**Facultad de Ingeniería**
**Asignatura Algoritmos de Optimización**

## Unidad de aprendizaje 1: Fundamentos y Métodos Clásicos de Optimización

## Introducción

Esta unidad introduce los conceptos fundamentales de la optimización, abordando la clasificación de los problemas, sus componentes matemáticos esenciales y los métodos clásicos más representativos, como el gradiente descendente y la programación lineal. El propósito es que el estudiante adquiera una comprensión conceptual y técnica de estas herramientas, y sea capaz de aplicarlas en la formulación y solución de problemas reales relacionados con la inteligencia artificial.

## 1. Introducción a la optimización y su relevancia en IA

### ¿Qué es optimizar?

Antes de entrar en definiciones técnicas, piensa en esta situación:

> Tienes que ir de tu casa a la universidad. Hay varias rutas posibles. Una es más rápida, otra tiene menos tráfico, otra pasa por un lugar más bonito. Según lo que quieras priorizar (tiempo, comodidad, paisaje), eliges **la mejor opción posible**.

Esa elección que haces es, en términos simples, un **problema de optimización**.

En matemáticas (y en inteligencia artificial), **optimizar** significa buscar **la mejor solución posible** a un problema, bajo ciertas condiciones.

### ¿Qué significa "la mejor solución"?

Depende del problema. En IA, podría significar:

- Obtener el **menor error posible** al entrenar un modelo de predicción.
- Maximizar la **precisión** de un clasificador.
- Reducir el **tiempo de entrenamiento** sin perder calidad en los resultados.
- Encontrar los **mejores parámetros** que permiten que una red neuronal funcione de manera óptima.

Por eso, **optimizar** siempre implica **definir un objetivo** (algo que queremos mejorar) y luego encontrar **qué decisiones debemos tomar** para lograrlo.

### La optimización en Inteligencia Artificial

En IA, optimizar no es un complemento: es el **corazón** de muchas tareas. Cuando entrenamos un modelo de machine learning, lo que hacemos realmente es **ajustar los parámetros internos** para que la salida (predicción) sea lo más precisa posible.

Veamos un ejemplo sencillo:

> Estás construyendo un modelo de regresión lineal que predice el precio de una casa a partir de su tamaño.
> El modelo empieza con una línea recta "inicial", pero seguramente esa línea no representa bien los datos. Entonces, el algoritmo **ajusta la pendiente y la intersección** de esa línea para que se acerque lo mejor posible a los puntos reales.
> ¿Cómo sabe cuánto ajustar? Mediante **optimización**: busca los valores de los parámetros que **minimicen el error** entre las predicciones y los datos reales.

**Componentes clave de un problema de optimización**

En términos formales, optimizar significa:

- **Definir una función objetivo**: Es aquello que queremos minimizar o maximizar (por ejemplo, el error cuadrático medio).
- **Definir variables de decisión**: Son los valores que podemos cambiar para influir en el resultado (por ejemplo, los pesos de un modelo).
- **Incluir restricciones (si existen)**: Son condiciones que se deben cumplir (por ejemplo, que la suma de probabilidades sea igual a 1).

En IA, esto se traduce en preguntas como:

- ¿Qué estoy tratando de mejorar (precisión, velocidad, consumo de memoria)?
- ¿Qué puedo modificar para lograrlo (número de capas, tasa de aprendizaje, selección de variables)?
- ¿Qué límites o restricciones tengo (tiempo de ejecución, datos disponibles, capacidad del hardware)?

**¿Por qué es importante en IA?**

Porque **la mayoría de los algoritmos de IA son entrenados mediante procesos de optimización**. Incluso los modelos más avanzados, como redes neuronales profundas, modelos de boosting o transformers, **no funcionan si no se optimizan bien**.

Además, hay tareas más allá del entrenamiento que también dependen de la optimización:

- Elegir los mejores hiperparámetros (grid search, random search)
- Seleccionar subconjuntos de características relevantes
- Agrupar datos sin etiquetas (clustering)
- Tomar decisiones autónomas (robótica, planificación)

**Reflexión final**

Aprender optimización no es solo aprender una técnica matemática. Es adquirir una herramienta para **tomar mejores decisiones**, de manera eficiente y con fundamento, dentro de sistemas inteligentes.

Durante esta unidad, vas a descubrir cómo funcionan los métodos clásicos de optimización, cómo se formulan los problemas, y cómo puedes resolverlos con herramientas computacionales.

## 2. Tipos de problemas: deterministas, estocásticos, lineales, no lineales

**¿Por qué es importante clasificar un problema de optimización?**

Antes de resolver un problema de optimización, necesitamos entender su naturaleza matemática. No todos los problemas se resuelven igual. Algunos se pueden resolver con métodos exactos, otros requieren aproximaciones o algoritmos inspirados en la naturaleza.

Saber si un problema es **determinista** o **estocástico**, **lineal** o **no lineal**, **con variables discretas o continuas, con o sin restricciones**, determina qué técnicas podemos aplicar y qué tan difícil será resolverlo.

**Optimización determinista vs. estocástica**

- **Determinista**

En este tipo de problemas, **todo está definido con certeza**. Si ingresas los mismos datos, obtienes siempre el mismo resultado. No hay elementos de azar.

Ejemplo (IA):

Ajustar una línea de regresión con mínimos cuadrados. Dado el conjunto de datos, siempre obtendrás los mismos coeficientes óptimos.

Se usan métodos como: gradiente descendente, programación lineal, métodos de Newton.

- **Estocástica**

Aquí, el problema involucra **elementos aleatorios o inciertos**. El resultado puede cambiar incluso con los mismos datos de entrada, porque el algoritmo incluye procesos aleatorios.

Ejemplo (IA):

Un algoritmo genético que busca el mejor conjunto de características para un modelo. El resultado puede variar entre ejecuciones, porque incluye mutación y cruza aleatoria.

Se usan métodos como: recocido simulado, algoritmos evolutivos, enjambre de partículas (PSO).

**Optimización lineal vs. no lineal**

- **Lineal**

Un problema es **lineal** si tanto la función objetivo como todas las restricciones son expresiones lineales (no tienen potencias, productos entre variables, logaritmos, etc.).

Ejemplo (IA):

Asignar recursos computacionales entre varios modelos para maximizar su rendimiento, bajo restricciones de tiempo y memoria.

$$\max Z = 5x + 4y$$
$$\text{sujeto a: } x + y \leq 10, \quad x, y \geq 0$$

Se resuelve con programación lineal y método simplex.

- **No lineal**

Un problema es **no lineal** si la función objetivo o alguna restricción tiene relaciones no lineales (por ejemplo, cuadráticas, exponenciales, logarítmicas).

Ejemplo (IA):

Entrenar una red neuronal: la función de pérdida depende de una combinación compleja de pesos, activaciones y derivadas. La superficie de error tiene valles, mesetas y múltiples mínimos.

Se resuelve con métodos como: gradiente descendente, BFGS, descenso conjugado, o incluso heurísticas si el problema es muy difícil.

**Variables discretas vs. continuas**

- **Variables continuas**

Las variables pueden tomar **cualquier valor real** dentro de un intervalo.

Ejemplo (IA):

Ajustar los pesos de una red neuronal: pueden valer 0.234, -1.5, 3.14, etc.

Se usan métodos que trabajan con derivadas y gradientes.

- **Variables discretas**

Las variables solo pueden tomar ciertos valores específicos (enteros, booleanos, categorías).

Ejemplo (IA):

Elegir qué características incluir en un modelo de clasificación. Cada variable tiene valor 1 (incluir) o 0 (excluir).

Se usan algoritmos como búsqueda exhaustiva, programación entera, algoritmos genéticos.

**Con o sin restricciones**

- **Con restricciones**: El problema incluye condiciones que deben cumplirse (por ejemplo, que el uso de memoria no supere cierto límite).
- **Sin restricciones**: Solo importa la función objetivo; no hay límites definidos para las variables.

**Resumen visual**

| **Tipo de problema** | **¿Qué lo caracteriza?** | **¿Qué técnicas aplicar?** |
|---|---|---|
| Determinista | Siempre da el mismo resultado | Gradiente descendente, métodos clásicos |
| Estocástico | Incluye aleatoriedad o simulación | Algoritmos evolutivos, heurísticas |
| Lineal | Función y restricciones son lineales | Programación lineal, método simplex |
| No lineal | Relación no lineal entre variables | Métodos numéricos, redes neuronales |
| Continuo | Variables con infinitos valores posibles | Derivadas, optimización continua |
| Discreto | Variables limitadas a ciertos valores | Combinatoria, programación entera |

**Conclusión**

Saber qué tipo de problema tienes enfrente es como hacer el diagnóstico correcto antes de aplicar un tratamiento.

En inteligencia artificial, esto es vital para elegir la técnica de optimización adecuada y no perder tiempo (ni potencia computacional) aplicando un método que no corresponde.

## 3. Funciones objetivo y restricciones

### ¿Qué es una función objetivo?

La **función objetivo** (también llamada función de costo o función de pérdida en IA) es el corazón de un problema de optimización. Es la expresión matemática que define lo que queremos mejorar.

> En pocas palabras, la función objetivo nos dice: “Esto es lo que quiero **minimizar** (como el error) o **maximizar** (como la precisión).”

En optimización, casi siempre trabajamos con problemas de **minimización**, incluso cuando queremos “maximizar” algo (porque maximizar una función es lo mismo que minimizar su inversa).

### ¿Cómo se ve una función objetivo?

Ejemplo general de un problema de minimización:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Aquí:

- $x$ son las variables de decisión (lo que podemos cambiar)
- $f(x)$ es la función objetivo (lo que queremos minimizar)

### Ejemplo aplicado en IA: regresión lineal

Supongamos que estamos ajustando una línea para predecir el precio de una casa según sus metros cuadrados. Nuestra función objetivo puede ser:

$$f(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Donde:

- $y_i$ es el valor real
- $\hat{y}_i = w_0 + w_1 x_i$ es la predicción del modelo
- $w$ son los parámetros (pendiente e intersección) que queremos ajustar

Aquí, lo que queremos es **minimizar el error cuadrático medio (MSE)** entre las predicciones del modelo y los datos reales.

**¿Qué son las restricciones?**

Las **restricciones** son **las condiciones que deben cumplirse mientras se optimiza**. No basta con encontrar la mejor solución, sino que esta **debe estar dentro de ciertos límites o reglas**.

En términos matemáticos, un problema con restricciones se ve así:

$$\min f(x) \quad \text{sujeto a:} \begin{cases} g_i(x) \leq 0 & \text{(restricciones de desigualdad)} \\ h_j(x) = 0 & \text{(restricciones de igualdad)} \end{cases}$$

**¿Para qué sirven las restricciones?**

Sirven para **modelar las limitaciones reales del problema**. Por ejemplo, en IA:

| Situación | Restricción posible |
|---|---|
| Que el modelo no tarde más de 2 horas en entrenarse | Tiempo de cómputo limitado |
| Que la suma de probabilidades sea igual a 1 | $\Sigma_{iPi} = 1$ |
| Que una variable esté entre 0 y 1 | $0 \leq x \leq 1$ |
| Que no se usen más de 5 características | $\Sigma x_i \leq 5$ |

Estas restricciones ayudan a que la solución no solo sea buena, sino **viable, ética o eficiente**.

**¿Qué pasa si un problema no tiene restricciones?**

Se llama **problema sin restricciones**. En IA, esto ocurre a menudo durante el entrenamiento inicial de un modelo, donde el único objetivo es minimizar la función de pérdida, sin ningún límite sobre los valores de los parámetros.

**Ejemplo**: entrenamiento de un perceptrón simple. Solo nos importa reducir el error.

**Visualizando un problema con función objetivo y restricciones**

Imagina una hoja de papel con un paisaje montañoso dibujado en 3D (la función objetivo). Tu tarea es encontrar el punto más bajo (mínimo).

Ahora imagina que hay una cinta delimitando un área: **ese es el espacio de restricciones**.

Tienes que encontrar el punto más bajo **dentro de esa área permitida**.

Esa es la esencia de la optimización con restricciones: **buscar el mejor resultado, dentro de lo permitido**.

**En resumen**

- La **función objetivo** nos dice qué estamos optimizando (error, costo, precisión).
- Las **restricciones** limitan las posibles soluciones (tiempo, recursos, valores).
- En IA, las funciones objetivo son generalmente funciones de pérdida (loss functions).
- Las restricciones pueden ser técnicas, computacionales, éticas o de diseño.

**4. Gradiente descendente y variantes (SGD, Mini-Batch)**

**¿Qué es el gradiente descendente?**

El **gradiente descendente** es uno de los algoritmos más importantes y utilizados en inteligencia artificial.

Lo usamos, por ejemplo, para **entrenar redes neuronales, ajustar modelos de regresión y minimizar funciones de pérdida**.

Se trata de una técnica **iterativa**, que sirve para **encontrar el punto mínimo** de una función, es decir, el conjunto de valores que **hacen que el error sea lo más pequeño posible**.

**Una analogía sencilla**

Imagina que estás en la cima de una montaña con los ojos vendados y necesitas llegar al punto más bajo (el valle), pero no puedes ver el camino.

Lo que haces es **tocar el terreno, sentir hacia qué lado baja más la pendiente, y dar un pequeño paso en esa dirección**. Repites eso muchas veces hasta que ya no puedas bajar más.

Eso, exactamente, es lo que hace el **gradiente descendente**:

- Calcula hacia dónde baja más la función (la dirección del **gradiente**).
- Da un paso en ese sentido.
- Repite el proceso hasta encontrar el mínimo.

**¿Cómo se expresa matemáticamente?**

El gradiente descendente actualiza los parámetros del modelo con la siguiente fórmula:

$$\theta := \theta - \eta \cdot \nabla f(\theta)$$

Donde:

- $\Theta$ son los **parámetros del modelo** (por ejemplo, los pesos de una red).
- $\nabla f(\theta)$ es el **gradiente**, es decir, la pendiente de la función en ese punto.
- $\eta$ es la **tasa de aprendizaje**: controla qué tan grande es cada paso.

El objetivo es ajustar los parámetros para **minimizar la función de pérdida**, paso a paso.

**Aplicación en IA: entrenamiento de modelos**

Cuando entrenamos un modelo de regresión o clasificación, queremos que sus predicciones sean lo más cercanas posibles a los valores reales. Para eso, definimos una **función de pérdida** (por ejemplo, el error cuadrático medio) y usamos gradiente descendente para **ajustar los pesos del modelo** y minimizar ese error.

**Variantes del gradiente descendente**

Existen tres formas comunes de aplicar este algoritmo, dependiendo de cuántos datos usamos para calcular el gradiente en cada paso:

a. **Gradiente descendente batch (por lotes completos)**

Calcula el gradiente usando **todo el conjunto de entrenamiento**.

**Ventajas:**

- Más preciso en cada paso.
- Convergencia más estable.

**Desventajas:**

- Muy lento en datasets grandes.
- Requiere mucha memoria.

b. **Gradiente descendente estocástico (SGD)**

Calcula el gradiente usando **solo un dato** a la vez (una muestra).
**Ventajas:**

- Mucho más rápido.
- Puede escapar de mínimos locales.

**Desventajas:**

- Puede ser inestable (saltos erráticos).
- No garantiza convergencia suave.

c. **Mini-Batch Gradient Descent**

Calcula el gradiente usando **un subconjunto de datos (batch pequeño)**.
**Ventajas:**

- Equilibrio entre velocidad y estabilidad.
- Muy usado en redes neuronales modernas (con batches de 32, 64, 128, etc.).

### Ejemplo en Python (simplificado)

python Copiar código

```python
import numpy as np

# Función de pérdida: f(x) = x^2
def f(x):
    return x**2

# Derivada (gradiente): f'(x) = 2x
def grad_f(x):
    return 2 * x

# Gradiente descendente
x = 5.0              # Valor inicial
learning_rate = 0.1
iterations = 10

for i in range(iterations):
    grad = grad_f(x)
    x = x - learning_rate * grad
    print(f"Iteración {i+1}: x = {x:.4f}, f(x) = {f(x):.4f}")
```

**¿Qué hace este código?**

Minimiza la función $f(x) = x^2$, cuyo mínimo está en $x = 0$. El gradiente es $f'_{(x)} = 2x$, y con cada paso, el algoritmo se acerca al mínimo.

**Consideraciones prácticas**

- La elección de la **tasa de aprendizaje (learning rate)** es clave:
  - Si es muy alta → el algoritmo puede "rebotar" y no converger.
  - Si es muy baja → avanza muy lentamente y tarda demasiado.
- Se pueden usar técnicas avanzadas como:
  - **Momentum**: agrega velocidad para evitar oscilaciones.
  - **RMSprop y Adam**: adaptan el tamaño del paso según la historia del gradiente.

**En resumen**

- El gradiente descendente es la base para **entrenar modelos ajustando sus parámetros.**
- Es un método numérico que **funciona por aproximaciones sucesivas**.
- Las variantes (batch, SGD, mini-batch) se adaptan a distintos tamaños de datos y problemas.
- Es fundamental entenderlo, porque aparece en casi todos los algoritmos de aprendizaje supervisado y redes neuronales.

5. **Programación lineal: formulación y método simplex**

**¿Qué es la programación lineal?**

La **programación lineal (PL)** es una técnica matemática utilizada para **optimizar una función lineal** (maximizar o minimizar), sujeta a **restricciones también lineales**. Se usa cuando queremos **asignar recursos limitados** de manera óptima.

Aunque tradicionalmente se aplica en problemas logísticos o industriales, en el contexto de la inteligencia artificial también se puede usar en tareas como:

- Optimización de recursos computacionales
- Planificación automática
- Asignación de tareas en sistemas multiagente
- Diseño de experimentos

**¿Qué significa "lineal"?**

Una función es **lineal** si las variables están elevadas solo a la **potencia 1** y **no se multiplican entre sí**.

Ejemplo de función objetivo lineal:

$$Z = 5x + 3y$$

Las restricciones también deben ser lineales, por ejemplo:

$$2x + y \leq 10 \quad \text{y} \quad x, y \geq 0$$

Esto hace que el problema se pueda representar gráficamente como una región poligonal y se pueda resolver de forma exacta.

**Componentes de un problema de PL**

1. **Función objetivo**: lo que queremos optimizar (ej. maximizar la precisión o minimizar el costo).
2. **Variables de decisión**: los valores que podemos ajustar (ej. uso de memoria, tiempo de CPU, asignación de tareas).
3. **Restricciones**: condiciones que deben cumplirse (ej. no pasar de cierto límite de recursos).
4. **Región factible**: conjunto de soluciones posibles que cumplen todas las restricciones.

**Ejemplo práctico (aplicado a IA)**

Supongamos que estás entrenando dos modelos de IA, pero tienes recursos limitados (tiempo de procesamiento). Quieres **maximizar el impacto** de tu sistema en un proyecto, sin pasarte del límite de horas disponibles.

- Modelo A genera 5 puntos de impacto por cada hora entrenada.
- Modelo B genera 3 puntos por hora.
- Tienes un máximo de 20 horas disponibles.
- El modelo A no puede usarse más de 8 horas.

**Formulación:**

$$\max Z = 5x + 3y$$

Sujeto a:

$$x + y \leq 20 \quad (\text{total de horas}) \quad x \leq 8 \quad (\text{límite para modelo A}) \quad x, y \geq 0$$

**Representación gráfica (cuando hay 2 variables)**

Cuando el problema tiene solo dos variables, se puede **graficar fácilmente**:

1. Se dibujan las restricciones en un plano x−yx-yx−y.
2. Se identifica la región factible (intersección de las restricciones).
3. Se grafica la función objetivo y se “mueve” la recta buscando el valor máximo.

El óptimo **siempre se encuentra en un vértice de la región factible**.

**¿Qué es el método simplex?**

Cuando el número de variables es mayor a dos (como en la mayoría de los problemas reales), no es posible resolver el problema gráficamente. Aquí es donde usamos el **método simplex**, que es un algoritmo exacto, paso a paso, para encontrar la **solución óptima**.

**¿Cómo funciona?**

1. **Reescribe el problema en forma estándar** (igualdades con variables de holgura).
2. **Construye una tabla simplex** con los coeficientes de la función objetivo y restricciones.
3. **Itera**: en cada paso, eliges qué variable entra y cuál sale de la base, hasta encontrar el máximo o mínimo.

Aunque los pasos pueden parecer mecánicos, el valor del método está en que **garantiza encontrar la mejor solución posible** dentro de los límites definidos.

**Aplicaciones en inteligencia artificial**

La programación lineal se puede aplicar en IA cuando los problemas requieren:

- **Asignar tareas** entre diferentes agentes de forma eficiente.
- **Planificar horarios o recursos** para tareas concurrentes.
- **Minimizar el tiempo total de entrenamiento** cumpliendo restricciones del sistema.
- **Optimizar combinaciones discretas** (a veces junto con métodos de programación entera).

Por ejemplo, en un sistema de planificación automatizada (como los utilizados en logística o robótica), la PL puede servir para decidir qué acciones ejecutar, en qué orden y con qué prioridad, maximizando el rendimiento y respetando las restricciones de tiempo o energía.

| **Concepto** | **Significado** |
|---|---|
| Función objetivo | Expresión lineal que queremos maximizar o minimizar |
| Restricciones | Límites también lineales que deben cumplirse |
| Región factible | Conjunto de soluciones posibles |
| Método simplex | Algoritmo que explora los vértices del espacio de soluciones para hallar el óptimo |
| Uso en IA | Optimización de tareas, recursos, horarios, cargas de trabajo |

**¿Y si el problema no es lineal?**

Si la función o las restricciones **no son lineales**, ya no se puede usar el método simplex. Ahí se requieren métodos de **optimización no lineal** o heurísticos, que veremos en la siguiente unidad.

## Bibliografía


- Boyd, S., & Vandenberghe, L. (2004). Convex Optimization. Cambridge University Press.
- Nocedal, J., & Wright, S. (2006). Numerical Optimization (2nd ed.). Springer.
- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (2nd ed.). O'Reilly Media.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.