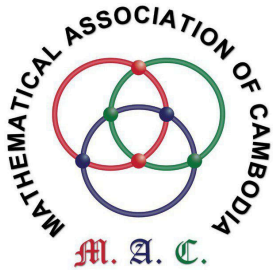
Gradient Descent Algorithm

KIMSIE PHAN

M.Sc Mathematics and Computing at IIT DHANDAB, INDIA

phankimsie03@gmail.com



Mathematical Association of Cambodia

May 10, 2025

Let's find minimizer point of function $f(x) = x^2 - 2x$.

Let's find minimizer point of function $f(x) = x^2 - 2x$.

How to find minimizer point of function
in multiple dimension ?

eg: $f(x_1, x_2) = x_1^2 + x_2^2 + 4x_1 - 6x_2$

Let's find minimizer point of function $f(x) = x^2 - 2x$.

How to find minimizer point of function
in multiple dimension ?
eg: $f(x_1, x_2) = x_1^2 + x_2^2 + 4x_1 - 6x_2$

$x_* = \arg\min_{x \in \mathbb{R}^n} f(x)$ in $n-$dimensional space!

I. Introduction

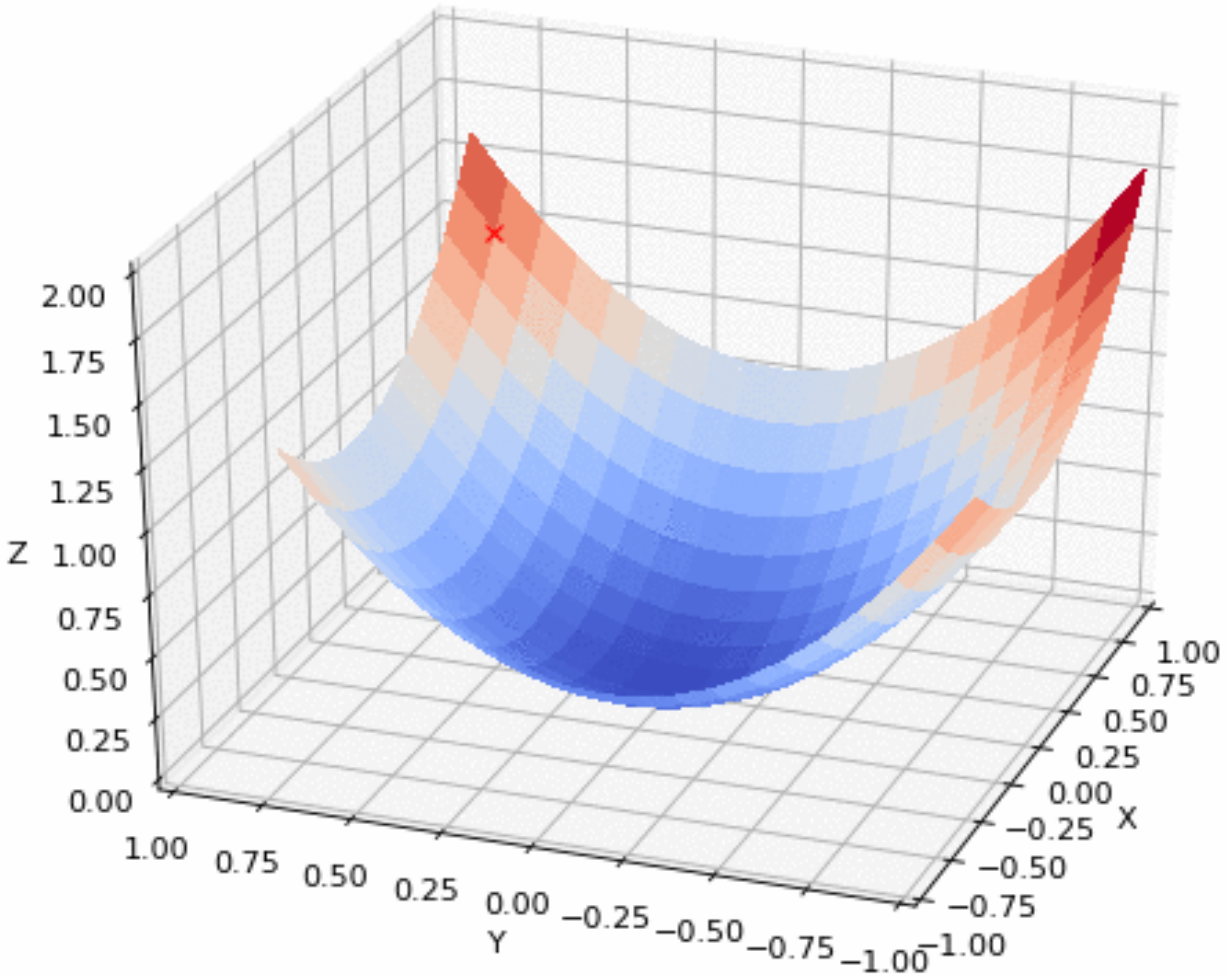
Gradient Descent is an optimization algorithm used to minimize a given objective function, often called a cost or loss function, by iteratively the parameters of the model. The essence of Gradient Descent lies in its ability to navigate the parameter space by following the direction of the steepest descent, guided by the gradient of the function. The algorithm updates the model parameters in small steps, guided by the gradient, to gradually converge to the minimum value of the objective function, where the model is most accurately fitted.

I. Introduction

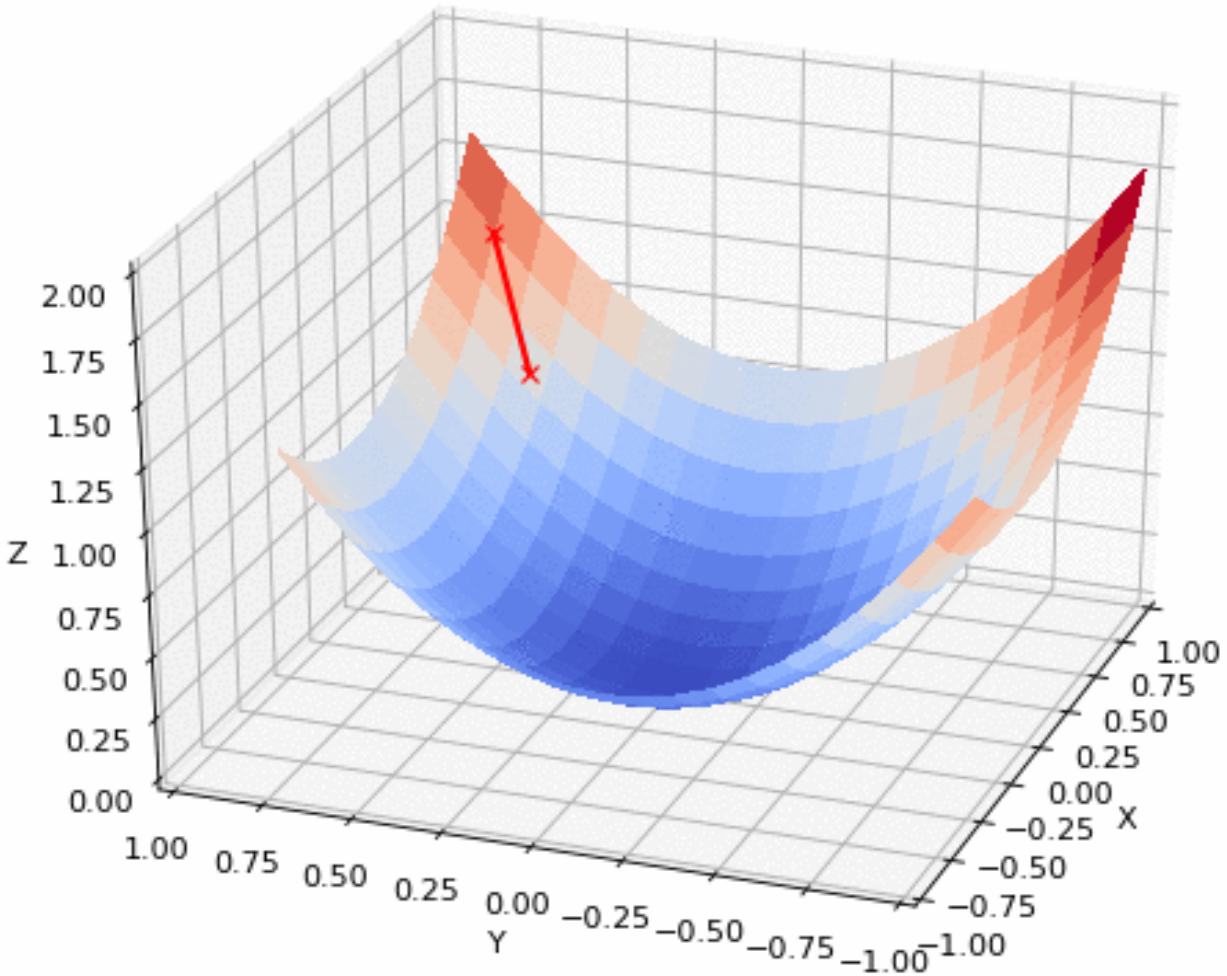
Gradient Descent is an optimization algorithm used to minimize a given objective function, often called a cost or loss function, by iteratively the parameters of the model. The essence of Gradient Descent lies in its ability to navigate the parameter space by following the direction of the steepest descent, guided by the gradient of the function. The algorithm updates the model parameters in small steps, guided by the gradient, to gradually converge to the minimum value of the objective function, where the model is most accurately fitted.

II. Motivation

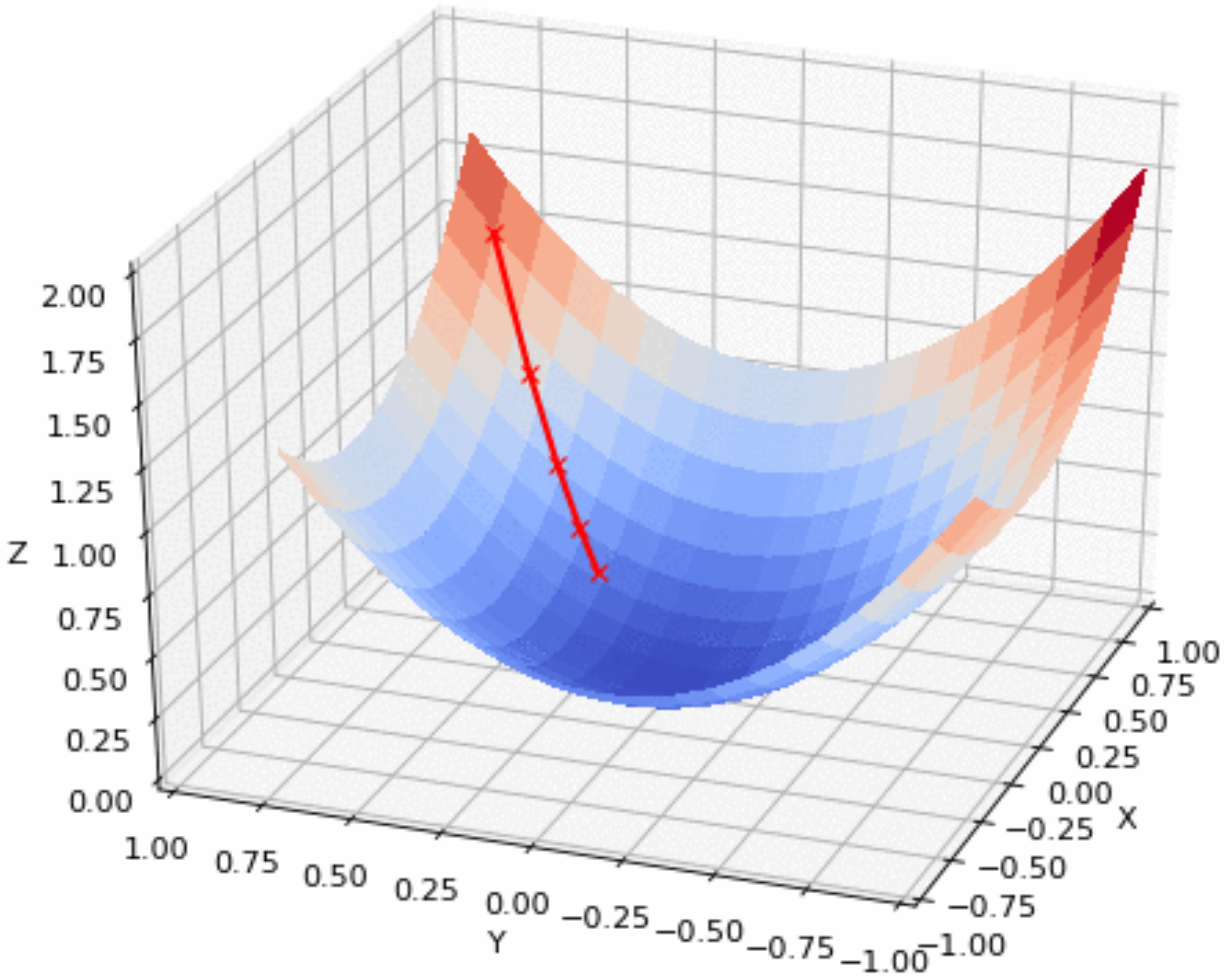
The primary goal of Gradient Descent is to minimize an objective function that is parameterized by a model's parameters. This is achieved by iteratively updating the parameters in the opposite direction of the gradient of the objective function with respect to those parameters.



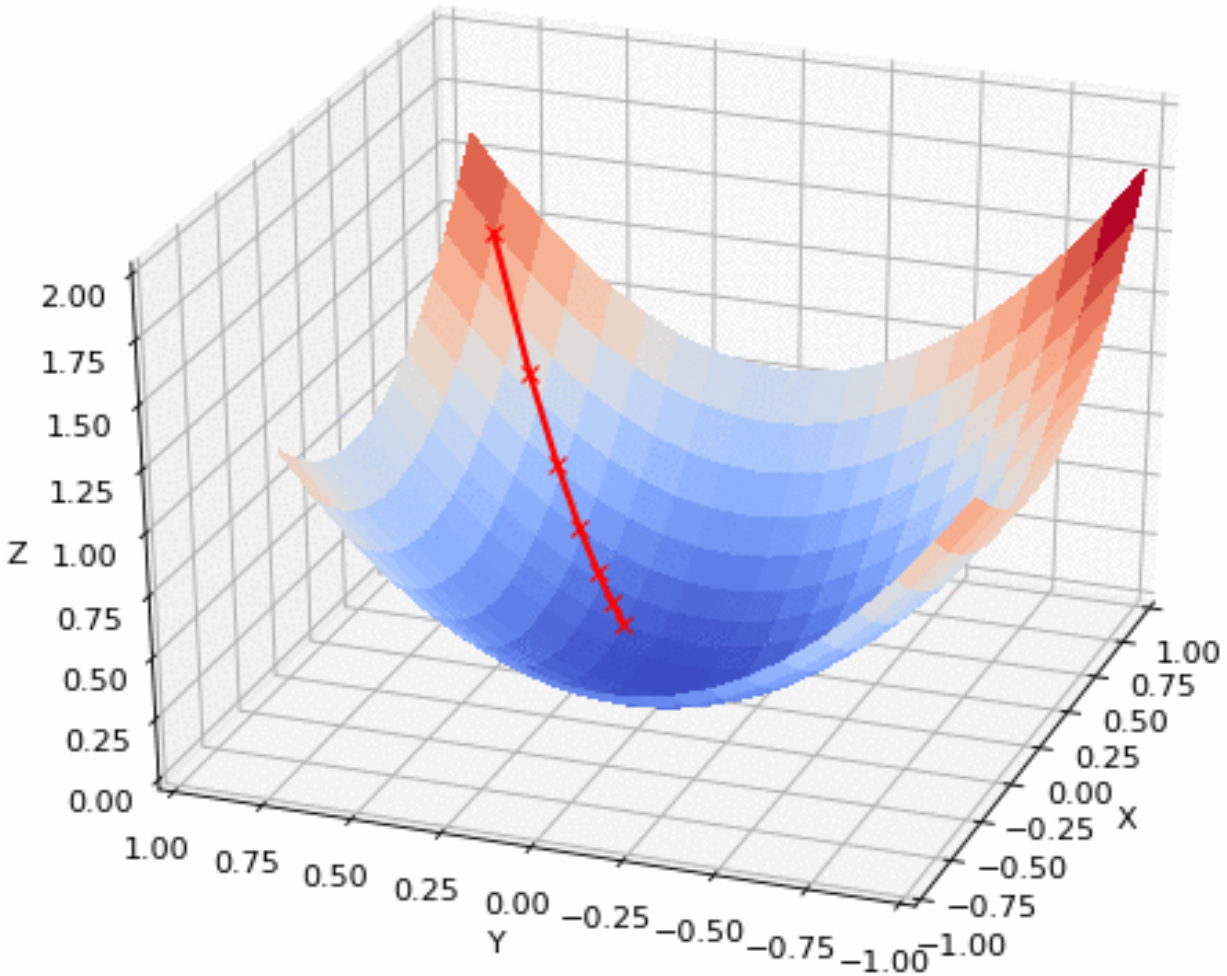
Gradient Descent in Action



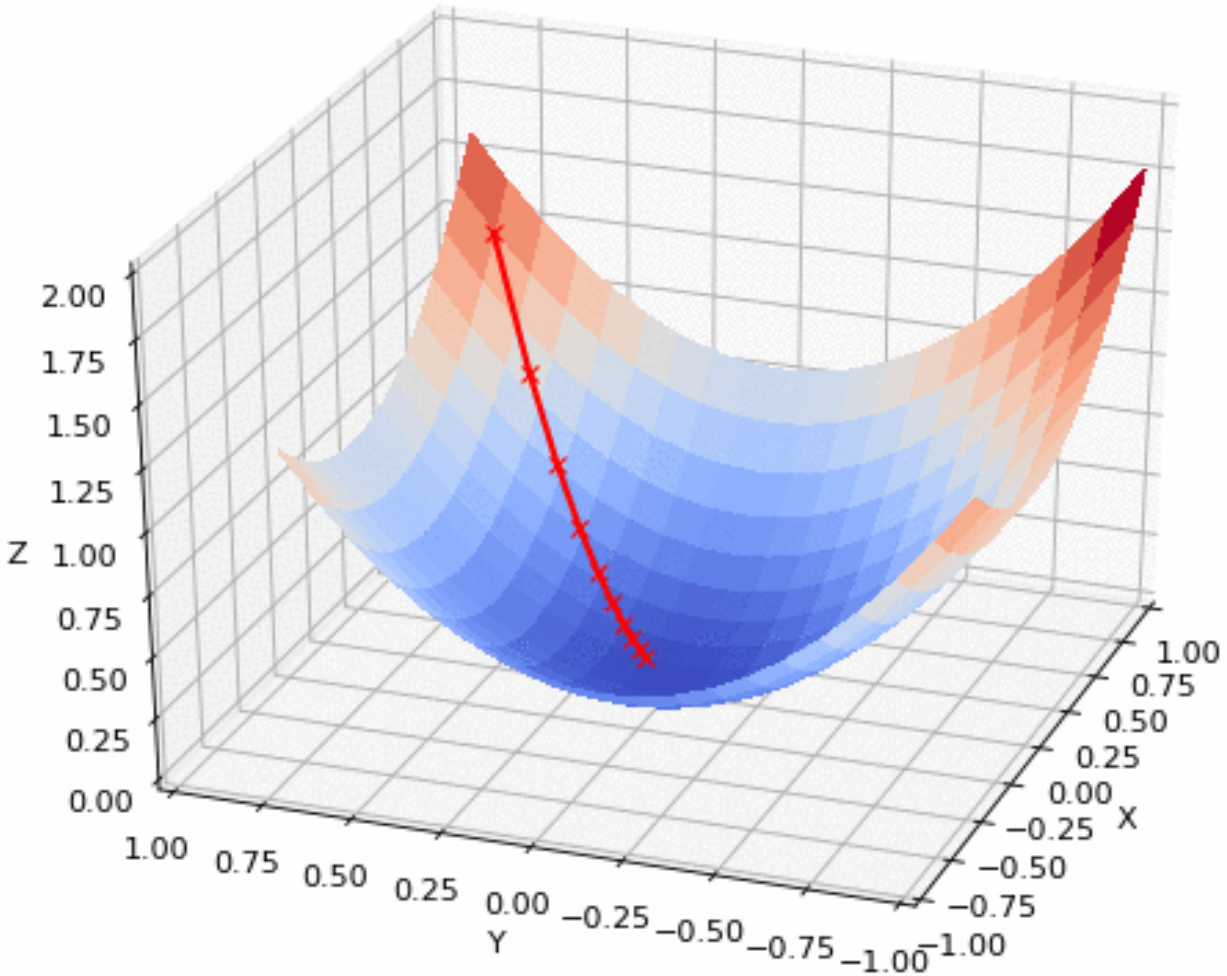
Gradient Descent in Action



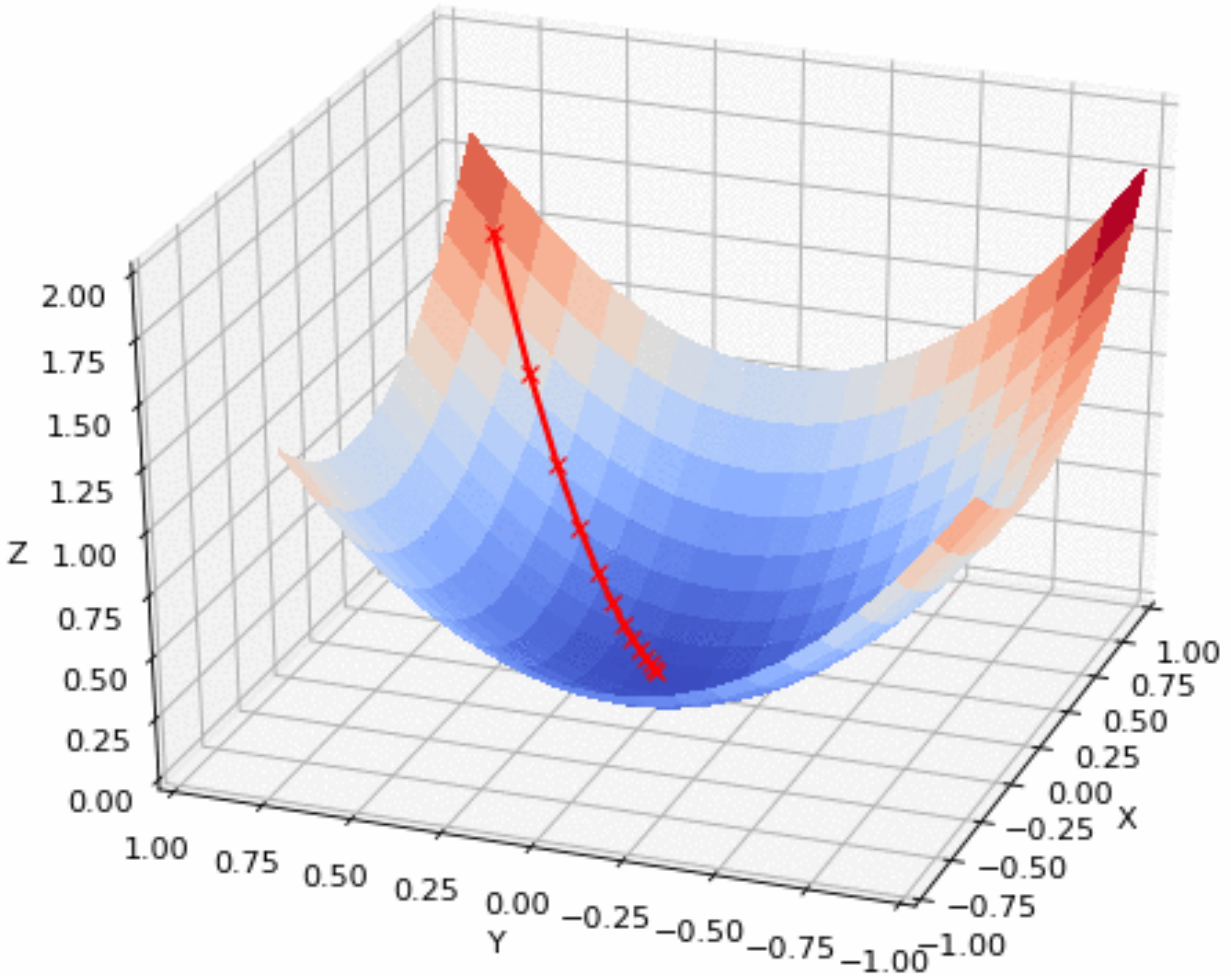
Gradient Descent in Action



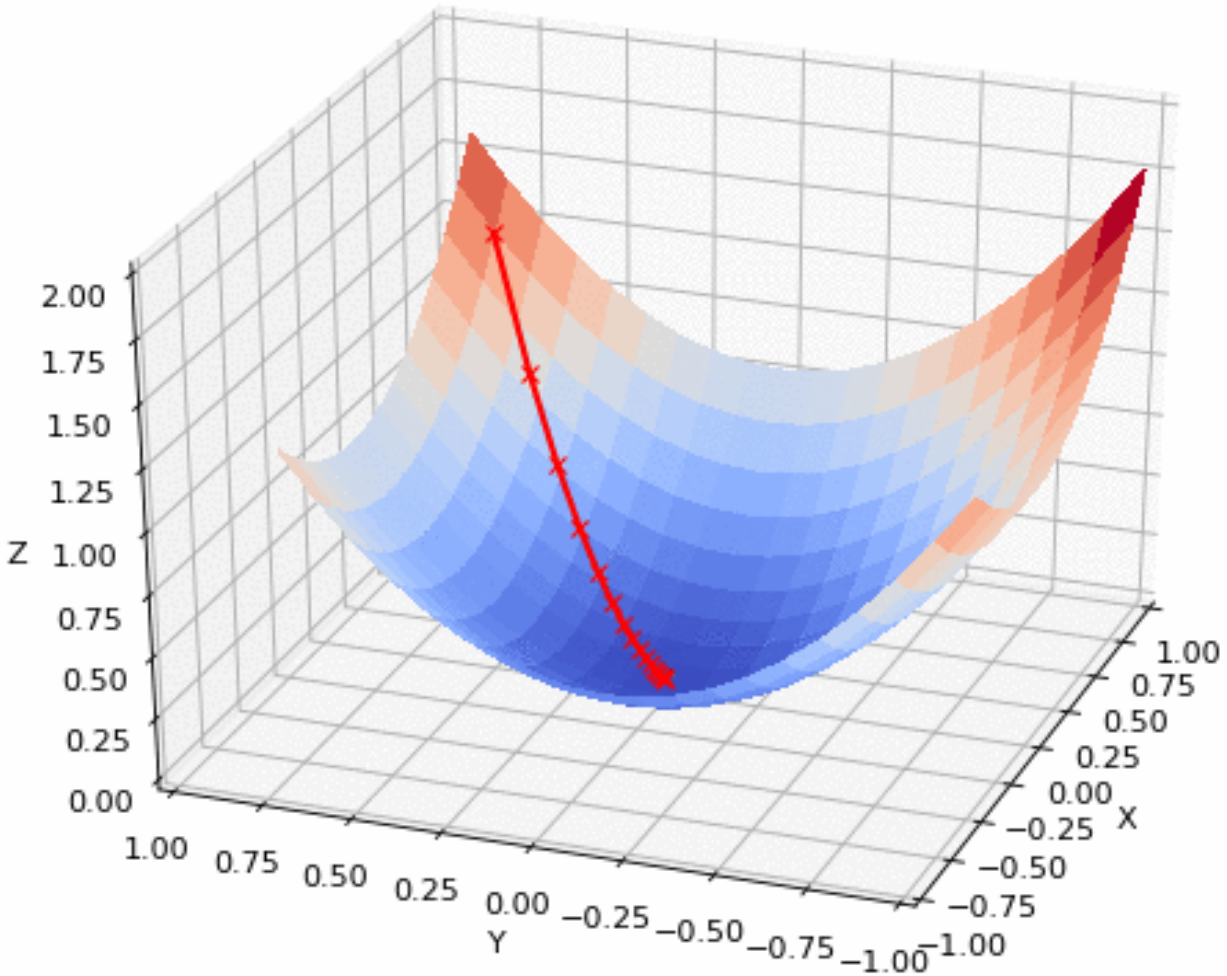
Gradient Descent in Action



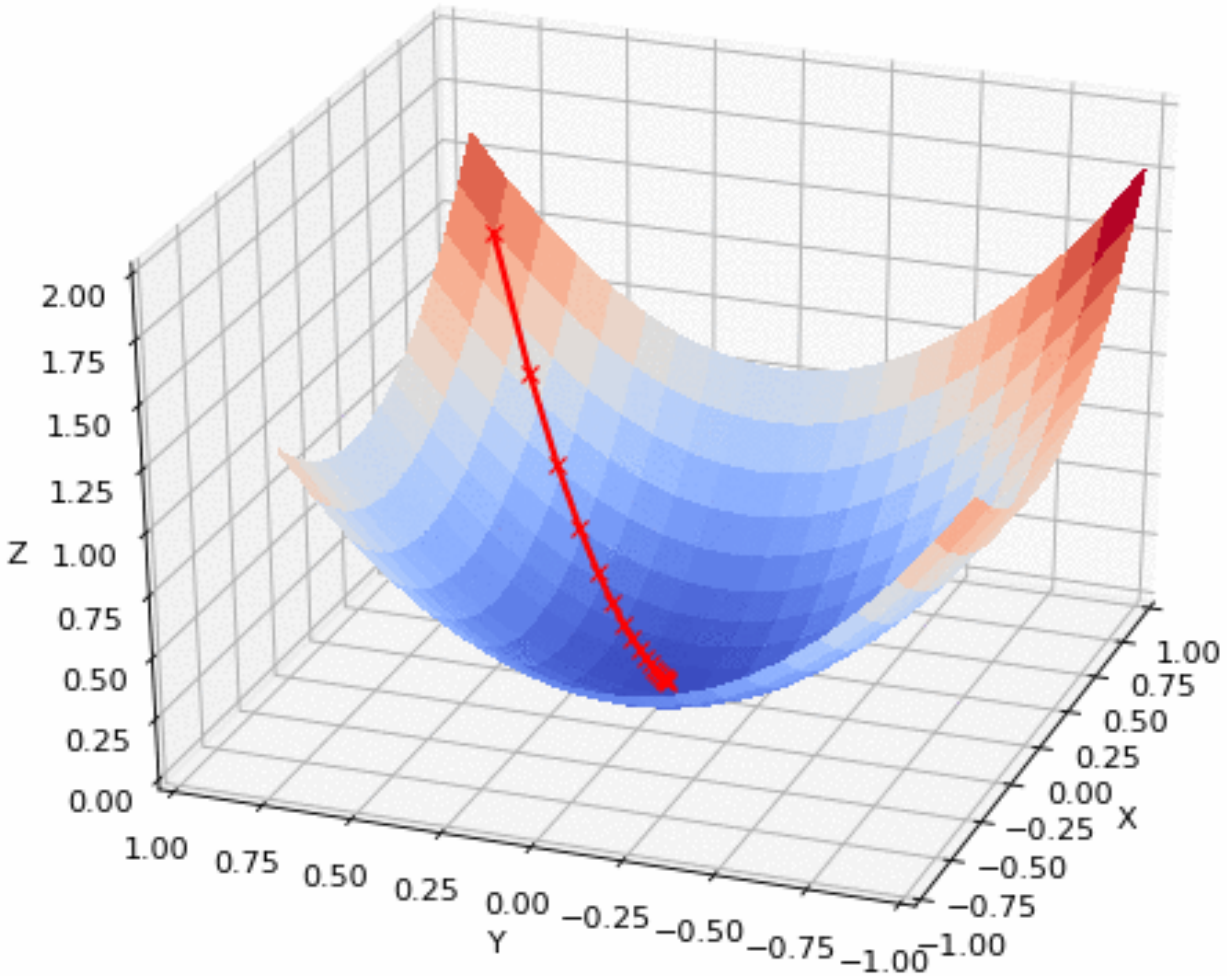
Gradient Descent in Action



Gradient Descent in Action



Gradient Descent in Action



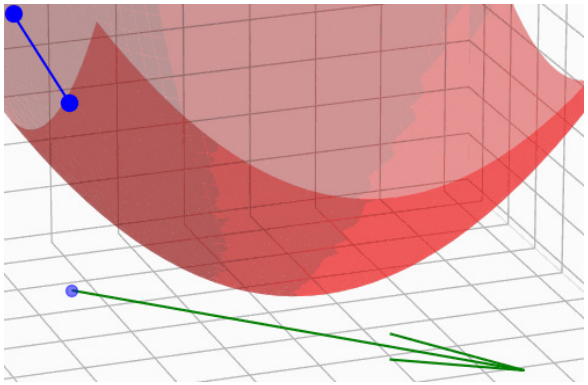
Gradient Descent in Action

Recall:

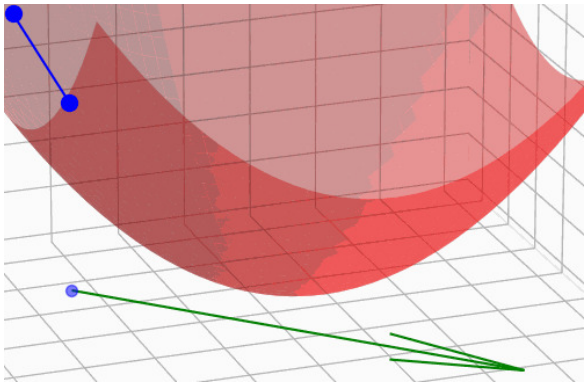
Definition (Gradient)

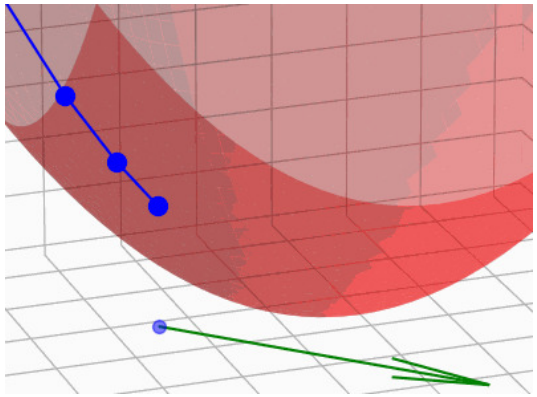
A gradient is a vector that represents the direction and rate of the steepest increase of a function. Define function $f : \mathbb{R}^n \to \mathbb{R}^n$ be differentiable. For any $x \in \mathbb{R}^n$, the **gradient** of function f at x is defined by

$$\nabla f(x_1, x_2, ..., x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n} \right) \in \mathbb{R}^n.$$

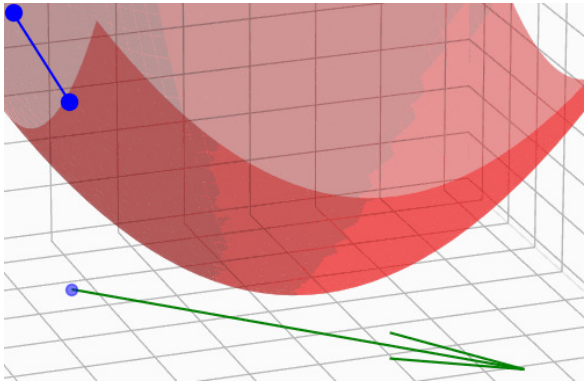


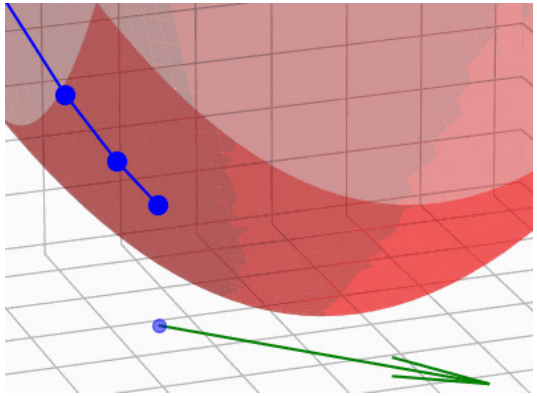
$$-\nabla f(x)$$

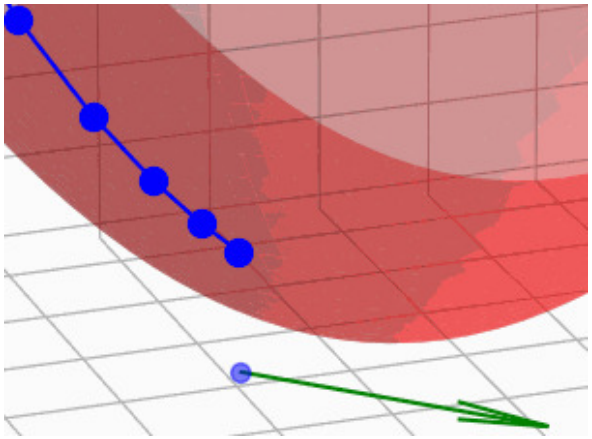




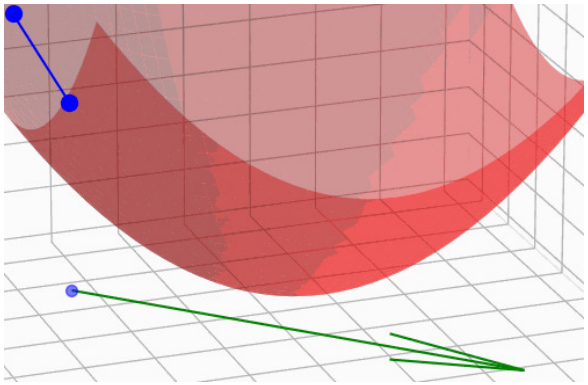
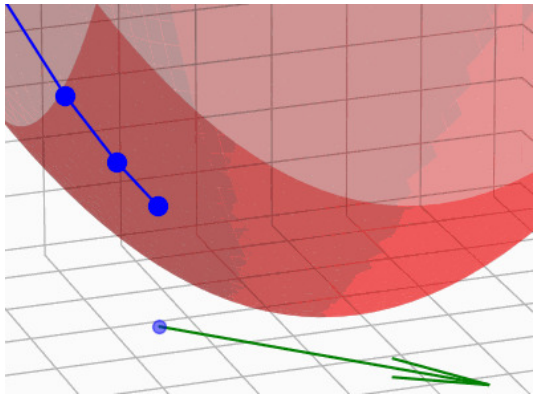
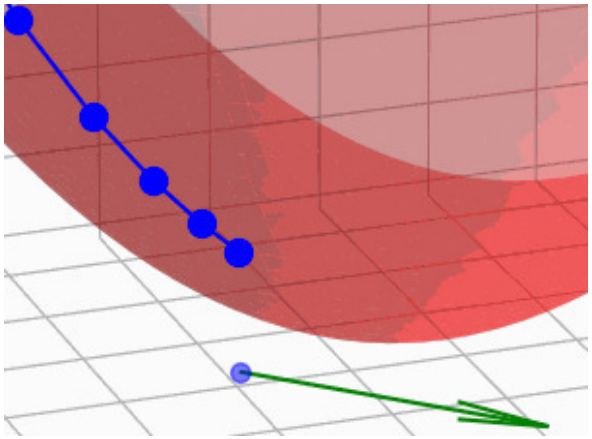
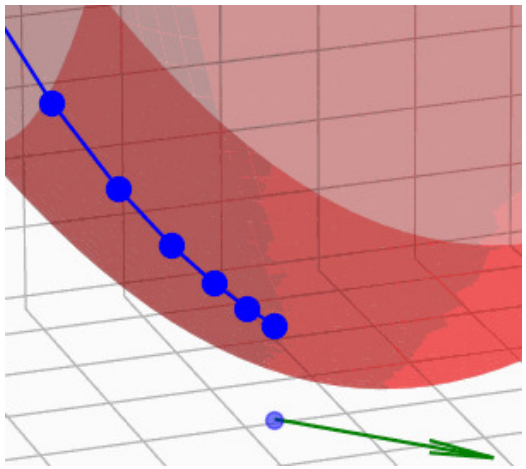
$$-\nabla f(x)$$







$$-\nabla f(x)$$

$$-\nabla f(x)$$

Definition (Gradient Descent)

For a given objective function $J(\theta)$, where θ represents the parameters, the update rule for **Gradient Descent** is

$$\theta := \theta - \eta \nabla_\theta J(\theta).$$

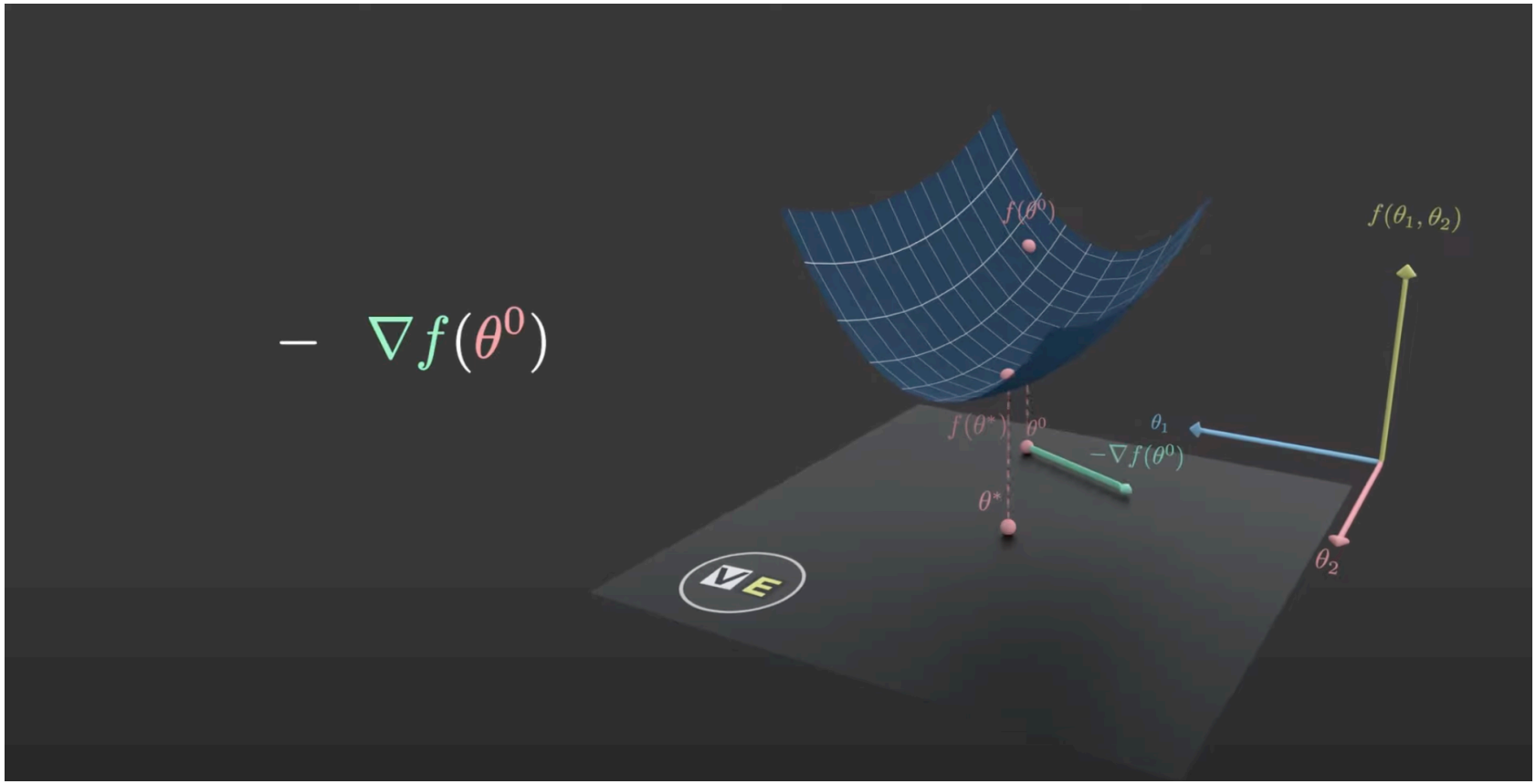
- θ is the parameters.
- $\eta > 0$ is the learning rate, or stepsizes.
- $\nabla_\theta J(\theta)$ is the direction or gradient of the cost function w.r.t θ.

Algorithm GradientDescent

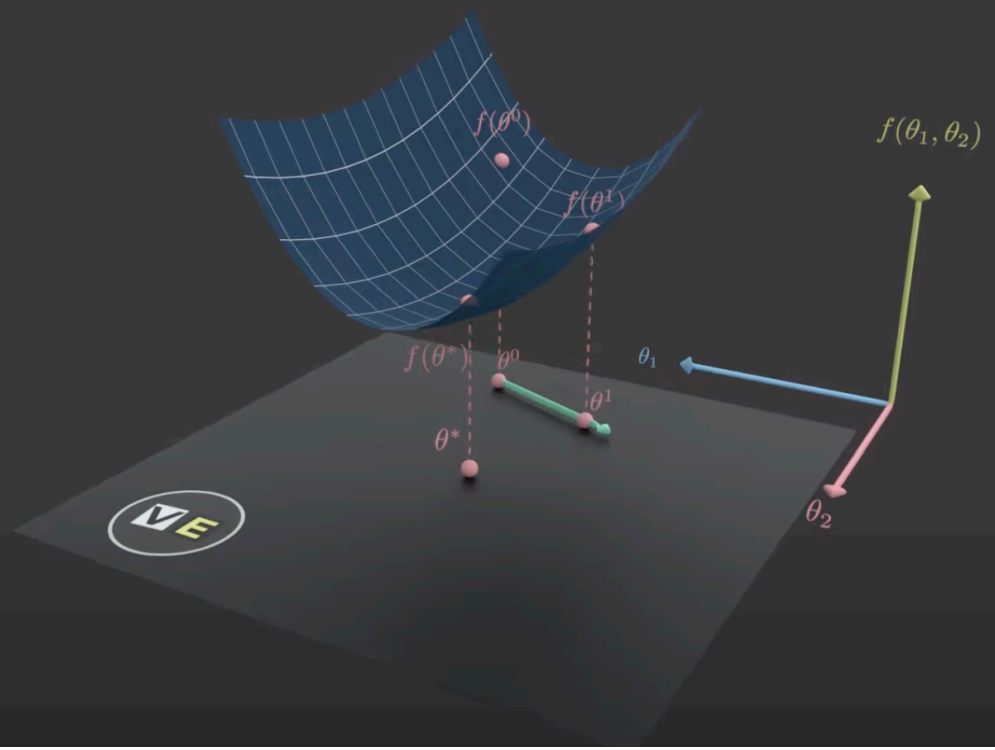
Input: Learning rate η, number of iterations N, initial parameters θ
Output: Optimized parameters θ

for i = 1 to N do
  Compute gradient $\nabla\theta\, J(\theta)$
  Update $\theta := \theta - \eta * \nabla\theta\, J(\theta)$
  (Optional) Compute the cost function $J(\theta)$ to monitor convergence
  (Optional) Check if $|\theta\_new - \theta\_old| <$ threshold, then break
end for

return θ
End Algorithm

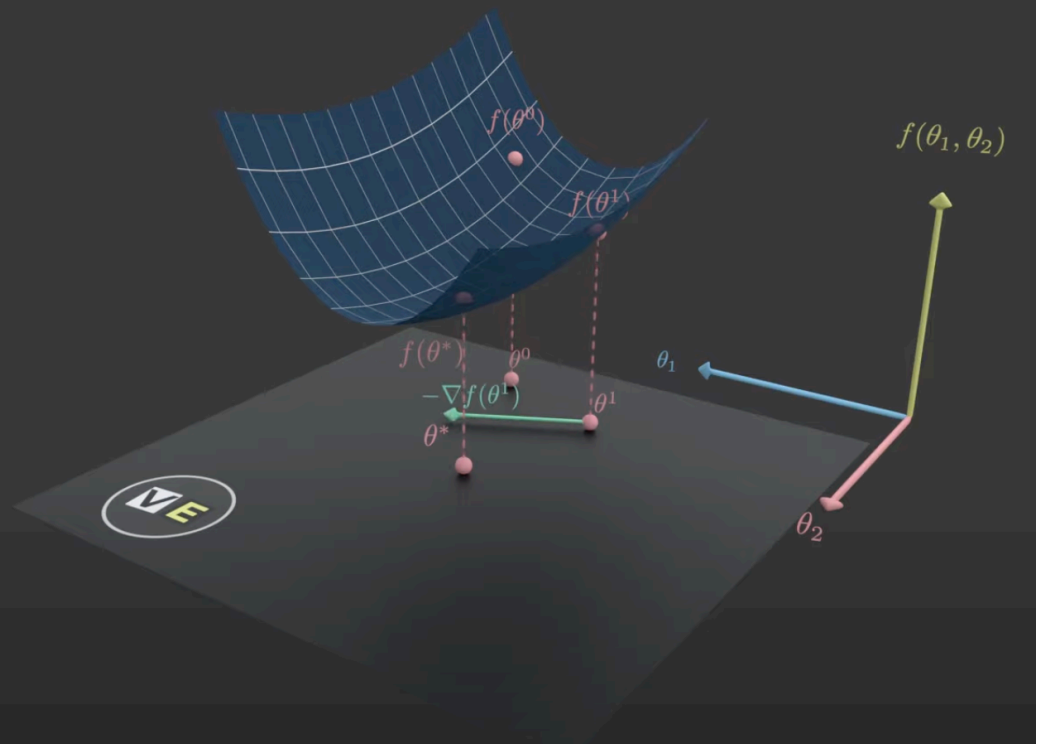

$-\nabla f(\theta^0)$
$f(\theta^0)$
$f(\theta_1, \theta_2)$
$f(\theta^*)$
θ^0
θ_1
$-\nabla f(\theta^0)$
θ^*
θ_2

$$\theta^1 = \theta^0 - \eta \nabla f(\theta^0)$$



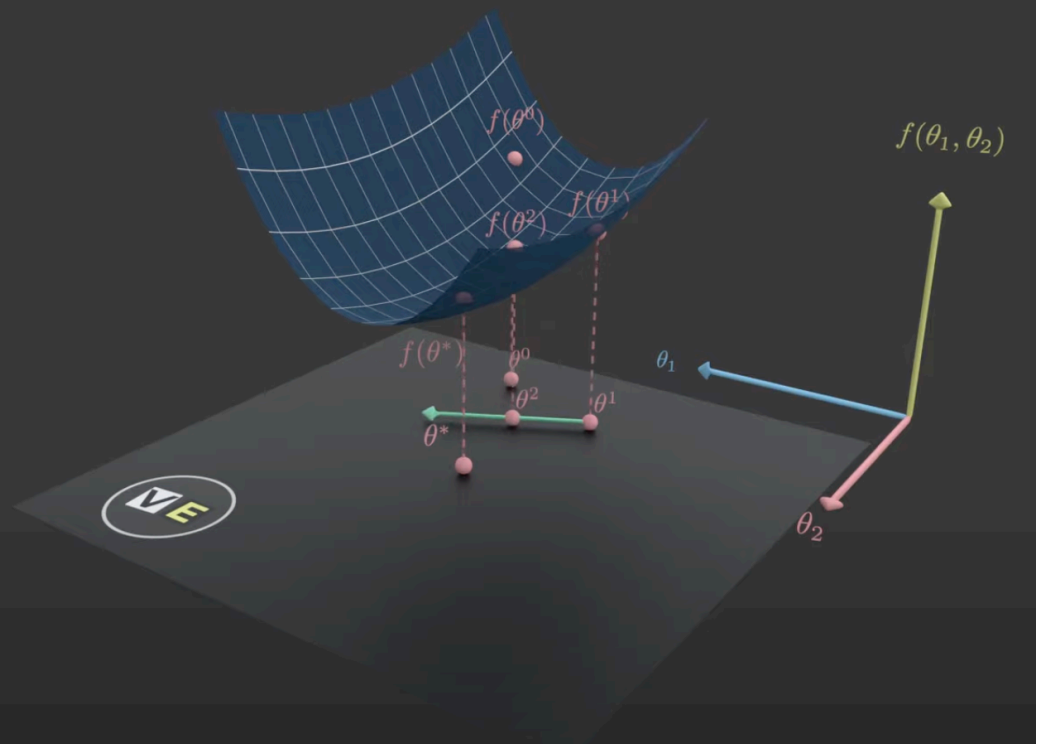
$$\theta^1 = \theta^0 - \eta \nabla f(\theta^0)$$

$$- \quad \nabla f(\theta^k)$$



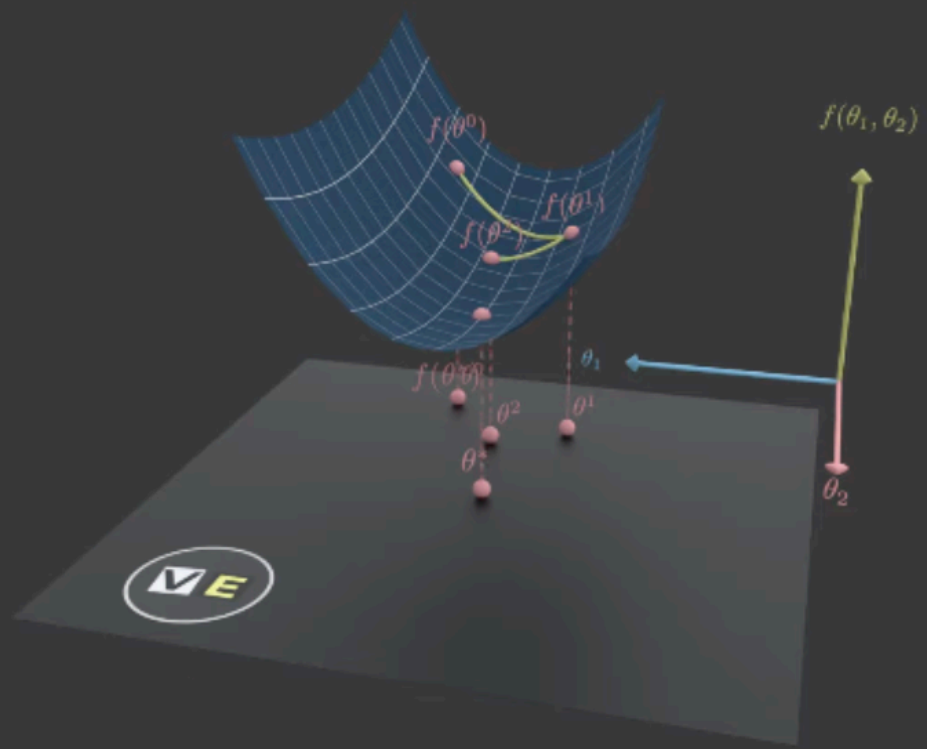
$$\theta^1 = \theta^0 - \eta \nabla f(\theta^0)$$

$$\theta^{k+1} = \theta^k - \eta \nabla f(\theta^k)$$



$$\theta^1 = \theta^0 - \eta \nabla f(\theta^0)$$

$$\theta^{k+1} = \theta^k - \eta \nabla f(\theta^k)$$



Choice of Stepsizes

The step size, also called the learning rate is a crucial parameter in gradient descent algorithms. It determines the size of the steps taken towards the minimum of the cost function in each iteration.

Choice of Stepsizes

The step size, also called the learning rate is a crucial parameter in gradient descent algorithms. It determines the size of the steps taken towards the minimum of the cost function in each iteration.

Let the stepsizes $(\alpha_t)_{t\in\mathbb{N}}$ is crucial :

- If they are too large, then θ_{t+1} is outside the domain, and the algorithm may diverge.
- If they are too small, θ_t needs many time steps to move away from θ_0, and convergence can be slow.

Choice of Stepsizes

The step size, also called the learning rate is a crucial parameter in gradient descent algorithms. It determines the size of the steps taken towards the minimum of the cost function in each iteration.

Let the stepsizes $(\alpha_t)_{t\in\mathbb{N}}$ is crucial :

- If they are too large, then θ_{t+1} is outside the domain, and the algorithm may diverge.
- If they are too small, θ_t needs many time steps to move away from θ_0, and convergence can be slow.

What a good stepsize choice is depends on the properties of J.

1. *Fixed schedule*: α_t generally depends on t through a simple equation,

$$\begin{aligned} \forall t, \quad & \alpha_t = \eta \text{ for some } \eta > 0, && \text{(Constant stepsize)} \\ \text{or } \forall t, \quad & \alpha_t = \tfrac{1}{t+1}. && \text{(Monotonically decreasing stepsize)} \end{aligned}$$

2. *Exact line search*: for any t, choose α_t such that

$$J\left(\theta_t - \alpha_t \nabla J(\theta_t)\right) = \min_{a \in \mathbb{R}} J\left(\theta_t - a \nabla J(\theta_t)\right).$$

2. *Exact line search*: for any t, choose α_t such that

$$J\left(\theta_t - \alpha_t \nabla J(\theta_t)\right) = \min_{a \in \mathbb{R}} J\left(\theta_t - a \nabla J(\theta_t)\right).$$

3. *Backtracking line search*: Choose α_t such that $J\left(\theta_t - \alpha_t \nabla J(\theta_t)\right)$ is sufficiently smaller than $J(\theta_t)$. Implies, for α_t small enough,

$$J\left(\theta_t - \alpha_t \nabla J(\theta_t)\right) \approx J(\theta_t) - \alpha_t ||\nabla J(\theta_t)||^2.$$

2. *Exact line search*: for any t, choose α_t such that

$$J\left(\theta_t - \alpha_t \nabla J(\theta_t)\right) = \min_{a \in \mathbb{R}} J\left(\theta_t - a \nabla J(\theta_t)\right).$$

3. *Backtracking line search*: Choose α_t such that $J\left(\theta_t - \alpha_t \nabla J(\theta_t)\right)$ is sufficiently smaller than $J(\theta_t)$. Implies, for α_t small enough,

$$J\left(\theta_t - \alpha_t \nabla J(\theta_t)\right) \approx J(\theta_t) - \alpha_t ||\nabla J(\theta_t)||^2.$$

Backtracking Line Search is generally a better and more practical option for finding the step size in many machine learning and optimization problems due to its adaptiveness and lower computational cost compared to exact line search. If simplicity is the primary concern and the problem is well-behaved, a fixed schedule with a well-tuned learning rate can work well. Exact Line Search is only recommended in special cases where the objective function is simple and can be solved exactly at each iteration.

Algorithm of Backtracking line search

Steps:

1. Initialize:

- $\beta \in (0,1)$ (commonly around 0.5).
- $c \in (0,1)$ (usually a small value like 0.0001).
- Set the initial step size $\alpha = \alpha_0$.

2. Condition Check:

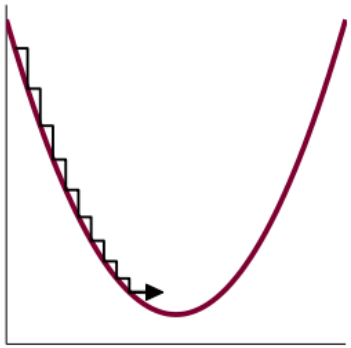
- Repeat until the following condition is satisfied:

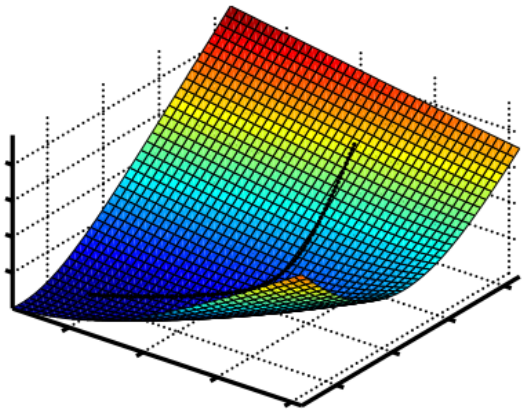
$$J(\theta_t - \alpha \nabla J(\theta_t)) \leq J(\theta_t) - c\alpha \, ||\nabla J(\theta_t)||^2$$

- If the condition is not satisfied:
  Reduce the step size: $\alpha = \beta\alpha$

3. Return the final step size α.

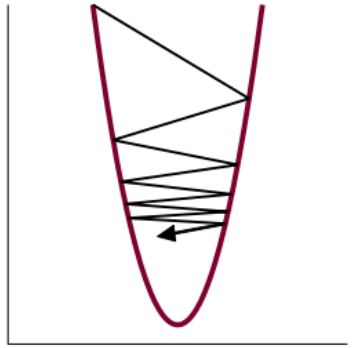
η too small

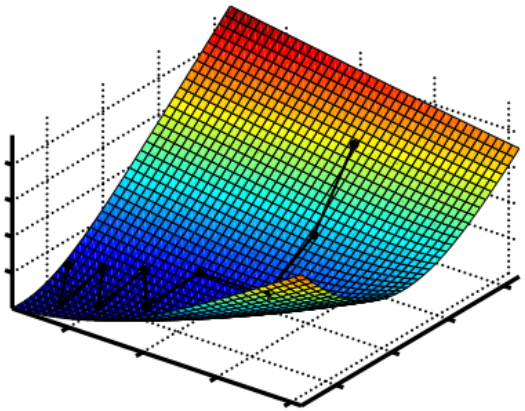




$\eta = 0.1$; 75 steps

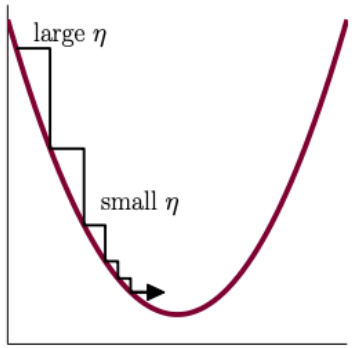
η too large

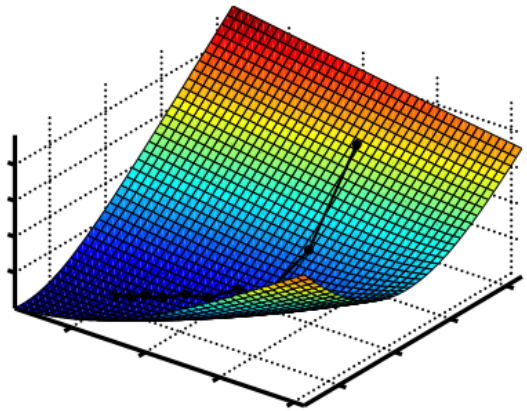




$\eta = 2$; 10 steps

variable η_t – just right





variable η_t; 10 steps

Lipschitz Continuous Gradient

If the objective function $J(\theta)$ has a Lipschitz continuous gradient, this provides important information about the appropriate choice of the learning rate α_t for Gradient Descent.

Lipschitz Continuous Gradient

If the objective function $J(\theta)$ has a Lipschitz continuous gradient, this provides important information about the appropriate choice of the learning rate α_t for Gradient Descent.

Definition

For any lipschitz constant $L > 0$, we say that J is L-smooth if ∇J is L-Lipschitz (Lipschitz continuous gradient), that is

$$\forall x, y \in \mathbb{R}^n, \quad ||\nabla J(x) - \nabla J(y)|| \leq L||x - y||.$$

Lipschitz Continuous Gradient

If the objective function $J(\theta)$ has a Lipschitz continuous gradient, this provides important information about the appropriate choice of the learning rate α_t for Gradient Descent.

Definition

For any lipschitz constant $L > 0$, we say that J is L-smooth if ∇J is L-Lipschitz (Lipschitz continuous gradient), that is

$$\forall x, y \in \mathbb{R}^n, \quad ||\nabla J(x) - \nabla J(y)|| \leq L||x - y||.$$

Lemma

Let $L > 0$ be fixed. If J is L-smooth, then for any $x, y \in \mathbb{R}^n$,

$$J(y) \leq J(x) + \langle \nabla J(x), y - x \rangle + \frac{L}{2}||y - x||^2.$$

where $\langle \nabla J(x), y - x \rangle = \sum_{i=1}^{n} \left(\frac{\partial J(x)}{\partial x_i}(y_i - x_i) \right)$.

Proof.

Consider the Taylor expansion of J around the point x

$$J(y) = J(x) + \langle \nabla J(x), y - x \rangle + R(y),$$

where remainder $R(y) = \int_0^1 \left(\nabla J(x + t(y - x)) - \nabla J(x) \right)^T (y - x) dt$

Proof.

Consider the Taylor expansion of J around the point x

$$J(y) = J(x) + \langle \nabla J(x), y - x \rangle + R(y),$$

where remainder $R(y) = \int_0^1 \left(\nabla J(x + t(y - x)) - \nabla J(x)\right)^T (y - x) dt$
Since J is L-smooth, then

$$||\nabla J(x + t(y - x)) - \nabla J(x)|| \leq L||t(y - x)|| = Lt||y - x||.$$

Proof.

Consider the Taylor expansion of J around the point x

$$J(y) = J(x) + \langle \nabla J(x), y - x \rangle + R(y),$$

where remainder $R(y) = \int_0^1 \left(\nabla J(x + t(y - x)) - \nabla J(x)\right)^T (y - x) dt$
Since J is L-smooth, then

$$||\nabla J(x + t(y - x)) - \nabla J(x)|| \leq L||t(y - x)|| = Lt||y - x||.$$

We get $$||R(y)|| \leq \int_0^1 Lt||y - x||^2 dt = \frac{L}{2}||y - x||^2.$$

Thus,

$$J(y) \leq J(x) + \langle \nabla J(x), y - x \rangle + \frac{L}{2}||y - x||^2.$$

□

Corollary

Let J be L-smooth, for some $L > 0$. We consider gradient descent with constant stepsize $\alpha_t = \frac{1}{L}$ for all t. Then, for any t,

$$J(\theta_{t+1}) \leq J(\theta_t) - \frac{1}{2L}||\nabla J(\theta_t)||^2.$$

Additionally assuming that J is lower bounded,

1. $(J(\theta_t))_{t\in\mathbb{N}}$ *converges to a finite value.*
2. $||\nabla J(\theta_t)|| \to 0$ *as* $t \to \infty$.

Corollary

Let J be L-smooth, for some $L > 0$. We consider gradient descent with constant stepsize $\alpha_t = \frac{1}{L}$ for all t. Then, for any t,

$$J(\theta_{t+1}) \leq J(\theta_t) - \frac{1}{2L}||\nabla J(\theta_t)||^2.$$

Additionally assuming that J is lower bounded,

1. $(J(\theta_t))_{t\in\mathbb{N}}$ *converges to a finite value.*
2. $||\nabla J(\theta_t)|| \to 0$ *as* $t \to \infty$.

Summary: For an objective function with a Lipschitz continuous gradient, the learning rate should be carefully chosen to be less than $\frac{2}{L}$.
This ensures that the Gradient Descent algorithm converges effectively, avoiding instability or divergence. Typically, a learning rate in the range $0 < \alpha \leq \frac{1}{L}$ is considered safe and effective for achieving convergence.

Definition (Convex function)

A function J is called a **convex function** if $\forall x, y \in \mathbb{R}^n$, $t \in [0, 1]$,

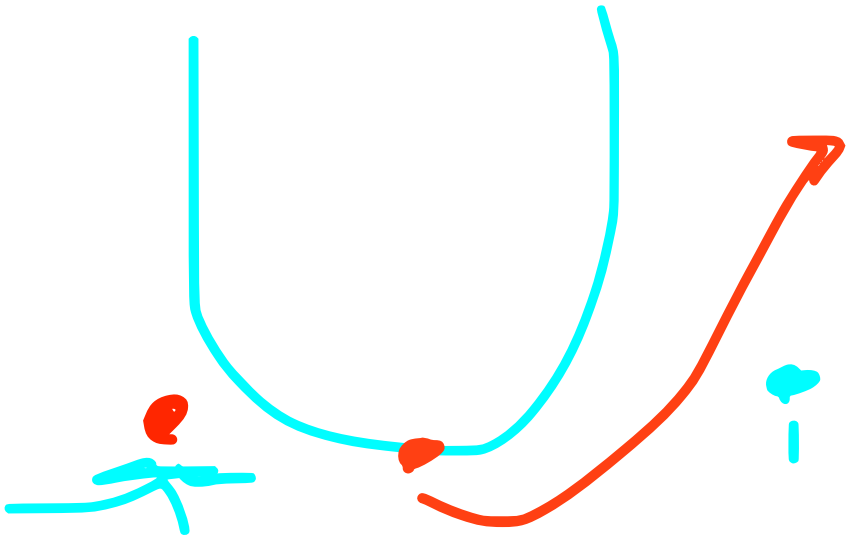
$$J((1-t)x + ty) \leq (1-t)J(x) + tJ(y).$$

Definition (Convex function)

A function J is called a **convex function** if $\forall x, y \in \mathbb{R}^n$, $t \in [0, 1]$,

$$J((1-t)x + ty) \leq (1-t)J(x) + tJ(y).$$

If J is convex, any local minimum is also a global minimum. This means that if gradient descent converges, it will converge to the global minimum of J which is often the desired outcome in optimization problems. Without convexity, gradient descent might converge to a local minimum instead, which may not be the best solution.



Definition (Convex function)

A function J is called a **convex function** if $\forall x, y \in \mathbb{R}^n$, $t \in [0, 1]$,

$$J((1-t)x + ty) \leq (1-t)J(x) + tJ(y).$$

If J is convex, any local minimum is also a global minimum. This means that if gradient descent converges, it will converge to the global minimum of J which is often the desired outcome in optimization problems. Without convexity, gradient descent might converge to a local minimum instead, which may not be the best solution.

Theorem

When J is differentiable, it is convex if and only if

$$\forall x, y \in \mathbb{R}^n, \quad J(y) \geq J(x) + \langle \nabla J(x), y - x \rangle.$$

Proof.

($\Rightarrow$) : Suppose that J is convex, then

$$
\begin{aligned}
J((1-t)x + ty) &\leq (1-t)J(x) + tJ(y) \\
J(x + t(y-x)) &\leq J(x) + t(J(y) - J(x)) \\
J(y) - J(x) &\geq \frac{J(x + t(y-x)) - J(x)}{t}, \quad \forall t \in (0, 1] \\
J(y) &\geq J(x) + \langle \nabla J(x), y - x \rangle, \quad \text{as } t \to 0
\end{aligned}
$$

Proof.

($\Rightarrow$) : Suppose that J is convex, then

$$\begin{aligned} J((1-t)x + ty) &\leq (1-t)J(x) + tJ(y) \\ J(x + t(y-x)) &\leq J(x) + t(J(y) - J(x)) \\ J(y) - J(x) &\geq \frac{J(x + t(y-x)) - J(x)}{t}, \quad \forall t \in (0, 1] \\ J(y) &\geq J(x) + \langle \nabla J(x), y - x \rangle, \quad \text{as } t \to 0 \end{aligned}$$

($\Leftarrow$) : Let $z = tx + (1-t)y$, then we have

$$\begin{aligned} J(x) &\geq J(z) + \langle \nabla J(z), x - z \rangle \\ J(y) &\geq J(z) + \langle \nabla J(z), y - z \rangle \end{aligned}$$

□

Proof.

($\Rightarrow$) : Suppose that J is convex, then

$$\begin{aligned} J((1-t)x+ty) &\leq (1-t)J(x)+tJ(y) \\ J(x+t(y-x)) &\leq J(x)+t(J(y)-J(x)) \\ J(y)-J(x) &\geq \frac{J(x+t(y-x))-J(x)}{t}, \quad \forall t \in (0,1] \\ J(y) &\geq J(x)+\langle \nabla J(x), y-x\rangle, \quad \text{as } t \to 0 \end{aligned}$$

($\Leftarrow$) : Let $z = tx + (1-t)y$, then we have

$$\begin{aligned} J(x) &\geq J(z)+\langle \nabla J(z), x-z\rangle \\ J(y) &\geq J(z)+\langle \nabla J(z), y-z\rangle \end{aligned}$$

Then,

$$\begin{aligned} tJ(y)+(1-t)J(x) &\geq J(z)+t\langle \nabla J(z), y-z\rangle + (1-t)\langle \nabla J(z), x-z\rangle \\ &\geq J(z)+\langle \nabla J(z), ty-tz+(1-t)x+tz-z\rangle \\ &\geq J(tx+(1-t)y) \end{aligned}$$

□

Theorem

Let J be convex and L-smooth, for some $L > 0$. We consider gradient descent with constant stepsize : $\alpha_t = \frac{1}{L}$ for all t. Then, for any $t \in \mathbb{N}$,

$$J(x_t) - J(x_*) \leq \frac{2L||x_0 - x_*||^2}{t+4}.$$

where x_* is a minimizer of J such that $J(x_*) = \min_{x \in \mathbb{R}^n} J(x)$.

Proof.

Let t be fixed. Since J is convex, by above theorem, we get

$$J(x_*) \geq J(x_t) + \langle \nabla J(x_t), x_* - x_t \rangle = J(x_t) + L\langle x_t - x_{t+1}, x_* - x_t \rangle. \quad (1)$$

Proof.

Let t be fixed. Since J is convex, by above theorem, we get

$$J(x_*) \geq J(x_t) + \langle \nabla J(x_t), x_* - x_t \rangle = J(x_t) + L\langle x_t - x_{t+1}, x_* - x_t \rangle. \quad (1)$$

And since x_* is minimizer, by using L-smoothness

$$\begin{aligned} J(x_*) &\leq J(x_{t+1}) \\ &\leq J(x_t) - \frac{1}{2L}||\nabla J(x_t)||^2 \\ &= J(x_t) - \frac{L}{2}||x_{t+1} - x_t||^2, \qquad (2) \end{aligned}$$

Proof.

Let t be fixed. Since J is convex, by above theorem, we get

$$J(x_*) \geq J(x_t) + \langle \nabla J(x_t), x_* - x_t \rangle = J(x_t) + L\langle x_t - x_{t+1}, x_* - x_t \rangle. \quad (1)$$

And since x_* is minimizer, by using L-smoothness

$$\begin{aligned} J(x_*) &\leq J(x_{t+1}) \\ &\leq J(x_t) - \frac{1}{2L}||\nabla J(x_t)||^2 \\ &= J(x_t) - \frac{L}{2}||x_{t+1} - x_t||^2, \qquad (2) \end{aligned}$$

From (1) and (2),

$$\begin{gathered} J(x_t) + L\langle x_t - x_{t+1}, x_* - x_t \rangle \leq J(x_*) \leq J(x_t) - \tfrac{L}{2}||x_{t+1} - x_t||^2 \\ 2\langle x_t - x_{t+1}, x_* - x_t \rangle + ||x_{t+1} - x_t||^2 \leq 0 \\ ||x_* - x_{t+1}||^2 \leq ||x_* - x_t||^2 \end{gathered}$$

□

Proof.

Now by above corollary,

$$J(x_{t+1}) - J(x_*) \leq J(x_t) - J(x_*) - \frac{1}{2L}||\nabla J(x_t)||^2., \quad (3)$$

Proof.

Now by above corollary,

$$J(x_{t+1}) - J(x_*) \leq J(x_t) - J(x_*) - \frac{1}{2L}||\nabla J(x_t)||^2., \quad (3)$$

By using Cauchy-schwarz, we can find

$$||\nabla J(x_t)|| \geq \frac{J(x_t) - J(x_*)}{||x_0 - x_*||}$$

Proof.

Now by above corollary,

$$J(x_{t+1}) - J(x_*) \leq J(x_t) - J(x_*) - \frac{1}{2L}||\nabla J(x_t)||^2., \quad (3)$$

By using Cauchy-schwarz, we can find

$$||\nabla J(x_t)|| \geq \frac{J(x_t) - J(x_*)}{||x_0 - x_*||}$$

$$J(x_{t+1}) - J(x_*) \leq J(x_t) - J(x_*) - \frac{1}{2L}\frac{(J(x_t) - J(x_*))^2}{||x_0 - x_*||^2}$$

$$\frac{1}{J(x_{t+1}) - J(x_*)} \geq \frac{1}{J(x_t) - J(x_*)} + \frac{1}{2L||x_0 - x_*||^2}$$

$$\frac{1}{J(x_t) - J(x_*)} \geq \frac{1}{J(x_0) - J(x_*)} + \frac{1}{2L||x_0 - x_*||^2}$$

□

Proof.

Since,

$$J(x_0) - J(x_*) \leq \frac{L||x_0 - x_*||^2}{2}$$

Therefore,

$$J(x_t) - J(x_*) \leq \frac{2L||x_0 - x_*||^2}{t+4}.$$

□

Proof.

Since,

$$J(x_0) - J(x_*) \leq \frac{L||x_0 - x_*||^2}{2}$$

Therefore,

$$J(x_t) - J(x_*) \leq \frac{2L||x_0 - x_*||^2}{t+4}.$$

□

Summary: The theorem provides a rigorous guarantee of how quickly gradient descent converges to the minimum for convex and smooth functions. It guides the choice of step size, highlights the importance of initial conditions, and serves as a critical result for optimizing machine learning models and other applications in convex optimization.

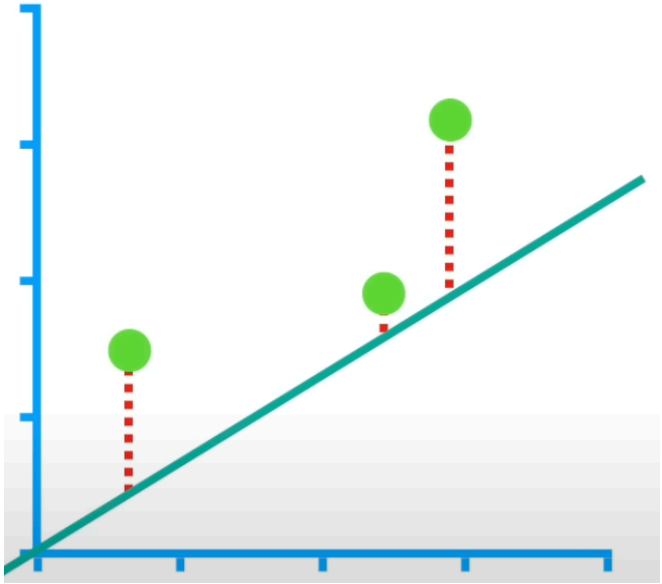
Application to Logistic Regression

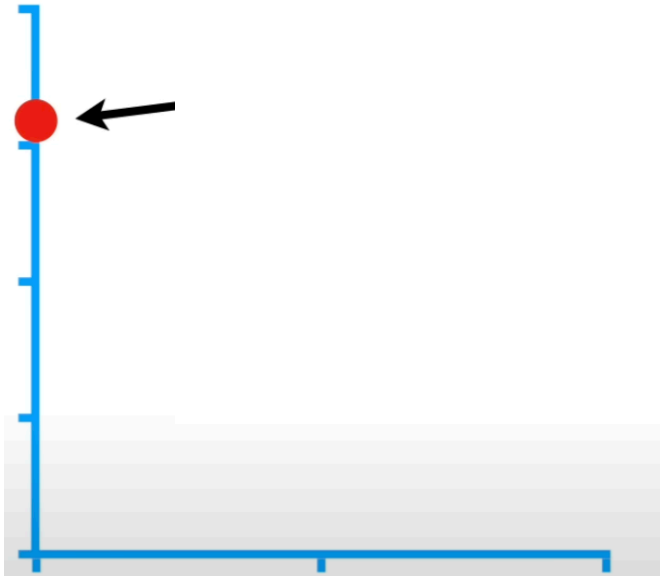
Gradient descent is an optimization algorithm used to minimize the cost function (also known as the loss function) in machine learning models. In logistic regression, the cost function is based on the difference between the predicted probabilities and the actual labels. The aim is to minimize this difference and find the best parameters that maximize the models accuracy.

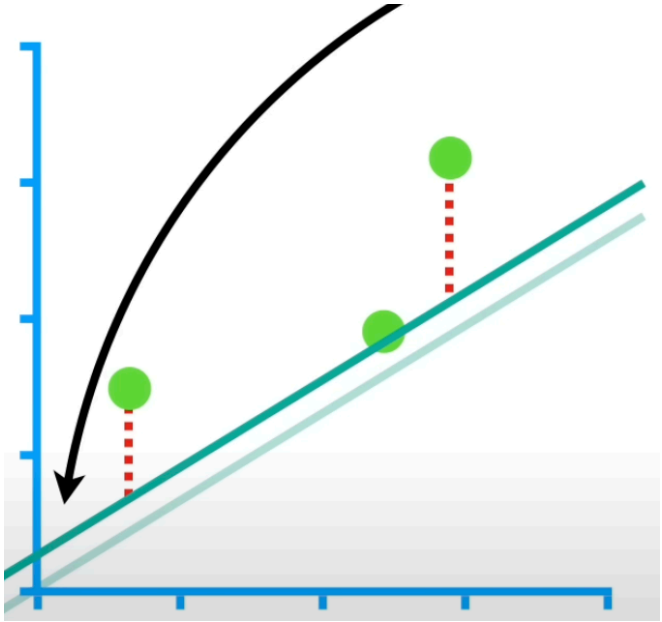
Application to Logistic Regression

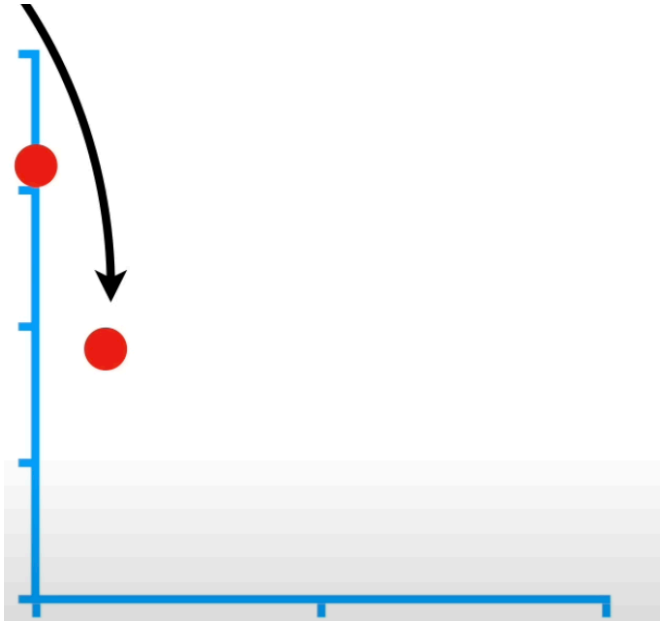
Gradient descent is an optimization algorithm used to minimize the cost function (also known as the loss function) in machine learning models. In logistic regression, the cost function is based on the difference between the predicted probabilities and the actual labels. The aim is to minimize this difference and find the best parameters that maximize the models accuracy.

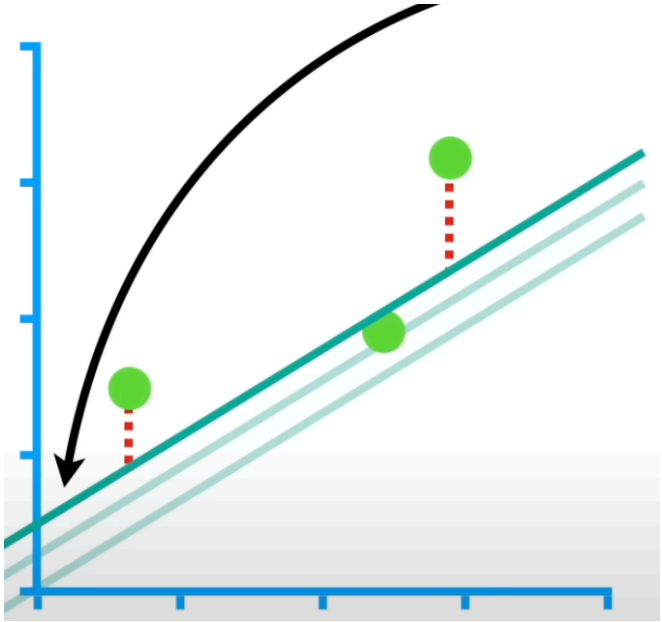
By using gradient descent to minimize the cost function of a machine learning model, we can find the best set of model parameters for accurate predictions. This means that it helps us find the best values for our models parameters so that our model can make accurate predictions.

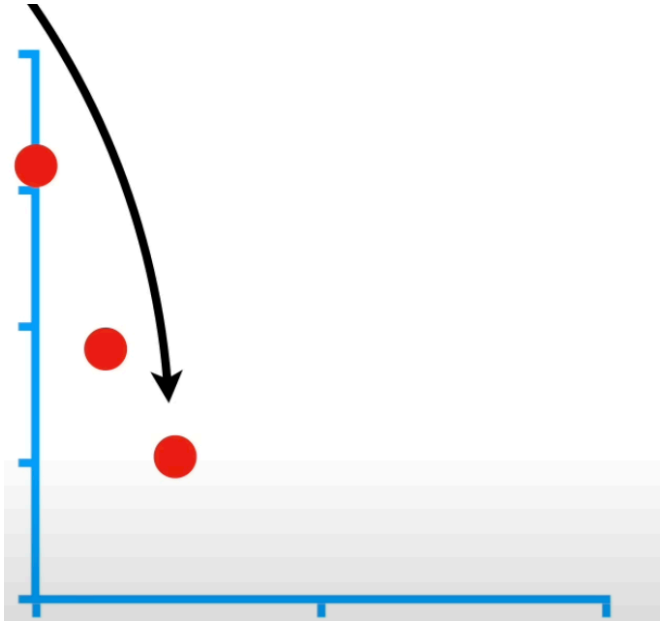


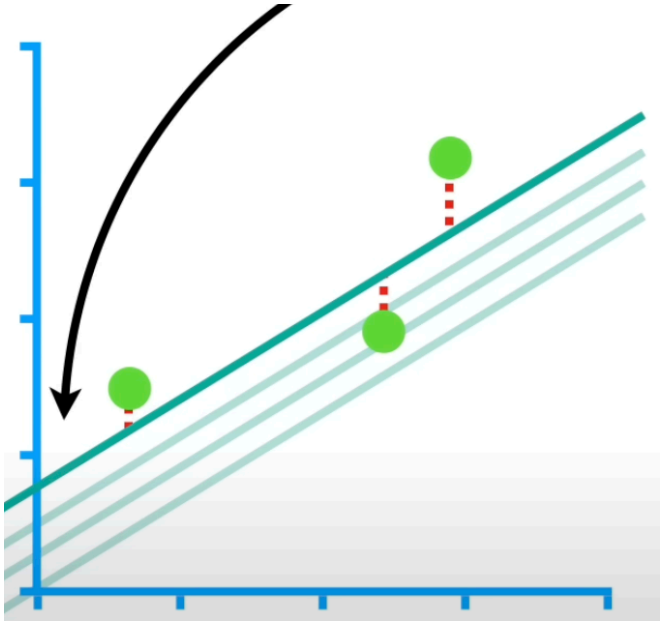


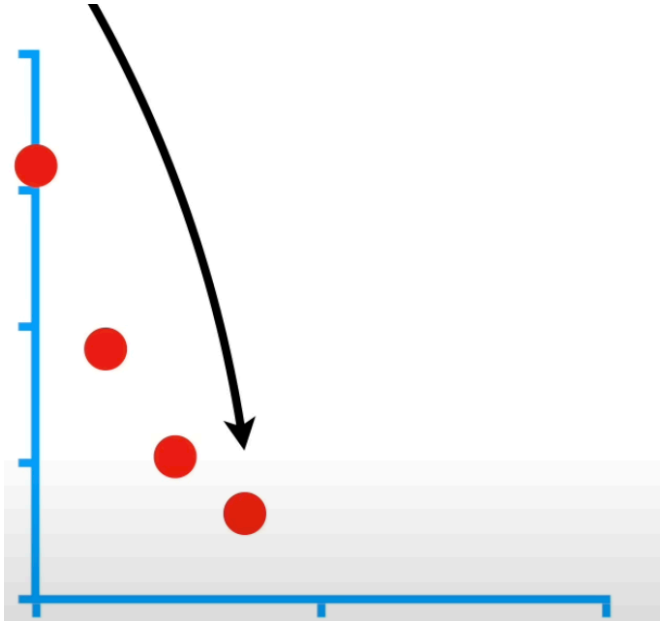


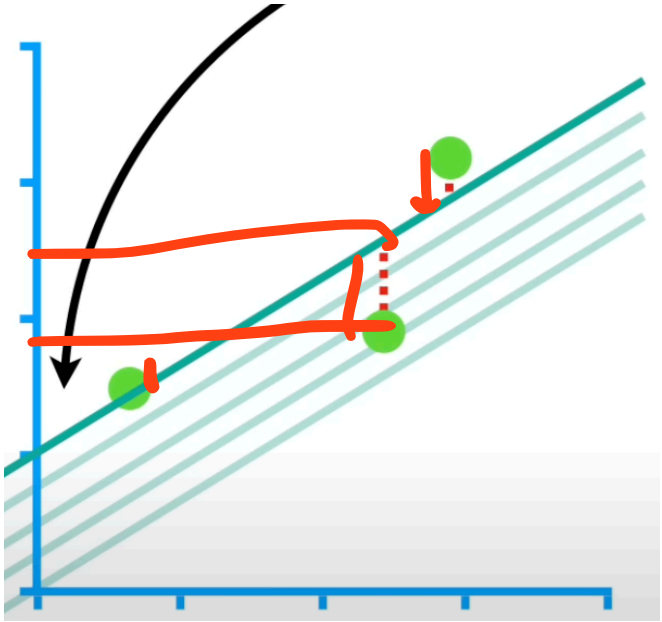


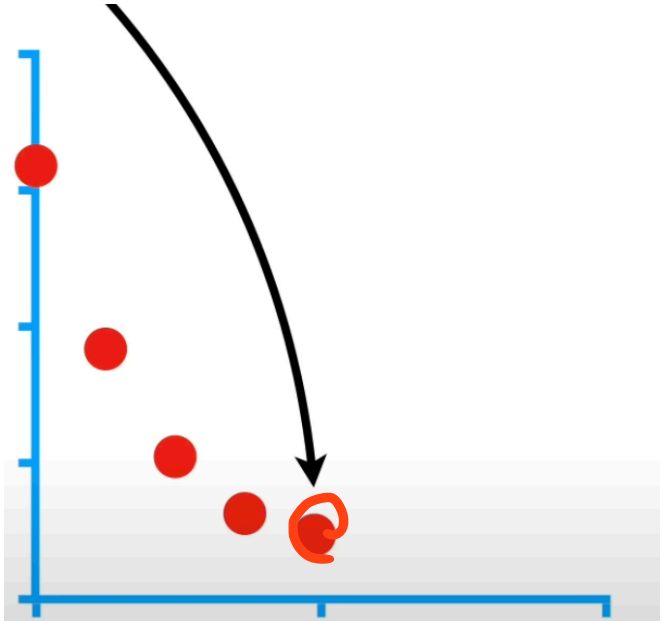


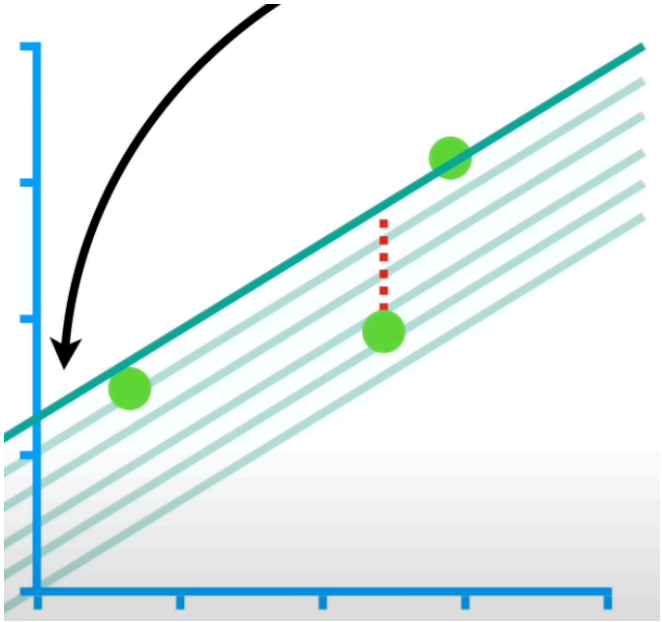


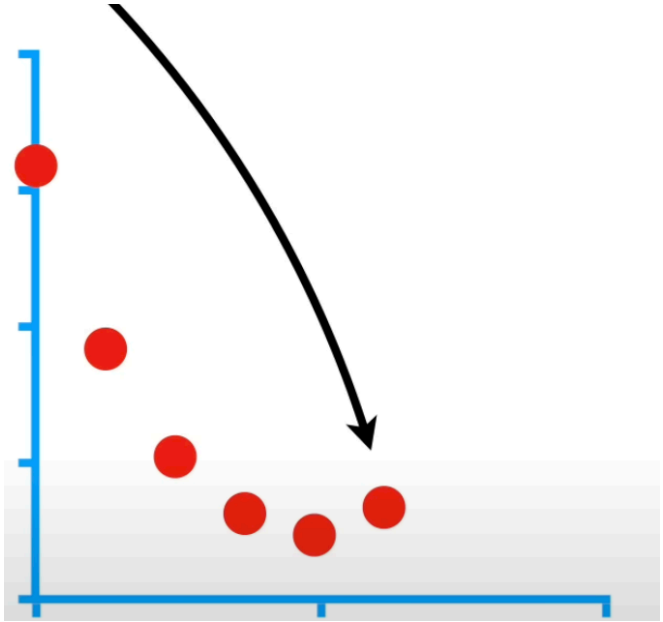


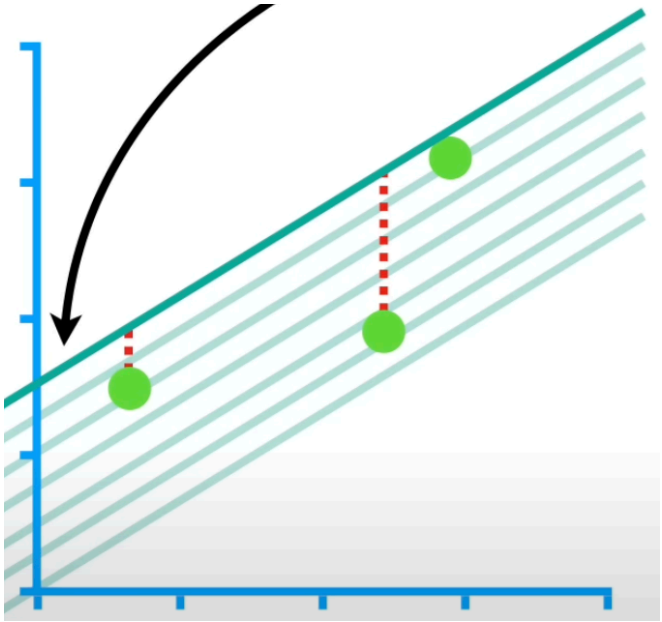


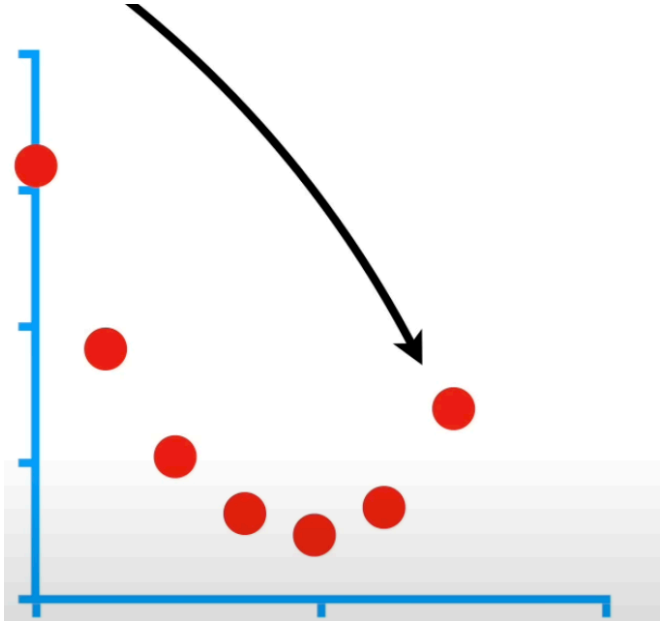


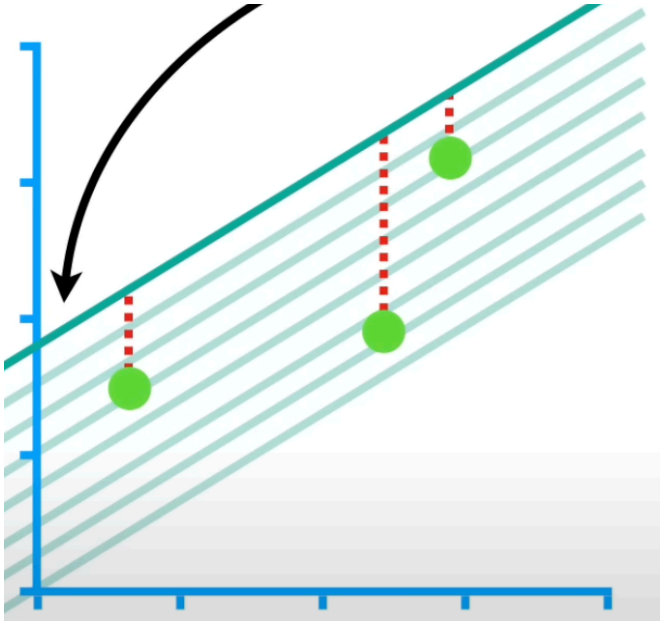


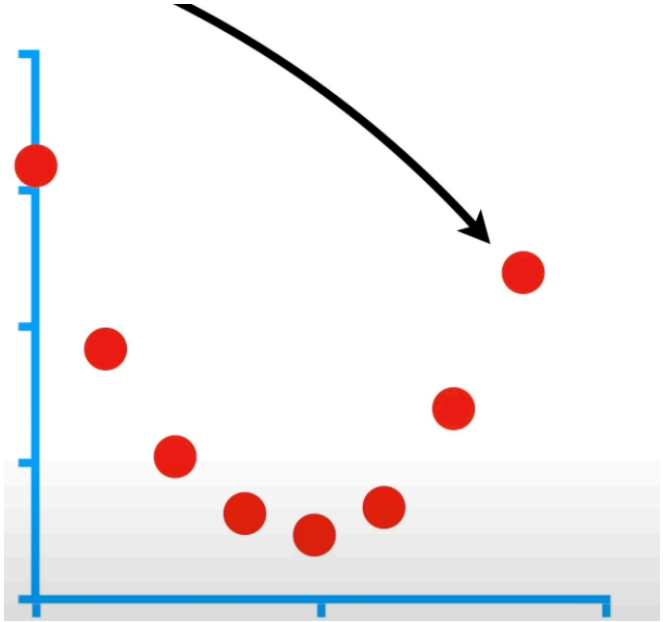












The cost function for logistic regression is :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

where:

- $h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$ is the hypothesis function (sigmoid function).
- $x^{(i)}$ is the feature vector for the i-th training.
- $y^{(i)}$ is the corresponding label.

The cost function for logistic regression is :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

where:

- $h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$ is the hypothesis function (sigmoid function).
- $x^{(i)}$ is the feature vector for the i-th training.
- $y^{(i)}$ is the corresponding label.

Now, compute the gradient of the cost function w.r.t x, we get

$$[\nabla J(\theta)]_j = \frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

The cost function for logistic regression is :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

where:

- $h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$ is the hypothesis function (sigmoid function).
- $x^{(i)}$ is the feature vector for the i-th training.
- $y^{(i)}$ is the corresponding label.

Now, compute the gradient of the cost function w.r.t x, we get

$$[\nabla J(\theta)]_j = \frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

Then the parameters are updated using Gradient Descent as:

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha\left[\nabla J(\theta)\right]_j, \quad t \text{ iterations}$$

Finally, the process is repeated until convergence, leading to the optimal parameter that minimize the logistic loss and reult in a well-fitted model.

Algorithm: Logistic Regression with Gradient Descent

Input:

- Training data $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ where $x^{(i)} \in \mathbb{R}^n$ and $y^{(i)} \in \{0, 1\}$.
- Learning rate α and number of iteration n.

STEP 1: Initialize $\theta = [\theta_0, \theta_1, ..., \theta_p]$ to small random values or zeros.

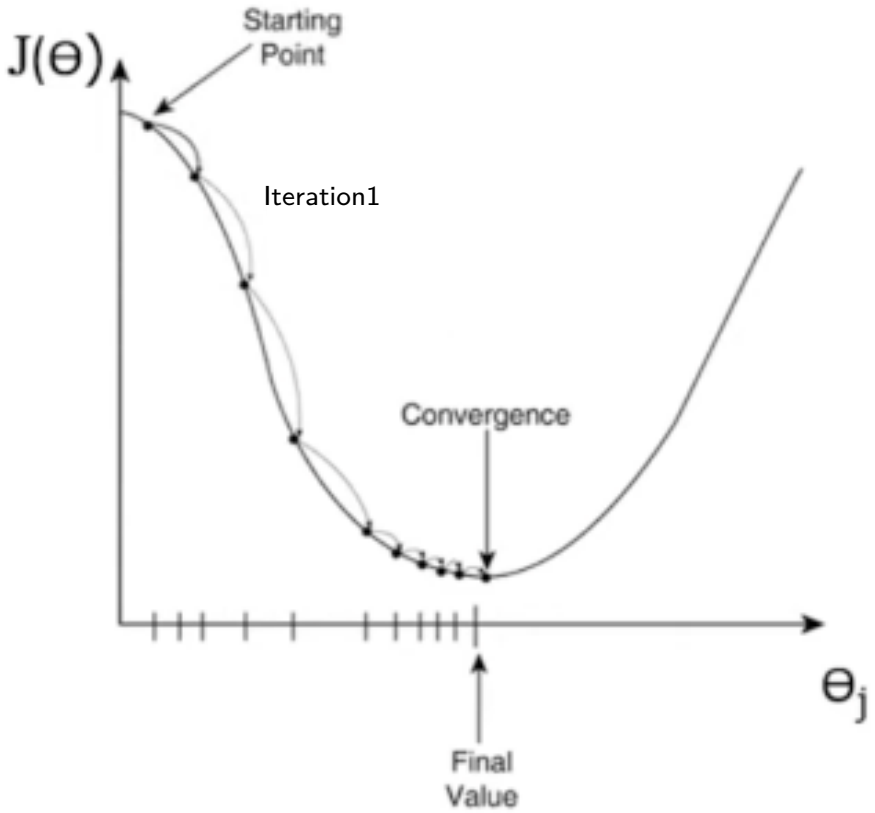
STEP 2: For iteration t from 1 to n:

compute $h_{\theta_j}(x^{(i)}) = \frac{1}{1+e^{-\theta_j^T x^{(i)}}}$

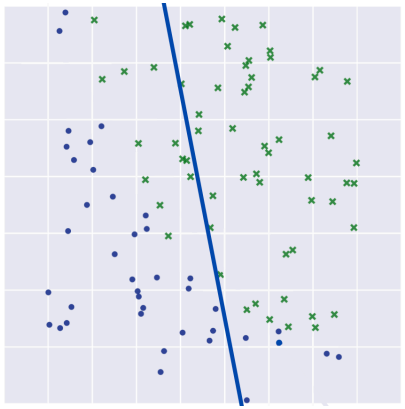
compute $\nabla [J(\theta)]_j = \frac{1}{m} \sum_{i=1}^{m} \left(h_{\theta_j}(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$

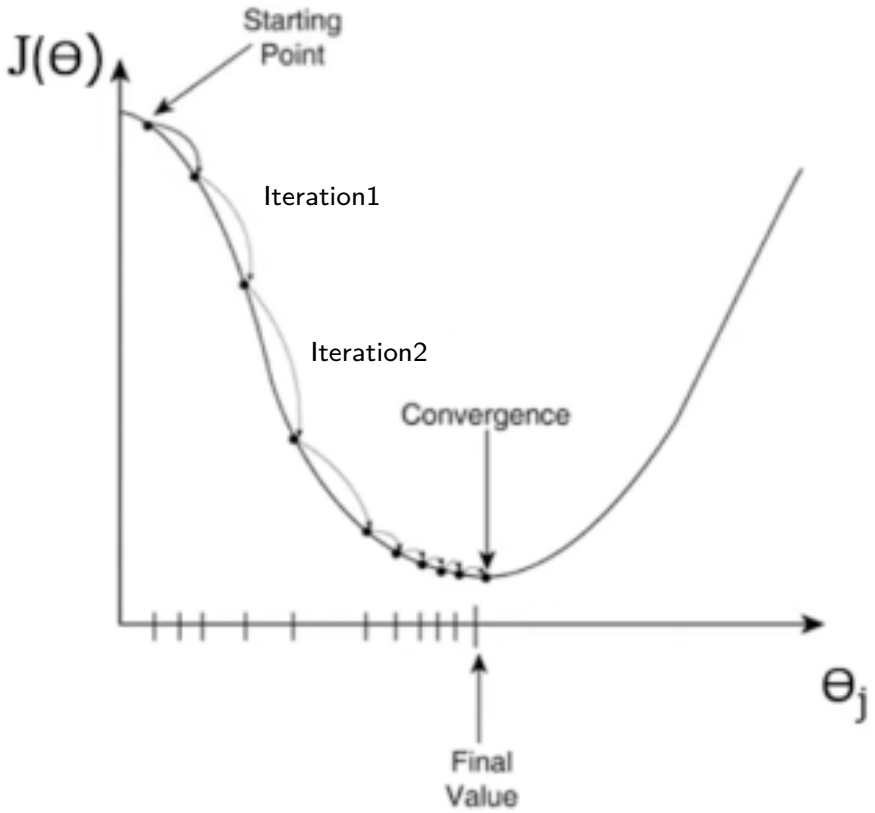
update $\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha [\nabla J(\theta)]_j$

STEP 3: After n iterations or convergence, return the optimized parameter vector θ.

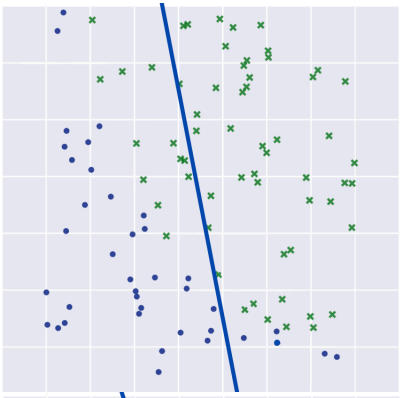


Iteration1

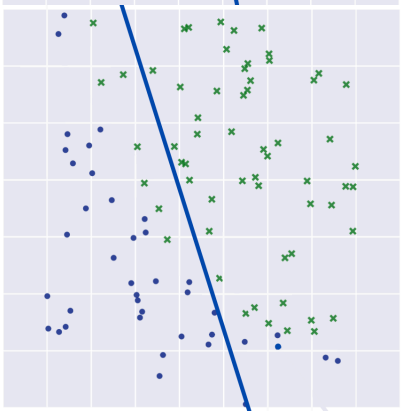


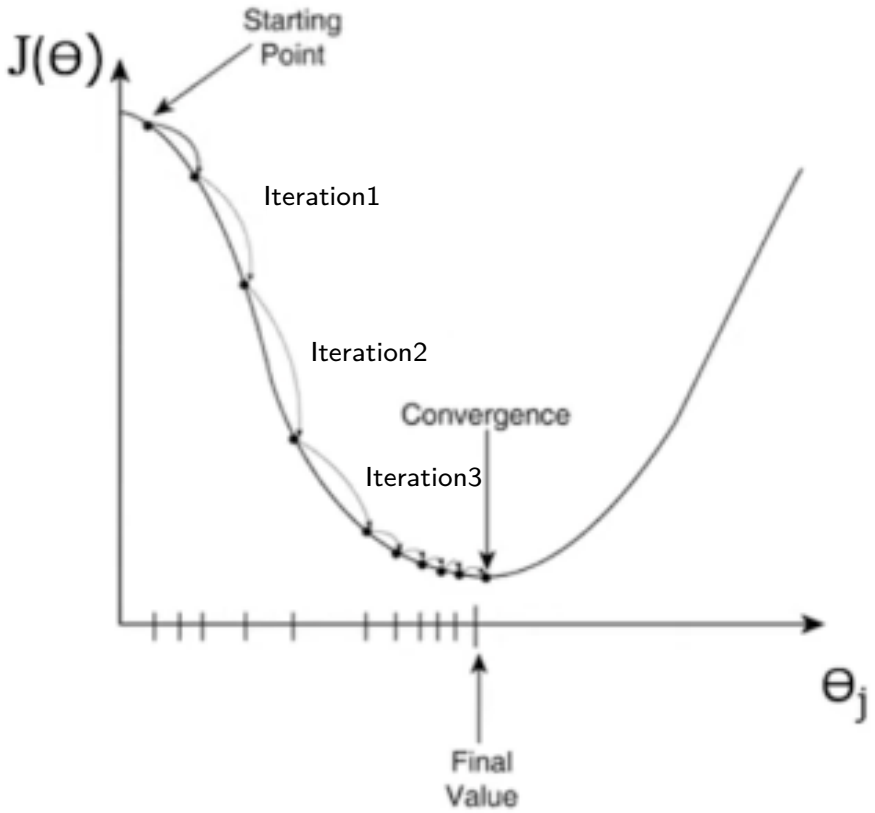


Iteration1

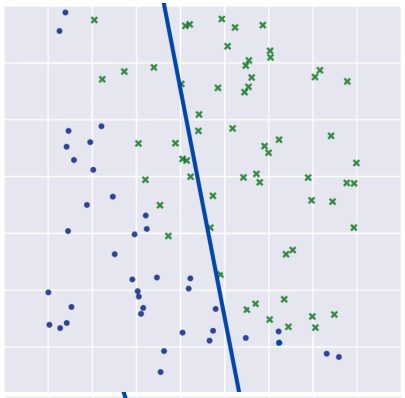


Iteration2

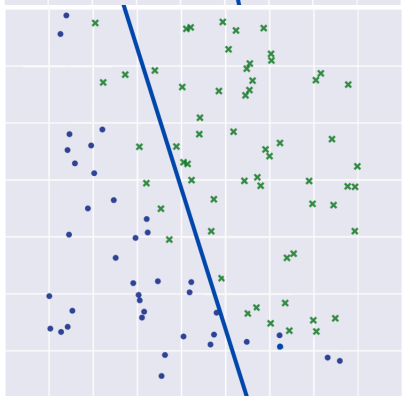




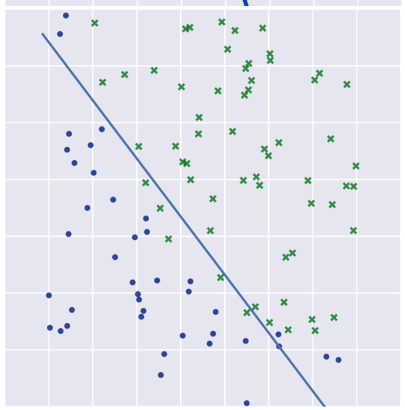
Iteration1



Iteration2



Iteration3



LR with GD in C++

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

// Sigmoid function to calculate the hypothesis
double sigmoid(double z) {
    return 1.0 / (1.0 + exp(-z));
}

// Function to compute the dot product of two vectors
double dotProduct(const vector<double>& a, const vector<double>&
b) {
    double result = 0.0;
    for (size_t i = 0; i < a.size(); ++i) {
        result += a[i] * b[i];
    }
    return result;
}

// Function to perform Logistic Regression using Gradient Descent
vector<double> logisticRegressionGD(const
vector<vector<double>>& X, const vector<int>& y, double alpha, int
iterations) {
 int m = X.size(); // Number of training examples
 int n = X[0].size(); // Number of features
 vector<double> theta(n, 0.0); // Initialize theta to zeros
```

```
// Gradient Descent loop
for (int iter = 0; iter < iterations; ++iter) {
vector<double> gradients(n, 0.0); // Initialize gradients to zero

// Compute gradients for each parameter
for (int i = 0; i < m; ++i) {
double h_theta = sigmoid(dotProduct(theta, X[i])); // Hypothesis for
example i
for (int j = 0; j < n; ++j) {
gradients[j] += (h_theta - y[i]) * X[i][j]; // Accumulate gradients
}
}
      // Update each parameter theta_j
      for (int j = 0; j < n; ++j) {
         theta[j] -= (alpha / m) * gradients[j];
      }
   }

   return theta; // Return the optimized theta values
}
```

```
int main() {
    // Example training data: 4 examples, 3 features (including bias
term)
    vector<vector<double>> X = {{1.0, 2.0, 3.0}, {1.0, 3.0, 4.0}, {1.0, 5.0,
6.0}, {1.0, 7.0, 8.0}};
    vector<int> y = {0, 0, 1, 1}; // Labels

    double alpha = 0.01;     // Learning rate
    int iterations = 1000;   // Number of iterations

    // Perform Logistic Regression with Gradient Descent
    vector<double> theta = logisticRegressionGD(X, y, alpha,
iterations);

    // Output the optimized parameters
    cout << "Optimized theta values: ";
    for (double val : theta) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```

Stochastic Gradient Descent (SGD): Instead of using the full dataset, SGD operates on a small random subset (batch) of the data at each iteration. It provides an approximation of the gradient:

- Faster iterations: Since SGD only uses a small batch of data, it updates weights more frequently, speeding up learning.
- Noise and variance: Due to the smaller batches, the gradients are noisier, leading to faster exploration of the parameter space but more instability in updates.
- Better generalization: The inherent noise in SGD's updates allows it to escape local minima and potentially find better solutions for generalization.

Stochastic Gradient Descent (SGD): Instead of using the full dataset, SGD operates on a small random subset (batch) of the data at each iteration. It provides an approximation of the gradient:

- Faster iterations: Since SGD only uses a small batch of data, it updates weights more frequently, speeding up learning.
- Noise and variance: Due to the smaller batches, the gradients are noisier, leading to faster exploration of the parameter space but more instability in updates.
- Better generalization: The inherent noise in SGD's updates allows it to escape local minima and potentially find better solutions for generalization.

Addition, in machine learning, different optimization algorithms are used to improve the efficiency and performance of training models. Some popular gradient-based optimization methods, specifically focusing on Momentum methods, Adagrad, and Adam, all of which are variations of Gradient Descent.

References

[1]. **Gareth James . Daniela Witten . Trevor Hastie . Robert Tibshirani**, An Introduction to Statistical Learning with Applications in R, Second Edition, 2017, Springer.

[2]. **Irène Waldspurger**, Gradient descent, (2020), *http://surl.li/hqlkru*.

[3]. **M. Magdon-Ismail**, Lecture 9- Logistic Regression and Gradient Descent, *http://surl.li/alxrjb*.

[4]. **Sebastian Ruder**, An overview of gradient descent optimization al gorithms*, (2016), *https://arxiv.org/pdf/1609.04747*.

[5]. **Mark Schmidt**, CPSC 540: Machine Learning Convergence of Gradient Descent, (2017), *http://surl.li/vkltec*.


THANK YOU
FOR YOUR
ATTENTION