Optimizer In Deep Learning

In deep learning, optimizers are algorithms used to adjust the model's parameters (weights and biases) to minimize the loss function and improve performance. They play a crucial role in training neural networks by guiding the model toward convergence.

Key Concepts:

1. Objective: Reduce the loss function by updating model parameters iteratively.
2. Learning Rate (η\etaη): Determines the size of the steps taken towards the minimum.
3. Gradient Descent: Core technique where gradients (derivatives of the loss function) guide updates to the parameters.

What are Optimizers in Deep Learning?

In deep learning, optimizers are crucial as algorithms that dynamically fine-tune a model's parameters throughout the training process, aiming to minimize a predefined loss function. These specialized algorithms facilitate the learning process of neural networks by iteratively refining the weights and biases based on the feedback received from the data. Well-known optimizers in deep learning encompass Stochastic Gradient Descent (SGD), Adam, and RMSprop, each equipped with distinct update rules, learning rates, and momentum strategies, all geared towards the overarching goal of discovering and converging upon optimal model parameters, thereby enhancing overall performance.

Before moving toward optimizer let's understand concept of

Exponential Weighted Moving Average (EWMA)

EWMA is a technique used to identify trends in time series data by giving more importance to recent values than older ones. This method is widely applied in:

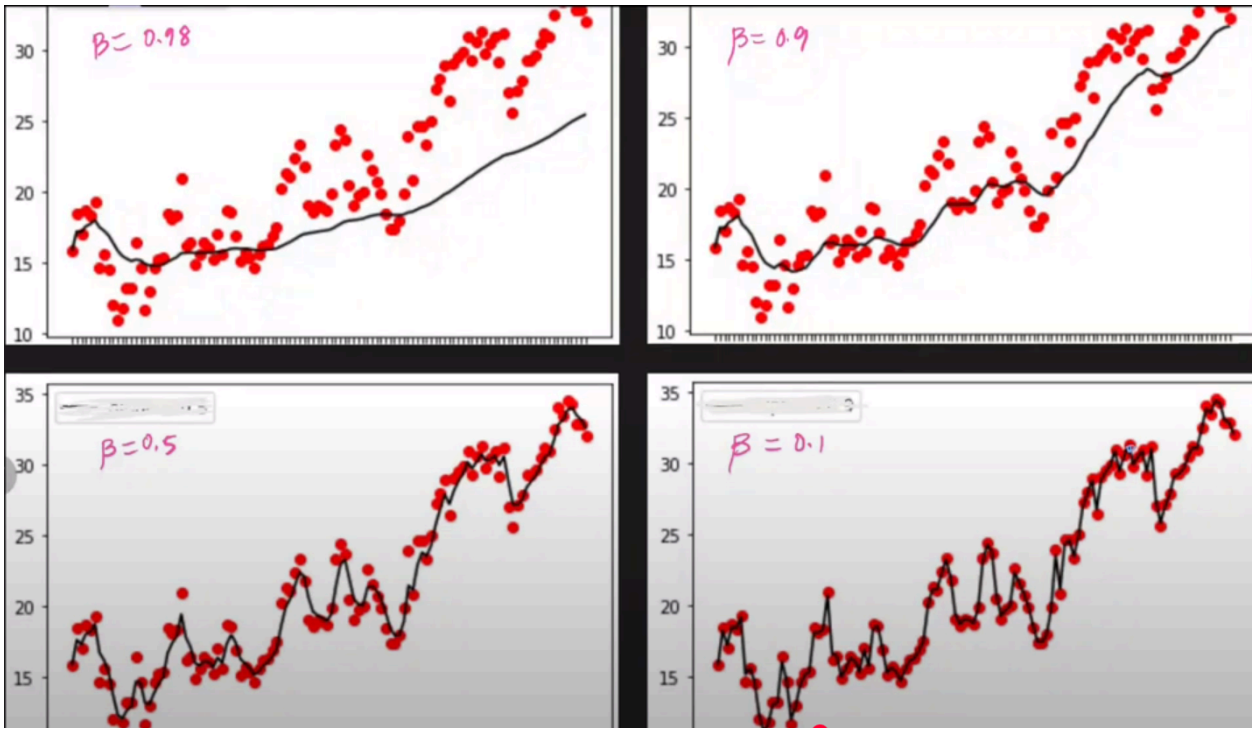
- **Time Series Forecasting**
- **Finance**
- **Signal Processing**

The goal of EWMA is to uncover hidden patterns in time series data by applying a decaying weight to past observations, favoring the most recent data points.

Key Characteristics of EWMA

1. Weighting Recent Data More: EWMA emphasizes recent data points over older ones.
2. Dynamic Trend Identification: Useful for finding trends in dynamic data.
3. Beta (β): A constant between 0 and 1 that controls the weight decay rate:
    - Higher β values give more weight to past data.
    - Lower β values emphasize recent data more.

While calculating EWMA we give more importance to current values in comparison to old points.



Mathematical Formulation

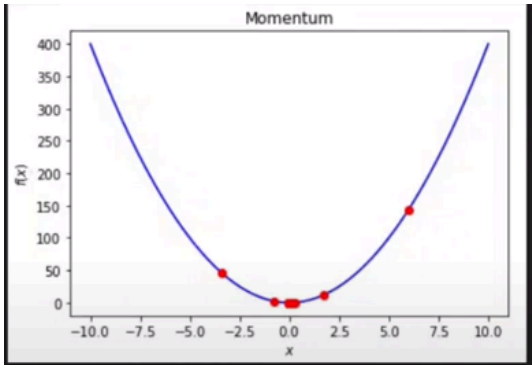
The EWMA at time , denoted as , is calculated as:

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t$$

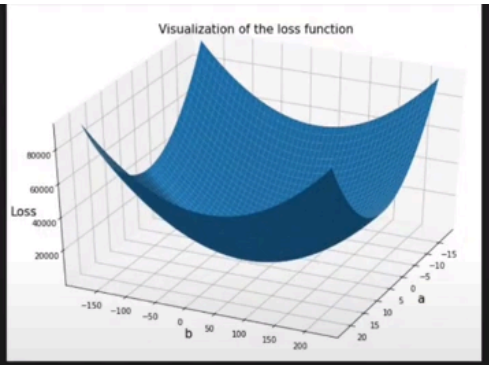
Note: Mathematical intuition from note

1. SGD with Momentum

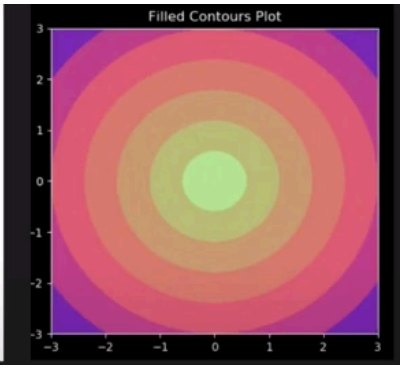
Understanding graphs



2D graph

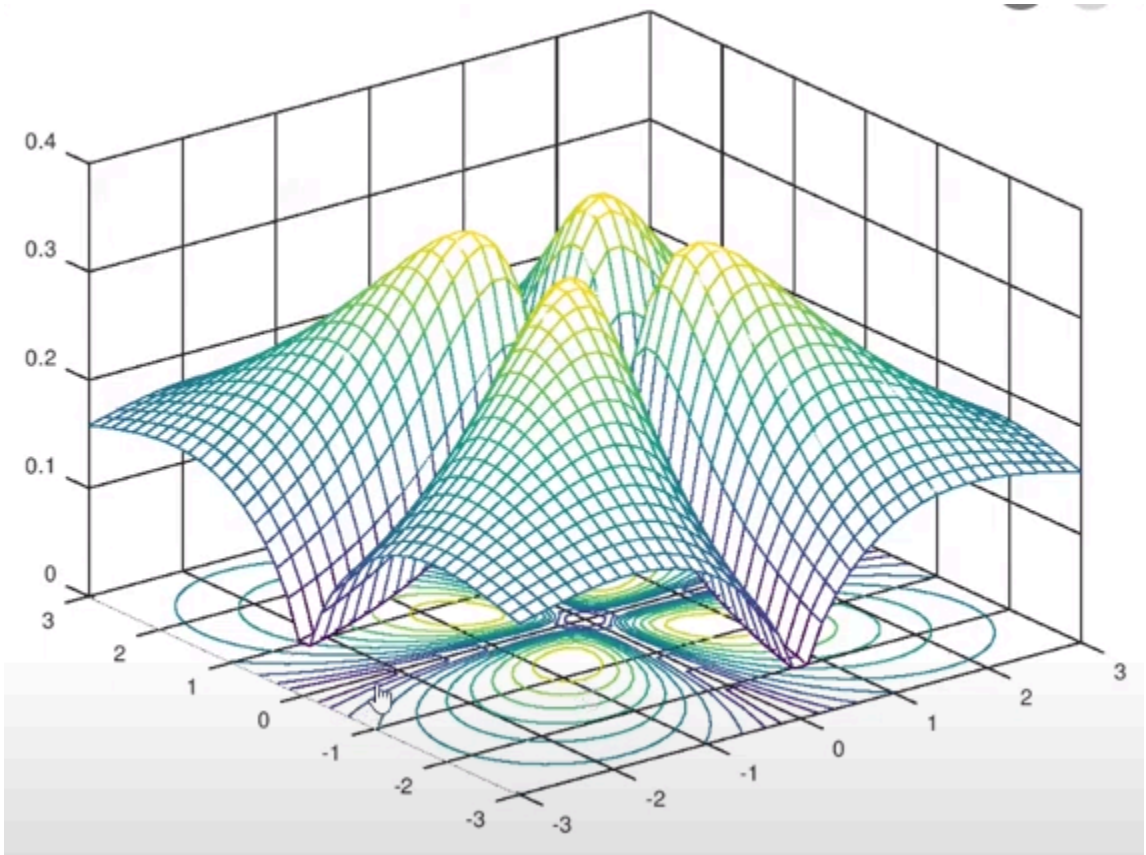


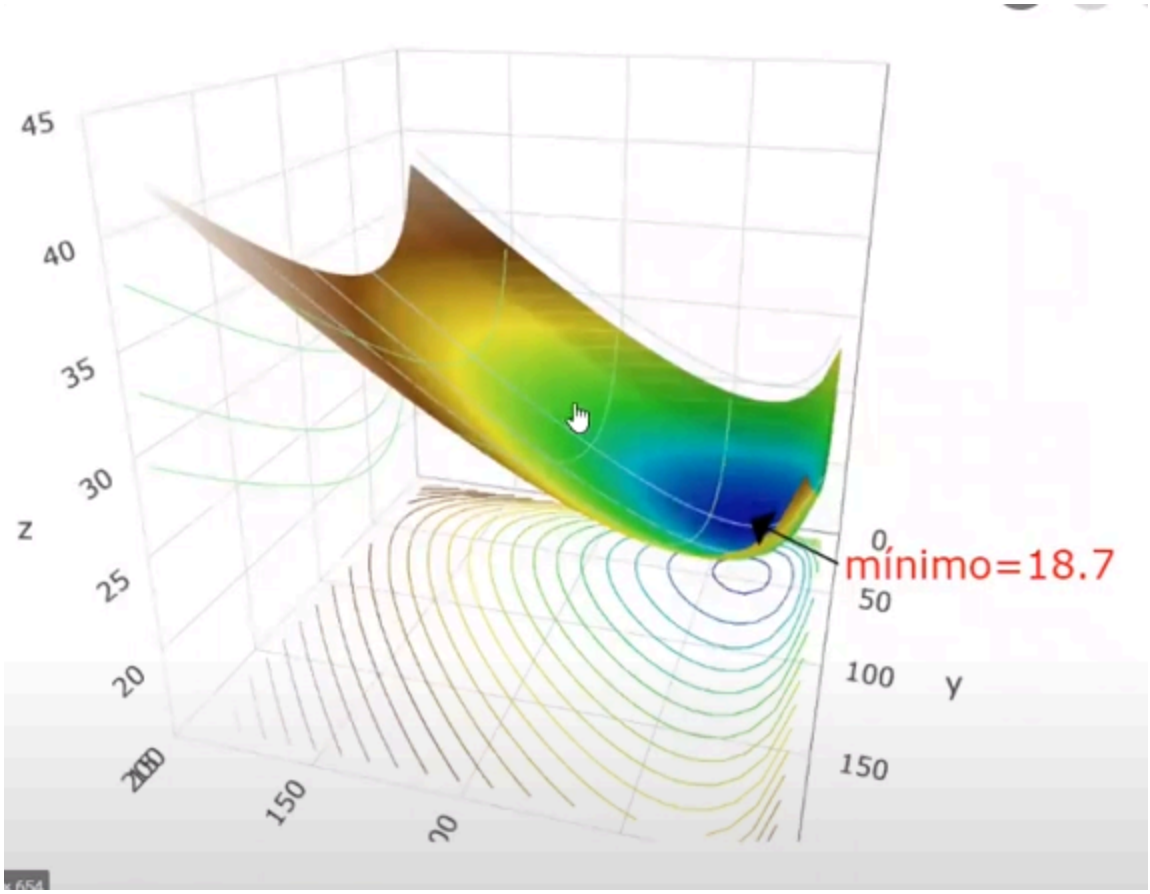
3D Graph

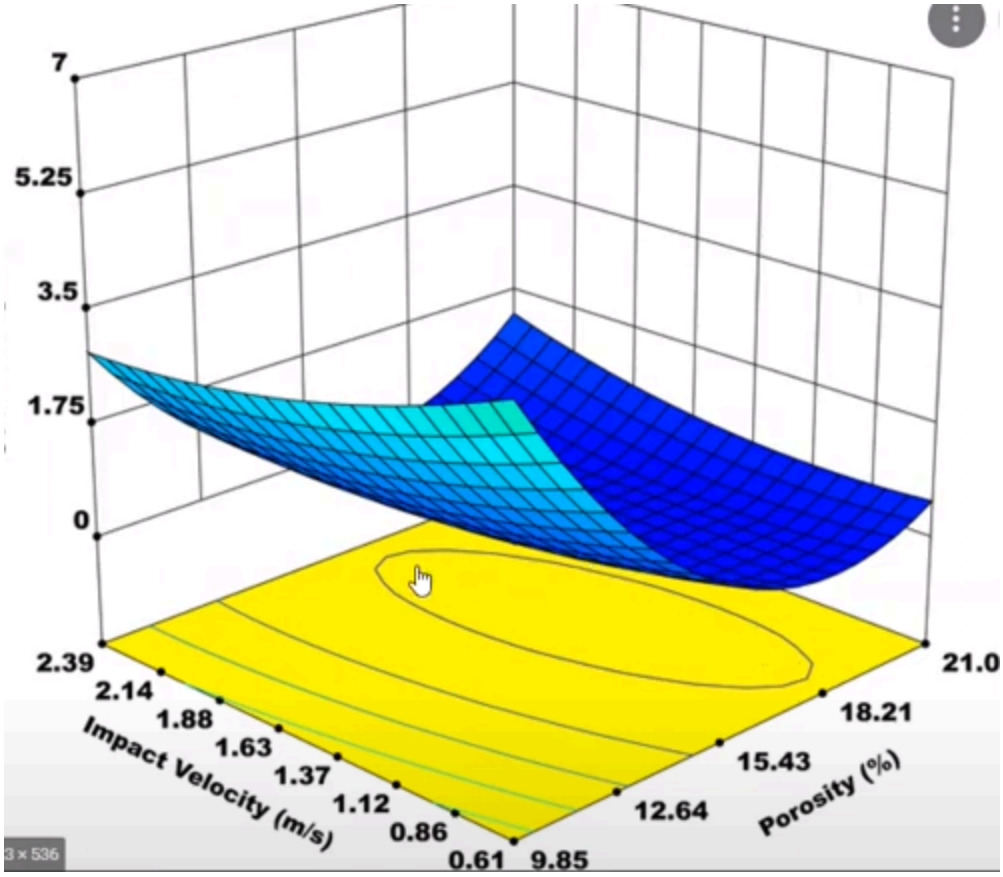


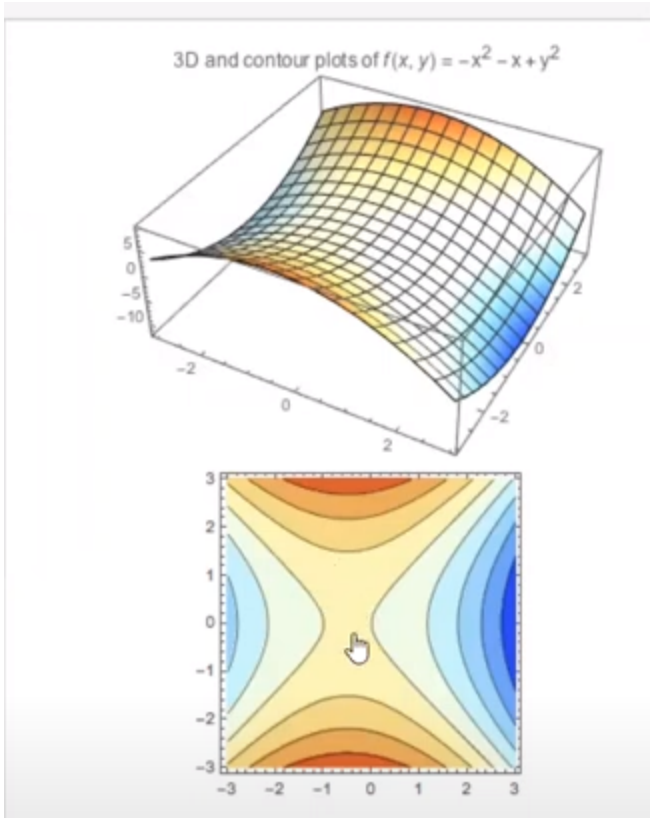
Contour Plot

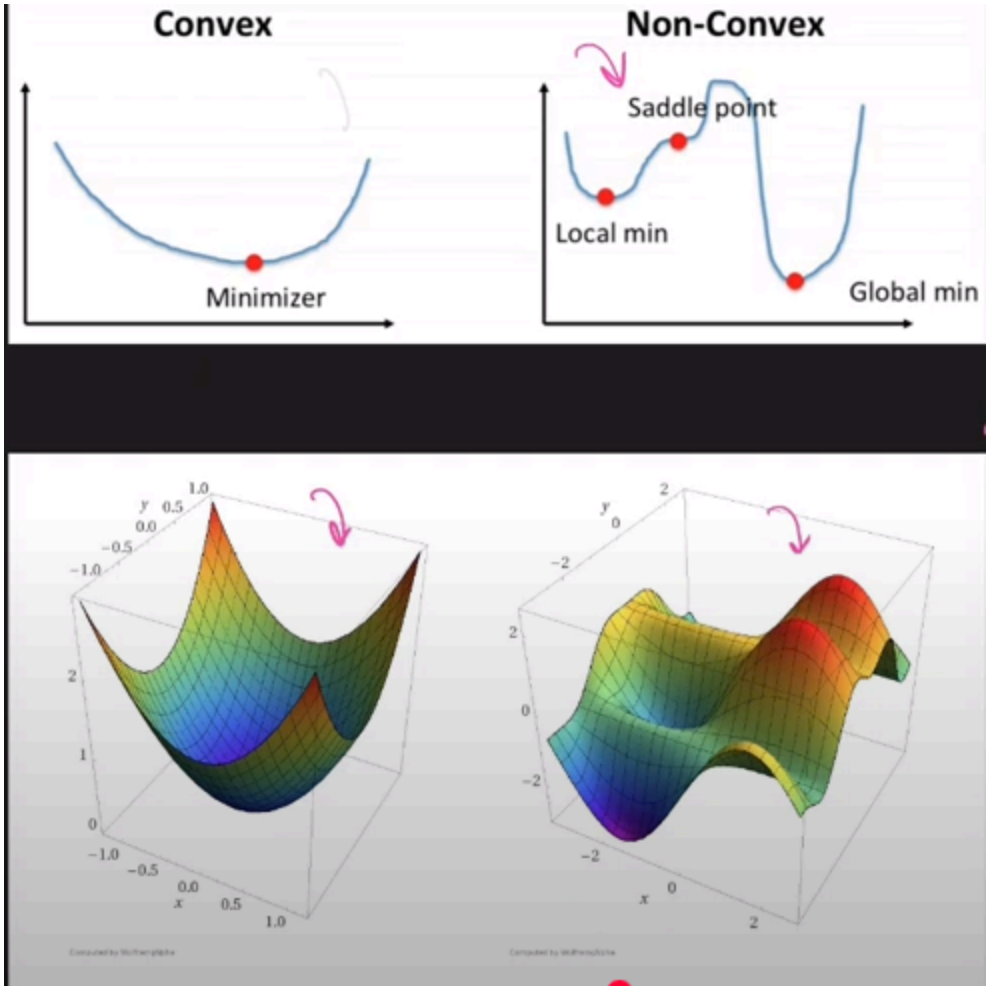
Visualizing contour




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

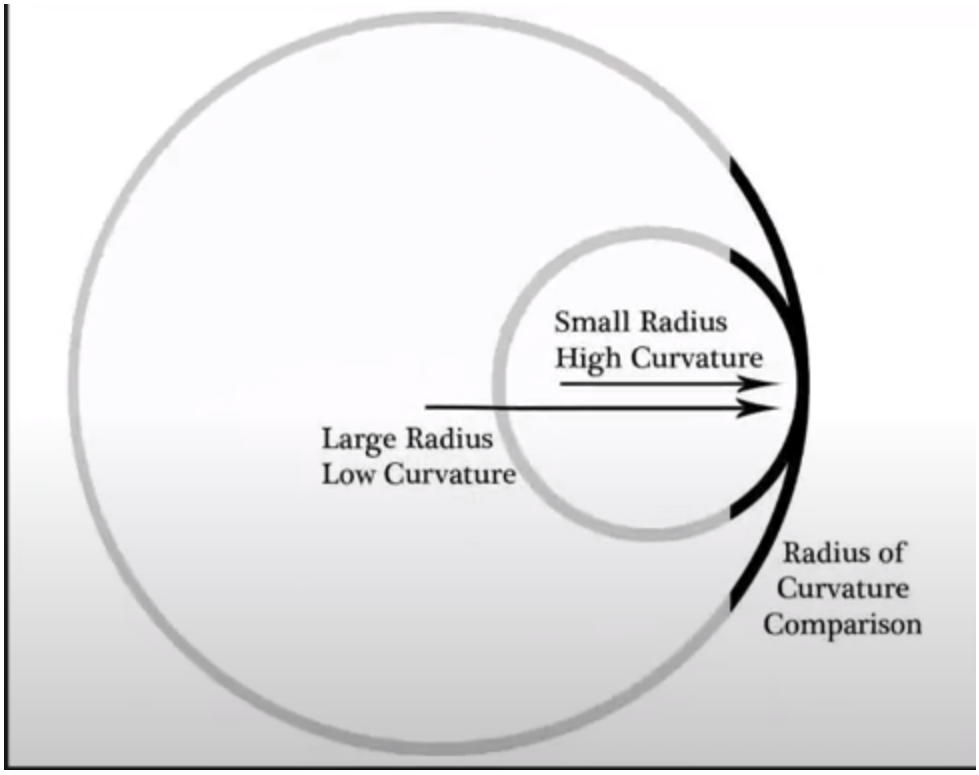

7
5.25
3.5
1.75
0
2.39
2.14
1.88
1.63
1.37
1.12
0.86
0.61
Impact Velocity (m/s)
9.85
12.64
15.43
18.21
21.0
Porosity (%)


3D and contour plots of $f(x, y) = -x^2 - x + y^2$
5
0
−5
−10
−2
0
2
2
0
−2
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3



For Non convex type of optimization problem to reach the minima it is very difficult. Due to following reason

1. Local minima
2. Saddle point
3. High Curvature



Momentum optimization technique solve the above three problems so we use this.

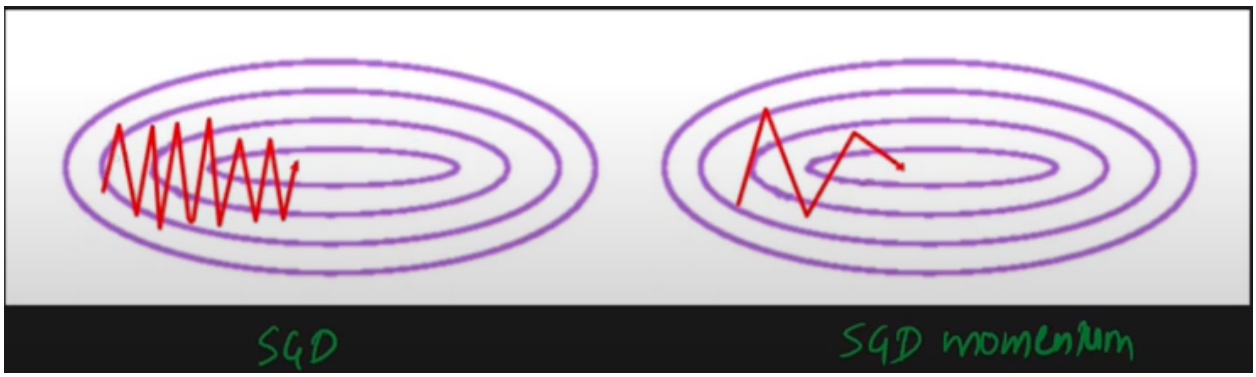
Stochastic Gradient Descent (SGD) with Momentum is an optimization technique designed to improve the efficiency and speed of the traditional Stochastic Gradient Descent method. It achieves this by incorporating a velocity term that helps accelerate convergence, especially in scenarios where the loss surface contains steep valleys or flat regions.

Key Concepts

1. **SGD Limitation**:
    - In traditional SGD, the optimization process can be slow and oscillatory, especially in regions where gradients vary significantly.
2. **Momentum Concept**:
    - Momentum introduces a memory of past gradients, smoothing the updates and reducing oscillations.

- It simulates the effect of a ball rolling down a hill, building speed in consistent directions and dampening oscillations.

3. **How Momentum Works**:
   - At each iteration, the optimizer combines the current gradient with a fraction of the previous update (velocity).



Advantages of Momentum

1. **Faster Convergence**:
   - Helps traverse flat regions and narrow valleys efficiently.
2. **Reduced Oscillations**:
   - Dampens fluctuations in the optimization path.
3. **Handles Complex Loss Surfaces**:
   - Performs well in scenarios with non-uniform gradients.

How Momentum is Gained:

1. **Incorporating Past Gradients**:
   - At each step, the optimizer remembers the *velocity* (a weighted sum of past gradients). This is updated using the formula:

$$v_t = \beta v_{t-1} - \eta \nabla L(\theta_t)$$

- Here:
  - β (momentum coefficient) determines how much of the previous velocity is carried forward.
  - $\eta \nabla L(\theta_t)$ adds the influence of the current gradient.

2. **Accumulation of Direction**:
    - If the gradients consistently point in a similar direction (e.g., downhill), the velocity term vt accumulates. This “build-up” allows the optimizer to move faster in the desired direction.
3. **Reducing Oscillations**:
    - If the gradients change direction frequently (e.g., moving across a narrow valley), the momentum helps to smooth out these changes. The optimizer retains part of its past motion, preventing drastic shifts caused by noisy gradients.
4. **Amplifying Updates**:
    - The accumulated velocity amplifies the update step, pushing the parameters further along the optimization path. This is particularly useful in flat regions or plateaus of the loss surface, where the gradients are small and progress would otherwise be slow.
5. **Decay Over Time**:
    - As the optimizer approaches a minimum, the gradients naturally become smaller. This reduces the incremental additions to vt, gradually slowing down the updates to avoid overshooting the minimum.

Intuitive Analogy:

Imagine rolling a ball down a bumpy hill:

- The ball starts slow but gains speed as it continues to roll in the downhill direction. This is analogous to the velocity term building up over consistent gradients.
- When the hill becomes flat or has small bumps, the ball’s momentum helps it glide over these regions without stopping. This resembles how momentum allows the optimizer to traverse flat regions of the loss surface.

Why Momentum is Beneficial:

1. **Accelerates Learning**:
    - By carrying forward past gradients, momentum increases the step size in directions with consistent gradients, speeding up convergence.
2. **Smooths Optimization Path**:

    - Momentum dampens oscillations caused by zig-zagging gradients, especially in narrow valleys of the loss surface.
3. **Handles Plateaus and Flat Regions**:
    - In areas where gradients are small, momentum helps maintain progress instead of getting stuck.
4. **Balances Stability and Speed**:
    - The β coefficient controls how much past velocity is retained, balancing between fast convergence and stable updates.

Summary in Simple Terms:

- Momentum is like a memory of past motion that helps the optimizer move faster in the right direction and smooth out changes.
- It works by combining the current gradient with a fraction of the previous step.
- The longer the optimizer moves in the same direction, the faster it goes, just like a ball gaining speed downhill.

2. Nesterov Accelerated Gradient Descent (NAG)

Nesterov Accelerated Gradient Descent (NAG) is an improvement over standard Momentum-based SGD. It enhances optimization by making updates more predictive, reducing oscillations, and improving convergence rates. This method provides a forward-looking mechanism to estimate gradients at a future position before updating the parameters.

Key Concepts

1. **Momentum Review**:
    - Standard momentum calculates the velocity based on the current gradient at the current parameter position .
    - It smooths updates and speeds up convergence.
2. **Nesterov Insight**:

- NAG predicts the future position of parameters using the momentum term before calculating the gradient.
- This approach anticipates the update, providing a more accurate direction for the parameter adjustment.

3. **Main Idea**:
    - Instead of computing the gradient at the current parameters , NAG computes the gradient at the “look-ahead” position, which incorporates the momentum term:

Advantages of NAG

1. **Faster Convergence**:
    - Anticipates updates, making the optimization process more efficient.
2. **Reduced Oscillations**:
    - Smooths updates in regions with steep or narrow valleys.
3. **Predictive Updates**:
    - Calculates gradients at the look-ahead position, leading to better directional accuracy.

Core Intuition: Why NAG?

Imagine you're running down a hill blindfolded (this is like optimizing a loss function). With basic **SGD**, you're just using the gradient at your current location to decide the next step. With **Momentum**, you factor in your past velocity, so you "roll" downhill faster. However, this can sometimes cause overshooting because you're always reacting to your current position.

NAG improves on this by **peeking ahead**. Instead of blindly calculating the gradient at your current position, NAG asks:
"If I took a step in the direction of my momentum, what would the gradient look like?"

By estimating the gradient at this “look-ahead” position, NAG makes smarter updates, avoiding overshooting and improving convergence.

Note: Copy

3. Ada Grad (Adaptive Gradient)

Adagrad, short for Adaptive Gradient Algorithm, is an optimization method designed to adaptively adjust the learning rate for each parameter during training. Its main goal is to improve performance on problems with sparse gradients or varying feature importance.

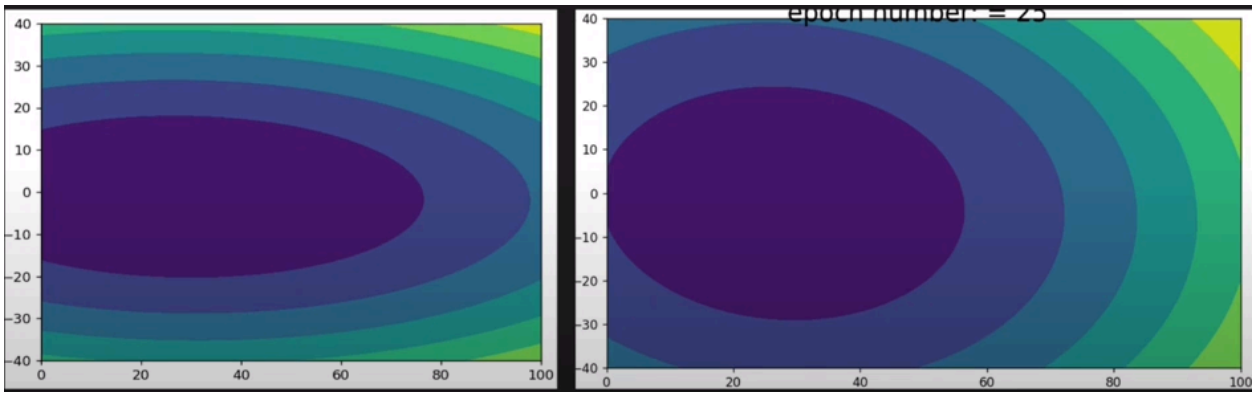
Why Adagrad?

In traditional gradient descent methods (like SGD), we use a fixed learning rate η for all parameters throughout training. However:

1. Different Features, Different Scales: Some features might be more sensitive than others, requiring smaller updates to avoid overshooting.
2. Sparsity: Features that appear rarely in the data may have very small gradients and require larger learning rates to make meaningful progress.

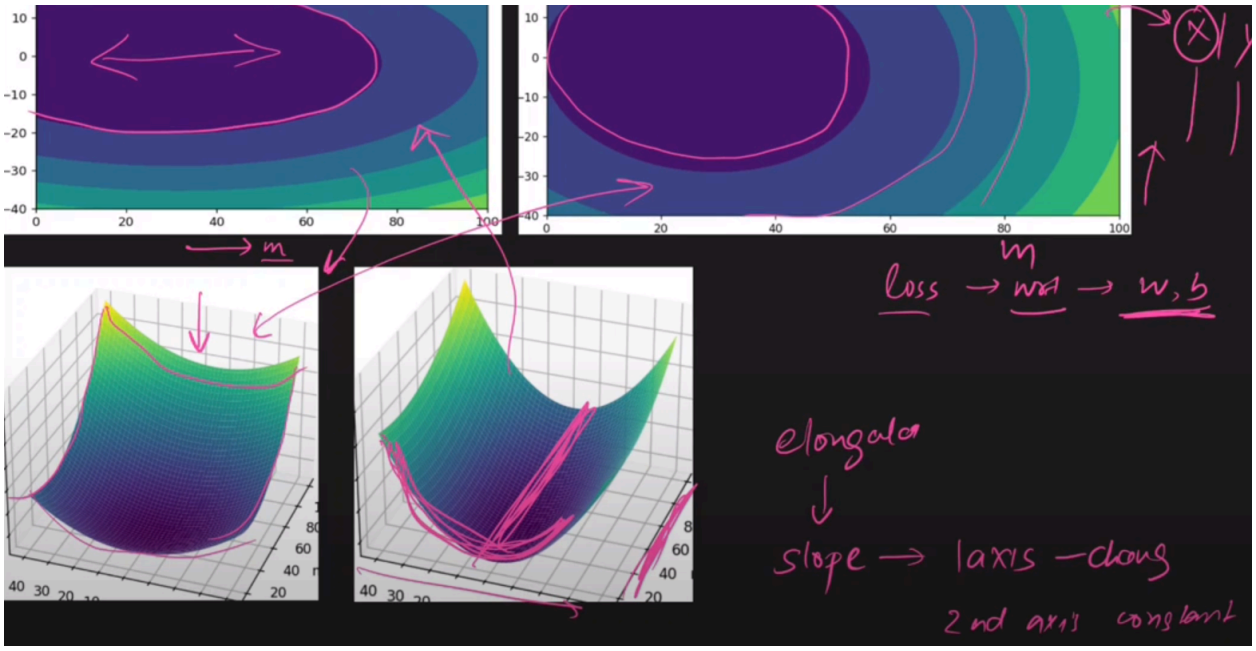
Adagrad solves these issues by scaling the learning rate for each parameter based on its historical gradient accumulation. Features with large gradients are scaled down, while those with small gradients are scaled up.

Elongated Bowl Problem



Sparse data set contour

normal dataset contour



Advantages of Adagrad

1. **Handles Sparse Data Well**:
   Ideal for problems like natural language processing (NLP) or recommendation systems where features may appear infrequently.
2. **No Manual Learning Rate Tuning**:
   Each parameter has its own learning rate that adapts over time.
3. **Stability**:
   Automatically scales updates, reducing the risk of overshooting.

Limitations of Adagrad

1. **Aggressive Decay**:
   Accumulated gradients keep growing, causing learning rates to shrink too much over time, which can stall progress.

4. RMSprop (Root Mean Square Propagation)

RMSprop is a popular optimization algorithm designed to address the **limitation of Adagrad**, which is the rapid decay of learning rates over time. It introduces a mechanism to control the effect of past gradients by using an **exponential moving average (EMA)** of squared gradients. This makes RMSprop highly effective for non-convex problems, such as training deep neural networks.

Why RMSprop?

Adagrad adjusts the learning rate for each parameter based on the sum of all past squared gradients. While this works well for sparse features, the **accumulated squared gradients can grow too large**, causing the learning rates to become excessively small, which slows or even halts learning.

RMSprop addresses this by:

1. Using a **moving average** of squared gradients rather than summing all gradients.
2. Giving more importance to recent gradients and diminishing the influence of older ones.

Key Intuition Behind RMSprop

1. **Control Learning Rate Decay**:
   By using an EMA of squared gradients, RMSprop ensures that the learning rate does not decrease indefinitely as in Adagrad. This allows for continued learning even in prolonged training processes.
2. **Feature Sensitivity**:
   Features with large gradients (frequent updates) are scaled down, while features with smaller gradients retain relatively larger effective learning rates.
3. **Smoother Optimization**:
   RMSprop effectively dampens oscillations in the parameter updates, particularly in directions with steep gradients, leading to more stable convergence.

Advantages of RMSprop

1. **No Aggressive Decay**:
   Learning rates remain dynamic without decaying excessively, unlike Adagrad.
2. **Handles Sparse Data**:
   Efficient for features with varying gradient frequencies, such as NLP and image data.
3. **Less Hyperparameter Tuning**:
   With a well-chosen β, RMSprop requires less manual adjustment of learning rates.

5. Adam Optimizer (Adaptive Moment Estimation)

Adam is one of the most widely used and effective optimization algorithms in deep learning. It combines the best properties of two popular optimization techniques: **Momentum** and **RMSprop**. Adam not only adapts the learning rate for each parameter but also incorporates the concept of momentum to smooth updates, making it robust and efficient for a wide variety of machine learning tasks.

Why Adam?

Adam stands out because it:

1. Combines the advantages of **momentum** (accelerating convergence) and **RMSprop** (adaptive learning rates).
2. Handles sparse gradients and noisy objectives efficiently.
3. Requires minimal hyperparameter tuning.

It is especially useful for training large-scale deep learning models with non-stationary objectives.