The data in this exercise is basically linear data plus noise. If we use a model that is too complicated, such as one with too many crosses, we give it the opportunity to fit to the noise in the training data, often at the cost of making the model perform badly on test data.

PLAYGROUND

Cada feature que metemos en playgroun es como un plano, x1,x2,coste. Naranja por debajo de 0 y azul por encima por ejemplo.

La función de coste suma estos planos.

Los pesos dice como de abruptas las pendientes en los planos son, si sube una parabola de plano hasta 7 o hasta 2. Esto influye al sumar los planos en a que plano se debe la forma predominante. Es decir, si tiene un peso grande lo que esta haciendo es ajustar mucho mi modelo a como son los datos justo para esa feature, haciendo overfiting y dejando la relación con otras features un poco de lado.

Regularizacion L2 intenta reducir la complejidad bajando el tamaño en general de los pesos, pero con el cuadrado del peso, es decir, penaliza mas a pesos grandes y menos a pequeños, pero a pesar de eso, el mas grande seguirá siendo el mas grande. Sin embargo las sumas de planos no darán lugar a precipicios, sino a formas mas suaves, es decir, intentaran que se ajusten menos a los datos.

Regularization: not trusting your examples too much. To avoid overfitting.

Several types of regularization, stop training when test loss starts increasing that’s also regularization.

L2 regularization

Don’t end up with weights bigger than they should be

We add a term in the loss function, this term doesn’t depend on the data, this term is the model complexity🡪 I want a simple model. Lambda is how we balance what we want, it depends on your problem, bigger if we want to penalize a lot model complexity:

If our training data are statistically equal distributed as our test data, i.e. we have lots and lots of training data, we don’t need regularization lambda=1e-6. But if we don’t have much training data or they are different, regularization is needed to avoid overfitting, and regularization might be tuned with cross validation or with a separate test set.

Each weight is updated with value of last time – algo proporcional al error - 2\*weight it had last time. Asi que el valor del tamaño del peso afecta por 2 el valor.

Model complexity:

* as a function of the weights of all the features in the model: a feature weight with a high absolute value is more complex
* as a function of the total number of features with nonzero weights

Encourages weight values toward 0 (but not exactly 0)

Encourages the mean of the weights toward 0, with a normal (bell-shaped or Gaussian) distribution.

* If your lambda value is too high, your model will be simple, but you run the risk of underfitting your data. Your model won't learn enough about the training data to make useful predictions.
* If your lambda value is too low, your model will be more complex, and you run the risk of overfitting your data. Your model will learn too much about the particularities of the training data, and won't be able to generalize to new data.

The ideal value of lambda produces a model that generalizes well to new, previously unseen data. Unfortunately, that ideal value of lambda is data-dependent, so you'll need to do some tuning.

Important: when using L2 regularization While test loss decreases, training loss actually increases. This is expected, because you've added another term to the loss function to penalize complexity. Ultimately, all that matters is test loss, as that's the true measure of the model's ability to make good predictions on new data.

L2 regularization encourages weights to be near 0.0, but not exactly 0.0.

Imagine a linear model with two strongly correlated features; that is, these two features are nearly identical copies of one another but one feature contains a small amount of random noise. If we train this model with L2 regularization, what will happen to the weights for these two features?

Both features will have roughly equal, moderate weights.

L2 regularization will force the features towards roughly equivalent weights that are approximately half of what they would have been had only one of the two features been in the model.

When the error that the two features introduce are more or less the same, L2 will give them the same weight more or less, because the loss function are compensated:

Feature without noise: in the loss function the term of its error will be slightly less error, and because of that, its weight will be bigger, so the term in the loss function regarding the model complexity (weight size) will be bigger, in total, the update of that weight will be more or less the same as for the feature with noise, because in that one what happens is the same but the other way around.

Without L2 the weight for the feature without noise would have been super high and the other 0 or close to 0, because it just takes into account the error it adds.

L1 REGULARIZATION

It removes features, every time it makes a weight zero it removes that feature. Its used to save RAM and may reduce noise in the model.

Increasing this count would only be justified if there was a sufficient gain in the model's ability to fit the data. Unfortunately, while this count-based approach is intuitively appealing, it would turn our convex optimization problem into a non-convex optimization problem that's [NP-hard](https://wikipedia.org/wiki/NP-hardness). (If you squint, you can see a connection to the knapsack problem.) So this idea, known as L0 regularization isn't something we can use effectively in practice.

However, there is a regularization term called L1 regularization that serves as an approximation to L0, but has the advantage of being convex and thus efficient to compute. So we can use L1 regularization to encourage many of the uninformative coefficients in our model to be exactly 0, and thus reap RAM savings at inference time.

L1 vs L2 regularization.

L2 and L1 penalize weights differently:

* L2 penalizes *weight*2.
* L1 penalizes |*weight*|.

Consequently, L2 and L1 have different derivatives:

* The derivative of L2 is 2 \* *weight*.
* The derivative of L1 is *k* (a constant, whose value is independent of weight).

You can think of the derivative of L2 as a force that removes x% of the weight every time. As [Zeno](https://wikipedia.org/wiki/Zeno%27s_paradoxes#Dichotomy_paradox) knew, even if you remove x percent of a number *billions of times*, the diminished number will still never quite reach zero. (Zeno was less familiar with floating-point precision limitations, which could possibly produce exactly zero.) At any rate, L2 does not normally drive weights to zero.

You can think of the derivative of L1 as a force that subtracts some constant from the weight every time. However, thanks to absolute values, L1 has a discontinuity at 0, which causes subtraction results that cross 0 to become zeroed out. For example, if subtraction would have forced a weight from +0.1 to -0.2, L1 will set the weight to exactly 0. Eureka, L1 zeroed out the weight.

L1 regularization—penalizing the absolute value of all the weights—turns out to be quite efficient for wide models.