OPTIMIZATION TECHNIQUES

Dr. Umarani Jayaraman

Optimizing Gradient Descents

Algorithm with Gradient Components

Momentum, Nestrov

Algorithm with Adaptive Learning Rates

AdaGrad, RMSProp, Adadelta

Algorithm with both Gradient Components & Learning Rates

Adam, AdaMax, Nadam, AMSGrad

What is optimization technique?

- Optimization techniques in deep learning refer to methods used to minimize the loss function during the training of neural networks.
- Most common optimization technique is SGD

Gradient descent

- Gradient descent is an **optimization method** for finding the **minimum of a function.**
- It is commonly used in deep learning models to update the weights of a neural network through back propagation.

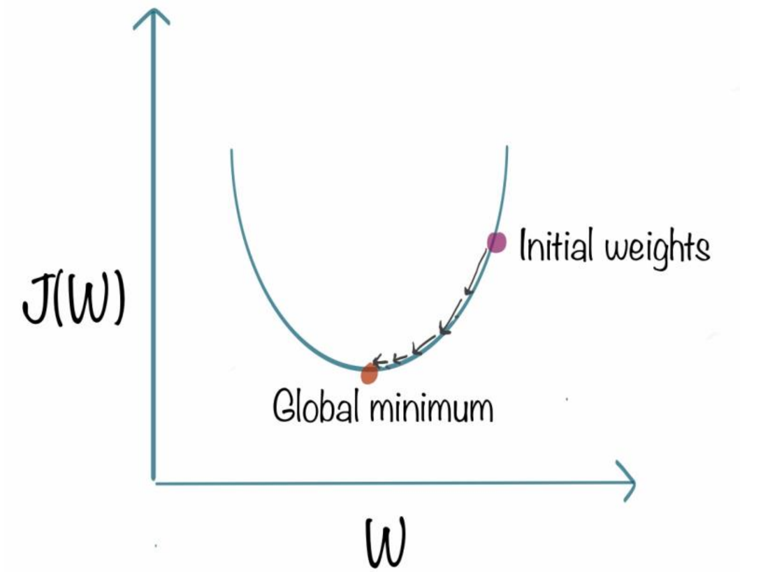
$$W_{new} = W_{old} - \eta\ (\partial L / \partial W_{old})$$

$\eta \rightarrow$ Learning rate

$(\partial L / \partial W_{old}) \rightarrow$ Derivation of loss function with respect to the old weight

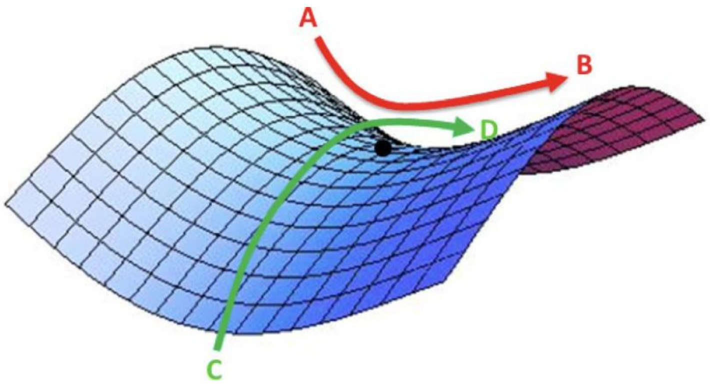
Gradient Descent Challenges

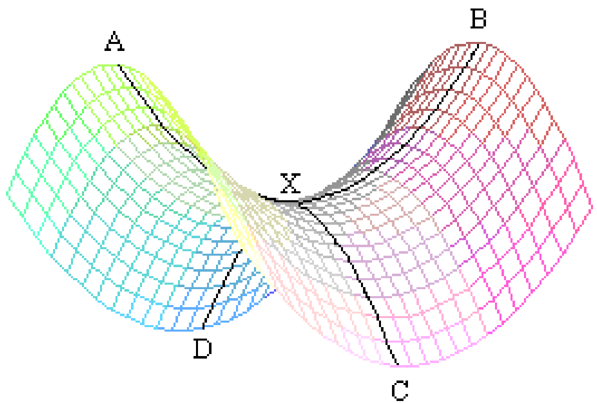
- Challenges of mini-batch Gradient Descent
- Choice of proper learning rate
  - Too small a learning rate leads to slow convergence
  - A large learning rate may leas to oscillation around the minima or it may diverge



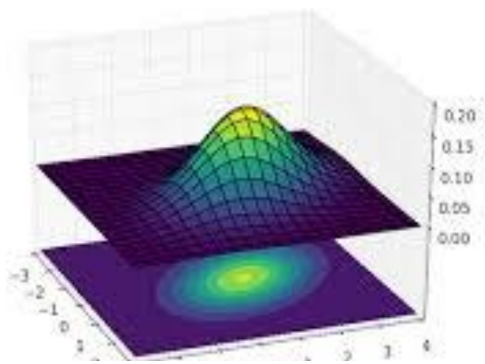
Gradient Descent Challenges

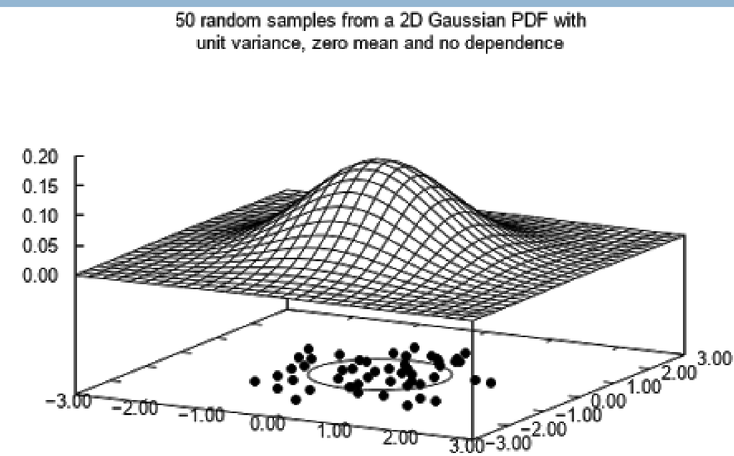
- Avoiding getting trapped in suboptimal local minima
  - Difficulty arises in from saddle points (i.e) points where one dimension slopes up and another slopes down
  - These saddle points which makes it hard for SGD to escape as the gradient is close to zero in all dimensions





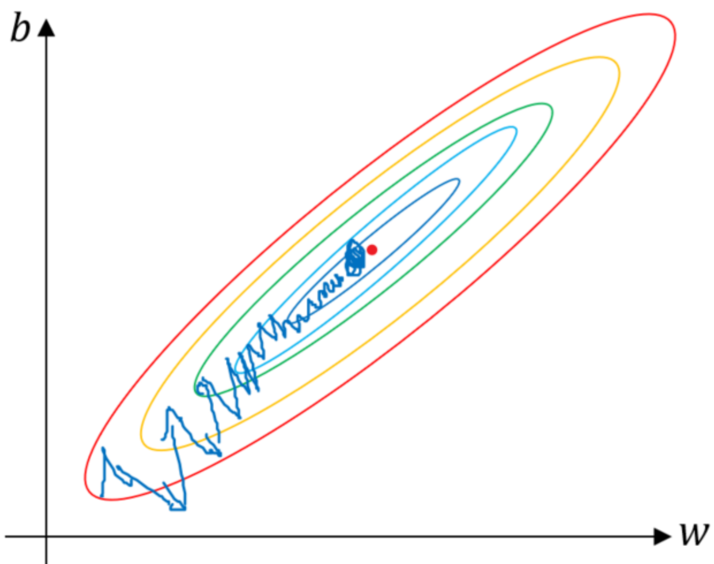
Gradient Descent

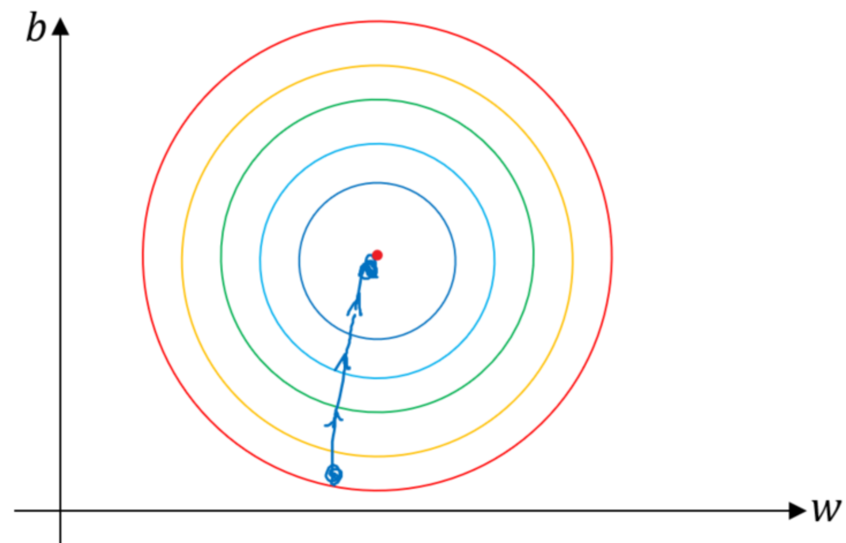




Unnormalized

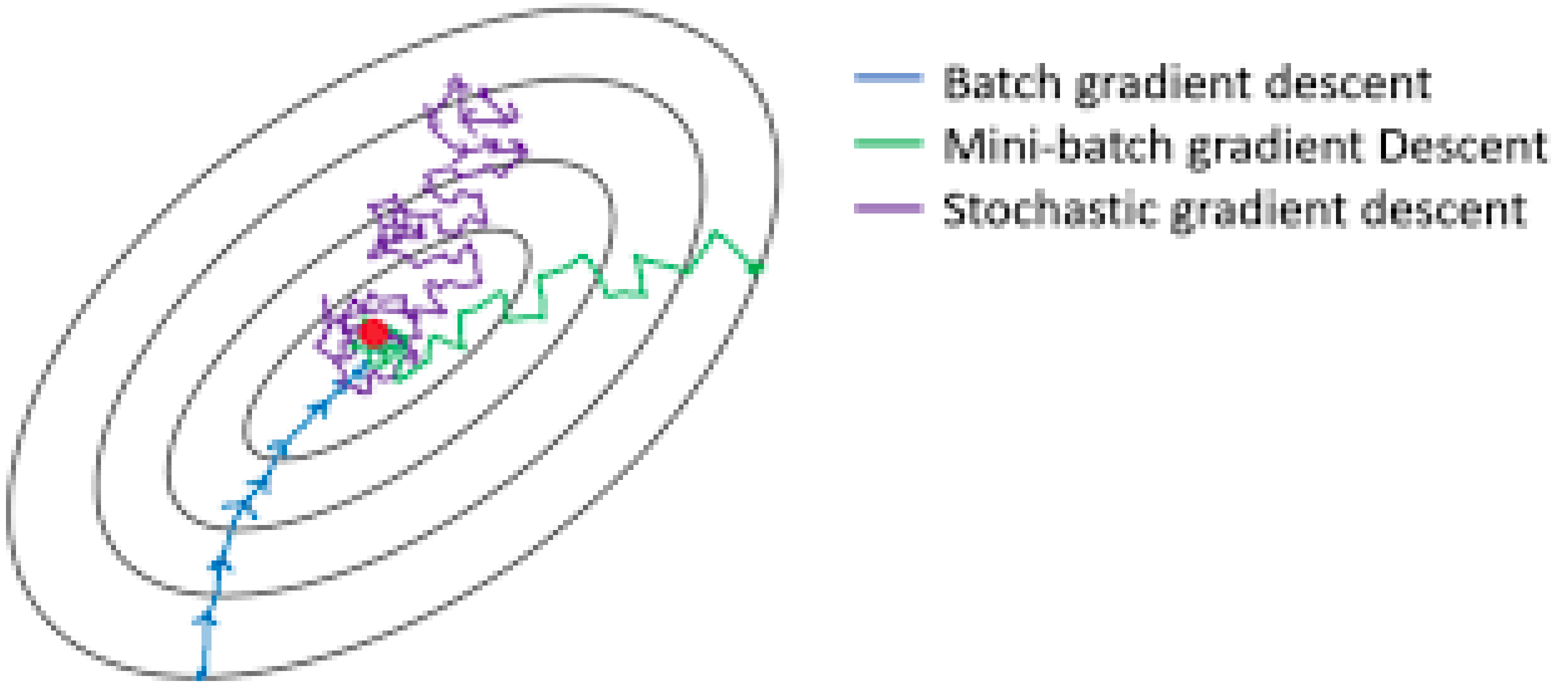
Normalized





Gradient Descent- Variants

- Batch
- Stochastic
- Mini-batch



1. Gradient descent (Batch GD)

- The standard GD updates the current weight using the current gradient $\partial L/\partial w$ multiplied by some factor called the learning rate, α.

$$w_{\text{new}} = w - \alpha \frac{\partial L}{\partial w}$$

.

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2.$$

2 . Stochastic gradient descent

- The vanilla SGD updates the current weight using the current gradient $\partial L/\partial w$ multiplied by some factor called the learning rate, α.

$$w_{\text{new}} = w - \alpha \frac{\partial L}{\partial w}$$

3 . Mini-Batch gradient descent

- The vanilla SGD updates the current weight using the current gradient $\partial L/\partial w$ multiplied by some factor called the learning rate, α.

$$w_{\text{new}} = w - \alpha \frac{\partial L}{\partial w}$$

Mini-batch gradient descent

Say $b = 10, m = 1000.$

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

}

}

Epoch vs Iterations

- For record of 10K
- Epoch 1 → GD ← 1iteration
- Epoch 1 → SGD ← 10 K (Total No of records)
- Epoch 1 → MBGD ← Total No of records / Batch size = 10K/1000 = 10 iterations
- Repeat this for multiple epoch to reach global minimum

Gradient Descent- Variants

- **1) Adapt the "gradient component"** ($\partial L/\partial w$)
  Instead of using only one (single) gradient like in stochastic vanilla gradient descent to update the weight, take an *aggregate of multiple gradients*. Specifically, these optimizers use the exponential moving average of gradients.
- **2) Adapt the "learning rate component"** (α)
  Instead of keeping a constant learning rate, adapt the learning rate according to the *magnitude* of the gradient(s).
- **Both (1) and (2)**
  Adapt both the gradient component and the learning rate component.

More Gradient Descent- Variants

- These optimizers try to improve the amount of information used to update the weights, mainly through using previous (and future) gradients, instead of only the present available gradient.

$$w_{\text{new}} = w - \alpha \frac{\partial L}{\partial w}$$

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$

Gradient Descent- Variants

- Below is a table that summarizes which "components" are being adapted:

Optimiser	Year	Learning Rate	Gradient
Momentum	1964		✓
AdaGrad	2011	✓	
RMSprop	2012	✓	
Adadelta	2012	✓	
Nesterov	2013		✓
Adam	2014	✓	✓
AdaMax	2015	✓	✓
Nadam	2015	✓	✓
AMSGrad	2018	✓	✓

Gradient Descent- Variants

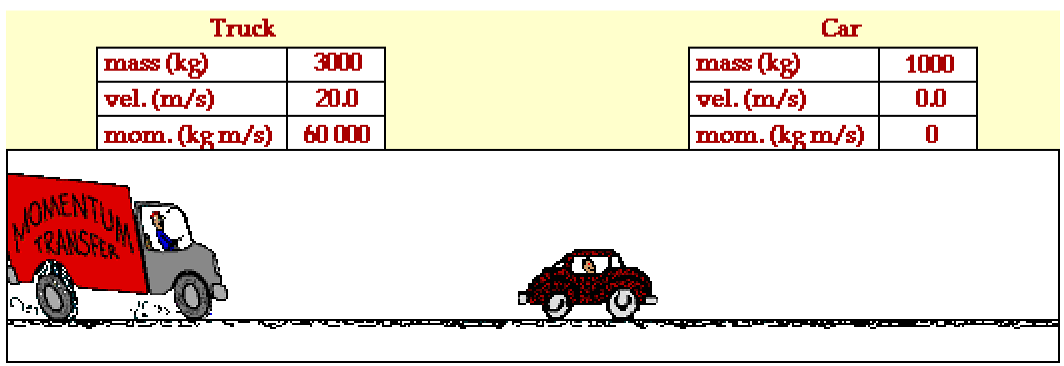
- Batch, Stochastic and Mini-batch gradient descent
- Momentum
- NAG
- AdaGrad
- RMSprop
- Adadelta
- Adam
- AdaMax
- Nadam
- AMSGrad

By changing the gradient

Momentum

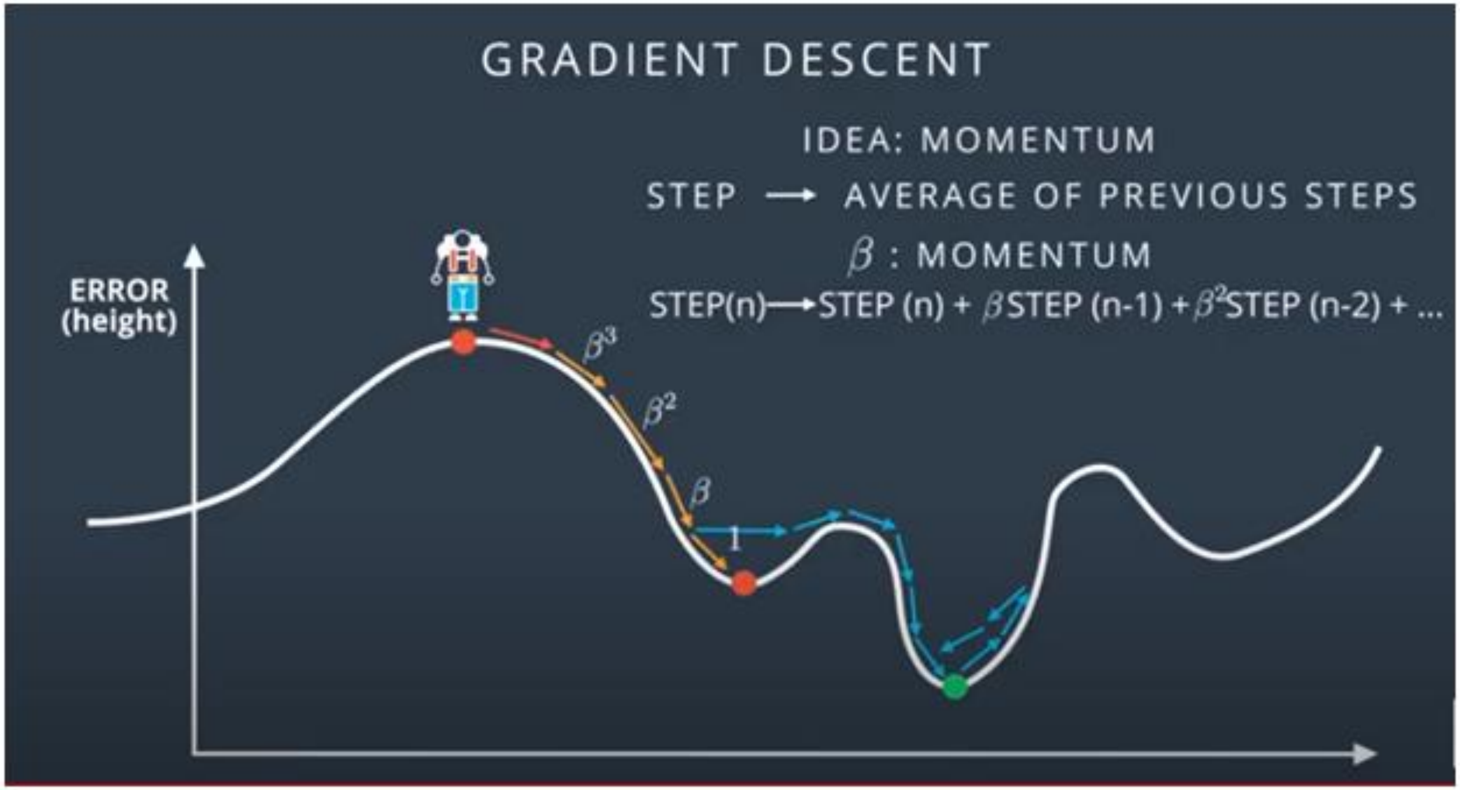
$$w_{\text{new}} = w - \alpha \frac{\partial L}{\partial w}$$

$$w_{t+1} = w_t - \alpha m_t$$



Momentum

- For series of data (time series data)
- $t_1, \quad t_2, \quad ,t_3, \ldots \; ,t_n$
- $a_1, \quad a_2, \quad ,a_3, \ldots ,a_n$
- $Vt_1 = a_1$
- $Vt_2 = \beta * Vt_1 + (1-\beta) * a_2 = \beta * a_1 + (1-\beta) * a_2$
- $Vt_3 = \beta * Vt_2 + (1-\beta) * a_3 = Vt_3 = \beta * [\beta * a_1 + (1-\beta) * a_2] + (1-\beta) * a_3$


GRADIENT DESCENT
IDEA: MOMENTUM
STEP ⟶ AVERAGE OF PREVIOUS STEPS
β : MOMENTUM
STEP(n) ⟶ STEP (n) + βSTEP (n-1) + β^2STEP (n-2) + ...
ERROR
(height)
β^3
β^2
β
1

4. Momentum Optimizer

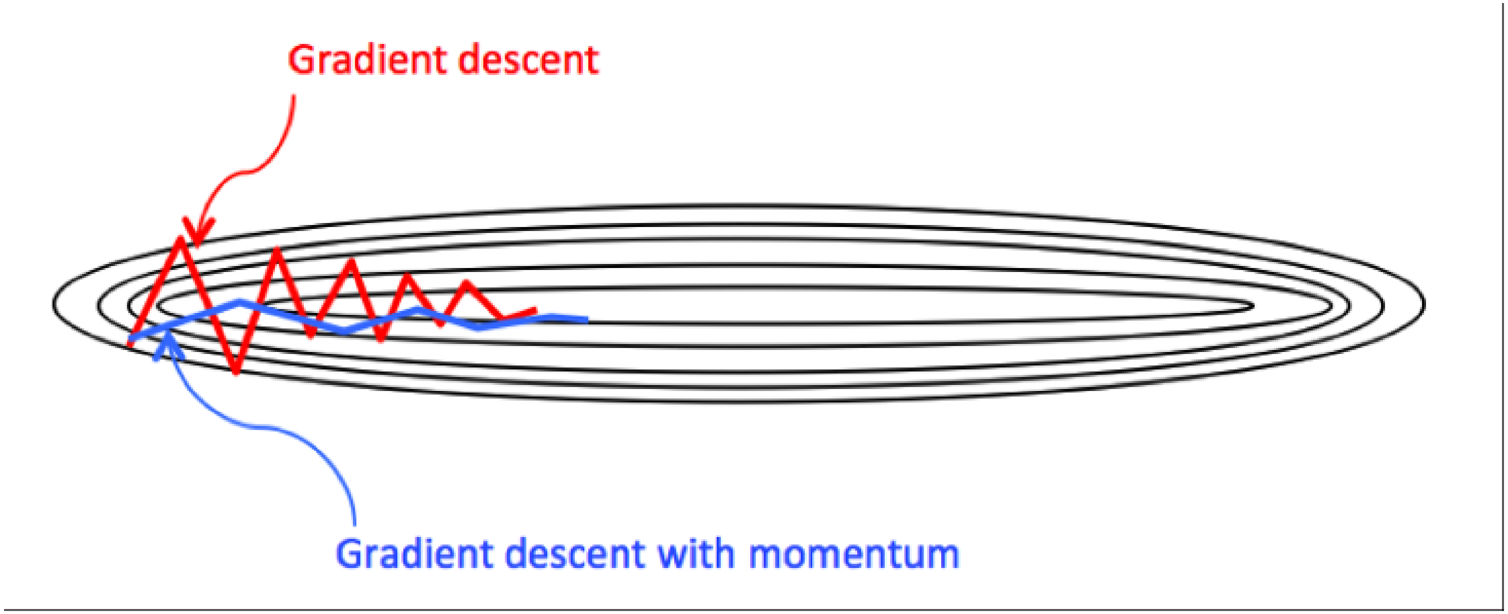
- Instead of depending only on the current gradient to update the weight, gradient descent with momentum (Polyak, 1964) replaces the current gradient with *m* ("momentum"), which is an aggregate of gradients.
- This aggregate is the exponential moving average of current and past gradients (i.e. up to time *t*).

$$w_{t+1} = w_t - \alpha m_t$$

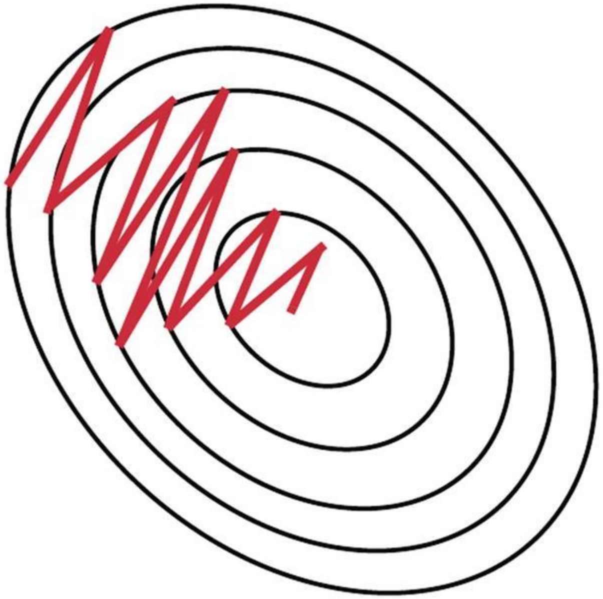
- Where, *m* initialized to 0 and $\beta = 0.9$

$$m_t = \beta m_{t-1} + (1 - \beta) \frac{\partial L}{\partial w_t}$$

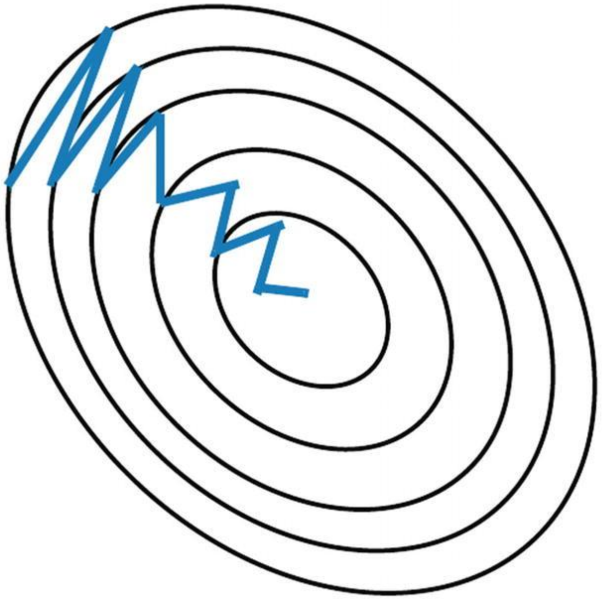
SGD with momentum



SGD with momentum

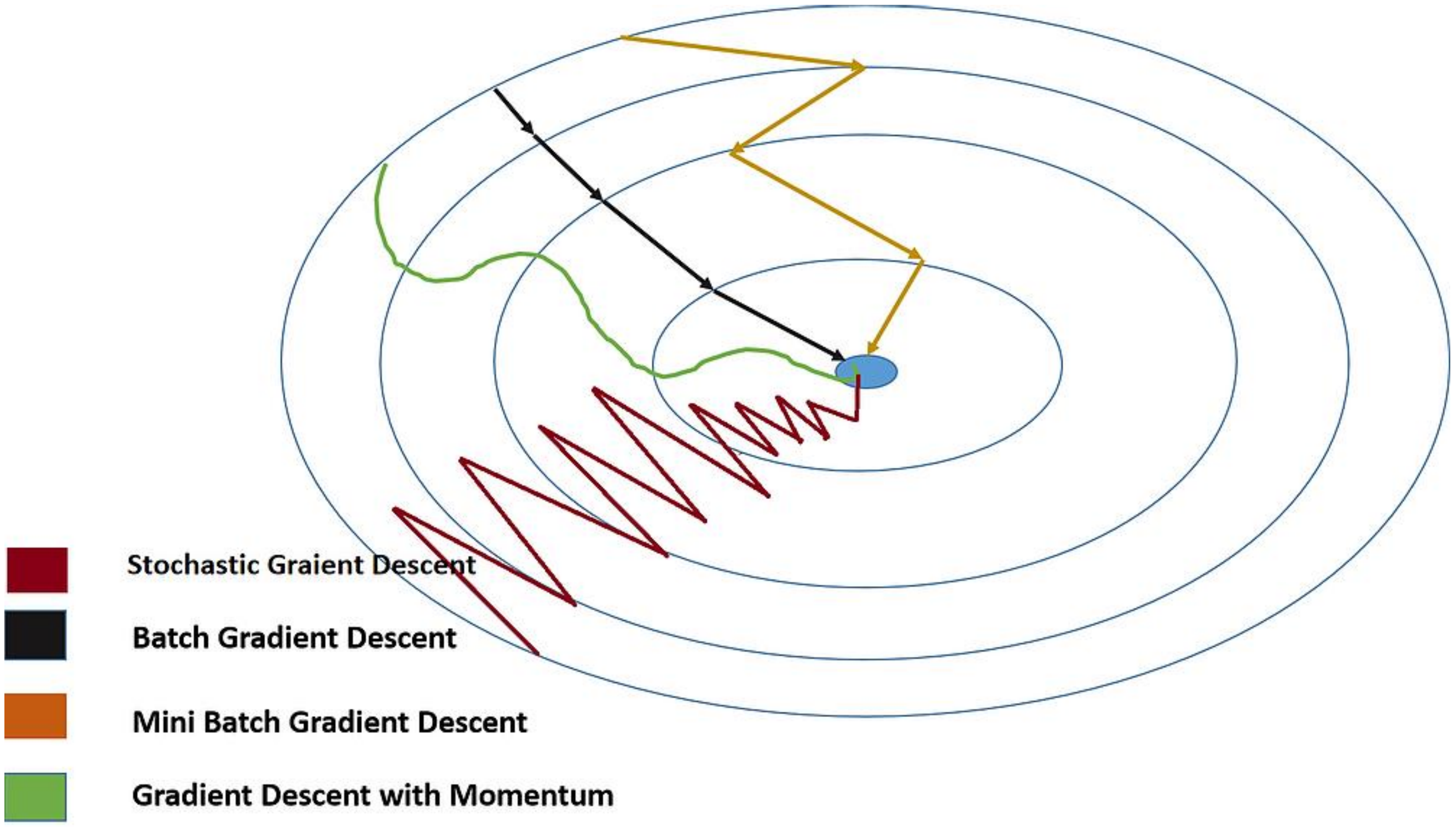


Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

Gradient Descents



5. Nesterov Accelerated Gradient (NAG)

- This update utilizes *m*, the exponential moving average of what we call *pro look ahead gradients*.

$$w_{t+1} = w_t - \alpha m_t$$

- Where, and *m* initialised to 0.

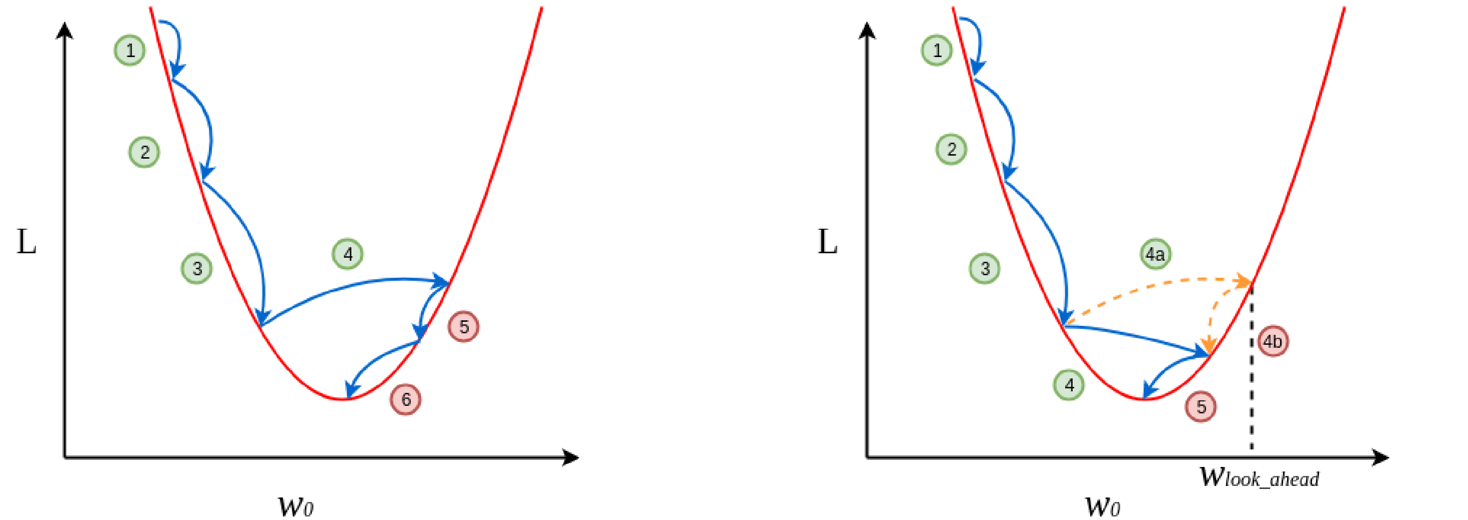
$$m_t = \beta m_{t-1} + (1 - \beta) \frac{\partial L}{\partial w^*}$$

$$w^* = w_t - \alpha m_{t-1}$$

NAG

- The last term in the second equation, w^* is a look ahead gradient
- This value can be obtained by going ‘one step ahead’ using the previous velocity.

Momentum vs NAG- Learning rate is fixed

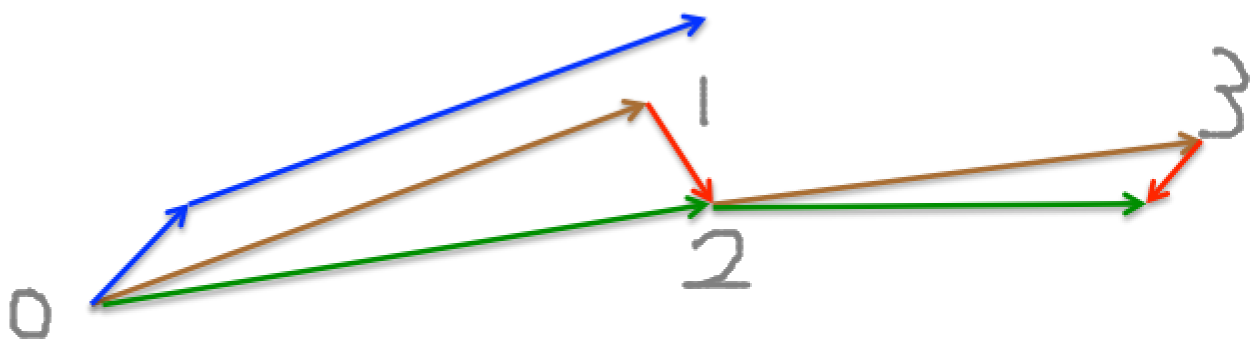


(a) Momentum-Based Gradient Descent (b) Nesterov Accelerated Gradient Descent

Nesterov Accelerated Gradient (NAG)

A picture of the Nesterov method

- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction.



brown vector = jump, red vector = correction, green vector = accumulated gradient

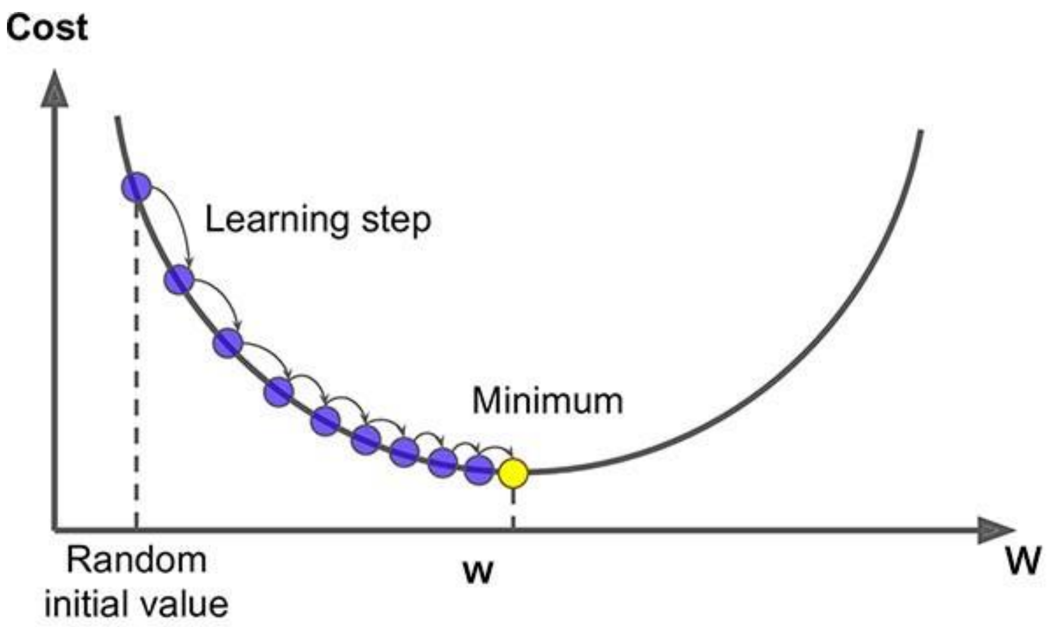
blue vectors = standard momentum

Problem with Momentum Optimizer/ NAG

- Both the algorithm requires the hyper parameters to be set manually which is beta (β).
- These hyper-parameter decide the learning rate
- The algorithm uses same learning rate for all dimensions
- We may require learning rate could be small in some dimension and large in another dimension

By changing the learning rate

Adaptive (various) learning rate



6. AdaGrad- Adaptive gradient

- Adaptive gradient (Duchi et al., 2011), acts on the learning rate component by dividing the learning rate by the square root of *v*, which is the cumulative sum of current and past squared gradients (i.e. up to time *t*).
- Note that the gradient component remains unchanged like in SGD.

$$A = \pi r^2$$

- Notice that ε is a small floating point value to ensure that we will never have to come across division by zero. *v* initialized to 0.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$v_t = v_{t-1} + \left[\frac{\partial L}{\partial w_t}\right]^2 = \sum_{i=1}^{t} \left[\frac{\partial L}{\partial w_i}\right]^2$$

AdaGrad- Adaptive gradient

- One dis-advantage of adgrad is that v_t becomes very high (squaring every time)
- As a result learning rate,α become very small
- No change in old weight and new weight

$$W_{new} \cong W_{old}$$

- This problem is fixed by slightly modifying the formula

7. RMSprop

- Root mean square prop or RMSprop (Hinton et al., 2012) is another adaptive learning rate that tries to improve AdaGrad.
- Instead of taking cumulative sum of squared gradients like in AdaGrad, we take the exponential moving (decay) average (Similar to momentum) of these gradients.

RMSprop (contd.)

- and v initialized to 0.
- $\alpha = 0.001$
- $\beta = 0.9$ (recommended by the authors of the paper)
- $\varepsilon = 10^{-6}$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$v_t = \beta v_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right]^2$$

8. Adadelta

- Like RMSprop, Adadelta (Zeiler, 2012) is also another improvement from AdaGrad, focusing on the learning rate component.
- Adadelta is probably short for 'adaptive delta', where *delta* here refers to the difference between the current weight and the newly updated weight.
- The difference between Adadelta and RMSprop is that Adadelta removes the use of the learning rate parameter completely by replacing it with *D,* the exponential moving average of squared *deltas*.

Adadelta

$$w_{t+1} = w_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

- where

$$D_t = \beta D_{t-1} + (1 - \beta)[\Delta w_t]^2$$

$$v_t = \beta v_{t-1} + (1 - \beta)\left[\frac{\partial L}{\partial w_t}\right]^2$$

- with *D* and *v* initialized to 0, and
- $\beta = 0.95$
- $\varepsilon = 10^{-6}$

$$\Delta w_t = w_t - w_{t-1}$$

9. Adam

- Adaptive moment estimation, or Adam (Kingma & Ba, 2014), is simply a combination of momentum and RMSprop. It acts upon
- the gradient component by using m, the exponential moving average of gradients (like in momentum), and
- the learning rate component by dividing the learning rate α by square root of v, the exponential moving average of squared gradients (like in RMSprop).

Adam

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

- Where

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- are the bias corrections, and

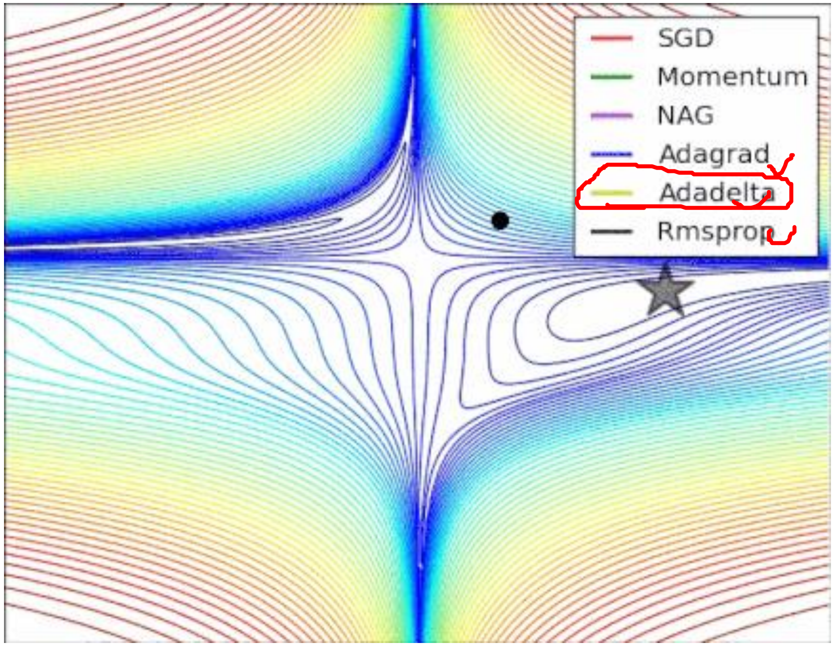
Adam

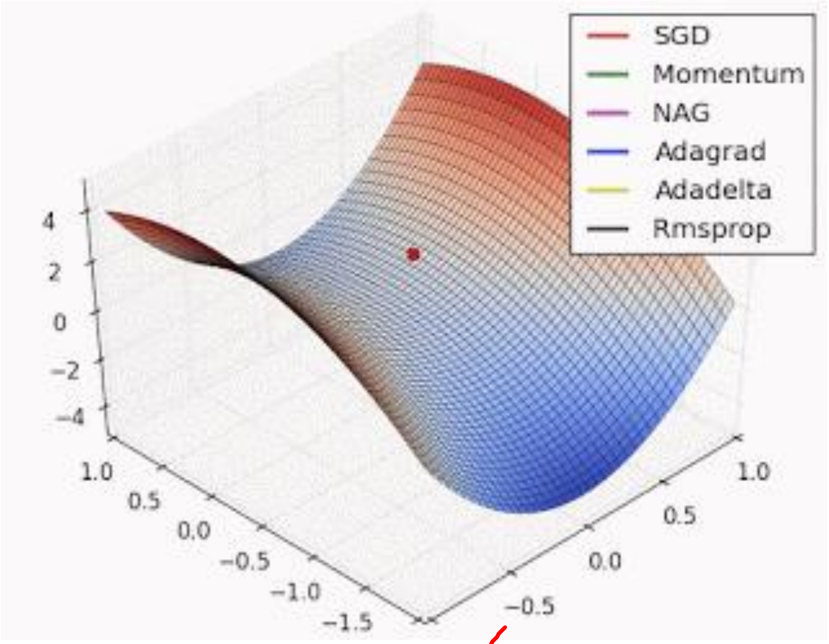
$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2$$

- with m and v initialized to 0.
- $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$

Gradient Descent Optimization-summary





Summary

Vanilla SGD

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$

Momentum

$$w_{t+1} = w_t - \alpha m_t$$

$$m_t = \beta m_{t-1} + (1-\beta) \frac{\partial L}{\partial w_t}$$

Adagrad

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$v_t = v_{t-1} + \left[\frac{\partial L}{\partial w_t}\right]^2$$

RMSprop

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$v_t = \beta v_{t-1} + (1-\beta) \left[\frac{\partial L}{\partial w_t}\right]^2$$

Adadelta

$$w_{t+1} = w_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$D_t = \beta D_{t-1} + (1-\beta)[\Delta w_t]^2$$

$$v_t = \beta v_{t-1} + (1-\beta) \left[\frac{\partial L}{\partial w_t}\right]^2$$

Nesterov

$$w_{t+1} = w_t - \alpha m_t$$

$$m_t = \beta m_{t-1} + (1-\beta) \frac{\partial L}{\partial w^*}$$

$$w^* = w_t - \alpha m_{t-1}$$

Adam

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \frac{\partial L}{\partial w_t}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) \left[\frac{\partial L}{\partial w_t}\right]^2$$

Sources

- https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c
- https://www.slideshare.net/ssuser77b8c6/an-overview-of-gradient-descent-optimization-algorithms
- https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9

THANK YOU