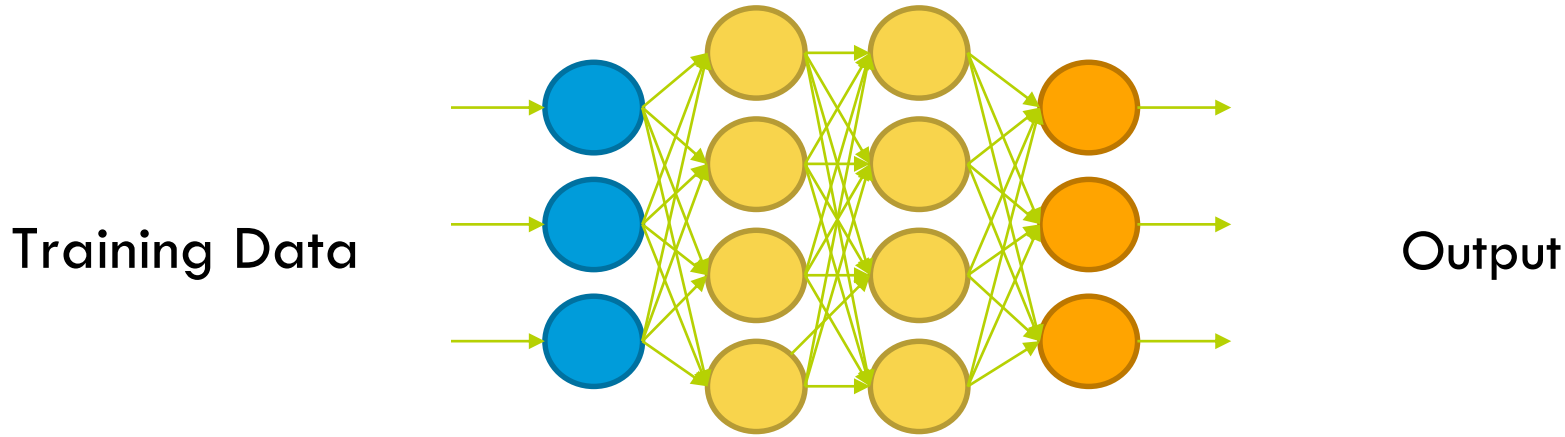


# Backpropagation in Neural Nets

# How to Train a Neural Net?



- Put in Training inputs, get the output
- Compare output to correct answers: Look at loss function  $J$
- Adjust and repeat!
- Backpropagation tells us how to make a single adjustment using calculus.

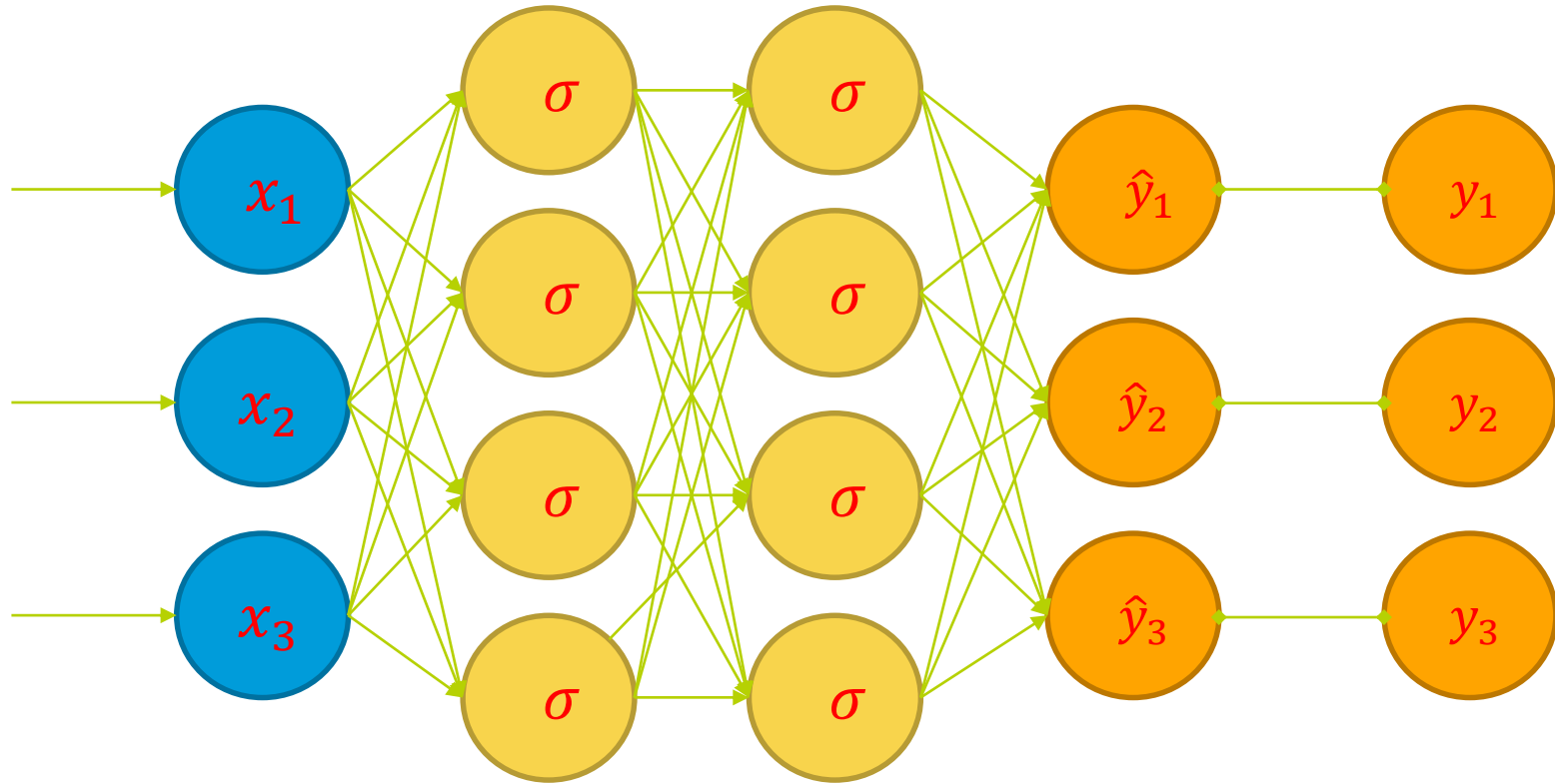
# How have we trained Gefore?

- Gradient Descent!
  1. Make prediction
  2. Calculate Loss
  3. Calculate gradient of the loss function w.r.t. parameters
  4. Update parameters  $\theta$  taking a step in the opposite direction
  5. Iterate

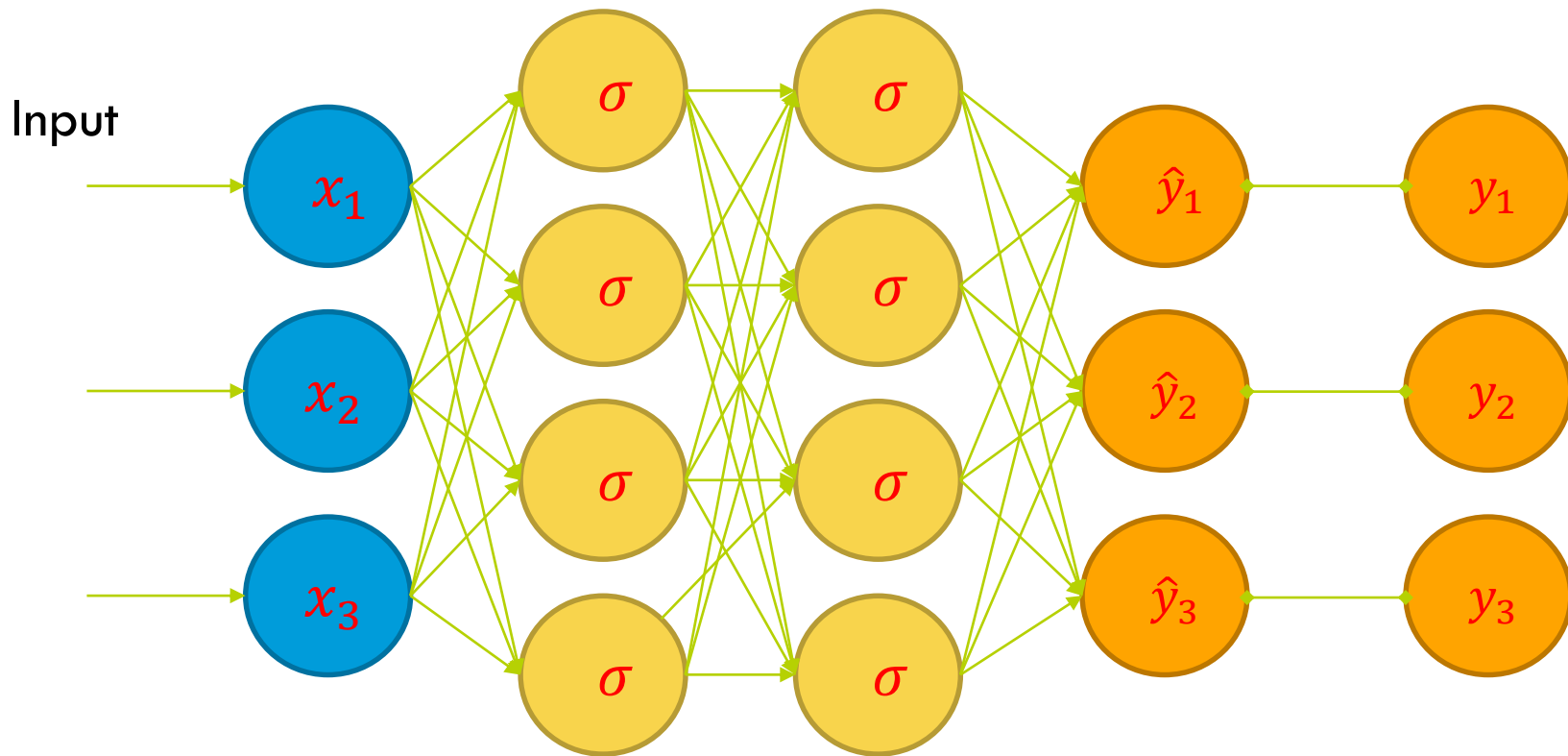
# How have we trained Gefore?

- Gradient Descent!
  1. Make prediction
  2. Calculate Loss } 'The Forward Pass'
  3. Calculate gradient of the loss function w.r.t. parameters
  4. Update parameters  $\theta$  taking a step in the opposite direction
  5. Iterate

# Feedforward Neural Network

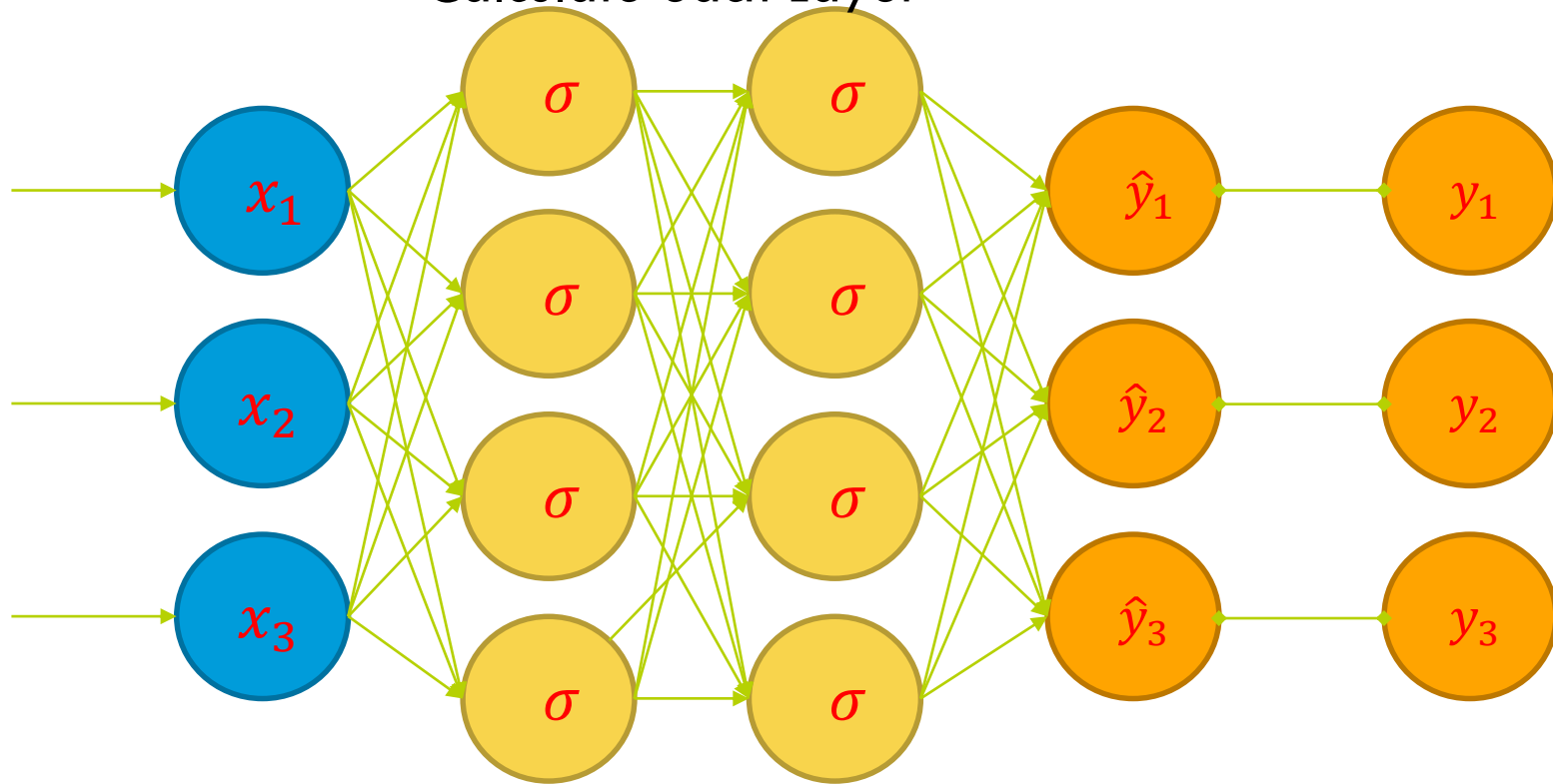


# Forward Propagation

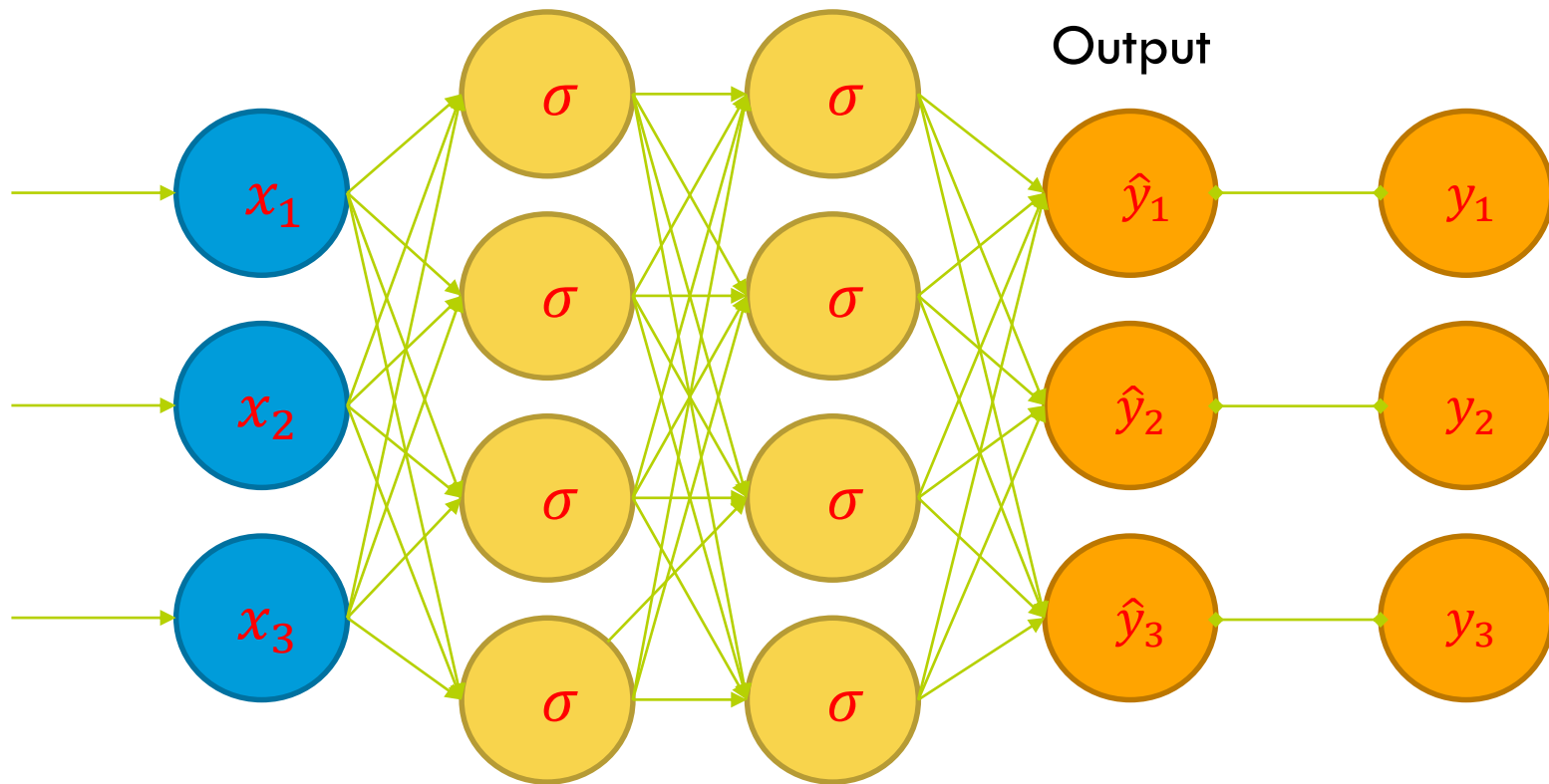


# Forward Propagation

Calculate each Layer

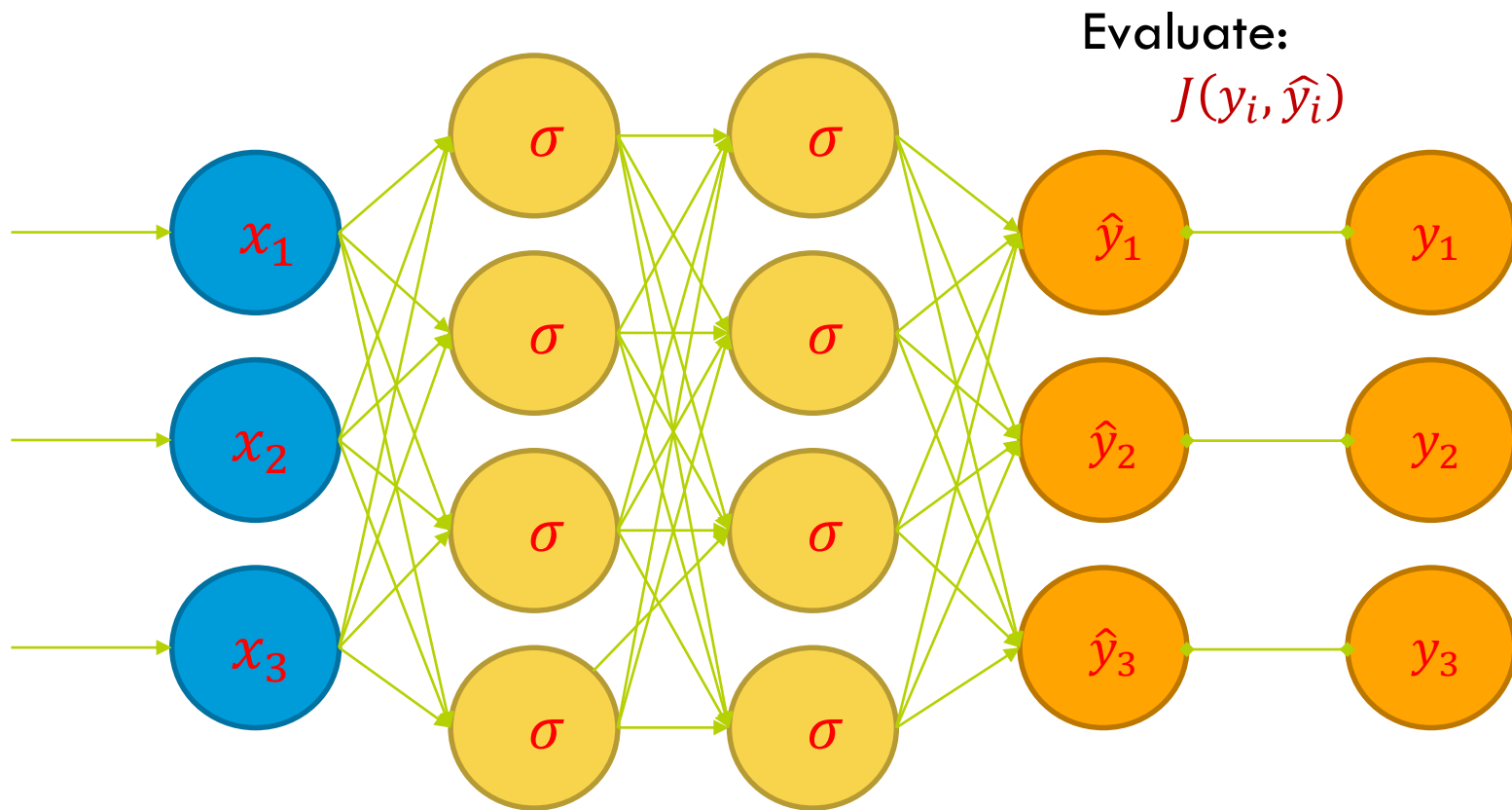


# Forward Propagation





# Forward Propagation



# How have we trained Gefore?

- Gradient Descent!
  1. Make prediction
  2. Calculate Loss
  3. Calculate gradient of the loss function w.r.t. parameters
  4. Update parameters  $\theta$  taking a step in the opposite direction
  5. Iterate

# How to calculate gradient?

---

- Chain rule



# How to Train a Neural Net?

- How could we change the weights to make our Loss Function lower?
- Think of neural net 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  $J(y, F(x))$

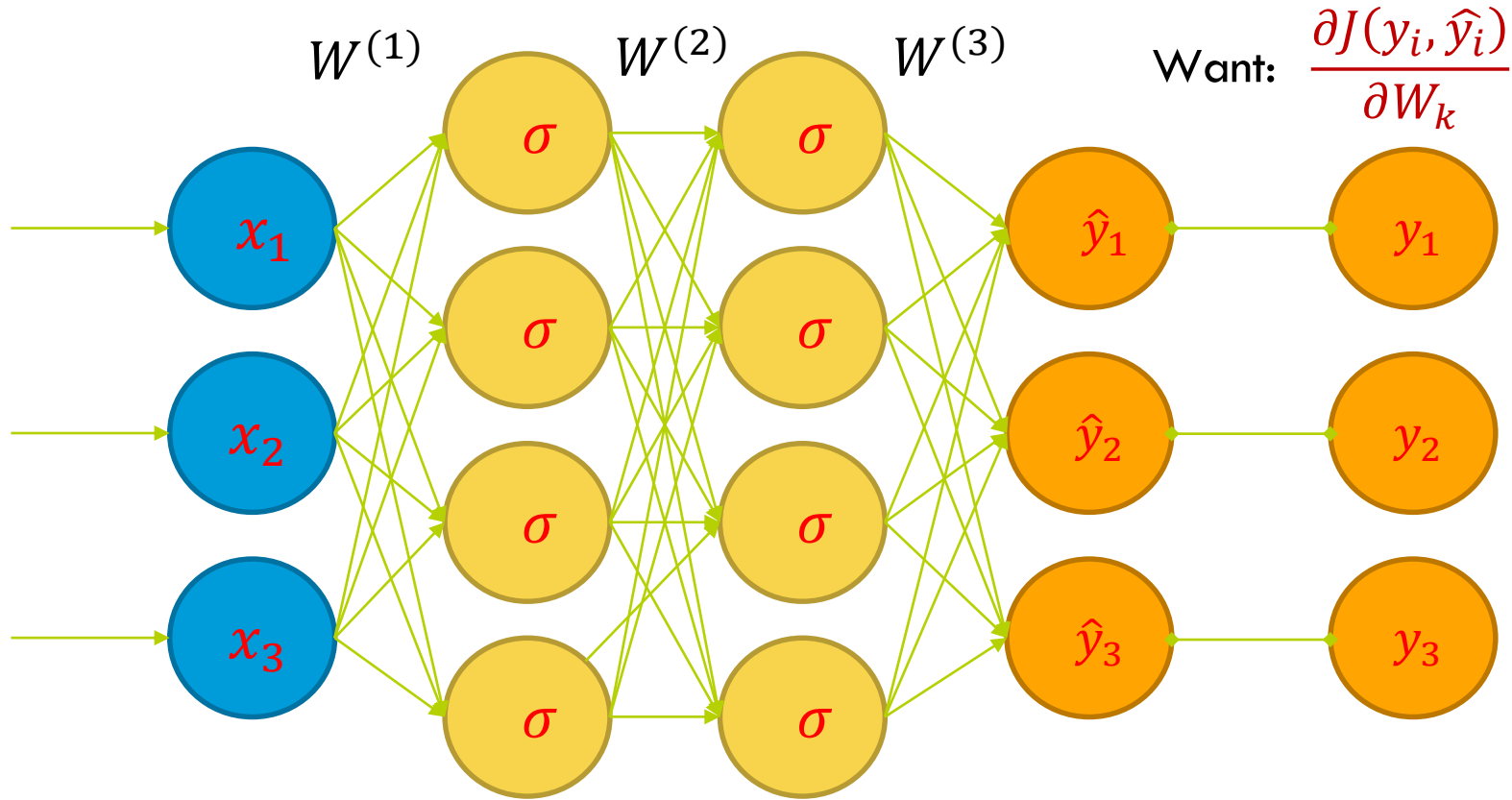
**PS. Loss function may also look like  $L(y, F(x))$**

**PS.  $F(x)$  is the current prediction or  $y^{\wedge}$ ;  $y$  is the ground truth (label)**

# How to Train a Neural Net?

- Get  $\frac{\partial J}{\partial W_k}$  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!

# Feedforward Neural Network



# Calculus to the Rescue

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

## Punchline

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

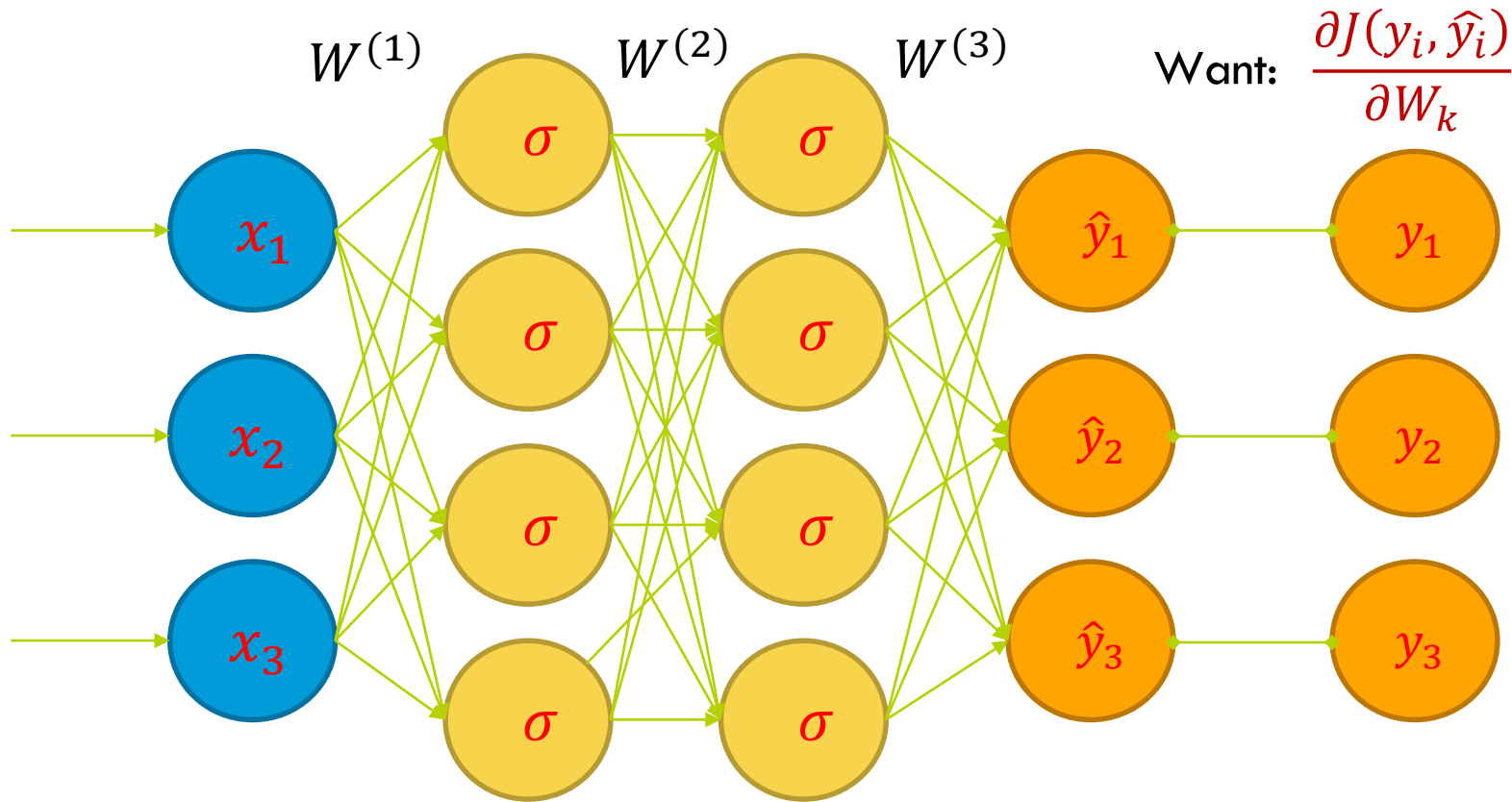
$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'(z^{(3)}) \cdot a^{(2)}$$

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

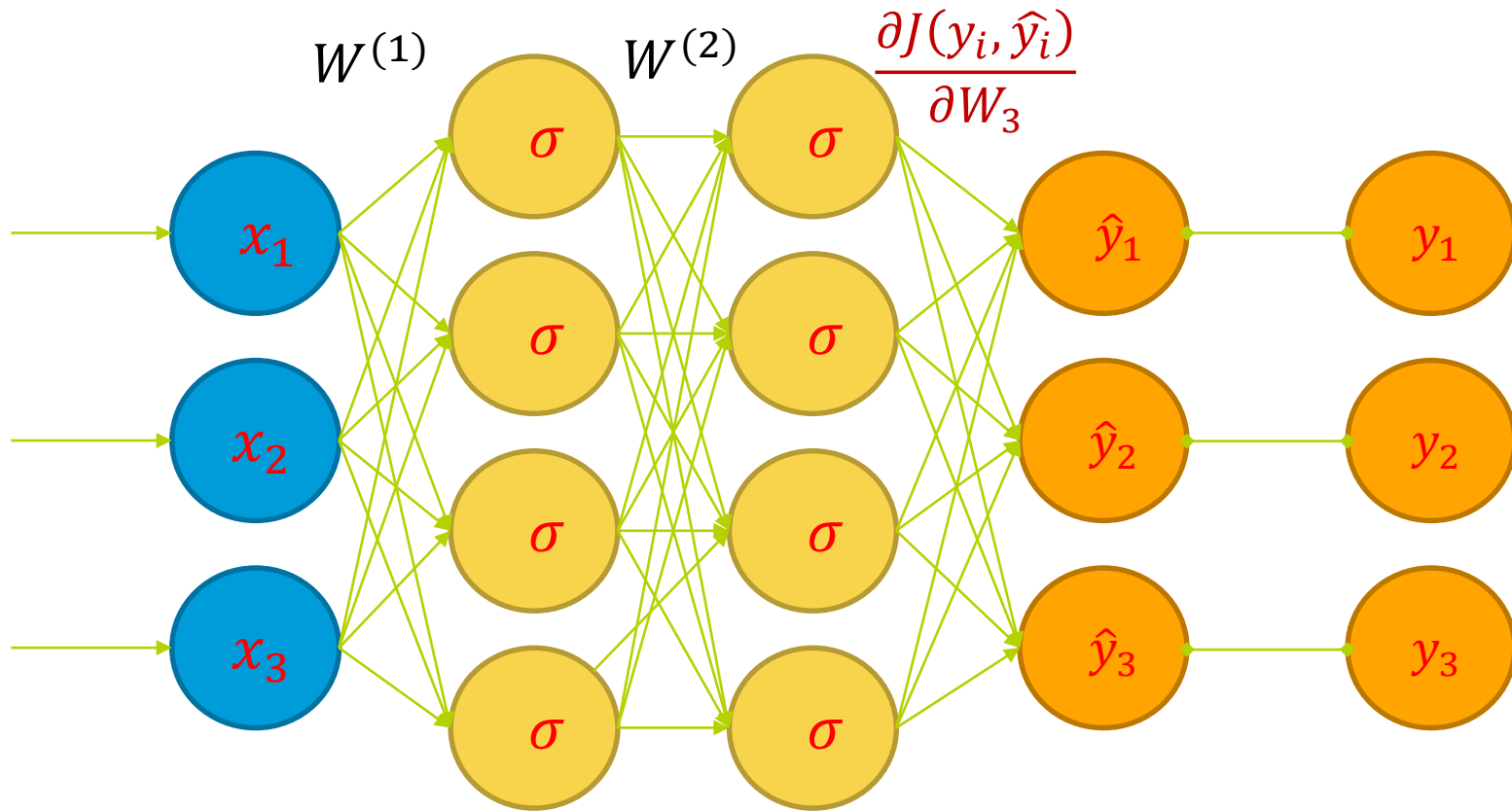
- Recall that:  $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Though they appear complex, aGove are easy to compute!



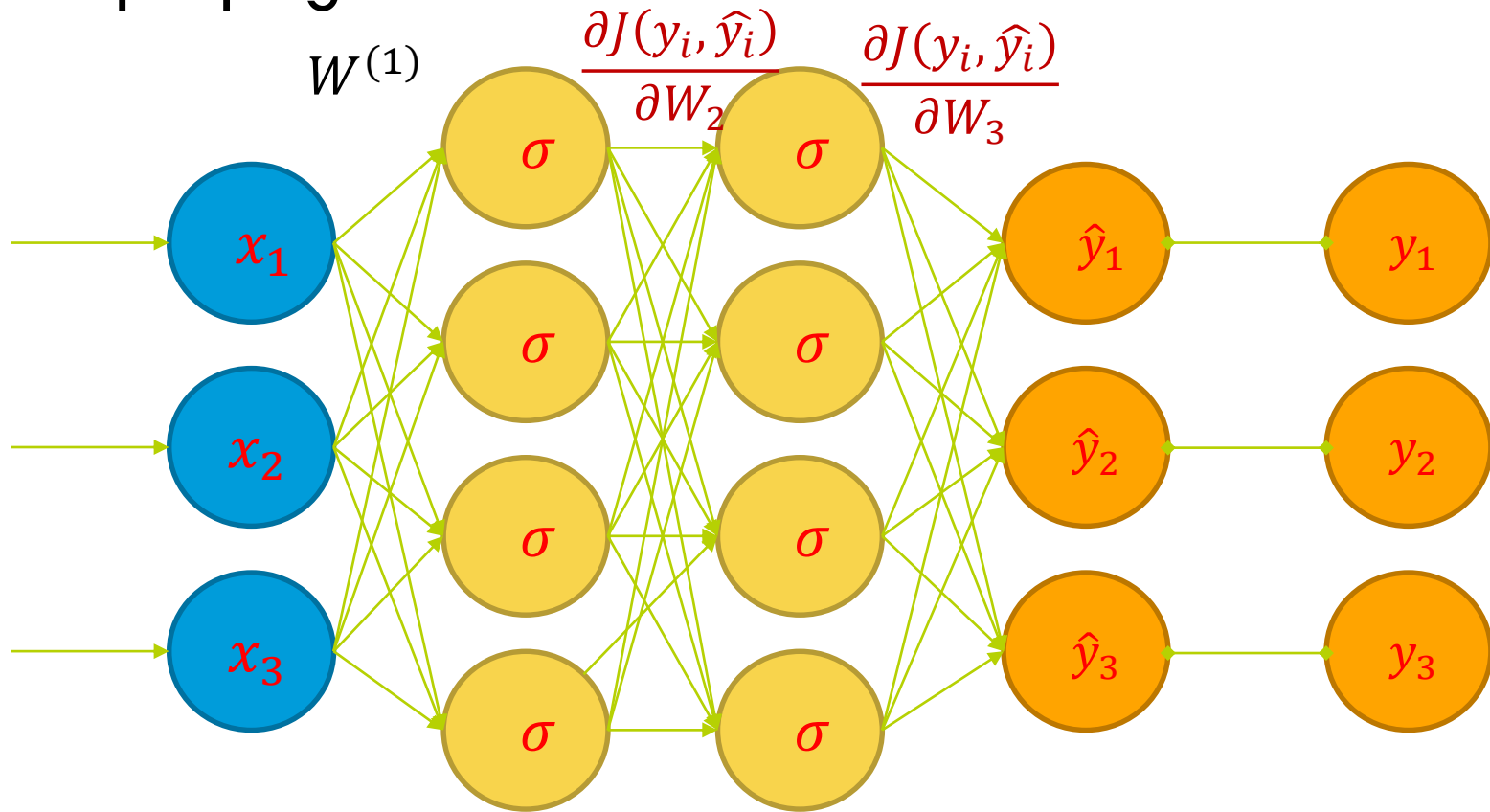
# Backpropagation



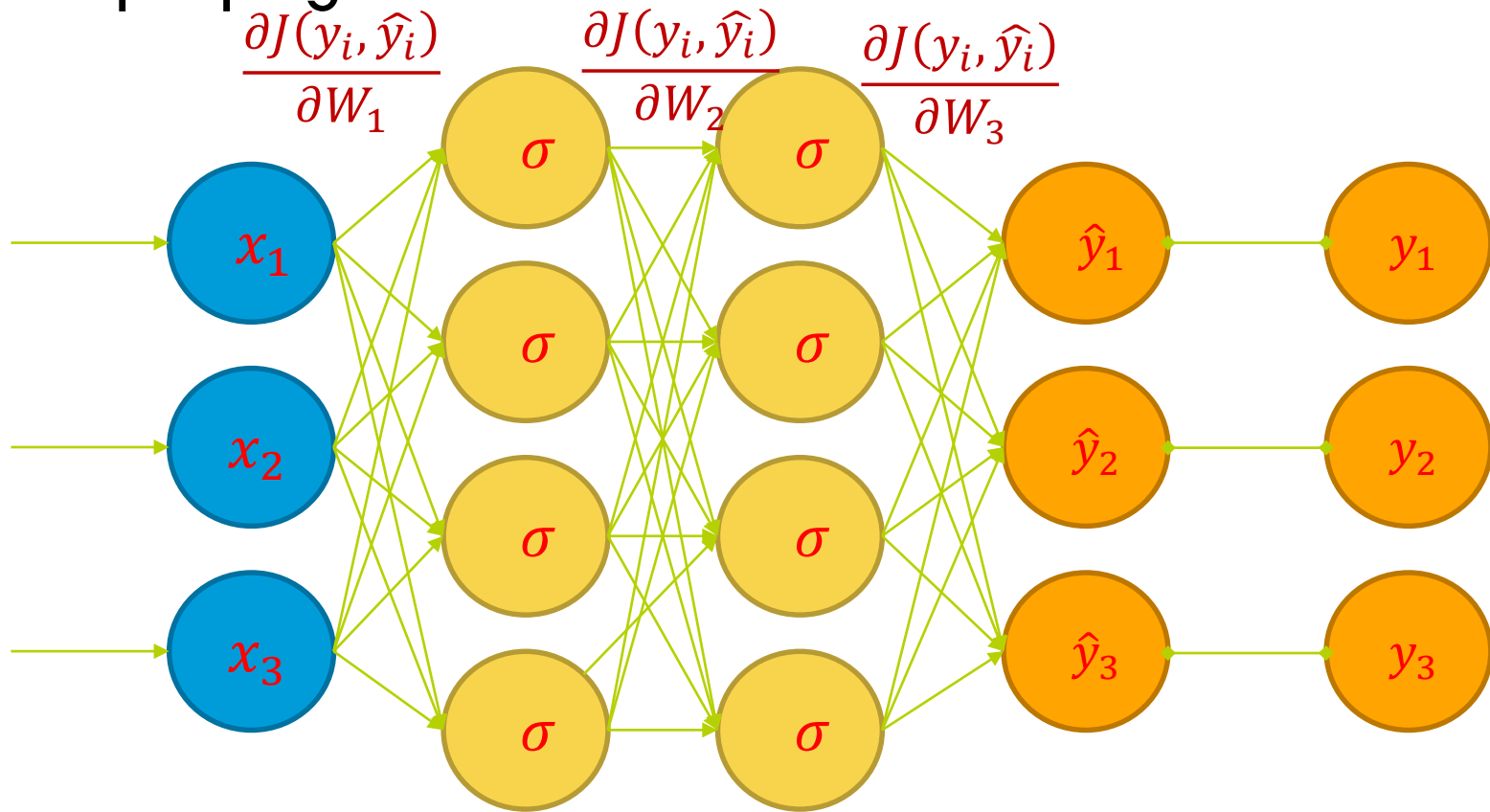
# Backpropagation



# Backpropagation



# Backpropagation



# How have we trained Gefore?

- Gradient Descent!
  1. Make prediction
  2. Calculate Loss
  3. Calculate gradient of the loss function w.r.t. parameters
  4. Update parameters  $G_y$  taking a step in the opposite direction
  5. Iterate

**In short, Backpropagation  $\sim$  passing back the loss**

Input

$\mathbf{x}$

3

$w_1=0.1$

$a(W^T \mathbf{x})$

Prediction

$\hat{y}$

True Output

$y$

1

**Activation Function:**  $a(z) = z$

**Mean Squared Error Function:**  $L(\hat{y}, y) = (\hat{y} - y)^2$

**Learning Rate:**  $\eta = 1$

Let's use the above example, 1D input, 1 scalar weight  $w_1$ , and identity activation function.

First, it's Forward-passing. We have the MSE loss regarding our input  $x$  as

$$L = (y - \hat{y})^2 = (y - W^T X)^2 = (1 - 0.3)^2 = 0.49$$

Next, backpropagation, we will solve the partial gradient of  $L$  regarding  $w_1$  using chain rule.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_1} = 2 (\hat{y} - y) x = 2 (-0.7) 3 = -4.2$$

Last, we will update  $w_1$  with the opposite of the gradient \* learning rate. The reason we go the opposite of the gradient is because the gradient points to the direction of the largest increase in  $L$ .

$$w_1^{new} = w_1 - \eta \frac{\partial L}{\partial w_1} = 0.1 + 1 \times 4.2 = 4.3$$

# Vanishing Gradients

Recall that:

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

- Remember:  $\sigma'(z) = \sigma(z)(1 - \sigma(z)) \leq .25$
- As we have more layers, the gradient gets very small at the early layers.
- This is known as the “vanishing gradient” problem.
- For this reason, other activations (such as ReLU) have become more common.



# Other Activation Functions

# Hyperbolic Tangent Function

- Hyperbolic tangent function
- Pronounced “tanh”

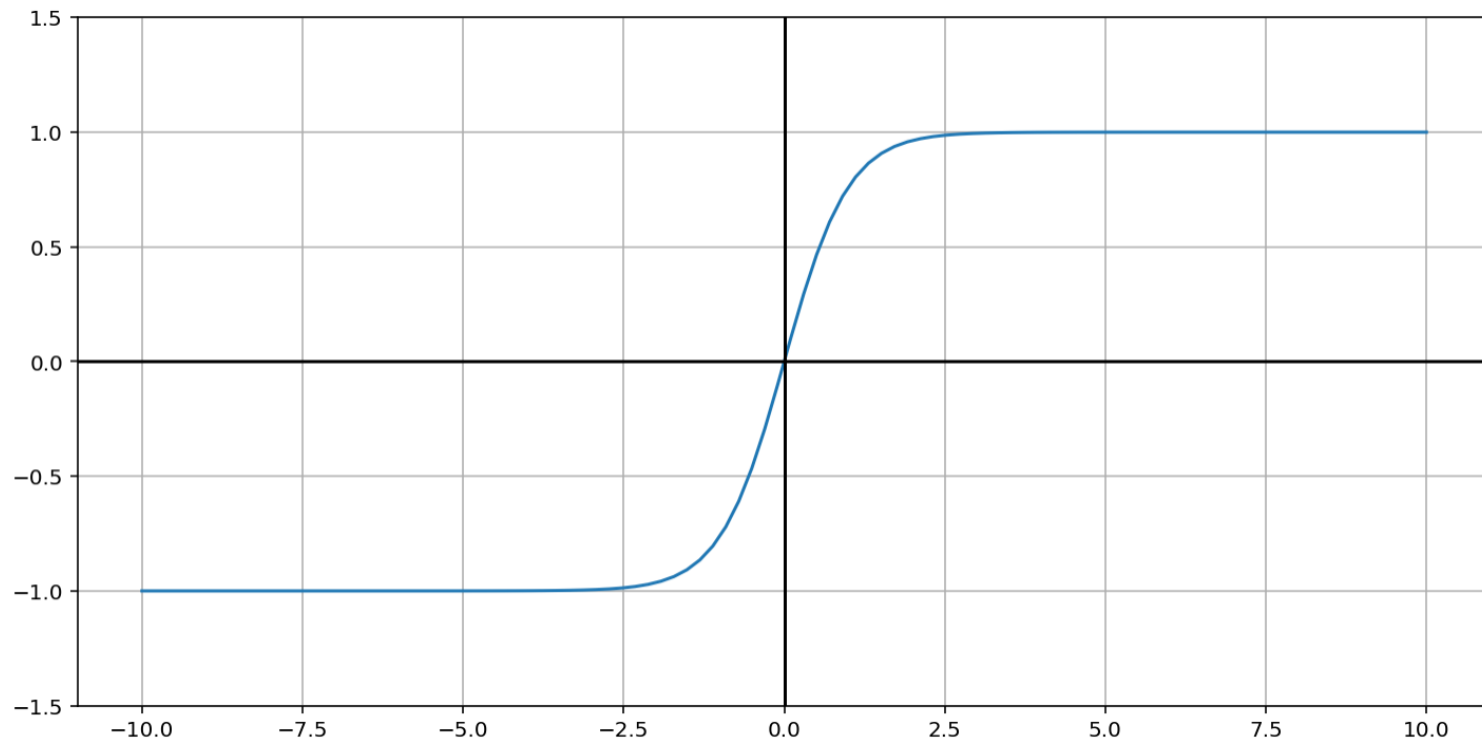
$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{2z} - 1}{e^{2z} + 1}$$

$$\tanh(0) = 0$$

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

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

# Hyperbolic Tangent Function



## Rectified Linear Unit (ReLU)

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

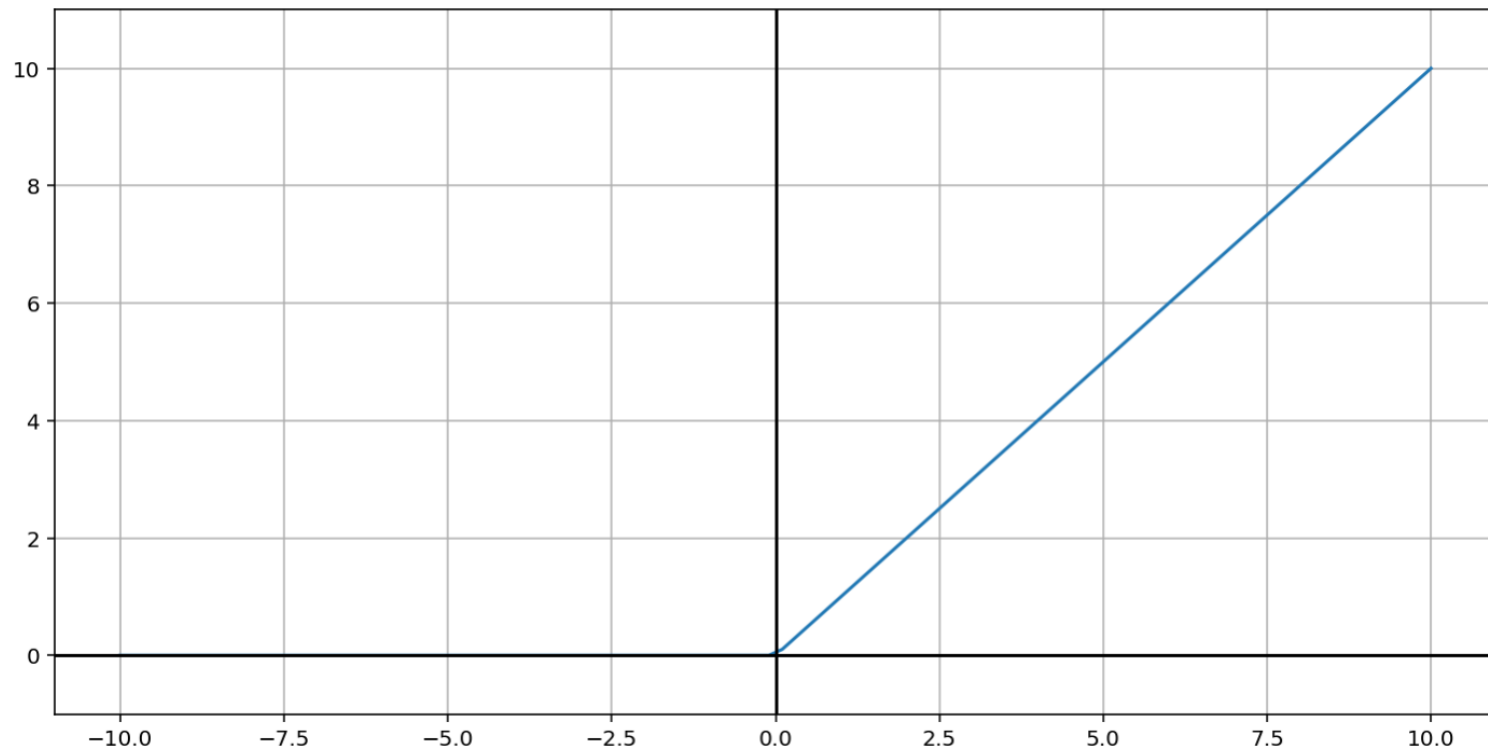
$$\text{ReLU}(0) = 0$$

$$\text{ReLU}(z) = z$$

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

for  $(z \gg 0)$

# Rectified Linear Unit (ReLU)



## “Leaky” Rectified Linear Unit (ReLU)

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

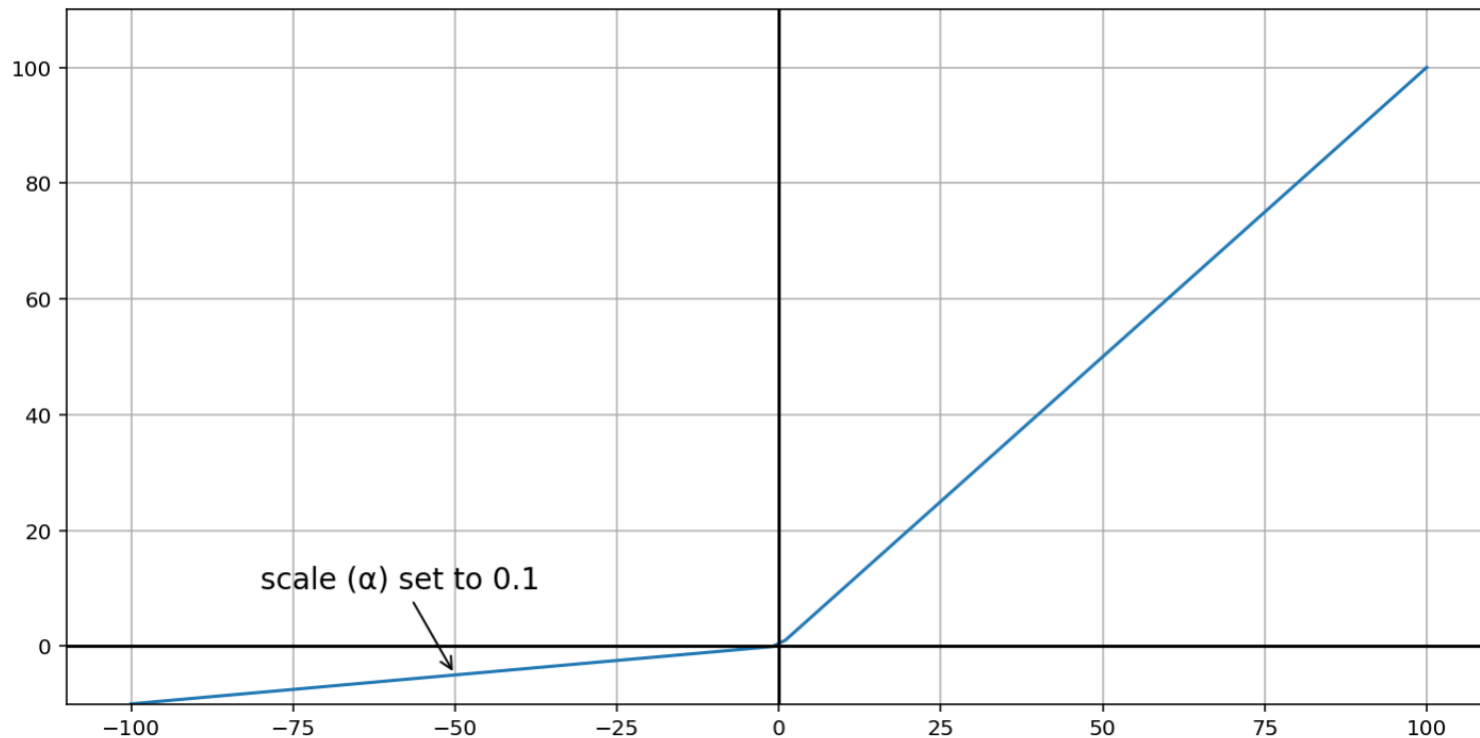
$$= (\alpha z, z) \quad \text{for } (\alpha < 1)$$

$$LReLU(0) = 0$$

$$LReLU(z) = z \quad \text{for } (z \gg 0)$$

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

# “Leaky” Rectified Linear Unit (ReLU)



# What next?

- We now know how to make a single update to a model given some data.
- But how do we do the full training?