Overfitting occurs when the model learns to "memorize" the training data, capturing noise or irrelevant patterns instead of generalizing to unseen data. In linear regression, this can result in **large weights** for some features, leading to extreme and unstable predictions.

L1 and L2 regularization are techniques used to prevent overfitting in machine learning models by adding a penalty term to the loss function. These penalties discourage the model from assigning overly large values to the weights.

X = np.array([1, 2, 3, 4, 5])

y = np.array([3, 4, 2, 4, 5])

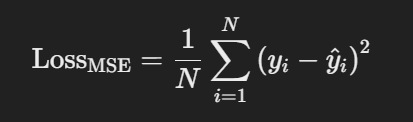
Suppose we are performing linear regression to fit the model:

Step 1 : YPred=w⋅X+b

where, w is the weight and b is the bias.

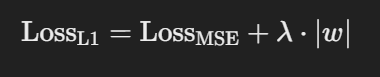
**Step 2 : Loss Function with Regularization**

1. **Without Regularization** (Mean Squared Error):

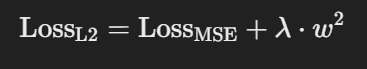


MSE measures how well the model fits the training data.

1. **With L1 Regularization (Lasso)**:



1. **With L2 Regularization (Ridge)**:



λ is a hyperparameter that controls the strength of the L2 penalty. It adds the **square of the weight** (w^2) to the loss function. Where, w^2 is the sum of the squares of all the weights in the model.

**Role of w^2:**

1. **Prevents Large Weights**:
   * The w^2 penalty term ensures that the optimization process discourages large values of w. A large weight w significantly increases the penalty, so the model tends to keep the weights small.
   * For example, if a feature X has too much influence (high w), the penalty term λ⋅w2 will reduce its impact, making the model more stable.
2. **Balances Model Complexity and Fit**: The penalty term forces the model to prioritize simplicity over complexity. This helps strike a balance between fitting the training data well and avoiding overfitting.
3. **Improves Generalization**: By keeping the weights small, the model avoids over-relying on specific features. This results in better predictions for unseen data

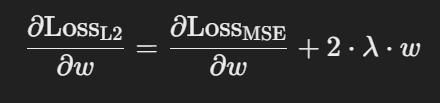
**Step 3 : Gradient Updates with Regularization.**

The gradients i.e. both w and b are adjusted based on the regularization type:

1. **L1 Regularization Gradient (absolute value)**: 

Where sign(w) is 1 if w>0, −1 if w<0, and 0 if w=0.

1. **L2 Regularization Gradient (squared)**:



**Key Differences**

* **L1 Regularization** (Lasso):
  + Encourages sparsity by pushing some weights to exactly zero, making it useful for feature selection.
  + The penalty grows linearly with the weight (∣w|).
* **L2 Regularization** (Ridge):
  + Discourages large weights but doesn't force them to zero.
  + The penalty grows quadratically with the weight (w^2).

After calculating the **regularized loss (step 2 or step 3)** and updating the model parameters (weights w and bias b) using gradient descent, the next steps are as follows:

**Step 4: Continue Gradient Descent Iterations**

To finalize the model parameters w and b, you need to:

1. **Perform more iterations (epochs)** of gradient descent.
   * For each data point, calculate:
     + Predictions (YPred​).
     + The regularized loss (MSE + L1/L2 penalty).
     + Gradients of the loss with respect to w and b.
   * Update w and b using the gradient descent update rules.
2. **Repeat until convergence**:
   * The model parameters are refined over multiple iterations until the loss function no longer decreases significantly (converges).
   * Alternatively, stop after a predefined number of epochs.

**Step 5: Final Prediction**

Once the model has been trained (i.e., w and b have been optimized), the **final predictions** for any new input X can be made using the learned parameters:

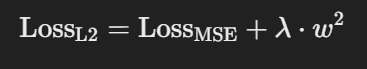
YPred=w⋅X+b

Step-by-step calculation for the given data using **L2 regularization** (Ridge Regression).

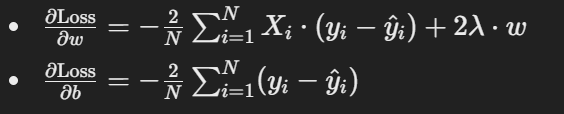
We'll follow the gradient descent process for **3 iterations (epochs)** with a learning rate (η=0.01) and regularization strength (λ=0.1).

**Given**

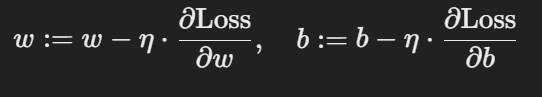
* Data: X=[1,2,3,4,5] , y=[3,4,2,4,5]
* Initialize: w=0.0, b=0.0
* Loss Function :



* Gradients :



* Update rules:



**Step 1: Epoch 1**

**1. Predicted values (y^i):**

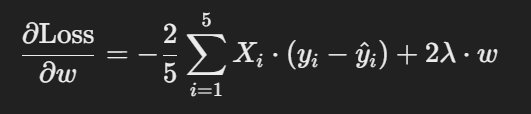
y^​i​ = w⋅Xi​+b

Substitute w=0.0 , b=0.0:

y^​=[0,0,0,0,0]

**2. Gradients**

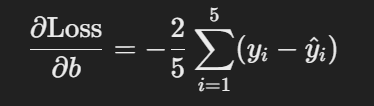
* Error: y−y^=[3,4,2,4,5]
* Compute ∂Loss/∂w :



So, ∂Loss/∂w ​=−2/5\*(​(1⋅3+2⋅4+3⋅2+4⋅4+5⋅5)+2⋅0.1⋅0)

Therefore, ∂Loss/∂w ​=−2/5​⋅54=−21.6

* Compute ∂Loss/∂b ​:



∂Loss​/∂b =−2/5​(3+4+2+4+5)

∂Loss/∂b=−2/5⋅18=−7.2

**3. Update w and b:**

w =w−η⋅∂Loss/∂w=0−0.01⋅(−21.6)=0.216

b =b−η⋅∂Loss/∂b ​=0−0.01⋅(−7.2)=0.072

**Step 2: Epoch 2**

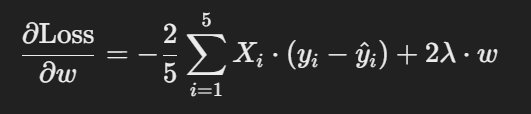
**1. Predicted values (y^i​):**

y^​i​= w⋅Xi​+b = 0.216⋅Xi​+0.072

y^=[0.288,0.504,0.72,0.936,1.152]

**2. Gradients**

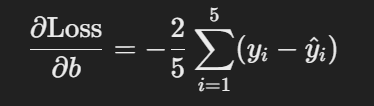
* Error: y−y^ = [3−0.288,4−0.504,2−0.72,4−0.936,5−1.152] = [2.712,3.496,1.28,3.064,3.848]
* Compute ∂Loss/∂w



∂Loss​/∂w =−2/5​\*((1⋅2.712+2⋅3.496+3⋅1.28+4⋅3.064+5⋅3.848)+2⋅0.1⋅0.216)

∂Loss/∂w=−2/5\*55.448+0.0432=−22.1792

* Compute ∂Loss/∂b ​:



∂Loss​/∂b =−2/5​⋅14.4=−5.76

**3. Update w and b:**

w = w−η⋅∂Loss/∂w=0.216−0.01⋅(−22.1792)=0.4388

b = b−η⋅ ∂Loss​/∂b =0.072−0.01⋅(−5.76)=0.1296

**Step 3: Epoch 3**

Repeat the same process:

* Predicted values:

y^=[0.5684,1.0072,1.446,1.8848,2.3236]

* Gradients:
  + ∂Loss/∂w=−19.358+2⋅0.4388=−19.2404
  + ∂Loss/∂b=−4.84
* Updated parameters:

w =0.4388−0.01⋅(−19.2404)=0.6312

b =0.1296−0.01⋅(−4.84)=0.178

**Step 4: Final Prediction**

After several epochs, we use the learned w and b to predict:

y^= w⋅X+b

For w=0.6312 , b=0.178 :

y^=[0.8092,1.4404,2.0716,2.7028,3.334]