ML models
Linear regression (10 min)
Linear regression (10 min)
Loss (10 min)
Interactive exercise: Loss (5 min)
Gradient descent (10 min)
Hyperparameters (10 min)
Interactive exercise: Gradient descent (5 min)
Programming exercise (10 min)
Test your knowledge (10 min)
What's next
Logistic regression (20 min)
Classification (20 min)
Data
Working with numerical data (60 min)
Working with categorical data (30 min)

Linear regression: Gradient descent

On this page
Click the plus icon to learn more about the math behind gradient descent
Model convergence and loss curves
Exercise: Check your understanding
Convergence and convex functions

Gradient descent is a mathematical technique that iteratively finds the weights and bias that produce the model with the lowest loss. Gradient descent finds the best weight and bias by repeating the following process for a number of user-defined iterations.

The model begins training with randomized weights and biases near zero, and then repeats the following steps:

1. Calculate the loss with the current weight and bias.
2. Determine the direction to move the weights and bias that reduce loss.
3. Move the weight and bias values a small amount in the direction that reduces loss.
4. Return to step one and repeat the process until the model can't reduce the loss any further.

The diagram below describe how iterative steps gradient descent performs to find the weights and bias that produce the model with the lowest loss.

Figure 12. Gradient descent is an iterative process that finds the weights and bias that produce the model with the lowest loss.

Click the plus icon to learn more about the math behind gradient descent.

At a concrete level, we can walk through the gradient descent steps using a small dataset with seven examples for a car's fuelness in pounds and its miles per gallon rating:

Pounds in 1000s (feature)	Miles per gallon (label)
3.3	18
3.69	15
3.44	18
3.43	16
4.34	15
4.42	14
2.37	24

1. The model starts training by setting the weight and bias to zero:

$$Weight = 0$$
$$Bias = 0$$
$$y = 0 + 0(x_1)$$

2. Calculates MSE loss with the current model parameters:

$$Loss = \frac{(24-0)^2 + (15-0)^2 + (18-0)^2 + (16-0)^2 + (15-0)^2 + (14-0)^2 + (24-0)^2}{7}$$
$$Loss = 303.71$$

3. Calculate the slope of the tangent to the loss function at each weight and the bias:

$$Weight\ slope = -119.7$$
$$Bias\ slope = -34.3$$

> **Click the plus icon to learn about calculating slope.**
>
> To get the slope for the line tangent to the weight and bias, we take the derivative of the loss function with respect to the weight and the bias, and then solve the equations.
>
> We'll write the equation for making a prediction as:
>
> $f_{w,b}(x) = (w * x) + b$
>
> We'll write the actual value as y.
>
> We'll calculate MSE using:
>
> $\frac{1}{M}\sum_{(x,y)\in D}(f_{w,b}(x) - y)^2$
>
> where Σ represents the sum of all training example and M represents the number of examples.
>
> **Weight derivative**
>
> The derivative of the loss function with respect to the weight is written as:
>
> $\frac{1}{M}\sum_{(x,y)\in D}(f_{w,b}(x) - y)^2 * 2x_j$
>
> and evaluates to:
>
> $\frac{1}{M}\sum_{(x,y)\in D}(f_{w,b}(x) - y)^2 * 2x_j$
>
> First we sum each predicted value minus the actual value and then multiply it by two times the feature value. Then we divide the sum by the number of examples. The result is the slope of the line tangent to the value of the weight.
>
> If we solve this equation with a weight and bias equal to zero, we get: 119.7 for the lines slope.
>
> **Bias derivative**
>
> The derivative of the loss function with respect to the bias is written as:
>
> $\frac{1}{M}\sum_{(x,y)\in D}(f_{w,b}(x) - y)^2$
>
> and evaluates to:
>
> $\frac{1}{M}\sum_{(x,y)\in D}(f_{w,b}(x) - y)^2 * 2$
>
> First we sum each predicted value minus the actual value and then multiply it by two. Then we divide the sum by the number of examples. The result is the slope of the line tangent to the value of the bias.
>
> If we solve this equation with a weight and bias equal to zero, we get: 34.3 for the line's slope.

4. Move a small amount in the direction of the negative slope to get the next weight and bias. For now, we'll arbitrarily define the "small amount" as 0.01:

$$New\ weight = old\ weight - (small\ amount * weight\ slope)$$
$$New\ bias = old\ bias - (small\ amount * bias\ slope)$$
$$New\ weight = 0 - (0.01 * -119.7)$$
$$New\ bias = 0 - (0.01 * -34.3)$$
$$New\ weight = 1.2$$
$$New\ bias = 0.34$$

Use the new weight and bias to calculate the loss and repeat. Completing the process for six iterations, we'll get the following weights, biases, and losses:

Iteration	Weight	Bias	Loss (MSE)
1	0	0	303.71
2	1.20	0.34	170.84
3	2.05	0.59	122.17
4	2.66	0.79	98.70
5	3.09	0.91	91.13
6	3.40	1.01	42.17

You can see that the loss gets lower with each updated weight and bias. In this example, we stopped after six iterations. In practice, a model trains until it **converges**. When a model converges, additional iterations don't reduce loss more because gradient descent has found the weights and bias that nearly minimize the loss.

If the model continues to train past convergence, loss begins to fluctuate in small amounts as the model continually updates the parameters around their lowest values. This can make it hard to verify that the model has actually converged. To confirm the model has converged, you'll want to continue training until the loss has stabilized.

Model convergence and loss curves

When training a model, you'll often look at a **loss curve** to determine if the model has **converged**. The loss curve shows how the loss changes as the model trains. The following is what a typical loss curve looks like. Loss is on the y-axis and iterations are on the x-axis:

Figure 13. Loss curve showing the model converging around the 1,000th iteration.

You can see that loss dramatically decreases during the first few iterations, then gradually decreases before flattening out around the 1,000th iteration mark. After 1,000 iterations, we can be mostly certain that the model has converged.

In the following figures, we show the model at three points during the training process: the beginning, the middle, and the end. Visualizing the model's state at snapshots during the training process solidifies the link between updating the weights and bias, reducing loss, and model convergence.

In the figures, we use the derived weights and bias at a particular iteration to represent the model. In the graph with the training data, we plot the model's predictions. Blue loss lines from the model's data points show the amount of loss. The longer the line, the more inaccurate the model's prediction.

In the following figure, we can see that around the second iteration the model would not be good at making predictions because of the high amount of loss.

Figure 14. Loss curve and snapshot of the model at the beginning of the training process.

At around the 400th iteration, we can see that gradient descent has found the weight and bias that produce a better model.

Figure 15. Loss curve and snapshot of model about midway through training.

And at around the 1,000th iteration, we can see that the model has converged, producing a model with the lowest possible loss.

Figure 16. Loss curve and snapshot of the model near the end of the training process.

Exercise: Check your understanding

What's the role of gradient descent in linear regression?

- Gradient descent removes outliers from the dataset to help the model make better predictions.
- Gradient descent helps to determine what type of loss to use when training a model, for example, L_1 or L_2.
- Gradient descent is an iterative process that finds the best weights and bias that minimize the loss. ✓

Correct answer.

Convergence and convex functions

The loss functions for linear models always produce a **convex** surface. As a result of this property, when a linear regression model converges, we know the model has found the weights and bias that produce the lowest loss.

If we graph the loss surface for a model with one feature, we can see its convex shape. The following is the loss surface for a hypothetical miles per gallon dataset. Weight is on the x-axis, bias is on the y-axis, and loss is on the z-axis:

Figure 17. Loss surface that shows its convex shape.

In this example, a weight of -5.44 and bias of 35.94 produce the lowest loss at 5.54:

Figure 18. Loss surface showing the weight and bias values that produce the lowest loss.

A linear model converges when it's found the minimum loss. Therefore, additional iterations only cause gradient descent to move the weight and bias values in very small amounts around the minimum. If we graphed the weights and bias during gradient descent, the points would look like a ball rolling down a hill. Finally stopping at the point where there's no more downward slope.

Figure 19. Loss graph showing gradient descent points stopping at the lowest point on the graph.

Notice that the black loss points create the exact shape of the loss curve: a steep decline before gradually sloping down until they've reached the lowest point on the loss surface.

It's important to note that the model almost never finds the exact minimum for each weight and bias, but instead finds a value very close to it. It's also important to note that the minimum for the weights and bias don't correspond to zero loss, only a value that produces the lowest loss for that parameter.

Using the weight and bias values that produce the lowest loss—in this case a weight of -5.44 and a bias of 35.94—we can graph the model to see how well it fits the data:

Figure 20. Model graphed using the weight and bias values that produce the lowest loss.

This would be the best model for this dataset because no other weight and bias values produce a model with lower loss.

Stay learned
Convergence
Convex function
Gradient descent
Iteration
Loss curve
Help Center
Previous
Interactive exercise: Parameters (5 min)
Next
Hyperparameters (10 min)
Was this helpful?
Send feedback

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License, and code samples are licensed under the Apache 2.0 License. For details, see the Google Developers Site Policies. Java is a registered trademark of Oracle and/or its affiliates.
Last updated 2026-04-24 UTC.