Lecture Notes:
Linear Regression

CIC1205 - Machine Learning

Eduardo Bezerra

February 26, 2026

0 Contents

1 Introduction

1.1 Regression vs. Classification

Supervised machine learning problems can be broadly divided into two categories: **classification** and **regression**.

In **classification**, the goal is to predict a *discrete* label or category. For example, deciding whether an email is spam or not spam, or identifying which digit appears in an image, are classification problems.

In **regression**, on the other hand, the goal is to predict a *continuous* numerical value. Predicting the price of a house given its size, or forecasting tomorrow's temperature given today's weather data, are examples of regression problems.

Regression

A **regression** task is a supervised learning problem in which the output variable (target) takes continuous numerical values. The goal is to learn a mapping from input features to a real-valued output.

1.2 What Is Linear Regression?

Linear Regression is one of the simplest and most widely used regression algorithms. The core idea is to model the relationship between one or more input variables (called *features*) and a continuous output variable (called the *target*) as a linear function.

In the univariate (single-feature) case, we assume that the target variable y depends approximately linearly on a single feature x. This means we are trying to fit a straight line through the data.

Why start with linear regression? Because:

- It provides an excellent conceptual foundation for understanding more complex models.
- It introduces key ideas such as the cost function and gradient descent that appear throughout machine learning.
- It is fast, interpretable, and surprisingly effective in many practical scenarios.

1.3 The Learning Pipeline

The typical workflow for supervised learning, including linear regression, follows these steps:

1. **Collect a training set**: a set of labelled examples $(x^{(i)}, y^{(i)})$, where $x^{(i)}$ is the feature value of the i-th example and $y^{(i)}$ is the corresponding target.
2. **Choose a hypothesis class**: decide on the form of the model (here, a linear function).
3. **Define a cost function**: a measure of how well (or poorly) the model fits the training data.

4. **Optimise the parameters**: find the values of the model parameters that minimise the cost.
5. **Predict**: apply the learned model to new, unseen inputs.

2 Representation of the Hypothesis

2.1 Notation

Before formalising the model, let us establish the notation that will be used throughout these notes. By simplicity, we will start considering the simplest case of univariate linear regression.

Symbol	Meaning
m	Number of training examples
$x^{(i)}$	Feature value of the i-th training example
$y^{(i)}$	Target value of the i-th training example
$(x^{(i)}, y^{(i)})$	The i-th training example
θ_0, θ_1	Model parameters (intercept and slope)
$h_\theta(x)$	Hypothesis function parameterised by θ

2.2 The Hypothesis Function

For the univariate linear regression model, the hypothesis is:

Hypothesis for Univariate Linear Regression

$$h_\theta(x) = \theta_0 + \theta_1 x \tag{1}$$

This is simply the equation of a straight line, where:

- θ_0 is the **intercept** (bias term), the value of $h_\theta(x)$ when $x = 0$.
- θ_1 is the **slope**, and corresponds to how much the prediction changes for a unit increase in x.

Different choices of θ_0 and θ_1 produce different lines, and therefore different hypotheses. The learning algorithm's task is to find the combination that best fits the data.

Note

The notation $h_\theta(x)$ emphasises that the hypothesis is a function of x *for fixed parameters* θ_0 and θ_1. When we talk about optimisation, we instead think of $J(\theta_0, \theta_1)$ as a function of the parameters.

Entry Ticket 1

Consider the hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$.

1. What is the predicted value $h_\theta(x)$ when $x = 0$? Which parameter does that correspond to?

2. If $\theta_1 = 3$, by how much does the prediction change when x increases by one unit?
3. Two hypotheses are defined by $(\theta_0, \theta_1) = (0, 2)$ and $(\theta_0, \theta_1) = (5, 2)$. They have the same slope. What is different about them geometrically?

3 The Cost Function

3.1 Motivation

We need an *objective* measure of how well a particular choice of parameters (θ_0, θ_1) explains the training data. Intuitively, a good model is one where, for each training example $(x^{(i)}, y^{(i)})$, the prediction $h_\theta(x^{(i)})$ is as close as possible to the actual target $y^{(i)}$.

3.2 Deriving the Mean Squared Error

The difference between the prediction and the actual target for example i is called the **residual**:

$$r^{(i)} = h_\theta(x^{(i)}) - y^{(i)}$$

Simply summing residuals is not useful, because positive and negative errors would cancel out. A natural fix is to square each residual:

$$\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Squaring has the additional desirable property of penalising large errors more heavily than small ones. To get an overall measure of error across all m training examples, we sum the squared residuals:

$$\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Finally, we divide by $2m$. The factor m converts the sum into an average (making the cost independent of the number of examples), and the factor 2 is a mathematical convenience: it cancels nicely with the exponent when we later take derivatives.

Mean Squared Error Cost Function

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \tag{2}$$

This is commonly called the **Mean Squared Error (MSE)** cost function (up to the factor of 2).

Entry Ticket 2

Refer to the derivation of the cost function $J(\theta_0, \theta_1)$.

1. Why is each residual $r^{(i)} = h_\theta(x^{(i)}) - y^{(i)}$ squared before summing? What problem would arise if we summed the residuals directly?
2. The denominator is $2m$, not m. What is the purpose of each factor: what

does dividing by m achieve, and why is the factor of 2 included?
3. If a model produces residuals $r^{(1)} = 3$ and $r^{(2)} = -3$ on a training set of $m = 2$ examples, compute J using the formula above.

3.3 The Optimisation Objective

Our learning goal is now formally stated as:

Optimisation Objective

$$\underset{\theta_0, \theta_1}{\text{minimise}} \quad J(\theta_0, \theta_1) \tag{3}$$

In other words, we seek the values θ_0^* and θ_1^* such that the cost function is as small as possible.

3.4 Geometric Intuition: The Single-Parameter Case

To develop geometric intuition, let us temporarily simplify by setting $\theta_0 = 0$, so the hypothesis becomes:

$$h_\theta(x) = \theta_1 x$$

and the cost reduces to:

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_1 x^{(i)} - y^{(i)}\right)^2$$

Let us expand this:

$$\begin{aligned} J(\theta_1) &= \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_1^2 (x^{(i)})^2 - 2\theta_1 x^{(i)} y^{(i)} + (y^{(i)})^2\right) \\ &= \underbrace{\left(\frac{1}{2m} \sum_{i=1}^{m} (x^{(i)})^2\right)}_{a} \theta_1^2 - \underbrace{\left(\frac{1}{m} \sum_{i=1}^{m} x^{(i)} y^{(i)}\right)}_{|b|} \theta_1 + \underbrace{\left(\frac{1}{2m} \sum_{i=1}^{m} (y^{(i)})^2\right)}_{c} \\ &= a\theta_1^2 + b\theta_1 + c \end{aligned} \tag{4}$$

This is a **quadratic (parabolic) function** of θ_1, with $a > 0$ (since it is a sum of squares). Such a parabola has a unique global minimum, which occurs at the vertex:

$$\theta_1^* = -\frac{b}{2a}$$

Geometrically: on the left, we have the scatter plot of training data together with the regression line for various θ_1; on the right, we have the corresponding value of $J(\theta_1)$ plotted as a curve. As θ_1 moves from 0 toward the optimal value, the line fits the data better and the point on the cost curve moves downward toward the minimum.

Entry Ticket 3

In the single-parameter case, the cost function takes the form $J(\theta_1) = a\theta_1^2 + b\theta_1 + c$.

1. What condition on a ensures that J has a *minimum* rather than a maximum? Is that condition satisfied here? Why?
2. What is the geometric shape of $J(\theta_1)$ as a function of θ_1?
3. As θ_1 approaches the optimal value θ_1^*, what happens to the slope of J? What does that imply for the size of each gradient descent step?

3.5 The Two-Parameter Case

When both θ_0 and θ_1 are free, the cost function $J(\theta_0, \theta_1)$ becomes a surface over the (θ_0, θ_1) plane.

Because the cost function is quadratic in both parameters, this surface has the shape of a **paraboloid**, a smooth bowl with a single global minimum.

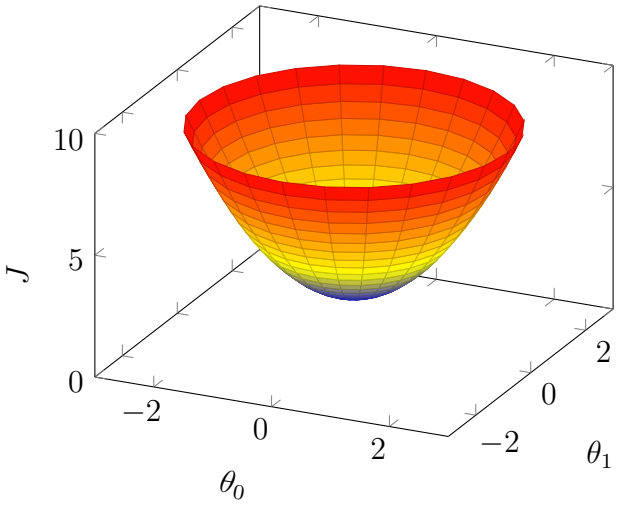

Figure 1: The cost surface $J(\theta_0, \theta_1)$ has the shape of a paraboloid. Each point represents a parameter pair, and the height represents the prediction error. The global minimum lies at the bottom of the bowl.

Figure 1 illustrates this geometry. Each point on the surface corresponds to a particular choice of parameters (θ_0, θ_1), and the height of the surface represents the cost J.

- Points high on the surface correspond to poor hypotheses (large prediction errors).
- Points lower on the surface correspond to better hypotheses.
- The bottom of the bowl corresponds to the optimal parameters, where the cost is minimal.

A useful way to understand this surface is to imagine slicing the bowl horizontally at different heights. Each slice produces a closed curve in the (θ_0, θ_1) plane.

These curves are the **level curves** (or contour lines) of J: they connect all parameter pairs that yield the same cost.

Because the surface is quadratic, these level curves are ellipses, all centred on the global minimum.

Thus, learning in linear regression can be interpreted geometrically: finding the best parameters means navigating the paraboloid surface until we reach its lowest point — the centre of the innermost ellipse.

3.6 Examples of Different Hypotheses

To build intuition, consider the small training set in Table 1, which records the size (in m^2) and selling price (in thousands of euros) of twelve houses.

Table 1: Training set: house sizes and prices (12 examples).

i	$x^{(i)}$ **(m²)**	$y^{(i)}$ **(k€)**
1	40	95.0
2	55	97.0
3	65	127.0
4	75	120.0
5	85	145.0
6	95	142.0
7	105	170.0
8	115	155.0
9	130	188.0
10	145	183.0
11	160	216.0
12	180	223.0

Three candidate hypotheses are described below, together with a visual description of what each one looks like on a scatter plot: see Figure 2.

- $h_\theta(x) = 0 + 0.5\,x$: a line through the origin with a gentle slope.Predictions are far too low for all examples (e.g. it predicts $h_\theta(40) = 20$ k€ against an actual price of 95 k€).
- $h_\theta(x) = 50 + 1.0\,x$: a line with a unit slope and a non-zero intercept. Predictions track the data closely (e.g. $h_\theta(40) = 90$ k€, $h_\theta(180) = 230$ k€).
- $h_\theta(x) = 100 + 2.0\,x$: a steep line shifted upwards. Predictions are too high (e.g. $h_\theta(40) = 180$ k€ against an actual price of 95 k€).

We can compute the cost $J(\theta_0, \theta_1)$ from Equation (2) for each candidate:

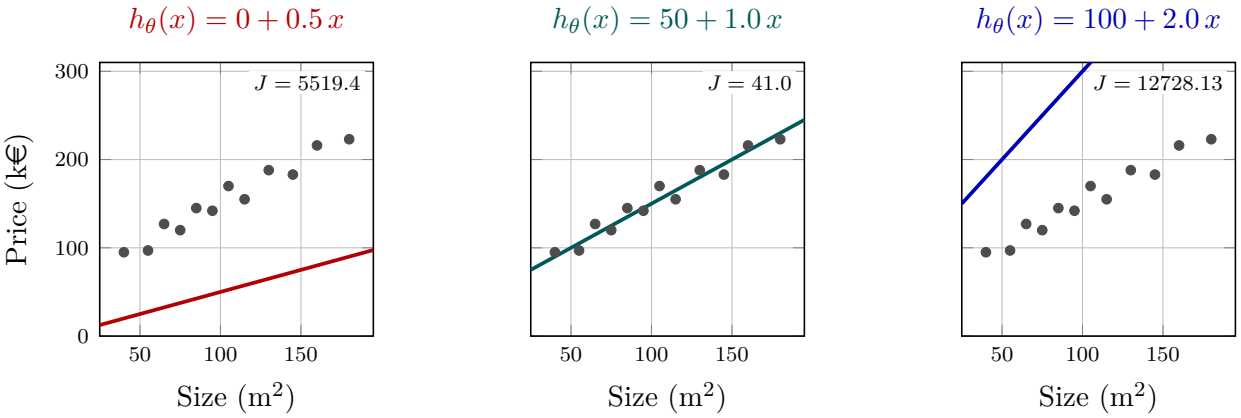

Figure 2: Three candidate hypotheses plotted against the 12 training examples (house size vs. price). Training data points are shown as filled circles. The second hypothesis ($\theta_0 = 50$, $\theta_1 = 1.0$) achieves a cost more than 100× lower than the first and nearly 800× lower than the third.

Hypothesis	$(\theta_0,\ \theta_1)$	$J(\theta_0, \theta_1)$
$h_\theta(x) = 0 + 0.5\,x$	$(0,\ 0.5)$	5519.38
$h_\theta(x) = 50 + 1.0\,x$	$(50,\ 1.0)$	41.04
$h_\theta(x) = 100 + 2.0\,x$	$(100,\ 2.0)$	12728.13

The second hypothesis achieves a cost more than **134 times smaller** than the first, and nearly **310 times smaller** than the third. This confirms quantitatively what our visual inspection suggested: $(\theta_0, \theta_1) = (50, 1.0)$ is by far the best of these three candidates.

Of course, there may exist even better parameter values than $(50, 1.0)$; the *exact* minimiser of J over the entire parameter space. We need a principled, automated way to find it. This is precisely the role of the **cost function minimisation** problem introduced in the next section.

3.7 The Cost Function Instantiated on the Toy Dataset

The general cost function $J(\theta_0, \theta_1)$ is an abstract expression. Let us make it fully concrete by substituting the 12 training examples from Table 1. The result is a specific real-valued function whose *only free variables* are the parameters θ_0 and θ_1; every other number comes directly from the data.

Step 1: Write out every squared residual

Substituting $x^{(i)}$ and $y^{(i)}$ into $\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ gives one term per training example:

$$J(\theta_0,\theta_1)=\frac{1}{24}\begin{bmatrix}(\theta_0+40\,\theta_1-95)^2\\+(\theta_0+55\,\theta_1-97)^2\\+(\theta_0+65\,\theta_1-127)^2\\+(\theta_0+75\,\theta_1-120)^2\\+(\theta_0+85\,\theta_1-145)^2\\+(\theta_0+95\,\theta_1-142)^2\\+(\theta_0+105\,\theta_1-170)^2\\+(\theta_0+115\,\theta_1-155)^2\\+(\theta_0+130\,\theta_1-188)^2\\+(\theta_0+145\,\theta_1-183)^2\\+(\theta_0+160\,\theta_1-216)^2\\+(\theta_0+180\,\theta_1-223)^2\end{bmatrix} \tag{5}$$

The denominator $24 = 2m = 2 \times 12$ comes from the definition of the cost function. Observe that θ_0 and θ_1 appear in *every* term; all other numbers ($40, 55, \ldots$ and $95, 97, \ldots$) are fixed constants read from the dataset.

Step 2: Expand and collect like terms

Each squared binomial $(\theta_0 + c_1\,\theta_1 + c_2)^2$ expands into $\theta_0^2 + c_1^2\,\theta_1^2 + c_2^2 + 2c_1\,\theta_0\theta_1 + 2c_2\,\theta_0 + 2c_1c_2\,\theta_1$. Summing over all 12 terms and collecting by monomial yields:

$$J(\theta_0,\theta_1)=\frac{1}{24}\Big[\underbrace{12}_{\sum 1}\,\theta_0^2+\underbrace{150{,}900}_{\sum x_i^2}\,\theta_1^2+\underbrace{2{,}500}_{2\sum x_i}\,\theta_0\theta_1-\underbrace{3{,}722}_{2\sum y_i}\,\theta_0-\underbrace{427{,}110}_{2\sum x_iy_i}\,\theta_1+\underbrace{308{,}295}_{\sum y_i^2}\Big] \tag{6}$$

where the annotations beneath each coefficient identify the dataset quantity it comes from:

Dataset quantity	**Value**	**Role in J**
m	12	number of examples
$\sum_{i=1}^{m} 1 = m$	12	coeff. of θ_0^2 (before $/2m$)
$\sum_{i=1}^{m} x_i^2$	150,900	coeff. of θ_1^2 (before $/2m$)
$2\sum_{i=1}^{m} x_i$	2,500	coeff. of $\theta_0\theta_1$ (before $/2m$)
$2\sum_{i=1}^{m} y_i$	3,722	coeff. of θ_0 (before $/2m$, negative)
$2\sum_{i=1}^{m} x_iy_i$	427,110	coeff. of θ_1 (before $/2m$, negative)
$\sum_{i=1}^{m} y_i^2$	308,295	constant term (before $/2m$)

Step 3: Simplified polynomial form

Dividing each coefficient by $2m = 24$ gives the final, fully numerical expression:

Cost Function: Explicit Polynomial Form

$$\boxed{J(\theta_0, \theta_1) = \frac{1}{2}\theta_0^2 + \frac{12{,}575}{2}\theta_1^2 + \frac{625}{6}\theta_0\theta_1 - \frac{1{,}861}{12}\theta_0 - \frac{71{,}185}{4}\theta_1 + \frac{102{,}765}{8}} \quad (7)$$

or equivalently in decimal form:

$$J(\theta_0, \theta_1) \approx 0.5\,\theta_0^2 + 6{,}287.5\,\theta_1^2 + 104.17\,\theta_0\theta_1 - 155.08\,\theta_0 - 17{,}796.25\,\theta_1 + 12{,}845.63 \quad (8)$$

What this expression tells us

Equation (7) is a **quadratic polynomial** in θ_0 and θ_1. Every coefficient is a fixed number computed once from the data; after that computation, J is purely a function of the two parameters. This is the surface we are trying to navigate downhill with gradient descent.
The fact that both θ_0^2 and θ_1^2 appear with *positive* coefficients ($\frac{1}{2}$ and $\frac{12{,}575}{2}$ respectively) confirms that the surface is a **convex paraboloid**, a bowl with a unique global minimum.

Sanity check

We can verify Equation (7) against Table 1 by substituting the three candidate parameter pairs:

$$J(0,\ 0.5) = \tfrac{1}{2}(0)^2 + \tfrac{12{,}575}{2}(0.25) + \tfrac{625}{6}(0)(0.5) - \tfrac{1{,}861}{12}(0) - \tfrac{71{,}185}{4}(0.5) + \tfrac{102{,}765}{8} \approx 5{,}519.4\checkmark$$

$$\begin{aligned} J(50,\ 1.0) &= \tfrac{1}{2}(50^2) + \tfrac{12{,}575}{2}(1^2) + \tfrac{625}{6}(50 \cdot 1) - \tfrac{1{,}861}{12}(50) - \tfrac{71{,}185}{4}(1) + \tfrac{102{,}765}{8} \\ &= 1{,}250 + \tfrac{12{,}575}{2} + \tfrac{15{,}625}{3} - \tfrac{46{,}525}{6} - \tfrac{71{,}185}{4} + \tfrac{102{,}765}{8} = \tfrac{985}{24} \approx 41.0\checkmark \end{aligned}$$

$$J(100,\ 2.0) = \tfrac{1}{2}(100^2) + \tfrac{12{,}575}{2}(2^2) + \tfrac{625}{6}(100 \cdot 2) - \tfrac{1{,}861}{12}(100) - \tfrac{71{,}185}{4}(2) + \tfrac{102{,}765}{8} \approx 11{,}074.0\checkmark$$

All three values match those computed directly from the training data, confirming that Equation (7) is correct.

4 Parameter Learning: Gradient Descent

4.1 The Big Picture

We now turn to the question of *how* to minimise $J(\theta_0, \theta_1)$ algorithmically. For the two-parameter linear regression case, an analytical solution exists (the Normal Equation), but gradient descent is the approach that scales to models with millions of parameters (such as deep neural networks), making it the most important optimisation technique to understand.

Gradient Descent is an iterative algorithm that starts from an initial guess for the parameters and repeatedly updates them in the direction that most steeply decreases the cost function.

4.2 The Algorithm

Gradient Descent Update Rule

Repeat until convergence:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{for } j = 0, 1 \tag{9}$$

where all θ_j must be updated **simultaneously**.

Let us unpack each component:

- $\alpha > 0$ is the **learning rate**, a hyperparameter that controls the step size. A small α leads to slow but reliable convergence; a large α can cause the algorithm to overshoot and diverge.
- $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ is the **partial derivative** of the cost with respect to θ_j. This is the slope of the cost surface in the θ_j direction at the current point.
- The minus sign ensures we move in the direction of *decreasing* cost (downhill).

Why Simultaneous Updates?

It is crucial that *both* parameters are updated simultaneously, using the values of θ_0 and θ_1 *before* the update step. Updating θ_0 first and then using the new θ_0 to update θ_1 would introduce an asymmetry and yield a different algorithm. In practice, this is implemented using temporary variables:

$$\begin{aligned} \text{temp0} &:= \theta_0 - \alpha \frac{\partial J}{\partial \theta_0} \\ \text{temp1} &:= \theta_1 - \alpha \frac{\partial J}{\partial \theta_1} \\ \theta_0 &:= \text{temp0} \\ \theta_1 &:= \text{temp1} \end{aligned}$$

Entry Ticket 4

Refer to the gradient descent update rule $\theta_j \leftarrow \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$.

1. What happens to the step size if α is very large? And if α is very small?
2. Suppose $\frac{\partial J}{\partial \theta_1} < 0$ at the current point. Does the update increase or decrease θ_1? Is that the correct direction to move toward the minimum?
3. Why must θ_0 and θ_1 be updated simultaneously rather than sequentially?

4.3 Intuition for the Single-Parameter Case

Consider minimising $J(\theta_1)$ (the single-parameter version). The update rule is:

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

There are three cases to consider:

1. **Derivative is negative** ($\frac{d}{d\theta_1} J < 0$): The cost function slopes downward at the current θ_1. Subtracting a negative number *increases* θ_1, moving it to the right (toward the minimum).
2. **Derivative is positive** ($\frac{d}{d\theta_1} J > 0$): The cost function slopes upward. Subtracting a positive number *decreases* θ_1, moving it to the left (again toward the minimum).
3. **Derivative is zero** ($\frac{d}{d\theta_1} J = 0$): We are already at the minimum (or a saddle point). The parameter does not change; the algorithm has converged.

Additionally, as we approach the minimum, the gradient naturally becomes smaller, so the step size automatically decreases, even with a fixed learning rate.

4.4 The Gradient in Higher Dimensions

When the model has $n+1$ parameters $(\theta_0, \theta_1, \ldots, \theta_n)$, the partial derivatives form a vector called the **gradient**:

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix}$$

The gradient points in the direction of steepest *ascent* of J. Therefore, to descend (minimise), we move in the direction of $-\nabla J$. The gradient descent update in vector form is:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla J(\boldsymbol{\theta})$$

Geometrically, imagine standing on a mountainous terrain (the cost surface) and always taking a step in the direction of steepest downhill slope. Eventually, you reach a valley, which is the minimum.

5 Gradient Descent for Linear Regression

5.1 Computing the Partial Derivatives

We now apply gradient descent specifically to linear regression by computing the partial derivatives of the MSE cost function.

Recall:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)^2$$

Derivative with respect to θ_0:

Using the chain rule:

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{2m} \sum_{i=1}^{m} 2 \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot \frac{\partial}{\partial \theta_0} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

Derivative with respect to θ_1:

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

5.2 The Update Rules

Substituting the partial derivatives into the gradient descent update rule yields the concrete algorithm for linear regression:

Gradient Descent for Linear Regression (Univariate)

Repeat until convergence:

$$\theta_0 \leftarrow \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \tag{10}$$

$$\theta_1 \leftarrow \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \tag{11}$$

(updates applied simultaneously)

Entry Ticket 5

Refer to the gradient descent update rules for linear regression (Equations (10) and (11)).

1. The update for θ_0 sums the residuals $(h_\theta(x^{(i)}) - y^{(i)})$ over all examples. The update for θ_1 sums the same residuals *multiplied by* $x^{(i)}$. Where does that extra $x^{(i)}$ come from? Refer to the partial derivative computation.
2. Suppose all residuals are positive, meaning the model systematically over-predicts. Will the updates increase or decrease θ_0 and θ_1? Does that make intuitive sense?
3. Why are these update rules called *Batch* Gradient Descent specifically?

5.3 Why This Is Called "Batch" Gradient Descent

Notice that each update uses *all* m training examples (the full batch). This variant is therefore called **Batch Gradient Descent**. Other variants include:

- **Stochastic Gradient Descent (SGD)**: uses one example at a time; faster per iteration but noisier.
- **Mini-batch Gradient Descent**: uses a small subset of examples; balances speed and stability.

5.4 Convergence Properties

An important property of the MSE cost function for linear regression is that it is **convex**: the paraboloid bowl has no local minima, only a single global minimum. This means:

- Gradient descent is *guaranteed* to converge to the global minimum (for a suitably small learning rate α).
- The final solution is the same regardless of the initialisation of θ_0 and θ_1.

This is in contrast to non-convex cost functions (e.g., those arising in neural networks), where gradient descent may get trapped in local minima or saddle points.

6 Summary

The table below summarises the key components of the linear regression model and the gradient descent algorithm.

Component	**Expression**
Hypothesis	$h_\theta(x) = \theta_0 + \theta_1 x$
Cost function	$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$
Objective	$\underset{\theta_0, \theta_1}{\text{minimise}}\ J(\theta_0, \theta_1)$
GD update (θ_0)	$\theta_0 \leftarrow \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$
GD update (θ_1)	$\theta_1 \leftarrow \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$

Key Takeaways

1. Linear regression models the relationship between a feature x and a continuous target y as a straight line.
2. The model parameters θ_0 (intercept) and θ_1 (slope) are learned by minimising the Mean Squared Error cost function.
3. The cost function for linear regression is convex (a paraboloid), guaranteeing a unique global minimum.
4. Gradient descent is an iterative algorithm that updates the parameters in the direction of steepest descent of the cost function.
5. The learning rate α controls the step size and must be chosen carefully.

A Appendix: Deriving the Gradient Descent Updates from Scratch

This appendix provides the full derivation of Equations (10) and (11) for completeness.

We wish to compute $\frac{\partial J}{\partial \theta_j}$ for $j \in \{0, 1\}$. Define:

$$f_i(\theta_0, \theta_1) = h_\theta(x^{(i)}) - y^{(i)} = \theta_0 + \theta_1 x^{(i)} - y^{(i)}$$

Then:

$$J = \frac{1}{2m} \sum_{i=1}^{m} [f_i]^2$$

By the chain rule:

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{2m} \sum_{i=1}^{m} 2 f_i \cdot \frac{\partial f_i}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} f_i \cdot \frac{\partial f_i}{\partial \theta_j}$$

For $j = 0$:

$$\frac{\partial f_i}{\partial \theta_0} = \frac{\partial}{\partial \theta_0}(\theta_0 + \theta_1 x^{(i)} - y^{(i)}) = 1$$

$$\Rightarrow \frac{\partial J}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

For $j = 1$:

$$\frac{\partial f_i}{\partial \theta_1} = \frac{\partial}{\partial \theta_1}(\theta_0 + \theta_1 x^{(i)} - y^{(i)}) = x^{(i)}$$

$$\Rightarrow \frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

These are exactly the expressions used in Equations (10) and (11). □