Neural Networks

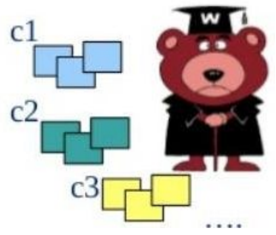
Dhaval Lunagariya

Agenda

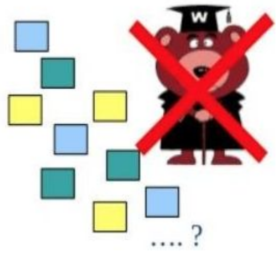
- Machine Learning Basics
- Supervised vs Unsupervised Learning
- AI vs ML vs DL
- Slope of Line
- Differentiation Chain Rule
- Components of NN
- Forward Propagation
- Loss Function
- Backward Propagation
- Forward Propagation
- Bias
- Learning Rate

Supervised vs Unsupervised

- **Supervised**
  - **Knowledge of output -** learning with the presence of an “expert” / teacher
    - Data is **labelled** with a class or value
    - **Goal :** predict class or vale
    - Eg. Neural Network, Support Vector machine, Decision Trees, Classification

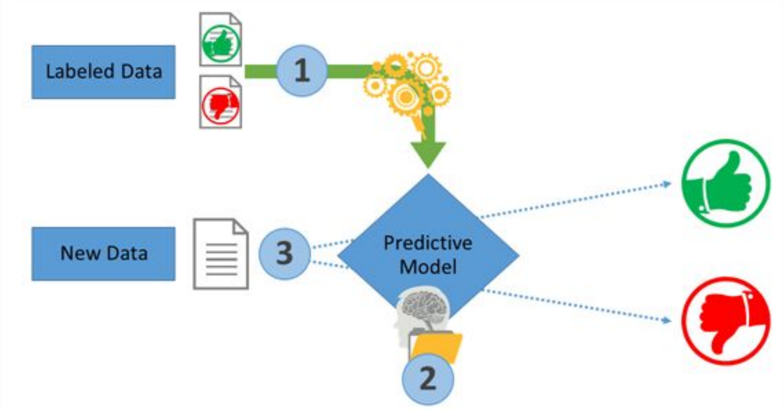


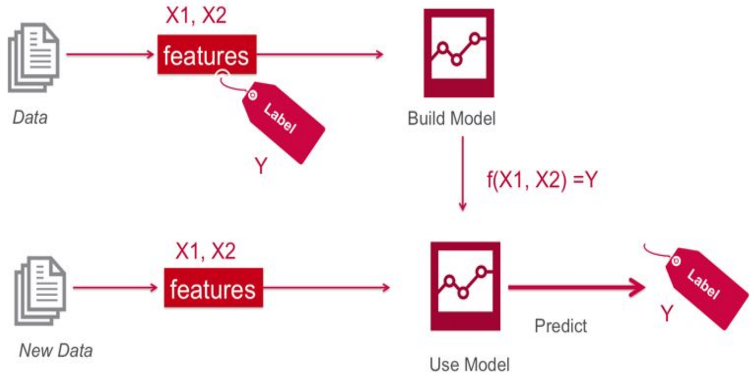
- **Unsupervised**
  - **No knowledge of output** class or value
    - Data is **unlabelled** or value unknown
    - Goal : Determine data patterns/groupings
  - Self-guided learning algorithm
    - Internal self-evaluation against some criteria
    - Eg. k-means, clusterring



source : researchgate.net

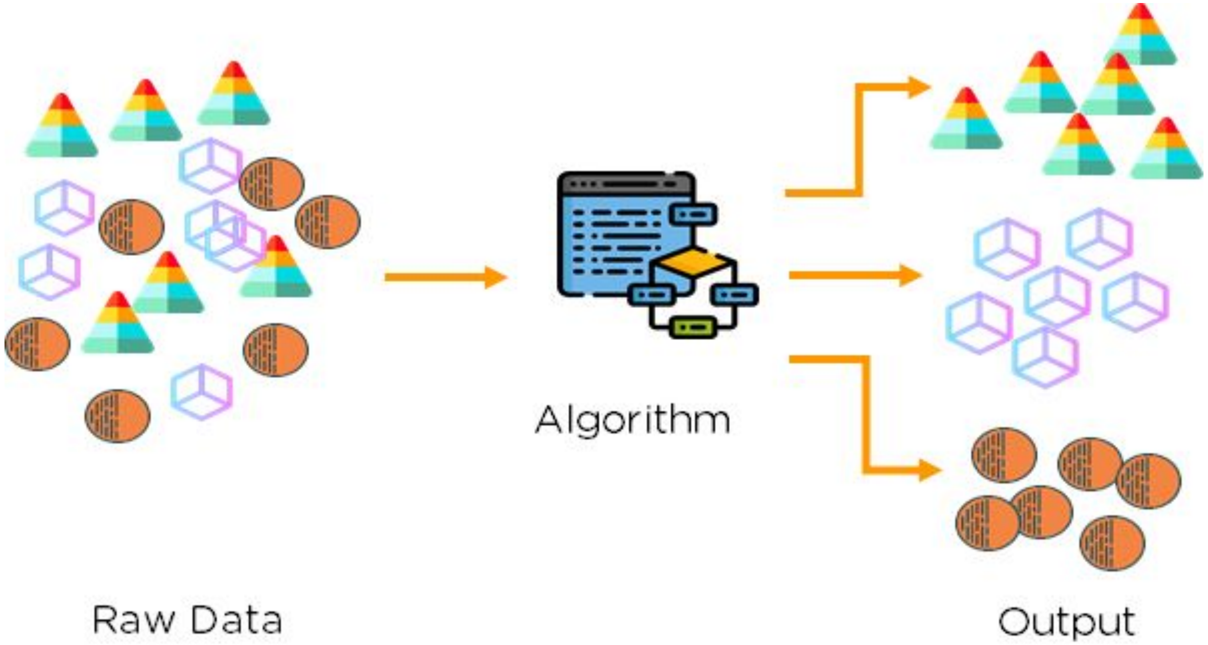
Supervised Machine Learning



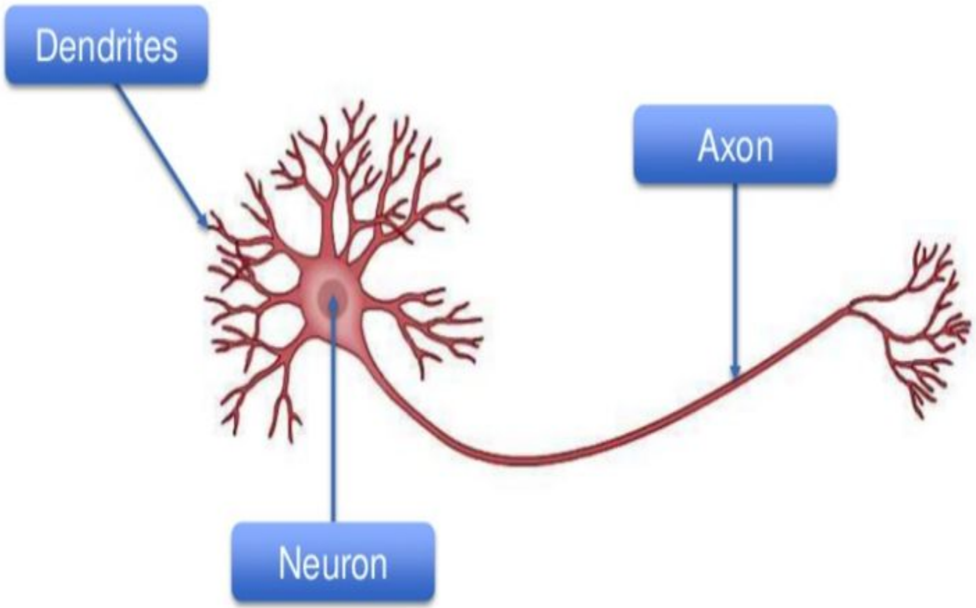


source: symantec.com | mapr.com

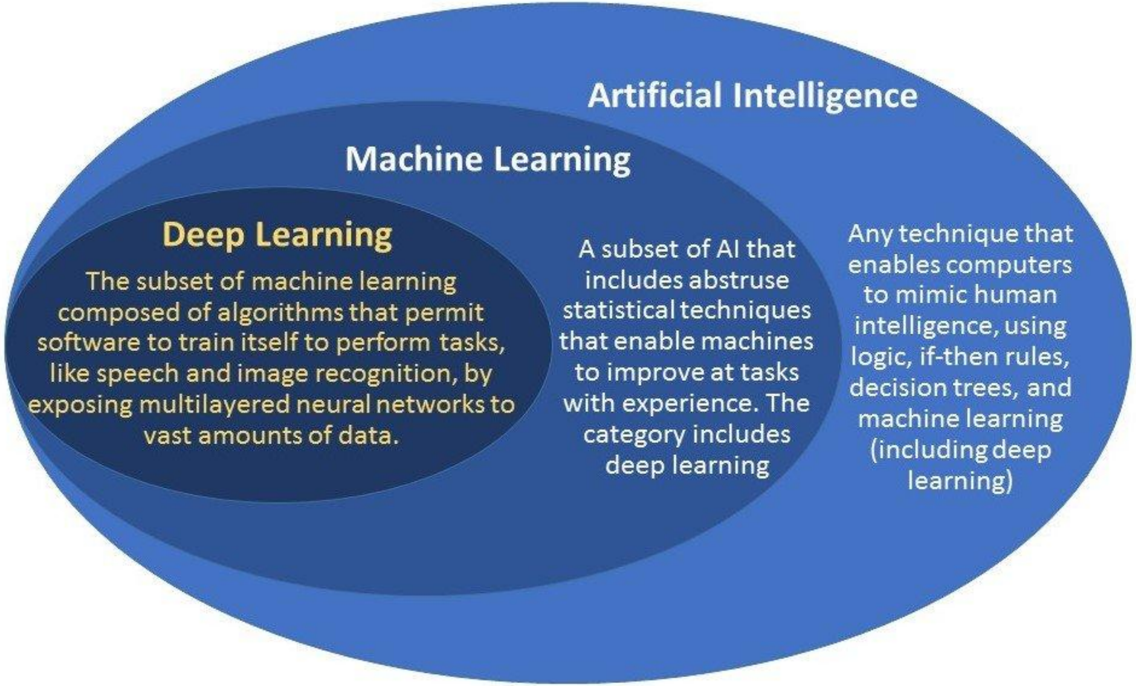
Unsupervised Learning



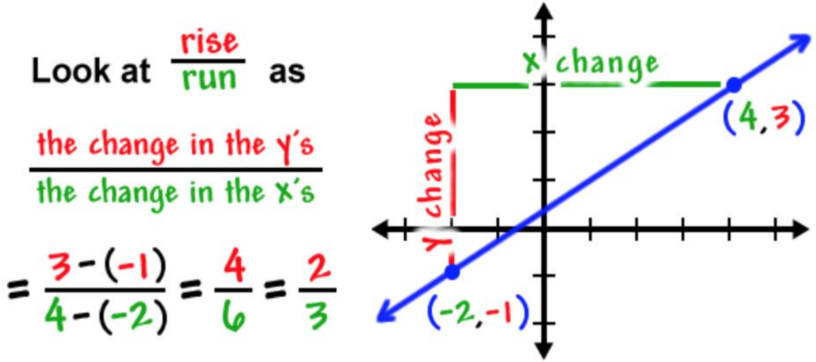
Human Brain

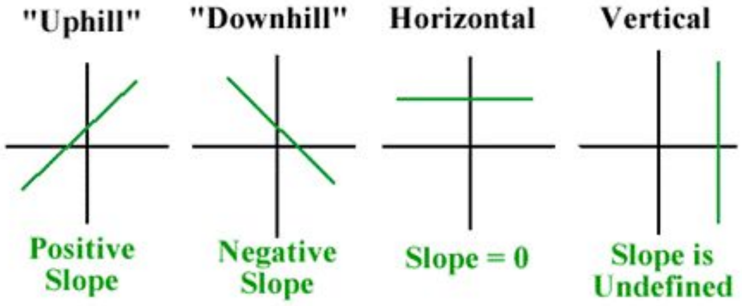


AI vs ML vs DL


Artificial Intelligence
Machine Learning
Deep Learning
The subset of machine learning composed of algorithms that permit software to train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data.
A subset of AI that includes abstruse statistical techniques that enable machines to improve at tasks with experience. The category includes deep learning
Any technique that enables computers to mimic human intelligence, using logic, if-then rules, decision trees, and machine learning (including deep learning)

Slope of Line





Chain Rule

If f and g are both differentiable and $F(x)$ is the composite function defined by $F(x) = f(g(x))$ then F is differentiable and F' is given by the product

$$F'(x) = f'(g(x))\, g'(x)$$

Differentiate outer function

Differentiate inner function

$$\frac{d}{dx}\left[f(g(x))\right] = \overbrace{f'(g(x))}\cdot\overbrace{g'(x)}$$

$$y' = f'(u)\cdot u'$$

Example: $y = (x^2+1)^3 \Rightarrow y = (\overset{g(x)}{x^2+1})^3$

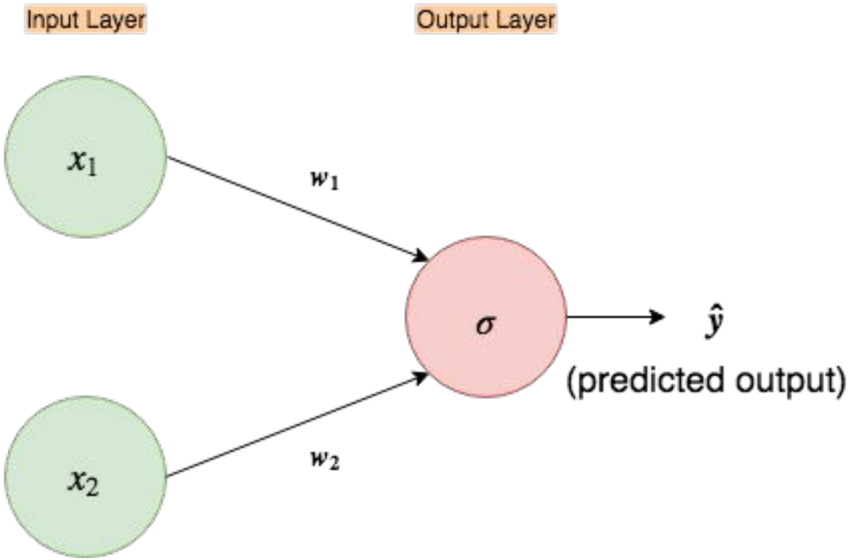
$$y = (t)^3$$

$$y = 3(t)^2\cdot(2x)$$

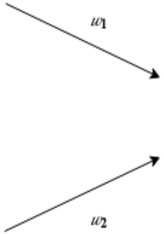
$$y = 3(x^2+1)^2\cdot(2x)$$

$$\to y = 6x(x^2+1)^2$$

Simple input-output neural network

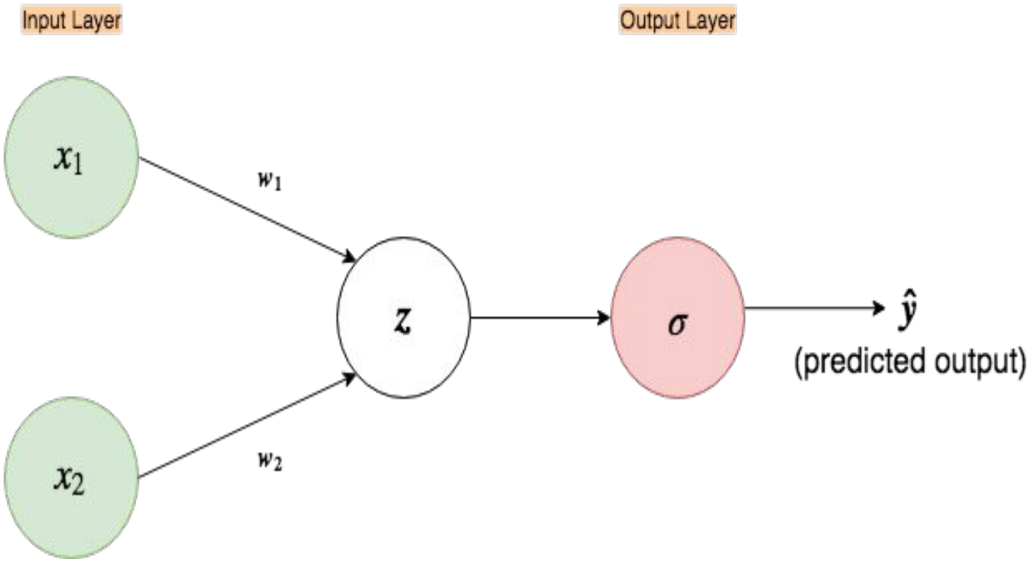


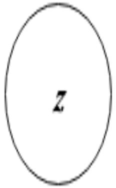
Input nodes



Weights

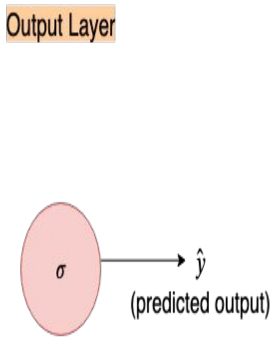
Expanded neural network





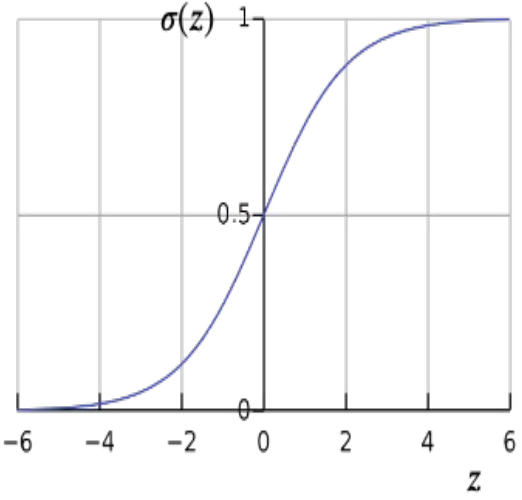
$$z = w_1x_1 + w_2x_2$$

Linear operation



`Input times weights and activate`

Sigmoid/logistic function

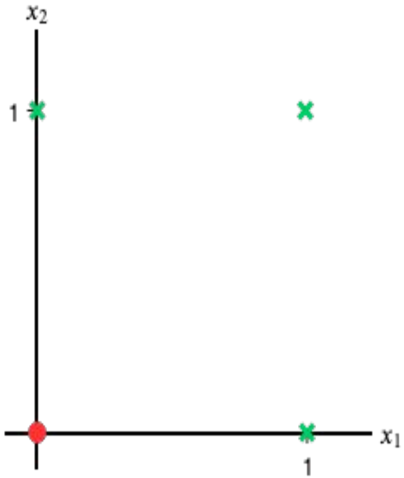


$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

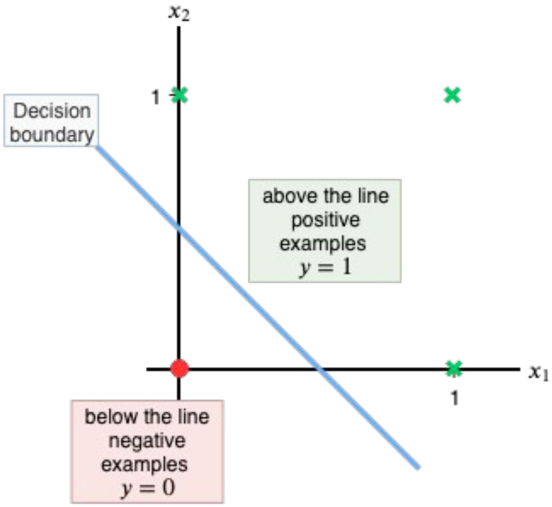
Other Popular Activation Functions:

- *Tanh — Hyperbolic tangent*
- *ReLu -Rectified linear units*

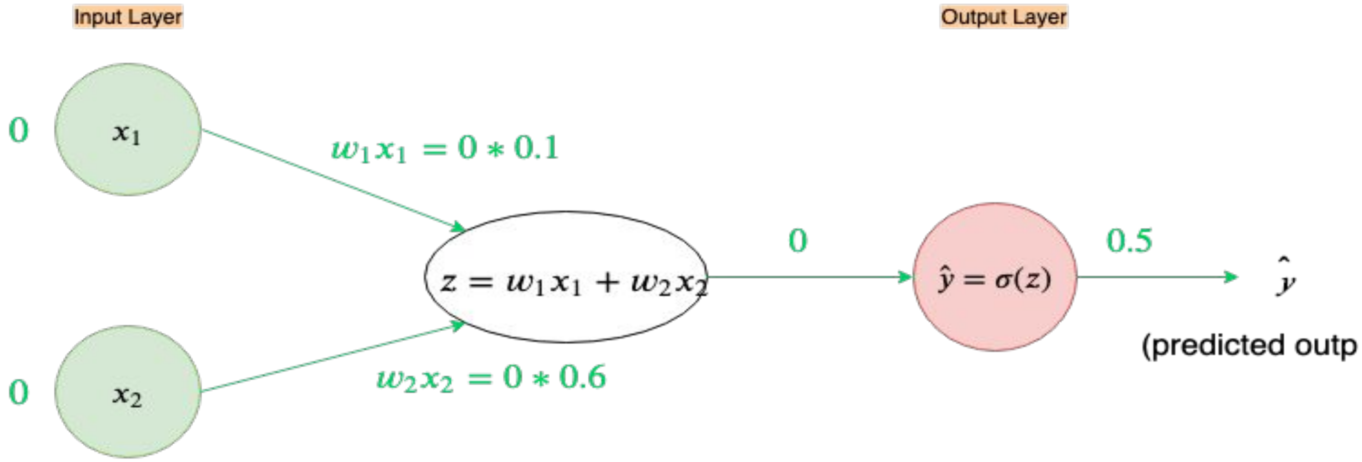
OR gate



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



Forward propagation of first input



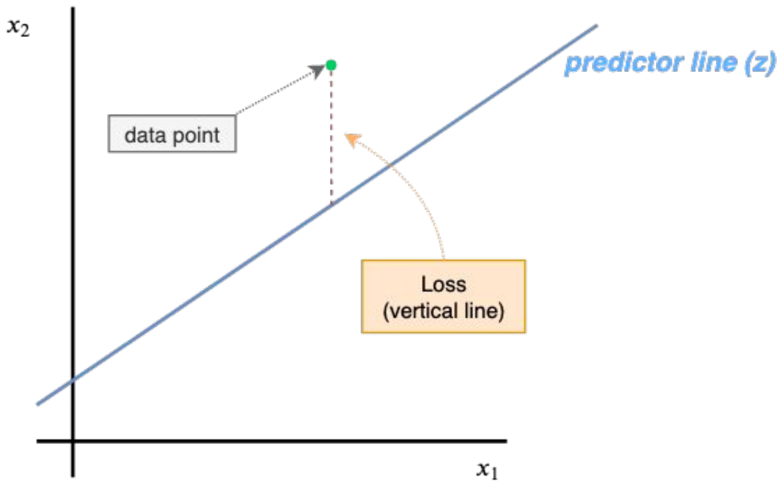
x_1=0
x2=0

Randomly selected
w1 =0.1
w2=0.6

The neural network is going through the following computations(forward computations marked in green):

- Our input for first example $x_1 = 0, x_2 = 0$
- Randomly initialized weights $w_1 = 0.1, w_2 = 0.6$
- $z = w_1 x_1 + w_2 x_2 = 0 * 0.1 + 0 * 0.6 = 0$
- $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{0}} = 0.5$

Loss Function



$$\mathbf{Loss = L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2}$$

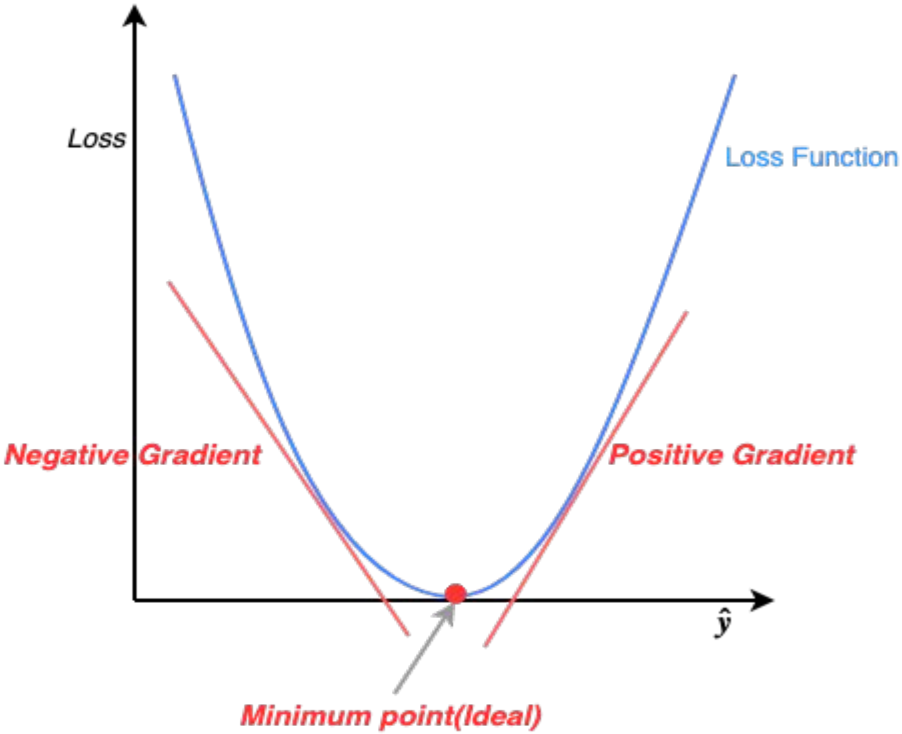
Where $\mathbf{y}$ is the actual desired output, and

$\hat{\mathbf{y}}$ is the predicted output

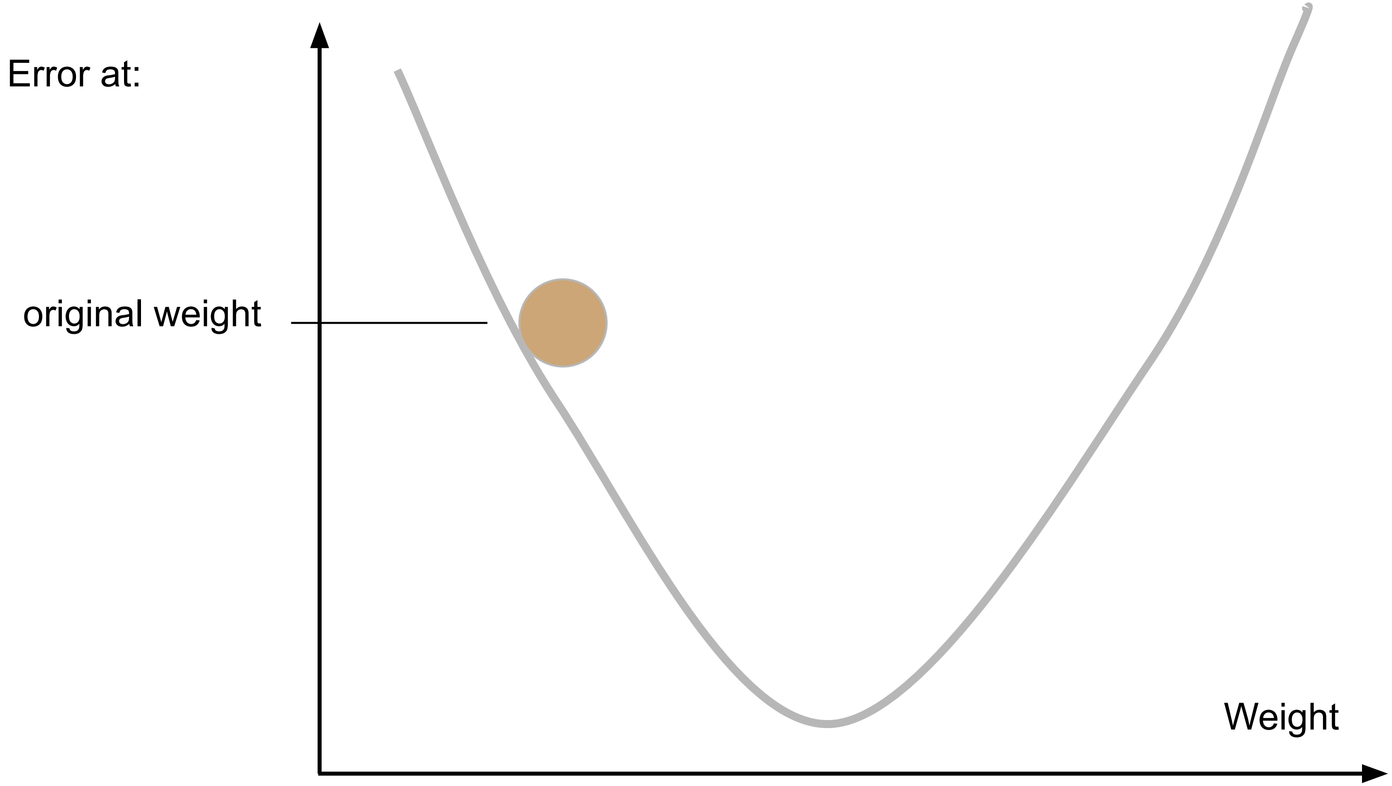
$$Loss = L(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(0 - 0.5)^2 = \frac{1}{2}(-0.5)^2 = \frac{1}{8} = 0.125$$

- For our current example $\hat{y} = 0.5$ and $y = 0$

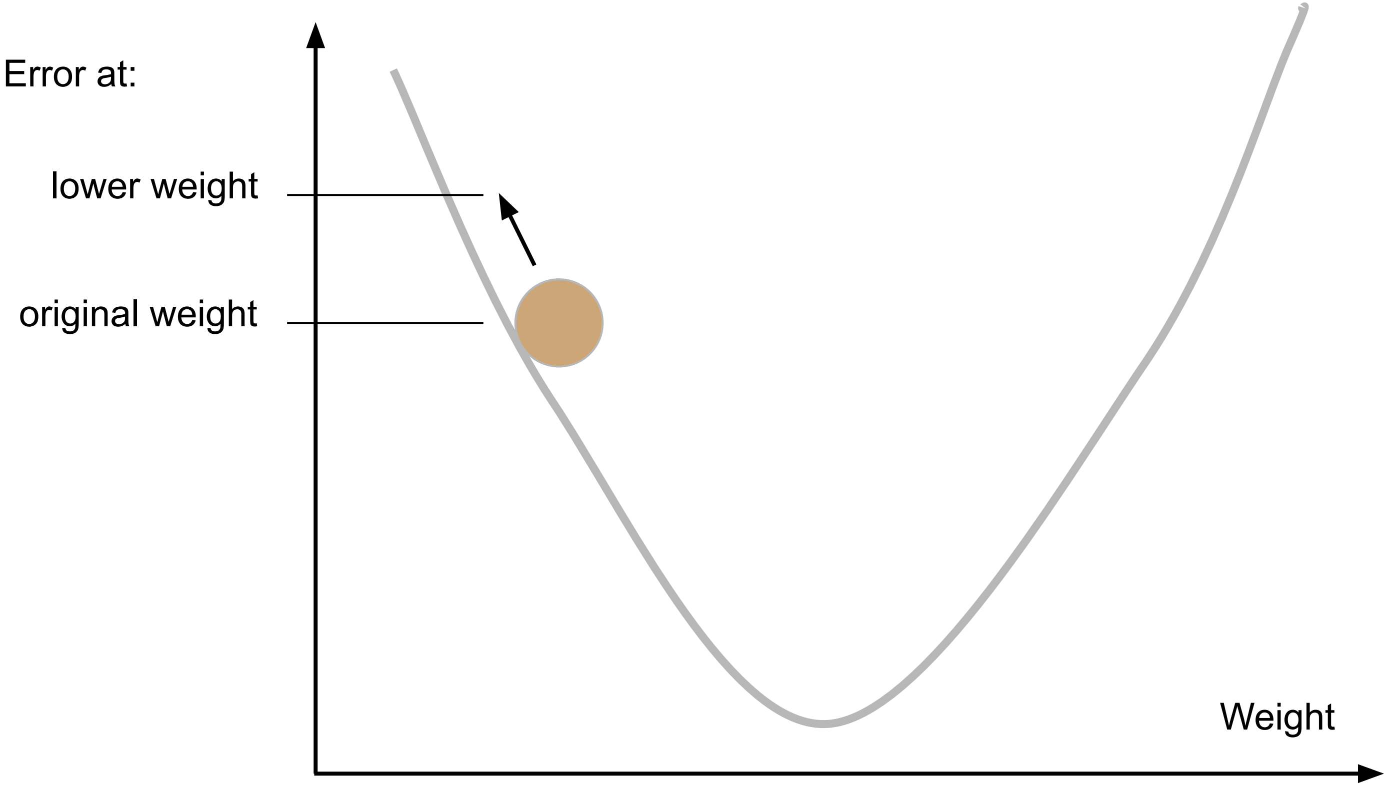
Loss function visualized



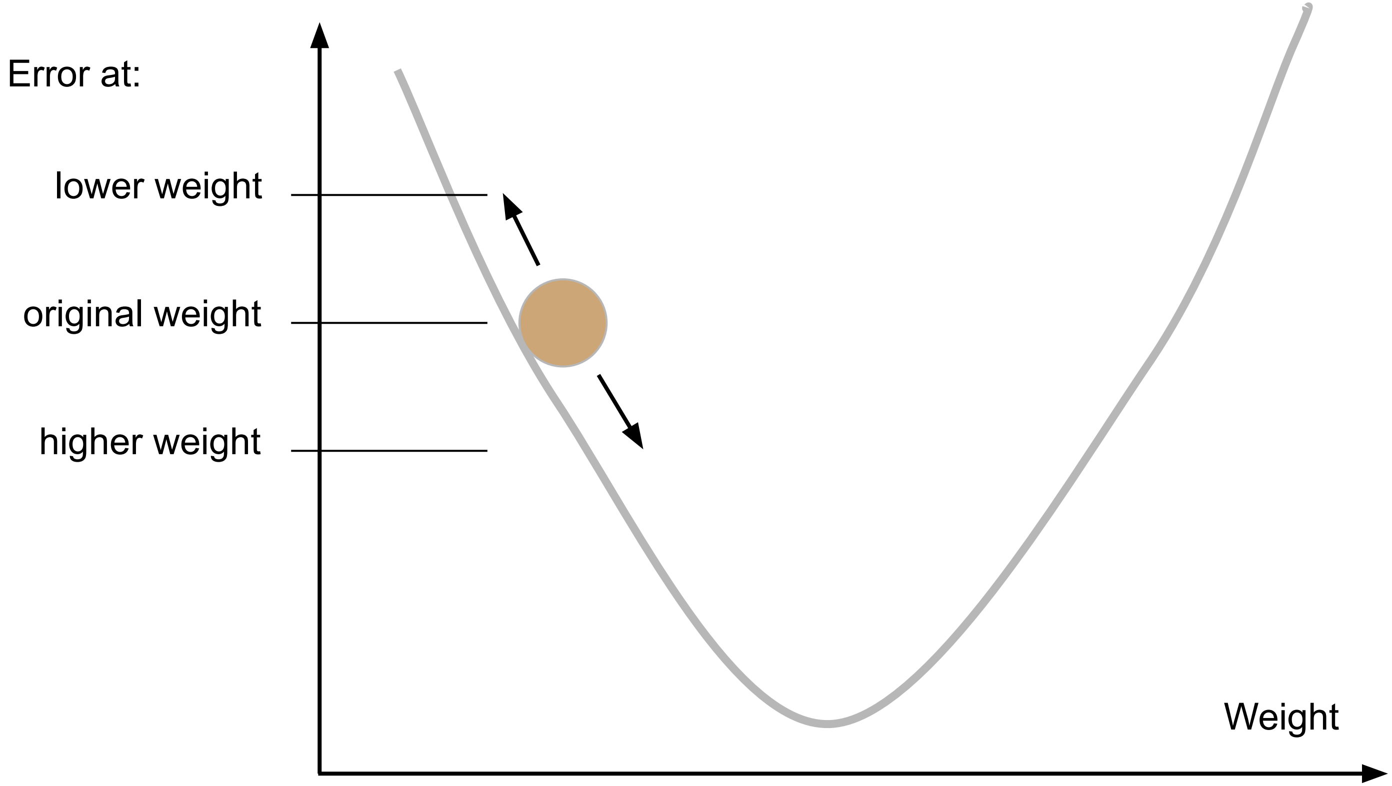
Learn all the weights: Gradient descent



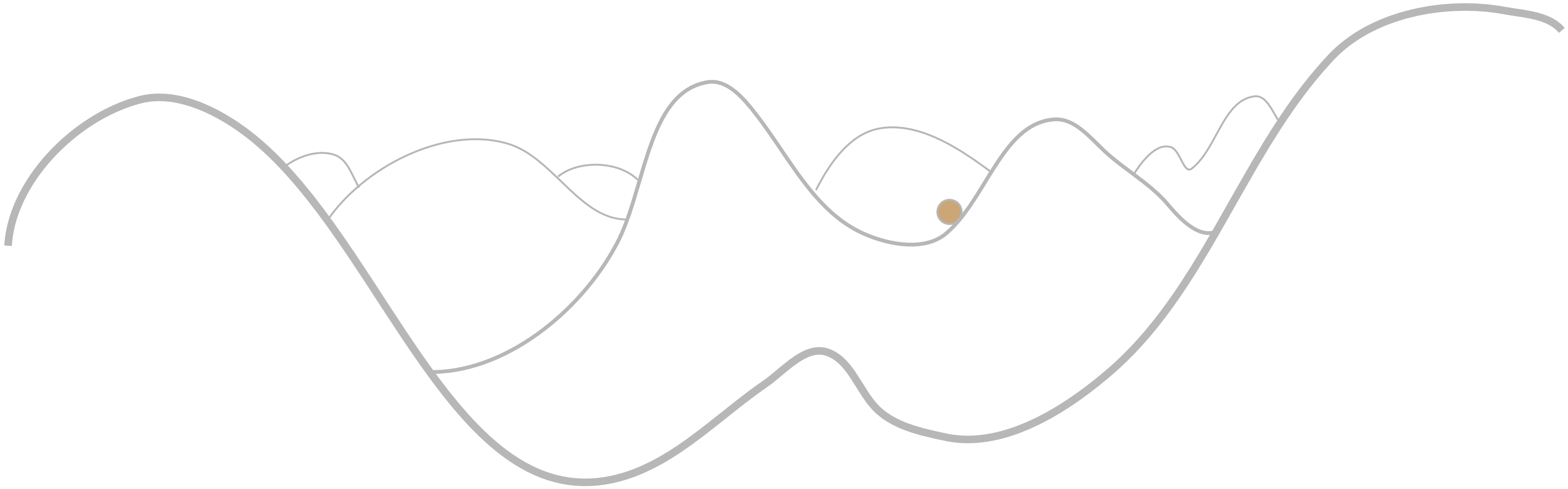
Learn all the weights: Gradient descent



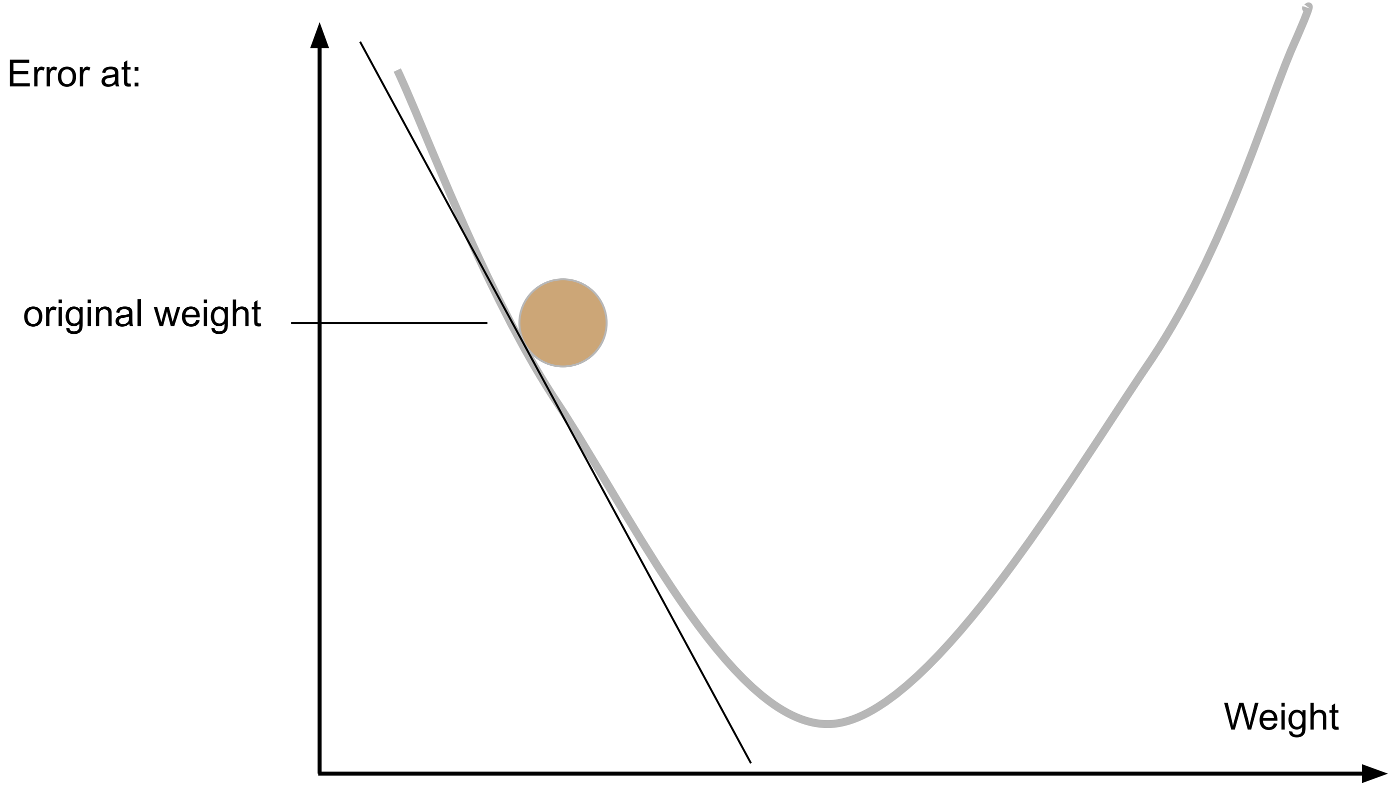
Learn all the weights: Gradient descent



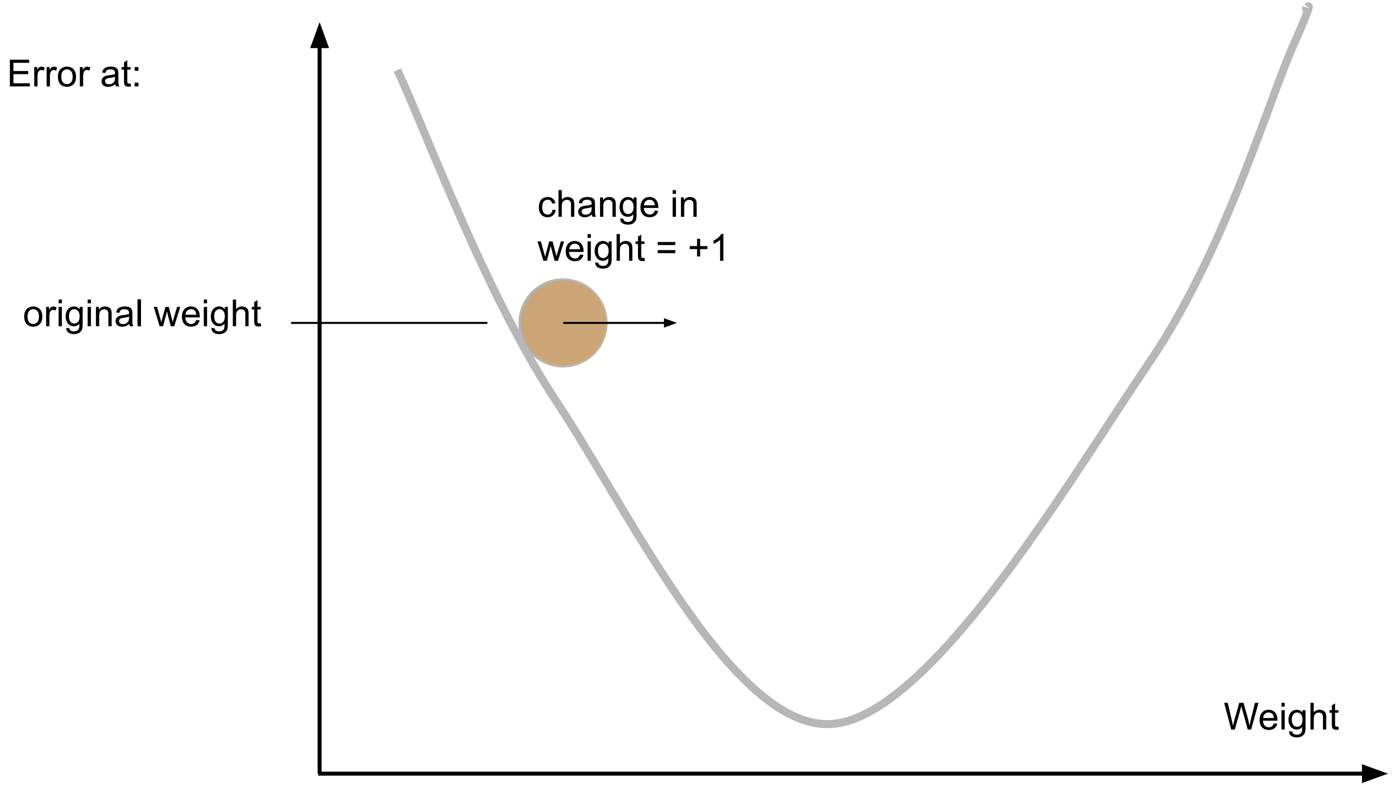
Numerically calculating the gradient is very expensive



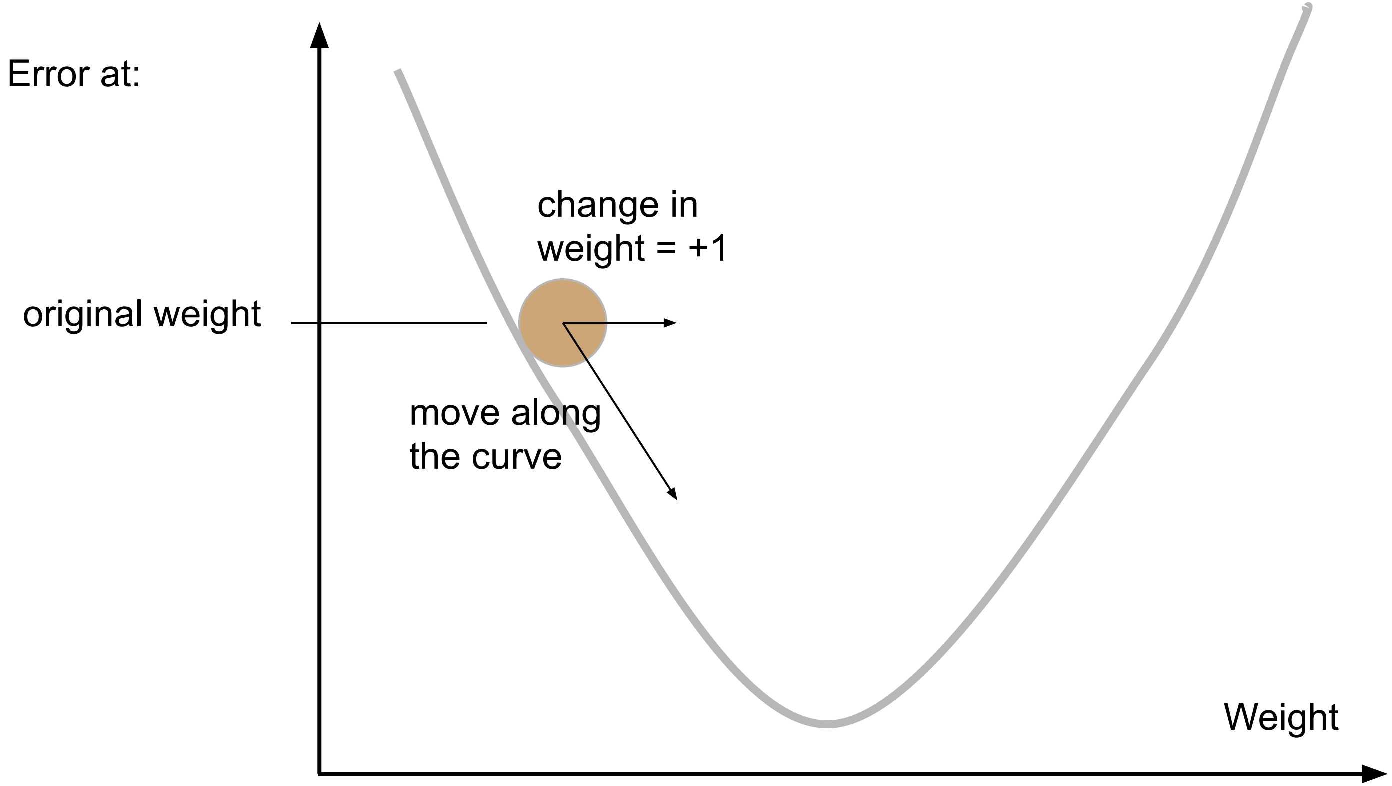
Calculate the gradient (slope) directly



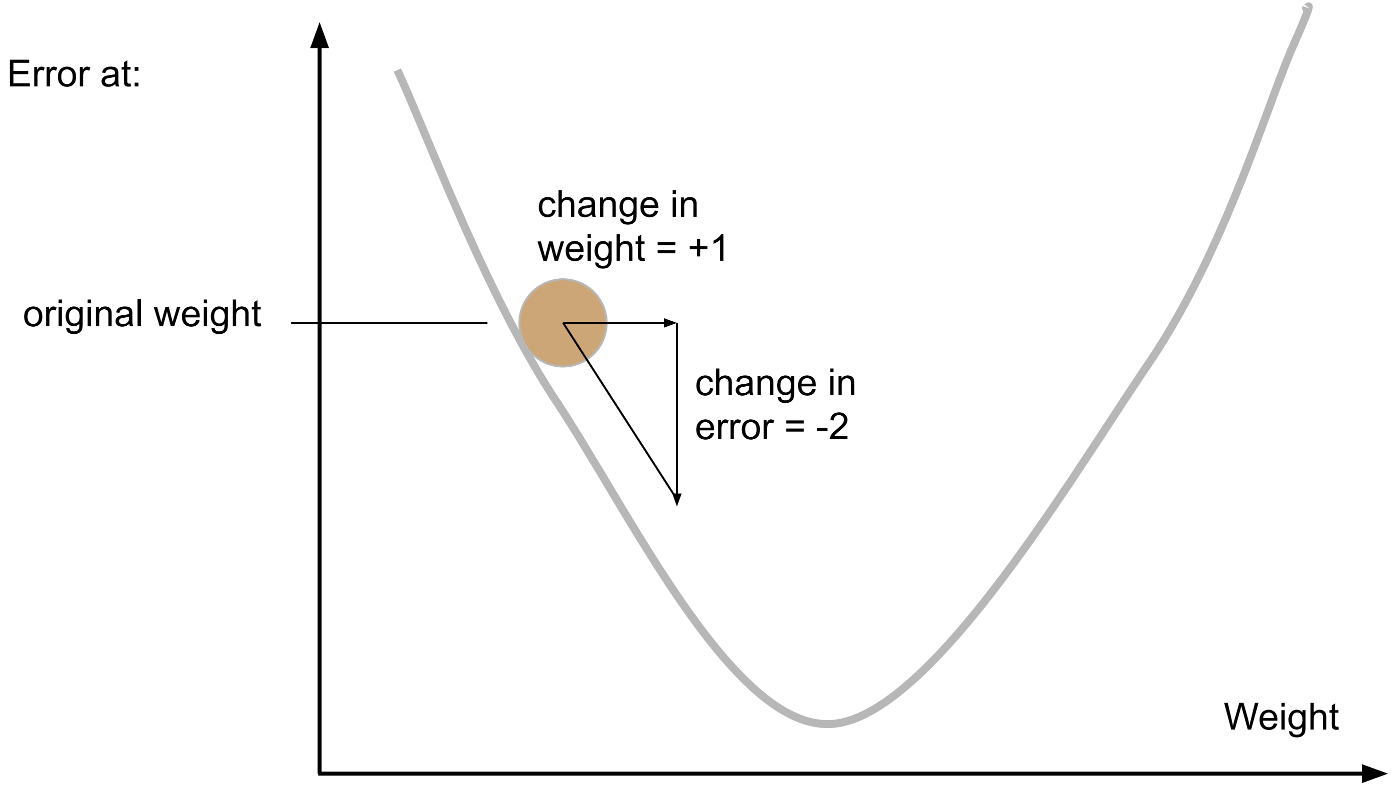
Slope



Slope


Error at:
change in
weight = +1
original weight
move along
the curve
Weight

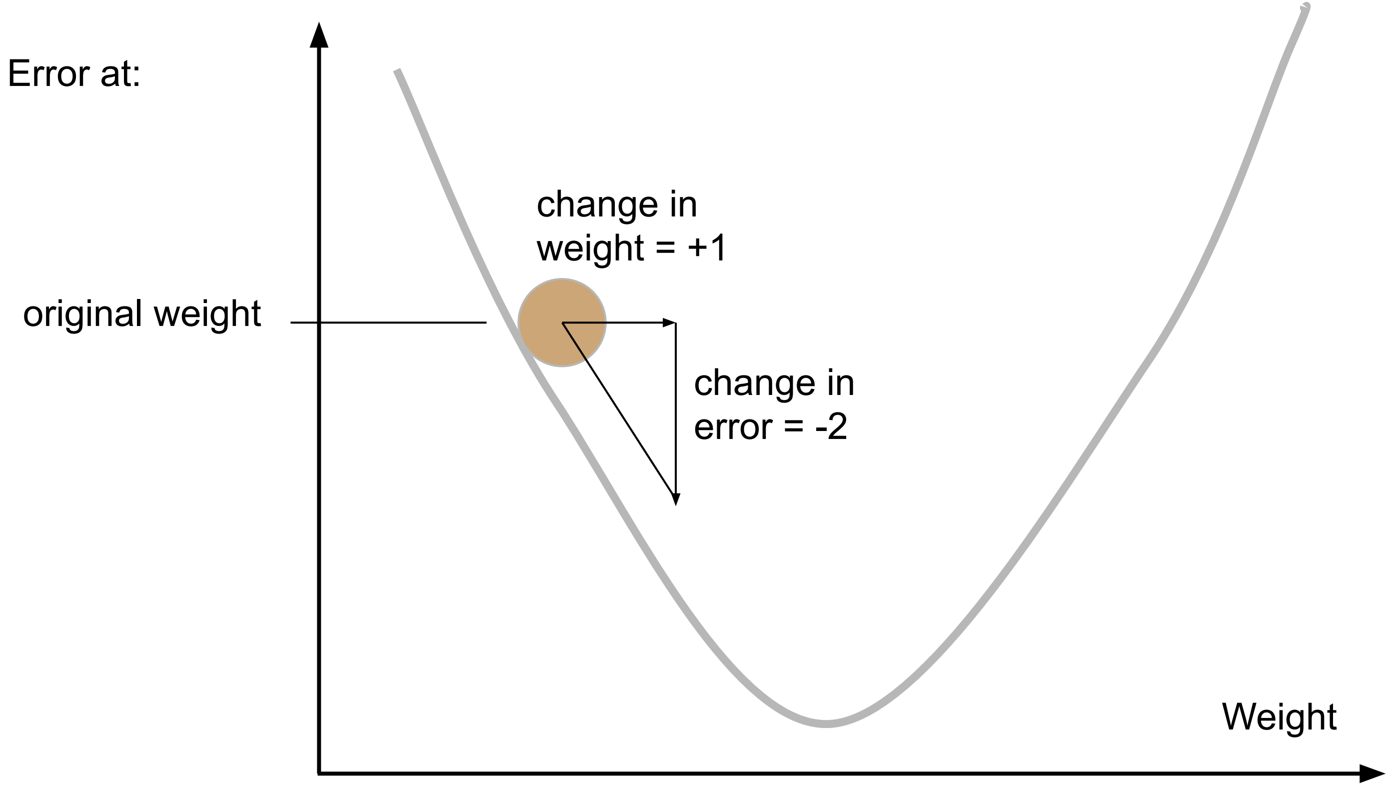
Slope



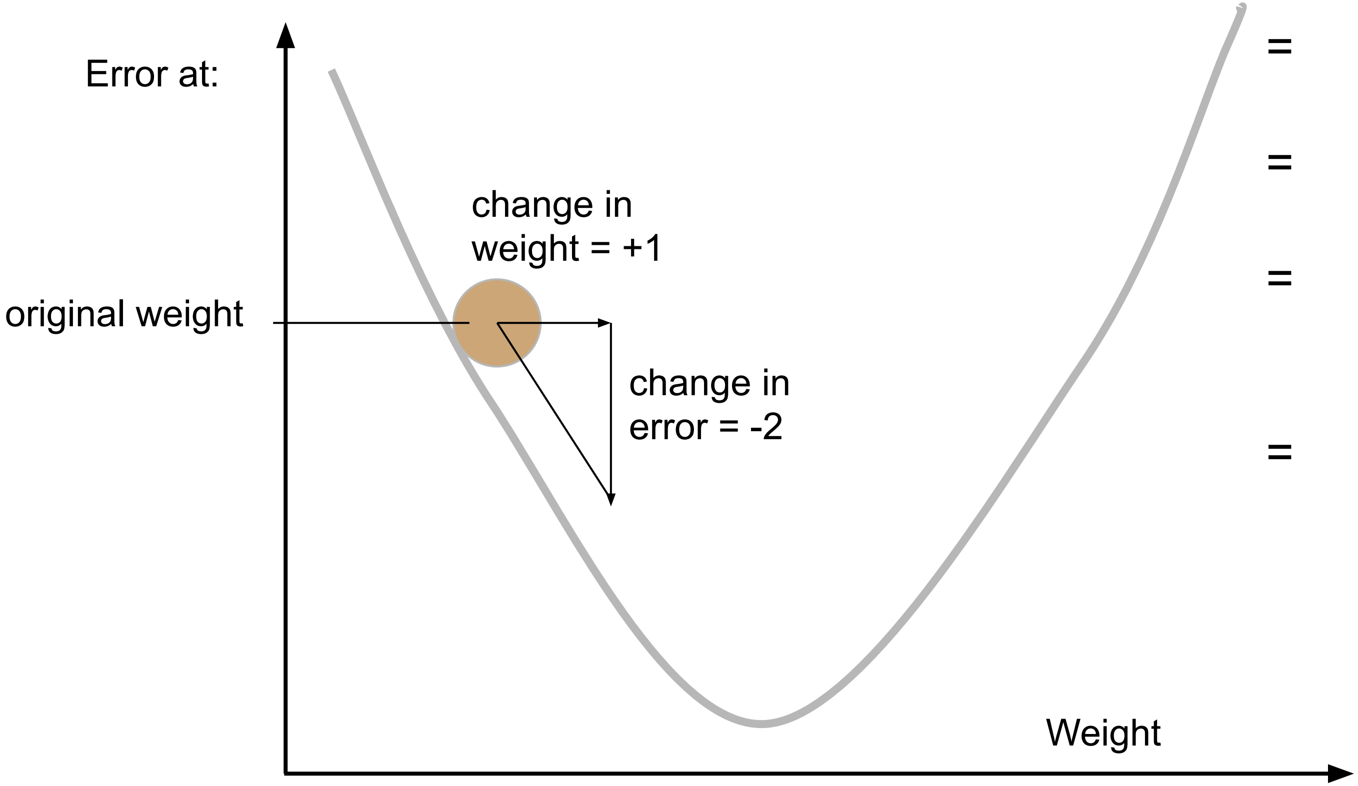
Error at:
change in
weight = +1
original weight
change in
error = -2
Weight

Slope

$$\text{slope} = \frac{\text{change in error}}{\text{change in weight}}$$



Slope



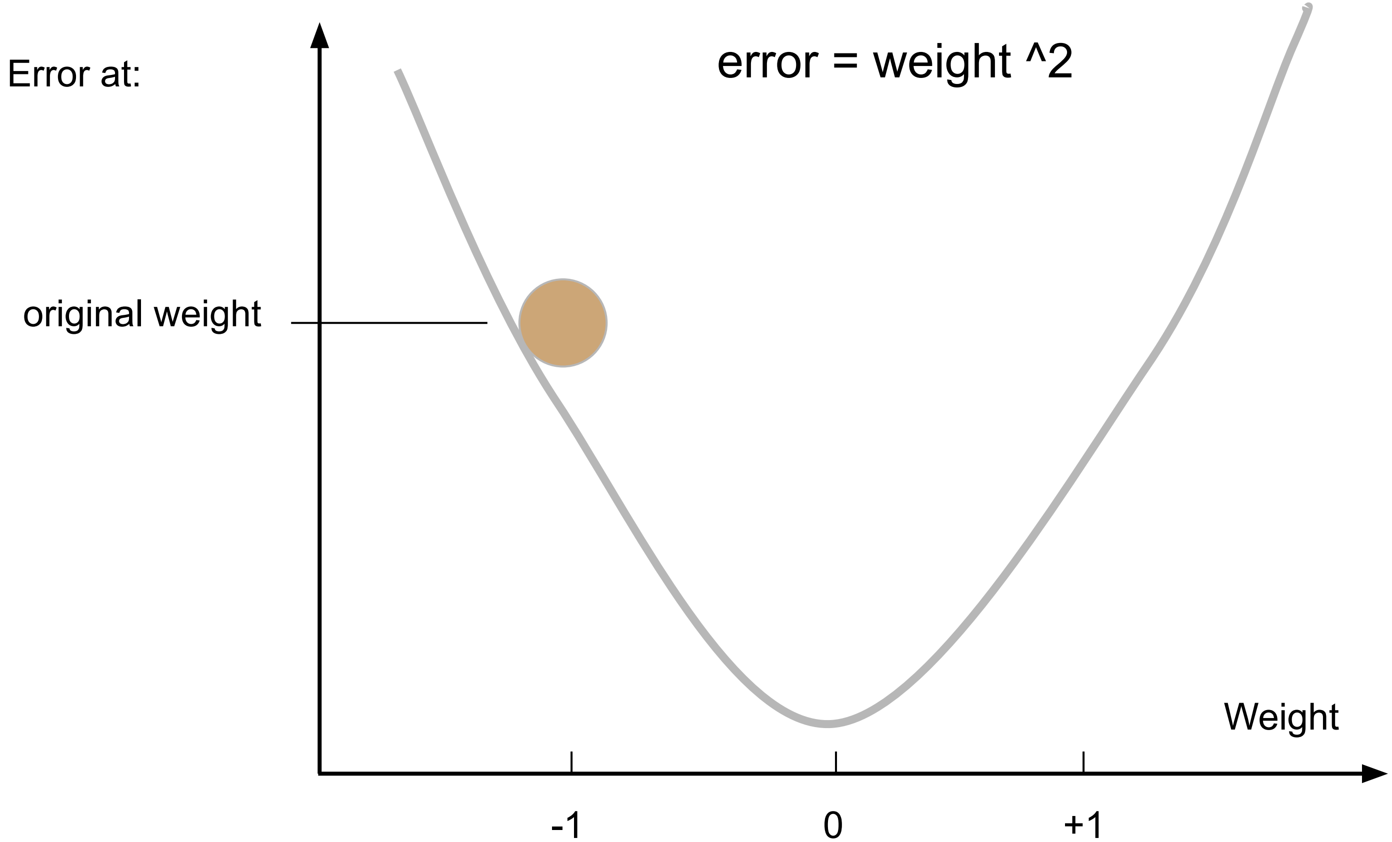
$$\text{slope} = \frac{\text{change in error}}{\text{change in weight}} = \frac{\Delta\ \text{error}}{\Delta\ \text{weight}} = \frac{d(\text{error})}{d(\text{weight})} = \frac{\partial e}{\partial w}$$

$$= \frac{-2}{+1} = -2$$

Slope

You have to know your error function.
For example:

error = weight ^2



Slope

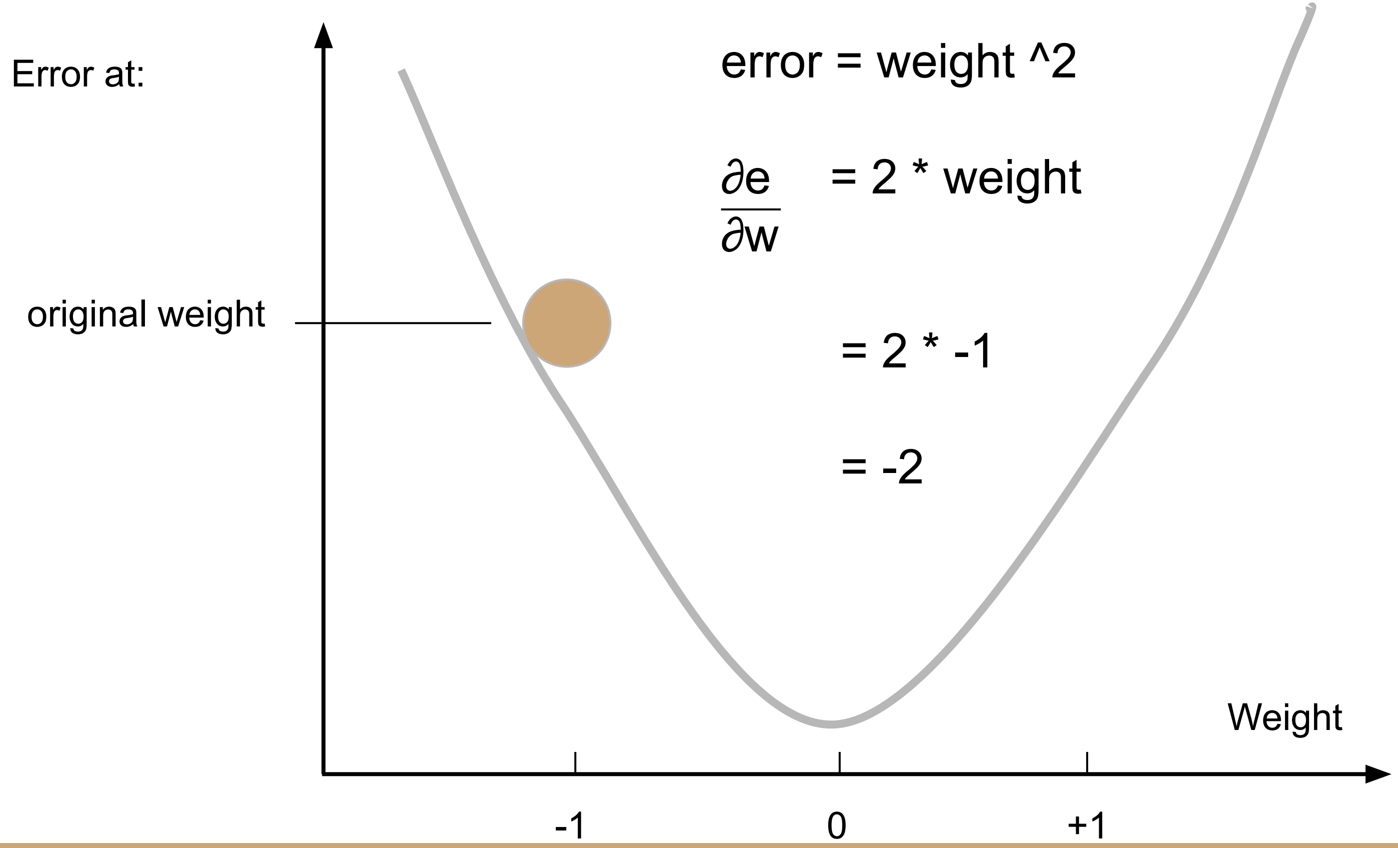
You have to know your error function.
For example:

error = weight ^2

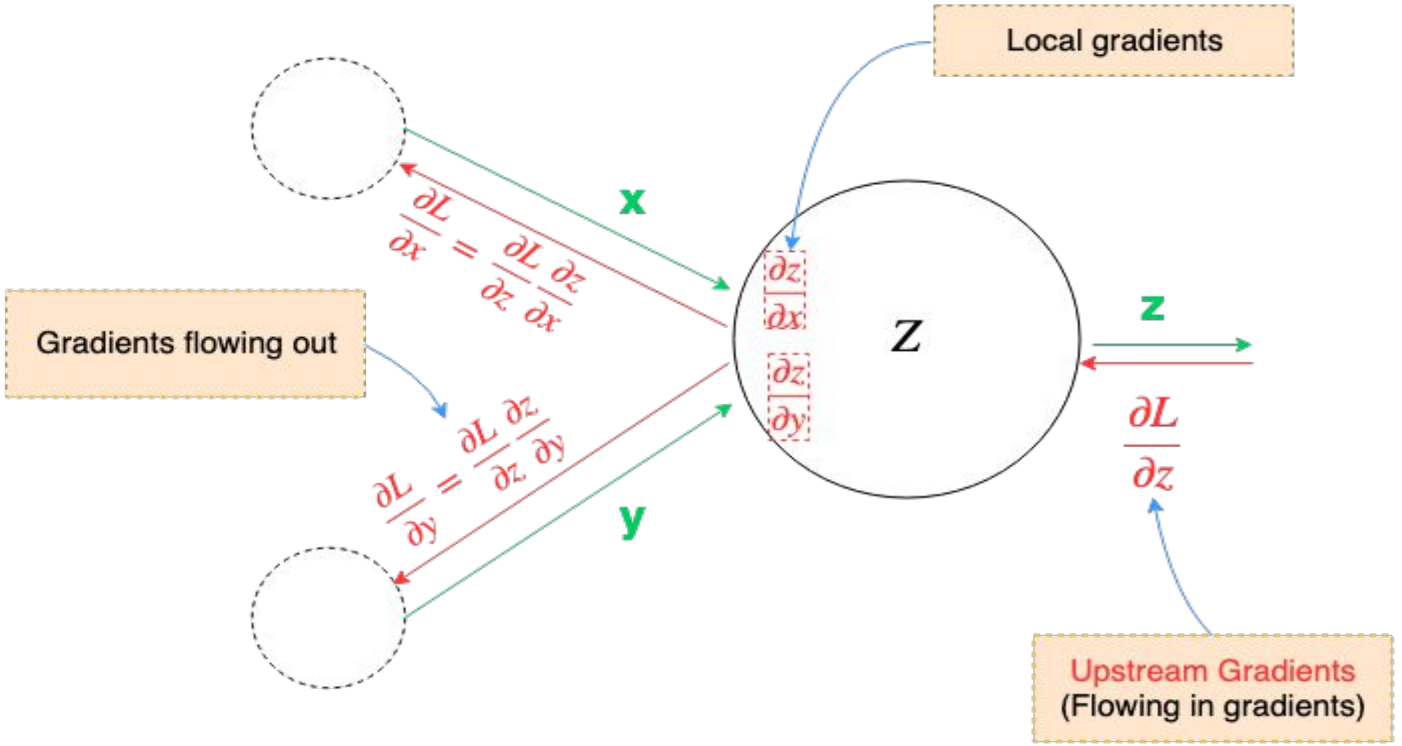
$$\frac{\partial e}{\partial w} = 2 * \text{weight}$$

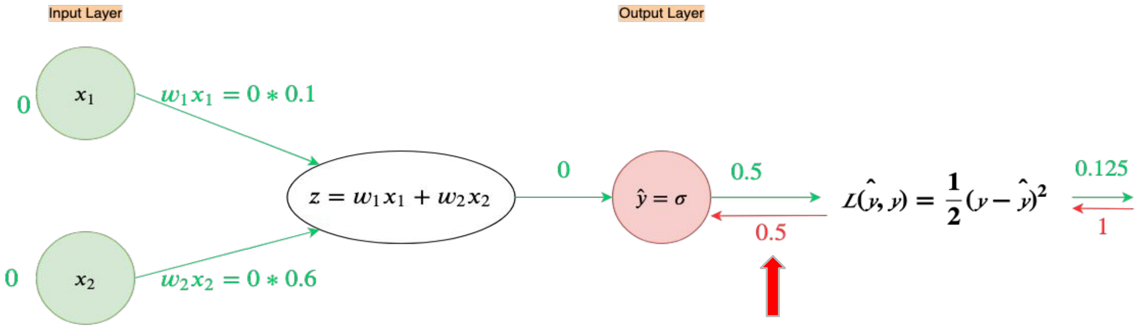
$$= 2 * -1$$

$$= -2$$



Gradient Flow





The neural network is going through the following computations(backward computations are marked in red):

- The first backward computation, fot the most part, is redundant, but for the sake of completeness we'll define it.
- $\frac{\partial L}{\partial L} = 1$ *- this forms the first Upstream gradient*
- The *Local gradient* at $L(\hat{y}, y)) = \frac{1}{2}(y - \hat{y})^2$ is:

$$\frac{\partial L}{\partial \hat{y}} = -(y - \hat{y})$$

Recall for current example $\hat{y} = 0.5$ and $y = 0$. So, numercal value of local gradient is:

$$\frac{\partial L}{\partial \hat{y}} = -(0 - 0.5) = 0.5$$

- Finally, we can combine these and send back to the red node:

$$\frac{\partial L}{\partial \hat{y}} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial L} * \frac{\partial L}{\partial \hat{y}} = 1 * 0.5 = 0.5$$

Following is the sigmoid function:

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

Let $u = 1 + e^{-z}$

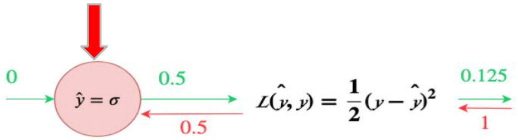
So, the equation becomes:

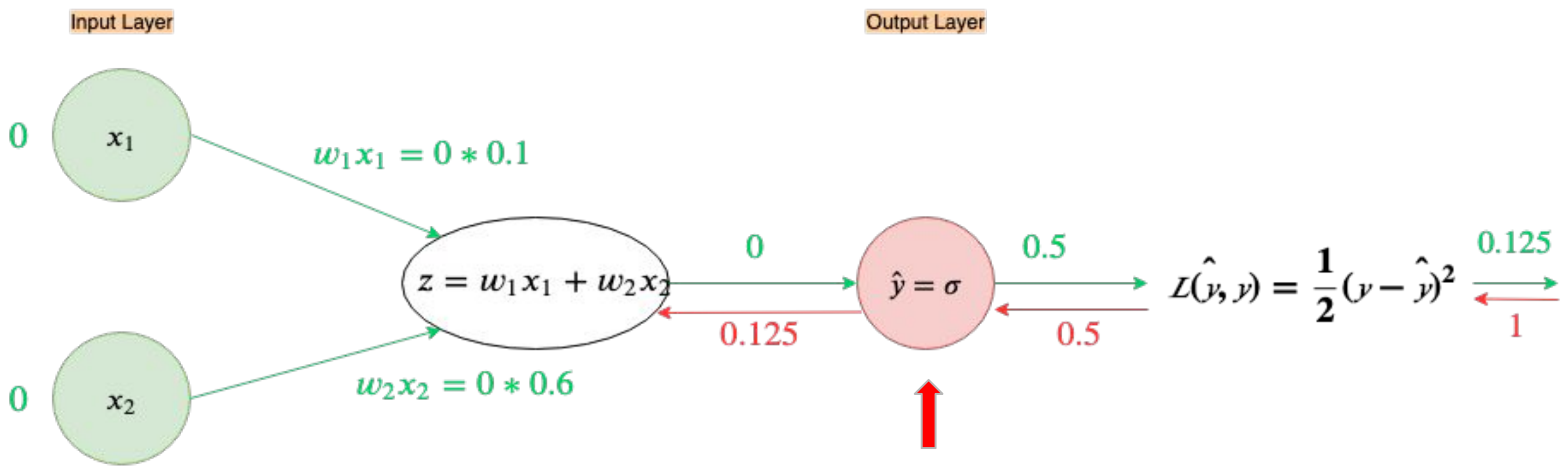
$$\hat{y} = \frac{1}{u}$$

This will be used in next backward calculation

Now, we can use the chain rule to easily derive the derivative:

$$\begin{aligned}
\frac{d\hat{y}}{dz} &= \frac{d\hat{y}}{du} * \frac{du}{dz} \\
&= \left(-\frac{1}{u^2}\right) * (-e^{-z}) \\
&\text{substitute } u = 1 + e^{-z} \\
&= \left(-\frac{1}{(1+e^{-z})^2}\right) * (-e^{-z}) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}} * \frac{e^{-z}}{(1+e^{-z})} \\
&= \frac{1}{1+e^{-z}} * \frac{1+e^{-z}-1}{(1+e^{-z})}, \text{ 1 added and subtracted, overall numerator remains same} \\
&= \frac{1}{1+e^{-z}} * \left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right) \\
&= \frac{1}{1+e^{-z}} * \left(1 - \frac{1}{1+e^{-z}}\right) \\
&\text{substitute } \frac{1}{1+e^{-z}} = \hat{y} \\
&= \hat{y} * (\mathbf{1} - \hat{y})
\end{aligned}$$

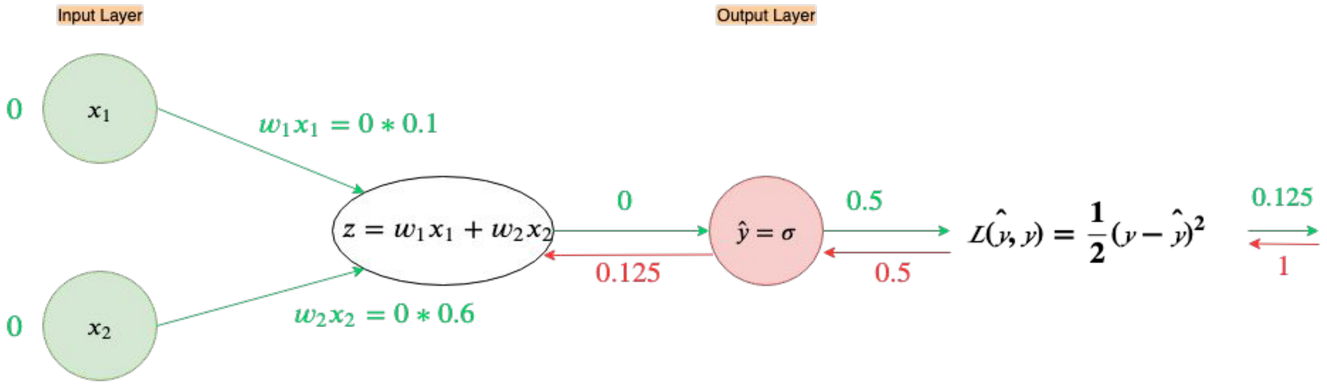




The neural network is going through the following computations(backward computations are marked in red):

- The *Upstream gradient* in this step is $\frac{\partial L}{\partial \hat{y}} = 0.5$
- The *Local gradient* at the red node is: $\frac{\partial \hat{y}}{\partial z} = \hat{y} * (1 - \hat{y}) = 0.5 * (1 - .05) = \frac{1}{4} = 0.25$
- Like previously, we will combine these and send them backwards to the white node:

$$\frac{\partial L}{\partial z} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial \hat{z}} = 0.5 * 0.25 = \frac{1}{8} = 0.125$$



The neural network is going through the following computations(backward computations are marked in red):

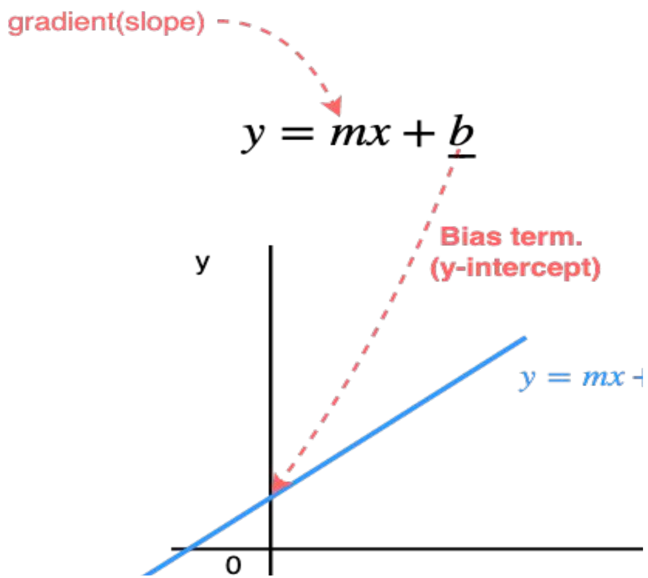
- Finally, we have now propagated the upstream gradient back enough to calculate the derivatives of weights w_1 and w_2.
- The *Upstream gradient* in this step is $\frac{\partial L}{\partial z} = 0.125$
- The *two Local gradients* are:

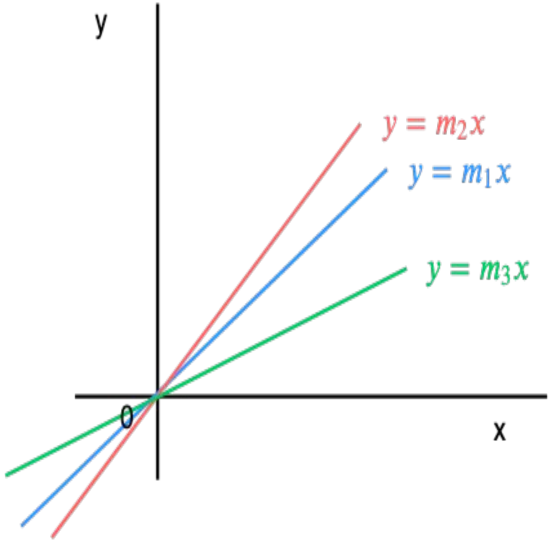
    1. $\frac{\partial z}{\partial w_1} = \frac{\partial(w_1x_1+w_2x_2)}{\partial w_1} = x_1 = 0$
    2. $\frac{\partial z}{\partial w_2} = \frac{\partial(w_1x_1+w_2x_2)}{\partial w_2} = x_2 = 0$

- We will again combine these, but this time not send them back to our input nodes, instead just figure out how much to change the weights w_1 and w_2:

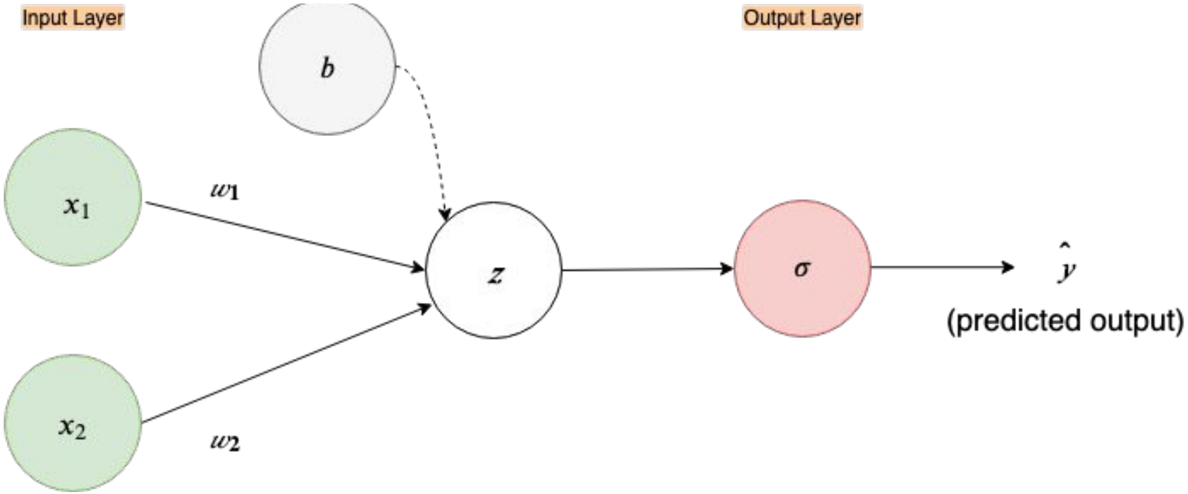
    1. $\frac{\partial L}{\partial w_1} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial w_1} = 0.125 * 0 = 0$
    2. $\frac{\partial L}{\partial w_2} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial w_2} = 0.125 * 0 = 0$

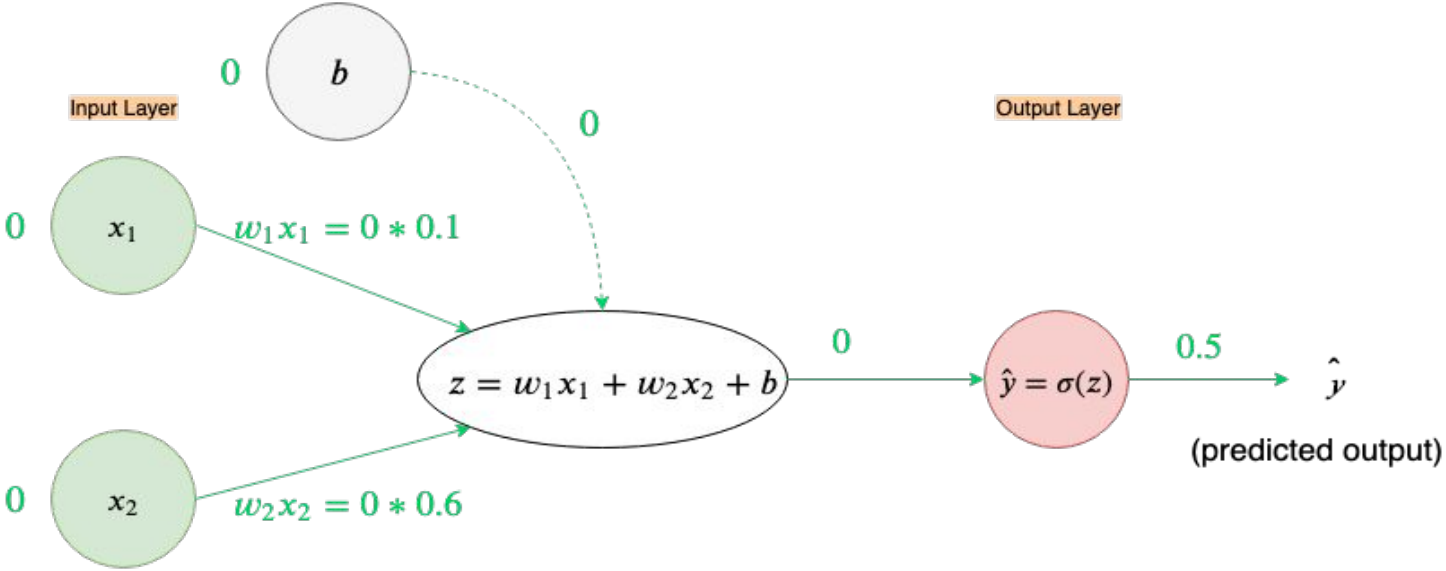
Bias


gradient(slope)
$y = mx + b$
Bias term.
(y-intercept)
y
$y = mx +$
0


y
$y = m_2x$
$y = m_1x$
$y = m_3x$
0
x

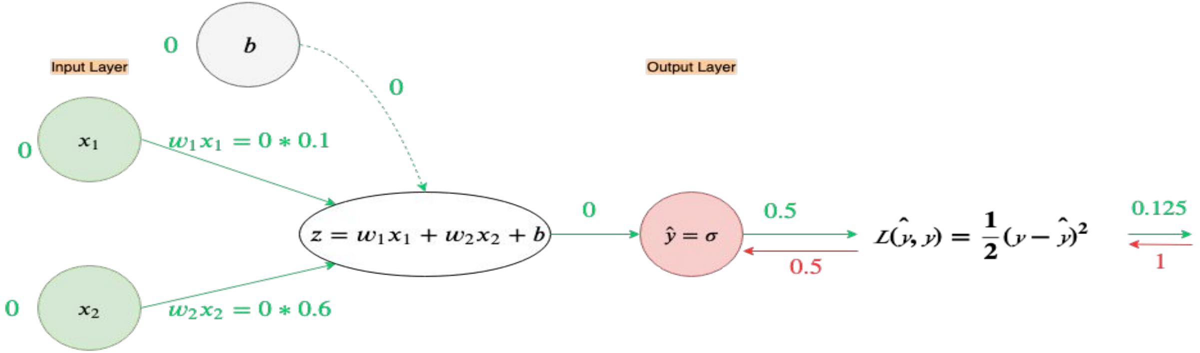
Expanded NN with bias node





The neural network is going through the following computations(forward computations marked in green):

- Our input for first example $x_1 = 0, x_2 = 0$
- Recall our randomly initialized weights $w_1 = 0.1, w_2 = 0.6.$
- We'll initialize our bias to be zero, $b = 0$
- $z = w_1 x_1 + w_2 x_2 + b = 0 * 0.1 + 0 * 0.6 + 0 = 0$
- $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{0}} = 0.5$



The neural network is going through the following computations(backward computations are marked in red):

- Again the first backward computation is redundant $\frac{\partial L}{\partial L} = 1$ *- this is still the first Upstream gradient*
- The *Local gradient* at $L(\hat{y}, y)) = \frac{1}{2}(y - \hat{y})^2$ still remains the same:

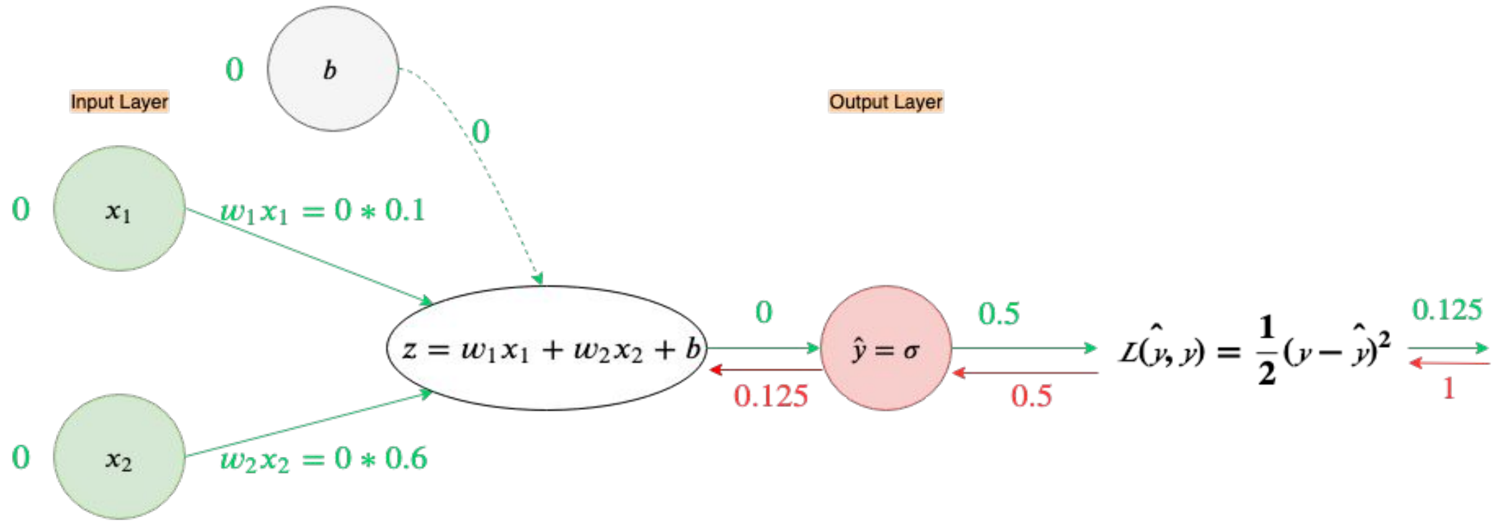
$$\frac{\partial L}{\partial \hat{y}} = -(y - \hat{y})$$

Recall, $\hat{y}$ for current example is $\hat{y} = 0.5$ and $y = 0$. So, numercal vale of local gradient also remains same:

$$\frac{\partial L}{\partial \hat{y}} = -(0 - 0.5) = 0.5$$

- As before, we'll combine these and send back to the red node:

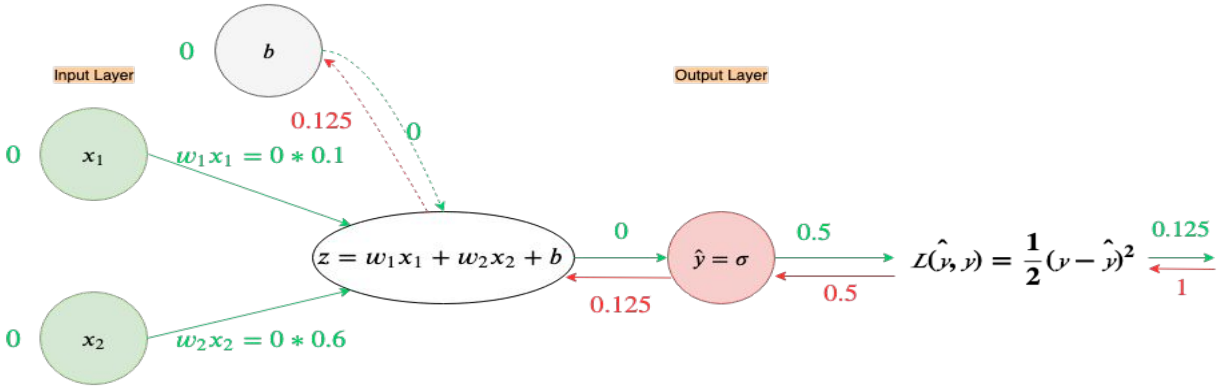
$$\frac{\partial L}{\partial \hat{y}} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial L} * \frac{\partial L}{\partial \hat{y}} = 1 * 0.5 = 0.5$$



The neural network is going through the following computations(backward computations are marked in red):

- The computations in this step remain the same as before.
- The *Upstream gradient* in this step is $\frac{\partial L}{\partial \hat{y}} = 0.5$
- The *Local gradient* at the red node is: $\frac{\partial \hat{y}}{\partial z} = \hat{y} * (1 - \hat{y}) = 0.5 * (1 - .05) = \frac{1}{4} = 0.25$
- Like previously, we will combine these and send them backwards to the white node:

$$\frac{\partial L}{\partial z} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial \hat{z}} = 0.5 * (0.5 - (1 - 0.5)) = \frac{1}{8} = 0.125$$



The neural network is going through the following computations(backward computations are marked in red):

- FInally, we have now propogated the upstream gradient back enough to calcalute w_1 ,w_2 and our bias b
- The *Upstream gradient* in this step is $\frac{\partial L}{\partial z} = 0.125$
- The *three Local gratients* are:

  1. $\frac{\partial z}{\partial w_1} = \frac{\partial(w_1x_1+w_2x_2+b)}{\partial w_1} = x_1 = 0$
  2. $\frac{\partial z}{\partial w_2} = \frac{\partial(w_1x_1+w_2x_2+b)}{\partial w_2} = x_2 = 0$
  3. $\frac{\partial z}{\partial b} = \frac{\partial(w_1x_1+w_2x_2+b)}{\partial b} = 1$

- We will again combine these, but this time not send them back to our input nodes, instead just figure out how much to change the weights w_1 , w_2 and b:

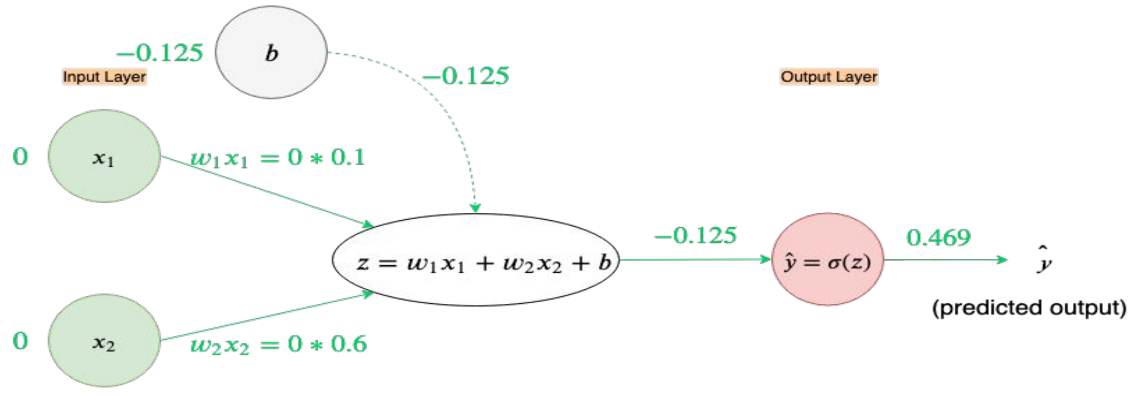
  1. $\frac{\partial L}{\partial w_1} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial w_1} = 0.125 * 0 = 0$
  2. $\frac{\partial L}{\partial w_2} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial w_2} = 0.125 * 0 = 0$
  3. $\frac{\partial L}{\partial b} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial b} = 0.125 * 1 = 0.125$

To calculate new bias we do the following:

Recall, current bias, $b = 0$ and $\frac{\partial L}{\partial b} = 0.125$

The new bias is:

$$b = b - \frac{\partial L}{\partial b} = 0 - 0.125 = \mathbf{-0.125}$$



The neural network is going through the following computations(forward computations marked in green):

- Our input for first example $x_1 = 0, x_2 = 0$
- Our weights remain the same $w_1 = 0.1, w_2 = 0.6$ but our new bias is $b = -0.125$
- $z = w_1 x_1 + w_2 x_2 + b = 0 * 0.1 + 0 * 0.6 + (-0.125) = -0.125$
- $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{0}} = 0.4687906... \approx \mathbf{0.469}$

Loss after newly calculated bias :

$$Loss = L(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(0 - 0.469)^2 = \frac{1}{2}(-0.469)^2 = \mathbf{0.10998005}$$

- For our current example $\hat{y} \approx \mathbf{0.469}$ and $y = \mathbf{0}$

Learning Rate

Equation for updating bias

$$b = b - \frac{\partial L}{\partial b}$$

Equation for updating bias showing step

$$b = b - 1\frac{\partial L}{\partial b}$$

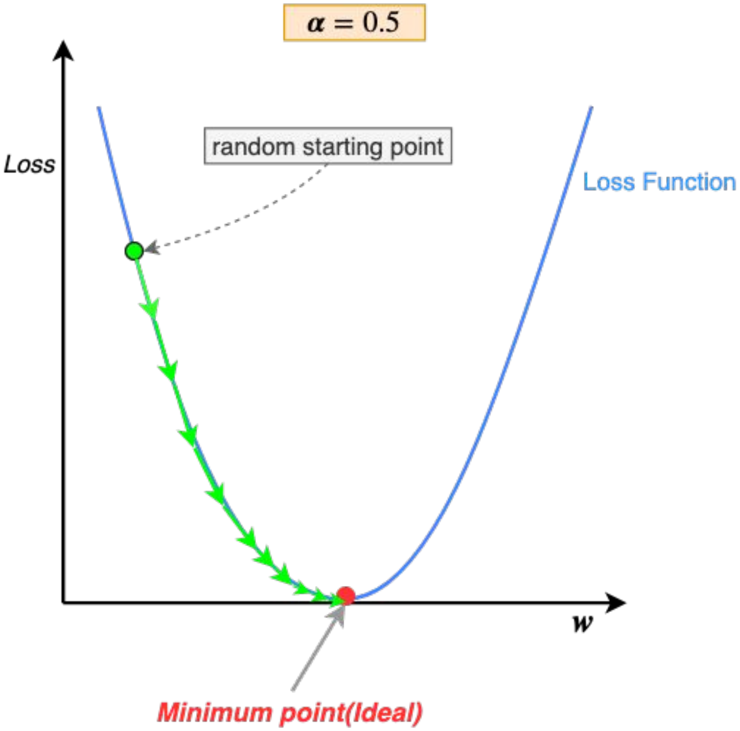
1 step

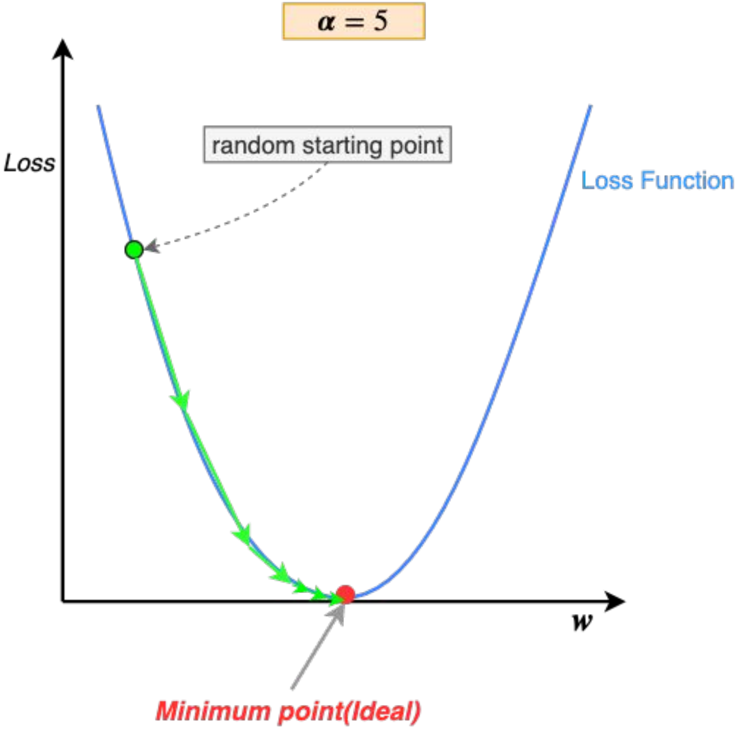
General Equation for Gradient Descent

$$w = w - \alpha\frac{\partial L}{\partial w}$$

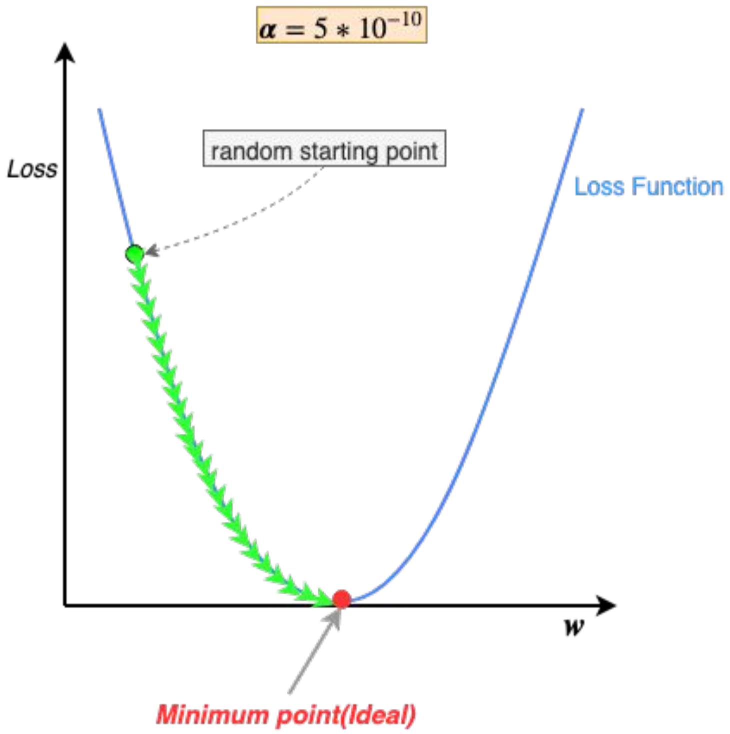
Learning Rate

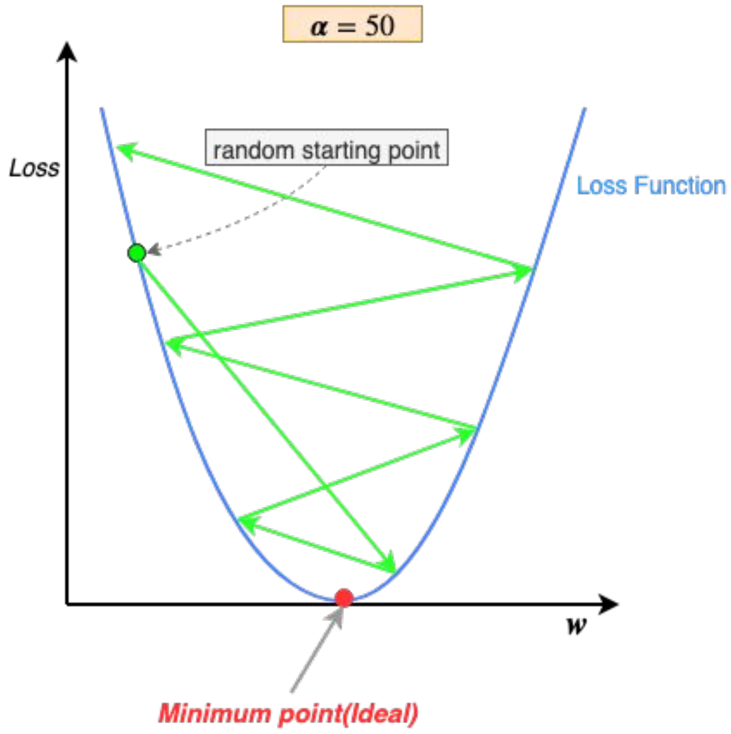
Effect of Learning Rate





Effect of very low vs. high learning rate





Gradient Descent

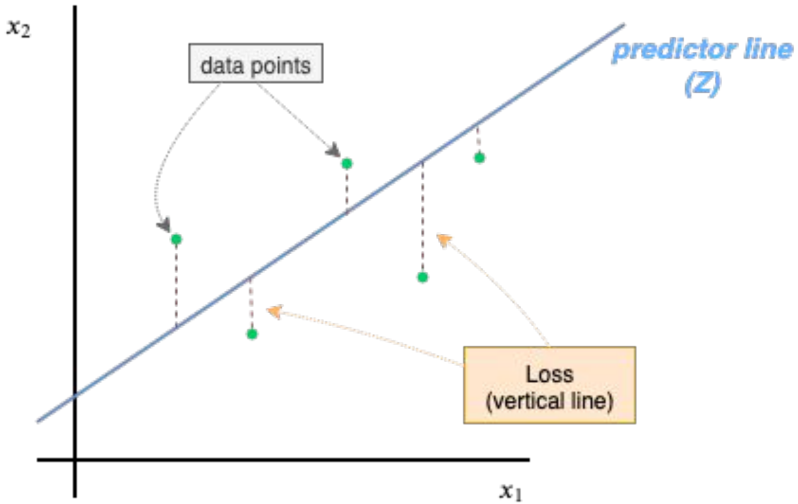
- **Batch Gradient Descent**
  - in one training iteration, it would reduce loss across all the training examples
- ***mini-batch gradient descent***
  - *use a subset of the data set in each iteration*
- ***stochastic gradient descent***
  - *only use one example per training iteration*

- ***Epoch***
  - *A training iteration where the neural network goes through all the training examples*

Cost Function

$$Cost = C(L(y^{(i)}, \hat{y}^{(i)}))$$
$$= \frac{1}{m}\sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)})$$
$$= \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}(y^{(i)} - \hat{y}^{(i)})^2$$
$$= \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$$

$i \rightarrow$ is the i^{th} training example
$m \rightarrow$ is the total number of training examples
$L(y^{(i)}, \hat{y}^{(i)}) \rightarrow$ is the loss in the i^{th} training example



Let's say, for example, our vectors $\vec{\mathbf{y}}$ and $\vec{\hat{\mathbf{y}}}$ are:

$$\vec{\mathbf{y}} = \begin{bmatrix} y^{(1)} & y^{(2)} \end{bmatrix} \text{ and } \vec{\hat{\mathbf{y}}} = \begin{bmatrix} \hat{y}^{(1)} & \hat{y}^{(2)} \end{bmatrix}$$

where $y^{(i)}$ or $\hat{y}^{(i)}$ is the i^{th} example in the vector. The number of examples, m, here is 2.

Now, let's calculate the ***Cost***.

$$\begin{aligned}
Cost(\vec{\mathbf{y}}, \vec{\hat{\mathbf{y}}}) &= \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \\
&= \frac{1}{2m} \sum (\vec{\mathbf{y}} - \vec{\hat{\mathbf{y}}}) \\
&= \frac{1}{2m} \sum \left(\begin{bmatrix} y^{(1)} & y^{(2)} \end{bmatrix} - \begin{bmatrix} \hat{y}^{(1)} & \hat{y}^{(2)} \end{bmatrix} \right)^{\circ 2} \\
&= \frac{1}{2m} \sum \left(\begin{bmatrix} (y^{(1)} - \hat{y}^{(1)}) & (y^{(2)} - \hat{y}^{(2)}) \end{bmatrix} \right)^{\circ 2} \\
&= \frac{1}{2m} \sum \left(\begin{bmatrix} (y^{(1)} - \hat{y}^{(1)})^2 & (y^{(2)} - \hat{y}^{(2)})^2 \end{bmatrix} \right) \\
&= \frac{\mathbf{1}}{\mathbf{2m}} [(\boldsymbol{y}^{(1)} - \hat{\boldsymbol{y}}^{(1)})^2 + (\boldsymbol{y}^{(2)} - \hat{\boldsymbol{y}}^{(2)})^2]
\end{aligned}$$

Element-wise square

Vector $\vec{\hat{\mathbf{y}}}$ has two examples in it $\hat{y}^{(1)}$ and $\hat{y}^{(2)}$.
To take $\frac{\partial Cost}{\partial \vec{\hat{\mathbf{y}}}}$, we'll have to take two partial derivatives; one with respect to each example.

The result is a vector of partial derivatives, called Jacobian. *(Jacobian is just a fancy name for a vector/ matrix full of derivatives):*

$$\frac{\partial Cost}{\partial \vec{\hat{\mathbf{y}}}} = \begin{bmatrix} \frac{\partial Cost}{\partial \hat{y}^{(1)}} & \frac{\partial Cost}{\partial \hat{y}^{(2)}} \end{bmatrix}$$

Let's do this in two parts so that we can understand how these simple derivatives are being computed:

$$\begin{aligned}\frac{\partial Cost}{\partial \hat{y}^{(1)}} &= \frac{\partial}{\partial \hat{y}^{(1)}}\left(\frac{1}{2m}[(y^{(1)} - \hat{y}^{(1)})^2 + (y^{(2)} - \hat{y}^{(2)})^2]\right) \\ &= \frac{1}{2m}\left[\frac{\partial}{\partial \hat{y}^{(1)}}\left((y^{(1)} - \hat{y}^{(1)})^2\right) + \frac{\partial}{\partial \hat{y}^{(1)}}\left((y^{(2)} - \hat{y}^{(2)})^2\right)\right] \\ &= \frac{1}{2m}\left[-2(y^{(1)} - \hat{y}^{(1)}) + 0\right] \\ &= -\frac{\mathbf{1}}{\mathbf{m}}(\mathbf{y}^{(1)} - \hat{\mathbf{y}}^{(1)})\end{aligned}$$

$$\begin{aligned}\frac{\partial Cost}{\partial \hat{y}^{(2)}} &= \frac{\partial}{\partial \hat{y}^{(2)}}\left(\frac{1}{2m}[(y^{(1)} - \hat{y}^{(1)})^2 + (y^{(2)} - \hat{y}^{(2)})^2]\right) \\ &= \frac{1}{2m}\left[\frac{\partial}{\partial \hat{y}^{(2)}}\left((y^{(1)} - \hat{y}^{(1)})^2\right) + \frac{\partial}{\partial \hat{y}^{(2)}}\left((y^{(2)} - \hat{y}^{(2)})^2\right)\right] \\ &= \frac{1}{2m}\left[0 + \left(-2(y^{(2)} - \hat{y}^{(2)})\right)\right] \\ &= -\frac{\mathbf{1}}{\mathbf{m}}(\mathbf{y}^{(2)} - \hat{\mathbf{y}}^{(2)})\end{aligned}$$

In the end, the Jacobian simply looks like this:

$$
\begin{aligned}
\frac{\partial Cost}{\partial \vec{\hat{\mathbf{y}}}} &= \begin{bmatrix} \frac{\partial Cost}{\partial \hat{y}^{(1)}} & \frac{\partial Cost}{\partial \hat{y}^{(2)}} \end{bmatrix} \\
&= \begin{bmatrix} -\frac{1}{m}\left(y^{(1)} - \hat{y}^{(1)}\right) & -\frac{1}{m}\left(y^{(2)} - \hat{y}^{(2)}\right) \end{bmatrix} \\
&= -\frac{\mathbf{1}}{m}\begin{bmatrix} \left(y^{(1)} - \hat{y}^{(1)}\right) & \left(y^{(2)} - \hat{y}^{(2)}\right) \end{bmatrix}
\end{aligned}
$$

Fig 36. Calculation of Jacobian on the simple example

From this, we can generalize the partial derivative equation.

Generalized derivative of the Cost(with squared error Loss) :

$$
\frac{\partial Cost}{\partial \hat{y}^{(i)}} = -\frac{\mathbf{1}}{m}\left(y^{(i)} - \hat{y}^{(i)}\right)
$$

Loss vs Cost

Partial derivative of ***Cost*** w.r.t $\hat{y}^{(i)}$	Partial derivative ***Loss*** w.r.t $\hat{y}^{(i)}$
$\dfrac{\partial Cost}{\partial \hat{y}^{(i)}} = -\dfrac{1}{m}\left(y^{(i)} - \hat{y}^{(i)}\right)$	$\dfrac{\partial L}{\partial \hat{y}} = -(y - \hat{y})$

$\frac{1}{m}$ is missing

Start the training loop for an arbitrary number of iterations, let's say 500

loop 500 times:

$\Delta w_1 = 0, \Delta w_2 = 0, \Delta b = 0$ →Define temporary gradient accumulator variables for our weights and bias with capital delta(Δ) as prefix

$\Delta C = 0$ → Temporary variable to accumulate all the losses, so that we can caluclate Cost at the end

Now loop over all, "m" training examples

foreach training example:

Perform Forward-propagation

Claculate Loss(L) on example

$\Delta C = \Delta C + L$ →accumulate loss of example in ΔC,

Perform Backpropagation

$\Delta w_1 = \Delta w_1 + \delta w_1$ → accumulate gradient of $w1(\delta w_1)$

$\Delta w_2 = \Delta w_2 + \delta w_2$ →accumulate gradient of $w2(\delta w_2)$

$\Delta b = \Delta b + \delta b$→accumulate gradient of $b(\delta b)$

Calculate the Cost(which is just the average loss across all examples)

$Cost = \frac{1}{m}\Delta C$

finally, perform gradeint descent for each parameter, recall α is the *learning rate* and m is the *total number of training examples*

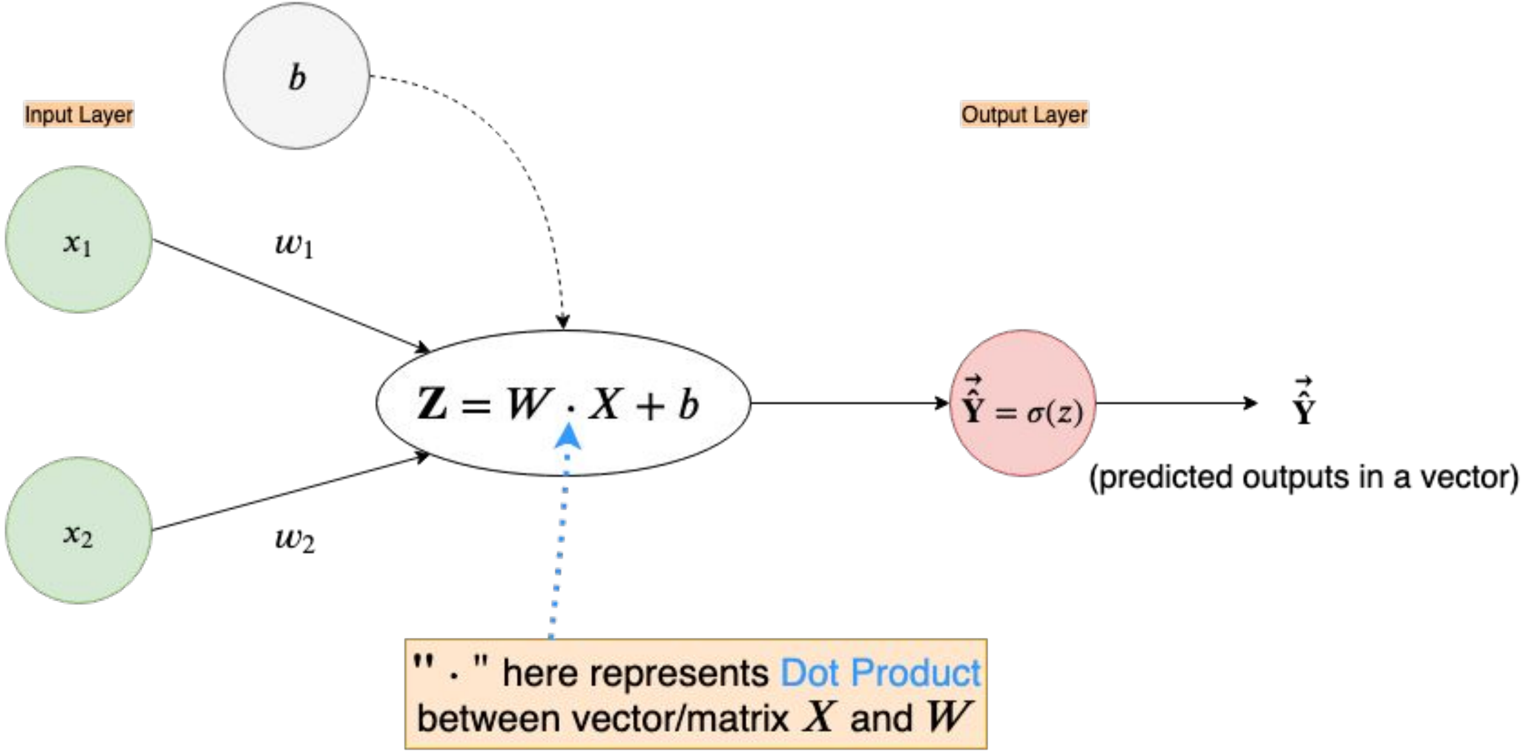
$$w_1 = w_1 - \frac{\alpha}{m}\Delta w_1$$

$$w_2 = w_2 - \frac{\alpha}{m}\Delta w_2$$

$$b = b - \frac{\alpha}{m}\Delta b$$

NOTE: diving by m gives us the average of the accumulated gradients in each case.

Vectorized Implementation



Data Setup

$\mathbf{W} = \begin{bmatrix} w_1 & w_2 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.6 \end{bmatrix}$, this makes $\mathbf{W}$ a (1×2) matrix

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ x_1^{(4)} & x_2^{(4)} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix},$$

here each row represents an example with x_1 and x_2 as its features. **X** is a (4×2) matrix.

Data set up in this way where each row of the matrix represents an individual example is called a Design Matrix

Similarly, $Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ y^{(4)} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix},$

each row represents the desired output for the respective example. **Y** is a (4×1) matrix

For dot-product we need to make sure that the matrices/vectors **W** and **X** are in the correct orientation/shape:

Matrix_A . Matrix_B = ResultMatrix_C

$$(a \times b) \;.\; (b \times c) \;=\; (a \times c)$$

Need to match

Right now, **W** is (1×2) and **X** is (4×2). So, we would need to fix the shape of our ***data*** (**X** and **Y**) to align with our computation of the dot-product.

So, we'll take the "transpose" of **X** and **Y**, which is simply flipping the matrix around its diagonal, so that rows become columns of the transposed matrix/vector.

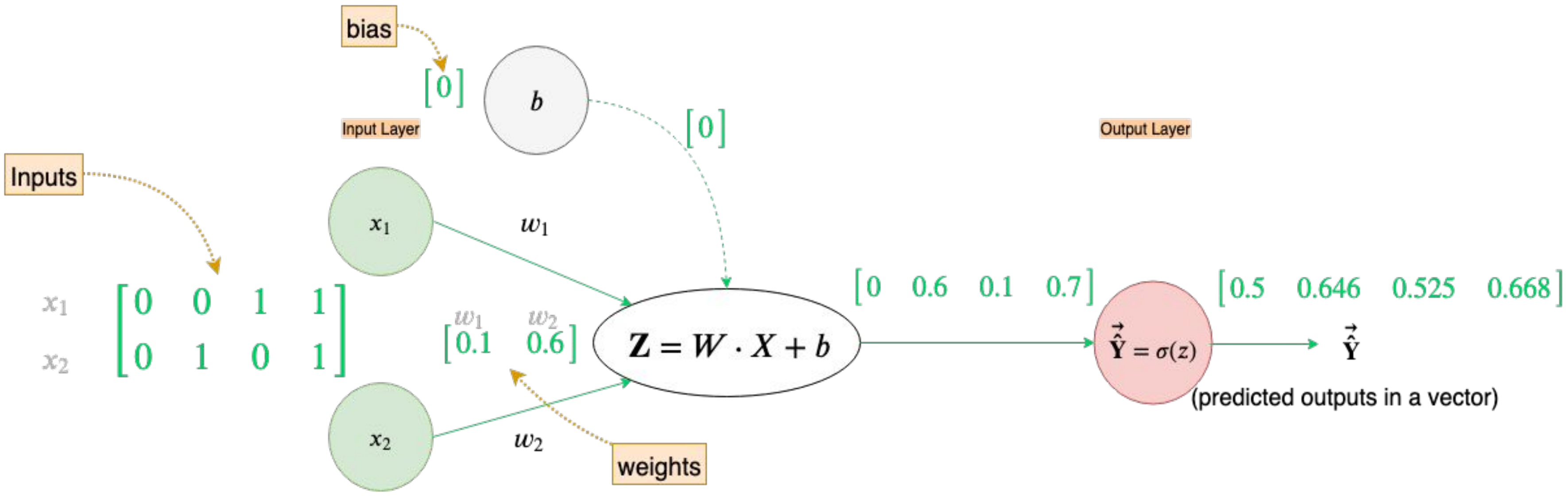
$$X_{train} = X^T = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix},$$

So, the data we'll train with, X_{train}, now has the shape (2×4)

Similarly, $Y_{train} = Y^T = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & y^{(4)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix},$

Y_{train}, now has the shape (1×4)

Bias, **b,** is simply, $b = \begin{bmatrix} b_1 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}$, a (1×1) matrix



bias
[0]
b
[0]
Input Layer
Output Layer
Inputs
x_1
w_1
x_1 [0 0 1 1]
x_2 [0 1 0 1]
w_1 w_2
[0.1 0.6]
$\mathbf{Z} = W \cdot X + b$
[0 0.6 0.1 0.7]
$\vec{\hat{\mathbf{Y}}} = \sigma(z)$
[0.5 0.646 0.525 0.668]
$\vec{\hat{\mathbf{Y}}}$
(predicted outputs in a vector)
x_2
w_2
weights

The neural network is going through the following computations(forward computations marked in green):

- Our input is $X_{train} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, weight is $\mathbf{W} = \begin{bmatrix} 0.1 & 0.6 \end{bmatrix}$ and bias is $b = \begin{bmatrix} 0 \end{bmatrix}$
- **Z**, the linear node, is calculated as follows:

NOTE: Exactly like the non-vectorized calculation from before for example#1

$$
\begin{aligned}
\mathbf{Z} &= W \cdot X + b \\
&= \begin{bmatrix} 0.1 & 0.6 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \\
&= \begin{bmatrix} (0.1*0+0.6*0) & (0.1*0+0.6*1) & (0.1*1+0.6*0) & (0.1*1+0.6*1) \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \\
&= \begin{bmatrix} (0.1*0+0.6*0+0) & (0.1*0+0.6*1+0) & (0.1*1+0.6*0+0) & (0.1*1+0.6*1+0) \end{bmatrix} \\
&= \begin{array}{c} \begin{bmatrix} 0 & 0.6 & 0.1 & 0.7 \end{bmatrix} \\ \begin{matrix} z^{(1)} & z^{(2)} & z^{(3)} & z^{(4)} \end{matrix} \end{array}
\end{aligned}
$$

Bias is added element-wise in **Z**. Every entry in **Z** is the result of the linear function on the i^{th} **example**. (So, z^i is the linear function applied to i^{th} example.)

- Let's run the output of **Z** through our sigmoid function(σ), to generate predictions for each example.

Again, same output as the non-vectorized calculation from before for example#1

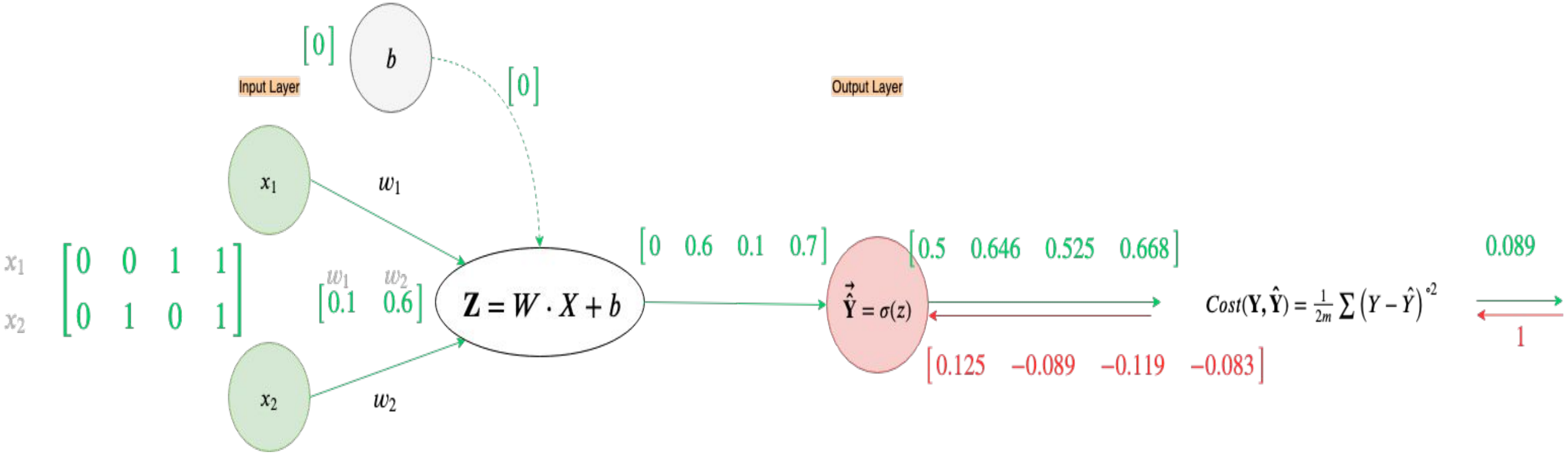
$$
\begin{aligned}
\hat{Y} &= \sigma(Z), \quad \boxed{\sigma \text{ function is applied element-wise}} \\
&= \begin{bmatrix} \frac{1}{1+e^{-z(1)}} & \frac{1}{1+e^{-z(2)}} & \frac{1}{1+e^{-z(3)}} & \frac{1}{1+e^{-z(4)}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{1+e^{-0}} & \frac{1}{1+e^{-0.6}} & \frac{1}{1+e^{-0.1}} & \frac{1}{1+e^{-0.7}} \end{bmatrix} \\
&= \begin{array}{c} \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \end{bmatrix} \\ \begin{matrix} \hat{y}^{(1)} & \hat{y}^{(2)} & \hat{y}^{(3)} & \hat{y}^{(4)} \end{matrix} \end{array}
\end{aligned}
$$

Element-wise square
(Hadamard Exponentiation)

$$
\begin{aligned}
Cost(\mathbf{Y}, \hat{\mathbf{Y}}) &= \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \\
&= \frac{1}{2m} \sum \left(Y - \hat{Y}\right)^{\circ 2} \\
&= \frac{1}{2(4)} \sum \left(\begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \end{bmatrix}\right)^{\circ 2} \\
&= \frac{1}{2(4)} \sum \begin{bmatrix} (0 - 0.5) & (1 - 0.646) & (1 - 0.525) & (1 - 0.668) \end{bmatrix}^{\circ 2} \\
&= \frac{1}{8} \sum \begin{bmatrix} (0 - 0.5)^2 & (1 - 0.646)^2 & (1 - 0.525)^2 & (1 - 0.668)^2 \end{bmatrix} \\
&= \frac{1}{8} \sum \begin{bmatrix} (-0.5)^2 & (0.354)^2 & (0.475)^2 & (0.332)^2 \end{bmatrix} \\
&= \frac{1}{8} \left((-0.5)^2 + (0.354)^2 + (0.475)^2 + (0.332)^2\right) \\
&= \frac{1}{8}(0.711) \\
&= \mathbf{0.089}
\end{aligned}
$$

Same calculation as Loss for example#1 from above

Backward Propagation



The neural network is going through the following computations(backward computations are marked in red):

- Again, the first backward computation is redundant, $\frac{\partial Cost}{\partial Cost} = 1$ - *this is the first **Upstream Gradient***
- Recall the derivative of the **Cost Function** $Cost(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$ we calculated above:

$$\frac{\partial Cost}{\partial \hat{y}^{(i)}} = -\frac{\mathbf{1}}{m}\left(y^{(i)} - \hat{y}^{(i)}\right)$$

- We can calculate the ***Local Gradient*** in one go, by also vectorizing the $\frac{Cost}{\partial \hat{y}^{(i)}}$ computation as below:

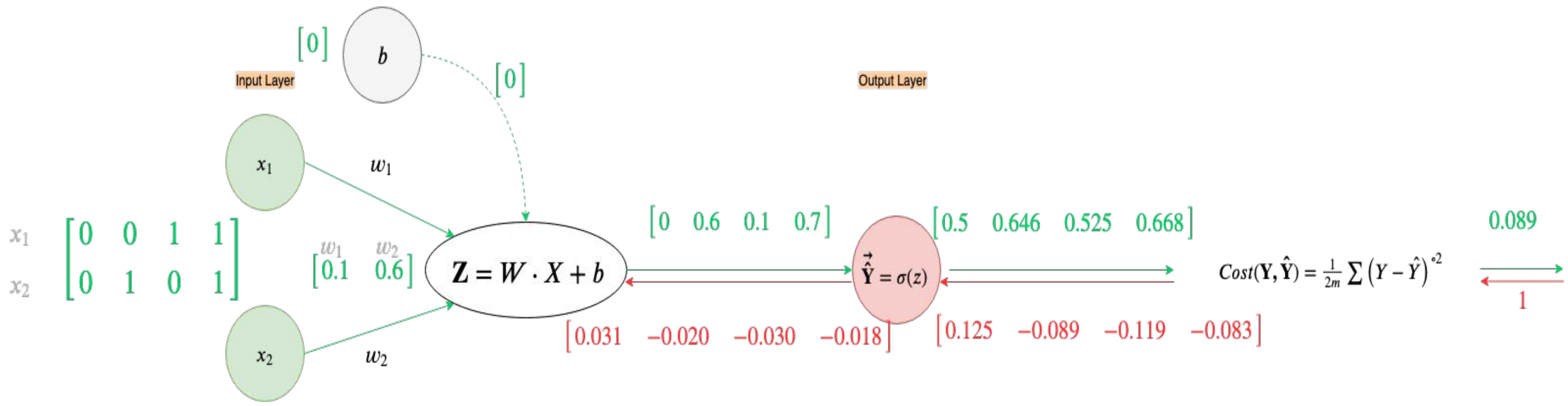
$$\begin{aligned} \frac{\partial Cost}{\partial \hat{Y}} &= -\frac{1}{m}(Y - \hat{Y}) \\ &= -\frac{1}{4}\left(\begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \end{bmatrix}\right) \\ &= -\frac{1}{4}\left(\begin{bmatrix} -0.5 & 0.354 & 0.475 & 0.332 \end{bmatrix}\right) \\ &= \begin{bmatrix} 0.125 & -0.089 & -0.119 & -0.083 \end{bmatrix} \end{aligned}$$

Same as the Loss calculation for example# 1 above

- As before we'll combine the Local and the upstream gradient and send it back to the red node:

$$\begin{aligned} \frac{\partial Cost}{\partial \hat{Y}} &= UpstreamGradient * LocalGradient \\ &= \frac{\partial Cost}{\partial Cost} * \frac{\partial Cost}{\partial \hat{Y}} \\ &= 1 * \begin{bmatrix} 0.125 & -0.089 & -0.119 & -0.083 \end{bmatrix} \\ &= \begin{bmatrix} 0.125 & -0.089 & -0.119 & -0.083 \end{bmatrix} \\ & \begin{bmatrix} \frac{\partial Cost}{\partial \hat{y}^{(1)}} & \frac{\partial Cost}{\partial \hat{y}^{(2)}} & \frac{\partial Cost}{\partial \hat{y}^{(3)}} & \frac{\partial Cost}{\partial \hat{y}^{(4)}} \end{bmatrix} \end{aligned}$$

Backward Propagation



Now, we can use the chain rule to easily derive the derivative:

$$
\begin{aligned}
\frac{\mathrm{d}\hat{y}}{\mathrm{d}z} &= \frac{\mathrm{d}\hat{y}}{\mathrm{d}u} * \frac{\mathrm{d}u}{\mathrm{d}z} \\
&= \left(-\frac{1}{u^2}\right) * (-e^{-z}) \\
&\text{substitute } u = 1 + e^{-z} \\
&= \left(-\frac{1}{(1+e^{-z})^2}\right) * (-e^{-z}) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}} * \frac{e^{-z}}{(1+e^{-z})} \\
&= \frac{1}{1+e^{-z}} * \frac{1+e^{-z}-1}{(1+e^{-z})}, \text{ 1 added and subtracted, overall numerator remains same} \\
&= \frac{1}{1+e^{-z}} * \left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right) \\
&= \frac{1}{1+e^{-z}} * \left(1 - \frac{1}{1+e^{-z}}\right) \\
&\text{substitute } \frac{1}{1+e^{-z}} = \hat{y} \\
&= \hat{y} * \left(\mathbf{1} - \hat{y}\right)
\end{aligned}
$$

The neural network is going through the following computations(backward computations are marked in red):

- Our ***Upstream Gradient*** in this step is :

$$\frac{\partial Cost}{\hat{Y}} = \begin{array}{c} \begin{bmatrix} 0.125 & -0.089 & -0.119 & -0.083 \end{bmatrix} \\ \begin{bmatrix} \frac{\partial Cost}{\partial \hat{y}^{(1)}} & \frac{\partial Cost}{\partial \hat{y}^{(2)}} & \frac{\partial Cost}{\partial \hat{y}^{(3)}} & \frac{\partial Cost}{\partial \hat{y}^{(4)}} \end{bmatrix} \end{array}$$

- Recall the derivative of the sigmoid/logistic function(σ): $\frac{\partial \hat{y}}{\partial z} = \hat{y} - (1 - \hat{y})$
- We'll use a vectorized version of the derivative of the sigmoid function as our ***Local Gradient***:

Hadamard Product
(element-wise multiplication)

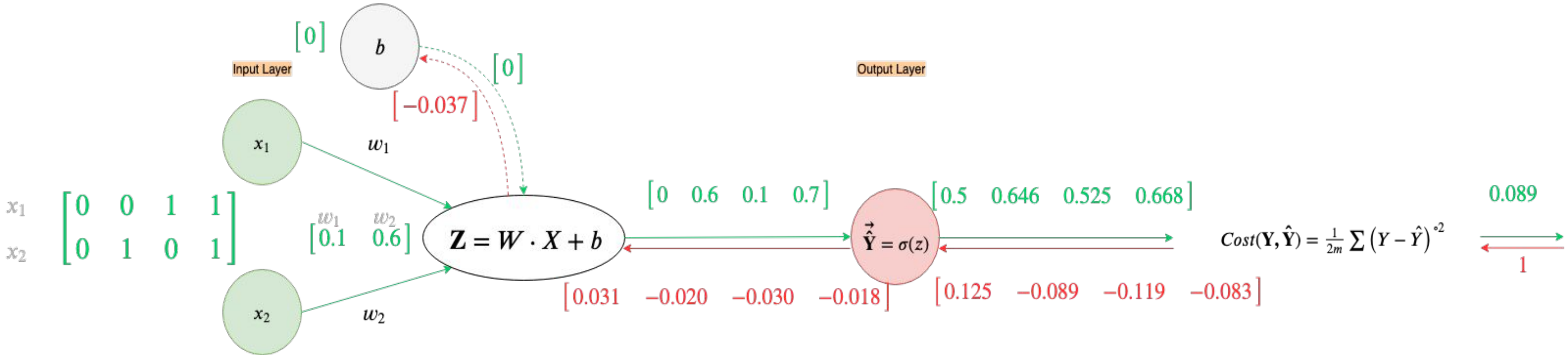
$$\begin{aligned} \frac{\partial \hat{Y}}{\partial Z} &= \hat{\mathbf{Y}}(\mathbf{1} - \hat{\mathbf{Y}}) \\ &= \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \end{bmatrix} \odot \left(1 - \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \end{bmatrix}\right) \\ &= \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \end{bmatrix} \odot \begin{bmatrix} 0.5 & 0.354 & 0.475 & 0.332 \end{bmatrix} \\ &= \begin{bmatrix} (0.5 * 0.5) & (0.646 * 0.354) & (0.525 * 0.475) & (0.668 * 0.332) \end{bmatrix} \\ &= \begin{bmatrix} 0.25 & 0.229 & 0.249 & 0.222 \end{bmatrix} \end{aligned}$$

Same local gradient calculation
as example#1, above

- We'll combine the upstream and local gradient and send them back to the white node(Z):

$$\begin{aligned} \frac{\partial Cost}{\partial Z} &= UpstreamGradient * LocalGradient \\ &= \frac{\partial Cost}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z} \\ &= \begin{bmatrix} 0.125 & -0.089 & -0.119 & -0.083 \end{bmatrix} \odot \begin{bmatrix} 0.25 & 0.229 & 0.249 & 0.222 \end{bmatrix} \\ &= \begin{bmatrix} (0.125 * 0.25) & (-0.089 * 0.229) & (-0.119 * 0.249) & (-0.083 * 0.222) \end{bmatrix} \\ &= \begin{array}{c} \begin{bmatrix} 0.031 & -0.020 & -0.030 & -0.018 \end{bmatrix} \\ \begin{bmatrix} \frac{\partial Cost}{\partial z^{(1)}} & \frac{\partial Cost}{\partial z^{(2)}} & \frac{\partial Cost}{\partial z^{(3)}} & \frac{\partial Cost}{\partial z^{(4)}} \end{bmatrix} \end{array} \end{aligned}$$

Back Propagation



The neural network is going through the following computations(backward computations are marked in red):

- We have propagated the upstream gradient back enough the calculate the gradient with respect to our weights W and bias b.
- Our ***Upstream Gradient*** in this step is :

$$\frac{\partial Cost}{\partial Z} = \begin{bmatrix} 0.031 & -0.020 & -0.030 & -0.018 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial Cost}{\partial z^{(1)}} & \frac{\partial Cost}{\partial z^{(2)}} & \frac{\partial Cost}{\partial z^{(3)}} & \frac{\partial Cost}{\partial z^{(4)}} \end{bmatrix}$$

- This time our ***Z*** node is the vectorized implementation of a linear function: $Z = W \cdot X + b$, where W(weights) and X(data) are being dotted(dot product) with bias added to each element of the dot product.
- The **Local Gradients** of this vectorized function are:

1. $\frac{\partial Z}{\partial W} = X^T = X_{train}^T = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$
2. $\frac{\partial Z}{\partial b} = 1$

*(Though we will not use this, as we don't want to change our input data, but the local gratient with respect to **X** is :*
$\frac{\partial Z}{\partial X} = W^T = \begin{bmatrix} 0.1 \\ 0.6 \end{bmatrix}$)

- We'll combine local and upstream gradients to figure out how much to change our weights and bias.

$$
\begin{aligned}
\frac{\partial Cost}{\partial W} &= UpstreamGradient * LocalGradient \\
&= \frac{\partial Cost}{\partial Z} \cdot \frac{\partial Z}{\partial W} \\
&= \begin{bmatrix} 0.031 & -0.02 & -0.03 & -0.018 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \\
&= \begin{bmatrix} (0 * 0.031 + 0 * -0.02 + 1 * -0.03 + 1 * -0.018) & (0 * -0.031 + 1 * -0.02 + 0 * -0.03 + 1 * -0.018) \end{bmatrix} \\
&= \begin{bmatrix} -0.048 & -0.038 \end{bmatrix} \\
& \quad \begin{bmatrix} \frac{\partial Cost}{\partial w_1} & \frac{\partial Cost}{\partial w_2} \end{bmatrix} \\
\frac{\partial Cost}{\partial b} &= \sum UpstreamGradient * LocalGradient \\
&= \sum \frac{\partial Cost}{\partial Z} * \frac{\partial Z}{\partial b} \\
&= \sum \begin{bmatrix} 0.031 & -0.02 & -0.03 & -0.018 \end{bmatrix} * 1 \\
&= \sum \begin{bmatrix} -0.031 & -0.02 & -0.03 & -0.018 \end{bmatrix} \\
&= [(0.031) + (-0.02) + (-0.03) + (-0.018)] \\
&= \mathbf{[-0.037]}
\end{aligned}
$$

Gradient Descent Update

To calculate new weights(*W*) and bias(*b*) we move in the negative direction of the gradient

Recall, our current Weight vector is $W = \begin{bmatrix} 0.1 & 0.6 \end{bmatrix}$, $\boldsymbol{\alpha} = \mathbf{1}$ and $\frac{\partial Cost}{\partial W} = \begin{bmatrix} -0.048 & -0.038 \end{bmatrix}$

The new Weights are:

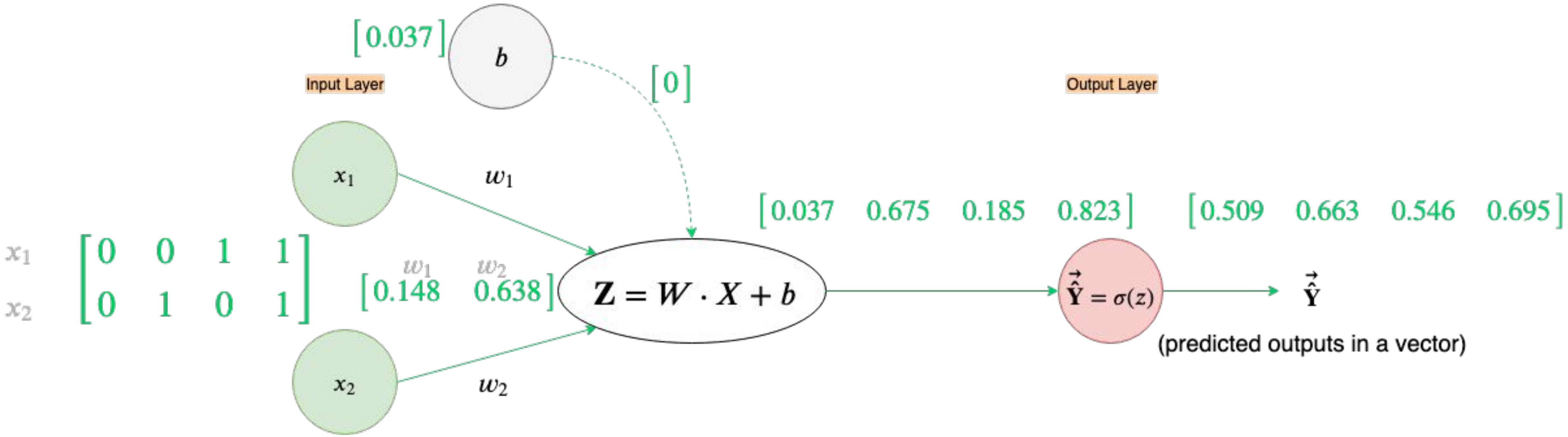
$$
\begin{aligned}
W &= W - \alpha \frac{\partial Cost}{\partial W} \\
&= \begin{bmatrix} 0.1 & 0.6 \end{bmatrix} - \left(1 * \begin{bmatrix} -0.048 & -0.038 \end{bmatrix}\right) \\
&= \begin{bmatrix} 0.1 & 0.6 \end{bmatrix} - \begin{bmatrix} -0.048 & -0.038 \end{bmatrix} \\
&= \begin{bmatrix} 0.1 + 0.048 & 0.6 + 0.038 \end{bmatrix} \\
&= \begin{bmatrix} 0.148 & 0.638 \end{bmatrix}
\end{aligned}
$$

Our current Bias vector is $b = \begin{bmatrix} 0 \end{bmatrix}$, $\boldsymbol{\alpha} = \mathbf{1}$ and $\frac{\partial Cost}{\partial b} = \mathbf{[-0.037]}$

The new Bias is:

$$
\begin{aligned}
b &= b - \alpha \frac{\partial Cost}{\partial b} \\
&= \begin{bmatrix} 0 \end{bmatrix} - \left(1 * \begin{bmatrix} -0.037 \end{bmatrix}\right) \\
&= \begin{bmatrix} 0 \end{bmatrix} - \begin{bmatrix} -0.037 \end{bmatrix} \\
&= \begin{bmatrix} 0 + 0.037 \end{bmatrix} \\
&= \begin{bmatrix} 0.037 \end{bmatrix}
\end{aligned}
$$

Forward Propagation



The neural network is going through the following computations(forward computations marked in green):

- Our input is $X_{train} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, weight is $\mathbf{W} = \begin{bmatrix} 0.148 & 0.638 \end{bmatrix}$ and bias is $b = \cancel{\begin{bmatrix} 0.099 \end{bmatrix}}$ **[0.037]**

$$
\begin{aligned}
\mathbf{Z} &= W \cdot X + b \\
&= \begin{bmatrix} 0.148 & 0.638 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0.037 \end{bmatrix} \\
&= \begin{bmatrix} (0.148*0+0.638*0) & (0.148*0+0.638*1) & (0.148*1+0.638*0) & (0.148*1+0.638*1) \end{bmatrix} + \begin{bmatrix} 0.037 \end{bmatrix} \\
&= \begin{bmatrix} (0.148*0+0.638*0+0.037) & (0.148*0+0.638*1+0.037) & (0.148*1+0.638*1+0.037) & (0.148*1+0.638*1+0.037) \end{bmatrix} \\
&= \begin{bmatrix} 0.037 & 0.675 & 0.185 & 0.823 \end{bmatrix} \\
& \quad \begin{bmatrix} z^{(1)} & z^{(2)} & z^{(3)} & z^{(4)} \end{bmatrix}
\end{aligned}
$$

Bias is added element-wise in **Z**. Every entry in **Z** is the result of the linear function on the i^{th} **example**.
(So, $z^{(i)}$ is the linear function applied to i^{th} example.)

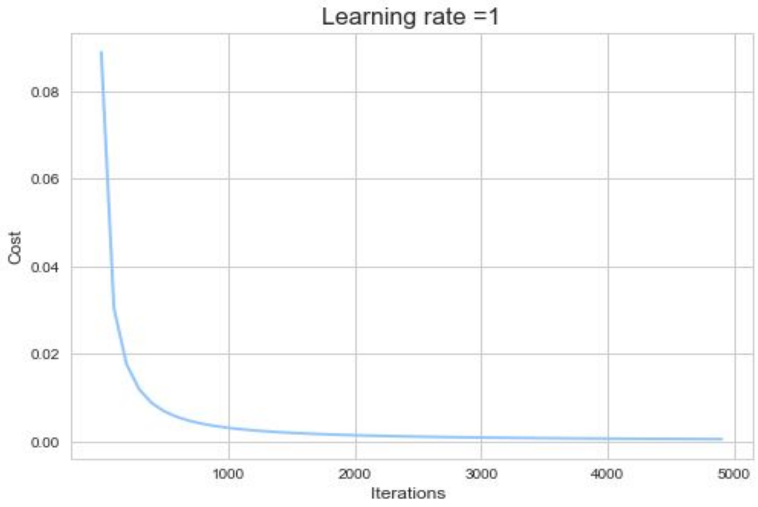
- Let's run the output of **Z** through our sigmoid function(σ), to generate predictions for each example.

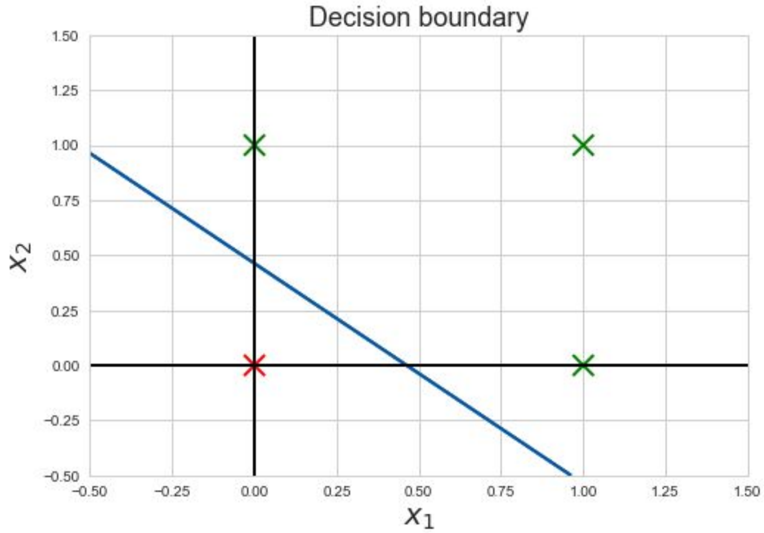
$$
\begin{aligned}
\hat{Y} &= \sigma(Z), \quad \boxed{\sigma \text{ function is applied element-wise}} \\
&= \begin{bmatrix} \frac{1}{1+e^{-z(1)}} & \frac{1}{1+e^{-z(2)}} & \frac{1}{1+e^{-z(3)}} & \frac{1}{1+e^{-z(4)}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{1+e^{-0.037}} & \frac{1}{1+e^{-0.675}} & \frac{1}{1+e^{-0.185}} & \frac{1}{1+e^{-0.823}} \end{bmatrix} \\
&= \begin{bmatrix} 0.509 & 0.663 & 0.546 & 0.695 \end{bmatrix}
\end{aligned}
$$

Cost - 2nd Iteration

$$
\begin{aligned}
Cost(\mathbf{Y}, \hat{\mathbf{Y}}) &= \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \\
&= \frac{1}{2m} \sum \left(Y - \hat{Y}\right)^{\circ 2} \\
&= \frac{1}{2(4)} \sum \left(\begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 0.509 & 0.663 & 0.546 & 0.695 \end{bmatrix}\right)^{\circ 2} \\
&= \frac{1}{2(4)} \sum \begin{bmatrix} (0 - 0.509) & (1 - 0.663) & (1 - 0.546) & (1 - 0.695) \end{bmatrix}^{\circ 2} \\
&= \frac{1}{8} \sum \begin{bmatrix} (0 - 0.509)^2 & (1 - 0.663)^2 & (1 - 0.546)^2 & (1 - 0.695)^2 \end{bmatrix} \\
&= \frac{1}{8} \sum \begin{bmatrix} (-0.509)^2 & (0.337)^2 & (0.454)^2 & (0.305)^2 \end{bmatrix} \\
&= \frac{1}{8} \left((-0.509)^2 + (0.337)^2 + (0.454)^2 + (0.305)^2\right) \\
&= \frac{1}{8}(0.672) \\
&= \mathbf{0.084}
\end{aligned}
$$

Cost Curve and Decision Boundary after 5k Epochs





Continue...

References

- https://end-to-end-machine-learning.teachable.com/
- https://medium.com/towards-artificial-intelligence/nothing-but-numpy-understanding-creating-neural-networks-with-computational-graphs-from-scratch-6299901091b0
-