Aprendizaje Automático (2018-2019)
Grado en Ingeniería Informática
Universidad de Granada

Informe práctica 1 : Programación

Montserrat Rodríguez Zamorano

25 de marzo de 2019

Índice

1. Ejercicio sobre la búsqueda iterativa de óptimos

Gradiente Descendente

El código de este ejercicio está implementado en el archivo `p1ej1.py`.

1. Implementar el algoritmo de gradiente descendente

El código implementado puede verse en (1). Los parámetros de la función son los siguientes:

- `fun`: función sobre la que aplicar el gradiente descendente.
- `learning_rate`: tasa de aprendizaje.
- `maxIterations`: número máximo de iteraciones.
- `error`: error mínimo a partir del cual detenemos la ejecución.
- `ini_point`: punto inicial.
- `grad`: gradiente de la función sobre la que se va a buscar el mínimo.

```
def gradient_descent(fun, learning_rate,maxIterations,error,
    ini_point, grad):
     #inicializar numero de iteraciones
     iterations = 0
     #inicializar w y w_arr
     w = np.array(np.float64([ini_point[0],ini_point[1]]))
     val_arr = [np.float64(fun(ini_point[0],ini_point[1]))]
     while iterations<maxIterations:
          #actualizar numero de iteraciones
          iterations +=1
          #guardamos el ultimo w calculado
          w_ant = np.array([w[0],w[1]])
          #actualizar w
          w = w - learning_rate*grad(w_ant[0],w_ant[1])
          #actualizar w_arr
          val_arr.insert(len(val_arr), np.float64(fun(w[0],w[1])))
          #si el error es minimo, detenemos la ejecucion
          if(abs(np.float64(fun(w[0],w[1])-fun(w_ant[0],w_ant[1])))<
             error):
              break
     return w, val_arr, iterations
```

Listing 1: Código gradiente descendente

La función devolverá las coordenadas calculadas, un array con las coordenadas que ha obtenido el algoritmo a lo largo de las iteraciones y el número de éstas que se han necesitado para alcanzarlas.

2. Considerar la función $E(u,v) = (u^2e^v - 2v^2e - u)^2$. Usar gradiente descendente para encontrar un mínimo de esta función, comenzando desde el punto (u,v) = $(1,1)$ y usando una tasa de aprendizaje $\eta = 0{,}01$.

Para resolver el ejercicio se usará la función `gradient_descent` como puede verse en (2).

```
eta = 0.01
maxIter = 10000000000
error2get = 1e-14
initial_point = np.array([1.0,1.0],dtype=np.float64)
w, w_arr, it = gradient_descent(E,eta,maxIter,error2get,
    initial_point, gradE)
```

Listing 2: Usar gradiente descendente para encontrar un mínimo

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

Calculamos las derivadas parciales:

$$\begin{aligned}\frac{\partial E}{\partial u}(u,v) &= 2(u^2e^v - 2v^2e^{-u})(2ue^v + 2v^2e^{-u}) = 2(u^2e^v - 2v^2e^{-u})(2ue^v + 2v^2e^{-u}) \\ &= 2(2u^3e^{2v} + 2v^2u^2e^{v-u} - 4v^2ue^{v-u} - 2v^4e^{-2u}) \\ &= 4(u^3e^{2v} + v^2u^2e^{v-u} - 2v^2ue^{v-u} - 2v^4e^{-2u})\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial v}(u,v) &= 2(u^2e^v - 2v^2e^{-u})(u^2e^v - 4ve^{-u}) \\ &= 2(u^4e^{2v} - 4u^2ve^{v-u} - 2v^2u^2e^{v-u} + 8v^3e^{-2u})\end{aligned}$$

El gradiente de la función será:

$$\nabla E(u,v) = \left(\frac{\partial E}{\partial u}(u,v), \frac{\partial E}{\partial v}(u,v)\right)$$

Si mostramos el gradiente en la consola de Python, el resultado será el que puede verse en la imagen (1.1).

```
Apartado (a)
Mostrar el gradiente:
 (2.71828182845905**v*u**2 - 2*2.71828182845905**(-u)*v**2)**2

--- Pulsar enter para continuar ---
```

Figura 1.1: Mostrar la expresión del gradiente en la consola de Python

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)

El algoritmo tarda 34 iteraciones en obtener un error mínimo. Puede verse el resultado de la ejecución en (1.2).

```
Apartado (b) Calcular número de iteraciones.

Numero de iteraciones:  34

--- Pulsar enter para continuar ---
```

Figura 1.2: Número de iteraciones en la consola de Python

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

Las coordenadas del punto mínimo obtenidas han sido

$$(u, v) = (0{,}6192076718344792, 0{,}9684482706760941)$$

. Puede verse el resultado de la ejecución en (1.3).

```
Apartado (c) Calcular coordenadas.

Coordenadas obtenidas: ( 0.6192076718344792 ,  0.9684482706760941 )

--- Pulsar enter para continuar ---
```

Figura 1.3: Mínimo en la consola de Python

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

a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial $x_0 = 0{,}1, y_0 = 0{,}1$, 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.

Los fragmentos de código que se utilizarán para generar las gráficas son (3) y (4). El resultado puede verse en las figuras (1.4) y (1.5), respectivamente

```
w, w_arr, it = gradient_descent(F, 0.01,50,1e-14,np.array
    ([0.1,0.1]),gradF)
print ('Numero de iteraciones: ', it)
print ('Coordenadas obtenidas: (', w[0], ', ', w[1],')')
plt.plot(w_arr)
plt.title("Experimento con tasa de aprendizaje eta = 0.01")
plt.xlabel("Numero de iteraciones")
plt.ylabel("f(x,y)")
plt.show()
```

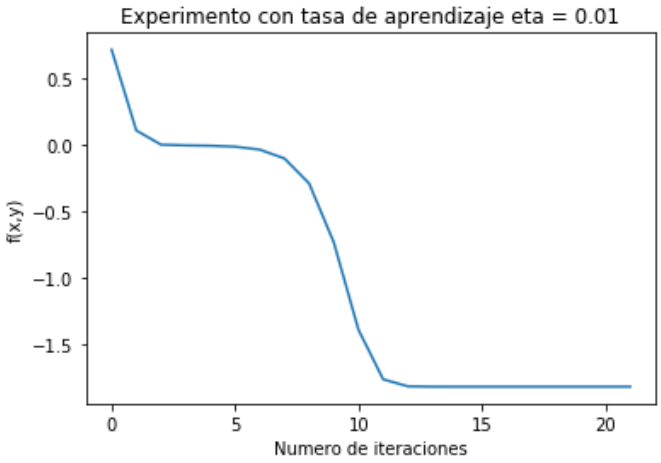
Listing 3: Código primera gráfica

```
w, w_arr, it = gradient_descent(F, 0.1,50,1e-14,np.array
    ([0.1,0.1]),gradF)
print ('Numero de iteraciones: ', it)
print ('Coordenadas obtenidas: (', w[0], ', ', w[1],')')
plt.plot(w_arr)
plt.title("Experimento con tasa de aprendizaje eta = 0.1")
```

```
plt.xlabel("Numero de iteraciones")
plt.ylabel("F(x,y)")
plt.show()
```

Listing 4: Código segunda gráfica

```
Numero de iteraciones:  21
Coordenadas obtenidas: ( 0.2438049668585023 ,  -0.23792582055545192 )
```




```
--- Pulsar enter para continuar ---
```

Figura 1.4: Descenso del valor de la función f(x,y) con $\eta = 0{,}01$

```
Numero de iteraciones:  50
Coordenadas obtenidas: ( 2.045834788227532 ,  -0.17033072304282185 )
```


```
--- Pulsar enter para continuar ---
```

Figura 1.5: Descenso del valor de la función f(x,y) con $\eta = 0{,}1$

Como puede verse, cuando la tasa de aprendizaje es $\eta = 0{,}01$, el valor de la función converge, obteniendo un mínimo y terminando la ejecución en 21 iteraciones. Por otro lado, si $\eta = 0{,}1$, se alcanza el número máximo de iteraciones sin obtener el mínimo. Como puede verse en la gráfica, el valor de la función no converge.

Como conclusión, la elección de la tasa de aprendizaje es importante a la hora de ejecutar la función del gradiente descendente, ya que una tasa de aprendizaje mal escogida hará que la función no converja.

b) Obtener el valor mínimo y los valores de las variables (x,y) en donde se alcanzan cuando el punto de inicio se fija: $(0{,}1, 0{,}1), (1, 1), (-0{,}5, -0{,}5), (-1, -1)$. Generar una tabla con los valores obtenidos.

Se ejecutará el código (5) para obtener el siguiente resultado en la consola de Python (1.6).

```
print("Valor␣inicial␣(0.1,0.1)")
w, w_arr, it = gradient_descent(F,0.01,50,1e-14,([0.1,0.1]),
    gradF)
print ('Coordenadas␣obtenidas:␣(', w[0], ',␣', w[1],')')
print ('Valor␣minimo:␣', F(w[0],w[1]))
print("Valor␣inicial␣(1,1)")
w, w_arr, it = gradient_descent(F,0.01,50,1e-14,([1,1]),gradF)
print ('Coordenadas␣obtenidas:␣(', w[0], ',␣', w[1],')')
print ('Valor␣minimo:␣', F(w[0],w[1]))
print("Valor␣inicial␣(-0.5,-0.5)")
w, w_arr, it = gradient_descent(F,0.01,50,1e-14,([-0.5,-0.5]),
    gradF)
print ('Coordenadas␣obtenidas:␣(', w[0], ',␣', w[1],')')
print ('Valor␣minimo:␣', F(w[0],w[1]))
print("Valor␣inicial␣(-1,-1)")
w, w_arr, it = gradient_descent(F,0.01,50,1e-14,([-1,-1]),gradF
    )
print ('Coordenadas␣obtenidas:␣(', w[0], ',␣', w[1],')')
print ('Valor␣minimo:␣', F(w[0],w[1]))
```

Listing 5: Código para aplicar gradiente descendente con distintos puntos de inicio

```
Ejercicio 3b
Valor inicial (0.1,0.1)
Coordenadas obtenidas: ( 0.2438049668585023 ,  -0.23792582055545192 )
Valor minimo:  -1.8200785415471559
Valor inicial (1,1)
Coordenadas obtenidas: ( 1.2180702996828958 ,  0.7128119496190772 )
Valor minimo:  0.5932693743258357
Valor inicial (-0.5,-0.5)
Coordenadas obtenidas: ( -0.731377457654812 ,  -0.23785536331334656 )
Valor minimo:  -1.3324810623309777
Valor inicial (-1,-1)
Coordenadas obtenidas: ( -1.2180702996828958 ,  -0.7128119496190772 )
Valor minimo:  0.5932693743258357


--- Pulsar enter para continuar ---
```

Figura 1.6: Resultado de la ejecución del código 5

Recogemos los valores obtenidos en la tabla (1.1). Como puede observarse, dependiendo del

Valor inicial	Valor mínimo	$f(x, y)$
(0.1,0.1)	-1.8200785415471559	(0.2438049668585023, -0.23792582055545192)
(1,1)	0.5932693743258357	(1.2180702996828958, 0.7128119496190772)
(-0.5,-0.5)	-1.3324810623309777	(-0.731377457654812, -0.23785536331334656)
(-1,-1)	0.5932693743258357	(-1.2180702996828958, -0.7128119496190772)

Tabla 1.1: Comparativa valores mínimos y coordenadas donde se obtienen

punto de inicio, se obtendrá un valor mínimo u otro. Incluso si el valor mínimo es el mismo, puede obtenerse desde puntos de inicio diferentes llegando a mínimos locales diferentes. Observar las filas correspondientes a los valores iniciales $(1, 1)$ y $(-1, -1)$).

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

La dificultad se encuentra en acertar escogiendo los parámetros que se le pasan a la función:

- **Tasa de aprendizaje:** si la tasa de aprendizaje es muy pequeña, la ejecución llevará demasiado tiempo. Sin embargo, si es demasiado grande, el valor nunca convergerá y no obtendremos el mínimo.
- **Punto de inicio:** dependiendo del punto de inicio escogido, el mínimo que obtenemos podrán ser distintos mínimos locales, obtendremos un mínimo local en lugar de uno global, etc.

2. Ejercicio sobre regresión lineal

1. Estimar un modelo de regresión lineal a partir de los datos proporcionados de dichos números (intensidad promedio, simetría) usando tanto el algoritmo de la pseudoinversa 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).

No se ha conseguido hacer este ejercicio debido a problemas en la ejecución del código, en concreto, con la función `Err(x,y,w)`. Se presentan las funciones que se han programado para ambos algoritmos.

En primer lugar, la función del gradiente descendente estocástico (6), siguiendo el pseudocódigo que puede verse en las transparencias de teoría. Se creará una secuencia de minibatches distinto para cada iteración e iteramos sobre el minibatch. Si el error es mínimo, detendremos la ejecución del algoritmo.

```
# Gradiente Descendente Estocastico
def sgd(x,y, learning_rate,maxIterations,error,m):
    #inicializar
    iterations = 0
    w = 0
    while iterations < maxIterations:
        cost = 0
        #generamos secuencia de minibatches
        minibatchX, minibatchY = minibatch(x,y,m)
        #iteramos en el minibatch
        for i in range(m):
```

```
            w_ant = w
            #actualizamos los pesos
            w = w_ant - (2/m)*learning_rate*gradient(minibatchX,
                minibatchY,w_ant)
        #si el error es minimo, detenemos la iteracion
        if Err(minibatchX,minibatchY,w)<error:
            break
        iterations += 1
    return w, iterations, cost
```

Listing 6: Código gradiente descendente estocástico

En esta función se hace uso de las funciones auxiliares:

- `minibatch(x,y,m)`: listing (7). Genera una secuencia de minibatches.
- `gradient(x,y,m)`: listing (8). Calcula el gradiente para la actualización de pesos.
- `Err(x,y,m)`: listing (9). Calcula el coste de la solución actual.
- `h(x,w)`: listing (10). Devuelve el valor predecido.

```
def minibatch(x,y,tam):
    minibX = []
    minibY = []
    for i in range(tam):
        temp = np.random.choice(len(x))
        minibX.append(x[temp])
        minibY.append(y[temp])
    return minibX,minibY
```

Listing 7: Código auxiliar (I)

```
def gradient(x,y,w):
    predminusy = h(np.transpose(x),w)-y
    return (np.dot(x,predminusy))
```

Listing 8: Código auxiliar (II)

```
def Err(x,y,w):
    m = len(x)
    cost = np.sum(np.square(h(x,w)-y))/m
    return cost
```

Listing 9: Código auxiliar (III)

```
def h(x,w):
    return np.dot(x,w)
```

Listing 10: Código auxiliar (IV)

A continuación, el código que calcula la pseudoinversa, siguiendo los pasos vistos en las diapositivas.

```
def pseudoinverse(x,y):
    t = np.transpose(x)
    pseudoinv = np.dot(np.linalg.inv(np.dot(t,x)),t)
    w = np.dot(pseudoinv,y)
    return w
```

Listing 11: Código pseudoinversa