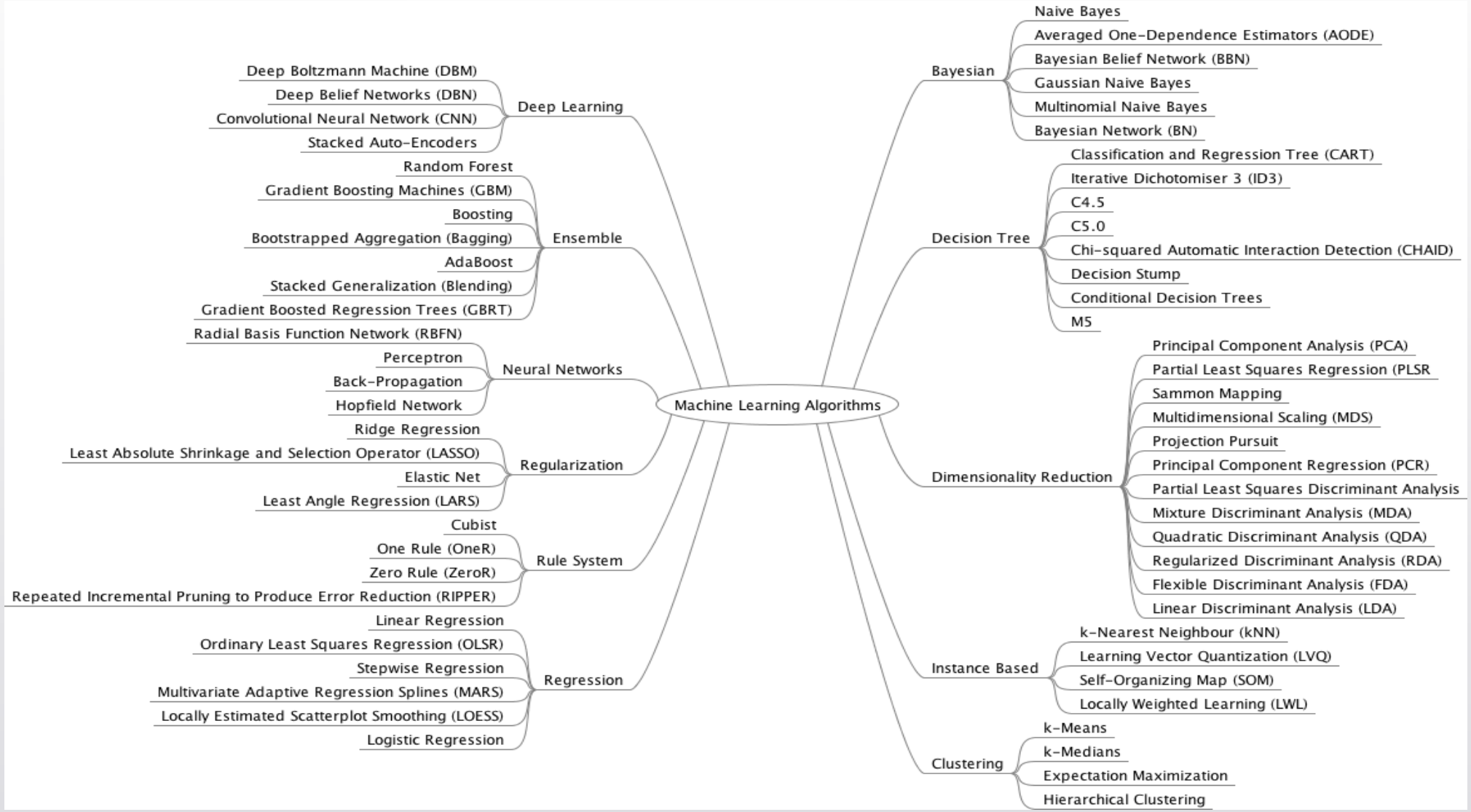
Algoritmos de optimización

¿Qué son?

Algoritmo: Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema. El algoritmo es ese conjunto de pasos que nos permiten alcanzar un objetivo. Es decir, no es más que un concepto teórico, una descripción de lo que haríamos para crear el modelo predictivo (como construir un árbol de decisión, por ejemplo)

En machine learning los algoritmos permiten a un ordenador aprender a realizar una actividad de forma autónoma, se usan los algoritmos programados que reciben y analizan los datos de entrada para predecir los valores de salida dentro del **rango aceptable,** cuando se llega a esta salida aceptable es cuando obtendremos un **modelo** aceptable.


Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

Algoritmos de optimización

Cuando estamos entrenado un **modelo** a partir de un conjunto de datos, es importante ser capaces de **medir la precisión o corrección del modelo**. Para ello vamos a definir lo que se denomina **función de pérdida o función de coste** que va a medir el **error producido**. Durante el entrenamiento del modelo, el objetivo va a ser minimizar dicha función de pérdida (es decir, encontrar el "mejor" modelo posible). Minimizar dicha función no es nada trivial, y el método que usemos para realizar dicha minimización es lo que vamos a llamar **optimizador**.

Función de perdida o función de coste

Mide el error de un modelo. En Machine Learning, no todos los errores tienen la misma importancia: Si clasifica fotografías de perros y gatos, podemos estar interesados en saber qué porcentaje de fotografías (sea cual sea su categoría) hemos clasificado correctamente, pero si clasifica tumores como benignos o malignos, no tiene la misma relevancia clasificar uno benigno como maligno que clasificar uno maligno como benigno, pues la vida de una persona puede estar en juego. Es por ello que para medir la eficacia de un algoritmo **existen diferentes funciones de pérdida o funciones de coste.**

Objetivo del algoritmo de optimización

Dada una función de pérdida o función de coste encontrar la configuración que la minimice.

Ejemplo

Minimización de la función de error

Entonces, si el error MSE cometido por nuestra regla de regresión es:

$$mse = \frac{1}{n}\sum_{1}^{n}(y_n - ax_n - b)^2$$

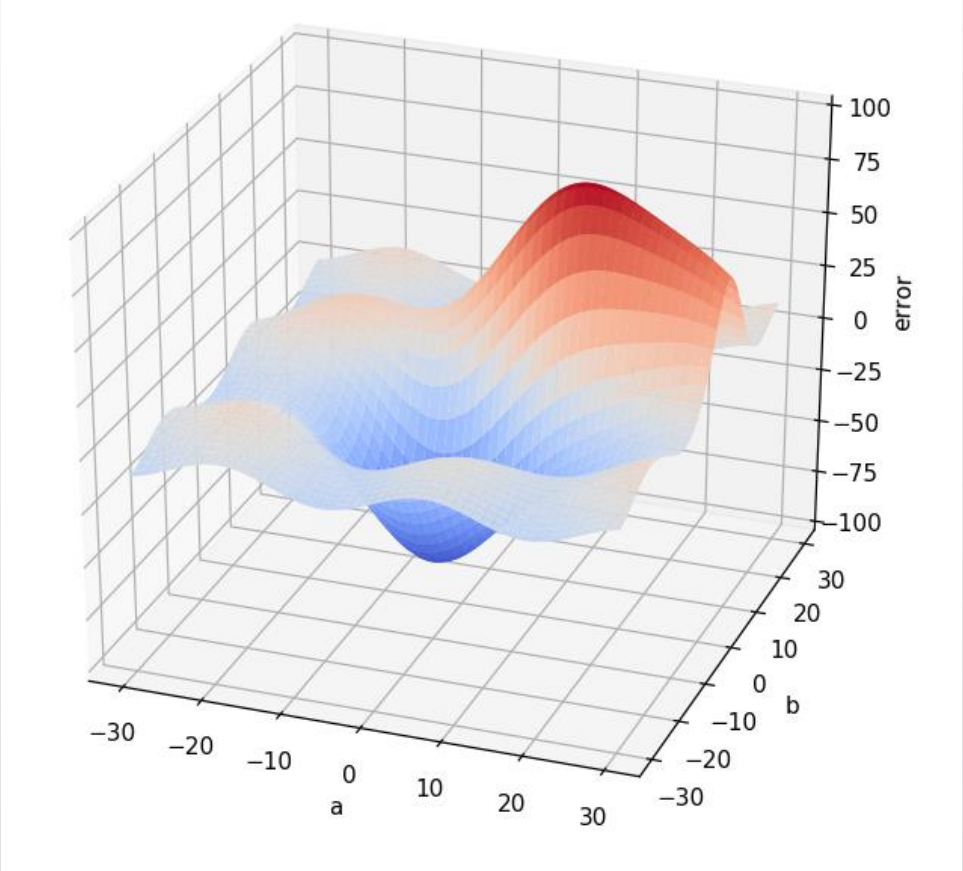
¿Qué valores de a y b minimizan el error? Lógicamente queremos encontrar la mejor recta de regresión, y si el criterio escogido es el error cuadrático medio, queremos encontrar la recta de regresión que minimice dicho error. Pues bien, la fórmula anterior define **la función de error o función de coste de nuestro modelo.**

Alternativamente podemos usar **cálculo infinitesimal** en funciones sencillas, pero, en general, ni el elevado numero de muestras en el dataset ni la función de coste lo van a permitir, alternativamente podemos usar lo que se conoce como **optimizadores.**

Optimizadores

Función de coste = MSE = Error

Para nuestra función anterior, el error dependía de a y b, si generamos muchos valores de a y b y los representamos en una grafica de 3 dimensiones en donde a y b se muestran en el eje horizontal y el **error** en el eje vertical tenemos la grafica siguiente:

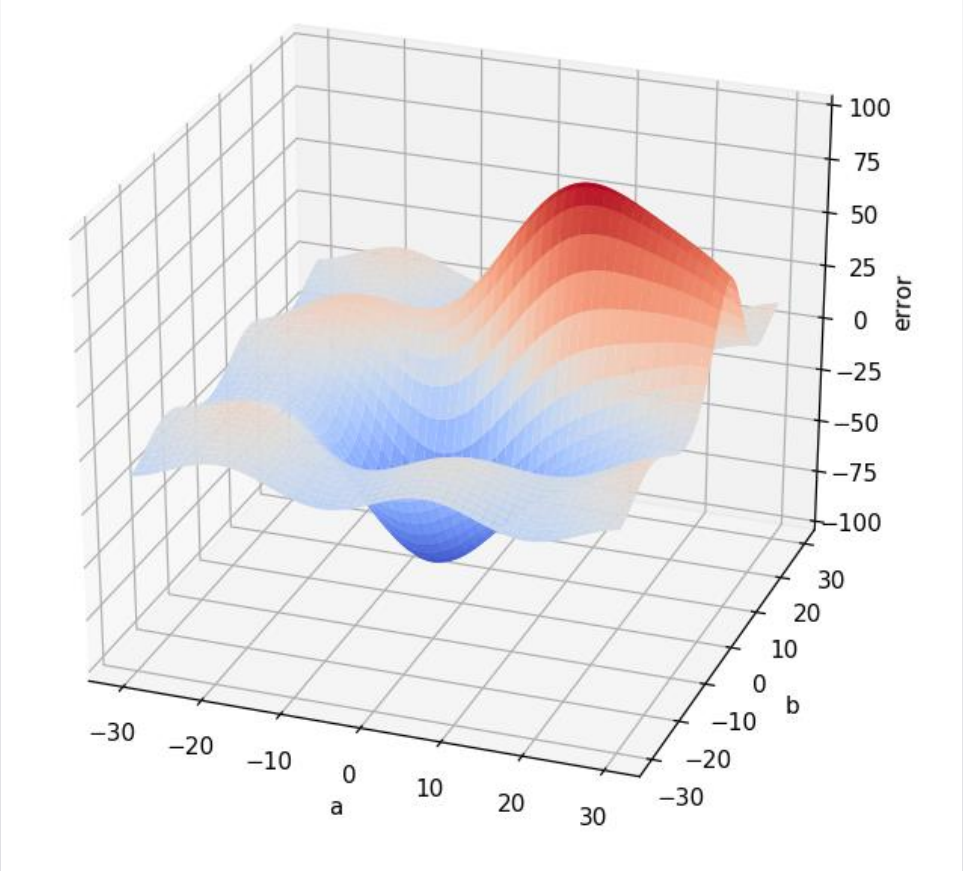


Objetivo del algoritmo optimizador

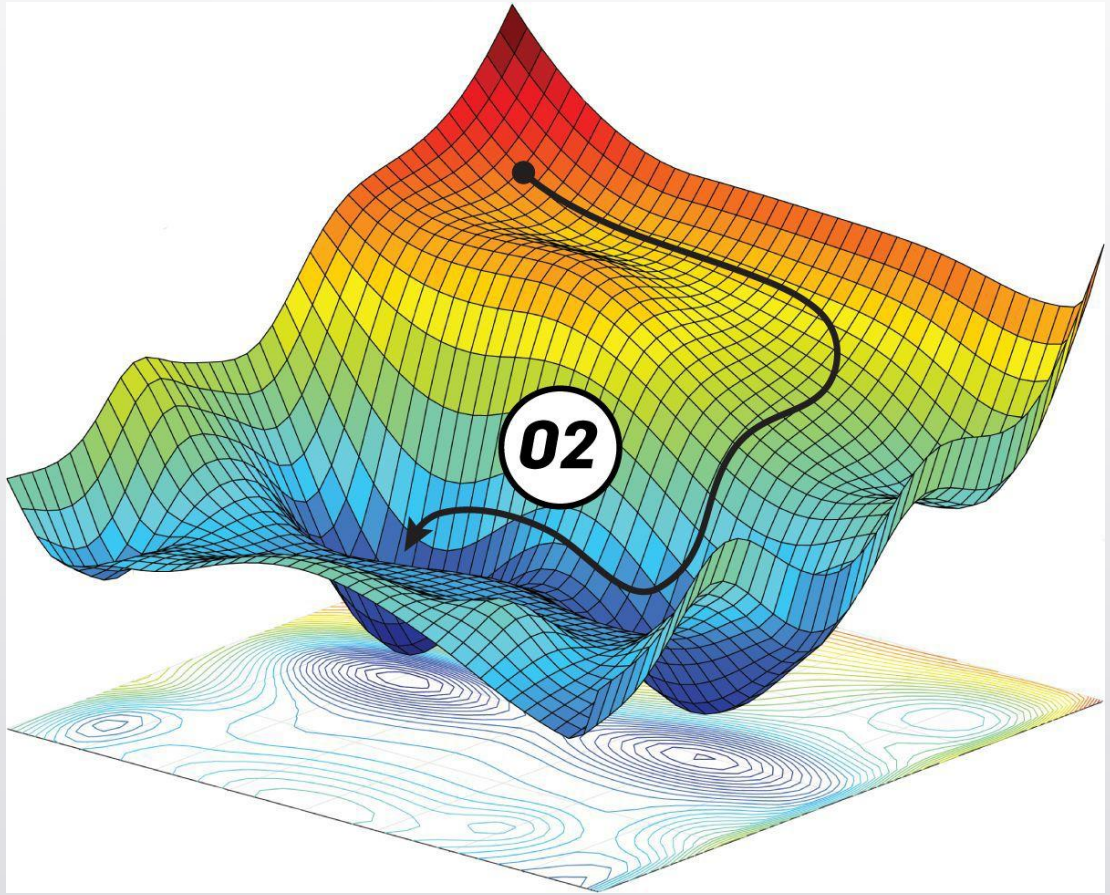
Se aprecia una combinación de valores a y b que generan el menor error posible y a simple vista, parece estar cerca del valor 0 tanto a como b.

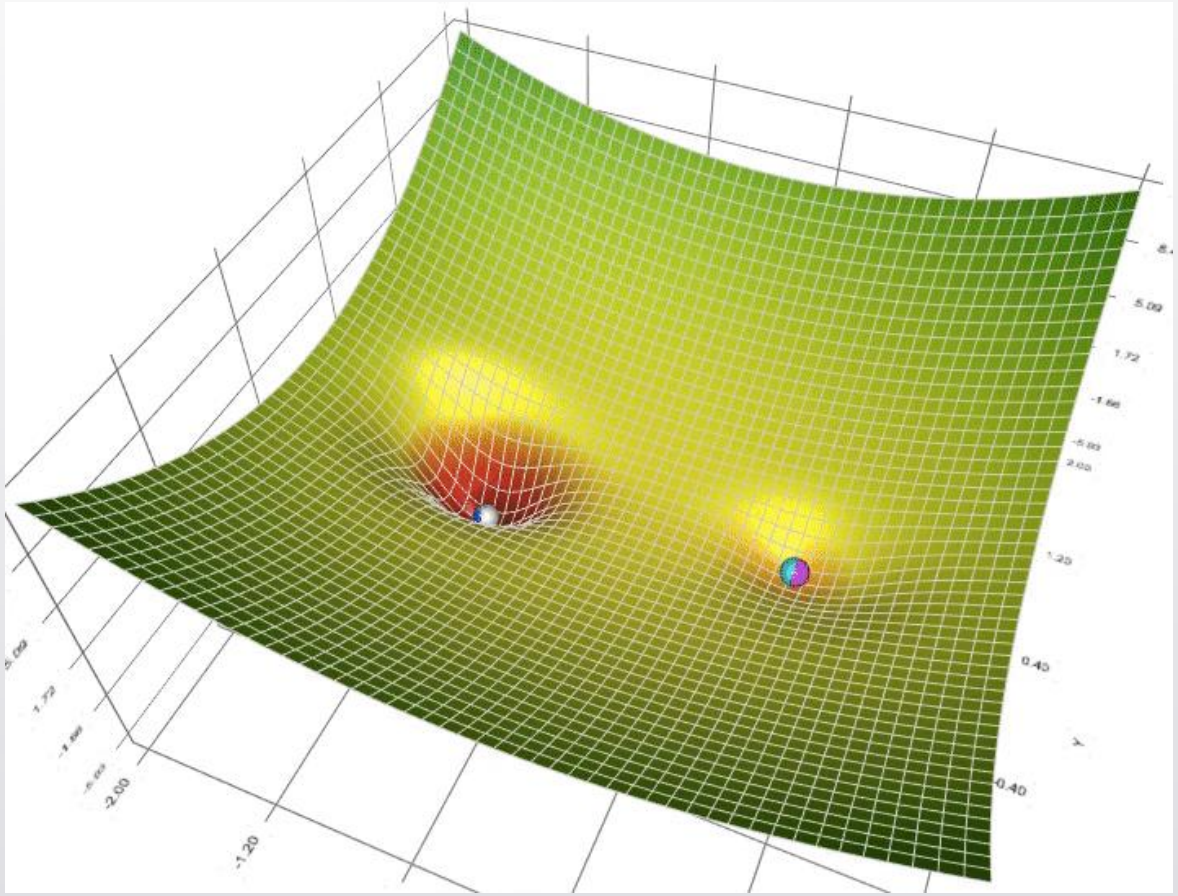
**** Imagen solamente demostrativa ****

El objetivo es averiguar la combinación de a y b **donde el error toma el valor más bajo posible**, al método para encontrar este valor lo llamaremos **optimizador**



Descenso Gradiente / Batch Gradient Descent

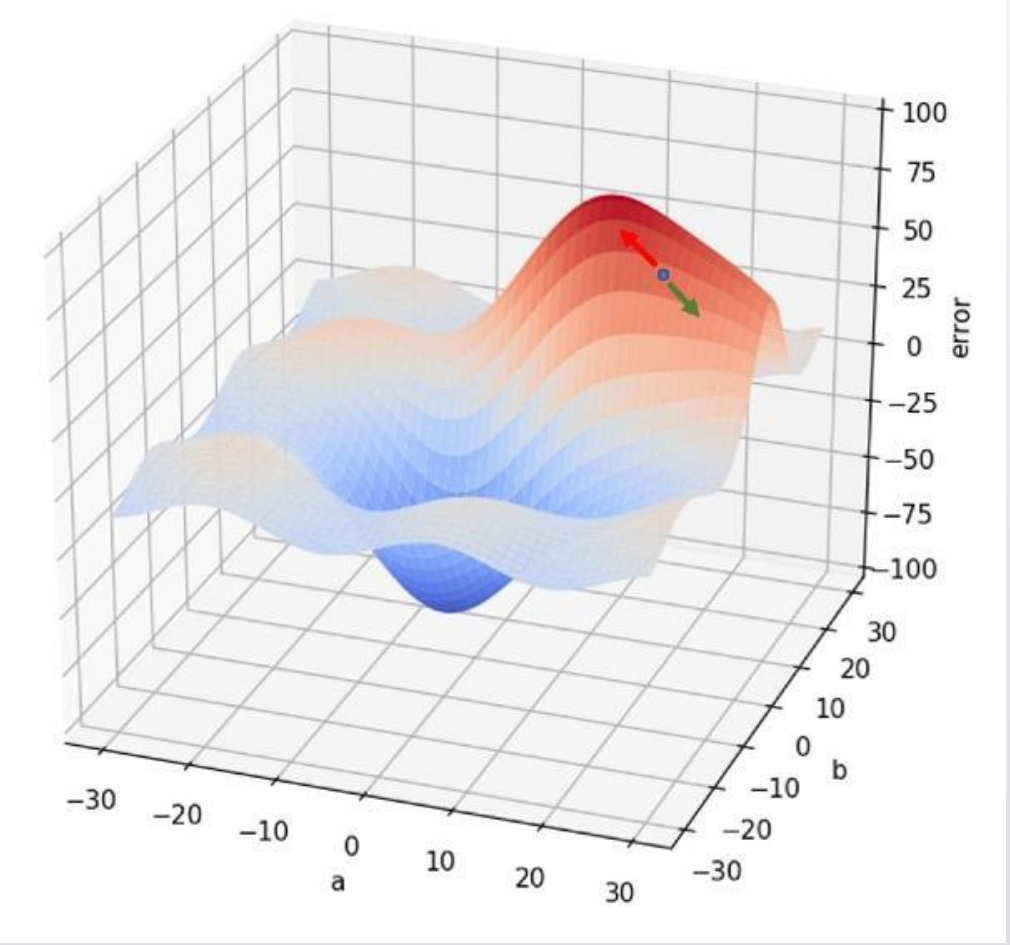




Descenso Gradiente / Batch Gradient Descent

a = 15, b = 20

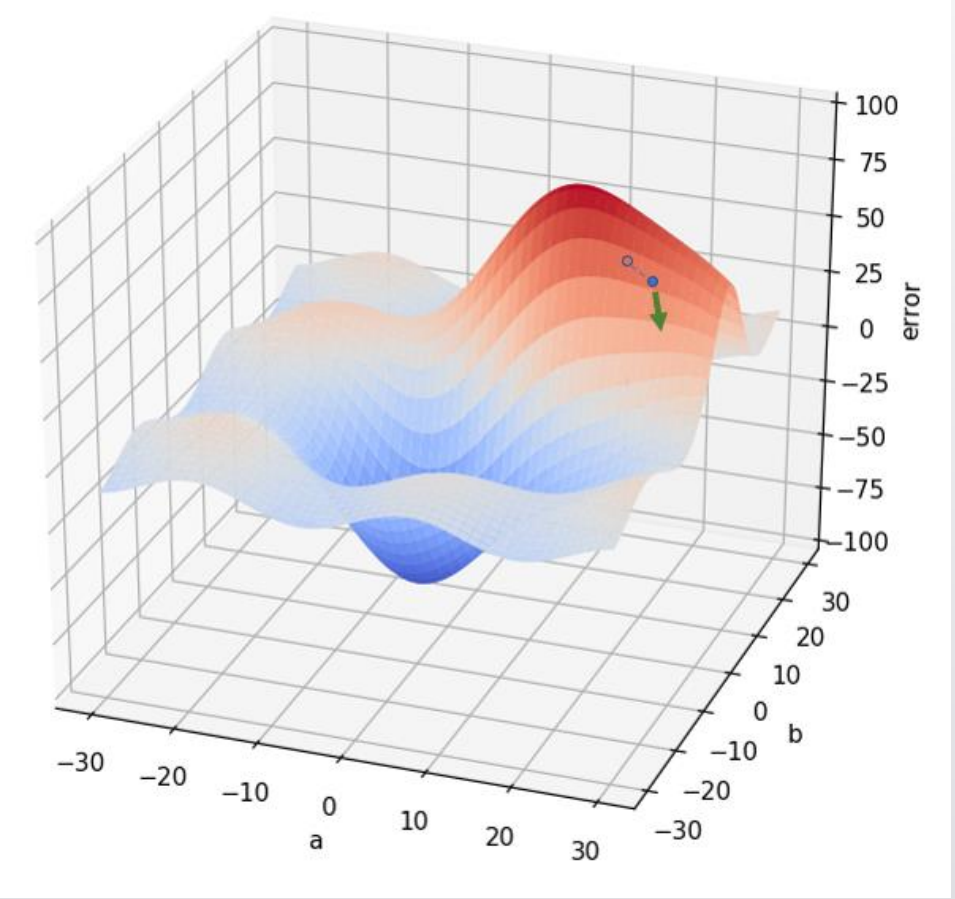
Calculamos el error en dicho punto y vemos la pendiente de la función de error a nuestro alrededor. Determinamos, por ejemplo, que hay una dirección en la que el terreno sube con la mayor pendiente posible (flecha roja) y que hay otra en la que la pendiente es la menor posible (flecha verde):



Descenso Gradiente / Batch Gradient Descent

a = 16, b = 19

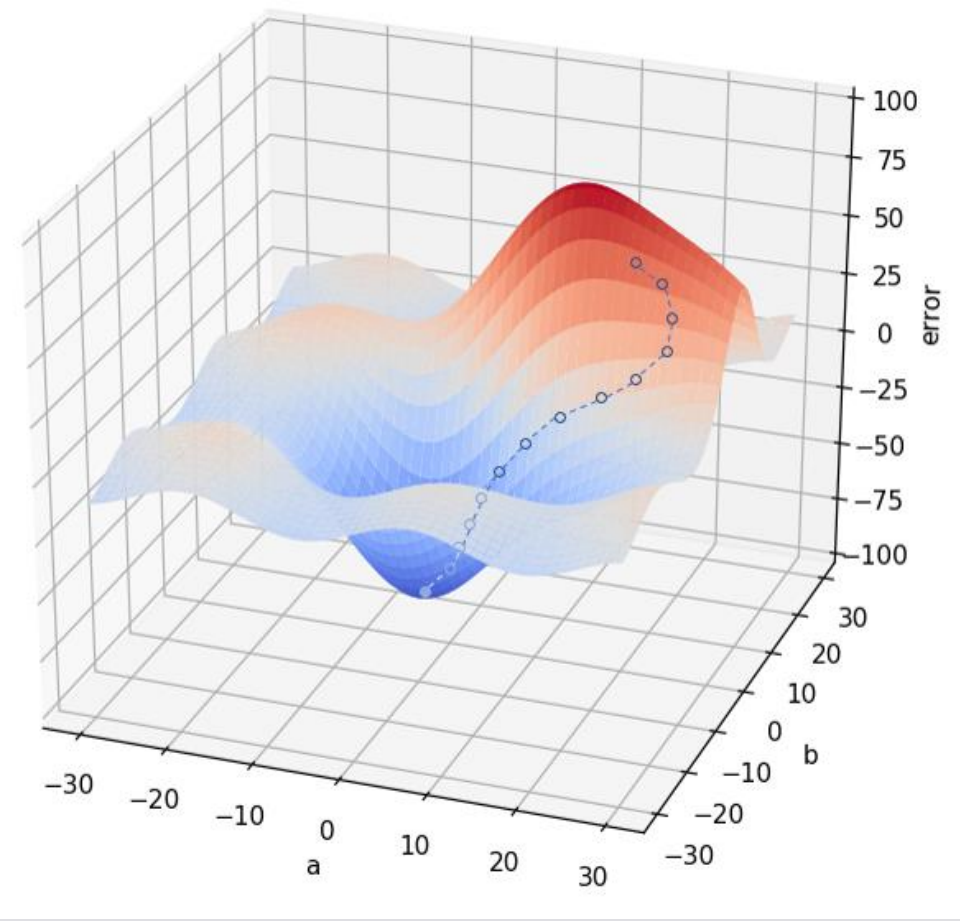
De forma que modificamos nuestros valores iniciales a y b "hacia la dirección de la flecha verde". Esto puede significar que a pase de 15 a 16 (por ejemplo), y que b pase de 20 a 19. En el nuevo punto, volvemos a comprobar cuál es la pendiente a nuestro alrededor:



Descenso Gradiente / Batch Gradient Descent

a = 16.5, b = 18

Este proceso se repite hasta que se alcance un punto en el que no sea posible modificar a y b disminuyendo el error:

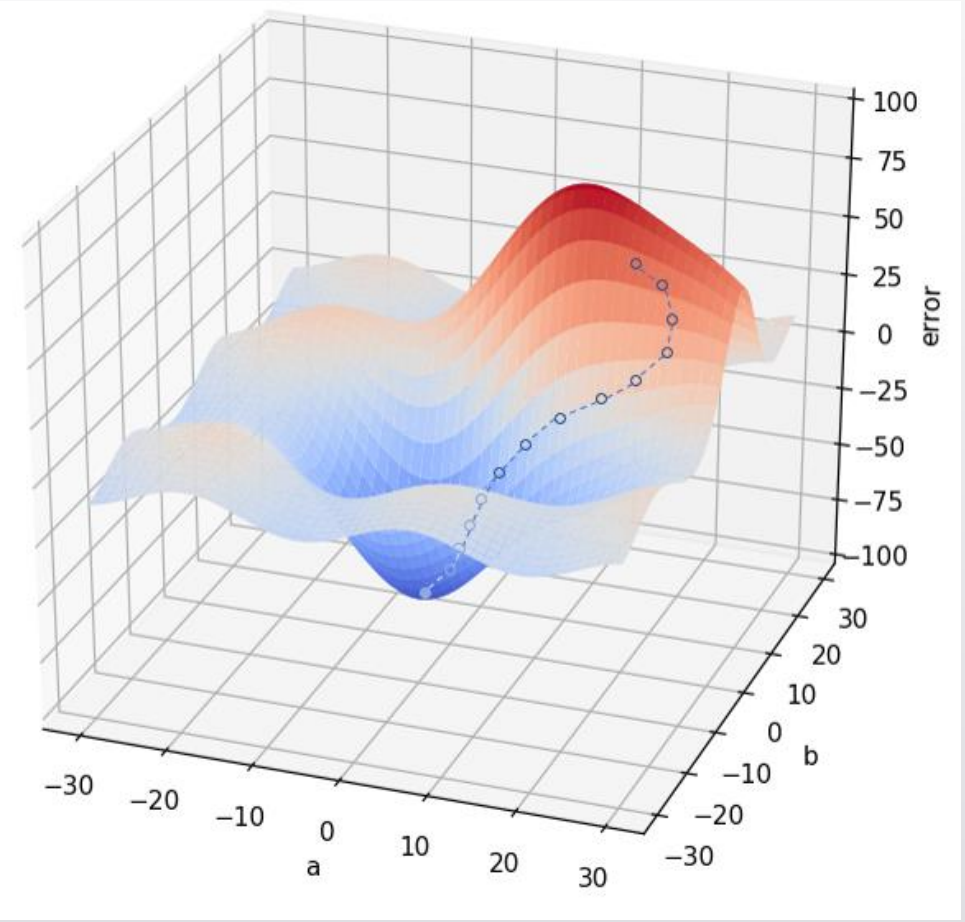


Taza de aprendizaje

la tasa de aprendizaje sería la longitud del paso que da cada vez que decide cambiar de posición.

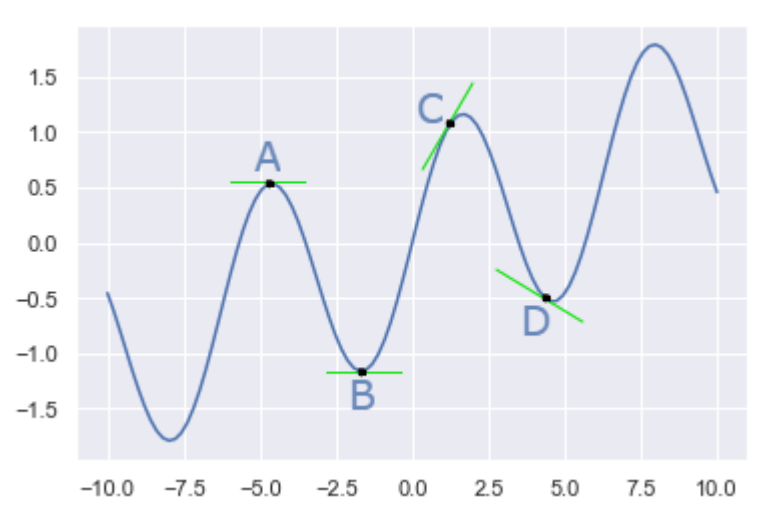
Si damos pasos muy grandes, es posible que lleguemos al punto de menor altura que estamos buscando y lo pasemos de largo.

Por el contrario, si damos pasos muy pequeños es más difícil que nos saltemos el mínimo buscado, pero tardaremos mucho más en llegar a él.



Derivadas y gradientes

$$error = f(a, b)$$



La derivada de f(x) en un punto determina la rapidez con la que la función cambia en dicho punto. Si la derivada en un punto es cero, significa que la función en dicho punto es **horizontal (ni sube ni baja)**. Si la derivada en un punto es un número **alto y positivo** significa que la función **crece** mucho en ese punto. Por último, si la derivada en un punto es un número alto pero **negativo**, significa que la función **decrece** mucho en ese punto. El concepto de derivada en un punto se muestra gráficamente dibujando la recta tangente a la función en dicho punto.

Gradientes

Trabajando con una función de más de una variable, el concepto de derivada se generaliza en el concepto de **gradiente.**
El gradiente en un punto se define como un vector formado por las **derivadas parciales** de la función con respecto a cada una de las variables de la función.

$$Gradiente\ de\ f(a,b) = \nabla f(a,b) = \left(\frac{\partial f}{\partial a}, \frac{\partial f}{\partial b}\right)$$

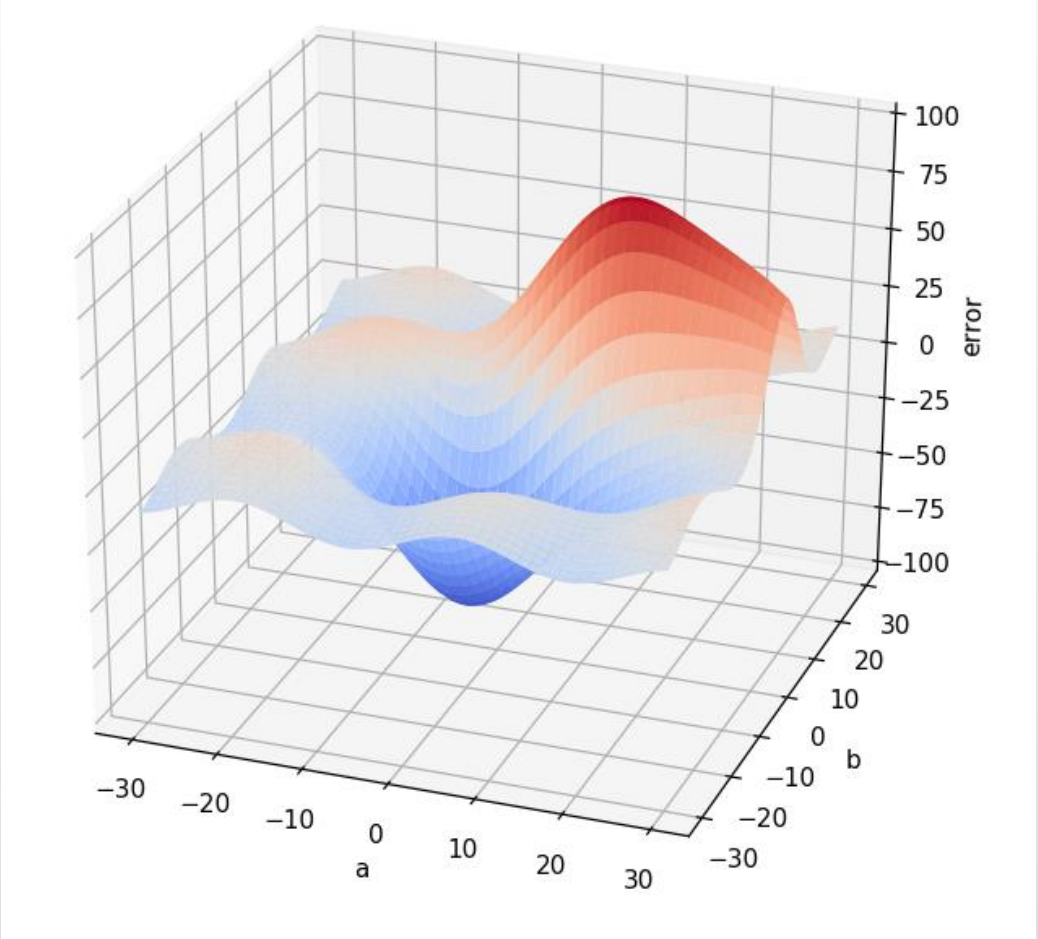
Esto supone que si tenemos una función de dos variables a y b, el gradiente en un punto estará formado por dos valores: el primero representando la derivada parcial con respecto a la variable a, y el segundo representando la derivada parcial con respecto a la variable b. Si tuviésemos una función de 5 variables, el gradiente estaría formado por 5 valores, uno para cada variable.

Ejemplo

Supongamos que **a** toma el valor **10** y **b** toma el valor **5**, en dicho punto el **error** es **50** (por ejemplo).

$$Gradiente\ de\ f(a,b) = \nabla f(a,b) = \left(\frac{\partial f}{\partial a}, \frac{\partial f}{\partial b}\right)$$

Calculamos el gradiente de la función de **error** con respecto a las variables **a** y **b** en el punto (10, 5) y obtenemos el **vector (2, 4).** El "2" significa que si **a** aumenta en una cierta cantidad, el **error** aumenta el doble. El "4" implica algo parecido: Si **b** aumenta en una cierta cantidad, el **error** aumenta la misma cantidad multiplicada por 4.

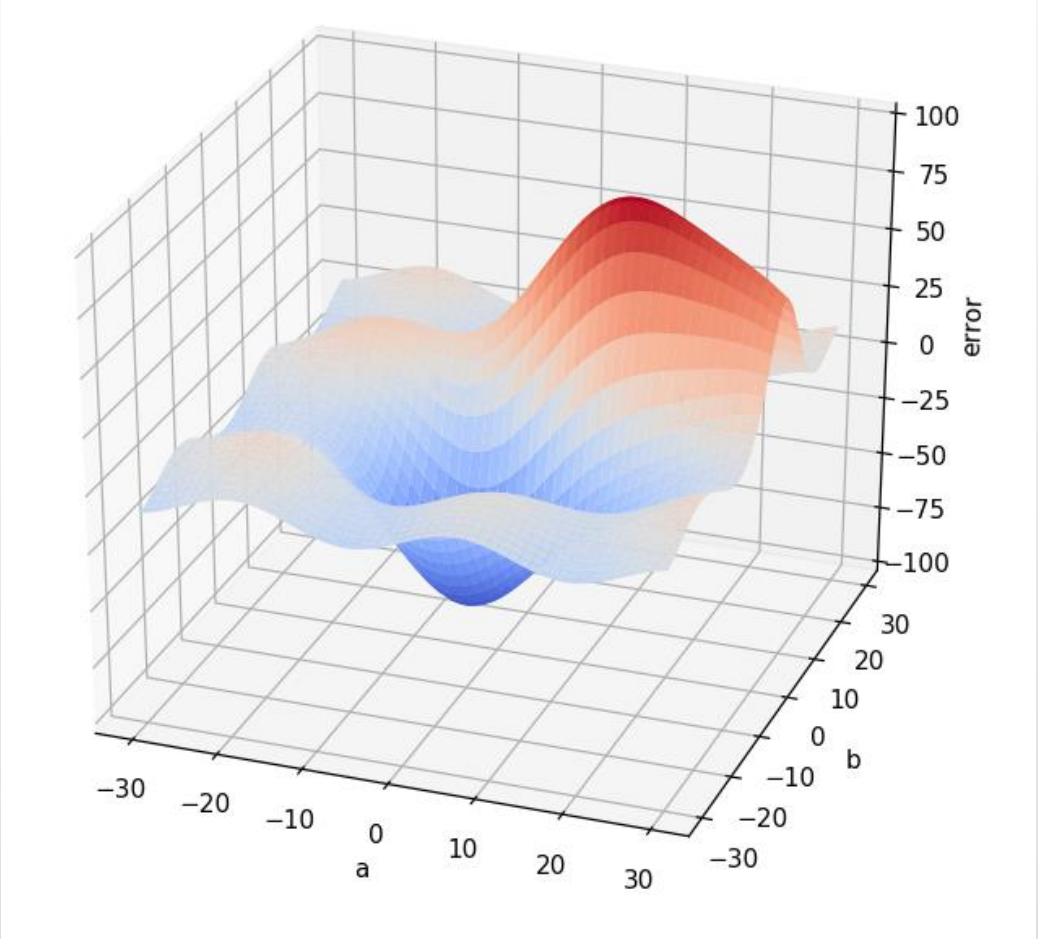


Ejemplo

Entonces, si al aumentar **a** y **b** aumenta el error, si disminuimos **a** y **b** ¡el error deberá disminuir!. Si por **Δ** entendemos "incremento", los incrementos a aplicar a **a** y a **b** para disminuir el error podrían ser los siguientes:

$$\Delta a = -\frac{\partial f}{\partial a}$$

$$\Delta b = -\frac{\partial f}{\partial b}$$



Elección de la tasa de aprendizaje

Restar la derivada parcial supone, en cualquier caso, restar una cantidad demasiado grande. Si estamos en el punto en el que la variable **a** toma el valor **10** y en dicho punto la derivada parcial con respecto a **a** vale **7**, restar a **a** el valor **7** supondría pasar **a** de **10** a **3**. Eso es un salto demasiado grande: no sabemos que forma tiene nuestra función de error entre dichos valores, tal vez tenga un mínimo que estemos saltando. Así que lo que hacemos es multiplicar las derivadas parciales por una pequeña cantidad.

Si la tasa de aprendizaje vale 0.001, por ejemplo, y la derivada parcial calculada era 7, esto supondría restar a **a** 7 x 0.001 = 0.007. Es decir, **a** pasaría de valer 10 a valer 9.993. Esto sí que es un pequeño paso hacia la dirección del valor mínimo de la función. Con la variable **b** habría que hacer lo mismo, claro. Los incrementos de **a** y de **b** quedan, por lo tanto, de la siguiente forma:

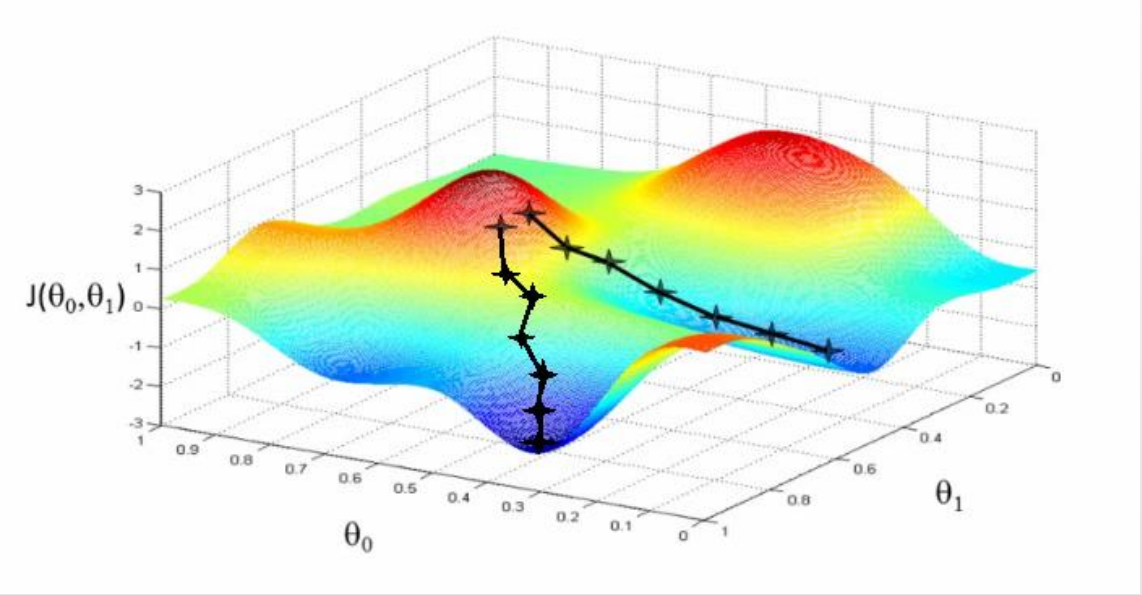
$$\Delta a = -\mu \frac{\partial f}{\partial a}$$

$$\Delta a = -\mu \frac{\partial f}{\partial a}$$

donde **μ** es la tasa de aprendizaje.

Funciones con más de un mínimo

Si la función es convexa (si solo tiene un mínimo) el método funciona. Pero ¿qué ocurre si la función tiene más de un mínimo? Algunos pueden ser mínimos locales (no mínimos absolutos) en los que podríamos "caer", y el algoritmo de descenso de gradiente no podría salir de él. Otra situación es cuando a partir de dos puntos iniciales próximos lleguemos a dos mínimos distintos como se ve en la siguiente imagen: en este caso es cuestión de suerte llegar a uno o a otro:



Otros optimizadores

El optimizador Descenso de Gradiente visto no el único utilizado. De hecho, existe toda una familia de optimizadores que, basados en Descenso de Gradiente, intentan mejorar el rendimiento de éste. Entre otros, tenemos:

- Stochastic Gradient Descent
- Mini-batch Gradient Descent
- Momentum
- AdaGrad
- RMSProp
- Adam

Stochastic Gradient Descent

$$mse = \frac{1}{n}\sum_{1}^{n}(y_n - ax_n - b)^2$$

El optimizador Stochastic Gradient Descent o Descenso de Gradiente Estocástico simplifica el cálculo considerando solo una muestra escogida de forma aleatoria cada vez que realiza el cálculo del gradiente, es decir, pasamos una única muestra, calculamos la función de error asociada y, a partir de esta, el gradiente y los incrementos a aplicar.

Esto supone que el tiempo necesario para el cálculo es muchísimo menor, aunque el algoritmo irá moviéndose hacia el mínimo de forma menos coherente en cada iteración.

Una ventaja de esta "incoherencia" es que puede resultarle más sencillo escapar de un mínimo local.

Mini-batch Gradient Descent

Otra alternativa es la que ofrece la versión "Mini-batch": si hacer pasar todos los datos en cada iteración exige demasiados recursos (aunque devuelva el resultado más exacto de la función de error) y hacer pasar tan solo un dato es la solución más rápida aunque el resultado de la función de error sea solo una aproximación al valor real ¿por qué no plantearse una solución intermedia y calcular el error cada n muestras?

Calcula el gradiente no a partir de todo el conjunto de entrenamiento ni a partir de una única muestra, sino a partir de un pequeño subconjunto aleatorio de muestras llamadas mini-batches. Este algoritmo tarda menos en alcanzar el mínimo que Stochastic Gradient Descent pero corre el riesgo de caer en un mínimo local más fácilmente.

Momentum

Stochastic Gradient Descent with Momentum o, simplemente, Momentum, recuerda el incremento aplicado a las variables en cada iteración y determina la siguiente actualización como una combinación lineal entre el gradiente y el incremento anterior. Es decir, aplica a los incrementos cierta "inercia" de forma que varíen más lentamente.

Nesterov Accelerated Gradient

Una versión de Momentum es la llamada Nesterov Accelerated Gradient (NAG). Este algoritmo añade también la "inercia" comentada, pero intenta que los incrementos disminuyan cuando el algoritmo se acerca al mínimo buscado.

Adagrad

AdaGrad (Adaptative Gradient Algorithm o Algoritmo de Gradiente Adaptativo) es una modificación de Stochastic Gradient Descent en la que se utilizan diferentes tasas de aprendizaje para las variables teniendo en cuenta gradiente acumulado en cada una de ellas. Un problema de este optimizador es que, en ocasiones, puede ocurrir que la tasa de aprendizaje para una variable decrezca demasiado rápidamente debido a la acumulación de altos valores del gradiente al comienzo del entrenamiento, lo que puede llevar a que el aprendiz no sea capaz de aproximarse al mínimo en dicha dimensión.

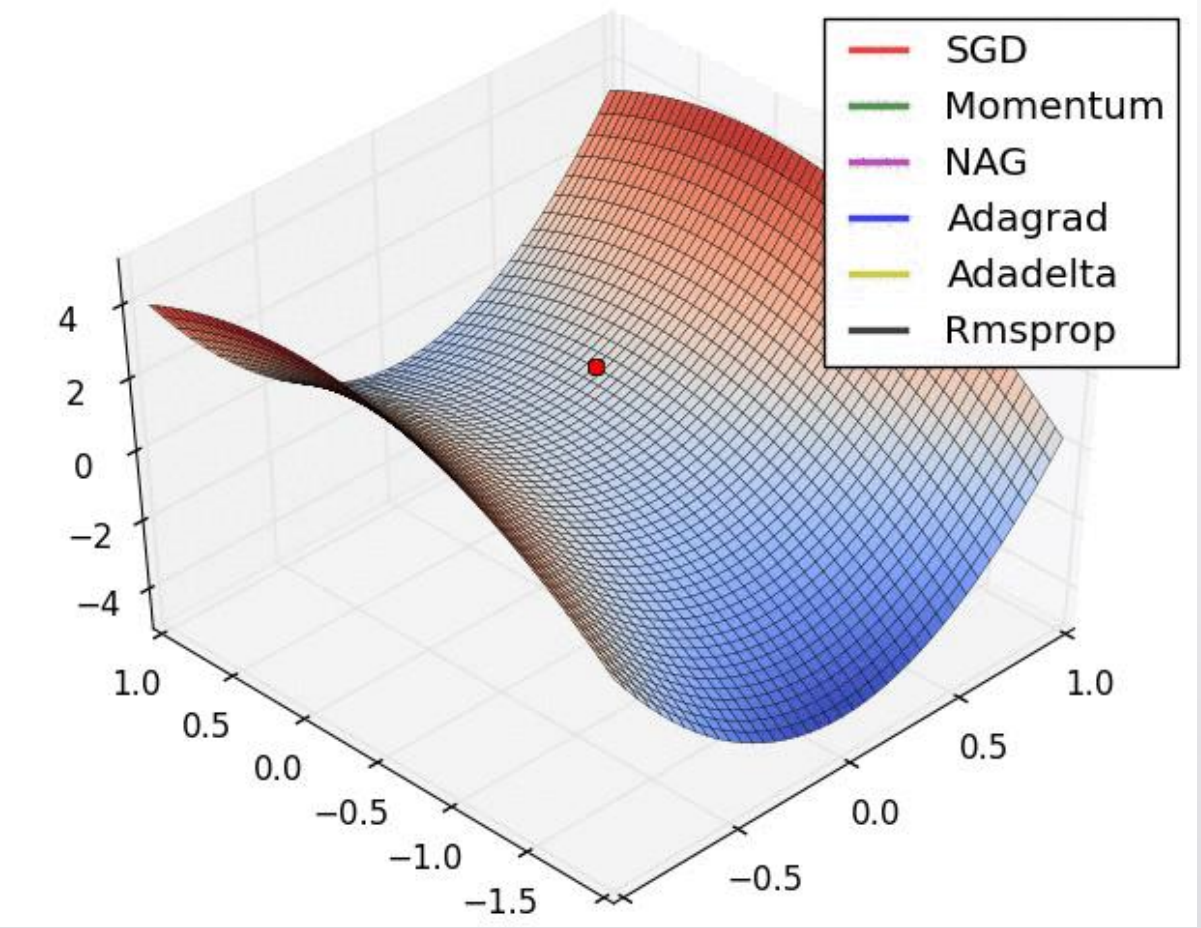
RMSProp

RMSProp o Root Mean Square Propagation es una variación de AdaGrad en la que, en lugar de mantener un acumulado de los gradientes, se utiliza el concepto de "ventana" para considerar solo los gradientes más recientes.

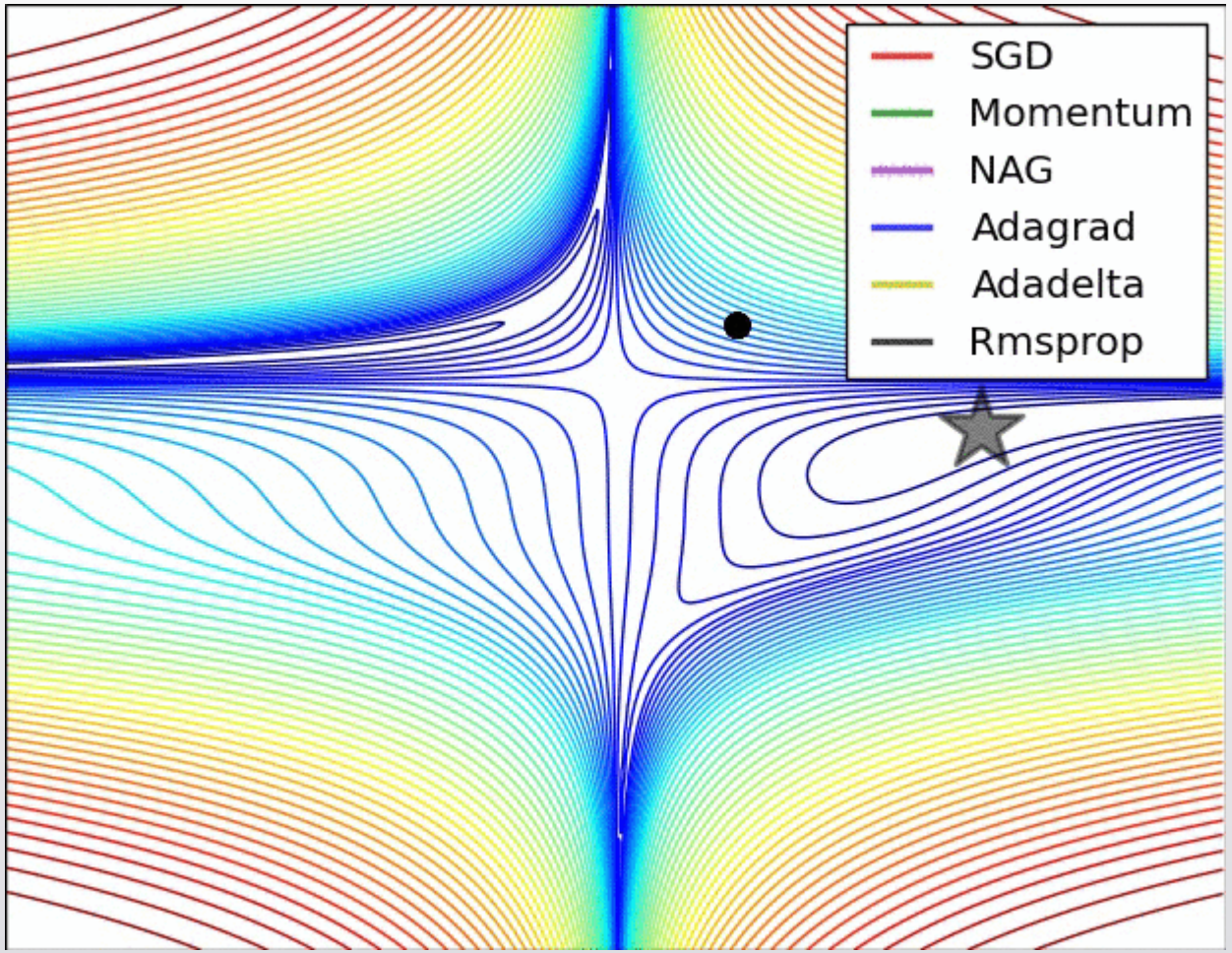
Adam

Adam o Adaptative Moment Optimization combina la metodología de Momentum y RMSProp, calculando una combinación lineal entre el gradiente y el incremento anterior, y considera los gradientes recientemente aparecidos en las actualizaciones para mantener diferentes tasas de aprendizaje por variable.

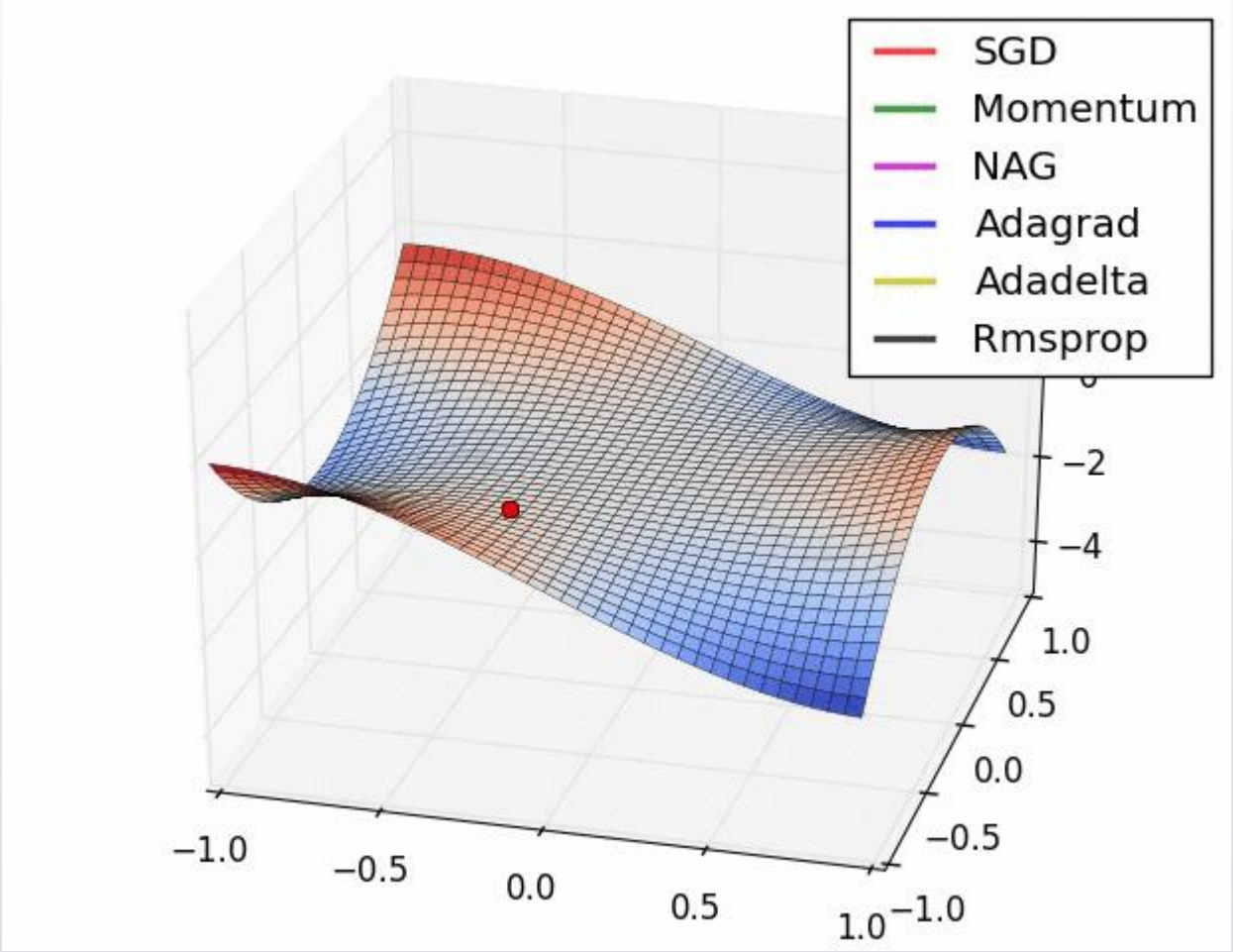
Impacto del optimizador. Valle largo



Impacto del optimizador. Función de Beale



Impacto del optimizador. Punto de silla



Gracias!