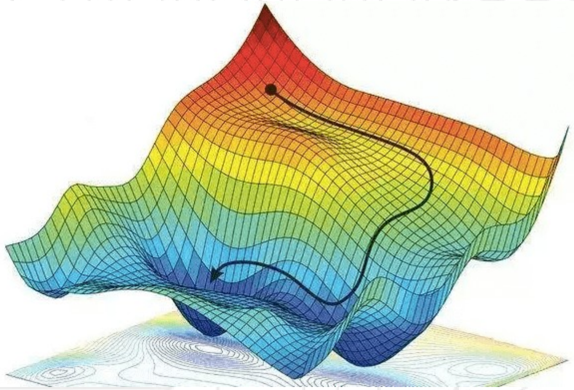
Module 1

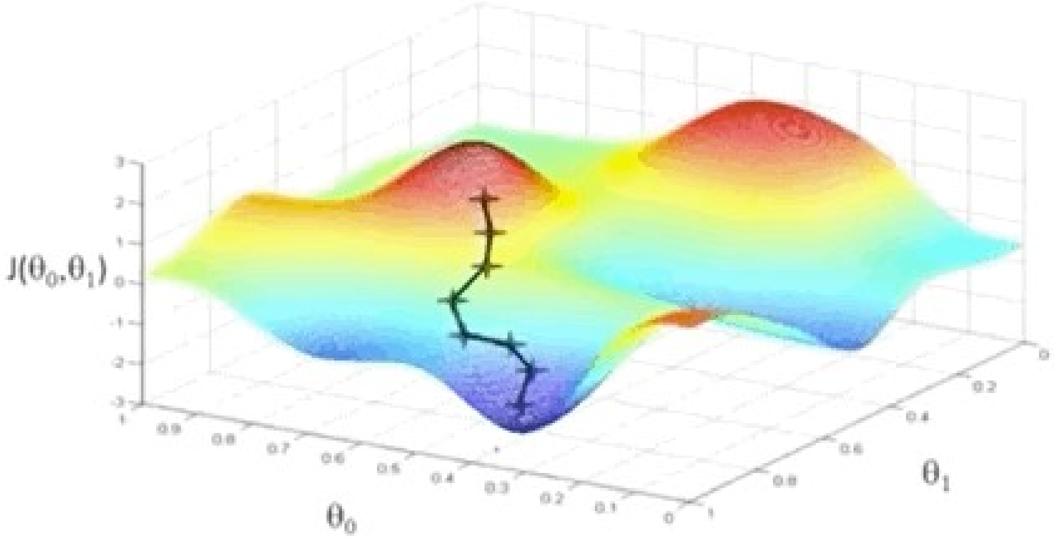
Regularization

The Gradient Descent Algorithm



Gradient Descent

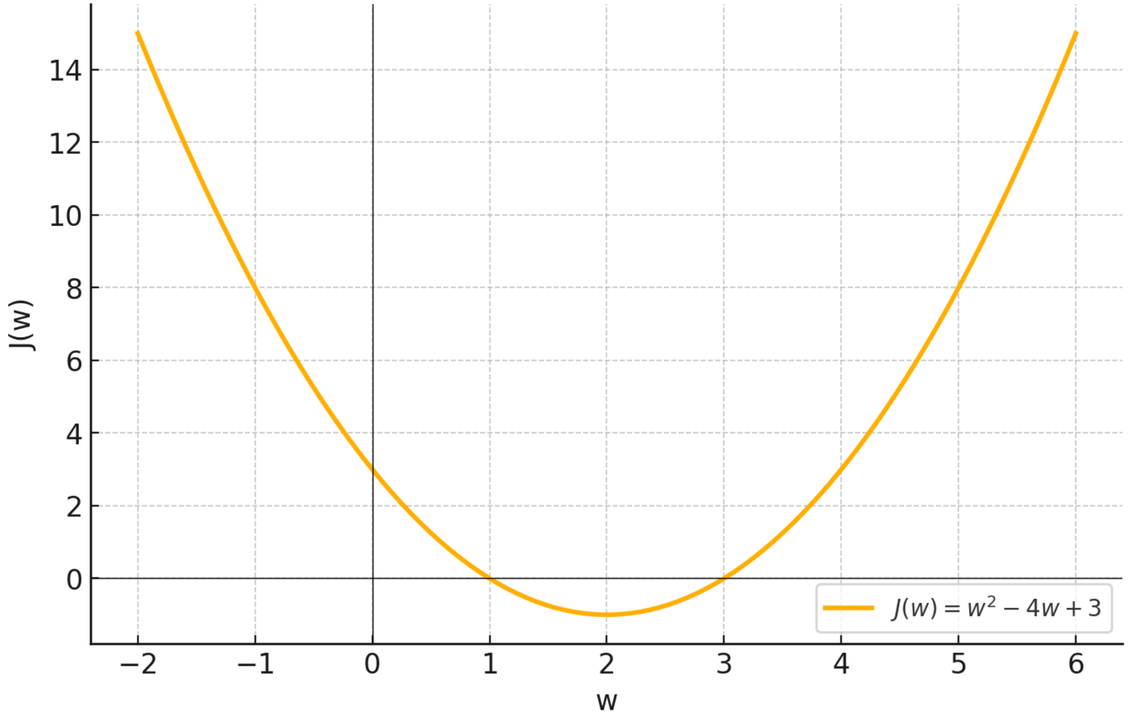
- Gradient descent is an optimization algorithm used to find the minimum of a cost function.
- It is based on moving downwards the gradient of the cost function in steps.



$$\nabla J(\boldsymbol{w}) = \frac{\delta J(\boldsymbol{w})}{\delta \boldsymbol{w}}$$

Gradient Descent

- Moving down the gradient.
- The negative gradient allows us to know how to update $\boldsymbol{w}$

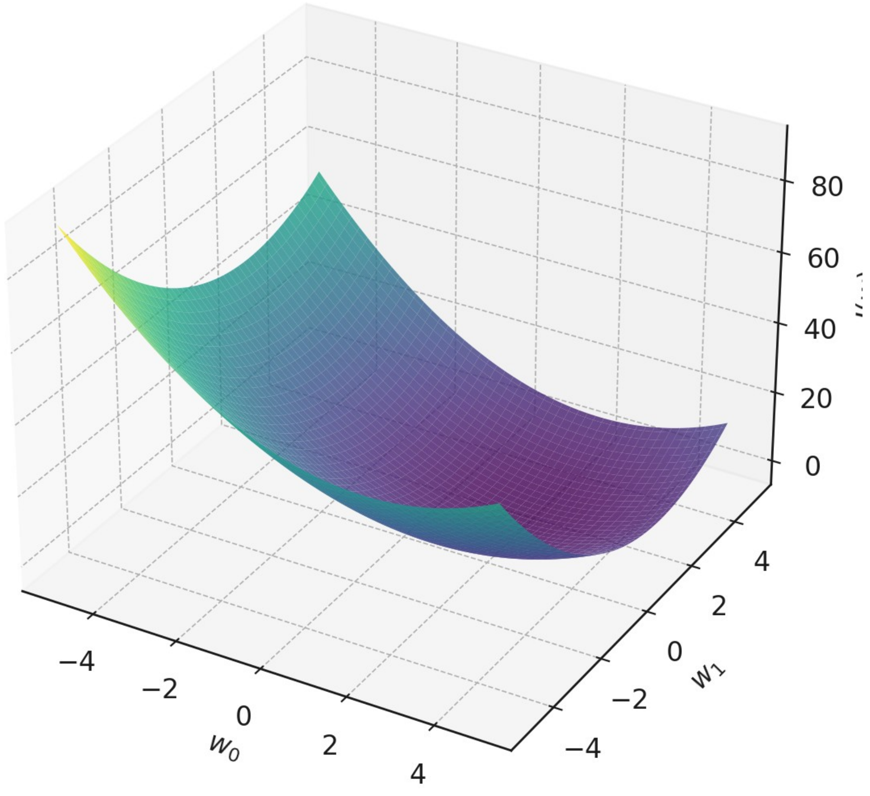


$$\nabla J(\boldsymbol{w}) = \frac{\delta J(w)}{\delta w}$$

$$-\nabla J(\boldsymbol{w})$$

Gradient Descent

- Moving down the gradient.
- The negative gradient allows us to know how to update $\boldsymbol{w}$



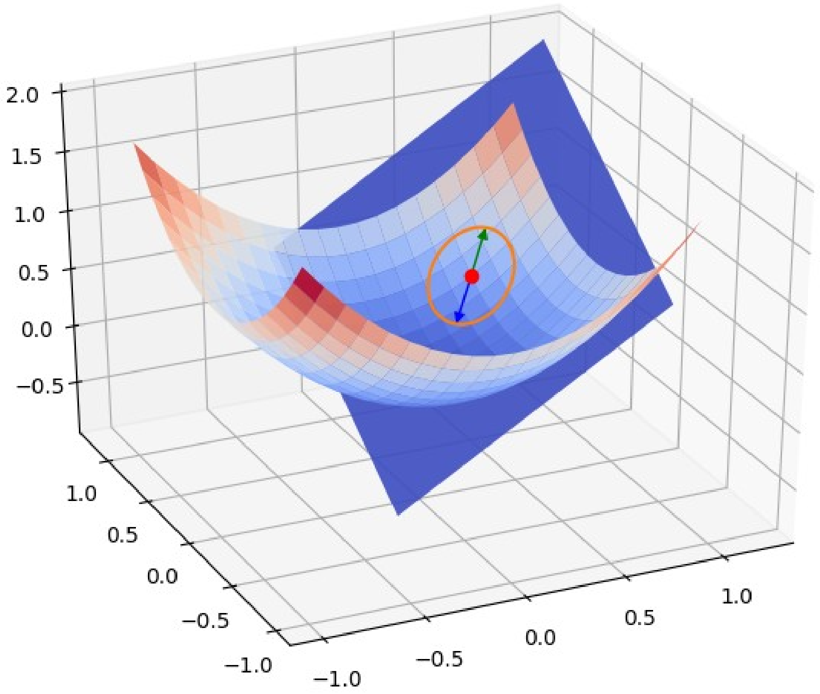
$$\nabla J(\boldsymbol{w}) = \frac{\delta J(w)}{\delta w}$$

$$-\nabla J(\boldsymbol{w})$$

Gradient Descent

- If $\nabla J(\boldsymbol{w})$ is negative or positive we need to keep updating

$-\nabla J(\boldsymbol{w})$



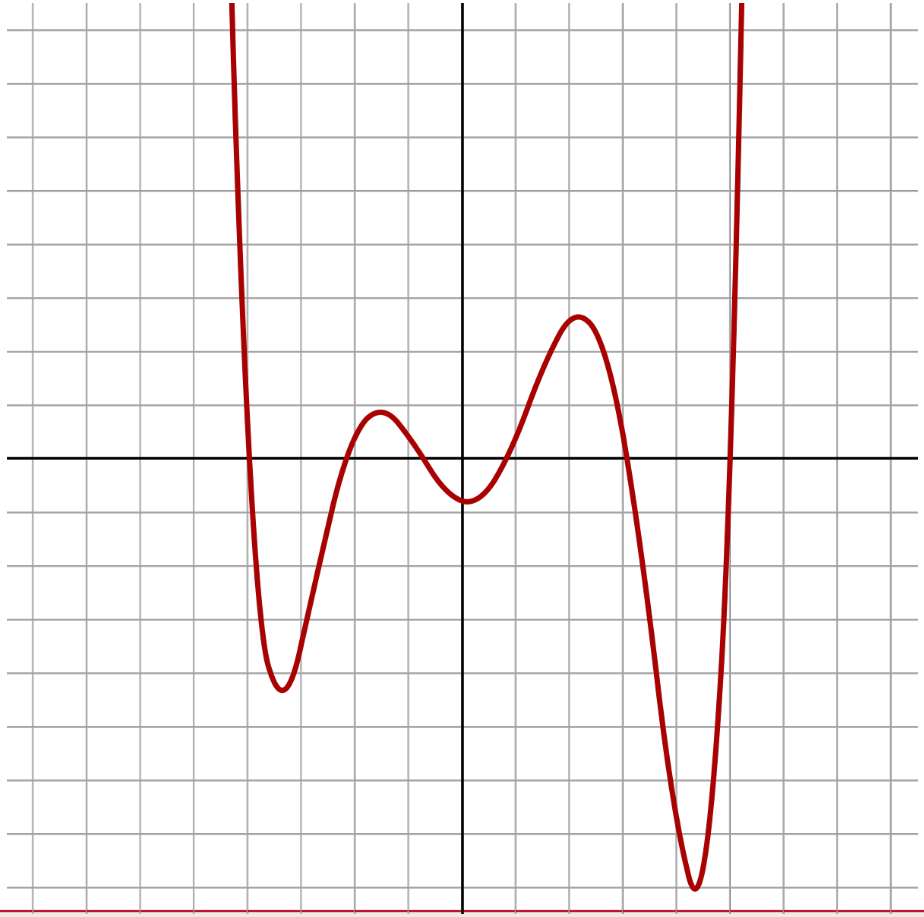
If $\nabla J(\boldsymbol{w})$ is negative or positive we need to keep updating

Gradient Descent Algorithm

- The algorithm to update $\boldsymbol{w}$

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla J(\boldsymbol{w}_\mathrm{i})$$

w at next step   ***w*** at current step   step size   gradient

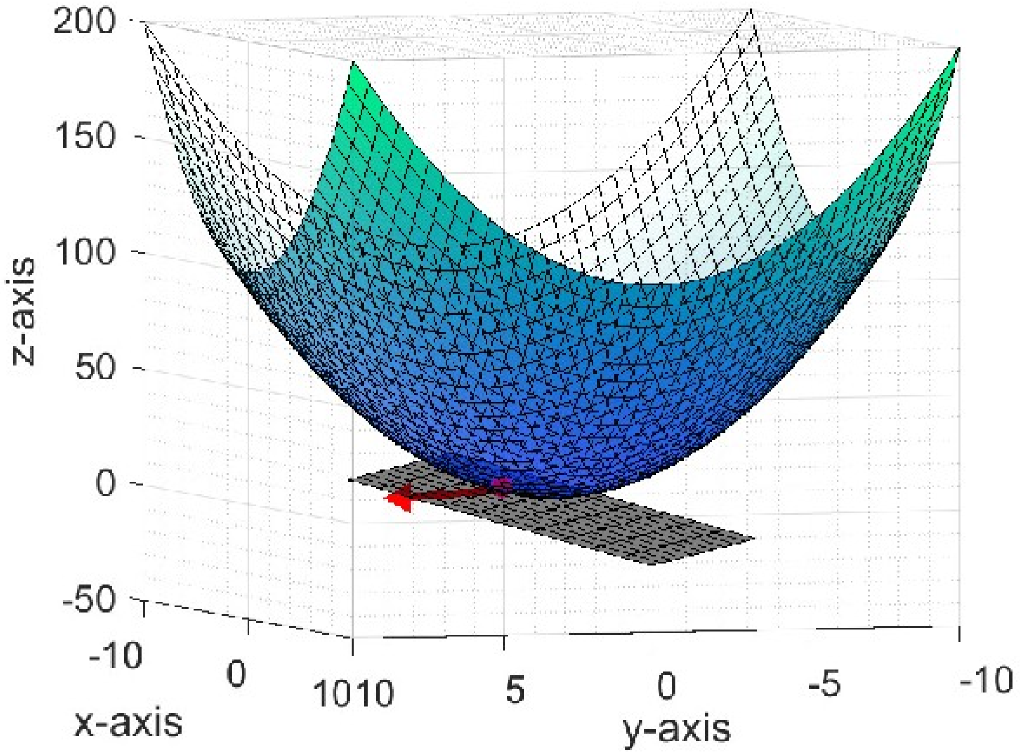


Gradient Descent Algorithm

- The algorithm to update $\boldsymbol{w}$

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla J(\boldsymbol{w}_\mathrm{i})$$

w **at next step** | ***w*** **at current step** | step size | gradient

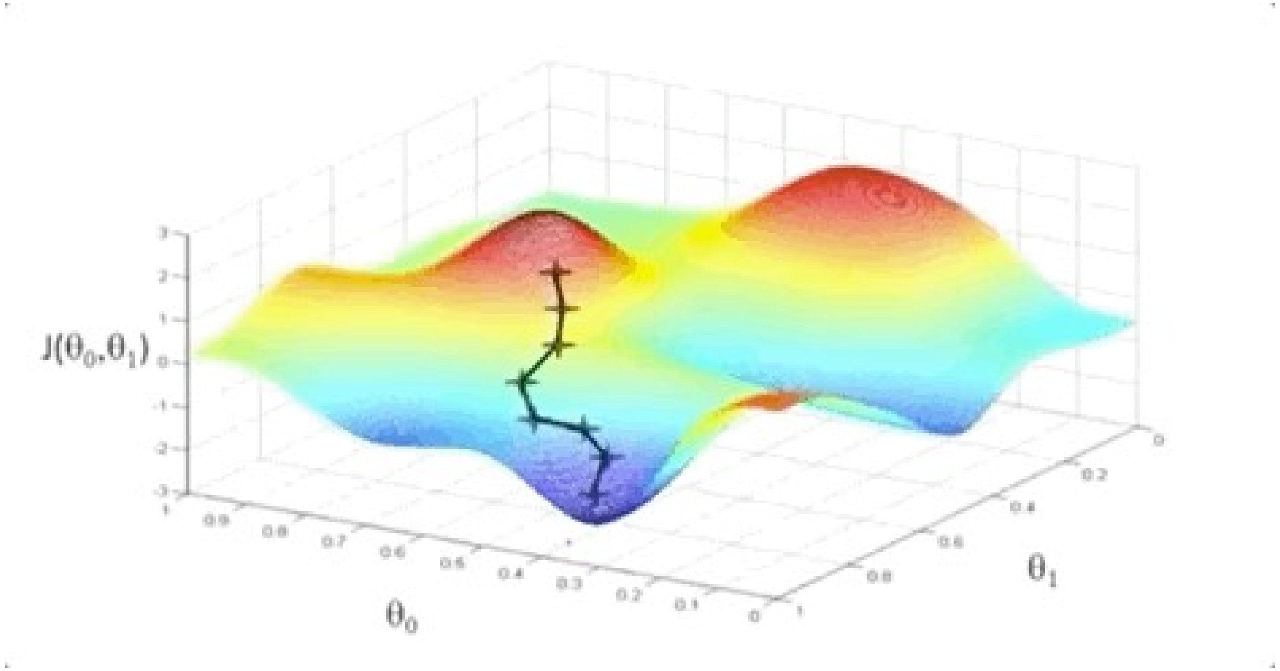


Gradient Descent Algorithm

- The algorithm to update ***w***

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla J(\boldsymbol{w}_\mathrm{i})$$

w at next step ***w*** at current step step size gradient



Algorithm Main Steps

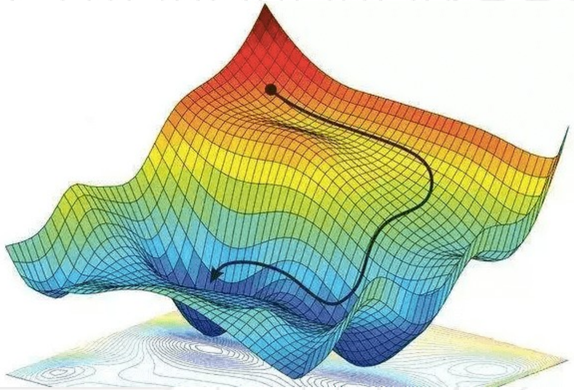
- Initialize $\boldsymbol{w}$ with random values
- We take one step $\boldsymbol{i}$ at a time
- Measure the local gradient of the cost function $\nabla J(\boldsymbol{w}_{\mathrm{i}})$
- Update $\boldsymbol{w}$ with direction of descent by the step size (learning rate)
- Stop when converging - the update is too small (say less than ϵ)
- Stop when diverging - maximum iterations reached (max_iter)

α : learning rate

$\nabla J(\boldsymbol{w}_{\mathrm{i}})$: gradient at value of $\boldsymbol{w}_i$

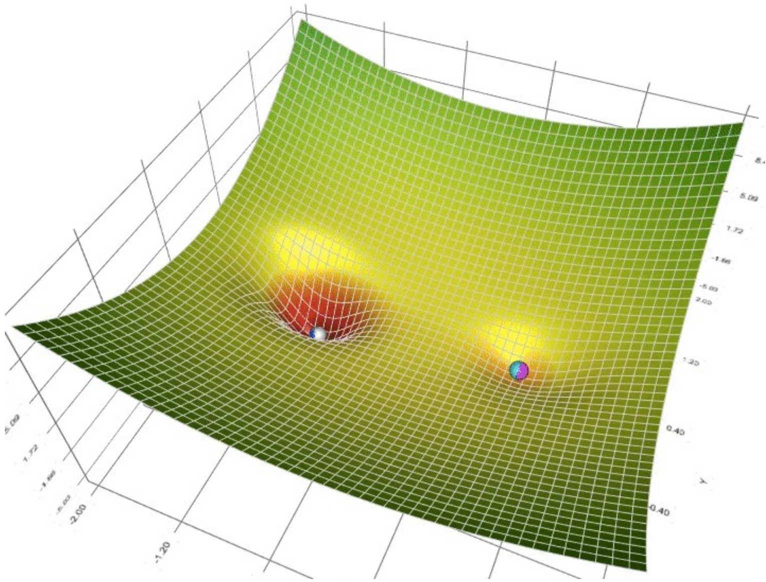
$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla J(\boldsymbol{w}_{\mathrm{i}})$: update

Elements of the GDA



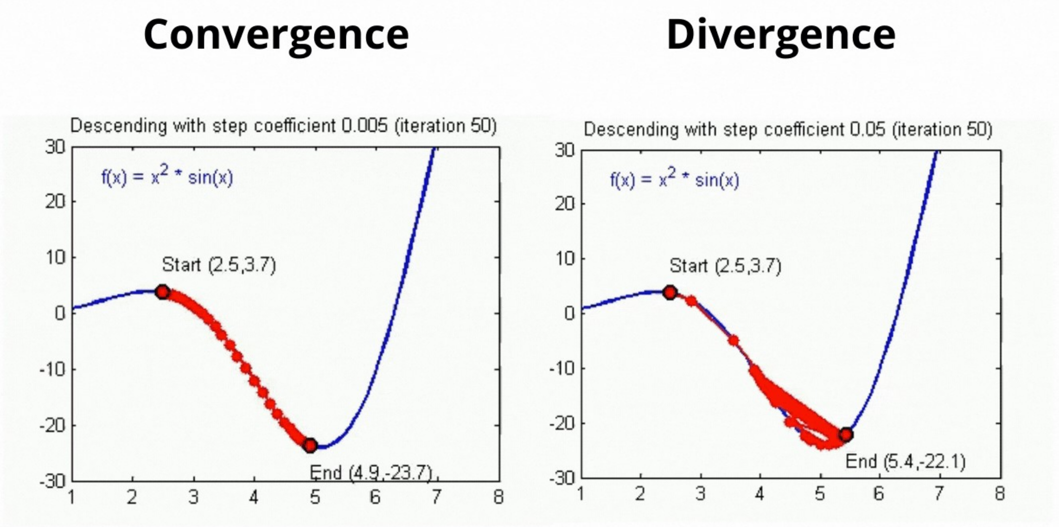
Initializing *w*

- The parameters are initialized randomly
- Random normal initializations for weights are common
- The initialization affects the convergence to a minimum



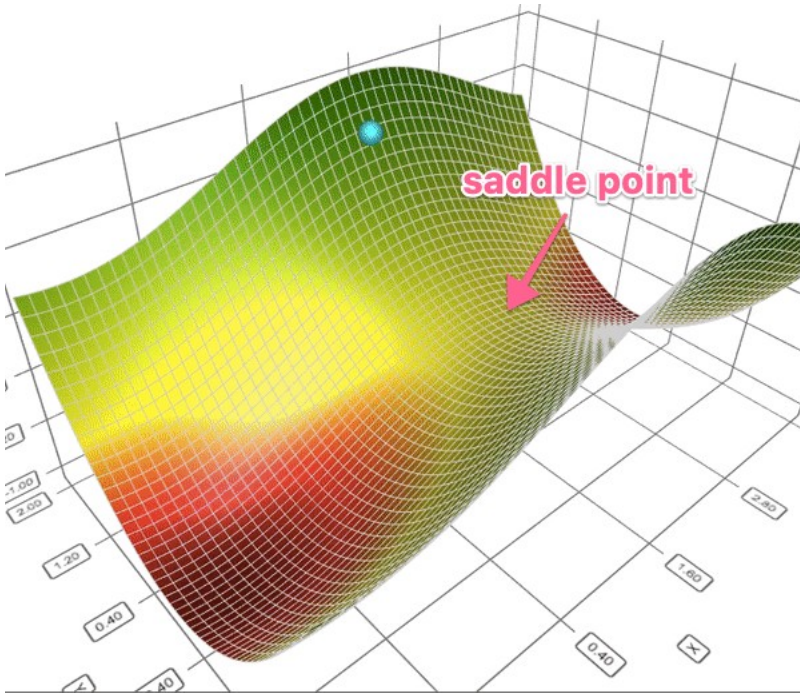
Learning Rate - α

- This is the size of the step taken
- A too small learning rate can cause the algorithm to take too many steps to converge to the minimum
- A too large learning rate can overshoot (diverge from) the minimum



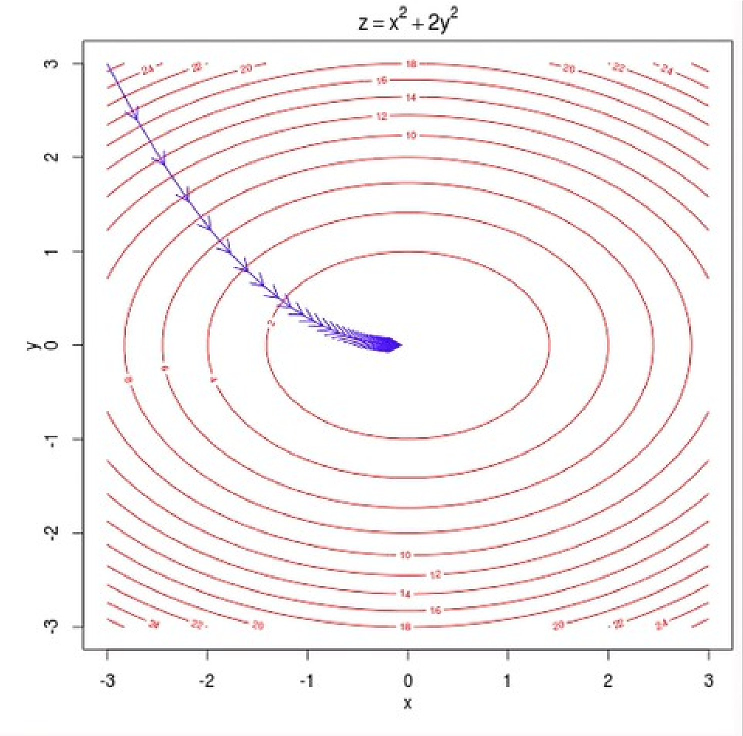
Gradients $\nabla J(\boldsymbol{w})$ Can be Zero at:

- ❑ A global minimum
- ❑ A local minimum
- ❑ A saddle point (neither a local or global minimum)



Number of Iterations:

- Gradient descent might continue to run infinitely if not stopped
- The maximum number of steps (**iterations)** are set in advance
- Most algorithms have parameters such as max_iter, iter, n_iter

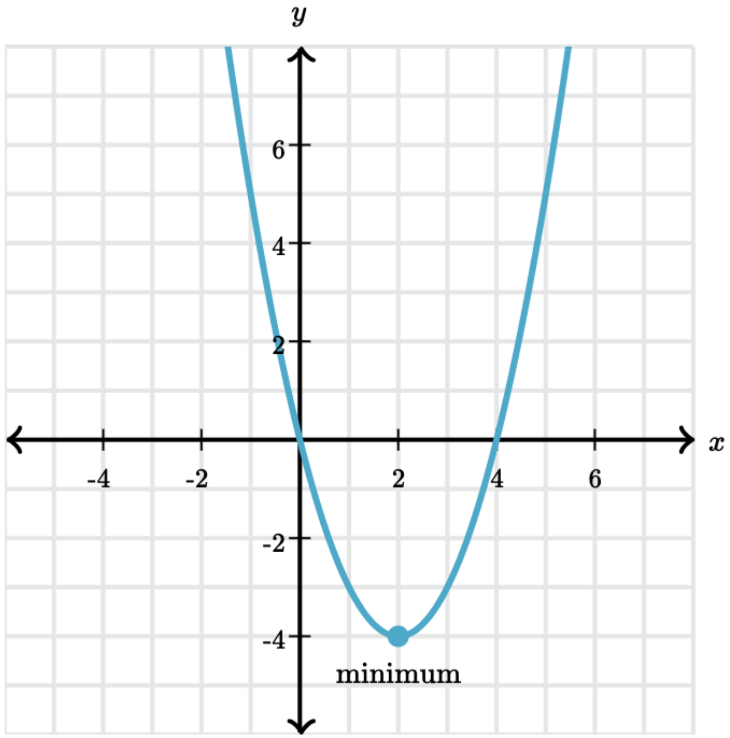


Python

Example: Convex

- Suppose we want to estimate just one parameter $\boldsymbol{w}$
- The cost function $\boldsymbol{J(w)}$ is given below:

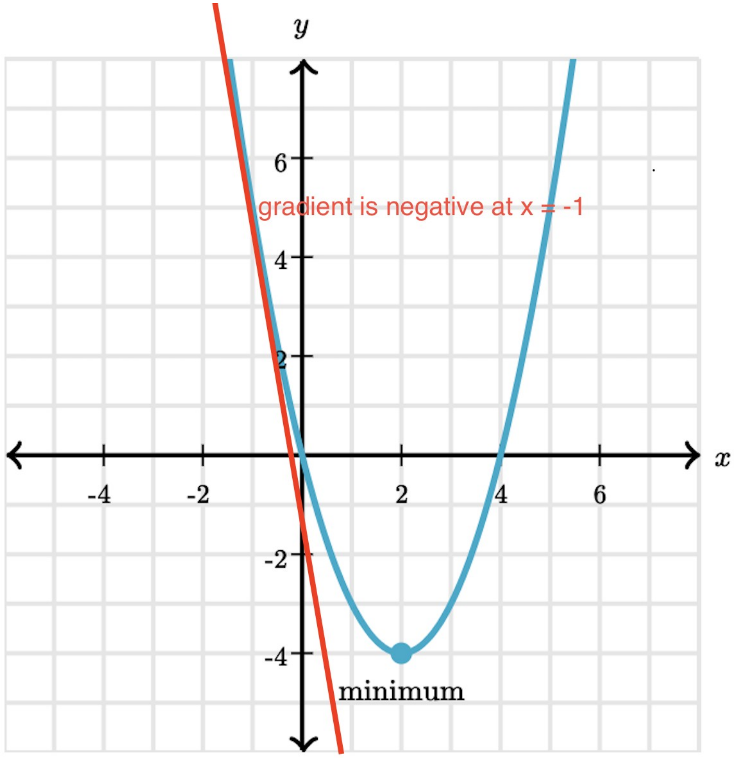
$$J(w) = w^2 - 4w$$



Example: Convex

- The gradient of the cost function $J(w)$ is given below:
- Let's also compute the gradient at $w = -1$

$$\nabla J(w) = 2w - 4$$
$$\nabla J(-1) = -6$$

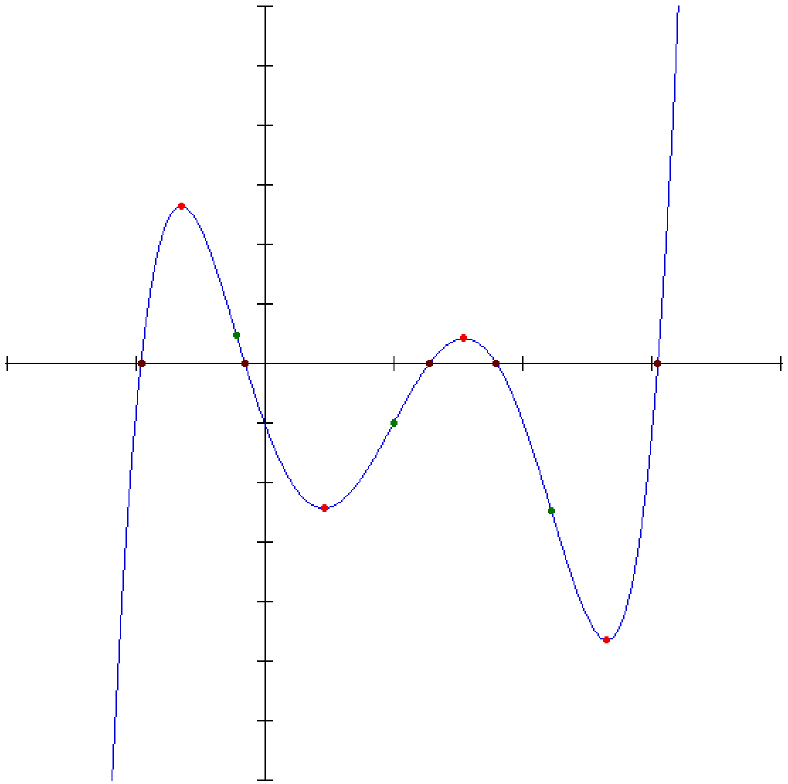


Python

Example: Non-Convex

- Suppose we want to estimate just one parameter $\boldsymbol{w}$
- The cost function $\boldsymbol{J(w)}$ is given below:

$$J(w) = w^5 - 5w^4 + 5w^3 + 5w^2 - 6$$



Python

Gradient Descent for Regression

- Using MSE cost function:

$$\frac{1}{n}||\mathbf{X}\mathbf{w} - y||_2^2$$

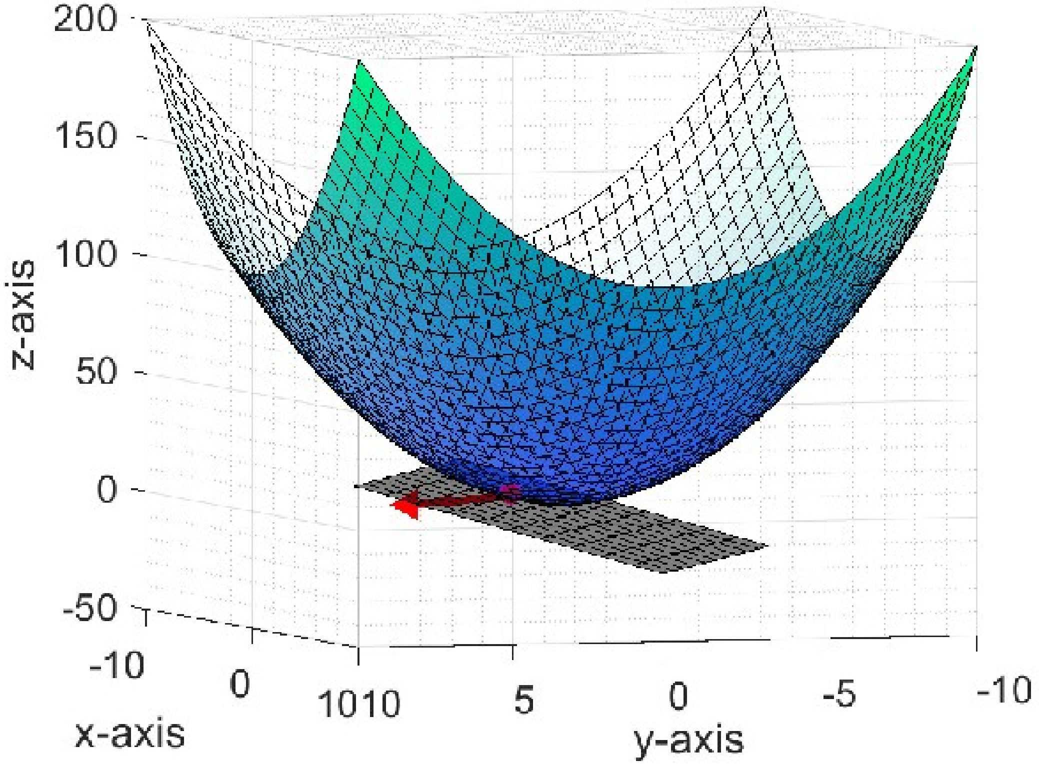
- Gradient

$$\nabla J(\mathbf{w}) = \frac{\delta J(\mathbf{w})}{\delta \mathbf{w}} = \frac{2}{n}\mathbf{X}^\top(\mathbf{X}\mathbf{w} - y)$$

- GDA

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \nabla J(\mathbf{w})$$

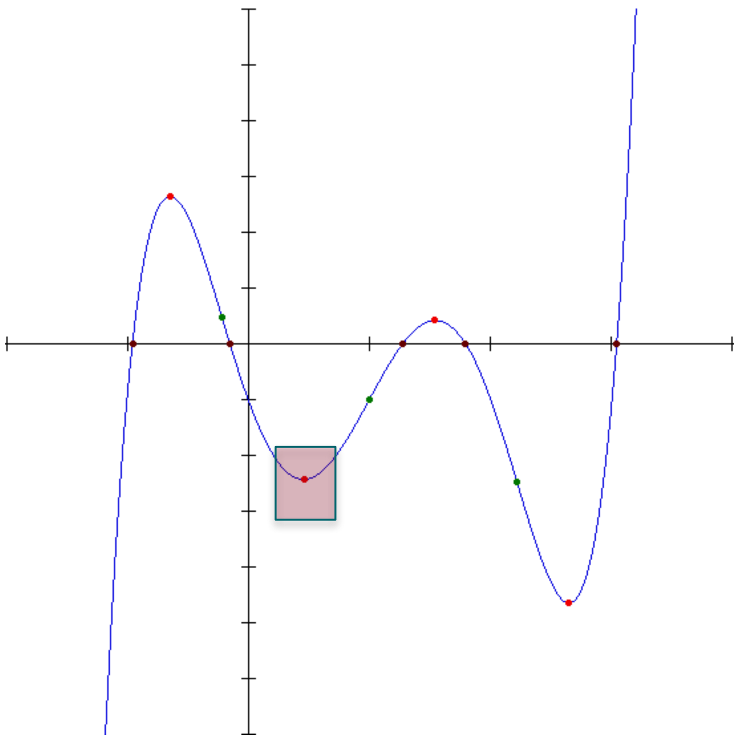
$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \frac{2}{n}\mathbf{X}^\top(\mathbf{X}\mathbf{w} - y)$$



Python

Optimizers:

- Get unstuck of local minima
- Speed up convergence



Batch Gradient Descent

- We can use parts of the data set to update $\boldsymbol{w}$

Sales	Sq_Ft	lot
360000	3032	22221
340000	2058	22912
250000	1780	21345
205500	1638	17342
275500	2196	21786
248000	1966	18902
229900	2216	18639

$\boldsymbol{y}$ $\boldsymbol{X}$

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \frac{2}{n} \boxed{\mathbf{X}}^\top (\boxed{\mathbf{X}}\mathbf{w} - \boxed{y})$$

Batch Gradient Descent

- We can use parts of the data set to update $\boldsymbol{w}$

Sales	Sq_Ft	lot
360000	3032	22221
340000	2058	22912
250000	1780	21345
205500	1638	17342
275500	2196	21786
248000	1966	18902
229900	2216	18639

$\{X_i, y_i\}_{i=1}^{2}$ - **Batch = 2**

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \frac{2}{n} \mathbf{X}^\top (\mathbf{X}\mathbf{w} - y)$$

$\boldsymbol{y}$ $\boldsymbol{X}$

Batch Gradient Descent

- We can use parts of the data set to update $\boldsymbol{w}$

Sales	Sq_Ft	lot
360000	3032	22221
340000	2058	22912
250000	1780	21345
205500	1638	17342
275500	2196	21786
248000	1966	18902
229900	2216	18639

$\boldsymbol{y}$ $\boldsymbol{X}$

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \frac{2}{n} \boxed{\mathbf{X}}^\top (\boxed{\mathbf{X}}\mathbf{w} - \boxed{y})$$

Batch Gradient Descent

- We can use parts of the data set to update $\boldsymbol{w}$

Sales	Sq_Ft	lot
360000	3032	22221
340000	2058	22912
250000	1780	21345
205500	1638	17342
275500	2196	21786
248000	1966	18902
229900	2216	18639

$\boldsymbol{y}$ $\underbrace{\hphantom{xxxxxxxx}}_{\boldsymbol{X}}$

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \frac{2}{n} \boxed{\mathbf{X}}^\top (\boxed{\mathbf{X}}\mathbf{w} - \boxed{y})$$

Batch Gradient Descent

- We can use parts of the data set to update ***w***

Sales	Sq_Ft	lot
360000	3032	22221
340000	2058	22912
250000	1780	21345
205500	1638	17342
275500	2196	21786
248000	1966	18902
229900	2216	18639

$\boldsymbol{y}$ $\qquad \underbrace{\qquad\qquad}_{\boldsymbol{X}}$

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \frac{2}{n} \boxed{\mathbf{X}}^\top (\boxed{\mathbf{X}}\mathbf{w} - \boxed{y})$$

Completed 1 epoch

Stochastic Gradient Descent (SGD)

- We can use parts of the data set to update $\boldsymbol{w}$

Sales	Sq_Ft	lot
360000	3032	22221
340000	2058	22912
250000	1780	21345
205500	1638	17342
275500	2196	21786
248000	1966	18902
229900	2216	18639

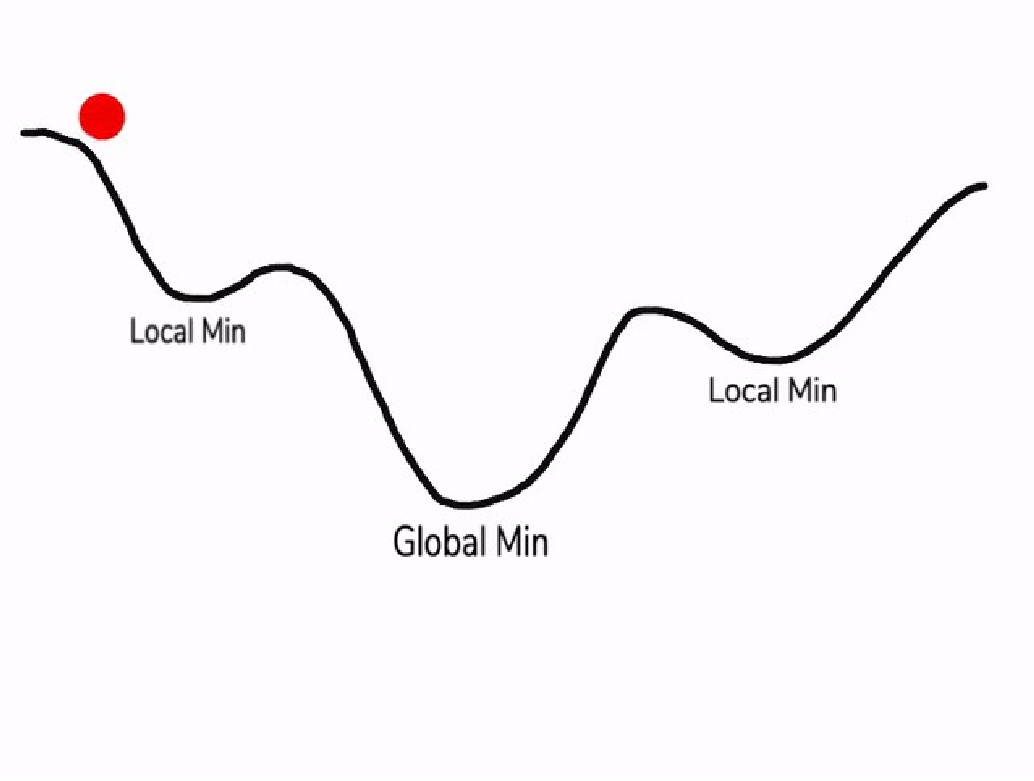
$\boldsymbol{y}$ $\qquad$ $\boldsymbol{X}$

Mini batch of 1 observation

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \frac{2}{n} \boxed{\mathbf{X}}^\top (\boxed{\mathbf{X}}\mathbf{w} - \boxed{y})$$

Momentum

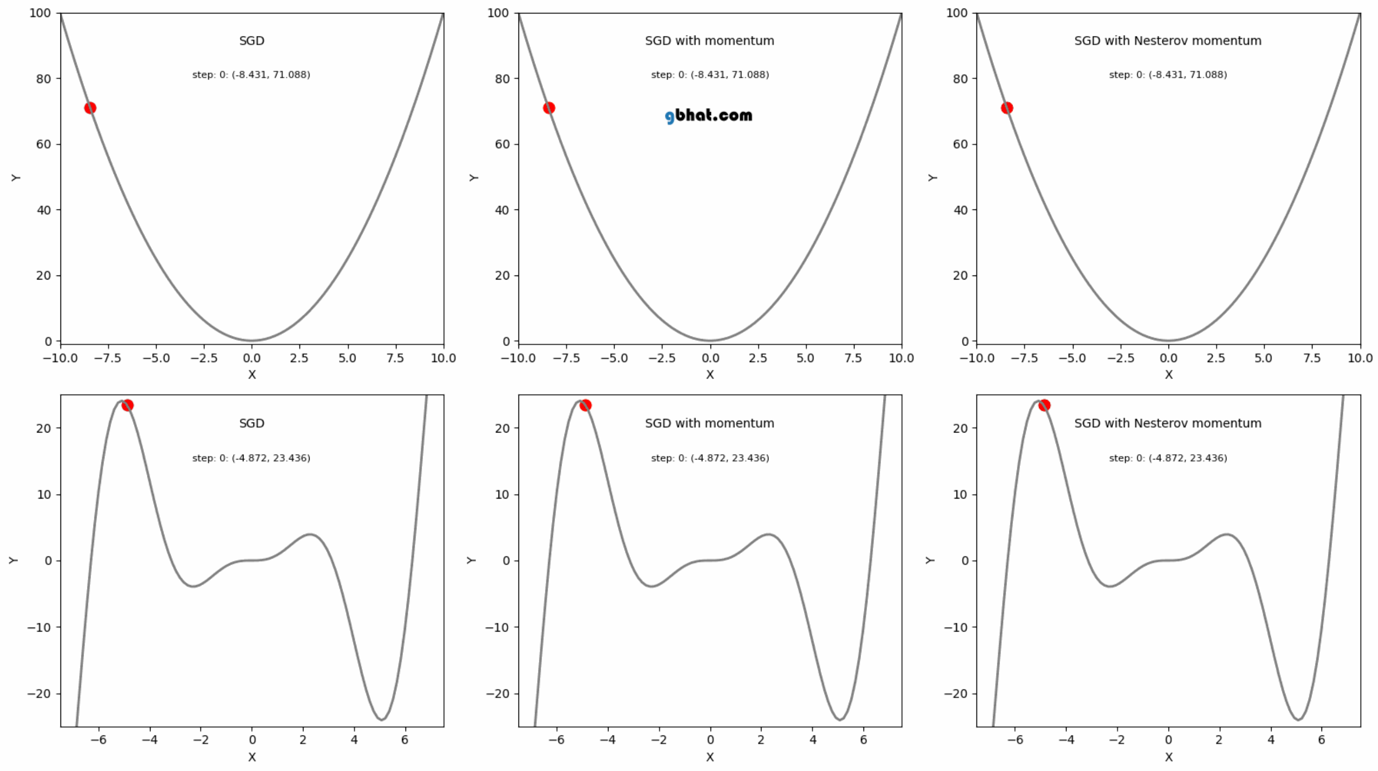
- Improves on gradient descent by using the acceleration of cost.
- Goal: get unstuck



	Derivative Form
Position	$r(t)$
Velocity	$v(t) = \frac{dr}{dt}$
Acceleration	$a(t) = \frac{dv}{dt} = \frac{d^2r}{dt^2}$

Adam Optimizer

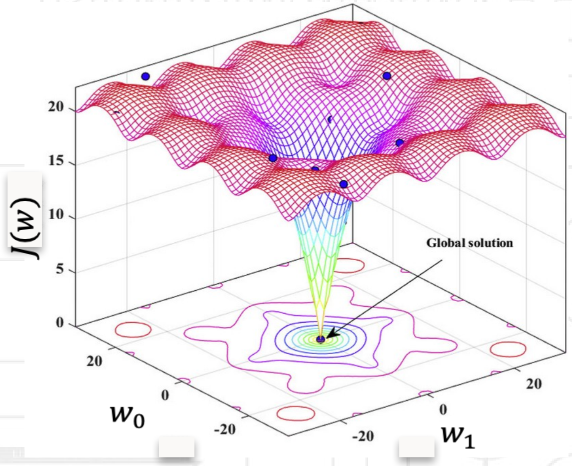
- Uses both velocity and acceleration



Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

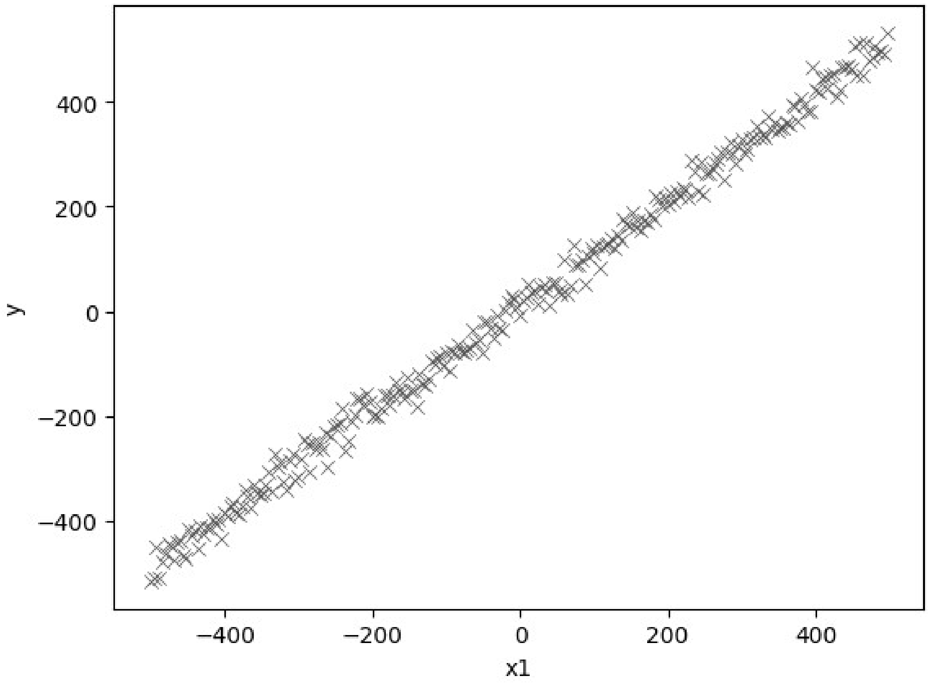
Python

Gradient Descent in LR



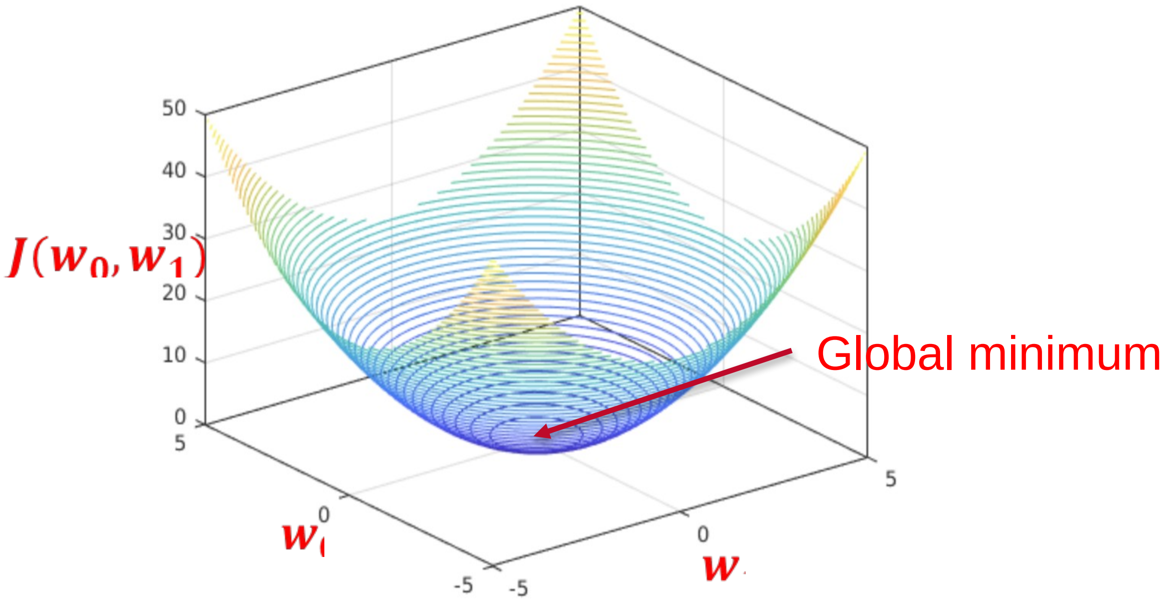
Example: Convex Cost Function

- ❑ Assume there is an unknown function $f: \boldsymbol{X} \xrightarrow{maps} y$
- ❑ Assume a linear estimator (hypothesis) $\boldsymbol{h}: \boldsymbol{X} \xrightarrow{maps} \boldsymbol{y}$ works well



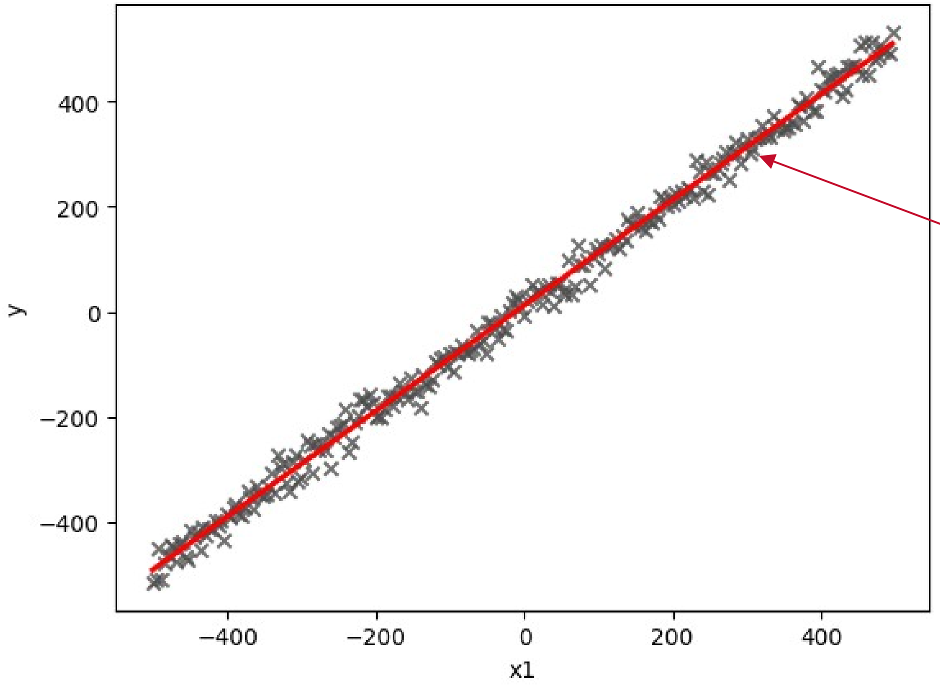
Example: Convex Cost Function

- Assume there is a unknown function $f: \boldsymbol{X} \xrightarrow{maps} y$
- Assume a linear estimator (hypothesis) $h: \boldsymbol{X} \xrightarrow{maps} y$ works well
- If h is linear regression, then cost function <u>SSE</u> is convex



Example: Convex Cost Function

- We can find the parameter vector $\boldsymbol{w} = [w_0, w_1]$ minimizing SSE
- We do need to create a design matrix $\boldsymbol{X} = [\mathbf{1}\ \boldsymbol{x}_1]$
- The model predictions would be: $\mathbf{h_w(X)} = \mathbf{X}\,\boldsymbol{w}$



$\mathbf{X}\,\boldsymbol{w}$

Example: Convex Cost Function

- We can find the parameter vector $\boldsymbol{w} = [w_0, w_1]$ minimizing SSE
- We do need to create a design matrix $\boldsymbol{X} = [\mathbf{1}\ \boldsymbol{x}_1]$
- The model predictions would be: $\mathbf{h}_{\mathbf{w}}(\mathbf{X}) = \mathbf{X}\,\boldsymbol{w}$

Bias or Intercept features

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & \cdots & x_{2n} \\ 1 & x_{31} & x_{32} & \cdots & \cdots & x_{3n} \\ \vdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ 1 & x_{m1} & x_{m2} & \cdots & \cdots & x_{mn} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \vdots \\ \beta_m \end{bmatrix} + \epsilon$$

1) Create y vector

2) Create X Matrix (also Called the Design/Model Matrix)

Example: Convex Cost Function

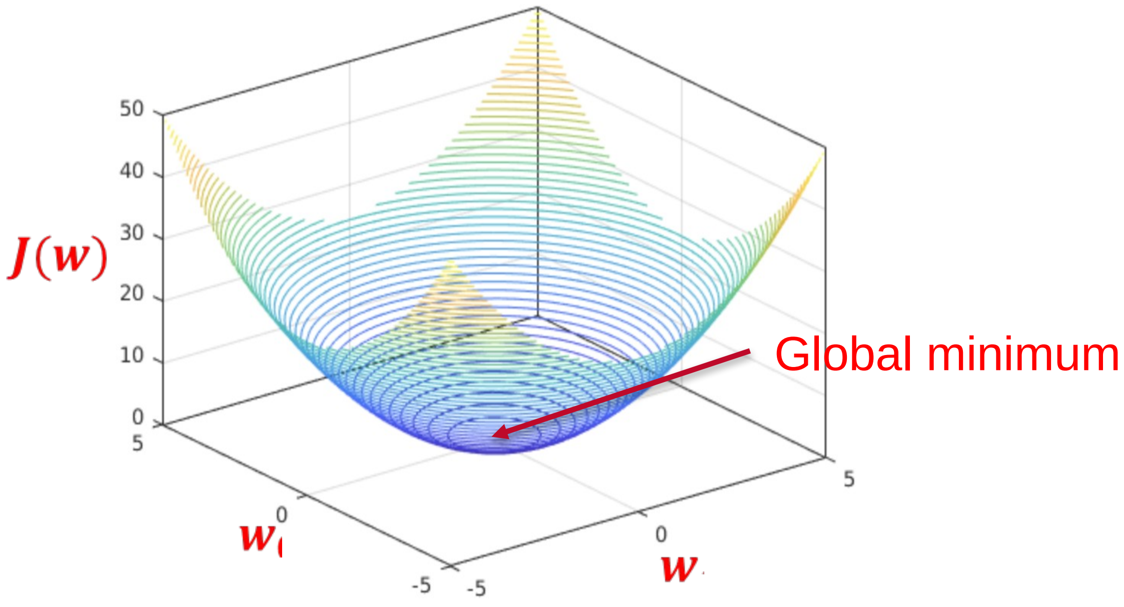
- We can find the parameter vector $\boldsymbol{w} = [w_0, w_1]$ minimizing SSE
- We do need to create a design matrix $\boldsymbol{X} = [\mathbf{1}\ \boldsymbol{x}_1]$
- The model predictions would be: $\mathbf{h_w}(\mathbf{X}) = \mathbf{X}\,\boldsymbol{w}$

$$\mathbf{X} = \begin{bmatrix} 1 & X_{11} & X_{12} & \cdots & X_{1k} \\ 1 & X_{21} & X_{22} & \cdots & X_{2k} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & X_{n1} & X_{n2} & \cdots & X_{nk} \end{bmatrix} \qquad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_k \end{bmatrix}$$

Example: Convex Cost Function

- Cost function is SSE

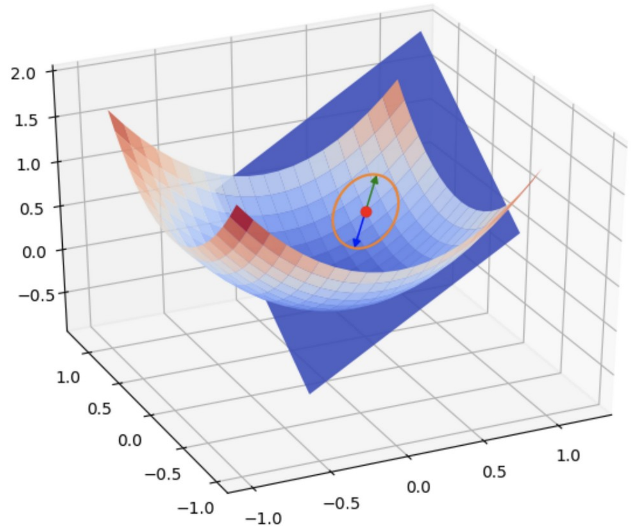
$$\mathbf{J(w)} = (\mathbf{y} - \mathbf{Xw})^{\top}(\mathbf{y} - \mathbf{Xw}) = \|\mathbf{y} - \mathbf{Xw}\|_2^2$$



Example: Convex Cost Function

- Gradient is the derivative of the cost function WRT parameters.

$$\nabla J(\mathbf{w}) = \frac{\delta J(\mathbf{w})}{\delta \mathbf{w}} = -2(\mathbf{X}^\top y) + 2(\mathbf{X}^\top \mathbf{X}\mathbf{w}) = \boxed{2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - y)}$$



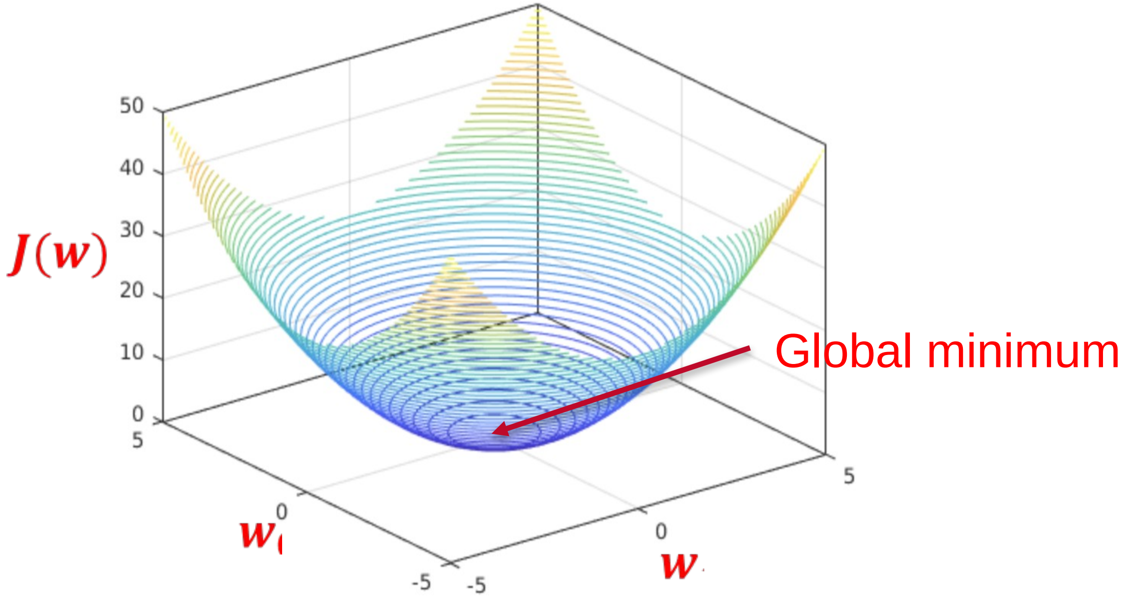
Example: Convex Cost Function

- Setting the gradient to zero and solving for w gets solution for min.

$$\nabla J(\mathbf{w}) = \frac{\delta J(\mathbf{w})}{\delta \mathbf{w}} = -2(\mathbf{X}^\top y) + 2(\mathbf{X}^\top \mathbf{X}\mathbf{w}) = \boxed{2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - y)}$$

Set to zero and solve for w

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1}(\mathbf{X}^\top \mathbf{y})$$



Python