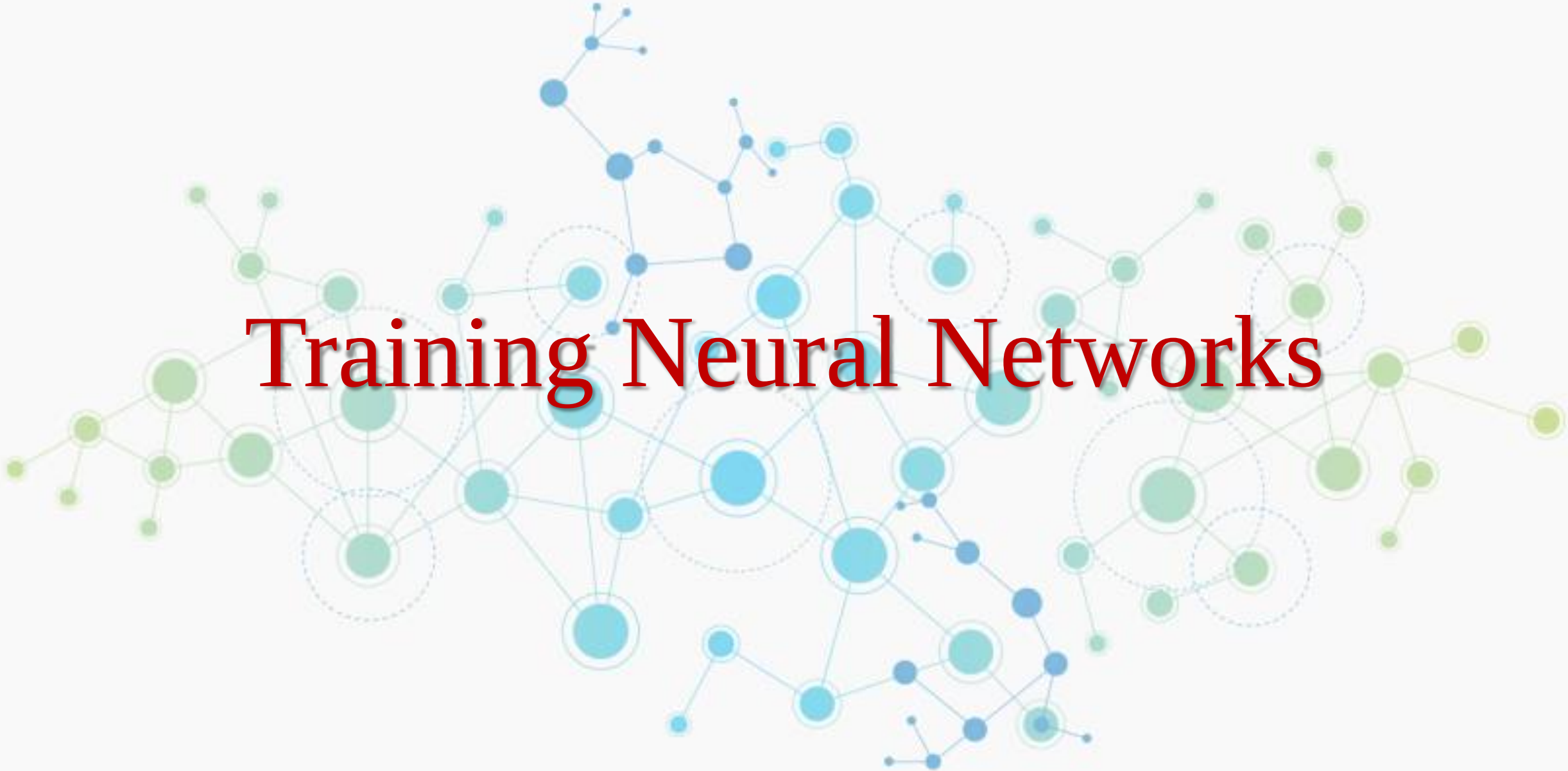
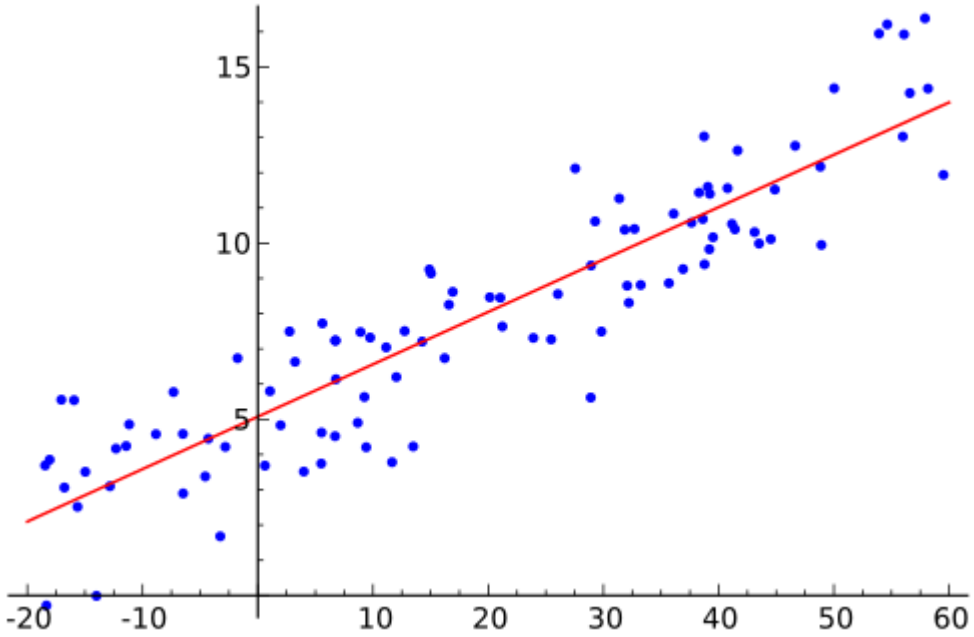


Training Neural Networks

Linear Regression



$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{Y}_i - Y_i)^2$$

Loss Function Squaring the Error difference

Benefits of squaring

Squaring always gives a positive value, so the sum will not be zero.

Squaring emphasizes larger differences—a feature that turns out to be both good and bad (think of the effect outliers have).

Loss Function for logistic regression

$L(\hat{y}, y) = -((y \log \hat{y}) + (1-y)(\log(1-\hat{y}))$

If $y=1$ $L(\hat{y}, y) = -(\log \hat{y})$ Here, we want $\log \hat{y}$ large , want $\hat{y}$ large that means $\hat{y} \approx 1$

If $y=0$ $L(\hat{y}, y) = -(\log(1-\hat{y})$ Here we want $(\log(1-\hat{y})$ large, want $\hat{y}$ small means $\hat{y} \approx 0$

It helps us also to resolve the local minima problem and gives us a convex problem

Cost Function: Average of Loss across the whole input

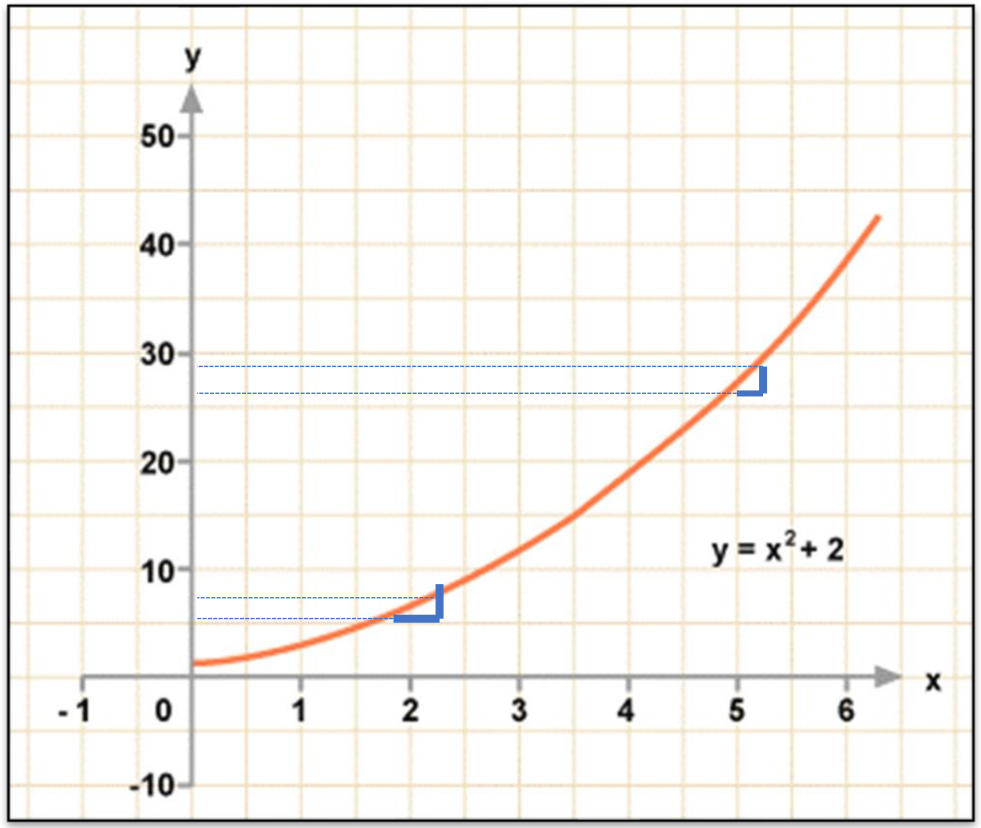
$\hat{y}^{(i)}=\sigma\,(w^{T}x^{(i)} + b)$ where $\sigma(z^{(i)})= {}^{1}/_{1+e^{-z^{(i)}}}$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L\,(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m} (y^{(i)}\log \hat{y}^{(i)} + (1-y^{(i)})\log (1-\hat{y}^{(i)}))$$

Our objective is to find w,b that minimize J(w,b)

Derivatives



$y=f(x)=x^2+2$

$\frac{df(x)}{dx} = 2x$

x=2 f(x)=6

x=2.0001 f(x)=6.00040001

Slope at x=2 is .0004/.0001= 4

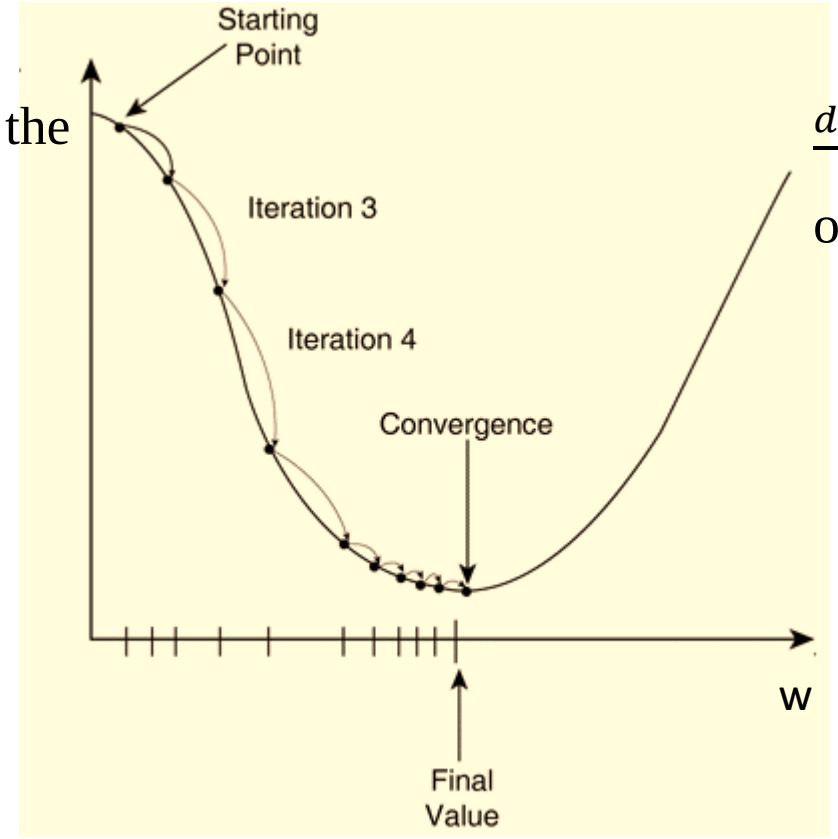
x=5 f(x) =27

x=5.0001 f(x) = 27.00100001

Slope at a =5 is .0010/.0001= 10

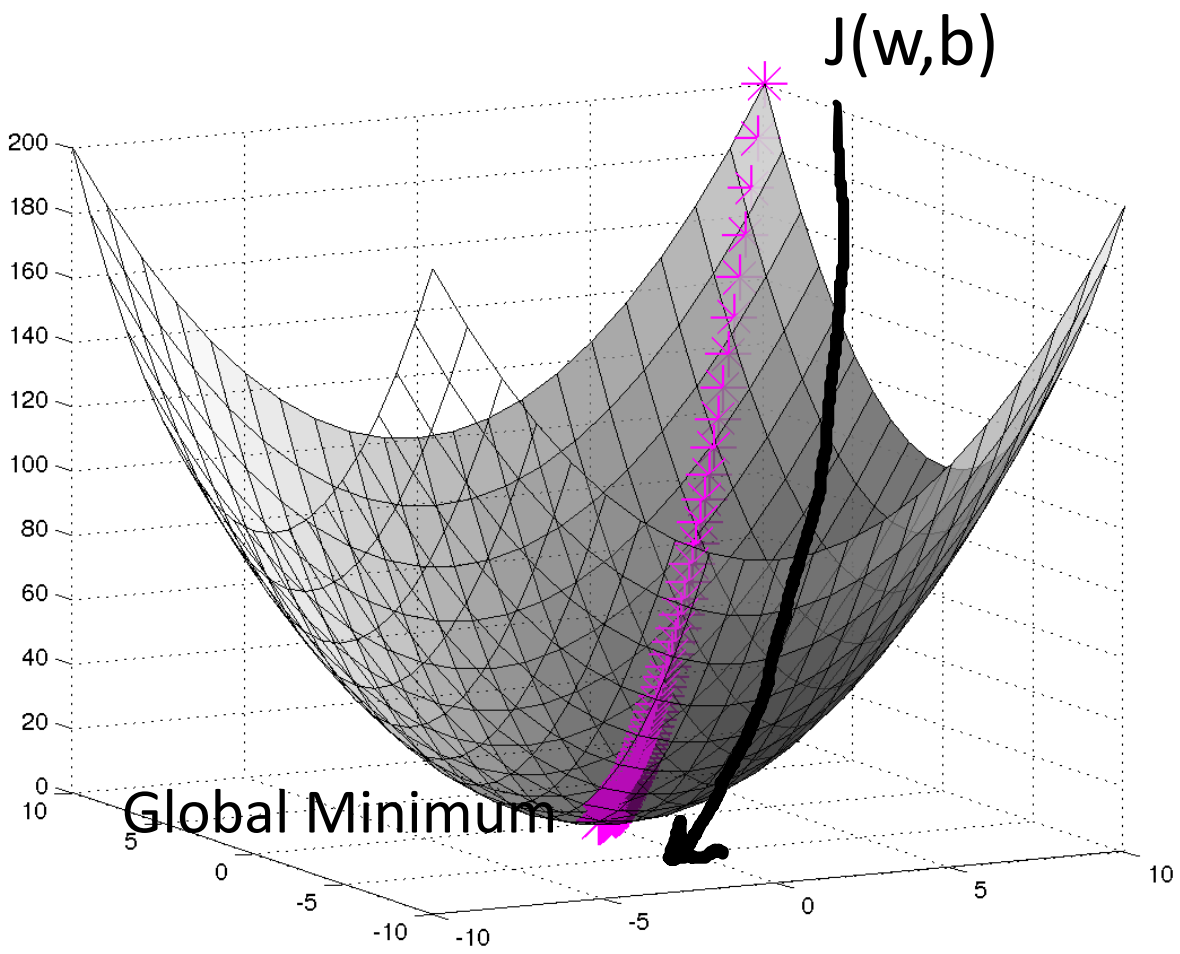
Gradient Descent single dimension

$\frac{dj(w)}{dw}$ is less than the optimum value



$\frac{dj(w)}{dw}$ is more than the optimum value

Gradient Descent



How the weights will change

w= w-α $\frac{dJ(w,b)}{dw}$ (Mathematicians write it as w= w-α $\frac{\partial(w,b)}{\partial w}$) Python w= w-α dw

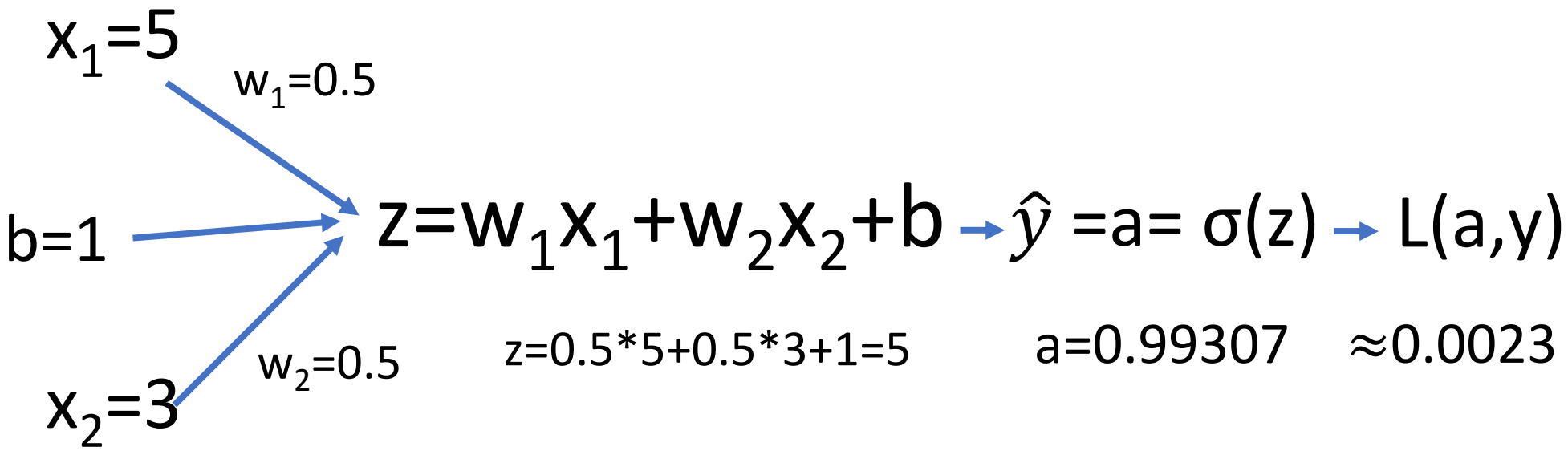
b= b-α $\frac{dJ(w,b)}{db}$ (Mathematicians write it as b= b-α $\frac{\partial(w,b)}{\partial b}$) Python b= b-α db

This process will repeat until we find or reach near the global minimum.

α is the learning rate which will decide that how fast or slow we are going towards the global minima. How big or small steps we want to take. Both have their own limitations and advantages.

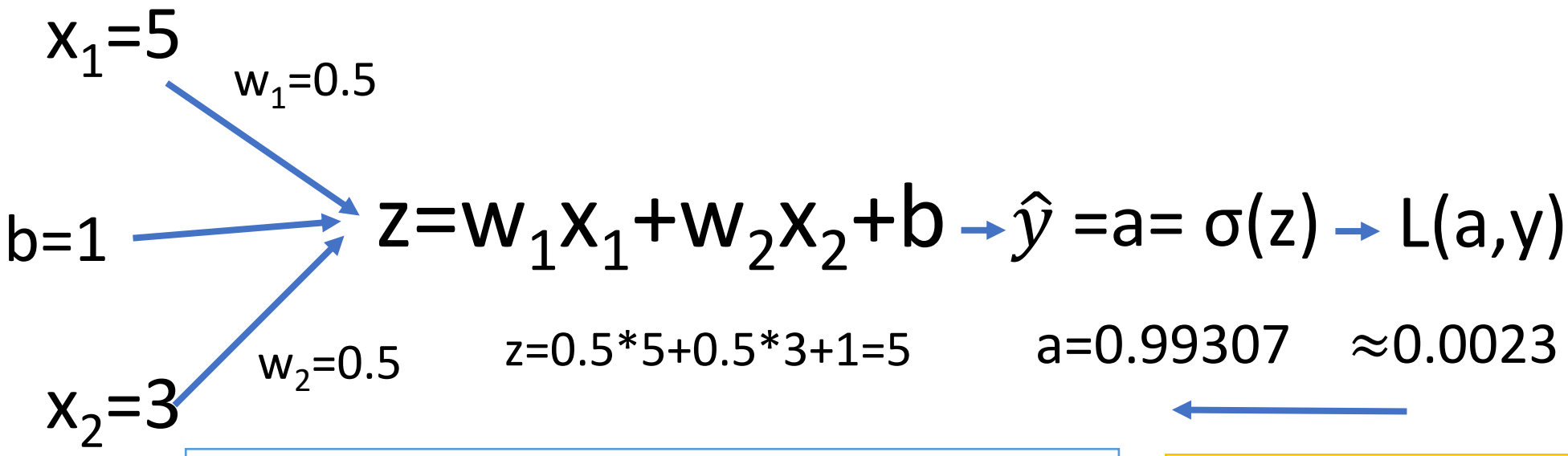
Derivatives in logistic regression

Original Value of y=1



Derivatives in logistic regression

Original Value of y=1



db=dz

dz=$\frac{dL(a,y)}{dz}$ = $\frac{dL}{da}\frac{da}{dz}$ = ($\frac{-y}{a}$ + $\frac{1-y}{1-a}$)*a(1-a)=a-y=-0.00693

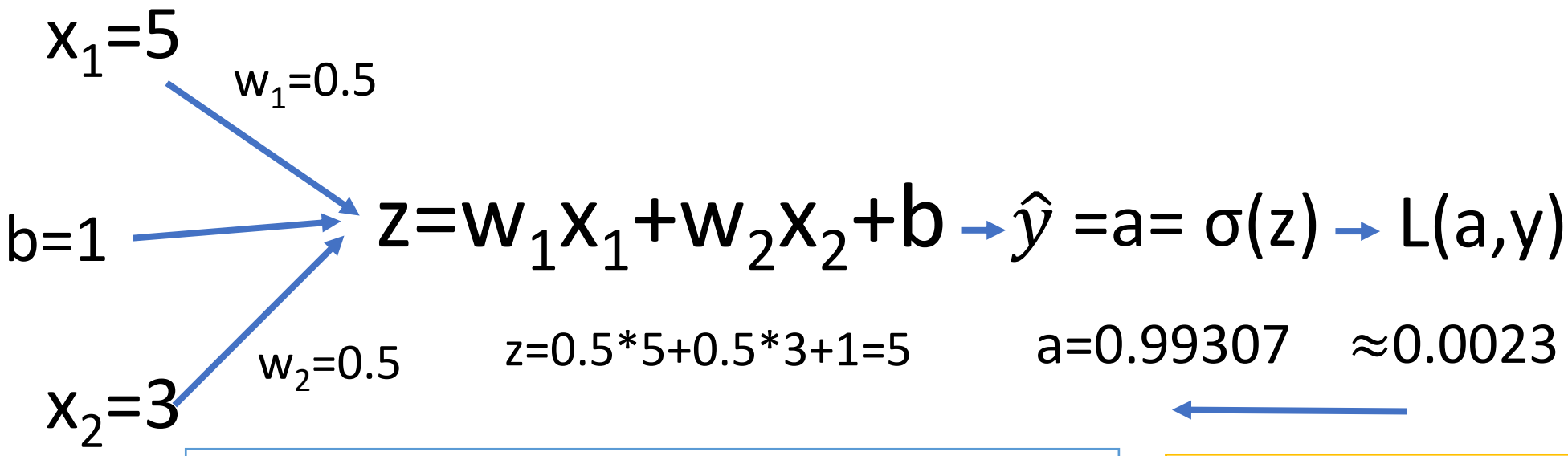
da=$\frac{dL(a,y)}{da}$ = $\frac{-y}{a}$ + $\frac{1-y}{1-a}$ = -1.006978

dw_1= $\frac{dL(a,y)}{dw_1}$ =x_1.dz=5*-0.00695=-0.03475

dw_2= $\frac{dL(a,y)}{dw_2}$ =x2.dz=3*-0.00695=-0.02085

Derivatives in logistic regression

Original Value of y=1



db=dz

dz=$\frac{dL(a,y)}{dz}$ = $\frac{dL}{da}\frac{da}{dz}$ = ($\frac{-y}{a}+\frac{1-y}{1-a}$)*a(1-a)=a-y=-0.00693

da=$\frac{dL(a,y)}{da}$ = $\frac{-y}{a}+\frac{1-y}{1-a}$ = -1.006978

α= 0.01

dw_1= $\frac{dL(a,y)}{dw_1}$ =x_1.dz=5*-0.00695=-0.03475

dw_2= $\frac{dL(a,y)}{dw_2}$ =x2.dz=3*-0.00695=-0.02085

$w_1=w_1-\alpha dw_1$ = 0.5-(0.01*-0.03475)=0.503475

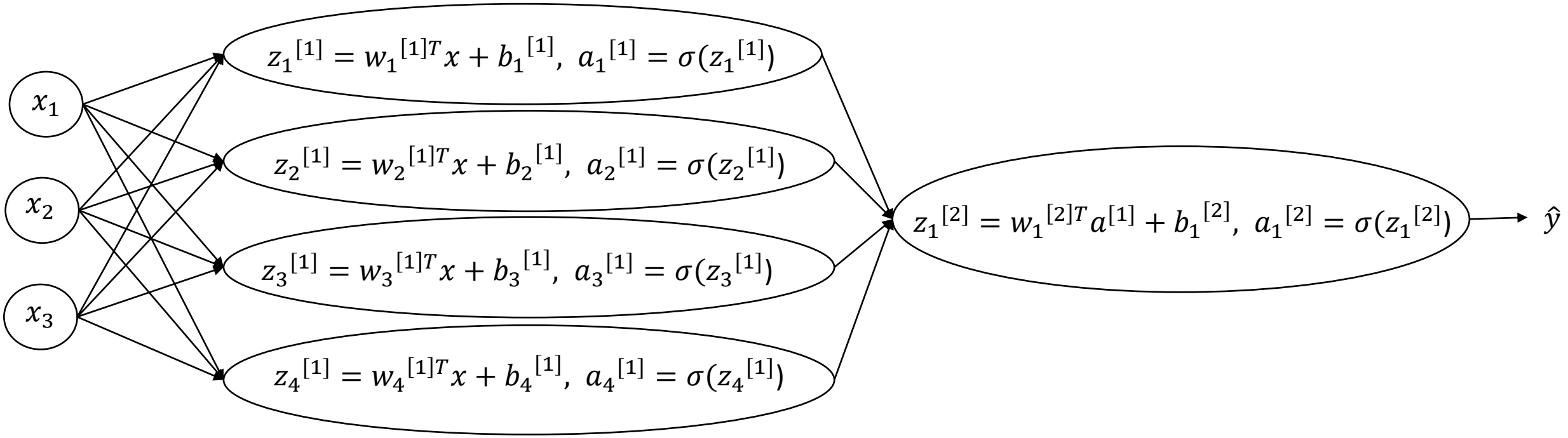
$w_2=w_2-\alpha dw_2$ = 0.5-(0.01*-0.02085)=0.502085

b=b- αdb =1-(0.01)*(-0.00693)=1.000693

Neural Network Representation

2 layer neural network

$a_i^{\,l}$ $a^{layer}_{node\ in\ layer}$



$a^{[0]}$ Layer 0
Input Layer

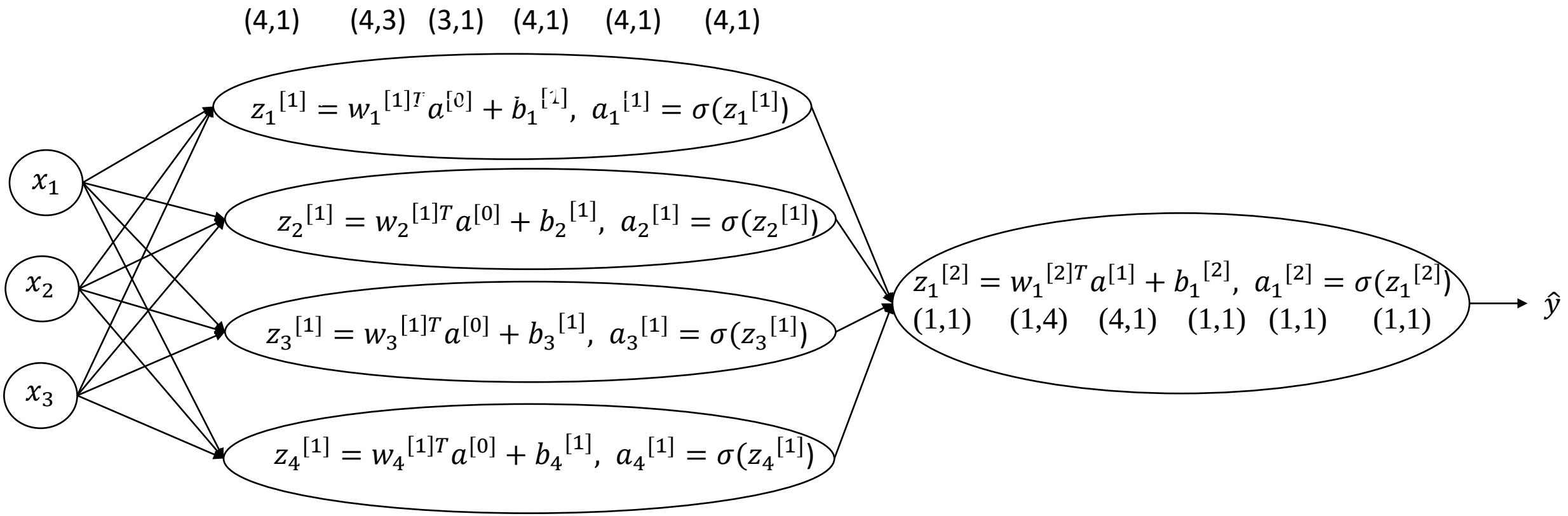
$a^{[1]}$ Layer 1 $w^{[1]}$ $b^{[1]}$ $z^{[1]}$
Hidden Layer

$a^{[2]}$ Layer 2 $w^{[2]}$ $b^{[2]}$ $z^{[2]}$
Output Layer

Neural Network Representation

2 layer neural network

$a_i^{\,l}$ $a^{layer}_{node\ in\ layer}$



$a^{[0]}$ Layer 0
Input Layer

$a^{[1]}$ Layer 1 $w^{[1]}$ $b^{[1]}$ $z^{[1]}$
Hidden Layer

$a^{[2]}$ Layer 2 $w^{[2]}$ $b^{[2]}$ $z^{[2]}$
Output Layer

Linearity Vs. Non-Linearity

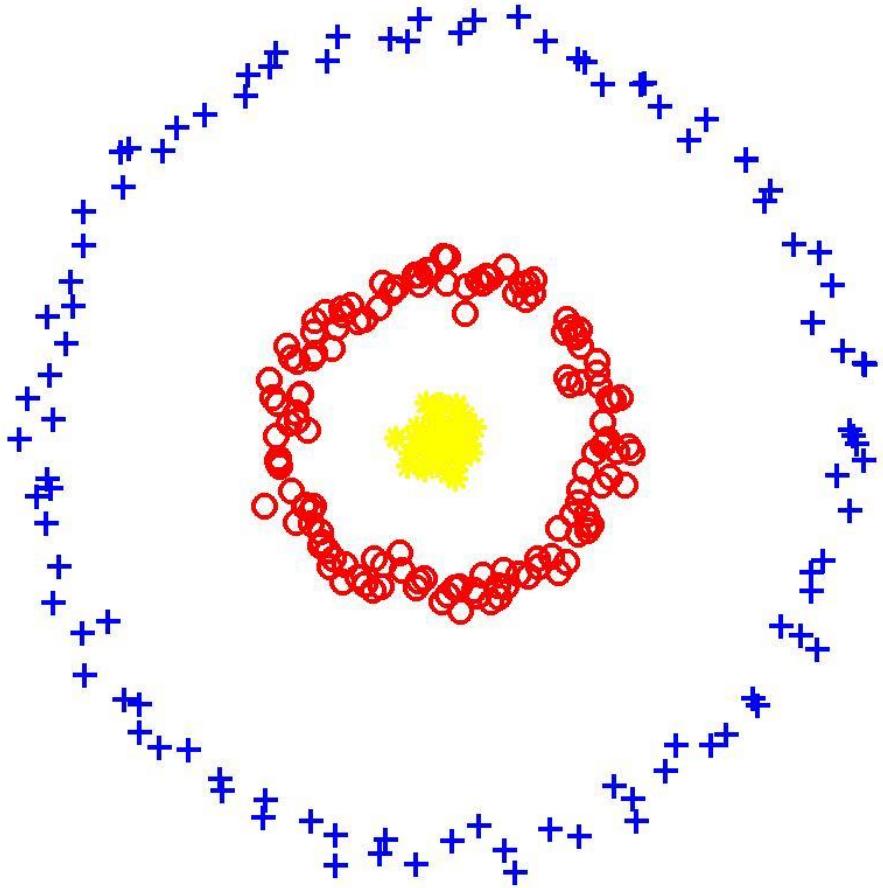
Simple multiplication of weights with inputs and giving the outputs will be linear function.

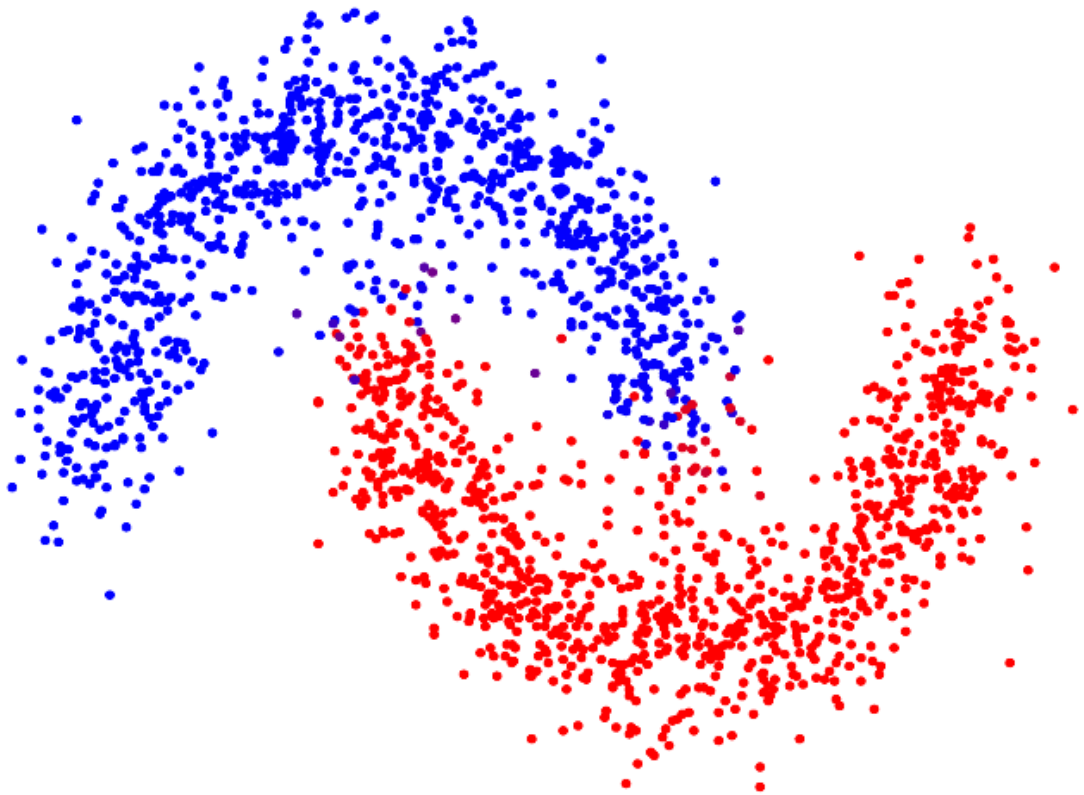
A linear function is just a polynomial of one degree and will not be able to learn complex functions.

Linear functions don’t have much expressive power and will loose out when solving complex problems.

Without activation functions (which will help us to achieve non-linearity) Neural Network will not be able to learn unstructured data like images, audio data, videos and text.

Non Linear Patterns





Activation Functions

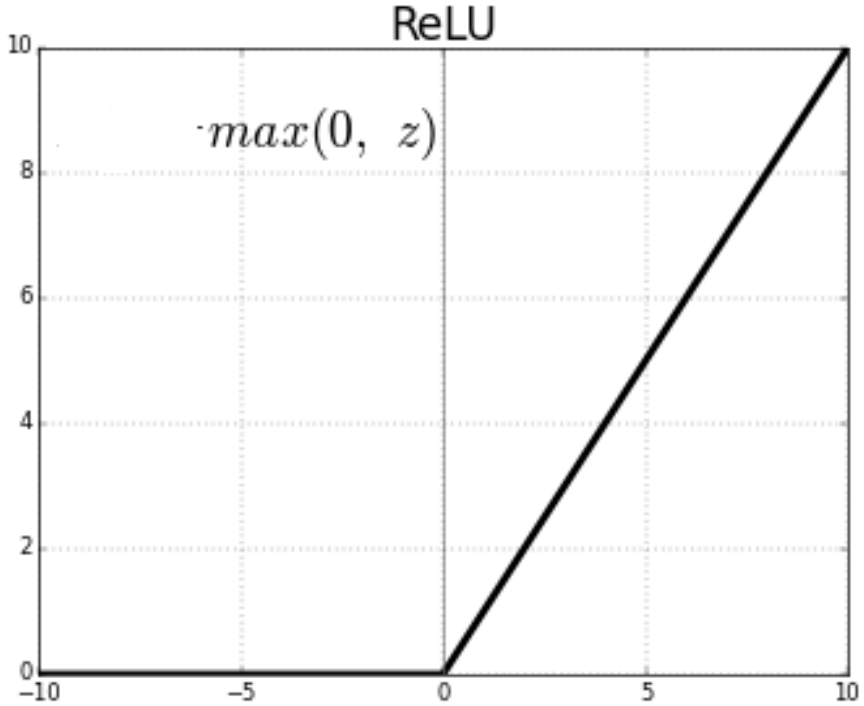
Output from every neuron is generated after applying activation Function to the values being calculated with set of inputs and their weights.

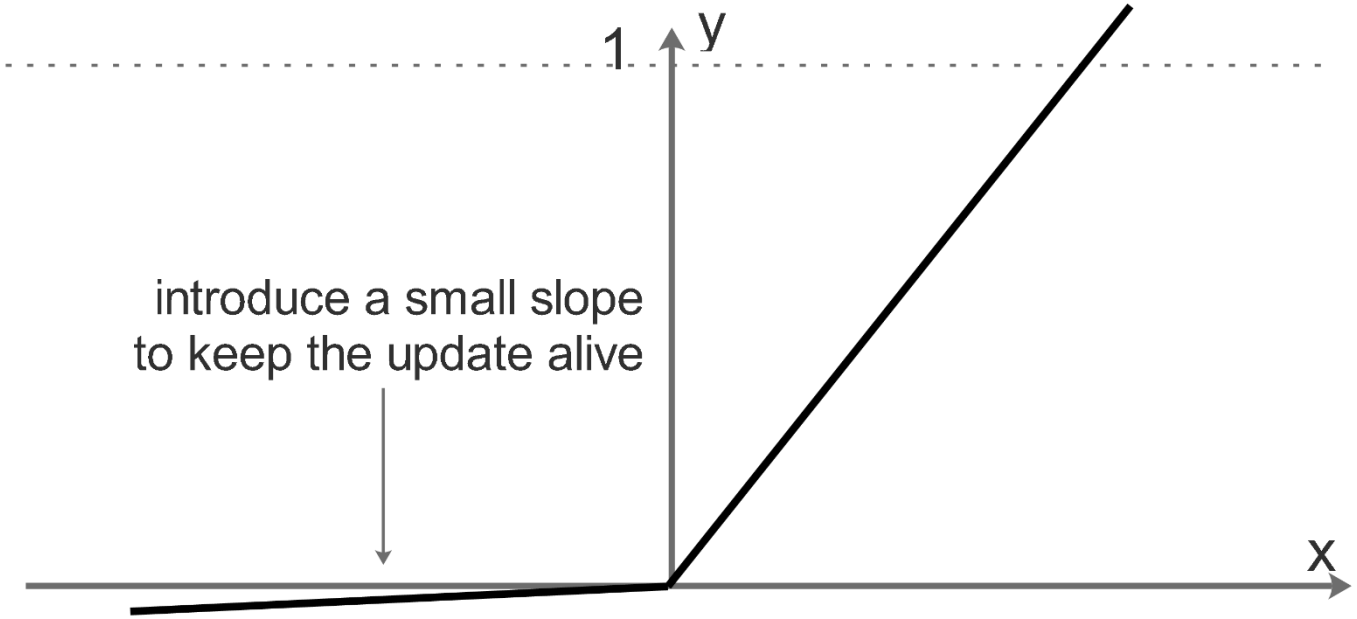
Most Popular Activation Functions are ReLU (Rectified Linear Units) and Sigmoid, Softmax and tanh

Activation functions should be differentiable. Non-linear Activation functions help you to bring non-linearity in the system

To transform/squash your input to a different space/domain and do some kind of thresholding

ReLU and Leaky ReLU





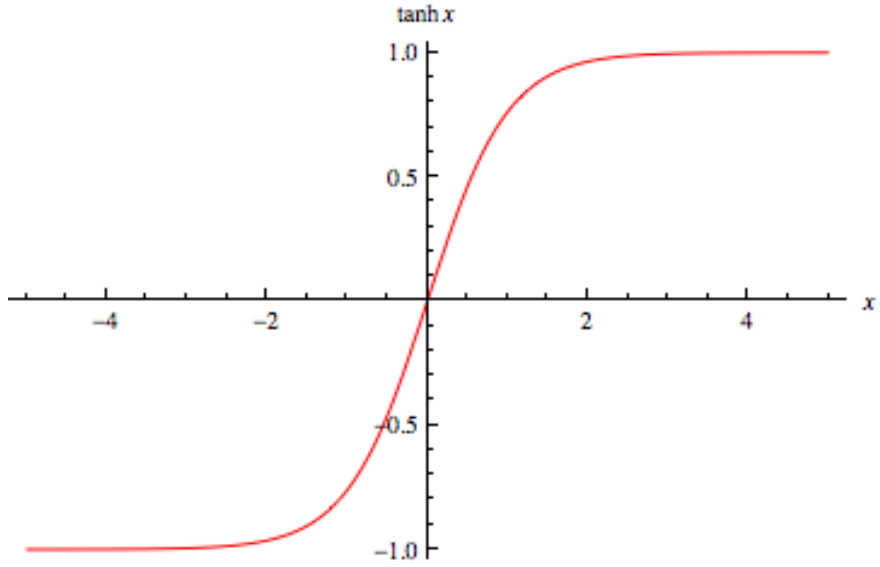
Leaky ReLU max(0.1z,z)

Rectified Linear Unit is the default activation function being used now. If you see the left part of the function, you can see that it is not zero, but almost zero. To resolve the issue of dead neurons people use Leaky ReLU

Activation Functions: tanh

Tanh activation function is more preferred than the sigmoid function.
It is the shifted version of sigmoid function with a mean value of zero.
It has better centering effect for the activation function to be used on the hidden layer.
For binary classification problem at the output layer we use the Sigmoid function.

$$\tanh = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Problem in both Sigmoid and tanh is that the slope of the curve except the middle region is too small and goes close to zero. This create a serious issue with gradient descent and learning becomes very slow.

Softmax Function

When we have to classify in multiple categories then softmax function is useful. For example if you want to categorize pictures into

A) scene b) Animals c) Humans d) Vehicles then in that case we will have four outputs from the softmax function which will give us the probabilities of each of these categories.

Sum of the probabilities will be one and that with the highest probability will be shown as the answer.

Understanding softmax

$$z^{[L]} = \begin{pmatrix} 5 \\ 2 \\ -1 \\ 3 \end{pmatrix} \qquad t = \begin{pmatrix} e^{5} \\ e^{2} \\ e^{-1} \\ e^{3} \end{pmatrix}$$

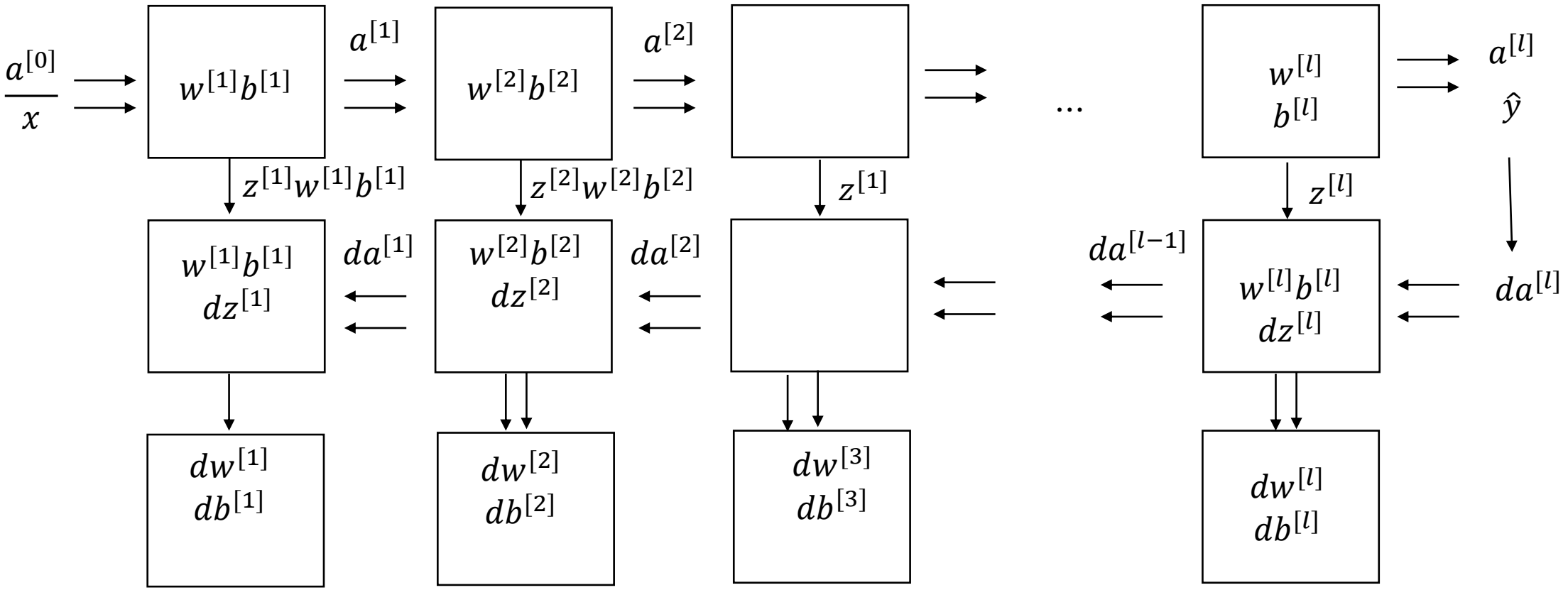
$$a^{[l]} = g^{[L]}\left(z^{[L]}\right) = \begin{pmatrix} e^{5}/(e^{5}+e^{2}+e^{-1}+e^{3}) \\ e^{2}/(e^{5}+e^{2}+e^{-1}+e^{3}) \\ e^{-1}/(e^{5}+e^{2}+e^{-1}+e^{3}) \\ e^{3}/(e^{5}+e^{2}+e^{-1}+e^{3}) \end{pmatrix} = \begin{pmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{pmatrix}$$

"Hard Max"

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Softmax regression generalizes logistic regression to C classes. If c=2, softmax reduces to logistic regression.

Forward and Backward Functions



$$w^{[l]} = w^{[l]} - \propto dw^{[l]}$$
$$b^{[l]} = b^{[l]} - \propto db^{[l]}$$

Public Notice regarding Use of Images/Information

This document contains images obtained by routine Google Images searches. Some of these images may perhaps be under copyright. They are included here for educational and noncommercial purposes and are considered to be covered by the doctrine of Fair Use. In any event they are easily available from Google Images.

It's not feasible to give full scholarly credit to the creators of the images/information. We hope they can be satisfied with the positive role they are playing in the educational process.


Thank
you!