**User Manual of MLP simulator (+NeuroSim) V1.0**

Developers: Pai-Yu Chen and Xiaochen Peng

PI: Prof. Shimeng Yu, Arizona State University

### Index

[1. Introduction 1](#_Toc520102579)

[2. System Requirements (Linux) 2](#_Toc520102580)

[3. Installation and Usage (Linux) 2](#_Toc520102581)

[4. Device Level: Extraction of Synaptic Device Characteristics (设备级别：提取突触设备特征) 3](#_Toc520102582)

[4.1 Non-ideal Analog eNVM Device Properties (非理想的模拟eNVM设备属性) 3](#_Toc520102583)

[4.2 Fitting by MATLAB script (nonlinear\_fit.m) 5](#_Toc520102584)

[5. Circuit Level: Synaptic Cores and Array Architectures 7](#_Toc520102585)

[5.1 Crossbar Array Architecture 7](#_Toc520102586)

[5.2 Pseudo-crossbar Array Architecture 9](#_Toc520102587)

[5.3 Array Peripheral Circuits 10](#_Toc520102588)

[6. Algorithm Level: Multilayer Perceptron (MLP) Neural Network Architecture 12](#_Toc520102589)

[7. How to run MLP simulator (+NeuroSim) 14](#_Toc520102590)

# 1. Introduction

MLP模拟器（+ NeuroSim）是用C ++开发的，用于在基于新兴非易失性存储器（eNVM）阵列架构的2层多层感知器（MLP）神经网络中使用MNIST手写数据集模拟在线学习/离线分类场景。该模拟器中的eNVM是指电阻式存储器设备的一个特殊子集，可以通过电压激励将电导调整为多级状态。 NeuroSim是一种电路级宏模型，用于根据电路级性能指标（如芯片面积，延迟，动态能量和泄漏功率）对神经启发架构进行基准测试。如果没有NeuroSim，MLP模拟器可以被视为一个独立的功能模拟器，能够在学习过程中评估学习准确性和电路级性能（但仅限于突触阵列）。使用NeuroSim，MLP模拟器（+ NeuroSim）成为一个集成框架，具有从设备级（晶体管和模拟eNVM设备属性）到电路级（具有外围电路模块的模拟eNVM阵列架构）然后到算法级（神经元）的分层组织网络拓扑），在学习运行时实现基于跟踪和周期精确的学习准确度评估以及电路级性能指标

在此发布的1.0版本中，目标用户是希望使用他/她自己的模拟突触设备数据快速估计系统级性能的设备工程师。期望用户具有随时可用的重量更新特性（电导与＃脉冲）。可以使用提供的MATLAB脚本提取设备级参数，例如级别数，权重更新非线性，设备到设备变化以及周期到周期变化。在电路级，可以使用一些设计选项，例如阵列架构是真正的交叉开关或带有1T1R的伪交叉开关。在算法级别，提供简单的2层MLP神经网络用于评估，因此用户仅可以使用有限的选项来修改，例如每层的大小和权重矩阵的大小。

# 2. System Requirements (Linux)

The tool is expected to run in Linux with required system dependencies installed. These include GCC, GNU make, GNU C libraries (glibc). We have tested the compatibility of the tool with a few different Linux environments, such as (1) Red Hat 5.11 (Tikanga), gcc v4.7.2, glibc 2.5, (2) Red Hat 7.3 (Maipo), gcc v4.8.5, glibc v2.1.7, (3) Ubuntu 16.04, gcc v5.4.0, glibc v2.23, and they are all workable.

※ The tool may not run correctly (stuck forever) if compiled with gcc 4.5 or below, because some C++11 features are not well supported.

# 3. Installation and Usage (Linux)

**Step 1:** Get the tool from GitHub

git clone https://github.com/neurosim/MLP\_NeuroSim.git

**Step 2:** Extract **MNIST\_data.zip** to it’s current directory

unzip MNIST\_data.zip

**Step 3:** Compile the codes

make

Summary of the useful commands is provided below. It is recommended to execute these commands under the tool’s directory.

|  |  |
| --- | --- |
| **Command** | **Description** |
| make | Compile the codes and build the “main” program |
| make clean | Clean up the directory by removing the object files and the “main” executable |
| ./main | Run simulation (after make) |
| make run | Run simulation (after make), and the results will be saved to a log file (filename appended with the current time info). This command does not work if “stdbuf” is not found. |

※ The simulation uses OpenMP for multithreading, and it will use up all the CPU cores by default.

# 4. Device Level: Extraction of Synaptic Device Characteristics (设备级别：提取突触设备特征)

## 4.1 Non-ideal Analog eNVM Device Properties (非理想的模拟eNVM设备属性)

As shown in Fig. 1, the framework considers the following non-ideal analog eNVM device properties:

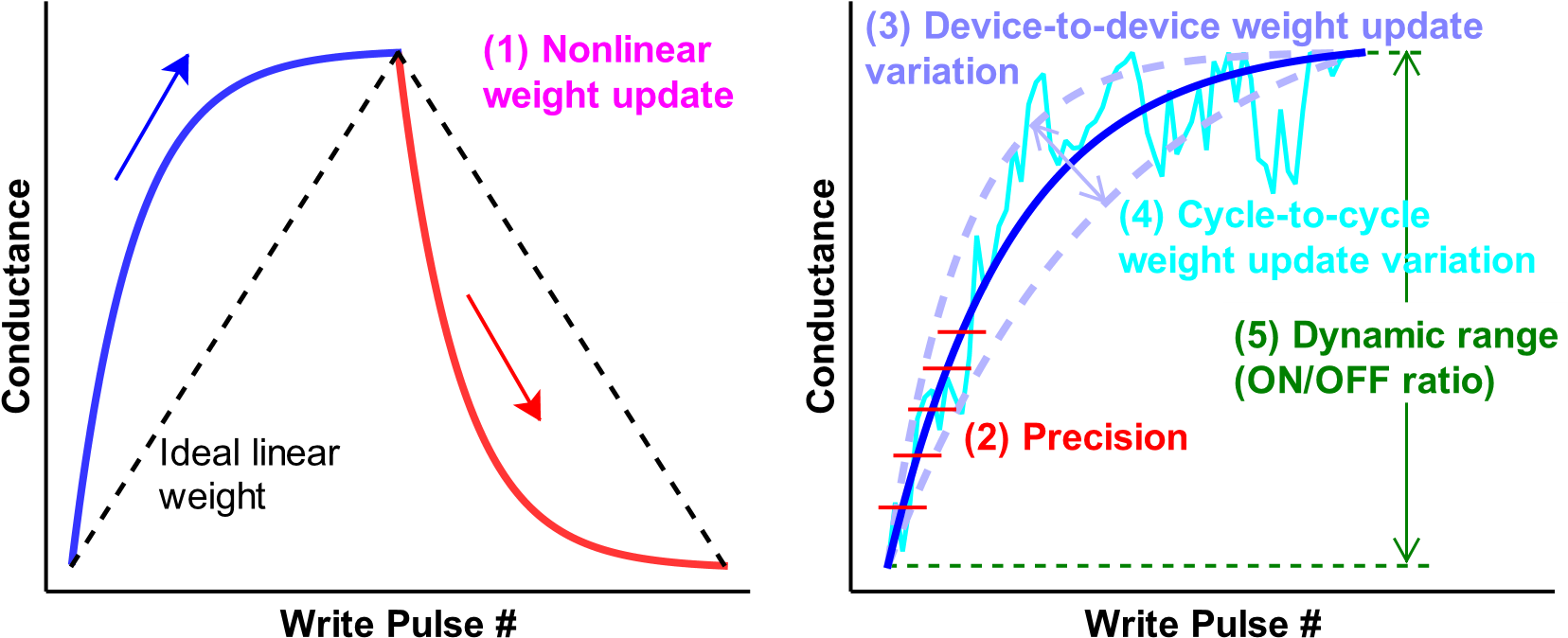
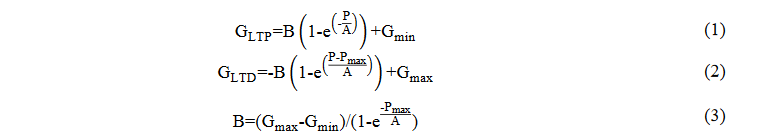


Fig. 1 Summary of non-ideal analog eNVM device properties.

##### 1) Nonlinear weight update

Ideally, the amount of weight increase (or long-term potentiation, LTP) and weight decrease (or long-term depression, LTD) should be linearly proportional to the number of write pulses. However, the realistic devices reported in literature do not follow such ideal trajectory, where the conductance typically changes rapidly at the beginning stages of LTP and LTD and then gradually saturates. We have built a device behavioral model to capture nonlinear weight update behavior, where the conductance change with number of pulses (*P*) is described with the following equations:

理想地，权值增加（或长期增强，LTP）和权值减少（或长期抑制，LTD）的量应与写脉冲的数量成线性比例。 然而，文献中报道的现实装置并未遵循这样的理想轨迹，其中电导通常在LTP和LTD的开始阶段快速变化，然后逐渐饱和。 我们已经建立了一个设备行为模型来捕获非线性权重更新行为，其中电导随脉冲数（P）的变化用以下公式描述：



*G*LTP and *G*LTD are the conductance for LTP and LTD, respectively. *G*max, *G*min, and *P*max are directly extracted from the experimental data, which represents the maximum conductance, minimum conductance and the maximum pulse number required to switch the device between the minimum and maximum conductance states. *A* is the parameter that controls the nonlinear behavior of weight update. *A* can be positive (blue) or negative (red). In Fig. 1, the *A* of LTP and LTD has the same magnitude but different signs. *B* is simply a function of *A* that fits the functions within the range of *G*max, *G*min, and *P*max. All these parameters can be different in LTP and LTD in the fitting by the MATLAB script. However, for simplicity, the simulator currently uses the smaller value of *G*max and the larger value of *G*min in LTP and LTD.

GLTP和GLTD分别是LTP和LTD的电导率。 从实验数据中直接提取Gmax，Gmin和Pmax，其表示在最小和最大电导状态之间切换器件所需的最大电导，最小电导和最大脉冲数。 A是控制权重更新的非线性行为的参数。 A可以是正（蓝色）或负（红色）。 在图1中，LTP和LTD的A具有相同的幅度但不同的符号。 B只是A的函数，适合Gmax，Gmin和Pmax范围内的函数。 在MATLAB脚本的拟合中，所有这些参数在LTP和LTD中可以是不同的。 然而，为简单起见，模拟器目前在LTP和LTD中使用较小的Gmax值和较大的Gmin值。

Using Eq. (1)-(3), a set of nonlinear weight increase (blue) and weight decrease (red) behavior can be obtained by adjusting *A* as shown in Fig. 2, where each nonlinear curve is labeled with a nonlinearity value from +6 to -6. It can be proved that Eq. (1) and (2) are equivalent with a different sign of *A*, thus we will just use Eq. (1) to calculate both nonlinear LTP and LTD weight update. Different than Fig. 1, all LTD curves are mirrored and shifted horizontally to make sure the curve starting from the pulse number 0 for simpler formulization.

使用公式（1） - （3），如图2所示调整A，可以得到一组非线性权重增加（蓝色）和权重减少（红色）集合，其中每个非线性曲线用+6到-6的非线性值标记。 可以证明，公式（1）和（2）等价于A的不同符号，因此我们将使用公式（1）计算非线性LTP和LTD重量更新。 与图1不同，所有LTD曲线都被镜像并水平移动，以确保曲线从脉冲数0开始，以便更简单地进行公式化。

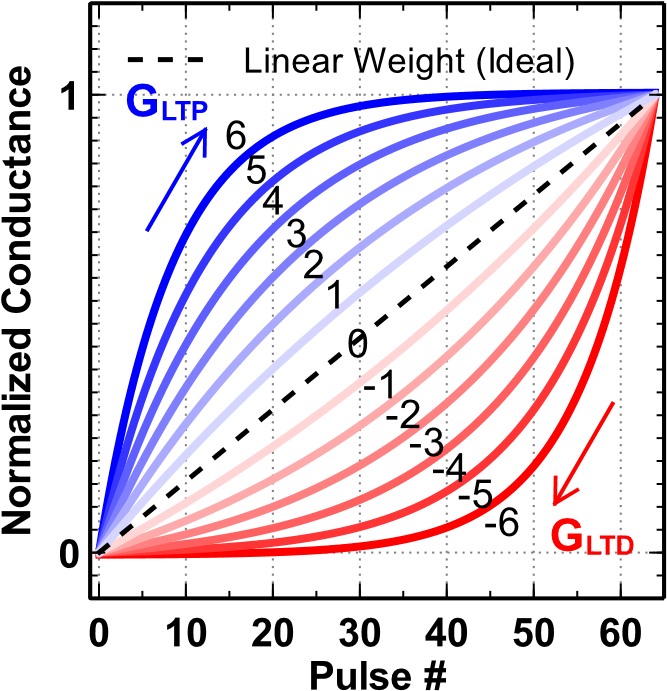


Fig. 2 Analog eNVM device behavioral model of the nonlinear weight update with the nonlinearity labeled from -6 to 6. （模拟eNVM设备非线性权重更新的行为模型，非线性标记为-6到6）

##### 2) Limited precision

The precision of an analog eNVM device is determined by the number of conductance states it has, which is *P*max in Eq. (1)-(3).

模拟eNVM设备的精度由它具有的电导状态的数量决定，即公式（1）-（3）中的Pmax。

##### 3) Device-to-device weight update variation

The effect of device-to-device weight update variation can be analyzed by introducing the variation into the nonlinearity baseline. This variation is defined as the nonlinearity baseline’s standard deviation (σ) respect to 1 step of the 6 steps in Fig. 2.

可以通过将变化引入非线性基线来分析设备到设备权值更新变化的影响。 该变化被定义为相对于图2中的6个步骤的1个步骤的非线性基线的标准偏差（σ）。

##### 4) Cycle-to-cycle weight update variation

The Cycle-to-cycle weight update variation is referred to as the variation in conductance change at every programming pulse. This variation (σ) is expressed in terms of the percentage of entire conductance range.

周期到周期权值更新变化被称为每个编程脉冲处的电导变化的变化。 该变化（σ）以整个电导范围的百分比表示。

**5) Dynamic range (ON/OFF ratio)**

Ideally, the weight values are represented by a normalized conductance of analog eNVM devices with the range from 0 to 1. However, the minimum conductance can be regarded as 0 only when the ratio between the maximum and minimum conductance (ON/OFF ratio) approaches infinity. With limited ON/OFF ratio, the cells with weight=0 still have leakage.

理想情况下，权重值由模拟eNVM设备的归一化电导表示，范围从0到1.但是，只有当最大和最小电导（ON / OFF比率）之间的比率接近时，最小电导才能被视为0 无穷。 在ON / OFF比有限的情况下，重量= 0的单元仍有泄漏。

## 4.2 Fitting by MATLAB script (nonlinear\_fit.m)

In this section, we will fit the experimental weight update data and extract the device parameters that will be used in the simulator. We have developed a MATLAB script **nonlinear\_fit.m** to do such a task, where it has been set up for fitting Ag:a-Si devices in the following reference as an example.

S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, “Nanoscale memristor device as synapse in neuromorphic systems,” *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, 2010.

Before the fitting, the user has to make sure the experimental weight update data are pre-processed in a format that is similar to Fig. 2. Namely, the LTD data should be mirrored horizontally and both LTP and LTD data should start from the pulse number 0 so that the data can be fit by Eq. (1)-(3). The user can look at the pre-processed data of Ag:a-Si devices as an example in the MATLAB script.

The shape of nonlinear weight update curves can look very different with the same *A* but different *P*max and *G*min, because *A* has to be scaled according to different *P*max. First, we normalize *P*max to be 1 by default definition, then we can tune the normalized *A* and the cycle-to-cycle weight update variation for both LTP and LTD in the generated **Figure 1** (normalized conductance vs. normalized number of pulses) to find the best fit, as shown in Fig. 3. A good procedure is that the user first finds out a reasonable normalized *A* for LTP and LTD curves without variation (by setting the variation in LTP and LTD to zero), and then try to fit the LTP and LTD data with good variation values and pseudorandom seeds (for example, **rng(103)** and **rng(898)** in script). In the script, the *A* values are defined as **A\_LTP** and **A\_LTD**, and *P*max is defined as **xf** which is set to be 1. These parameters are shown in Fig. 4. It should be noted that the device-to-device weight update variation is not considered in this script.

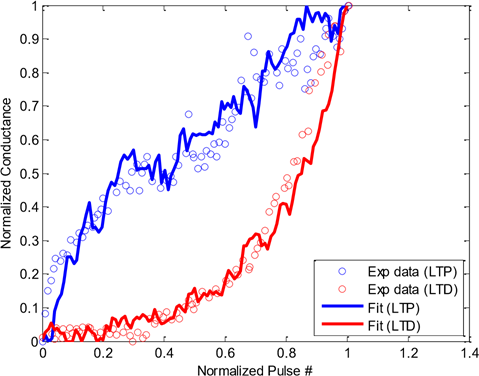


Fig. 3 Fitting of Ag:a-Si weight update data with normalized *A* in the plot of normalized conductance vs. normalized number of pulses.

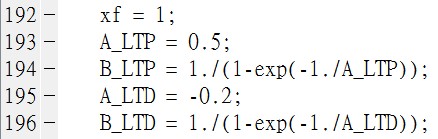


Fig. 4 Code snippet of parameters in **nonlinear\_fit.m**

After the fitting is done, the user can look up the magnitude of the normalized *A* value to figure out its corresponding nonlinearity label. We provide the information of one-to-one mapping of the nonlinearity label values to the normalized *A* values in the file **Nonlinearity-NormA.htm**. The nonlinearity label value is from 0 to 9 with a step of 0.01, which is precise enough. If the normalized *A* value is negative, the nonlinearity label value will also simply be negative. In the example of Ag:a-Si, the fitted normalized *A* values are 0.5 and -0.2, and we can figure out their corresponding nonlinearities are 2.40 and -4.88 for LTP and LTD, respectively, as shown in Table 1. In the last section, *MLP simulator* (*+NeuroSim*) will take these two values as eNVM cell parameters.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nonlinearity | Norm. A | Nonlinearity | Norm. A | Nonlinearity | Norm. A | Nonlinearity | Norm. A | Nonlinearity | Norm. A |
| 2.31 | 0.5207 | 2.38 | 0.5038 | 2.45 | 0.4879 | 4.84 | 0.2030 | 4.91 | 0.1983 |
| 2.32 | 0.5183 | 2.39 | 0.5015 | 2.46 | 0.4856 | 4.85 | 0.2023 | 4.92 | 0.1976 |
| 2.33 | 0.5158 | 2.40 | 0.4992 | … | … | 4.86 | 0.2016 | 4.93 | 0.1970 |
| 2.34 | 0.5134 | 2.41 | 0.4969 | … | … | 4.87 | 0.2010 | 4.94 | 0.1963 |
| 2.35 | 0.5110 | 2.42 | 0.4946 | … | … | 4.88 | 0.2003 | 4.95 | 0.1957 |
| 2.36 | 0.5086 | 2.43 | 0.4932 | 4.82 | 0.2044 | 4.89 | 0.1996 | 4.96 | 0.1950 |
| 2.37 | 0.5062 | 2.44 | 0.4901 | 4.83 | 0.2037 | 4.90 | 0.1990 | 4.97 | 0.1944 |

Table 1 Snippet of **Nonlinearity-NormA.htm**

# 5. Circuit Level: Synaptic Cores and Array Architectures

In this framework, we consider two synaptic cores of 2-layer MLP in the evaluation for circuit-level metrics such as area, latency, energy, etc. A synaptic core is a computation unit that is specifically designed for weighted sum and weight update. It consists of the synaptic array and array periphery. In the simulator, a synaptic core can be instantiated from **SubArray** class in **SubArray.cpp**. In this released version, there are two available design options for the synaptic core. One is based on the crossbar array architecture, and other one is based on the pseudo-crossbar array architecture, as shown in Fig. 5. The details of both architectures and their peripheral circuits are introduced below.

在此框架中，我们在评估电路级指标（如面积，延迟，能量等）时考虑2层MLP的两个突触核心。突触核心是专为加权和和权重更新而设计的计算单元。 它由突触阵列和阵列外围组成。 在模拟器中，可以从SubArray.cpp中的SubArray类实例化突触核心。 在此发布的版本中，synaptic核心有两种可用的设计选项。 一种基于交叉开关阵列架构，另一种基于伪交叉开关阵列架构，如图5所示。下面介绍两种架构及其外围电路的细节。

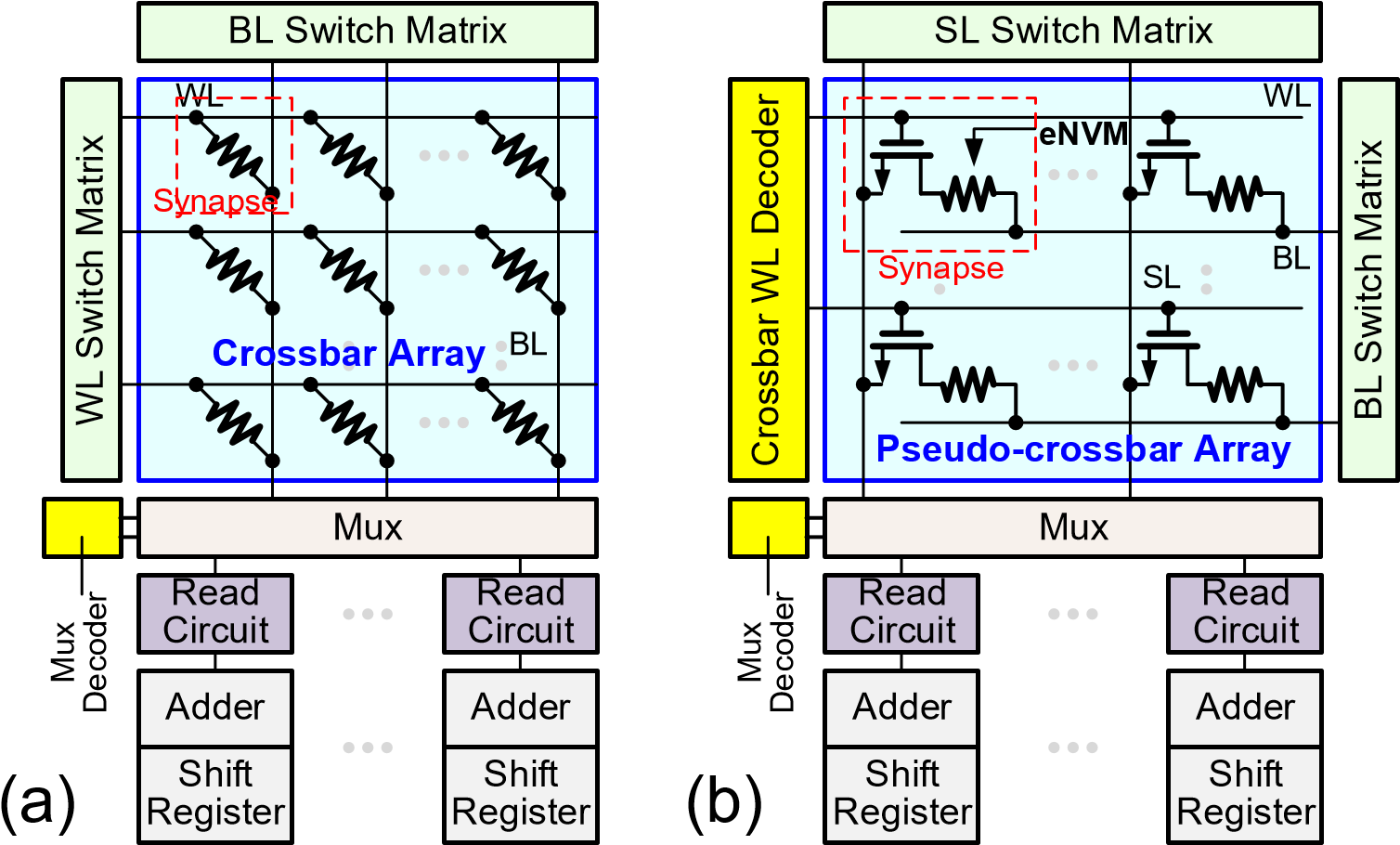


Fig. 5 Synaptic cores based on (a) crossbar and (b) pseudo-crossbar array architectures

（突触核心基于（a）交叉开关和（b）伪交叉开关阵列架构）

## 5.1 Crossbar Array Architecture

The crossbar array structure has the most compact and simplest array structure for eNVM devices to form a weight matrix, where each eNVM device is located at the cross point of a word line (WL) and a bit line (BL). The crossbar array structure can achieve a high integration density of 4F2/cell (F is the lithography feature size). If the input vector is encoded by read voltage signals, the weighted sum operation (matrixvector multiplication) can be performed in a parallel fashion with the crossbar array. However, as there is no isolation between cells, it is necessary to apply some voltage (smaller than the programming voltage, e.g. V/2) at all the unselected rows and columns to prevent the write disturbance on unselected cells during weight update. The voltage bias schemes for weight update is shown in Fig. 6. In addition, a two-terminal threshold switching selector device is desired to minimize the write disturbance and sneak path problem. In our model, a simple I-V nonlinearity (**NL**) in **Cell.cpp** is used to define the effect of selector or built-in self-selection. As the weight increase and decrease need different programming voltage polarities, the weight update process requires 2 steps with different voltage bias schemes. In weight update, the selected cells will be on the same row, and programming pulses or biases (if no update) are provided from the BL, allowing the selected cells to be tuned differently in parallel. To perform weight update for the entire array, a row-by-row operation is necessary. Ideally, the entire row is selected at a time to ensure the maximum parallelism. In the simulator, the crossbar array architecture can be designated by setting **cmosAccess=false** in **RealDevice** class of **Cell.cpp**.

交叉开关阵列结构具有最紧凑和最简单的阵列结构，用于eNVM设备形成权重矩阵，其中每个eNVM设备位于字线（WL）和位线（BL）的交叉点。横杆阵列结构可以实现4F2 /单元的高集成密度（F是光刻特征尺寸）。如果输入矢量由读取电压信号编码，则加权和操作（矩阵向量乘法）可以与交叉开关阵列以并行方式执行。然而，由于单元之间没有隔离，因此必须在所有未选择的行和列上施加一些电压（小于编程电压，例如V / 2），以防止在重量更新期间对未选择的单元的写入干扰。用于重量更新的电压偏置方案如图6所示。此外，需要双端阈值开关选择器装置以最小化写入干扰和潜行路径问题。在我们的模型中，Cell.cpp中的简单I-V非线性（NL）用于定义选择器或内置自选的效果。由于重量增加和减少需要不同的编程电压极性，因此重量更新过程需要具有不同电压偏置方案的两个步骤。在权重更新中，所选单元将在同一行上，并且从BL提供编程脉冲或偏置（如果没有更新），允许所选单元被并行地不同地调谐。要对整个阵列执行权重更新，需要逐行操作。理想情况下，一次选择整行以确保最大的并行度。在模拟器中，可以通过在Cell.cpp的RealDevice类中设置cmosAccess = false来指定交叉开关阵列架构。

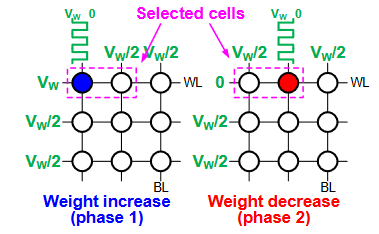


Fig. 6 Voltage bias scheme in the write operation of crossbar array. Two separate phases for weight increase and decrease are required. In this example, the left cell of the selected cells will be updated in phase 1, while the right one will be updated in phase 2. （交叉阵列写操作中的电压偏置方案。 需要两个单独的阶段来增加和减少重量。 在此示例中，所选单元格的左侧单元格将在阶段1中更新，而右侧单元格将在阶段2中更新）

## 5.2 Pseudo-crossbar Array Architecture

Another common design solution to the write disturbance and sneak path problem is to add a cell selection transistor in series with the eNVM device, forming the one-transistor one-resistor (1T1R) array architecture, as shown in Fig. 7(a). The WL controls the gate of the transistor, which can be viewed as a switch for the cell. The source line (SL) connects to the source of the transistor. The eNVM cell’s top electrode connects to the BL, while its bottom electrode connects to the drain of the transistor through a contact via. In such case, the cell area of 1T1R array is then determined by the transistor size, which is typically >6F2 depending on the maximum current required to be delivered into the eNVM cell. Larger current needs larger transistor gate width/length (W/L). However, conventional 1T1R array is not able to perform the parallel weighted sum operation. To solve this problem, we modify the conventional 1T1R array by rotating the BLs by 90o, which is known as the pseudo-crossbar array architecture, as shown in Fig. 7(b). In weighted sum operation, all the transistors will be transparent when all WLs are turned on. Thus, the input vector voltages are provided to the BLs, and the weighted sum currents are read out through SLs in parallel. The weight update operation in pseudo-crossbar array is similar to that in crossbar array, as shown in Fig. 8. As the unselected WLs can turn off the transistors on unselected rows, no voltage bias is required for these unselected BLs thus pseudo-crossbar array can have save a lot of weight update energy compared to the crossbar array. In the simulator, the pseudo-crossbar array architecture can be designated by setting **cmosAccess=true** in **RealDevice** class of **Cell.cpp**.

写入干扰和潜行路径问题的另一种常见设计解决方案是添加与eNVM器件串联的单元选择晶体管，形成单晶体管单电阻（1T1R）阵列架构，如图7（a）所示。 WL控制晶体管的栅极，可以将其视为电池的开关。源极线（SL）连接到晶体管的源极。 eNVM单元的顶部电极连接到BL，而其底部电极通过接触通孔连接到晶体管的漏极。在这种情况下，1T1R阵列的单元面积然后由晶体管尺寸确定，晶体管尺寸通常> 6F2，这取决于输送到eNVM单元所需的最大电流。较大的电流需要较大的晶体管栅极宽度/长度（W / L）。然而，传统的1T1R阵列不能执行并行加权和运算。为了解决这个问题，我们通过将BLs旋转90°来修改传统的1T1R阵列，这被称为伪交叉阵列架构，如图7（b）所示。在加权求和操作中，当所有WL导通时，所有晶体管将是透明的。因此，输入矢量电压被提供给BL，并且加权的和电流通过SL并行读出。伪交叉开关阵列中的权重更新操作类似于交叉开关阵列中的权重更新操作，如图8所示。由于未选择的WL可以关闭未选择的行上的晶体管，因此这些未选择的BL不需要电压偏置，因此伪交叉开关与交叉开关阵列相比，阵列可以节省大量的重量更新能量。在模拟器中，可以通过在Cell.cpp的RealDevice类中设置cmosAccess = true来指定伪交叉开关阵列架构。

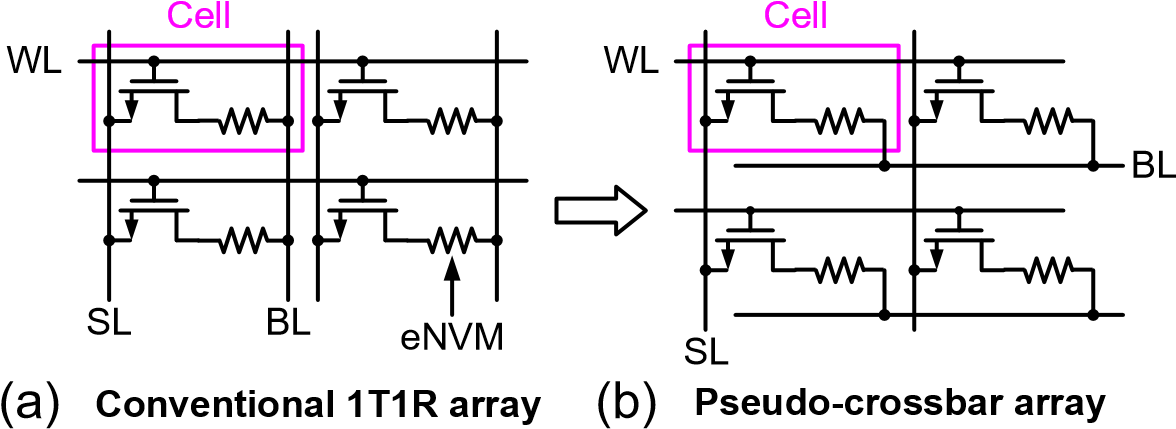


Fig. 7 Transformation from (a) conventional 1T1R array to (b) pseudo-crossbar array by 90o rotation of BL to enable weighted sum operation.

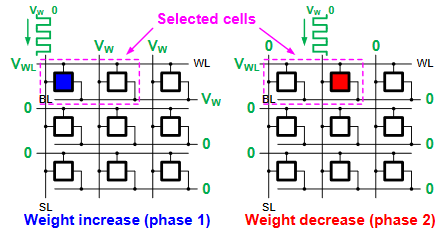


Fig. 8 Voltage bias scheme in the write operation of pseudo-crossbar array. Two separate phases for weight increase and decrease are required. In this example, the left cell of the selected cells will be updated in phase 1, while the right one will be updated in phase 2. （伪交叉阵列写操作中的电压偏置方案。 需要两个单独的阶段来增加和减少重量。 在此示例中，所选单元格的左侧单元格将在阶段1中更新，而右侧单元格将在阶段2中更新。）

## 5.3 Array Peripheral Circuits （阵列外围电路）

The periphery circuit modules used in both eNVM crossbar and pseudo-crossbar synaptic cores in Fig. 5 are described below:

##### 1) Switch matrix

Switch matrixes are used for fully parallel voltage input to the array rows or columns. Fig. 9(a) shows the BL switch matrix for example. It consists of transmission gates that are connected to all the BLs, with control signals (B1 to Bn) of the transmission gates stored in the registers (not shown here). In the weighted sum operation, the input vector signal is loaded to B1 to Bn, which decide the BLs to be connected to either the read voltage or ground. In this way, the read voltage that is applied at the input of transmission gates can pass to the BLs and the weighted sums are read out through SLs in parallel. If the input vector is not 1 bit, it should be encoded using multiple clock cycles, as shown in Fig. 9(b). The reason why we do not use analog voltage to represent the input vector precision is the I-V nonlinearity of eNVM cell, which will cause the weighted sum distortion or inaccuracy. In the simulator, all the switch matrixes (**slSwitchMatrix**, **blSwitchMatrix** and **wlSwitchMatrix**) are instantiated from **SwitchMatrix** class in **SwitchMatrix.cpp**.

开关矩阵用于输入到阵列行或列的完全并联电压。图9（a）示出了例如BL开关矩阵。它由连接到所有BL的传输门组成，传输门的控制信号（B1到Bn）存储在寄存器（这里未示出）中。在加权求和操作中，输入矢量信号被加载到B1到Bn，这决定了BL连接到读取电压或地。以这种方式，在传输门的输入处施加的读取电压可以传递到BL，并且通过SL并行地读出加权的和。如果输入向量不是1位，则应使用多个时钟周期对其进行编码，如图9（b）所示。我们不使用模拟电压来表示输入矢量精度的原因是eNVM单元的I-V非线性，这将导致加权和失真或不准确。在模拟器中，所有开关矩阵（slSwitchMatrix，blSwitchMatrix和wlSwitchMatrix）都是从SwitchMatrix.cpp中的SwitchMatrix类实例化的。

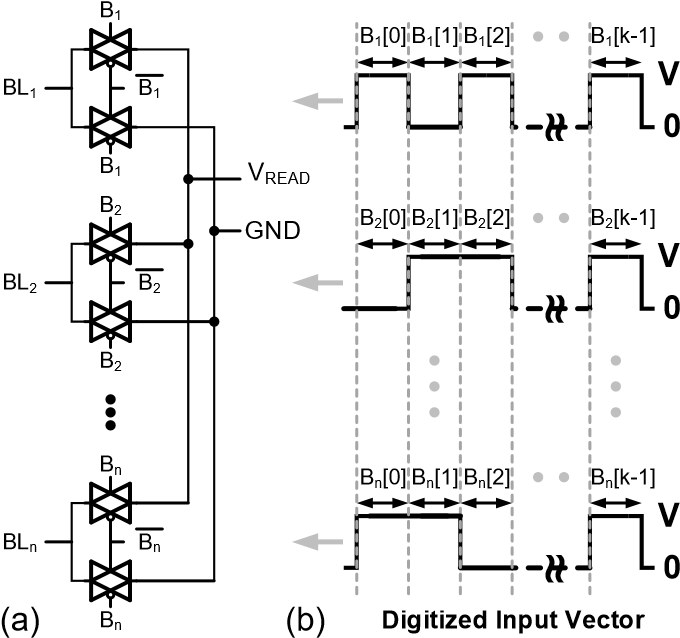


Fig. 9 (a) Transmission gates of the BL switch matrix in the weighted sum operation. A vector of control signals (B1 to Bn) from the registers (not shown here) decide the BLs to be connected to either a voltage source or ground. (b) Control signals in a bit stream to represent the precision of the input vector.

##### 2) Crossbar WL decoder

The crossbar WL decoder is modified from the traditional WL decoder. It has an additional feature to activate all the WLs for making all the transistors transparent for weighted sum. The crossbar WL decoder is constructed by attaching the follower circuits to every output row of the traditional decoder, as shown in Fig. 10. If *ALLOPEN*=1, the crossbar WL decoder will activate all the WLs no matter what input address is given, otherwise it will function as a traditional WL decoder. In the simulator, the crossbar WL decoder contains a traditional decoder (**wlDecoder**) instantiated from **RowDecoder** class in **RowDecoder.cpp** and a collection of follower circuits (**wlDecoderOutput**) instantiated from **WLDecoderOutput** class in **WLDecoderOutput.cpp**.

从传统的WL解码器修改纵横WL解码器。 它还具有激活所有WL的附加功能，使所有晶体管对加权和都透明。 通过将跟随器电路连接到传统解码器的每个输出行来构造纵横WL解码器，如图10所示。如果ALLOPEN = 1，则无论给出什么输入地址，交叉开关WL解码器将激活所有WL，否则 它将作为传统的WL解码器。 在模拟器中，交叉开关WL解码器包含从RowDecoder.cpp中的RowDecoder类实例化的传统解码器（wlDecoder）和从WLDecoderOutput.cpp中的WLDecoderOutput类实例化的跟随器电路（wlDecoderOutput）的集合。

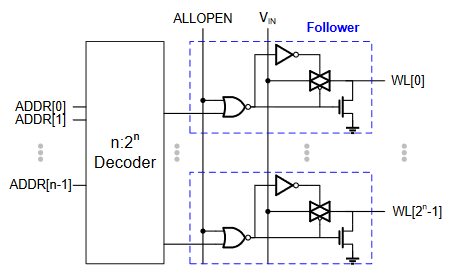


Fig. 10 Circuit diagram of the crossbar WL decoder. Follower circuit is attached to every row of the decoder to enable activation of all WLs when *ALLOPEN*=1.

##### 3) Multiplexer (Mux) and Mux decoder

The Multiplexer (Mux) is used for sharing the read periphery circuits among synaptic array columns, because the array cell size is much smaller than the size of read periphery circuits and it will not be areaefficient to put all the read periphery circuits underneath the array. However, sharing the read periphery circuits among synaptic array columns inevitably increases the latency of weighted sum as time multiplexing is needed, which is controlled by the Mux decoder. In the simulator, the Mux (**mux**) is instantiated from **Mux** class in **Mux.cpp** and the Mux decoder (**muxDecoder**) is instantiated from **RowDecoder** class in **RowDecoder.cpp**.

多路复用器（Mux）用于在突触阵列列之间共享读取外围电路，因为阵列单元尺寸远小于读取外围电路的尺寸，并且将所有读取外围电路放置在阵列下方将不是区域效率。 然而，在需要时间复用时，在突触阵列列之间共享读取外围电路不可避免地增加加权和的等待时间，其由Mux解码器控制。 在模拟器中，Mux（mux）从Mux.cpp中的Mux类实例化，Mux解码器（muxDecoder）从RowDecoder.cpp中的RowDecoder类实例化。

##### 4) Analog-to-digital read circuit

To convert these analog weighted sum currents to digital outputs, we use the read circuit in the following reference to employ the principle of the integrate-and-fire neuron model, as shown in Fig. 11(a). The read circuit integrates the weighted sum current on the finite capacitance of the array column. Once the voltage charges up above a certain threshold, the read circuit fires an output pulse and the capacitance is discharged back. The simulated waveform of integrated input voltage and the digital output spikes of the read circuit is shown in Fig. 11(b). The number of output spikes is proportional to the weight sum current. The precision required for this analog-to-digital conversion (ADC) determines the pulse width in each bit of the input vector. In the simulator, a collection of read circuits (**readCircuit**) is instantiated from **ReadCircuit** class in **ReadCircuit.cpp**.

为了将这些模拟加权和电流转换为数字输出，我们使用以下参考中的读取电路来采用积分和激发神经元模型的原理，如图11（a）所示。 读取电路在阵列列的有限电容上积分加权和电流。 一旦电压充电超过某个阈值，读取电路就会触发输出脉冲并释放电容。 集成输入电压的模拟波形和读电路的数字输出尖峰如图11（b）所示。 输出尖峰的数量与重量和电流成比例。 该模数转换（ADC）所需的精度决定了输入矢量每一位的脉冲宽度。 在模拟器中，读取电路的集合（readCircuit）从ReadCircuit.cpp中的ReadCircuit类实例化。

D. Kadetotad, Z. Xu, A. Mohanty, P.-Y. Chen, B. Lin, J. Ye, S. Vrudhula, S. Yu, Y. Cao, J.-S. Seo, “Parallel architecture with resistive crosspoint array for dictionary learning acceleration,” *IEEE J. Emerg. Sel. Topics Circuits Syst. (JETCAS)*, vol. 5, no. 2, pp. 194-204, 2015.



Fig. 11 (a) Design of a read circuit that employs the principle of the integrate-and-fire neuron model. (b) Simulated waveform of integrated input voltage and the digital output spikes of the read circuit.

##### 5) Adder and shift register

The adder and shift register pair at the bottom of synaptic core performs shift and add of the weighted sum result at each input vector bit cycle (B1 to Bn in Fig. 9(b)) to get the final weighted sum. The bit-width of the adder and shift register needs to be further extended depending on the precision of input vector. If the values in the input vector are only 1 bit, then the adder and shift register pair is not required. In the simulator, a collection of the adder and shift register pairs (**ShiftAdd**) is instantiated from **ShiftAdd** class in **ShiftAdd.cpp**, where **ShiftAdd** further contains a collection of adders (**adder**) instantiated from **Adder** class in **Adder.cpp** and a collection of registers (**dff**) instantiated from **DFF** class in **DFF.cpp**.

突触核心底部的加法器和移位寄存器对在每个输入矢量比特周期（图9（b）中的B1到Bn）执行加权和加法的移位和相加，以得到最终的加权和。 加法器和移位寄存器的位宽需要根据输入矢量的精度进一步扩展。 如果输入向量中的值仅为1位，则不需要加法器和移位寄存器对。 在模拟器中，加法器和移位寄存器对（ShiftAdd）的集合从ShiftAdd.cpp中的ShiftAdd类实例化，其中ShiftAdd还包含从Adder.cpp中的Adder类实例化的加法器（加法器）的集合以及寄存器的集合 （dff）在DFF.cpp中从DFF类实例化。

# 6. Algorithm Level: Multilayer Perceptron (MLP) Neural Network Architecture

At the algorithm level, we provide a simple 2-layer multilayer perceptron (MLP) neural network for performance benchmark. As shown in Fig. 12(a), the network consists of an input layer, hidden layer and output layer (the input layer is not included when counting the number of layers). MLP is a fully connected neural network, where each neuron node in one layer connects to every neuron node in the following layer. The connection between two neuron nodes is through a synapse with its strength representing the weight, where *W*IH and *W*HO are the weight matrix between input and hidden layer and between hidden and output layer, respectively. For the input image data, we post-processed the MNIST handwritten digits by cropping the edges of each image (making it 20×20 pixels). For the favor of hardware implementation, we also convert the images to black and white data in the simulator to reduce the design complexity of input encoding. By default, the network topology is 400(input layer)-100(hidden layer)-10(output layer). 400 neurons of input layer correspond to the 20×20 MNIST image, and 10 neurons of output layer correspond to 10 classes of digits. The users could change the network topology as needed. However, a new learning rate may be required to optimize the learning accuracy.

在算法级别，我们提供了一个简单的2层多层感知器（MLP）神经网络，用于性能基准测试。如图12（a）所示，网络由输入层，隐藏层和输出层组成（在计算层数时不包括输入层）。 MLP是完全连接的神经网络，其中一层中的每个神经元节点连接到下一层中的每个神经元节点。两个神经元节点之间的连接是通过突触来实现的，其强度代表权重，其中WIH和WHO分别是输入层和隐藏层之间以及隐藏层和输出层之间的权重矩阵。对于输入图像数据，我们通过裁剪每个图像的边缘（使其为20×20像素）对MNIST手写数字进行后处理。为了有利于硬件实现，我们还在模拟器中将图像转换为黑白数据，以降低输入编码的设计复杂性。默认情况下，网络拓扑为400（输入层）-100（隐藏层）-10（输出层）。输入层的400个神经元对应20×20 MNIST图像，输出层的10个神经元对应10个数字类。用户可以根据需要更改网络拓扑。然而，可能需要新的学习速率来优化学习准确性。

The learning applications that the network can implement in this simulator include the online learning and offline training with classification only. In online learning, the simulator emulates hardware to train the network with images randomly picked from the training dataset (60k images) and classify the testing dataset (10k images). In offline training with classification only, the network is pre-trained by software, and the MLP simulator only emulates hardware to classify the testing dataset. The training data file is **patch60000\_train.txt** and its label file (the correct answers of the training data) is **label60000\_train.txt**. The testing data file is **patch10000\_test.txt** and its label file (the correct answers of the testing data) is **label60000\_train.txt**.

网络可以在该模拟器中实现的学习应用程序包括仅具有分类的在线学习和离线培训。 在线学习中，模拟器模拟硬件以使用从训练数据集中随机挑选的图像（60k图像）训练网络，并对测试数据集（10k图像）进行分类。 在仅具有分类的离线培训中，网络由软件预先训练，并且MLP模拟器仅模拟硬件以对测试数据集进行分类。 训练数据文件是patch60000\_train.txt，其标签文件（训练数据的正确答案）是label60000\_train.txt。 测试数据文件是patch10000\_test.txt，其标签文件（测试数据的正确答案）是label60000\_train.txt。

The training process consists of two key operations, the feed forward (FF) and back propagation (BP). In feed forward, the input data are fed from the input layer and they will travel in a forward direction to the output layer via a series of weighted sum operation and neuron activation function along the way. The feed forward result at the output layer will then be compared with its correct answer (the label) to calculate its prediction error (the deviation). In back propagation, this error is propagated backward from the output layer to adjust the weights of each layer in a way that the prediction error is minimized. In this simulator, we use stochastic gradient decent method to update the weights in the back propagation. Different than the traditional gradient decent, the back propagation is performed after the feed forward of every image rather than that of the entire image dataset. On the other hand, the testing (classification) process only has the feed forward operation to make predictions. The weights in this process will not be changed.

训练过程包括两个关键操作，前馈（FF）和反向传播（BP）。在前馈中，输入数据从输入层馈送，并且它们将通过一系列加权求和操作和沿途的神经元激活功能在前向方向上行进到输出层。然后将输出层的前馈结果与其正确答案（标签）进行比较，以计算其预测误差（偏差）。在反向传播中，该误差从输出层向后传播，以便以预测误差最小化的方式调整每层的权重。在这个模拟器中，我们使用随机梯度体面方法来更新反向传播中的权重。与传统的梯度体面不同，反向传播是在每个图像的前馈而不是整个图像数据集的前馈之后执行的。另一方面，测试（分类）过程仅具有前馈操作以进行预测。此过程中的权重不会更改。

Fig. 12(b) shows a schematic of a neuron node, which encapsulates the principles discussed above. The neuron takes the weighted sum result from its inward synapses and pass it through a 1-bit low-precision activation function. In this way, the offline classification, which is purely feed forward, can be realized in 1 bit. However, the computation on the back propagation of weight update generally needs higher precision to update the small errors, thus a high-precision activation function for the back propagation is still necessary.

Fig. 12(c) shows the circuit block diagram for hardware implementation of this 2-layer MLP neural network. The weighted sum operation is performed using the synaptic cores. However, the weights used in a regular synaptic array can only represent positive values (*W*H=0~1), while the weights in algorithm can be either positive or negative values (*W*A=-1~1). The algorithm’s weighted sum is then expressed as

图12（b）示出了神经元节点的示意图，其封装了上述原理。 神经元从其向内突触中获取加权和结果，并通过1位低精度激活函数。 通过这种方式，可以以1比特实现纯粹前馈的离线分类。 然而，对权重更新的反向传播的计算通常需要更高的精度来更新小误差，因此仍然需要用于反向传播的高精度激活函数。图12（c）示出了该2层MLP神经网络的硬件实现的电路框图。 使用突触核心执行加权求和操作。 然而，常规突触阵列中使用的权重只能表示正值（WH = 0~1），而算法中的权重可以是正值或负值（WA = -1~1）。 然后将算法的加权和表示为

*W*A*V* = (2*W*H - *J*)*V* = 2*W*H*V* - *JV* (4)

where *V* is the input vector and *J* is the matrix of all ones that has the same dimension as *W*A and *W*H. In Eq. (4), *W*H*V* is the weighted sum output from the synaptic core. Therefore, we squeeze *W*A from (-1~1) to the range of *W*H (0~1): i.e. -1 is mapped to 0, 0 is mapped to 0.5, and 1 is mapped to 1. To reconstruct *W*A*V*, we have to perform a two-step read from the array: first, we read out *W*H*V*, and then multiply *W*H*V* by 2 using a 1-bit left-shift, and then subtract *JV* (basically the sum of vector) from *W*H*V* through the adder at the periphery. The MSB (sign bit in 2’s complement notation) of the adder output will be the 1-bit output of the low-precision activation function. It should be noted that we only consider the main sub-circuit modules for the neuron periphery at current stage of this simulator, and the hardware for BP error calculation as well as the detailed control logics will be included in the future release.

其中V是输入向量，J是与WA和WH具有相同维度的所有1的矩阵。 在Eq。 （4），WHV是突触核心的加权和输出。 因此，我们将WA从（-1~1）压缩到WH（0~1）的范围：即-1映射到0,0映射到0.5，1映射到1.要重建WAV，我们有 从数组执行两步读取：首先，我们读出WHV，然后使用1位左移将WHV乘以2，然后从WHV通过加法器减去JV（基本上是矢量之和） 外围。 加法器输出的MSB（2的补码表示符号位）将是低精度激活功能的1位输出。 应该注意的是，我们只考虑该模拟器当前阶段神经元外围的主要子电路模块，BP错误计算的硬件以及详细的控制逻辑将包含在未来版本中。

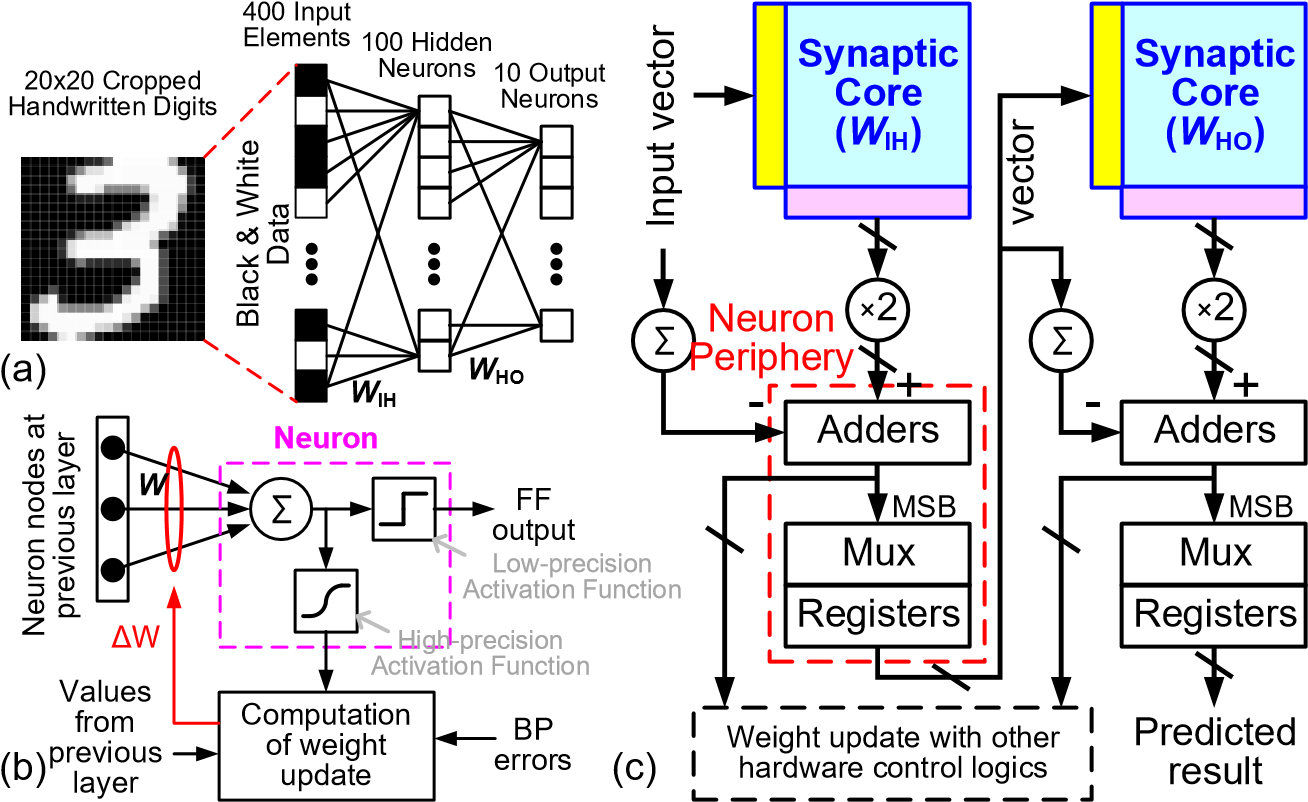


Fig. 12 (a) The 2-layer MLP neural network. (b) Schematic of a neuron node. (c) Circuit block diagram for hardware implementation of the 2-layer MLP network.

In the back propagation phase, the weight update values (ΔW) will be translated to the number of LTP or LTD write pulses (Fig. 12(b)) and applied to the synaptic array following the voltage bias scheme in Fig. 6 or Fig. 8. To reduce the hardware design complexity, we assume that all the selected cells in each write batch have to wait for the full number of write pulse cycles regardless of their ΔW. For example, if the device has 64 conductance levels, then it needs to go through the whole 64 write pulse cycles even when its ΔW=0. For this naïve weight update scheme, the total weight update latency of a synaptic array can be straightforwardly calculated as (number of conductance states) × (number of write batches in an array row) × (number of array rows). The second term, number of write batches, is related to **numWriteColMuxed** in **Param.cpp**, which will be introduced later in the next section.

在反向传播阶段，权重更新值（ΔW）将被转换为LTP或LTD写入脉冲的数量（图12（b））并且在图6或图8中的电压偏置方案之后应用于突触阵列。 为了降低硬件设计的复杂性，我们假设每个写批处理中的所有选定单元必须等待全部写脉冲周期，而不管它们的ΔW如何。 例如，如果器件具有64个电导水平，那么即使其ΔW= 0，它也需要经历整个64个写脉冲周期。 对于这种天真的权重更新方案，突触阵列的总权重更新等待时间可以直接计算为（电导状态的数量）×（阵列行中的写批次的数量）×（阵列行的数量）。 第二个术语，写入批次的数量与Param.cpp中的numWriteColMuxed有关，稍后将在下一节中介绍。

# 7. How to run MLP simulator (+NeuroSim)

Before running the simulator, the device-level and array-level parameters that the user may wish to modify are in **RealDevice** class of **Cell.cpp**, as shown in Fig. 13. The default values are for the Ag:Si example.

在运行模拟器之前，用户可能希望修改的设备级和数组级参数位于Cell.cpp的RealDevice类中，如图13所示。默认值适用于Ag：Si示例。



Fig. 13 Code snippet of **RealDevice** class in **Cell.cpp**

where **maxConductance** and **minConductance** are defined as 1/Ron and 1/Roff respectively, **readVoltage** and **readPulseWidth** are on-chip read voltage (V) and read pulse width (s). The specified value of **readPulseWidth** does not matter because it will be modified later by the read circuit module when it calculates the required pulse width for each integration cycle based on the ADC precision. **writeVoltageLTP** and **writePulseWidthLTP** are the write voltage (V) and the write pulse width (s) during LTP or weight increase. **writeVoltageLTD** and **writePulseWidthLTD** are also defined in the same way. **maxNumLevelLTP** and **maxNumLevelLTD** mean the maximum number of write pulse for LTP and LTD of the real device, which represents the precision. Besides, **NL\_LTP** and **NL\_LTD** represent the nonlinearity values for LTP and LTD, and **sigmaDtoD** and **sigmaCtoC** represent the device-to-device and cycle-to-cycle weight update variation, respectively. Currently the simulator only takes one value of the cycle-to-cycle weight update variation for **sigmaCtoC**. It is encouraged that the user selects the larger one in LTP and LTD for conservative estimation. In the example of Ag:a-Si, **NL\_LTP** and **NL\_LTD** are set to

2.40 and -4.88, which are obtained from the MATLAB fitting results. The cycle-to-cycle weight update variation is set to be 0.035 as the maximum variation of LTP and LTD is 3.5%.

Also, **cmosAccess** is used to choose the cell structure, or synaptic core type in other words. **cmosAccess=true** means the pseudo-crossbar (1T1R) array, while **cmosAccess=false** means the true crossbar array. If the cell is 1T1R, we need to define **resistanceAccess**, which is the turn-on resistance value of the transistor in 1T1R array. The **FeFET** option for the ferroelectric FET is not released yet, thus the default configuration is **FeFET=false**. If the cell is crossbar, I-V nonlinearity **NL** can be specified as the current ratio between write voltage and half write voltage considering if a selector is added. The **nonIdenticalPulse** option is for non-identical write pulse scheme where the write pulse amplitude or width linearly increases or decreases with the pulse number. **VinitLTP**, **VstepLTP**, **VinitLTD**, **VstepLTD**, **PWinitLTP**, **PWstepLTP**, **PWinitLTD** and **PWstepLTD** are essential parameters that need to be defined by the users when **nonIdenticalPulse=true**.

其中maxConductance和minConductance分别定义为1 / Ron和1 / Roff，readVoltage和readPulseWidth是片上读取电压（V）和读取脉冲宽度（s）。 readPulseWidth的指定值无关紧要，因为当读取电路模块根据ADC精度计算每个积分周期所需的脉冲宽度时，它将稍后被修改。 writeVoltageLTP和writePulseWidthLTP是LTP或权重增加期间的写电压（V）和写脉冲宽度（s）。 writeVoltageLTD和writePulseWidthLTD也以相同的方式定义。 maxNumLevelLTP和maxNumLevelLTD表示实际器件的LTP和LTD的最大写脉冲数，表示精度。

此外，NL\_LTP和NL\_LTD表示LTP和LTD的非线性值，sigmaDtoD和sigmaCtoC分别表示设备到设备和周期到周期的权重更新变化。目前，模拟器仅获取sigmaCtoC的循环到循环重量更新变化的一个值。鼓励用户在LTP和LTD中选择较大的一个用于保守估计。在Ag的例子中：a-Si，NL\_LTP和NL\_LTD被设置为2.40和-4.88，它们是从MATLAB拟合结果中获得的。循环到循环重量更新变化设定为0.035，因为LTP和LTD的最大变化为3.5％。

此外，cmosAccess用于选择单元格结构，换句话说就是突触核心类型。 cmosAccess = true表示伪交叉开关（1T1R）数组，而cmosAccess = false表示真正的交叉开关数组。 如果单元是1T1R，我们需要定义resistanceAccess，这是1T1R阵列中晶体管的导通电阻值。 铁电FET的FeFET选项尚未发布，因此默认配置为FeFET = false。 如果单元是交叉开关，则考虑是否添加了选择器，可以将I-V非线性NL指定为写入电压和半写入电压之间的电流比。 nonIdenticalPulse选项用于不相同的写脉冲方案，其中写脉冲幅度或宽度随脉冲数线性增加或减少。 当nonIdenticalPulse = true时，VinitLTP，VstepLTP，VinitLTD，VstepLTD，PWinitLTP，PWstepLTP，PWinitLTD和PWstepLTD是用户需要定义的基本参数。

In the file **Param.cpp**, there are also network-level parameters that the user may wish to modify, as shown in Fig. 14.

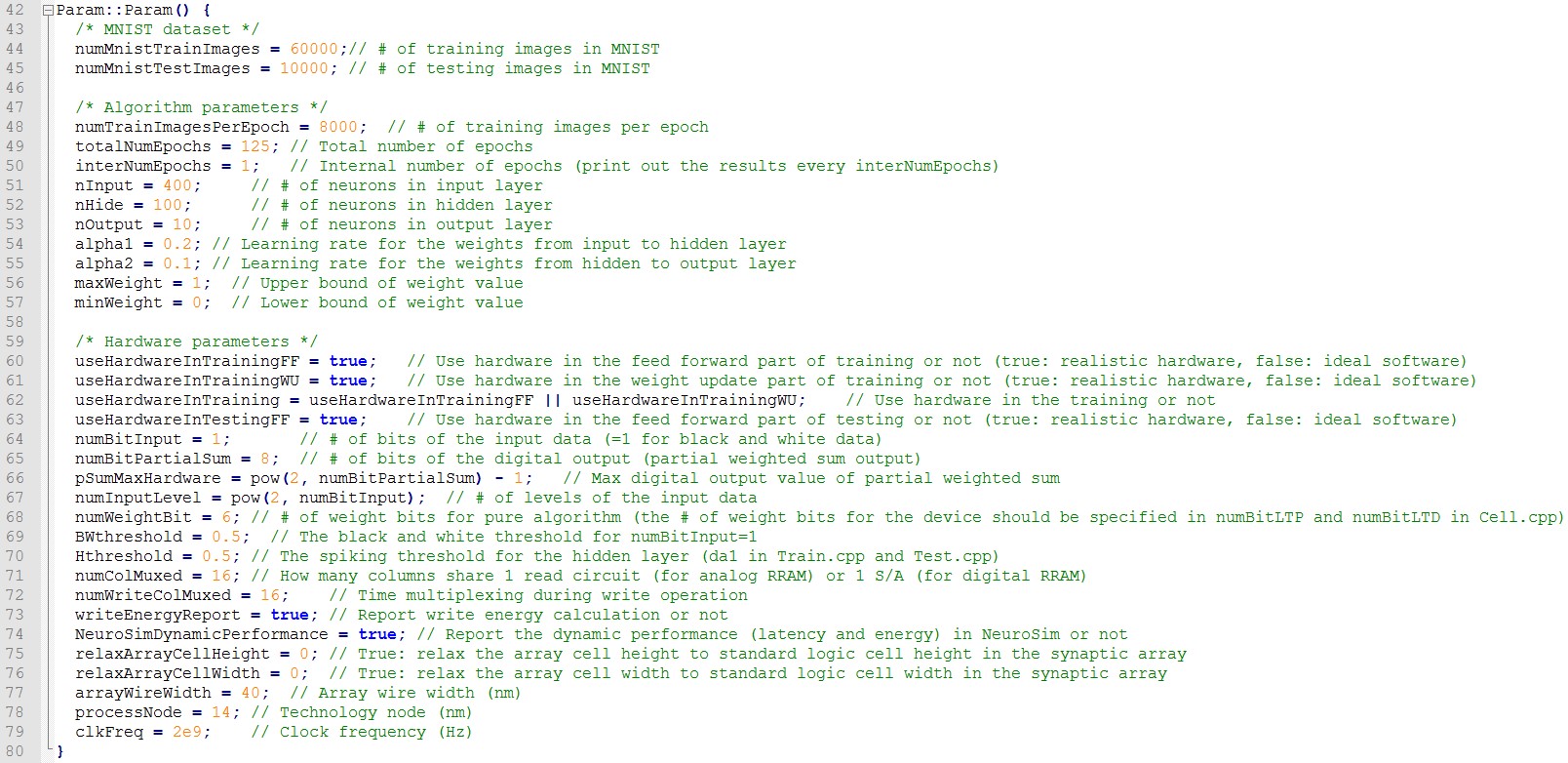


Fig. 14 Code snippet of **Param.cpp**

where **numMnistTrainImages** and **numMnistTestImages** are the number of images in MNIST during training and testing respectively, **numTrainImagesPerEpoch** means the number of training images per epoch, while **interNumEpochs** represents the internal number of epochs within each printed epoch shown on the screen. In addition, **nInput**, **nHide** and **nOutput** are the number of neurons in input, hidden and output layers in the 2-layer MLP neural network, respectively.

其中numMnistTrainImages和numMnistTestImages分别是训练和测试期间MNIST中图像的数量，numTrainImagesPerEpoch表示每个时期的训练图像数，而interNumEpochs表示屏幕上显示的每个打印时期内的内部时期数。 此外，nInput，nHide和nOutput分别是2层MLP神经网络中输入，隐藏和输出层中神经元的数量。

The first four hardware parameters determine the learning configuration, which can be the following cases:

#### 1. Online learning in hardware: **useHardwareInTrainingFF, useHardwareInTrainingWU** and **useHardwareInTestingFF** are all **true**

2. Offline learning in software and then classification only in hardware: **useHardwareInTrainingFF** and **useHardwareInTrainingWU** are **false**,while **useHardwareInTestingFF** is **true**

#### 3. Pure learning in software: **useHardwareInTrainingFF, useHardwareInTrainingWU** and **useHardwareInTestingFF** are all **false**

For other hardware parameters, **numBitInput** means the number of bits of the input data. The hardware architecture design in this released version only allows **numBitInput=1** (black and white data), which should not be changed. **numBitPartialSum** represents the number of bits in the digital output (partial weighted sum output) of read circuit (ADC) at the periphery of the synaptic array. **numWeightBit** means the number of weight bits for pure algorithm without consideration of hardware, and **numColMuxed** means the number of columns of the synaptic array sharing one read circuit in the array. Time-multiplexing is required if **numColMuxed** is greater than 1. For example, the total weighted sum latency will be increased by roughly 16 times if **numColMuxed=16**. In the weight update, there might also be limited throughput for the weight update information to be provided from outside. In this case, time-multiplexing is implemented by setting **numWriteColMuxed**. For example, **numWriteColMuxed=16** means updating every row will need roughly 16 weight update operations.

对于其他硬件参数，numBitInput表示输入数据的位数。此发布版本中的硬件架构设计仅允许numBitInput = 1（黑白数据），不应更改。 numBitPartialSum表示在突触阵列外围的读取电路（ADC）的数字输出（部分加权和输出）中的位数。 numWeightBit表示纯算法的权重位数，不考虑硬件，numColMuxed表示在阵列中共享一个读取电路的突触阵列的列数。如果numColMuxed大于1，则需要时间复用。例如，如果numColMuxed = 16，则总加权和延迟将增加大约16倍。在权重更新中，从外部提供的权重更新信息的吞吐量可能也有限。在这种情况下，通过设置numWriteColMuxed来实现时间复用。例如，numWriteColMuxed = 16意味着更新每一行将需要大约16个权重更新操作。

Basically, the parameters for running the simulator are all included in the file **Cell.cpp** and **Param.cpp**. Whenever any change is made in the files, the codes has to be recompiled by using make command as stated in **Usage (Linux)** section. If the compilation is successful, the following screenshot of Fig. 15 can be expected:

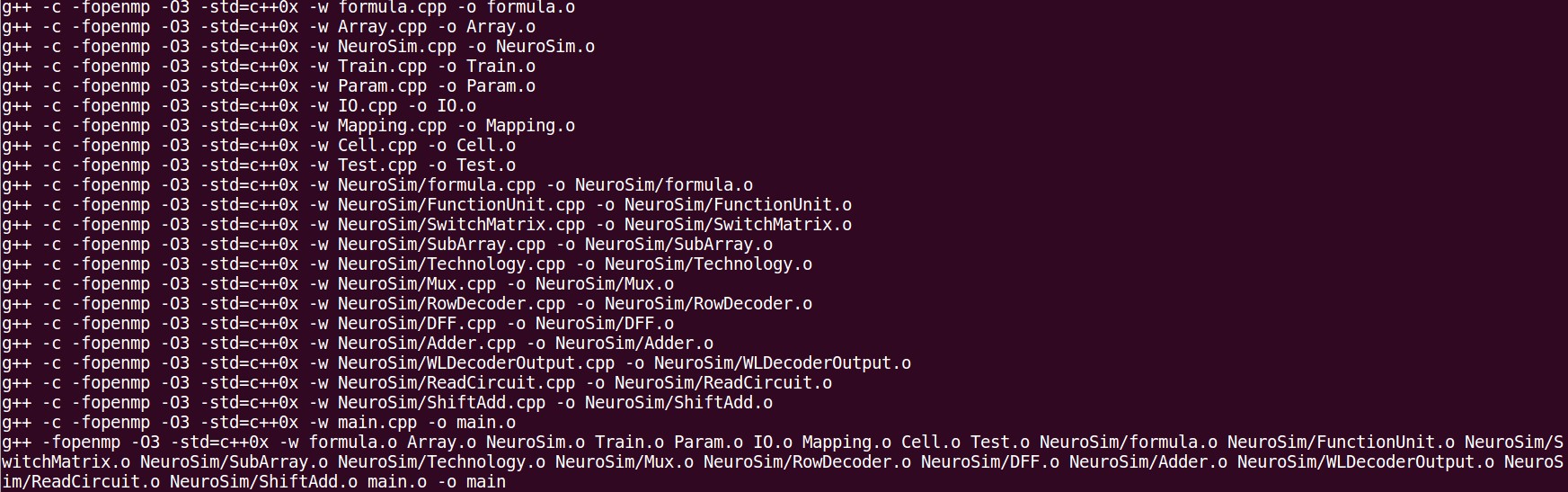


Fig. 15 Output of make

Then, use ./main or make run to run the program. For both commands, the program will print out the results at every epoch during the simulation. Compared to ./main, make run can save the results to a log file with filename appended with the current time info, as shown in Fig. 16. With the default value of **totalNumEpochs=125** and **numTrainImagesPerEpoch=8000** for a total 1 million MNIST images, the simulation will approximately take about 40 mins with an Intel i7 CPU and 32 GB RAM.

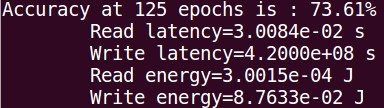
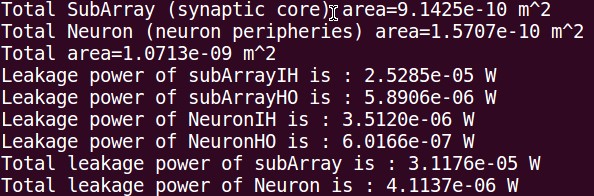


Fig. 16 Output of the simulation

At the end of simulation, it is expected to have similar results for the Ag:a-Si example in Table 2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Accuracy | Area (m2) | Read Latency (s) | Write Latency (s) | Read Energy (J) | Write Energy (J) |
| 73.61% | 1.0713e-9 | 3.0084e-2 | 4.2000e8 | 3.0015e-4 | 8.76e-2 |

Table 2 Final results of learning accuracy and circuit-level performance

For the accuracy of pure software learning as the reference, the users could change the learning modes in **Param.cpp** as discussed above,and it is 96~97% for a network topology 400-100-10.