2. Linear Regression

Outline

Problem setup

Solve the OLS problem

Problem setup

- example: predicting house prices from features
- *features*: size, number of bedrooms, etc.
- *response*: price
- goal: learn a model that predicts price from features
- *training data*: pairs of features and prices

Living area (ft^2)	# Bedrooms	Price ($1000s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Problem setup

- data:
$$\{(x^{(i)}, y^{(i)}) \mid i = 1, \cdots, n\},$$
- $x^{(i)} \in \mathbf{R}^d$ is a *feature* vector
- $y^{(i)} \in \mathbf{R}$ is the *response* variable
- *linear* model (*hypothesis*):
$$h(x; \theta) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d = \theta^T x$$
- $\theta = (\theta_0, \theta_1, \cdots, \theta_d) \in \mathbf{R}^{d+1}$ is the *parameter* vector.
- $x = (1, x_1, \cdots, x_d) \in \mathbf{R}^{d+1}$ is the *augmented* feature vector.
- objective: learning θ^* from the data so that $h(x; \theta^*)$ predicts y *well*.

Loss function

- squared loss:

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h(x^{(i)}, \theta) - y^{(i)} \right)^2$$

- learn θ^* by minimizing $J(\theta)$.
- this is called *ordinary least squares* (OLS) regression model.

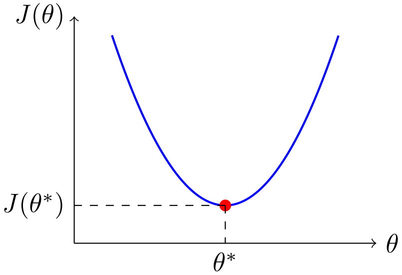
Outline

Problem setup

Solve the OLS problem

Solve the OLS problem

- solve for θ^* that minimizes $J(\theta)$.



- $J(\theta)$ is a *convex* function of θ.
- $J(\theta)$ has a shape like a *bowl* with a single *minimum point*.
- analytical solution
- gradient descent

Analytical solution

- express $J(\theta)$ using matrix notation:

$$J(\theta) = \frac{1}{2n}(X\theta - y)^T(X\theta - y),$$

  where X is the *design matrix* and y is the *response vector*.

- differentiate $J(\theta)$ with respect to θ:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{n}X^T(X\theta - y).$$

- set the derivative to zero and solve for θ^*:

$$\theta^* = (X^T X)^{-1} X^T y.$$

Computing the analytical solution

- it is attempting to directly use $\theta^* = (X^T X)^{-1} X^T y$ to compute θ^*
- not recommended because it involves inverting a matrix
- in practice, direct matrix inversion is rarely used
- alternative methods that are numerically more stable are used
- they often involve matrix factorization techniques
- *QR decomposition*, *SVD*, *Cholesky decomposition*

Computing the analytical solution

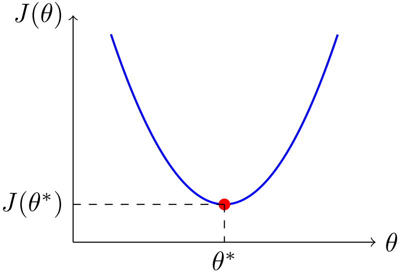
- *QR decomposition*: $X = QR$
- Q is an orthonormal matrix, i.e., $Q^T Q = I$
- R is an upper triangular matrix
- $\theta^* = R^{-1} Q^T y \iff R\theta^* = Q^T y$
-
```
Q, R = jnp.linalg.qr(x)
theta_qr = jax.scipy.linalg.solve_triangular(R, Q.T @ y)
```

Gradient descent

- update rule:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j},$$

  for all $j = 0, 1, \cdots, d$.
- α is the *learning rate*.
- repeat until convergence.



Tutorial

Linear regression tutorial