

# 18: Hardware Acceleration I

Engr 315: Hardware / Software Codesign

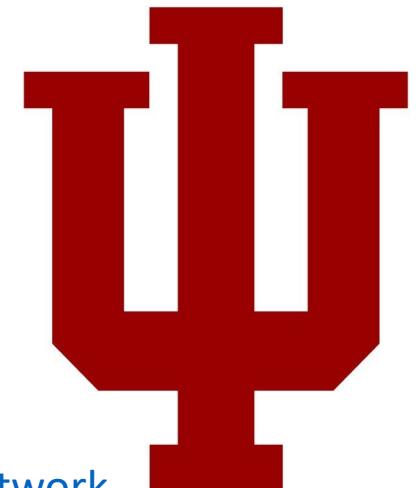
Andrew Lukefahr

*Indiana University*

Some material taken from:

[https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural\\_network](https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural_network)

<http://cs231n.github.io/neural-networks-1/>



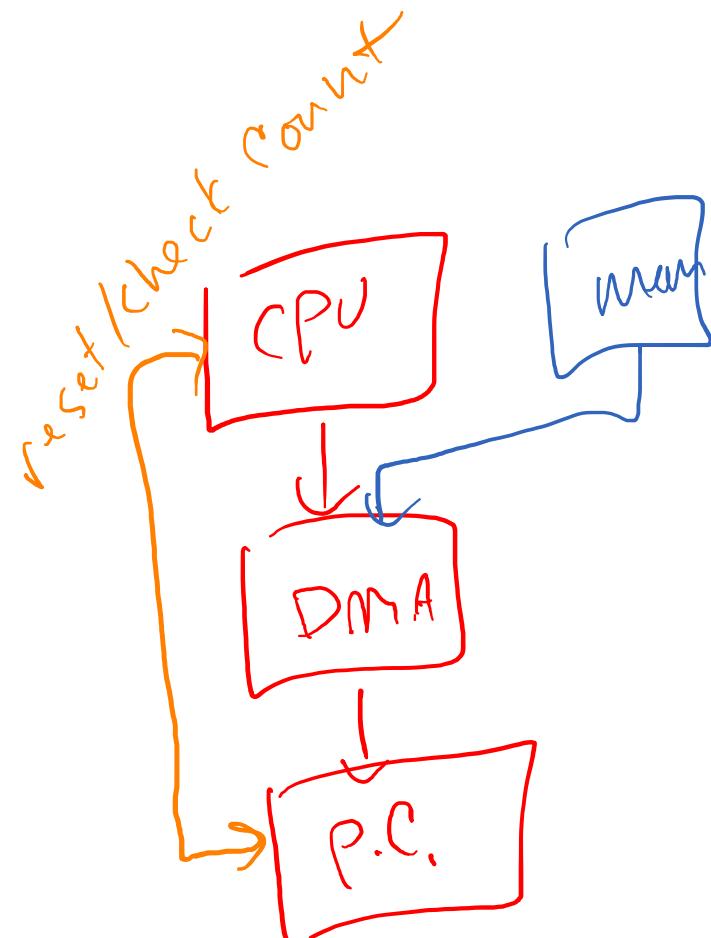
# Announcements

- P4 is due TOMORROW (not today)
  - Bitstream / hwh files added
  - Password: ‘iuxilinx’
- P5 is out
- Exam in 3 weeks

P6: More  
P7: Ropcount DMA

P4: volatile

# P5: Adds DMA + AXI-Stream to Popcount



- DMA
  - Add DMA engine to move data via AXI4-Full to AXI-Stream interface
- Popcount.sv:
  - Add AXI-Stream Interface
  - Keep AXI4-Lite Interface to read result

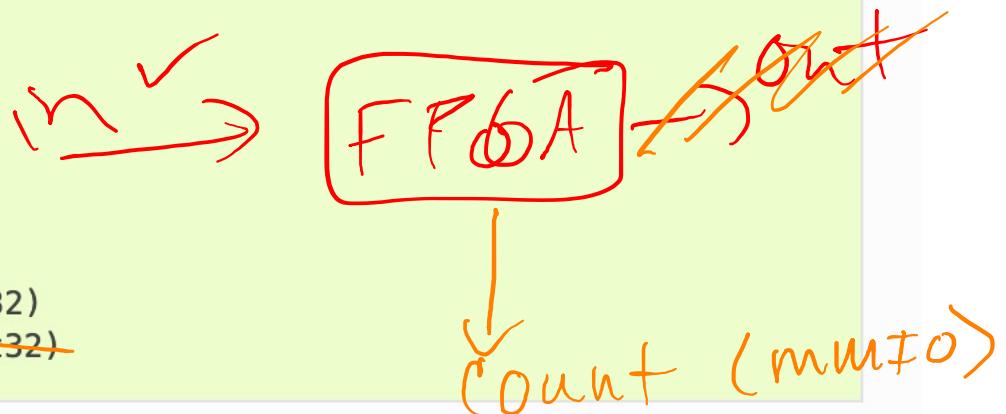
# Example DMA

[https://pynq.readthedocs.io/en/v2.6.1/pynq\\_libraries/dma.html](https://pynq.readthedocs.io/en/v2.6.1/pynq_libraries/dma.html)

```
import numpy as np
from pynq import allocate
from pynq import Overlay

overlay = Overlay('example.bit')
dma = overlay.axi_dma

input_buffer = allocate(shape=(5,), dtype=np.uint32)
output_buffer = allocate(shape=(5,), dtype=np.uint32)
```

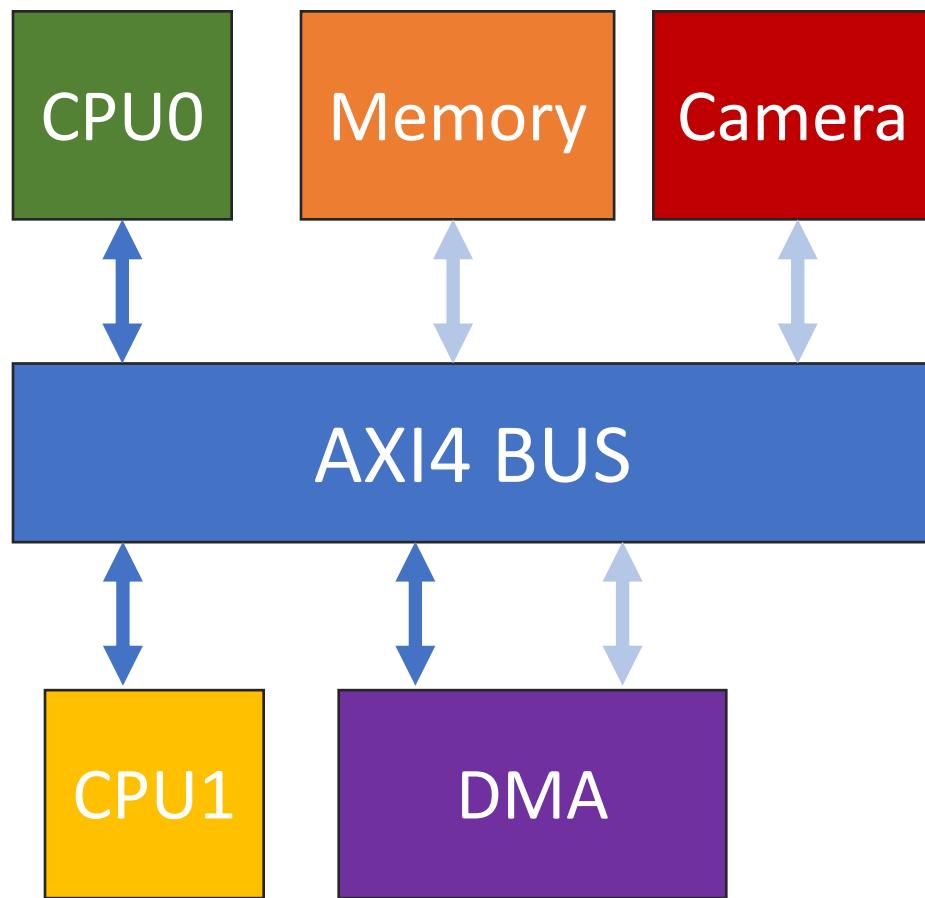


```
for i in range(5):
    input_buffer[i] = i
```

```
dma.sendchannel.transfer(input_buffer)
dma.recvchannel.transfer(output_buffer)
dma.sendchannel.wait()
dma.recvchannel.wait()
```

Output buffer will contain: [0 1 2 3 4]

# DMA has 2 Memory Interfaces



- Interface 1: Memory Copy
  - Data Interface
  - Fast
  - Master
- Interface 2: Tell DMA what to copy
  - Control Interface
  - Slower
  - Slave

# All MMIO Registers

CPU Store  
Order

Control - 0x0400 <i>R/W</i>	31-1 Reserved	0 Start	4
Status - 0x0404 <i>RD</i>	31-1 Reserved	0 Done	
Source - 0x0408 <i>R/W</i>	31-0 DMA Source Address		1-3
Destination - 0x040C <i>R/W</i>	31-1 DMA Destination Address		1-3
Size - 0x0410 <i>R/W</i>	31-16 Reserved	15-0 DMA Transfer Size (in Bytes)	1-3

# Using DMA from the CPU:

0x0400: Control Register  
0x0404: Status Register  
0x0408: Source Address  
0x040C: Destination Address  
0x0410: Transfer Size in Bytes

```
void dma_copy ( uint32_t * src,
                uint32_t * dest,
                uint32_t size) {

    * ( (volatile uint32_t *) (0x0408))=src;
    * ( (volatile uint32_t *) (0x040C))=dest;
    * ( (volatile uint32_t *) (0x0410))=size;
    * ( (volatile uint32_t *) (0x0400))= 0x1; //start

    //spin until copy done
    while( * ( (volatile uint32_t *) (0x0404)) != 0x1) { ; }

}
```

# P6 – DMA from C

A screenshot of a search results page. The search bar at the top contains the query "xilinx dma ip". Below the search bar are navigation links: All (highlighted), News, Shopping, Images, Videos, More, and Tools. A message indicates "About 294,000 results (0.50 seconds)". The first result is a link to the "AXI DMA v7.1 LogiCORE IP Product Guide - Xilinx" document, which was published on Jun 14, 2019. The document provides high-bandwidth direct memory access between AXI4 memory mapped and AXI4-Stream IP cores, spanning 97 pages. The URL for the document is [https://www.xilinx.com/support/documentation/ip\\_documentation/axi\\_dma/v7\\_1/pg021\\_axi\\_dma.pdf](https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf).

[https://www.xilinx.com/support/documentation/ip\\_documentation/axi\\_dma/v7\\_1/pg021\\_axi\\_dma.pdf](https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf)

P6 – DMA from C

## **Programming Sequence**

### **Direct Register Mode (Simple DMA)**

## P6 – DMA from C

1. Start the MM2S channel running by setting the run/stop bit to 1 (MM2S\_DMACR.RS = 1). The halted bit (DMASR.Halted) should deassert indicating the MM2S channel is running.

Set MM2S\_DMACR.RS = 1

wait until DMASR.Halted = 0

# P6 – DMA from C

1. Start the MM2S channel running by setting the run/stop bit to 1 (MM2S\_DMACR.RS = 1). The halted bit (DMASR.Halted) should deassert indicating the MM2S channel is running.
2. Skip

# P6 – DMA from C

1. Start the MM2S channel running by setting the run/stop bit to 1 (MM2S\_DMACR.RS = 1). The halted bit (DMASR.Halted) should deassert indicating the MM2S channel is running.
2. Skip
3. Write a valid source address to the MM2S\_SA register.

# P6 – DMA from C

1. Start the MM2S channel running by setting the run/stop bit to 1 (MM2S\_DMACR.RS = 1). The halted bit (DMASR.Halted) should deassert indicating the MM2S channel is running.
2. Skip
3. Write a valid source address to the MM2S\_SA register.
4. Write the number of bytes to transfer in the MM2S\_LENGTH register.  
The MM2S\_LENGTH register must be written last.

*← triggers  
the  
DMA*

# P6 – DMA from C

1. Start the MM2S channel running by setting the run/stop bit to 1 (MM2S\_DMACR.RS = 1). The halted bit (DMASR.Halted) should deassert indicating the MM2S channel is running.

## 2. Skip

3. Write a valid source address to the MM2S\_SA register.
4. Write the number of bytes to transfer in the MM2S\_LENGTH register.

The MM2S\_LENGTH register must be written last.

5. ~~poll~~ MM2S\_DMASR.Idle bit for completion

Wait until MM2S\_DMASR.Idle = 1

# P7+ Accelerate Machine Learning

- Goal: Accelerate reference neural network
- Harder, more open-ended projects
- Groups of 2 allowed.

# Accelerate Machine Learning

- Given: Python starter code to classify digits
- Your Task: Accelerate it!
- How: Up to you!

Grade: Based on overall Speedup!

# Project 6: Dot Product

- Given: Slow Dot-Product in Hardware
- Your Task: Accelerate it!
- How? Pipelining + Parallelism
- Why? Project X ↩

# What Number is this?

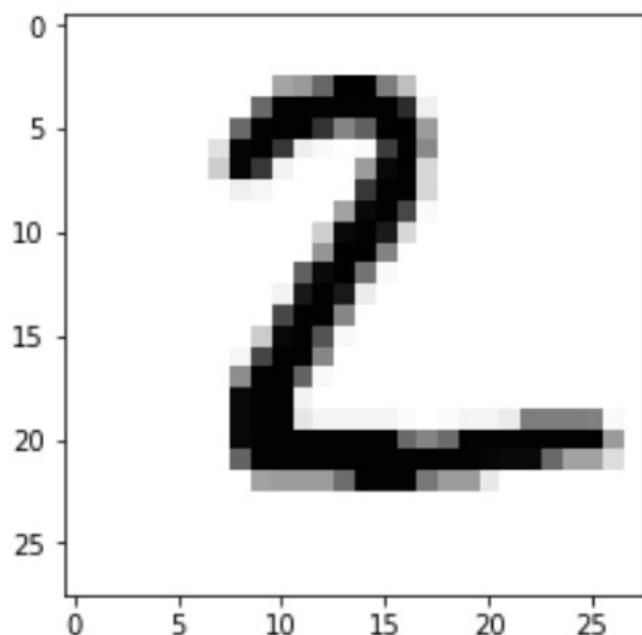
## Raw data, in hex

# Simple Neural Network

=====

Index: 0

Image:



- Takes in image of number
- Returns integer value
- How? artificial neural network

→ ML Classification Result: 2

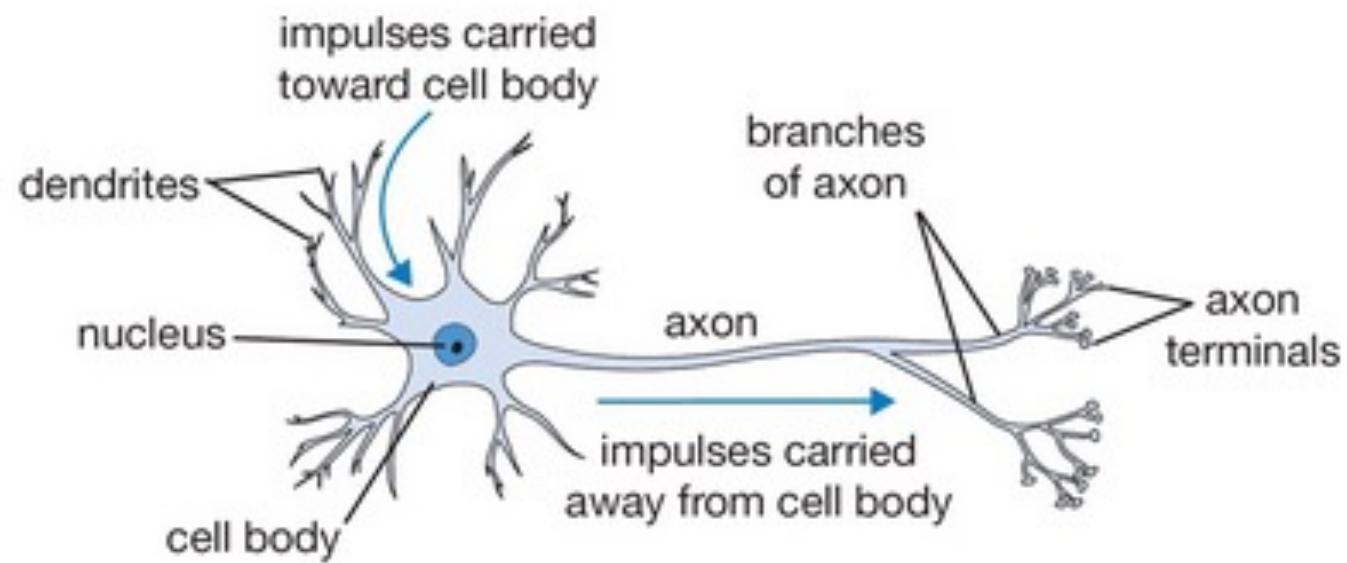
Real Value: 2

Correct Result: True

=====

# Fully-Connected Neural Networks

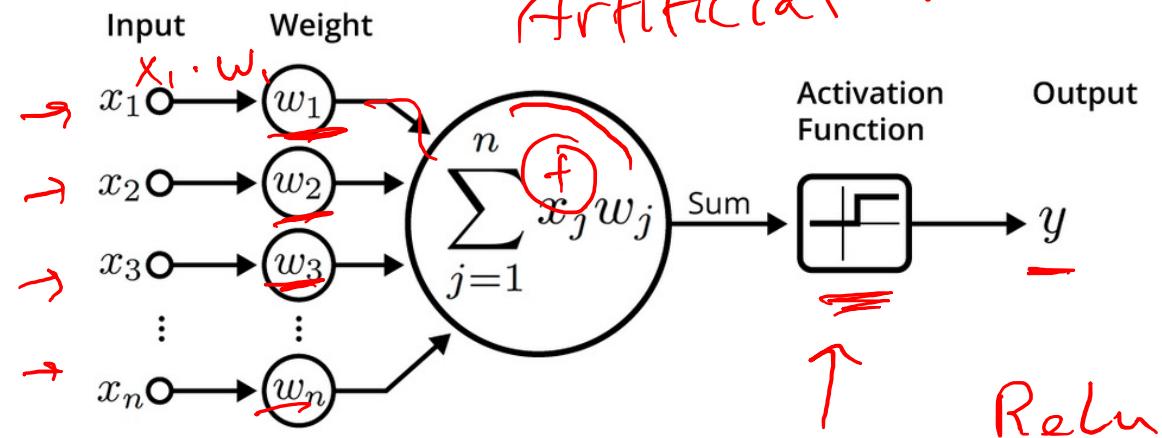
# The Biological Neuron



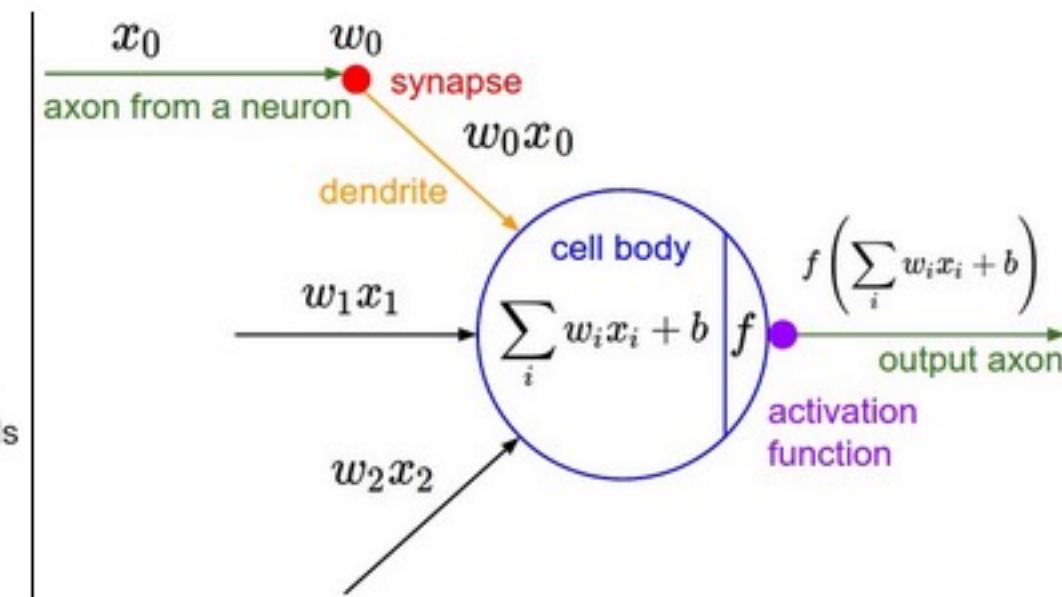
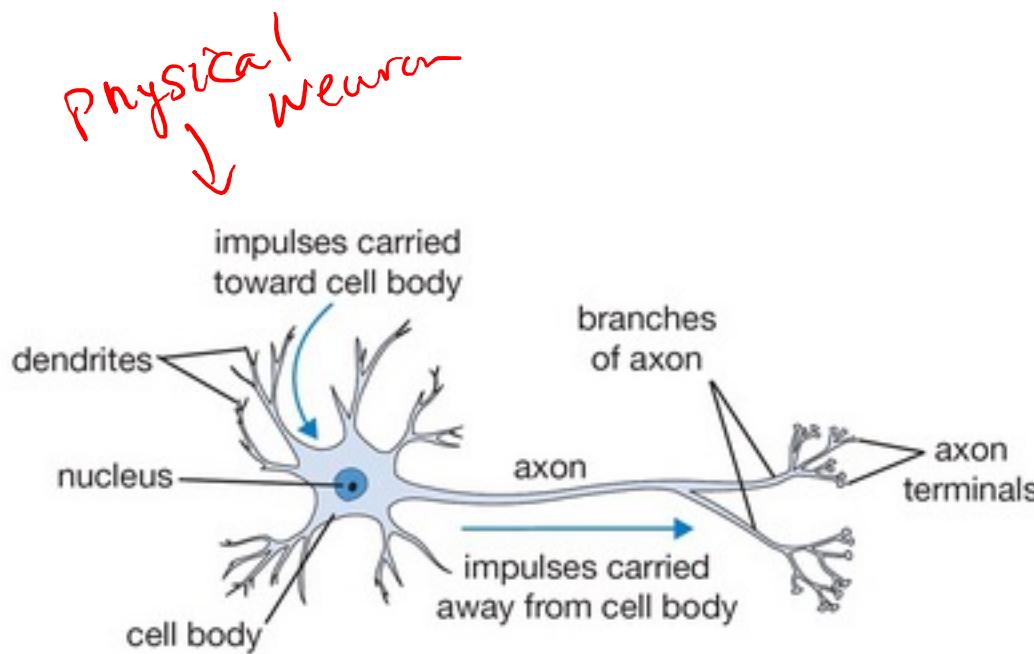
# The Math Neuron

# The Math Neuron

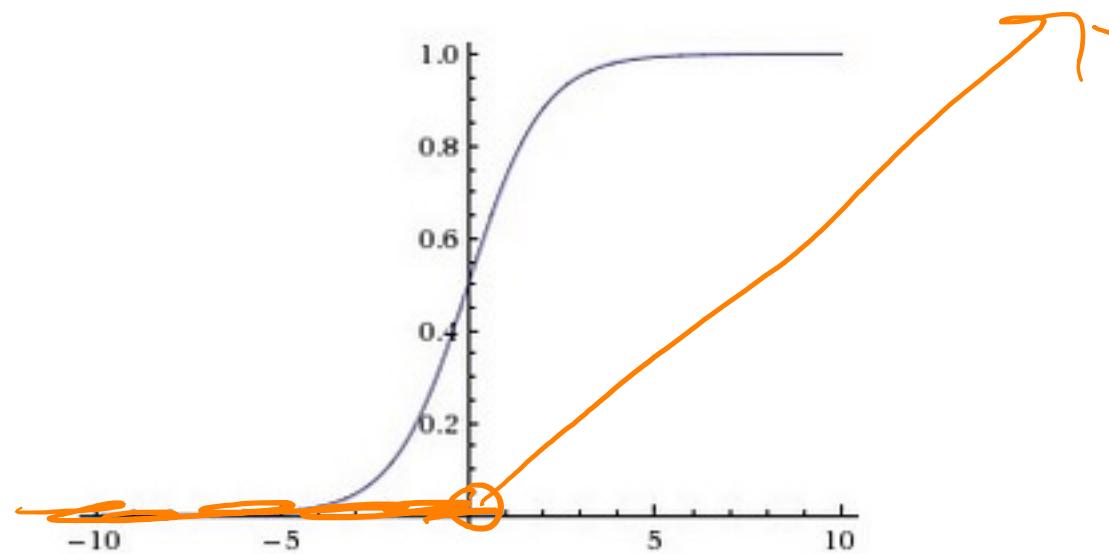
Artificial  
Artificial Neuron



An illustration of an artificial neuron. Source: Becoming Human.



# Activation Function: Sigmoid

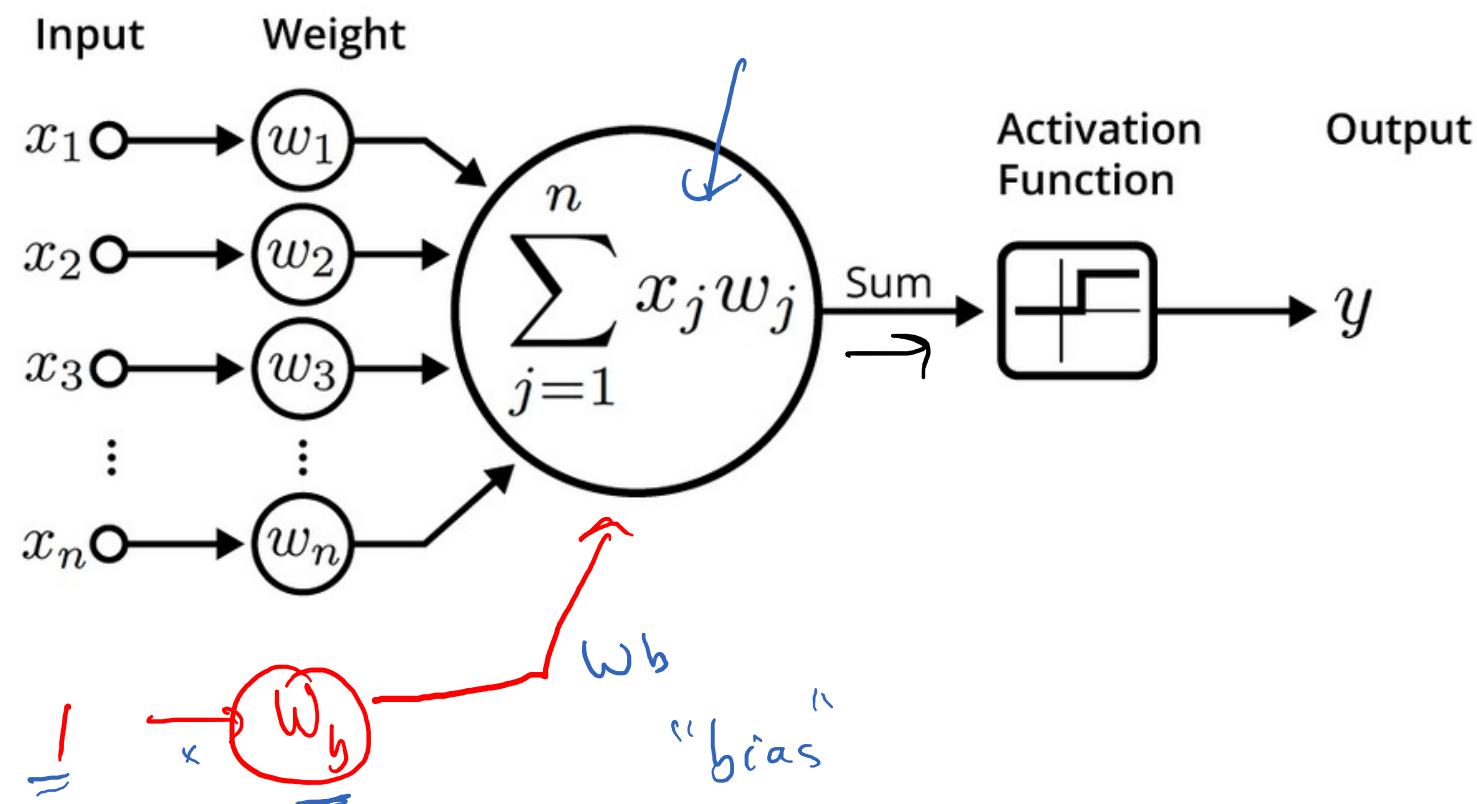


if  $x \leq 0 \Rightarrow 0$   
else  $\Rightarrow K$

$$\sigma(x) = 1/(1+e^{-x})$$

# Adding Bias

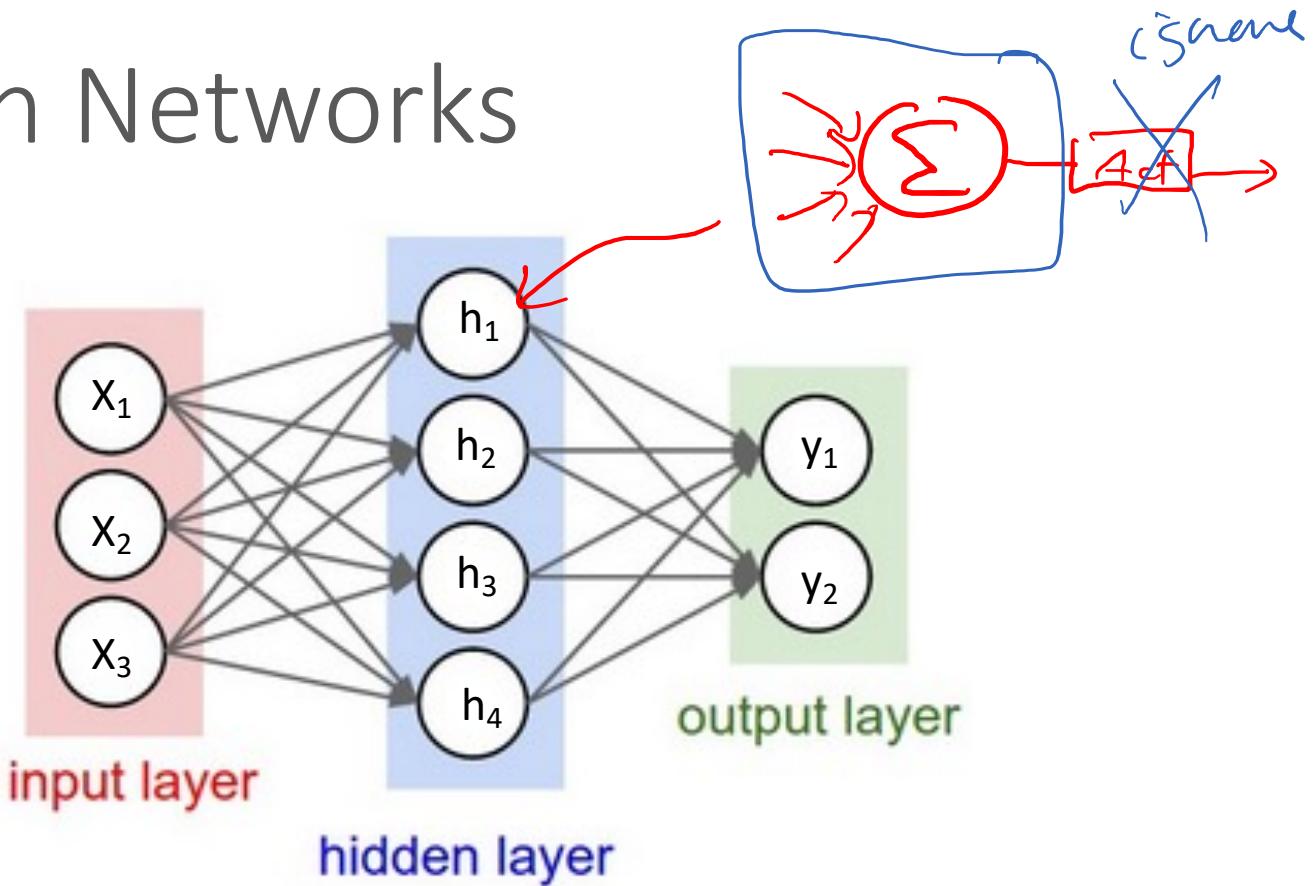
$$\sum_{j=1}^n x_j w_j + b \text{ (or } w_b)$$



# Python Neuron

```
class Neuron(object):
    # ...
    def forward(self, inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
        return firing_rate
```

# Multi-Neuron Networks



Input Layer: Just input values

Output Layers: No Activation Function

Let's focus on just the Matrix for now.

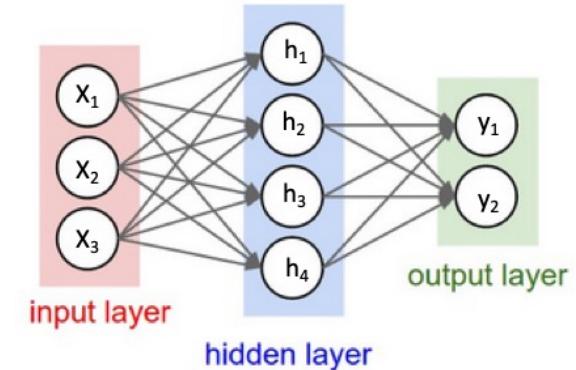
$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \cdot \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix}$$

$$h_1 = x_1 \cdot w_{11} + x_2 \cdot w_{12} + x_3 \cdot w_{13}$$

$$h_2 = x_1 \cdot w_{21} + x_2 \cdot w_{22} + x_3 \cdot w_{23}$$

$$h_3 = x_1 \cdot w_{31} + x_2 \cdot w_{32} + x_3 \cdot w_{33}$$

$h_4$  is similar..

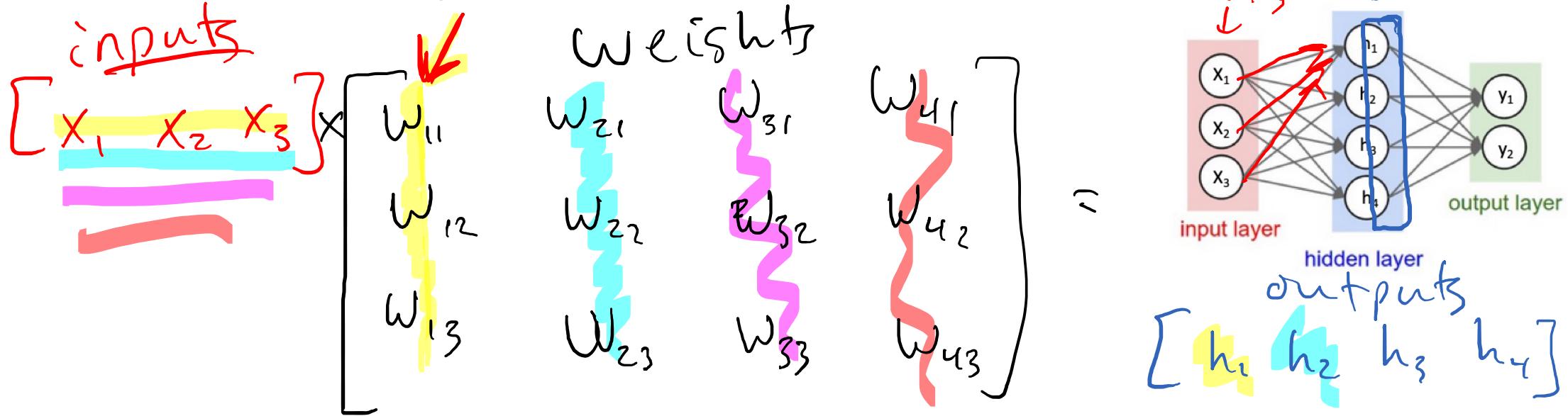


true  $h_i =$   
 $\underbrace{\text{sigmoid}}_{\leftarrow}(\underbrace{\dots}_{\sim} + b)$

vector dot product

Matrix multiplication

Let's focus on just the Matrix for now.



$$\rightarrow h_1 = x_1 \cdot w_{11} + x_2 \cdot w_{12} + x_3 \cdot w_{13}$$

$$h_2 = x_1 \cdot w_{21} + x_2 \cdot w_{22} + x_3 \cdot w_{23}$$

$$h_3 = x_1 \cdot w_{31} + x_2 \cdot w_{32} + x_3 \cdot w_{33}$$

$$h_4 = x_1 \cdot w_{41} + x_2 \cdot w_{42} + x_3 \cdot w_{43}$$

## Matrix Multiplication (Dot Product)

$$\begin{bmatrix} \text{inputs} \\ i_0 \quad i_1 \end{bmatrix} \times \begin{bmatrix} \text{weights} \\ w_{00} \quad w_{10} \quad w_{20} \\ w_{01} \quad w_{11} \quad w_{21} \end{bmatrix} = \begin{bmatrix} \text{outputs} \\ o_0 \quad o_1 \quad o_2 \end{bmatrix}$$

$$o_0 = i_0 \cdot w_{00} + i_1 \cdot w_{01}$$

$$o_1 = i_0 \cdot w_{10} + i_1 \cdot w_{11}$$

$$o_2 = i_0 \cdot w_{20} + i_1 \cdot w_{21}$$

# Matrix Multiplication (Dot Product)

$$\begin{bmatrix} i_0 & i_1 \end{bmatrix} \times \begin{bmatrix} w_{00} & w_{10} & w_{20} \\ w_{01} & w_{11} & w_{21} \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} o_0 & o_1 & o_2 \end{bmatrix}$$

$$o_0 = i_0 \cdot w_{00} + i_1 \cdot w_{01}$$

$$o_1 = i_0 \cdot w_{10} + i_1 \cdot w_{11}$$

$$o_2 = i_0 \cdot w_{20} + i_1 \cdot w_{21}$$

# Matrix Multiplication (Dot Product)

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} =$$

$$\left[ \underbrace{0.1 \cdot 1}_{\text{row} \cdot \text{column}} + 0.2 \cdot 4, \quad 0.1 \cdot 2 + 0.2 \cdot 5, \right. \\ \left. 0.1 \cdot 3 + 0.2 \cdot 6 \right]$$

$$\begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}$$

# Matrix Multiplication (Dot Product)

inputs

$$\begin{bmatrix} 0.1 \\ \underline{0.2} \end{bmatrix} \times \begin{bmatrix} 1 & \overset{\text{weights}}{2} & 3 \\ 4 & 5 & 6 \end{bmatrix} =$$

$$= \begin{bmatrix} (0.1 \times 1 + 0.2 \times 4) & (0.1 \times 2 + 0.2 \times 5) & (0.1 \times 3 + 0.2 \times 6) \end{bmatrix}$$

(Answer)

$$= \begin{bmatrix} \underline{0.9} \\ \underline{1.2} \\ \underline{1.5} \end{bmatrix}$$

## Alternative Dot Computations

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}$$

$$\begin{array}{r} 0.1 \cdot 1 & 0.1 \cdot 2 & 0.1 \cdot 3 \\ + 0.2 \cdot 4 & + 0.2 \cdot 5 & + 0.2 \cdot 6 \\ \hline 0.9 & 1.2 & 1.5 \end{array}$$

$$\begin{array}{r} [0.1 \quad 0.2 \quad 0.3] \\ + [0.8 \quad 1.0 \quad 1.2] \\ \hline 0.9 \quad 1.2 \quad 1.5 \end{array}$$

## Alternative Dot Computations

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}$$

$$0.1 \cdot 1 \quad 0.1 \cdot 2 \quad 0.1 \cdot 3 = \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix}$$

$$0.2 \cdot 4 \quad 0.2 \cdot 5 \quad 0.2 \cdot 6 = \begin{bmatrix} 0.8 & 1.0 & 1.2 \end{bmatrix}$$

(input @ a time?)

$$\begin{array}{r} \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \\ + \begin{bmatrix} 0.8 & 1.0 & 1.2 \end{bmatrix} \\ \hline \begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix} \end{array}$$

## Alternative Dot Computations

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}$$

Diagram illustrating the computation of the dot product using the Multiply-Accumulate (MAC) operation:

The input vectors are:  
Row vector:  $0.1 \cdot 1 + 0$ ,  $0.1 \cdot 2 + 0$ ,  $0.1 \cdot 3 + 0$   
Column vector:  $0$ ,  $0$ ,  $0$

The result is:  
 $0$ ,  $0.1$ ,  $0.2$ ,  $0.3$

The final result is:  
 $0.9$ ,  $1.2$ ,  $1.5$

Annotations:  
- The first row of the column vector is labeled  $0.1 \cdot 1 + 0$ ,  $0.1 \cdot 2 + 0$ ,  $0.1 \cdot 3 + 0$ .  
- The second row of the column vector is labeled  $0$ ,  $0$ ,  $0$ .  
- The third row of the column vector is labeled  $0.1$ ,  $0.2$ ,  $0.3$ .  
- A bracket under the first three elements of the result vector is labeled  $0.2 \cdot 4 + 0.1$ ,  $0.2 \cdot 5 + 0.2$ ,  $0.2 \cdot 6 + 0.3$ .  
- The text "Multiply-Accumulate (MAC)" is written below the first three elements of the result vector.

# Python Time

inputs

$[0.1 \ 0.2 \ 0.3] \times \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = [h_1 \ h_2 \ h_3 \ h_4]$

weights

```
weights = np.array( [[1,2,3,4],[5,6,7,8],[9,10,11,12]], dtype=np.float32)
inputs = np.array([[0.1,0.2,0.3]], dtype=np.float32)
outputs = np.dot(inputs, weights)
```

Input	Weights	Output
[0.1 0.2 0.3]	[1. 2. 3. 4.] [5. 6. 7. 8.] [ 9. 10. 11. 12.]	= [3.8000002 4.4 5. 5.6000004]

Input [[0.1 0.2 0.3]] .	Weights [1. 2. 3. 4.] [5. 6. 7. 8.] [ 9. 10. 11. 12.]	Output = [3.8000002 4.4	5.	5.6000004]
----------------------------	--	----------------------------	----	------------

# Alternative Dot

```
# how its done in dot.sv
def pydot(inputs,weights):
    inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

Inputs (Shape):  
(1, 3)  
Output (Shape):  
(1, 4)  
Weights (Shape):  
(3, 4)



Input [[0.1 0.2 0.3]] .	Weights [1. 2. 3. 4.] [5. 6. 7. 8.] [ 9. 10. 11. 12.]	Output = [3.8000002 4.4 5. 5.6000004]	
----------------------------	--	---	--

# Alternative Dot

# how its done in dot.sv

```
def pydot(inputs,weights):
    → inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        ↗ for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

$$\text{outs} = [0 \quad 0 \quad 0 \quad 0]$$

$$i=0, j=0$$

$$\text{outs} = [0.1 \quad 0 \quad 0 \quad 0]$$

$$i=0, j=1$$

$$\text{outs} = [0.1 \quad 0 + 0.1 \cdot 2 \quad 0 \quad 0]$$

$$= [0.1 \quad 0.2 \quad 0 \quad 0]$$

$i=0, j=2$

$$\text{outs} = [0.1 \quad 0.2 \quad 0.3 \quad 0]$$

$i=0, j=3$

$$\text{outs} = [0.1 \quad 0.2 \quad 0.3 \quad 0.4]$$

$i=1$

$$\text{outs} = 0.1 + 0.2 \cdot 5, \quad 0.2 + 0.2 \cdot 6, \quad 0.3 + 0.2 \cdot 7$$

$$= [1.1 \quad 1.4 \quad 1.7 \quad 2.0]$$

Inputs (Shape):  
(1, 3)  
Output (Shape):  
(1, 4)  
Weights (Shape):  
(3, 4)

$i=2$

$$\text{outs} = [3.8 \quad 4.4 \quad 5.5 \quad 5.6]$$

# Floating-Point Multiply-Accumulate (FMAC)

- Math:  $a * b + c$  ← 8 cycles

$$\begin{array}{r} 0.1 \\ \times 1.0 \\ \hline 0.1 \\ + 0.0 \\ \hline \rightarrow 0.0 \end{array}$$

8 cycles

# Floating-Point math takes 8 cycles.

- Floating-Point is complicated.
- How do we work around an 8 cycle latency?
- Pipelining!

# Pipelining

- FMAC takes 8 cycles for 1 value
- But can accept a new value every cycle.

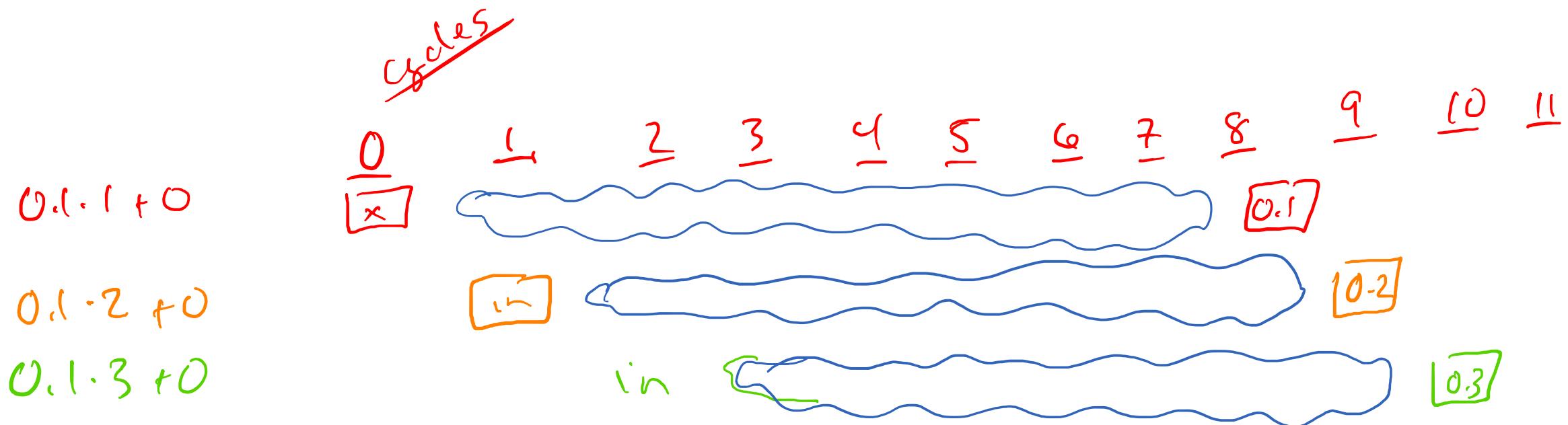
cycles for  
value |

→ • Latency: 8 cycles / value ↩

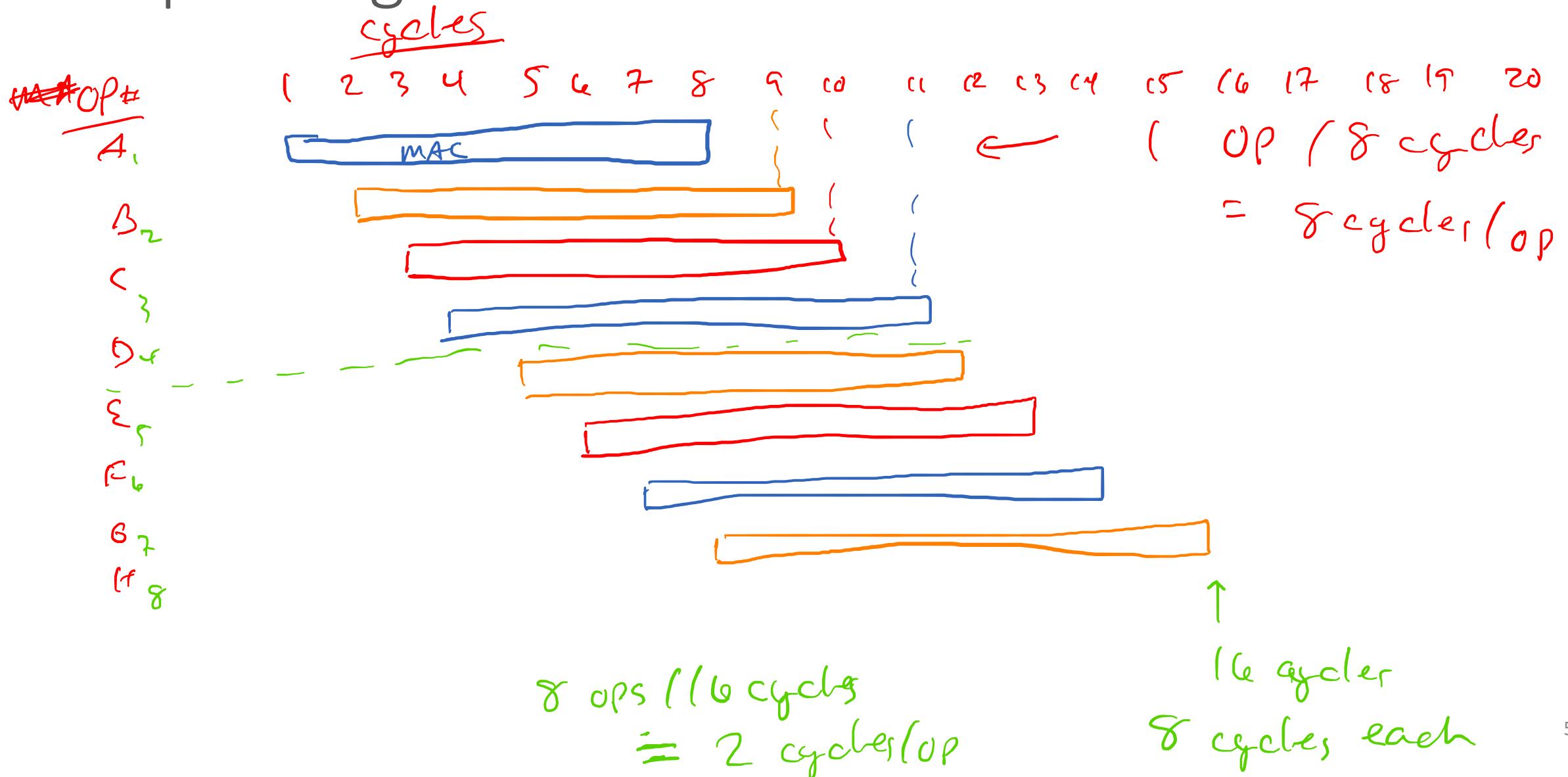
↙ • Throughput: 1 value / cycle

values per cycle

# Pipelining



# Pipelining



# Next Time: More Hardware Parallelism

⇒ more pipelining

⇒ parallel hardware

parallelizing dot product.

# 18: Hardware Acceleration I

Engr 315: Hardware / Software Codesign

Andrew Lukefahr

*Indiana University*

