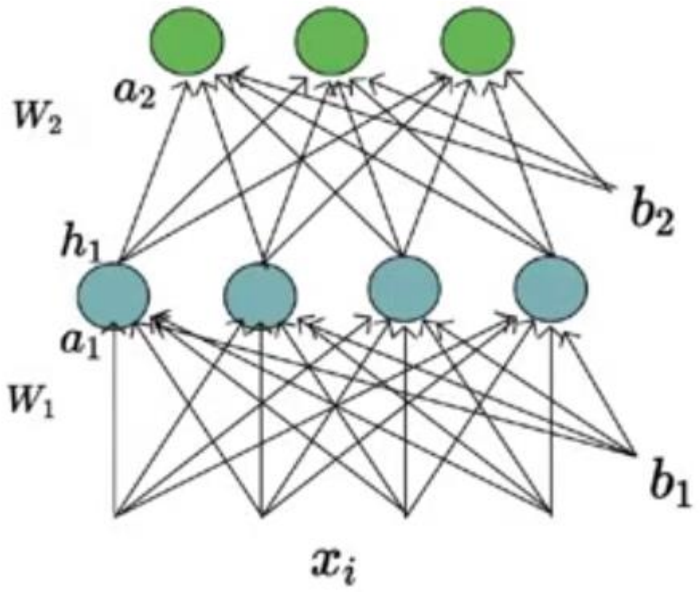
Deep Learning:
Feed Forward Neural Network

राष्ट्रीय प्रौद्योगिकी संस्थान सिक्किम
NATIONAL INSTITUTE OF TECHNOLOGY SIKKIM

Course Instructor:
Dr. Bam Bahadur Sinha
Assistant Professor
Computer Science & Engineering
National Institute of Technology
Sikkim

What is the **LOSS FUNCTION** that you use for a multi-class classification problem?



$$b = [\ 0 \quad 0\ ]$$

$$W_1 = \begin{bmatrix} 0.1 & 0.3 & 0.8 & -0.4 \\ -0.3 & -0.2 & 0.5 & 0.5 \\ -0.3 & 0.1 & 0.5 & 0.4 \\ 0.2 & 0.5 & -0.9 & 0.7 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 0.3 & 0.8 & -0.2 & -0.4 \\ 0.5 & -0.2 & -0.3 & 0.5 \\ 0.3 & 0.1 & 0.6 & 0.6 \end{bmatrix}$$

$$x = [\ 0.3 \quad -0.4 \quad 0.6 \quad 0.2\ ] \quad y = [\ 0 \quad 0 \quad 1\ ]$$

Output :

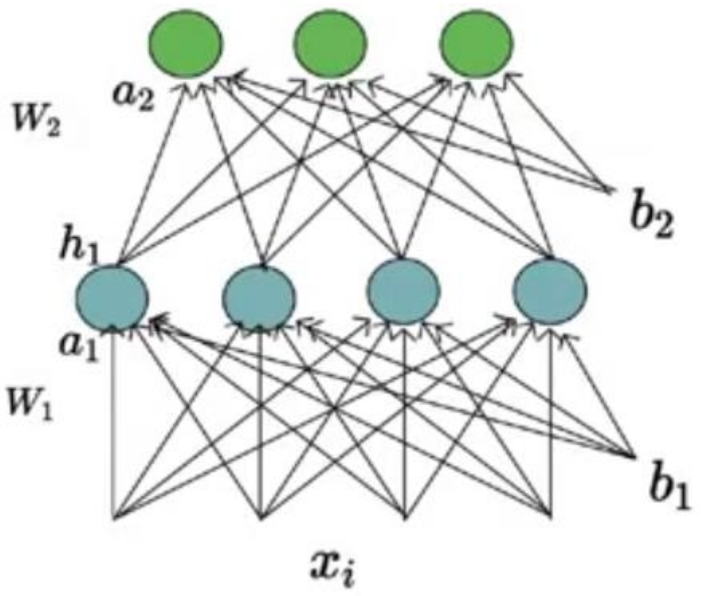
$$\begin{aligned} a_1 &= W_1 * x + b_1 &&= [\ 0.31 \quad 0.39 \quad 0.25 \quad -0.54\ ] \\ h_1 &= sigmoid(a_1) &&= [\ 0.58 \quad 0.60 \quad 0.56 \quad 0.37\ ] \\ a_2 &= W_2 * h_1 + b_2 &&= [\ 0.39 \quad 0.18 \quad 0.79\ ] \\ \hat{y} &= softmax(a_2) &&= [\ 0.3024 \quad 0.2462 \quad 0.4514\ ] \end{aligned}$$

Cross Entropy Loss:

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

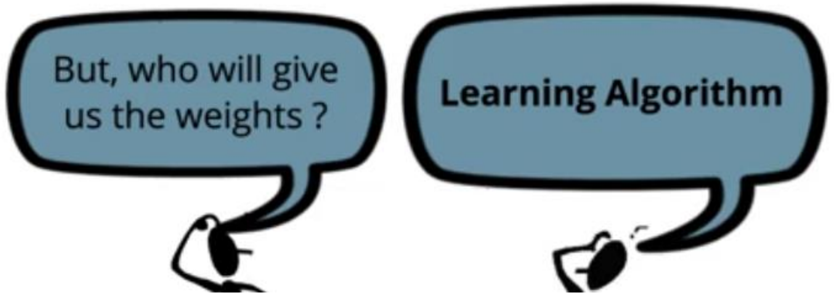
$$\begin{aligned} L(\Theta) &= -1 * \log(0.4514) \\ &= 0.7954 \end{aligned}$$

Multi-class classification problem



Given weights, we know how to compute the model's output for a given input

Given weights, we know how to compute the model's loss for a given input



Learning Algorithm

Initialise w, b

Iterate over data:

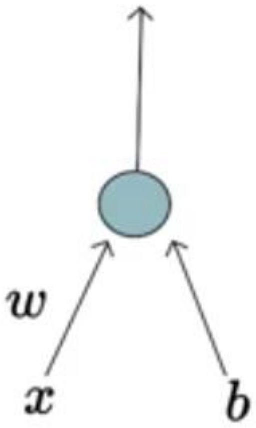
$compute\ \hat{y}$

$compute\ \mathscr{L}(w, b)$

$w = w - \eta \Delta w$

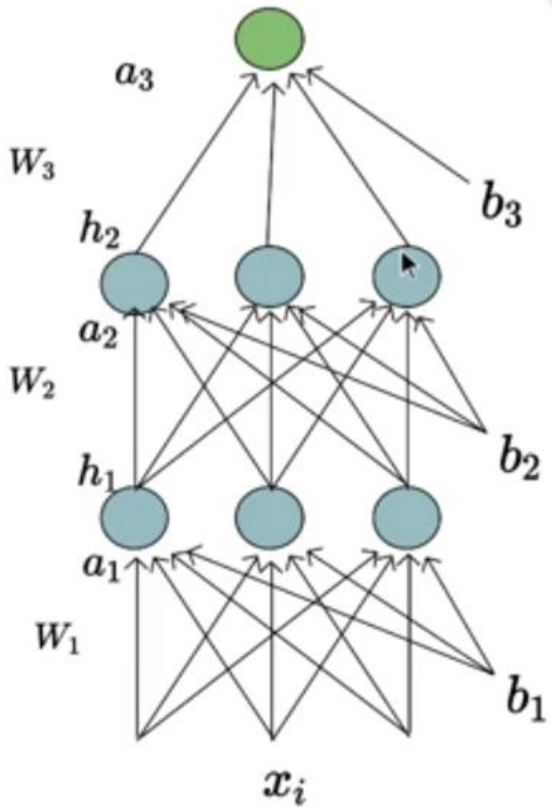
$b = b - \eta \Delta b$

till satisfied



$$Earlier : w, b$$

$$Now : w_{111}, w_{112}, \ldots$$



$$Earlier : L(w, b)$$

$$Now : L(w_{111}, w_{112}, \ldots)$$

Gradient Descent Learning Algorithm for Multi-class Classification problem

Initialise ~~w, b~~ w_{111}, w_{112}

Iterate over data:

$compute\ \hat{y}$

$compute\ \mathscr{L}(w, b)$

$$w_{111} = w_{111} - \eta \Delta w_{111}$$

$$w_{112} = w_{112} - \eta \Delta w_{112}$$

....

$$w_{313} = w_{313} - \eta \Delta w_{313}$$

till satisfied

How do you check the performance of a Deep Neural Network?

(Binary Classification)

Indian Liver Patient Records * - whether person needs to be diagnosed or not ?

Test Data

Age	Albumin	T_Bilirubin
65	3.3	0.7
62	3.2	10.9
20	4	1.1
84	3.2	0.7

. . .

y	Predicted
0	0
0	1
1	1
1	0

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$= \frac{2}{4} = 50\%$$

How do you check the performance of a Deep Neural Network?

(Multi-class Classification)

Test Data

y	Predicted
0	0
1	7
3	8
5	5
1	1

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$= \frac{3}{5} = 60\%$$

Summary and Roadmap

Real inputs

$x_i \in \mathbb{R}$

Squared Error Loss :

$$L(\Theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{i=1}^{d}(\hat{y}_{ij} - y_{ij})^2$$

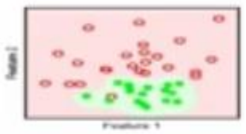
Cross Entropy Loss:

$$L(\Theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{i=1}^{d} y_{ij}\log(\hat{y}_{ij})$$

- Binary Classification
- Multi-class classification
- Regression

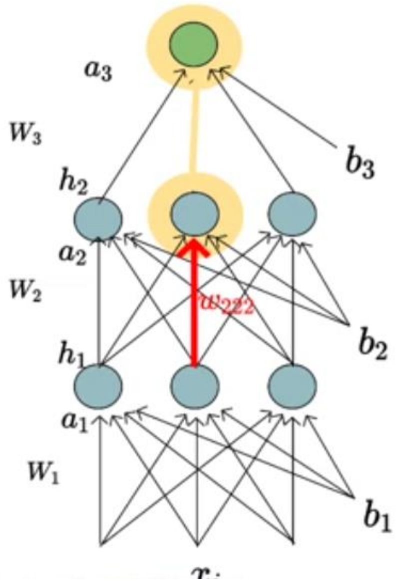
Gradient Descent with backpropagation

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$



$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

How many derivatives do we need to compute and how do we compute them?



- Let us focus on the highlighted weight (w_{222})
- To learn this weight, we have to compute partial derivative w.r.t loss function

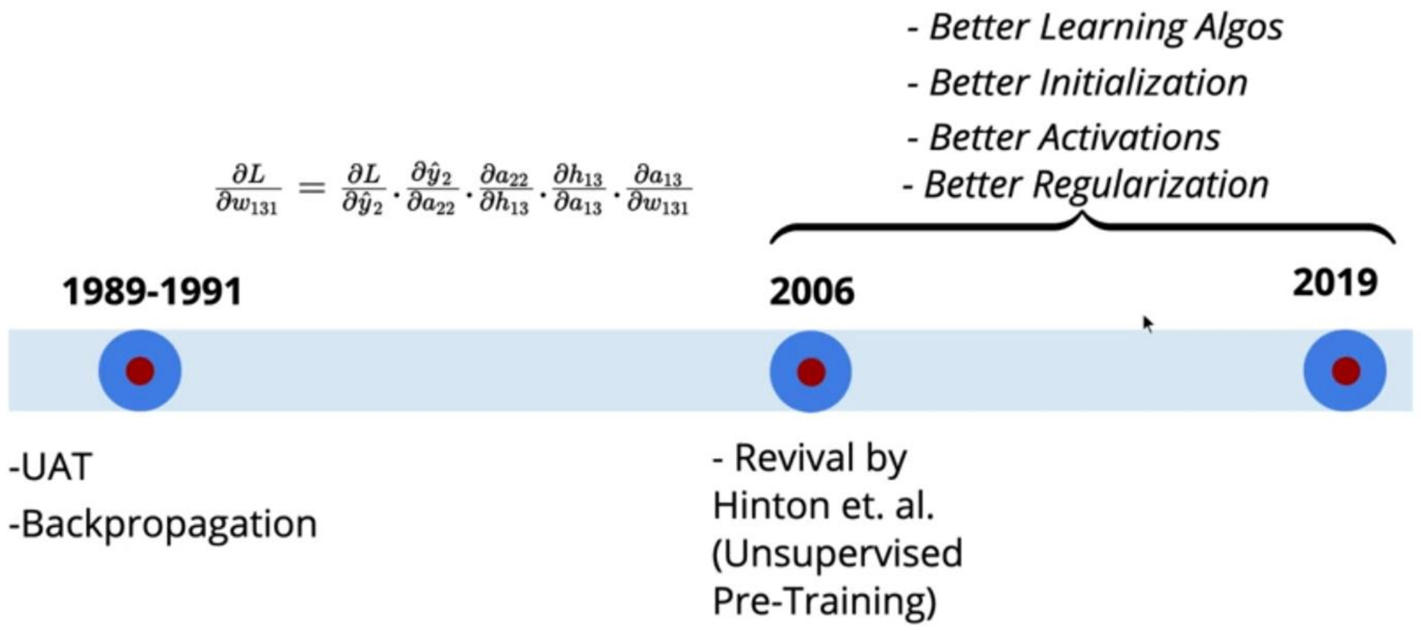
$$(w_{222})_{t+1} = (w_{222})_t - \eta * \left(\frac{\partial L}{\partial w_{222}}\right)$$

$$\begin{aligned}
\frac{\partial L}{\partial w_{222}} &= \left(\frac{\partial L}{\partial a_{22}}\right).\left(\frac{\partial a_{22}}{\partial w_{222}}\right) \\
&= \left(\frac{\partial L}{\partial h_{22}}\right).\left(\frac{\partial h_{22}}{\partial a_{22}}\right).\left(\frac{\partial a_{22}}{\partial w_{222}}\right) \\
&= \left(\frac{\partial L}{\partial a_{31}}\right).\left(\frac{\partial a_{31}}{\partial h_{22}}\right).\left(\frac{\partial h_{22}}{\partial a_{22}}\right).\left(\frac{\partial a_{22}}{\partial w_{222}}\right) \\
&= \left(\frac{\partial L}{\partial \hat{y}}\right).\left(\frac{\partial \hat{y}}{\partial a_{31}}\right).\left(\frac{\partial a_{31}}{\partial h_{22}}\right).\left(\frac{\partial h_{22}}{\partial a_{22}}\right).\left(\frac{\partial a_{22}}{\partial w_{222}}\right)
\end{aligned}$$

If DL is not working appropriate:

- Better Optimization
- Better Activation Function
- Better Weight Initialization
- Better Regularizer
- Better Compute
- Better Data

Deep Learning Timeline



Better Learning Algorithm

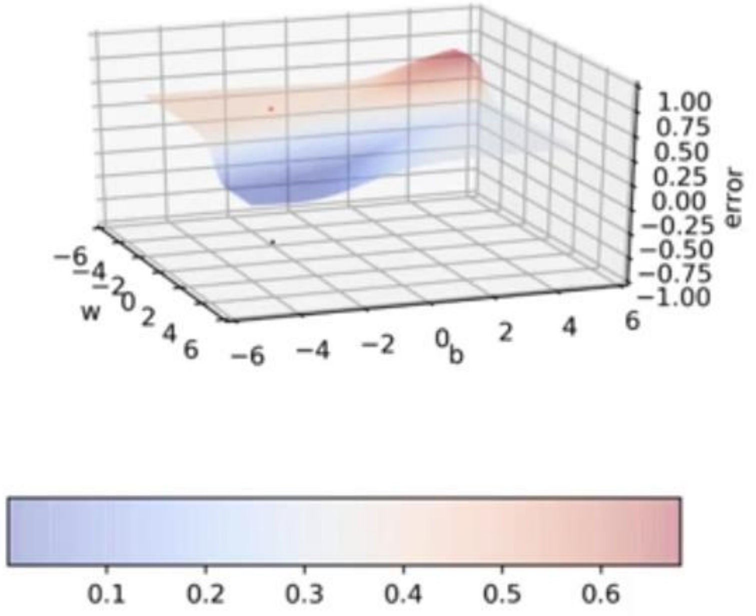
Gradient Descent Update Rule

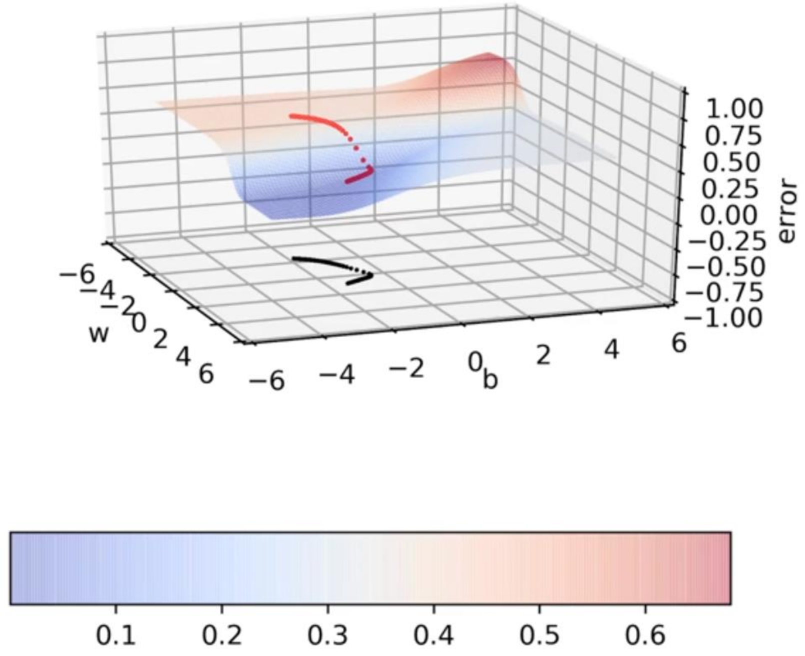
$$w = w - \eta \frac{\partial \mathscr{L}(w)}{\partial w}$$

- How do we compute the gradients?
- What data should we use for computing the gradients?
- How do we use the gradients?
- Can we come-up with a better update rule?

Gradient Descent Update

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x):
   #sigmoid with parameters w, b
   return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error(w, b):
     err = 0.0
     for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5* (fx - y) ** 2
     return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x


def do_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



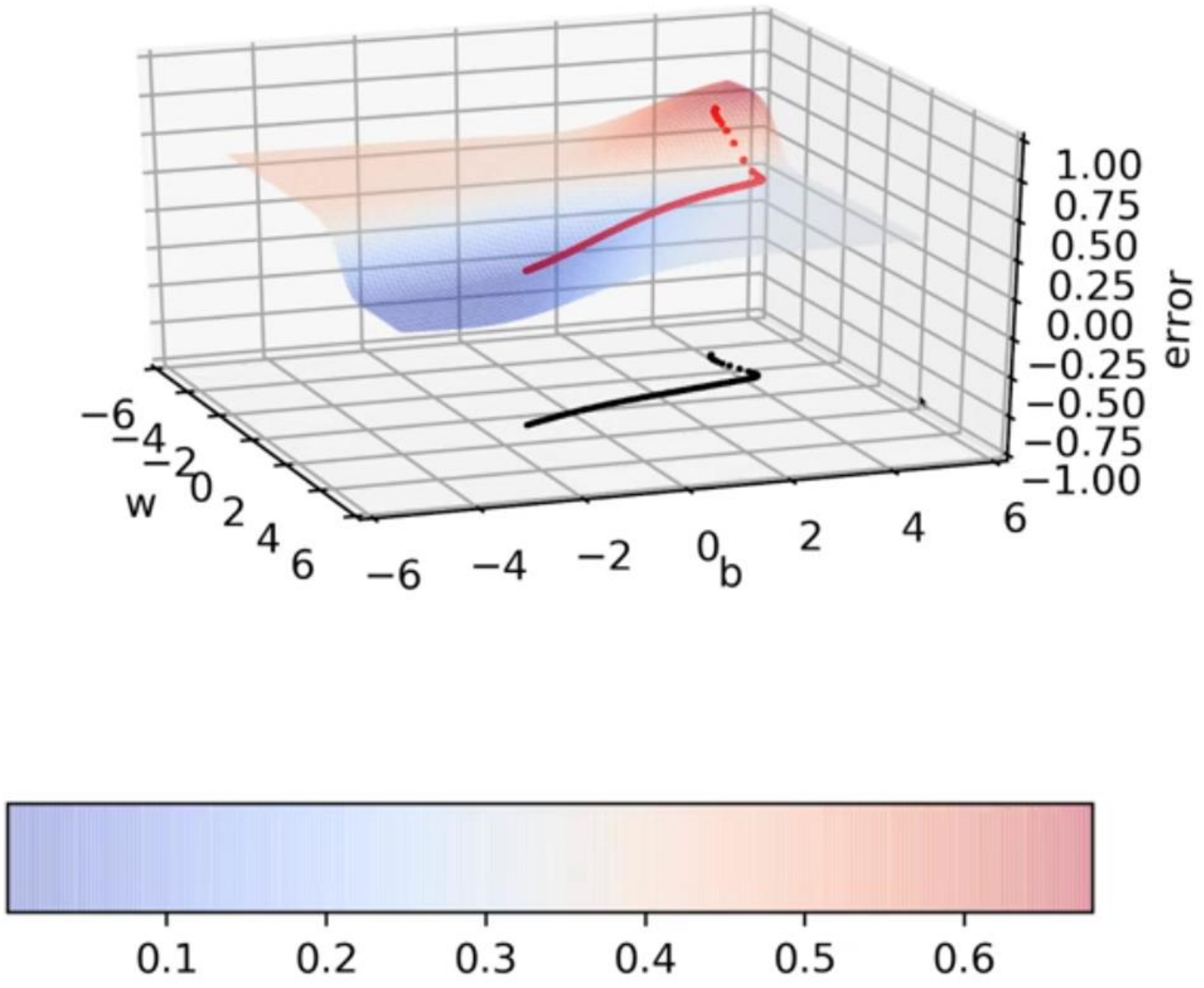


Drawback of Gradient Computation

Initialise w, b **randomly**

Iterate over data:

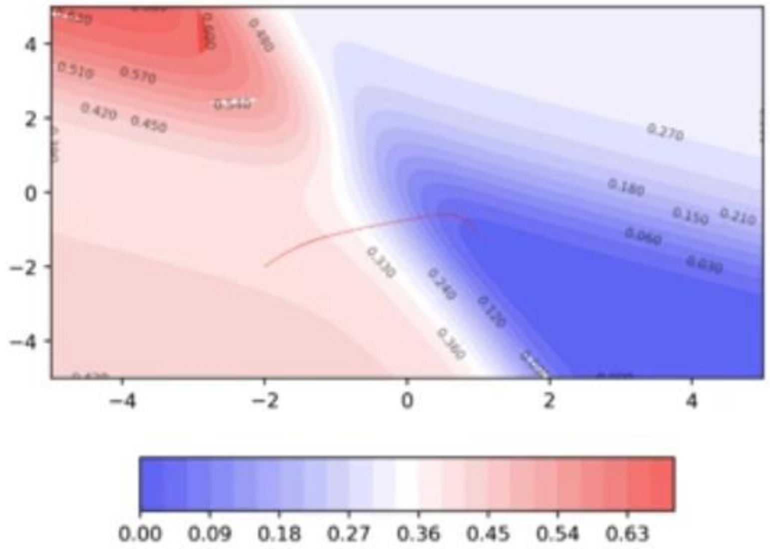
till satisfied



Momentum based Gradient Descent

Issues

It takes a lot of time to navigate regions having gentle slope (because the gradient in these regions is very small)



Intuitive Solution

If I am repeatedly being asked to go in the same direction, then I should probably gain some confidence & start taking bigger steps in that direction.

Mathematical Intuition

Gradient Descent Update Rule

$$w_{t+1} = w_t - \eta \nabla w_t$$

Momentum based Gradient Descent Update Rule

$$v_t = \gamma * v_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - v_t$$

Momentum Based Gradient Descent

$$v_t = \gamma \cdot v_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - v_t$$

$$
\begin{aligned}
v_0 &= 0 \\
v_1 &= \gamma \cdot v_0 + \eta \nabla w_1 = \eta \nabla w_1 \\
v_2 &= \gamma \cdot v_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2 \\
v_3 &= \gamma \cdot v_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\
&= \gamma \cdot v_2 + \eta \nabla w_3 = \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3 \\
v_4 &= \gamma \cdot v_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \cdot \eta \nabla w_2 + \gamma \cdot \eta \nabla w_3 + \eta \nabla w_4 \\
&\vdots \\
v_t &= \gamma \cdot v_{t-1} + \eta \nabla w_t = \gamma^{t-1} \cdot \eta \nabla w_1 + \gamma^{t-2} \cdot \eta \nabla w_1 + \ldots + \eta \nabla w_t
\end{aligned}
$$

*Exponential Decaying Average

Gradient Descent Update Rule

Vs

Momentum Based Gradient Descent

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x):
    #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error(w, b):
      err = 0.0
      for x, y in zip(X, Y):
         fx = f(w, b, x)
         err += 0.5* (fx - y) ** 2
      return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient Descent Update Rule

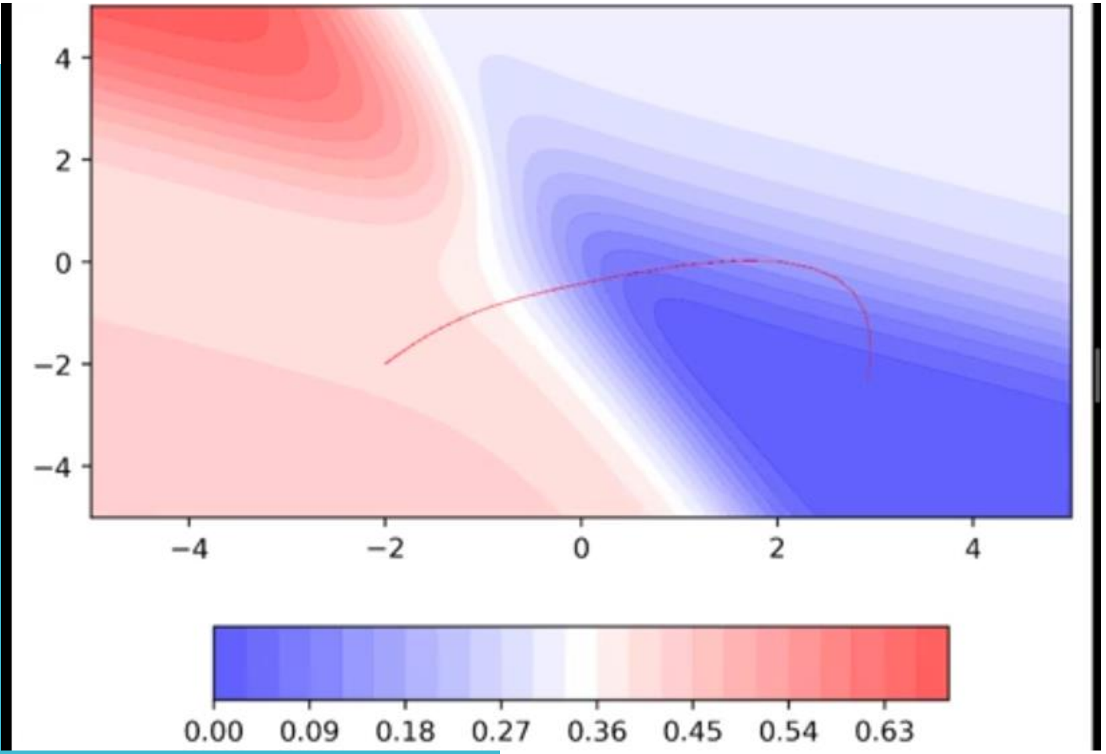
Momentum based Gradient Descent Update Rule

$$v_t = \gamma * v_{t-1} + \eta \nabla w_t$$

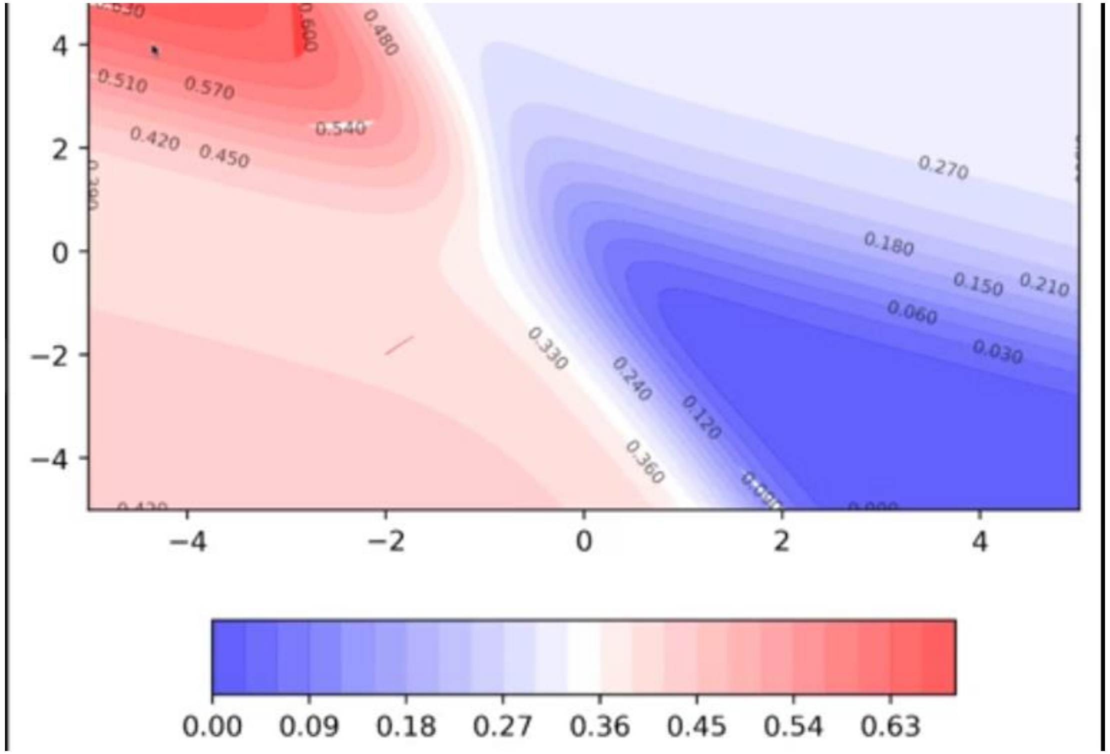
$$w_{t+1} = w_t - v_t$$

```
def do_momentum_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    v_w, v_b = 0, 0
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        v_w = gamma*v_w + eta * dw
        v_b = gamma*v_b + eta * dw

        w = w - v_w
        b = b - v_w
```

Momentum Gradient Descent



Momentum Gradient Descent



Gradient Descent

Convergence Graph - GD Vs MGD

Gradient Descent Update Rule