# METODOS AVANZADOS BASADOS EN GRADIENTE

Melissa Macedo Ramos

January 2025

## 1 METODOS AVANZADOS BASADOS EN GRADIENTE

### 1.1 ¿Qué son los métodos basados en gradiente?

Los métodos basados en gradiente son algoritmos de optimización que utilizan la derivada o el gradiente de una función para encontrar sus mínimos o máximos. Se basan en la idea de que el gradiente de una función señala la dirección de mayor incremento, por lo que, para minimizar una función, se avanza en la dirección opuesta al gradiente.Estos métodos son ampliamente utilizados en diversas áreas como el aprendizaje automático, la optimización matemática y la inteligencia artificial, ya que permiten ajustar parámetros en modelos complejos de manera eficiente.

article [2024]Descenso gradual: ¡un algoritmo de optimización esencial!

Dado un problema de minimización de una función f(x), los métodos de gradiente siguen la ecuación de actualización:

$$x_{k+1} = x_k - \eta \nabla f(x_k)$$

Donde:

- $x_k$ es la iteración actual del parámetro a optimizar.
- $\eta$ es la tasa de aprendizaje (*learning rate*), que controla el tamaño del paso en cada iteración.
- $\nabla f(x_k)$ es el gradiente de la función en el punto $x_k$.

Este proceso se repite iterativamente hasta alcanzar un punto donde el gradiente sea cercano a cero, lo que indica un mínimo.

Imagine que tiene un problema de aprendizaje automático y desea entrenar su algoritmo con descenso de gradiente para minimizar su función de costo J(w, b) y alcanzar su mínimo local ajustando sus parámetros (w y b). La siguiente imagen muestra los ejes horizontales que representan los parámetros (w y b),

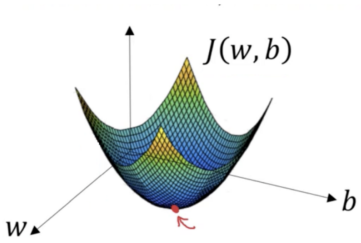


Figure 1: Caption

mientras que la función de costo J(w, b) se representa en los ejes verticales. article [Aug 01-2024]Matthew Urwin

### 1.1.1 Comparación con otros métodos de optimización

Estos métodos utilizan tanto la primera como la segunda derivada (Hessiano) de la función objetivo para guiar la optimización. Entre ellos se encuentran:

- Método de Newton: Utiliza la información del Hessiano para ajustar los pasos de actualización, lo que puede conducir a una convergencia más rápida. article [2021]Emiola, I., Adem, R. l:

- Métodos Cuasi-Newton (como BFGS): Aproximan el Hessiano para reducir el costo computacional, manteniendo algunas ventajas de los métodos de segundo orden.

**Ventajas:**
- Pueden converger en menos iteraciones debido a la utilización de información de curvatura.
- Son más efectivos en problemas donde el paisaje de la función objetivo es complejo y presenta curvaturas pronunciadas.

**Desventajas:**
- El cálculo y almacenamiento del Hessiano es computacionalmente costoso y requiere mucha memoria, especialmente para problemas de alta dimensión.
- Pueden ser inestables si el Hessiano no es positivo definido.

## 1.2 Gradiente Descendiente y sus Variantes

### 1.2.1 Gradiente Descendiente Estándar

El Gradiente Descendiente Estándar, también conocido como Batch Gradient Descent, es un método de optimización que actualiza los parámetros de un

modelo utilizando el gradiente de la función de costo calculado sobre el conjunto de datos completo. article [2021]Emiola, I., Adem, R. l:

**EJEMPLO**

**Paso 1: Calcular el Gradiente** El gradiente de la función $f(x)$ es la derivada:

$$\nabla f(x) = 2(x - 3)$$

Esta derivada nos indica en qué dirección debemos movernos para minimizar la función.

**Paso 2: Elegir un Punto Inicial** Tomemos un valor inicial $x_0 = 10$ (un punto alejado del mínimo).

Definimos la tasa de aprendizaje $\eta = 0.1$, que nos dice qué tan grande será cada paso

**Paso 3: Aplicar la Regla del Gradiente Descendiente** La actualización del valor de $x$ en cada iteración se hace con:

$$x_{\text{nuevo}} = x_{\text{actual}} - \eta \nabla f(x_{\text{actual}})$$

Ahora calculamos cada iteración

**conclusion** El Gradiente Descendiente Estándar permite encontrar el mínimo de la función f(x) siguiendo la dirección opuesta al gradiente, actualizando x en pequeños pasos.

### 1.2.2 Gradiente Descendiente Estocástico (SGD)

El Gradiente Descendiente Estocástico (SGD) es una variante del método de gradiente descendente estándar, donde, en lugar de usar todo el conjunto de datos para calcular el gradiente en cada iteración, se utiliza solo un ejemplo o una pequeña muestra aleatoria de los datos.

El SGD se basa en la misma idea básica del gradiente descendiente: actualizar los parámetros del modelo en la dirección opuesta al gradiente de la función de costo. Sin embargo, en lugar de calcular el gradiente en función de todos los datos, calcula el gradiente para un solo dato o un pequeño subconjunto (mini-lote) de los datos. article [2023]PROGRAMACIÓN MATEMÁTICA Métodos de optimización l

En el SGD, la actualización se realiza de la siguiente forma:

$$x_{k+1} = x_k - \eta \nabla f(x_k, x^{(i)})$$

Donde $x^{(i)}$ es un solo ejemplo de los datos de entrenamiento (o un mini-lote en variantes del SGD), y el gradiente $\nabla f(x_k, x^{(i)})$

**EJEMPLO**

Imaginemos que estamos tratando de minimizar una función de costo en un conjunto de datos que tiene 1000 muestras. En el Gradiente Descendiente Estándar, se calcularía el gradiente utilizando todas las 1000 muestras y luego se actualizarían los parámetros. Este proceso podría ser lento si el conjunto de datos es grande.

En cambio, con el SGD, en cada iteración se selecciona aleatoriamente una muestra del conjunto de datos y se calcula el gradiente usando solo esa muestra. Esto acelera las actualizaciones, pero introduce algo de ruido, ya que el gradiente calculado a partir de solo una muestra no es tan preciso como el gradiente calculado sobre todo el conjunto de datos.

### 1.2.3 Gradiente Descendiente Mini-Batch

El Gradiente Descendiente Mini-Batch es una variante del Gradiente Descendiente Estocástico (SGD), que intenta combinar las ventajas del Gradiente Descendiente Estándar y del SGD, al usar una pequeña cantidad de ejemplos (mini-lote) en cada iteración para calcular el gradiente, en lugar de usar todo el conjunto de datos como en el gradiente descendente estándar o solo un único ejemplo como en el SGD. article [2023]PROGRAMACIÓN MATEMÁTICA Métodos de optimización l

**EJEMPLO:** Imaginemos que tenemos un conjunto de datos con 1000 ejemplos y queremos usar el Gradiente Descendiente Mini-Batch para entrenar un modelo. Si decidimos usar un mini-lote de tamaño 100, el conjunto de datos se dividirá en 10 mini-lotes, y en cada iteración, el gradiente se calculará usando 100 ejemplos seleccionados aleatoriamente de estos 1000 datos.

Después de cada mini-lote, los parámetros del modelo se actualizan, y luego se pasa al siguiente mini-lote. Matemáticamente, la actualización se realiza de la siguiente manera:

$$x_{k+1} = x_k - \eta \nabla f\left(x_k, \{x^{(i_1)}, x^{(i_2)}, \ldots, x^{(i_m)}\}\right)$$

Donde:

- $\eta$ es la tasa de aprendizaje. - $\nabla f\left(x_k, \{x^{(i_1)}, x^{(i_2)}, \ldots, x^{(i_m)}\}\right)$ es el gradiente calculado en función del mini-lote de tamaño $m$.

## 1.3 Aplicaciones Prácticas del metodo de Gradiente

### 1.3.1 Entrenamiento de Redes Neuronales Profundas

El entrenamiento de redes neuronales profundas es uno de los campos más comunes y efectivos en los que se aplica el método de gradiente descendente. Las redes neuronales profundas (también conocidas como redes profundas o DNN, por sus siglas en inglés) son redes con múltiples capas ocultas que permiten aprender representaciones complejas y abstraídas de los datos. El entrenamiento de estas redes se basa en el algoritmo de retropropagación, que es una forma de gradiente descendente aplicada a cada capa de la red.

Durante el proceso de entrenamiento, el gradiente descendente se utiliza para ajustar los pesos de la red neuronal, minimizando una función de pérdida. A medida que las redes neuronales se vuelven más profundas (con más capas ocultas), los gradientes se calculan y actualizan de manera iterativa en cada capa a través de retropropagación, lo que permite que la red aprenda patrones y representaciones complejas a partir de datos no estructurados como imágenes, texto

o sonidos.article [2024]Descenso gradual: ¡un algoritmo de optimización esencial!

### 1.3.2 Optimización de Funciones en Problemas de Optimización Convexa y No Convexa

El método de gradiente es muy utilizado en problemas de optimización convexa y no convexa.

En optimización convexa, el objetivo es minimizar una función convexa, que garantiza que cualquier mínimo local es también un mínimo global. Esto hace que el gradiente descendente sea particularmente efectivo, ya que es más probable que el algoritmo encuentre un mínimo global de forma eficiente.

En optimización no convexa, los problemas son más complejos, ya que la función objetivo puede tener múltiples mínimos locales. Aunque el gradiente descendente no garantiza encontrar el mínimo global en estos casos, sigue siendo muy útil debido a su capacidad para encontrar buenos mínimos locales, especialmente si se combinan con técnicas como inicio aleatorio, momentum, o técnicas de regularización.

Ambos tipos de problemas, convexos y no convexos, surgen en una amplia gama de aplicaciones, como la optimización de parámetros en modelos de aprendizaje automático o la programación en ingeniería.article [2024]Descenso gradual: ¡un algoritmo de optimización esencial!

### 1.3.3 Aplicaciones en Regresión y Clasificación

El gradiente descendente también se aplica de manera efectiva en regresión y clasificación:

En regresión, el objetivo es modelar la relación entre una o más variables independientes (features) y una variable dependiente (target). El gradiente descendente se usa para minimizar la función de pérdida, como el error cuadrático medio (MSE), para ajustar los parámetros del modelo (por ejemplo, los coeficientes en la regresión lineal).

En clasificación, el gradiente descendente se utiliza para optimizar modelos como regresiones logísticas o máquinas de soporte vectorial (SVM). En este caso, la función de pérdida más común es la entropía cruzada, que mide la diferencia entre las probabilidades predichas por el modelo y las etiquetas reales. Este tipo de optimización permite ajustar los parámetros de modelos que predicen probabilidades de clase.

Ambas aplicaciones utilizan gradiente descendente para ajustar los parámetros del modelo iterativamente, mejorando las predicciones del modelo a lo largo de las iteraciones.article [2024]Descenso gradual: ¡un algoritmo de optimización esencial!

#### 1.3.4 Implementación en Librerías como TensorFlow, PyTorch y Scikit-Learn

Las librerías de TensorFlow, PyTorch, y Scikit-Learn son ampliamente utilizadas para implementar el gradiente descendente en tareas de aprendizaje automático y aprendizaje profundo:

**TensorFlow:** Es una de las librerías más populares para la construcción y entrenamiento de redes neuronales profundas. Ofrece una implementación eficiente del gradiente descendente y sus variantes, como SGD, Adam, y RMSprop, utilizando optimizadores de alto rendimiento y soporte para GPU/TPU.
**PyTorch:** Al igual que TensorFlow, PyTorch es otro marco de trabajo para redes neuronales profundas. PyTorch proporciona un enfoque dinámico y flexible para definir redes y realizar optimizaciones utilizando gradiente descendente. Su API es conocida por ser más fácil de usar y más intuitiva para los investigadores.
**Scikit-Learn:** Es una librería ampliamente utilizada para tareas de aprendizaje automático clásico. Aunque no está diseñada específicamente para redes neuronales profundas, Scikit-Learn incluye implementaciones de gradiente descendente para regresión logística, máquinas de soporte vectorial, y otros algoritmos de optimización que se benefician de este método. article [2024]Descenso gradual: ¡un algoritmo de optimización esencial!

# 2 TECNICAS BASADAS EN MOMENTO (Aceleracion de Nesterov

## 2.1 Introduccion

La aceleración de Nesterov es una técnica basada en el concepto de momento, utilizada para mejorar el rendimiento de los métodos de optimización basados en gradiente. Su objetivo es acelerar la convergencia hacia el óptimo de una función objetivo, especialmente en problemas donde las soluciones son complicadas de alcanzar debido a terrenos con curvaturas pronunciadas o mesetas.

En optimización, las técnicas basadas en momento utilizan el concepto de acumulación de velocidades para mejorar la eficiencia del gradiente descendente. En lugar de avanzar directamente en la dirección del gradiente en cada iteración, se consideran los gradientes anteriores para darle al sistema un "impulso" que permita un avance más rápido en las regiones planas y una mayor estabilidad en zonas oscilantes.

La aceleración de Nesterov es una mejora del gradiente descendente con momento que introduce la idea de predecir la posición futura antes de calcular el gradiente. Esto ayuda a ajustar el paso de forma más precisa y evita que el optimizador "sobrepase" el óptimo.

## 2.2 Fundamentos Matemáticos de la Aceleración de Nesterov

### 2.2.1 Principio de Aceleración de Nesterov

El principio de aceleración de Nesterov se refiere a un enfoque mejorado de optimización basado en el método de gradiente descendente. Mientras que el gradiente descendente tradicional solo se mueve en la dirección del gradiente en cada iteración, la técnica de Nesterov introduce una aceleración al utilizar una estimación de la próxima posición antes de calcular el gradiente. Esta técnica mejora la velocidad de convergencia al hacer que el algoritmo "adelante" el paso antes de hacer la actualización final, lo que permite un mejor aprovechamiento de la información disponible.article [2024]Descenso gradual: ¡un algoritmo de optimización esencial!

### 2.2.2 Derivación Matemática de la Técnica de Nesterov

La derivación matemática de la técnica de aceleración de Nesterov se basa en el método de gradiente con momento, pero con un ajuste para realizar una predicción más informada. La derivación de Nesterov se describe a través de la siguiente forma: article [2024]Descenso gradual: ¡un algoritmo de optimización esencial!

Consideramos el problema de optimización donde queremos minimizar una función f(x) con respecto a 'x'. El gradiente descendente clásico se actualiza según:

$$x_{k+1} = x_k - \eta \nabla f(x_k)$$

donde $\eta$ es la tasa de aprendizaje y $\nabla f(x_k)$ es el gradiente de la función en el punto $x_k$.

Nesterov introduce la predicción de la próxima posición antes de actualizar el valor de $x$. La fórmula de la aceleración de Nesterov se da por:

$$v_{k+1} = \beta v_k + \eta \nabla f(x_k - \beta v_k)$$

$$x_{k+1} = x_k - v_{k+1}$$

Aquí, $v_k$ representa la "velocidad" o el término de momento, $\beta$ es un parámetro de amortiguamiento (generalmente cercano a 1), y el gradiente $\nabla f(x_k - \beta v_k)$ se calcula en el punto $x_k - \beta v_k$, la predicción de la próxima posición.

### 2.2.3 ejemplo de aceleracion de Nesterov

El principio de aceleración de Nesterov se puede ejemplificar en una situación sencilla donde buscamos minimizar una función cuadrática. A continuación, se presenta un ejemplo básico que ilustra cómo se realiza la actualización de los parámetros utilizando este método:

Consideremos la función cuadrática de un solo parámetro $f(x) = (x-3)^2$, cuyo mínimo se encuentra en $x = 3$. Queremos utilizar el algoritmo de Nesterov para encontrar este mínimo de manera más eficiente.

**Paso 1: Inicialización**

1. Empezamos con un valor inicial $x_0 = 0$ (un valor alejado del mínimo).
2. La tasa de aprendizaje $\eta = 0.1$.
3. El parámetro de momento $\beta = 0.9$, que es un valor comúnmente utilizado.

**Paso 2: Cálculo del Gradiente**

El gradiente de la función es simplemente la derivada de $f(x)$:

$$\nabla f(x) = 2(x-3)$$

**Paso 3: Aplicación de la Técnica de Nesterov**

1. **Inicialización del momento**: Comenzamos con $v_0 = 0$, ya que no hay información de las iteraciones anteriores.
2. **Iteración 1**:
   (a) La predicción de la próxima posición es:
   $$\hat{x}_1 = x_0 - \beta v_0 = 0 - 0.9(0) = 0$$
   (b) Calculamos el gradiente en la posición predicha $\hat{x}_1 = 0$:
   $$\nabla f(\hat{x}_1) = 2(0-3) = -6$$
   (c) Actualizamos el valor de $v_1$:
   $$v_1 = \beta v_0 + \eta \nabla f(\hat{x}_1) = 0.9(0) + 0.1(-6) = -0.6$$
   (d) Finalmente, actualizamos $x_1$:
   $$x_1 = x_0 - v_1 = 0 - (-0.6) = 0.6$$
3. **Iteración 2**:
   (a) La predicción de la próxima posición es:
   $$\hat{x}_2 = x_1 - \beta v_1 = 0.6 - 0.9(-0.6) = 1.14$$
   (b) Calculamos el gradiente en la posición predicha $\hat{x}_2 = 1.14$:
   $$\nabla f(\hat{x}_2) = 2(1.14-3) = -3.72$$
   (c) Actualizamos el valor de $v_2$:
   $$v_2 = \beta v_1 + \eta \nabla f(\hat{x}_2) = 0.9(-0.6) + 0.1(-3.72) = -0.906$$

(d) Finalmente, actualizamos $x_2$:

$$x_2 = x_1 - v_2 = 0.6 - (-0.906) = 1.506$$

A medida que avanzamos con más iteraciones, el valor de $x_k$ se acercará al mínimo $x = 3$, y la técnica de Nesterov acelerará la convergencia en comparación con el gradiente descendente tradicional. Esto es especialmente útil cuando se trabaja con funciones más complejas, como en el caso de redes neuronales profundas, donde la convergencia eficiente es crucial.

# 3 METODOS EN SEGUNDO ORDEB(Newton, enfoques cuasi-Newton

## 3.1 ¿Que es el algoritmo Quasi-Newton?

El algoritmo Quasi-Newton es un método de optimización que se utiliza principalmente para resolver problemas de optimización no lineal sin restricciones. Es una opción popular en varios campos, como la estadística, análisis de los datos, y la ciencia de datos debido a su eficiencia y eficacia en la aproximación de la matriz de Hesse, que es crucial para determinar la curvatura de la función objetivo. A diferencia del método tradicional de Newton, que requiere el cálculo de las segundas derivadas, el algoritmo Quasi-Newton construye una aproximación de la matriz de Hesse utilizando solo las primeras derivadas, lo que lo hace computacionalmente menos intensivo. article [2020]APPLIED STATISTICS: Data Analysis VOLUME I: ANALYSISl

## 3.2 Formula matematicamente

El método de Newton es un algoritmo iterativo que se utiliza para encontrar raíces de una función o para minimizar funciones. En el contexto de optimización, se utiliza para encontrar el mínimo de una función diferenciable. La fórmula para actualizar la iteración en el método de Newton es la siguiente:

$$x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k)$$

donde:

1. $x_k$ es la estimación actual del parámetro,
2. $\nabla f(x_k)$ es el gradiente de la función en $x_k$,
3. $H(x_k)$ es la matriz Hessiana de la función en $x_k$, que es la matriz de segundas derivadas de la función.

   article [2020]APPLIED STATISTICS: Data Analysis VOLUME I: ANALYSISl

## 3.3 Características principales del algoritmo Quasi-Newton

Una de las características clave del algoritmo Quasi-Newton es su capacidad de converger más rápido que los métodos de primer orden, como el descenso de gradiente. Esto se debe en gran medida a su uso de información de segundo orden a través del hessiano aproximado. Además, el algoritmo es menos sensible a la elección de la estimación inicial en comparación con otros métodos. Los métodos Quasi-Newton más utilizados incluyen el algoritmo Broyden-Fletcher-Goldfarb-Shanno (BFGS) y su variante de memoria limitada, L-BFGS, que es particularmente útil para problemas de optimización a gran escala.article [2020]APPLIED STATISTICS: Data Analysis VOLUME I: ANALYSISl

## 3.4

sectionAplicaciones del algoritmo Quasi-Newton

El algoritmo Quasi-Newton encuentra aplicaciones en varios dominios, incluidos máquina de aprendizaje, econometría e ingeniería. En el aprendizaje automático, se utiliza a menudo para entrenar modelos en los que se requiere la optimización de una función de pérdida. Su eficiencia lo hace adecuado para el análisis de datos de alta dimensión, donde los métodos tradicionales pueden tener dificultades. En econometría, el algoritmo se utiliza para estimar parámetros en modelos complejos, lo que permite a los investigadores obtener información de grandes conjuntos de datos.article [2020]APPLIED STATISTICS: Data Analysis VOLUME I: ANALYSISl

### 3.4.1 Ventajas de utilizar el algoritmo Quasi-Newton

Una de las principales ventajas del algoritmo Quasi-Newton es su equilibrio entre eficiencia computacional y velocidad de convergencia. Al evitar el cálculo directo de las derivadas secundarias, reduce significativamente la carga computacional y, al mismo tiempo, aprovecha la información de curvatura. Esto lo hace particularmente ventajoso en escenarios donde las evaluaciones de funciones son costosas o requieren mucho tiempo. Además, los métodos Quasi-Newton son robustos y pueden manejar una amplia variedad de problemas de optimización de manera efectiva.article [2020]APPLIED STATISTICS: Data Analysis VOLUME I: ANALYSISl

### 3.4.2 Limitaciones del algoritmo Quasi-Newton

A pesar de sus ventajas, el algoritmo Quasi-Newton tiene limitaciones. La precisión de la aproximación hessiana puede degradarse en casos en los que la función objetivo es altamente no lineal o tiene discontinuidades.

Además, si bien el algoritmo es generalmente robusto, aún puede converger a puntos de silla o mínimos locales, especialmente en entornos complejos. Los usuarios deben ser conscientes de estos posibles obstáculos y es posible que deban implementar estrategias como reinicios o métodos híbridos para mejorar el rendimiento. article [2020]APPLIED STATISTICS: Data Analysis VOLUME I: ANALYSISl

## 3.5 Ejemplo

Ejemplo práctico de Método de Newton

Consideremos la función cuadrática simple:

$$f(x) = (x-3)^2$$

El mínimo de esta función está en $x = 3$. Para ilustrar el método de Newton, seguimos estos pasos:

(a) **Paso 1: Derivadas de la función**
La primera derivada (gradiente) es:

$$\nabla f(x) = 2(x-3)$$

La segunda derivada (Hessiana) es:

$$H(x) = 2$$

(b) **Paso 2: Inicialización**
Tomemos un valor inicial $x_0 = 0$, es decir, empezamos con una estimación de $x$ lejos del mínimo.

(c) **Paso 3: Aplicar la fórmula de Newton**
El siguiente paso se calcula de la siguiente manera:

$$x_1 = x_0 - \frac{\nabla f(x_0)}{H(x_0)} = 0 - \frac{2(0-3)}{2} = 3$$

Así que en una sola iteración, el método de Newton encuentra el mínimo $x = 3$.

article [2020]APPLIED STATISTICS: Data Analysis VOLUME I: ANALYSISl

# 4 PROBLEMAS PRACTICOS RESTRICCIONES DE MEMORIA Y COMPLEJIDAD-COMPUTACIONAL

**Complejidad computacional**

La complejidad computacional es un tema muy importante y se refiere a la estimación de qué tan difícil o fácil se puede resolver computacionalmente un problema. Sin embargo, la complejidad computacional no responderá al tiempo de cálculo real que se tomará para resolver un problema particular o una instancia de problema porque las implementaciones reales dependerán de otros factores, como el hardware y el software. Por tanto, la complejidad es una estimación del orden o número de operaciones computacionales, más que del tiempo. En términos generales, la complejidad computacional y las clases están estrechamente asociadas con las máquinas de Turing. Una máquina de Turing puede considerarse como una máquina abstracta que puede leer una entrada y manipular una operación a la vez, de acuerdo con reglas predefinidas. En general, una máquina de Turing es capaz de realizar el cálculo de cualquier función computable. Dado que las reglas son fijas, se lleva a cabo una acción a la vez, por lo que dicha máquina de Turing se vuelve determinista, llamada máquina de Turing determinista.

*****

Optimización de Algoritmos en Machine Learning **Problema 1:** Entrenar un modelo de **Red Neuronal** en una computadora con GPU de **2 GB de VRAM** para clasificar imágenes.

**Restricción:** Las redes neuronales requieren mucha memoria para almacenar **pesos, gradientes y activaciones**.

**Solución:**

(a) **Reducir la dimensión de entrada:** Aplicar técnicas como **PCA** o **compresión de imágenes**.

(b) **Mini-batch training:** Procesar los datos en **lotes pequeños** en lugar de toda la base de datos.

(c) **Cuantización de pesos:** Reducir la precisión de los pesos de **32 bits** a **16 bits** o **8 bits**.

(d) **Pruning:** Eliminar conexiones innecesarias para reducir el tamaño de la red.

*

1. Optimización de Algoritmos en Machine Learning **Problema:** Entrenar un modelo de **Red Neuronal** en una computadora con GPU de **2 GB de VRAM** para clasificar imágenes.

**Restricción:** Las redes neuronales requieren mucha memoria para almacenar **pesos, gradientes y activaciones**.

**Solución:**

(a) **Reducir la dimensión de entrada:** Aplicar técnicas como **PCA** o **compresión de imágenes**.

(b) **Mini-batch training:** Procesar los datos en **lotes pequeños** en lugar de toda la base de datos.

(c) **Cuantización de pesos:** Reducir la precisión de los pesos de **32 bits** a **16 bits** o **8 bits**.

(d) **Pruning:** Eliminar conexiones innecesarias para reducir el tamaño de la red.

*

2. Algoritmos en Procesamiento de Big Data **Problema:** Procesar grandes volúmenes de datos en un sistema con recursos limitados de CPU y memoria RAM.

**Restricción:** Los algoritmos tradicionales pueden volverse ineficientes al escalar grandes conjuntos de datos.

**Solución:**

(a) **Uso de procesamiento distribuido:** Implementar herramientas como **Apache Spark** o **Hadoop**.

(b) **Filtrado de datos:** Aplicar **técnicas de muestreo** o selección de características antes del procesamiento.

(c) **Optimización de estructuras de datos:** Uso de estructuras más eficientes como **árboles de búsqueda** o **hashing**.

(d) **Estrategias de paginación y memoria virtual:** Cargar datos en memoria solo cuando sea necesario.

*

3. Compresión de Datos en Dispositivos con Bajo Almacenamiento **Problema:** Almacenar imágenes y videos en dispositivos con almacenamiento limitado.

**Restricción:** Los formatos de archivo sin compresión ocupan demasiado espacio.

**Solución:**

(a) **Compresión con pérdida:** Uso de formatos como **JPEG** para imágenes y **MP4** para videos.

(b) **Compresión sin pérdida:** Implementar formatos como **PNG** o **FLAC** según la necesidad.

(c) **Codificación eficiente:** Uso de **algoritmos de Huffman** o **Run-Length Encoding (RLE)**.

(d) **Almacenamiento en la nube:** Sincronización de archivos con servicios como **Google Drive** o **Dropbox**.

*

4. Simulación de Sistemas Complejos con Limitaciones Computacionales **Problema:** Realizar simulaciones de modelos físicos o climáticos en computadoras con recursos limitados.

**Restricción:** Las simulaciones requieren una gran cantidad de cálculos y almacenamiento de datos intermedios.

**Solución:**

(a) **Aproximaciones numéricas:** Implementación de métodos como **Monte Carlo** o **diferencias finitas**.

(b) **Uso de supercomputadoras o clústeres:** Ejecutar procesos en paralelo mediante **HPC (High Performance Computing)**.

(c) **Reducir resolución espacial o temporal:** Utilizar mallas más grandes en simulaciones de dinámica de fluidos.

(d) **Uso de modelos reducidos:** Aplicar técnicas de **reducción de dimensionalidad** para minimizar cálculos.

# 5 EVALUACION DE DATOS DE ENCUESTAS PERUANAS

*

Rjemplo: Análisis de Factores Socioeconómicos que Influyen en el Nivel Educativo en Perú usando Métodos Basados en Gradiente

**1. Contexto del Problema** En Perú, la educación es un factor clave en el desarrollo social y económico. Sin embargo, existen desigualdades en

el acceso y nivel educativo alcanzado por la población. Usando datos de encuestas nacionales como la Encuesta Nacional de Hogares (ENAHO), se busca identificar los factores socioeconómicos que influyen en el nivel educativo de los ciudadanos.

**2. Datos Utilizados** Se pueden utilizar variables de la ENAHO como:

- Nivel educativo alcanzado (variable objetivo) - Ingreso mensual del hogar - Ubicación geográfica (urbano/rural) - Edad y género del encuestado - Acceso a servicios básicos (agua, electricidad, internet) - Ocupación de los padres

**Método Avanzado Basado en Gradiente** Para analizar los datos, se puede emplear el método de Gradiente Boosting Machine (GBM), como XGBoost, LightGBM o CatBoost, que optimizan la clasificación o regresión utilizando técnicas de descenso de gradiente. El modelo busca minimizar el error iterativamente ajustando árboles de decisión. **Archivo de excel 'Enaho01-2022-100.csv encuesta.csv**'

*

codigo python

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Cargar datos (simulación, en un caso real se usaría la ENAHO)
data = pd.read_csv("encuesta_peruana.csv")

# Definir variables predictoras y objetivo
X = data[['ingreso', 'zona', 'edad', 'genero', 'servicios_basicos', 'ocupacion_padres']
y = data['nivel_educativo']

# Dividir en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Crear modelo XGBoost
model = xgb.XGBClassifier(objective="multi:softmax", num_class=5, eval_metric="mlogloss

# Entrenar modelo
model.fit(X_train, y_train)

# Predecir y evaluar
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo: {accuracy:.2f}")
```

### Interpretación de Resultados

Se puede analizar la importancia de las variables para identificar cuáles tienen mayor impacto en el nivel educativo. Se pueden visualizar relaciones entre el nivel educativo y factores como ingreso y zona geográfica. Si la precisión del modelo es alta, se pueden tomar decisiones de política pública basadas en estos resultados.

**Aplicaciones y Conclusión** Este análisis ayuda a entender las barreras socioeconómicas que limitan el acceso a la educación en Perú y puede ser utilizado para desarrollar políticas de inclusión educativa. Además, demuestra cómo los métodos basados en gradiente pueden ser útiles para analizar datos de encuestas de manera eficiente.