{desafío}
latam_

Optimización _

Motivación

Preliminares

- Hasta el momento hemos ignorado (deliberadamente) cómo aprende un algoritmo.
- Ahora aprenderemos cómo se escogen los parámetros en un conjunto de datos de entrenamiento para posteriormente generar predicciones.
- Para entender cómo funcionan las redes neuronales en el contexto de TensorFlow y Keras, debemos tener nociones sobre cómo funcionan los algoritmos de optimización.

El problema

- Resulta que la finalidad es encontrar un óptimo de una función objetivo de un algoritmo.
- Por lo general encontraremos **problemas de minimización** respecto al costo/pérdida del desempeño algorítmico.
- Métodos:
  - **De Gradiente:** Descenso de Gradiente, Gradiente Conjugado, Cuasi-Newtoniano
  - **Sin Gradiente:** Simplex/Nelder Mead, Genética.
- Dada una función de pérdida, nuestro objetivo es encontrar una combinación de parámetros del modelo que minimice ésta.
- Para efectos prácticos, implementaremos métodos basados en gradientes.

Métodos basados en Gradiente

{desafío} latam_

¿Qué son?

- El gradiente es la representación de la máxima inclinación de una superficie respecto a un punto evaluado.
- Por defecto, el gradiente nos informará de cuál es la dirección de mayor inclinación de una función.
- Dado que nuestro problema es minimizar esta función de pérdida, nuestro objetivo es tomar la máxima inclinación para disminuir nuestra función de pérdida.

El caso de la regresión lineal

Problema clásico: ajustar una regresión en función a un conjunto de datos.

$$\hat{y}(\gamma, \beta) = \gamma + \beta \cdot x$$

Parámetros

Objetivo: encontrar una combinación de parámetros.
Esta combinación de parámetros debe satisfacer algún problema de **optimización**.

El caso de la regresión lineal

La función de pérdida minimizar responde al mínimo cuadrados ordinarios. La métrica va a ser el **Promedio del Error Cuadrático**:

$$\text{RSS} = J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\beta^{\mathsf{T}}x_i - y_1)^2$$

Deseamos encontrar una función objetivo que resuelva el problema de optimización:

$$\underset{\beta \in \boldsymbol{\beta}}{\operatorname{argmin}} J(\beta)$$

Para ello recorreremos la superficie de la función de pérdida J.

Descenso de Gradiente

Iniciamos nuestro parámetro en un punto de inicio (generalmente aleatorio)
Iteramos la siguiente expresión hasta que se satisfaga algún criterio de convergencia:

$$\hat{\theta}_{i+1} \leftarrow \hat{\theta}_i - \alpha \nabla_\theta \mathbb{E}\left[J(\theta)\right]$$

Descenso de Gradiente

Iniciamos nuestro parámetro en un punto de inicio (generalmente aleatorio)
Iteramos la siguiente expresión hasta que se satisfaga algún criterio de convergencia:

$$\hat{\theta}_{i+1} \leftarrow \hat{\theta}_i - \alpha \nabla_\theta \mathbb{E}[J(\theta)]$$

Valor esperado de nuestra función de costo elegida.

Descenso de Gradiente

Iniciamos nuestro parámetro en un punto de inicio (generalmente aleatorio)
Iteramos la siguiente expresión hasta que se satisfaga algún criterio de convergencia:

$$\hat{\theta}_{i+1} \leftarrow \hat{\theta}_i - \alpha \nabla_\theta \mathbb{E}[J(\theta)]$$

Vector con la diferenciación de la esperanza de la función de error (Gradiente)

Descenso de Gradiente

Iniciamos nuestro parámetro en un punto de inicio (generalmente aleatorio)
Iteramos la siguiente expresión hasta que se satisfaga algún criterio de convergencia:

$$\hat{\theta}_{i+1} \leftarrow \hat{\theta}_i - \alpha \nabla_\theta \mathbb{E}[J(\theta)]$$

Tasa de aprendizaje en la siguiente iteración.

Descenso de Gradiente

Iniciamos nuestro parámetro en un punto de inicio (generalmente aleatorio)
Iteramos la siguiente expresión hasta que se satisfaga algún criterio de convergencia:

$$\hat{\theta}_{i+1} \leftarrow \hat{\theta}_i - \alpha \nabla_\theta \mathbb{E}[J(\theta)]$$

Forzamos la dirección de manera descendiente.

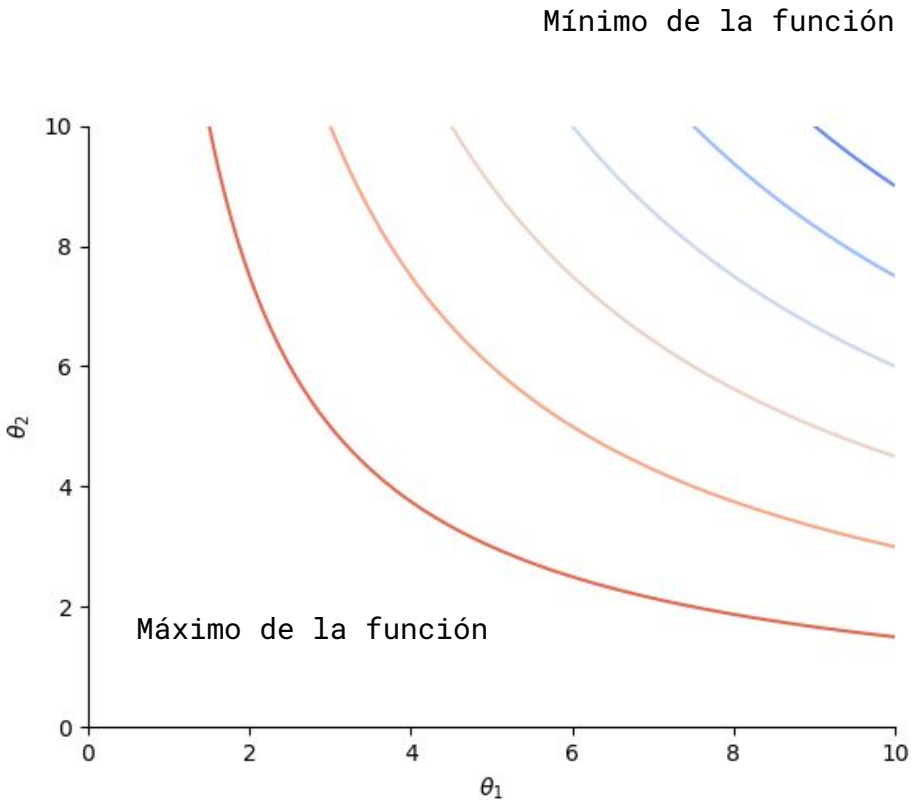
Descenso de Gradiente

Iniciamos nuestro parámetro en un punto de inicio (generalmente aleatorio)
Iteramos la siguiente expresión hasta que se satisfaga algún criterio de convergencia:

$$\hat{\theta}_{i+1} \leftarrow \hat{\theta}_i - \alpha \nabla_\theta \mathbb{E}[J(\theta)]$$

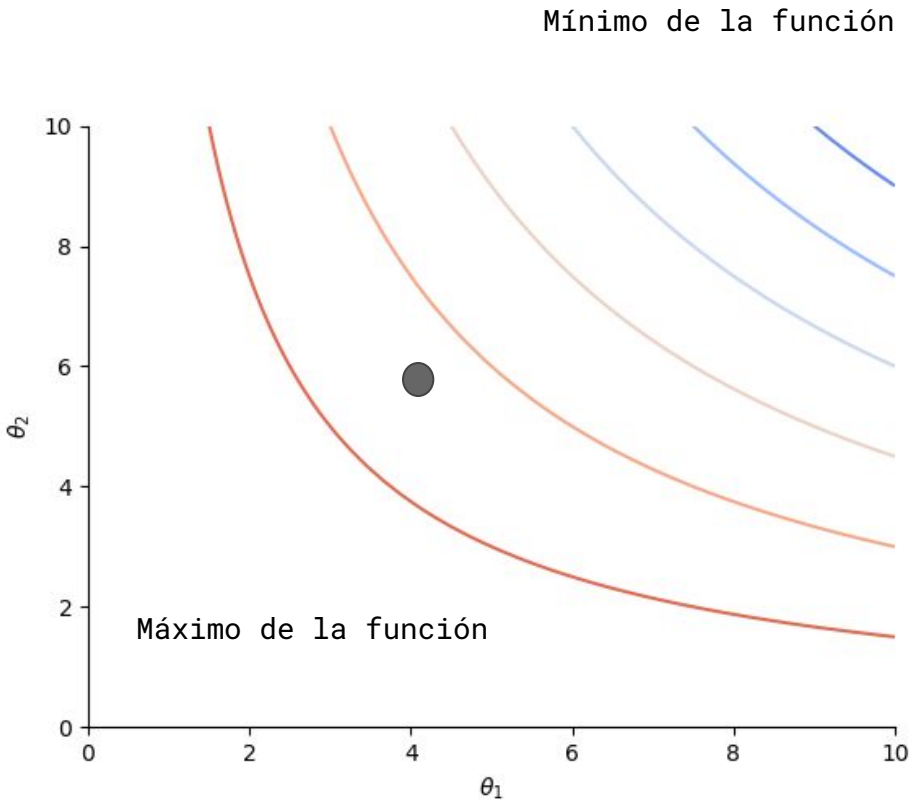
Función candidata contra la cual evaluamos el descenso

Visualización del proceso algorítmico



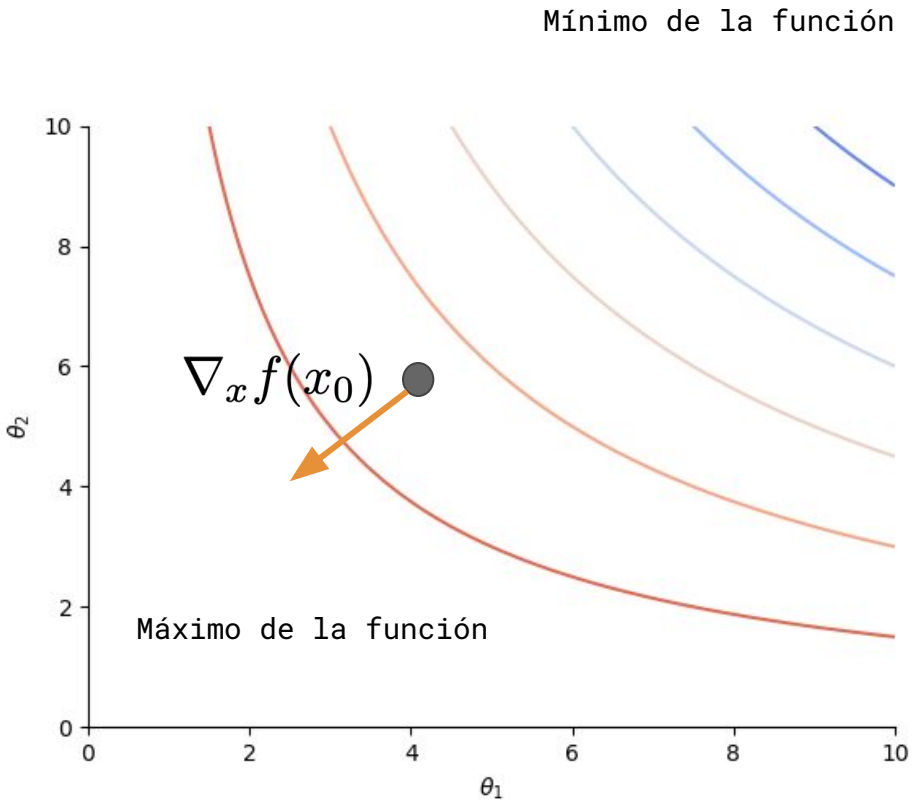
Evaluamos nuestra función de pérdida en múltiples puntos, dando paso a una superficie de la función de pérdida.

Visualización del proceso algorítmico



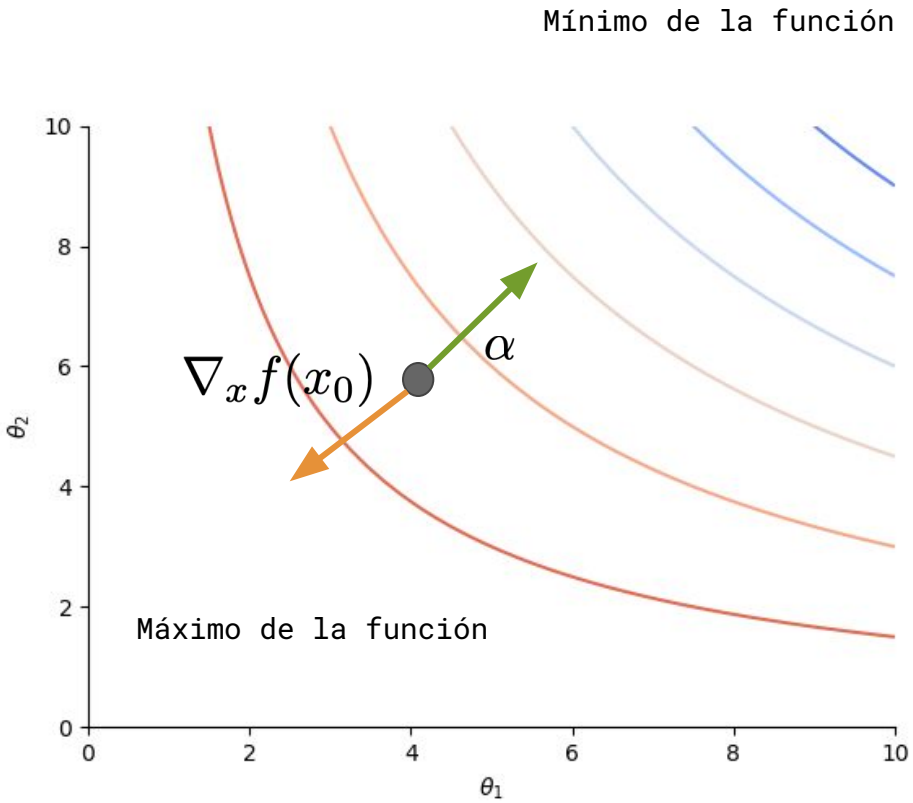
Tomamos aleatoriamente, un punto de inicio a considerar que representará una combinación de parámetros.

Visualización del proceso algorítmico



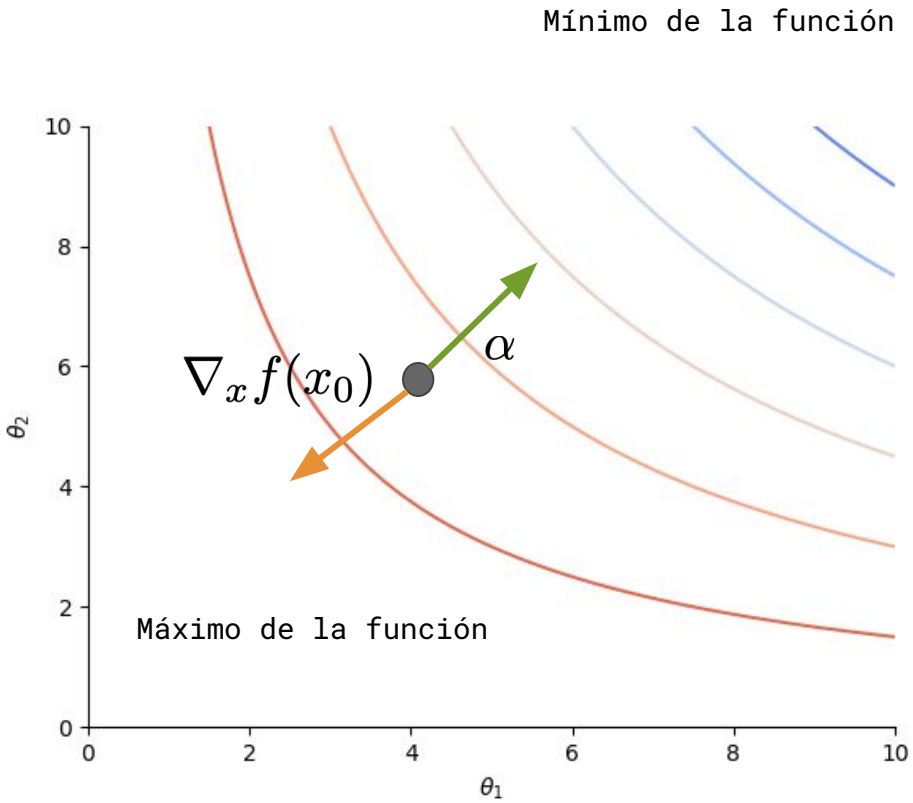
Evaluamos el gradiente del punto de inicio, el cual nos indicará cuál es la dirección de máximo ascenso en nuestra superficie de pérdida.

Visualización del proceso algorítmico



Dado que buscamos minimizar la pérdida dado una combinación de parámetros, tomamos la dirección inversa del gradiente

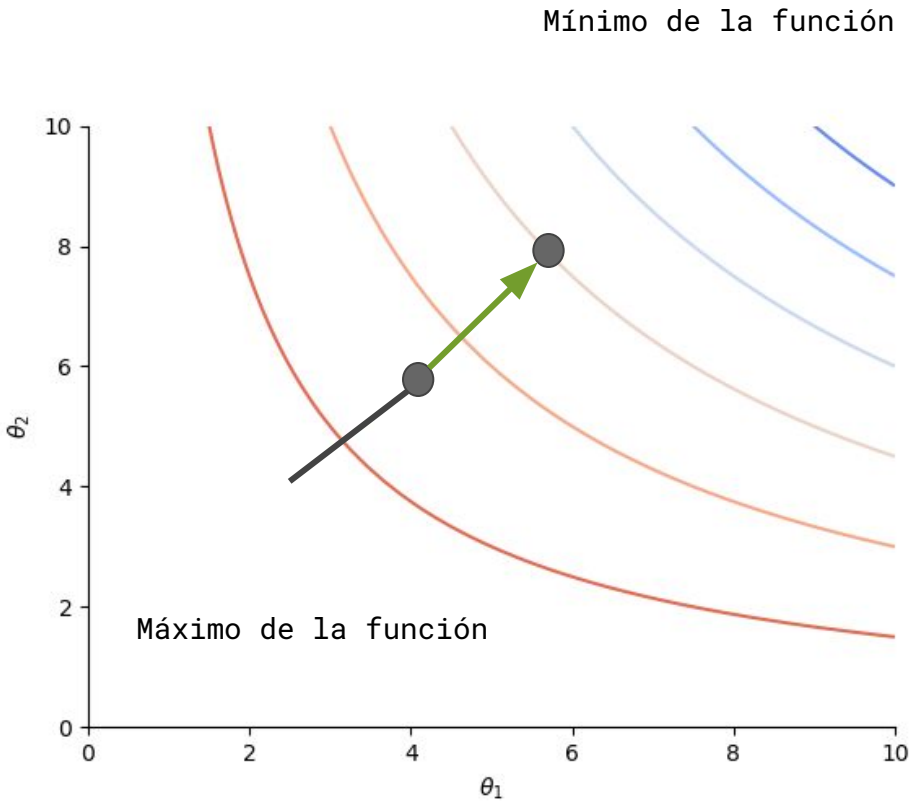
Visualización del proceso algorítmico



Dado que buscamos minimizar la pérdida dado una combinación de parámetros, tomamos la dirección inversa del gradiente.

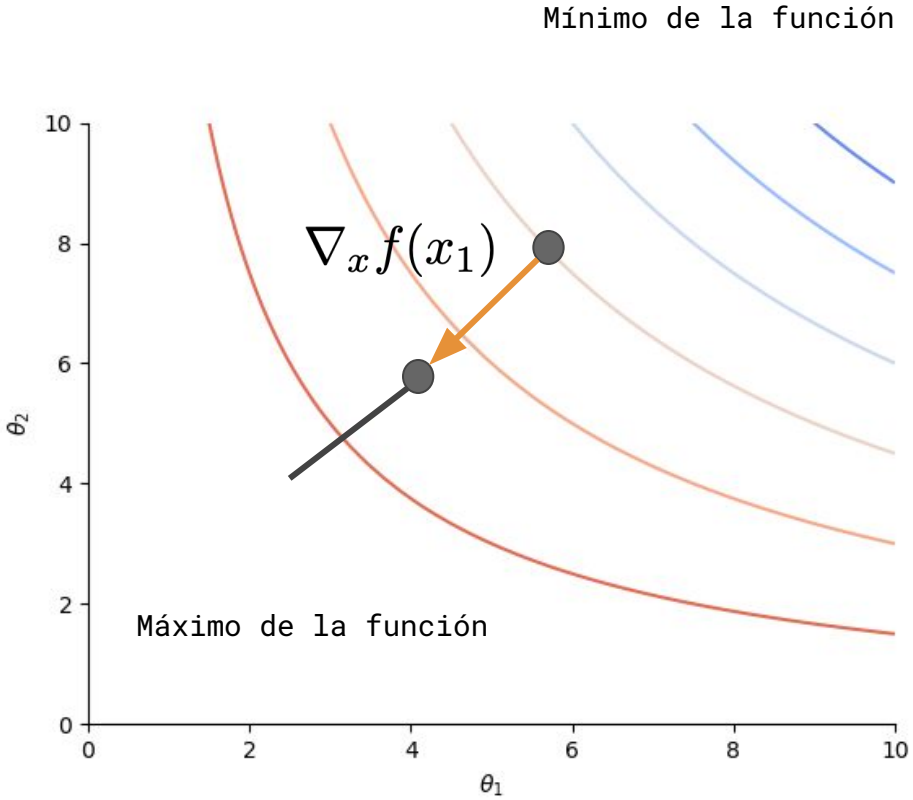
Considerando una tasa de aprendizaje fija, actualizamos nuestro punto óptimo.

Visualización del proceso algorítmico



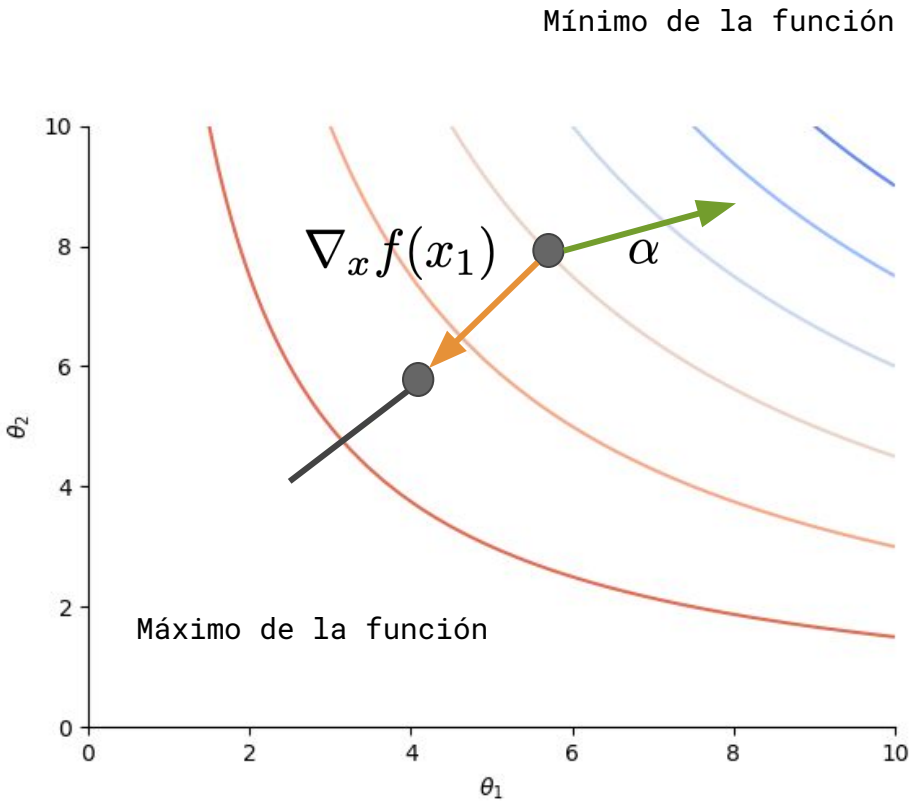
Actualizamos el punto de nuestra función a evaluar.

Visualización del proceso algorítmico



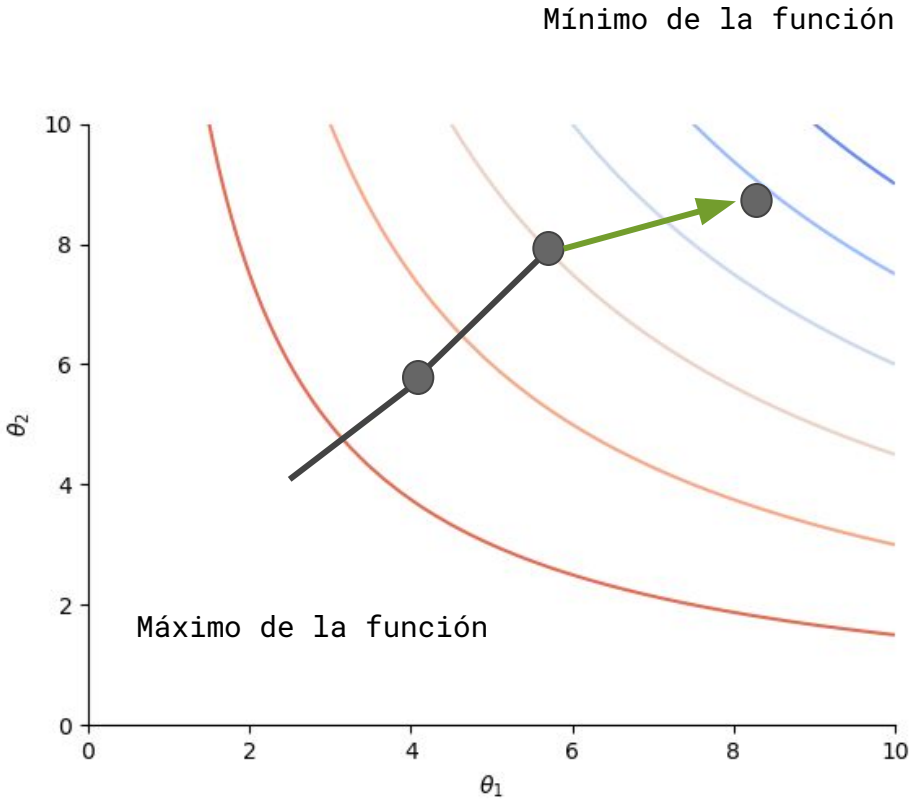
Evaluamos el gradiente en esta siguiente iteración.

Visualización del proceso algorítmico



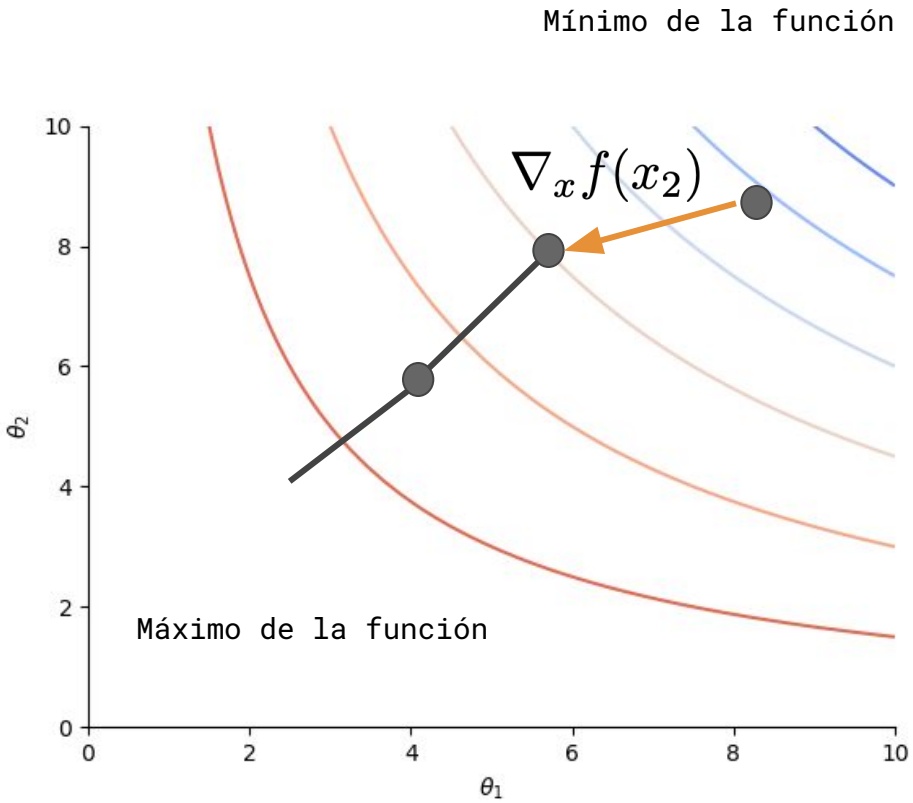
Actualizamos nuestra dirección de descenso del gradiente

Visualización del proceso algorítmico



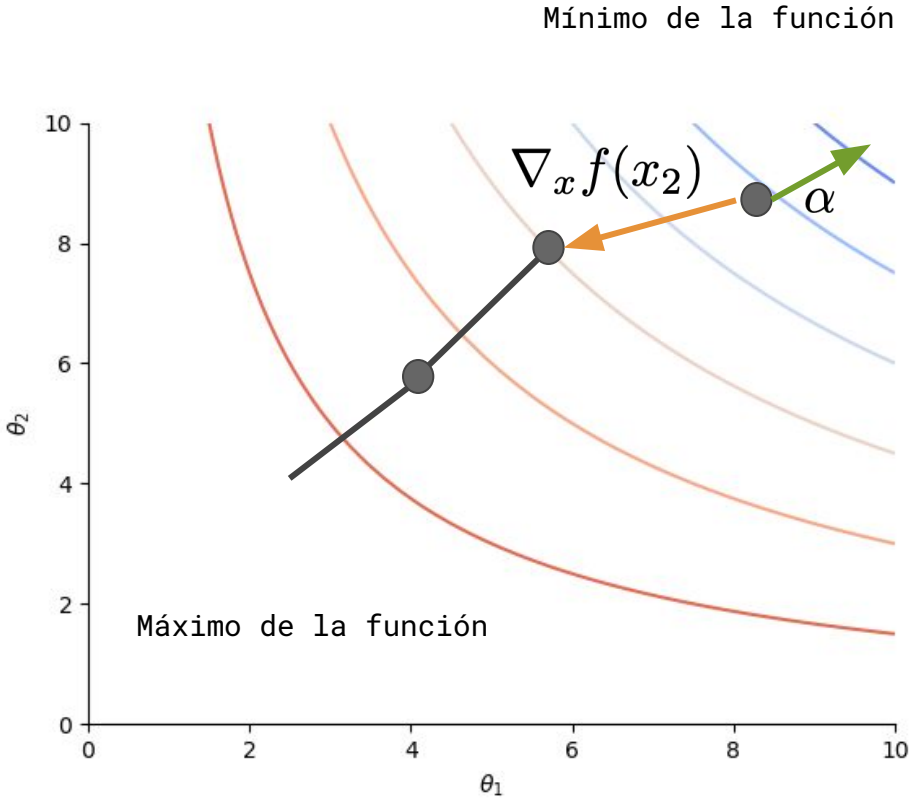
Iteramos

Visualización del proceso algorítmico



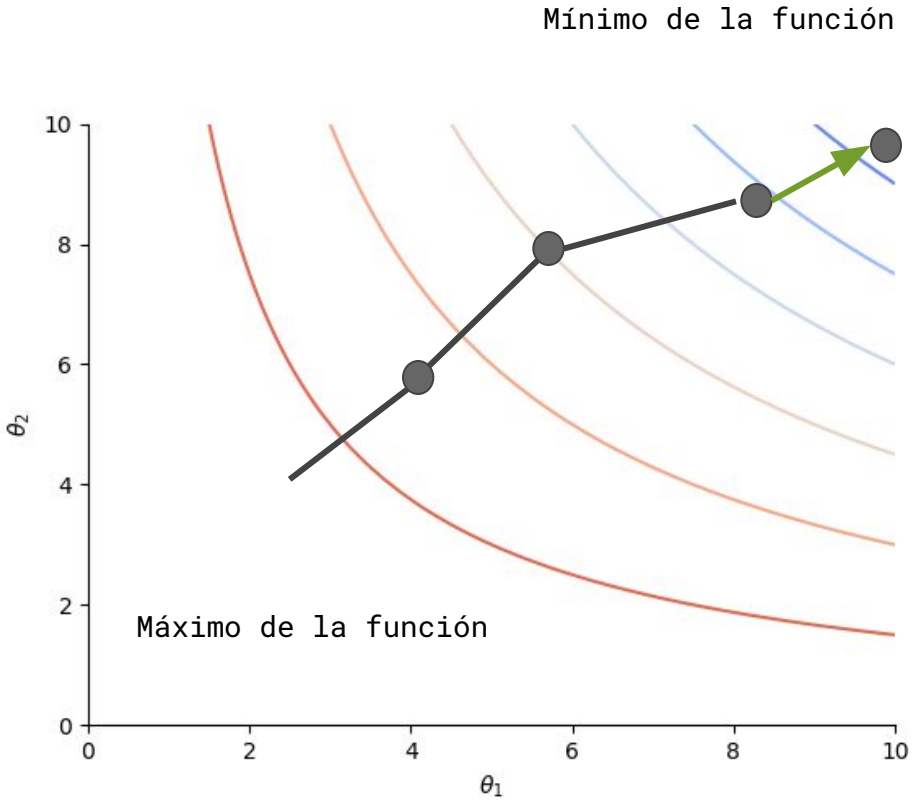
Iteramos

Visualización del proceso algorítmico



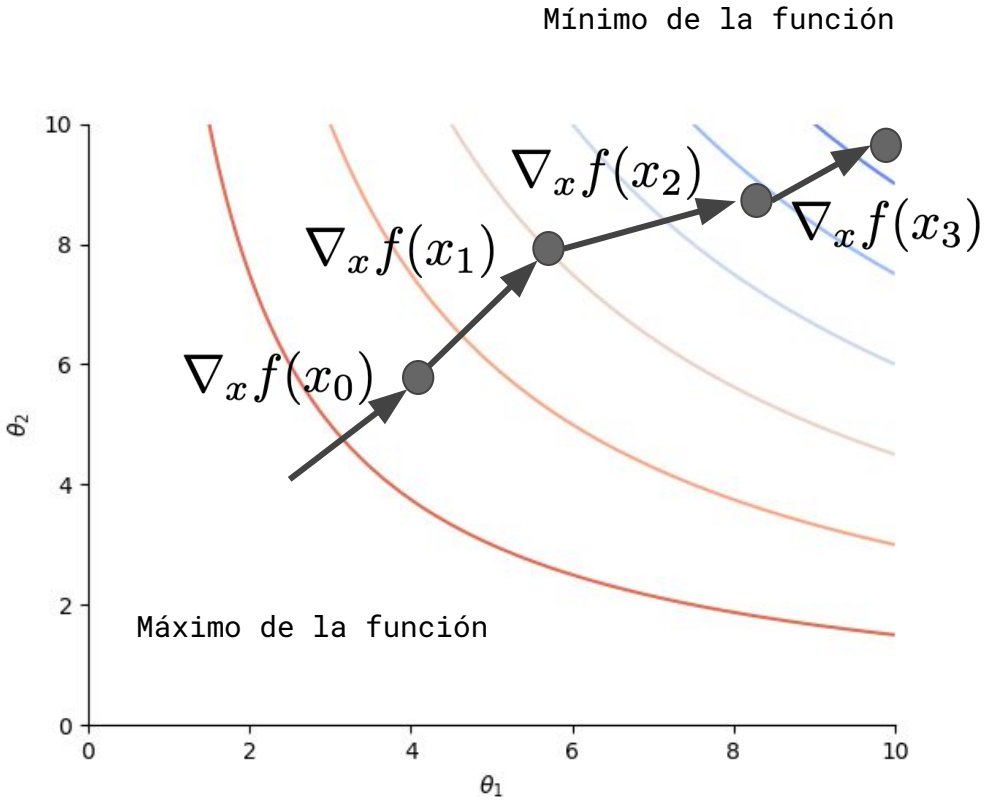
Iteramos

Visualización del proceso algorítmico



Iteramos

Visualización del proceso algorítmico



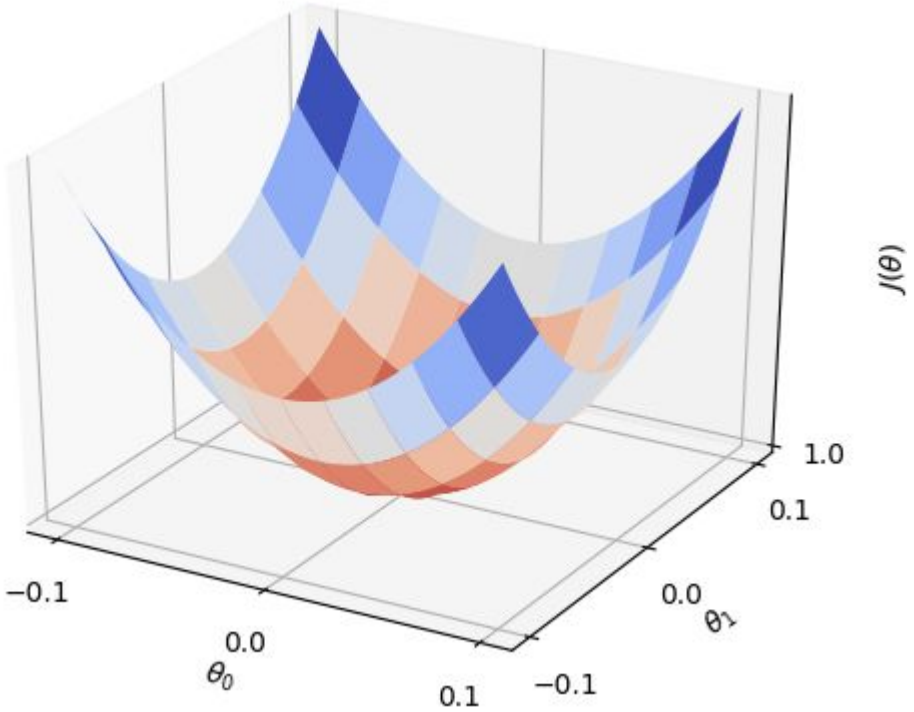
Iteramos

Variantes del Descenso de Gradiente

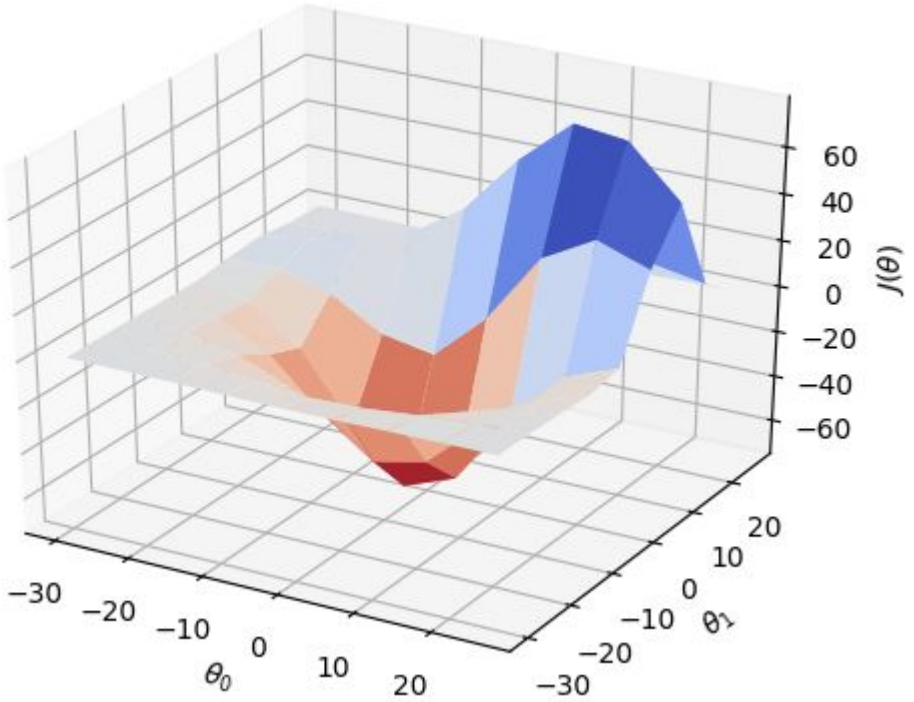
Problemas con Descenso de Gradiente

- La tasa de aprendizaje es un hiperparámetro: No estamos exentos de implementar heurísticas para encontrar el óptimo.
- No tenemos certeza sobre cuál es la topología específica de la superficie del espacio de búsqueda.
- La tasa de aprendizaje se comporta idéntica para todos los atributos.
- Hace uso de todo el conjunto de entrenamiento para efectuar una sola actualización del paso. Esto presenta complicaciones asociadas al tiempo de ejecución

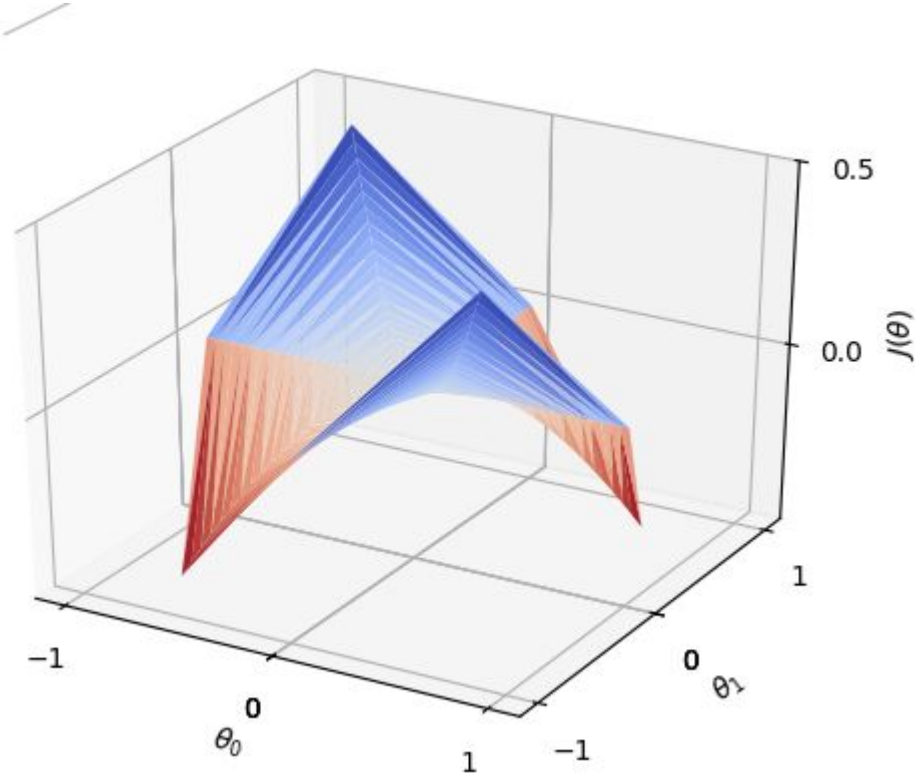
Topología del espacio de búsqueda: Caso óptimo



Topología del espacio de búsqueda: Valle



Topología del espacio de búsqueda: Saddle



Descenso de Gradiente Estocástico

- **Piedra angular en los métodos de Machine Learning**: La mayoría de los algoritmos implementados utilizan Gradiente Estocástico para encontrar la mejor combinación de parámetros.
- A diferencia de Descenso de Gradiente clásico, en SGD no se utilizan todos los datos de entrenamiento para actualizar los parámetros.
- Actualiza los parámetros una vez por cada elemento del conjunto de entrenamiento.
- Esto permite tener una tasa de actualización más alta que GD, siendo más eficiente para encontrar mínimos/máximos.

Descenso de Gradiente Estocástico

Tomemos un punto de inicio similar respecto al Descenso de Gradiente Clásico

$$\hat{\theta}_{j+1} \leftarrow \hat{\theta} - \alpha \nabla_{\theta} J(\theta; x_i, y_i)$$

El proceso de actualización respecto a parámetros candidatos, gradiente y función de pérdida se mantienen idénticos.

Descenso de Gradiente Estocástico

Tomemos un punto de inicio similar respecto al Descenso de Gradiente Clásico

$$\hat{\theta}_{j+1} \leftarrow \hat{\theta} - \alpha \nabla_{\theta} J(\theta; x_i, y_i)$$

A diferencia de GD Clásico, acá la evaluación de la función de pérdida se realiza por cada uno de los ejemplos en el conjunto de entrenamiento.

Variantes de SGD: Batch Training

- Entrenamiento por “partes”: Estimamos el error asociado para cada ejemplo en el conjunto de entrenamiento.
- Una vez que recolectamos información suficiente sobre el comportamiento de los ejemplos en la parte, actualizamos los parámetros y nuestra búsqueda.

Variantes de SGD: Mini Batch Training

- Escenarios conocidos:
  - Entrenamiento con todos los datos (GD/Batch Training).
  - Entrenamiento con cada uno de los datos de manera individual (SGD).
- Mini Batch se presenta como un punto intermedio entre ambos.
- Entrenamos con una partición específica de los datos de manera tal de actualizar los parámetros con más de un ejemplo de entrenamiento específico.

Variantes de SGD: Mini Batch Training

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(x^{j:j+b}, y^{j:j+b})$$

El proceso de actualización respecto a parámetros candidatos, gradiente y función de pérdida se mantienen idénticos.

Variantes de SGD: Mini Batch Training

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(x^{j:j+b}, y^{j:j+b})$$

La selección de ejemplos dependerá de un punto de partida j, y un rango superior b.

Solo seleccionamos el rango de datos, no los ejemplos en sí.

Variantes actuales de optimización

Variantes actuales: Momentum

- Presenta similitudes frente al Descenso de Gradiente Estocástico.
- La principal diferencia es que el Momentum controla la cantidad de movimiento generada por SGD.
- Es una **variante robusta** de SGD frente a la **inyección excesiva de aleatoriedad** en el proceso de búsqueda.

Variantes actuales: Momentum de Nesterov

- Problema con la optimización por el momentum: puede que la regularización de los movimientos de Momentum dificulte el proceso de estacionarse en un mínimo.
- **Solución propuesta por Nesterov**: generar una predicción de los futuros pasos de actualización en la búsqueda para evitar el exceso de inercia en el momentum.

Variantes actuales: Adagrad

- **Problema con la optimización con Nesterov**: No diferencia la frecuencia de actualización de los parámetros.
- Ventajas de Adagrad:
  - Facilita la especificación del Learning Rate a nivel de parámetro. Esta es una situación deseable cuando trabajamos con matrices dispersas.
  - A grandes rasgos, es una variante adaptativa de SGD, donde regularizamos el learning rate del gradiente.
  - No requiere de calibración detallada del Learning Rate.

Variantes actuales: Adadelta

- **Problema con la optimización con Adagrad**: adapta el learning rate en función a la suma de todos los gradientes anteriormente evaluados.
- El problema de esto es que sobreregulariza la búsqueda en la superficie de la función de pérdida.
- Adadelta busca adaptar el learning rate en función de un criterio arbitrario sobre la cantidad de gradientes anteriores a considerar en su paso adaptativo.
- Cuando Adadelta presenta una combinación específica de hiperparámetros, se conoce como **RMSprop**.

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com