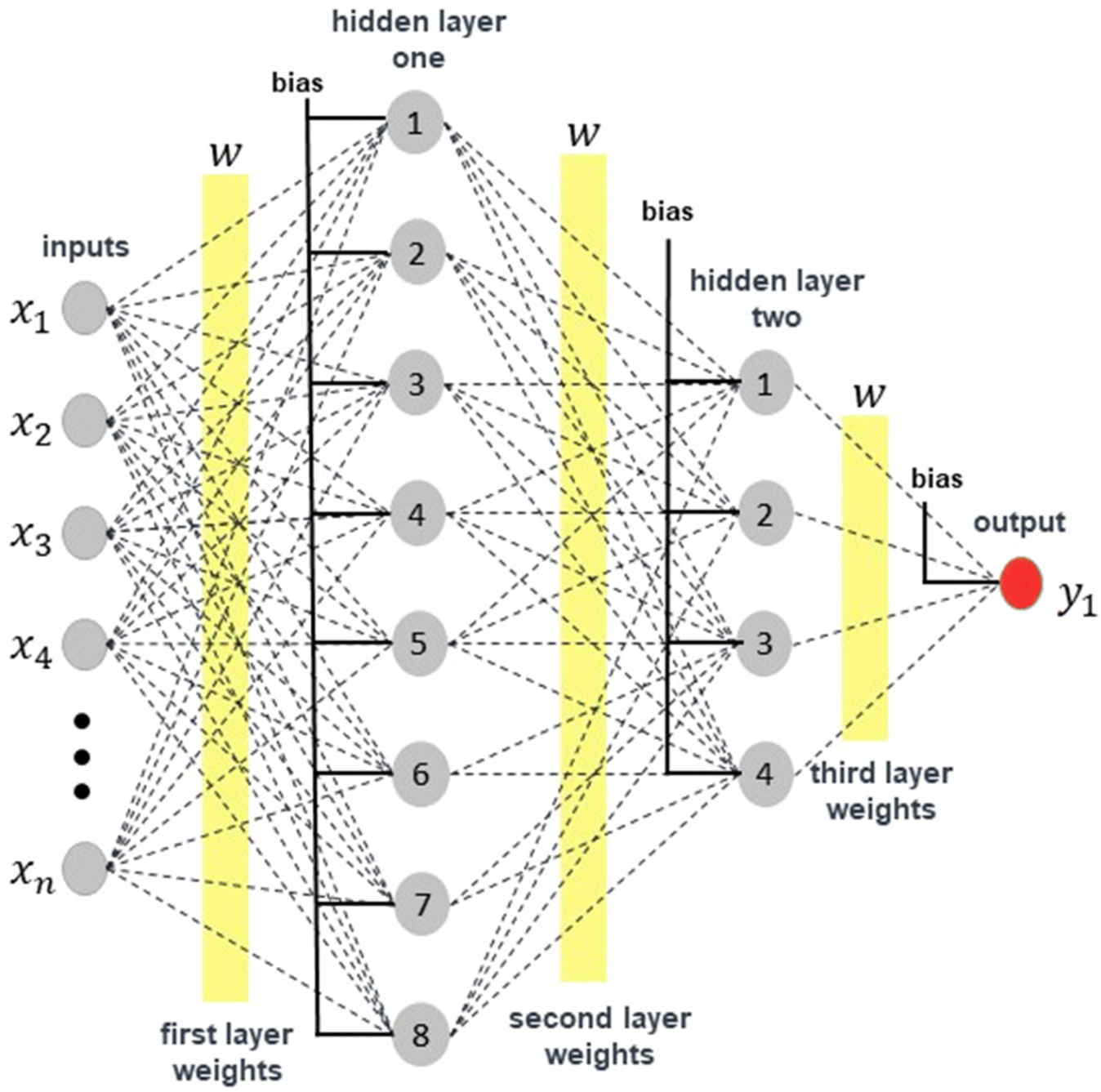
Neural Network Math

1. Structure



1.1. Layers

Assume:

- 784 input nodes (a flatten array (list) from a image which is 28x28)
- 200 hidden nodes
- 10 output notes (representing 0, 1, 2, 3, 4, 5, 6, 7, 8 9)

2. Notation

- input nodes = $x_i \; where \; i = 1, 784$; in matrix form: $\boldsymbol{X}$ which is a 784 x 1 vector
- hidden notes = $h_j \; where \; j = 1, 200$; in matrix form: $\boldsymbol{H}$ which is a 200 x 1 vector
- output notes = $o_k \; where \; k = 1, 10$; in matrix form: $\boldsymbol{O}$ which is a 10 x 1 vector
- target = y: given a particular input X where X represents the entire input vector for one training input. 10 x 1 vector
- calculated target = $\hat{y}$: network calculated value for input X. 10 x 1 vector
- learning rate = lr: a small constant (a scale factor) which adjusts the weights on each

training pass. scalar.
- bias = b: an addition constant weight added to each layer (e.g., x_0, h_0). scalar. We are going to ignore this for now.

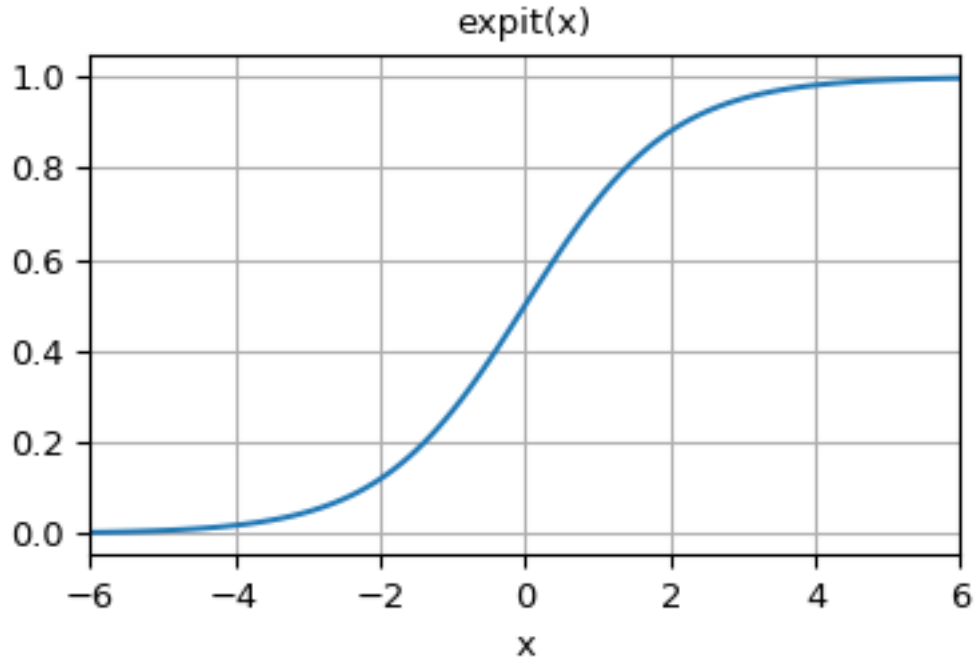
2.1. Weights (linkages between layers)

- W_{ij} represents the link between Node i in input layer to Node j next layer
- *wih* is the weight matrix between the input and hidden layers. wih = 200 x 784
- *who* is the weight matrix between the hidden layer and the output layers. who = 10 x 200

3. Activation function -- the *Sigmoid function*

$g(x) = \frac{1}{1+e^{-x}}$ This is used to normalize the output of each neuron between 0 and

1.



4. Hypothesis: Calculating $\hat{y}$

- $h_j = g(\sum_{i=1}^{784} w_{ij} * x_i)$ *or* $H = g(X * wih.T)$ where ".T" means transpose
- $\hat{y}_j = g(\sum_{i=1}^{200} w_{ij} * h_i)$ *or* $\hat{y} = g(H * who.T)$

- Therefore: $\hat{y} = g(g(X * wih.T) * who.T)$ This is also known as *forward propagation*.

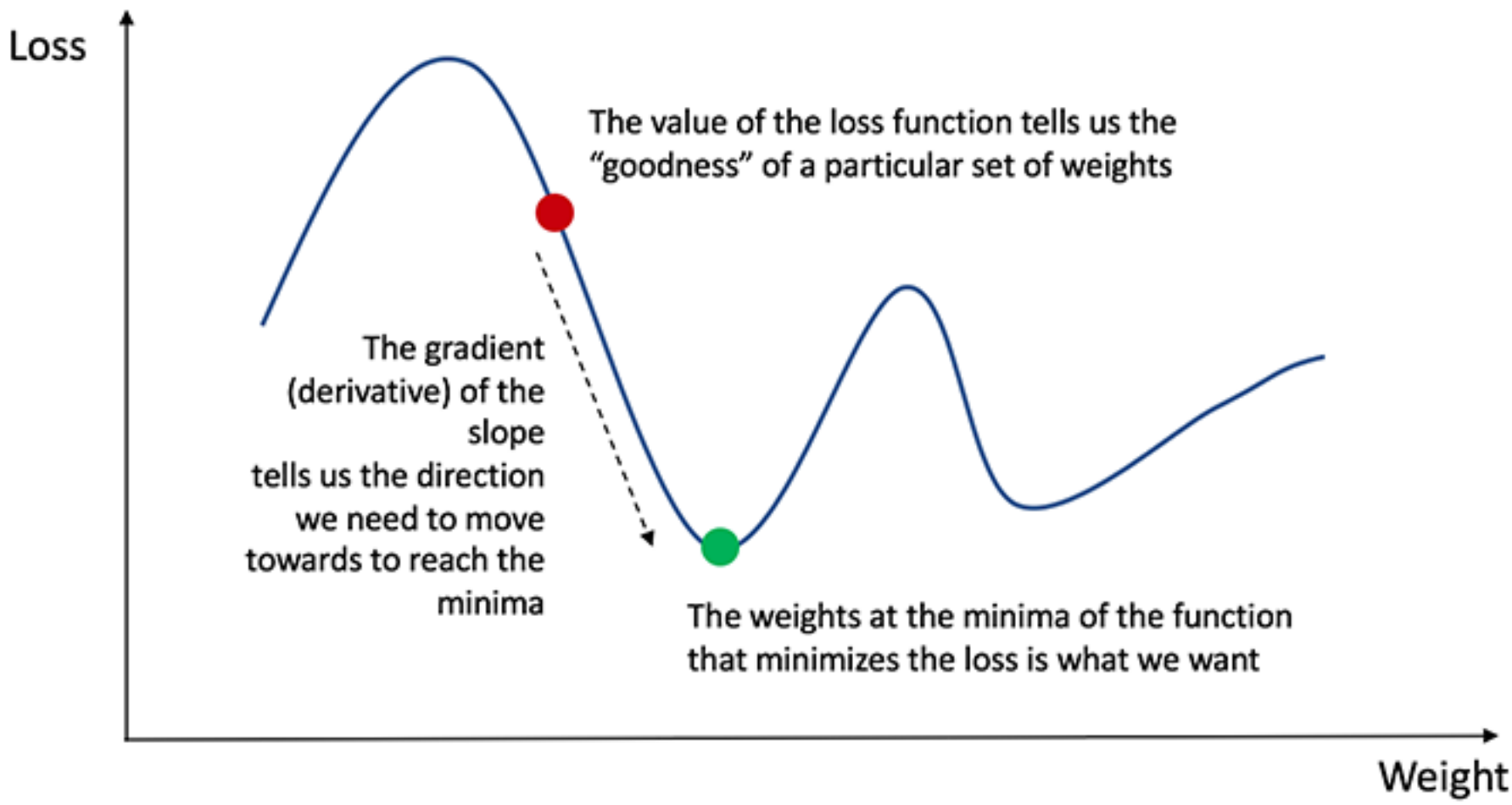
5. Error Calculation

There are many available loss functions, and the nature of our problem should dictate our choice of loss function. The usual error function used is a sum-of-sqaures error function.

$$Loss(y, \hat{y}) = \sum_{i=1}^{n} (y - \hat{y})^2$$

$(y - \hat{y})$ is a 10 x 1 vector and it is the amount of error associated with each output node

The goal is to minimize this function. We do this by figuring out which the direction is lower by looking at the *gradient* or derivative of this function and stepping toward a lower value. This is done by adjusting the weights w_{ij} by the a small amount (the learning rate lr) based on how much of the error is associated with this particular weight. This is determined via the partial derviative of the particular weight to the hypothsis equation.



However, we cannot directly calculate the derivative of the loss function with respect to the weights and biases because the equation of the loss function does not contain the weights and biases. Therefore, we need the chain rule to help us calculate it.

$$\begin{aligned}\frac{\partial Loss(y,\hat{y})}{\partial W} &= \frac{Loss(y,\hat{y})}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial W} \; where \; z = Wx + b \\ &= 2(y - \hat{y}) \times derviative \; of \; sigmoid \; function \times x \\ &= 2(y - \hat{y}) \times z(1 - z) \times x\end{aligned}$$

This has to be done for each of the two layers. You start with the error between the output layer and the hidden layer adjusting the values (weights) of who. Then between the hidden layer and the input layer adjusting the values (weights) of wih. This is called *back propagation*. The calculation above was only for a single layer. Specifically,

$$wih = wih + deltawih.T$$

$$who = who + deltawho.T$$

$$where$$

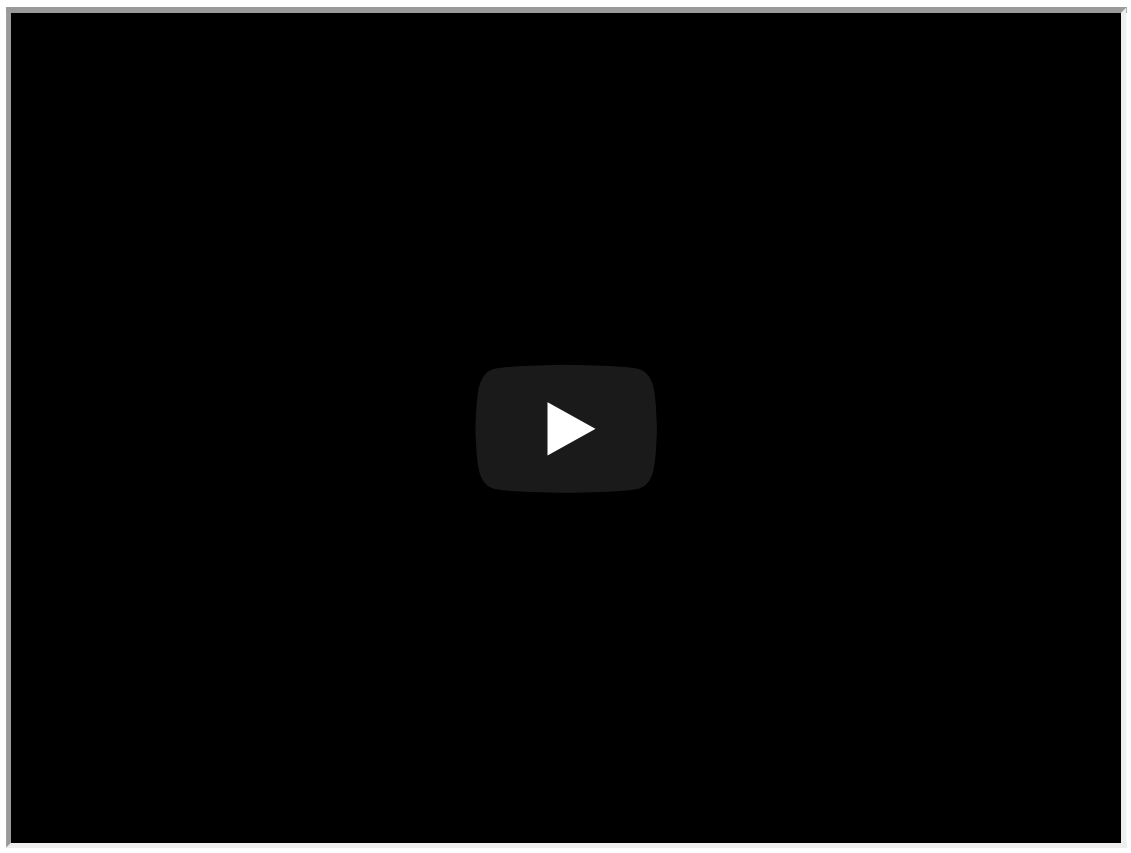
$$deltawho = lr \times (H.T * (2(y - \hat{y})\hat{y}\prime))$$

$$deltawih = lr \times [X.T * (((2(y - \hat{y})\hat{y}\prime) * who) \times H\prime)]$$

And to calculate the derviative of the sigmoid function:

$$\begin{aligned}
\frac{dg(x)}{dx}\frac{1}{1+e^{-x}} &= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{e^{-x}+1-1}{(1+e^{-x})^2} \\
&= \frac{e^{-x}+1}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} \\
&= \frac{1}{(1+e^{-x})} - \frac{1}{(1+e^{-x})^2} \\
&= g(x) - g(x)^2 \\
&= g(x)(1-g(x))
\end{aligned}$$

Watch this video for more background on the calculus:



6. Training the network

```
For each record in the training set X and each expected value (target) y:

Do while training examples {

   forwardProp(X)

   backProp(X,y)

}
```

7. Predicting once the network is trained

Simply (a) read in 28x28 picture and put into scalar form (784 x 1). (b) Then call the function $\hat{y} = g(g(X * wih.T) * who.T)$ where wih and who are the saved values from the training. The function would be something like:

def guess(self, X):

$$\hat{y} = g(g(X * wih.T) * who.T)$$

return $\hat{y}$ # which is a 10 x 1 array with a single 1 in an array element. The index to that 1 is the number predicted.