Sesión 3: Cálculo Diferencial en una Variable

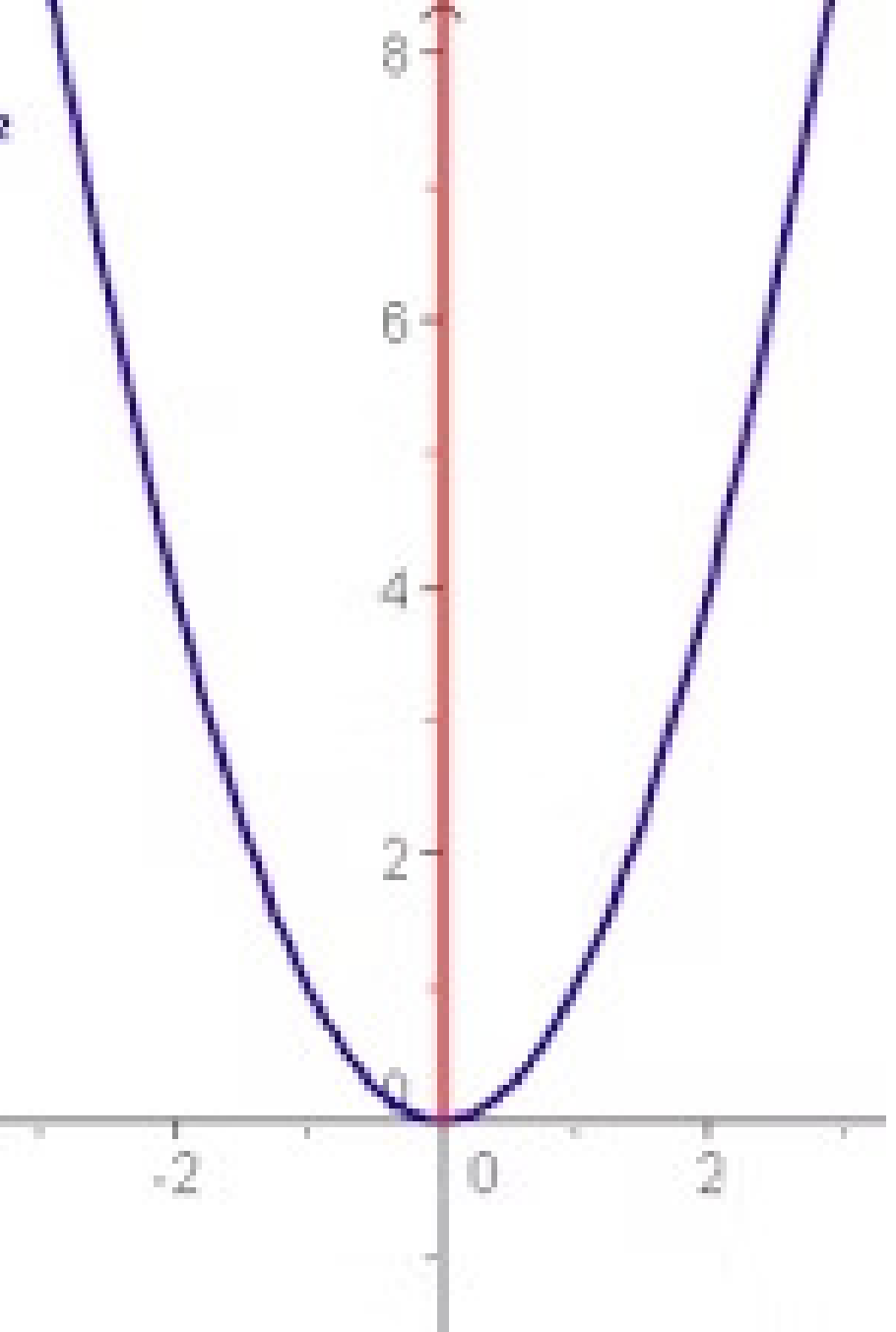
El Aprendizaje de Máquina (Machine Learning) se basa en gran medida en la formulación de problemas de optimización, donde se buscan los valores de los parámetros de un modelo que minimicen (o maximicen) una determinada función de costo. El cálculo diferencial resulta esencial para abordar estos desafíos, pues permite analizar cómo cambian las funciones y determinar sus puntos críticos (máximos o mínimos).

por Kibernum Capacitación S.A.

Introducción al Cálculo Diferencial en una Variable

En esta sesión, nos centraremos en los fundamentos del cálculo diferencial en una variable, estableciendo la conexión entre la teoría y su implementación práctica en entornos de programación. A continuación, se describen los temas clave que se tratarán:

1. Conceptos Básicos de Cálculo Diferencial
2. Derivadas
3. Cálculo de Derivadas
4. Aplicaciones en Optimización
5. Uso de Python para Cálculo Diferencial



Conceptos Básicos de Cálculo Diferencial: Funciones

Se revisará cómo las funciones de una variable real pueden modelar relaciones cuantitativas en el contexto del Aprendizaje de Máquina (por ejemplo, funciones de costo o pérdida).

Una función f(x) asigna a cada valor de x un valor de salida f(x).

Ejemplo: $f(x) = x^2$

Conceptos Básicos de Cálculo Diferencial: Límites y Continuidad

Entenderemos cómo se definen los límites de una función y la importancia de la continuidad para garantizar la existencia de la derivada.

El límite es el valor al que se acerca una función f(x) cuando la variable x se aproxima a cierto valor a. Es fundamental para definir la continuidad de la función y el concepto de derivada. Por ejemplo, para la función f(x)= sin x/x cuando x→0, se tiene:

$$\lim_{x \to 0} \frac{\sin x}{x} = 1.$$

Definición de la Derivada

Se presentará la derivada como la tasa de cambio instantánea de una función, y se explicará por qué es crucial en la optimización de modelos.

La derivada de f(x) en x = a se define como:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$$

Representa la tasa de cambio instantánea de la función en ese punto.

La derivada de una función en un punto mide la tasa de cambio instantánea o la pendiente de la recta tangente en ese punto.
Se define como:

f'(x) = lim(h→0) [f(x+h) - f(x)]/h

Esta definición permite determinar cómo varía la función en torno a un punto específico.

Reglas Básicas de Derivación

Aprenderemos a derivar funciones polinómicas, exponenciales y trigonométricas usando reglas como la suma, el producto y el cociente.

Las reglas de derivación simplifican el proceso de hallar derivadas de funciones complejas. Algunas de las reglas más importantes son:

- Regla de la Suma: [f(x) + g(x)]' = f'(x) + g'(x)

$$(f+g)' = f' + g'$$

- Regla del Producto: [f(x) · g(x)]' = f'(x) · g(x) + f(x) · g'(x)

$$(f \cdot g)' = f'g + fg'$$

- Regla del Cociente: [f(x)/g(x)]' = [f'(x) · g(x) - f(x) · g'(x)]/[g(x)]²

$$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$$

- Regla de la Cadena: Si y = f(g(x)), entonces dy/dx = f'(g(x)) · g'(x)

Si $h(x) = f(g(x))$, entonces:

$$h'(x) = f'(g(x)) \cdot g'(x)$$

Si $f(x) = \sin(2x)$, aplicando la regla de la cadena,

$$f'(x) = \cos(2x) \cdot 2.$$

Cálculo de Derivadas de Funciones Básicas

Para resolver problemas de baja complejidad, es necesario calcular las derivadas de funciones simples. Por ejemplo, consideremos las siguientes funciones:

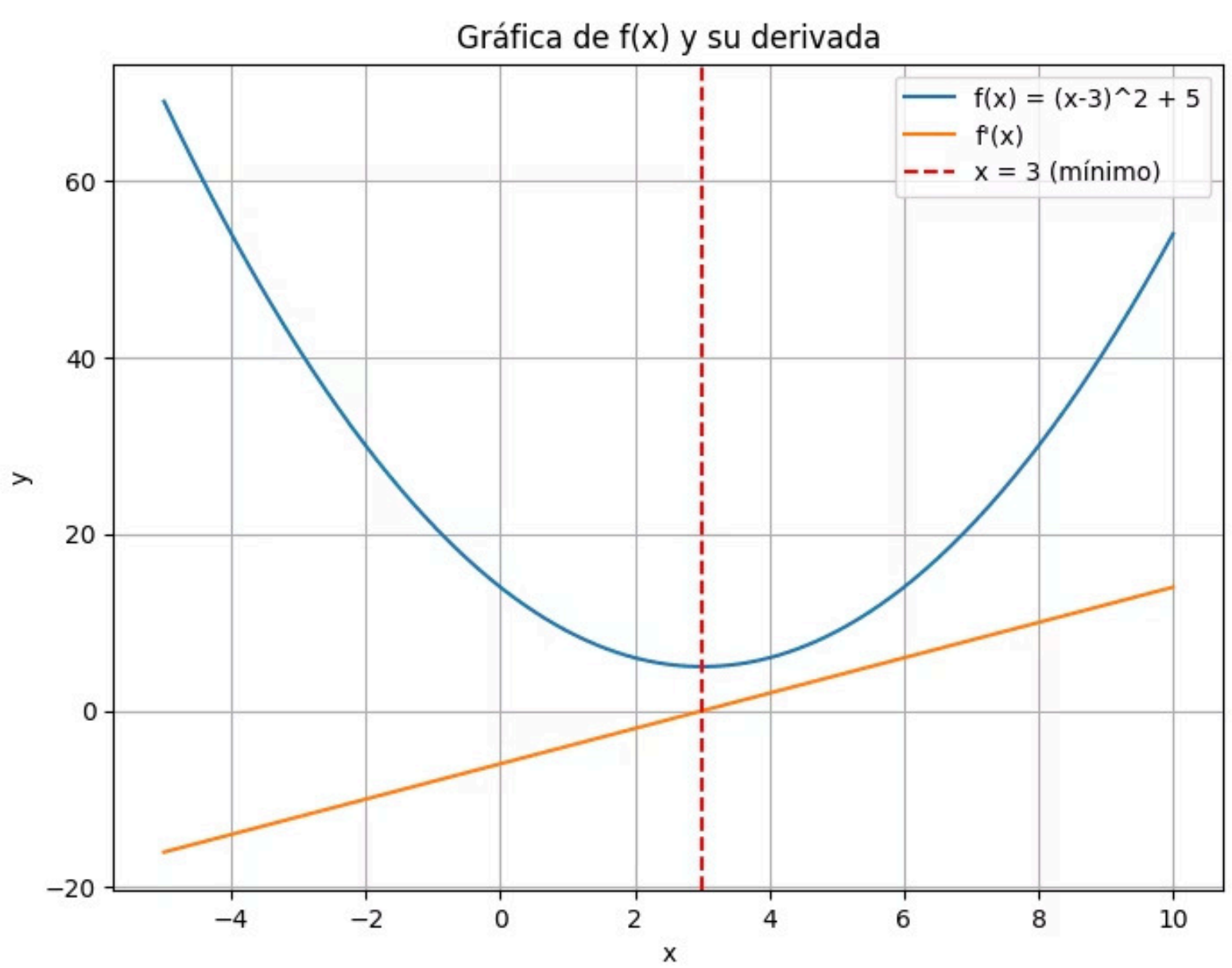
Función Polinómica: $f(x) = 3x^2 + 2x - 5$, $f'(x) = 6x + 2$

Función Trigonométrica: $g(x) = \sin(x) + \cos(x)$, $g'(x) = \cos(x) - \sin(x)$

Estos cálculos se realizan aplicando las reglas básicas de derivación, y son fundamentales para el análisis de comportamiento de las funciones.

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# Definir la variable simbólica y la función f(x)
x = sp.symbols('x')
f_expr = (x - 3)**2 + 5

# Calcular la derivada simbólica de f(x)
f_prime_expr = sp.diff(f_expr, x)

# Convertir las expresiones simbólicas en funciones numéricas con lambdify
f = sp.lambdify(x, f_expr, 'numpy')
f_prime = sp.lambdify(x, f_prime_expr, 'numpy')

# Generar un rango de valores para x
x_vals = np.linspace(-5, 10, 400)

# Evaluar la función y su derivada en los valores generados
y_vals = f(x_vals)
y_prime_vals = f_prime(x_vals)

# Crear la gráfica
plt.figure(figsize=(8, 6))
plt.plot(x_vals, y_vals, label='f(x) = (x-3)^2 + 5')
plt.plot(x_vals, y_prime_vals, label="f'(x)")
plt.axvline(x=3, color='red', linestyle='--', label='x = 3 (mínimo)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gráfica de f(x) y su derivada')
plt.legend()
plt.grid(True)
plt.show()
```

Ejemplo sencillo en Python que utiliza **Sympy** para obtener la derivada simbólica de una función y **Matplotlib** para graficar tanto la función como su derivada:


Gráfica de f(x) y su derivada
f(x) = (x-3)^2 + 5
f'(x)
x = 3 (mínimo)
60
40
20
0
−20
y
−4
−2
0
2
4
6
8
10
x

Aplicación de la Regla de la Cadena en Funciones Compuestas

La regla de la cadena es crucial para derivar funciones compuestas, ya que permite descomponer la derivación de una función compleja en derivadas más simples. En problemas de baja complejidad, esto significa que si tenemos una función:

$f(x) = \sin(3x)$

se identifica:

$u = 3x$, $f(u) = \sin(u)$

La derivada de $f(u)$ es $\cos(u)$ La derivada de u es 3

Por lo tanto, aplicando la regla de la cadena:

$f'(x) = \cos(3x) \cdot 3 = 3\cos(3x)$

Esta técnica es fundamental para resolver problemas de baja complejidad que involucren funciones compuestas.

La Regla de la Cadena en Machine Learning

Esta técnica nos permite tratar cada componente de la función por separado y luego combinar los resultados. Por ejemplo, si necesitas derivar una función que modela un proceso sencillo pero que, a la vez, es el resultado de la composición de dos procesos más simples, la regla de la cadena hace que el problema sea mucho más manejable y sistemático.

En el contexto del machine learning, la regla de la cadena es fundamental en el proceso de backpropagation en redes neuronales. Durante el entrenamiento, es necesario calcular la derivada del error con respecto a cada parámetro del modelo. Dado que la salida de una red neuronal es el resultado de aplicar múltiples funciones compuestas (cada capa de la red aplica una transformación lineal seguida de una función de activación), la regla de la cadena permite propagar de forma eficiente estos gradientes a través de la red. Esto es vital para ajustar los pesos y minimizar la función de costo, optimizando así el rendimiento del modelo.

En resumen, la regla de la cadena no solo simplifica la derivación de funciones compuestas en problemas matemáticos de baja complejidad, sino que también es un pilar en la optimización de modelos de machine learning, facilitando el entrenamiento y la actualización de parámetros en algoritmos de aprendizaje profundo.

Backpropagation en Redes Neuronales

El backpropagation es un algoritmo esencial en el entrenamiento de redes neuronales. Su función principal es calcular de manera eficiente los gradientes de la función de pérdida respecto a cada uno de los parámetros (pesos y sesgos) de la red. Estos gradientes indican cómo deben ajustarse los parámetros para minimizar el error del modelo.

El proceso se lleva a cabo en dos fases:

1. Propagación hacia adelante (forward pass): Se introduce una entrada a la red, y cada capa procesa esa información para generar una salida final. En esta fase se calcula el valor de la función de pérdida, que mide la diferencia entre la salida predicha y la salida deseada.
2. Propagación hacia atrás (backward pass): Utilizando la regla de la cadena, el algoritmo retropropaga el error desde la capa de salida hasta la capa de entrada. Esto implica:
    - Calcular la derivada de la función de pérdida respecto a la salida de la red.
    - Determinar, mediante la regla de la cadena, cómo ese error se distribuye en cada capa.
    - Obtener los gradientes de la pérdida con respecto a cada peso y sesgo.

Con estos gradientes, se pueden actualizar los parámetros de la red utilizando métodos de optimización, como el descenso de gradiente, para reducir progresivamente el error del modelo.

Ejemplo de Backpropagation

Ilustra el proceso de backpropagation utilizando la regla de la cadena en una red neuronal simple de una sola neurona. En este ejemplo, la neurona toma una entrada x, calcula una combinación lineal con un peso w y un sesgo b, y luego aplica la función de activación sigmoide. La salida de la neurona se compara con un valor objetivo y mediante el error cuadrático. Posteriormente, se calculan los gradientes utilizando la regla de la cadena y se actualizan los parámetros.

```python
import numpy as np

# Definir la función sigmoide y su derivada
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_deriv(x):
    s = sigmoid(x)
    return s * (1 - s)

# Parámetros iniciales
x = 0.5         # entrada
y = 1.0         # valor objetivo
w = 0.1         # peso inicial
b = 0.0         # sesgo inicial
learning_rate = 0.1

# PROCESO DE PROPAGACIÓN HACIA ADELANTE (Forward Pass)
z = w * x + b                   # combinación lineal
y_hat = sigmoid(z)              # salida de la neurona después de la activación
loss = 0.5 * (y - y_hat) ** 2     # error cuadrático
print("Forward Pass:")
print("z =", z)
print("y_hat =", y_hat)
print("Loss =", loss)

# PROCESO DE RETROPROPAGACIÓN (Backward Pass) USANDO LA REGLA DE LA CADENA
# Paso 1: Derivada del error respecto a la salida (y_hat)
dL_dyhat = -(y - y_hat)

# Paso 2: Derivada de la función sigmoide respecto a z
dyhat_dz = sigmoid_deriv(z)

# Paso 3: Derivada de z respecto a los parámetros w y b
dz_dw = x         # ya que z = w*x + b
dz_db = 1         # ya que z = w*x + b

# Aplicar la regla de la cadena para calcular los gradientes
dL_dw = dL_dyhat * dyhat_dz * dz_dw
dL_db = dL_dyhat * dyhat_dz * dz_db
print("\nBackward Pass:")
print("dL/dw =", dL_dw)
print("dL/db =", dL_db)

# Actualizar los parámetros utilizando el descenso de gradiente
w_new = w - learning_rate * dL_dw
b_new = b - learning_rate * dL_db
print("\nUpdated Parameters:")
print("Nuevo w =", w_new)
print("Nuevo b =", b_new)
```

```
Forward Pass:
z = 0.05
y_hat = 0.5124973964842103
Loss = 0.11882939421733661

Backward Pass:
dL/dw = -0.060899755162180894
dL/db = -0.12179951032436179

Updated Parameters:
Nuevo w = 0.1060899755162181
Nuevo b = 0.01217995103243618
```

Este ejemplo ilustra de manera sencilla cómo se utiliza la regla de la cadena para descomponer la derivación de una función compuesta y, a partir de ella, calcular los gradientes necesarios para ajustar los parámetros en el entrenamiento de una red neuronal.

Explicación del Ejemplo

1. **Forward Pass:**

- Se calcula z=w×x+b .
- Se aplica la función sigmoide para obtener y_hat=sigmoid(z)
- Se mide el error con la función de pérdida (error cuadrático):

$$\text{loss} = \frac{1}{2}(y - y\_hat)^2.$$

1. **Backward Pass (Retropropagación):**

- Se calcula la derivada del error respecto a la salida,

$$\frac{\partial L}{\partial y\_hat}.$$

- Se utiliza la derivada de la función sigmoide, para saber cómo varía la salida con respecto a la combinación lineal z.

- Se multiplican estas derivadas junto con las derivadas de ***z*** respecto a ***w*** y ***b*** (respectivamente, x y 1), aplicando la regla de la cadena.
- Finalmente, se actualizan los parámetros ***w*** y ***b*** utilizando un paso de descenso de gradiente.

Uso del Cálculo Diferencial para Encontrar Máximos y Mínimos

El cálculo diferencial se utiliza para determinar los puntos de máximo y mínimo de una función, lo cual es esencial en optimización. El proceso implica:

1. Calcular la derivada $f'(x)$ de la función $f(x)$.
2. Encontrar los puntos críticos donde $f'(x)=0$ o la derivada no está definida.
3. Utilizar la prueba de la segunda derivada o el análisis de la variación de $f'(x)$ para determinar si se trata de un máximo o un mínimo.

Ejemplo: Para la función $f(x)=x^2-4x+5$:

- La derivada es $f'(x)=2x-4$.
- Igualando a cero, $2x-4=0$ se obtiene $x=2$.
- Evaluar la segunda derivada $f''(x)=2$ (positivo) indica que en $x=2$ se tiene un mínimo.

Aplicaciones en Optimización

En Aprendizaje de Máquina, las derivadas se utilizan para:

- Encontrar mínimos de funciones de costo: Minimizar la pérdida en modelos de regresión o clasificación.
- Actualizar parámetros de un modelo: Gradient Descent y métodos relacionados (Adam, RMSProp, etc.) se basan en las derivadas parciales del costo con respecto a cada parámetro.

Ejemplo Sencillo de Minimización

Considere $f(x)=x^2+2x+1$.

- Su derivada es $f'(x)=2x+2$.
- El mínimo ocurre cuando $f'(x)=0$, es decir $x=-1$.

Simulación de Problemas de Cálculo Diferencial en Python

El entorno Python es ampliamente utilizado en la industria para simular y resolver problemas matemáticos, incluyendo el cálculo diferencial.

Python ofrece varias librerías que facilitan el cálculo de derivadas y la optimización:

1. Sympy: Permite realizar derivación simbólica.
2. SciPy: Ofrece funciones de optimización (por ejemplo, scipy.optimize).

A continuación, se presentan dos ejemplos de cómo aplicar estos conceptos:

Derivación Simbólica con Sympy

```python
import sympy as sp

# Definir la variable simbólica
x = sp.Symbol('x')

# Definir la función
f = sp.sin(3*x + 1)

# Calcular la derivada usando la regla de la cadena
f_prime = sp.diff(f, x)

print("La derivada de f(x) es:", f_prime)
```

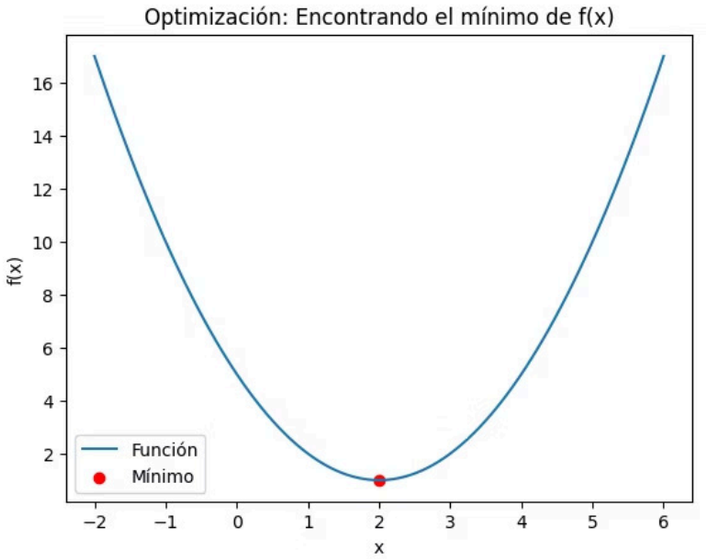
```
La derivada de f(x) es: 3*cos(3*x + 1)
```

Explicación: Este código utiliza Sympy para calcular la derivada de sin(3x + 1), aplicando internamente la regla de la cadena.

Optimización con SciPy

```python
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# Definir la función a optimizar (en este caso, un polinomio)
def funcion_objetivo(x):
    return (x - 2)**2 + 1

# Usar 'minimize' para encontrar el mínimo
resultado = minimize(funcion_objetivo, x0=[0])

print("El mínimo se encuentra en x =", resultado.x[0])

# Visualización
x_vals = np.linspace(-2, 6, 100)
y_vals = funcion_objetivo(x_vals)
plt.plot(x_vals, y_vals, label='Función')
plt.scatter(resultado.x, funcion_objetivo(resultado.x), color='red', label='Mínimo')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Optimización: Encontrando el mínimo de f(x)')
plt.legend()
plt.show()
```

```
El mínimo se encuentra en x = 1.9999999746237656
```



Explicación: En este ejemplo, se define una función cuadrática simple y se utiliza minimize de SciPy para encontrar su mínimo. Además, se grafica la función junto con el punto mínimo.

Actividad Práctica Guiada: Aplicando cálculo diferencial

Objetivo: Aplicar el cálculo diferencial para derivar funciones simbólicamente y resolver un problema de minimización en Python.

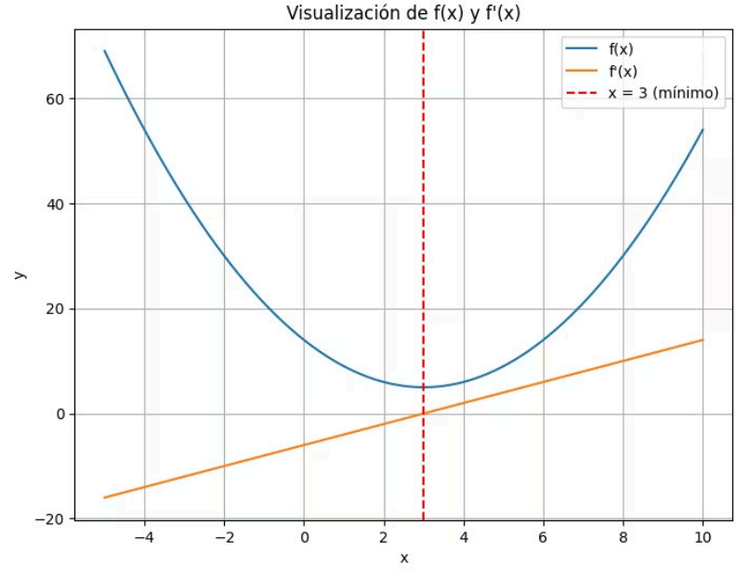
Paso a Paso:

1. Definir Función y Calcular su Derivada con Sympy
    - Crear una función $f(x)=(x-3)^2+5$
    - Usar sp.diff para obtener su derivada y confirmar el punto crítico.
2. Visualizar la Función y su Derivada
    - Generar valores en un rango, por ejemplo $[-5,10]$.
    - Graficar $f(x)$ y $f'(x)$ con matplotlib.
3. Optimizar la Función con SciPy
    - Utilizar minimize para confirmar que el mínimo se encuentra en $x=3$.

Implementación de la Actividad Práctica

```python
# Importación de librerías
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# 1. Definir la función y calcular su derivada con Sympy
# Crear la variable simbólica
x = sp.Symbol('x')
# Definir la función f(x) = (x - 3)^2 + 5
f = (x - 3)**2 + 5
# Calcular la derivada de f(x) usando sp.diff
f_prime = sp.diff(f, x)
# Mostrar la función y su derivada en consola
print("f(x) =", f)
print("f'(x) =", f_prime)
# Confirmar el punto crítico resolviendo f'(x) = 0
critical_points = sp.solve(f_prime, x)
print("Punto(s) crítico(s):", critical_points)

# 2. Visualizar la función y su derivada con Matplotlib
# Convertir f y f_prime a funciones numéricas (usando lambdify)
f_numeric = sp.lambdify(x, f, 'numpy')
f_prime_numeric = sp.lambdify(x, f_prime, 'numpy')
# Generar valores en el rango [-5, 10]
x_vals = np.linspace(-5, 10, 400)
y_vals = f_numeric(x_vals)
y_prime_vals = f_prime_numeric(x_vals)
# Graficar f(x) y f'(x)
plt.figure(figsize=(8, 6))
plt.plot(x_vals, y_vals, label="f(x)")
plt.plot(x_vals, y_prime_vals, label="f'(x)")
plt.axvline(x=3, color='red', linestyle='--', label="x = 3 (mínimo)")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Visualización de f(x) y f'(x)")
plt.legend()
plt.grid(True)
plt.show()

# 3. Optimizar la función con SciPy
# Definir la función objetivo para el optimizador
def objective(x_val):
    return f_numeric(x_val)

# Usar minimize para encontrar el mínimo de la función (valor inicial x0 = 0)
resultado = minimize(objective, x0=[0])
print("Resultado de la optimización:")
print(resultado)
```

```
f(x) = (x - 3)**2 + 5
f'(x) = 2*x - 6
Punto(s) crítico(s): [3]
```




```
Resultado de la optimización:
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 5.000000000000001
        x: [ 3.000e+00]
      nit: 2
      jac: [ 5.960e-08]
 hess_inv: [[ 5.000e-01]]
     nfev: 6
     njev: 3
```

Explicación del Código

Definición y Derivación Simbólica:

- Se define la variable simbólica ***x*** y la función **f(x)=(x−3)2+5**
- Se calcula la derivada ***f_prime*** utilizando ***sp.diff***.
- Se resuelve ***f_prime = 0*** para confirmar que el punto crítico se encuentra en ***x***=3.

Visualización con Matplotlib:

- Se convierte la función y su derivada en funciones numéricas con ***sp.lambdify.***
- Se genera un conjunto de valores de x en el rango [−5,10] y se calculan los correspondientes f(x) y f′(x)
- Se grafican ambas funciones y se marca verticalmente el punto x=3, indicando el mínimo.

Optimización con SciPy:

- Se define la función objetivo ***objective*** que devuelve f(x) para un valor dado.
- Se utiliza ***minimize*** con un valor inicial x0=0 para encontrar el mínimo, que se espera esté en x=3.

Resumen y Conclusiones

En esta sesión hemos explorado los fundamentos del cálculo diferencial en una variable y su aplicación en el contexto del Aprendizaje de Máquina. Hemos cubierto:

- Los conceptos básicos de funciones, límites y continuidad
- La definición formal de la derivada y su interpretación geométrica
- Las reglas básicas de derivación, incluyendo la regla de la cadena
- La aplicación de derivadas para encontrar máximos y mínimos
- El uso de Python para implementar cálculo diferencial y optimización
- La conexión entre el cálculo diferencial y el algoritmo de backpropagation en redes neuronales

Estos conceptos son fundamentales para comprender los algoritmos de optimización utilizados en Machine Learning, como el descenso de gradiente, que permiten ajustar los parámetros de los modelos para minimizar las funciones de costo.

La implementación práctica en Python, utilizando bibliotecas como SymPy, Matplotlib y SciPy, nos ha permitido visualizar y resolver problemas de cálculo diferencial de manera eficiente, estableciendo así una base sólida para abordar problemas más complejos en el ámbito del Aprendizaje de Máquina.