intel Nervana™ AI Academy

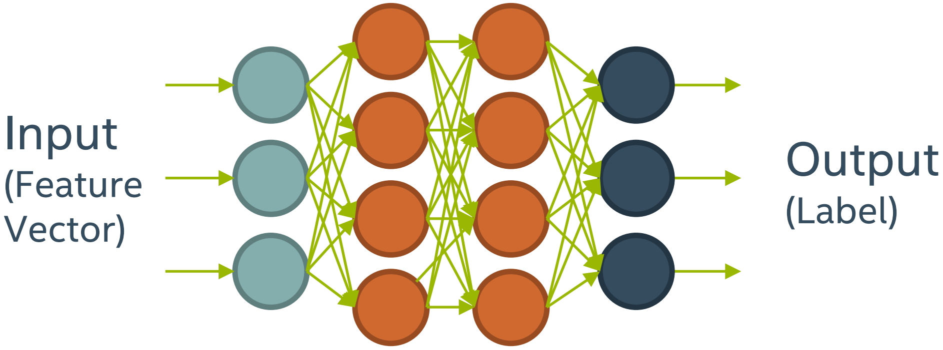
BACKPROPAGATION IN NEURAL NETS

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 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 BEFORE?

Gradient Descent!

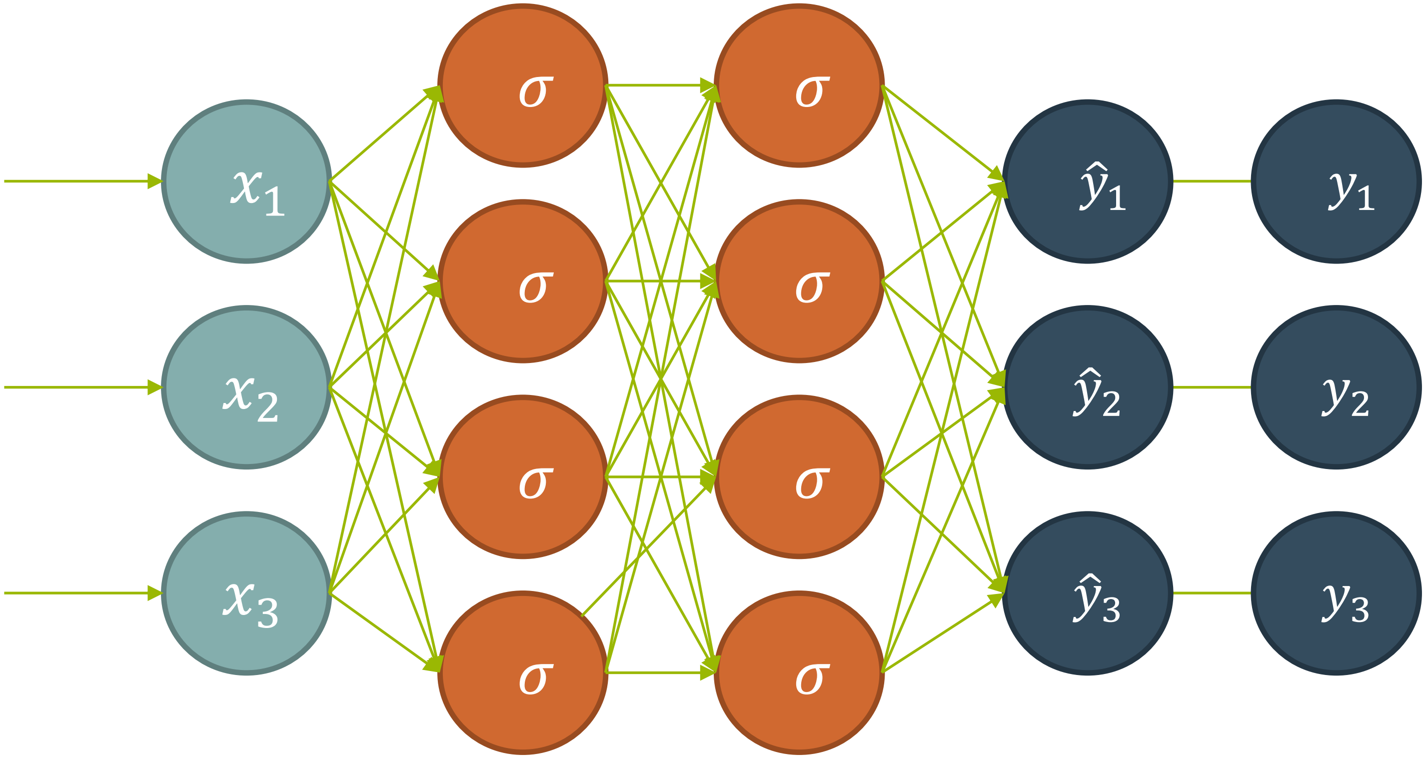
1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

HOW HAVE WE TRAINED BEFORE?

Gradient Descent!

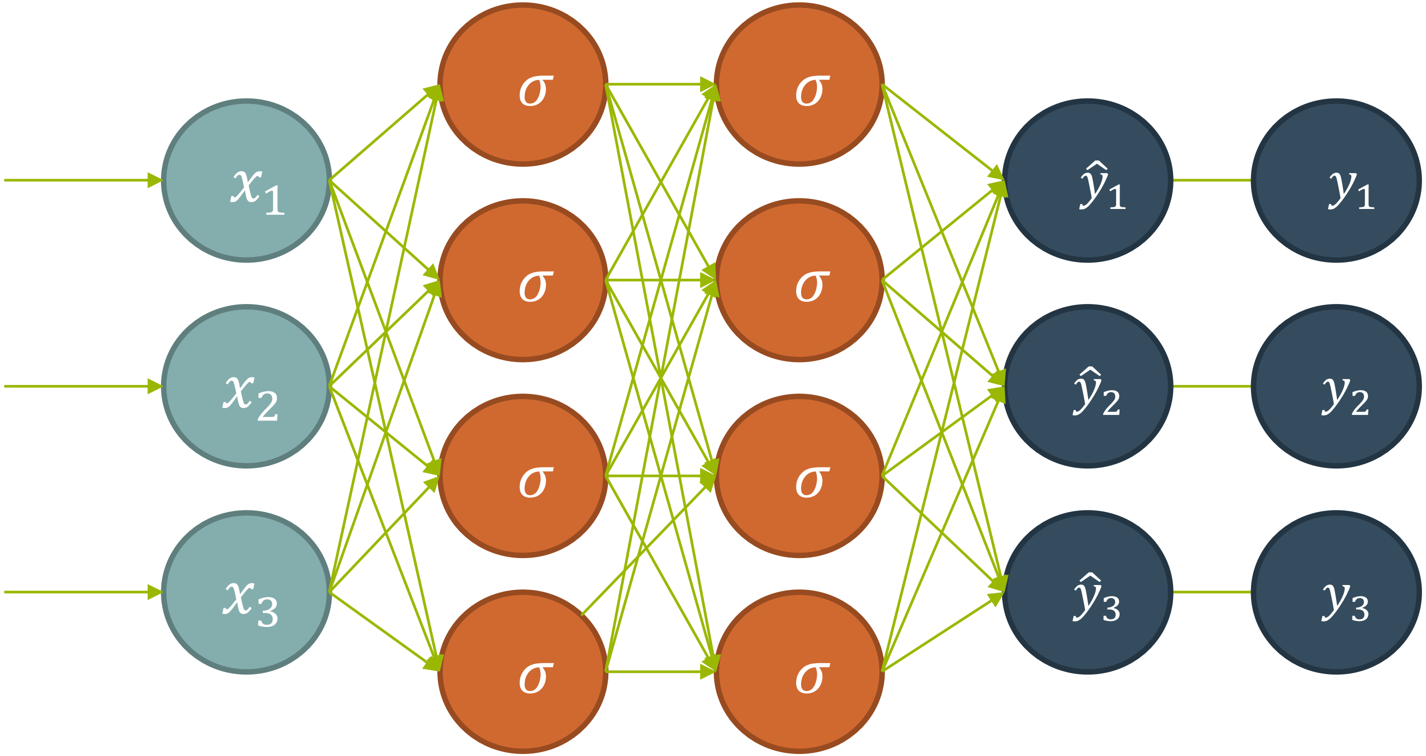
1. **Make prediction**
2. **Calculate Loss**
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

FEEDFORWARD NEURAL NETWORK



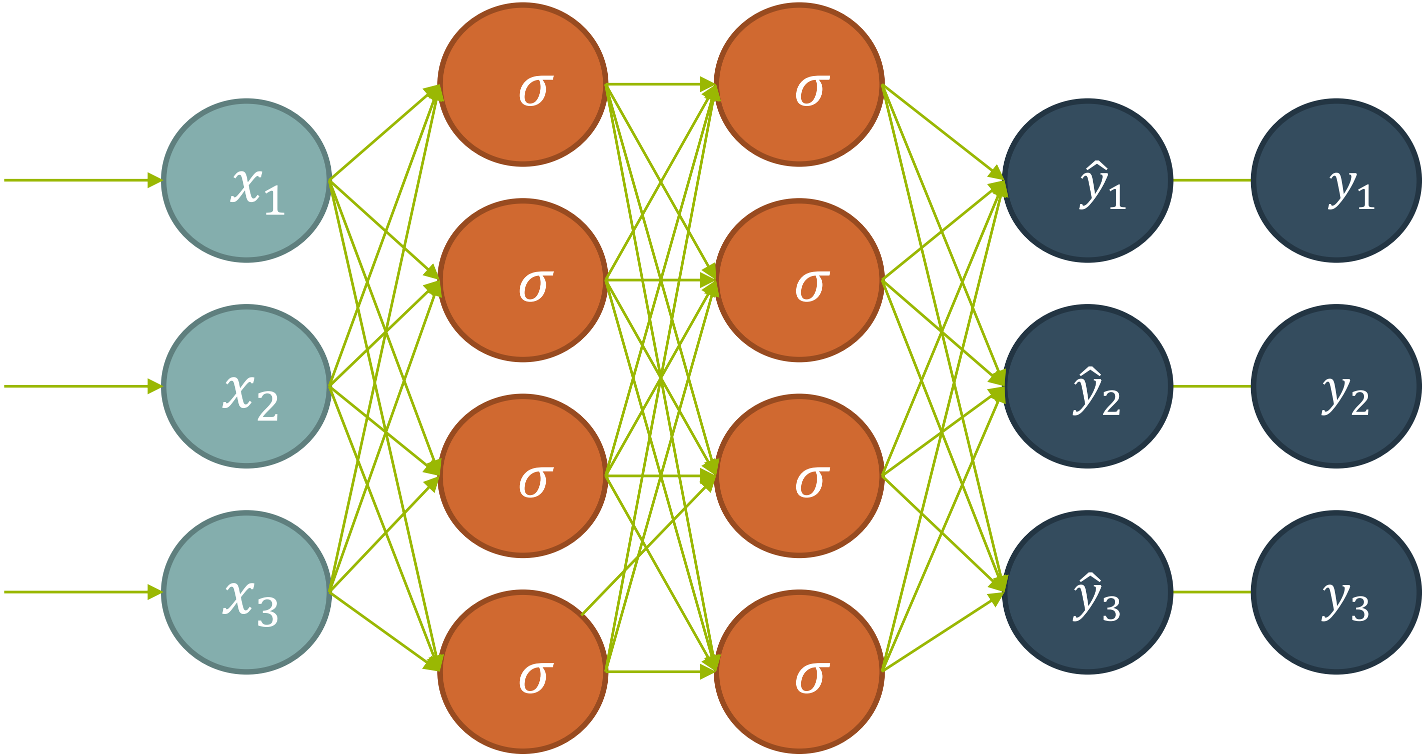
FORWARD PROPAGATION

Pass in Input

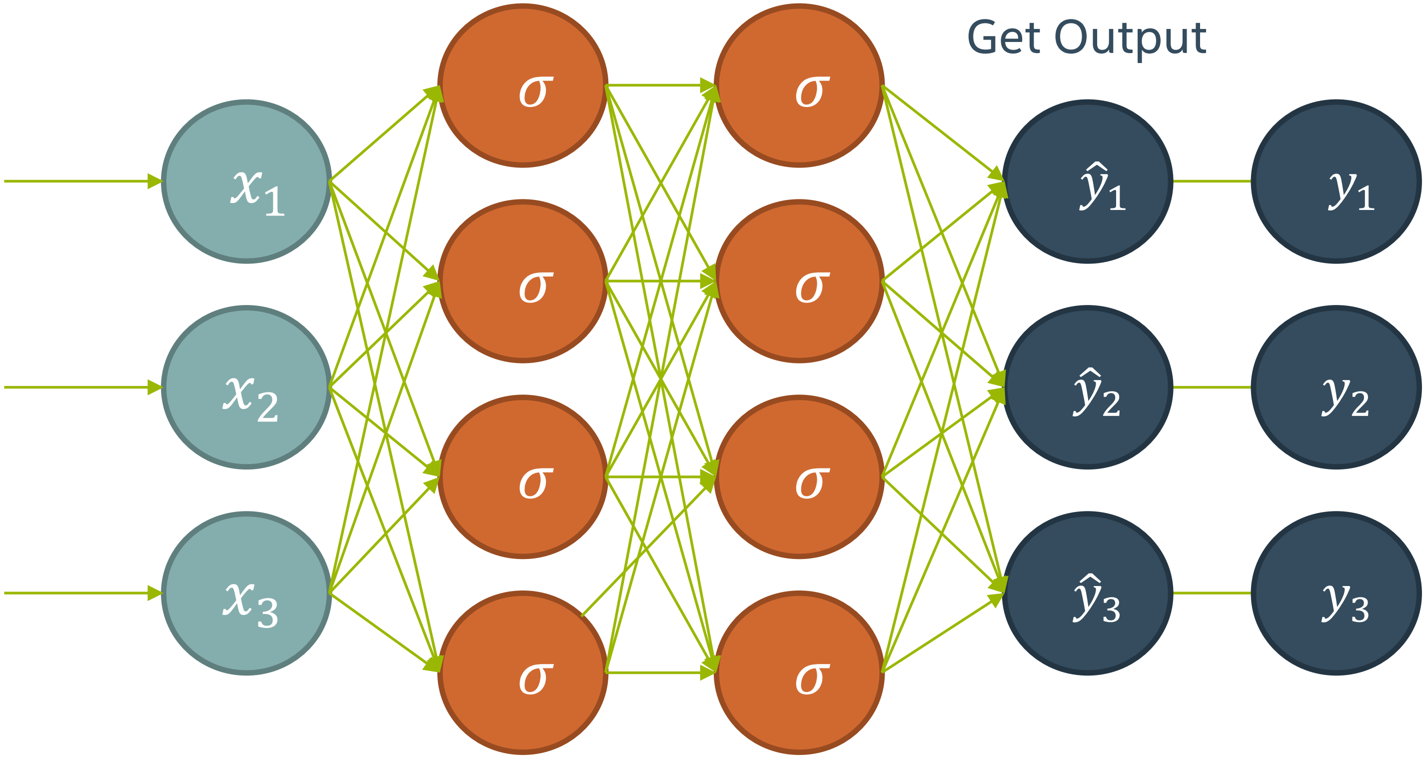


FORWARD PROPAGATION

Calculate each Layer

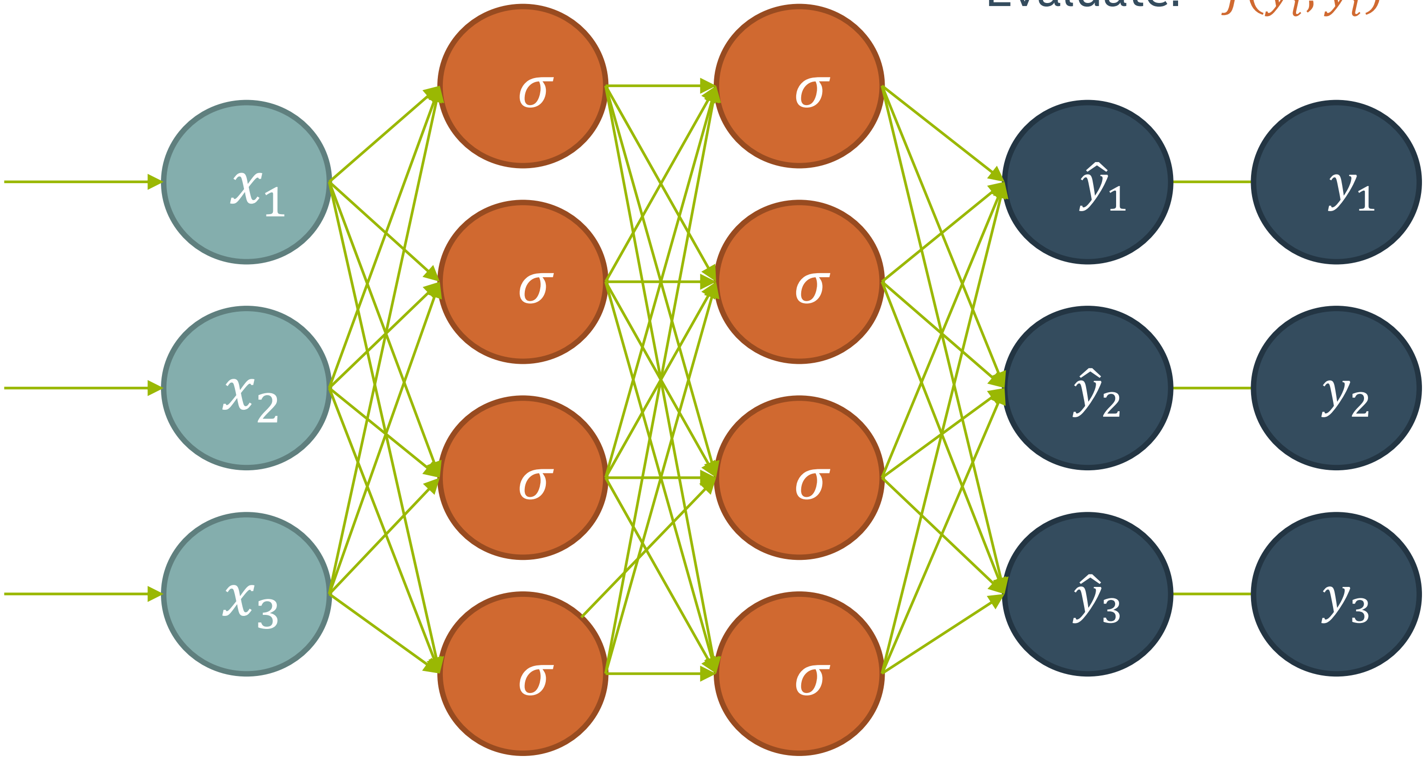


FORWARD PROPAGATION



FORWARD PROPAGATION

Evaluate: $J(y_i, \hat{y}_i)$



HOW HAVE WE TRAINED BEFORE?

Gradient Descent!

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

HOW TO CALCULATE GRADIENT?

Chain rule

HOW TO TRAIN A NEURAL NET?

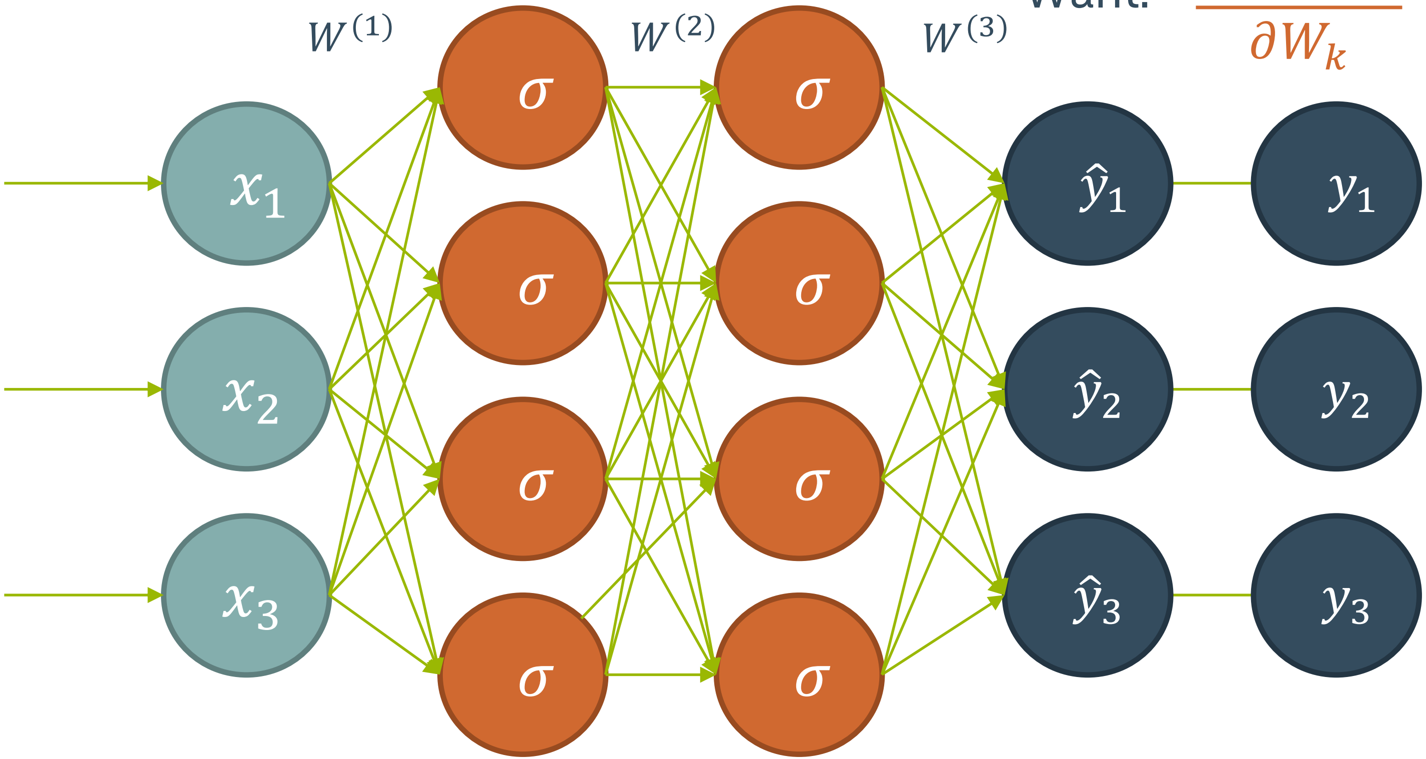
- How could we change the weights to make our Loss Function lower?
- Think of neural net as a function F: X -> Y
- F is a complex computation involving many weights W_k
- Given the structure, the weights “define” the function F (and therefore define our model)
- Loss Function is J(y,F(x))

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 Wk if we want to lower our loss function.
- Make an adjustment and repeat!

FEEDFORWARD NEURAL NETWORK

Want: $\dfrac{\partial J(y_i, \hat{y}_i)}{\partial W_k}$



CALCULUS TO THE RESCUE

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

PUNCHLINE

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

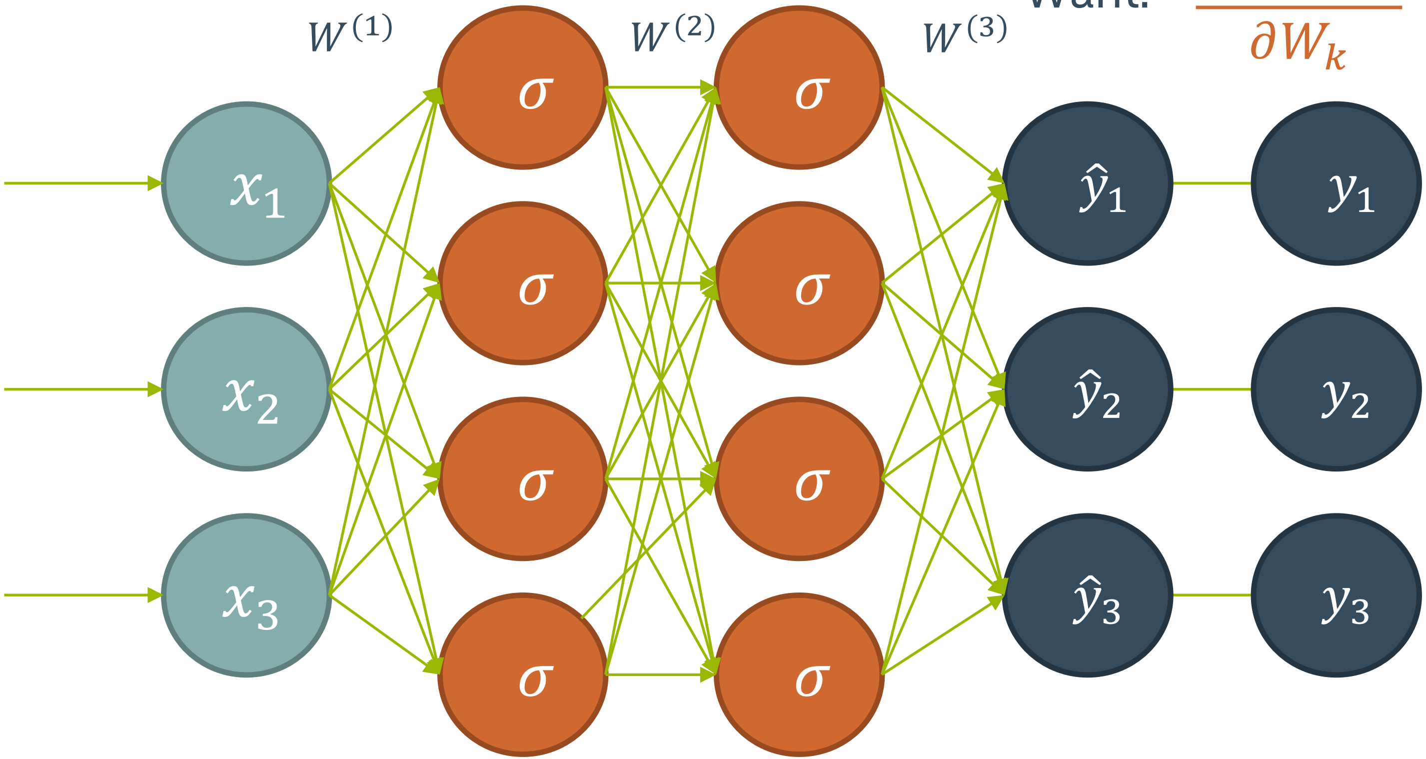
$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

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

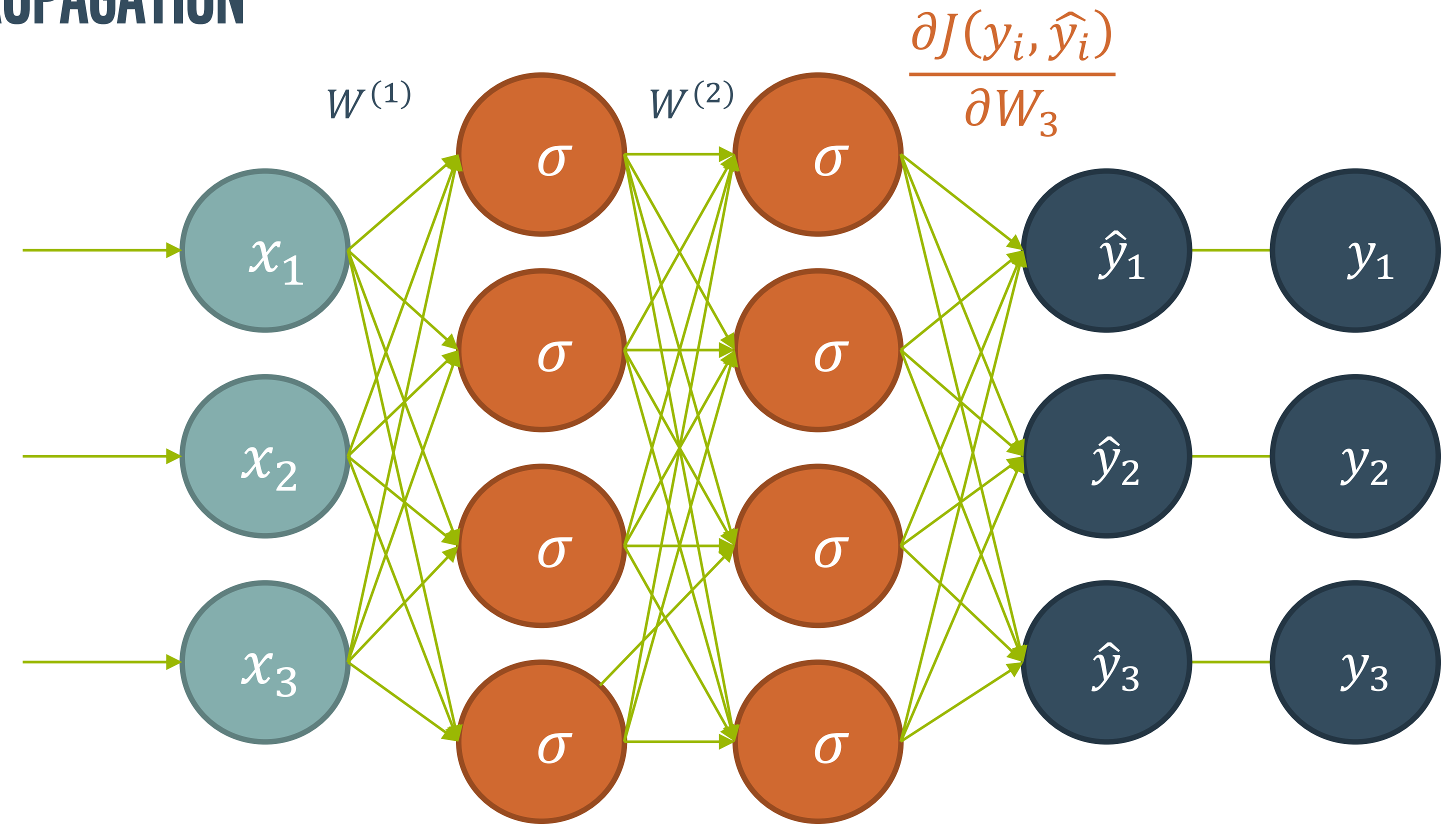
- Recall that: $\sigma'(z) = \sigma(z)(1-\sigma(z))$
- Though they appear complex, above are easy to compute!

BACKPROPAGATION

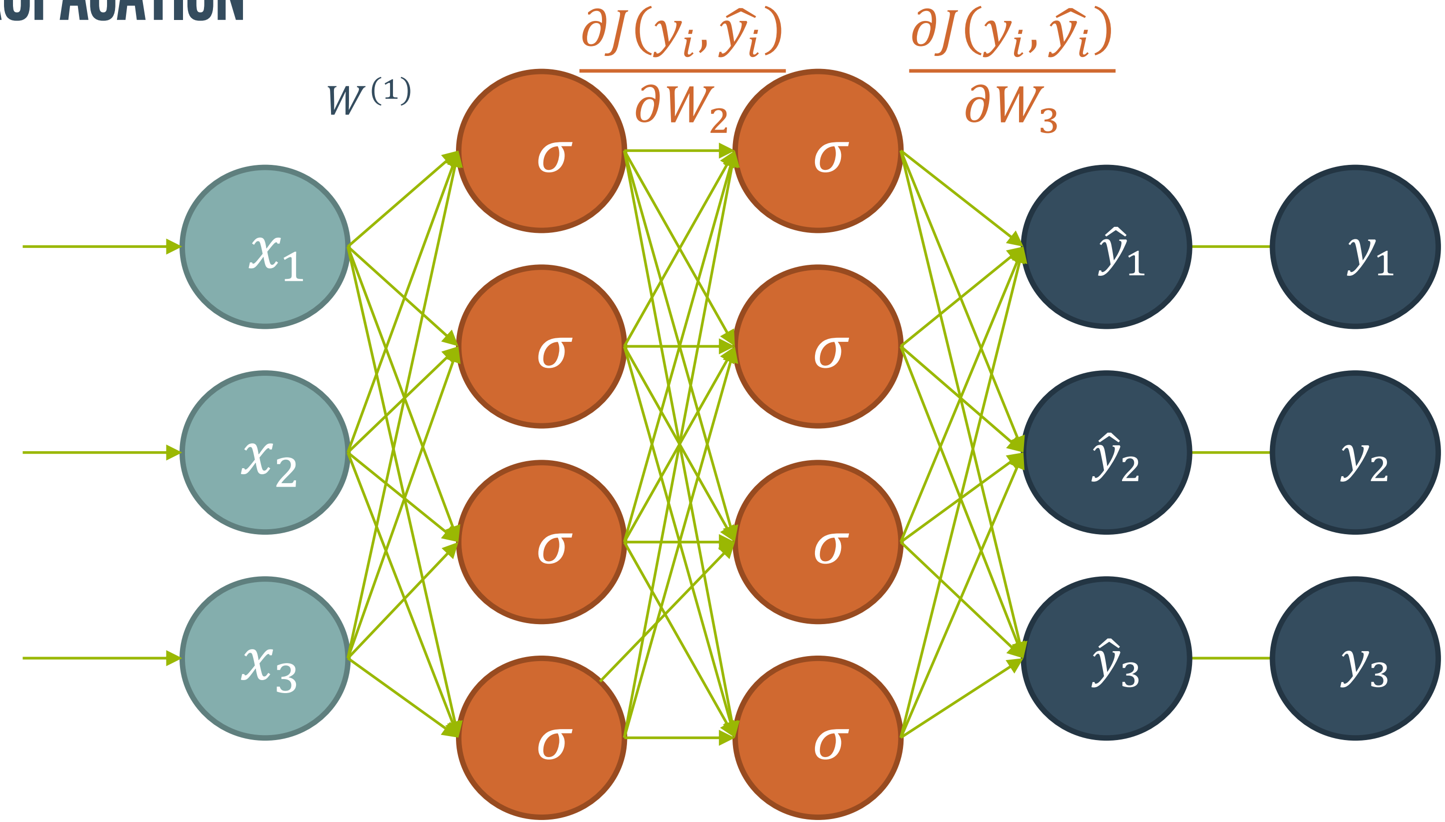
Want: $\dfrac{\partial J(y_i, \hat{y}_i)}{\partial W_k}$



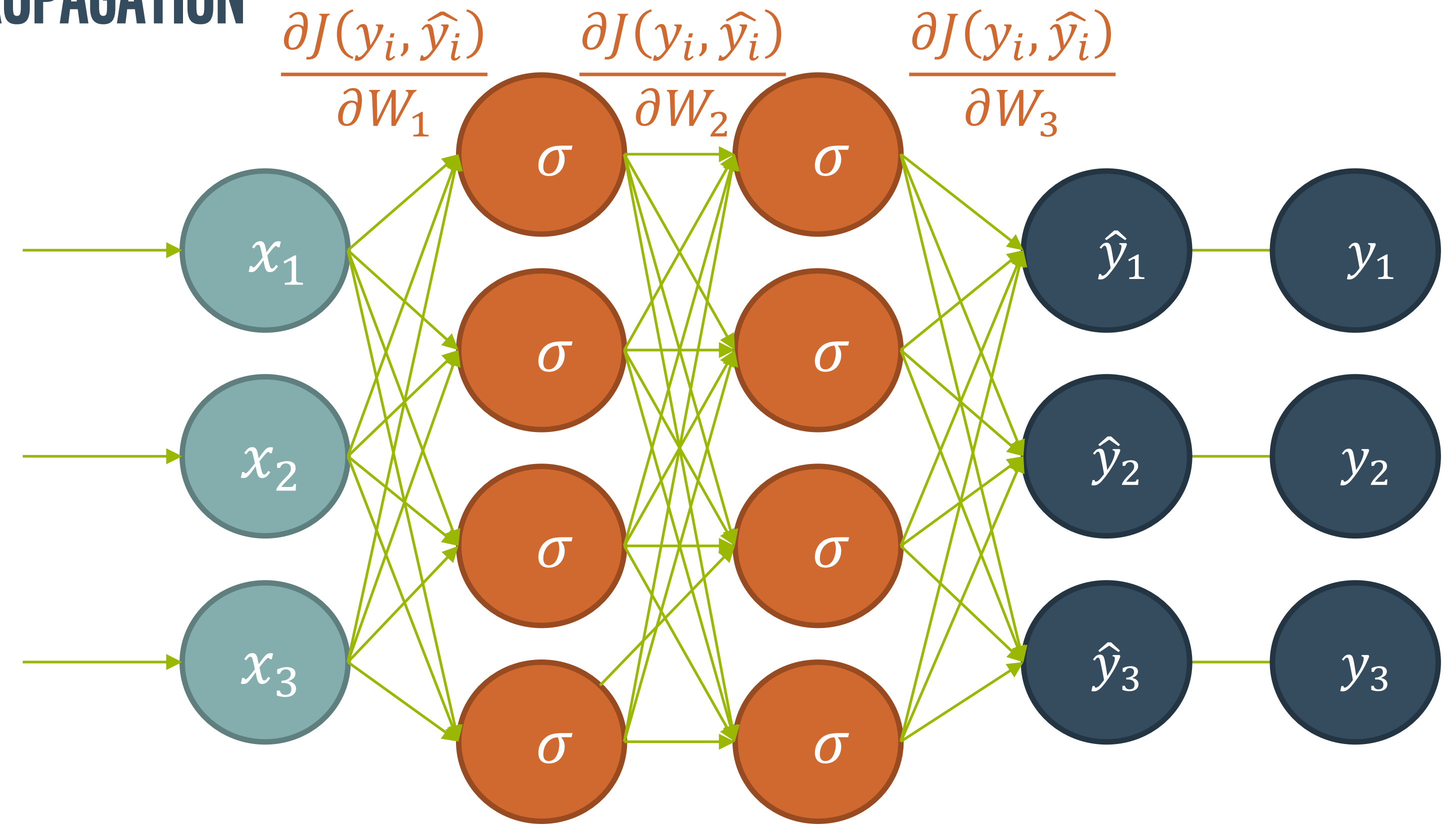
BACKPROPAGATION



BACKPROPAGATION



BACKPROPAGATION



HOW HAVE WE TRAINED BEFORE?

Gradient Descent!

1. Make prediction
2. Calculate Loss
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

Recall that:

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

- 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

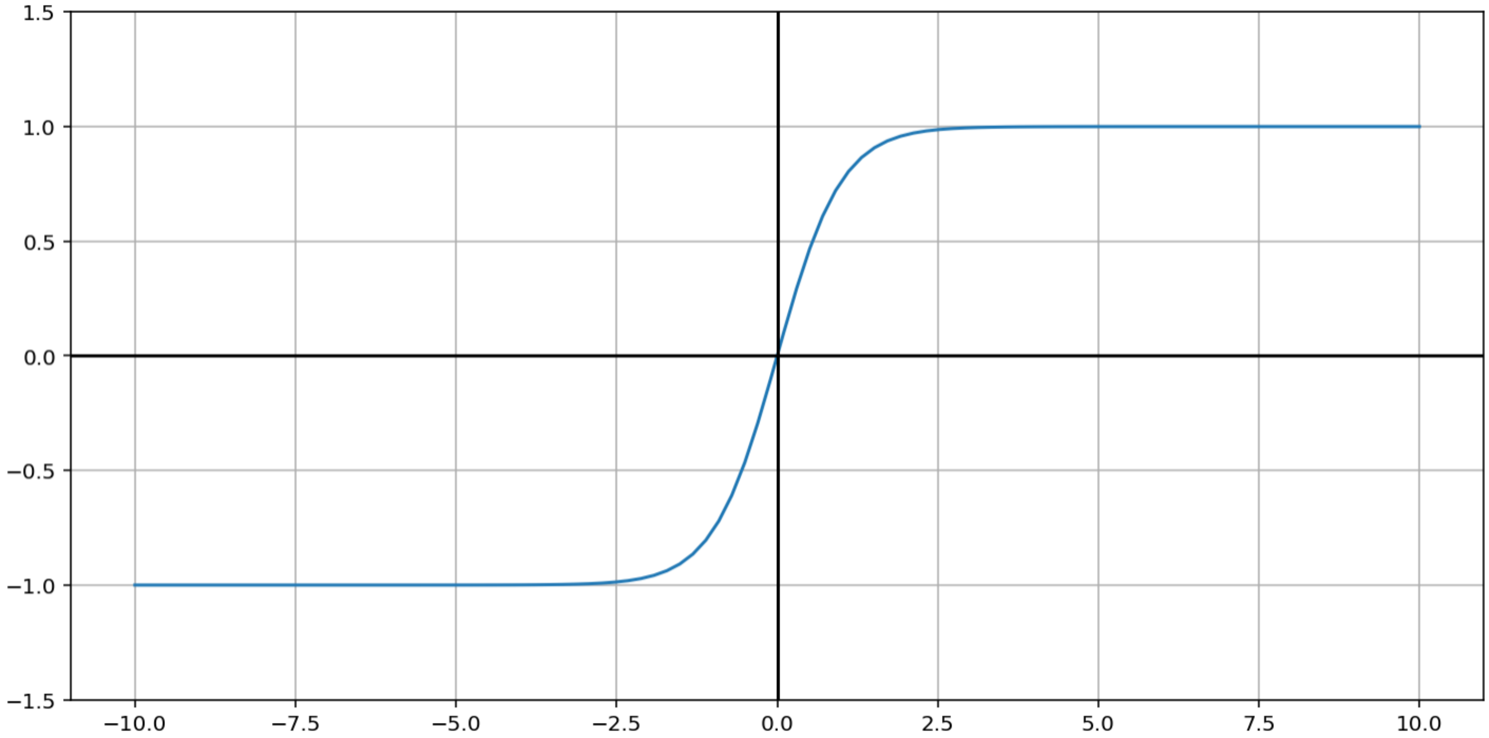
HYPERBOLIC TANGENT FUNCTION

- Hyperbolic tangent function
- Pronounced “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



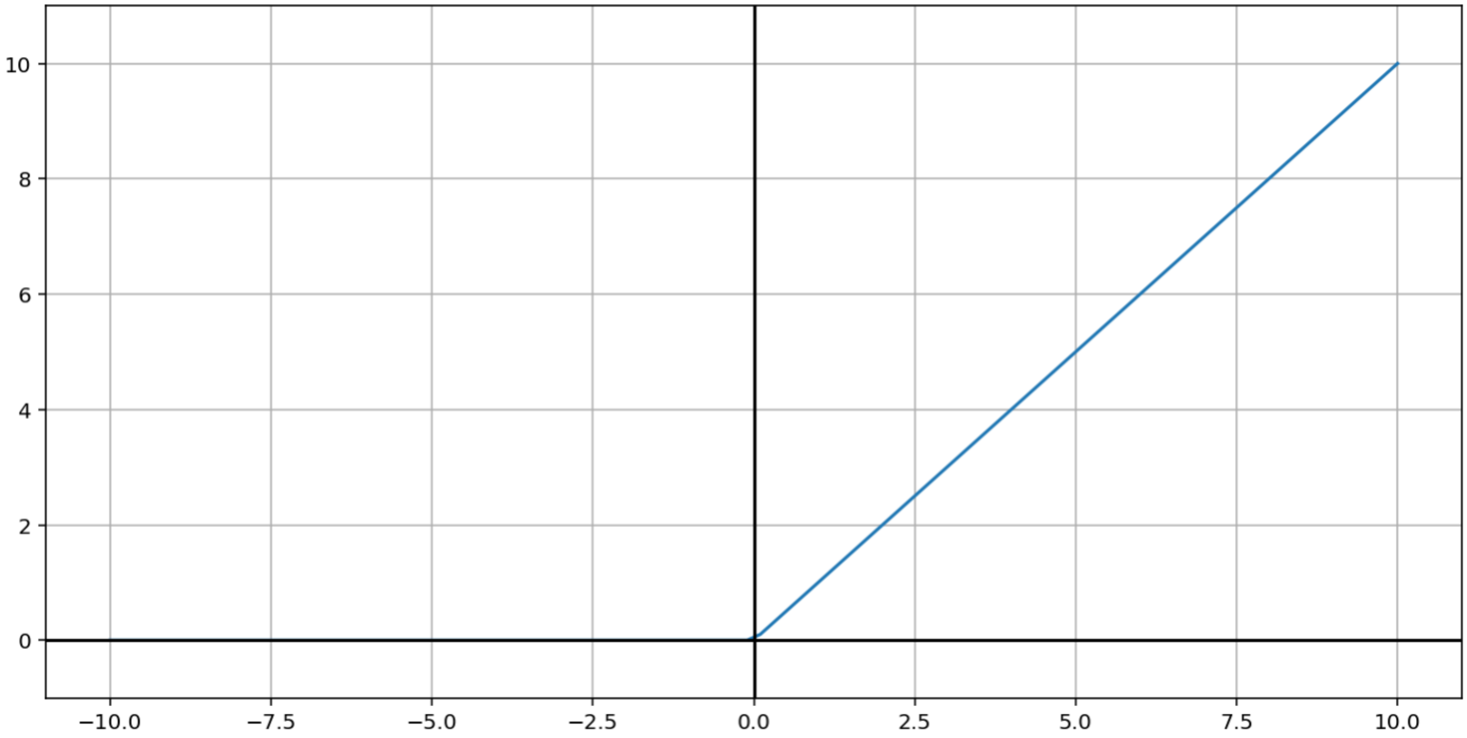
RECTIFIED LINEAR UNIT (RELU)

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

$$= \max(0, z)$$

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

RECTIFIED LINEAR UNIT (RELU)



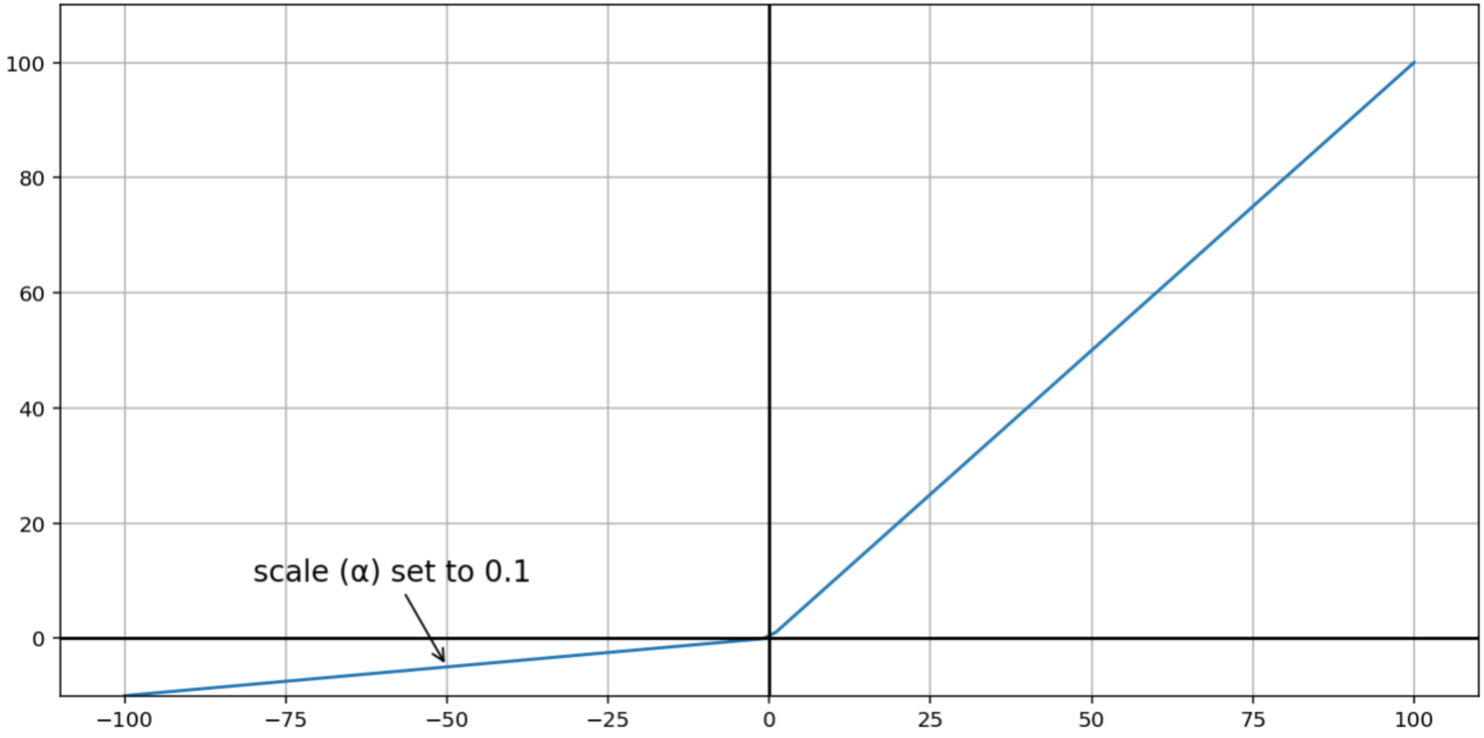
"LEAKY" RECTIFIED LINEAR UNIT (RELU)

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

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

$$LReLU(0) = 0$$
$$LReLU(z) = z \qquad \text{for } (z \gg 0)$$
$$LReLU(-z) = -\alpha z$$

"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?

We will dive into these details in the next lecture.

Nervana™ AI Academy