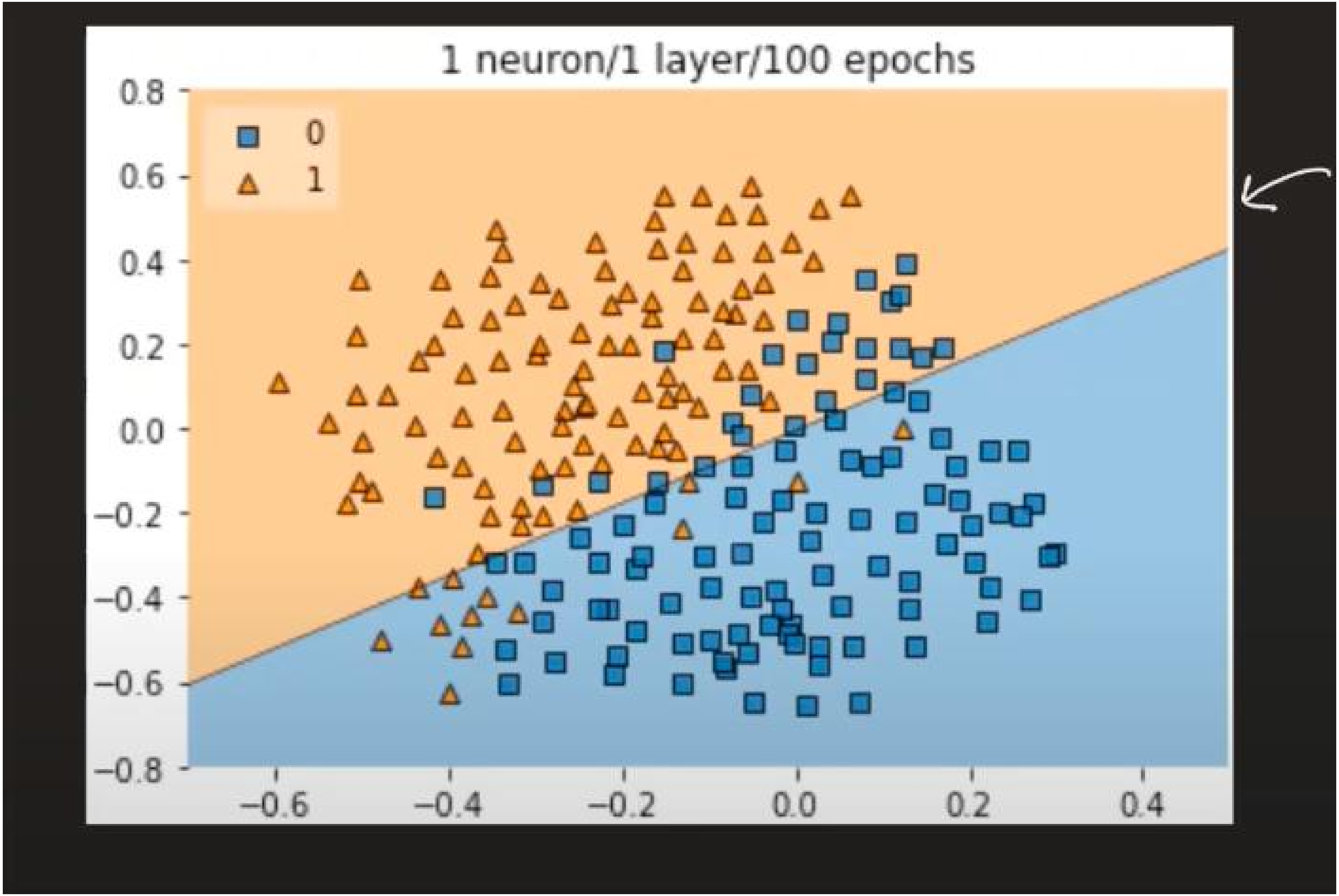
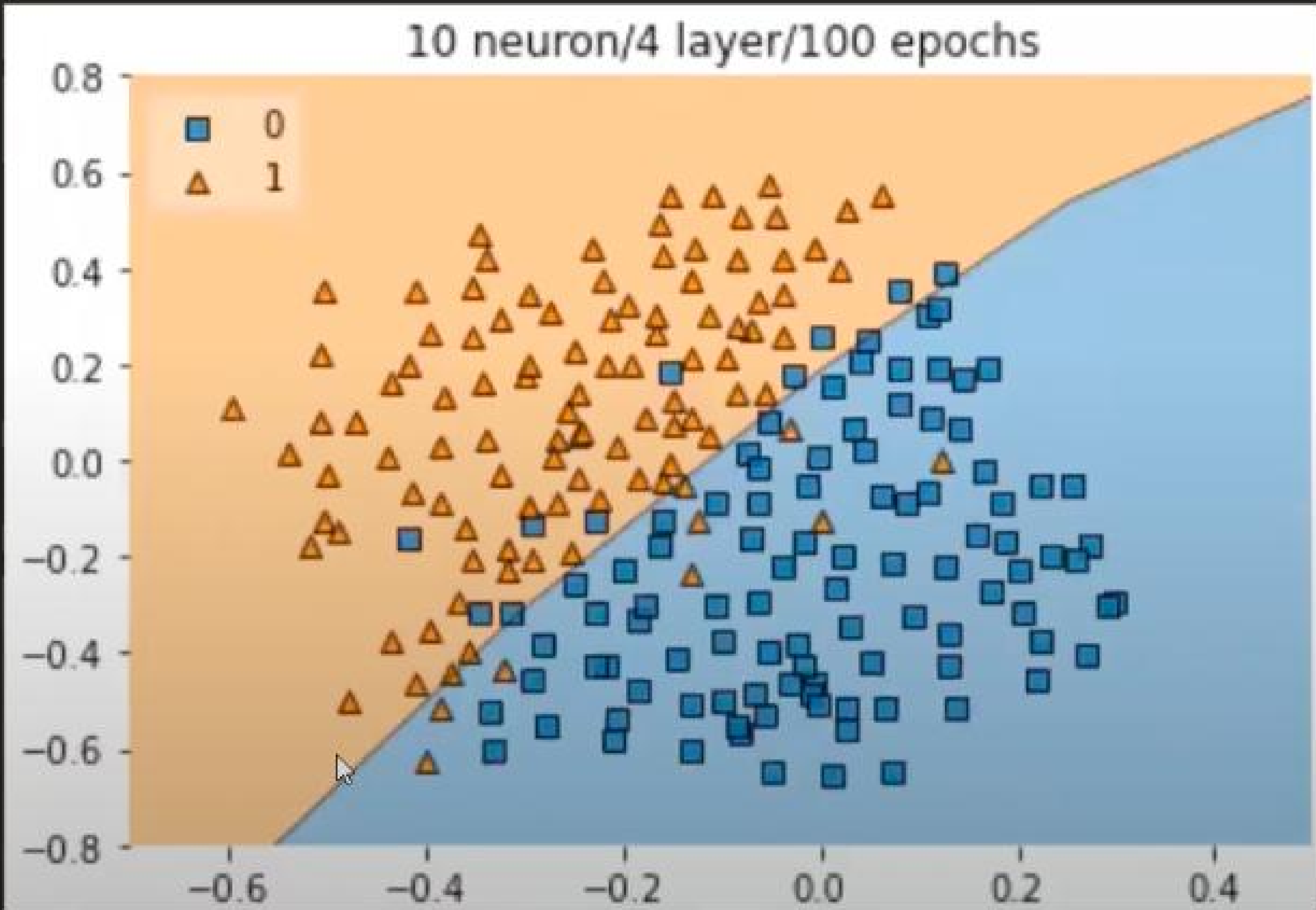
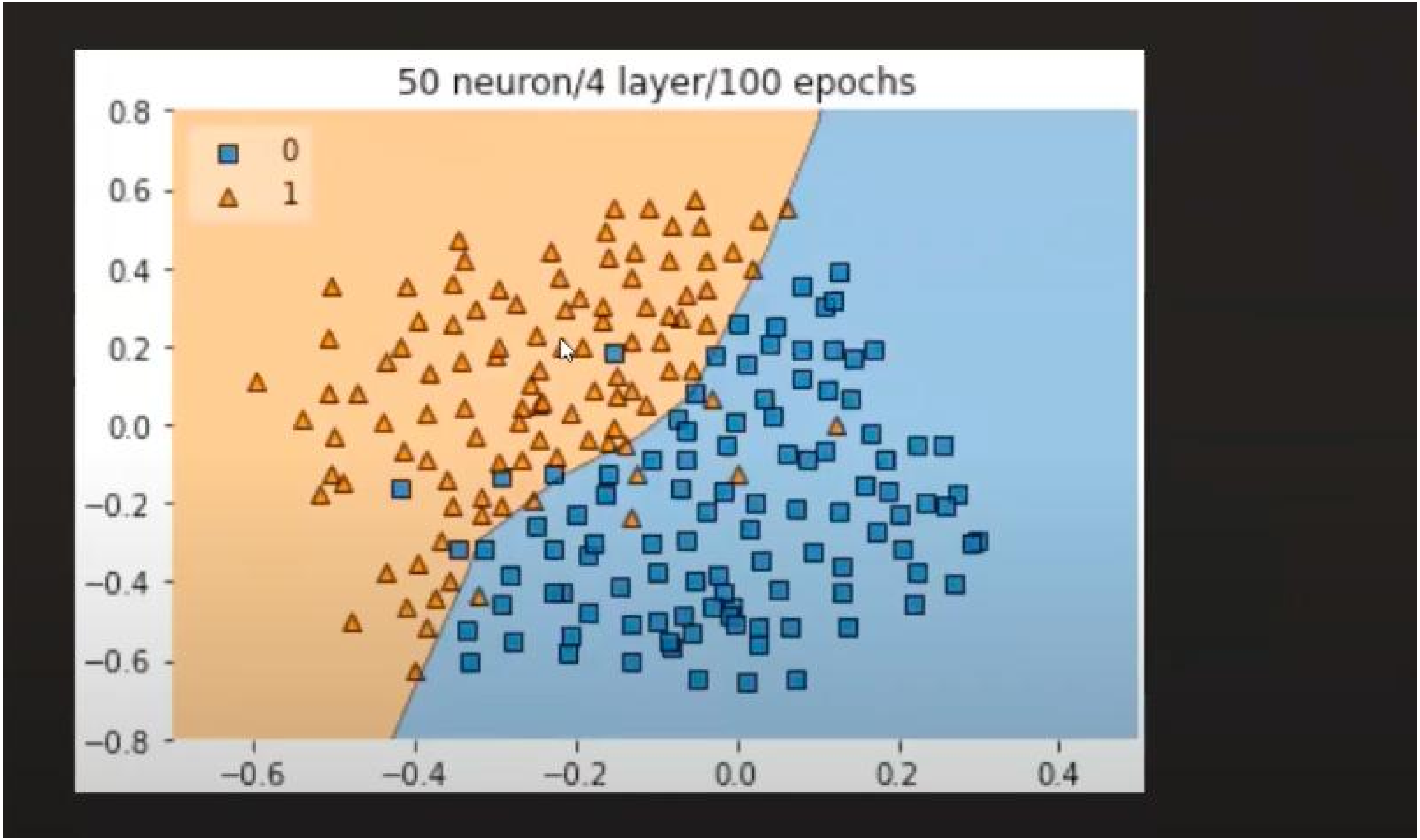
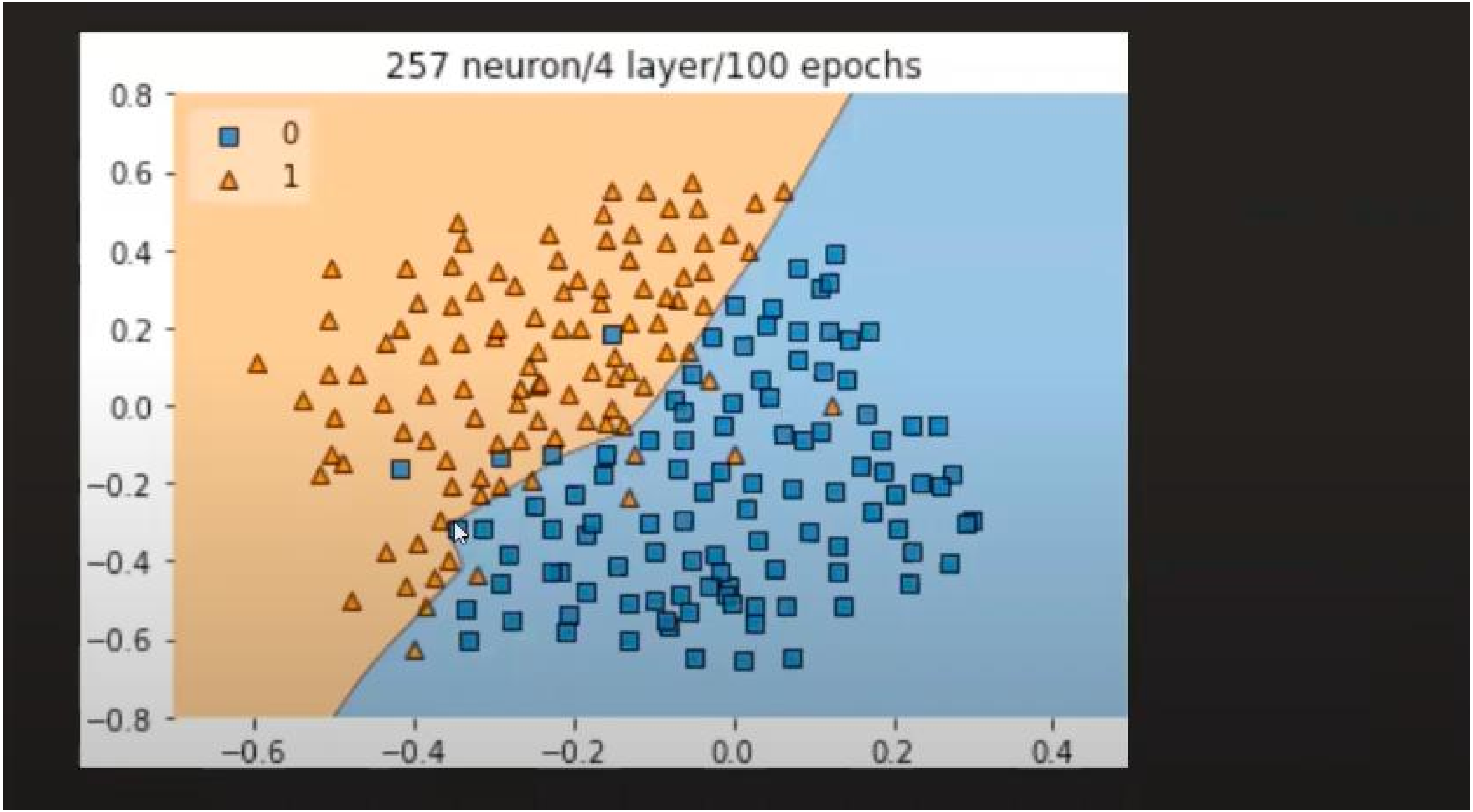


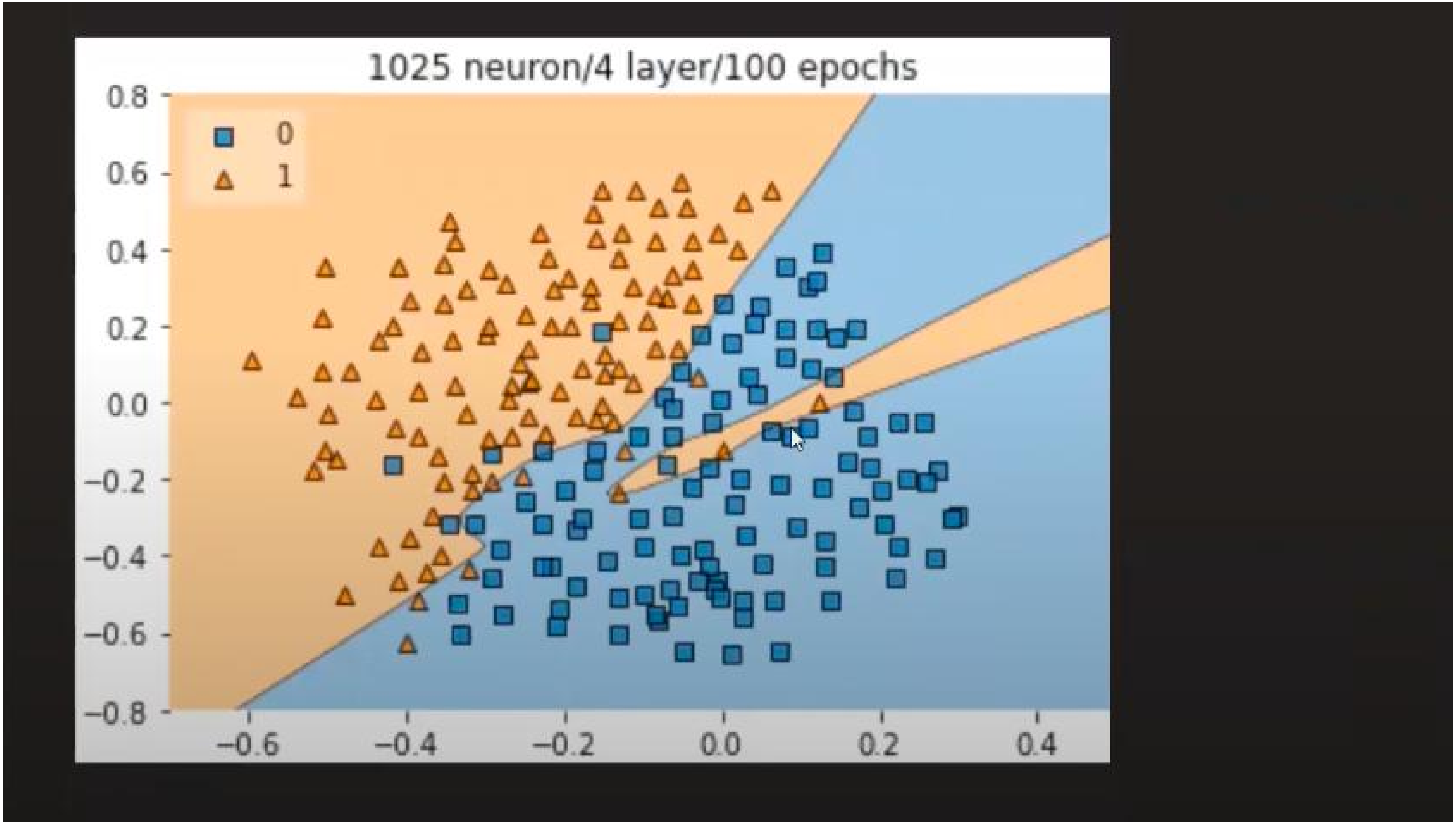
1 neuron/1 layer/100 epochs
0
1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4



10 neuron/4 layer/100 epochs
0
1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4


50 neuron/4 layer/100 epochs
0
1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4


257 neuron/4 layer/100 epochs
0
1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4

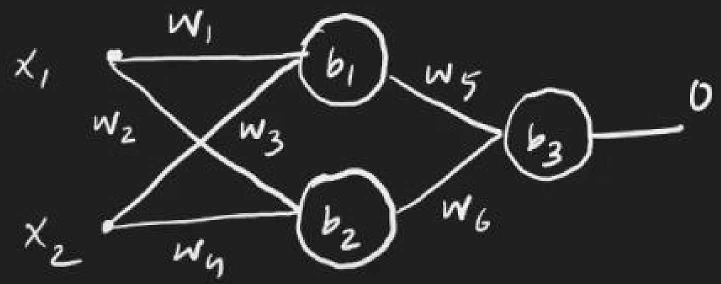

1025 neuron/4 layer/100 epochs
0
1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4

What are Activation Functions?

31 May 2022 14:49

In artificial neural networks, each neuron forms a weighted sum of its inputs and passes the resulting scalar value through a function referred to as an activation function or transfer function. If a neuron has n inputs then the output or activation of a neuron is

$$a = g(w_1x_1 + w_2x_2 + w_3x_3 + \dots w_nx_n + b)$$

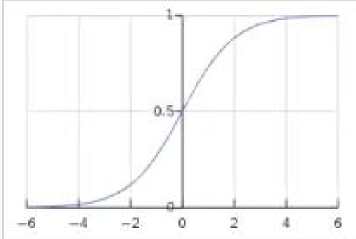


This function g is referred to as the activation function.

Activation function

From Wikipedia, the free encyclopedia

For the formalism used to approximate the influence of an extracellular electrical field on neurons, see activating function. For a linear system's transfer function, see transfer function.



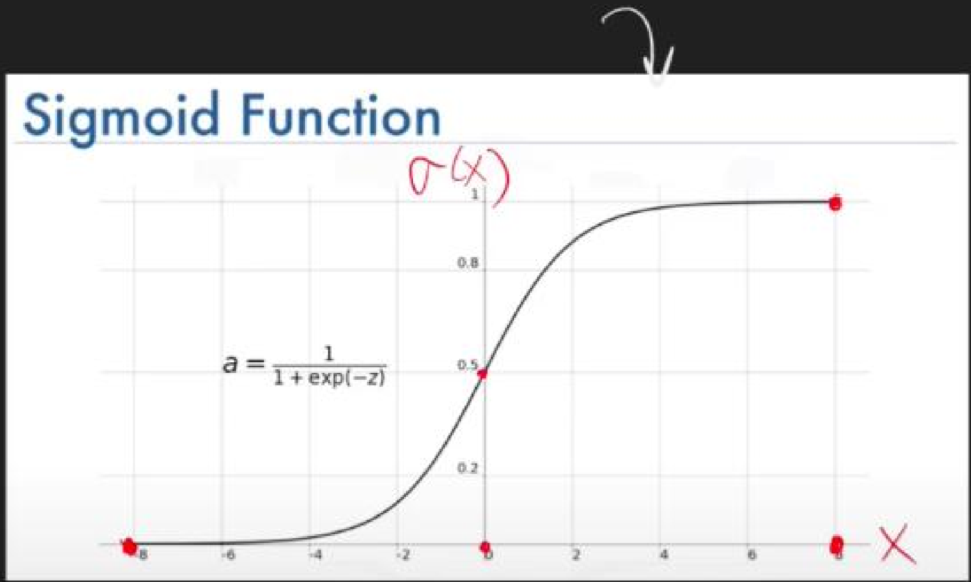
Logistic activation function

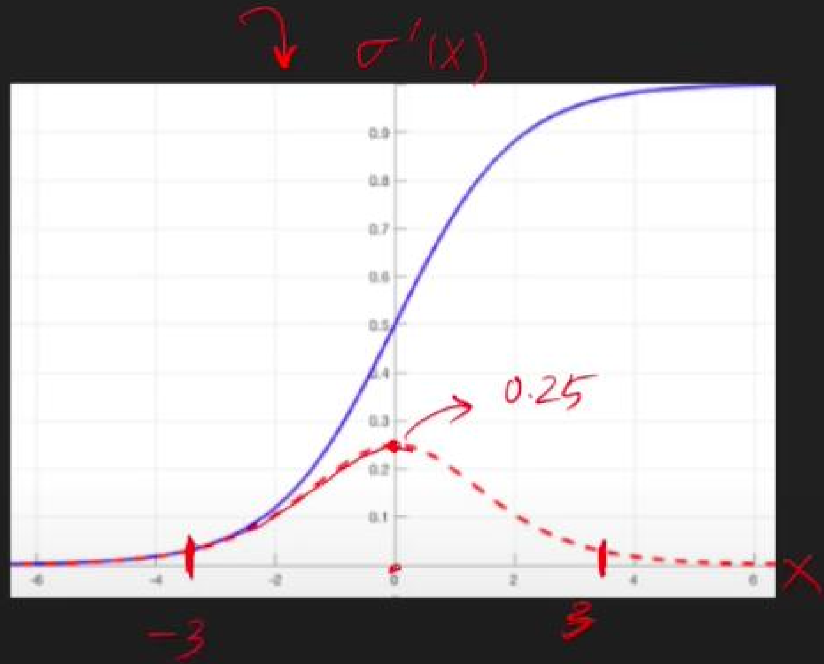
In artificial neural networks, each neuron forms a weighted sum of its inputs and passes the resulting scalar value through a function referred to as an activation function or transfer function. If a neuron has n inputs $x_1, x_2, \ldots x_n$ then the output or activation of a neuron is $a = g(w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots w_n x_n + b)$. This function g is referred to as the activation function. If the function g is taken as the linear function $g(z) = z$ then the neuron performs linear regression or classification. In general g is taken to be a nonlinear function to do nonlinear regression and solve classification problems that are not linearly separable. When g is taken to be a sigmoidal or 's' shaped function varying from 0 to 1 or -1 to 1, the output value of the neuron can be interpreted as a YES/NO answer or binary decision. However saturating activation function can cause the vanishing gradient problem in deep networks. Replacing saturating sigmoidal activation functions with activation functions like ReLU that have larger derivative values allowed deeper networks to be trained for the first time. Non-monotonic and oscillating activation functions that significantly outperform ReLU have since been found.[1] In particular oscillating activation functions improve gradient flow, speedup training and allow single neurons to learn the XOR function like certain human cerebral neurons.[2][3]

Contents [hide]

Sigmoid Activation Function

31 May 2022 14:50

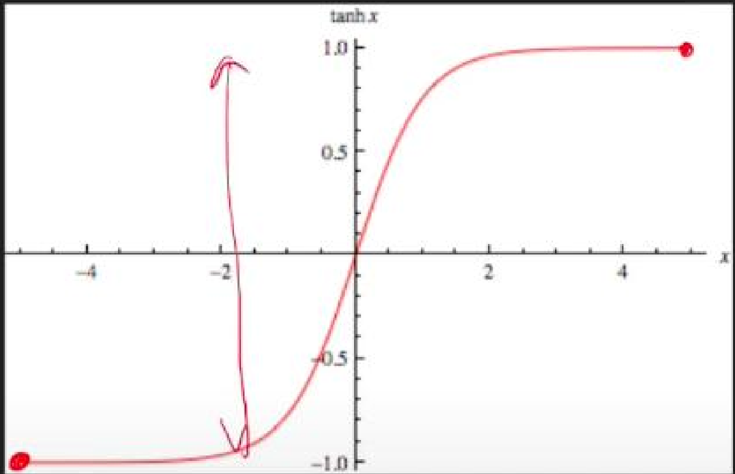


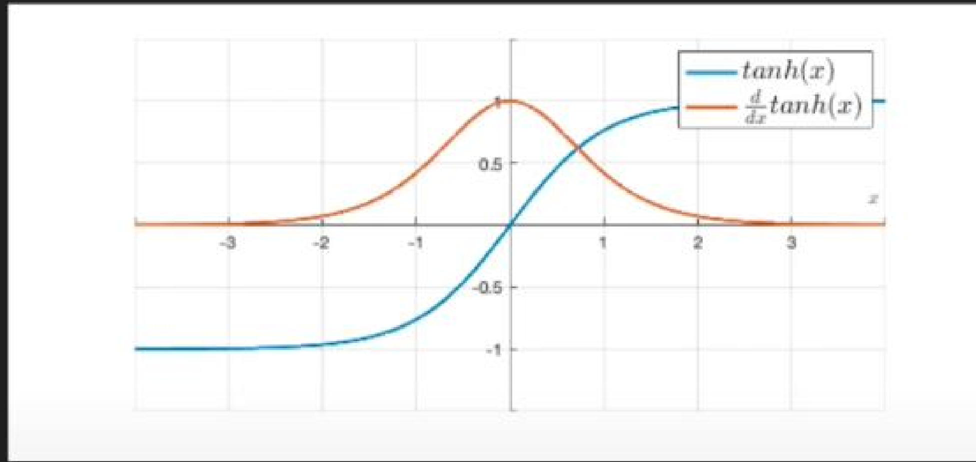


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Tanh Activation Function

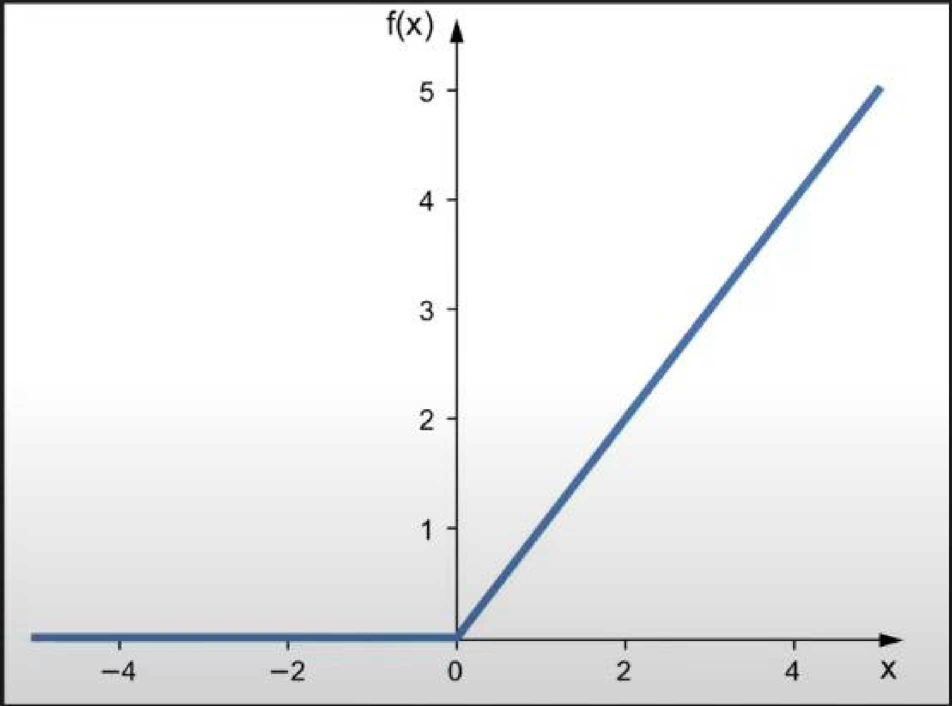
31 May 2022 14:50





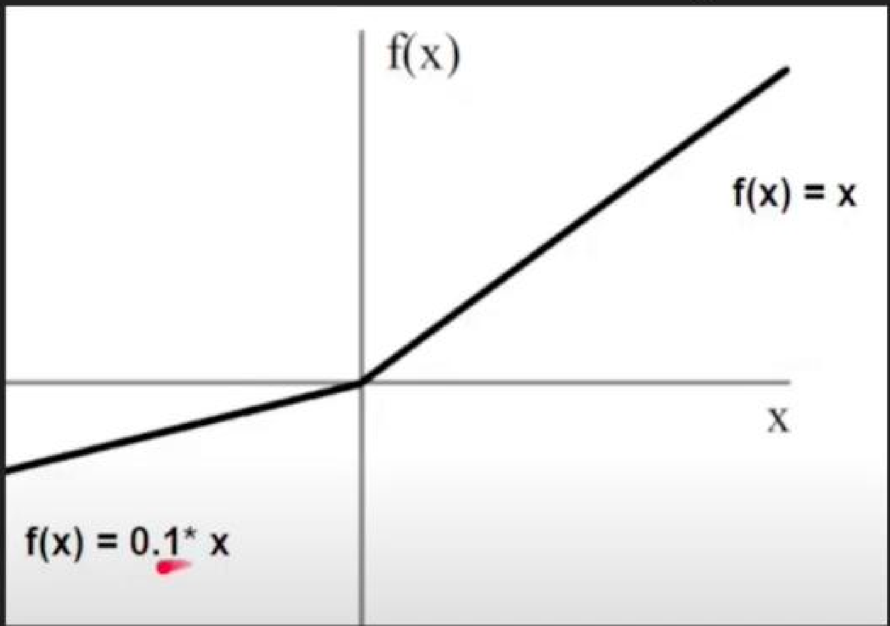
Relu Activation Function

01 June 2022 16:43



Leaky Relu

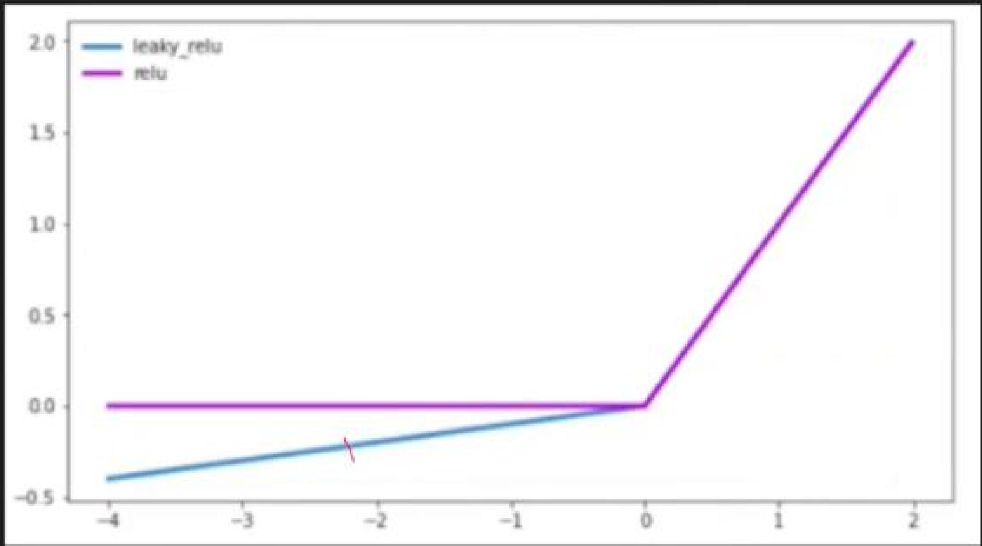
08 June 2022 22:53

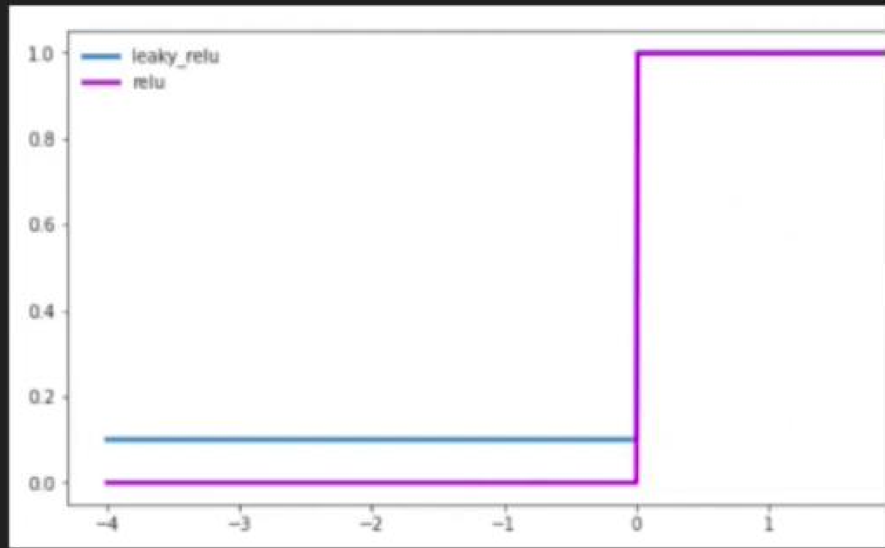


$$f(z) = \max(0.01z, z)$$

$$z \geq 0 \rightarrow z$$

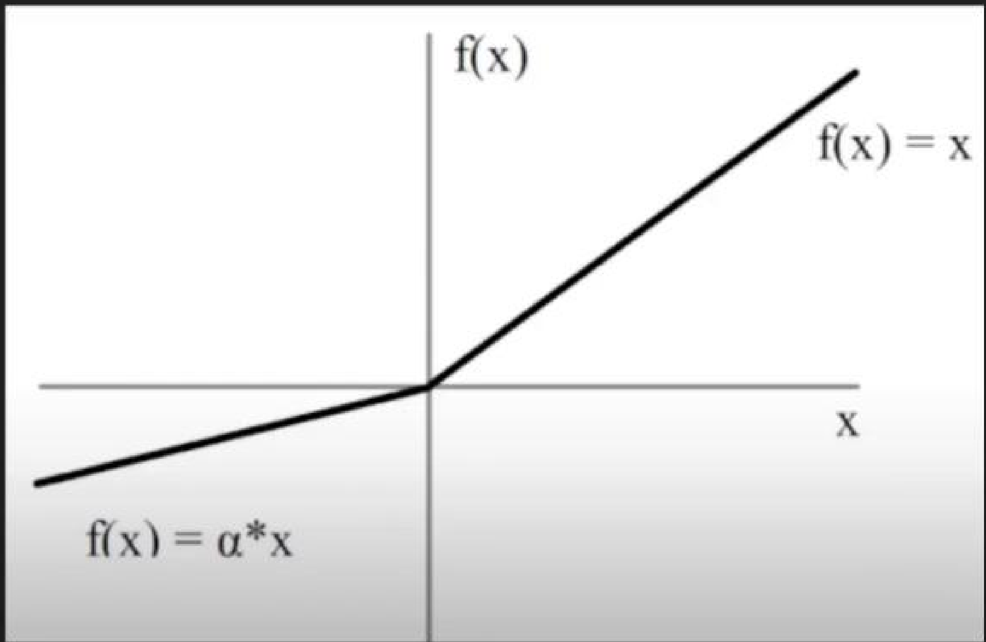
$$z < 0 \rightarrow \frac{1}{100} z \text{ (fraction of } z)$$


leaky_relu
relu
2.0
1.5
1.0
0.5
0.0
−0.5
−4
−3
−2
−1
0
1
2


leaky_relu
relu
1.0
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0
1

Parametric Relu

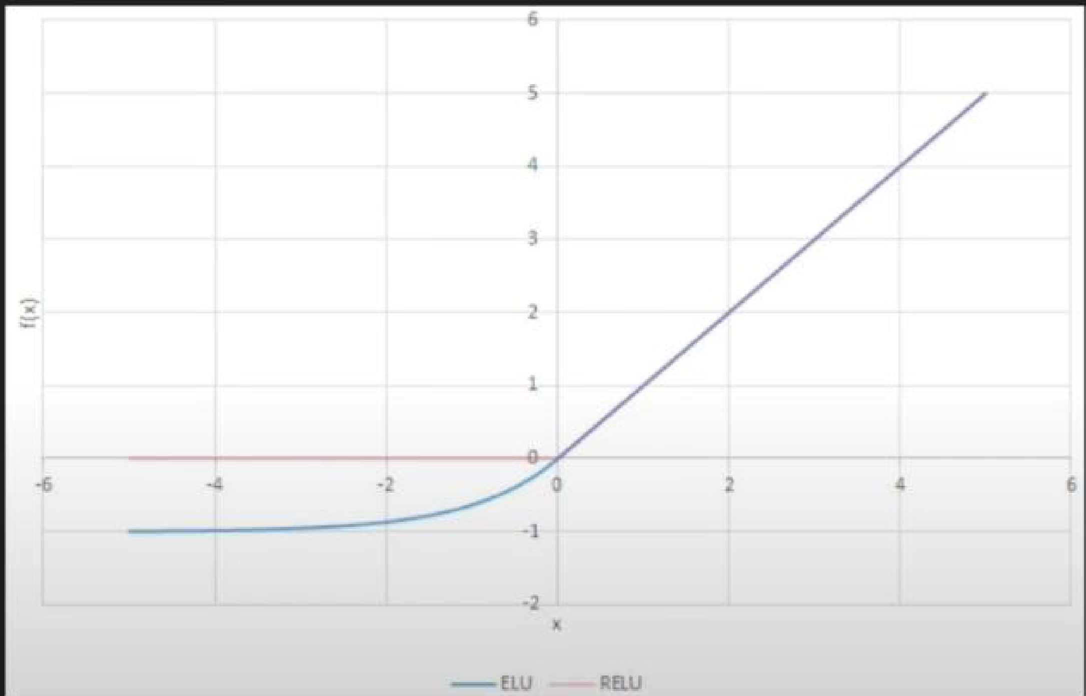
08 June 2022 22:53



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

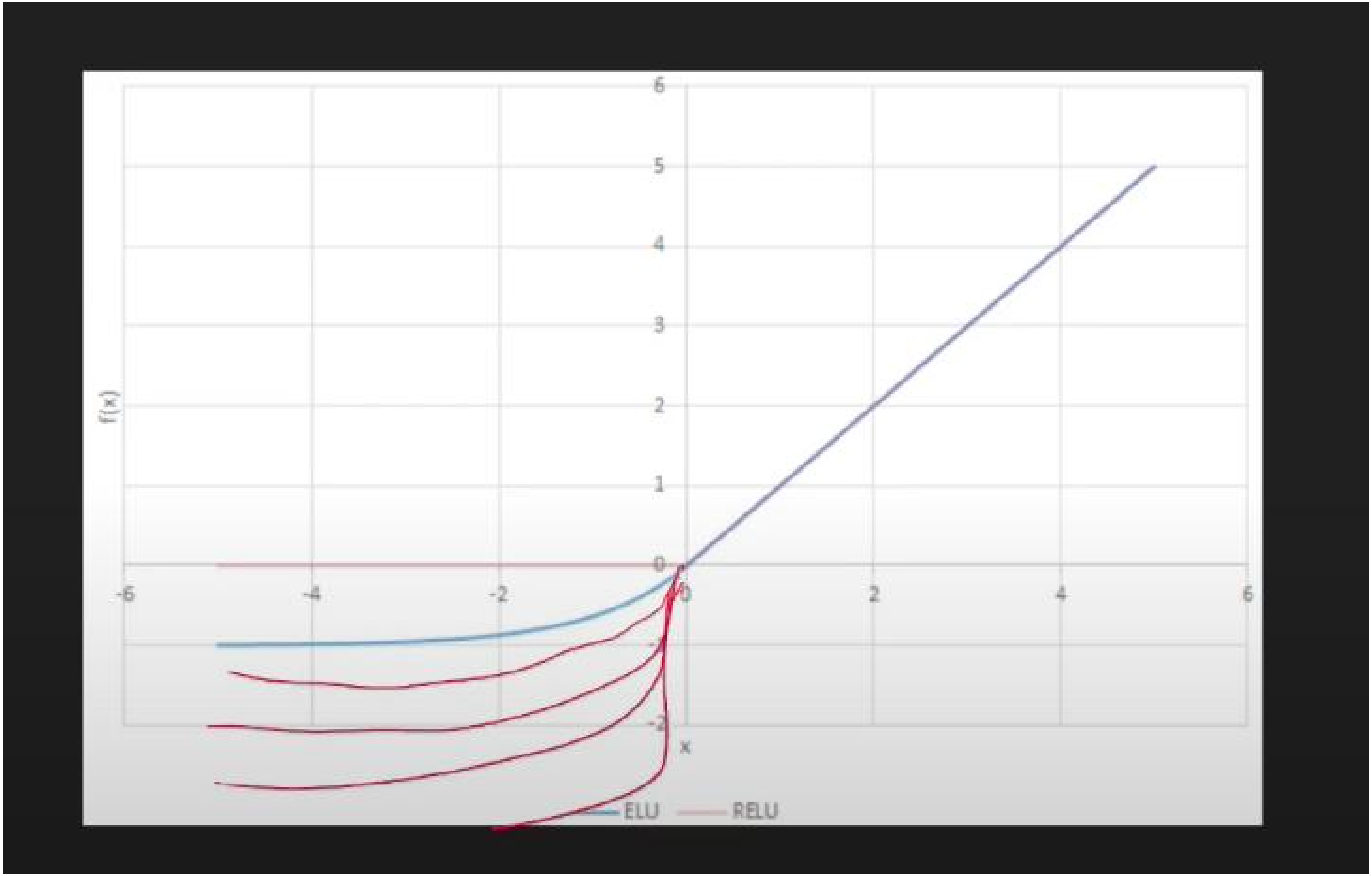
Elu - Exponential Linear Unit

09 June 2022 00:29



$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

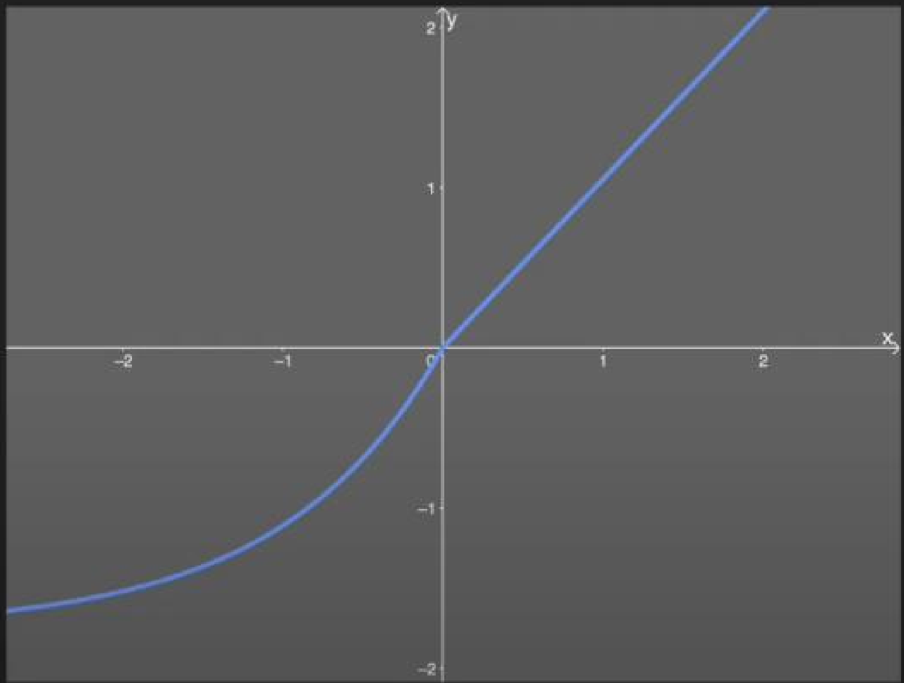
$$\mathrm{ELU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \mathrm{ELU}(x) + \alpha & \text{if } x \leq 0 \end{cases}$$



f(x)
x
6
5
4
3
2
1
0
-1
-2
-6
-4
-2
0
2
4
6
ELU
RELU

Selu - Scaled Exponential Linear Unit

09 June 2022 00:29

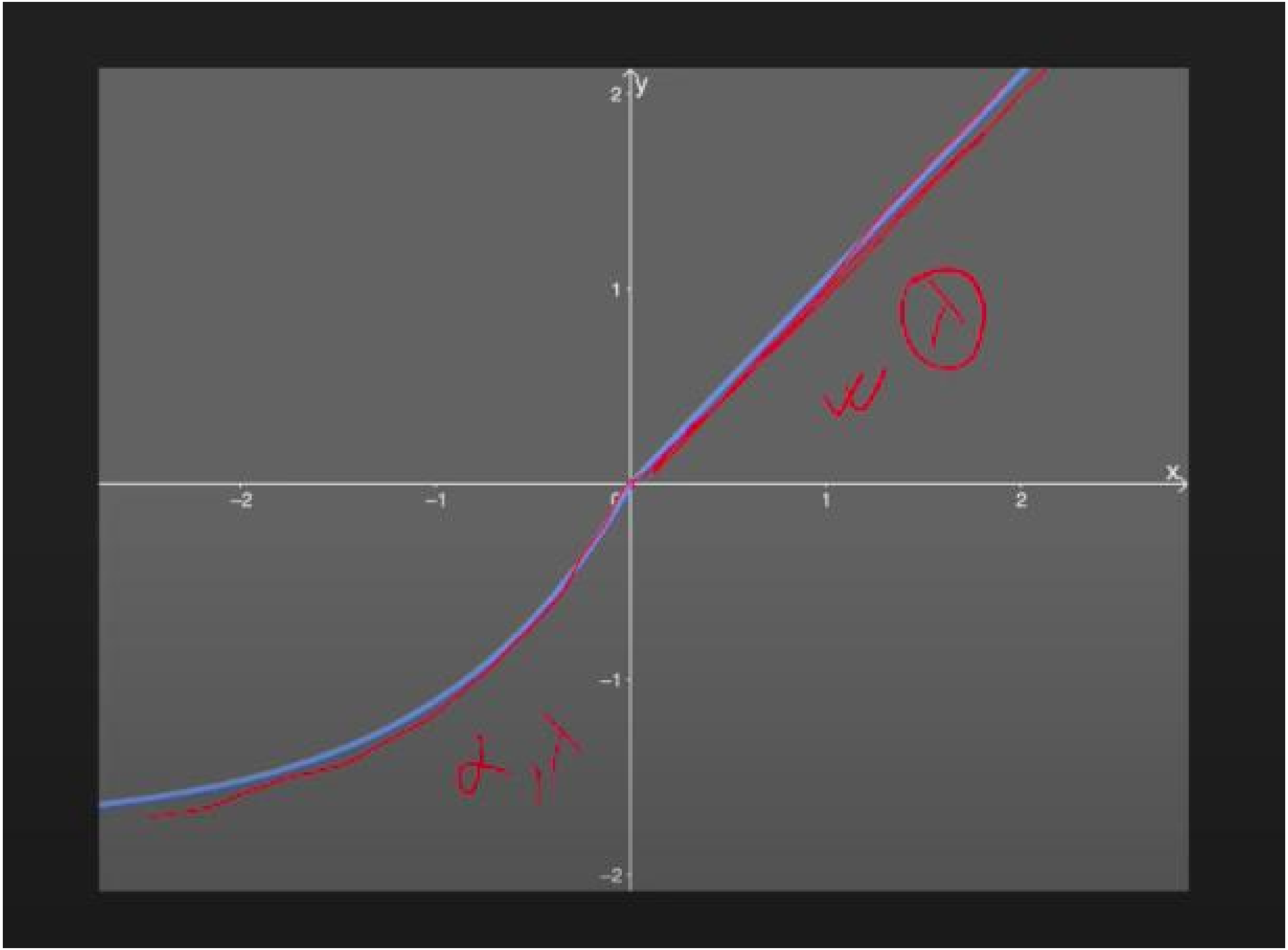


$$\mathrm{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^{x} - \alpha & \text{if } x \leq 0 \end{cases}$$

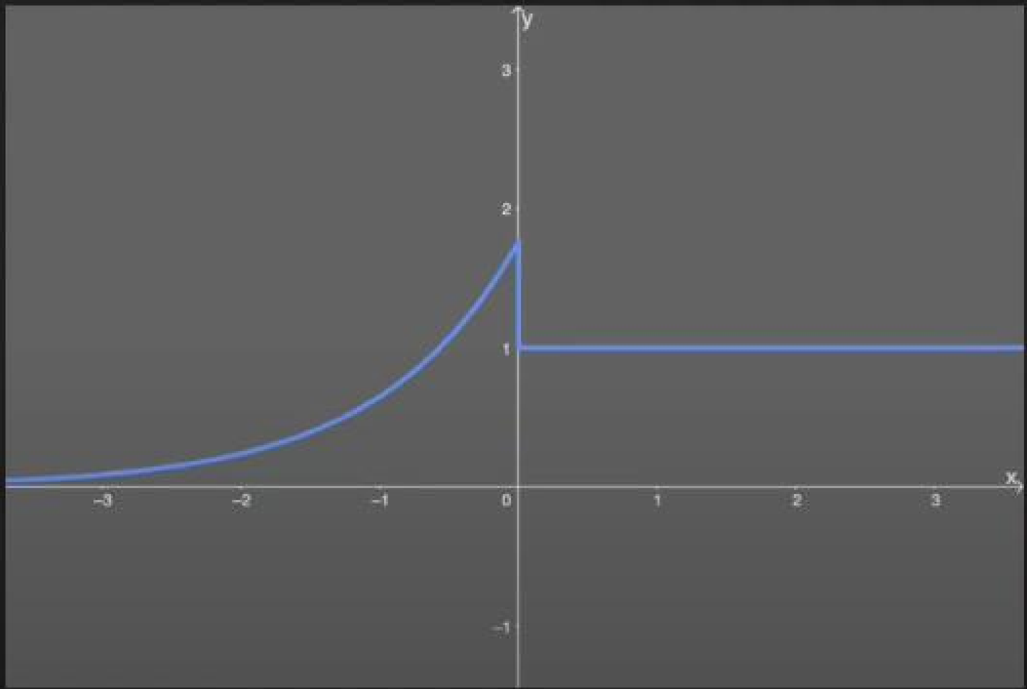
$$a \approx 1.6732632423543772848170429916717$$

$$\lambda \approx 1.0507009873554804934193349852946$$

$$\mathrm{SELU}'(x) = \lambda \begin{cases} 1 & \text{if } x > 0 \\ \alpha e^{x} & \text{if } x \leq 0 \end{cases}$$

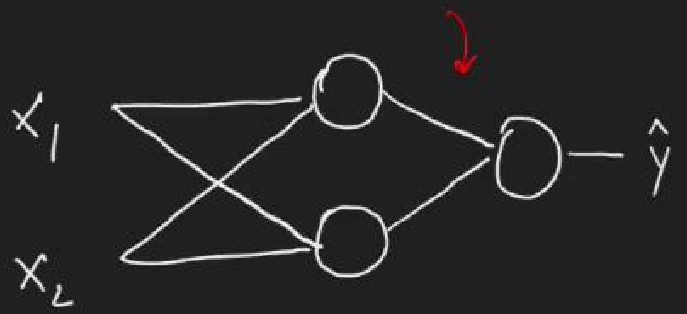


y
x
2
1
−1
−2
−2
−1
0
1
2


y
3
2
1
−1
x
−3
−2
−1
0
1
2
3

Why Weight Initialization is Important?

22 June 2022 12:48



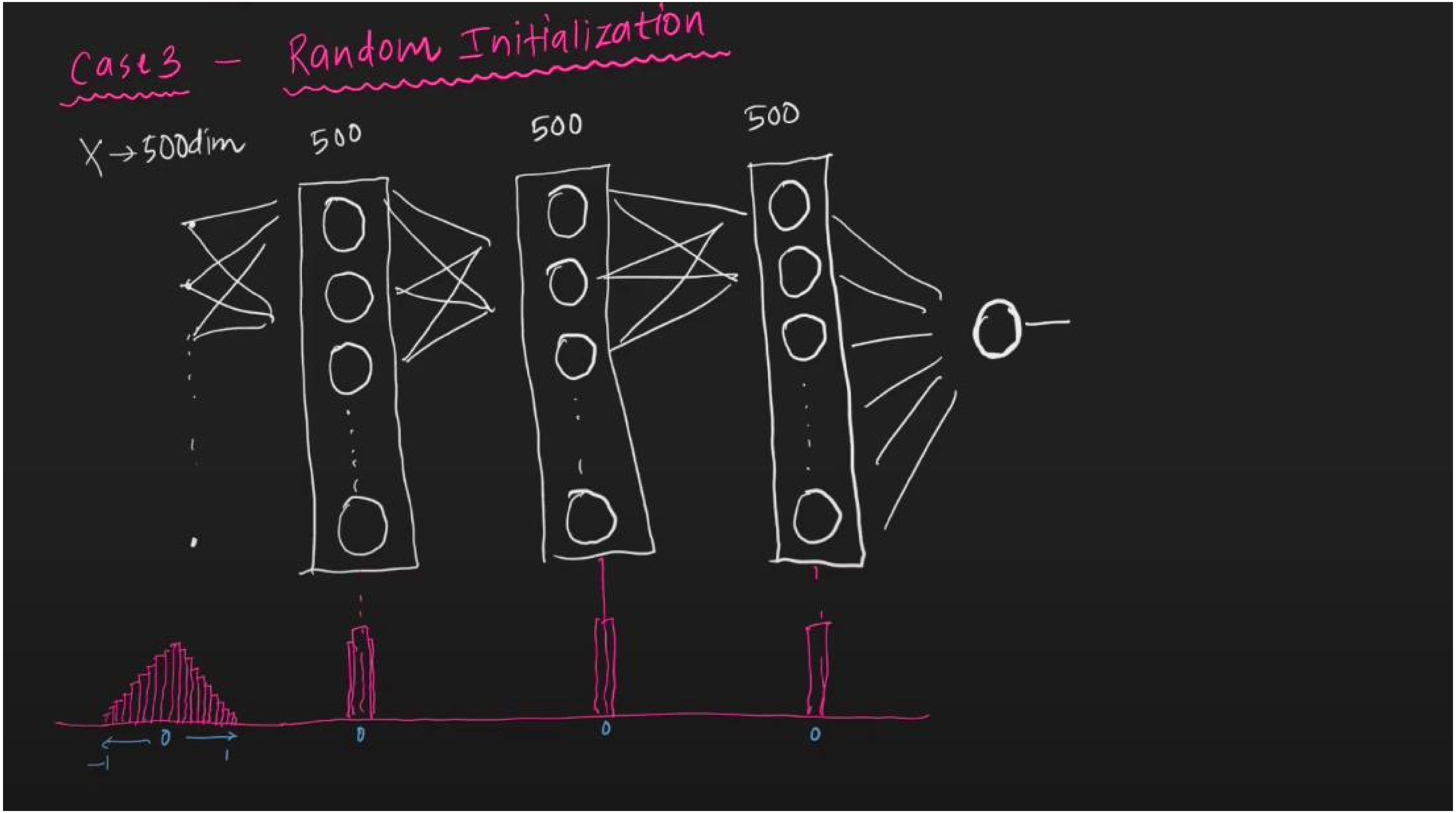
1. Initialize the parameters
2. Choose an *optimization algorithm*
3. Repeat these steps:
    1. Forward propagate an input
    2. Compute the cost function
    3. Compute the gradients of the cost with respect to parameters using backpropagation
    4. Update each parameter using the gradients, according to the optimization algorithm

$$\boxed{\frac{\partial L}{\partial w^1_{11}}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} X_1$$

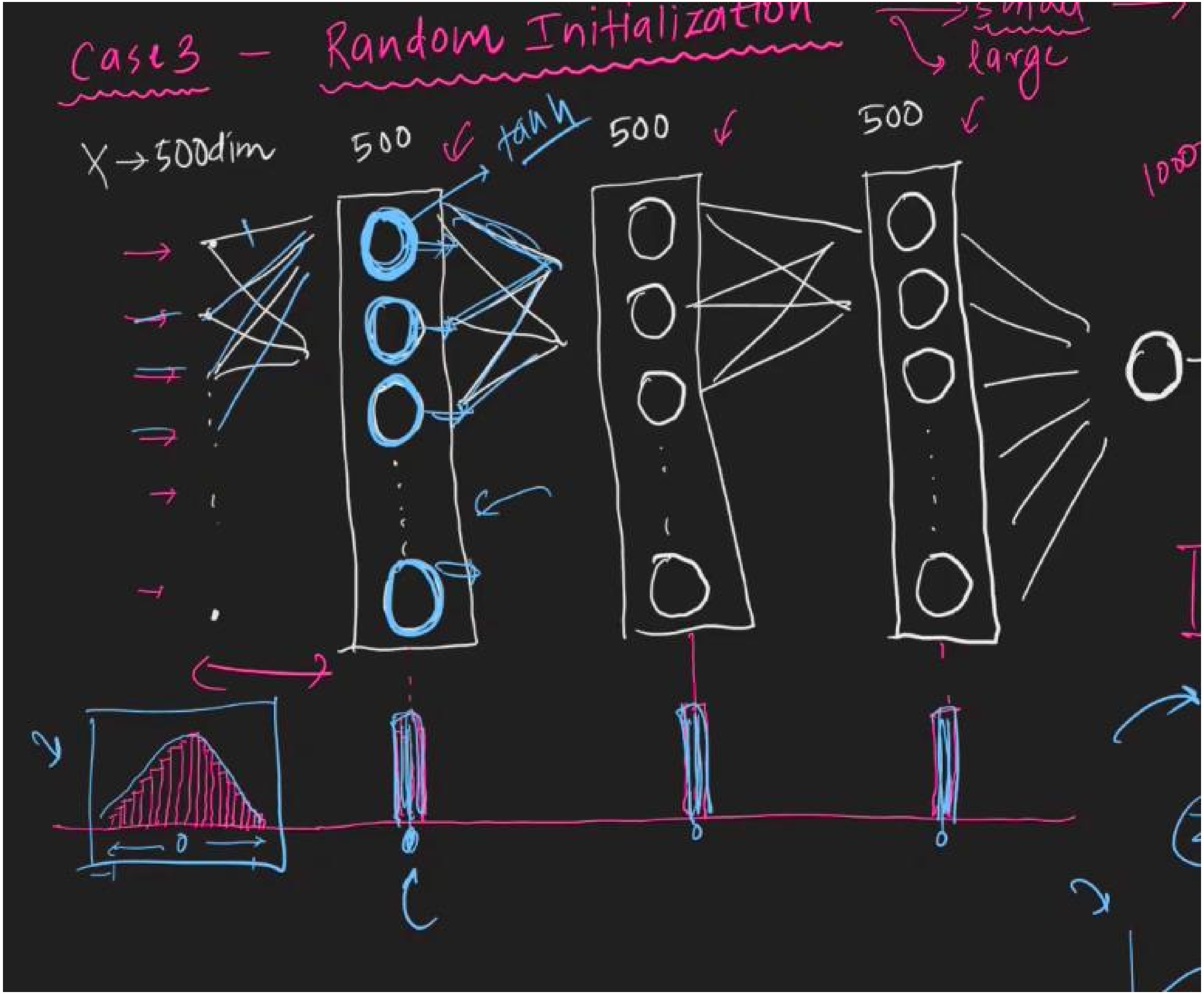
$$\boxed{\frac{\partial L}{\partial w^1_{12}}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{12}} \frac{\partial a_{12}}{\partial z_{12}} X_1$$

$$\frac{\partial L}{\partial w^1_{21}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} X_2$$

$$\frac{\partial L}{\partial w^1_{22}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{12}} \frac{\partial a_{12}}{\partial z_{12}} X_2$$

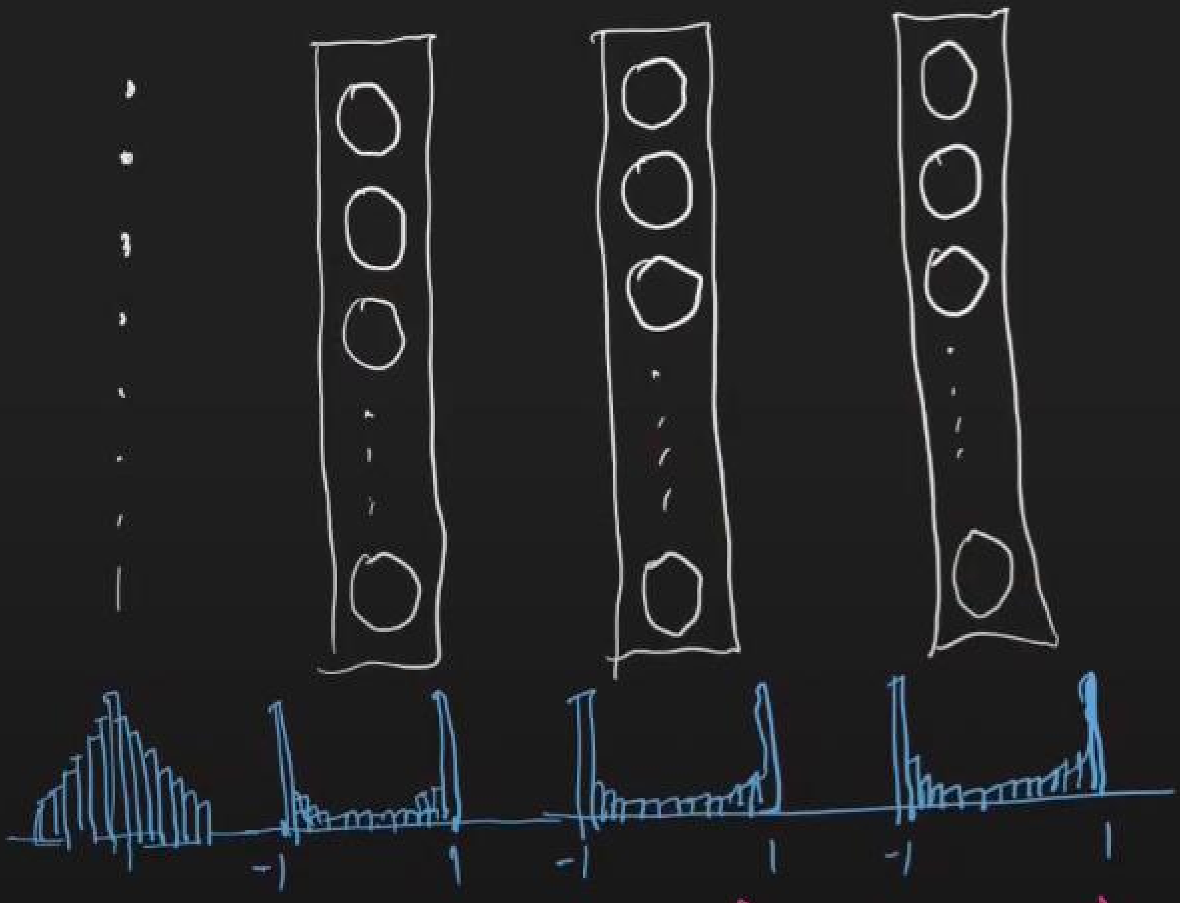


Case 3 - Random Initialization
X → 500dim
500
500
500
0
-1
1
0
0
0



Case3 - Random Initialization
small
large
X → 500dim
500
tanh
500
500
0
0
0

Random Initialization (Large values)



Batch-Normalization (BN) is an algorithmic method which makes the training of Deep Neural Networks (DNN) faster and more stable.

It consists of normalizing activation vectors from hidden layers using the mean and variance of the current batch. This normalization step is applied right before (or right after) the nonlinear function.

Covarite Shift


peas
bees
knees
cheese
fleas?

Keras Implementation

27 June 2022 11:03

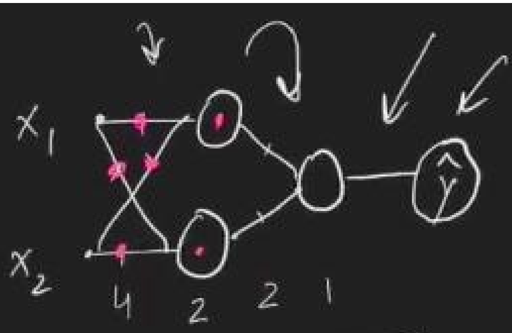
```
model = Sequential()

model.add(Dense(3,activation='relu',input_dim=2))
model.add(BatchNormalization())
model.add(Dense(2,activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1,activation='sigmoid'))
```

Role of Optimizer

05 July 2022 10:01

cgpa	iq	placed
8	80	1
9	90	1
7	70	0

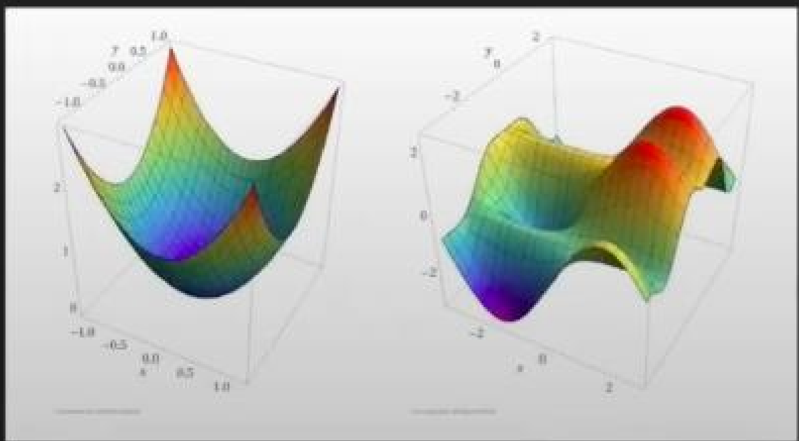


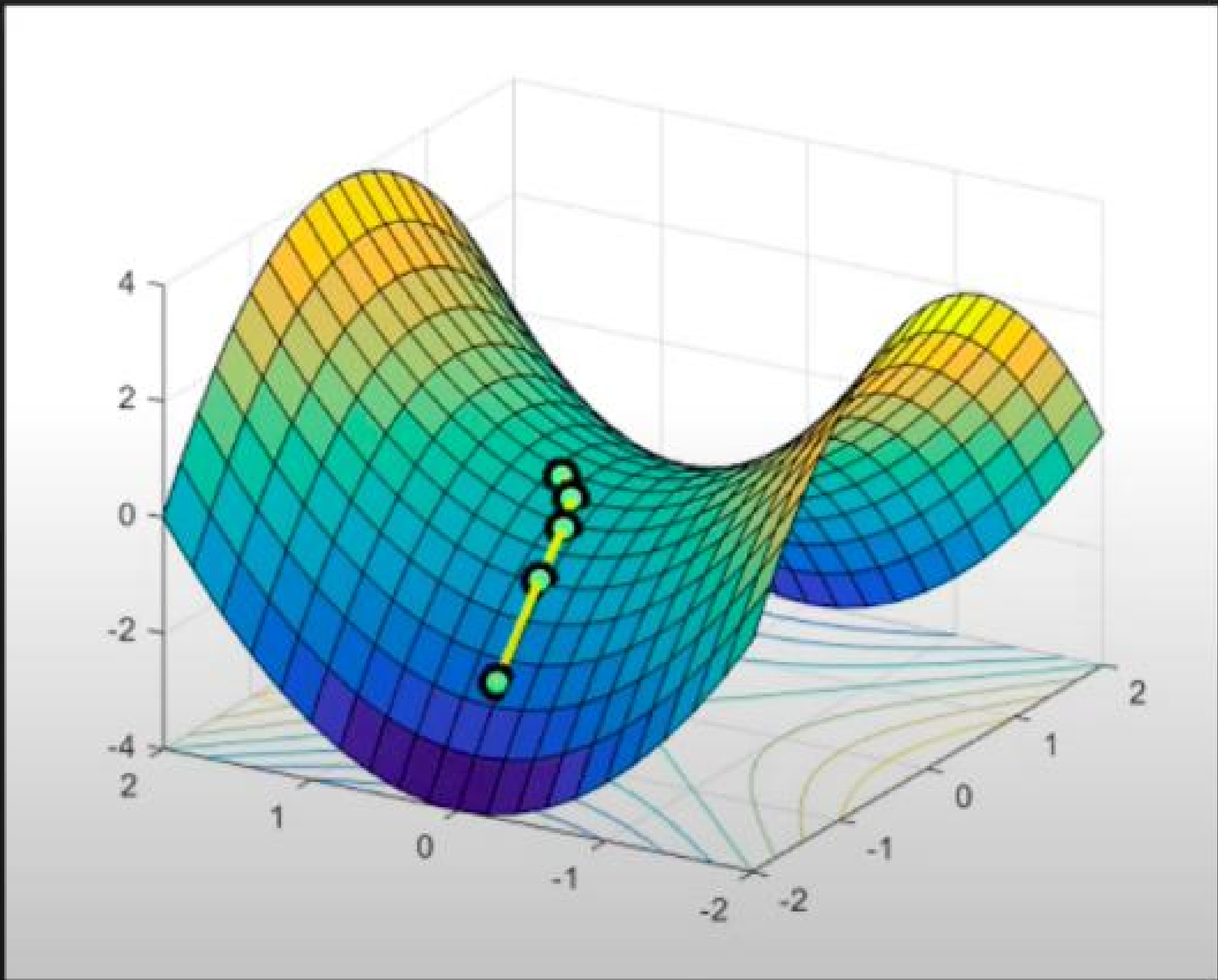
$\boxed{\text{9 parameters}}$

optimization

2 min

$y \quad \hat{y} \quad (w, b)$

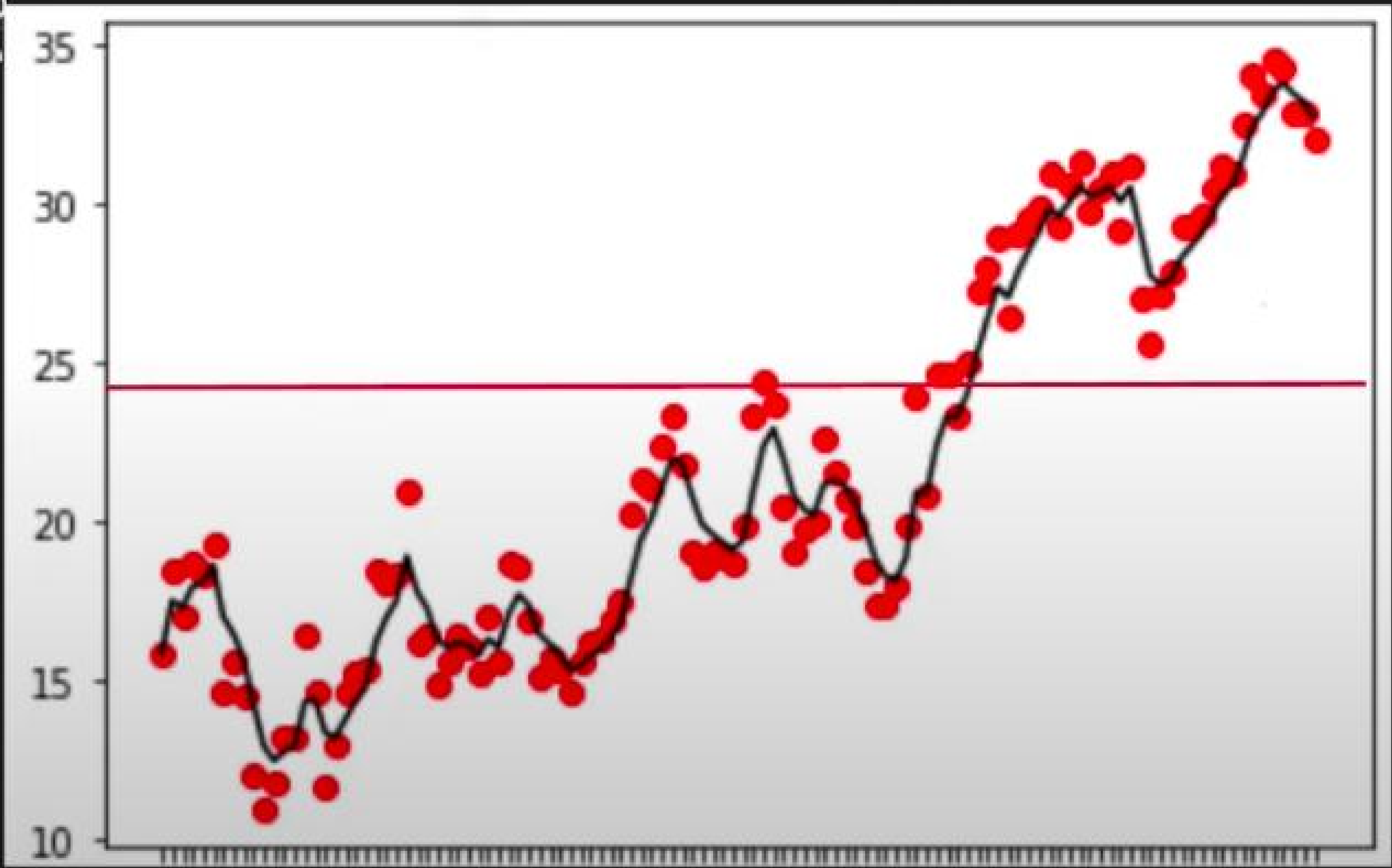



4
2
0
-2
-4
2
1
0
-1
-2
2
1
0
-1
-2

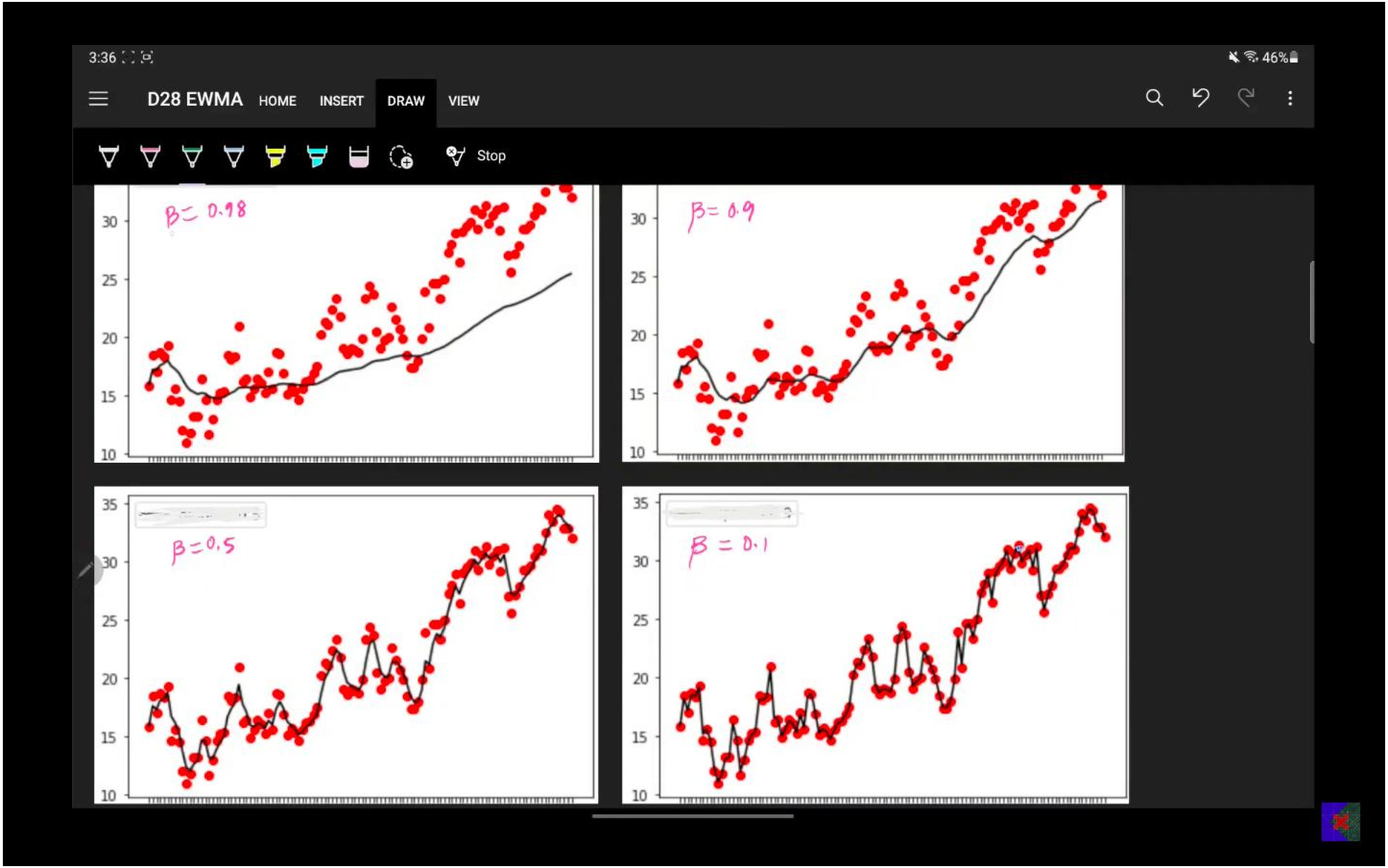
05 July 2022 20:15

time series

temp



dates


3:36
46%
D28 EWMA
HOME
INSERT
DRAW
VIEW
Stop
$\beta = 0.98$
$\beta = 0.9$
$\beta = 0.5$
$\beta = 0.1$
10
15
20
25
30
35

D28 EWMA HOME INSERT DRAW VIEW

Stop

Mathematical Intuition

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$$V_0 = 0$$

$$V_1 = (1-\beta)\theta_1$$

$$V_2 = \beta V_1 + (1-\beta)\theta_2$$

$$= \beta(1-\beta)\theta_1 + (1-\beta)\theta_2$$

$$V_3 = \beta V_2 + (1-\beta)\theta_3$$

$$= \beta^2(1-\beta)\theta_1 + \beta(1-\beta)\theta_2 + (1-\beta)\theta_3$$

$$V_4 = \beta V_3 + (1-\beta)\theta_4$$

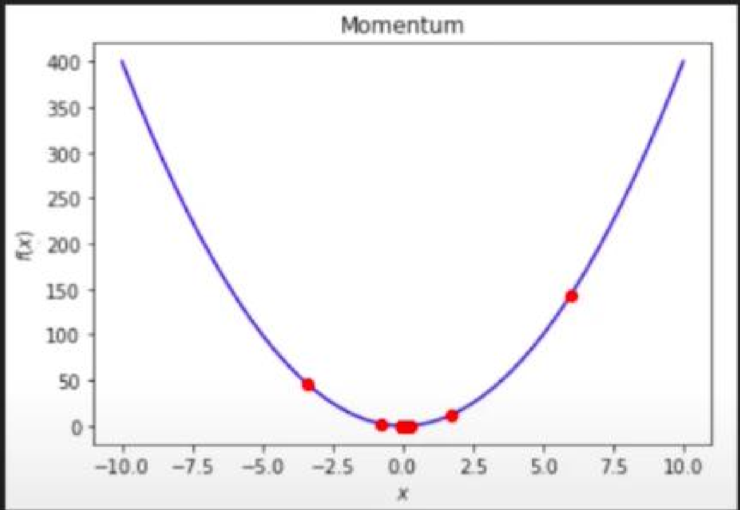
$$V_4 = \beta^3(1-\beta)\theta_1 + \beta^2(1-\beta)\theta_2 + \beta(1-\beta)\theta_3 + (1-\beta)\theta_4$$

$$= (1-\beta)\left[\beta^3\theta_1 + \beta^2\theta_2 + \beta\theta_3 + \theta_4\right]$$

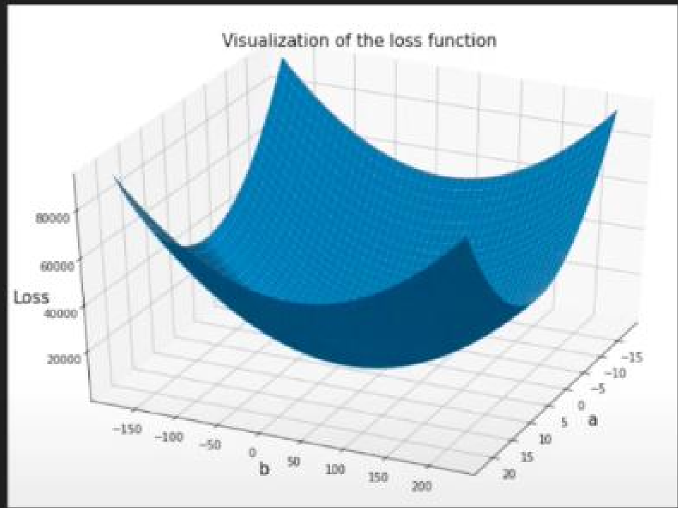
$$\beta^3 < \beta^2 < \beta \qquad 0 < \beta < 1$$

Understanding Graphs

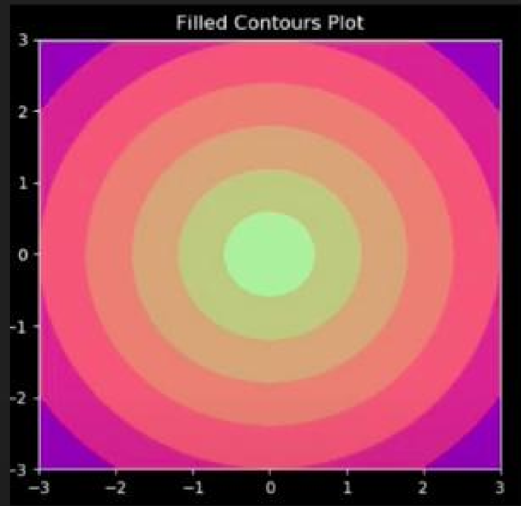
20 July 2022 12:03



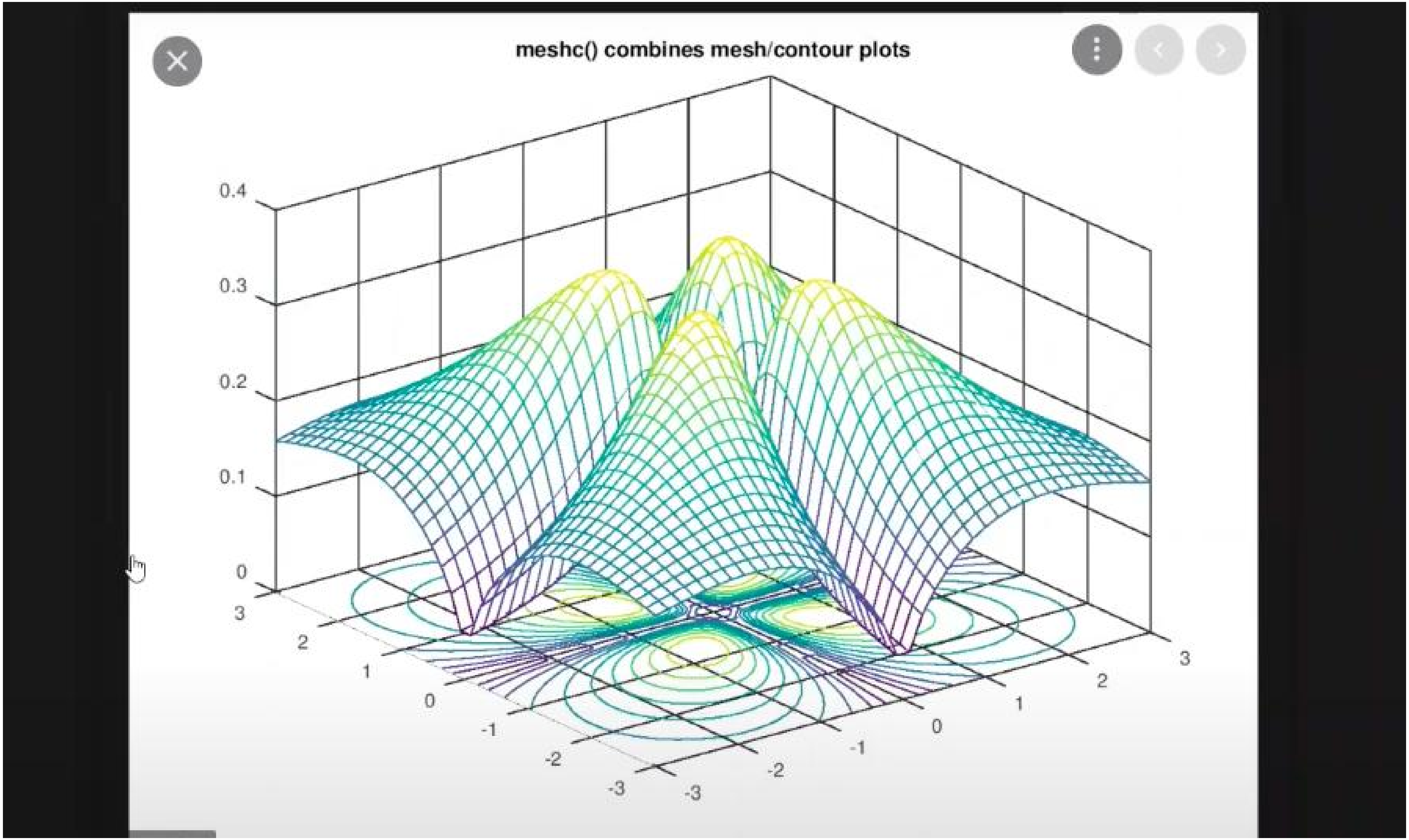
2D

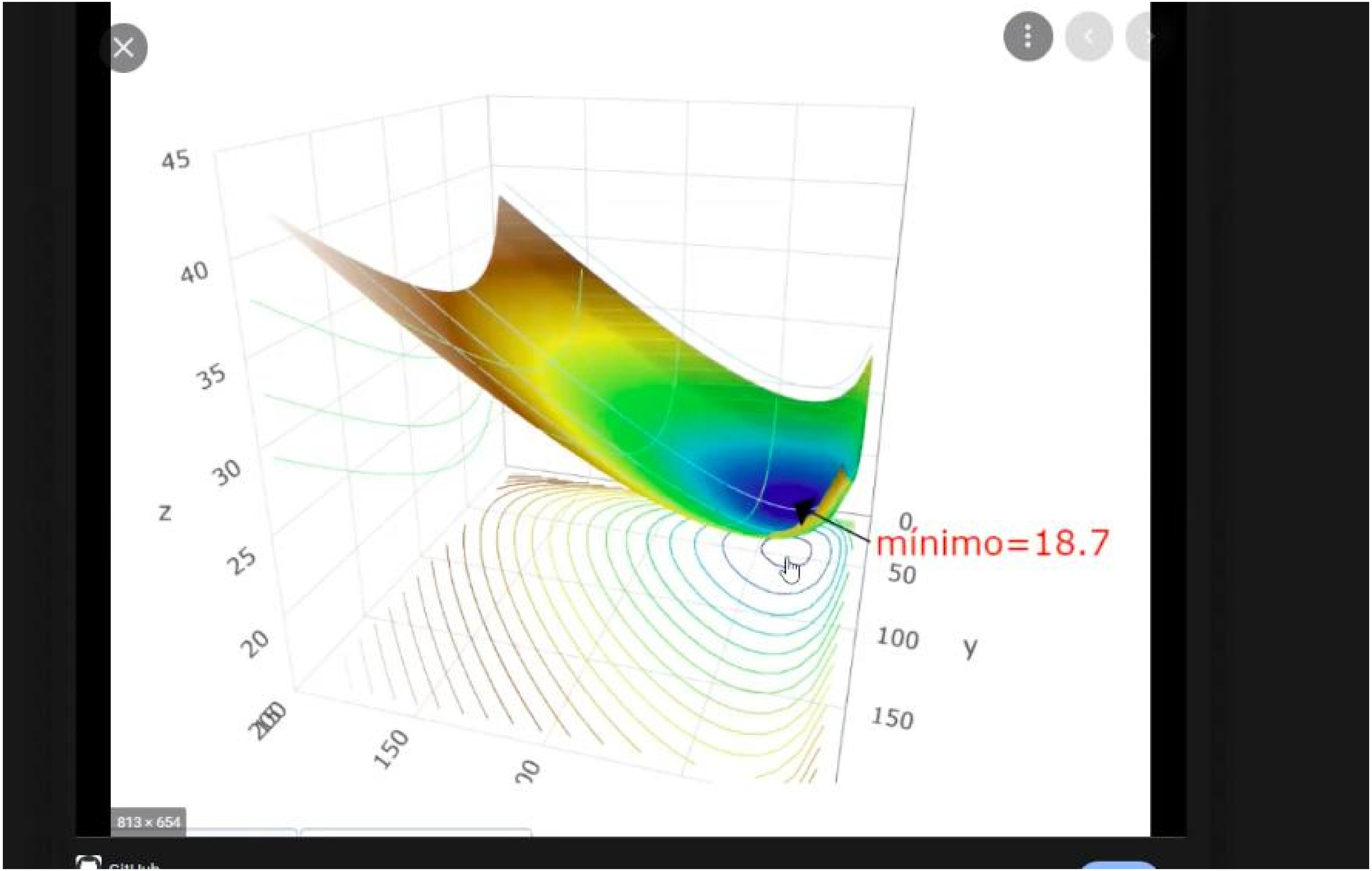


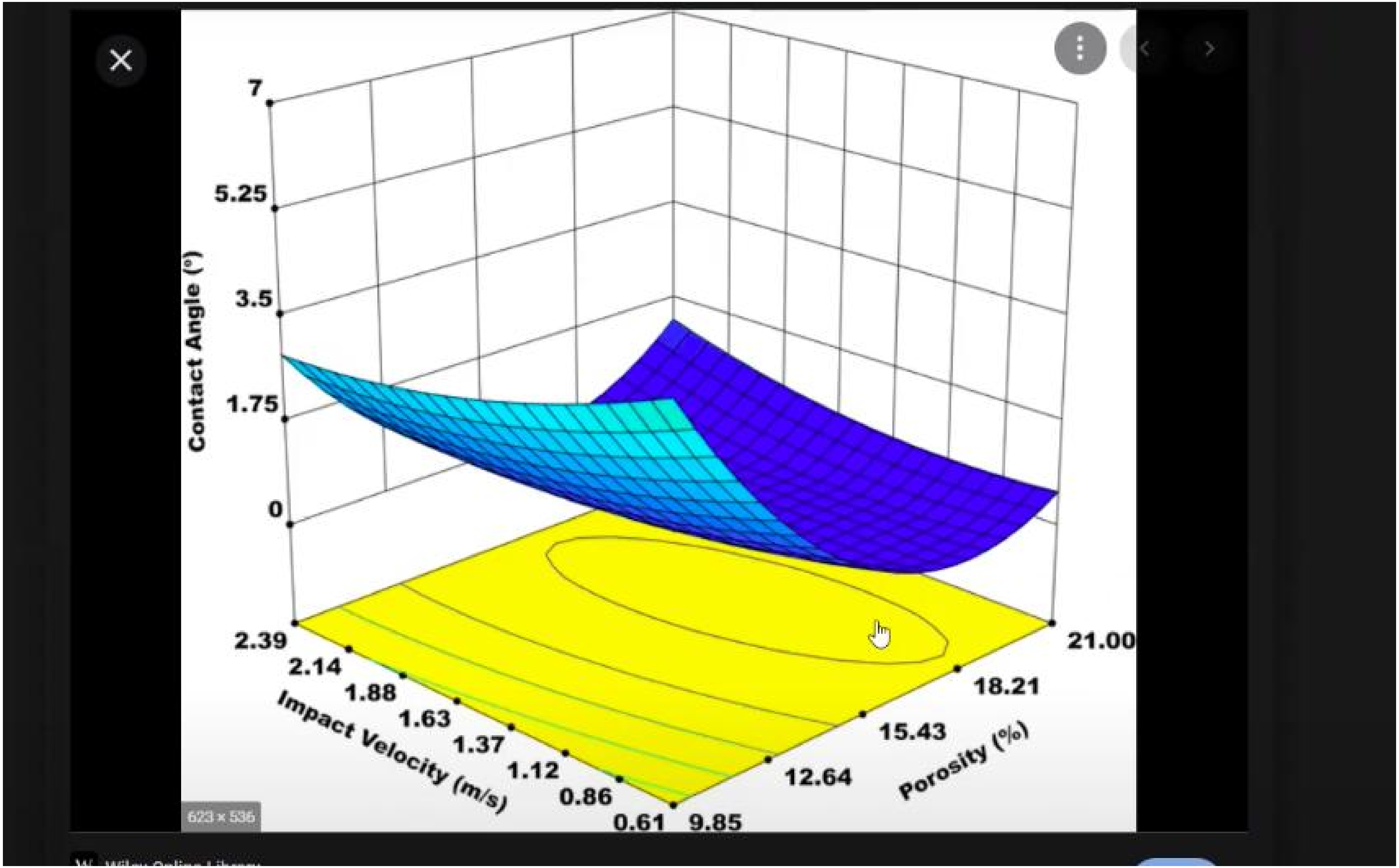
3D

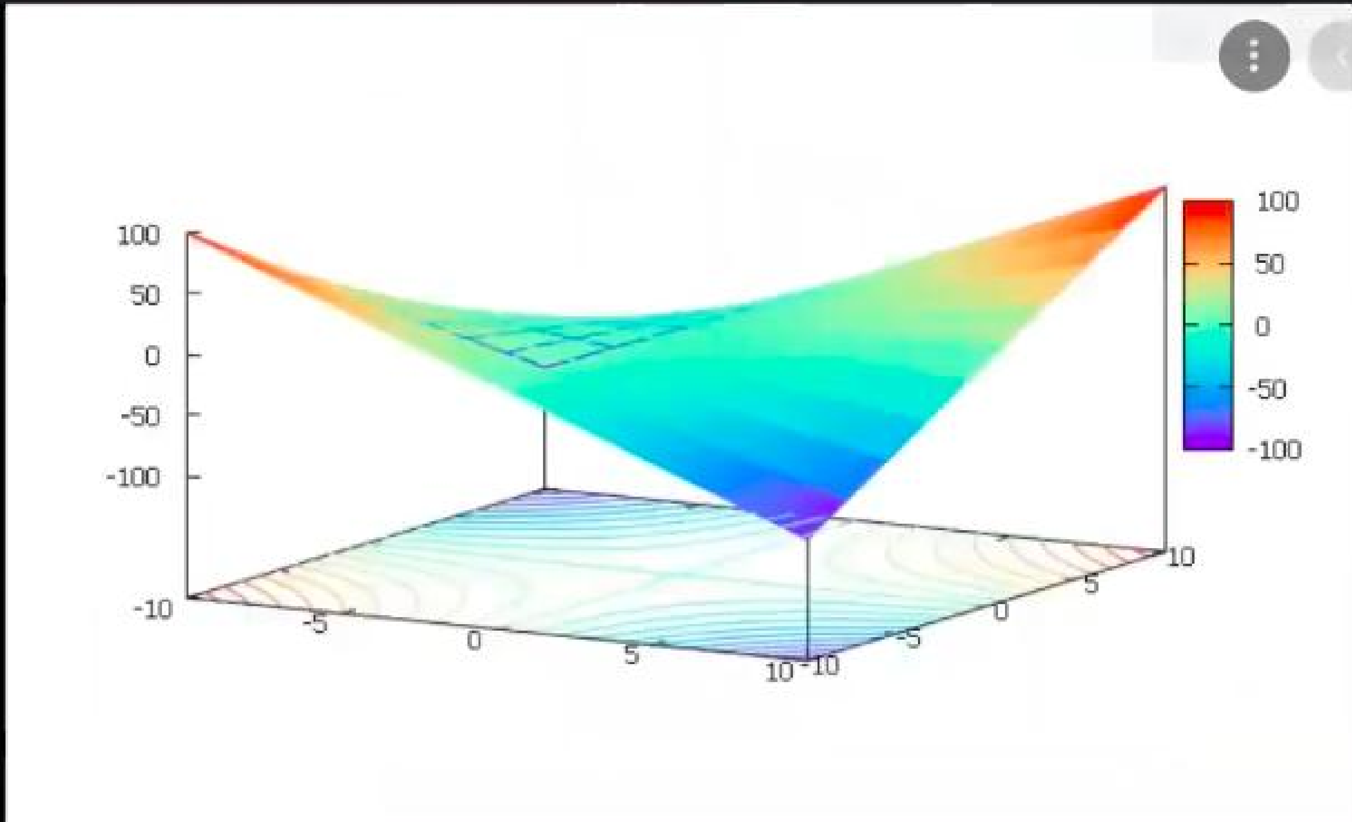


Contour


meshc() combines mesh/contour plots
0.4
0.3
0.2
0.1
0
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


45
40
35
30
25
20
z
0
50
100
150
y
150
mínimo=18.7
813 × 654


Contact Angle (°)
7
5.25
3.5
1.75
0
2.39
2.14
1.88
1.63
1.37
1.12
0.86
0.61
Impact Velocity (m/s)
9.85
12.64
15.43
18.21
21.00
Porosity (%)
623 × 536



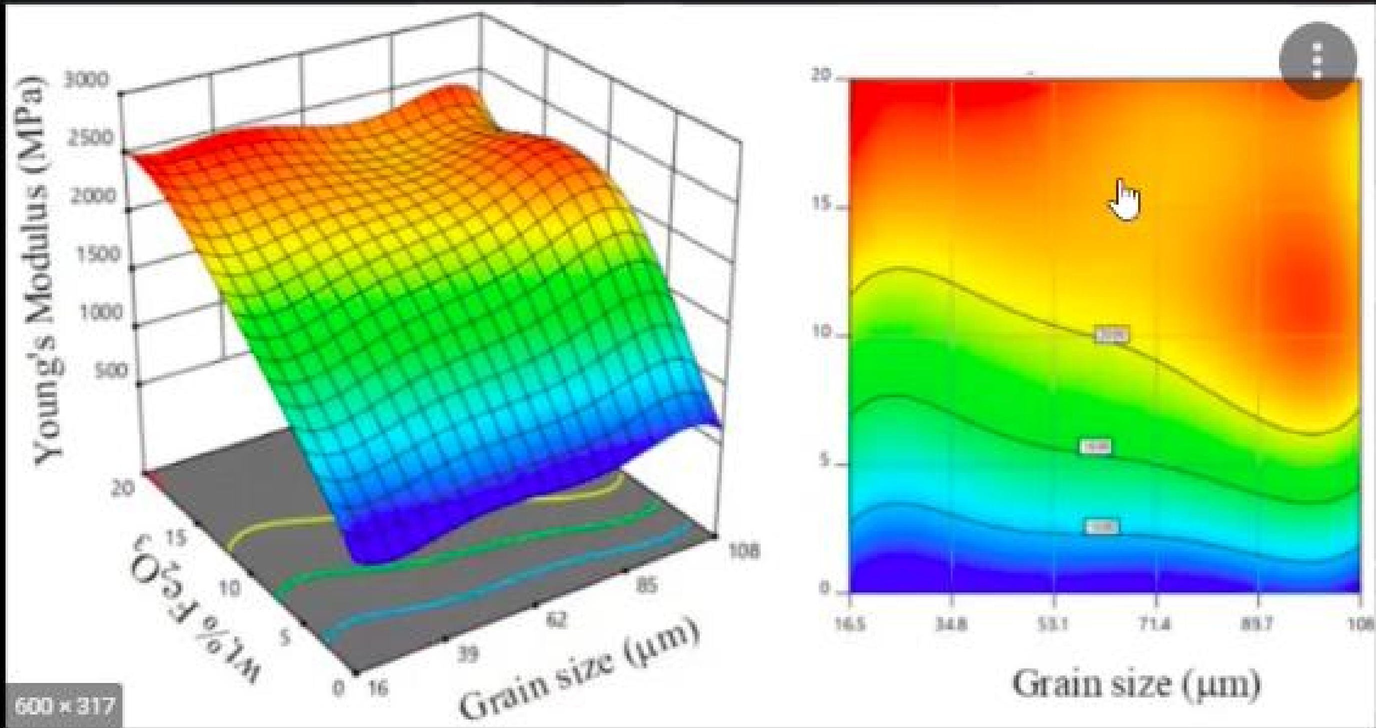
Stack Overflow

gnuplot - how to obtain contour lines with the same level color of the 3d plot - Stack Overflow

Visit

Related images

See more

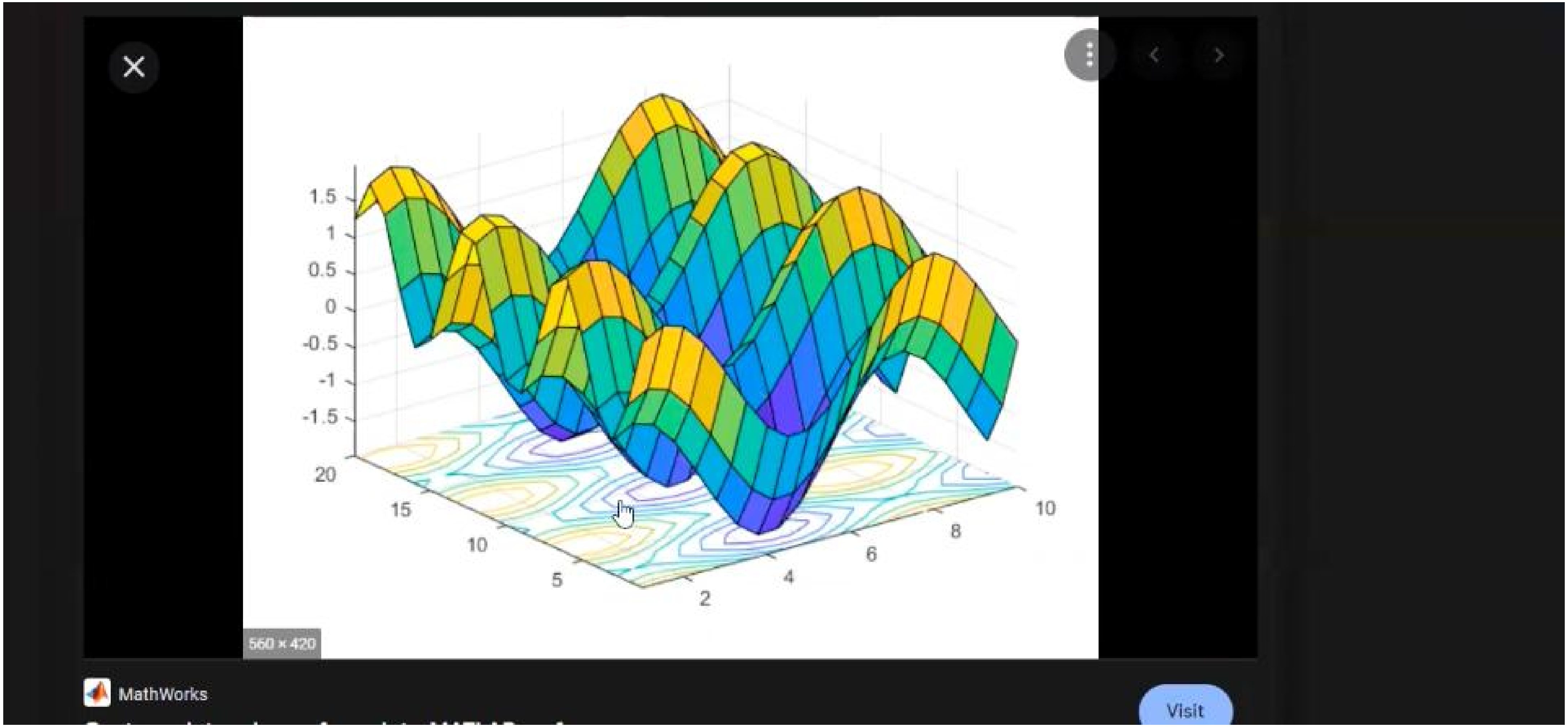


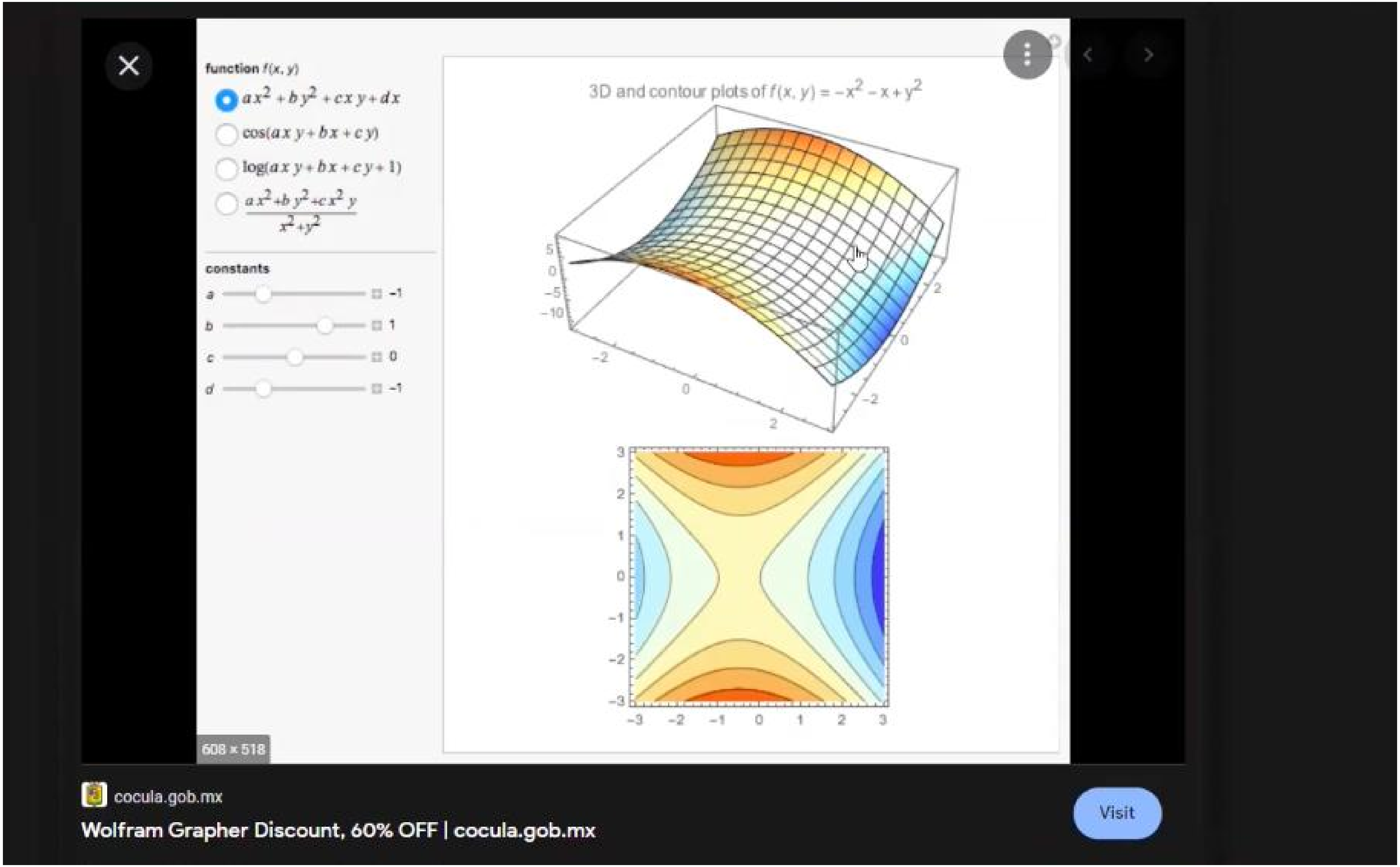
600 × 317

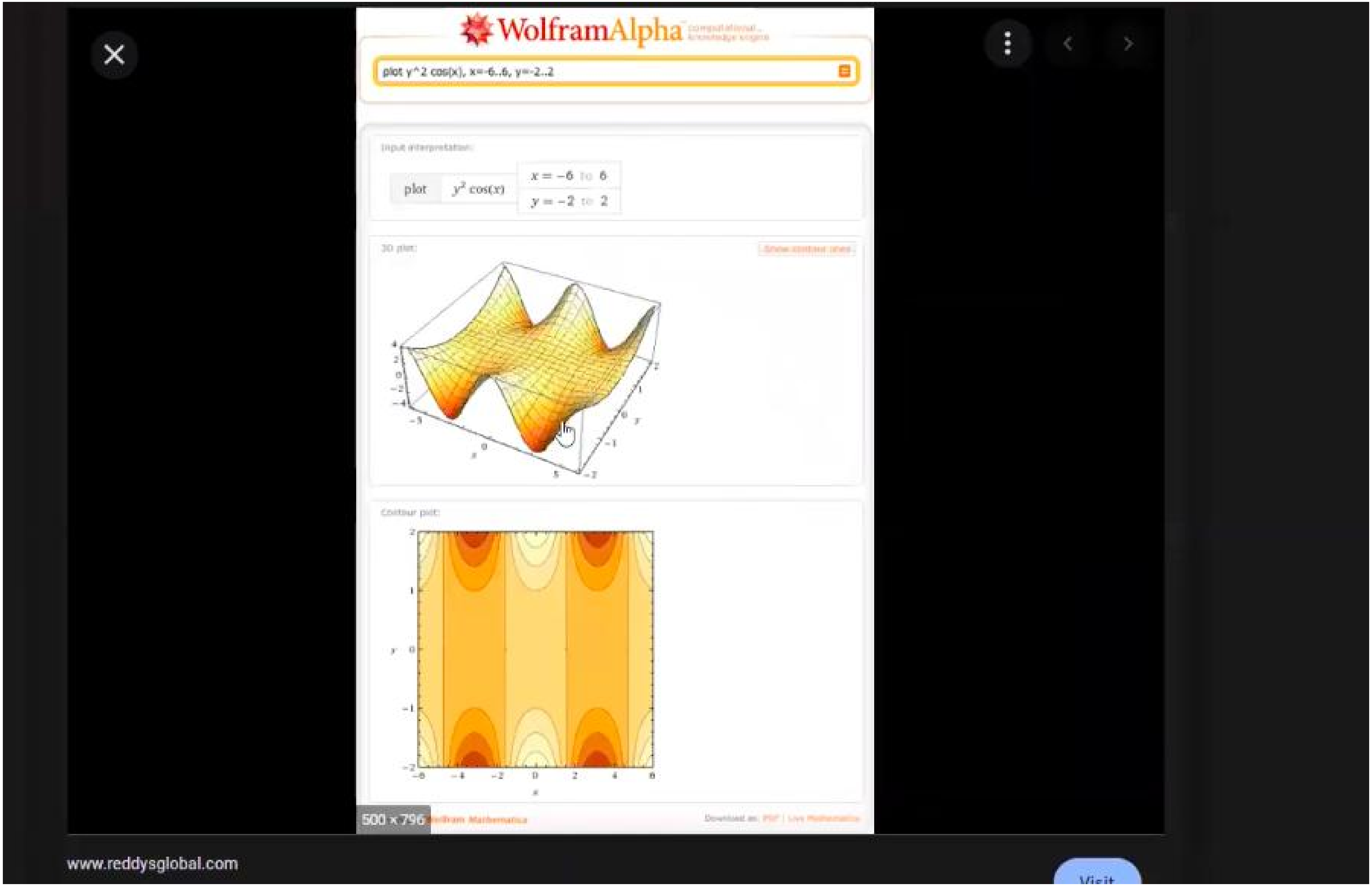
ScienceDirect.com

An experimental investigation on mechanical properties of Fe2O3 microparticles reinforced

Visit

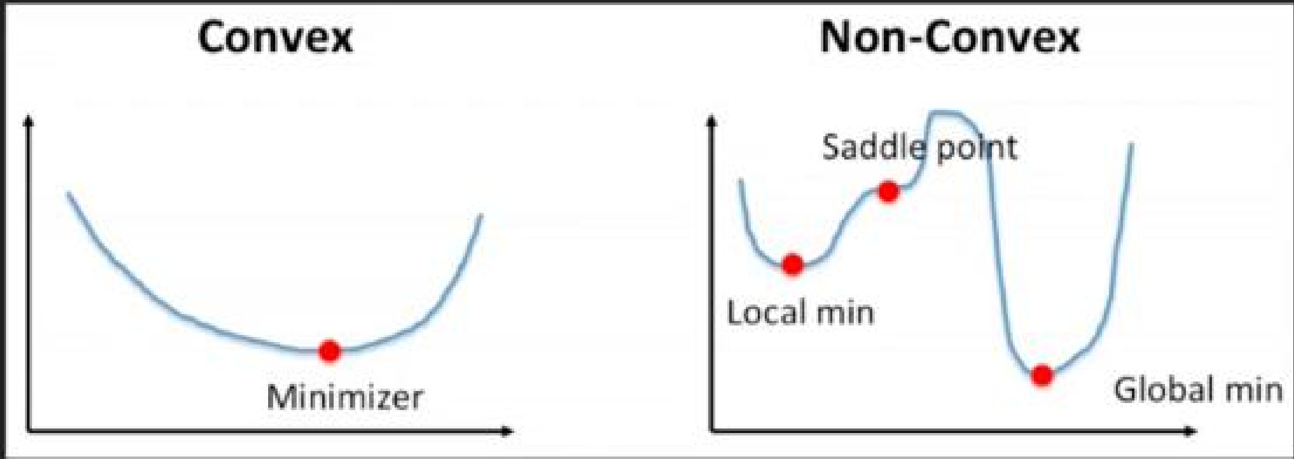

1.5
1
0.5
0
-0.5
-1
-1.5
20
15
10
5
2
4
6
8
10
560 × 420
MathWorks
Visit

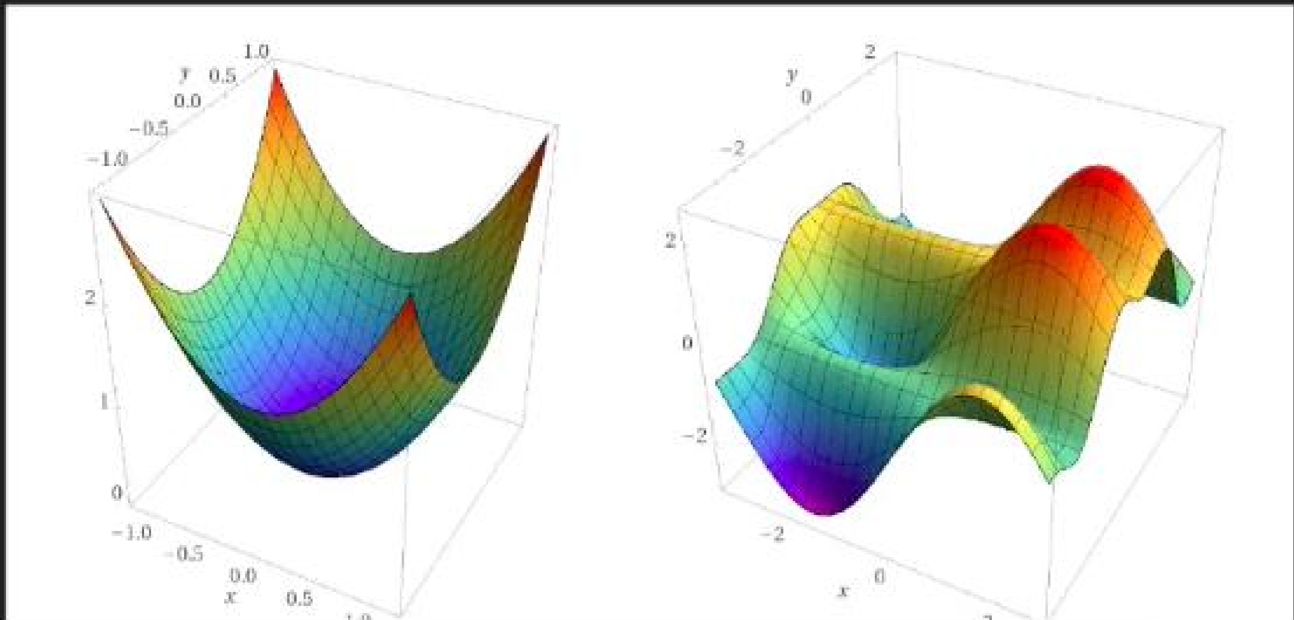

function $f(x, y)$
$a x^2 + b y^2 + c x y + d x$
$\cos(a x y + b x + c y)$
$\log(a x y + b x + c y + 1)$
$\frac{a x^2 + b y^2 + c x^2 y}{x^2 + y^2}$
constants
3D and contour plots of $f(x, y) = -x^2 - x + y^2$
608 × 518
cocula.gob.mx
Wolfram Grapher Discount, 60% OFF | cocula.gob.mx
Visit

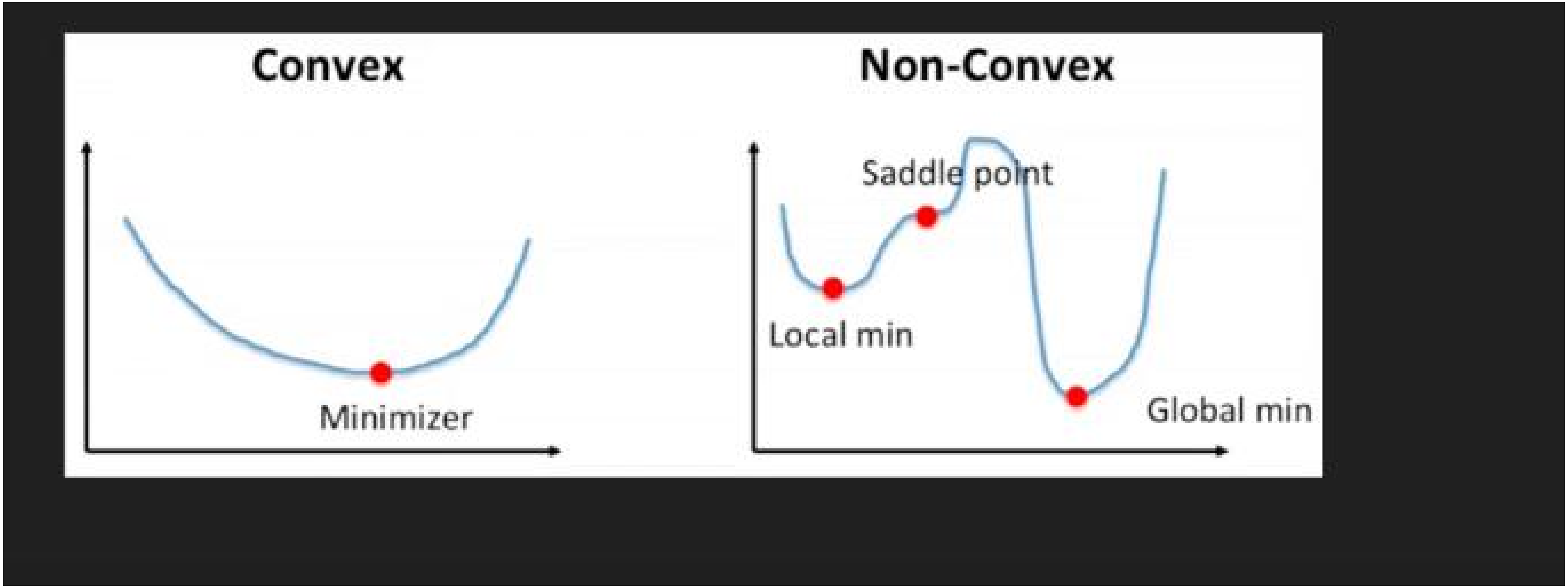

WolframAlpha
plot y^2 cos(x), x=-6..6, y=-2..2
Input interpretation:
plot
y^2 cos(x)
x = -6 to 6
y = -2 to 2
3D plot:
Contour plot:
500 × 796
www.reddysglobal.com

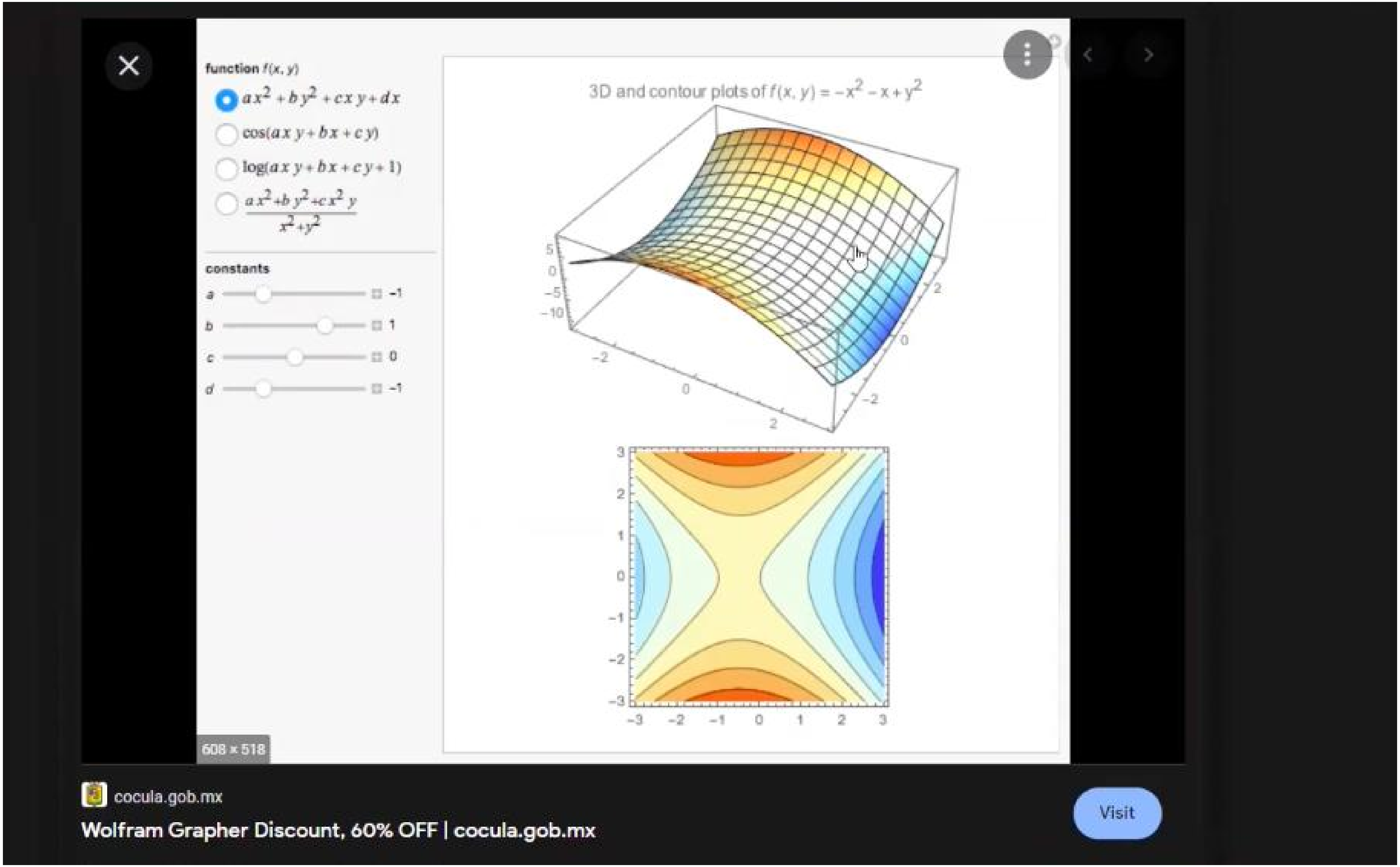
Convex Vs Non-Convex Optimization

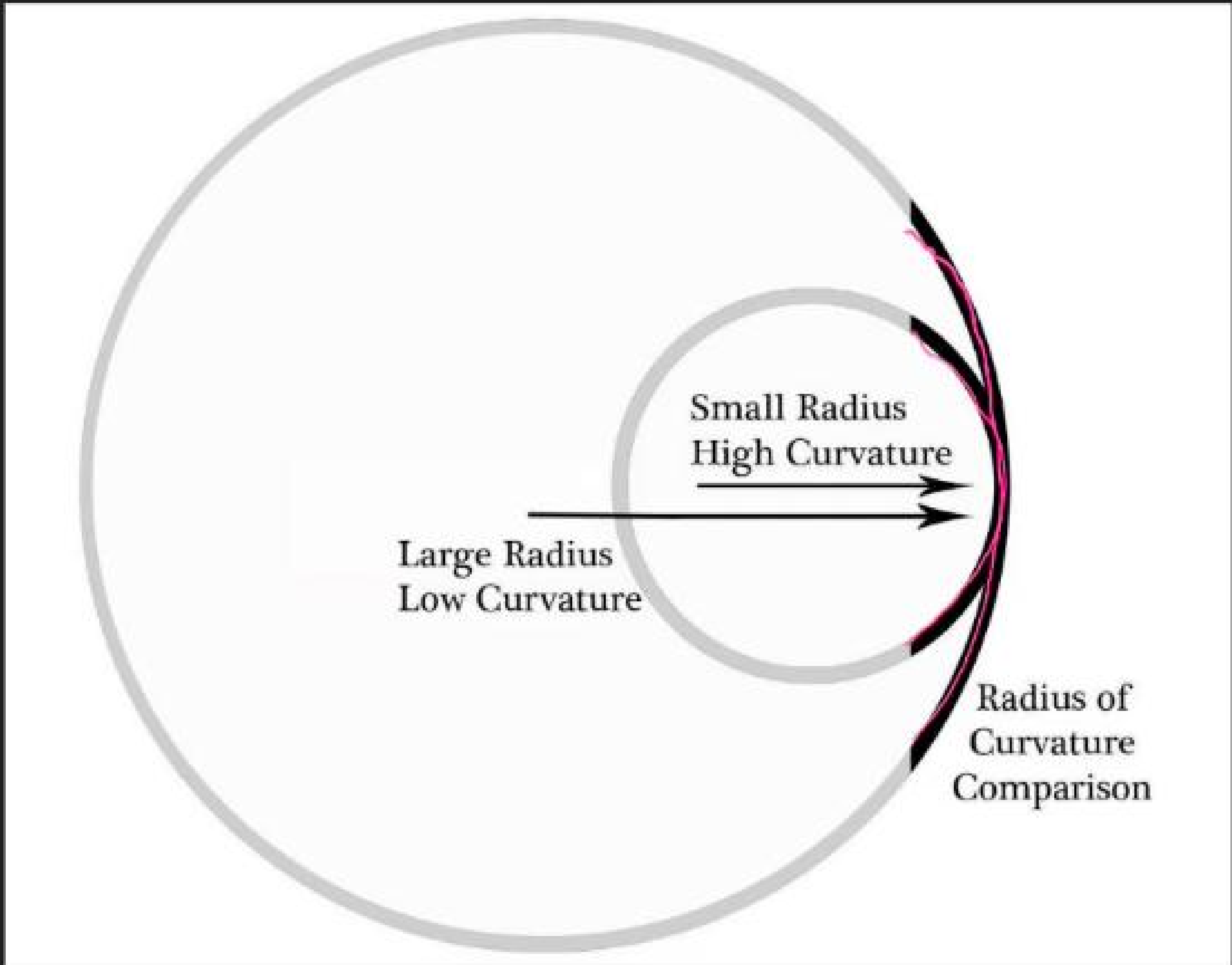
20 July 2022 13:06

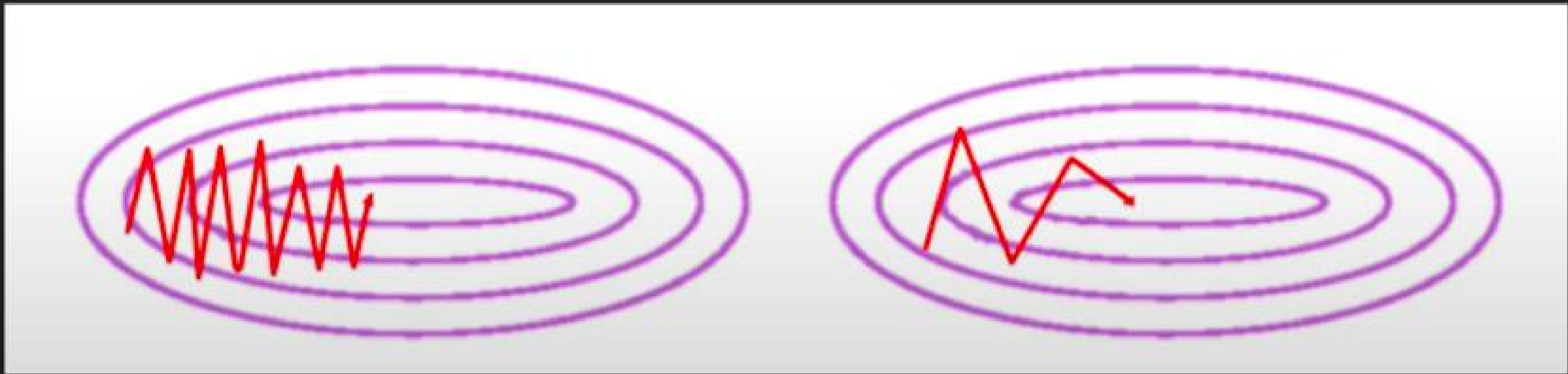





Convex
Minimizer
Non-Convex
Saddle point
Local min
Global min

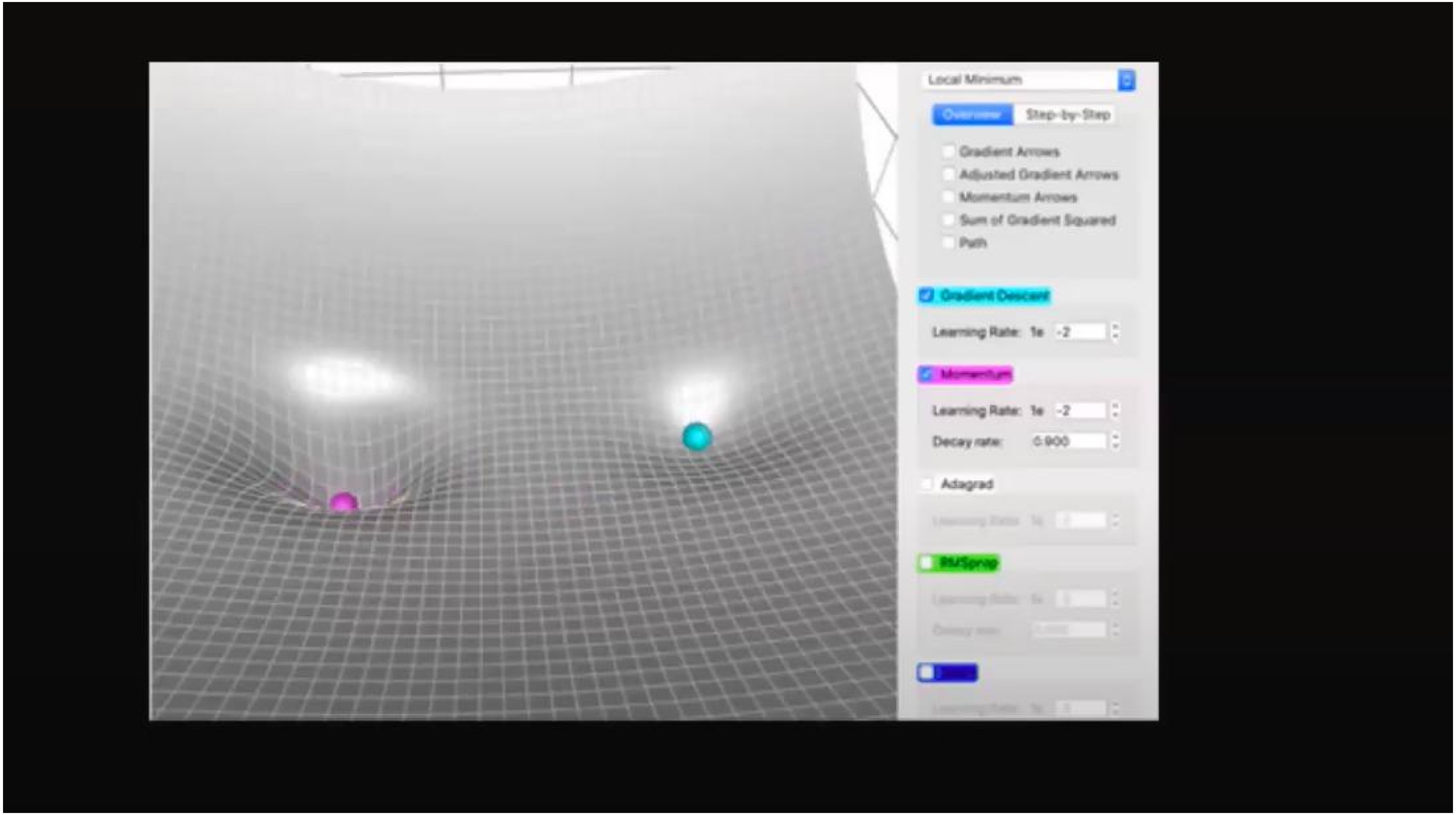

function f(x, y)
$a x^2 + b y^2 + c x y + d x$
$\cos(a x y + b x + c y)$
$\log(a x y + b x + c y + 1)$
$\frac{a x^2 + b y^2 + c x^2 y}{x^2 + y^2}$
constants
a −1
b 1
c 0
d −1
3D and contour plots of $f(x, y) = -x^2 - x + y^2$
608 × 518
cocula.gob.mx
Wolfram Grapher Discount, 60% OFF | cocula.gob.mx
Visit


Small Radius
High Curvature
Large Radius
Low Curvature
Radius of
Curvature
Comparison

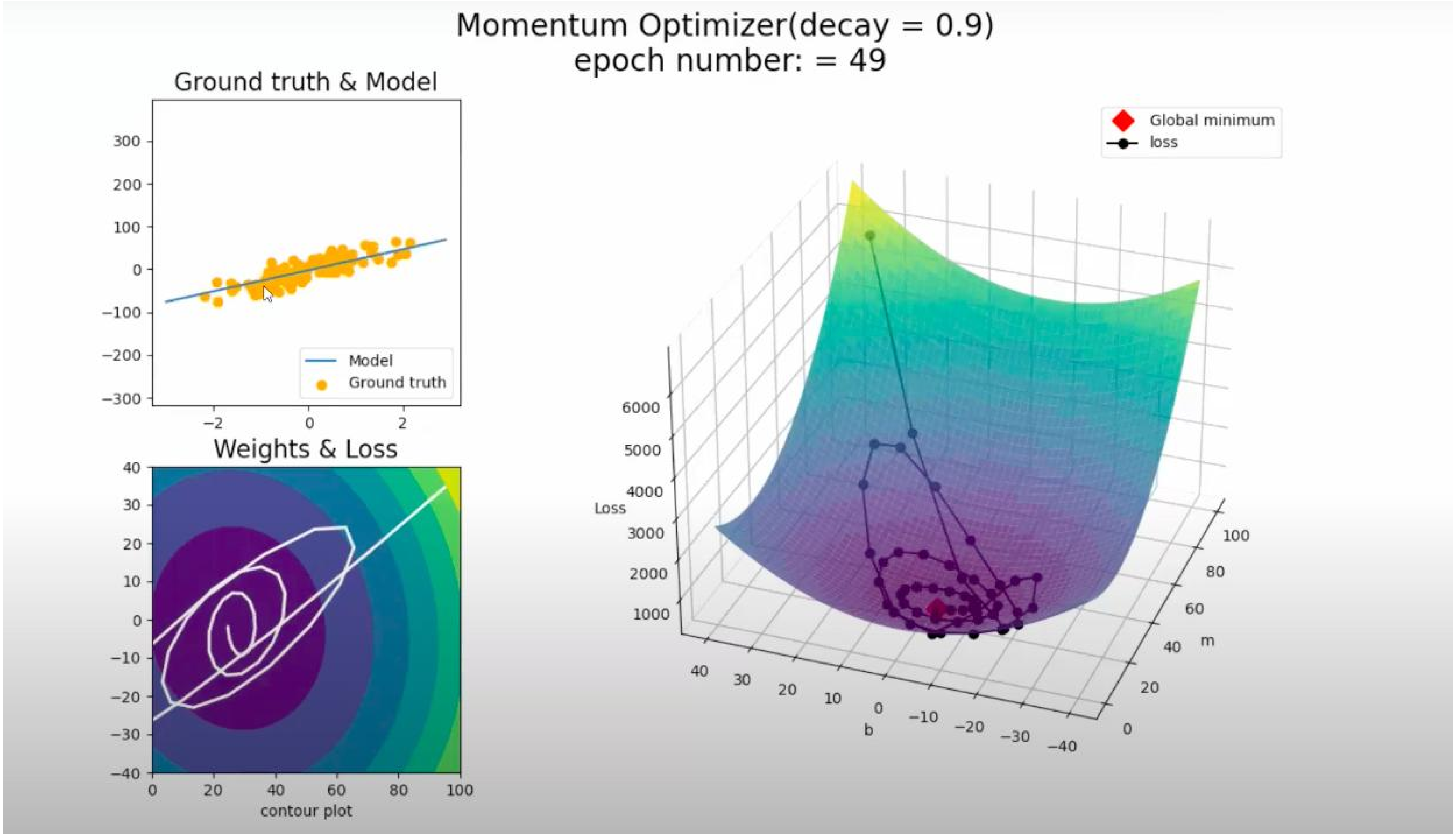


SGD

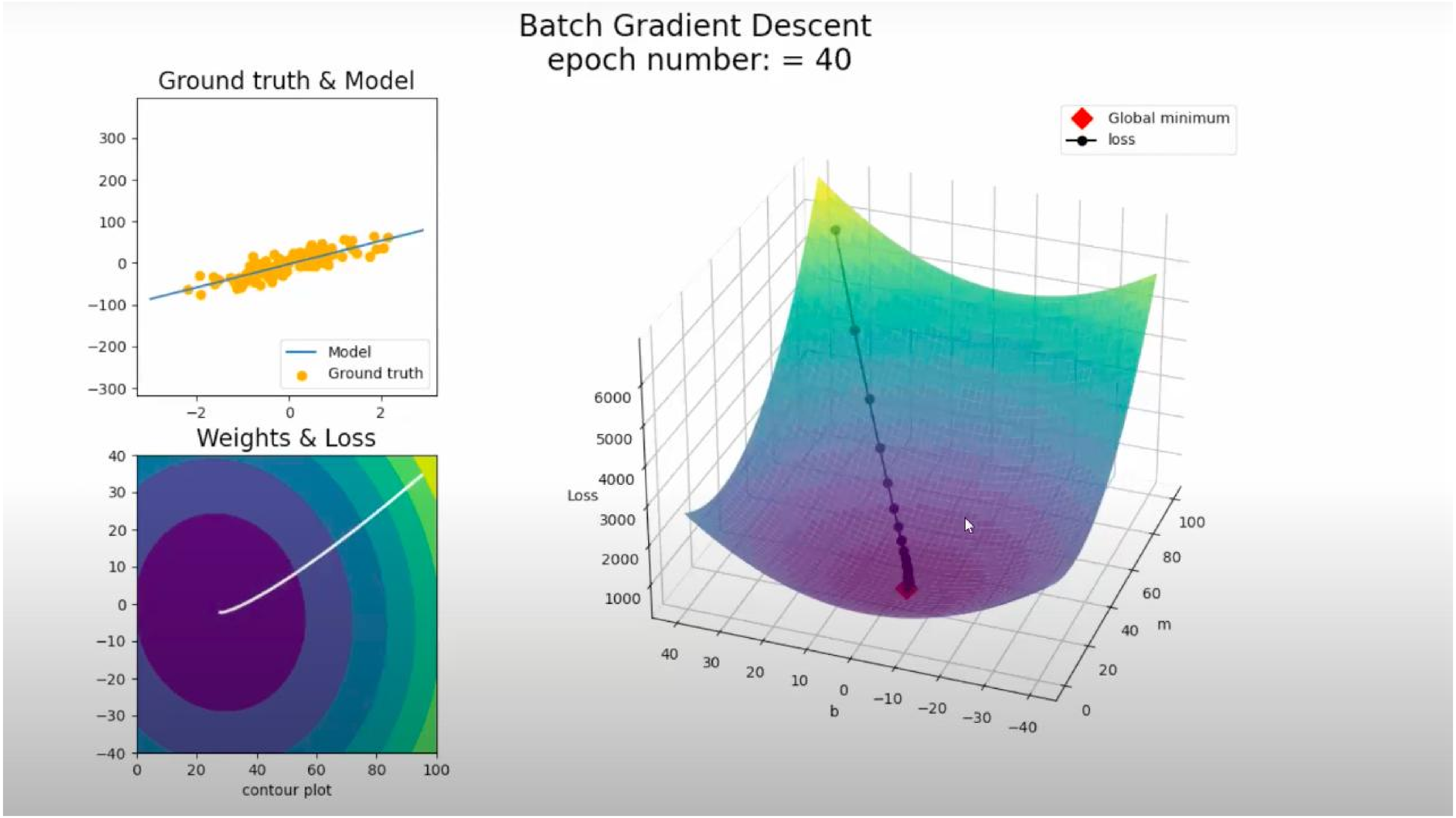
SGD momentum



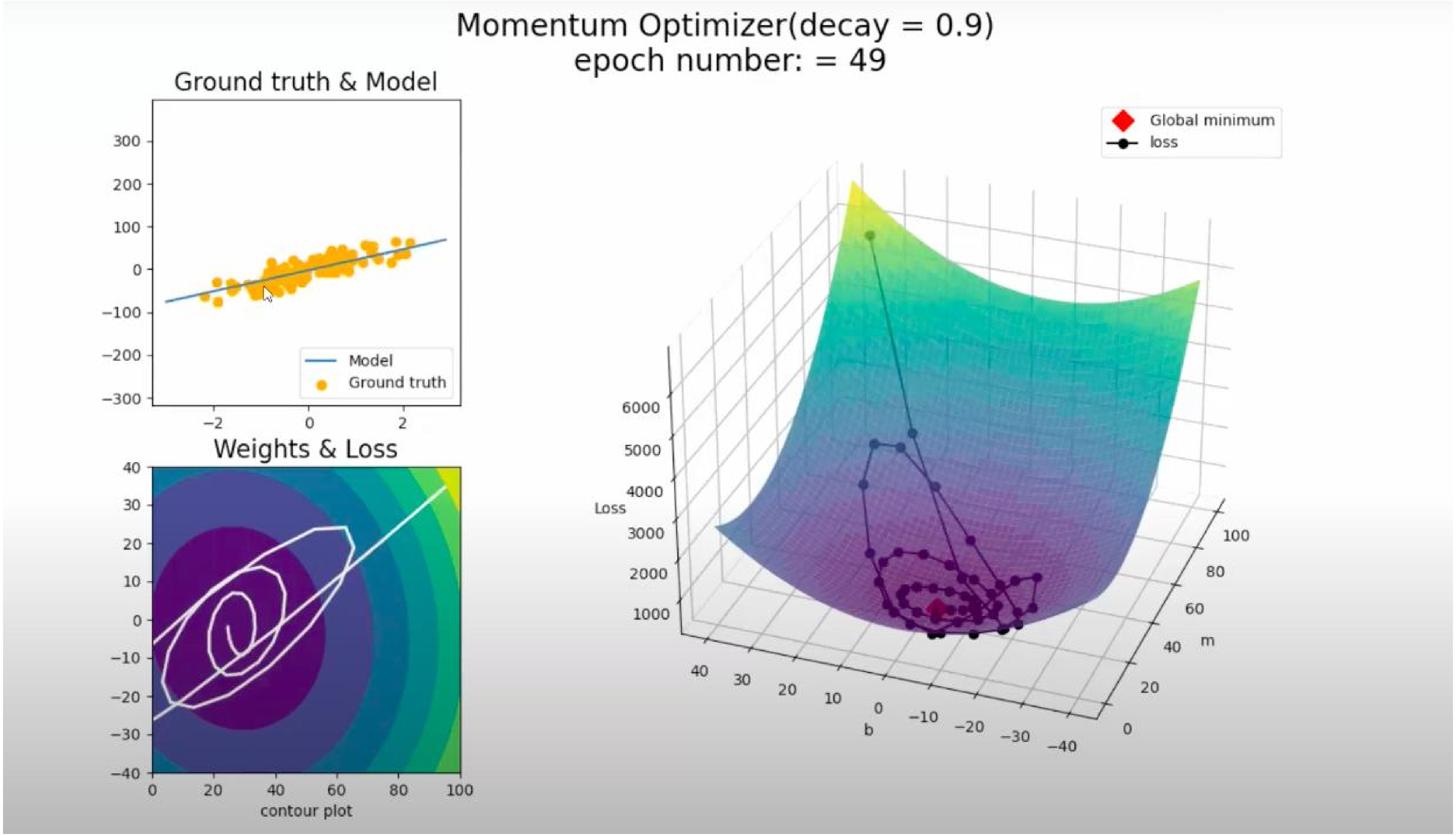
Local Minimum
Overview
Step-by-Step
Gradient Arrows
Adjusted Gradient Arrows
Momentum Arrows
Sum of Gradient Squared
Path
Gradient Descent
Learning Rate: 1e -2
Momentum
Learning Rate: 1e -2
Decay rate: 0.900
Adagrad
RMSprop



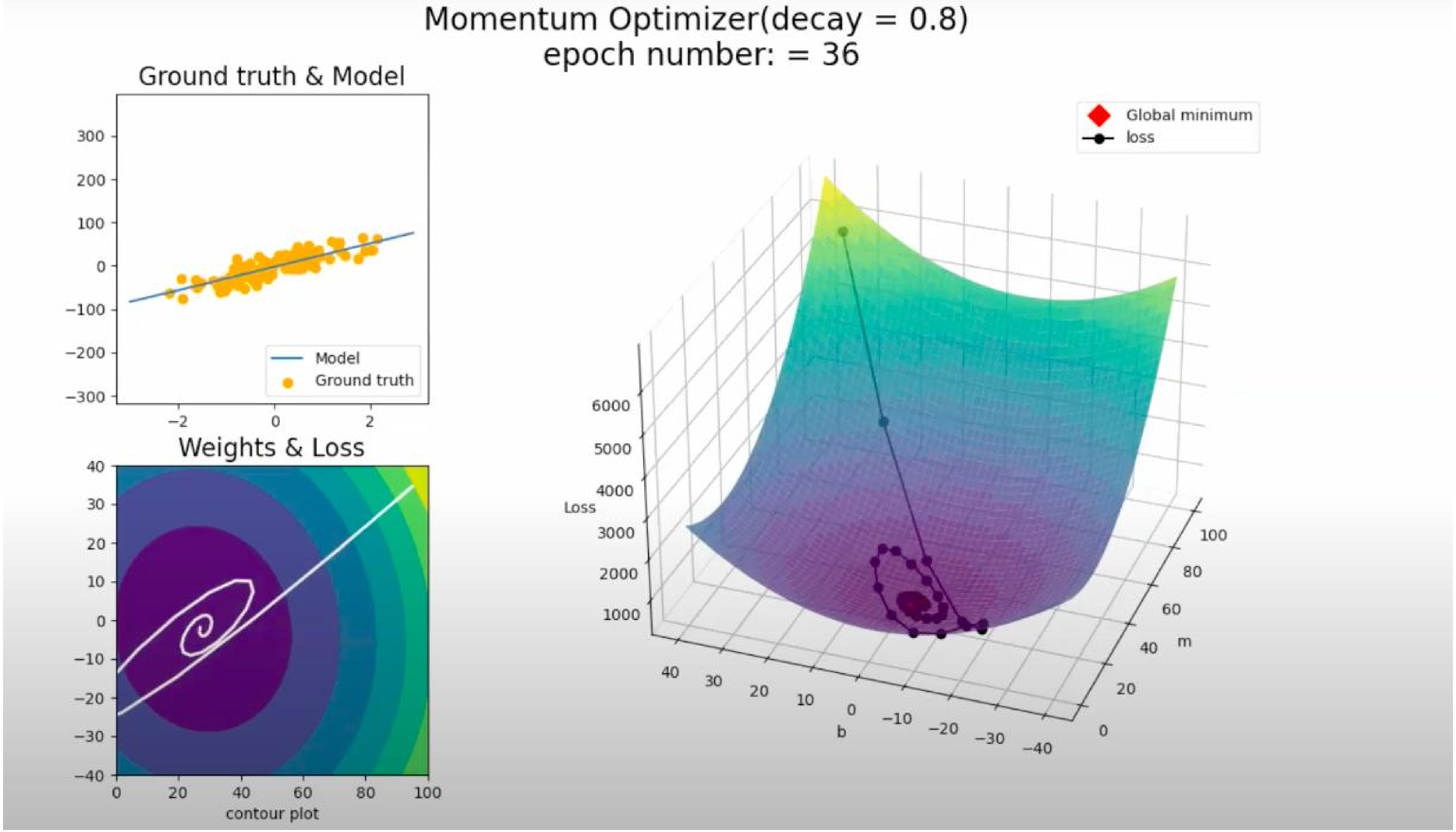
Momentum Optimizer(decay = 0.9)
epoch number: = 49
Ground truth & Model
300
200
100
0
−100
−200
−300
−2
0
2
Model
Ground truth
Weights & Loss
40
30
20
10
0
−10
−20
−30
−40
0
20
40
60
80
100
contour plot
Global minimum
loss
Loss
6000
5000
4000
3000
2000
1000
40
30
20
10
0
−10
−20
−30
−40
b
100
80
60
40
20
0
m



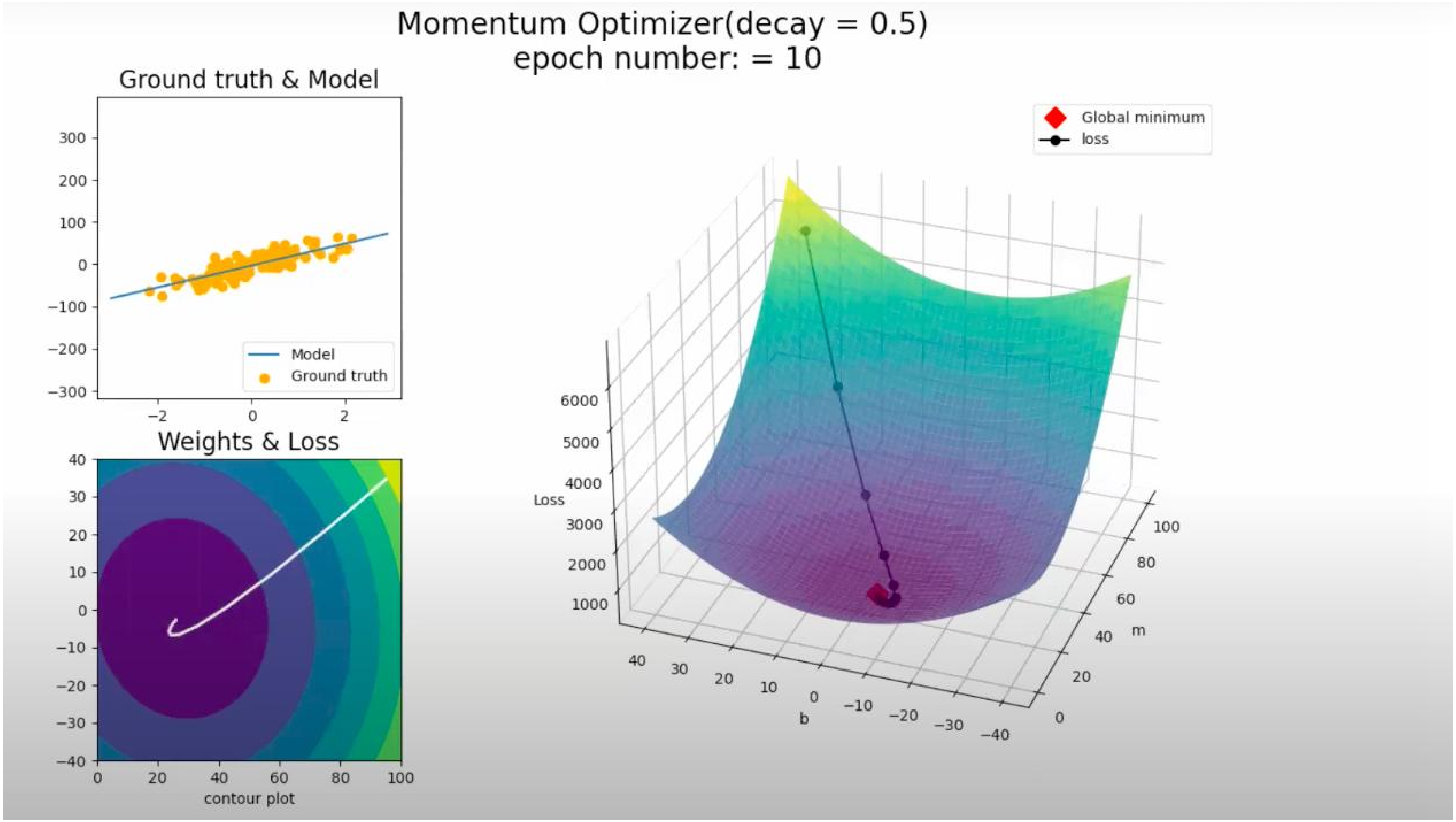
Batch Gradient Descent
epoch number: = 40
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m

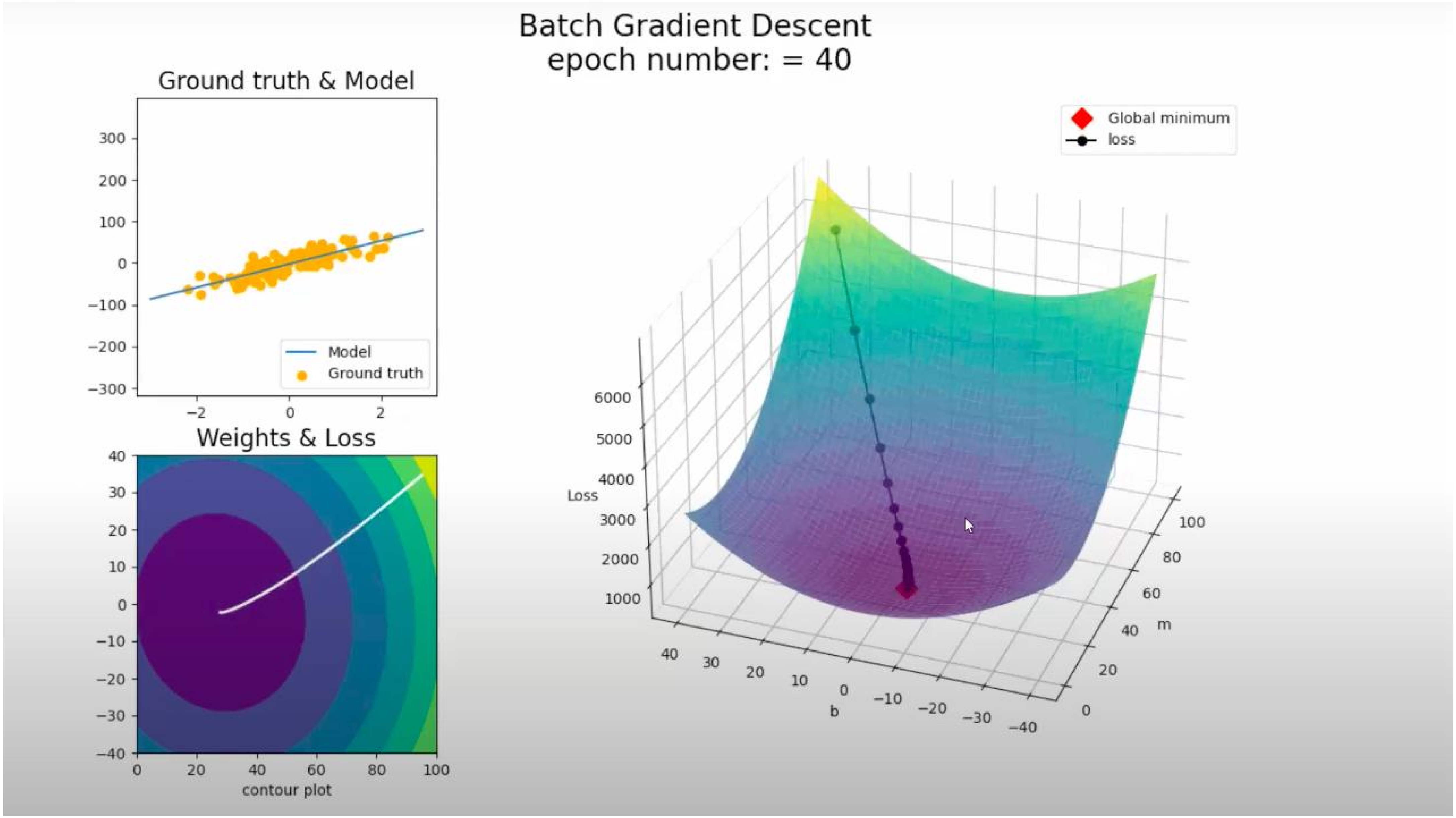


Momentum Optimizer(decay = 0.9)
epoch number: = 49
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m

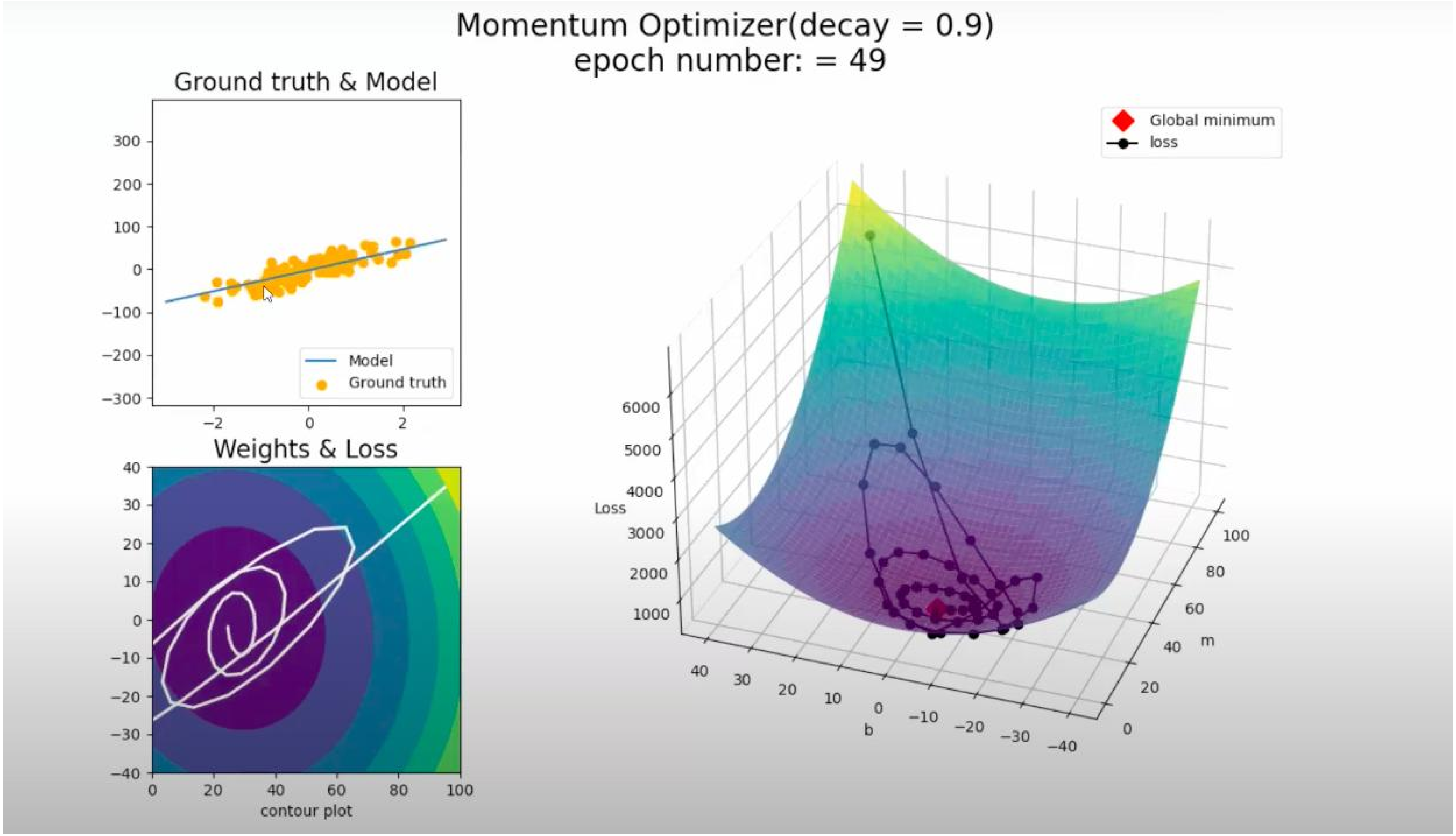


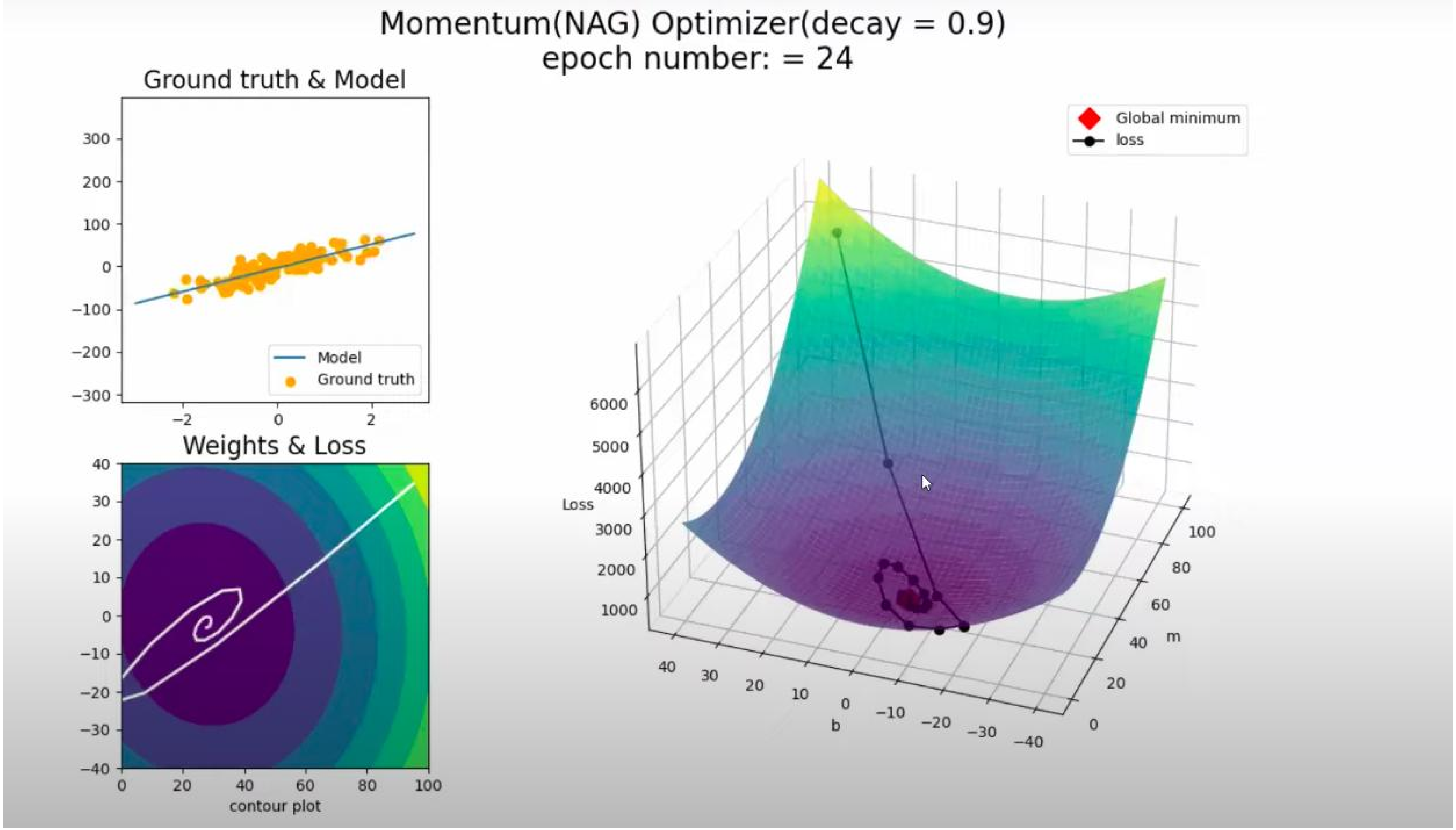
Momentum Optimizer(decay = 0.8)
epoch number: = 36
Ground truth & Model
Model
Ground truth
300
200
100
0
−100
−200
−300
−2
0
2
Weights & Loss
40
30
20
10
0
−10
−20
−30
−40
0
20
40
60
80
100
contour plot
Global minimum
loss
Loss
6000
5000
4000
3000
2000
1000
b
40
30
20
10
0
−10
−20
−30
−40
m
100
80
60
40
20
0


Momentum Optimizer(decay = 0.5)
epoch number: = 10
Ground truth & Model
300
200
100
0
−100
−200
−300
−2
0
2
Model
Ground truth
Weights & Loss
40
30
20
10
0
−10
−20
−30
−40
0
20
40
60
80
100
contour plot
Global minimum
loss
6000
5000
4000
3000
2000
1000
Loss
40
30
20
10
0
−10
−20
−30
−40
b
0
20
40
60
80
100
m



Batch Gradient Descent
epoch number: = 40
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m


Momentum Optimizer(decay = 0.9)
epoch number: = 49
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m



Momentum(NAG) Optimizer(decay = 0.9)
epoch number: = 24
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m

Keras Code

24 July 2022 13:18

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD", **kwargs
)
```

SGD

Keras Code

24 July 2022 13:18

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD", **kwargs
)
```

SGD

Momentum

momentum = 0.9

nesterov = False

Keras Code

24 July 2022 13:18

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD", **kwargs
)
```

SGD ✓

Momentum

momentum = 0.9

nesterov = False

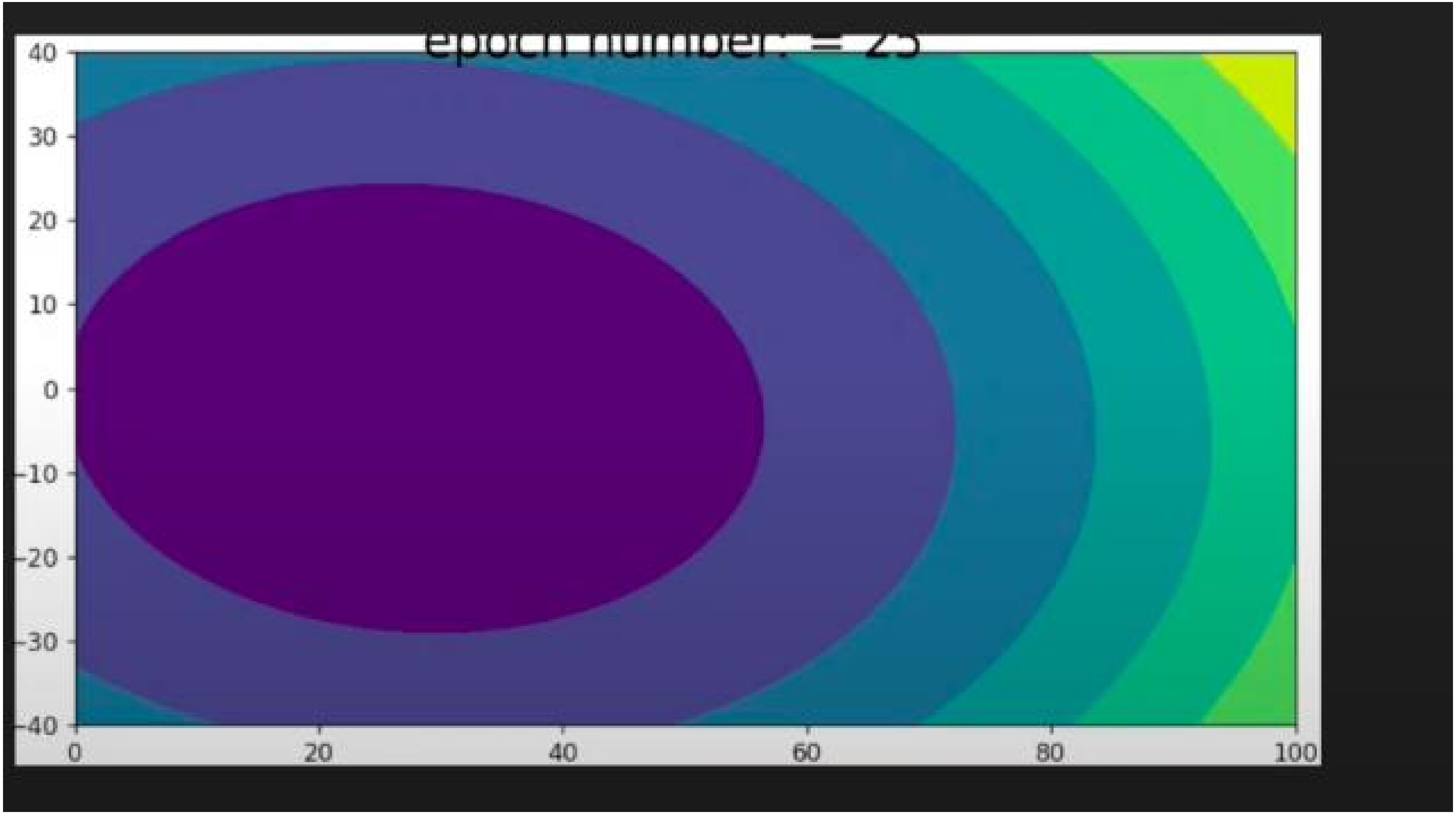
✓

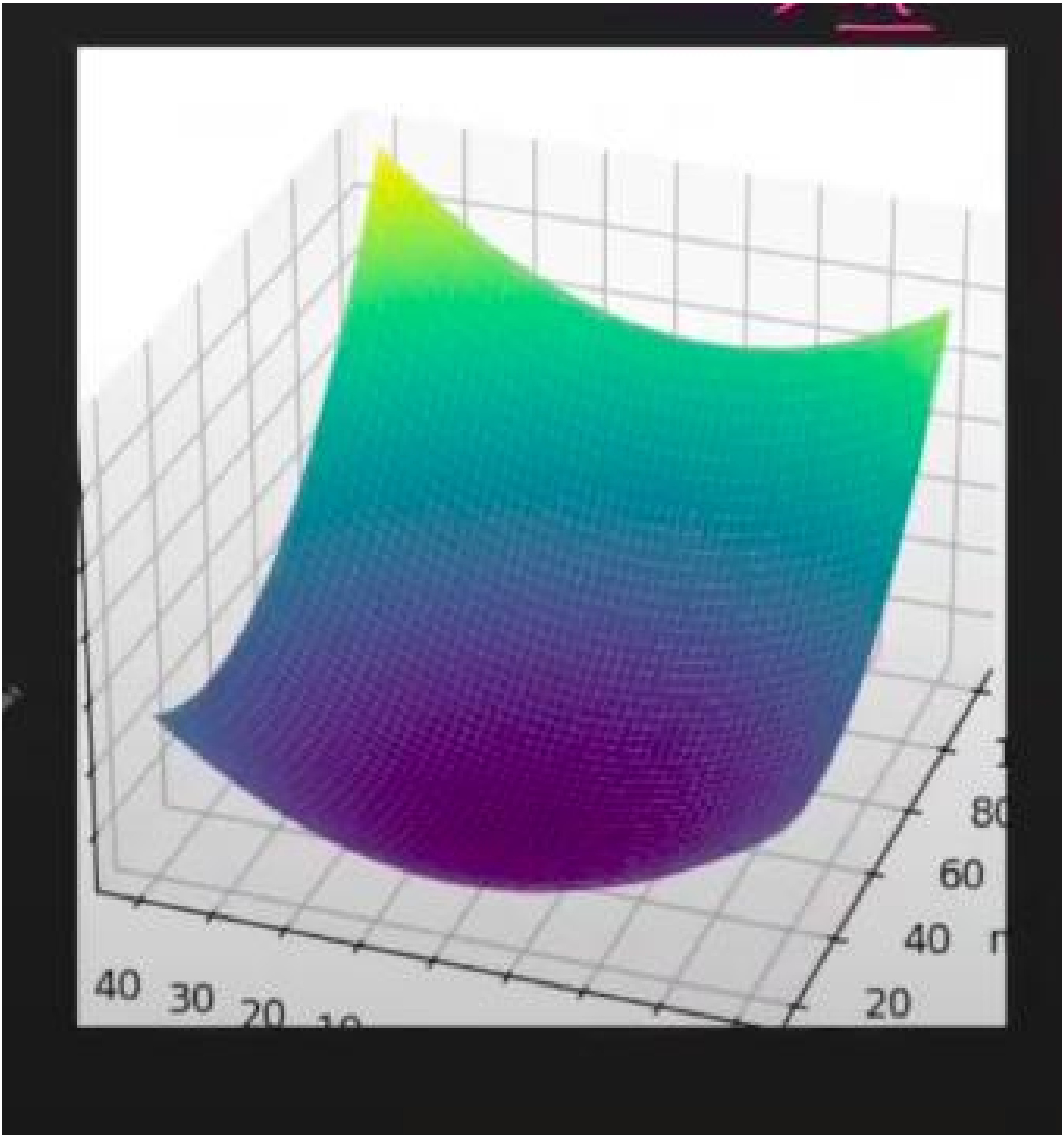
NAG

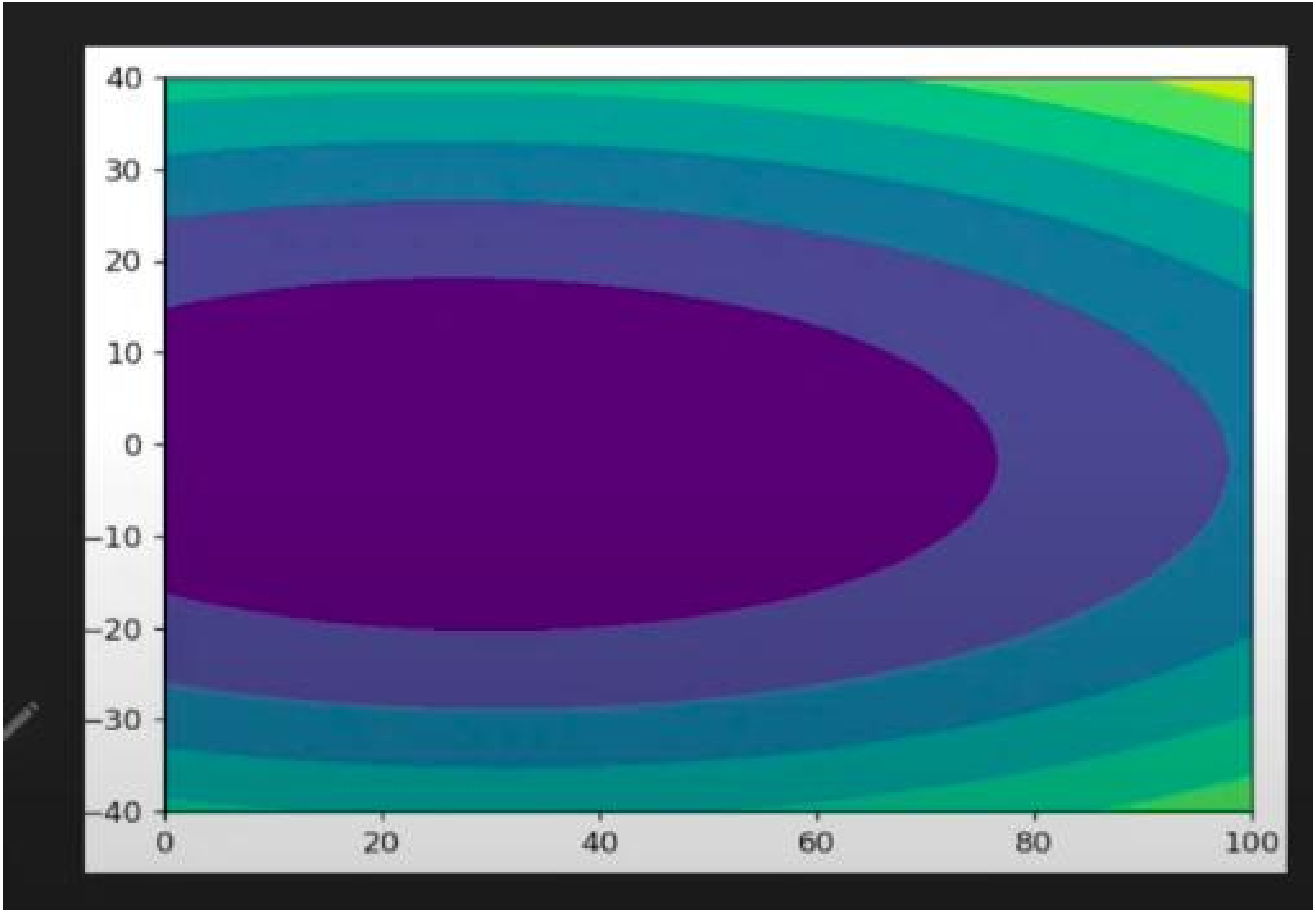
momentum = 0.9

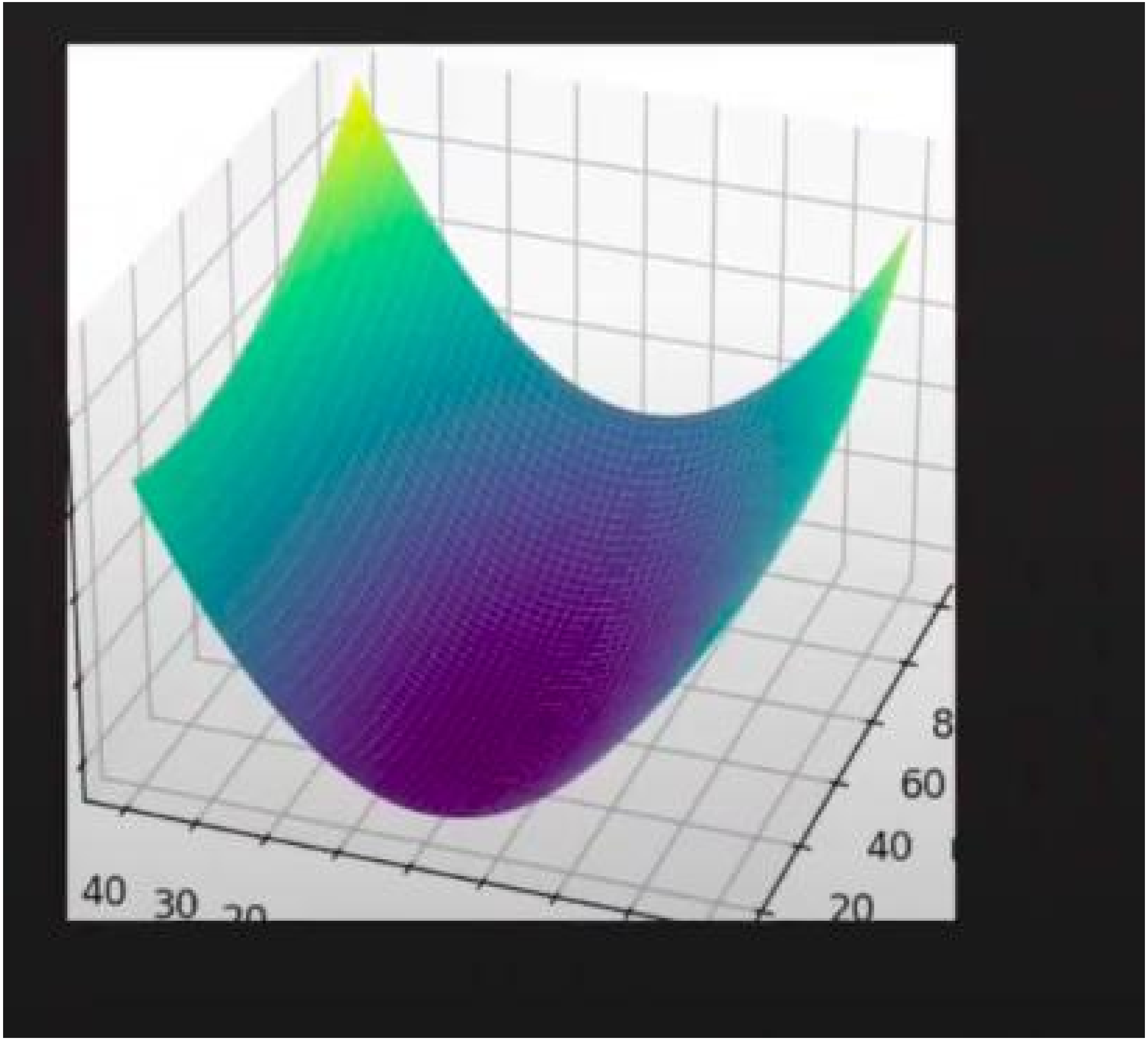
nesterov = True

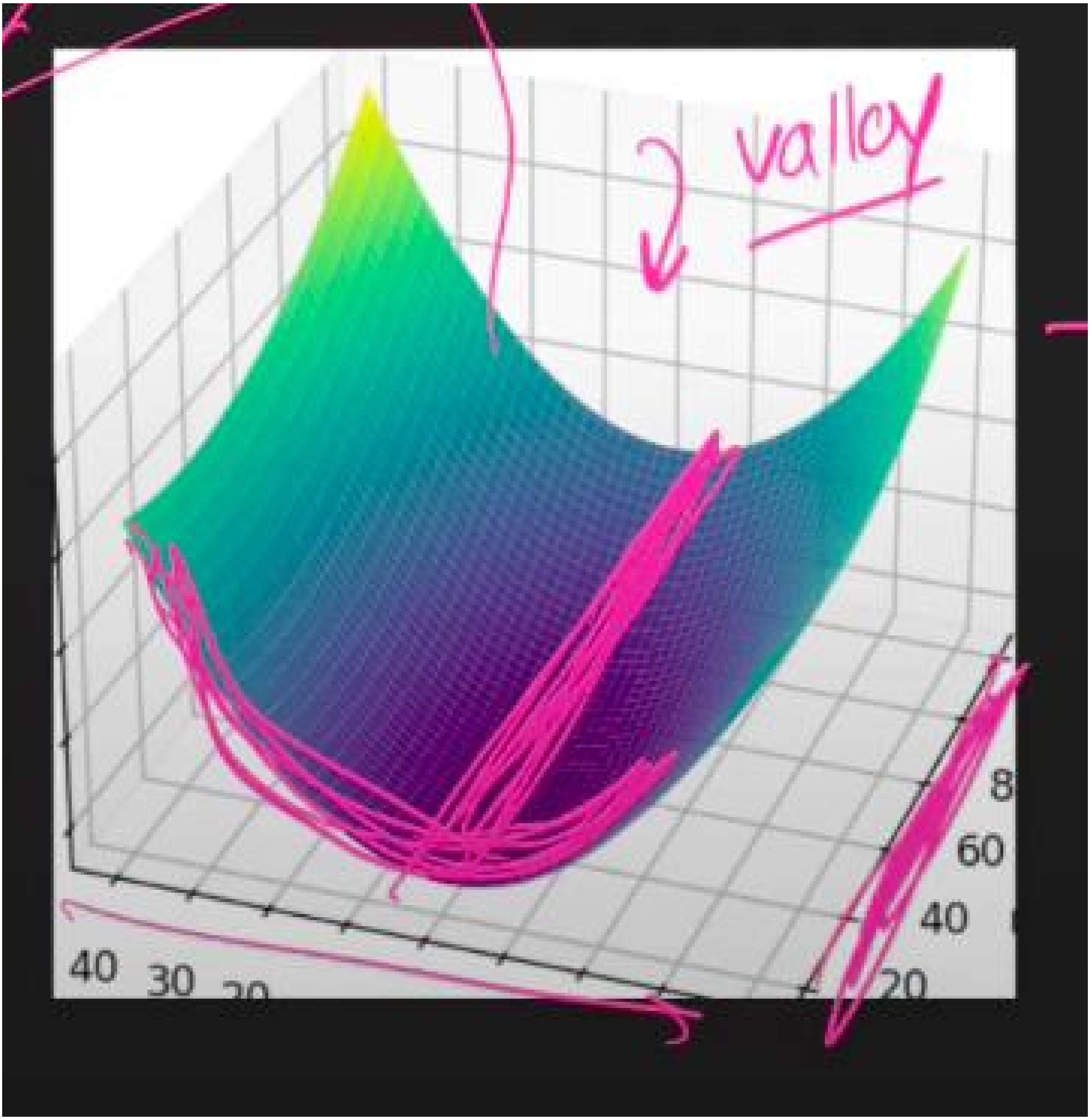
✓

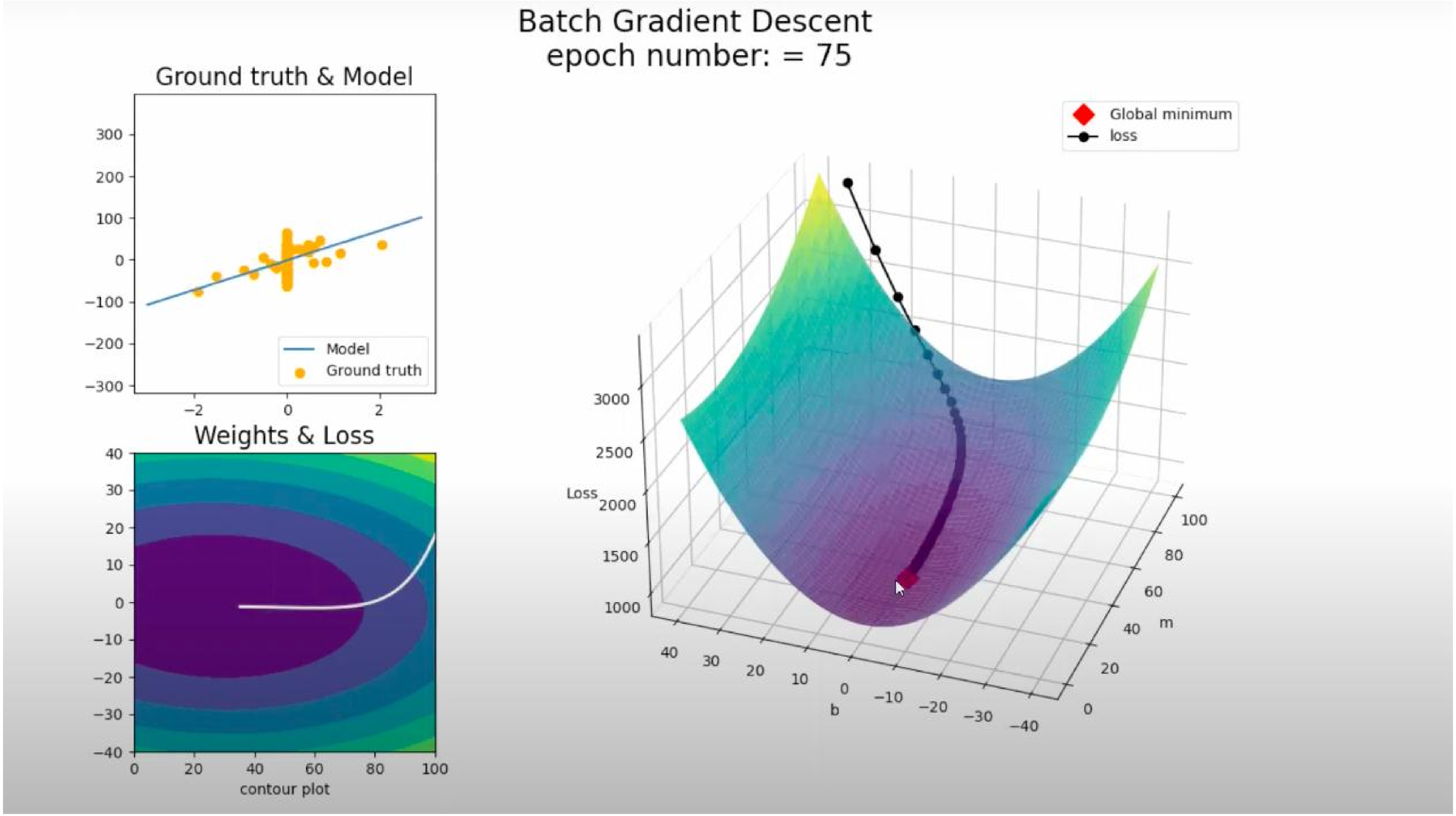

epoch number: = 25
40
30
20
10
0
-10
-20
-30
-40
0
20
40
60
80
100


40
30
20
80
60
40
20

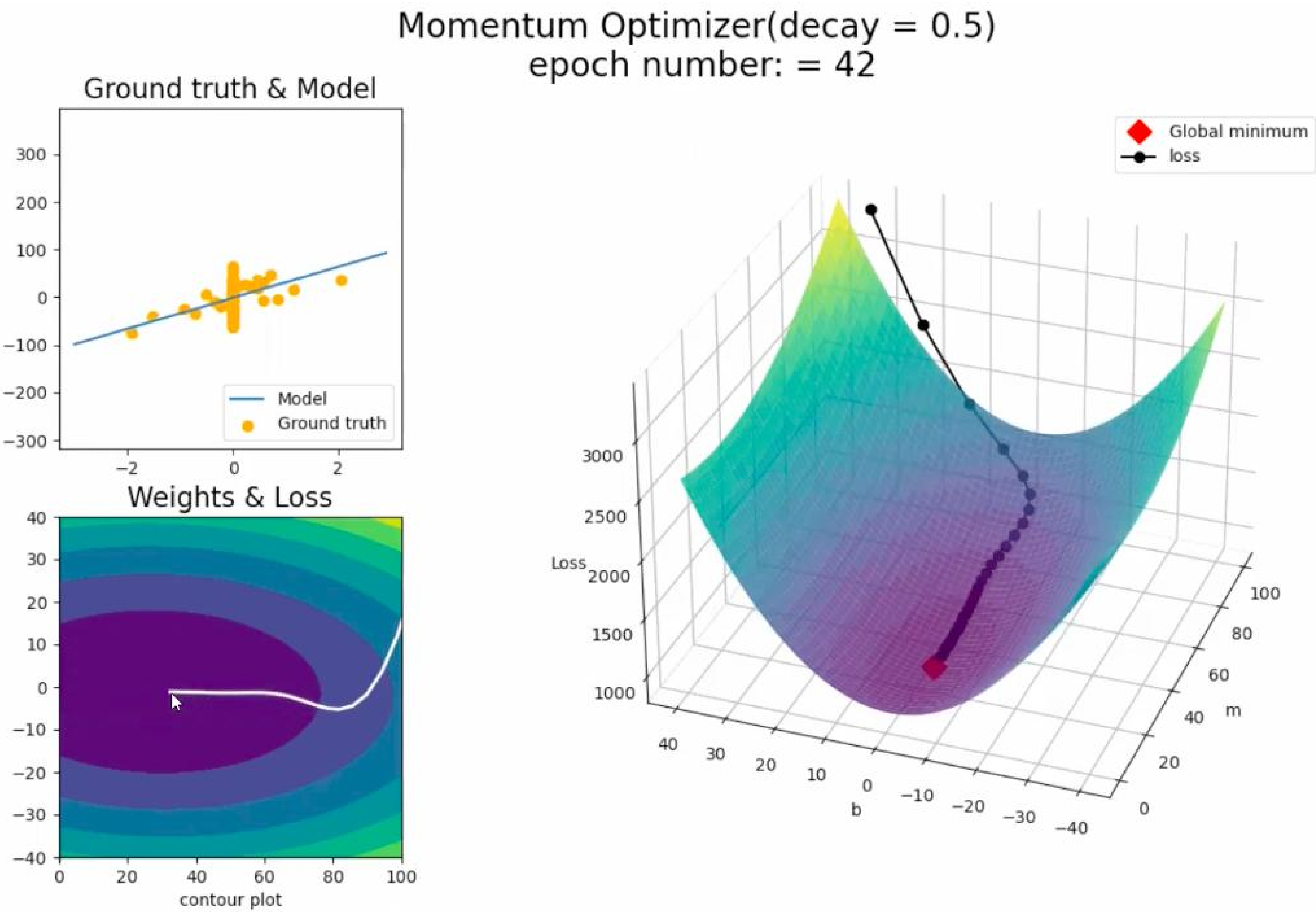

40
30
20
10
0
−10
−20
−30
−40
0
20
40
60
80
100


40
30
60
40
20


valley
40
30
80
60
40
20

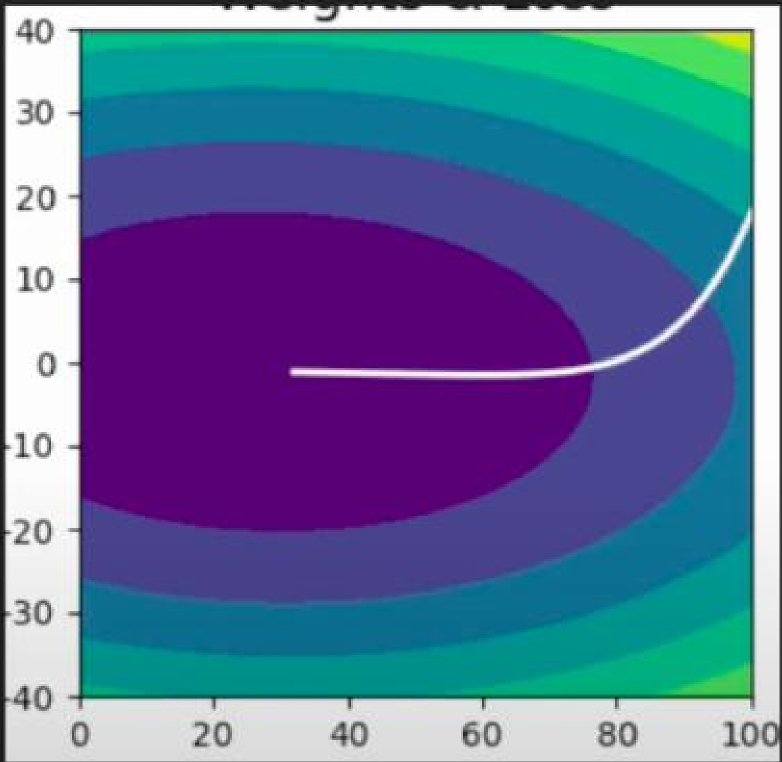


Batch Gradient Descent
epoch number: = 75
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m

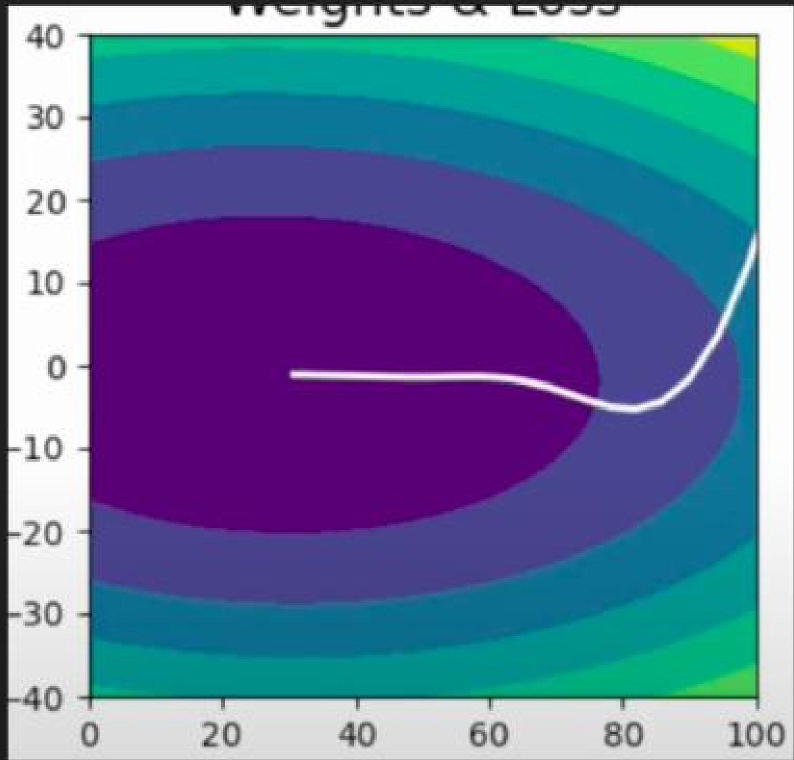

Momentum Optimizer(decay = 0.5)
epoch number: = 42
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m

How optimizers behave(Why?)

02 August 2022 18:44

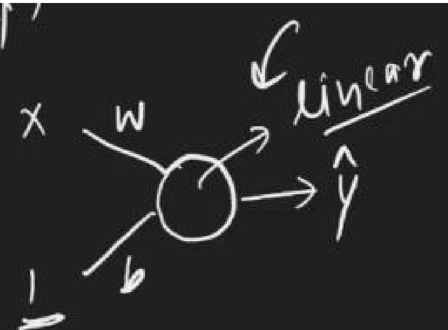


Batch GD



Momentum

$(y-\hat{y})$



$$\boxed{\frac{\partial L}{\partial w}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w}$$

$$= -2(y-\hat{y})\boxed{x}$$ add

$$\frac{\partial L}{\partial b} = \boxed{-2(y-\hat{y}) * 1}$$ bignum

BGD → sparspe

for i in epochs:

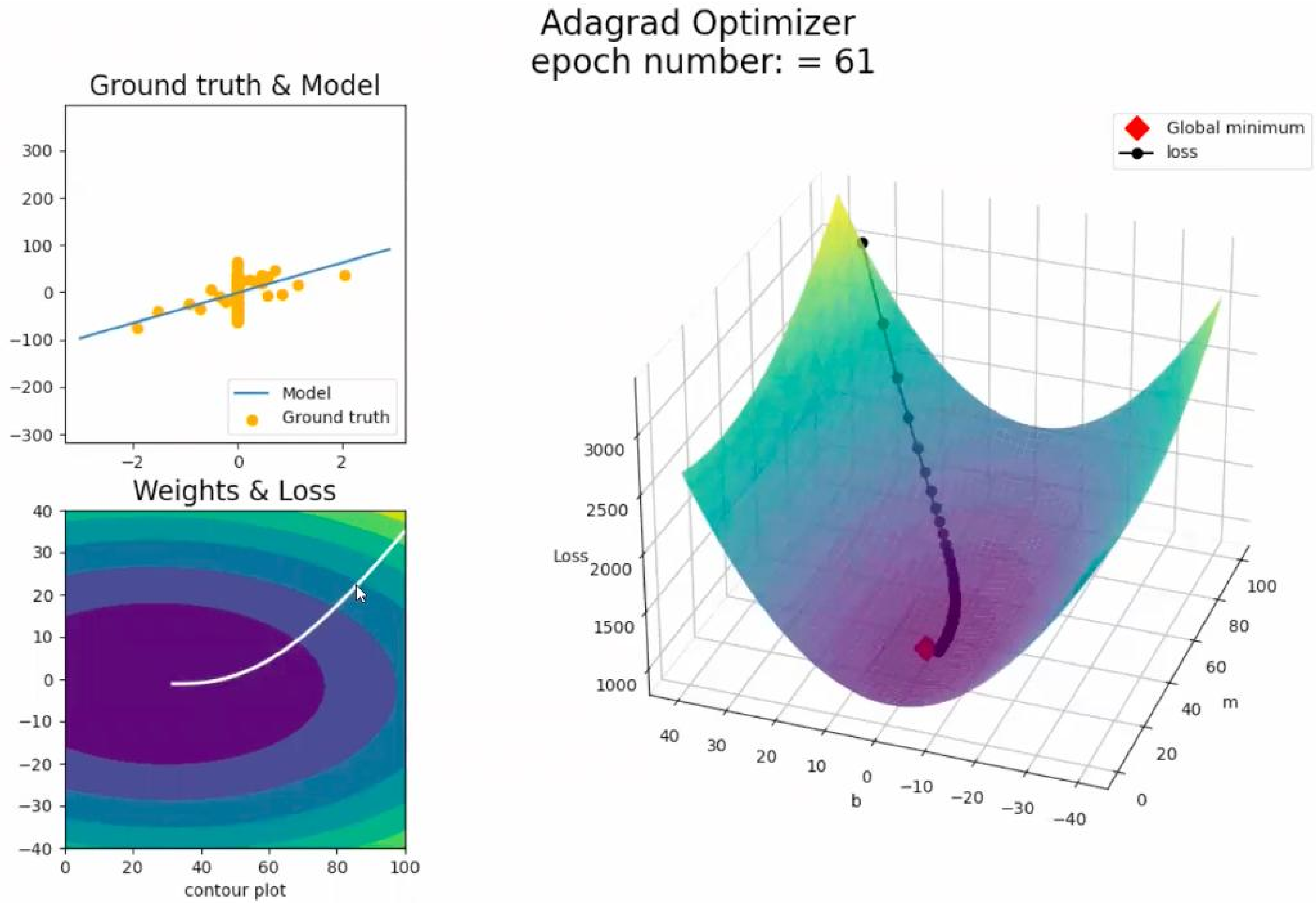
$$W = W - \eta \boxed{\frac{\partial L}{\partial w}}$$

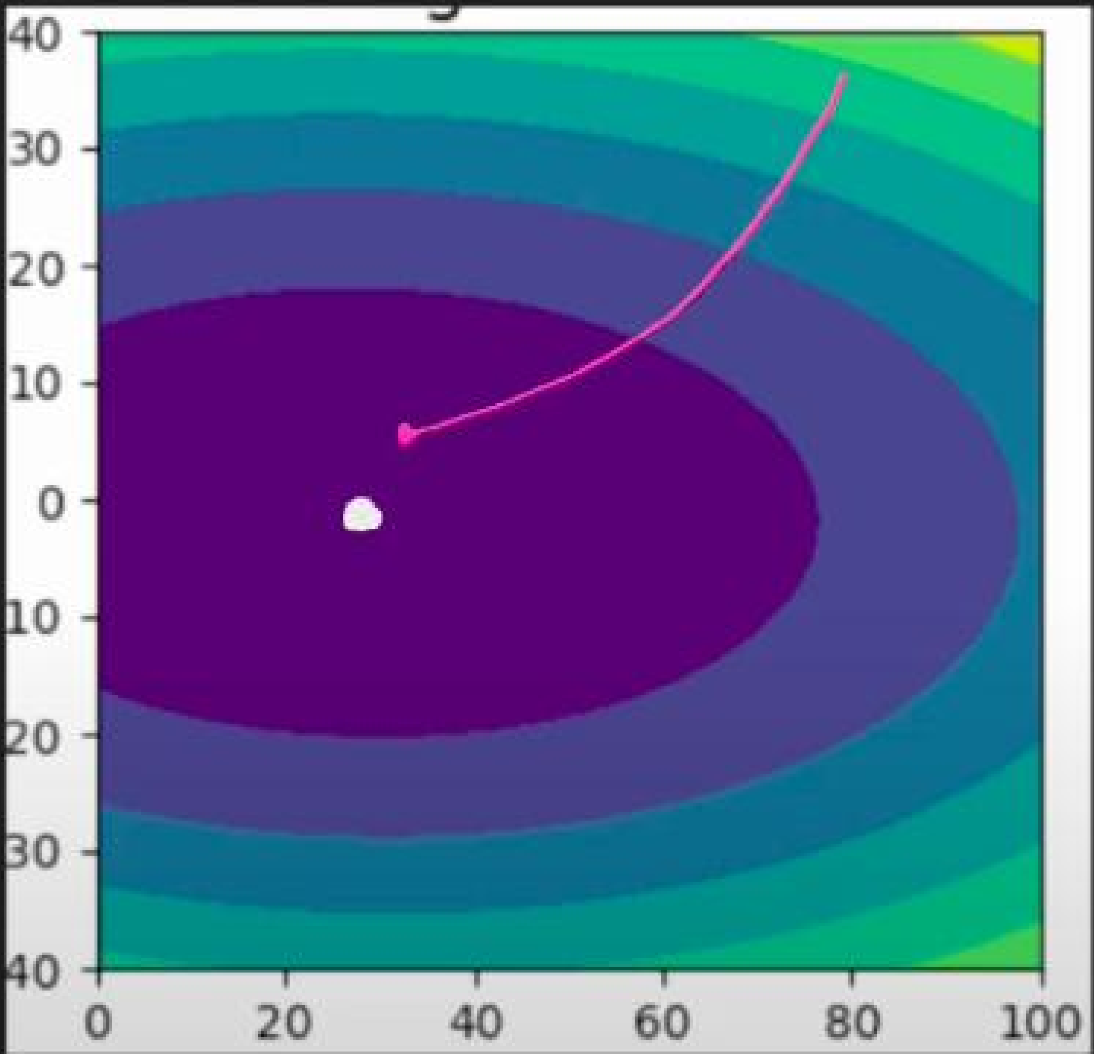
$$b = b - \eta \boxed{\frac{\partial L}{\partial b}}$$ → big update in

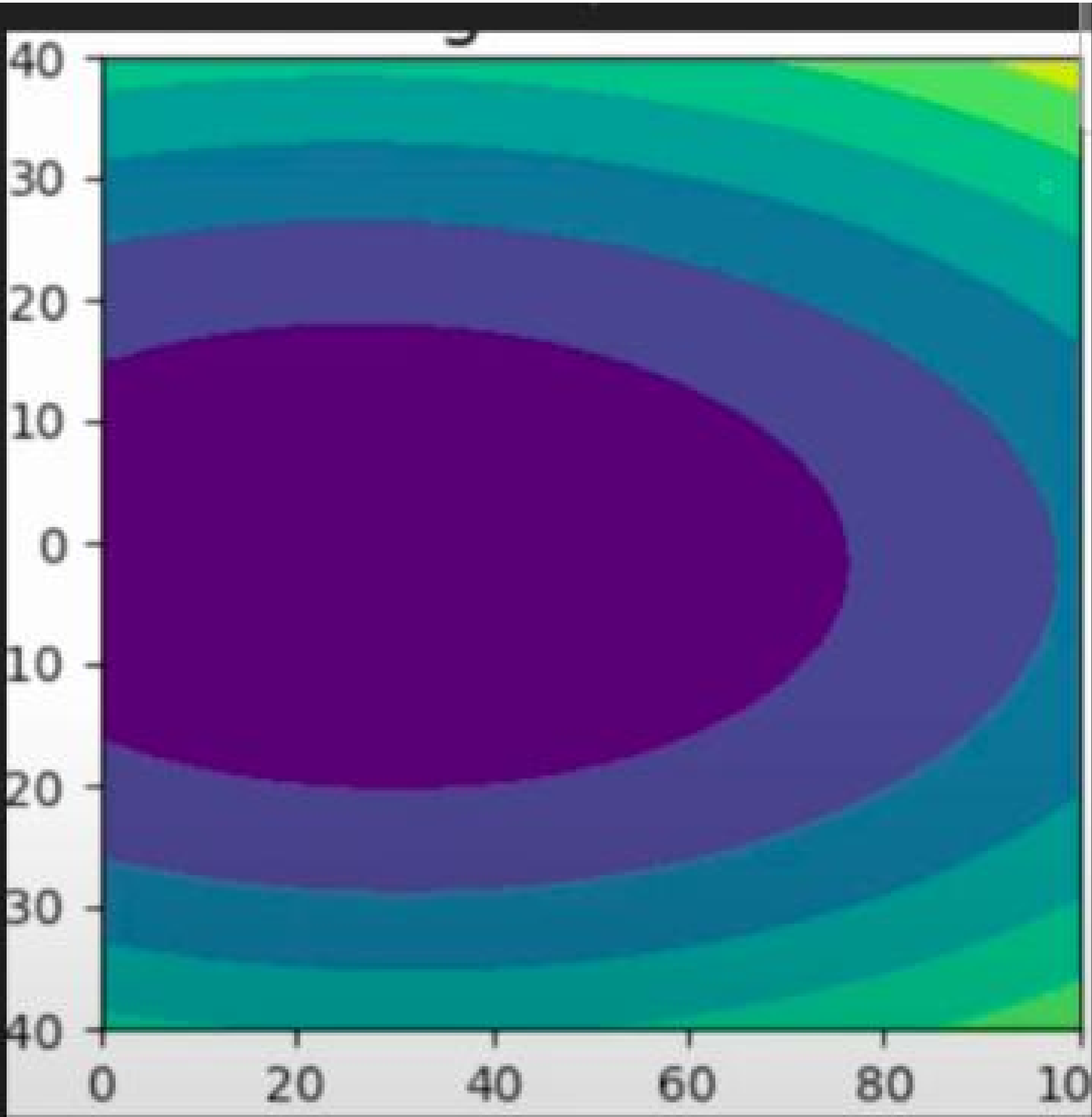
$\boxed{100 \text{ rows}}$ → small number

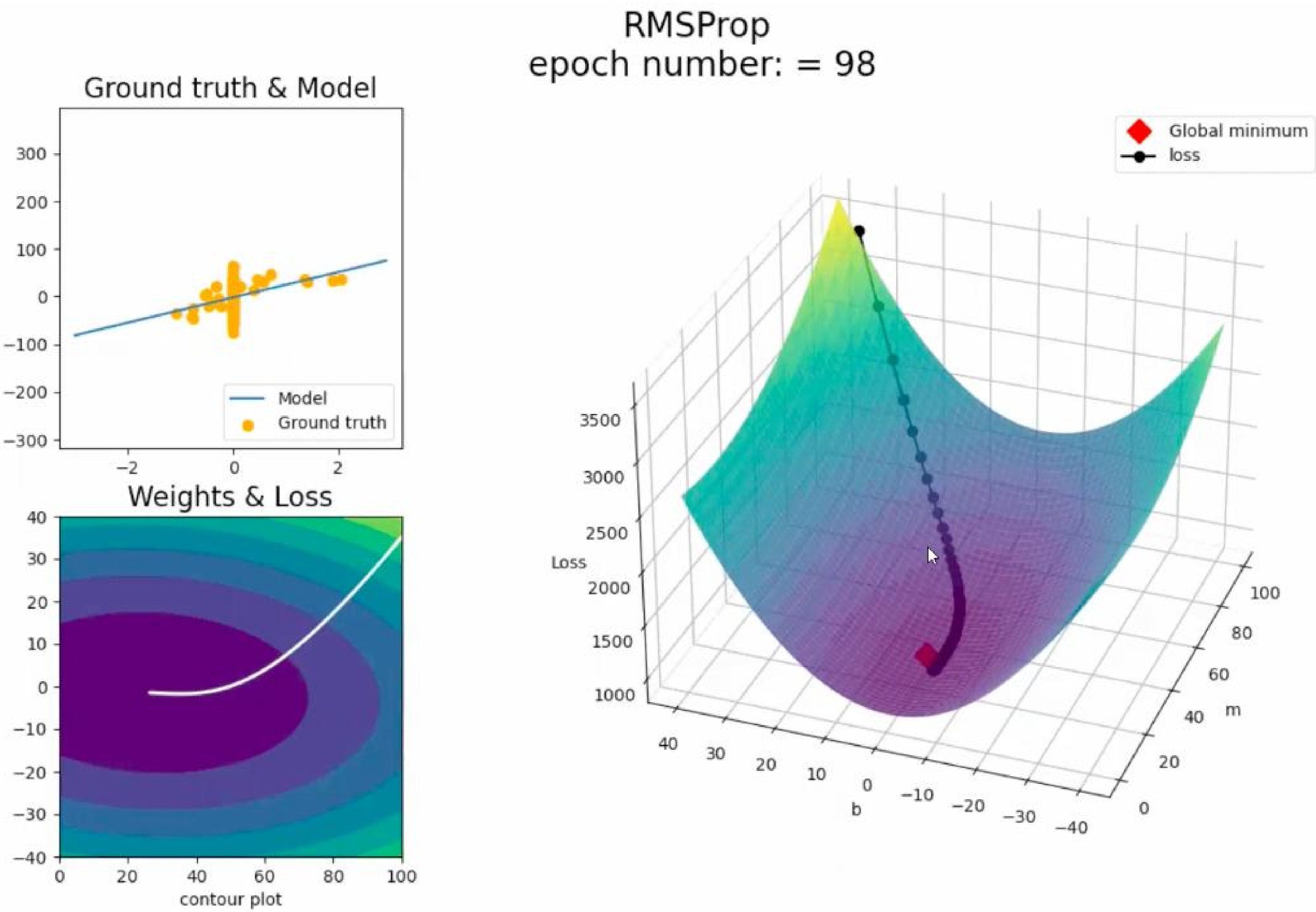
0   x sparse

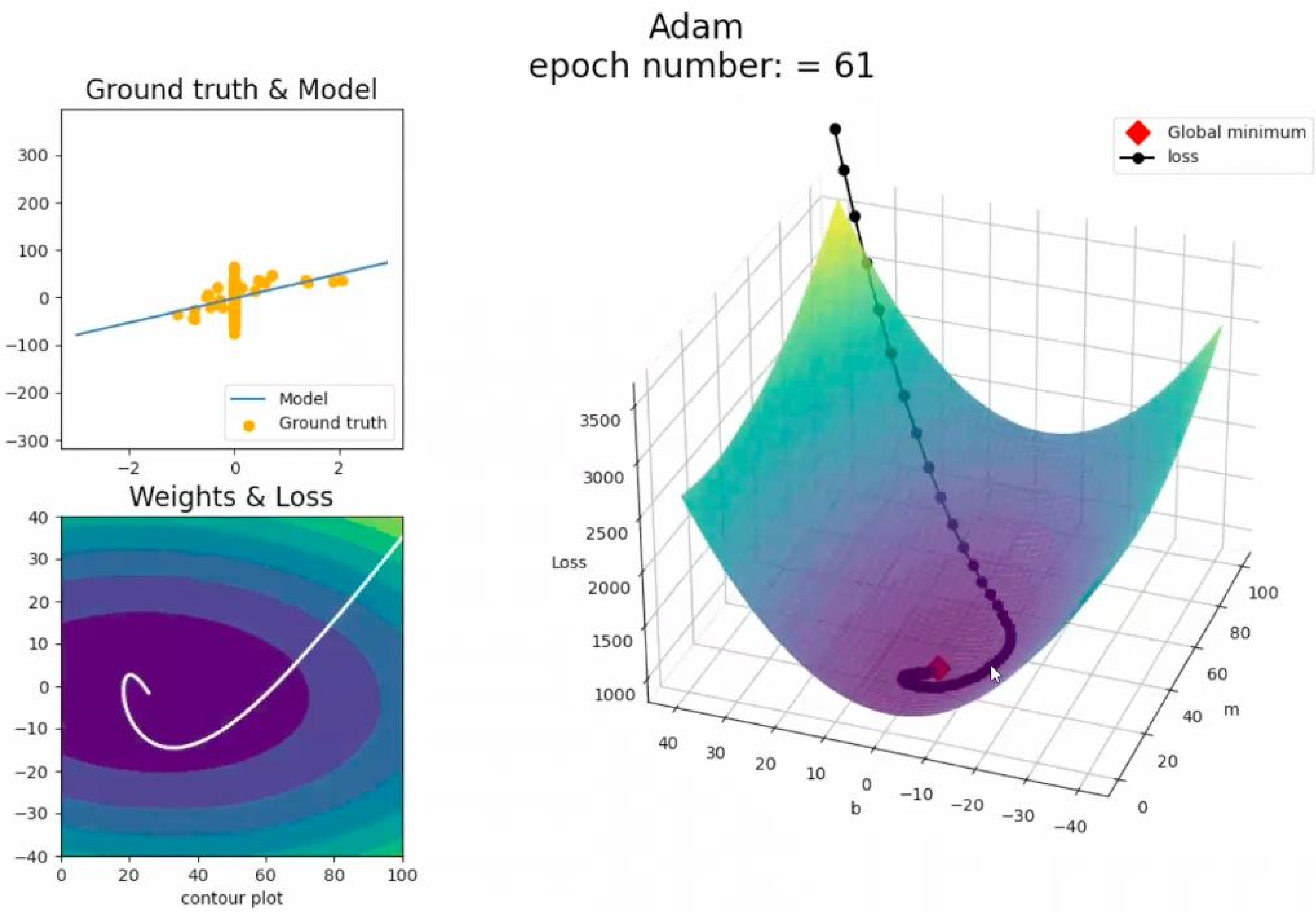
small in every epoch


Adagrad Optimizer
epoch number: = 61
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m


40
30
20
10
0
10
20
30
40
0
20
40
60
80
100


40
30
20
10
0
10
20
30
40
0
20
40
60
80
10


RMSProp
epoch number: = 98
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m



Adam
epoch number: = 61
Ground truth & Model
Model
Ground truth
Weights & Loss
contour plot
Global minimum
loss
Loss
b
m