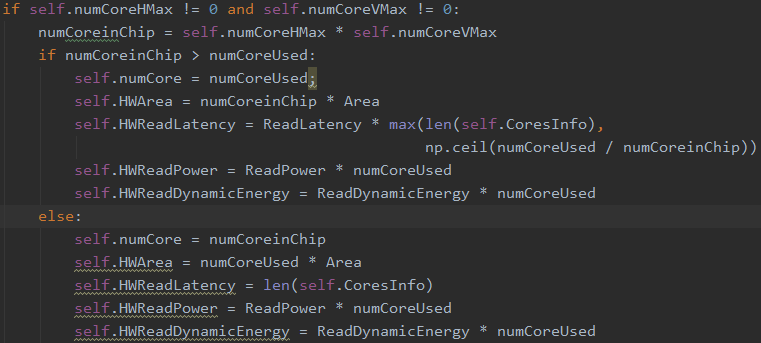
计算功耗和能量



具体函数的封装和实现同上，计算得到中间结果:单片面积、单核功耗等

#### 2. 返回上层HWEvaluate方法

由中间结果计算得到硬件参数值：面积=片数\*单片面积

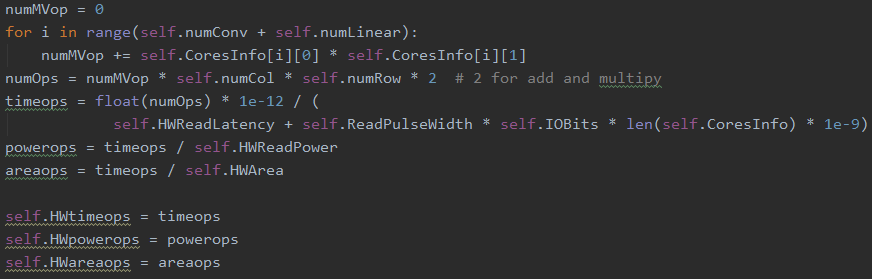


计算timeops powerops areaops的公式是固定的；

都是用来评价处理器运算能力的性能指标。

powerops 用于度量在1W功耗的情况下，处理器能进行多少万亿次操作。

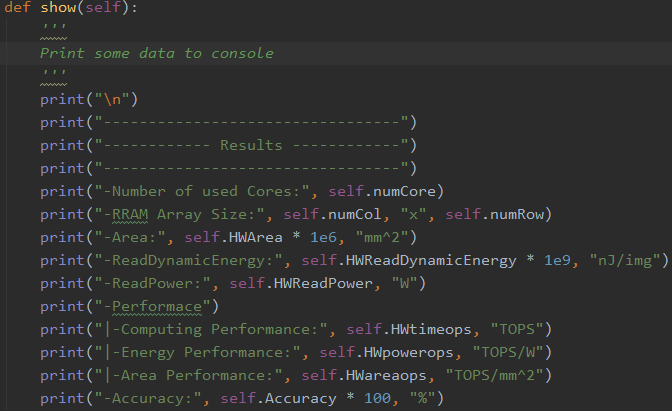
areaops 用于度量在单位面积上，处理器能进行多少万亿次操作。



## 结果展示

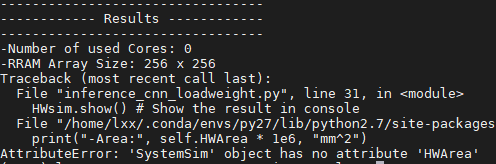
调用SyetemSim类的show方法

将参数打印到终端



# 二、CNN测试文件

五层CNN模型的测试文件运行报错



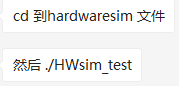
修改对应文件后依然报错

解决办法：在命令行cd到/home/lxx/xpe/XPE/simulator/simulator/hardwaresim/目录下

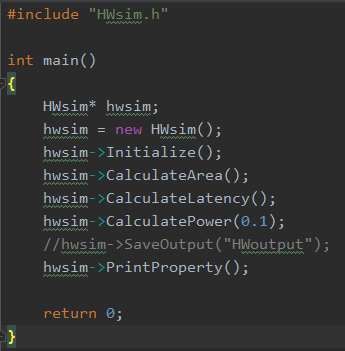
在命令行中直接运行 HWsim\_test



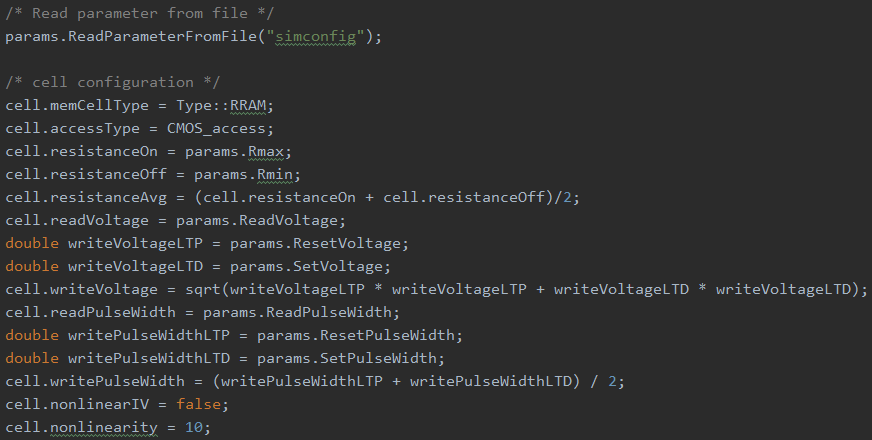




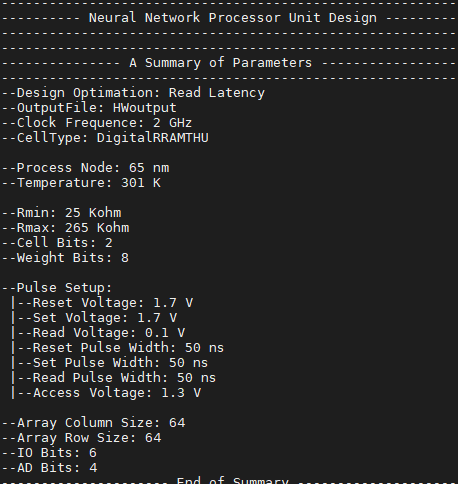
HWsim\_test.cpp文件内容如下：因为是C++直接调用，计算参数不知道如何获取的

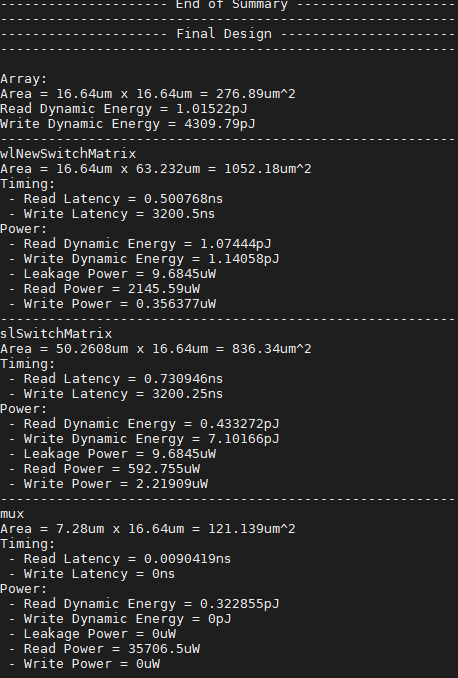


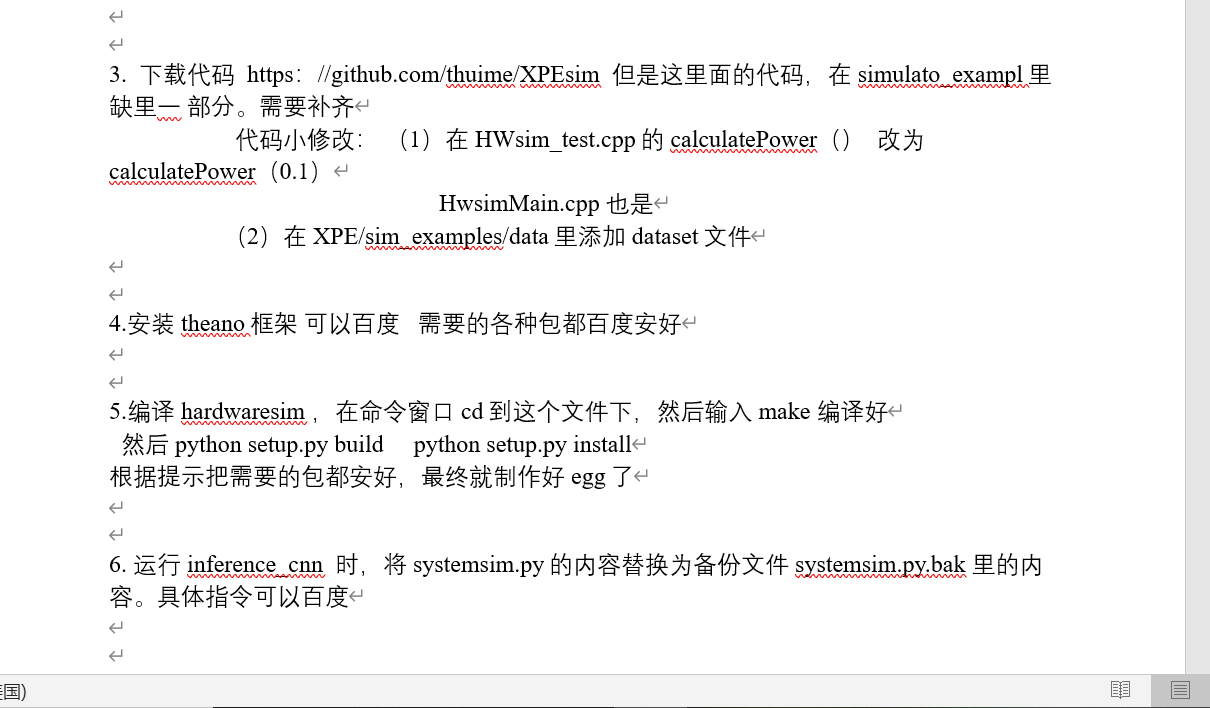
计算参数的初始化如下：从文件中读入和内嵌的参数



运行结果如下：







## 单步调试inference\_cnn\_loadweight.py文件

# 三、模型修改

## 1.修改数据文件

### (1) 模型参数分析

npz结尾的数据集是压缩文件，包含多个数组，查看子文件夹目录：



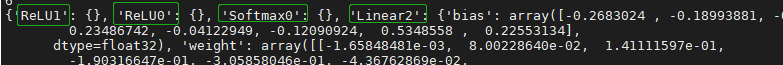


读取对应子文件夹内容：

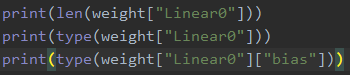


mnist.npz实例中有一个数组 arr\_0，数组内包含6个数组项：

这里weight是字典 包含6个数据项（也是字典）



数组项的类型是字典，字典值为数组：





还有未显示的linear1, linear0 每个小字典中还包括bias、weight两部分

制作自己的npz模型需要将参数转换成字典：

键：卷积层改为Conv2dx 全连接改为Linearx x为所在层数

值：值是两个字典：键：weight/bias 值：数组

### (2) 使用alexnet.pth模型文件实例进行npz参数文件的制作

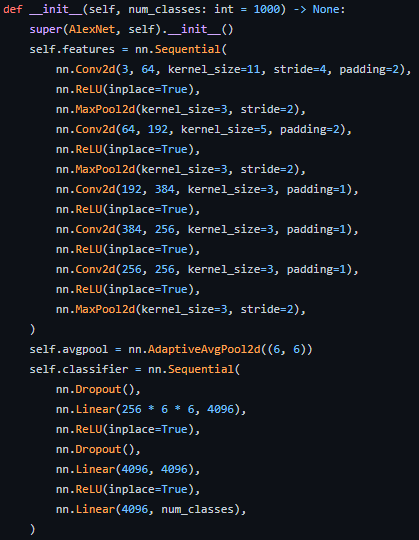
模型下载地址：



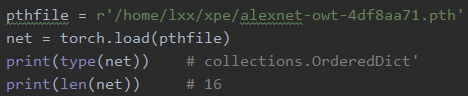
该模型只包含网络参数，不包含网络结构信息，需要预训练的时候加载模型

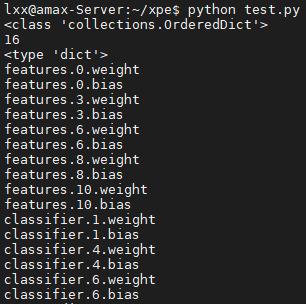
网络结构：五层卷积+三层全连接

具体参数如下：

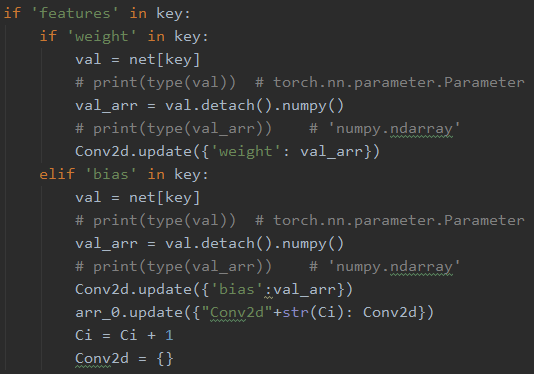


模型参数加载：



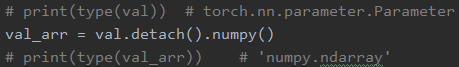


取出的数据为有序字典类型，按键取值，修改对应层名称，将每层内的权重和偏置分开存放。



数据类型修改为对应数据类型：

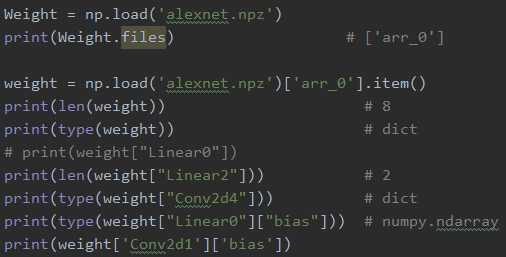


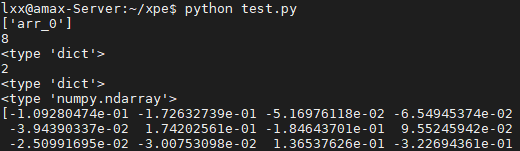


将重组的数据转存为npz格式文件

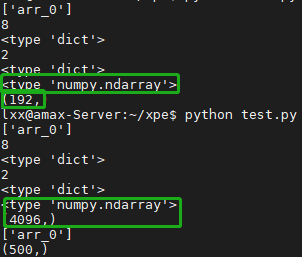


对结果进行验证：





数组维度对比

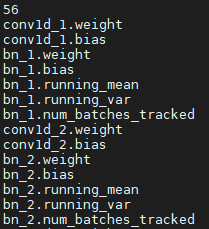


参数文件制作完成。

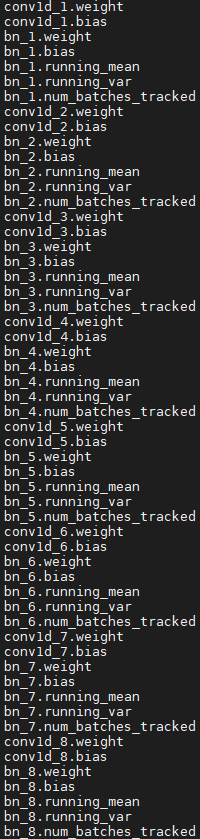
### (3) 使用best-270.pth制作npz文件

模型参数加载

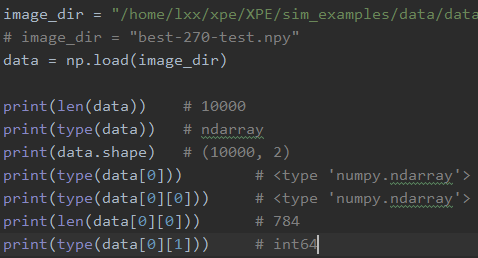
8层卷积 最后一层全局池化



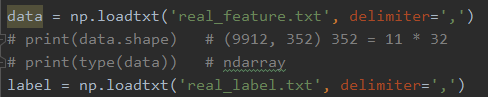
没有全连接层 bn层保存参数：



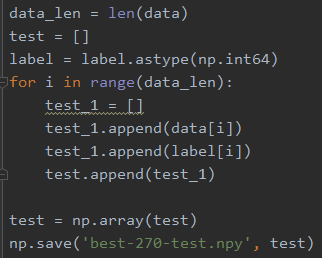
### (4) 把测试图像和标签按指定维度存为npy格式（自己制作）



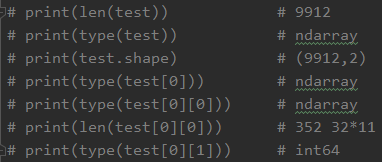
测试demo的参数维度核类型



从文本读入图像数据和标签



对齐数据格式，保存npy文件



验证测试结果