CS 331
Computer Vision

10 / 29 / 2026

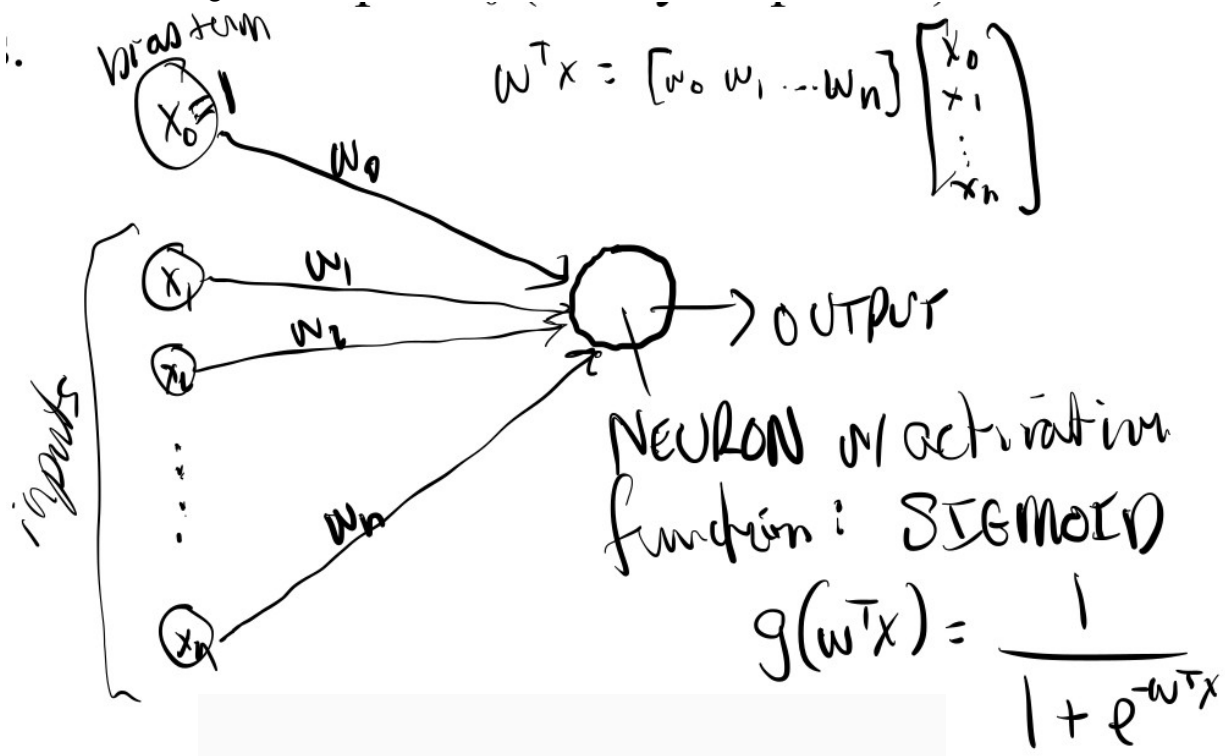
Instructor: Michael Eckmann

Topics

- Questions / Comments?
- Announcement: In-class exam on Thursday November 5. Topics list to be sent out by tomorrow.
- Quick review of Neural Network parts and terminology
- More about Neural Networks

Artificial Neuron

- **weights** on the links
- how the neuron uses the **inputs** and weights to compute an output via an ***activation function***.
- First input x_0 (always equals 1) is called the bias.



Topics for Neural Networks

Neural networks

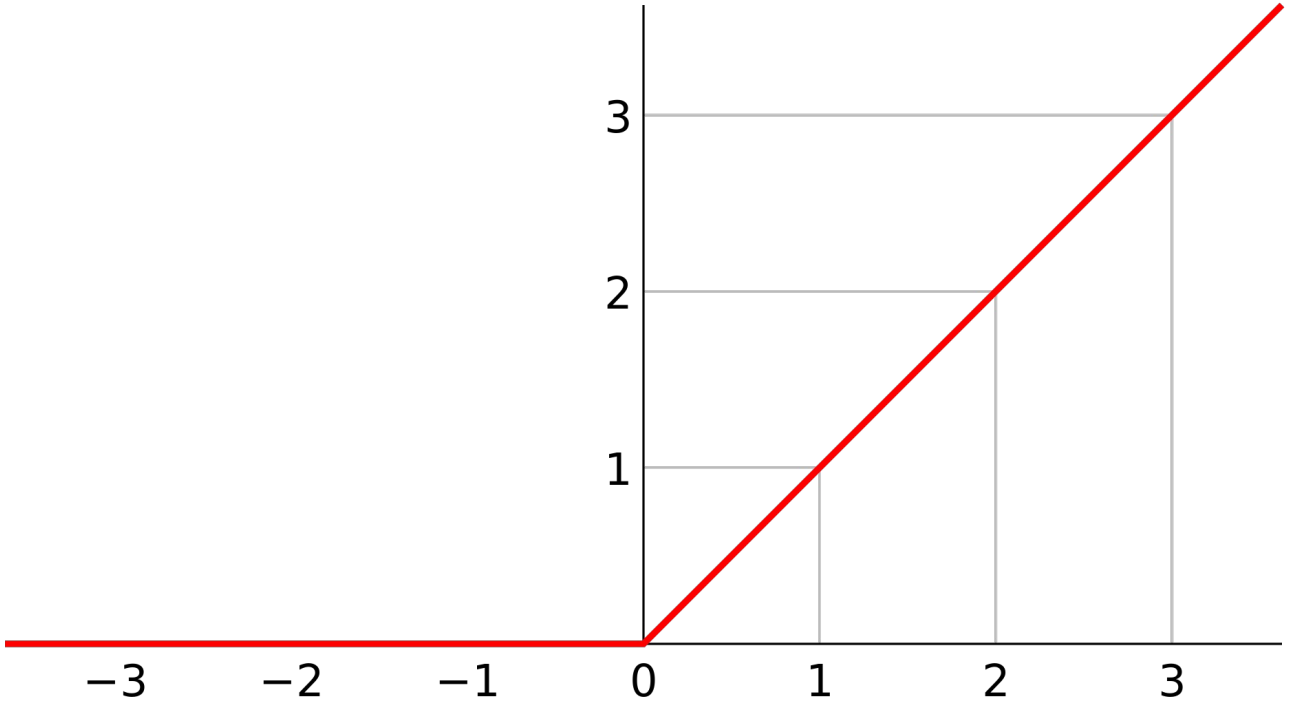
- Review: Input layer, hidden layer(s), output layer, bias units, Sizes of weight matrices
- Introduce another activation function: ReLU
- Forward propagation and an example
- What happens if we create a Neural Network without using activation functions?
- Show how a Neural Network with ReLUs can approximate arbitrary functions

Layers and subscripts

- $W^{(j)}$ are the weights from layer j to layer j+1
- Layer numbers (j) will be superscripts
- Number of a unit within a layer will be subscript
- The bias term at each layer will not generally be shown and will be numbered with a subscript of 0
- For the weight matrices, the reason the number of columns is 1 more than the number of units in the layer is due to the bias unit.

ReLU activation function

- Another activation function that is widely used is the ReLU = Rectified Linear Unit
- It is the ramp function which is simply y = max(0,x)



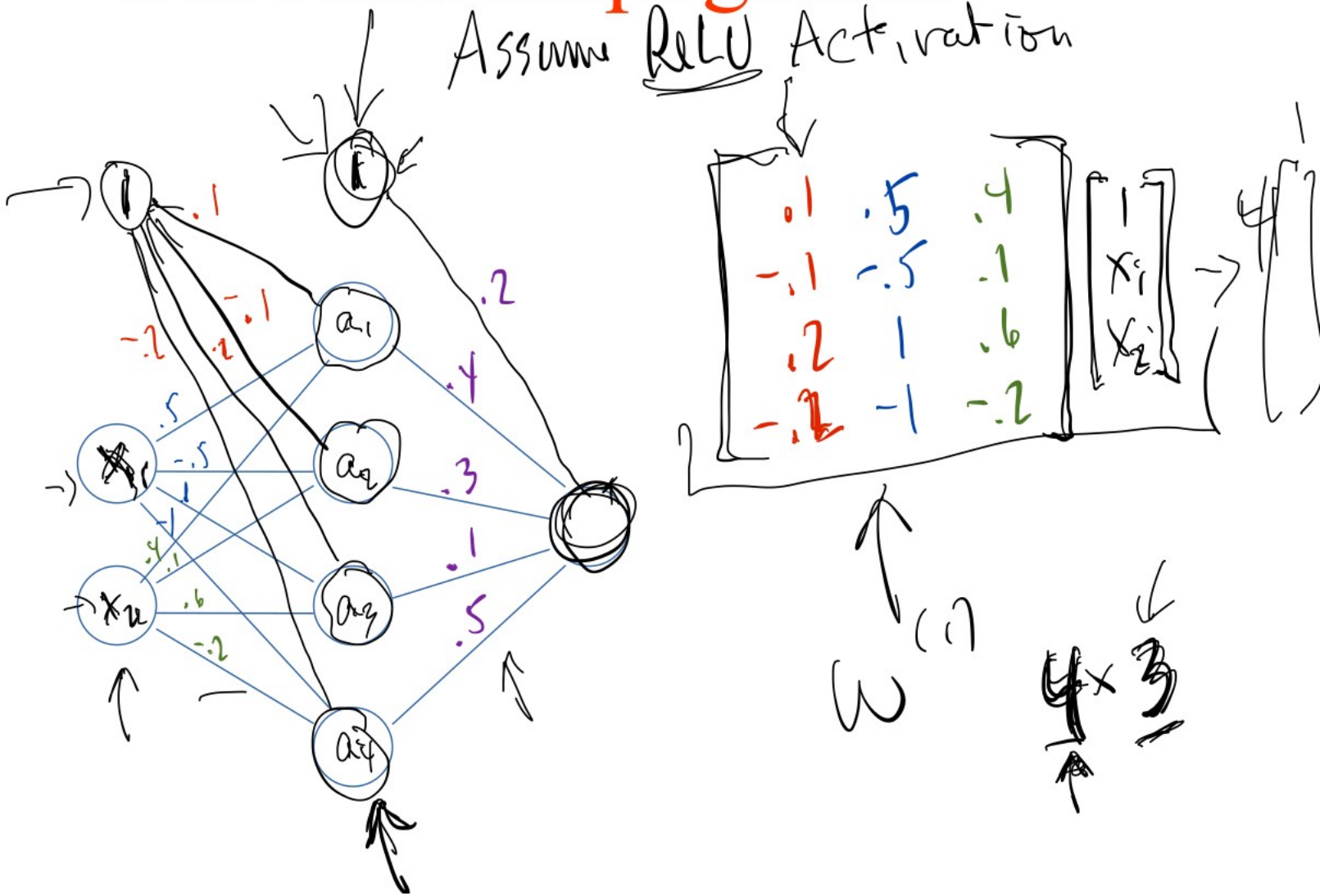
Forward Propogation

- Forward Propogation works in a forward manner from the input layer to the output layer.
- Between each layer, we compute the weight matrix times the previous layer as the input to our activation functions in the later layer. Each activation function takes the input and computes an output.
- The weights after a layer times the outputs of that layer act as inputs to the next layer. And so on.

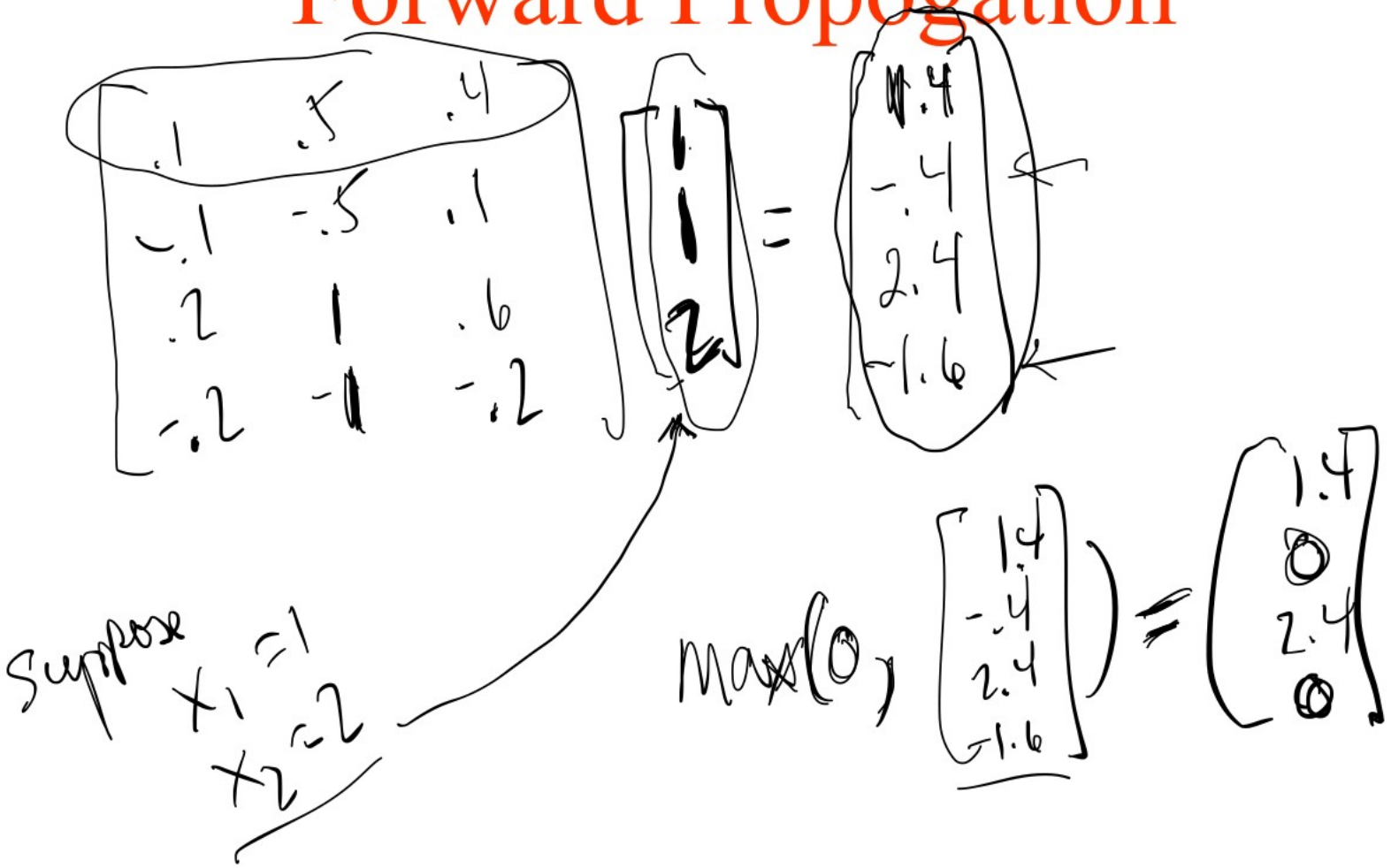
Forward Propogation

- The forward propogation should be implemented in a “vectorized” fashion, that is without loops, to do the processing. To do this, we use matrix multiplication and elementwise operations.

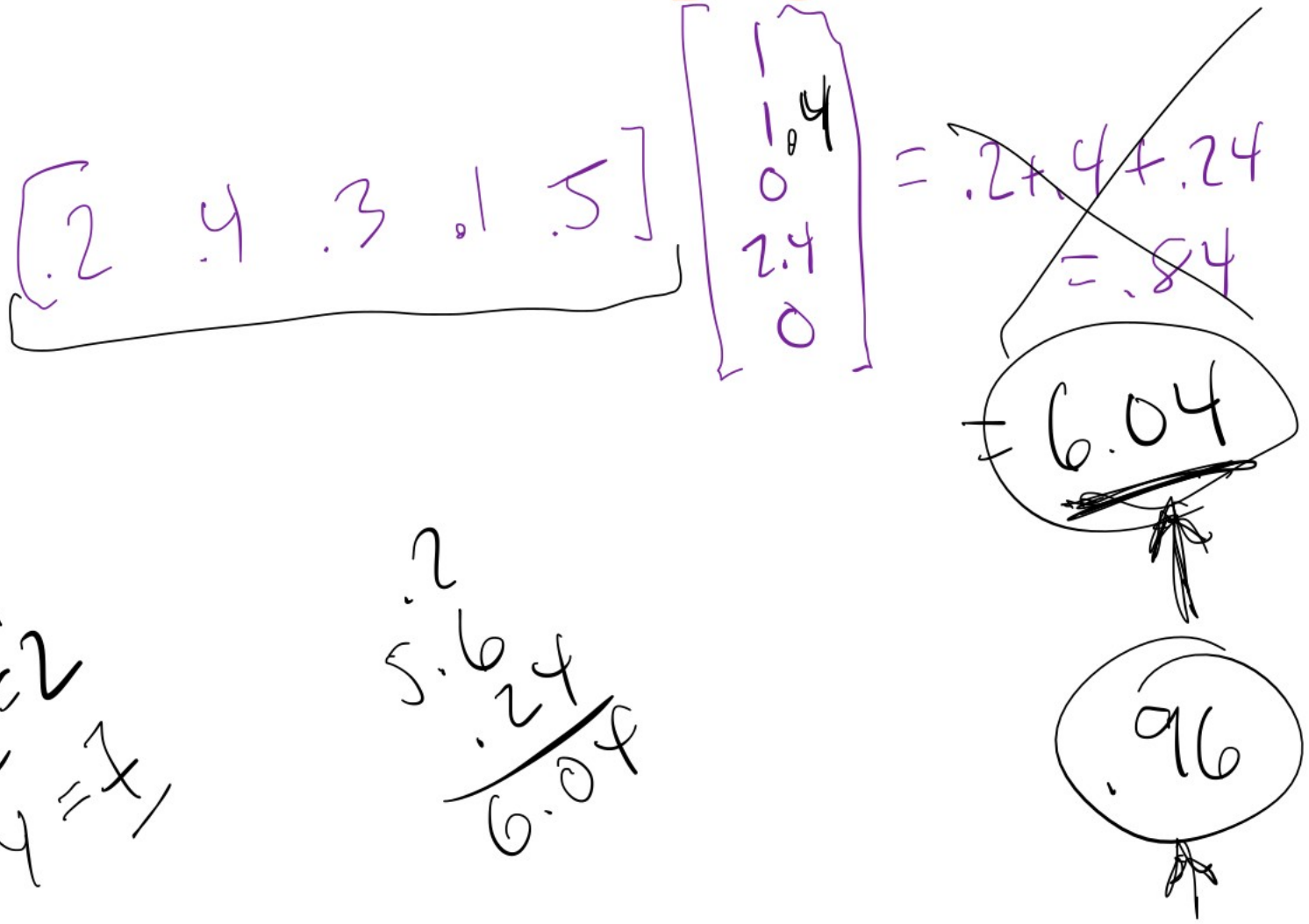
Forward Propogation



Forward Propogation



Forward Propogation



Neural Network without activation

- Now that we know how forward propagation works, suppose we have 1 hidden layer like before.
- With the ReLU activation we end up computing $W^{(1)}$ @ x which gets fed into ReLU at the hidden layer and results in

max(0, $W^{(1)}$ @ x).

- Forward propagation causes this result to get multiplied by the weight matrix in the next layer:

$$W^{(2)} @ \max(0, W^{(1)} @ x)$$

- Without ReLU we would have at the next layer the following:

$$W^{(2)} @ (W^{(1)} @ x)$$

- Note: @ is numpy matrix multiplication operator

Neural Network without activation

- Without ReLU we would have at the next layer the following:

$$W^{(2)}@W^{(1)}@x = W^{(21)}@x$$

  - where $W^{(21)} = W^{(2)}@W^{(1)}$

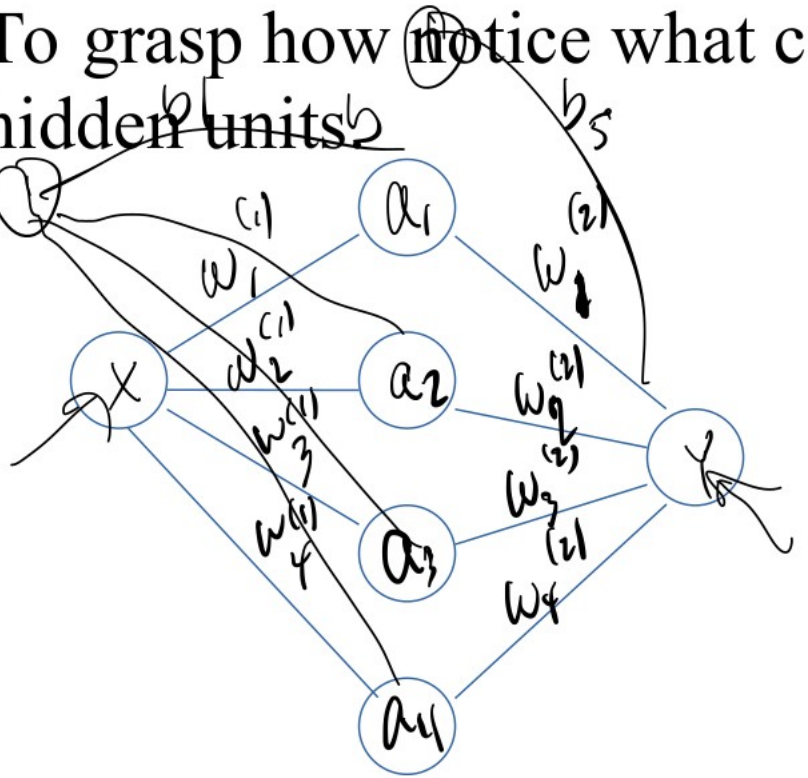
- The two matrices (one for each layer) can be replaced by one matrix. Think back to the linear regression equation for y hat. This is linear.
- The takeaway: the activation function provides non-linearity which is a critical part of the functioning of Neural Networks.

Universal Approximation

- Neural Networks can approximate a wide variety of functions (Universal Approximation Theorem)
- Let’s look at part of chapter 12 in a book online titled “Foundations of Computer Vision” by Torralba, Isoba, and Freeman.
- https://visionbook.mit.edu/neural_nets.html

Universal Approximation

- Neural Networks can approximate a wide variety of functions (see Universal Approximation Theorem)
- To grasp how notice what can be done with just 4 hidden units

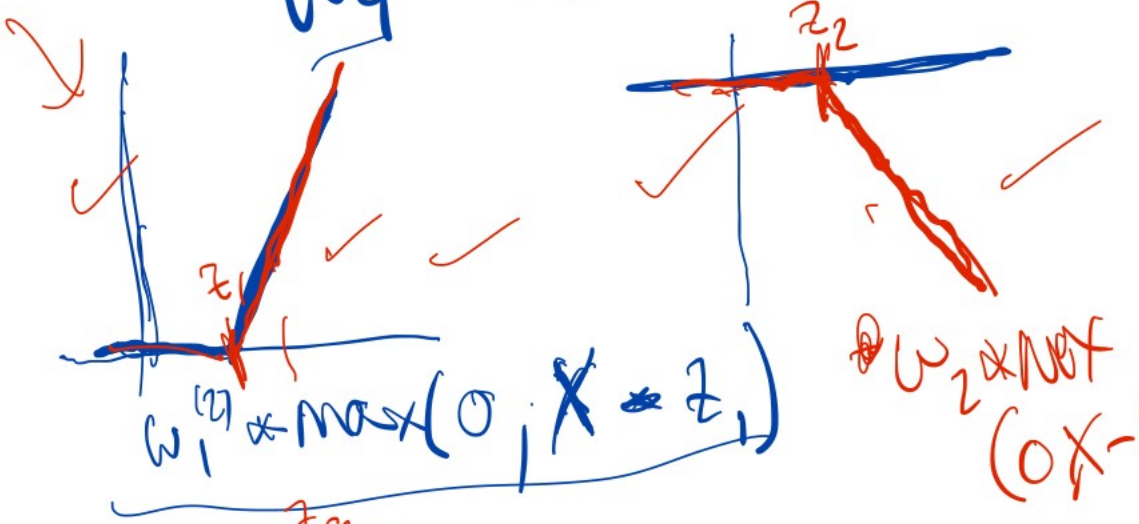


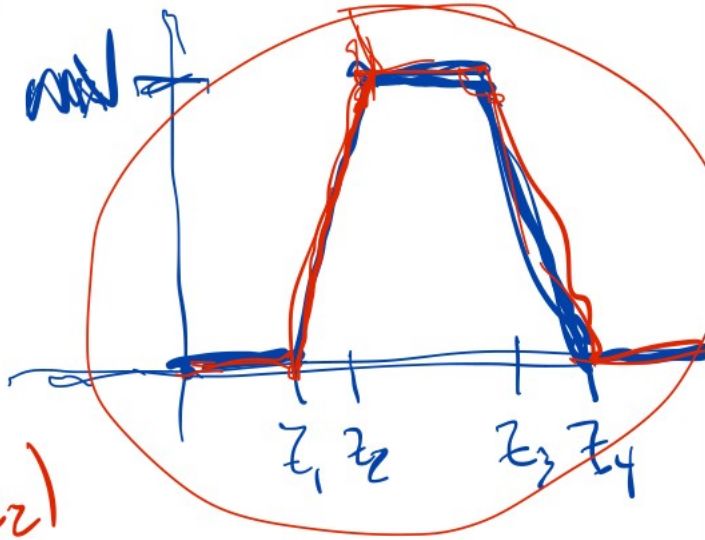
$$a_1 = \max(0, w_1^{(1)} x + b_1)$$
$$a_2 = \max(0, w_2^{(1)} x + b_2)$$
$$a_3 = \max(0, w_3^{(1)} x + b_3)$$
$$a_4 = \max(0, w_4^{(1)} x + b_4)$$

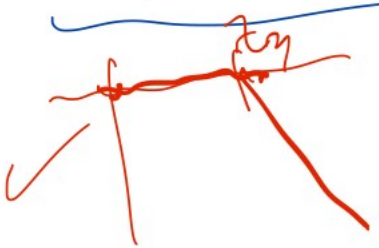
$$y = w_1^{(2)} a_1 + w_2^{(2)} a_2 + w_3^{(2)} a_3 + w_4^{(2)} a_4 + b_5$$

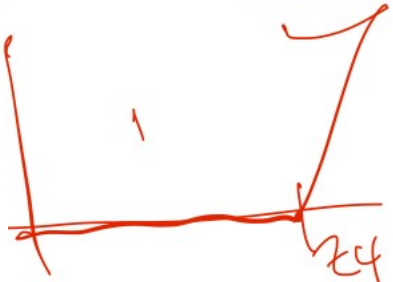
Universal Approximation

$$y = w_1^{(2)} * \max(0, w_1^{(1)} x + b1) + z_1$$
$$w_2^{(2)} * \max(0, w_2^{(1)} x + b2) +$$
$$w_3^{(2)} * \max(0, w_3^{(1)} x + b3) +$$
$$w_4^{(2)} * \max(0, w_4^{(1)} x + b4) + b5$$

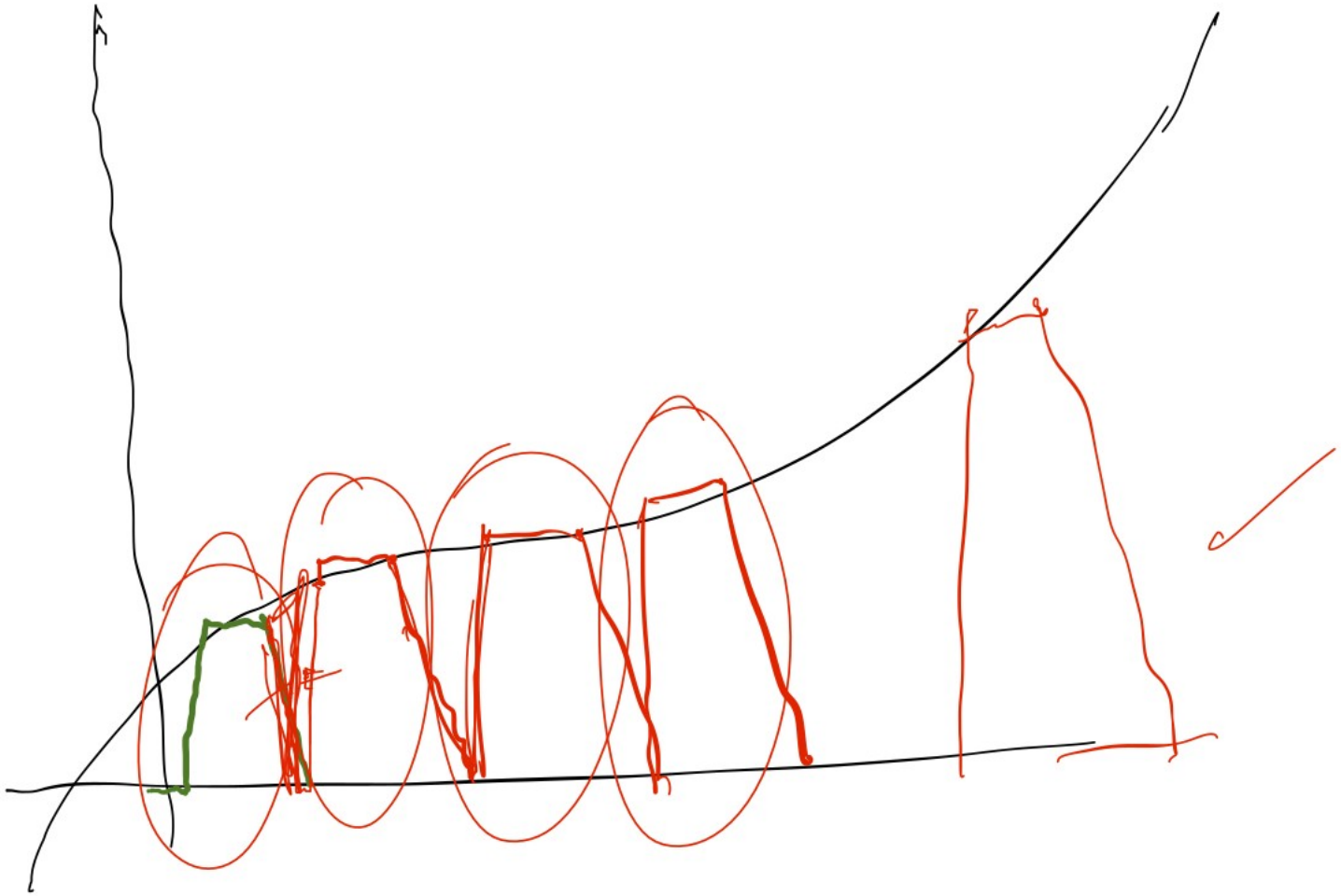








Universal Approximation



How a Neural Network learns

- The steps are:

Initialize weights randomly

For all training examples

    Forward Propagation to make prediction

    Backward Propagation to compute the gradient

    Gradient Descent to alter the weights

- We covered forward propagation already.

Next Topics

- Introduce Backward propagation (aka backprop), what it does for us and how
- Go through an example
- Show the example with forward prop, backprop and gradient descent in a spreadsheet
- Outline of the overall learning process again including where backprop fits into it
- Computational Graphs
- Larger Network w/ activation functions and backprop

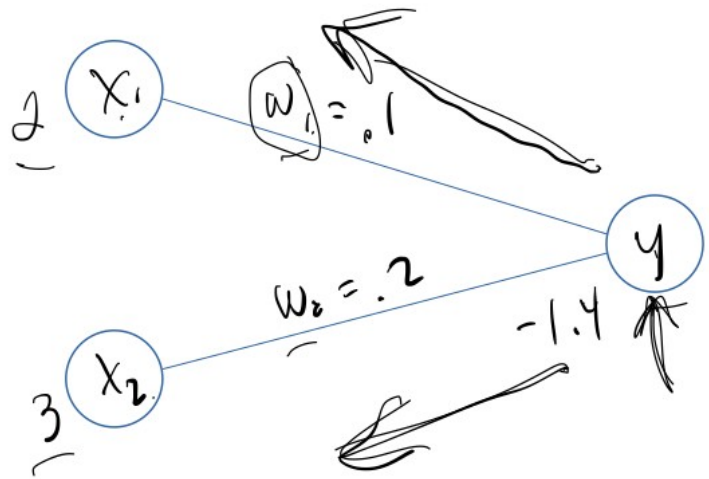
Backward Propogation

- The gradient could be computed analytically but would be inefficient for neural networks in general. **Backward Propogation**, simply backprop, is an efficient way to compute the gradient of a Neural Network. We will compute a partial derivative of the loss function with respect to each weight in the network and use these in gradient descent to update the weights.
- Backpropagation works by determining the partial derivatives at each layer starting at the last layer and going backwards.
- Think of each partial derivative with respect to a variable, say q, representing how much the loss function is affected by a change in variable q.

Backward Propogation

- Let’s go through an example of a simple network with 2 input units, 1 output unit, no activation function and only 1 training example.
- The loss function we’ll use is the familiar squared error (doesn’t have to be mean squared error b/c only 1 training example).

Backprop example



$3 \cdot -1.4 = -4.2$

$2 \cdot .1 + 3 \cdot .2 = .8$

x_1	x_2	Y
2	3	1.5

$y = w_1 x_1 + w_2 x_2$

$COST = (y - y_{train})^2$

$\frac{\partial COST}{\partial y} = 2(y - y_{train})$

$\frac{\partial COST}{\partial w_2} = \frac{\partial y}{\partial w_2} \cdot \frac{\partial COST}{\partial y} = 2 \cdot (.8 - 1.5) = -1.4$

$\frac{\partial COST}{\partial w_1} = \frac{\partial y}{\partial w_1} \cdot \frac{\partial COST}{\partial y} = -1.4 \cdot -1.4 = -2.8$

$\frac{\partial y}{\partial w_1} = x_1 \quad 2 \times -1.4$