ECE 464 / 564 Project : Fall 2020 2021:
Binary Artificial Neural Network

Change Log
- 9/10/21 original spec
- 10/12/21 Problem size limits added

This project is to be conducted individually. You can collaborate on the paper version of the design, including discussion of ideas, design approach, etc. However, you are forbidden to share code or to reuse code of others. We will be running code comparison tools on your submitted code.

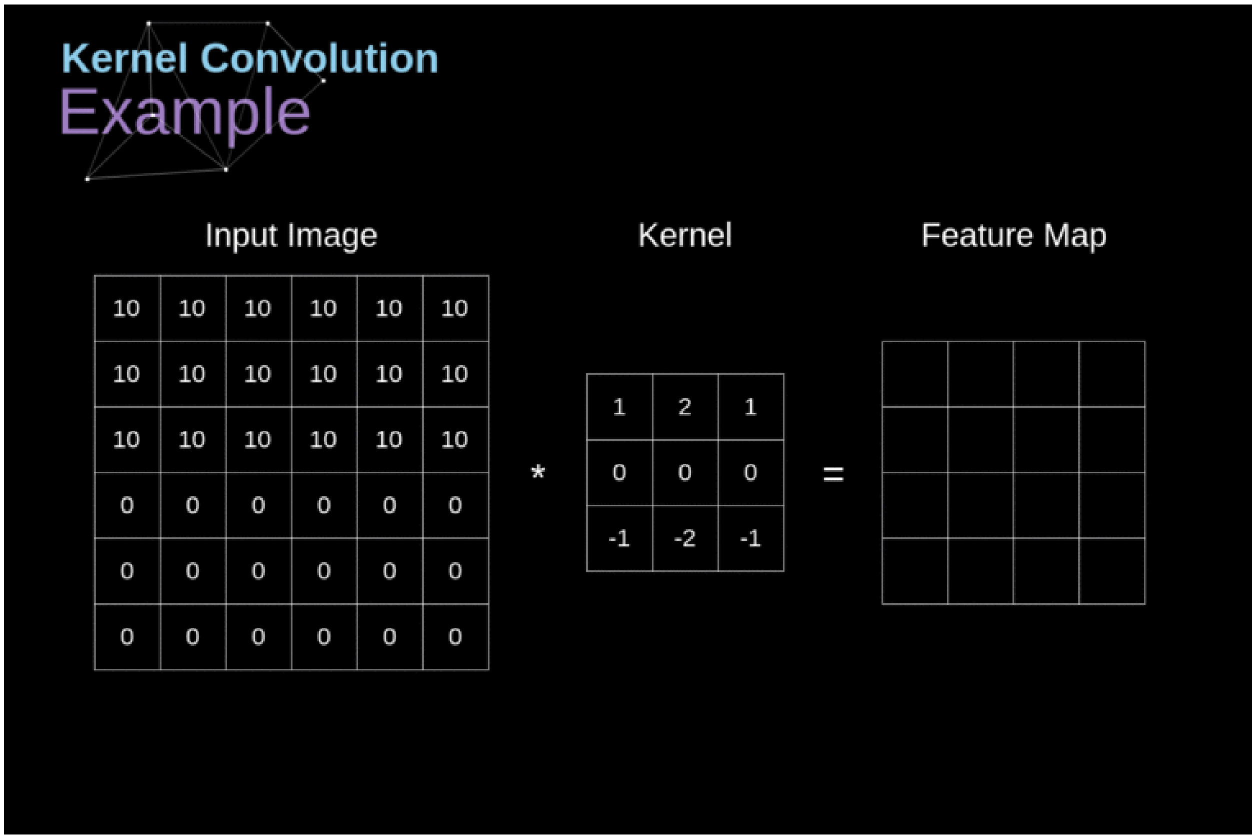
EOL students are expected to do the ECE 464 project.

ECE 564: Your task is to implement a single stage of an all binary convolutional neural network. All the weights and input data is binary. They take on values of -1 and 1, which are represented by 0 and 1 respectively. The inputs are going to come from an input SRAM, the weights from a weight SRAM and the outputs written to an output SRAM memory. The sizes of each matrix are stored in the SRAMs and can change, and multiple problems might be stored in the memories.

ECE 464 and EOL students: Your task is to implement a fixed size single stage of a binary convolutional neural network.

Convolutional Neural Network

The network above is fully connected. In a convolutional neural network, the weight matrix is smaller than the input matrix, and an output matrix is produced. An example is taken from the figure below. This figure is taken from https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9 which includes a good animation of how this calculation is done.



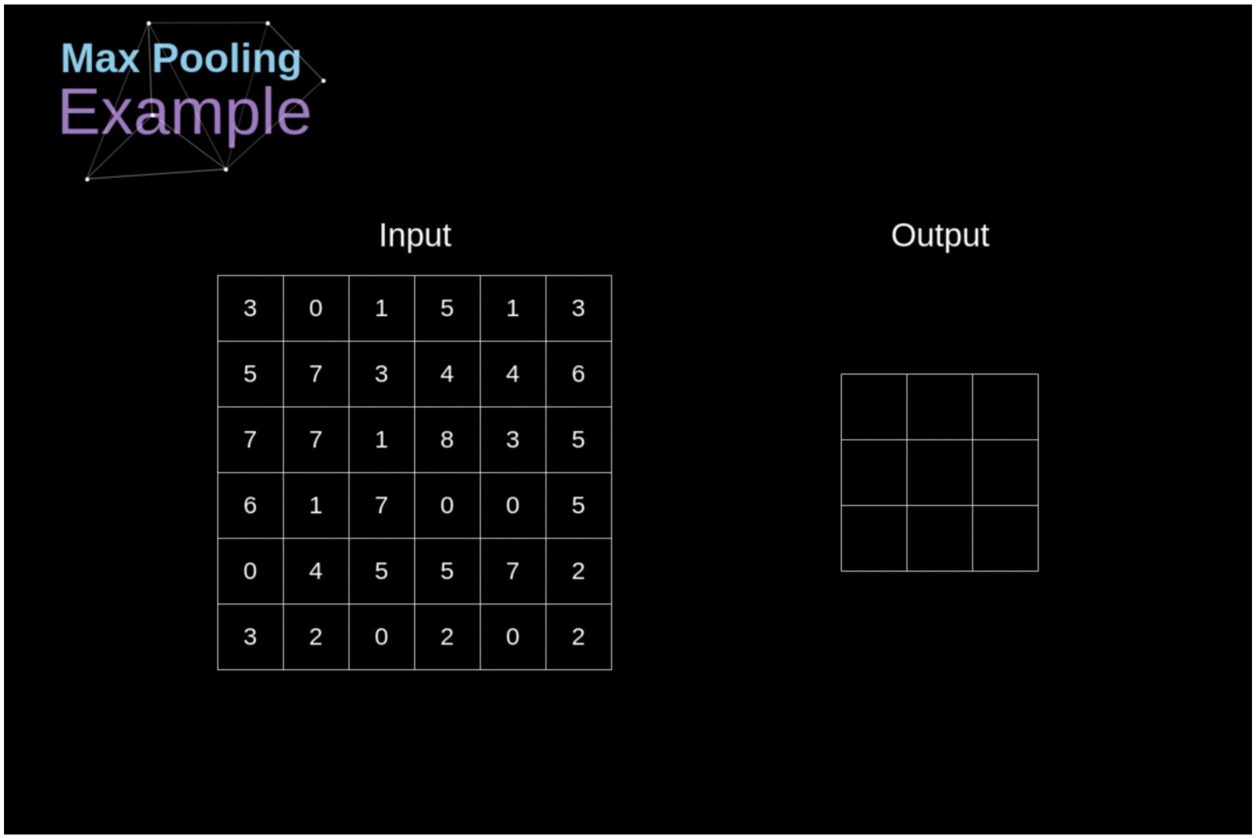
10	10	10	10	10	10
10	10	10	10	10	10
10	10	10	10	10	10
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

1	2	1
0	0	0
-1	-2	-1

Pooling

Pooling reduces the matrix size by taking the max, min or average of each region of the matrix. For example, max pooling each 2x2 region of a 6x6 array produces a 3x3 array. In the example below the top left output array element is max(3, 0 5, 7) = 7.

Pooling is not needed in the 2021 project.



Max Pooling
Example

Input

3	0	1	5	1	3
5	7	3	4	4	6
7	7	1	8	3	5
6	1	7	0	0	5
0	4	5	5	7	2
3	2	0	2	0	2

Output

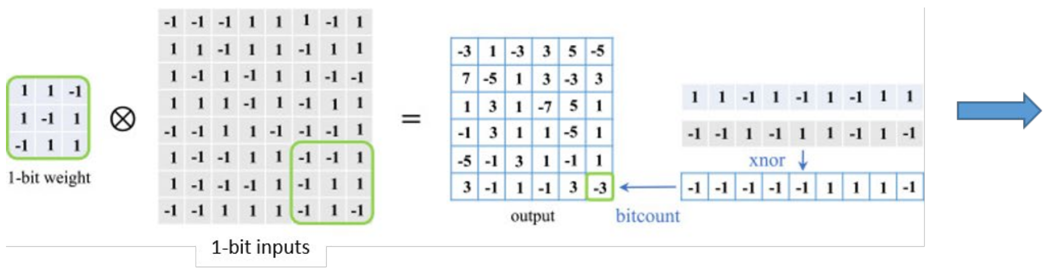
Binary Neural Networks

About five years ago, researchers realized that the neural network can be trained so that weights and inputs can be rounded to just their sign, -1 and 1. That allows multiply and add to be replaced by XNORs and bit counting. Details can be found in the following articles, as well as other sources:

- https://arxiv.org/abs/1603.05279
- https://arxiv.org/abs/1511.00363

We will be implementing a slightly simplified version than what is described in these papers. Below is an example of a binary convolutional stage. (Adapted from https://www.sciencedirect.com/science/article/abs/pii/S0031320320300856). Remember -1 is represented by 0 in the hardware.

In this project we will implement only a single convolutional stage.



```
-1  1 -1  1  1 -1
 1 -1  1  1 -1  1
 1  1  1 -1  1  1
-1  1  1  1 -1  1
-1 -1  1  1 -1  1
 1 -1  1 -1  1 -1
```

Stage output after sign activation

An example of a two stage binary neural network is given as follows: This will be illustrated for the following simplified example.

- Convolution. 3x3 weight matrix; 4x4 input matrix. Thus the output matrix is 2x2.

$$\begin{matrix} w00 & w01 & w02 \\ w10 & w11 & w12 \\ w20 & w21 & w22 \end{matrix} \otimes \begin{matrix} i00 & i01 & i02 & i03 \\ i10 & i11 & i12 & i13 \\ i20 & i21 & i22 & i23 \\ i30 & i31 & i32 & i33 \end{matrix} = \begin{matrix} o00 & o01 \\ o10 & o11 \end{matrix}$$

*o00 = sign(w00*i00+w01*i01+ … +w22*i22)*

Etc.

Example

$$\begin{matrix} -1 & 1 & -1 \\ 1 & 1 & 1 \\ -1 & 1 & -1 \end{matrix} \otimes \begin{matrix} 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{matrix} = \begin{matrix} -1 & 1 \\ 1 & 1 \end{matrix}$$

For the ECE 564 project, the sizes, weights and inputs are stored in weight and input memory as follows. Each word of memory is 16 bits wide. The weights and inputs are stored in an SRAM. i.e. each word is 16-bits wide. The weight memory for this example is as follows:

Generic structure of weight memory. All the data is packed. The msb is the highest order bit in the word.

Address	Data	Comment
0000	dimen	Weight matrix is dimen x dimen in size (integer)
0001	packed	Binary weights in packed format

For this example since there are only 9 weights, they will fit into one word. Assume a maximum of 16 weights, so it will always fit in one word.

In this example, the packed data entry is

0000 000w22 w21w20w12w11 w10w02w01w00

Symbolically, or in binary

0000 0000 1011 1010

Or in hex

00BA

Generic Structure of input memory. All data is packed.

Address	Data	Comment
0000	dimen1	Input matrix is dimen1 (x) x dimen2 (y)
0001	dimen2	
0002		Packed inputs. Add words as needed.
…		
000n	dimen1	Next input
000m	dimen2	

In the example above, dimen1=dimen2 = 4, and only one word is needed to store the input data. That word is generically

I33 i32 … i01 i00

i.e. the msb is i33 while the lsb is i00

or in binary,

0000 1111 0110 0101

Or in hex

0FA9

Assume the largest input you need to support is 1024x1024.

Problem Sizes

The weight matrix is limited to 3x3.
The input matrix is one of the following sizes: 16x16; 12x12; 10x10.
See the readme_564 file to see how these are organized in memory.

For the ECE 464 The weight memory stores a 3x3 weight matrix packed as follows…

0000 000w22 w21w20w12w11 w10w02w01w00

Symbolically, or in binary, for this example

0000 0000 1011 1010

Or in hex

00BA

The input is sized to support a 16x16 input matrix. i.e. its 16 words, each 16-bits long. The first word consists of the following entries, msb to lsb

Address	Data
0000	i0,16 i0,15 … i0,1 i0,0
0001	i1,16 i1,15 … i1,1 i1,0

etc.

Upon going high, you process this one example and then stop.

Problem Sizes

The weight matrix is limited to 3x3.
The input matrix is limited to 4x4.
See the readme_464 file to see how these are organized in memory.

Overall Schema

The overall schema is as follows:

Control Interface
There will be three control signals for the unit.

input	reset_b	: Active low reset signal, will clear the machine state
input	clk	: System clock forwarded from the test fixture.
input	go	; Start processing if high

output busy : The test bench will halt when busy is high, waiting for the computation.

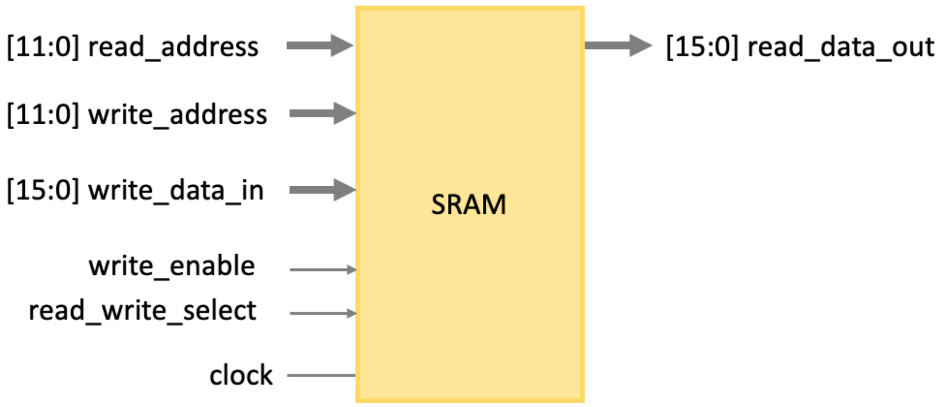
Memory structure

There are three SRAMs provided, one for weights, one for input data, and one for output results. Each memory is 16 bits wide, and is 16-bit word addressable. The weight matrices won't change during a run. Multiple inputs will be presented though.

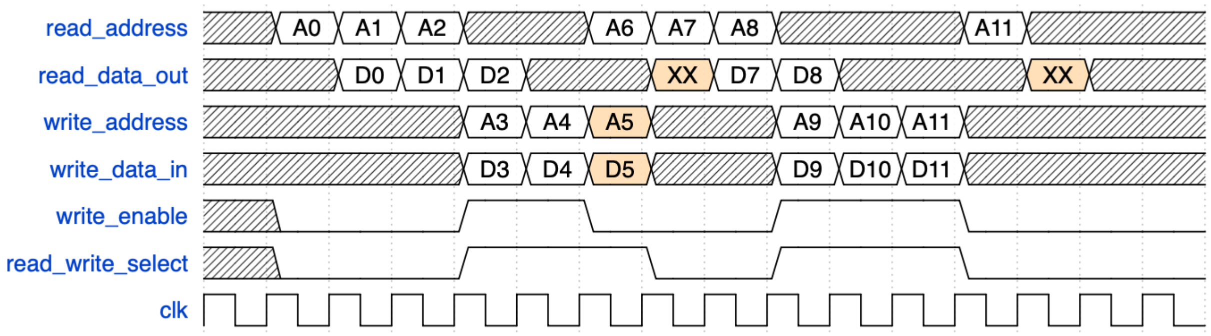
Zero Padding

Since only complete 16-bit words have to be written to the memory you need to zero pad any words in which the outputs are less than 16 bits wide. For example if the output is 4-bits wide the high order 12 bits must be zeros.

SRAM interface



The SRAM is word addressable and has a one cycle delay between address and data. When writing to the SRAM, you would have to set the "write_enable" to high. The SRAM will write the data in the next cycle.



As shown in the example above, since “write_enable” is set to low when A5 and D5 is on the write bus, D5 will not be written to the SRAM. Also, because “read_write_select” is set to high, the read request for A6 will not be valid.
Note that the SRAM cannot handle consecutive read after write (RAW) to the same address (shown as A11 and D11 in the timing diagram). You would have to either manage the timing of your access, or write the data forwarding mechanism yourself. As long as the read and write address are different, the request can be pipelined.