Doble Grado en Ingeniería Informática y Matemáticas

APRENDIZAJE AUTOMÁTICO
(E. Computación y Sistemas Inteligentes)

TRABAJO-1: Programación

UNIVERSIDAD DE GRANADA

Carlos Santiago Sánchez Muñoz

Grupo de prácticas 3 - Lunes

Email: carlossamu7@correo.ugr.es

7 de marzo de 2020

Índice

1. Ejercicio sobre la búsqueda iterativa de óptimos

Gradiente Descendente.

Apartado 1

Implementar el algoritmo de gradiente descendente.

Comenzamos con la implementación de la función gradiente descendente, `gd`, y para ello es fundamental entender a la perfección los diferentes parámetros de dicha función:

- `w`: Punto inicial.
- `lr`: *Learning rate* o tasa de aprendizaje.
- `grad_fun`: Gradiente de la función `fun`.
- `fun`: Función (diferenciable) a minimizar.
- `epsilon`: Error permitido. Si el valor de la función está por debajo de éste entonces acabamos y no hace falta agotar `max_iters`.
- `max_iters`: Número máximo de iteraciones.

Los argumentos `epsilon` y `max_iters` son opcionales, es decir, si no le pasamos un valor tomarán el valor por defecto $-\infty$ y 100000 respectivamente. Mostramos el código del algoritmo:

```
def gd(w, lr, grad_fun, fun, epsilon=-math.inf, max_iters=100000):
    it = 0
    while fun(w)>epsilon and it<max_iters:
        w = w - lr*grad_fun(w)
        it += 1
    return w, it
```

La expresión que rige la minimización de gradiente descendente es $w = w - \eta \nabla E(w)$. Por tanto los dos ingredientes más importantes para que la minimización sea efectiva es una buena elección de la tasa de aprendizaje (η), ya que un valor muy grande podría hacer que el algoritmo no convergiese y un valor muy pequeño podría necesitar demasiadas iteraciones para alcanzar el mínimo, y un buen punto inicial. Por supuesto el cálculo del gradiente ha de ser correcto, en otro caso, es como si no hiciéramos nada.

La condición de parada que hemos impuesto es que no se sobrepase el número máximo de iteraciones o que el valor de la función en un punto sea menor o igual que `epsilon`. La función nos devuelve el vector final donde se alcanza el mínimo y el número de iteraciones que se han usado.

Apartado 2

Considerar la función $E(u,v) = (ue^v - 2ve^{-u})^2$. Usar gradiente descendente para encontrar un mínimo de esta función, comenzando desde el punto $(u,v) = (1,1)$ y usando una tasa de aprendizaje $\eta = 0,1$.

a) Calcular analíticamente y mostrar la expresión del gradiente de la función $E(u,v)$.

b) ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de $E(u,v)$ inferior a 10^{-14}. (Usar flotantes de 64 bits).

c) ¿En qué coordenadas (u,v) se alcanzó por primera vez un valor igual o menor a 10^{-14} en el apartado anterior?

Calculamos las derivadas parciales de E respecto a u y v para formar la expresión del gradiente ∇E:

$$\nabla E(u,v) = \left(\frac{\partial E}{\partial u}, \frac{\partial E}{\partial v}\right) = (2(ue^v - 2ve^{-u})(e^v + 2ve^{-u}), 2(ue^v - 2ve^{-u})(ue^v - 2e^{-u}))$$

Mostramos el código de la función E, sus derivadas parciales y ∇E:

```
""" Funcion E del apartado 2 """
def E(w):
    return (w[0]*np.exp(w[1]) - 2*w[1]*np.exp(-w[0]))**2

""" Derivada parcial de E respecto de u"""
def Eu(w):
    return 2 * (w[0]*np.exp(w[1]) - 2*w[1]*np.exp(-w[0]))
           * (np.exp(w[1]) + 2*w[1]*np.exp(-w[0]))

""" Derivada parcial de E respecto de v """
def Ev(w):
    return  2 * (w[0]*np.exp(w[1]) - 2*w[1]*np.exp(-w[0]))
            * (w[0]*np.exp(w[1])-2*np.exp(-w[0]))

""" Gradiente de E """
def gradE(w):
    return np.array([Eu(w), Ev(w)])
```

Para saber el número de iteraciones que tarda el algoritmo y el punto donde se alcanza el mínimo, llamamos a la función `gd` implementada, usando $(u,v) = (1,1)$ y $\eta = 0{,}1$.

```
w, num_ite = gd(np.array([1.0, 1.0]), 0.1, gradE, E, 1e-14)
print("\nb) Numero de iteraciones: {}".format(num_ite))
print("\nc) Coordenadas obtenidas: ({}, {}).".format(w[0], w[1]))
print("    Valor en el punto: {}.".format(E(w)))
```

El número de iteraciones necesarias para alcanzar un error menor que 10^{-14} es 10. Las coordenadas del mínimo son $(u_{min}, v_{min}) = (0{,}04473629039778207, 0{,}023958714099141746)$ y su imagen $E(u_{min}, v_{min}) = 1{,}2068339442207 47 \cdot 10^{-15}$.

Apartado 3

Considerar ahora la función $f(x,y) = (x-2)^2 + 2(y+2)^2 + 2\sin(2\pi x)\sin(2\pi y)$.

a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial $(x_0 = 1, y_0 = -1)$**, (tasa de aprendizaje** $\eta = 0,01$ **y un máximo de** 50 **iteraciones. Generar un gráfico de cómo desciende el valor de la función con las iteraciones. Repetir el experimento pero usando** $\eta = 0,1$**, comentar las diferencias y su dependencia de** η**.**

b) Obtener el valor mínimo y los valores de las variables (x,y) **en donde se alcanzan cuando el punto de inicio se fija en:** $(2{,}1, -2{,}1)$**,** $(3,-3)$**,** $(1{,}5, 1{,}5)$**,** $(1,-1)$**. Generar una tabla con los valores obtenidos.**

De manera análoga al ejercicio anterior calculamos las derivadas parciales de f respecto a x e y para formar la expresión del gradiente ∇f:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (2(x-2) + 4\pi\cos(2\pi x)\sin(2\pi y), 4(y+2) + 4\pi\sin(2\pi x)\cos(2\pi y))$$

Continuamos mostrando el código asociado a la función f, sus derivadas parciales y ∇f:

```
""" Funcion f del apartado 3 """
def f(w):
    return (w[0]-2)**2 + 2*(w[1]+2)**2 +
           2*np.sin(2*np.pi*w[0])*np.sin(2*np.pi*w[1])

""" Derivada parcial de f respecto de x """
def fx(w):
    return 2*(w[0]-2) + 4*np.pi*np.cos(2*np.pi*w[0])*np.sin(2*np.pi*w[1])

""" Derivada parcial de f respecto de y """
def fy(w):
    return 4*(w[1]+2) + 4*np.pi*np.sin(2*np.pi*w[0])*np.cos(2*np.pi*w[1])

""" Gradiente de f """
def gradf(w):
    return np.array([fx(w), fy(w)])
```

Para el apartado a) se ha implementado una función `gd_grafica`:

```
""" Funcion de GD que almacena los resultados para construir una grafica """
def gd_grafica(w, lr, grad_fun, fun, max_iters = 100000):
    it = 0
    graf = np.zeros(max_iters)
    while it < max_iters:
        graf[it] = fun(w)    # Guardamos el resultado de la iteración
        w = w - lr*gradf(w)
        it += 1

    # Dibujamos la gráfica
    plt.plot(range(0,max_iters), graf, 'b-o', label=r"$\eta$ = {}".format(lr))
    plt.xlabel('Iteraciones'); plt.ylabel('f(x,y)')
    plt.legend()
    plt.title("Curva de gradiente descendente")
    plt.gcf().canvas.set_window_title('Ejercicio 1 - Apartado 3a)')
    plt.show()
    return graf
```

```
print ("a) Grafica con learning rate igual a 0.01")
    g1 = gd_grafica(np.array([1.0, -1.0]), 0.01, gradf, f, 50)
    print ("   Grafica con learning rate igual a 0.1")
    g2 = gd_grafica(np.array([1.0, -1.0]), 0.1, gradf, f, 50)
    # Comparamos las gráficas
    plt.plot(g1, 'b-o', label=r"$\eta$ = {}".format(0.01))
    plt.plot(g2, 'r-o', label=r"$\eta$ = {}".format(0.1))
    plt.xlabel('Iteraciones'); plt.ylabel('f(x,y)');
    plt.legend()
    plt.title("óComparacin de las curvas de gradiente descendente")
    plt.gcf().canvas.set_window_title('Ejercicio 1 - Apartado 3a)')
    plt.show()
```

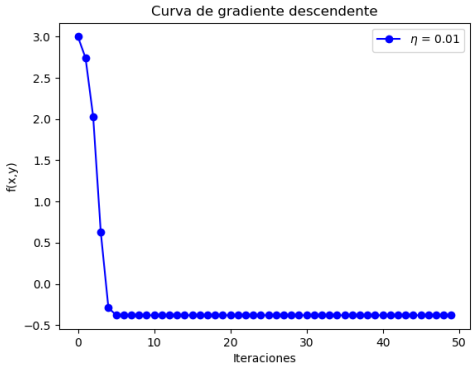

Imagen 1: Curva de GD con $\eta = 0{,}01$

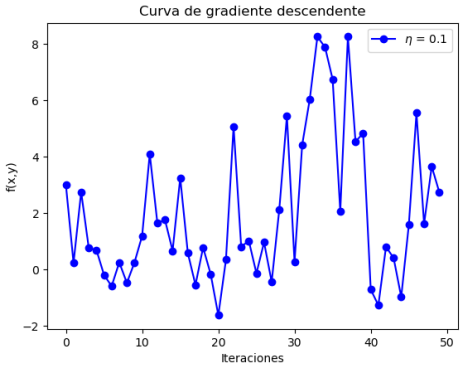

Imagen 2: Curva de GD con $\eta = 0{,}1$

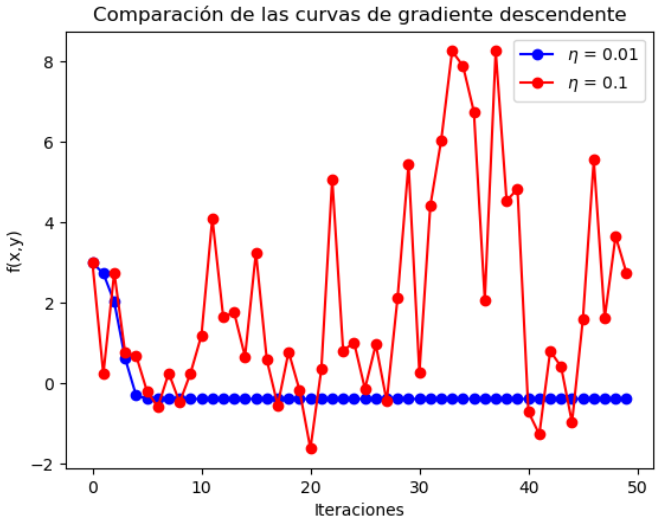

Imagen 3: Comparativa de curvas de GD

```
""" Usando GD muestra el punto inicial, el minimo y el valor del minimo """
def print_gd(w, lr, grad_fun, fun, epsilon, max_iters = 1000000000):
    print("\n   Punto de inicio: ({}, {})".format(w[0], w[1]))
    w, _ = gd(w, lr, grad_fun, fun, epsilon, max_iters)
```

```
    print("    (x,y) = ({}  , {})".format(w[0], w[1]))
    print("    Valor minimo: {}".format(f(w)))


print("\nb) Minimo y valor donde se alcanza segun el punto inicial:")
    lrate = 0.01; eps = 0.0001; max_it = 1000
    print_gd(np.array([2.1, -2.1]), lrate, gradf, f, eps, max_it)
    print_gd(np.array([3.0, -3.0]), lrate, gradf, f, eps, max_it)
    print_gd(np.array([1.5, 1.5]), lrate, gradf, f, eps, max_it)
    print_gd(np.array([1.0, -1.0]), lrate, gradf, f, eps, max_it)
```

(x_0, y_0)	(x_{min}, y_{min})	$E(x_{min}, y_{min})$
(2.1, -2.1)	(2.1, -2.1)	-0.660983
(3, .-3)	(2.766339, -2.745475)	-0.289924
(1.5, 1.5)	(1.779120, 1.0309456)	18.042072
(-1, -1)	(1.233661, -1.254525)	-0.2899238

Apartado 4

¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?

El método de gradiente descendente es un método en principio aplicable a cualquier función diferenciable y para el que podemos asegurar de forma teórica que, bajo ciertas condiciones sobre la función como la convexidad, este método convergerá a un mínimo global de la función. En la práctica estas condiciones no tienen por qué cumplirse en nuestros problemas (por ejemplo, las funciones f y E no son convexas en su dominio), por lo que no siempre obtenemos buenos resultados.

En este ejercicio vemos dos ejemplos de comportamiento muy diferentes ya que 1. en el caso de la función E el algoritmo de gradiente descendente tiene una rápida convergencia hacia el mínimo global de la función, de forma robusta con independencia de cómo ajustemos los parámetros mientras que 2. en el caso de la función f esta convergencia depende enormemente de la tasa de aprendizaje y del punto inicial.

La representación de ambas funciones nos ayuda a entender el por qué de este comportamiento: la función f tiene muchos mínimos locales a los que converge el método de gradiente descendiente (ya que el gradiente sólo nos da información local sobre el comportamiento de la función) y esto provoca su mal comportamiento, mientras que la función E tiene menos mínimos locales, lo que lo convierte en un problema mucho más tratable.

De esto deducimos que la verdadera dificultad reside en elegir los parámetros adecuados, cuyos valores óptimos dependerán del aspecto de la función y su cantidad de mínimos locales en los que el gradiente pueda quedarse «atrapado».

En la práctica es posible que no podamos obtener fácilmente información sobre el aspecto de las funciones que vamos a minimizar, que probablemente tengan alta dimensionalidad y no vengan dadas por una expresión analítica sencilla, por lo que tendremos que experimentar con distintas tasas de aprendizaje y puntos iniciales para ajustar el método a la función a minimizar.

2. Ejercicio sobre Regresión Lineal

Este ejercicio ajusta modelos de regresión a vectores de características extraídos de imágenes de dígitos manuscritos. En particular se extraen dos características concretas: el valor medio del nivel de gris y simetría del número respecto de su eje vertical. Solo se seleccionarán para este ejercicio las imágenes de los números 1 y 5.

Apartado 1

Estimar un modelo de regresión lineal a partir de los datos proporcionados de dichos números (Intensidad promedio, Simetria) usando tanto el algoritmo de la pseudo-inversa como Gradiente descendente estocástico (SGD). Las etiquetas serán $-1, 1$, una para cada vector de cada uno de los números. Pintar las soluciones obtenidas junto con los datos usados en el ajuste. Valorar la bondad del resultado usando E_{in} y E_{out} (para E_{out} calcular las predicciones usando los datos del fichero de test). (usar `Regress_Lin(datos,label)` como llamada para la función (opcional)).

Apartado 2

En este apartado exploramos como se transforman los errores E_{in} y E_{out} cuando aumentamos la complejidad del modelo lineal usado. Ahora hacemos uso de la función `simula_unif(N,2,size)` que nos devuelve N coordenadas 2D de puntos uniformemente muestreados dentro del cuadrado definido por $[-size, size] \times [-size, size]$.

- **EXPERIMENTO**

  a) **Generar una muestra de entrenamiento de $N = 1000$ puntos en el cuadrado $\mathcal{X} = [-1, 1] \times [-1, 1]$. Pintar el mapa de puntos 2D. (ver función de ayuda)**

  b) **Consideremos la función $f(x_1, x_2) = sign((x_1 - 0{,}2)^2 + x_2^2 - 0{,}6)$ que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un $10\,\%$ de las mismas. Pintar el mapa de etiquetas obtenido.**

  c) **Usando como vector de características $(1, x_1, x_2)$ ajustar un modelo de regresión lineal al conjunto de datos generado y estimar los pesos w. Estimar el error de ajuste E_{in} usando Gradiente Descendente Estocástico (SGD).**

  d) **Ejecutar todo el experimento definido por (a)-(c) 1000 veces (generamos 1000 muestras diferentes) y**
     - **Calcular el valor medio de los errores E_{in} de las 1000 muestras.**
     - **Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de E_{out} en dicha iteración. Calcular el valor medio de E_{out} en todas las iteraciones.**

  e) **Valore que tan bueno considera que es el ajuste con este modelo lineal a la vista de los valores medios obtenidos de E_{in} y E_{out}.**

3. Bonus

<u>Método de Newton.</u> Implementar el algoritmo de minimización de Newton y aplicarlo a la función $f(x, y)$ dada en el ejercicio 3. Desarrolle los mismos experimentos usando los mismos puntos de inicio.

- **Generar un gráfico de como desciende el valor de la función con las iteraciones.**
- **Extraer conclusiones sobre las conductas de los algoritmos comparando la curva de decrecimiento de la función calculada en el apartado anterior y la correspondiente obtenida con gradiente descendente.**