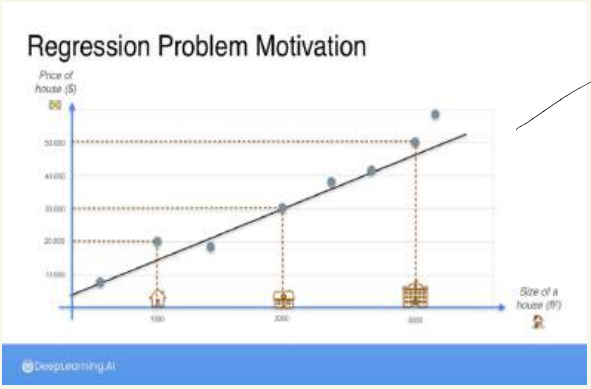
Week 3 Optimizing In Neural Networks

Regression with a perceptron

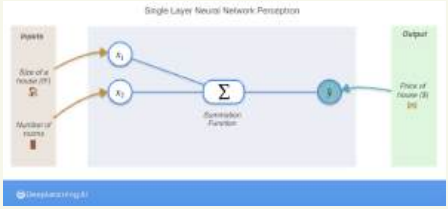


The goal is to find the best possible line

we have 1 feature here what if we add another feature

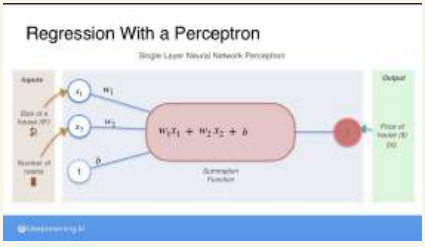
	Size of a house (ft²)	Number of rooms	Price of house ($)
	1000ft²	2	$20,000
	2000ft²	4	$30,000
	3000ft²	7	$50,000

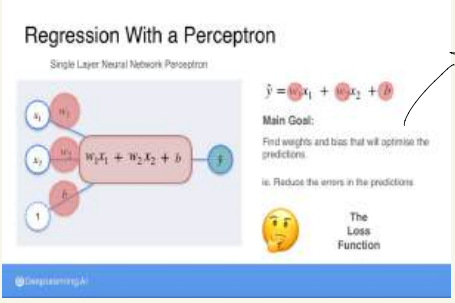
Now this becomes more complicated what if we had 10 features or 100.
This is where the perceptron comes in.



Now what is happening in the summation function each i/p is multiplied by a weight to determine how important it is to the output.

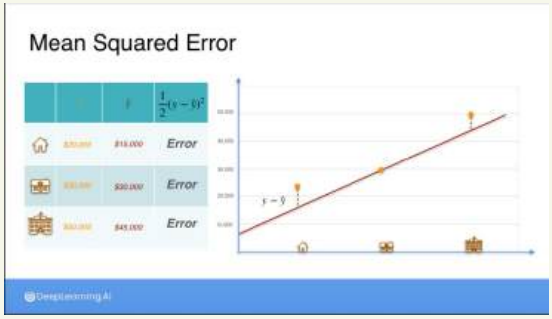
let's say we have w_1, w_2 & a bias term that gets added to this.





Therefore $\hat{y}$ is represented as a function of w_1, w_2, b

The loss function tells us how far we are from a prediction.



In this $(y-\hat{y})$ is the distance from accurate prediction. or the Error.

Now why do we take $(y-\hat{y})^2$

1) to quantify the big outliers
2) If the error could be +ve or -ve those numbers when summed can be zero or even small no.s which becomes harder to compute.

$1/2$ is simply to remove the 2 after Differentiation

$\hat{y} = w_1x_1 + w_2x_2 + b$, Loss function: $L(y,\hat{y}) = \frac{1}{2}(y-\hat{y})^2$

Main goal → Find w_1, w_2, b that gives $\hat{y}$ with the least error

To optimize w_1, w_2, b you need gradient descent

$$w_1 \to w_1 - \alpha\frac{\partial L}{\partial w_1} \qquad w_2 \to w_2 - \alpha\frac{\partial L}{\partial w_2} \qquad b = b - \alpha\frac{\partial L}{\partial b}$$

Some initial starting values

Prediction function: $\hat{y} = w_1x_1 + w_2x_2 + b$ Loss function: $L(y,\hat{y}) = \frac{1}{2}(y-\hat{y})^2$

$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b}$ > chain rule $\quad \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1}$ > chain rule $\quad \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$

$\frac{\partial L}{\partial \hat{y}} = \frac{1}{2} \times 2 \times (-1)\cdot(y-\hat{y}) = -(y-\hat{y})$ $\quad \frac{\partial \hat{y}}{\partial b} = 1$ $\quad \frac{\partial \hat{y}}{\partial w_1} = x_1$ $\quad \frac{\partial \hat{y}}{\partial w_2} = x_2$

$\frac{\partial L}{\partial b} = -(y-\hat{y})\cdot 1$ — (1) $\quad \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = -(y-\hat{y})x_1$ — (2) $\quad \frac{\partial L}{\partial w_2} = -(y-\hat{y})x_2$ — (3)

We use gradient descent

$w_1 = w_1 - \alpha\frac{\partial L}{\partial w_1} = w_1 - \alpha[-x_1(y-\hat{y})]$

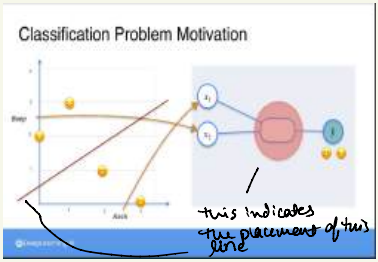
$w_2 \to w_2 - \alpha[-x_2(y-\hat{y})]$

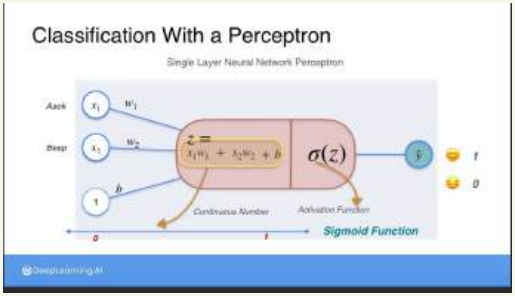
$b \to b - \alpha(-(y-\hat{y}))$

If we do this many times we get good values to minimize loss

Classification Problem Motivation

Dataset for the sentiment analysis. This can be plotted & shown as a perceptron →

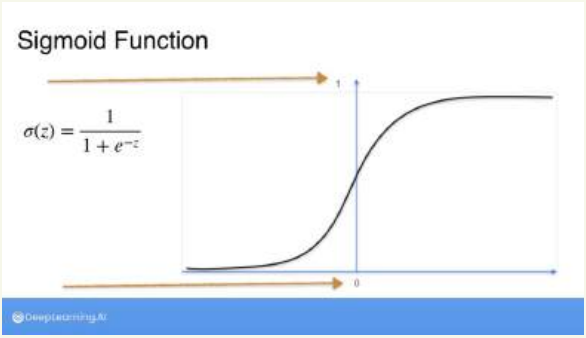




z is a continuous no. that can be anywhere on the number line. We want that in linear regression because we want any no. to be the value but in this we have happy (sad so we want the number to be 1 or 0 so we use a sigmoid function. to turn the entire no. line b/w 1 or 0.

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

If you look at the graph you will notice everything b/w 0 & 1 for y or o/p whereas the whole number line for input never really touches 1 or 0.



Derivative

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

$$\rightarrow \frac{\partial}{\partial z}\sigma(z) = -1(1+e^{-z})^{-1-1}\left(\frac{\partial}{\partial z}(1+e^{-z})\right)$$

$$\Rightarrow -1(1+e^{-z})^{-2}\left(\frac{\partial}{\partial z}(1) + \frac{\partial}{\partial z}(e^{-z})\right)$$

$$\Rightarrow -1(1+e^{-z})^{-2}\left(0 + e^{-z}\left(\frac{\partial}{\partial z}(-z)\right)\right)$$

$$\Rightarrow -1(1+e^{-z})^{-2}(e^{-z})(-1)$$

$$\frac{\partial}{\partial z}\sigma(z) = -1\,(1+e^{-z})^{-2}\cdot(e^{-z})(-1)$$

$$= (1+e^{-z})^{-2}(e^{-z}) = \frac{1}{(1+e^{-z})^2}\cdot(e^{-z})$$

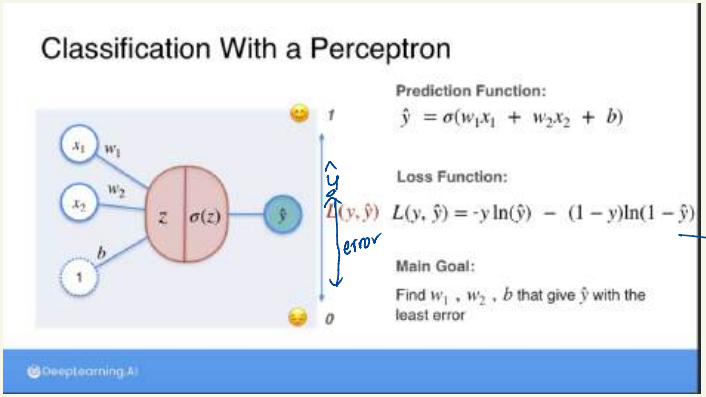
$$= \frac{e^{-z}}{(1+e^{-z})^2}$$

Add & subtract 1

$$\rightarrow \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2} \Rightarrow \frac{1}{1+e^{-z}}\left(1 - \frac{1}{(1+e^{-z})}\right)$$

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

- Since it's easy to calculate the differentiation by this it is imp. for ML.



For classification what works best is log loss.

$L(y, \hat{y})$, we use this to update our variables to bring loss down.

This is the loss function.

$L(y,\hat{y})$ is large if $y \& \hat{y}$ are far.

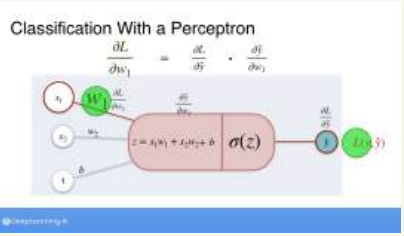
$L(y,\hat{y})$ small if $y \& \hat{y}$ are close.

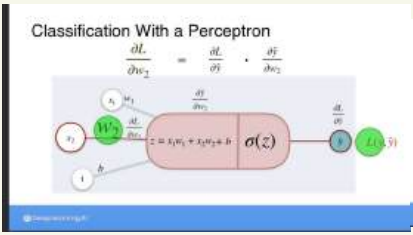
So now for optimal values of w_1, w_2, b we need gradient descent

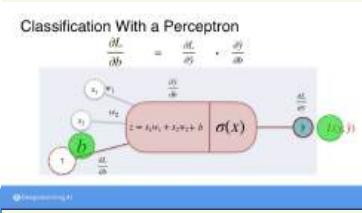
$$w_1 \rightarrow w_1 - \alpha\frac{\partial L}{\partial w_1} \qquad w_2 \rightarrow w_2 - \alpha\frac{\partial L}{\partial w_2} \qquad b \rightarrow b - \alpha\frac{\partial L}{\partial b}$$

So we need to find all the derivatives & perform gradient descent.

We need to

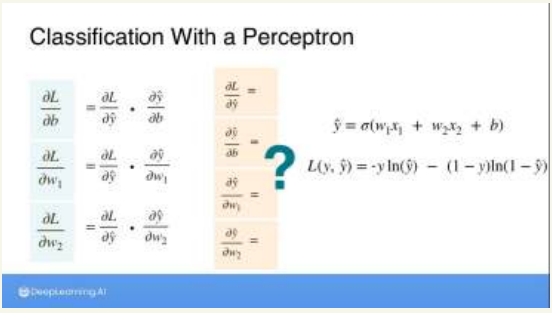


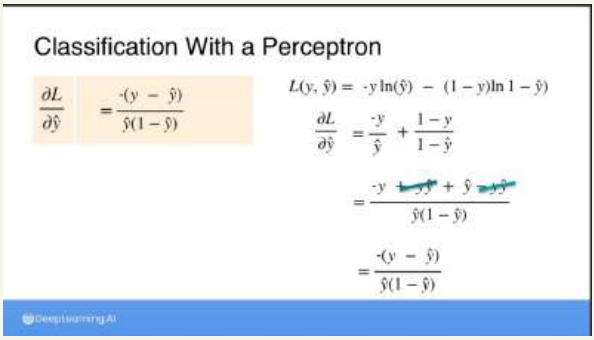




Classification witth a Perceptron – Key Concepts

- **Network Flow**
  - Inputs: x_1, x_2
  - Linear combin z ꜛ$w_1 + b$
  - Activation: $y = \sigma(z) = \frac{1}{1}$ $(1-y)\,g\,(1-y)$
  - Output: $\hat{y} \in (0,1)$
- **Loss Function**
  - Used for binarfication: $L\,(y,\, y_1) = -[y \log\,(\hat{y})) + (1)$
  - Compares predicted probability $\hat{y}$ to actual label y
- **Why Log Loss Comes After Sigmoid**
  - Sigmoid gives the **probability**
  - Log loss evaluates how close $\hat{y}$is to the true label
  - The model's goal: minimize this loss
- **Backpropagation (Gradient Flow)**
  - Chain rule: $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \,.\, \hat{a}$
  - This helps compute how much t to adjust w bias to reduce loss





Then we calculate.

$\frac{\partial \hat{y}}{\partial w_1} = \hat{y}\,(1-\hat{y})\,x_1$ - due to derivative of sigmoid → $\sigma(1-\sigma)$

$\frac{\partial \hat{y}}{\partial w_2} = \hat{y}\,(1-\hat{y})\,x_2$

$\frac{\partial \hat{y}}{\partial b} = \hat{y}\,(1-\hat{y})$ that's where the $\hat{y}(1-\hat{y})$ come in

Classification With a Perceptron

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})} \hat{y}(1-\hat{y})$$

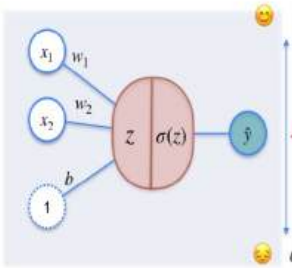
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})} \hat{y}(1-\hat{y}) x_1$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})} \hat{y}(1-\hat{y}) x_2$$

$$\frac{\partial L}{\partial b} = -(y-\hat{y}) \quad \frac{\partial L}{\partial w_1} = -(y-\hat{y}) x_1 \quad \frac{\partial L}{\partial w_2} = -(y-\hat{y}) x_2$$

So for a good prediction $(y-\hat{y})$ minimal.

Classification With a Perceptron



To find optimal values for:

w_1, w_2, b

You need gradient descent

$$w_1 \rightarrow w_1 - \alpha \frac{\partial L}{\partial w_1} \qquad w_2 \rightarrow w_2 - \alpha \frac{\partial L}{\partial w_2} \qquad b \rightarrow b - \alpha \frac{\partial L}{\partial b}$$

$w_1 \rightarrow w_1 - \alpha(-x_1(y-\hat{y}))$ $\quad b \rightarrow b - \alpha(-(y-\hat{y}))$

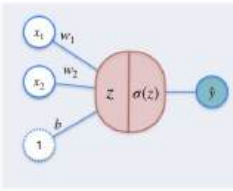
$w_2 \rightarrow w_2 - \alpha(-x_2(y-\hat{y}))$

If you do this many times you get a good model.

Now we saw a perceptron, a neural network is a network of perceptrons in many layers.

Classification Problem Motivation

Sentence	Aack	Beep	Mood
Aack aack aack!	3	0	Happy
Beep beep!	0	2	Sad
Aack beep beep beep!	1	3	Sad
Aack beep aack!	2	1	Happy



$$z = x_1 w_1 + x_2 w_2 + b$$

DeepLearning.AI

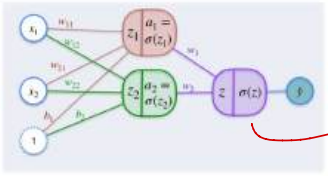
- This builds a linear boundary. However it sometimes is more complicated than just 1 line.

then we use multiple perceptrons.

2,2,1 Neural Network

Neural network of depth 2

- one input layer
- one hidden layer
- one output layer



DeepLearning.AI

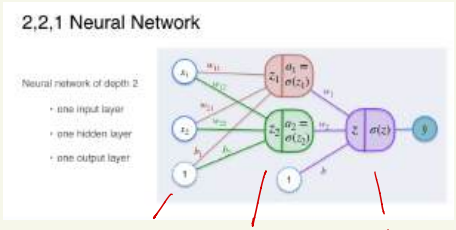
here $z_1 = w_{11} x_1 + w_{21} x_2 + b_1$

$z_2 = w_{12} x_1 + w_{22} x_2 + b_2$

$a_1 = \sigma(z_1) \quad a_2 = \sigma(z_2)$

then $z = a_1 w_1 + a_2 w_2$

$\sigma(z) \Rightarrow \sigma(z)$



We add a bias to output layer

i/p hidden o/p

2,2,1 Neural Network

$$a_1 = \sigma(z_1)$$

$$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

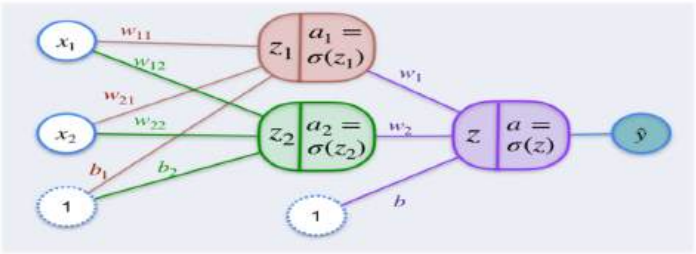
$$a_2 = \sigma(z_2)$$

$$z_2 = x_1 w_{12} + x_2 w_{22} + b_2$$

$$\hat{y} = \sigma(z)$$

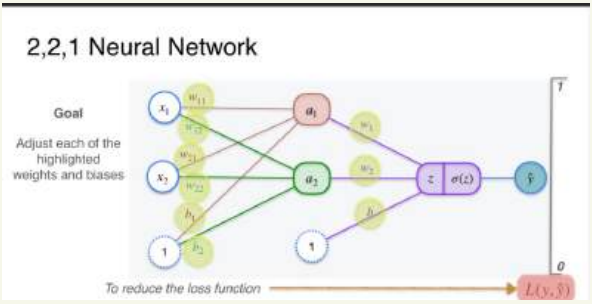
$$z = a_1 w_1 + a_2 w_2 + b$$

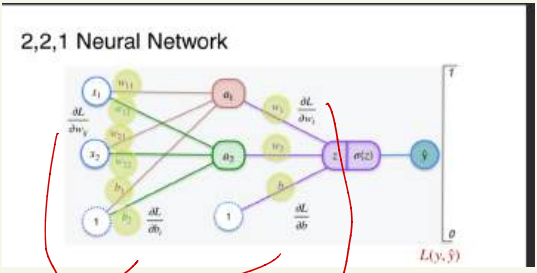
$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$



↳ This is the code in which a NN works.

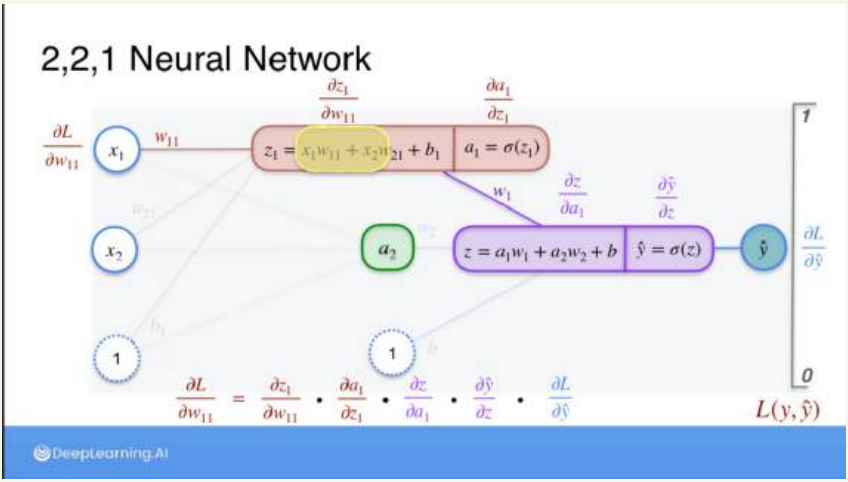
Let's train them now:





These

Partial derivatives tell us exactly in what direction to move each w & b to reduce the log loss function



$\frac{\partial L}{\partial \hat{y}}$ depends on $\frac{\partial \hat{y}}{\partial z}$ which depends on $\frac{\partial z}{\partial a_1}$ or $\frac{\partial a_1}{\partial z_1}$ or $\frac{\partial z_1}{\partial w_{11}}$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial z_1}{\partial w_{11}} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z}{\partial a_1} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\Rightarrow x_1 \cdot a_1(1-a_1) \cdot w_1 \cdot \hat{y}(1-\hat{y}) \cdot \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})}$$

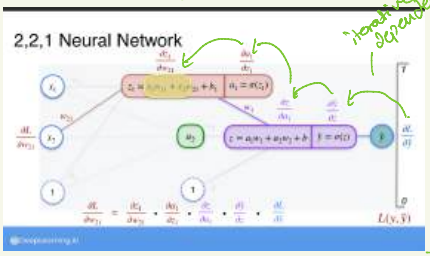
$$\Rightarrow -x_1 w_1 a_1 (1-a_1)(y-\hat{y})$$

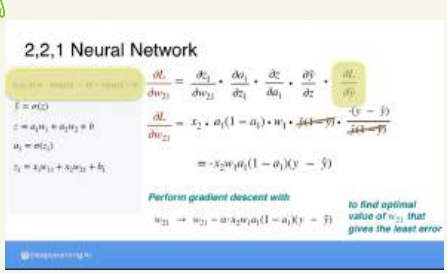
$L(y, \hat{y}) = -y \log(\hat{y}) \cdot (1-y)(\log(1-\hat{y}))$
$\hat{y} = \sigma(z)$
$z = a_1 w_1 + a_2 w_2 + b$
$a_1 = \sigma(z_1)$
$z_1 = x_1 w_{11} + x_2 w_{21} + b_1$

Now perform gradient descent.

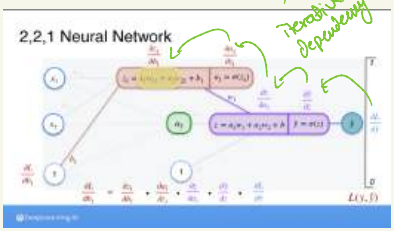
$$w_{11} \to w_{11} - \alpha \cdot x_1 w_1 a_1 (1-a_1)(y-\hat{y})$$

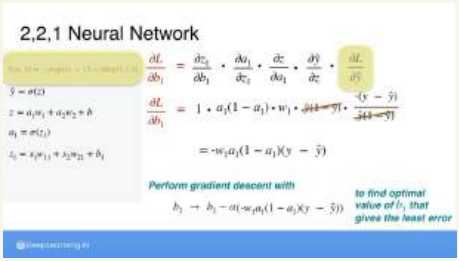
for w_{12}



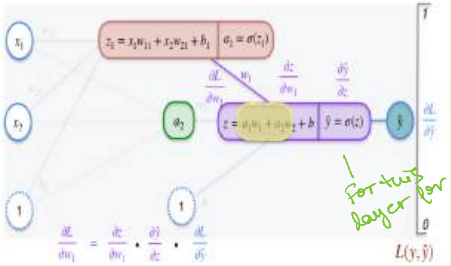


For b_1





2,2,1 Neural Network



For this layer for w_1

$$\frac{\partial L}{\partial w_1} = \frac{\partial z}{\partial w_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

For b

2,2,1 Neural Network

$$\hat{y} = \sigma(z)$$

$$z = a_1w_1 + a_2w_2 + b$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial z}{\partial w_1} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_1} = a_1 \cdot \hat{y}(1-\hat{y}) \cdot \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}$$

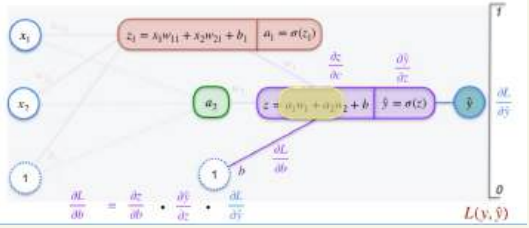
$$= -a_1(y - \hat{y})$$

Perform gradient descent with

$$w_1 \rightarrow w_1 - \alpha(-a_1(y - \hat{y}))$$

to find optimal value of w_1 that gives the least error

2,2,1 Neural Network



2,2,1 Neural Network

$$\hat{y} = \sigma(z)$$

$$z = a_1w_1 + a_2w_2 + b$$

$$\frac{\partial L}{\partial b} = \frac{\partial z}{\partial b} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}}$$

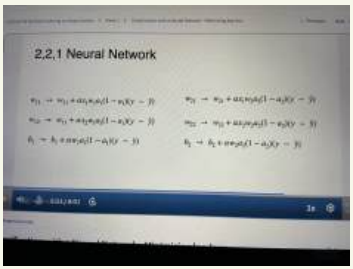
$$\frac{\partial L}{\partial b} = 1 \cdot \hat{y}(1-\hat{y}) \cdot \frac{-(y - \hat{y})}{\hat{y}(1-\hat{y})}$$

$$= -(y - \hat{y})$$

Perform gradient descent with

$$b \rightarrow b - \alpha(-(y - \hat{y}))$$

to find optimal value of b that gives the least error



→ These are the formulas

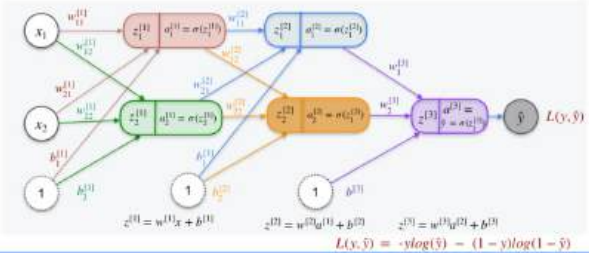
2,2,1 Neural Network

$$w_1 \rightarrow w_1 + \alpha a_1(y - \hat{y}))$$

$$w_2 \rightarrow w_2 + \alpha a_2(y - \hat{y}))$$

$$b \rightarrow b + \alpha(y - \hat{y}))$$

Back Propagation Introduction



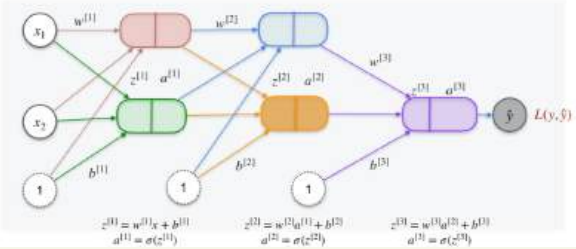
DeepLearning.AI

From this we are able to derive

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

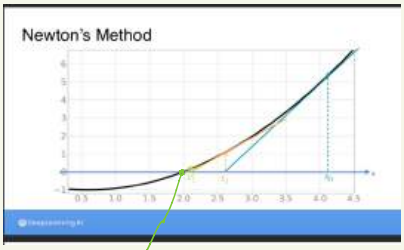
And we try to minimize it

Back Propagation Introduction



Newton's Method

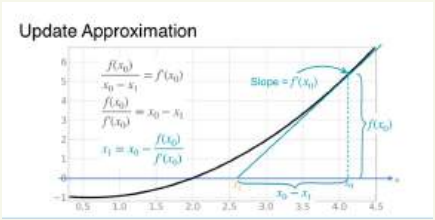
In principle newton's method is used to find zeroes of a function



goal is to find where function = 0 $f(x)=0$

we take x_0 randomly take the derivative & draw the tangent see that the place it intersects the x-axis at x_1 which is closer to the zero of function.
we do something at x_1 & the slope intersects now at x_2 which is very close.

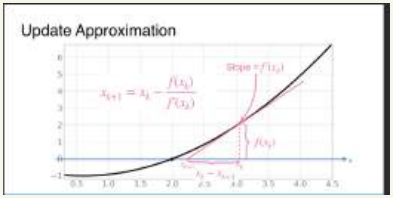
Found 0 at x_2



here we can see that

$$\text{Slope} = f'(x_0) = \frac{\text{rise}}{\text{run}} = \frac{f(x_0)}{x_0 - x_1}$$

$$\frac{f(x_0)}{f'(x_0)} = x_0 - x_1 \qquad x_1 = x_0 - \frac{f(x_0)}{f'(x_1)}$$



$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Newton's Method for Optimization

Newton's method

Goal: find a zero of $f(x)$

1) Start with some x_0

2) Update:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

3) Repeat 2) until you find the root.

NM for Optimization

Goal: minimize $g(x)$ ⇒ find zeros of $g'(x)$

$f(x) \mapsto g'(x)$ $\quad$ $f'(x) \mapsto (g'(x))'$

1) Start with some x_0

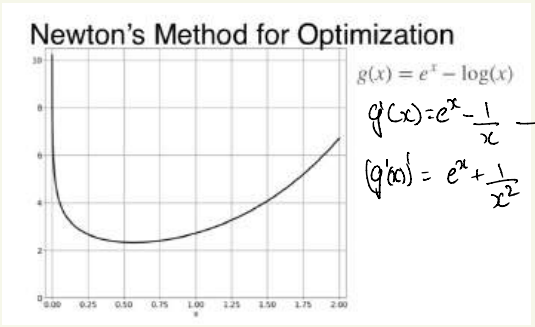
2) Update:

$$x_{k+1} = x_k - \frac{g'(x_k)}{(g'(x_k))'}$$

3) Repeat 2) until you find the root.

DeepLearning.AI

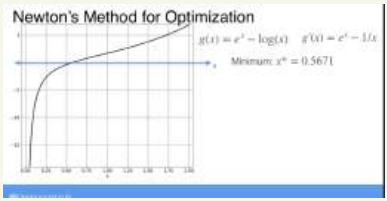
Newton's Method for Optimisation



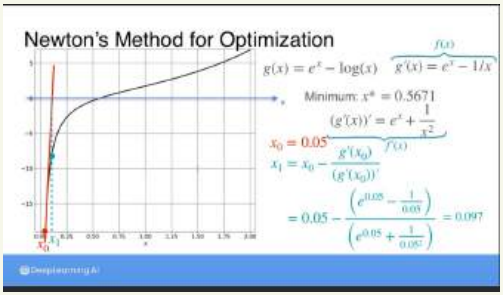
→ Now to find minimum using NM we need to use it to find the zeroes of the derivative.

$g(x) = e^x - \frac{1}{x}$ → $f(x) \to g'(x)$

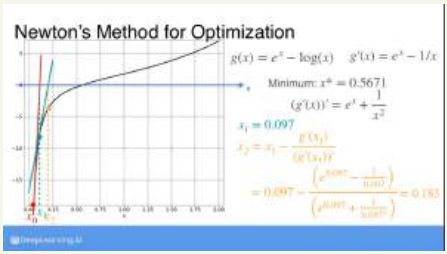
$(g'(x))' = e^x + \frac{1}{x^2}$ → $f'(x) \to (g'(x))'$

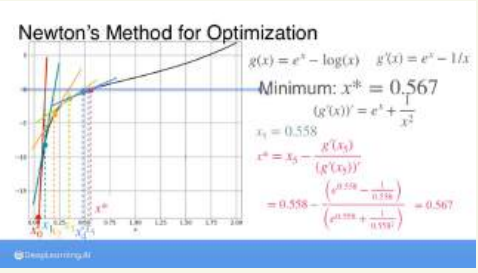


This is the graph of the derivative. we need to find it's zero.



→ we iterate again with x_1 we get x_2





On the 6th iteration we reached the minimum value which is 0.567

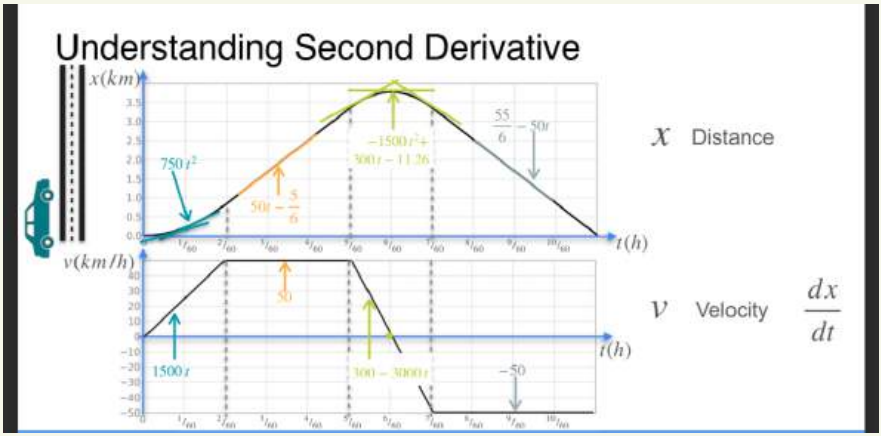
Second Derivative → $x_{k+1} = x_k - \frac{g'(x_k)}{(g'(x_k))'}$ ← second derivative

Leibniz notation: $\frac{d^2 f(x)}{dx^2} = \frac{d}{dx}\left(\frac{df(x)}{dx}\right)$ Lagrange notation: $f''(x)$

Understanding Second derivative

1st derivative → velocity for x distance → $\frac{dx}{dt}$ → rate of change of distance w.r.t time

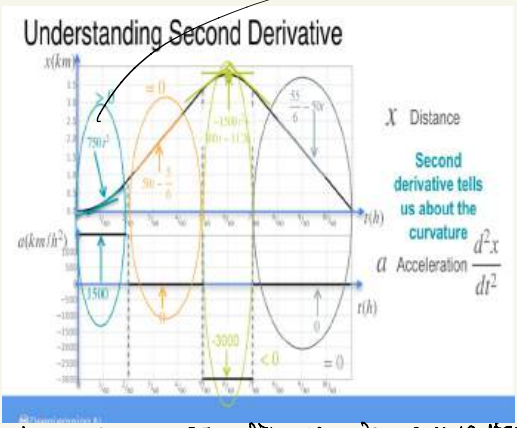
and derivative → acceleration for time t → $\frac{dv}{dt} = \frac{d^2x}{dt^2}$ ← acceleration +ve → gives increasing velocity
- -ve → decreasing velocity
- 0 → constant



comparison b/w distance velocity graph.

constant increase in vel so acceleration > 0

at constant velocity.



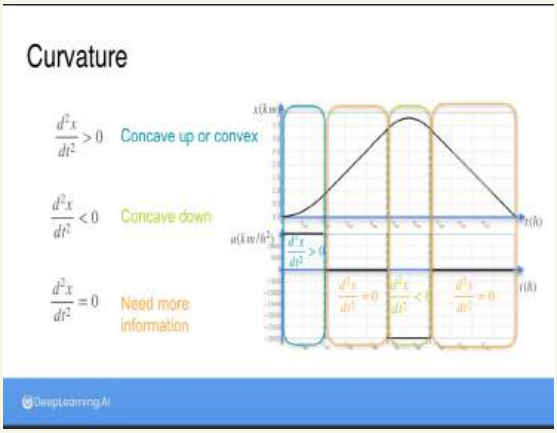
→ Notice in the first 2 mins when the distance increases at an increasing rate +ve acceleration

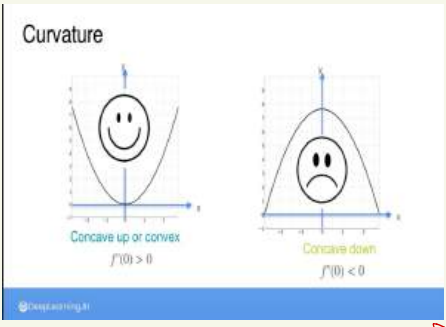
2–5 constant distance covered so acceleration = 0

5–7 the speed or distance slows down & decreases. So acceleration < 0.

again from 7–10 → the distance decreases constantly so 0 acceleration

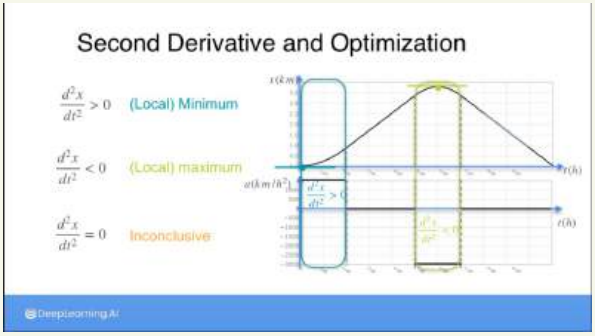
The second derivative gives a measure of the amount the curve deviates from being a straight line.

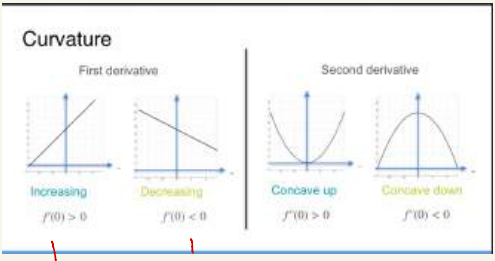




∪ = smile concave up

∩ = frown concave down





increasing function

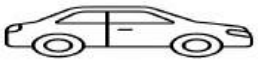
decreasing function

Second Derivative in Optimization – Summary for ML

- **First Derivative** $f'(x)$
  - Tells the slope (increasing or decreasing)
  - Used to find stationary points (where slope = 0)
- **Second Derivative** $f''(x)$
  - Helps classify those stationary points:

Condition	Interpretation
$f'(x) = 0,\ f''(x) > 0$	Local Minimum 😊 (concave up)
$f'(x) = 0,\ f''(x) < 0$	Local Maximum 😞 (concave down)
$f''(x) = 0$	Inconclusive – check more

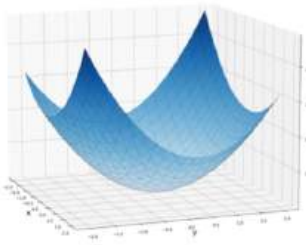
- **Analogy (Car Example)**
  - Position $f(x)$
  - Velocity $f'(x)$
  - Acceleration $f''(x)$



  - Positive acceleration = speeding up (curve bends up)
  - Negative acceleration = slowing down (curve bends down)

Second derivative in multiple variables

Second Derivative



$$f(x,y) = 2x^2 + 3y^2 - xy$$

Differentiate w.r.t x

- x → $4x - y$
  - x → 4 — Rate of change of $f'_x(x,y)$ w.r.t x
  - y → -1 — Rate of change of $f'_x(x,y)$ w.r.t y
- y → $6y - x$
  - x → -1 — Rate of change of $f'_y(x,y)$ w.r.t x
  - y → 6 — Rate of change of $f'_y(x,y)$ w.r.t y

Same

w.r.t y

DeepLearning.AI

What Do These Mean?

The first two w.r.t x, y w.r.t y

Rate of change of $f_x(x,y)$ w.r.t x
Rate of change of $f_y(x,y)$ w.r.t y

Change in the change in the function w.r.t tiny changes in x and y

Same idea as with one variable!

Rate of change of $f_x(x,y)$ w.r.t y
Rate of change of $f_y(x,y)$ w.r.t x

1. Change in the slope along one coordinate axis w.r.t tiny changes along an orthogonal coordinate axis

DeepLearning.AI

What Do These Mean?

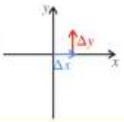
Rate of change of $f_x(x,y)$ w.r.t x
Rate of change of $f_y(x,y)$ w.r.t y

Change in the change in the function w.r.t tiny changes in x and y

Same idea as with one variable!

Rate of change of $f_x(x,y)$ w.r.t y
Rate of change of $f_y(x,y)$ w.r.t x

1. Change in the slope along one coordinate axis w.r.t tiny changes along an orthogonal coordinate axis



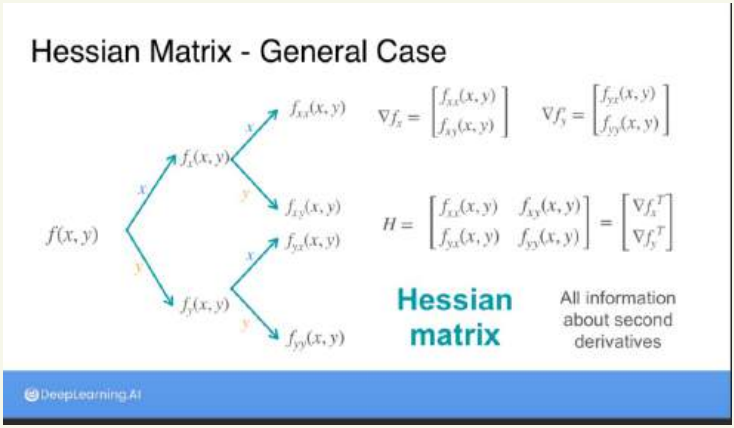
2. They are the same!

(In most cases)

Notation

	Leibniz's notation	Lagrange's notation
Rate of change of $f_x(x,y)$ w.r.t x	$\frac{\partial^2 f}{\partial x^2}$	$f_{xx}(x,y)$
Rate of change of $f'_y(x,y)$ w.r.t y	$\frac{\partial^2 f}{\partial y^2}$	$f_{yy}(x,y)$
Rate of change of $f'_x(x,y)$ w.r.t y	$\frac{\partial^2 f}{\partial x \partial y}$	$f_{xy}(x,y)$
Rate of change of $f'_y(x,y)$ w.r.t x	$\frac{\partial^2 f}{\partial y \partial x}$	$f_{yx}(x,y)$

DeepLearning.AI

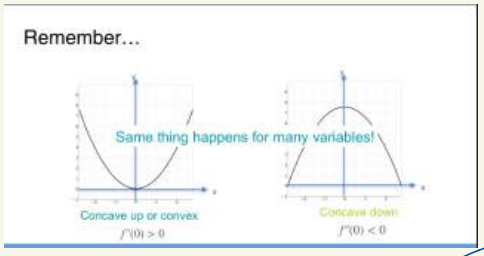


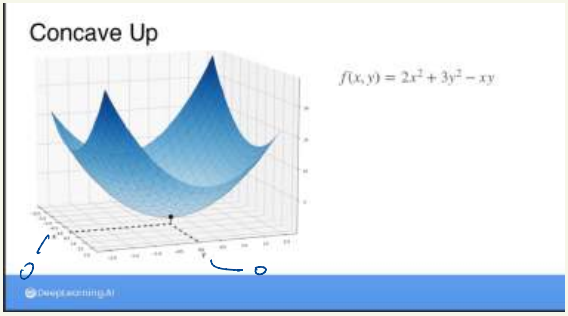
Question : $f(x,y) = x^2 y^2$ Hessian matrix = ?

$$\frac{\partial f}{\partial x} = 2x \qquad \frac{\partial f}{\partial y} = 2y$$

$$\frac{\partial^2 f}{\partial x^2} = 2 \qquad \frac{\partial^2 f}{\partial x \partial y} = 0 \qquad \frac{\partial^2 f}{\partial y^2} = 2 \qquad \frac{\partial^2 f}{\partial y \partial x} = 0$$

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$





$F(x,y) = 2x^2 + 3y^2 - xy$

$H(0,0) = \begin{bmatrix} 4 & -1 \\ -1 & 6 \end{bmatrix}$

For this now do we check if the function is +ve or negative we have a matrix. For that we use eigen values

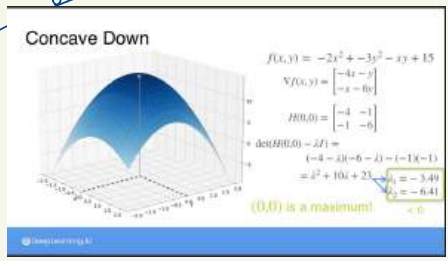
$\det(H(0,0) - \lambda I) = \det \begin{bmatrix} 4-\lambda & -1 \\ -1 & 6-\lambda \end{bmatrix}$

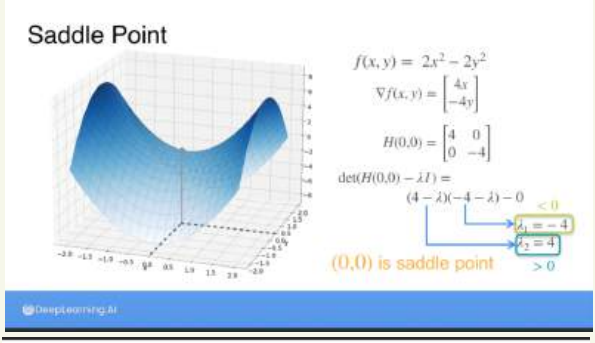
$\rightarrow (4-\lambda)(6-\lambda) - (-1)(-1)$

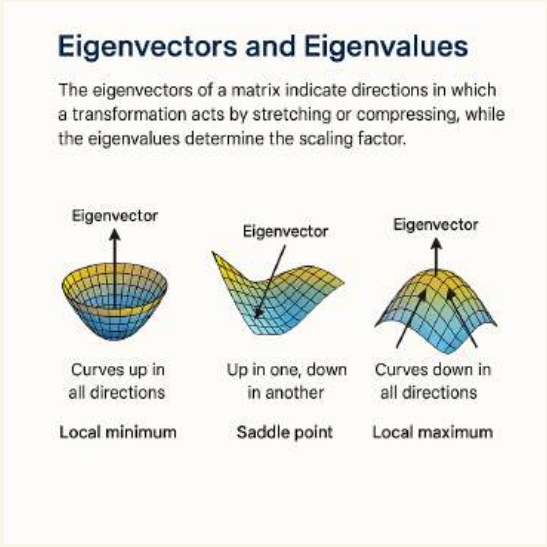
$= \lambda^2 + 10\lambda + 23 \quad \lambda_1 = 6.41$

$\lambda_2 = 3.59$

$(0,0)$ is min as these are the functions with concave up.







Summary

	1 variable $f(x)$	2 variables $f(x, y)$	More variables $f(x_1, x_2, \ldots, x_n)$
(Local) minima	Happy face $f''(x) > 0$	Upper paraboloid $\lambda_1 > 0$ & $\lambda_2 > 0$	All $\lambda_i > 0$
(Local) maxima	Sad face $f''(x) < 0$	Down paraboloid $\lambda_1 < 0$ & $\lambda_2 < 0$	All $\lambda_i < 0$
Need more information	$f''(x) = 0$	Saddle point $\lambda_1 > 0$ & $\lambda_2 < 0$ $\lambda_1 < 0$ & $\lambda_2 > 0$ Or some $\lambda_i = 0$	Some $\lambda_i > 0$ and some $\lambda_j < 0$ OR At least one $\lambda_i = 0$

DeepLearning.AI

Newton's Method

1 variable $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$ $\quad x_{k+1} = x_k - f''(x_k)^{-1} f'(x_k)$

2 variables $\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \underbrace{H^{-1}(x_k, y_k)}_{2\times 2} \cdot \underbrace{\nabla f(x_k, y_k)}_{2\times 1}$

e.g. $f(x,y) = x^4 + 0.8y^4 + 4x^2 + 2y^2 - xy - 0.2x^2y$

$\frac{\partial f}{\partial x} = 4x^3 + 8x - y - 0.4xy$ $\quad \frac{\partial f}{\partial y} = 3.2y^3 + 4y - x - 0.2x^2$

gradient $(\nabla f(x,y))$

$\frac{\partial^2 f}{\partial x^2} = 12x^2 + 8 - 0.4y$ $\quad \frac{\partial^2 f}{\partial x \partial y} = -1 - 0.4x$

$-1 - 0.4x$ $\quad 9.6y^2 + 4$

H

Now let's start at some point $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$

$\nabla f(4,4) = \begin{bmatrix} 277.6 \\ 213.6 \end{bmatrix}$ $\quad H(4,4) = \begin{bmatrix} 198.4 & -2.6 \\ -2.6 & 157.6 \end{bmatrix}$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} - \begin{bmatrix} 198.4 & -2.6 \\ -2.6 & 157.6 \end{bmatrix}^{-1} \begin{bmatrix} 277.6 \\ 213.6 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 2.58 \\ 2.62 \end{bmatrix} \rightarrow \text{closer to } 0,0.$$

Now 2nd iteration $\rightarrow \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 2.58 \\ 2.62 \end{bmatrix}$.

$\nabla f(2.58, 2.61) = \begin{bmatrix} 84.25 \\ 63.4 \end{bmatrix}$ $\quad H(2.58, 2.61) = \begin{bmatrix} 86.83 & -2.032 \\ -2.032 & 69.39 \end{bmatrix}$, $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 2.58 \\ 2.61 \end{bmatrix} - \begin{bmatrix} 86.83 & -2.032 \\ -2.032 & 69.39 \end{bmatrix}^{-1} \begin{bmatrix} 84.25 \\ 63.4 \end{bmatrix}$

$= \begin{bmatrix} 1.59 \\ 1.67 \end{bmatrix}$, repeat until you reach zero