BITS Pilani
Pilani Campus

BITS Pilani presentation

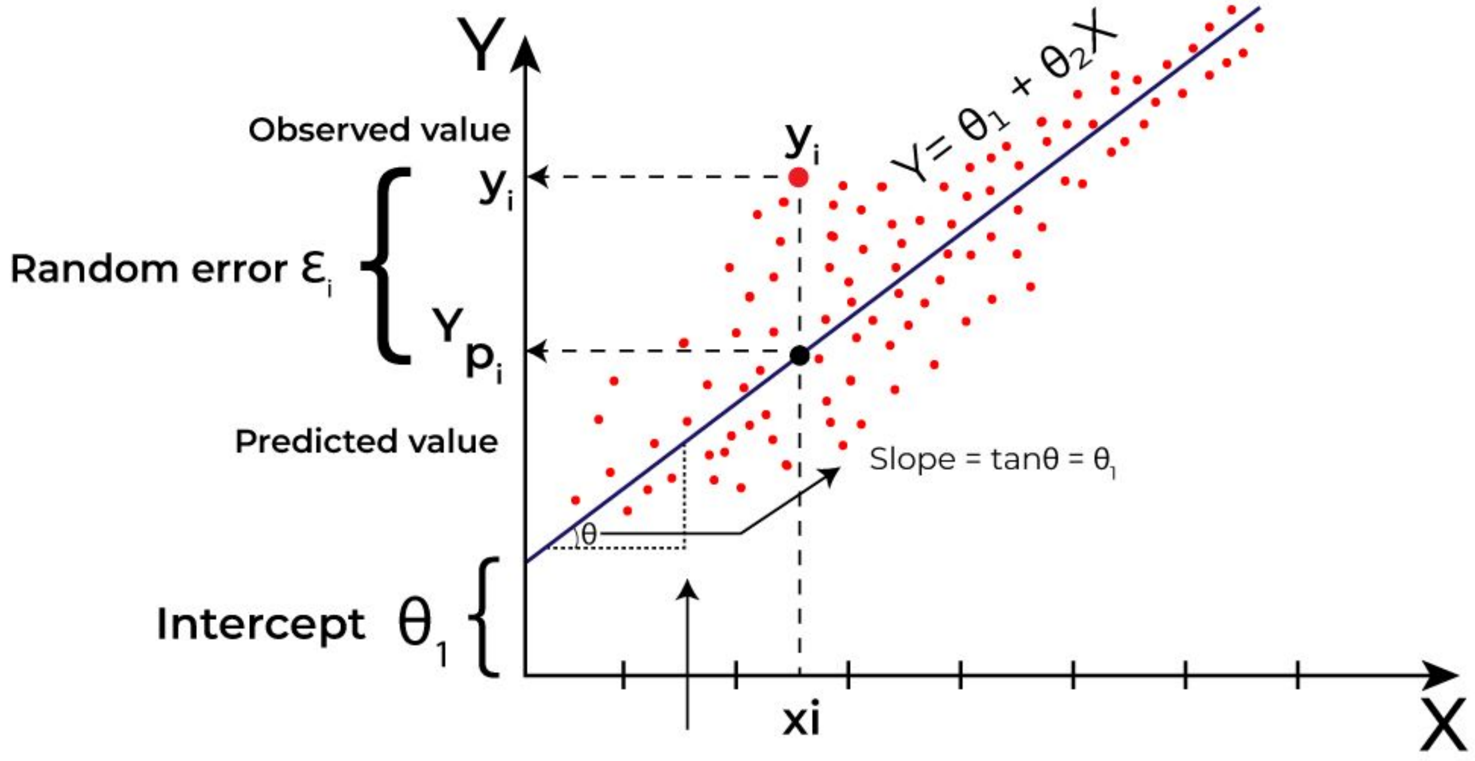
Tanmay Tulsidas Verlekar
CSIS

innovate achieve lead

BITS Pilani
Pilani Campus

Applied Machine Learning SE ZG568 / SS ZG568 Lecture No. 4

Regression



Y
Observed value
y_i
Random error ε_i
Y_{p_i}
Predicted value
Intercept θ_1
y_i
$Y = \theta_1 + \theta_2 X$
θ
Slope = tanθ = θ_1
xi
X

Generalised form

Equation 4-1. Linear Regression model prediction

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- $\hat{y}$ is the predicted value.
- n is the number of features.
- x_i is the ith feature value.
- θ_j is the jth model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

Equation 4-2. Linear Regression model prediction (vectorized form)

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

Error

The MSE of a Linear Regression hypothesis $h_{\boldsymbol{\theta}}$ on a training set $\mathbf{X}$ is calculated using Equation 4-3.

Equation 4-3. MSE cost function for a Linear Regression model

$$\text{MSE}\left(\mathbf{X}, h_{\boldsymbol{\theta}}\right) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Normal Equation

$$\left|\begin{array}{l} \theta_0 \begin{pmatrix} 0 \\ x_0 \end{pmatrix} + \theta_1 \begin{pmatrix} 0 \\ x_1 \end{pmatrix} + \ldots.\ \theta_n \begin{pmatrix} 0 \\ x_n \end{pmatrix} \\ \theta_0 \begin{pmatrix} 1 \\ x_0 \end{pmatrix} + \theta_1 \begin{pmatrix} 1 \\ x_1 \end{pmatrix} + \ldots.\ \theta_n \begin{pmatrix} 1 \\ x_n \end{pmatrix} \\ \ldots \\ \theta_0 \begin{pmatrix} m \\ x_0 \end{pmatrix} + \theta_1 \begin{pmatrix} m \\ x_1 \end{pmatrix} + \ldots.\ \theta_n \begin{pmatrix} m \\ x_n \end{pmatrix} \end{array}\right| - y$$

We cannot simply square the above expression. As the square of a vector/matrix is not equal to the square of each of its values. So to get the squared value, multiply the vector/matrix with its transpose.

$$Cost = (X\theta - y)^T (X\theta - y)$$

Calculating the value of θ using the partial derivative

$$\frac{\partial J}{\partial \theta} = \frac{\partial}{\partial \theta}\left((X\theta - y)^T(X\theta - y)\right)$$

$$\frac{\partial J}{\partial \theta} = 2X^T(X\theta - y)$$

We know to find the optimum value of any partial derivative equation we have to equate it to 0.

$$2X^TX\theta - X^Ty = 0$$

Equation 4-4. Normal Equation

$$\hat{\theta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1} \ \mathbf{X}^T \ \mathbf{y}$$

Solved example

X= $\begin{bmatrix} 1 & -1 \\ 3 & 2 \\ -2 & 4 \end{bmatrix}$ Y= $\begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix}$

We have

$$A^T A = \begin{bmatrix} 1 & 3 & -2 \\ -1 & 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 3 & 2 \\ -2 & 4 \end{bmatrix} = \begin{bmatrix} 14 & -3 \\ -3 & 21 \end{bmatrix}$$

$$A^T \mathbf{b} = \begin{bmatrix} 1 & 3 & -2 \\ -1 & 2 & 4 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 10 \end{bmatrix}$$

so the normal system $A^{\mathrm{T}}A\mathbf{x} = A^T\mathbf{b}$ in this case is $\begin{bmatrix} 14 & -3 \\ -3 & 21 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 10 \end{bmatrix}$

Computational Complexity

The Normal Equation computes the inverse of $\mathbf{X}^T \mathbf{X}$, which is an $(n + 1) \times (n + 1)$ matrix (where n is the number of features). The *computational complexity* of inverting such a matrix is typically about $O(n^{2.4})$ to $O(n^3)$ (depending on the implementation). In other words, if you double the number of features, you multiply the computation time by roughly $2^{2.4} = 5.3$ to $2^3 = 8$.

Gradient Descent

Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

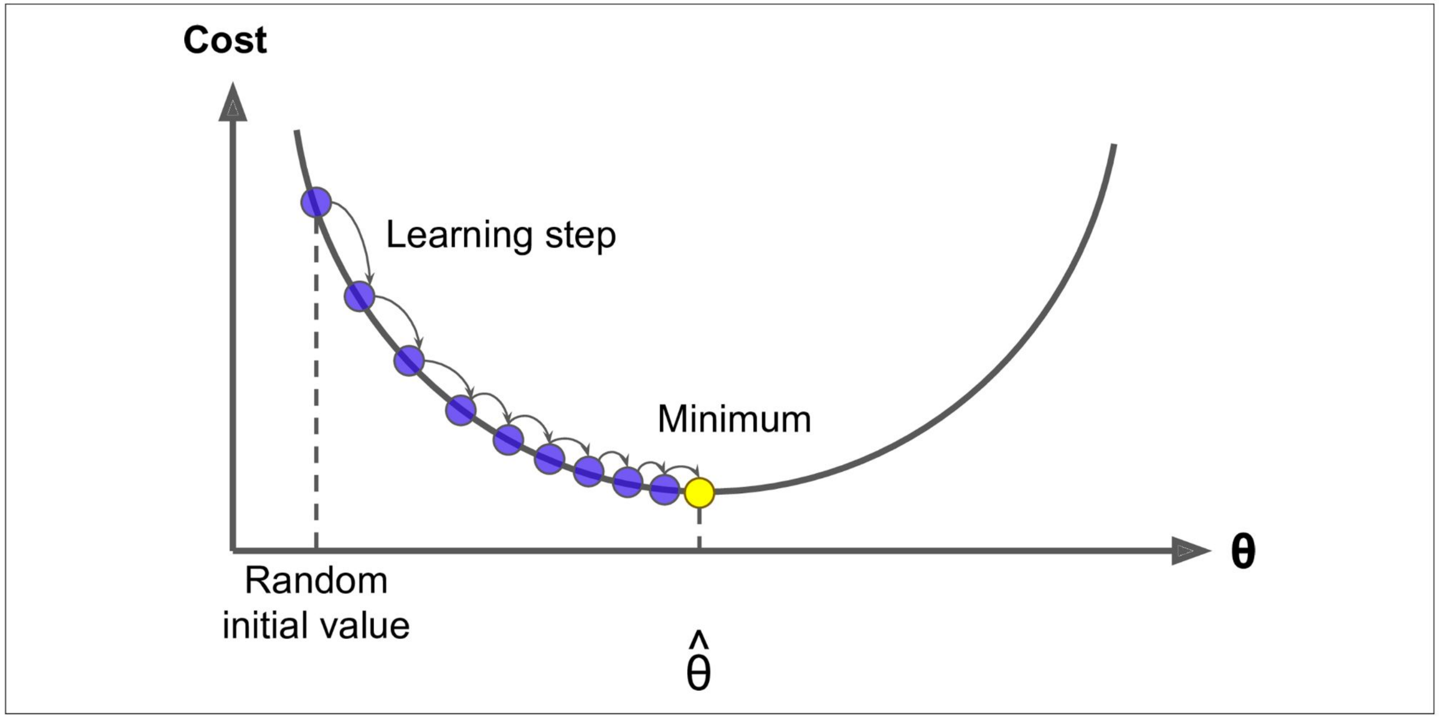
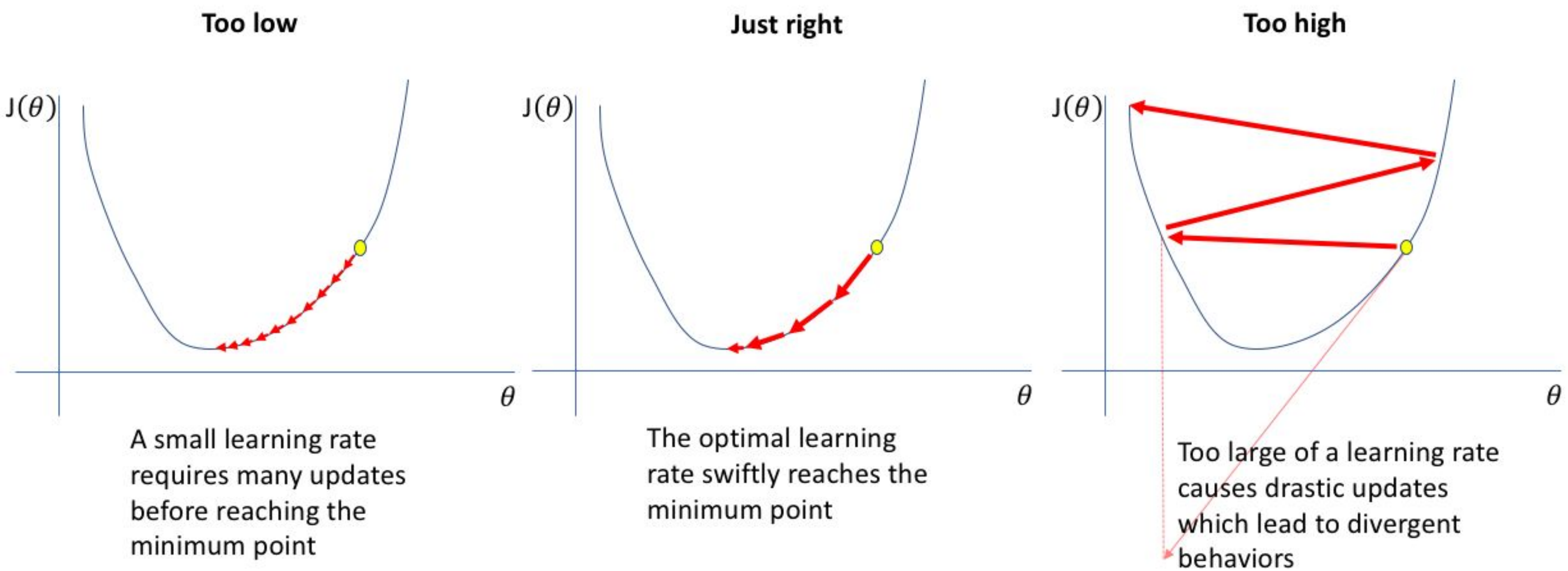


Figure 4-3. Gradient Descent

Learning rate



A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

Too large of a learning rate causes drastic updates which lead to divergent behaviors

Irregular terrains

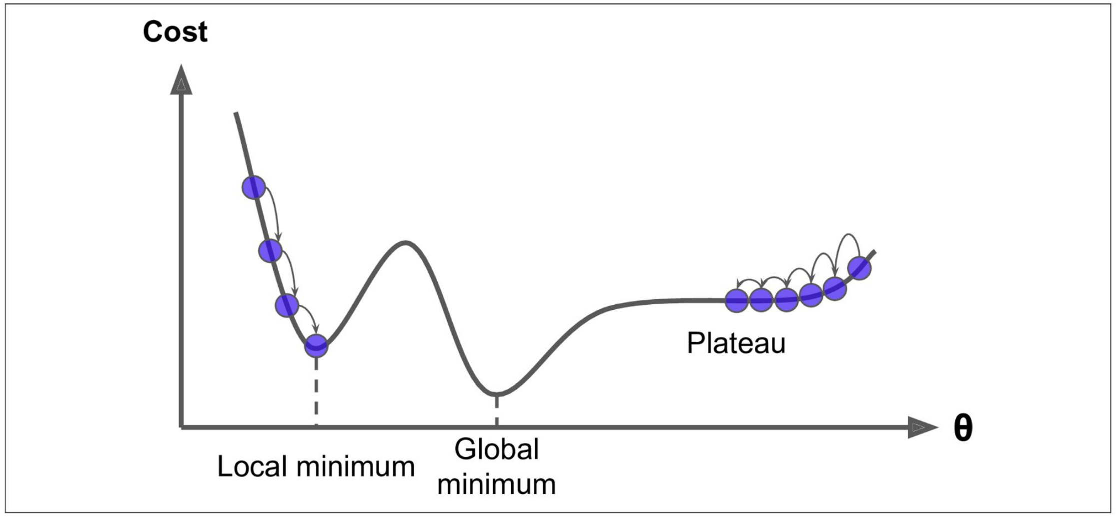


Figure 4-6. Gradient Descent pitfalls

Fortunately, the MSE cost function for a Linear Regression model happens to be a convex function.

Gradient Descent is guaranteed to approach arbitrarily close the global minimum (if you wait long enough and if the learning rate is not too high).

The need for scaling

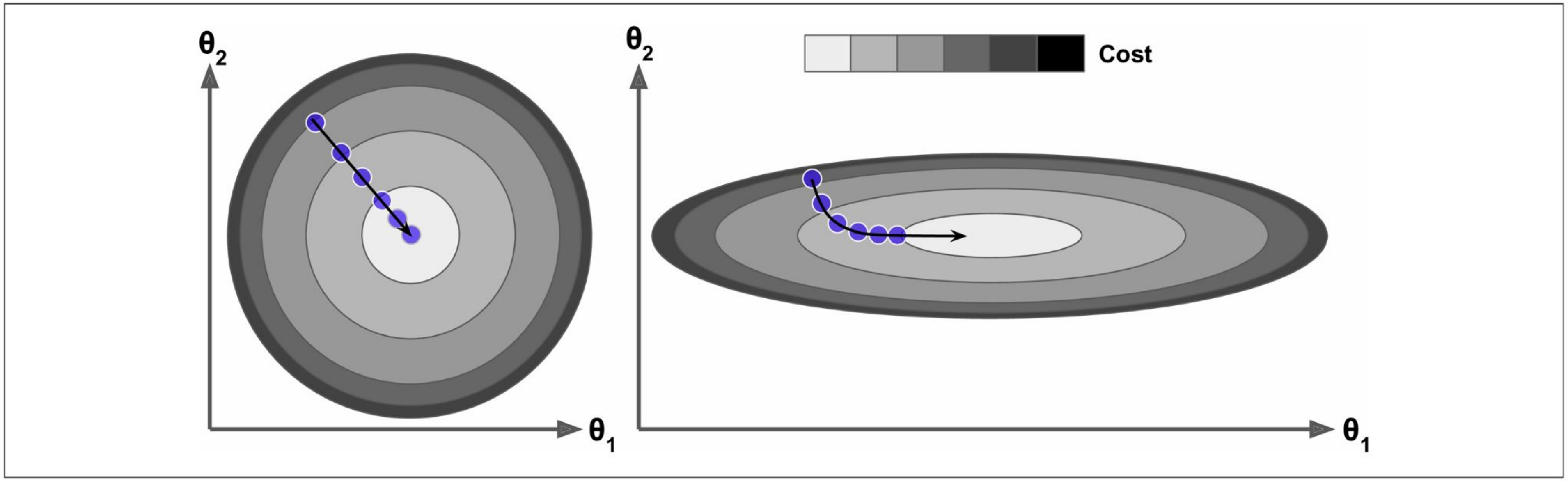


Figure 4-7. Gradient Descent with and without feature scaling

Batch Gradient Descent

We need to calculate how much the cost function will change if you change θj just a little bit.

Equation 4-5. Partial derivatives of the cost function

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Equation 4-6. Gradient vector of the cost function

$$\nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Equation 4-7. Gradient Descent step

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$$

Stochastic Gradient Descent

It picks a random instance in the training set at every step and computes the gradients based only on that single instance.

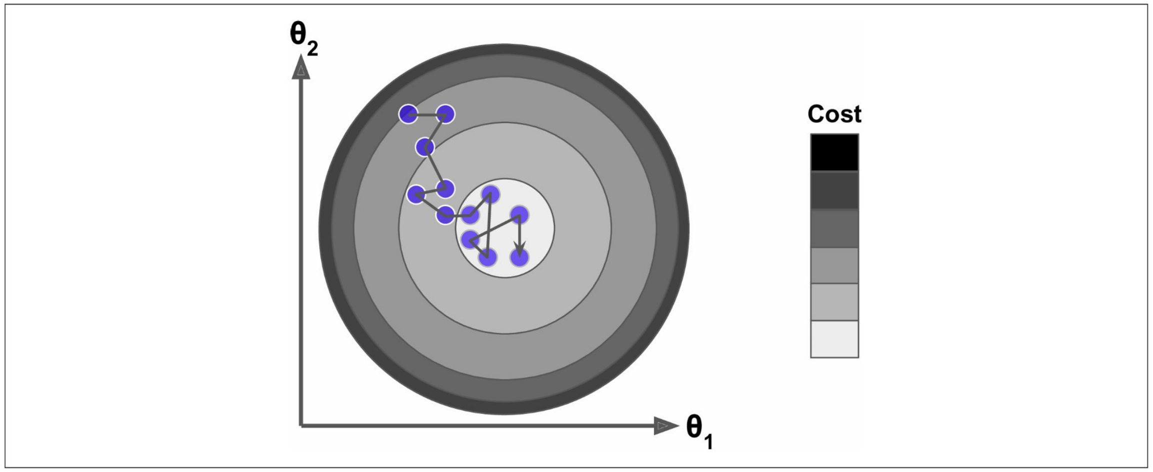


Figure 4-9. Stochastic Gradient Descent

The learning rate is determined at each iteration.

Example

$w_1 = w_2 = b = 0;$

$x_1 = 3; \quad x_2 = 2$

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

$\eta = 0.1;$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Mini-batch Gradient Descent

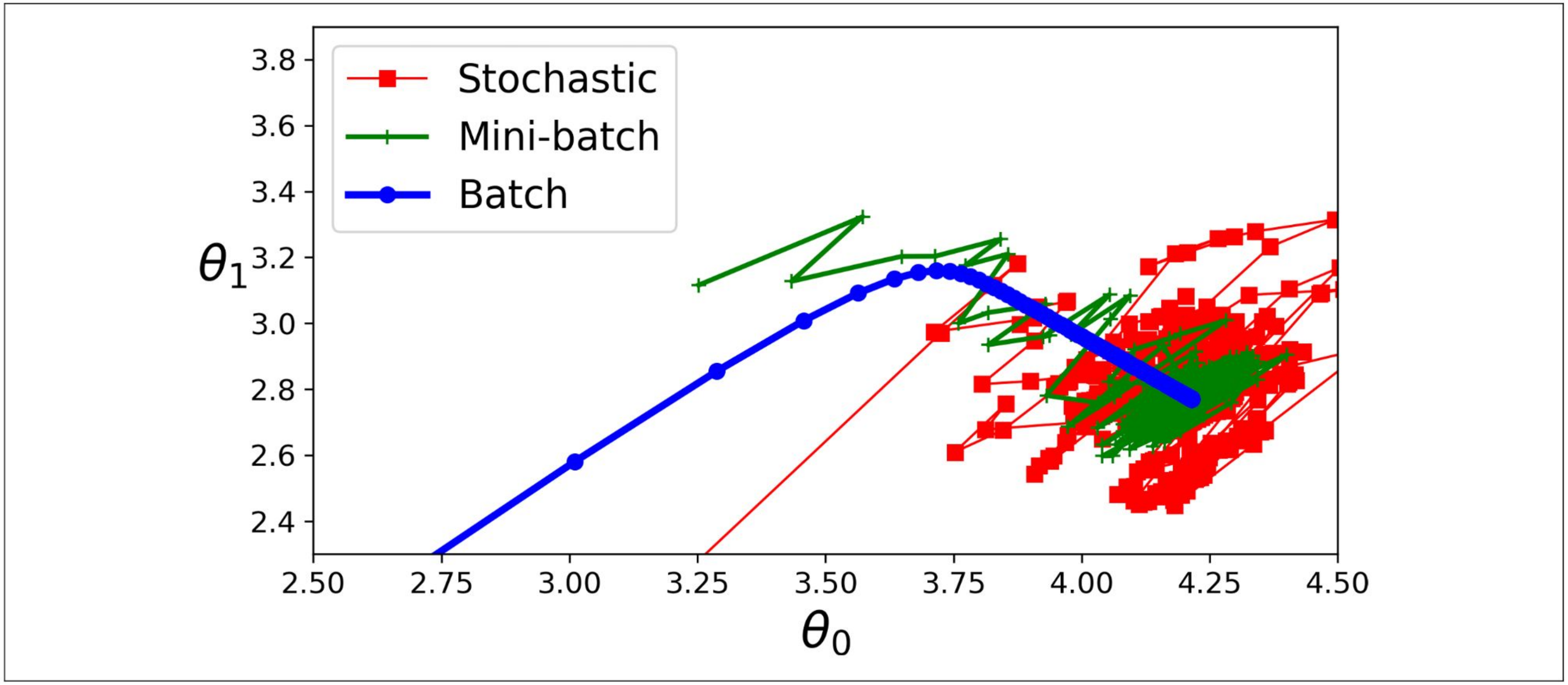


Figure 4-11. Gradient Descent paths in parameter space

Polynomial Regression

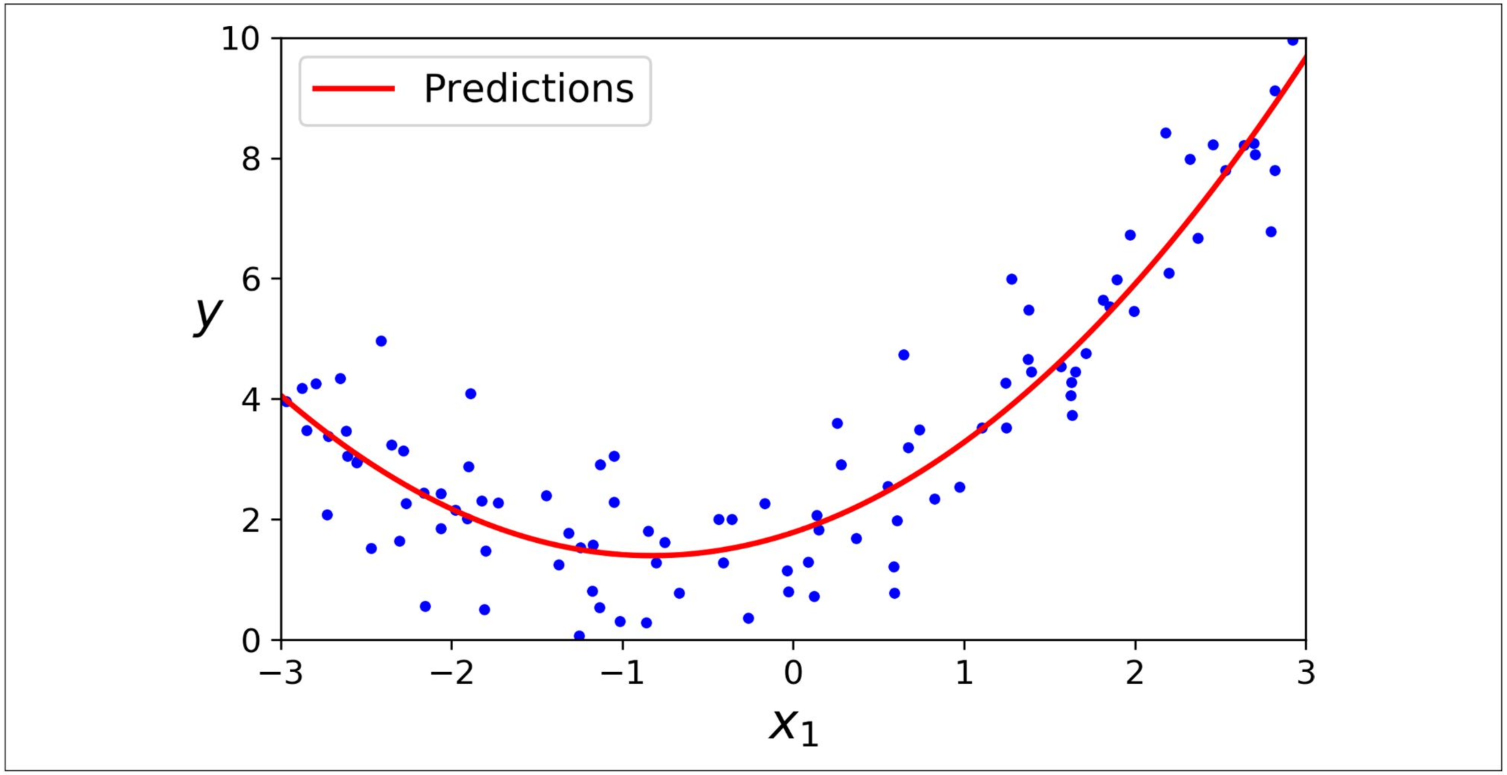


Figure 4-13. Polynomial Regression model predictions

Not bad: the model estimates $\hat{y} = 0.56x_1^2 + 0.93x_1 + 1.78$ when in fact the original function was $y = 0.5x_1^2 + 1.0x_1 + 2.0 +$ Gaussian noise.

Learning Curves

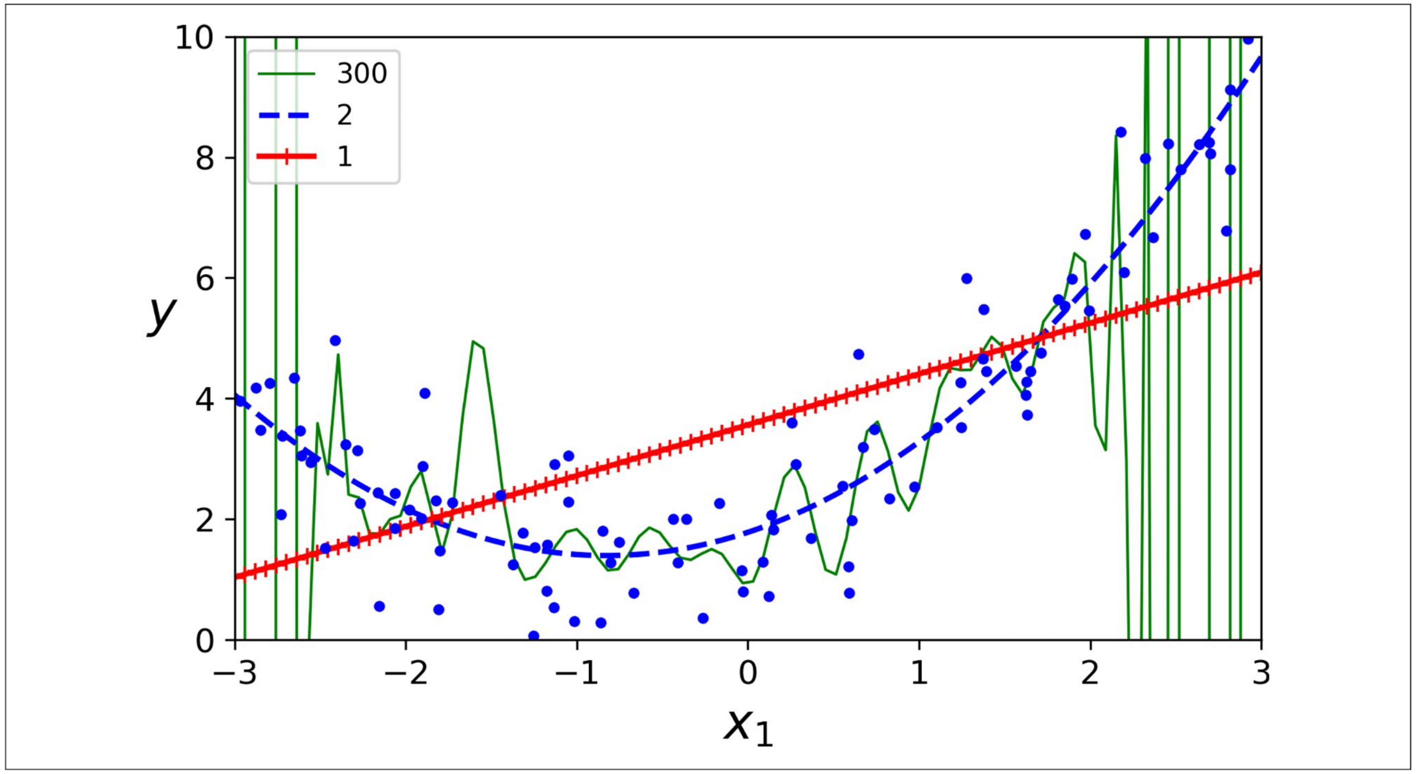
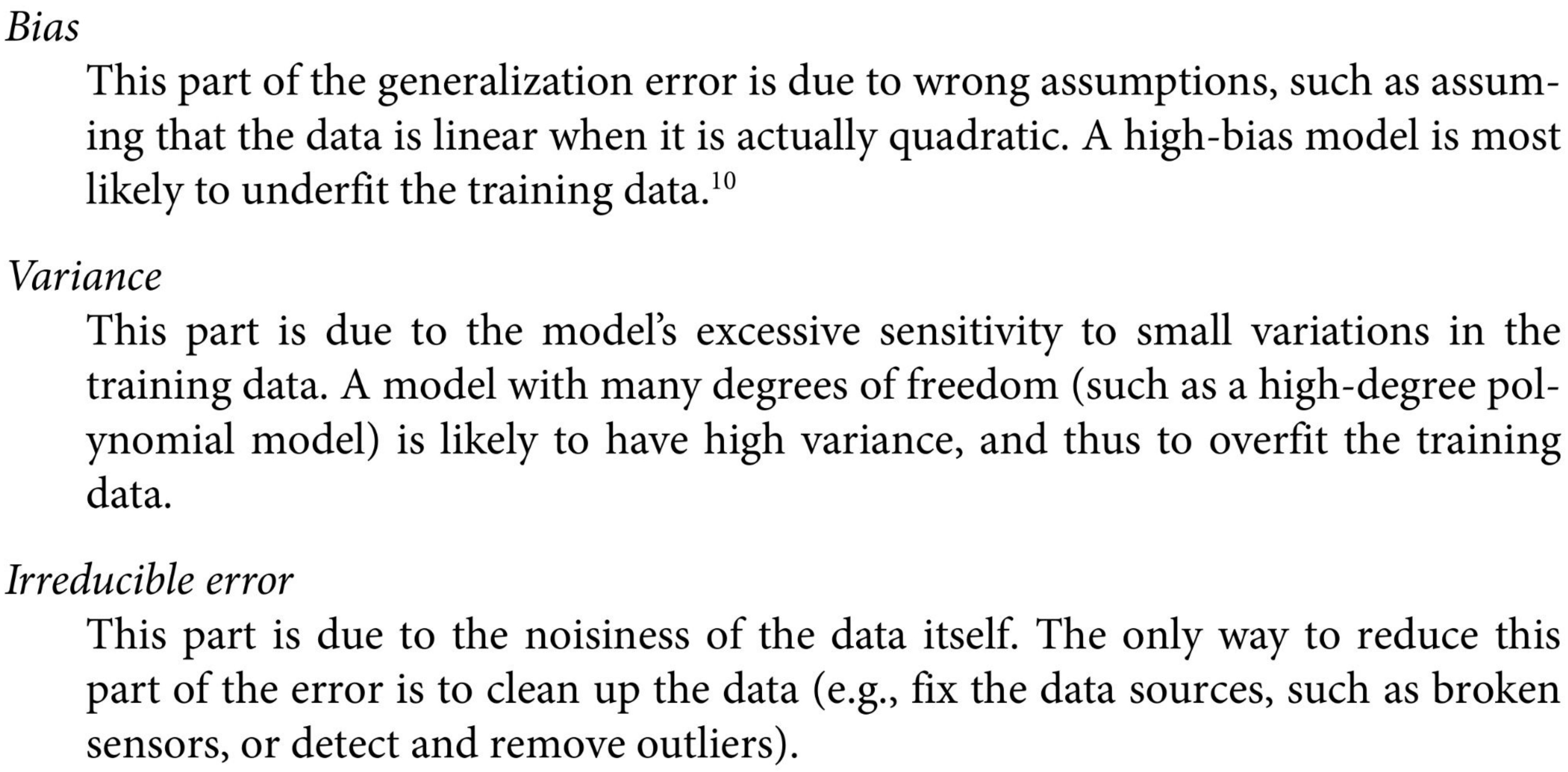


Figure 4-14. High-degree Polynomial Regression

The Bias/Variance Tradeoff



Bias

This part of the generalization error is due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic. A high-bias model is most likely to underfit the training data.[10]

Variance

This part is due to the model's excessive sensitivity to small variations in the training data. A model with many degrees of freedom (such as a high-degree polynomial model) is likely to have high variance, and thus to overfit the training data.

Irreducible error

This part is due to the noisiness of the data itself. The only way to reduce this part of the error is to clean up the data (e.g., fix the data sources, such as broken sensors, or detect and remove outliers).