Machine Learning

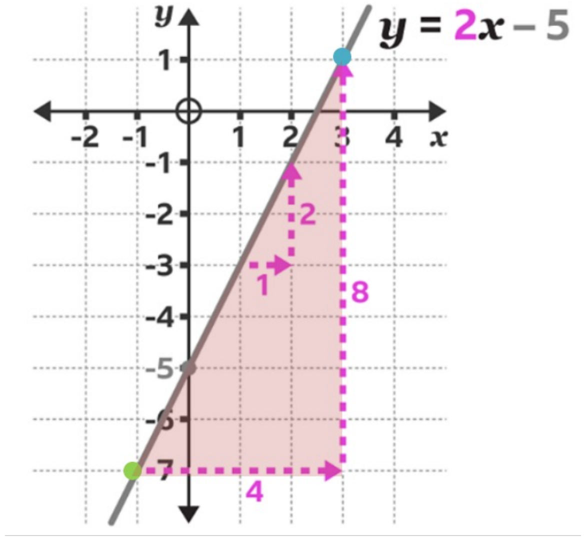
Gradient Descent

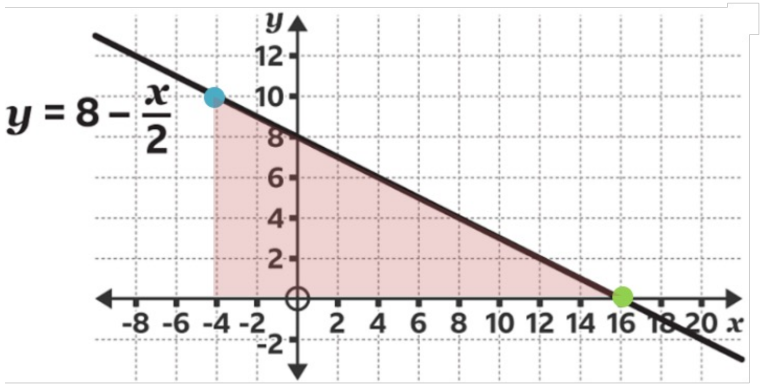
Tarapong Sreenuch

8 February 2024

克明峻德，格物致知

Recap: Gradients

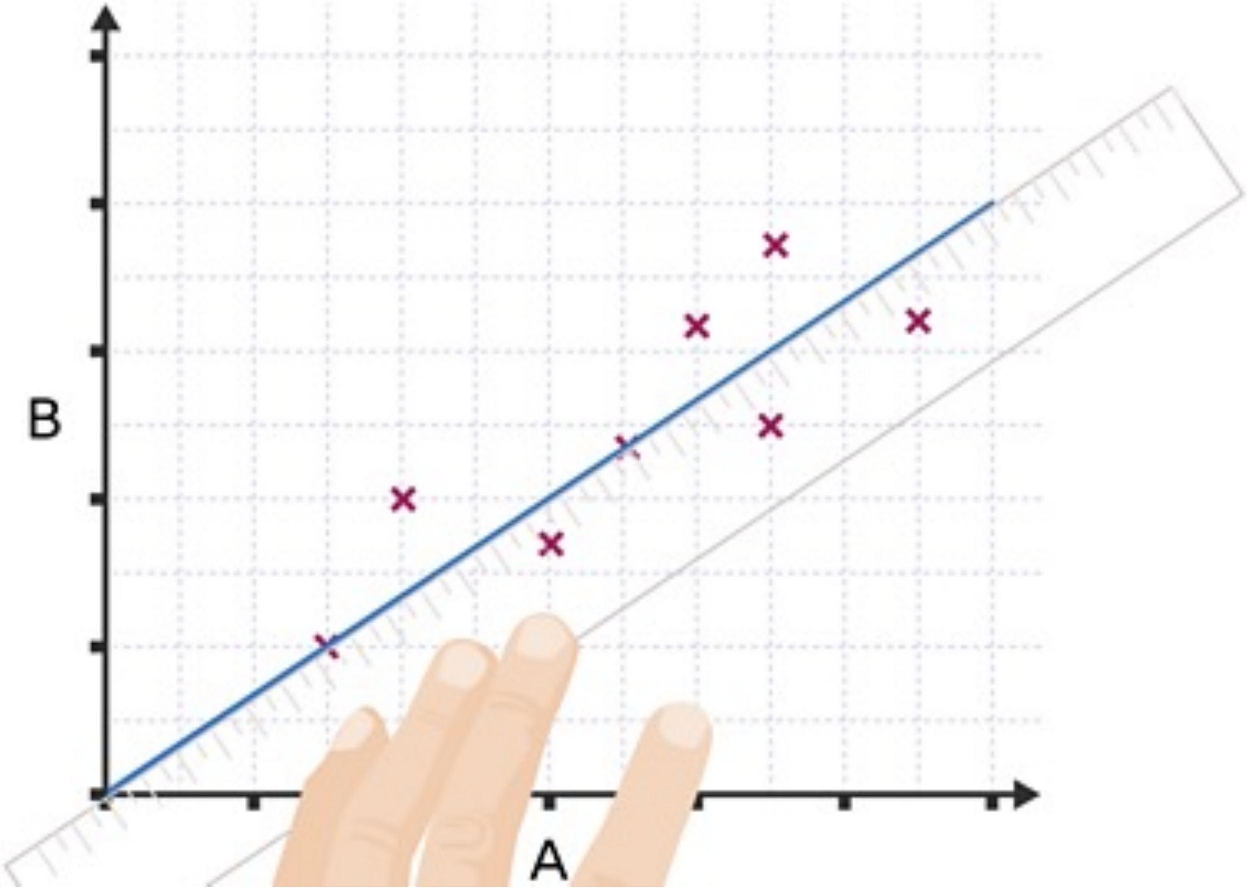




x ⬆ *y* ⬆ ➡ ⊕ *gradient*

x ⬆ *y* ⬇ ➡ ⊖ *gradient*

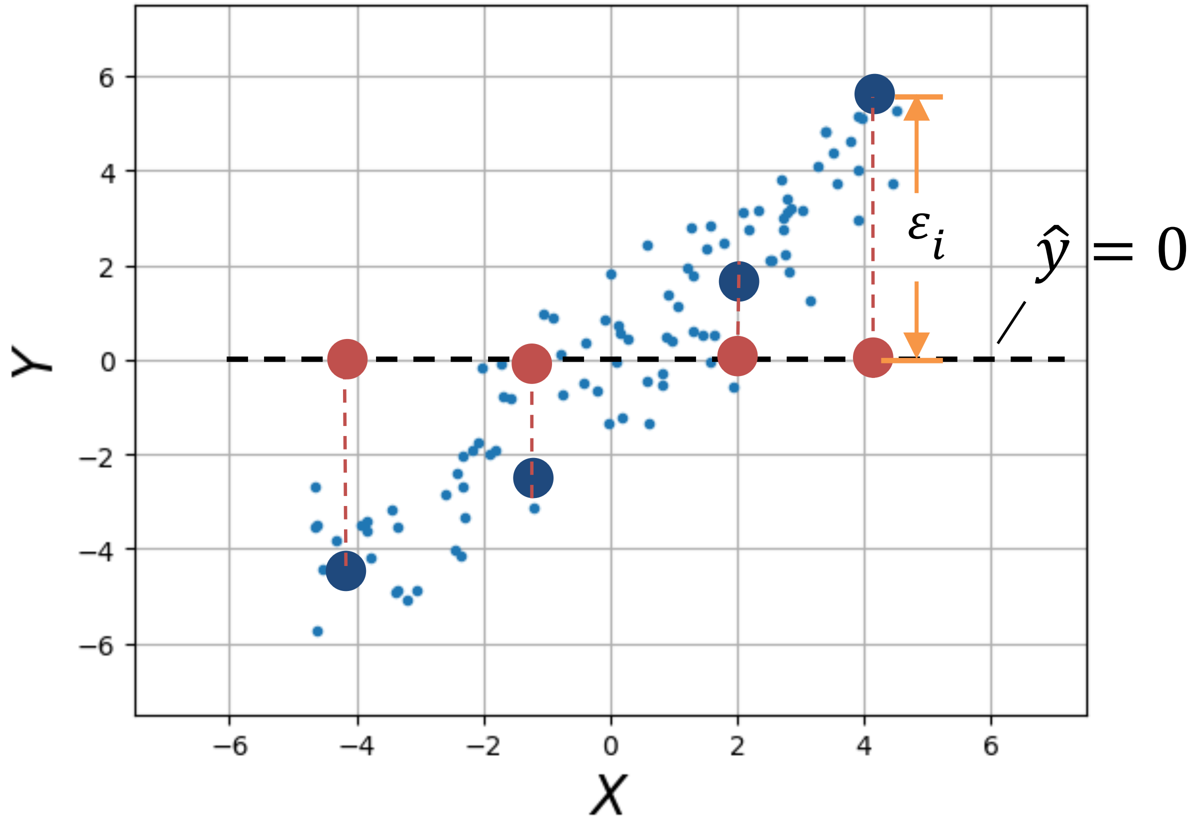
Drawing a Line of Best Fit

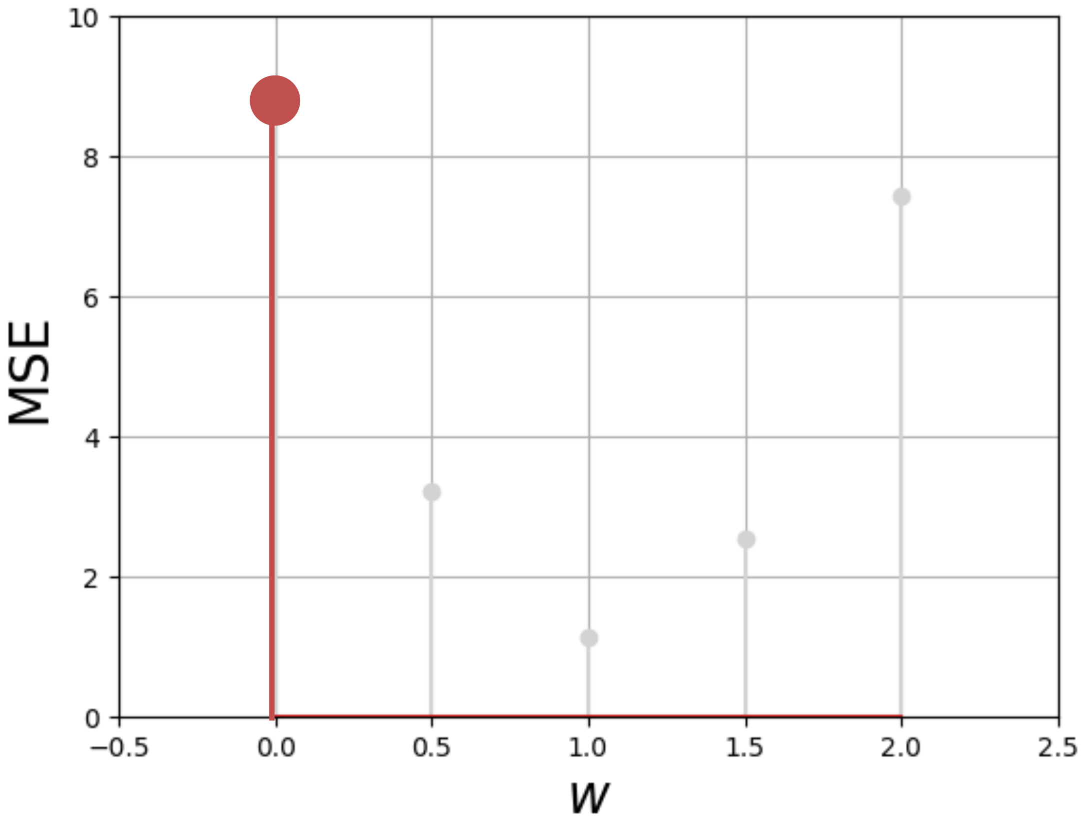


Q: How do we find a line of best fit?

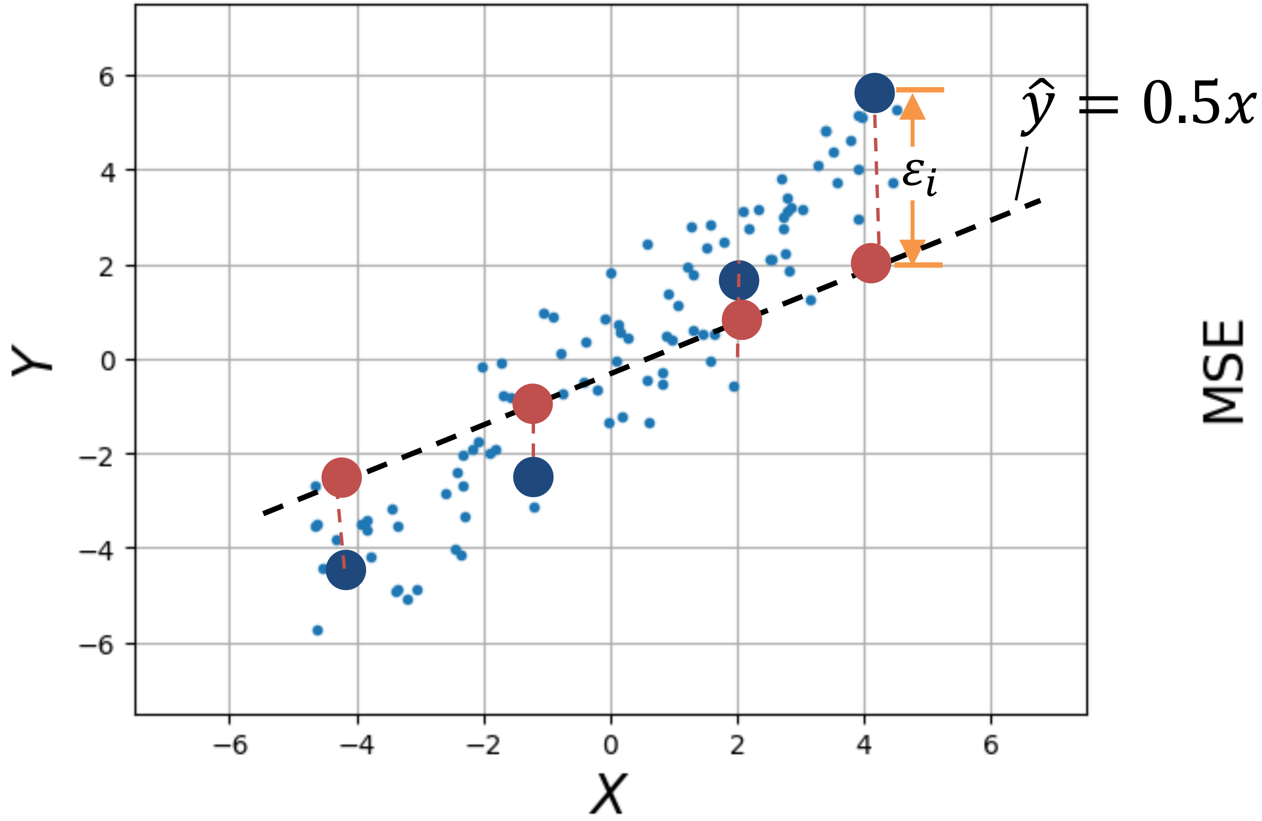
A: By rotating (clockwise/anti-clockwise) and/or shifting (up/down) the ruler, we find a line that goes roughly through the middle of all the middele of all the scatter plots.

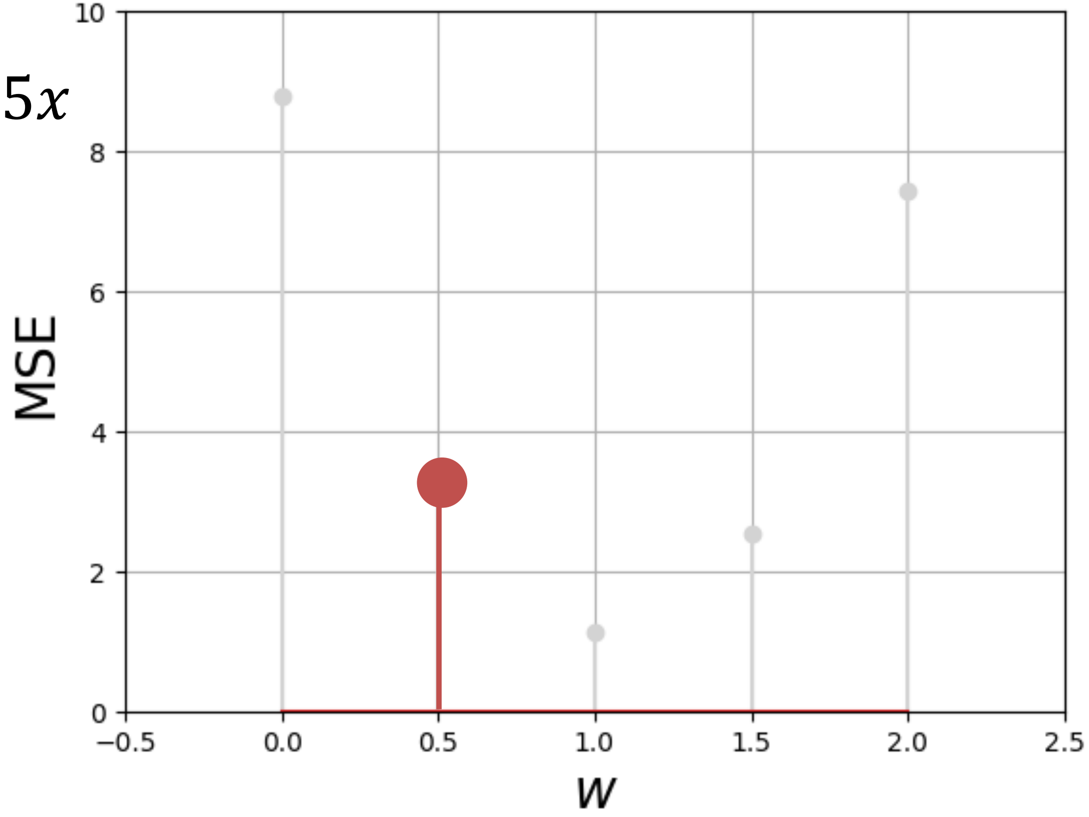
Forming a Linear Model



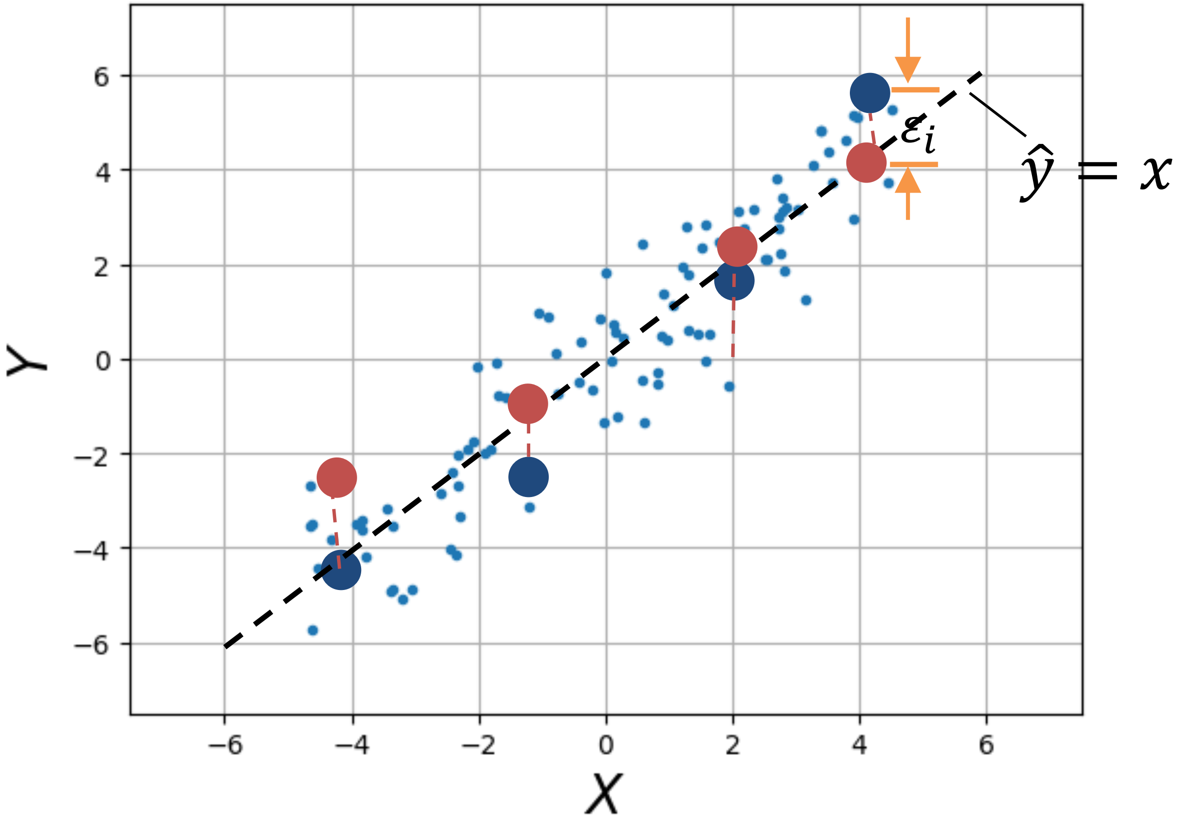


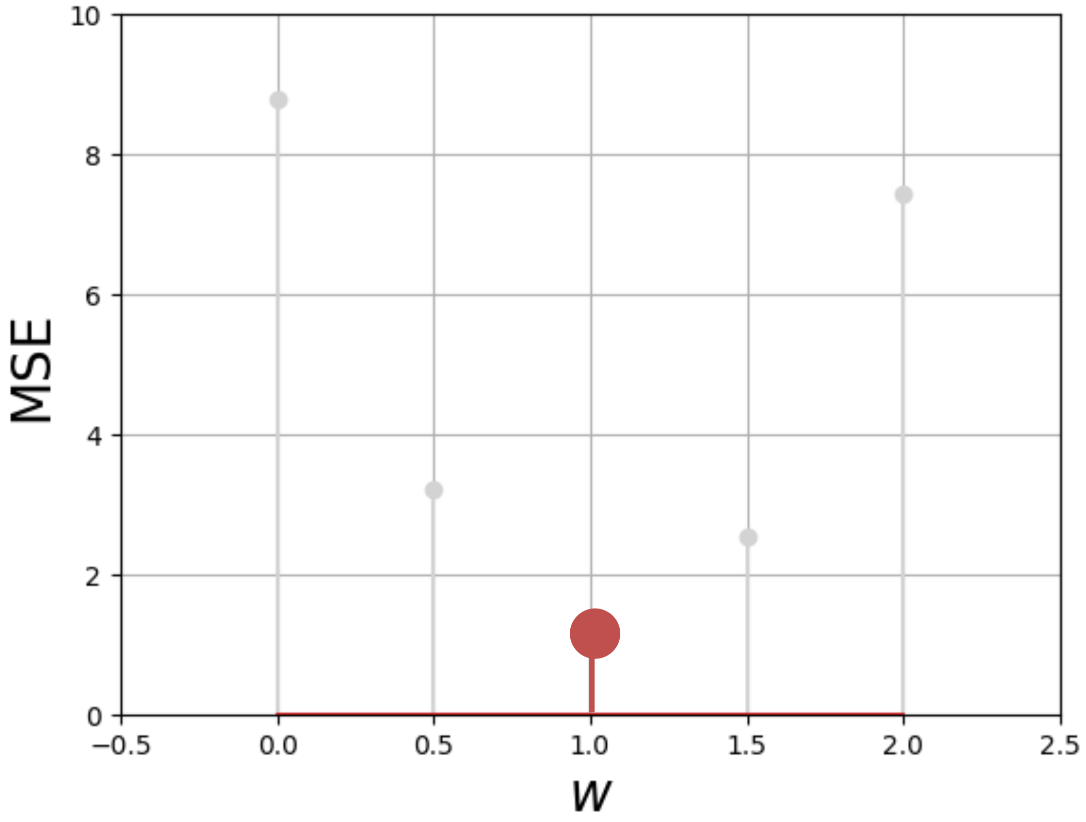
Forming a Linear Model



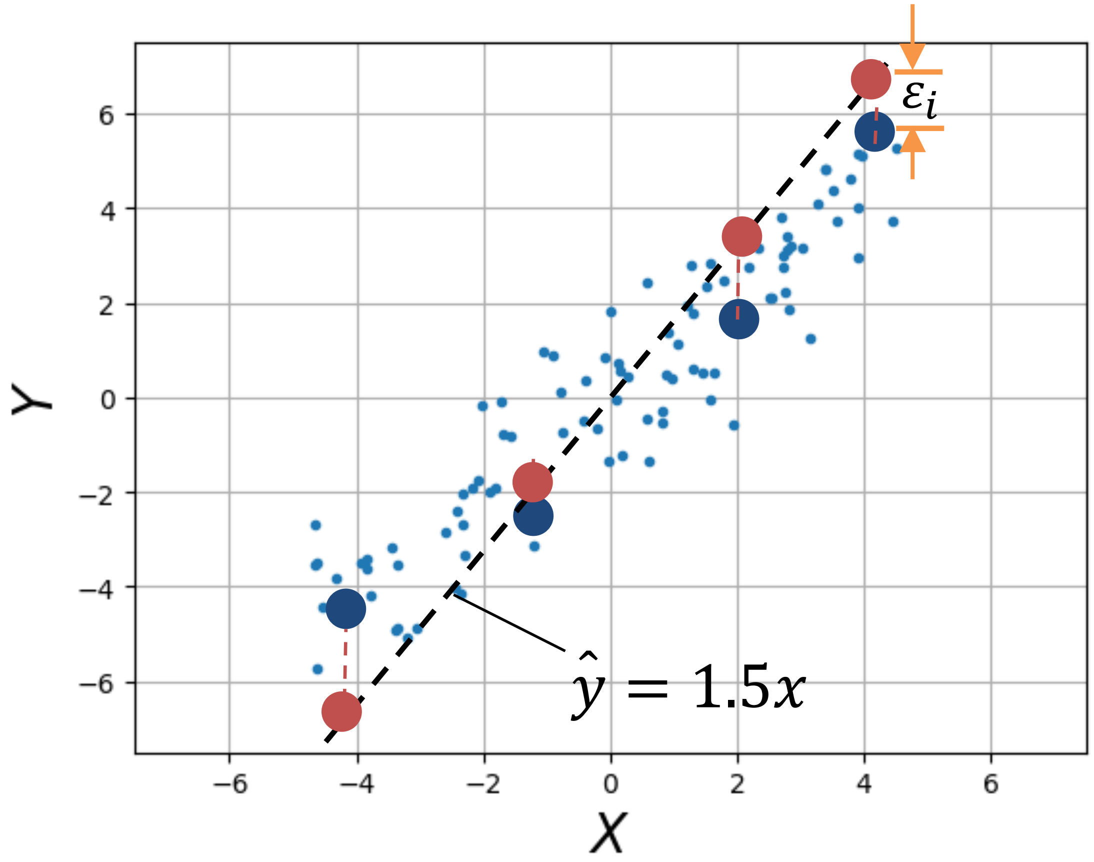


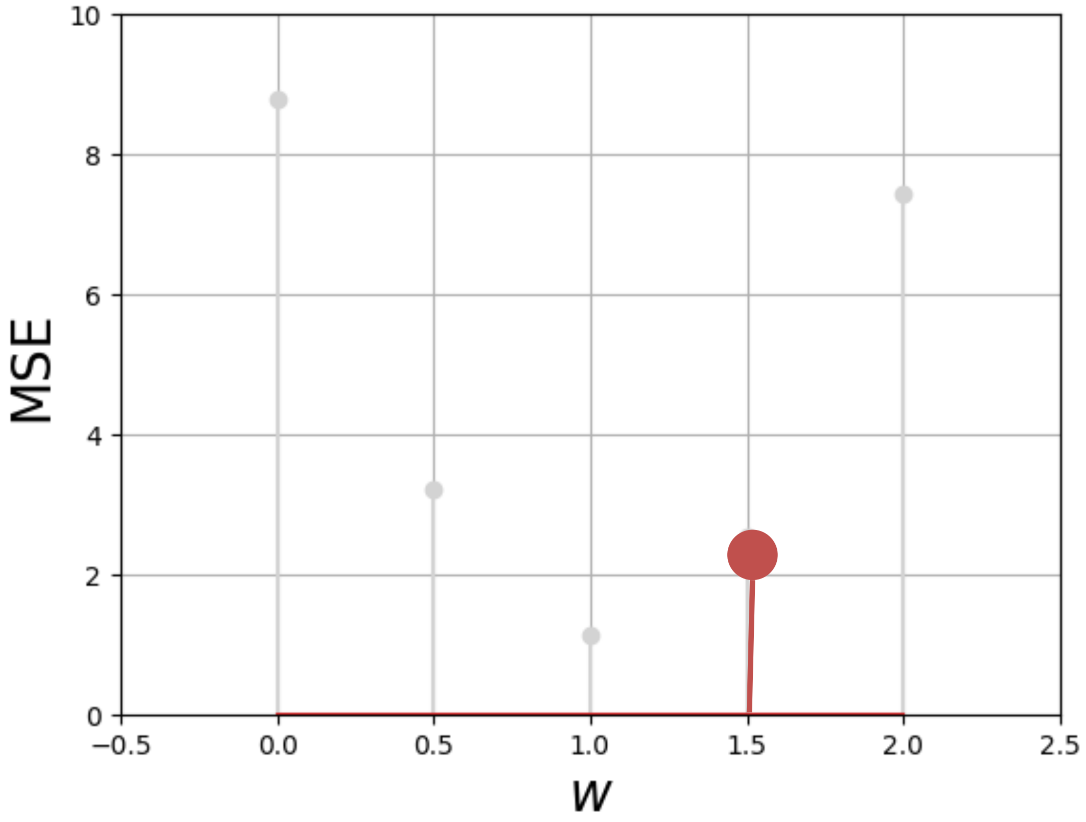
Forming a Linear Model



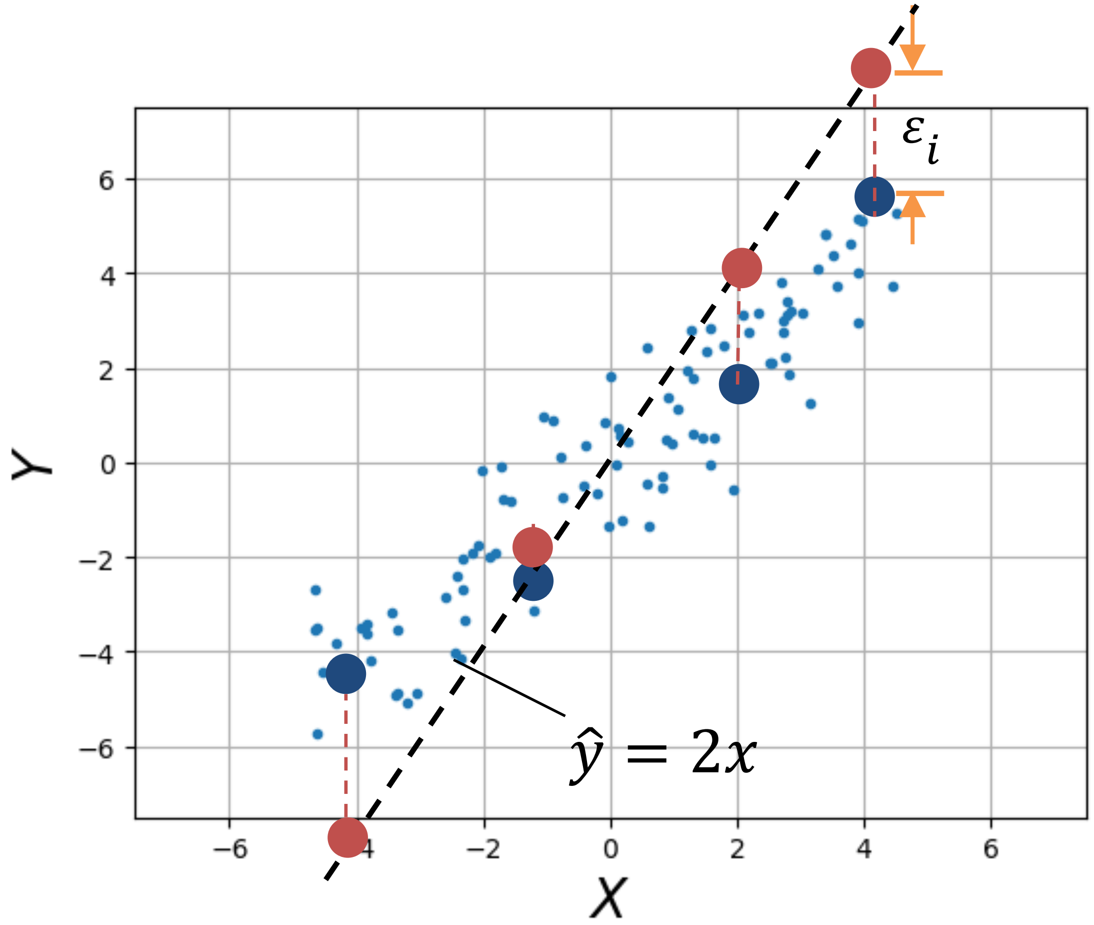


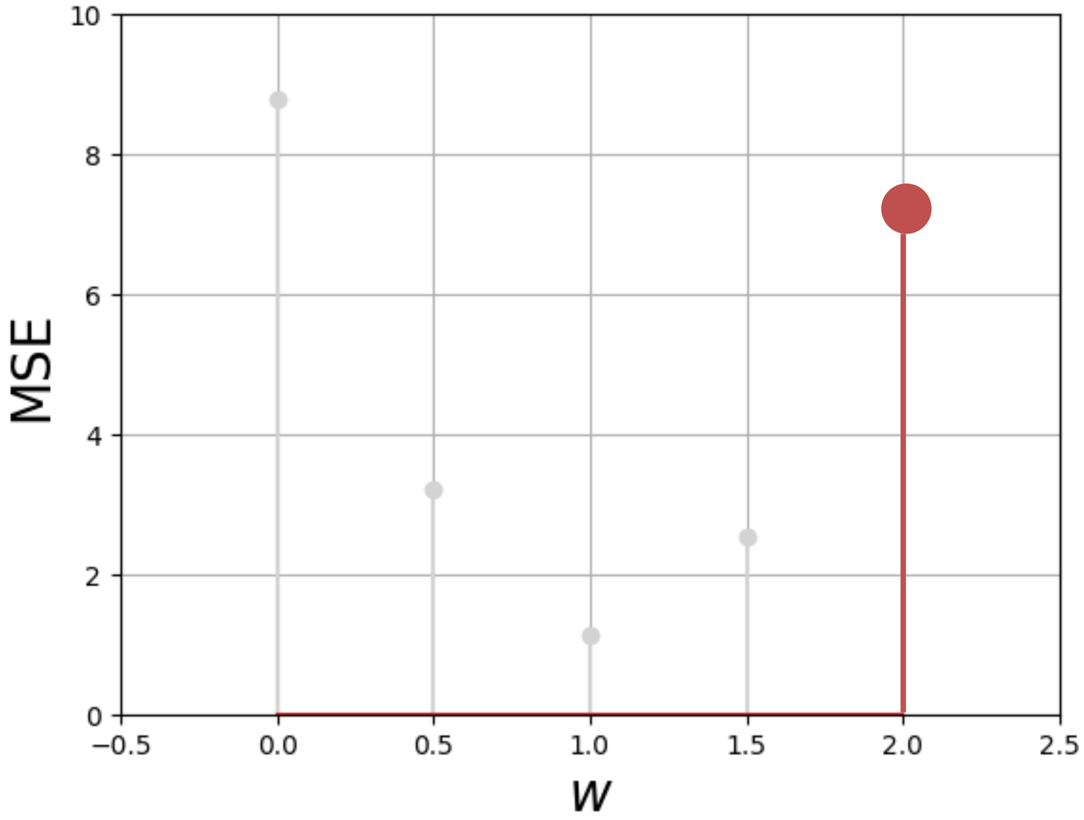
Forming a Linear Model



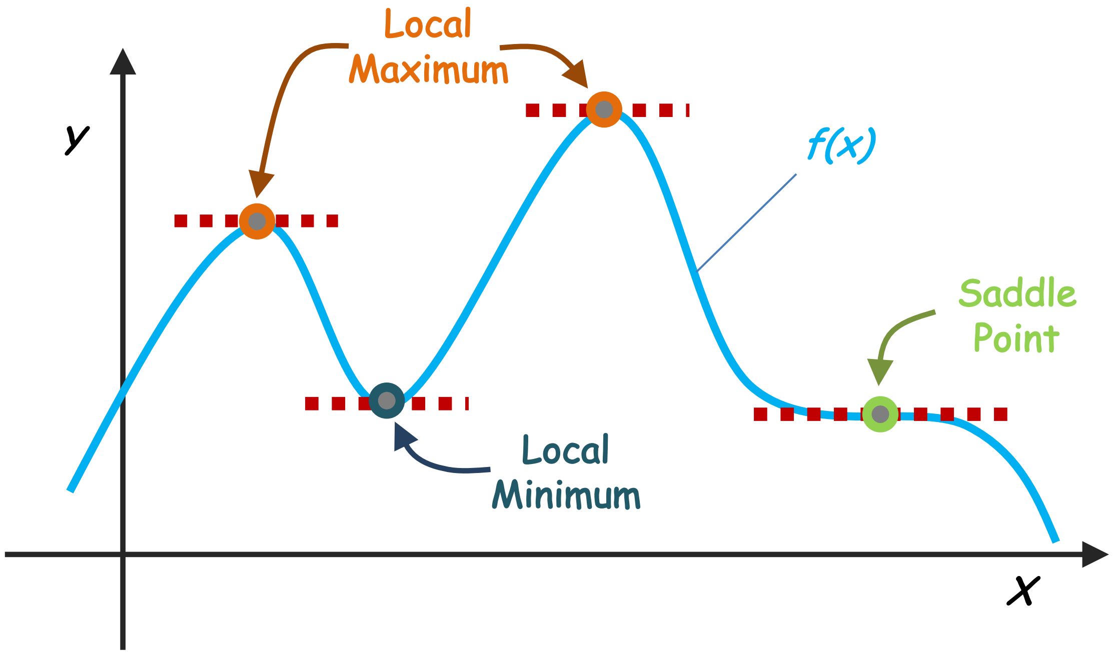


Forming a Linear Model





Calculus



Q: Which value of x will f(x) be either minimum or maximum?

Hint: What will happen to Slope (or Gradient) at those x(s)?

A: … Slope (or Gradient) = 0 …

Best Fit Line

Sum Squared Error (SSE):

$$\mathrm{SSE} \quad = \quad (Y - X\times\vec{w})^T(Y - X\times\vec{w})$$

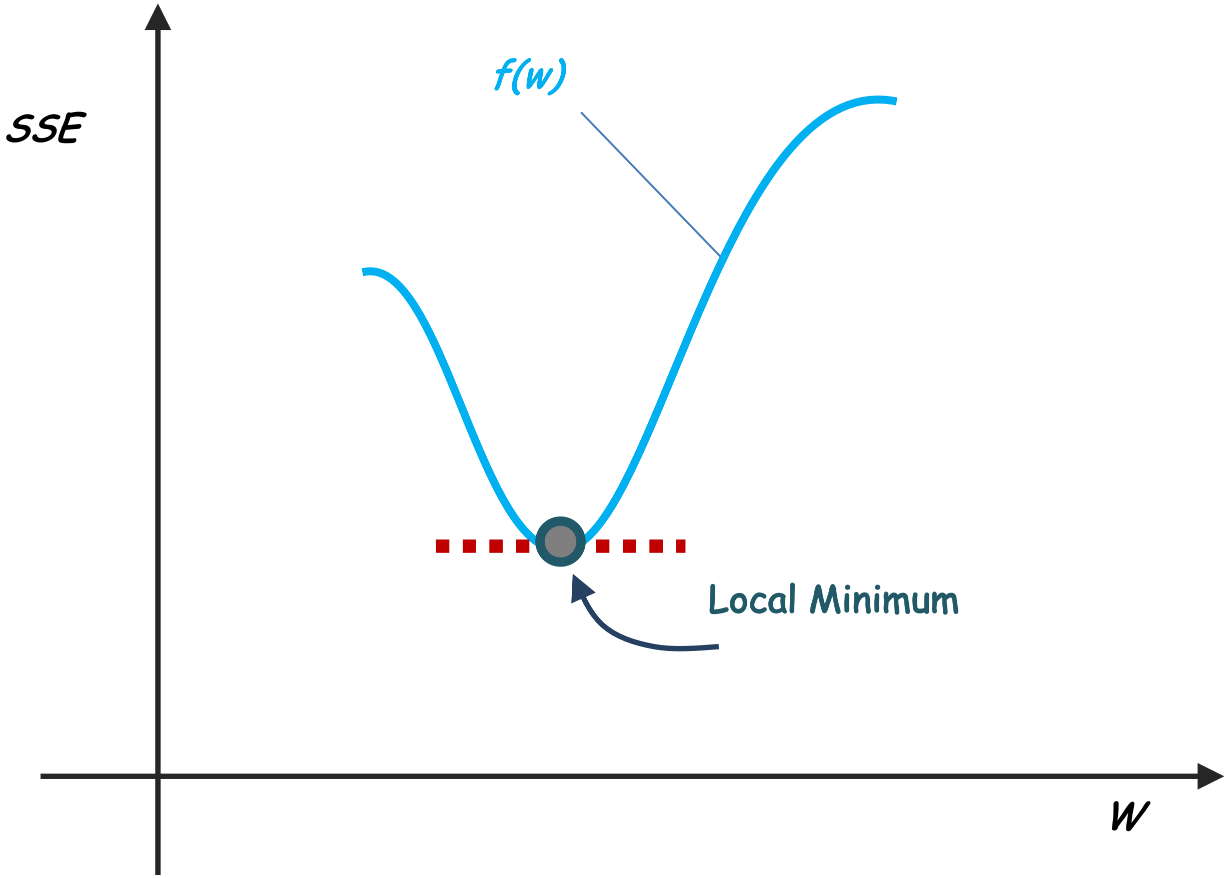
SSE Derivative:

$$\nabla\mathrm{SSE} \quad = \quad 2X^T(Y - X\times\vec{w})$$

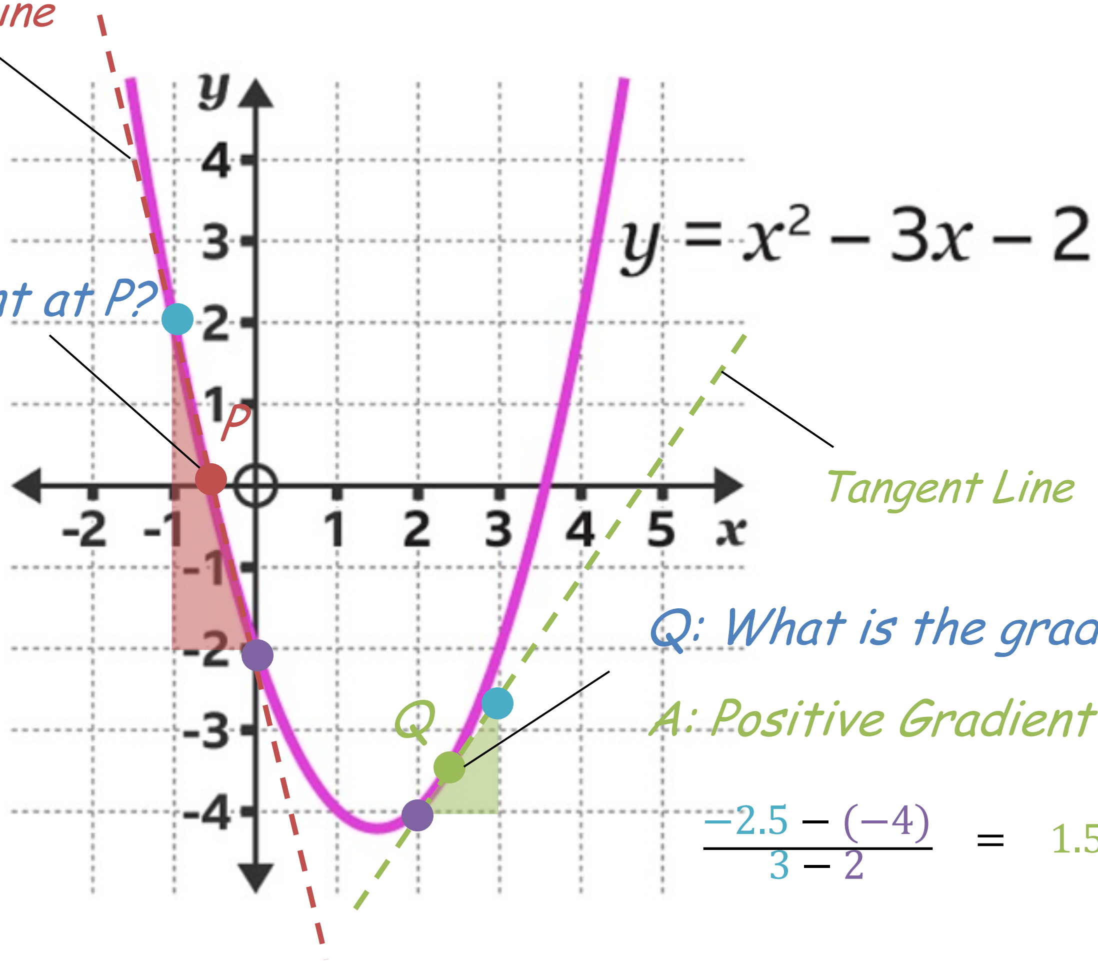
To minimise SSE, we find $\vec{w}$ which results in $\nabla\mathrm{SSE} = 0$.

Q: Which value of w will f(w) be minimum?

A: … Slope (or Gradient) = 0 …



Non-Linear Equation: Gradients



Q: What is the gradient at P?

A: Negative Gradient

$$\frac{-2 - 2}{0 - (-1)} = -4$$

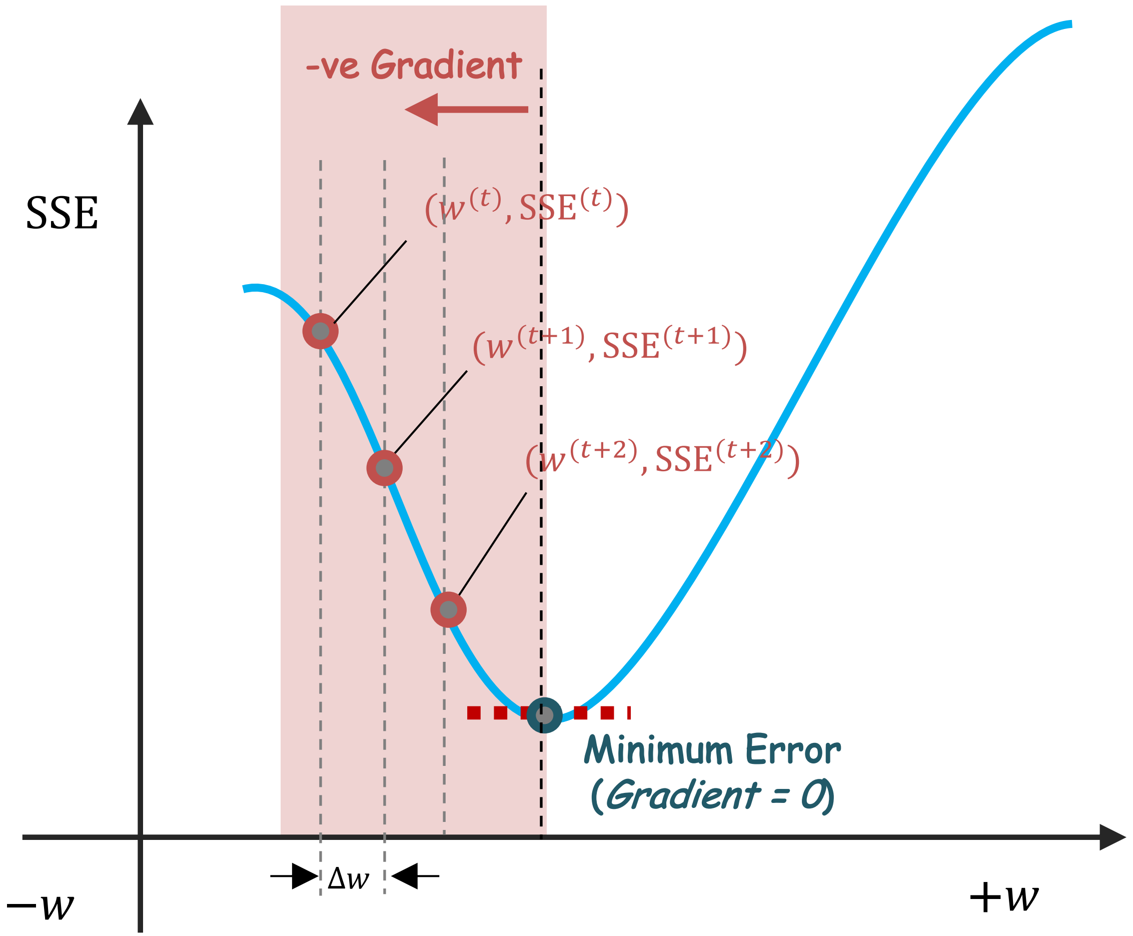
Q: What is the gradient at Q?

A: Positive Gradient

$$\frac{-2.5 - (-4)}{3 - 2} = 1.5$$

The tangent to the curve at P (or Q) has the same gradient as the curve at that point.

Gradient Descent: Intuition


-ve Gradient
SSE
$(w^{(t)}, \text{SSE}^{(t)})$
$(w^{(t+1)}, \text{SSE}^{(t+1)})$
$(w^{(t+2)}, \text{SSE}^{(t+2)})$
Minimum Error
(Gradient = 0)
Δw
$-w$
$+w$

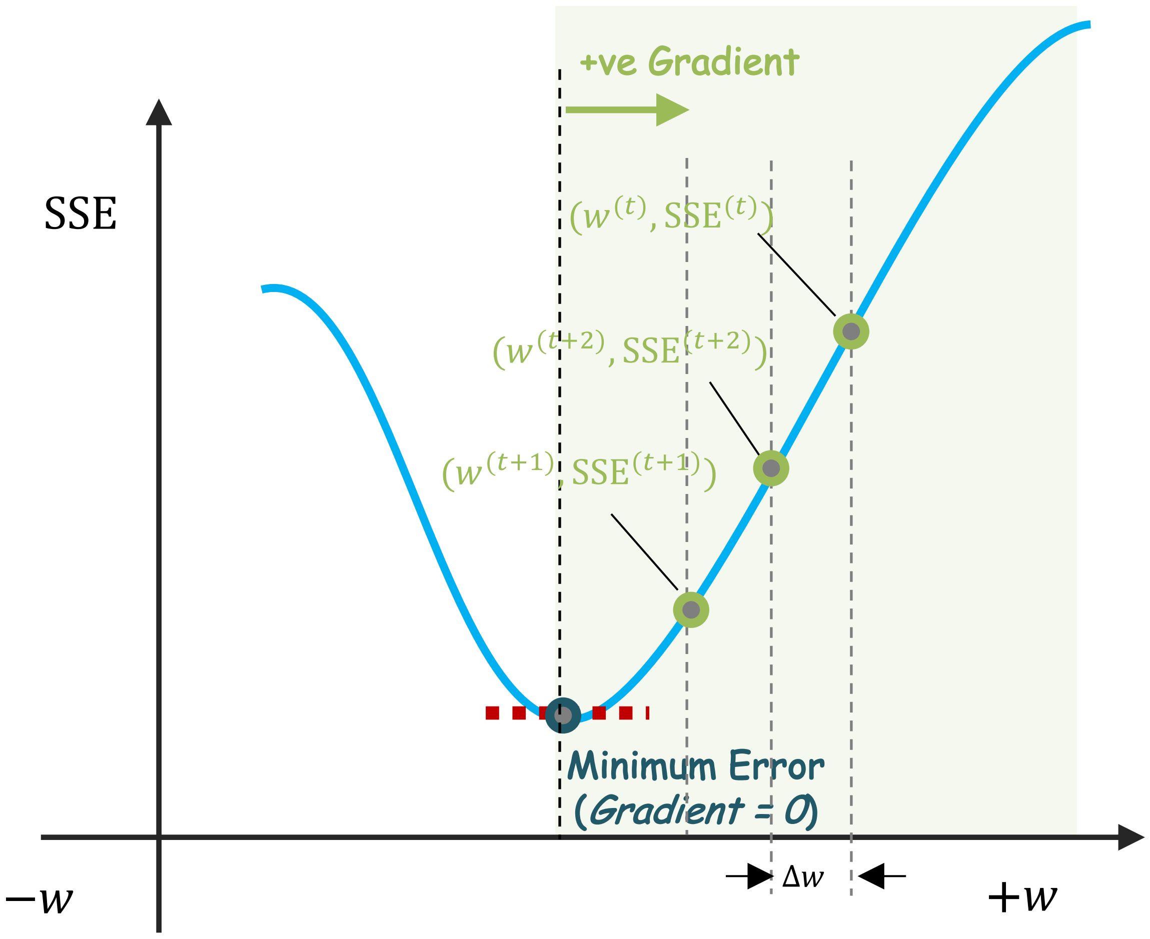
$-\mathbf{ev}$ gradient implies if w *increased* (move in the $+w$ direction) the SSe would also be *decreased*.

If SSE was to *decrease*, then we must move in the $+\ w$ direction.

If SSE was continuing to *decrease*, then eventually we would reach the *minimum* point.

Hence, along as we have $-\mathbf{ev}$ gradient, for each step we are going to keep increasing w, i.e. $w^{(t+1)} = w^{(t)} + \Delta w$.

Gradient Descent: Intuition


+ve Gradient
SSE
$(w^{(t)}, \mathrm{SSE}^{(t)})$
$(w^{(t+2)}, \mathrm{SSE}^{(t+2)})$
$(w^{(t+1)}, \mathrm{SSE}^{(t+1)})$
Minimum Error
(Gradient = 0)
$-w$
Δw
$+w$

$+\mathrm{ev}$ gradient implies if w *increased* (move in $+w$ direction) the SSE would be *decreased.*

If SSE was to *decrease*, then w must be decreasing, i.e. move in the $-w$ direction.

If w continued to decrease, then eventually SSE would reach the *minimum* point.

Hence, along as we have $+\mathrm{ev}$ gradient, for each step we are going to keep *decreasing* w, i.e. $w^{(t+1)} = w^{(t)} - \Delta w$.

Gradient Descent: Intuition

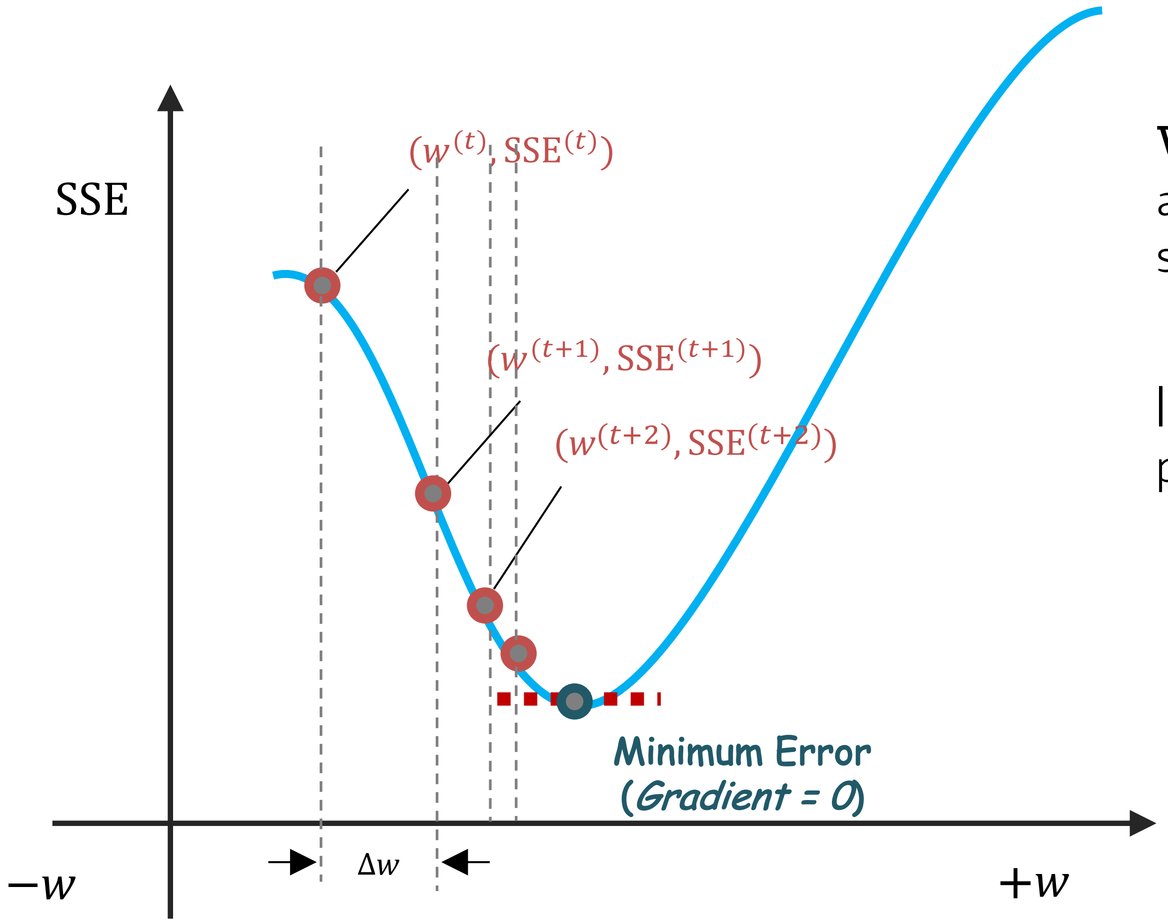
If +ve gradient, then $w^{(t+1)} = w^{(t)} - \Delta w$ else (-ev) $w^{(t+1)} = w^{(t)} + \Delta w$.

$$w^{(t+1)} = w^{(t)} - \mathrm{sign}(\nabla \mathrm{SSE}(w^{(t)})) \times \Delta w$$

gradient

Δw is fixed step. Unless is very small, w will *unlikely* be *very close* at the minimum point.

Gradient Descent: Observation

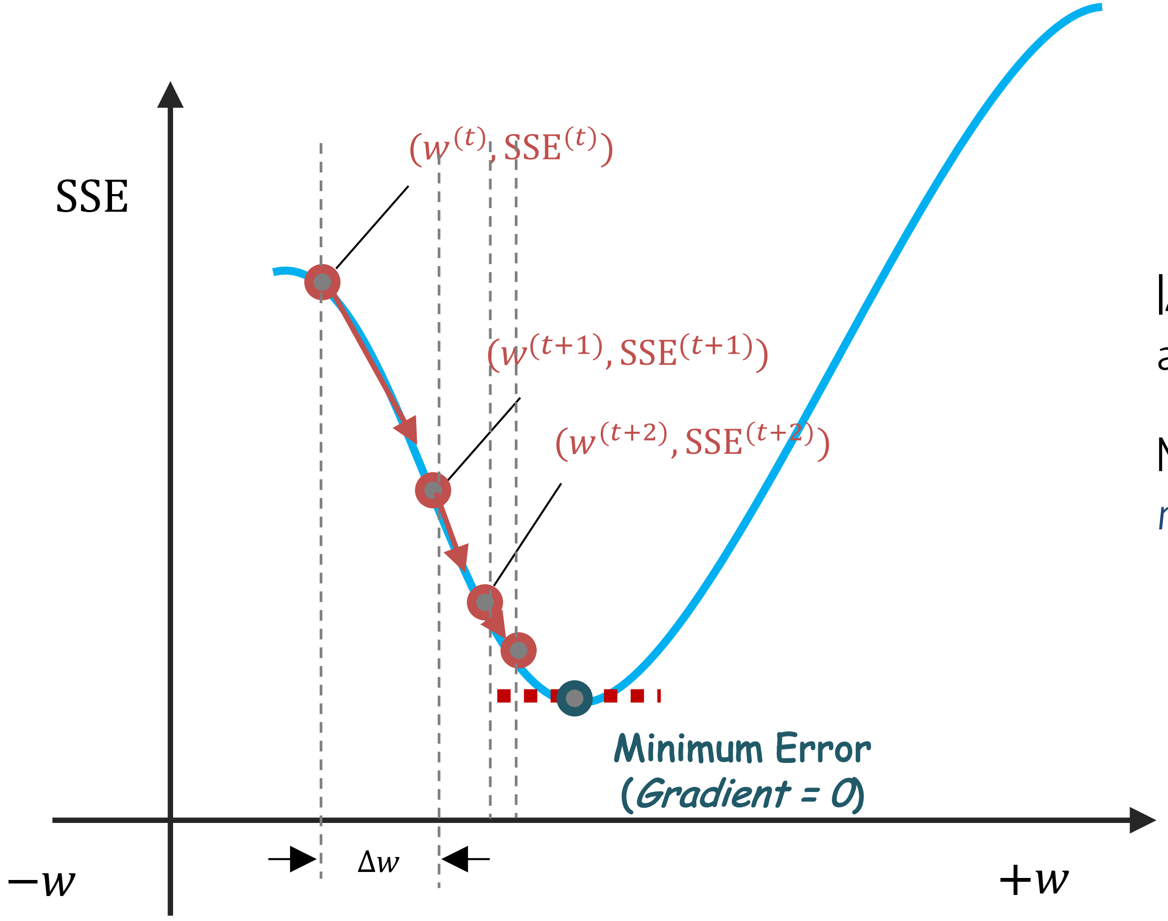

SSE
$(w^{(t)}, \text{SSE}^{(t)})$
$(w^{(t+1)}, \text{SSE}^{(t+1)})$
$(w^{(t+2)}, \text{SSE}^{(t+2)})$
Minimum Error
(*Gradient = 0*)
$-w$
Δw
$+w$

Wish List: We'd like Δw to be large when w is further away from the *minimum* point, and in the opposite *smaller* step when *closer* to the *minimum* point.

$|\nabla \text{SSE}(w)|$ becomes smaller when closer to the minimum point.

$$|\nabla \text{SSE}(w^{(t)})| \geq |\nabla \text{SSe}(w^{(t+1)})| \geq \cdots \geq |\nabla \text{SSE}(w^{(t+N)})|$$

Gradient Descent: Revised


SSE
$(w^{(t)}, \mathrm{SSE}^{(t)})$
$(w^{(t+1)}, \mathrm{SSE}^{(t+1)})$
$(w^{(t+2)}, \mathrm{SSE}^{(t+2)})$
Minimum Error
(Gradient = 0)
$-w$
Δw
$+w$

Learning Rate

$$w^{(t+1)} \leftarrow w^{(t)} - \boxed{\eta} \times \nabla \mathrm{SSE}(w^{(t)})$$

$|\Delta w|$ is *large* when w is further away from the *miininum* point as $|\nabla \mathrm{SSE}(w)|$ large.

Meanwhile, $|\Delta w|$ becomes smaller when w is closer to the *mininum* point as $|\nabla \mathrm{SSE}(w)|$ is small.

Step: $|\Delta w| = |\eta \times \nabla \mathrm{SSE}(w)|$

Sign(∇SSE) → Direction | $|\nabla SSE|$ → *Step Size* | η → *Convergent Time*

Pseudocode for Gradient Descent

```
# Inputs
#  f(w)        ← objective to minimize (w → scalar)
#  grad(w)     ← gradient of f at w  (w → ∇f(w))
#  w0          ← initial parameters
#  n           ← learning rate (constant step size)
#  max_iter    ← maximum number of iterations
#  tol         ← stopping threshold on ‖∇f(w)‖

# ----- optimize -----
w ← w0

FOR t = 1 TO max_iter DO
  g ← grad(w)                  # compute gradient ∇f(w)

  IF norm(g) ≤ tol THEN        # stopping rule (first-order)
     BREAK
  END IF

  w ← w - n · g                # gradient descent update
END FOR

RETURN w
```

Gradient Descent from Scratch

```python
from typing import Callable, Tuple
import numpy as np

def gradient_descent(
    grad: Callable[[np.ndarray], np.ndarray],
    w0: np.ndarray,
    eta: float = 0.05,
    max_iter: int = 1000,
    tol: float = 1e-6,
) -> Tuple[np.ndarray, int, float]:
    w = np.asarray(w0, dtype=float).ravel()
    n_iter = 0

    for t in range(max_iter):
        g = grad(w)
        gn = float(np.linalg.norm(g))
        if gn <= tol:
            n_iter = t
            break
        w = w - eta * g
        n_iter = t + 1
    else:
        # loop exhausted without break → recompute final grad norm
        gn = float(np.linalg.norm(grad(w)))

    return w, n_iter, gn
```

Linear Regression: Gradient Descent as a Minimiser

```
from sklearn.base import BaseEstimator, RegressorMixin
import numpy as np

class MyLinearRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, eta=0.05, max_iter=1000, tol=1e-6):
        self.eta, self.max_iter, self.tol = eta, max_iter, tol

    def fit(self, X, y):
        # Design matrix with bias
        Phi = np.c_[np.ones(X.shape[0]), X]
        N, D = Phi.shape

        # ∇ SSE(w) = 2 · Φᵀ(Φw - y)
        def grad(w):
            r = Phi @ w - y
            return 2.0 * (Phi.T @ r)

        # Run the GD minimiser
        w0 = np.zeros(D)
        w, n_iter, gn = gradient_descent(
            grad, w0, eta=self.eta, max_iter=self.max_iter, tol=self.tol
        )

        self.weights_, self.n_iter_, self.grad_norm_ = w, n_iter, gn
        return self

    def predict(self, X):
        Xs = np.c_[np.ones(X.shape[0]), X]
        return Xs @ self.weights_
```

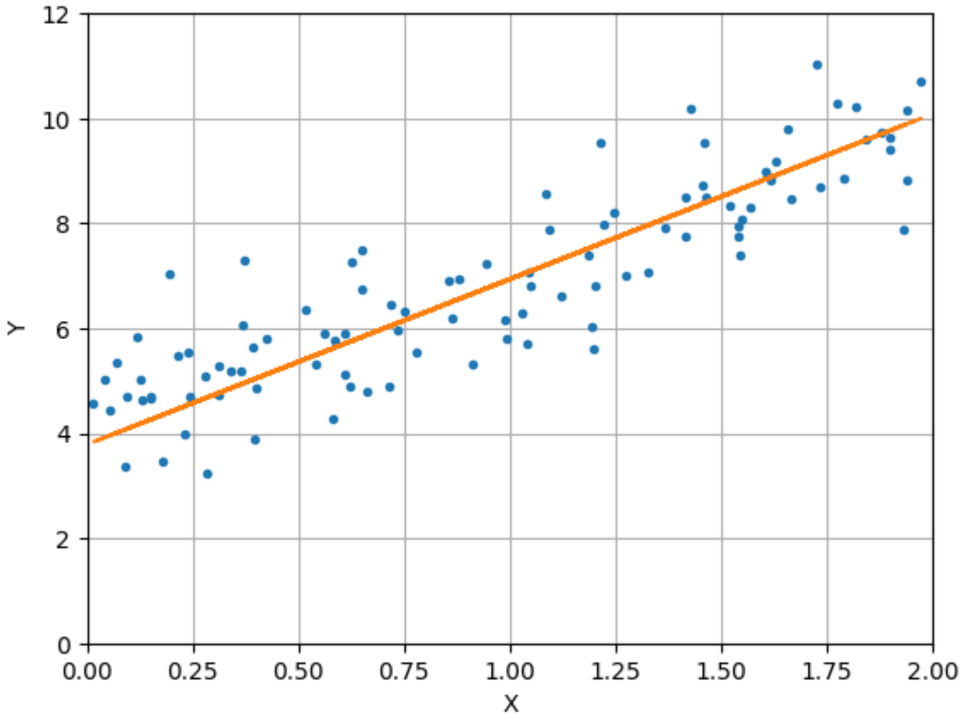
Usage Example

```
# gradient descent related settings
learning_rate  = 0.01
max_iterations = 1000
tolerance      = 1e-6

# train with our sklearn-style class
model = MyLinearRegressor(eta=learning_rate, max_iter=max_iterations, tol=tolerance)
model.fit(X_train, y_train)

# optimised weights [bias, coefficients...]
print("Optimised Weights:", model.weights_)
```

```
Optimised Weights: [3.79008501 3.14520414]
```



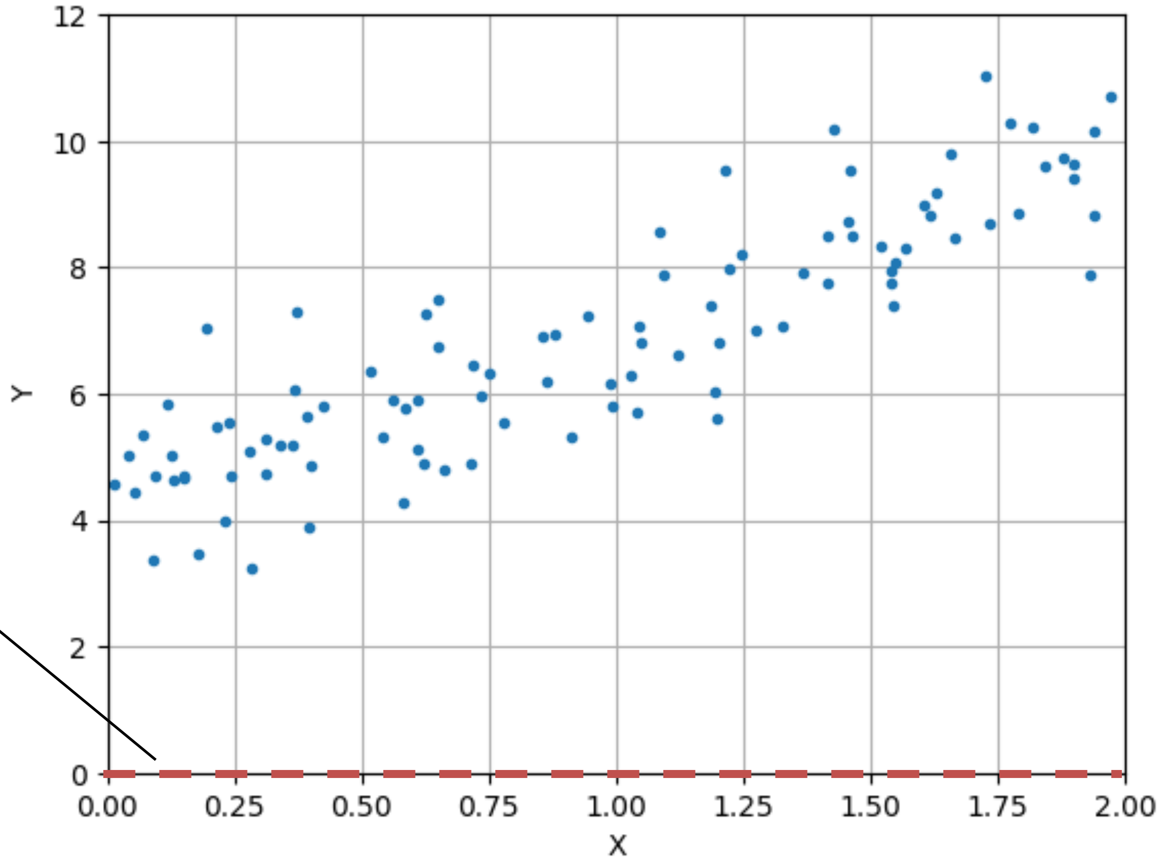
Step 1: Weight Initialisation

```
# Step 1: Initialize weights
w0 = np.zeros(D)
```

```
w, n_iter, gn = gradient_descent(
    grad, w0, eta=self.eta, max_iter=self.max_iter,
    tol=self.tol, callback=self.callback
)
```

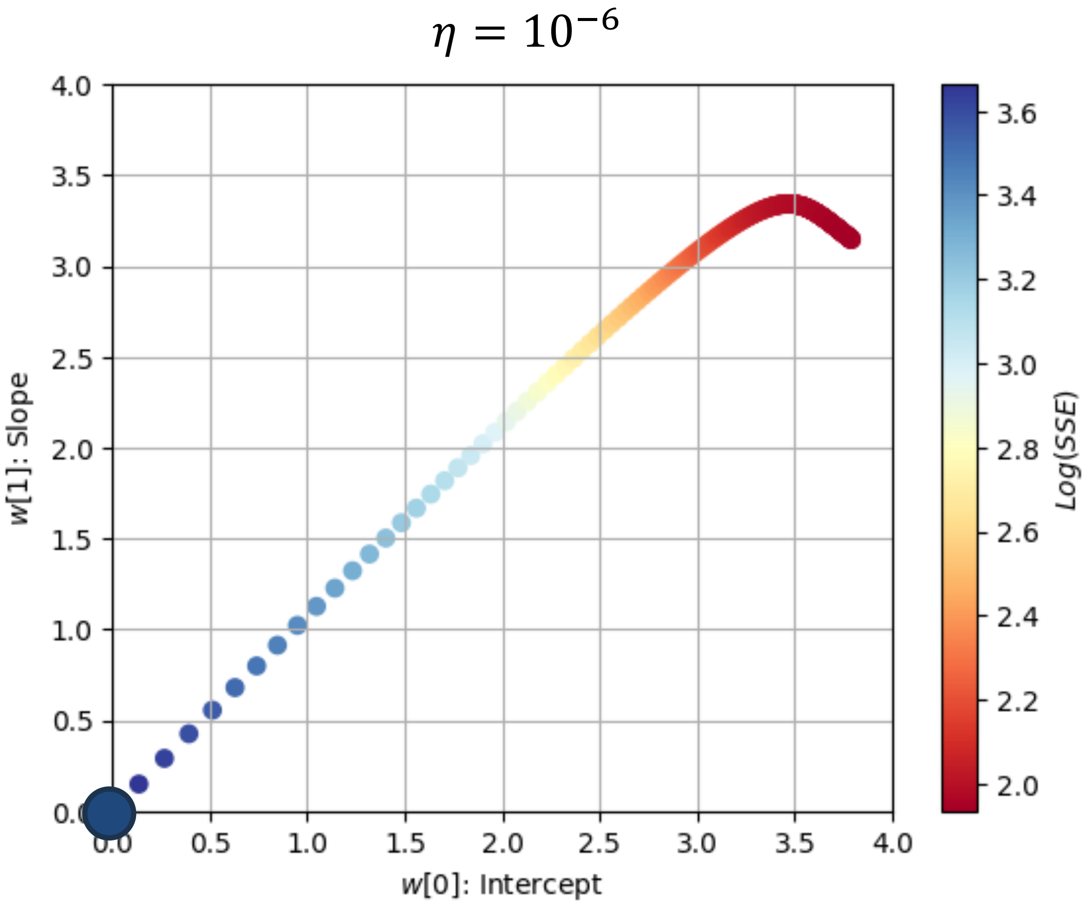
$$\vec{w} = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

$$\begin{aligned} \hat{y} &= 0 \cdot x + 0 \\ &= 0 \end{aligned}$$



Step 2: Gradient Descent Loop

```
# Step 2: Start the optimisation loop
for t in range(max_iter):
    g = grad(w)
    gn = float(np.linalg.norm(g))
    if gn ⩽ tol:
        n_iter = t
        break
    w = w - eta * g
    n_iter = t + 1
    if callback is not None:
        callback(t, w, g, gn)
else:
    # if not broken, recompute gn for reporting
    gn = float(np.linalg.norm(grad(w)))
```

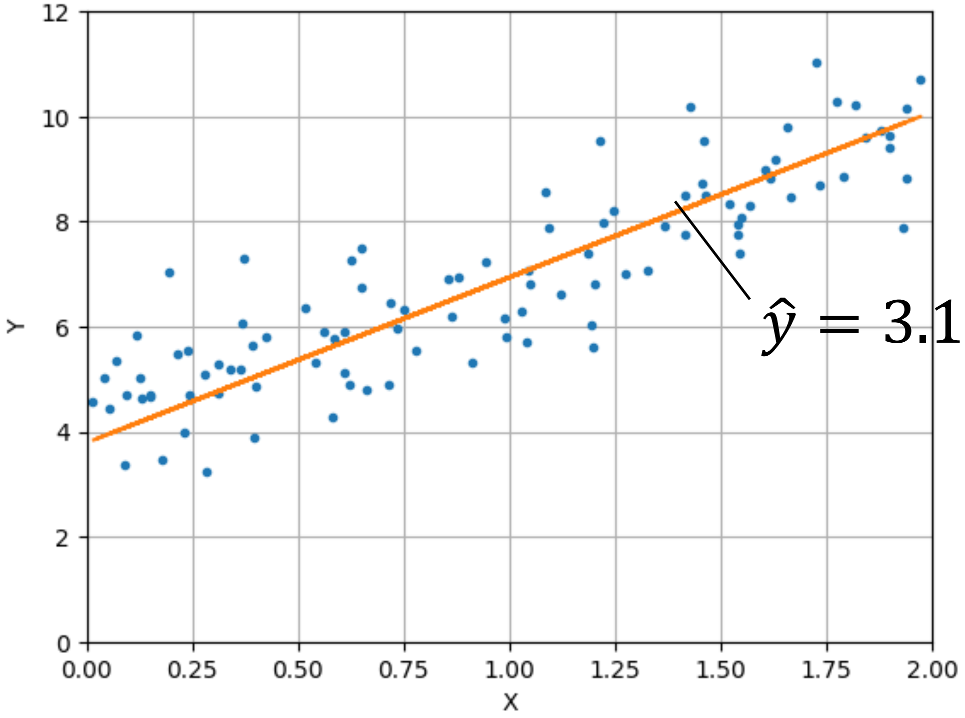


Step 3: Return Optimised Weights

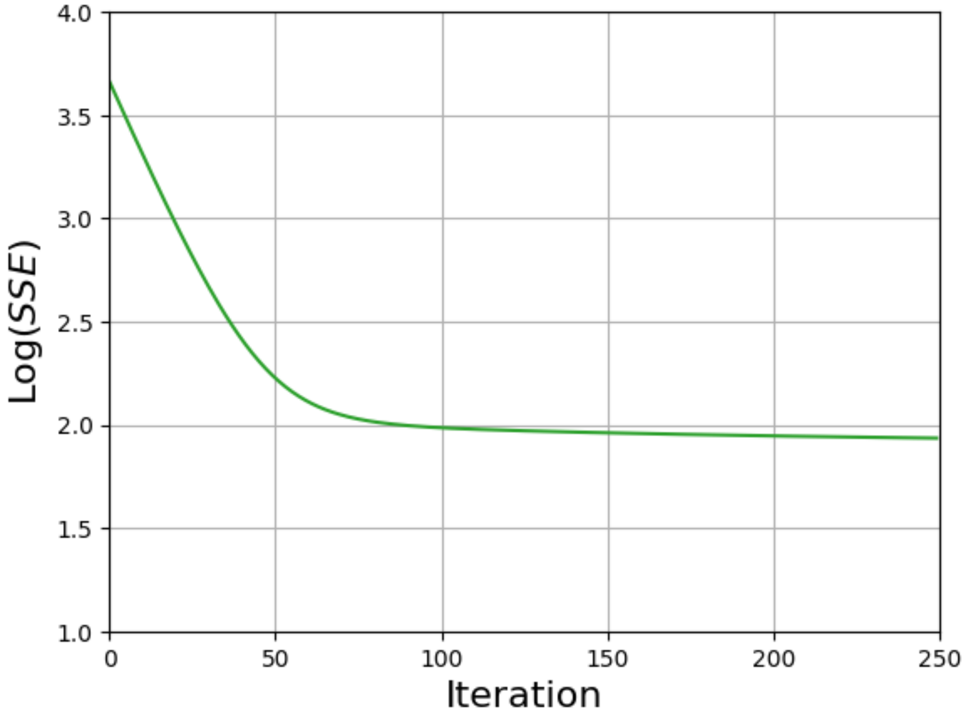
```
# Step 3: Return the optimized weights
return w, n_iter, gn
```

```
# optimised weights [bias, coefficients...]
print("Optimised Weights:", model.weights_)
```

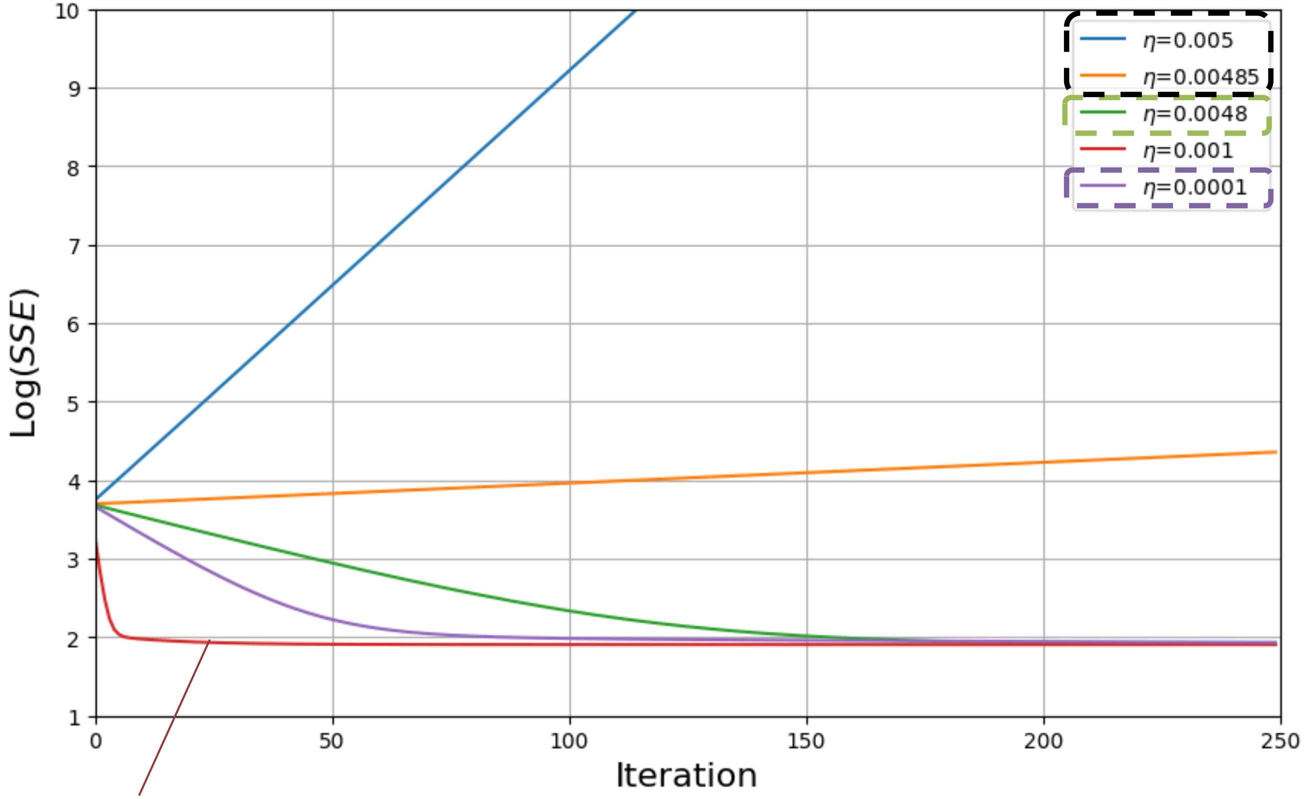
```
Optimised Weights: [3.79008501 3.14520414]
```



$$\hat{y} = 3.15x + 3.79$$



Learning Rate



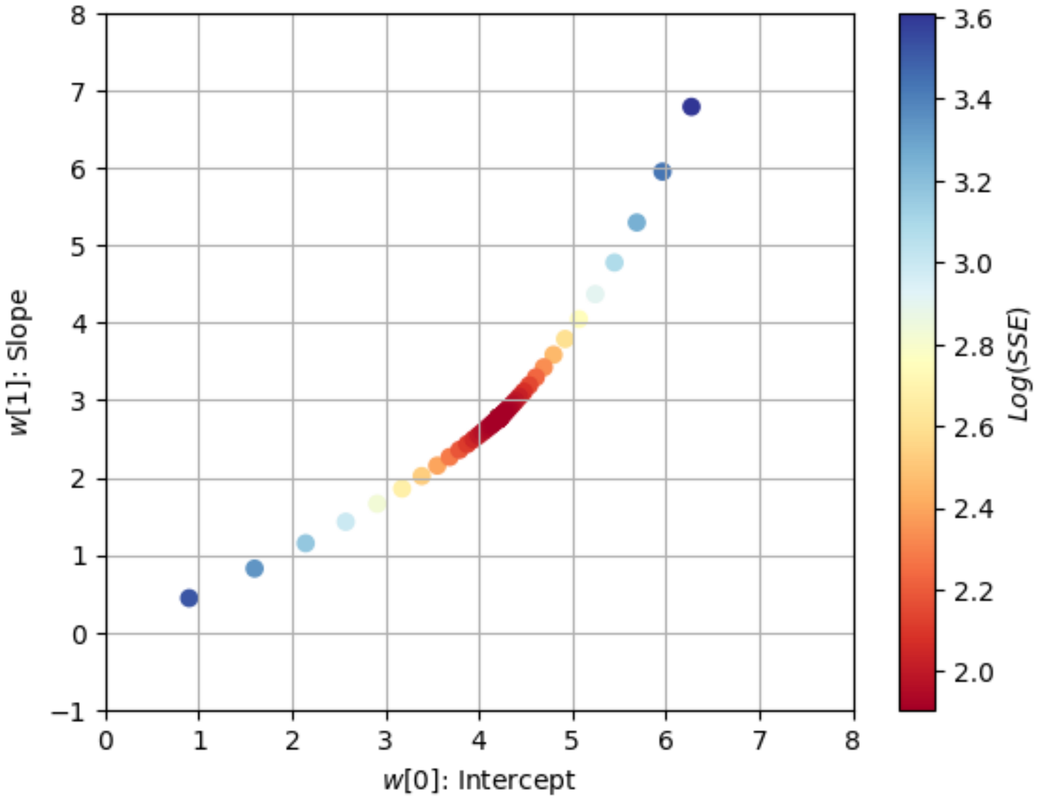
...just right...

- Low learning rate leads to slow convergence, which will require iterations.
- High learning rate can also lead to slow convergence, which is caused by oscillations.
- Very high learning rate will cause gradient descent not to converge.

If things go wrong, then just lower the learning rate.

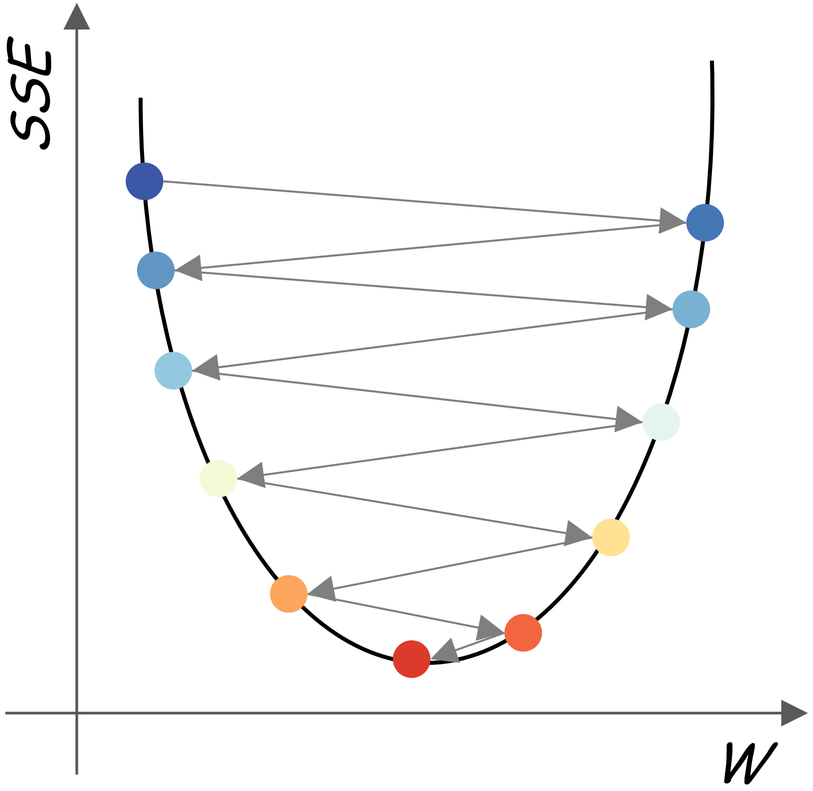
High Learning Rate

$$\eta = 4.6\times10^{-3}$$

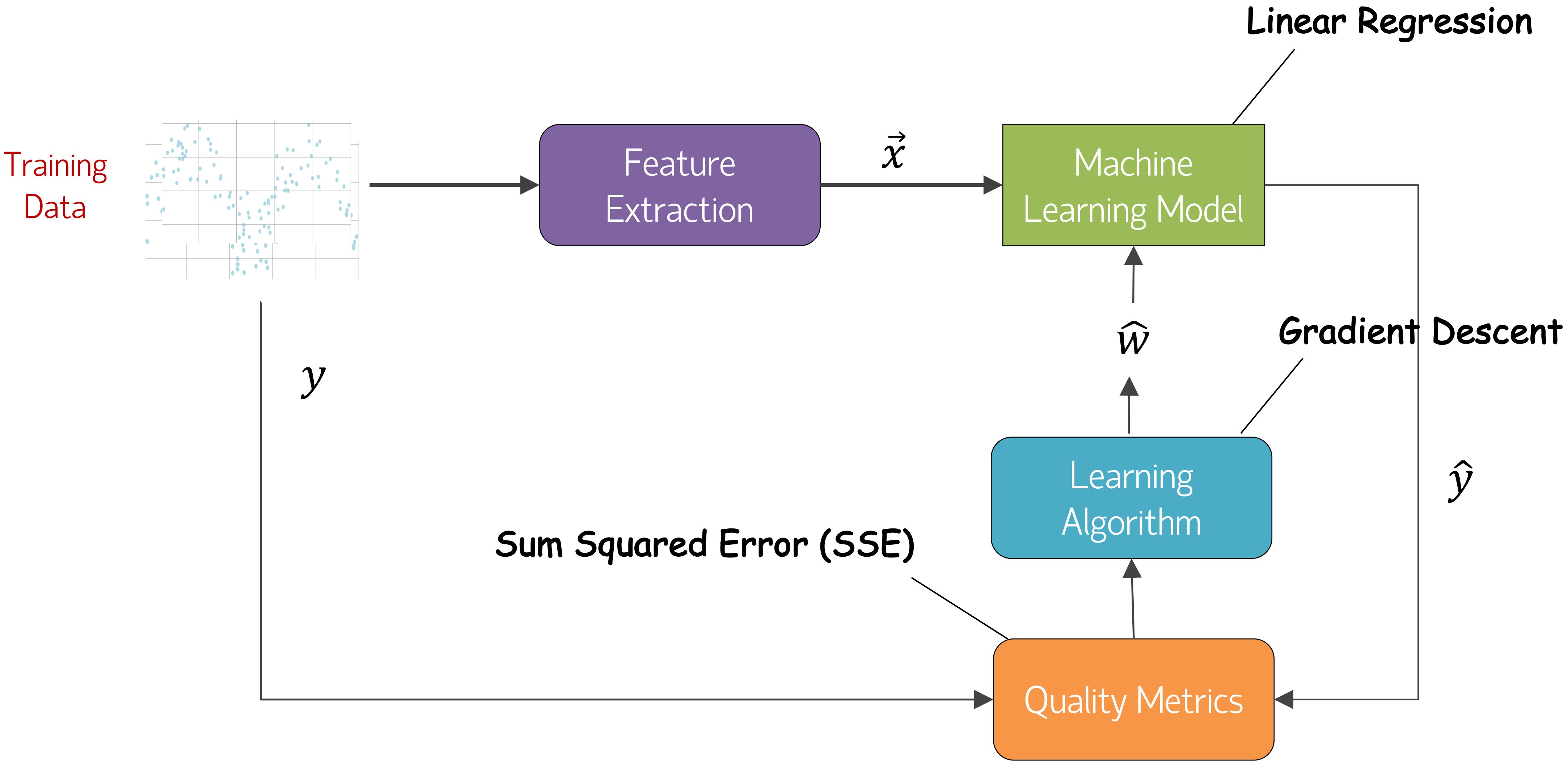


Q: It looks like there are 2 trajectories, i.e. converges from bottom left and from top right. What happened?

A: …Oscillation…



Workflow: Linear Regression



Summary

- Gradient Descent is an iterative optimization technique: It is widely used to minimize error functions by iteratively updating parameters in the direction of the negative gradient, aiming to reach the point where the function has the lowest error.

- Learning Rate determines the step size: The learning rate controls how much the parameters are adjusted with each iteration. Choosing an appropriate learning rate is crucial, as a high rate might cause overshooting, while a low rate could slow down convergence.

- Stopping Criteria ensure convergence: Gradient Descent uses stopping criteria to terminate the process once changes in the error fall below a specified threshold or a set number of iterations is reached, indicating that a minimum is close.

- Gradient Descent is in fact a the learning algorithm that iteratively adjusts model parameters to minimize error. Rather than being unique to any single model type (e.g., linear regression or neural networks), Gradient Descent is a general optimization technique used across various models to find the best parameter values that reduce the error metric.