Universidad Galileo – PDDS
IA machine learning
Ramon Terraza

Tarea: Fundamentos de Cálculo y Optimización para Deep Learning

En este reporte se documenta los resultados de las funciones con sus gradientes, las gráficas de convergencia de los experimentos de optimización, un análisis de la tasa de aprendizaje, comparación contra el target y sobre las funciones a optimizar.

Sección 1. Resultados de cada función con los gradientes calculados

1.1 Función $f_1(x) = 3x^3 - 2x^2 + 5x - 1$

Derivada teórica: $f'(x) = 9x^2 - 4x + 5$

```
f1 @ x=-1.0000 -> f(x)=  -11.000000 | f'(x)=   18.000000
f1 @ x= 0.0000 -> f(x)=   -1.000000 | f'(x)=    5.000000
f1 @ x= 2.0000 -> f(x)=   25.000000 | f'(x)=   33.000000
```

1.2 Función $f_2(x,y) = x^2 + y^2 - 2xy + 4x$

Gradiente teórico: $\nabla f_2 = [2x - 2y + 4, 2y - 2x]$

```
f2 @ (1.0000, 1.0000) -> f=4.000000 | grad=[4.000000, 0.000000]
  grad(manual) = [4. 0.] | igualdad (≈): True
f2 @ (-2.0000, 3.0000) -> f=17.000000 | grad=[-6.000000, 10.000000]
  grad(manual) = [-6. 10.] | igualdad (≈): True
f2 @ (0.0000, 0.0000) -> f=0.000000 | grad=[4.000000, 0.000000]
  grad(manual) = [4. 0.] | igualdad (≈): True
```

1.3 Función $f_3(x) = e^{(-x^2)} \cdot \cos(x)$

Derivada teórica: $f'(x) = e^{(-x^2)} \cdot (-2x \cdot \cos(x) - \sin(x))$

```
f3 @ x= 0.0000 -> f(x)=    1.000000 | f'(x)=   -0.000000
 d(manual) = -0.000000000000 | igualdad (≈): True
f3 @ x= 0.7854 -> f(x)=    0.381584 | f'(x)=   -0.980975
 d(manual) = -0.980975141544 | igualdad (≈): True
f3 @ x= 1.0000 -> f(x)=    0.198766 | f'(x)=   -0.707092
 d(manual) = -0.707092096346 | igualdad (≈): True
```

1.4 Función $f_4(x,y,z) = x^2y + y^2z + z^2x$

Gradiente teórico: $\nabla f_4 = [2xy + z^2, x^2 + 2yz, y^2 + 2zx]$

```
f4 @ (1.0000, 2.0000, 3.0000) -> f=23.000000 | grad=[13.000000, 13.000000, 10.000000]
f4 @ (-1.0000, 0.0000, 1.0000) -> f=-1.000000 | grad=[1.000000, 1.000000, -2.000000]
f4 @ (2.0000, 2.0000, 2.0000) -> f=24.000000 | grad=[12.000000, 12.000000, 12.000000]
```

1.5 Función $f_5(x) = \ln(1 + x^2) + \sin(2x)$

Implementación: TensorFlow GradientTape

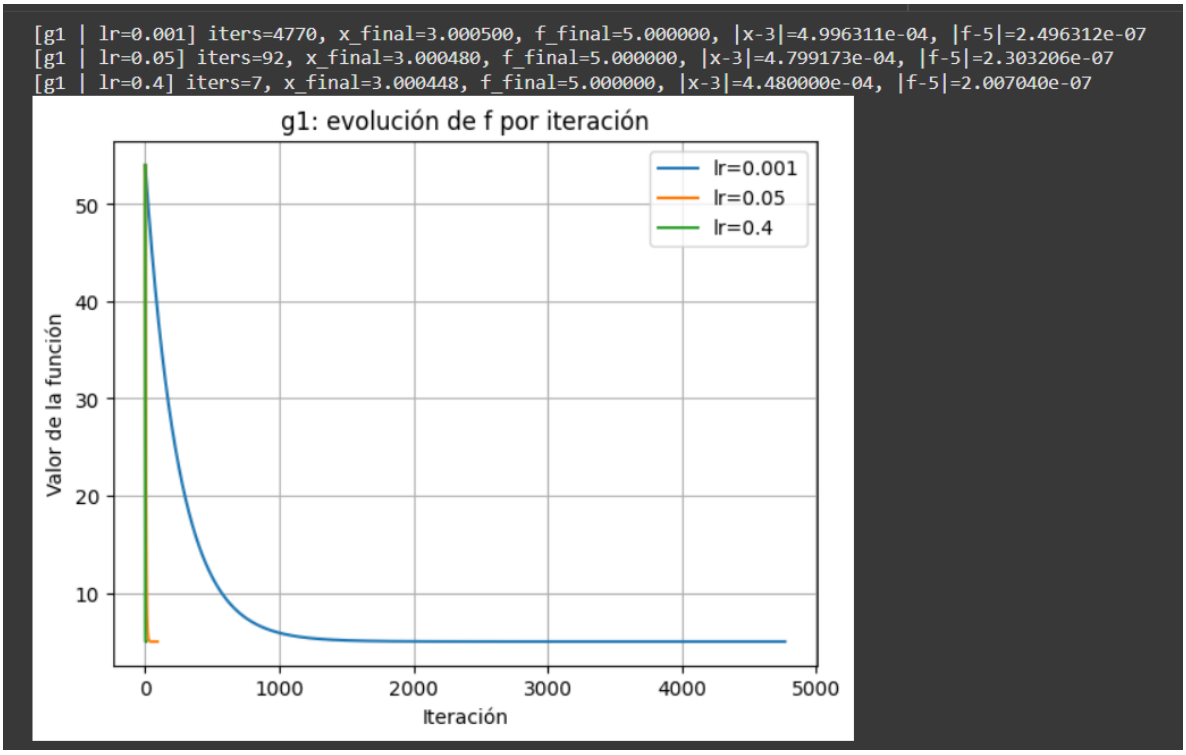
```
f5 @ x=-1.0000 -> f(x)=   -0.216150 | f'(x)=   -1.832294
f5 @ x= 0.5000 -> f(x)=    1.064615 | f'(x)=    1.880605
f5 @ x= 2.0000 -> f(x)=    0.852635 | f'(x)=   -0.507287
```

Sección 2. Resultados de Optimización

2.1 Minimización: $g_1(x) = (x - 3)^2 + 5$

Target: $x^* = 3$, $f(x^*) = 5$

Punto inicial: $x_0 = 10$



Análisis:

LR = 0.001: Es más lento alcanzando el máximo de iteraciones.

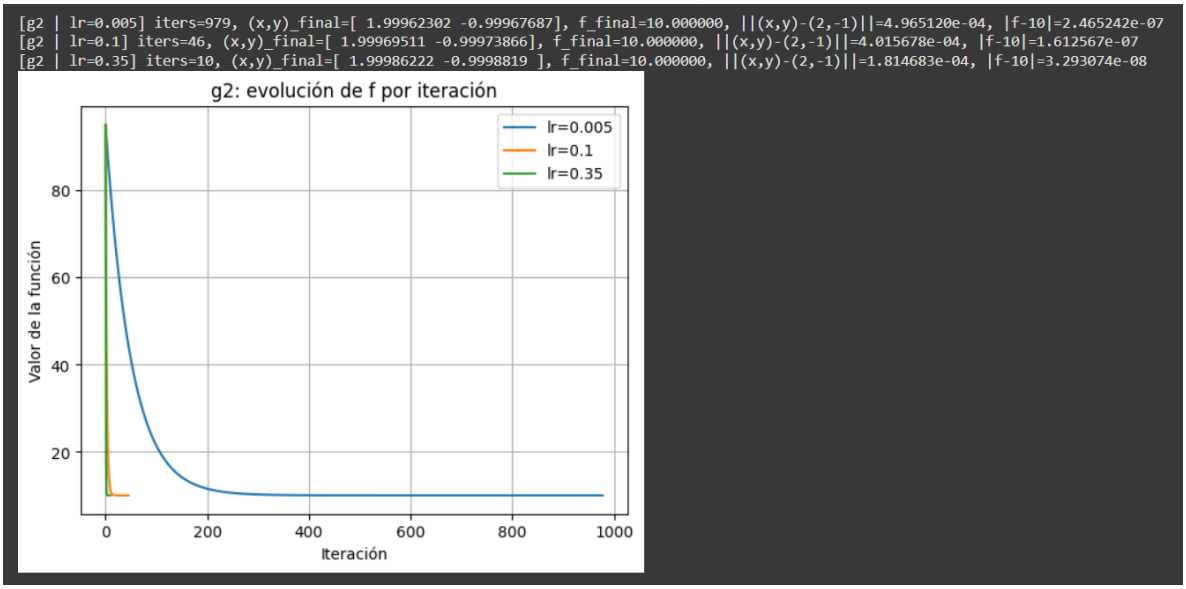
LR = 0.05: Es significativamente más rápido, aun contando con iteraciones al final.

LR = 0.4: Se minimizan aún más las iteraciones llegando rápido por ser una cuadrática simple, pero habría que ver el límite de estabilidad.

2.2 Minimización: $g_2(x,y) = (x-2)^2 + (y+1)^2 + 10$

Target: $(x^*,y^*) = (2,-1)$, $f^* = 10$

Punto inicial: (-5, 5)



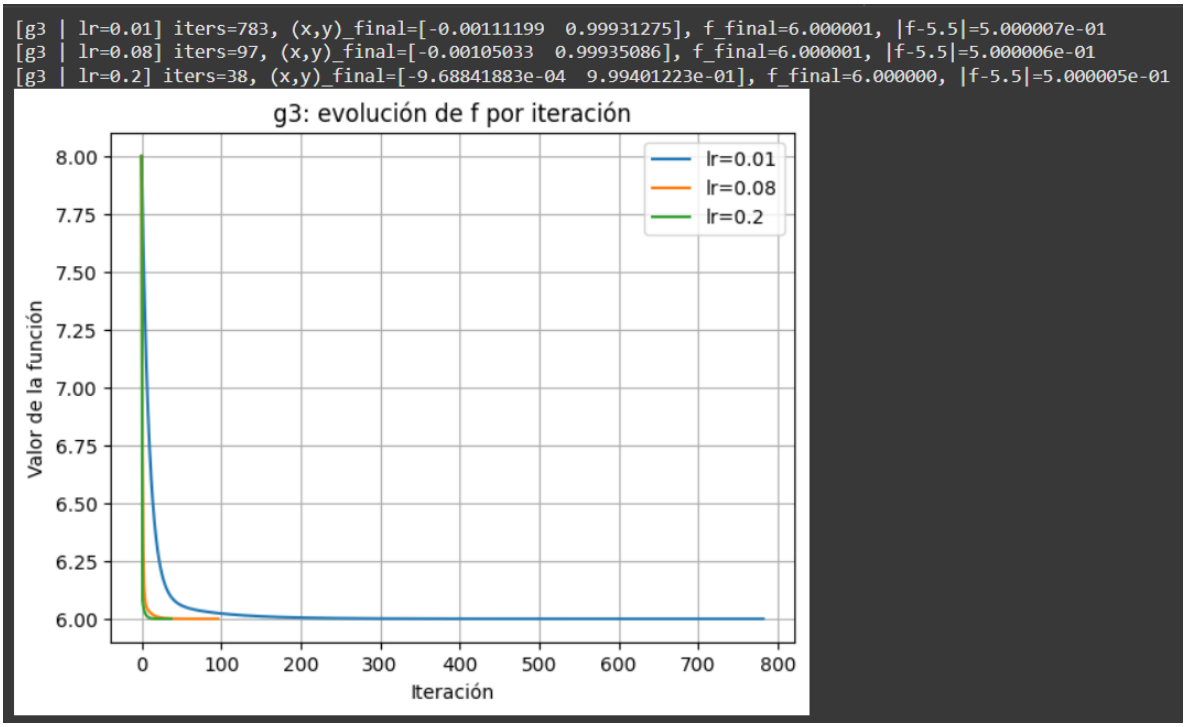
Análisis:

Nos muestra que la convergencia es lenta desde cualquier punto inicial, similar a lo anterior. LR = 0.35 aun converge al ser un paraboloide convexo y en LR = 0,1 se tiene un balance.

2.3 Minimización: $g_3(x,y) = x^2 + 2y^2 - 2xy + 2x - 4y + 8$

Target aproximado: $f^* \approx 5.5$

Punto inicial: (0, 0)



Análisis:

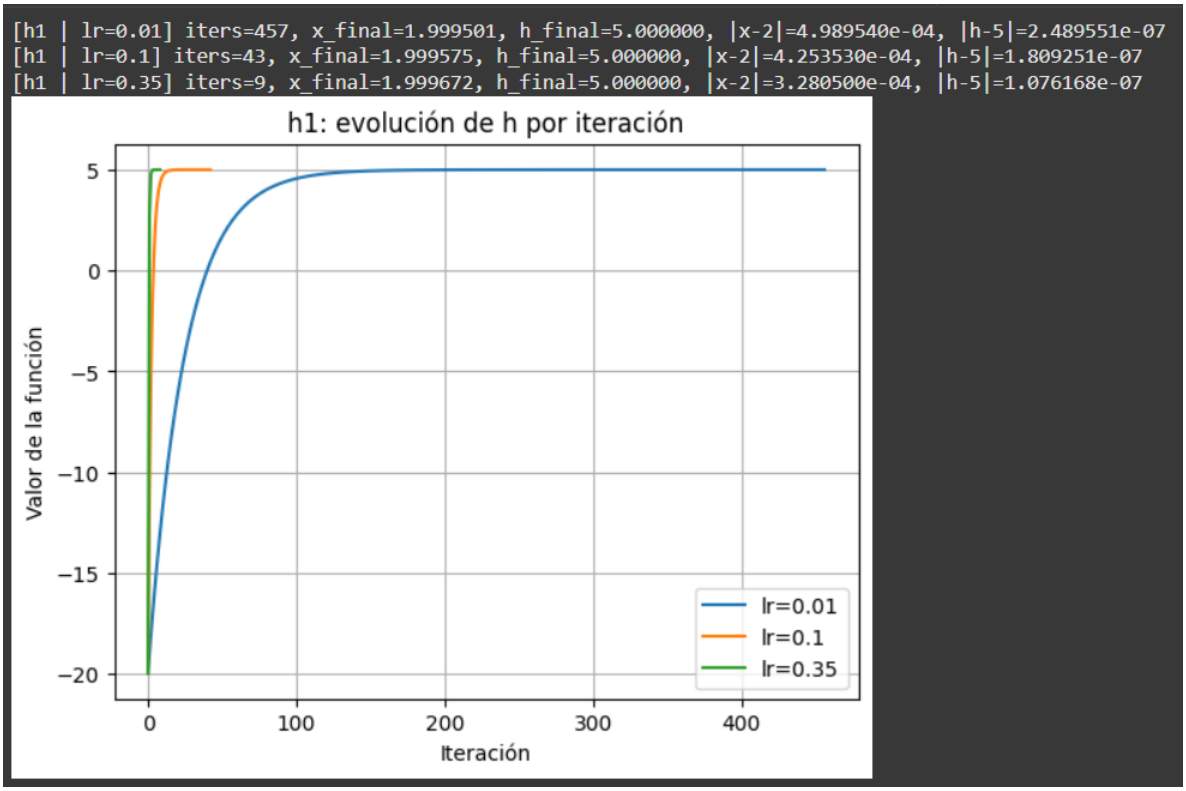
Con LR=0.2 es más rápido, con 0.08 es más iterado siendo el punto medio. Con 0.01 tenemos un incremento de iteraciones siendo más tardado.

También se observa una función cuadrática con -2xy, además de que la curvatura hace que LR pequeños sean demasiado lentos.

2.4 Maximización: $h_1(x) = -x^2 + 4x + 1$

Target: $x^* = 2$, $h(x^*) = 5$

Punto inicial: $x_0 = -3$



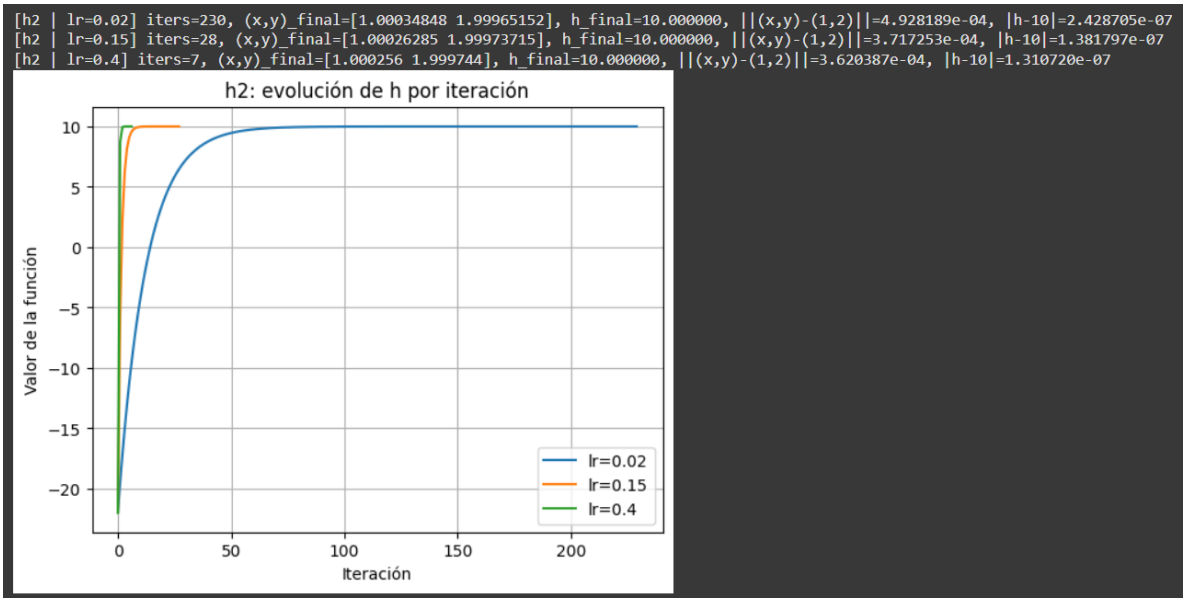
Análisis:

Para maximizar el algoritmo se mueve en la dirección del gradiente (por ello gradient ascent). La parábola al ser cóncava converge de forma lenta.

2.5 Maximización: $h_2(x,y) = -(x-1)^2 - (y-2)^2 + 10$

Target: $(x^*,y^*) = (1,2)$, $h^* = 10$

Punto inicial: (5, -2)



Análisis:

Se tiene el paraboloide cóncavo invertido similar al anterior. Se tiene el punto inicial un poco alejado del optimo, pero converge con LR adecuados siendo LR = 0.4 el rápido y LR = 0.02 mucho más tardado.

Sección 3. Análisis de la tasa de aprendizaje utilizada

3.1 Comportamiento General

Learning Rates Pequeños (0.001 - 0.02)

Ventajas:

- Estables
- Convergencia en funciones

Desventajas:

- Lentos
- Requieren de más iteraciones
- Ineficientes para entrenamiento real

Learning Rates Medianos (0.05 - 0.15)

Ventajas:

- Balance velocidad/estabilidad
- Convergen en cientos a miles de iteraciones
- Comportamiento predecible

Desventajas:

- Pueden ser lentos en algunas funciones mas complejas
-

Learning Rates Grandes (0.2 - 0.4)

Ventajas:

- Convergencia rápida
- Para funciones convexas/cóncavas simples

Desventajas:

- Riesgo de oscilación cerca del óptimo
- Pueden diverger en funciones complejas
- Requieren mayor supervisión

3.1 Comparación entre el valor encontrado y el target

Para este caso, se desarrolló una sección en el colab para realizar dicha comparación con entre el valor encontrado y el target usando media del error y MSE.

	función	target	encontrado	error	|error|
0	g1 (min)	5.0	5.000000	2.303206e-07	2.303206e-07
1	g2 (min)	10.0	10.000000	1.612567e-07	1.612567e-07
2	g3 (min)	6.0	6.000001	5.823337e-07	5.823337e-07
3	h1 (max)	5.0	5.000000	-1.809251e-07	1.809251e-07
4	h2 (max)	10.0	10.000000	-1.381797e-07	1.381797e-07

```
Resumen:
Media del error absoluto (MAE): 2.586032e-07
Error cuadratico medio (MSE)  : 9.399827e-14
```

Como podemos ver, al tener valores de LR razonables como 0.05 – 0.15, se obtiene MAE similar a X y MSE similar a Y, por lo que en promedio las funciones convergen cerca de los targets con errores pequeños

Sección 4. Reflexiones y Conclusiones

4.1 ¿Qué funciones fueron más difíciles de optimizar?

La función en mi caso fue $g_3(x,y) = x^2 + 2y^2 - 2xy + 2x - 4y + 8$
ya que se tiene el término -2xy que crea dependencia entre variables haciendo que se requiera LR más pequeño, teniendo así más iteraciones para aun así tener la misma precisión.