

Test

18: Hardware Acceleration I

Engr 315: Hardware / Software Codesign

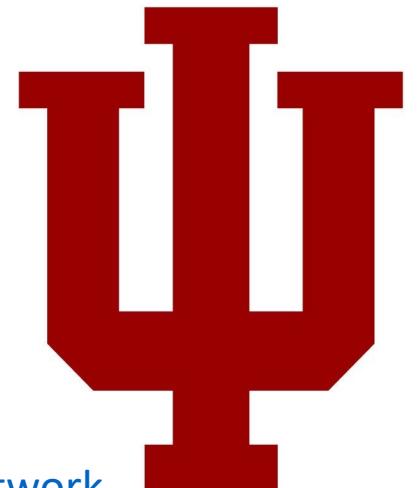
Andrew Lukefahr

Indiana University

Some material taken from:

https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural_network

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



Announcements

- P6 out
 - AG not up yet?
- No Class Wednesday, Oct 26th.
- New SD Cards coming... *for P7*
- Exam-Nov 7th

P6: 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

```
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)
```

must use special allocate

```
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]

P7 – DMA from C

A screenshot of a search results page. The search bar at the top contains the text "xilinx dma ip". Below the search bar are navigation links: "All" (highlighted in blue), "News", "Shopping", "Images", "Videos", "More", and "Tools". A message below the links states "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 is a PDF file from Jun 14, 2019. The document title is purple, and the link text is "https://www.xilinx.com › support › pg021_axi_dma PDF".

xilinx dma ip

All News Shopping Images Videos More Tools

About 294,000 results (0.50 seconds)

[https://www.xilinx.com › support › pg021_axi_dma PDF](https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf)

AXI DMA v7.1 LogiCORE IP Product Guide - Xilinx

Jun 14, 2019 — The AXI Direct Memory Access (AXI DMA) IP core provides high-bandwidth direct memory access between the AXI4 memory mapped and AXI4-Stream IP ...
97 pages

https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf

P7 – DMA from C

Programming Sequence

Direct Register Mode (Simple DMA)

P7 – 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

↳ while loop

P7 – 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 (No Interrupts)

P7 – 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.

P7 – 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*

P7 – 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

↳ while loop

P8+ Accelerate Machine Learning

- Goal: Accelerate reference neural network
- Harder, more open-ended projects

MV
ZZ1, ZZ2
Neural Nets

Accelerate Machine Learning

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

Grade: Based on overall Speedup!

Project 8: 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

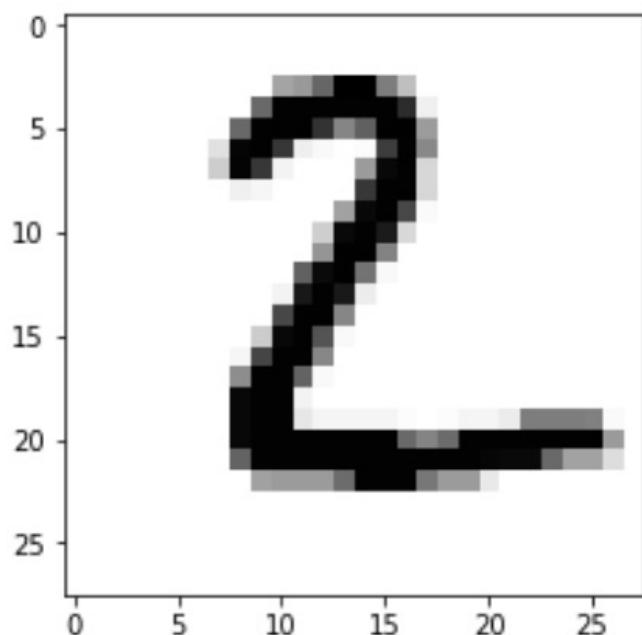
MNIST

Simple Neural Network

=====

Index: 0

Image:



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

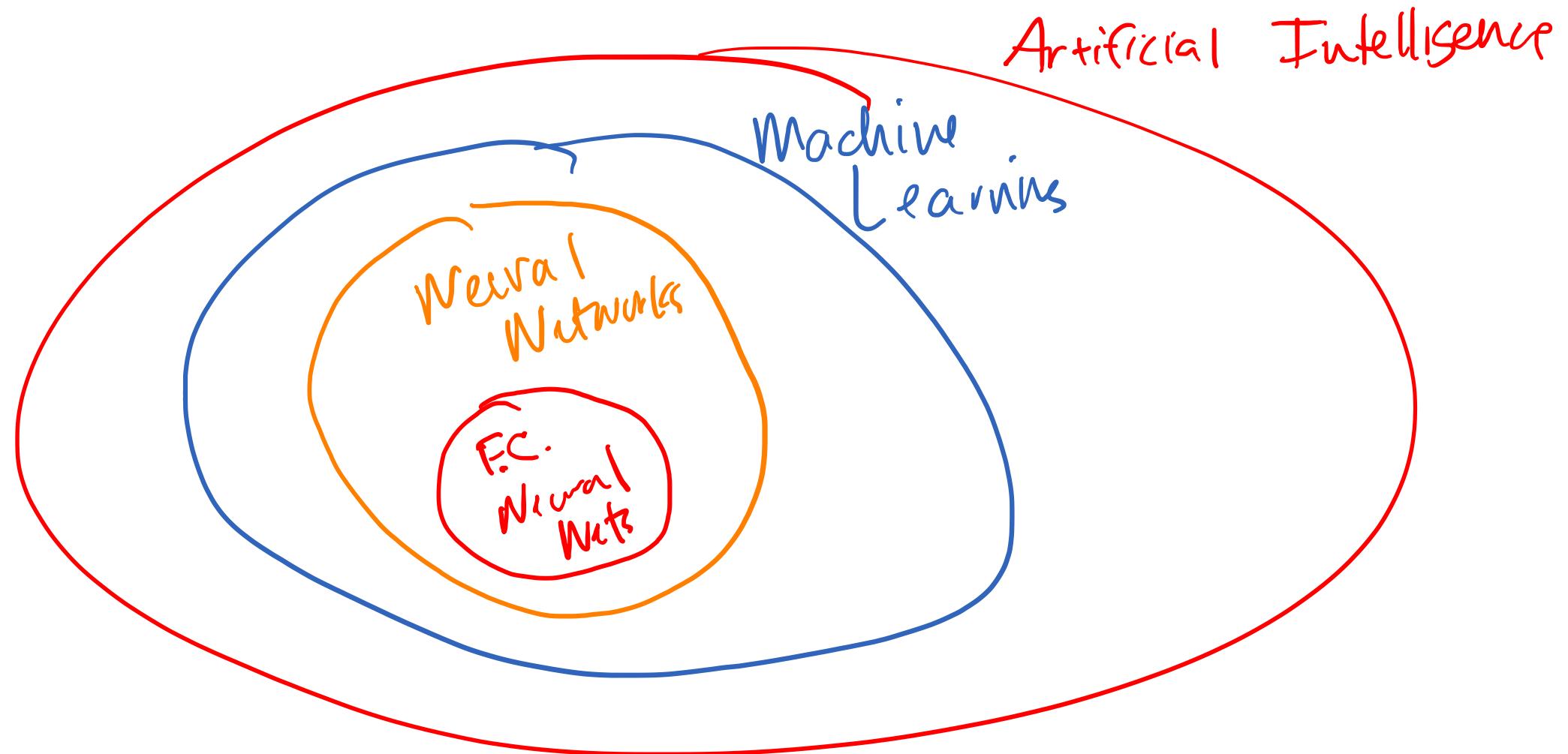
→ ML Classification Result: 2

Real Value: 2

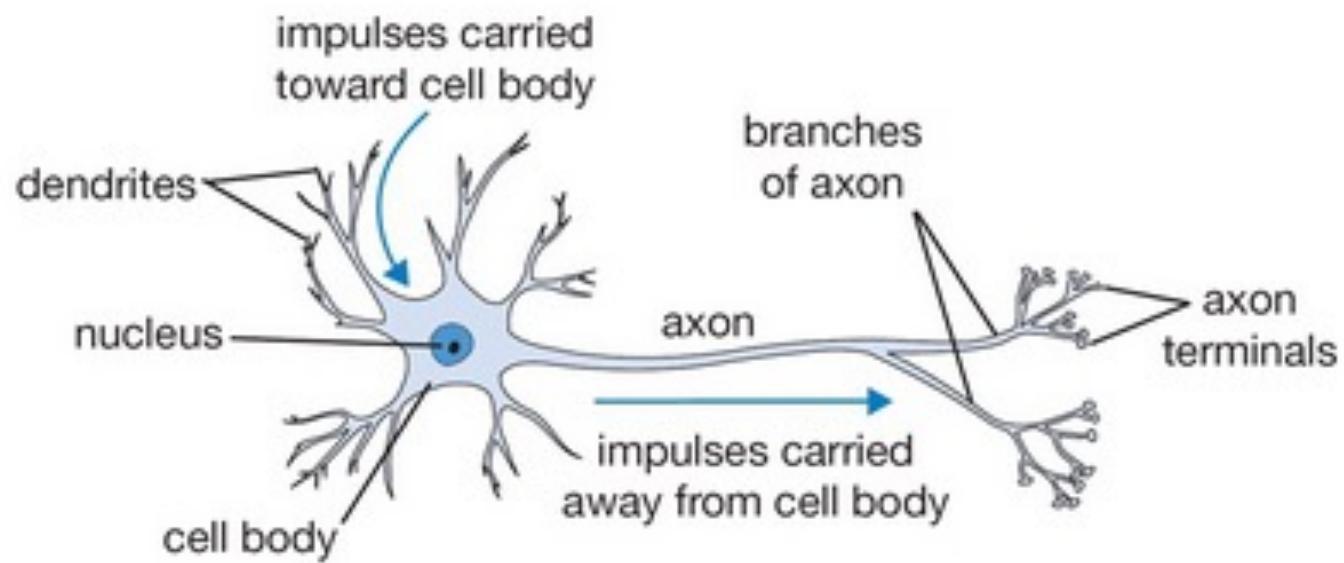
Correct Result: True

=====

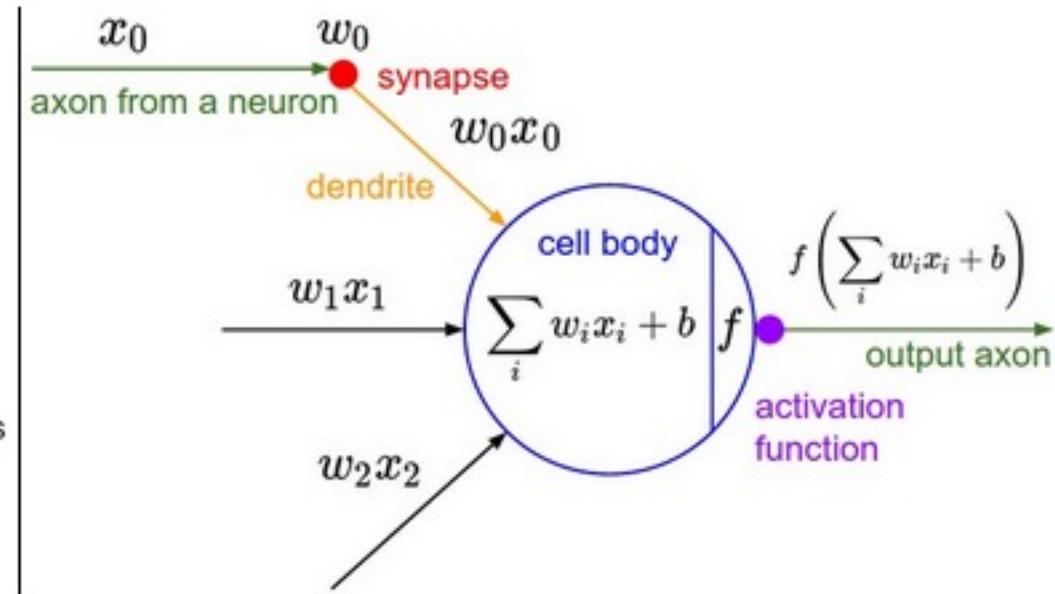
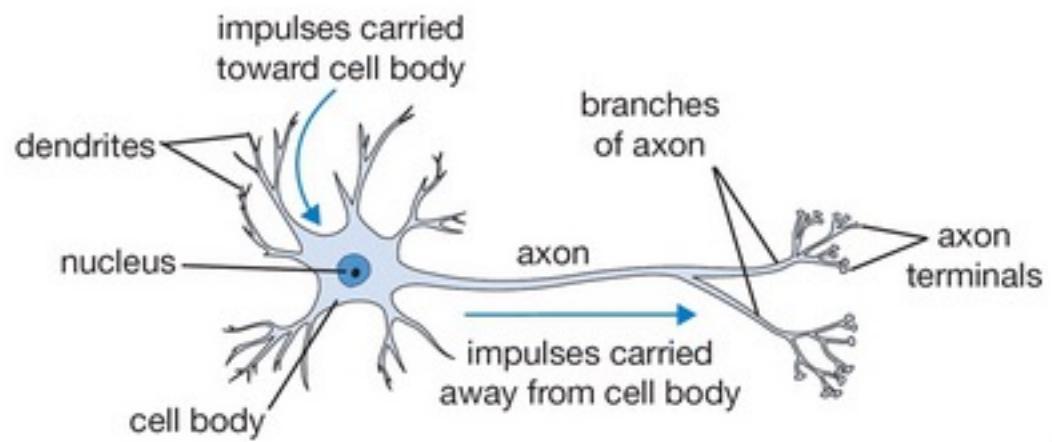
Fully-Connected Neural Networks



The Biological Neuron

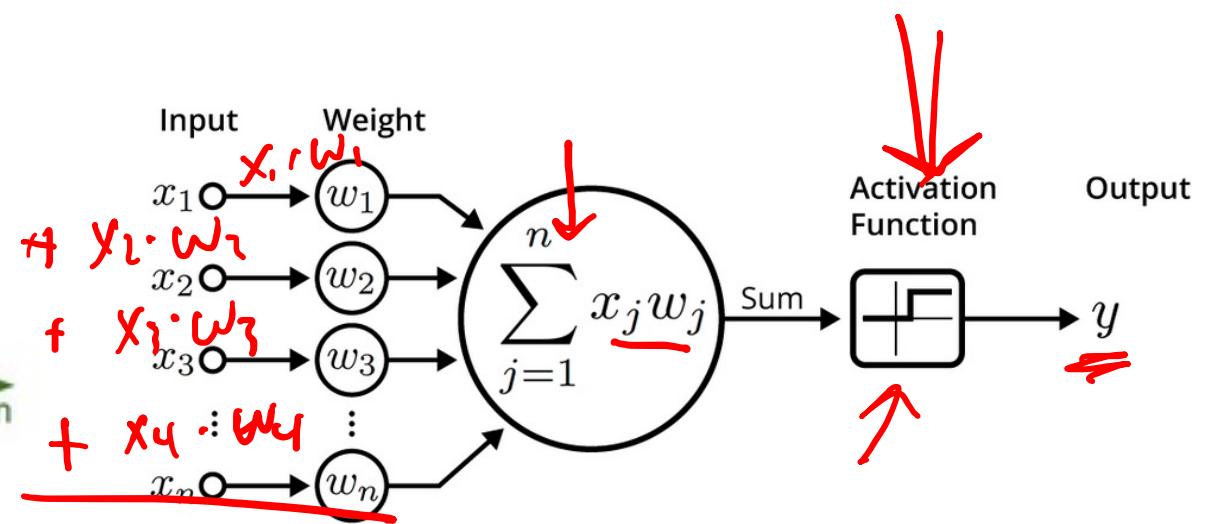
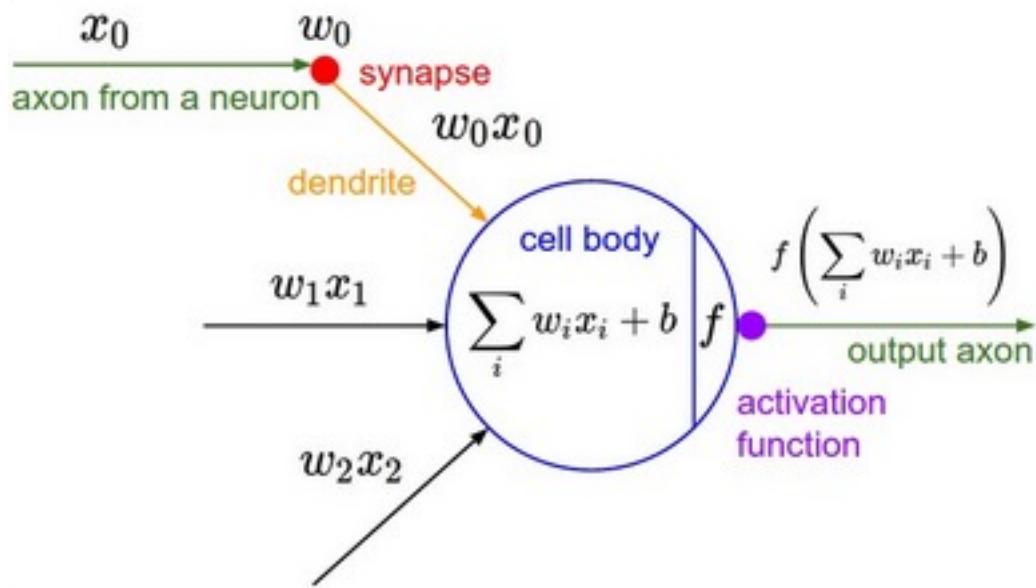
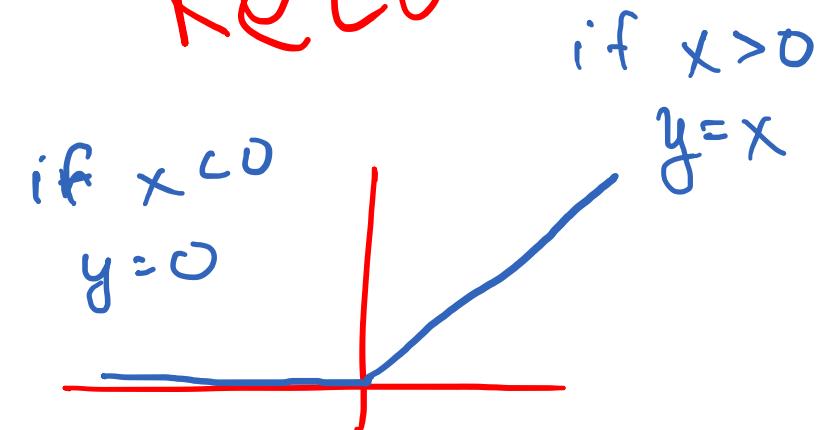


The Math Neuron



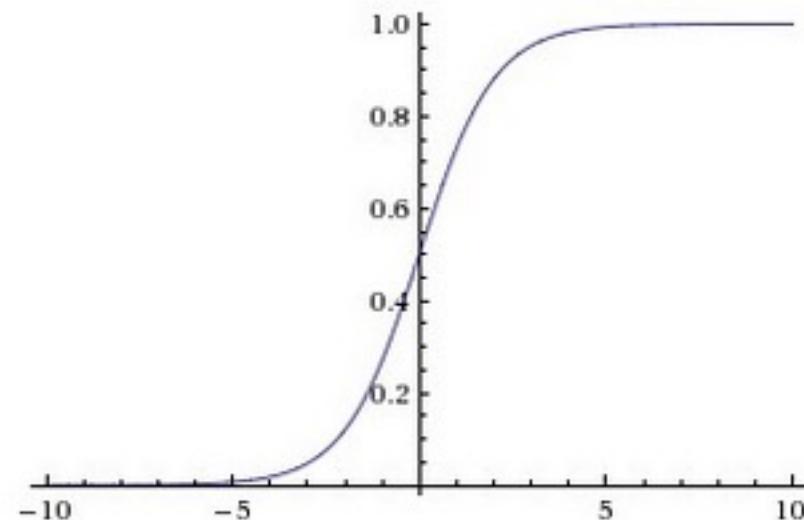
The Math Neuron

ReLU



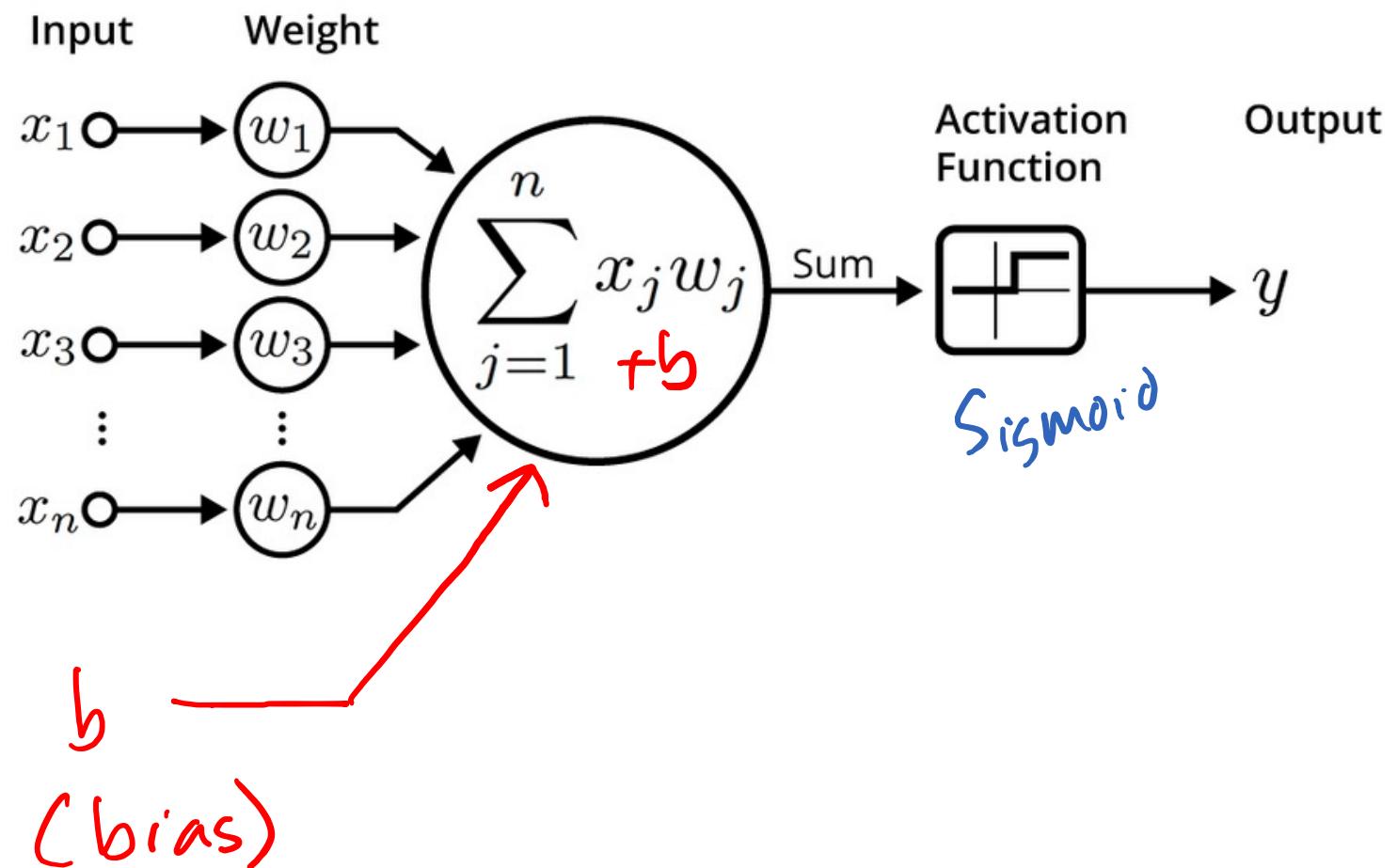
An illustration of an artificial neuron. Source: Becoming Human.

Activation Function: Sigmoid



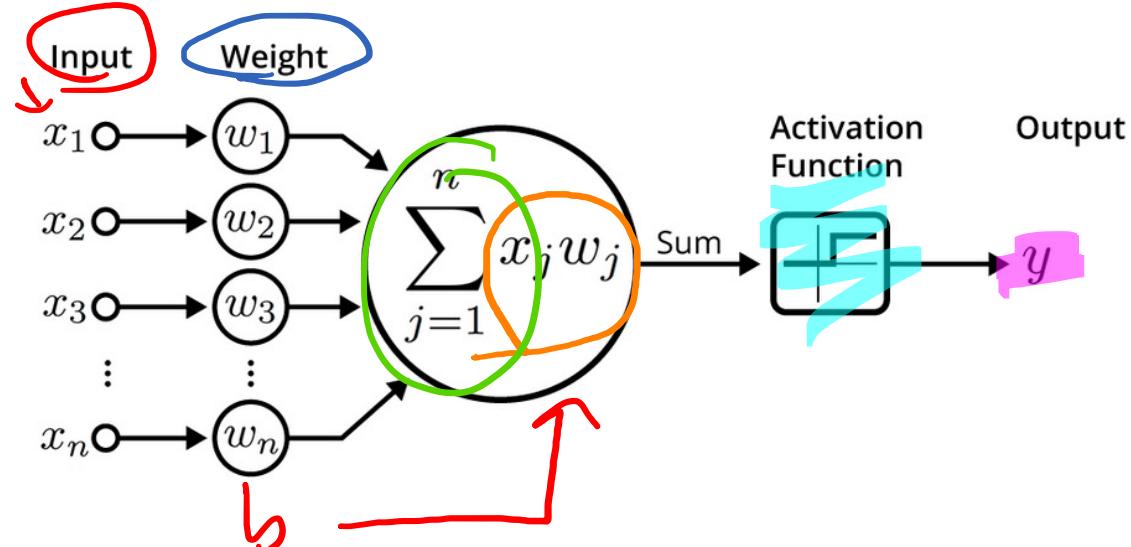
$$\sigma(x) = 1/(1+e^{-x})$$

Adding Bias



Fixme. odd profile

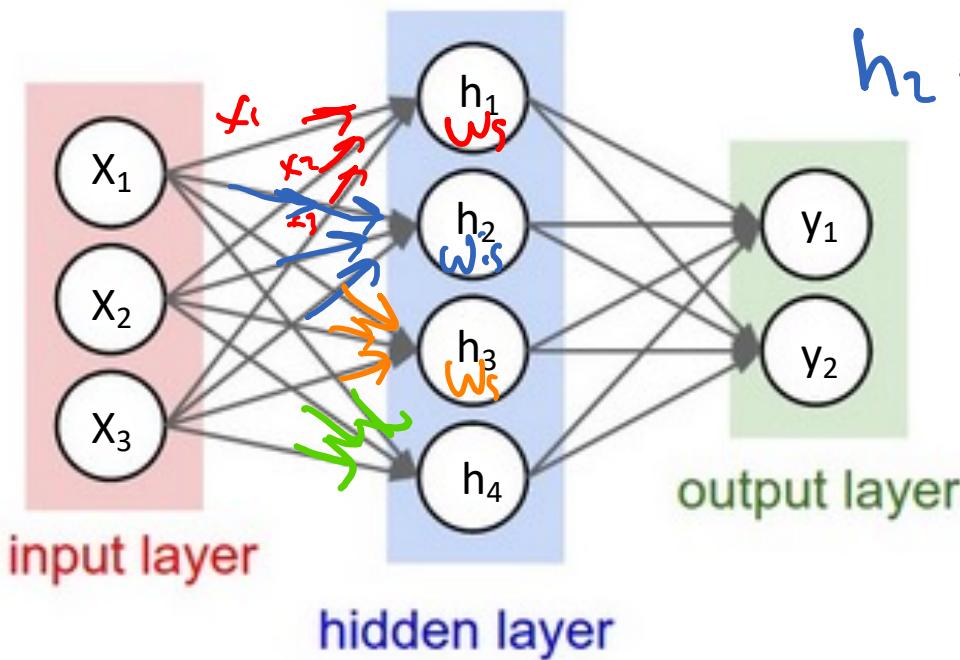
Python Neuron



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

Profile code!

Neuron Networks



$$h_1 = f(x_1 \cdot w_{11} + x_2 \cdot w_{12} + x_3 \cdot w_{13} + b_1)$$

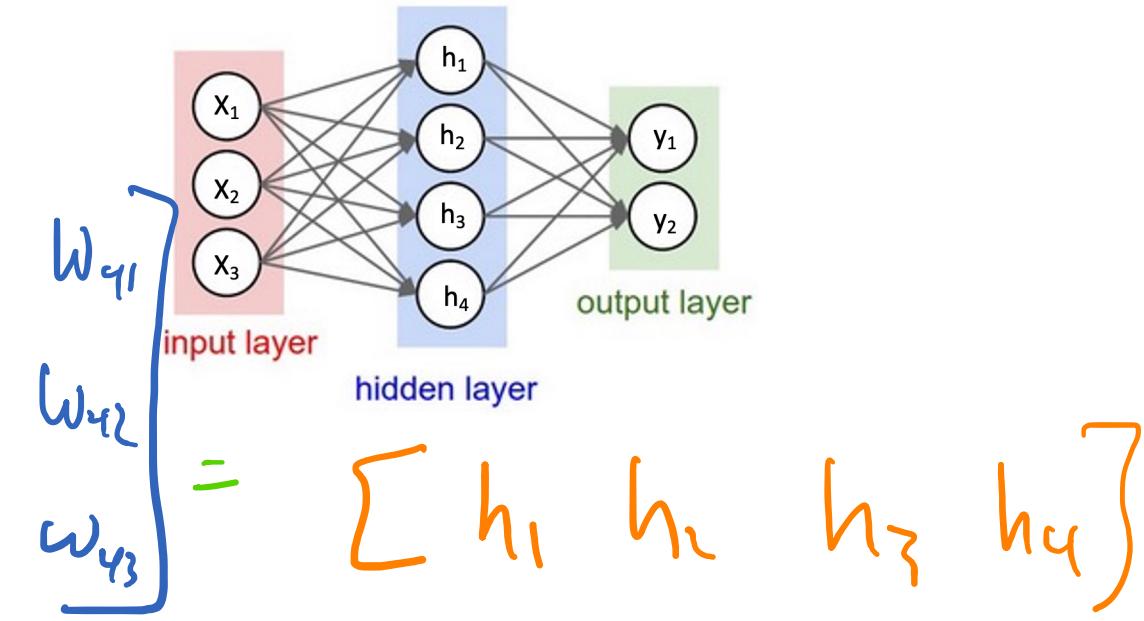
$$h_2 = f(x_1 \cdot w_{21} + x_2 \cdot w_{22} + x_3 \cdot w_{23} + b_2)$$

$$\begin{aligned} h_3 = f(& x_1 \cdot w_{31} \\ & + x_2 \cdot w_{32} \\ & + x_3 \cdot w_{33} \\ & + b_3) \end{aligned}$$

Input Layer: Just input values
Output Layers: No Activation Function

Neuron Networks

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



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

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

⋮
⋮
⋮

Let's focus on just the Matrix for now.

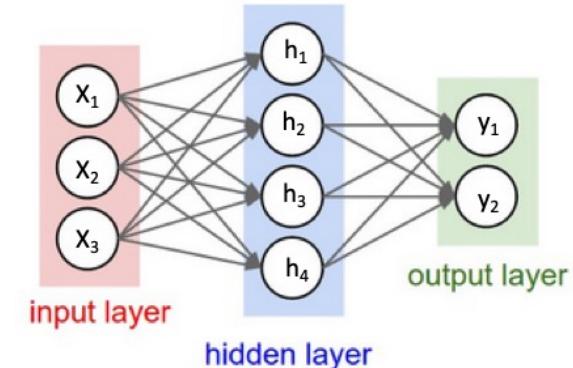
$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \cdot \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix}$$

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

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

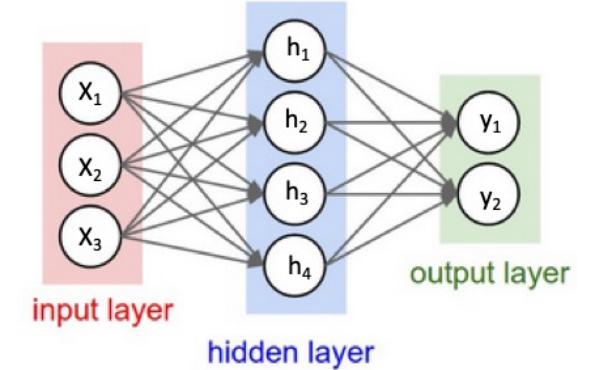
$$h_3 = x_1 \cdot w_{31} + x_2 \cdot w_{32} + x_3 \cdot w_{33}$$

h_4 is similar..



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

Matrix Multiplication (Dot Product)



Matrix Multiplication (Dot Product)

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

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

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

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

Matrix Multiplication (Dot Product)

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

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

Partial result

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

$$\begin{array}{r} 0.2 \cdot 4 \\ 0.2 \cdot 5 \\ 0.2 \cdot 6 \\ \hline + \end{array} \begin{bmatrix} 0.8 & 1.0 & 1.2 \end{bmatrix} \overbrace{\hphantom{0.8 1.0 1.2}}^{\begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}}$$

Alternative Dot Computations

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

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

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

(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

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

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.

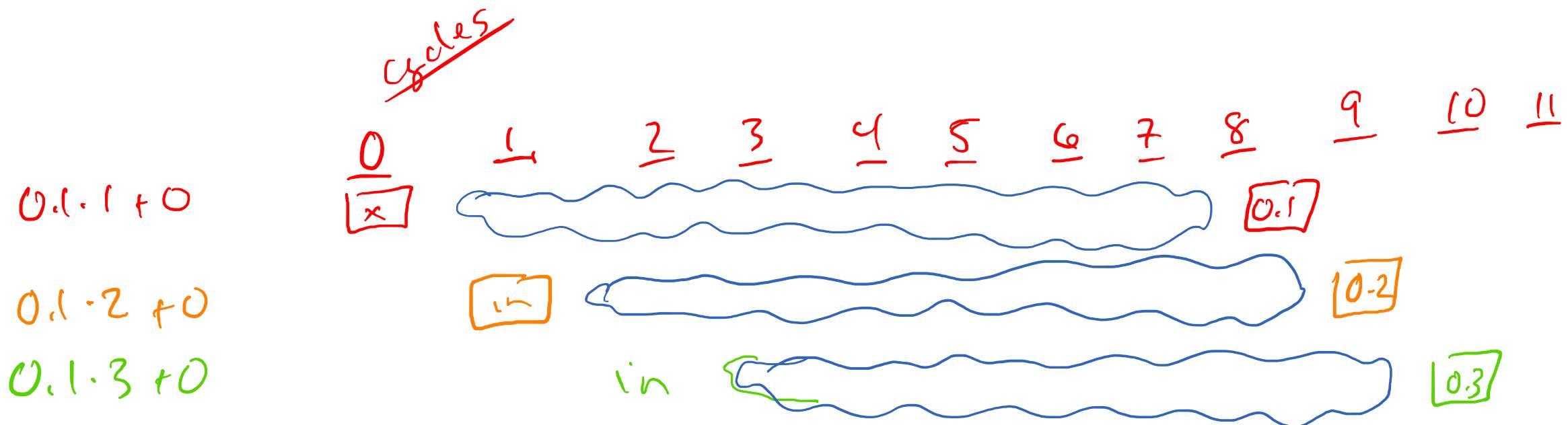
cycles for
value |

→ • Latency: 8 cycles / value ↩

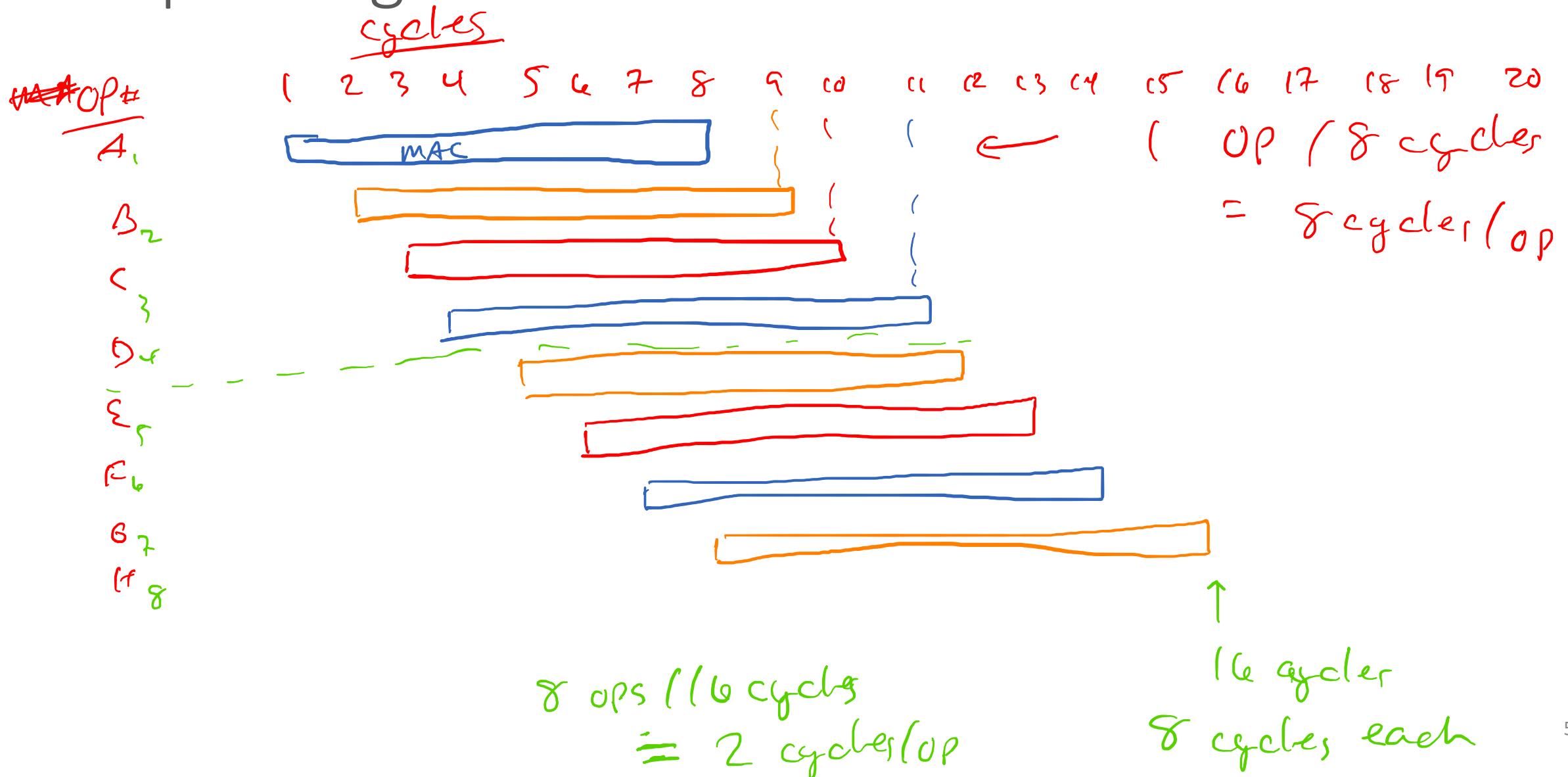
↙ • Throughput: 1 value / cycle

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

