Optimization
MGD and AdaGrad (Adaptive Gradient)

Momentum-based Gradient Descent (MGD

- Momentum-based Gradient Descent (MGD) is a variation of Gradient Descent that helps accelerate convergence and improve stability during the optimization process.
- In traditional Gradient Descent, the update rule for the parameters at each iteration is based solely on the gradient of the cost function with respect to the parameters. However, in MGD, we also introduce a momentum term that accumulates the gradients over time and helps to smooth out the updates and prevent oscillations in the optimization process.
- The update rule for MGD can be written as:

$$v = \beta v + \alpha \nabla J(\theta),$$

$$\theta = \theta - v$$

- where θ is the vector of parameters, $\nabla J(\theta)$ is the gradient of the cost function with respect to θ, α is the learning rate, β is the momentum parameter (usually set to a value between 0.9 and 0.99), and v is the velocity vector.

Momentum-based Gradient Descent (MGD)

- At each iteration, we calculate the gradient of the cost function with respect to the parameters and update the velocity vector v by taking a weighted average of the previous velocity and the current gradient. The new parameters θ are then updated based on the velocity vector and the learning rate.
- Introducing momentum helps MGD to build up velocity in directions where the gradients consistently point, and dampens the oscillations that occur in directions where the gradients vary in sign or magnitude. This can lead to faster convergence and more stable optimization, especially when the cost function is noisy or has a lot of curvatures.
- Overall, MGD is a powerful optimization algorithm that can help improve the performance of machine learning models by accelerating convergence and improving stability.

(MGD)

Given Dataset

x	y
1	3
2	5
3	7

Hyperparameters

- Initial Weight: $w = 0.5$
- Initial Bias: $b = 0.1$
- Learning Rate: $\alpha = 0.1$
- Momentum Coefficient: $\beta = 0.9$
- Initial Velocities: $V_w = 0, V_b = 0$

General MGD Formulas

1. **Velocity Updates:**

$$V_w = \beta V_w + \alpha \frac{\partial L}{\partial w}$$

$$V_b = \beta V_b + \alpha \frac{\partial L}{\partial b}$$

2. **Parameter Updates:**

$$w = w - V_w$$

$$b = b - V_b$$

- We have a dataset of 5 samples with the following values

x	y
1	2
2	4
3	6
4	8
5	10

Our goal is to minimize the mean squared error (MSE) loss function:
L = 1/2m * (y_pred - y)^2
where y_pred is the predicted value of y based on the current values of the slope and intercept.
The MGD algorithm involves maintaining a momentum variable v, which is a moving average of the gradients of the loss function with respect to the parameters. The update rule for the parameters becomes:
w = w - lr * v_w
b = b - lr * v_b

where lr is the learning rate hyperparameter, v_w and v_b are the momentum variables for the slope and intercept respectively, and w and b are the current values of the slope and intercept.

AdaGrad (Adaptive Gradient)

- AdaGrad (Adaptive Gradient) is a gradient descent optimization algorithm that adapts the learning rate of each parameter based on the historical gradient information.

- The basic idea of AdaGrad is to use a different learning rate for each parameter, which is adapted during the training process.

- The learning rate is decreased for frequently occurring features and increased for infrequent features, which can improve the performance of the model.

AdaGrad (Adaptive Gradient)

- The AdaGrad algorithm can be summarized in the following steps:
- Initialize the weight vector w and the gradient sum G to zero.

For each iteration t:

a. Compute the gradient gt of the loss function with respect to the weight w(t).

b. Add the squared gradient gt^2 to the diagonal of G.

c. Compute the update vector δt as: δt = (η / √(Gt + ε)) * gt

d. Update the weight vector as: w(t+1) = w(t) - δt

AdaGrad (Adaptive Gradient)

$$w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial w(t-1)}$$

Where **alpha(t)** denotes different learning rates for each weight at each iteration.

$$\eta'_t = \frac{\eta}{\mathrm{sqrt}(\alpha_t + \epsilon)}$$

Here, **η** is a constant number, **epsilon** is a small positive value number to avoid divide by zero error if in case **alpha(t)** becomes 0 because if alpha(t) become zero then the learning rate will become zero which in turn after multiplying by derivative will make w(old) = w(new), and this will lead to small convergence.

$$\alpha_t = \sum_{i=1}^{t} g_i^2 \quad g_i = \frac{\partial L}{\partial w(old)}$$

g_i is derivative of loss with respect to weight and g_i^2 will always be positive since its a square term, which means that alpha(t) will also remain positive, this implies that *alpha(t) >= alpha(t-1)*.

- Second input Input (*x*,*y*)=(4,5)

- Weight (*w*) = 0.594 (updated from the previous step)
- learning rate =0.1

- dL_dw =2*([0.594×4]−5)×4= −20.992

$\alpha_t = \sum_{i=1}^{t} g_i^2$ = 440.832

So, $0.5 - \left[\frac{0.1}{\sqrt{440.832+10^{-8}}} \times (-20.992)\right] \approx 0.6.$

AdaGrad (Adaptive Gradient)

$$w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial w(t-1)} \qquad \eta'_t = \frac{\eta}{\text{sqrt}(\alpha_t + \epsilon)}$$

- Given:Weight (*w*) = 0.5, Input (*x*,*y*)=(3,4), learning rate =0.1 $\epsilon = 10^{-8}$
- Our goal is to minimize the mean squared error (MSE) loss function:
- L = 1/n* (y_pred - y)^2
- y_pred = 0.5×3=1.5
- dL_dw = 2/1 * (y_pred - y) * x
- dL_dw = 2*(1.5-4)*3=-15

$\alpha_t = \sum_{i=1}^{t} g_i^2$ =(−15)^2=225

- Update the parameter using the Adagrad update rule.
- $w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial w(t-1)} =$ $0.5 - \left[\frac{0.1}{\sqrt{255+10^{-8}}} \cdot (-15)\right]$
- = ≈0.59405

- One advantage of AdaGrad is that it does not require a manual tuning of the learning rate, as the learning rate is automatically adjusted based on the historical gradient information. However, one disadvantage is that the learning rate can become too small over time, which can lead to slow convergence or premature stopping of the algorithm.