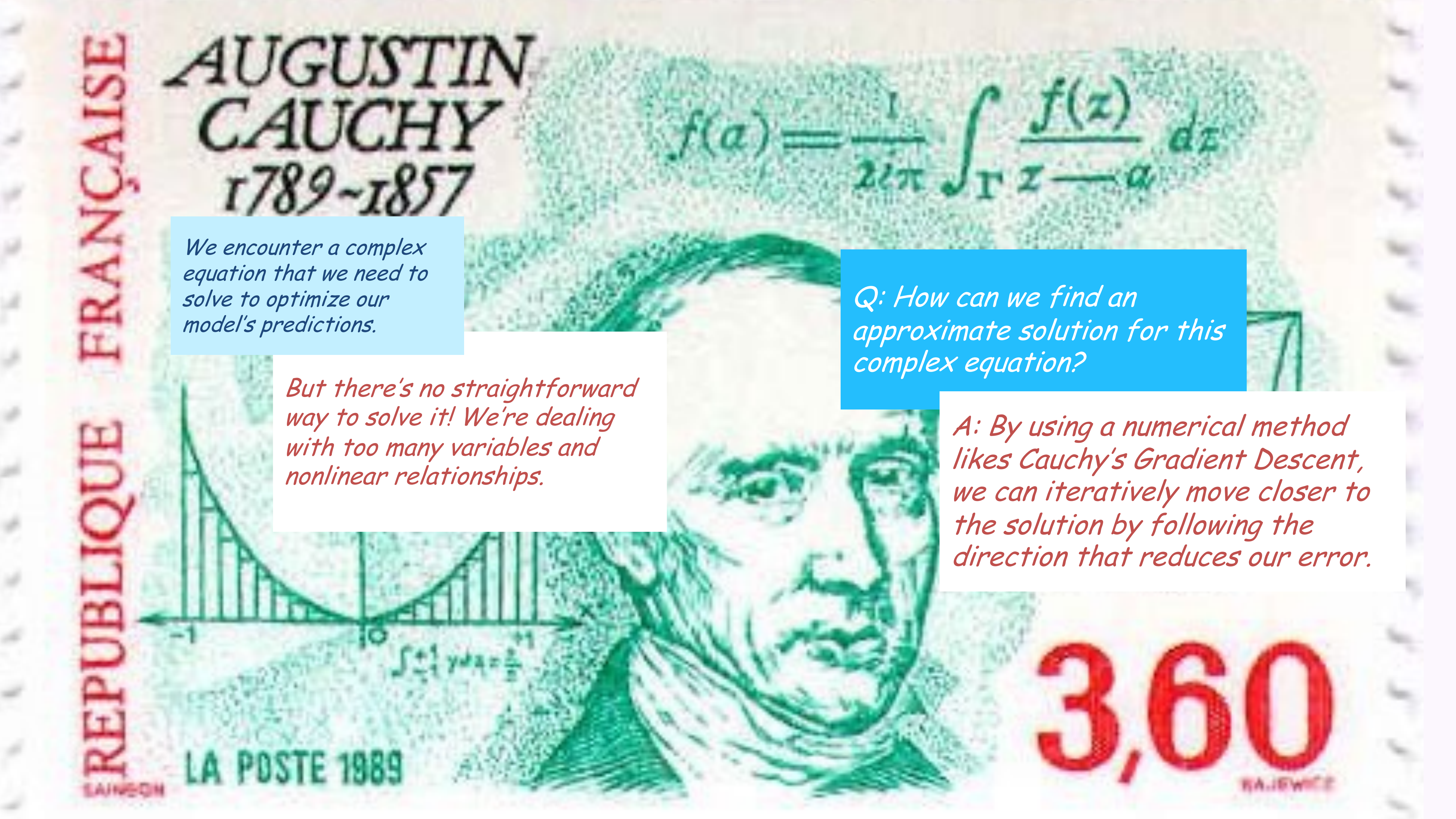
Machine Learning

Gradient Descent

Tarapong Sreenuch

8 February 2024

克明峻德，格物致知



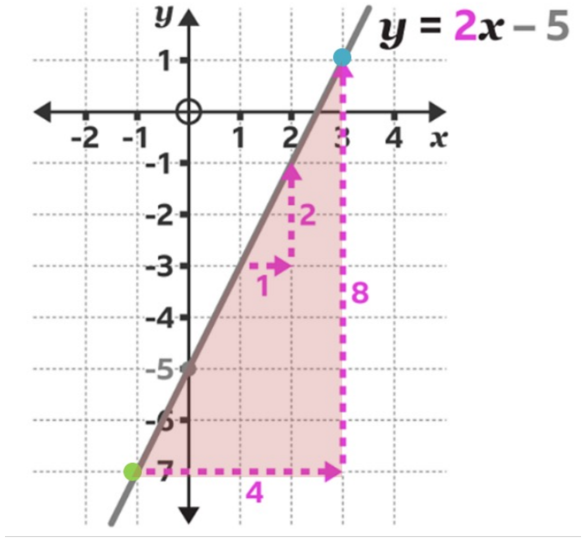
We encounter a complex equation that we need to solve to optimize our model's predictions.

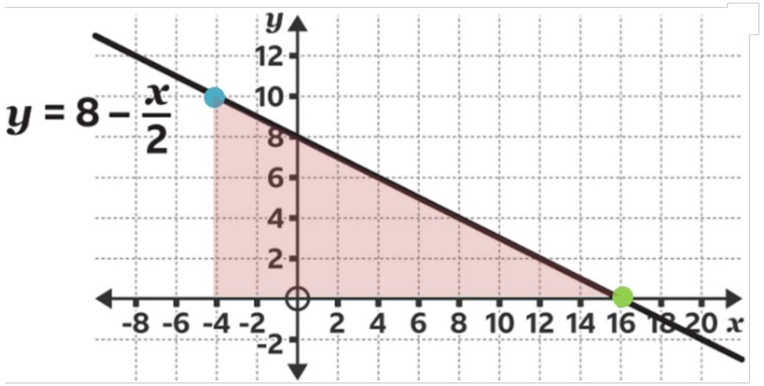
But there's no straightforward way to solve it! We're dealing with too many variables and nonlinear relationships.

Q: How can we find an approximate solution for this complex equation?

A: By using a numerical method likes Cauchy's Gradient Descent, we can iteratively move closer to the solution by following the direction that reduces our error.

Recap: Gradients

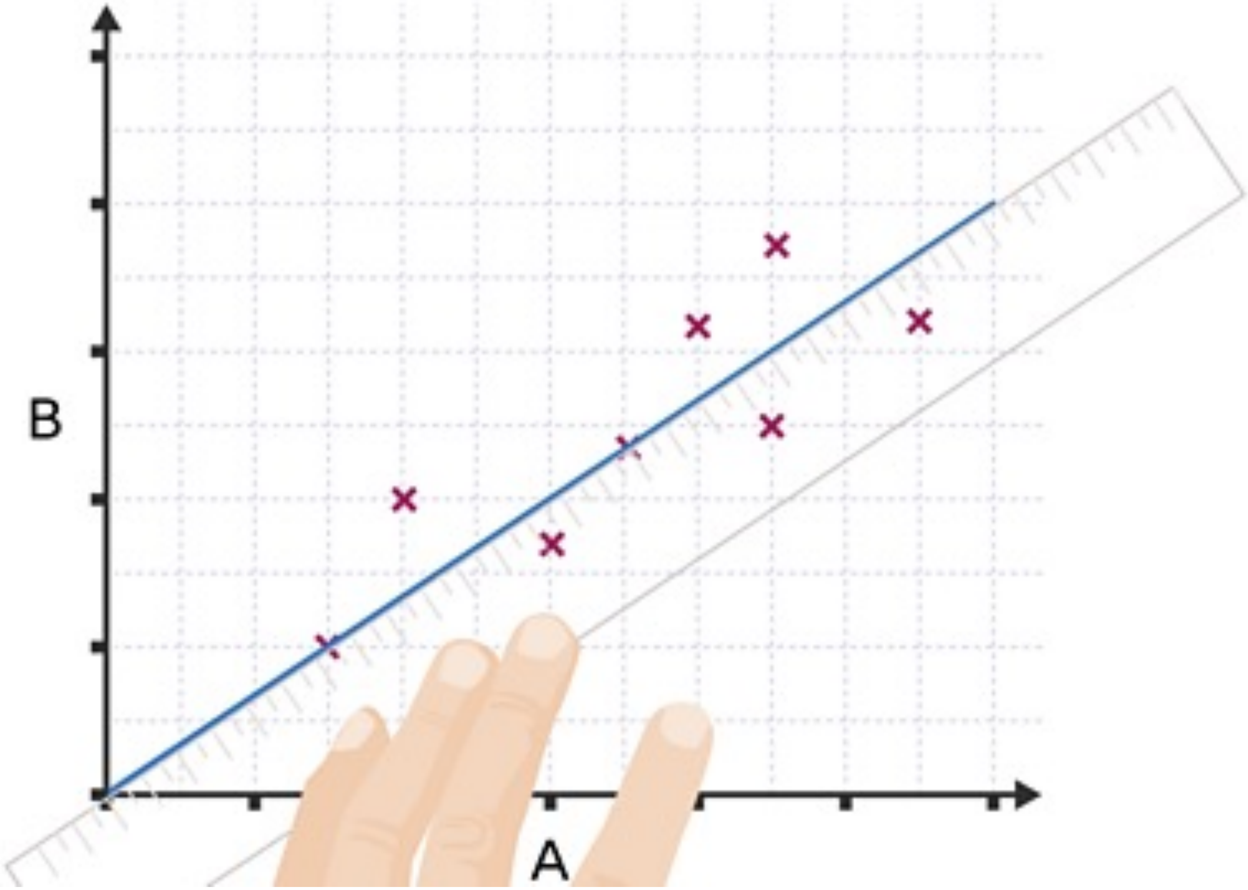




x ⬆ y ⬆ ➡ ⊕ gradient

x ⬆ y ⬇ ➡ ⊖ gradient

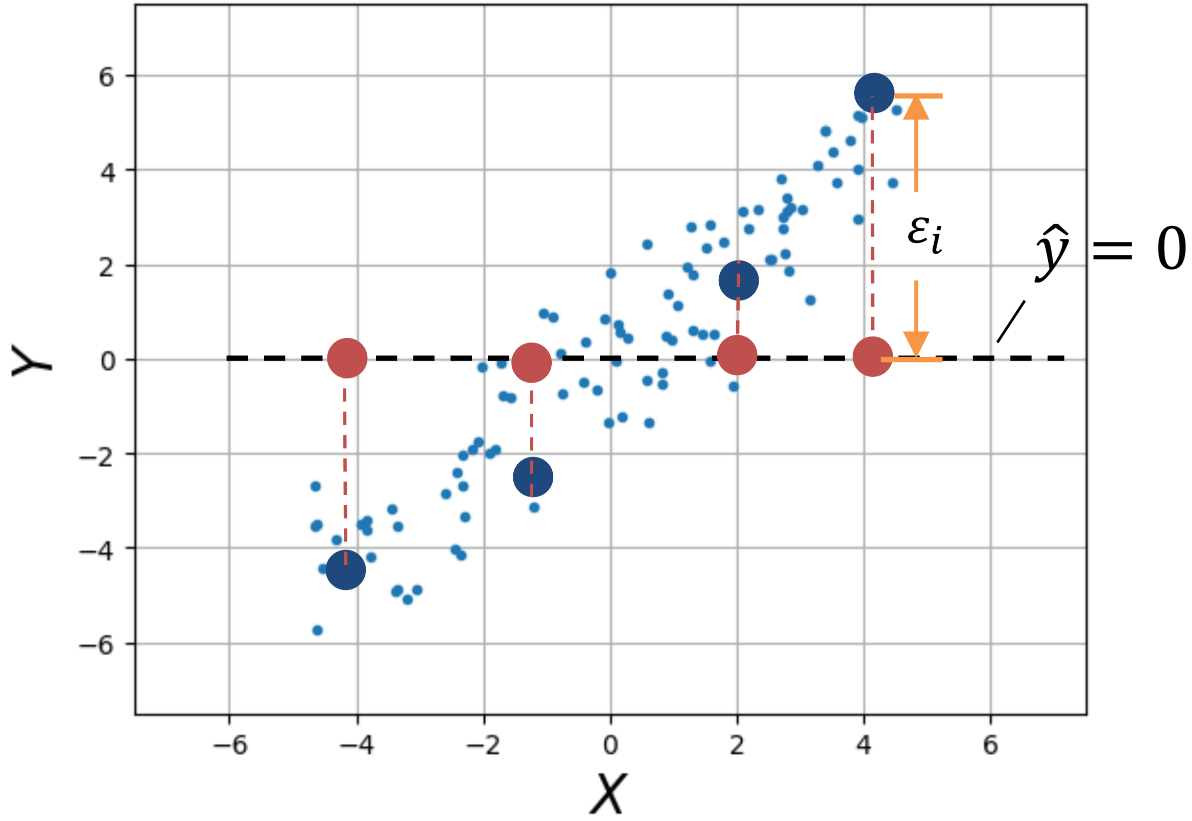
Drawing a Line of Best Fit

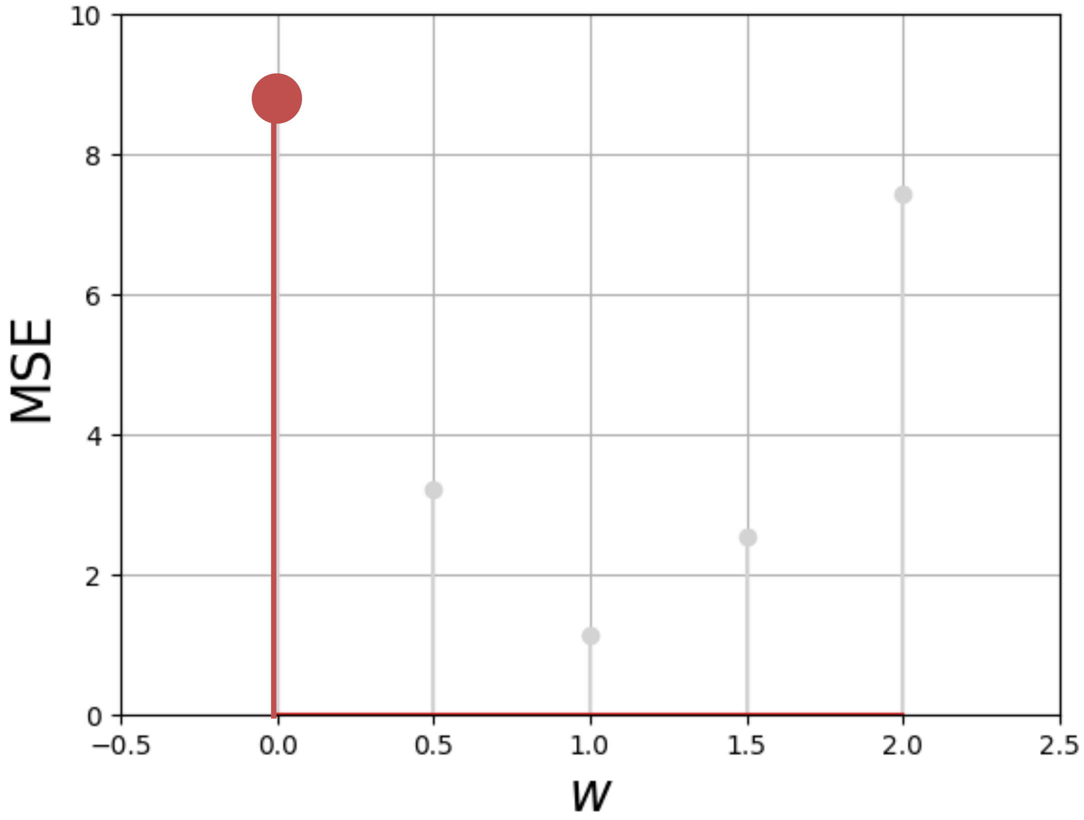


Q: How do we find a line of best fit?

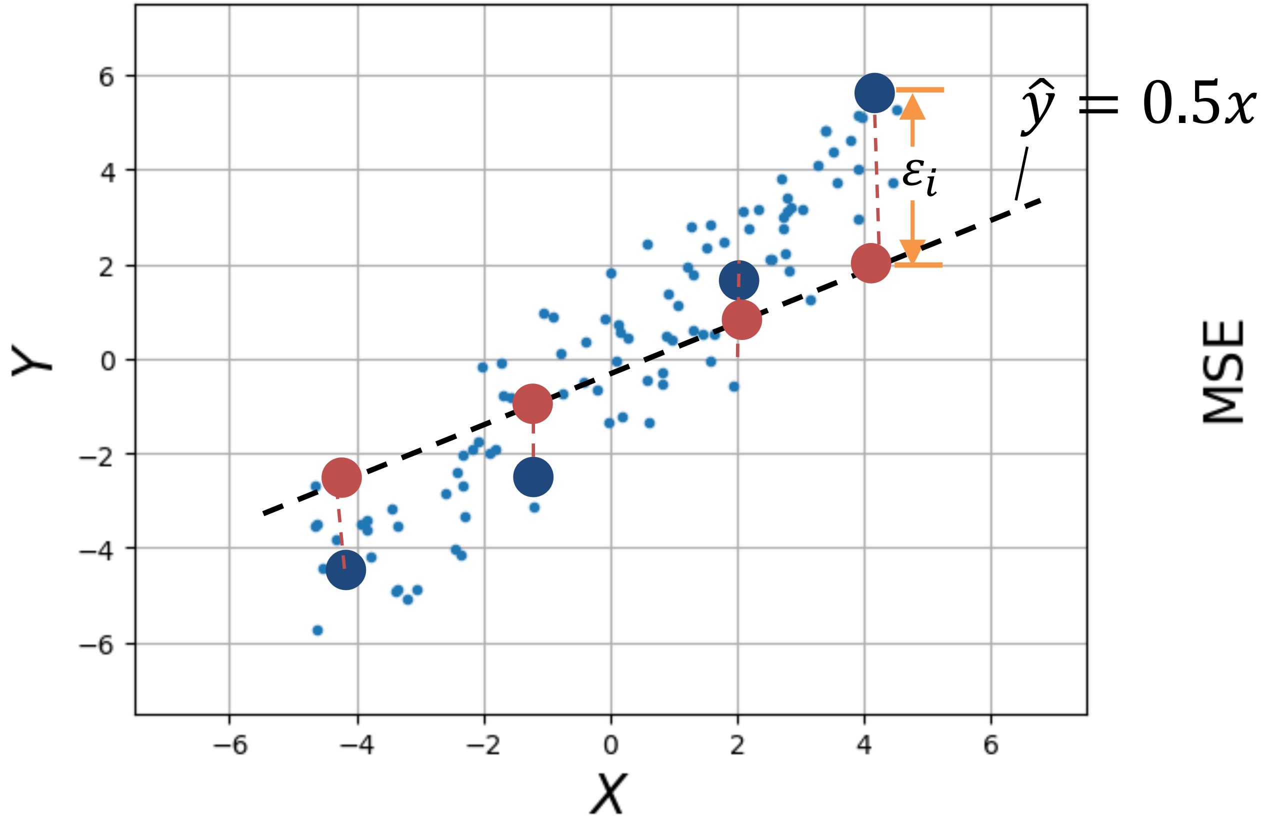
A: By rotating (clockwise/anti-clockwise) and/or shifting (up/down) the ruler, we find a line that goes roughly through the middle of all the middele of all the scatter plots.

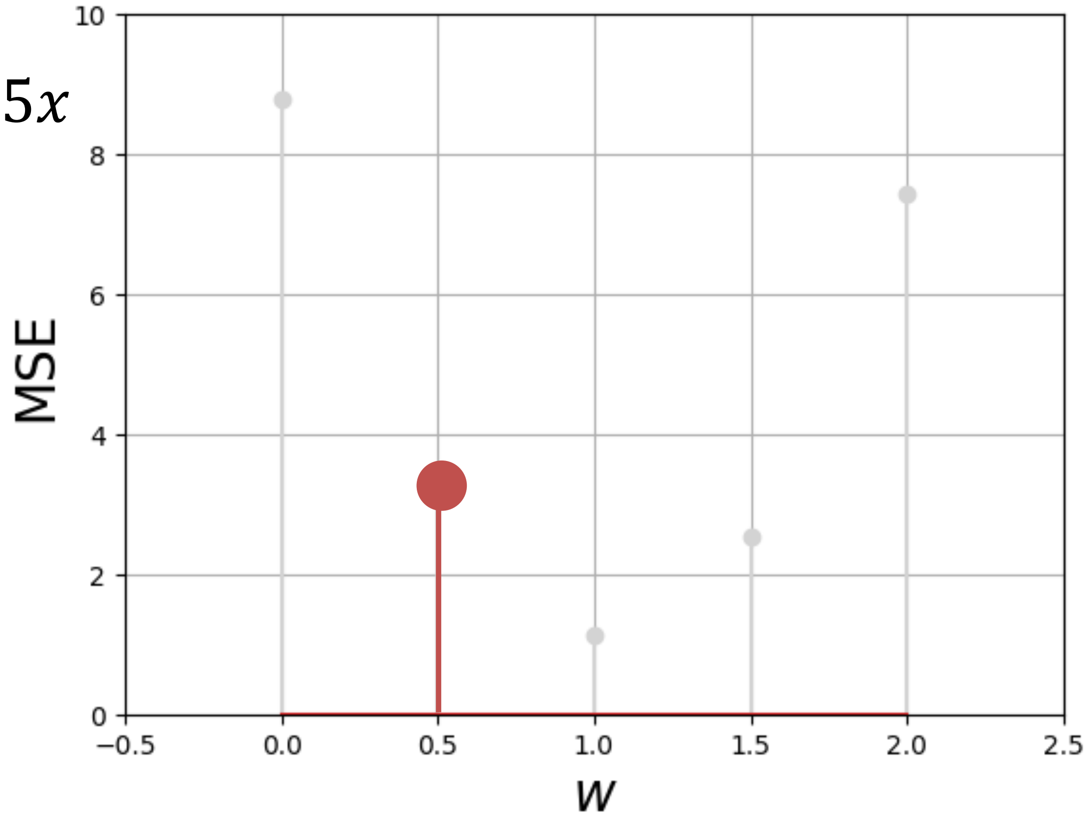
Forming a Linear Model



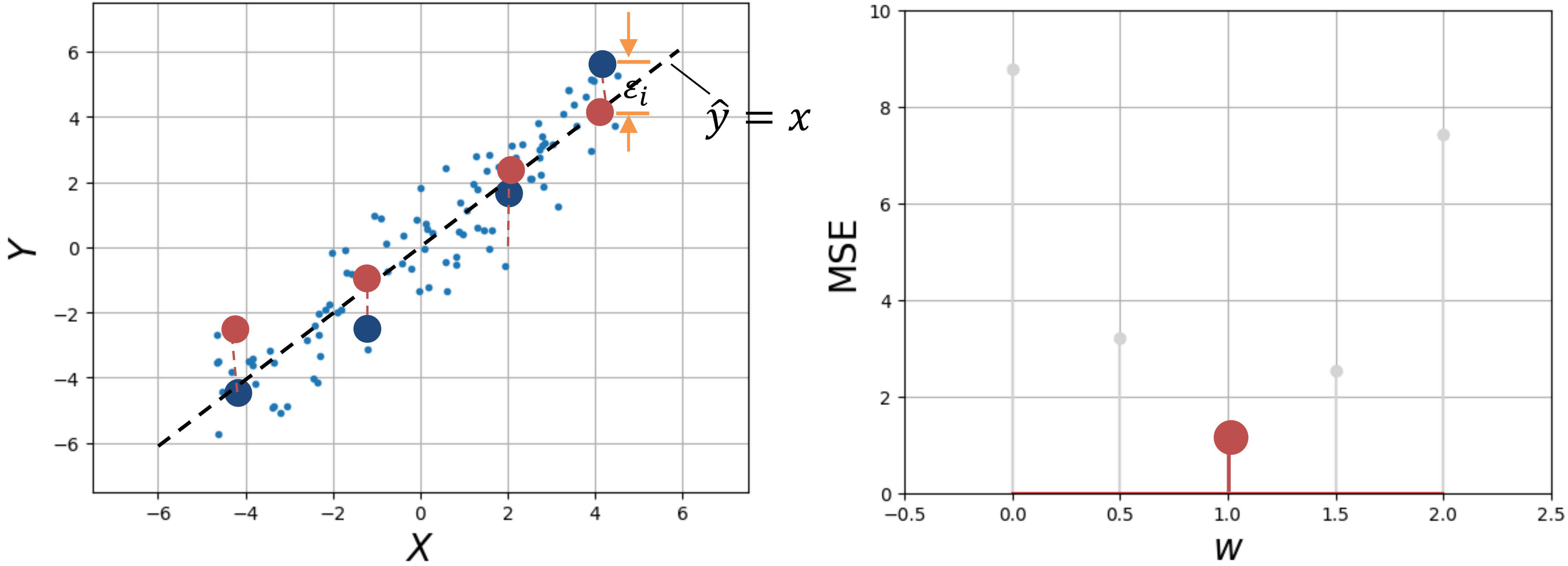


Forming a Linear Model

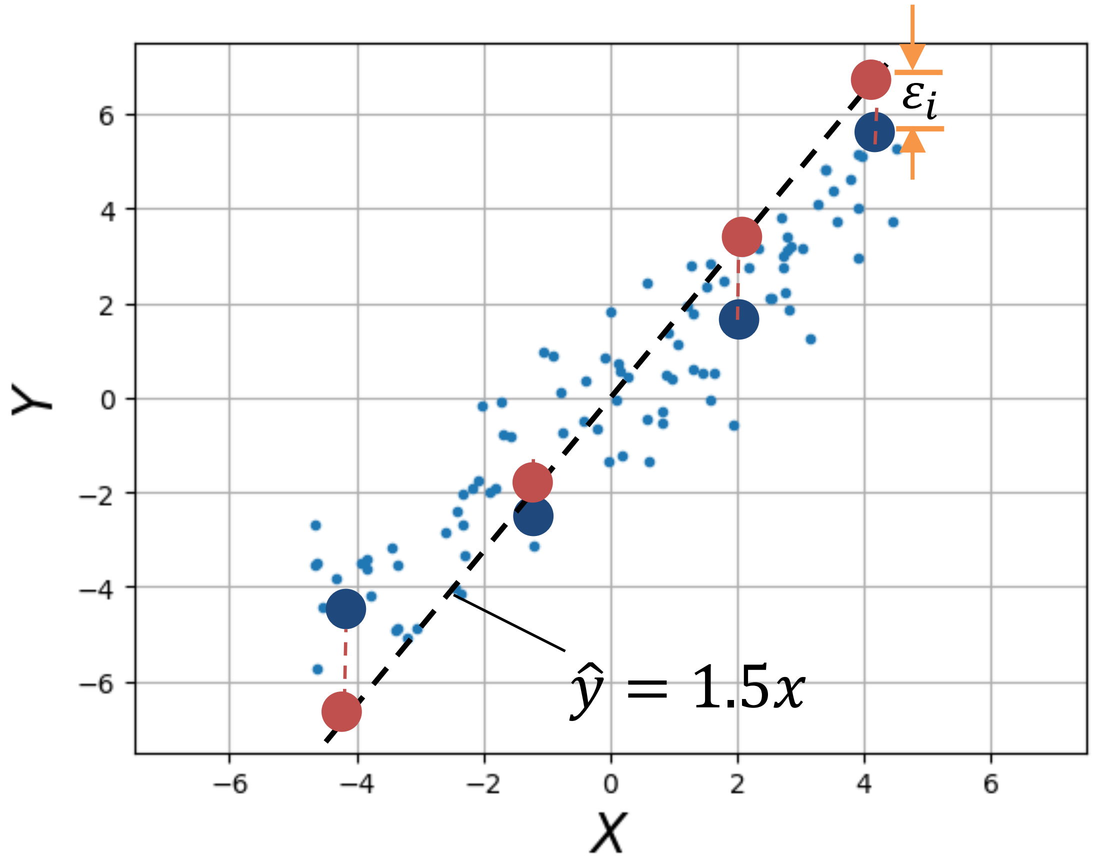


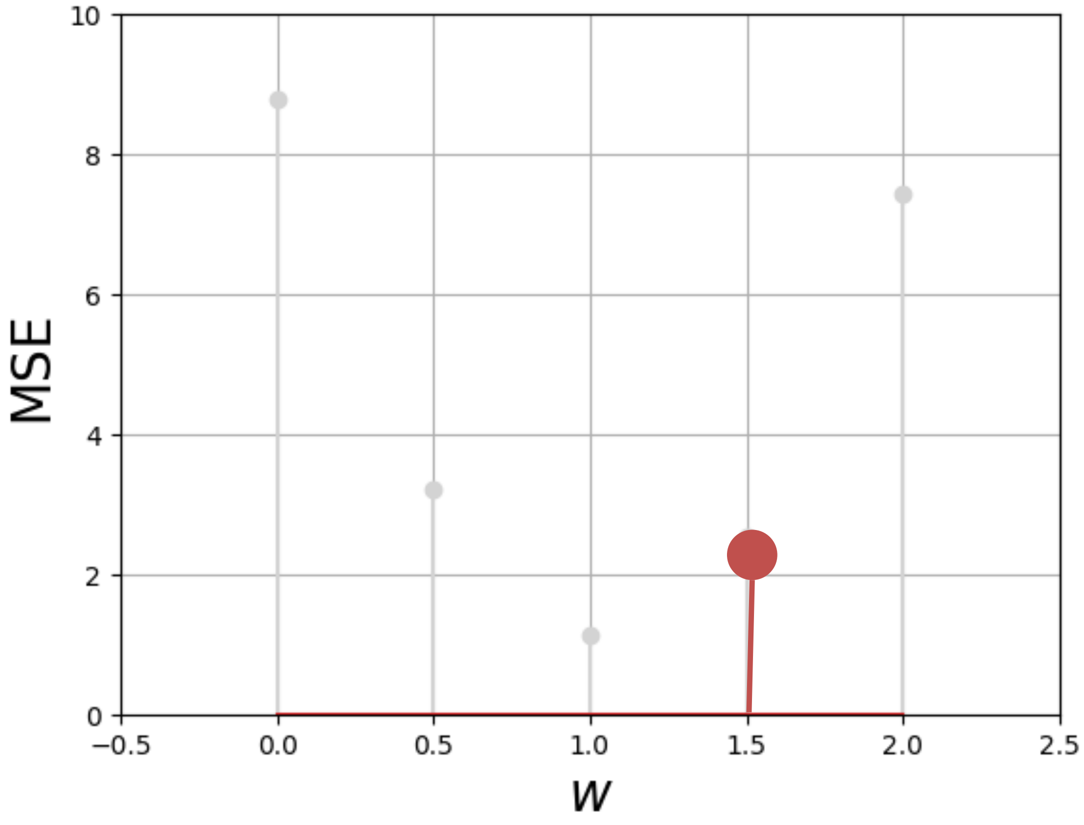


Forming a Linear Model

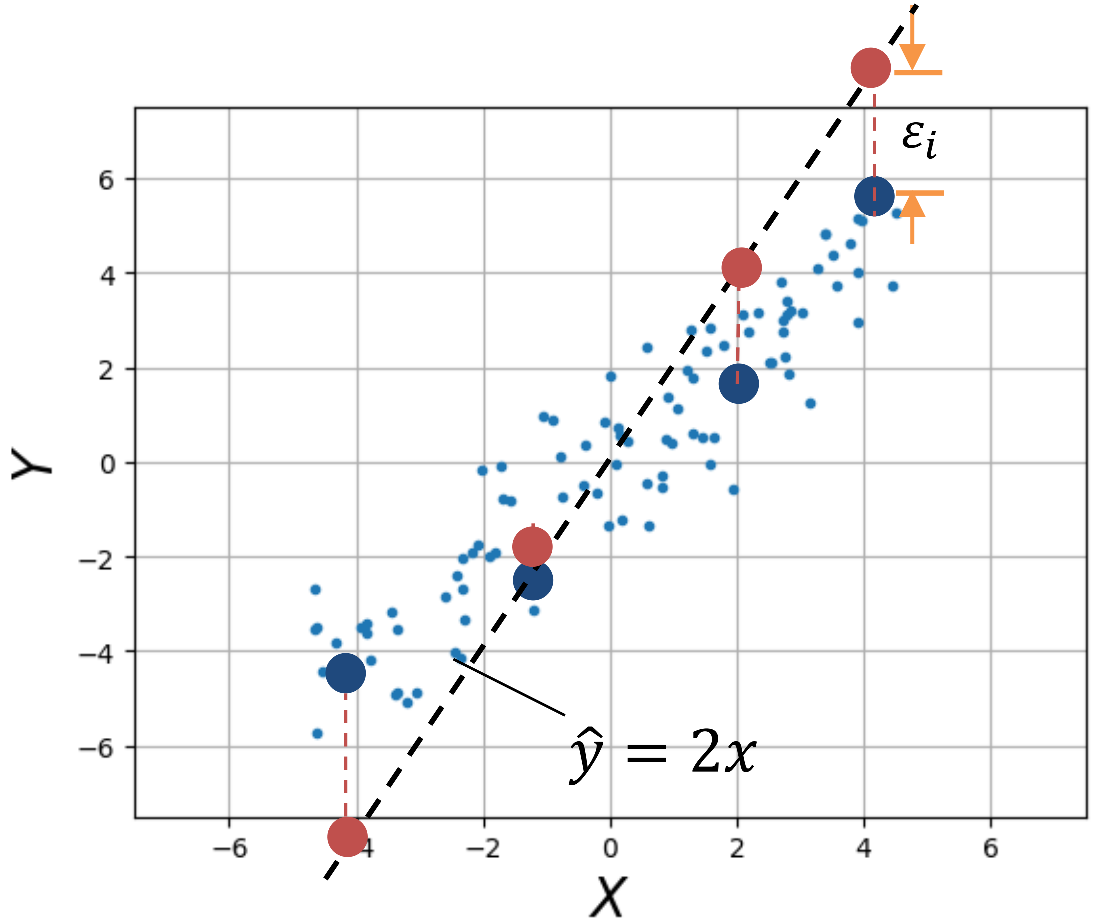


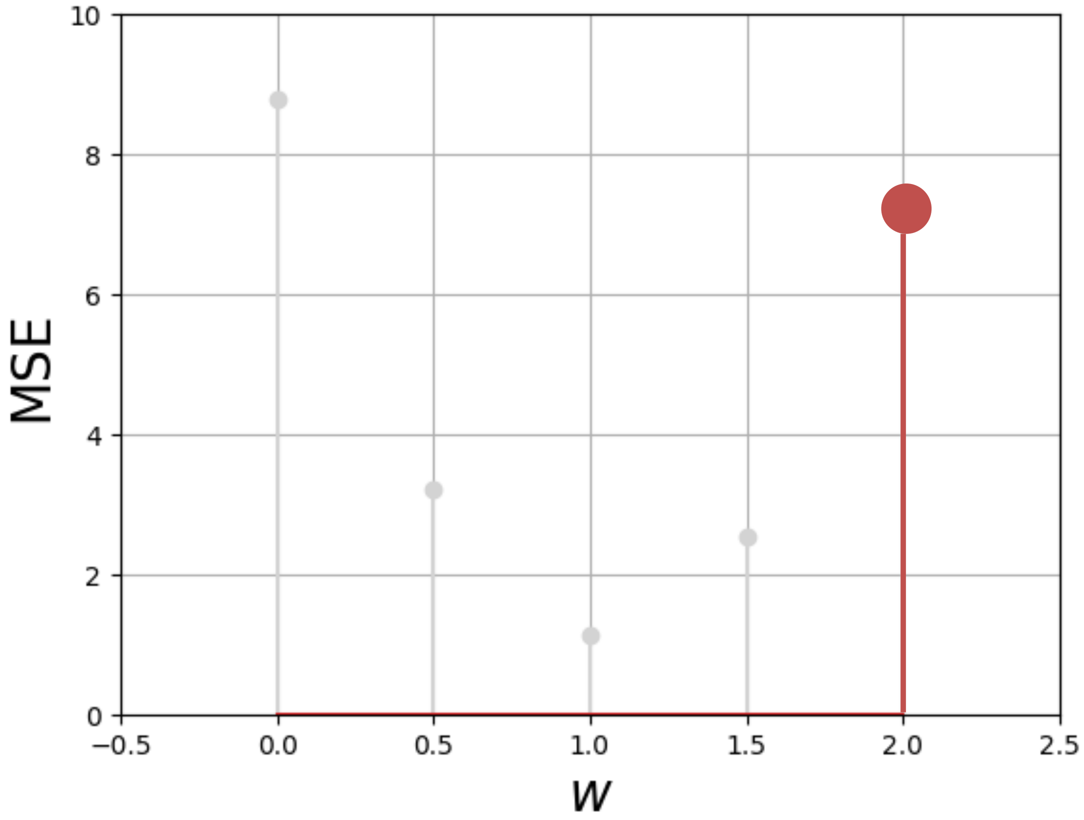
Forming a Linear Model



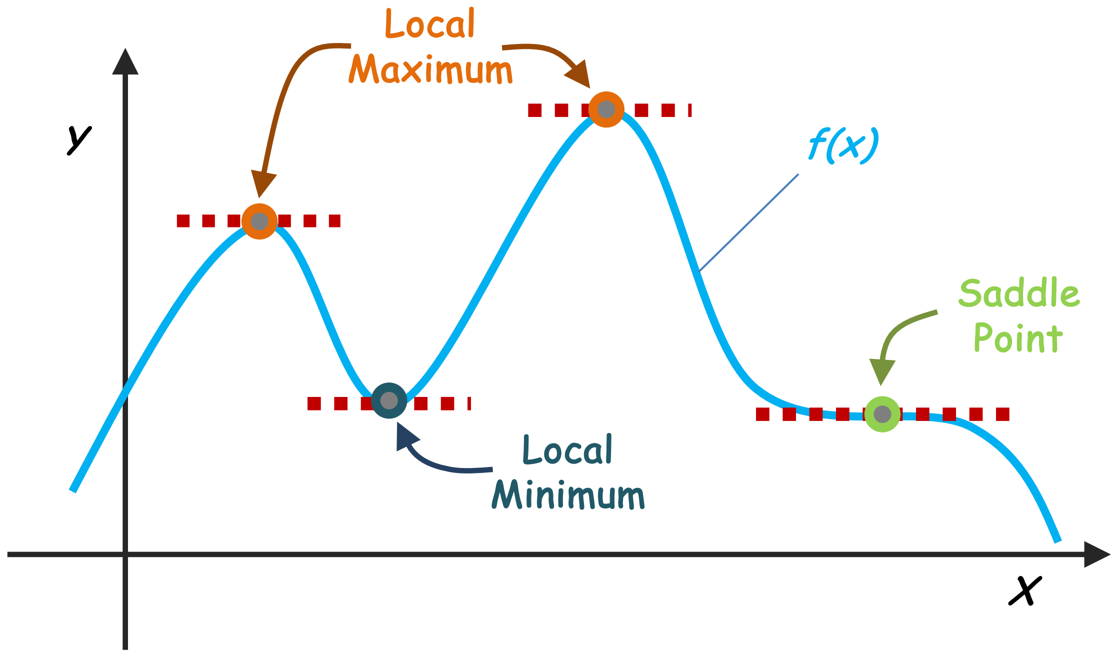


Forming a Linear Model





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):

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

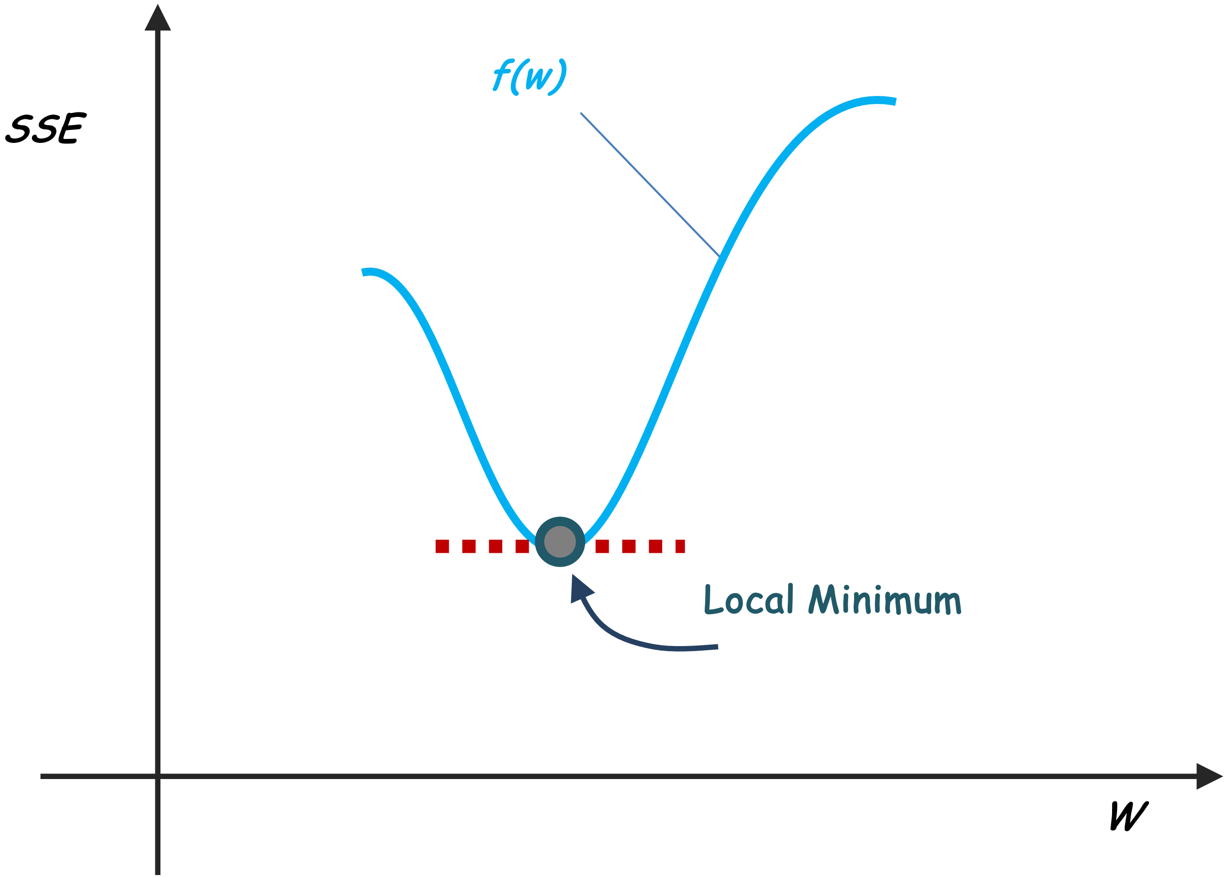
SSE Derivative:

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

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

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

A: … Slope (or Gradient) = 0 …



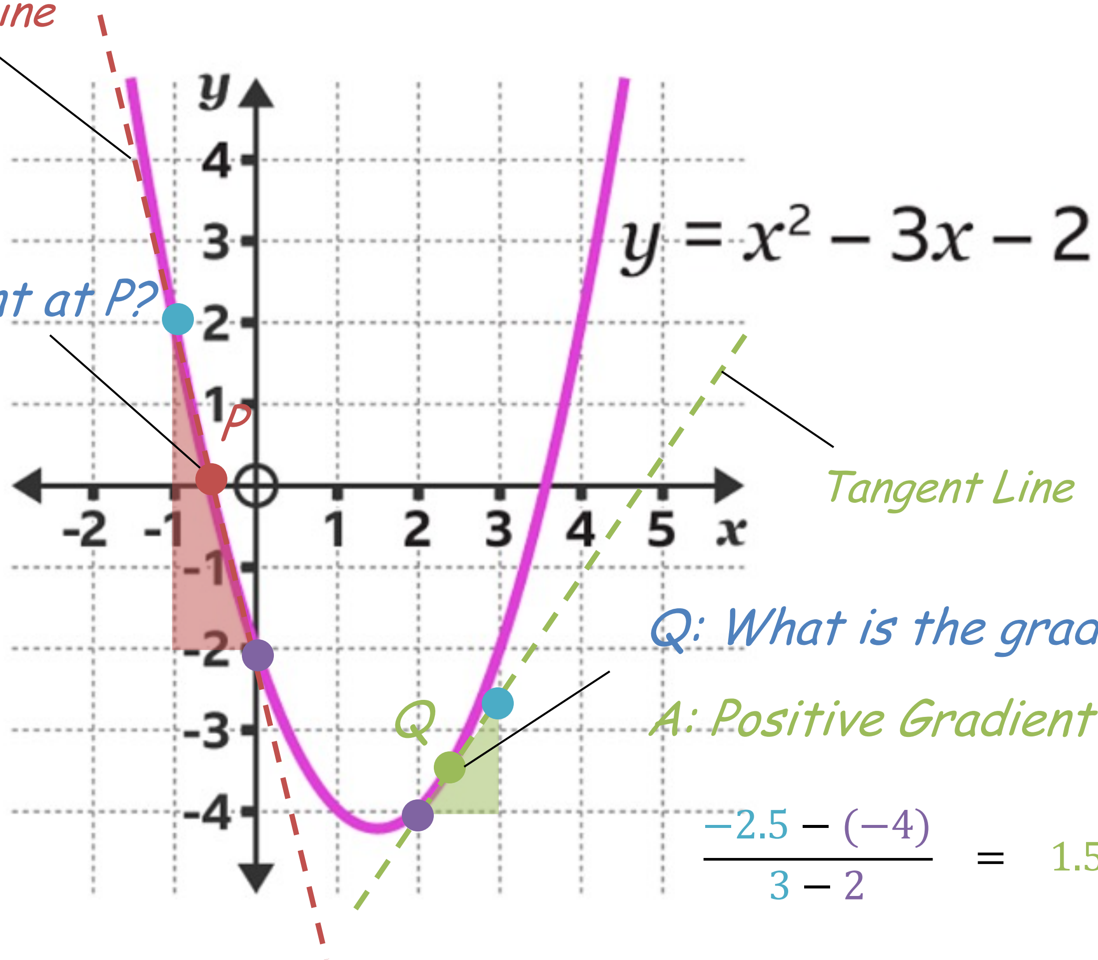
Non-Linear Equation: Gradients

Tangent Line

Q: What is the gradient at P?

A: Negative Gradient

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



Tangent Line

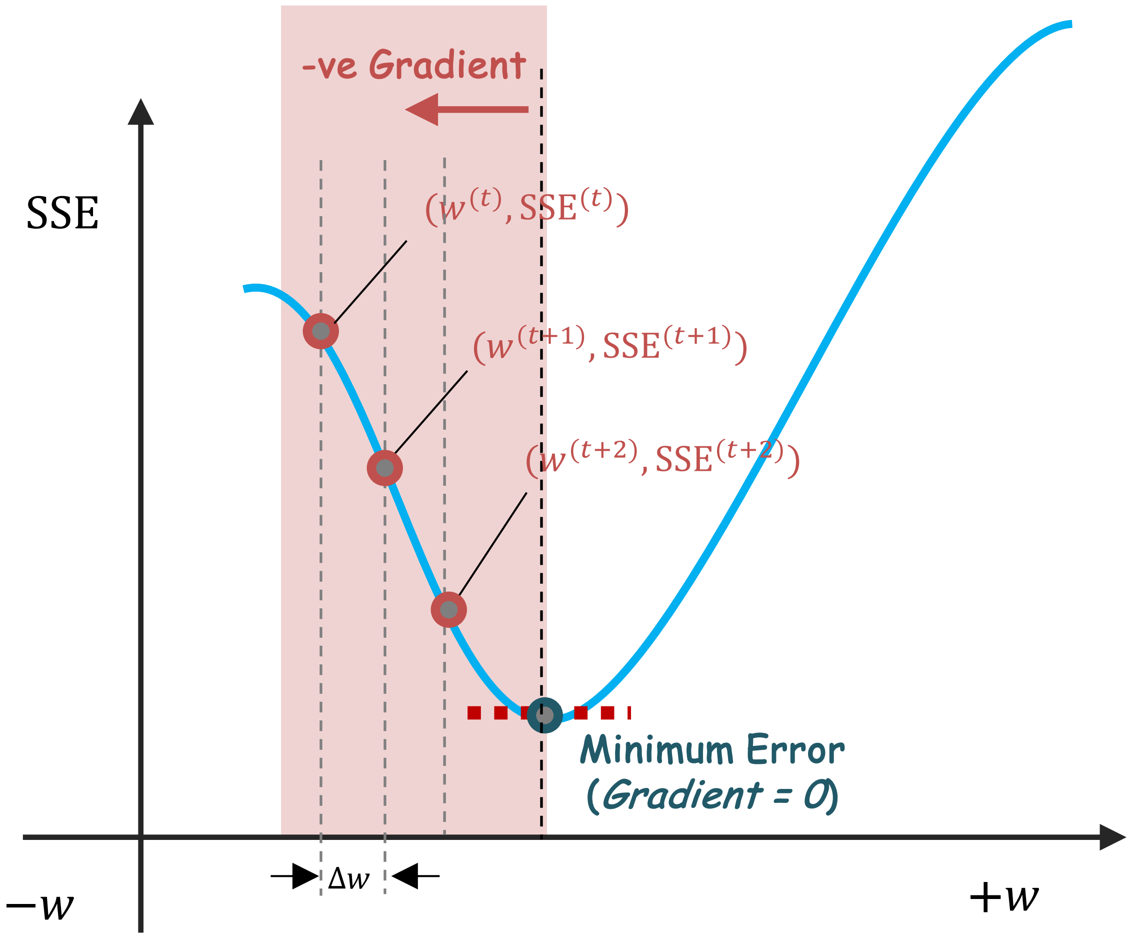
Q: What is the gradient at Q?

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



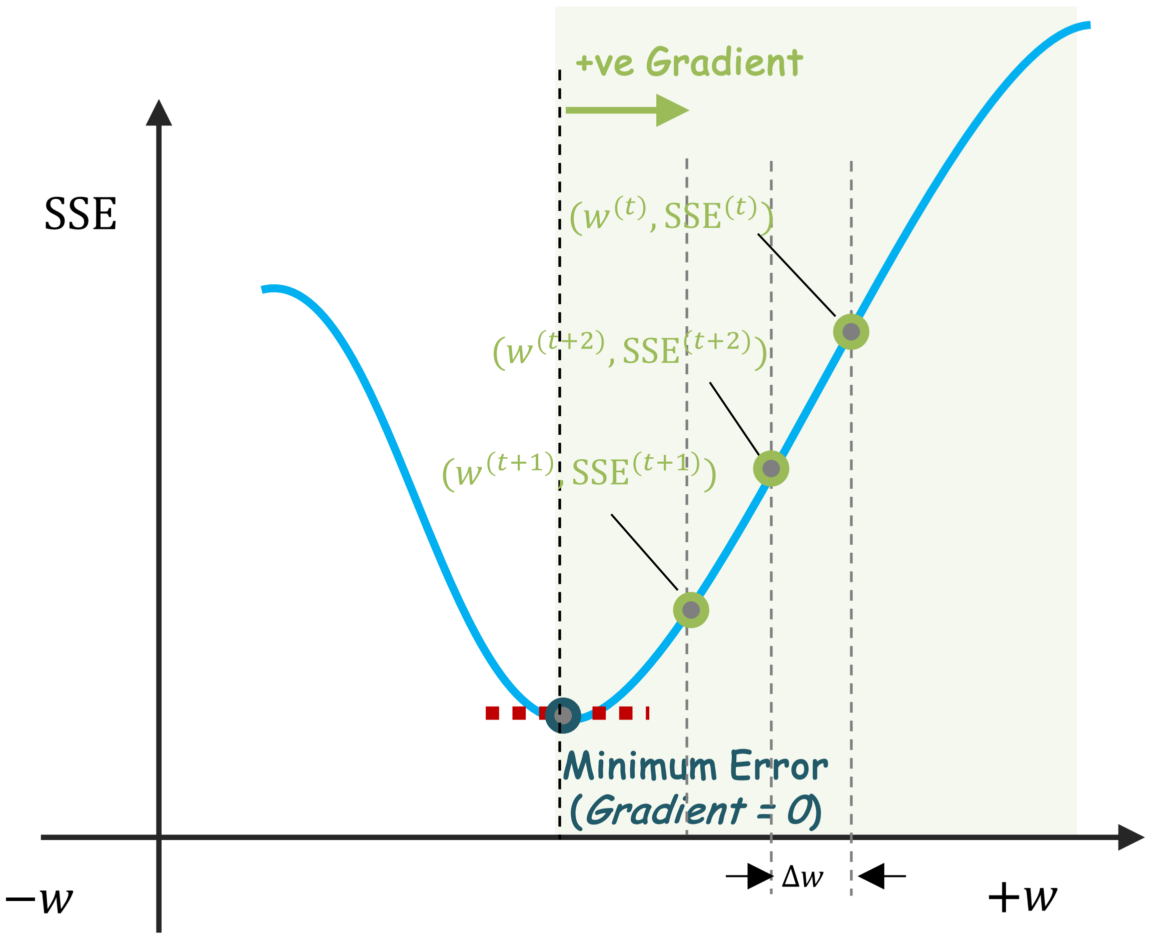
$-\mathrm{ev}$ gradient implies if w *increased* (move in the $+w$ direction) the SSe would also be *decreased*.

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

Gradient Descent: Intuition



$+\mathrm{ev}$ 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. $w^{(t+1)} = w^{(t)} - \Delta w$.

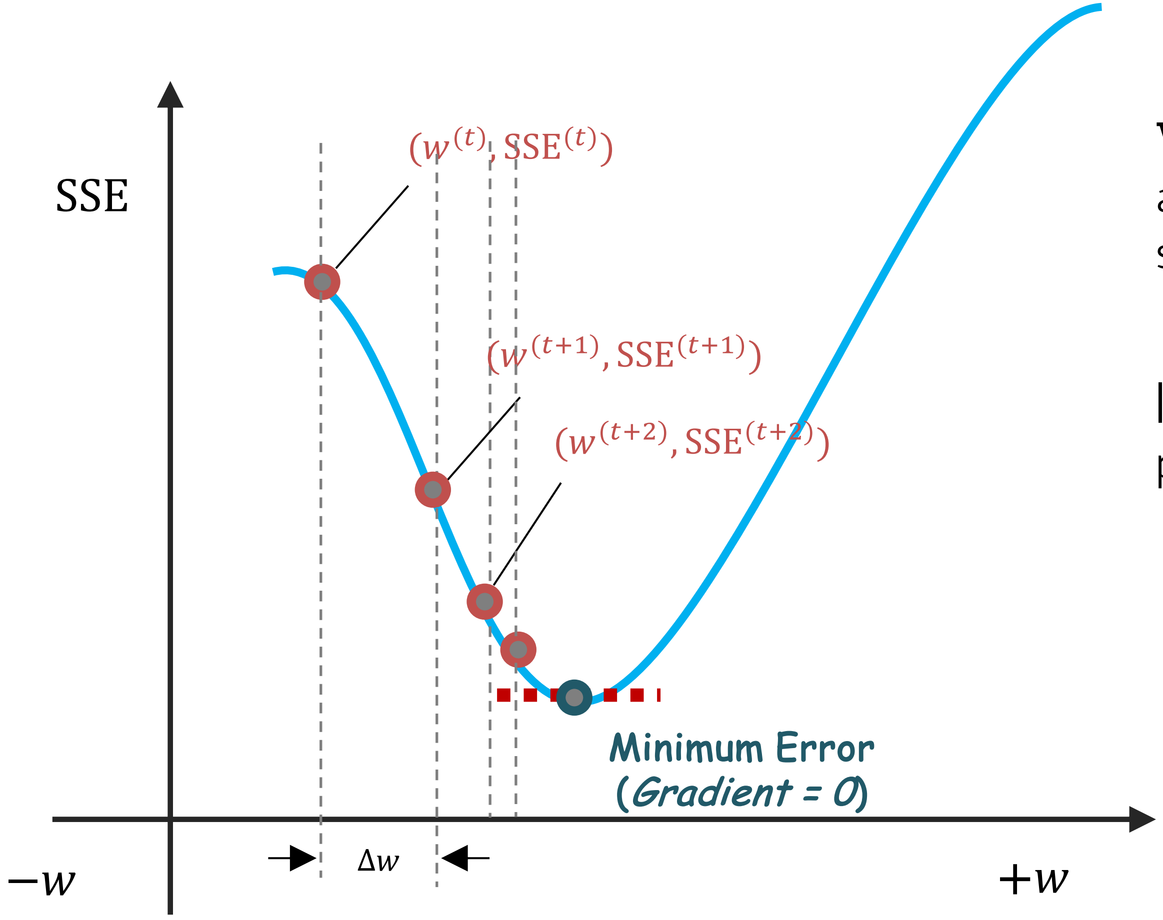
Gradient Descent: Intuition

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

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

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

Gradient Descent: Observation

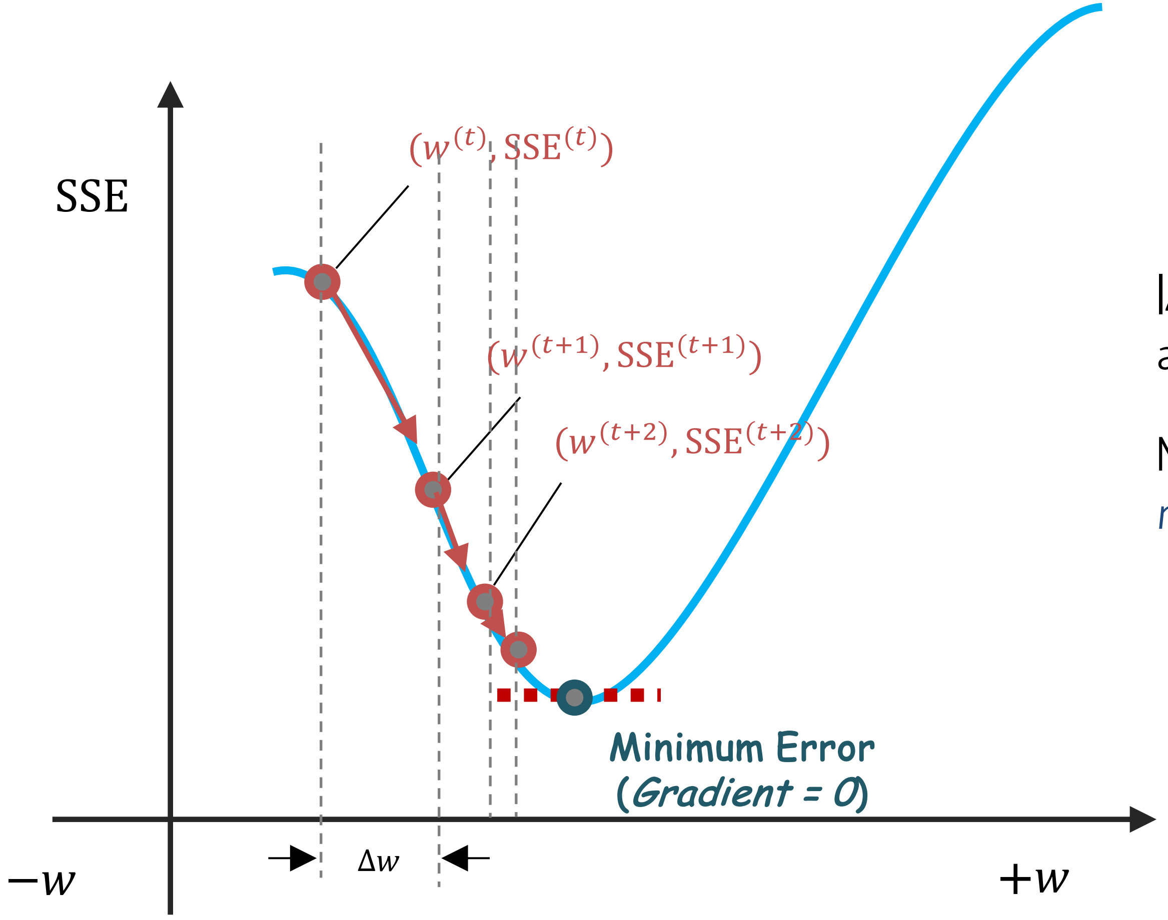


Wish List: We'd like Δw to be large when w is further away from the *minimum* point, and in the opposite *smaller* step when *closer* to the *minimum* point.

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

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

Gradient Descent: Revised



Learning Rate

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

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

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

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

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

Pseudocode for Gradient Descent

```
Function Gradient_Descent(learning_rate, max_iterations, initial_weights, tolerance)
    Begin
        // Step 1: Initialize parameters
        Set weights = initial_weights
        Set iteration = 0

        // Step 2: Start the optimization loop
        While iteration < max_iterations do
            // Step 2.1: Calculate gradients
            Initialize an empty list: gradients
            For each weight in weights do
                Compute the gradient of the error function with respect to the weight
                Add the computed gradient to the gradients list

            // Step 2.2: Check for convergence
            If the magnitude of all gradients < tolerance then
                Break the loop

            // Step 2.3: Update weights
            For each weight in weights do
                Update weight = weight - (learning_rate * corresponding gradient)

            Increment iteration by 1

        // Step 3: Return the optimized weights
        Return weights
    End
```

Python Code Snippet

```python
import numpy as np

# Gradient Descent function
def gradient_descent(X, y, learning_rate=0.01, max_iterations=1000, tolerance=1e-6):
    # Step 1: Initialize parameters
    weights = np.zeros(X.shape[1])
    history = []

    # Step 2: Start the optimization loop
    for iteration in range(max_iterations):
        # Step 2.1: Compute the gradient
        gradient = compute_gradient(X, y, weights)

        # Step 2.2: Update weights
        weights = weights - learning_rate * gradient

        # Step 2.3: Calculate and record the cost (Sum of Squared Errors)
        cost = np.sum((y - X @ weights) ** 2)
        history.append(cost)

        # Step 2.4: Check for convergence
        if np.linalg.norm(gradient) < tolerance:
            print(f"Convergence achieved at iteration {iteration}")
            break

    # Step 3: Return the optimized weights and cost history
    return weights, history
```

Python Code Snippet (cont.)

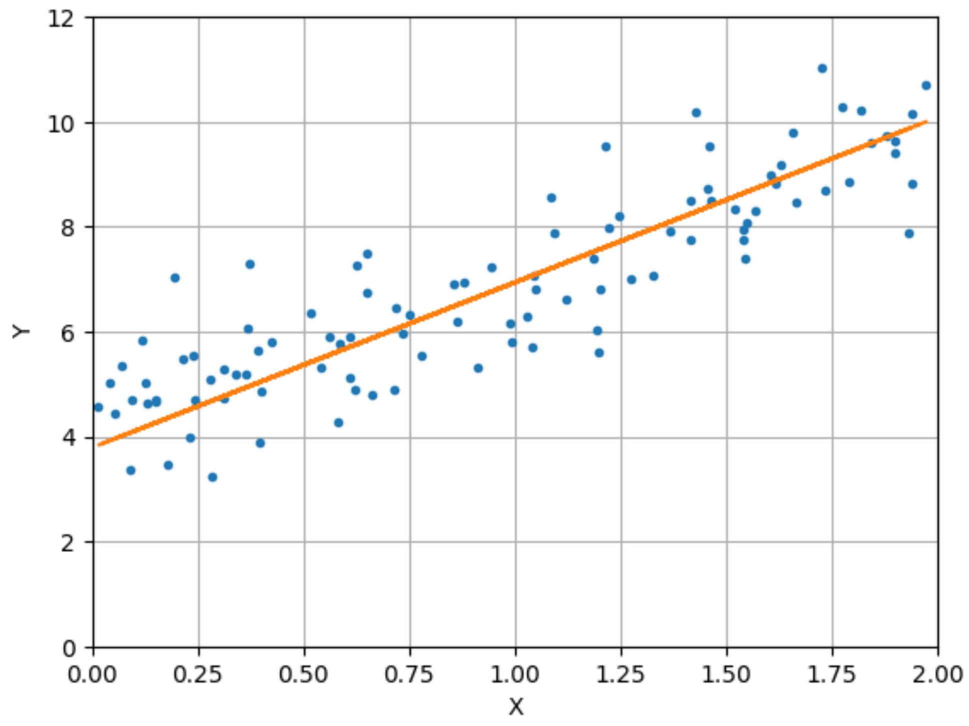
```
# Function to compute gradient for linear regression
def compute_gradient(X, y, weights):
    """
    Computes the gradient of the Sum of Squared Errors (SSE) for linear regression.
    """
    # Step 1.1: Calculate predictions
    predictions = X @ weights

    # Step 1.2: Compute gradient using the formula ∇SSE = -2 * X^T * (y - X * weights)
    gradient = -2 * X.T @ (y - predictions)

    return gradient
```

Usage Example

```
# Set parameters for gradient descent
learning_rate = 0.01
max_iterations = 1000
tolerance = 1e-6

# Perform gradient descent optimization
weights, cost_history = gradient_descent(X_b, y, learning_rate, max_iterations, tolerance)

# Print the optimized weights
print("Optimized Weights:", weights)
```

η

```
Optimized Weights: [3.79008501 3.14520414]
```

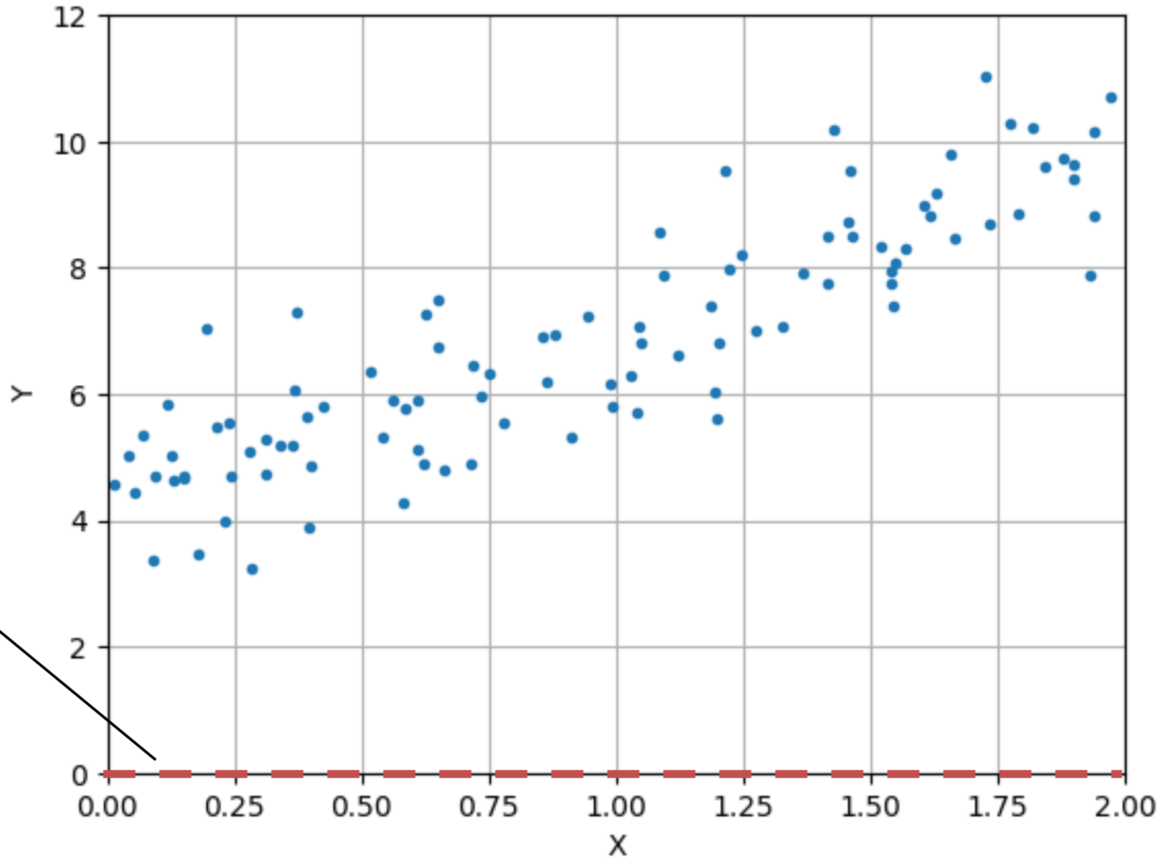


Step 1: Weight Initialisation

```
# Step 1: Initialize parameters
weights = np.zeros(X.shape[1])
history = []
```

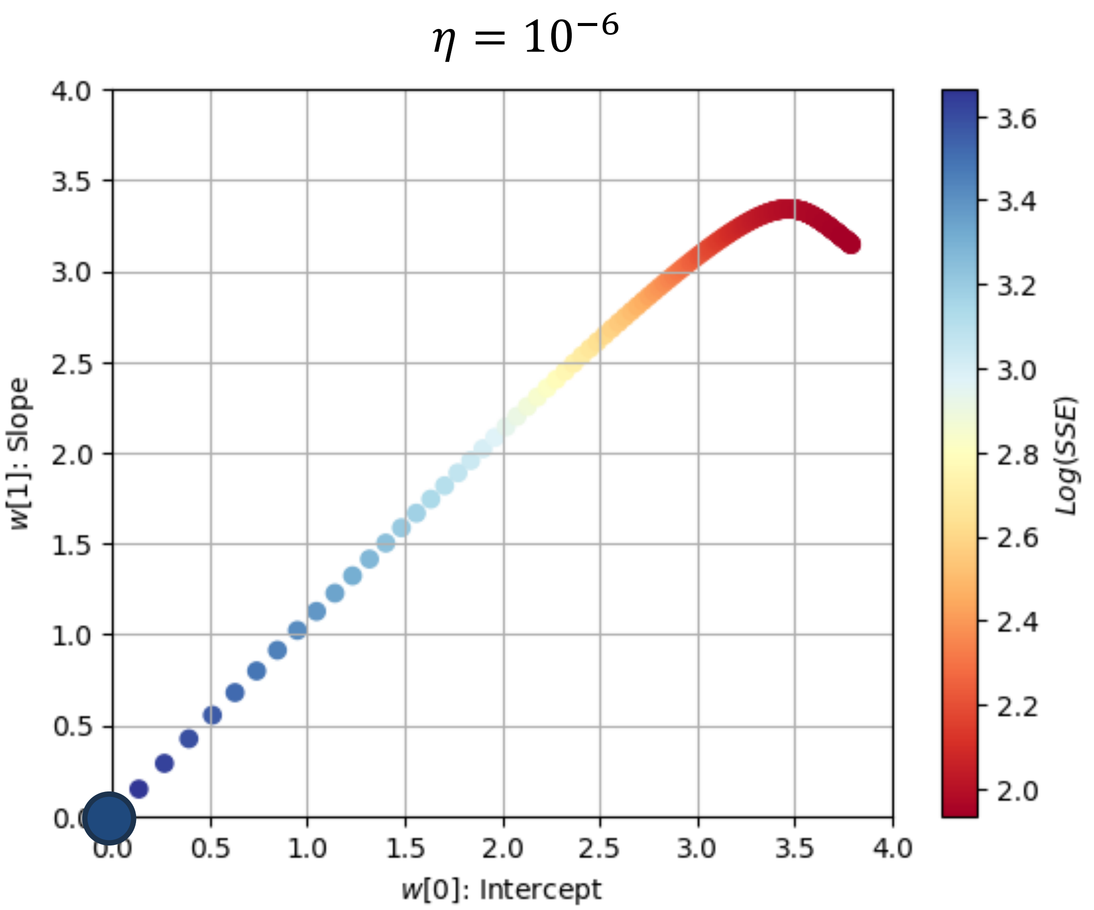
$$\vec{w} = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

$$\hat{y} = 0 \cdot x + 0$$
$$= 0$$



Step 2: Optimisation Loop

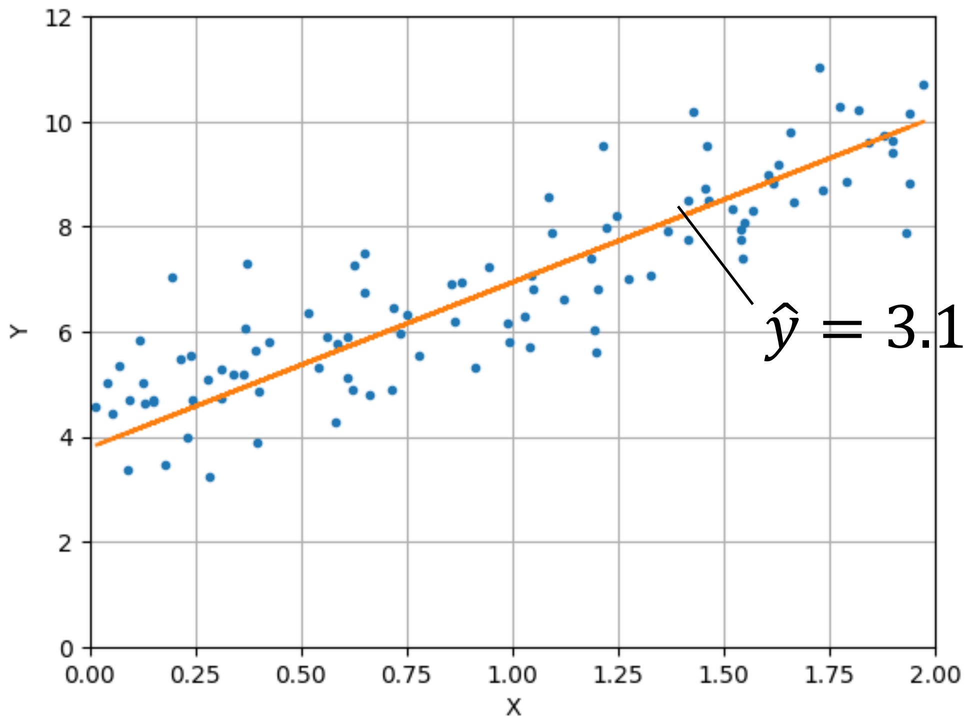
```
# Step 2: Start the optimization loop
for iteration in range(max_iterations):
    # Step 2.1: Compute the gradient
    gradient = compute_gradient(X, y, weights)

    # Step 2.2: Update weights
    weights -= learning_rate * gradient

    # Step 2.3: Calculate and record the cost (Sum of
Squared Errors)
    cost = np.sum((y - X @ weights) ** 2)
    history.append(cost)

    # Step 2.4: Check for convergence
    if np.linalg.norm(gradient) < tolerance:
        print(f"Convergence achieved at iteration
{iteration}")
        break
```



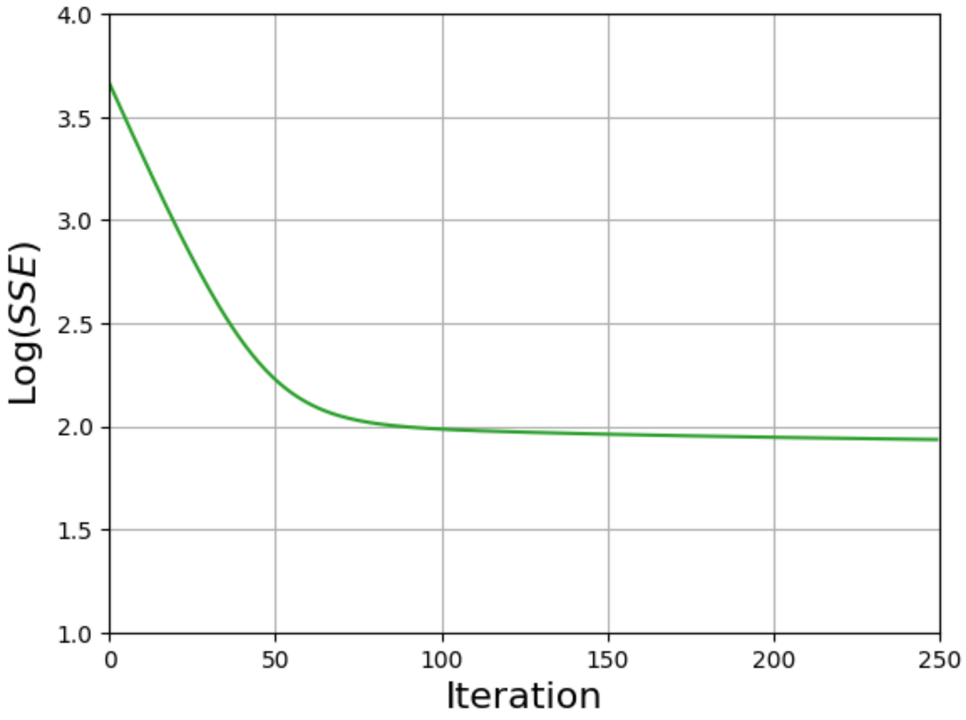
Step 4 & 5: Return Optimised Weights

```
# Step 3: Return the optimized weights and cost history
return weights, history
```

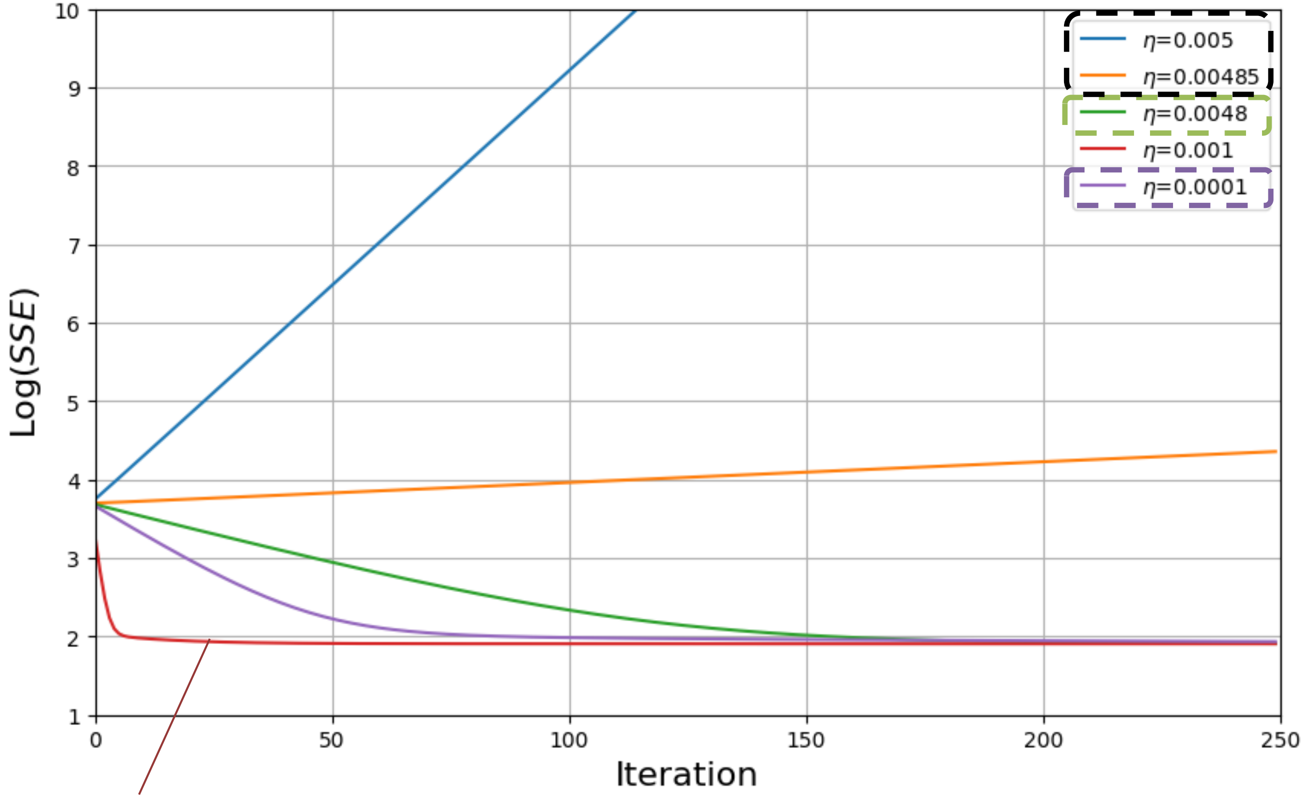
```
Optimized Weights: [3.79008501 3.14520414]
```



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



Learning Rate



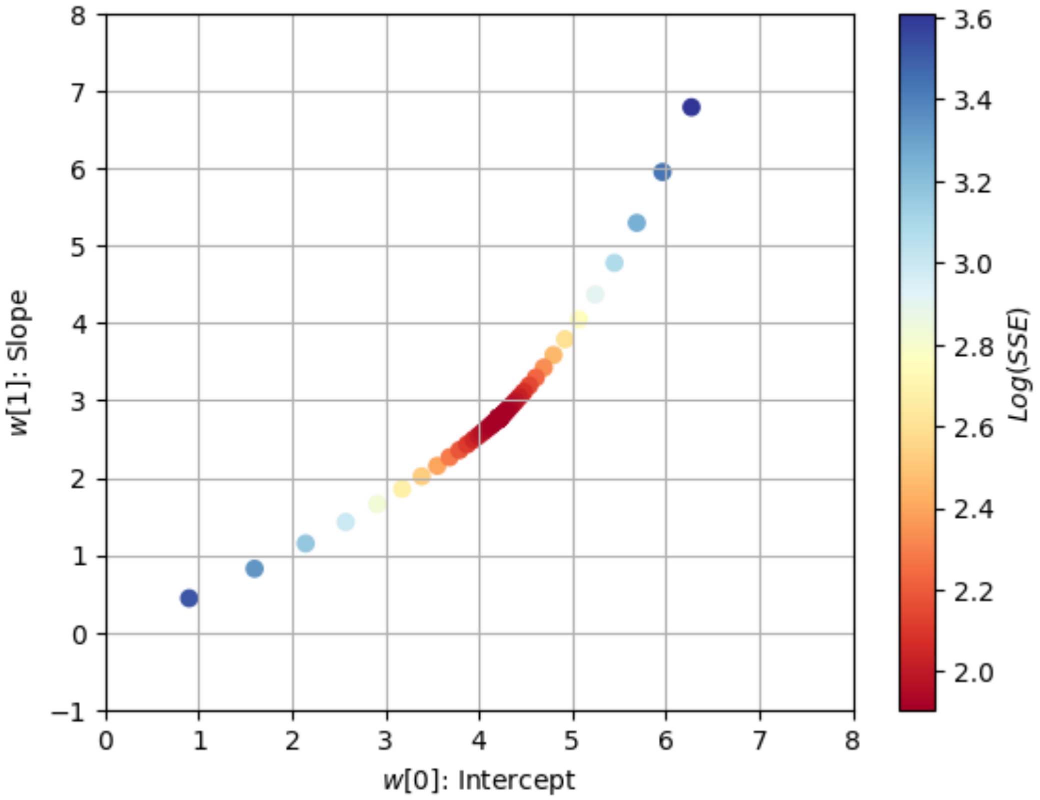
…just right…

- 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.

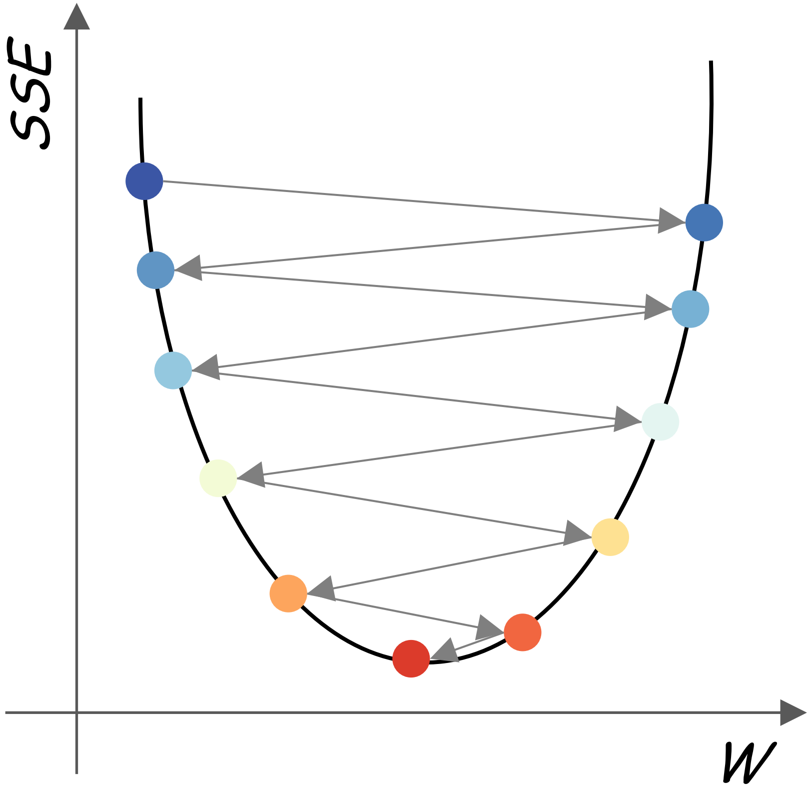
High Learning Rate

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

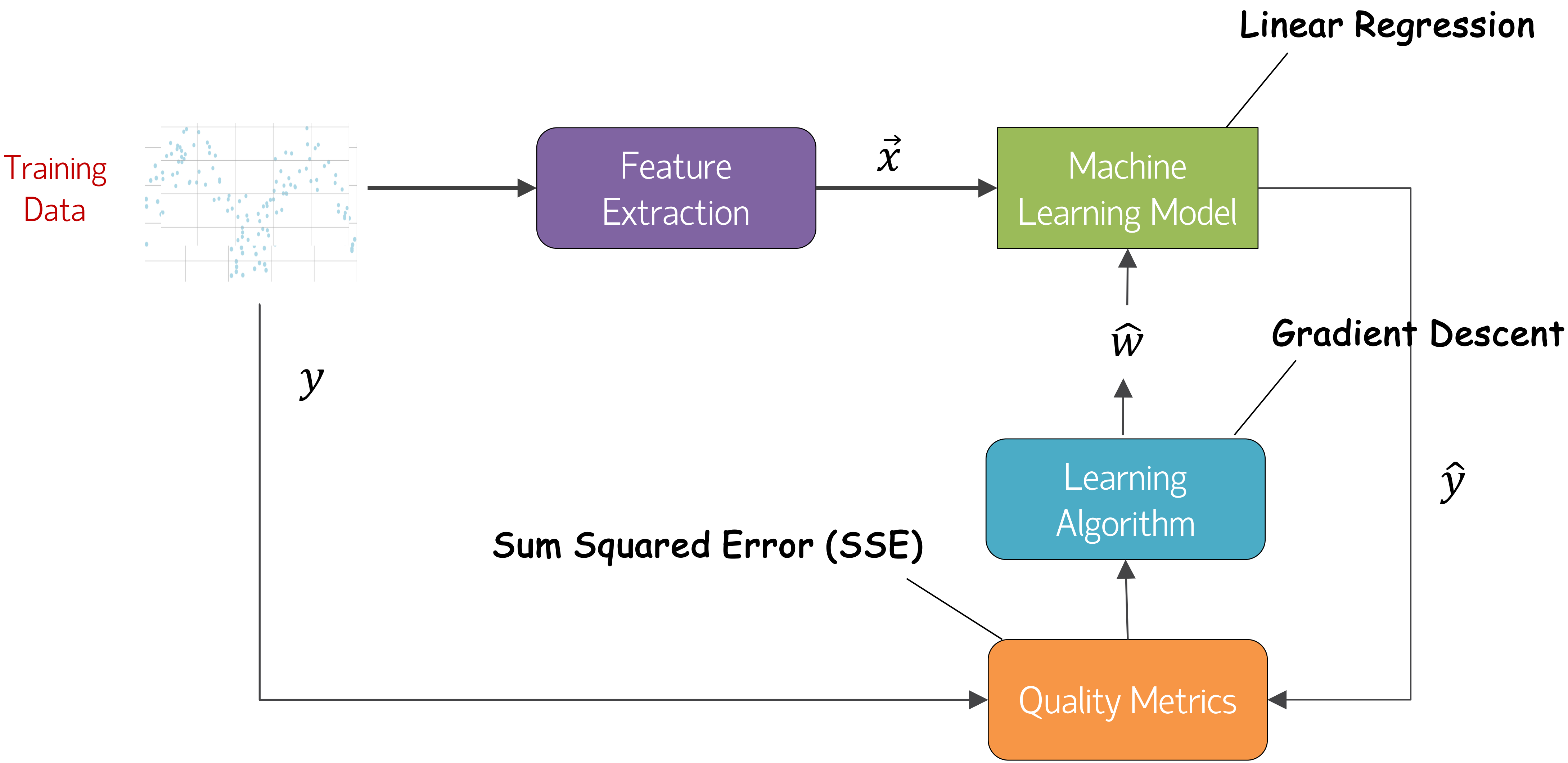


Q: It looks like there are 2 trajectories, i.e. converges from bottom left and from top right. What happened?

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.