the model optimization is achieved through **gradient descent**, which is an iterative optimization algorithm commonly used for minimizing the loss function (error) in machine learning models like linear regression. Here's a breakdown of how the optimization works:

Key Concepts for Optimization:

1. **Objective**: The goal of linear regression is to minimize the difference between the predicted values (ypredictedy_{\text{predicted}}ypredicted) and the actual values (yyy) by adjusting the model parameters (weights and bias). This difference is quantified by a loss function, typically the **Mean Squared Error (MSE)** in linear regression.
   The loss function LLL can be expressed as:
   L(weights,bias)=1nsamples∑i=1nsamples(ypredictedi−yi)2L(\text{weights}, \text{bias}) = \frac{1}{n_{\text{samples}}} \sum_{i=1}^{n_{\text{samples}}} (y_{\text{predicted}_i} - y_i)^2L(weights,bias)=nsamples1i=1∑nsamples(ypredictedi−yi)2
   The **weights** and **bias** are updated iteratively to minimize this loss function using gradient descent.
2. **Gradient Descent**: Gradient descent optimizes the model by adjusting the weights and bias in small steps proportional to the negative gradient of the loss function. This moves the parameters towards values that reduce the error.
   The update rules for the parameters (weights and bias) in each iteration are:
   weights=weights−α⋅∇weightsL\text{weights} = \text{weights} - \alpha \cdot \nabla_{\text{weights}} Lweights=weights−α⋅∇weightsL
   bias=bias−α⋅∇biasL\text{bias} = \text{bias} - \alpha \cdot \nabla_{\text{bias}} Lbias=bias−α⋅∇biasL
   where:
   - α\alphaα is the learning rate, controlling the step size of the updates.
   - ∇weightsL\nabla_{\text{weights}} L∇weightsL and ∇biasL\nabla_{\text{bias}} L∇biasL are the gradients of the loss function with respect to the weights and bias, respectively.

Breakdown of the `fit` Function:

Initialize Parameters:
python
Copy code

```
n_samples, n_features = X.shape
self.weights = np.zeros(n_features)
self.bias = 0
```

1.
   - The weights are initialized to a zero vector of size equal to the number of features (`n_features`), and the bias is initialized to 0.

Gradient Descent Loop:
python

Copy code

```
for _ in range(self.n_iters):
    y_predicted = np.dot(X, self.weights) + self.bias
    dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
    db = (1 / n_samples) * np.sum(y_predicted - y)
    self.weights -= self.lr * dw
    self.bias -= self.lr * db
```

2.
    - **Predicted values (`y_predicted`)**:
      ypredicted=X⋅weights+biasy_{\text{predicted}} = X \cdot \text{weights} + \text{bias}ypredicted=X⋅weights+bias
      This is the model's prediction based on the current weights and bias.
    - **Calculate Gradients (`dw` and `db`)**:
        - **Gradient of weights (`dw`)**:
          dw=1nsamples⋅XT⋅(ypredicted−y)dw = \frac{1}{n_{\text{samples}}} \cdot X^T \cdot (y_{\text{predicted}} - y)dw=nsamples1⋅XT⋅(ypredicted−y)
          This computes how much the weights should change in order to reduce the loss. It's derived by calculating the derivative of the loss function with respect to the weights.
        - **Gradient of bias (`db`)**:
          db=1nsamples⋅∑(ypredicted−y)db = \frac{1}{n_{\text{samples}}} \cdot \sum (y_{\text{predicted}} - y)db=nsamples1⋅∑(ypredicted−y)
          This computes the gradient for the bias, which is derived by taking the derivative of the loss function with respect to the bias.

Update Weights and Bias:
python
Copy code

```
self.weights -= self.lr * dw
self.bias -= self.lr * db
```

    -
        - The weights are updated by subtracting the learning rate multiplied by the gradient (`dw`). This moves the weights in the direction that reduces the error.
        - The bias is updated similarly by subtracting the learning rate multiplied by `db`.

3. **Iteration**:
    - The loop repeats for `n_iters` iterations, continually updating the weights and bias until the error is minimized or reduced significantly. The learning rate (`self.lr`) controls how big the steps are, preventing overshooting the optimal point while ensuring the optimization converges.

Optimization Process:

- **Starting Point**: Initially, the weights and bias are set to zero. In the first iteration, the model makes predictions using these initial values.
- **Gradients Computation**: For each iteration, the gradient of the loss function is computed with respect to both weights and bias. The gradient indicates the slope or direction in which the loss function increases. By moving in the opposite direction of the gradient, the model reduces the error.
- **Parameter Update**: The weights and bias are updated using gradient descent, gradually moving toward values that minimize the loss function. Each iteration moves closer to the optimal solution.
- **Stopping Condition**: After `n_iters` iterations, the algorithm stops. At this point, the model has ideally learned the best-fitting line for the given data by minimizing the error.

Summary:

The model optimization in this code is carried out using **gradient descent**, where:

- The loss function is minimized by iteratively adjusting the weights and bias.
- The updates are controlled by the learning rate, and they follow the gradients of the loss function.
- This process continues for `n_iters` iterations, refining the model to fit the training data.

This optimization technique allows the model to find the optimal weights and bias that minimize the prediction error.