Introduction To Neural Networks

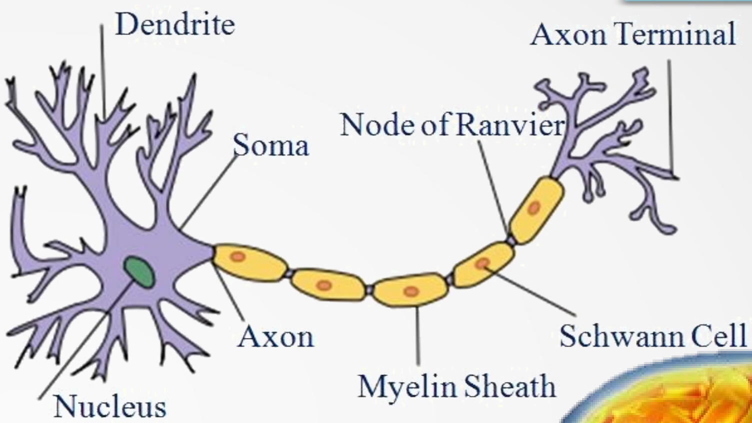
UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Learning Objectives

After completing this lecture, you will be able to:-

- Describe the biological motivation for neural networks
- Explain the link between artificial neurons and logistic regression
- Calculate a single neural forward-pass from input to output
- Explain the backpropagation process as applied to standard sigmoidal transfer functions
- Describe alternative activation functions and their potential benefits

Motivation for Neural Networks



Neuron

- Very simple structure
  - Dendrites receive signals
  - Soma processes signals
  - Axon emits signals

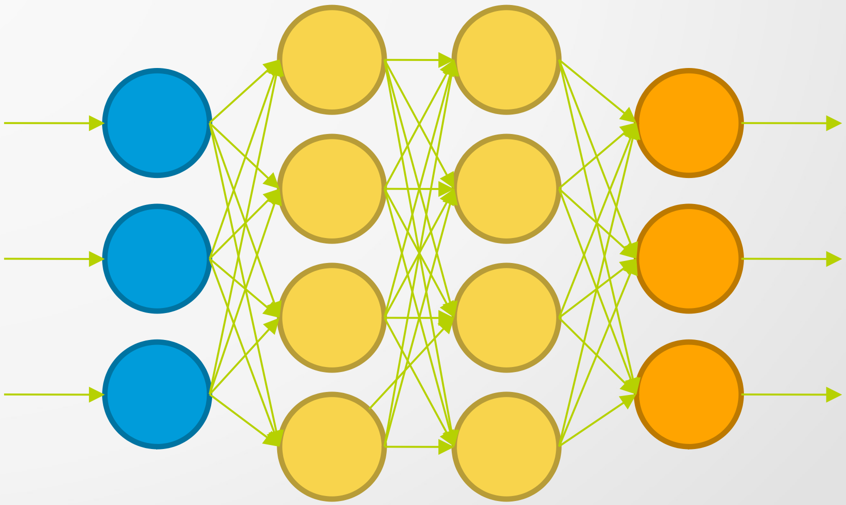
Human brain

- > 100 billion neuron
- Heavy interconnection (> 60 trillion)
- Average of a few thousand connections per neuron

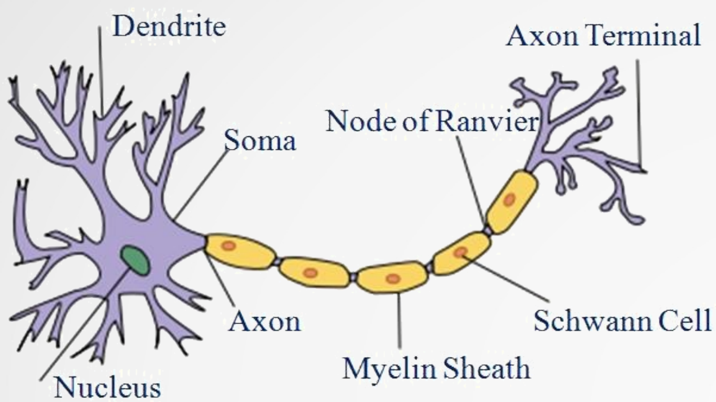
Motivation for Neural Networks

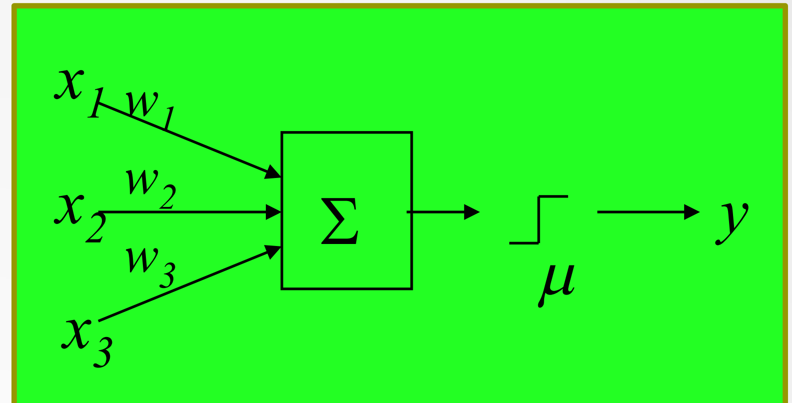
Artificial Neural Networks

- Use biology as inspiration for mathematical model
- Get signals from previous neurons
- Generate signals according to inputs
- Pass signals on to next neurons
- Layering many neurons allows complex models to be created



Motivation for Neural Networks






```
float sum = 0.0;
for (int i=1; i<=3, i++)
    sum += w[i]*x[i];
sum -= u;
if (sum >= 0)
    y = 1;
else
    y = 0;
```

$$
\begin{aligned}
y &= g(w_1 x_1 + w_2 x_2 + w_3 x_3 - \mu) \\
  &= g(\sum_i w_i x_i - \mu)
\end{aligned}
$$

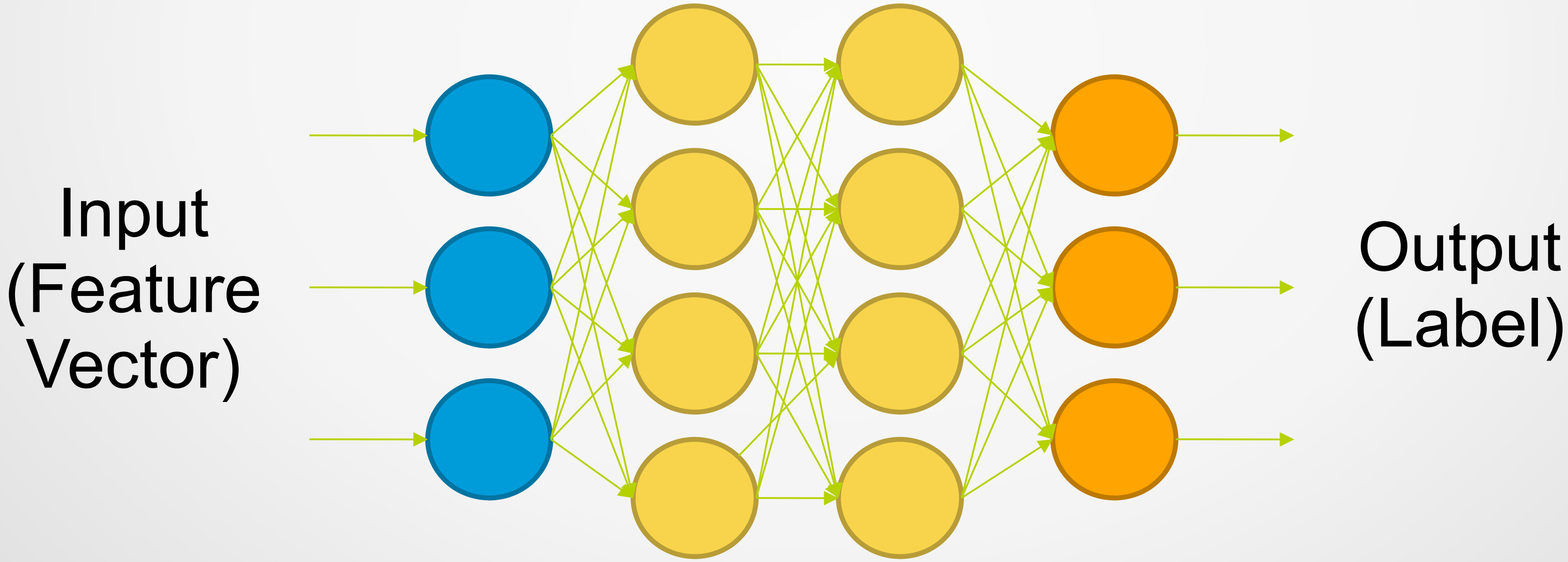
Where *g* is the unit step function:

$$
g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}
$$

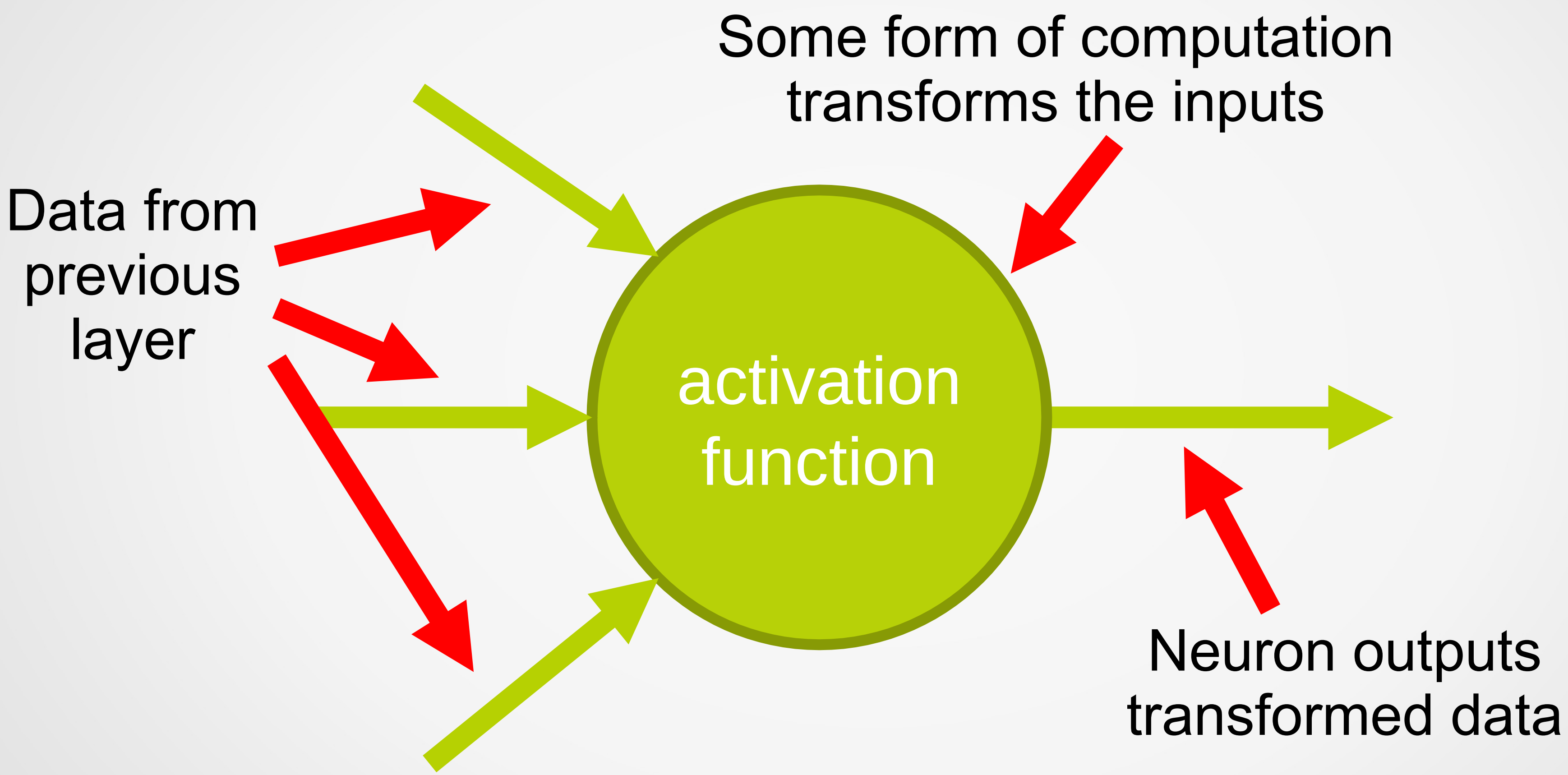
μ = the **threshold level**
w = **weight** of the connection

Structure of Neural Networks

- Can think of it as a complicated computation engine
- We will "train it" using our training data
- Then (hopefully) it will give good answers on new data

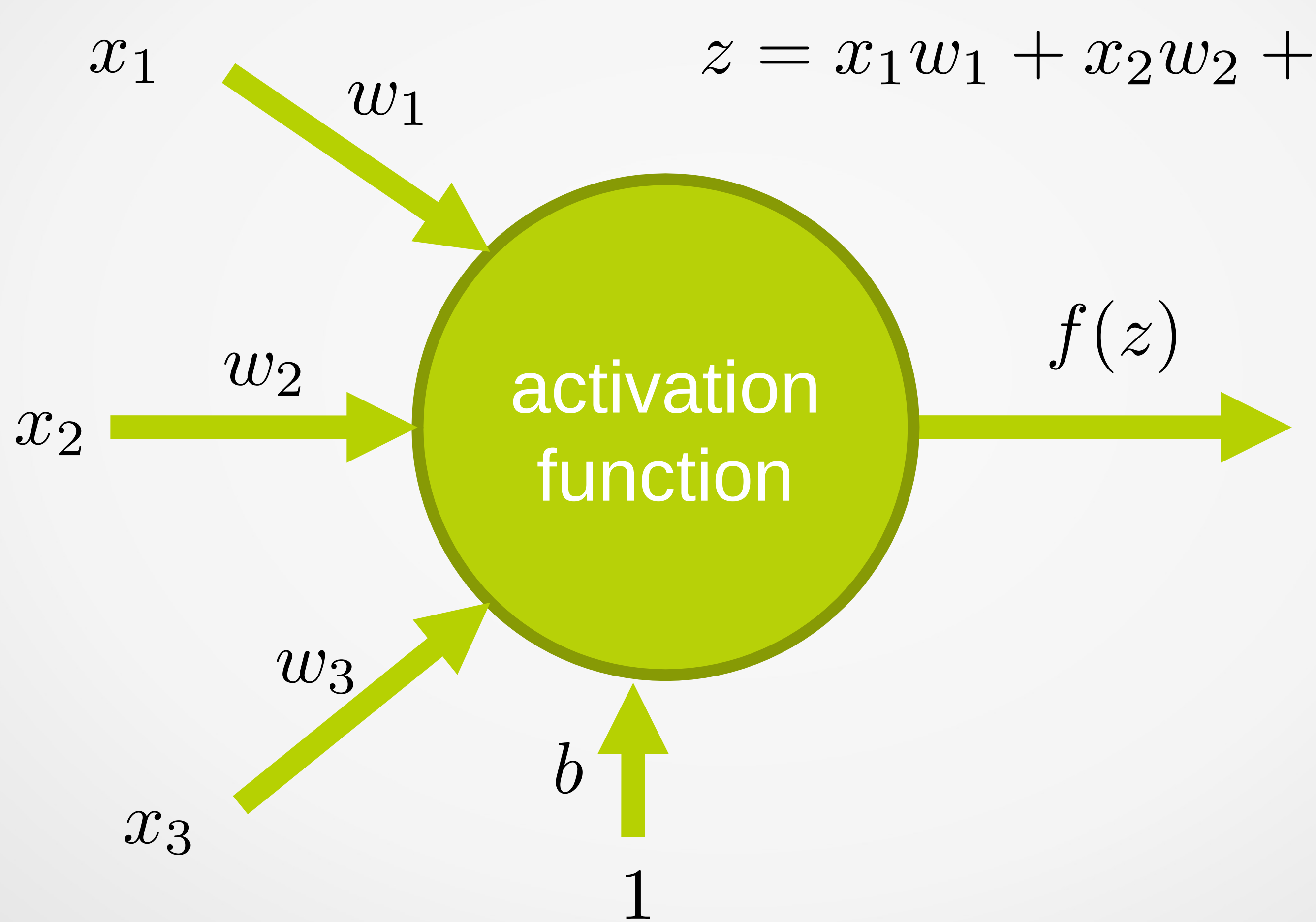


Artificial Neuron



Artificial Neuron

$$z = x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$



Artificial Neuron

Vector notation

- z = net input
- b = bias term
- f = activation function
- a = output to next layer

$$z = b + \sum_{i=1}^{m} x_i w_i$$

$$z = b + x^T w$$

$$a = f(z)$$

Neuron vs Logistic Regression

When we choose: $$f(z) = \frac{1}{1+e^{-z}}$$

$$z = b + \sum_{i=1}^{m} x_i w_i = x_1 w_1 + x_2 w_2 + \ldots + x_m w_m + b$$

The neuron is then simply a “unit” of logistic regression

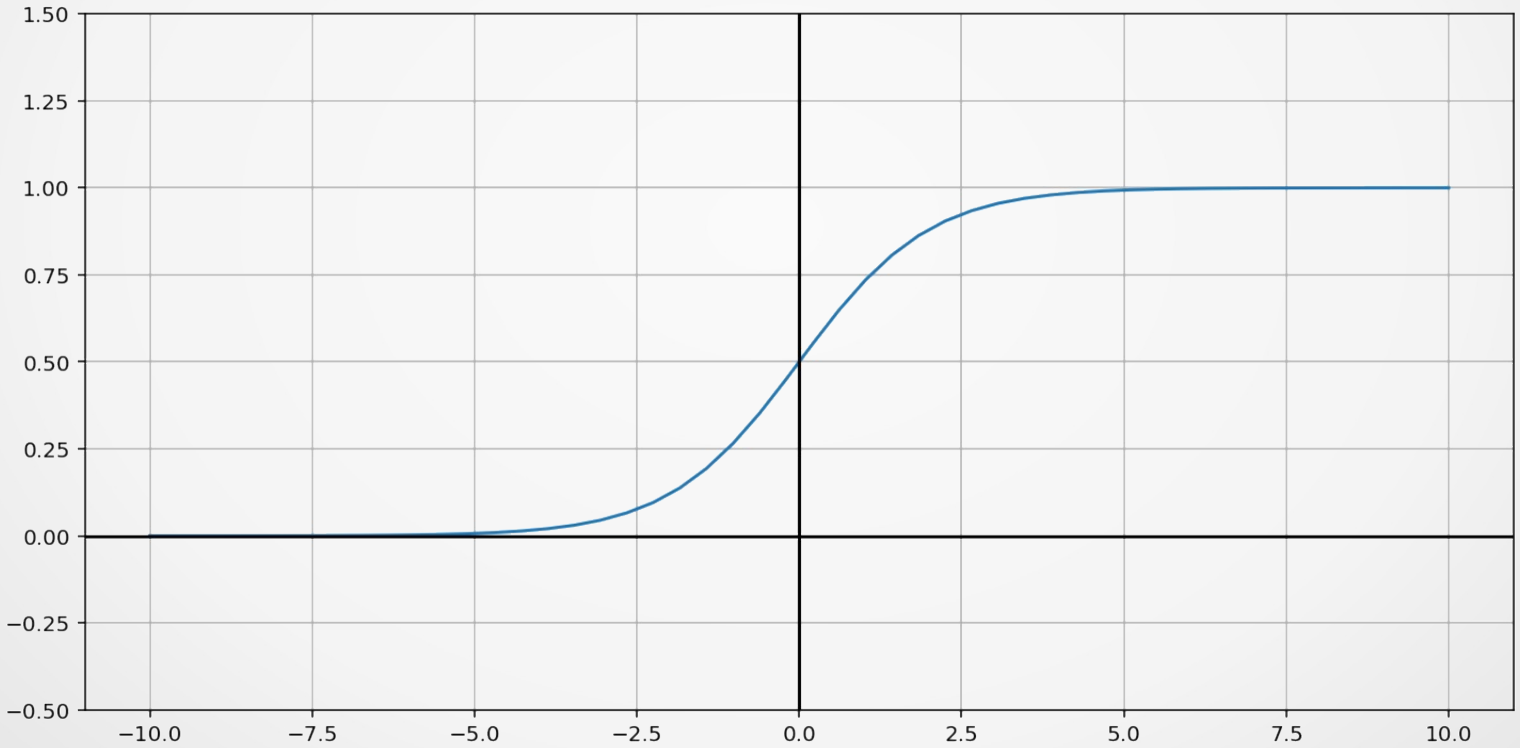
weights $\leftrightarrow$ coefficients

inputs $\leftrightarrow$ variables

bias term $\leftrightarrow$ constant term

Neuron vs Logistic Regression

This is called the “sigmoid” function: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$



Neuron vs Logistic Regression

A nice property of the sigmoid function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Quotient rule

$$\frac{d}{dx} \cdot \frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

$$\sigma'(z) = \frac{0 - (-e^{-z})}{(1+e^{-z})^2} = \frac{-e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{\cancel{1+e^{-z}}}{(1+e^{-z})^{\cancel{2}}} - \frac{1}{(1+e^{-z})^2}$$

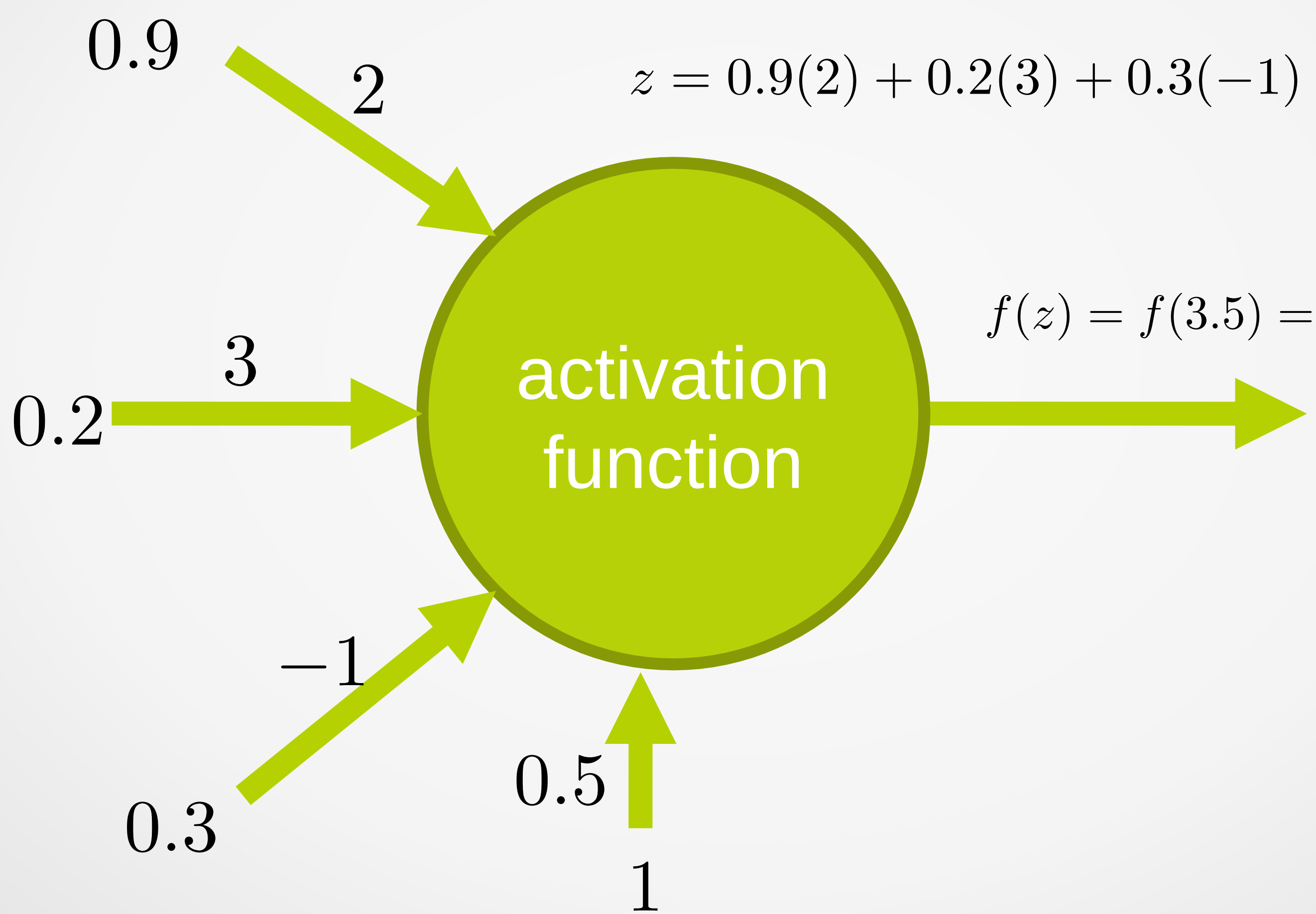
$$= \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$ This will be helpful!

Artificial Neuron Example

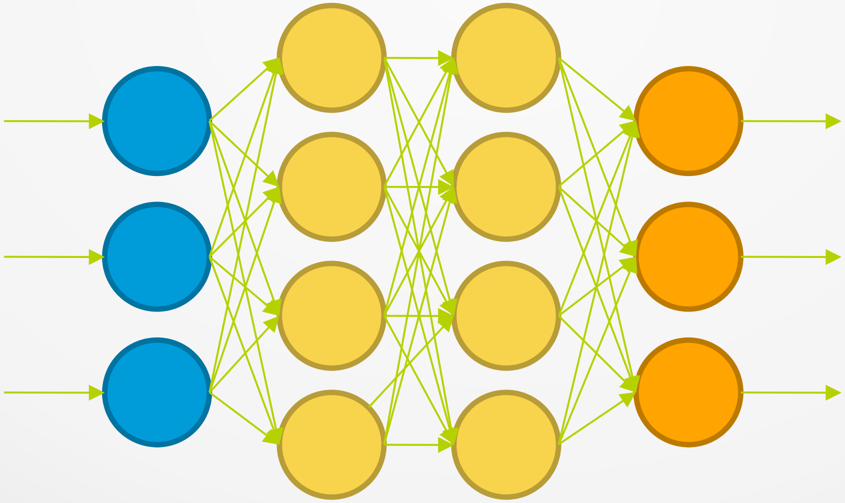
$$z = 0.9(2) + 0.2(3) + 0.3(-1) + 0.5 = 2.6$$

$$f(z) = f(3.5) = \frac{1}{1 + e^{(-2.6)}} = 0.93$$

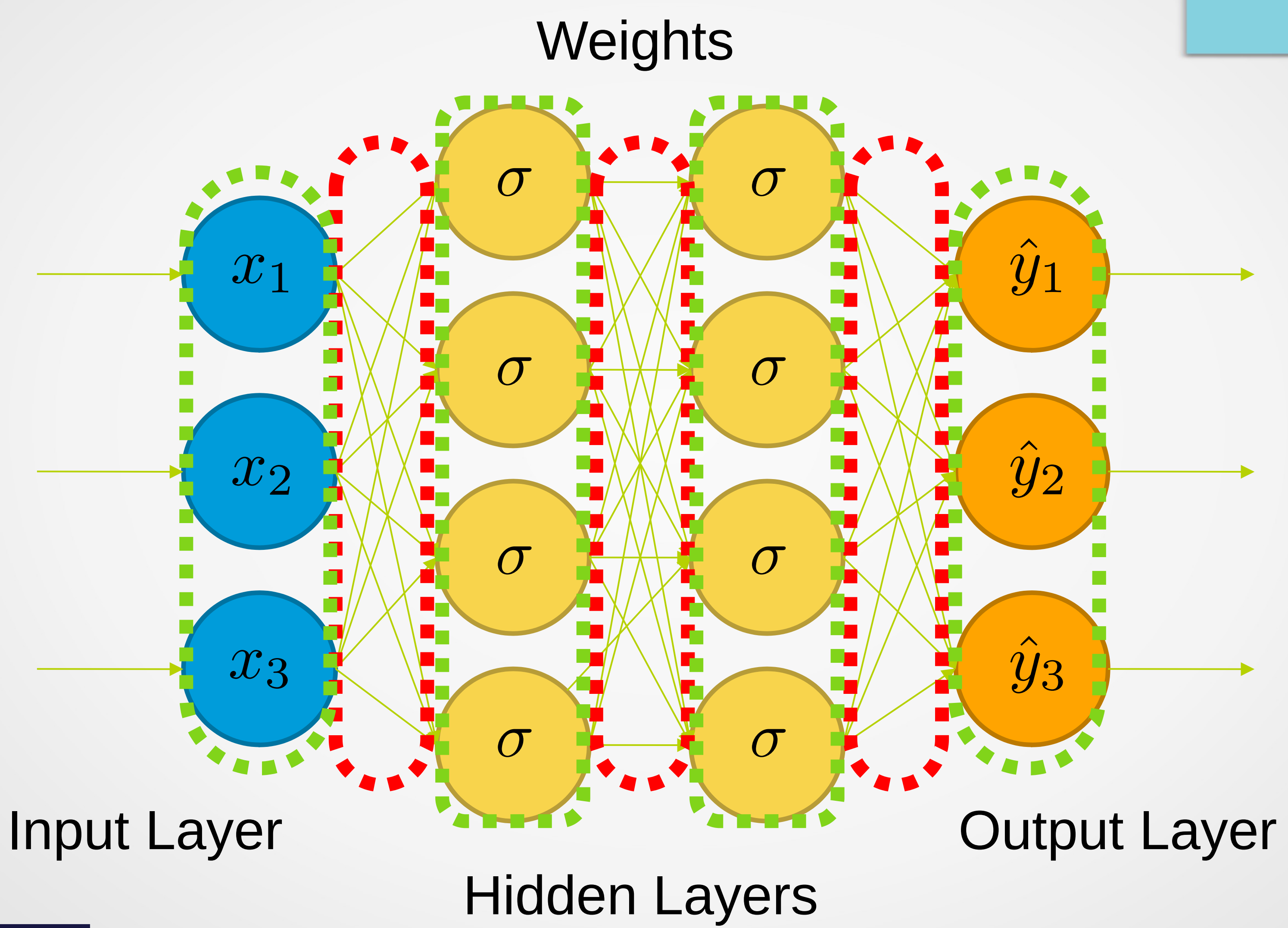


Network of Neurons

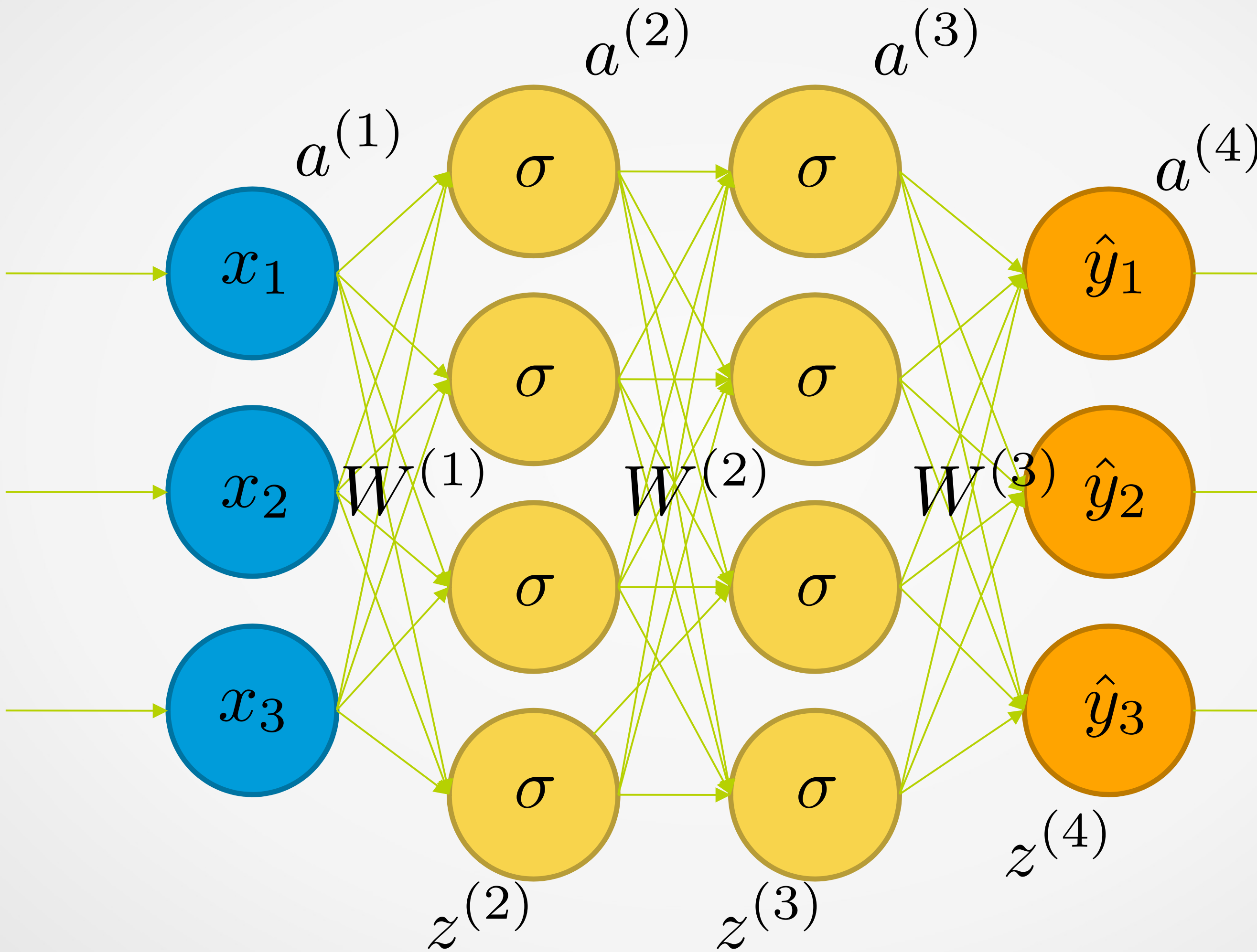
- What are the limitations of a single neuron?
  - Linear decision boundary (just like logistic regression)
- Most real-world problems are considerably more complicated!



Feedforward Neural Network



Feedforward Neural Network



Activations (output to next layer)

Weights (matrices)

Net Input (sum of weighted input)

Computing with Matrices

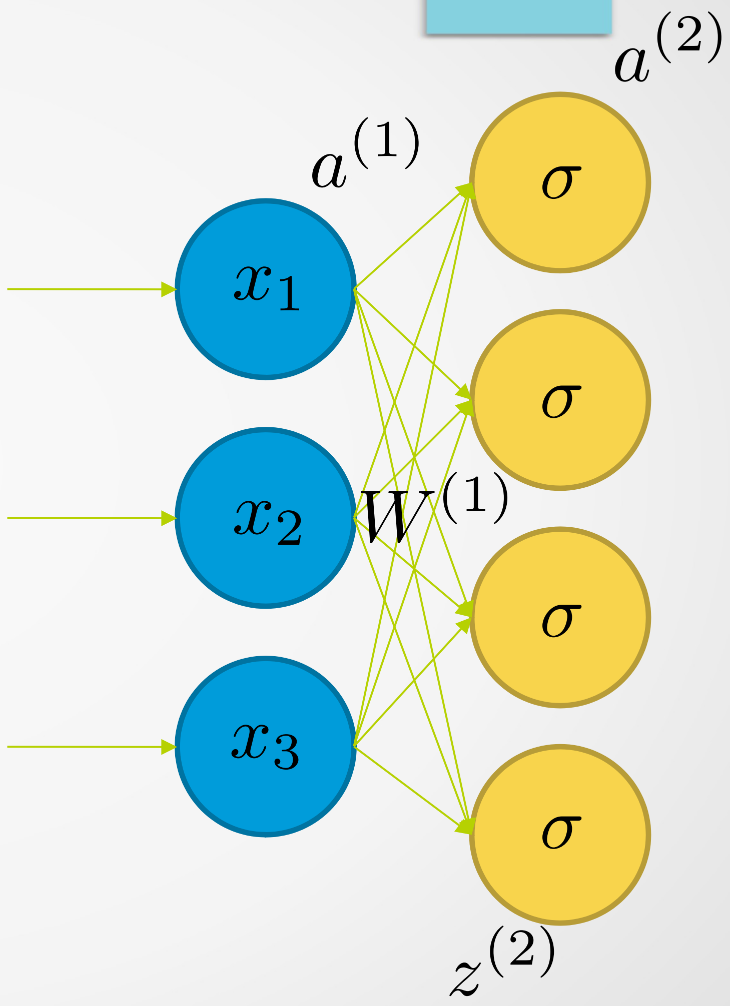
$W^{(1)}$ is a 3-by-4 matrix

$z^{(2)}$ is a 4-by-1 vector

$a^{(2)}$ is a 4-by-1 vector

$$a^{(1)} = x \qquad z^{(2)} = a^{(1)} W^{(1)}$$

$$a^{(2)} = \sigma\left(z^{(2)}\right)$$



Computing with Matrices

For a single training instance (data point)

Input: vector *x* (row vector with length 3)

Output: vector *y* (row vector with length 3)

$$a^{(1)} = x \qquad z^{(2)} = a^{(1)} W^{(1)}$$

$$a^{(2)} = \sigma\left(z^{(2)}\right) \qquad z^{(3)} = a^{(2)} W^{(2)}$$

$$a^{(3)} = \sigma\left(z^{(3)}\right) \qquad z^{(4)} = a^{(3)} W^{(3)}$$

$$\hat{y} = softmax\left(z^{(4)}\right)$$

Computing with Matrices

In practice, many data points are done at the same time by ‘stacking’ rows into a matrix

Input: matrix *x* (3-by-*n* matrix, each row one data point)

Output: matrix *y* (3-by-*n* matrix, each row one data point)

$$a^{(1)} = x \qquad z^{(2)} = a^{(1)} W^{(1)}$$

$$a^{(2)} = \sigma\left(z^{(2)}\right) \quad z^{(3)} = a^{(2)} W^{(2)}$$

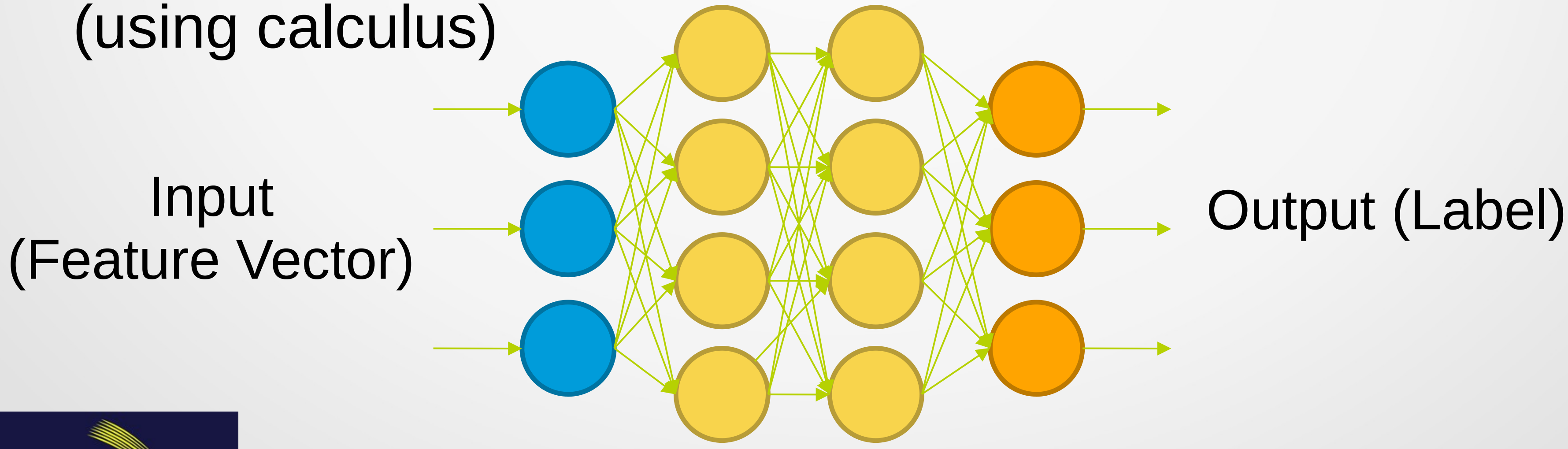
$$a^{(3)} = \sigma\left(z^{(3)}\right) \quad z^{(4)} = a^{(3)} W^{(3)}$$

$$\hat{y} = softmax\left(z^{(4)}\right)$$

Equations still look exactly the same!

Training a Neural Network

- Put in training inputs, get the output
- Compare output to correct answers (target): look at the loss function *J*
- Adjust and repeat
- Backpropagation tells us how to make the adjustment (using calculus)

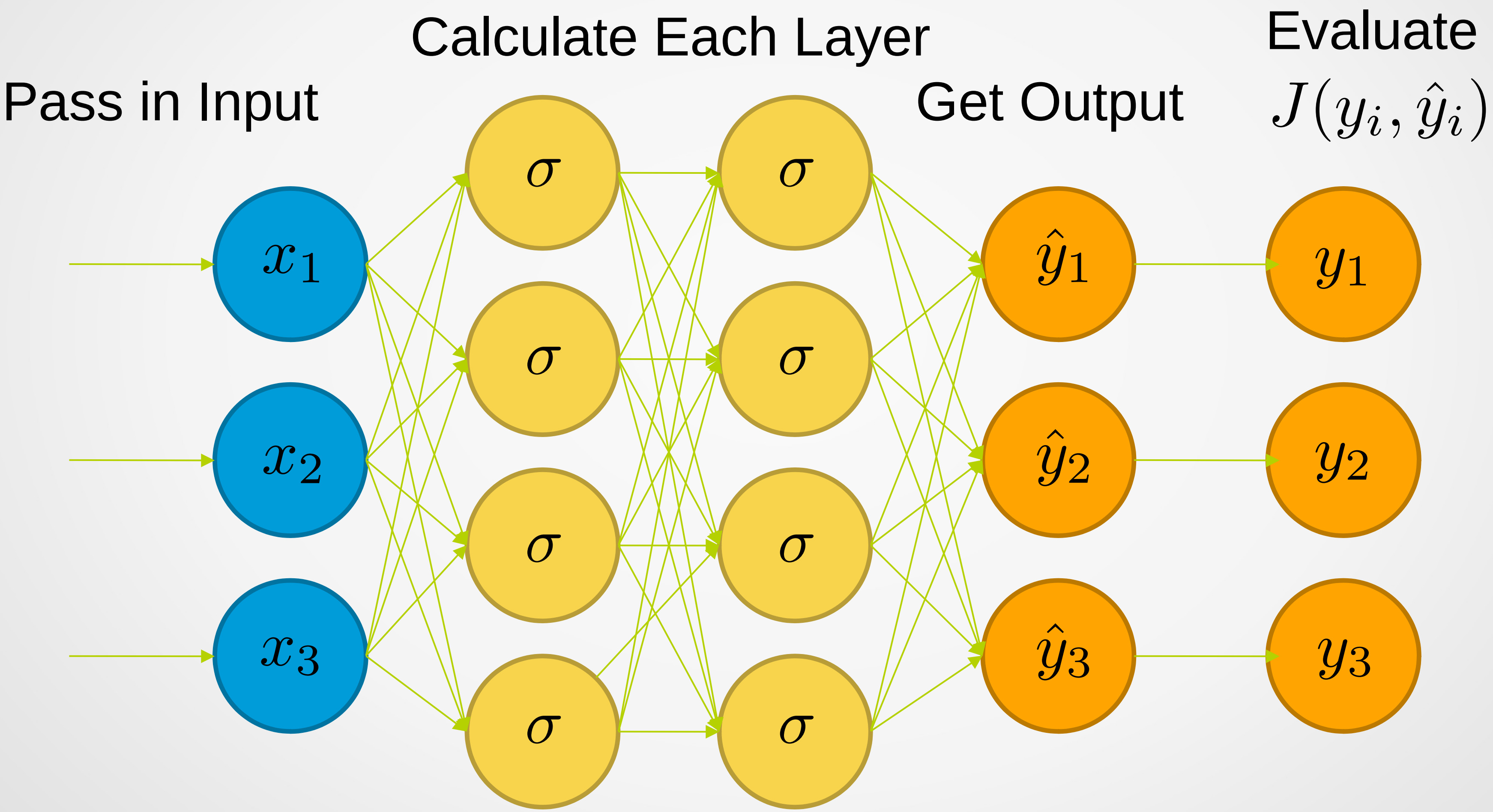


Training a Neural Network

Gradient Descent!

1) Make prediction
2) Calculate Loss function
3) Calculate gradient of the loss function w.r.t parameters
4) Update parameters by taking a step in the opposite direction
5) Iterate

Training a Neural Network



Training a Neural Network

Gradient Descent!

1) Make prediction
2) Calculate Loss function
3) Calculate gradient of the loss function w.r.t parameters
4) Update parameters by taking a step in the opposite direction
5) Iterate

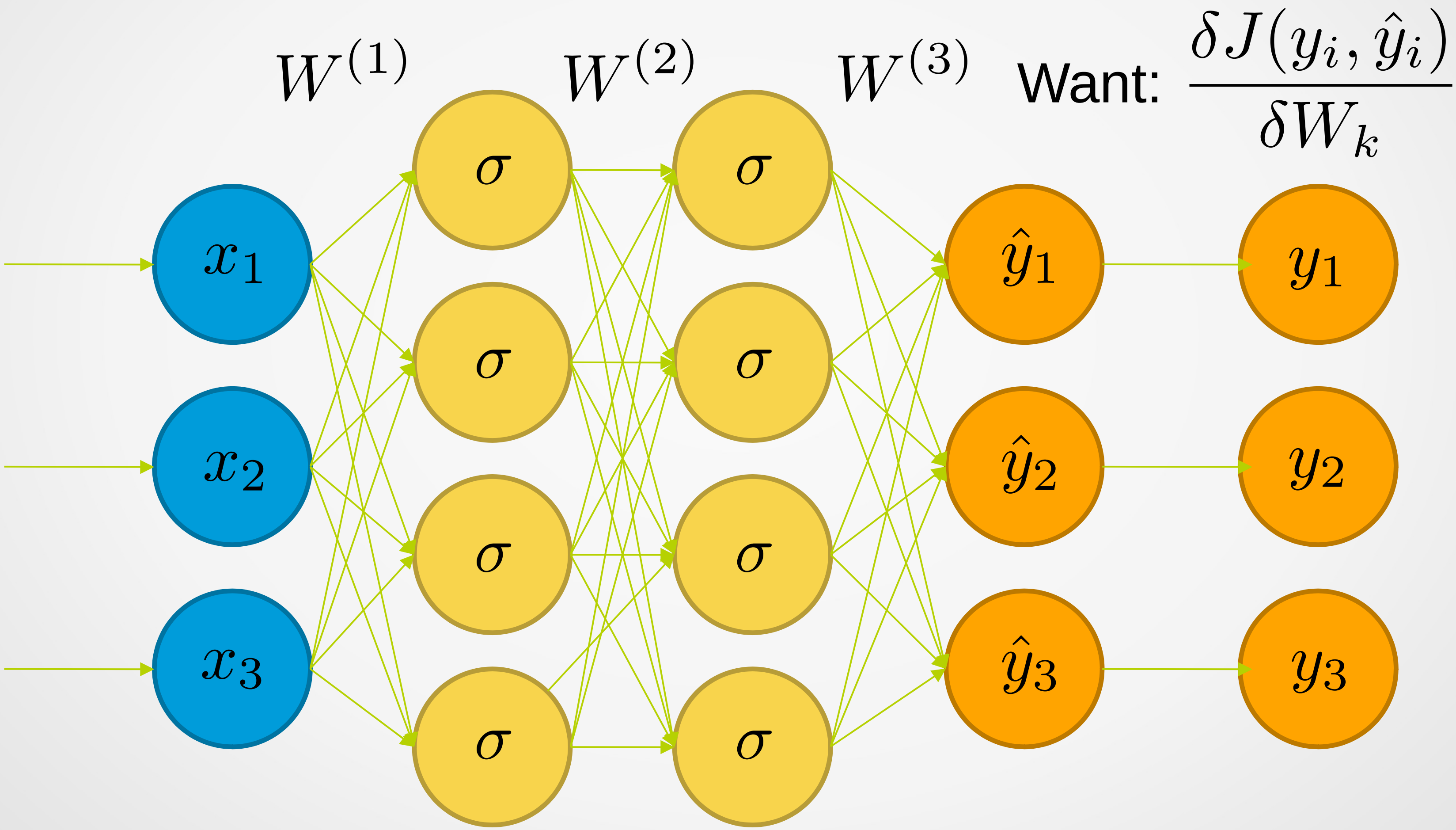
Training a Neural Network

- How do we change the weights to reduce our Loss Function?
- Think of the neural network as a function $F : X \Rightarrow Y$
- *F* is a complex computation involving many weights W_k
- Given the structure, the weights "define" the function *F* (and therefore define our model)
- Loss Function is $J\left(y, F(x)\right)$

Training a Neural Network

- Get $\frac{\delta J}{\delta W_k}$ for every weight in the network
- This tells us what direction to adjust each W_k if we want to lower our loss function
- Make an adjustment and repeat!

Training a Neural Network



Training a Neural Network

Calculus to the rescue!

- Use calculus, chain rule, etc.
- Functions are chosen to have “nice” derivatives
- Numerical issues to be considered

Training a Neural Network

Spoiler/punchline

$$\frac{\delta J}{\delta W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

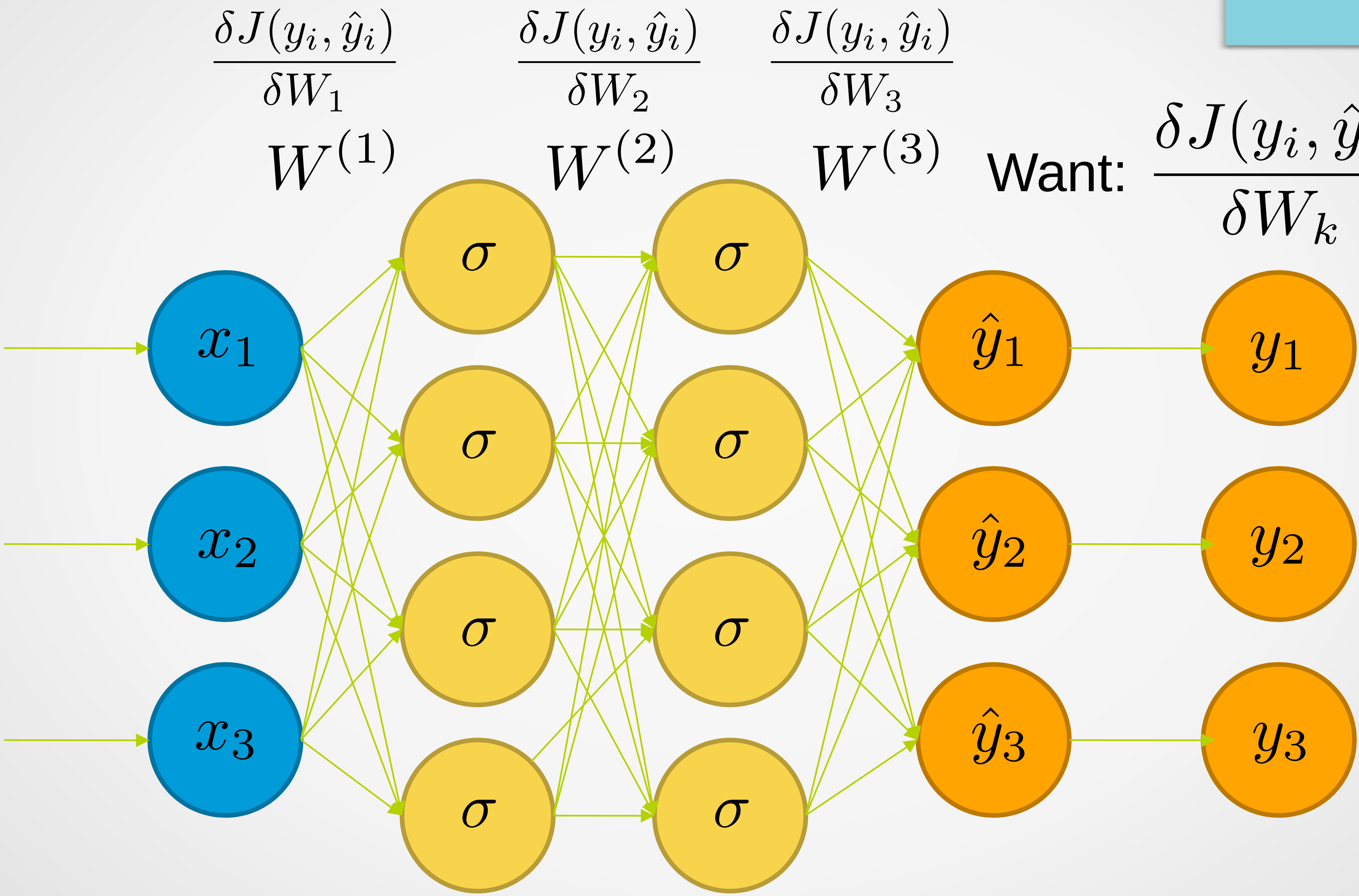
$$\frac{\delta J}{\delta W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma' \left(z^{(3)}\right) \cdot a^{(2)}$$

$$\frac{\delta J}{\delta W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma' \left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma' \left(z^{(2)}\right) \cdot X$$

Recall that: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Although they appear complex, above are easy to compute!

Training a Neural Network



Want: $\frac{\delta J(y_i, \hat{y}_i)}{\delta W_k}$

Training a Neural Network

Gradient Descent!

1) Make prediction
2) Calculate Loss function
3) Calculate gradient of the loss function w.r.t parameters
4) Update parameters by taking a step in the opposite direction
5) Iterate

Vanishing Gradients

$$\frac{\delta J}{\delta W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma' \left(z^{(3)} \right) \cdot W^{(2)} \cdot \sigma' \left(z^{(2)} \right) \cdot X$$

- The above is the gradient for the first layer
- Remember: $\sigma'(z) = \sigma(z)(1 - \sigma(z)) \leq 0.25$
- As we have more layers, the gradient gets very small in the early layers
- This is known as the “vanishing gradient” problem
- Other activations (such as ReLU) have become common because of this

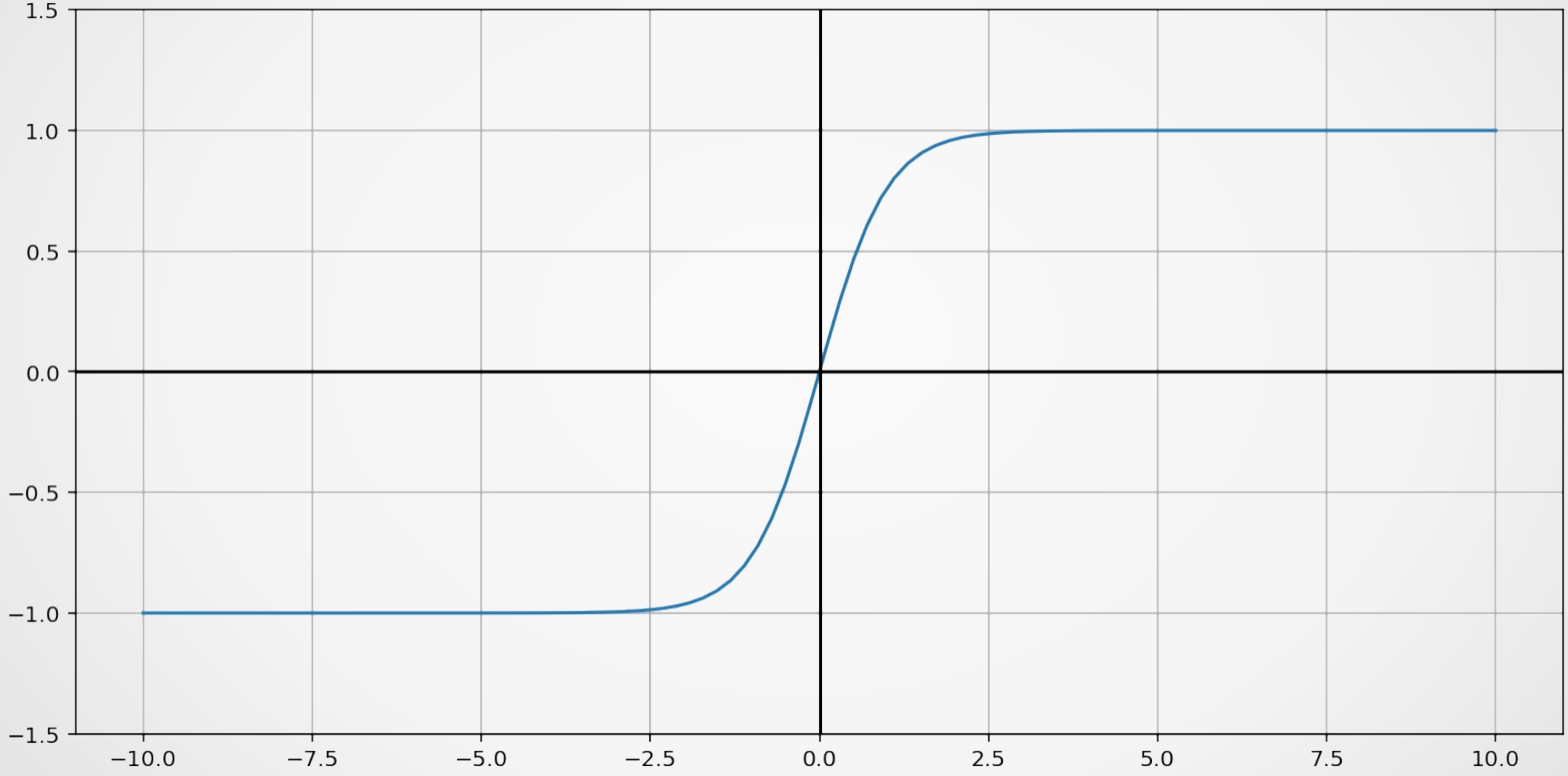
Hyperbolic Tangent Function

- Pronounced “tanch”

$$tanh(z) = \frac{sinh(z)}{cosh(z)} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$tanh(0) = 0$$
$$tanh(\infty) = 1$$
$$tanh(-\infty) = -1$$

Hyperbolic Tangent Function

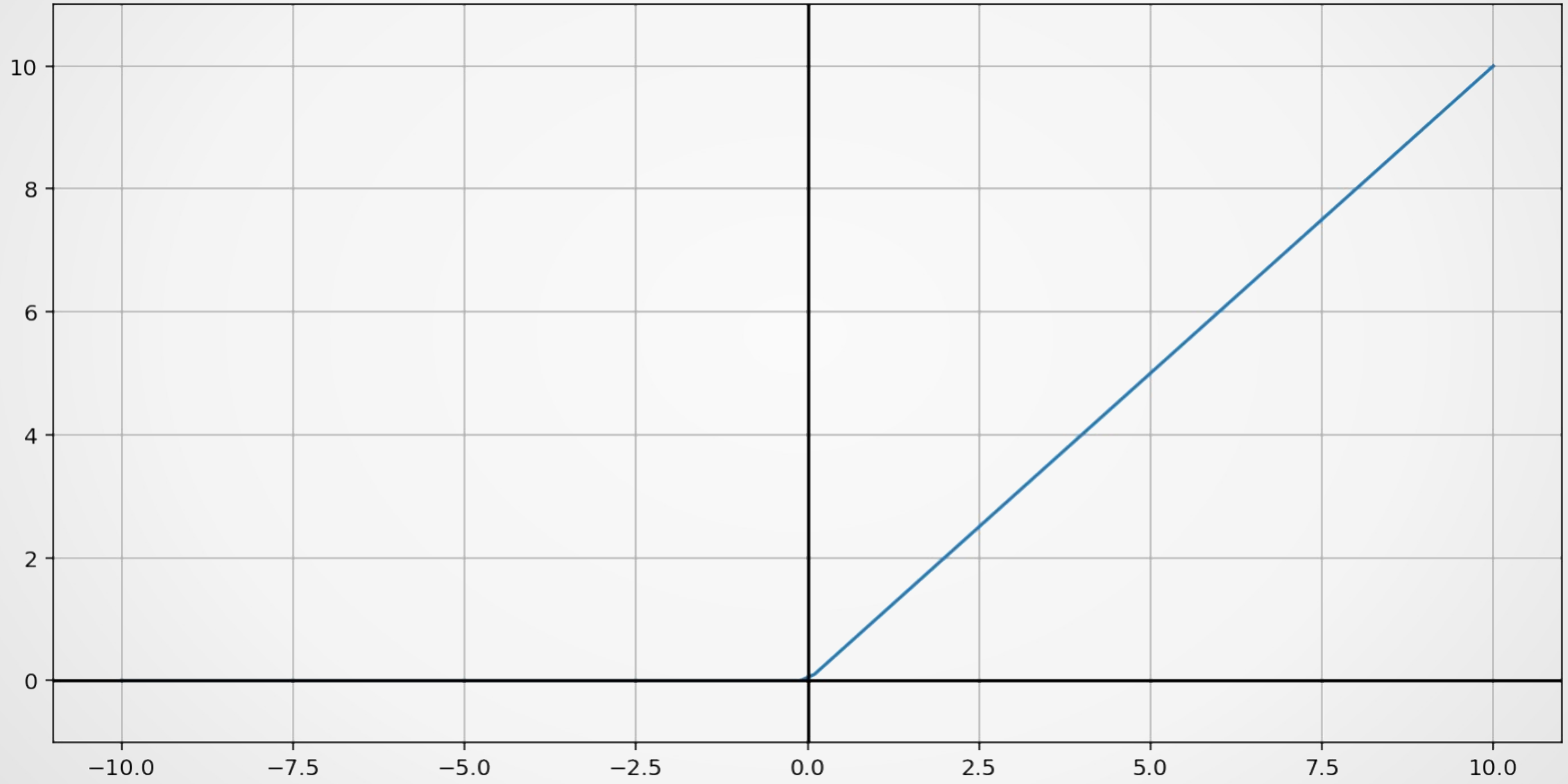


Rectified Linear Unit (ReLU)

$$ReLU(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$$
$$= \max(0, z)$$

$$ReLU(0) = 0$$
$$ReLU(z) = z \qquad \text{for } (z >> 0)$$
$$ReLU(-z) = 0$$

Rectified Linear Unit (ReLU)



“Leaky” Rectified Linear Unit (ReLU)

$$LReLU(z) = \begin{cases} \alpha z, & z < 0 \\ z, & z \geq 0 \end{cases}$$

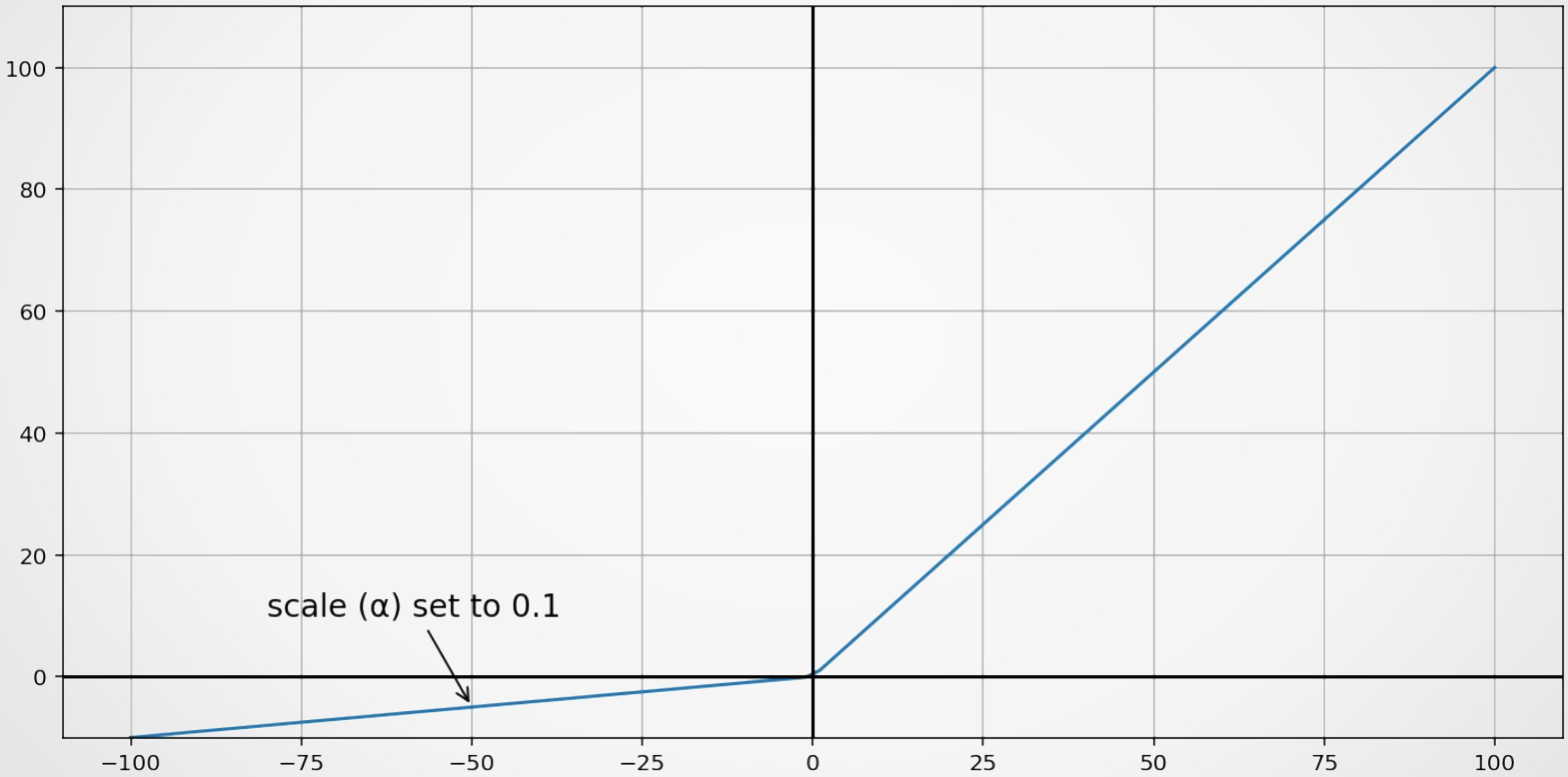
$$= \max(\alpha z, z) \qquad \text{for } (\alpha < 1)$$

$$LReLU(0) = 0$$

$$LReLU(z) = z \qquad \text{for } (z >> 0)$$

$$LReLU(-z) = -\alpha z$$

“Leaky” Rectified Linear Unit (ReLU)



End of Lecture

Many thanks to Intel Software for providing a variety of resources for this lecture series