INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

EVOLUTIONARY COMPUTING

PRÁCTICA 2: DESCENSO DEL GRADIENTE

ALUMNO: ORTEGA VICTORIANO IVAN

PROF.: JORGE LUIS ROSAS TRIGUEROS

FECHA DE REALIZACIÓN DE LA PRÁCTICA: 27/02/2018

FECHA DE ENTREGA DEL REPORTE: 06/03/2018

MARCO TEÓRICO.

La optimización de funciones objetivo-multivariables no lineales necesita el empleo de técnicas robustas y eficientes. La eficiencia es importante debido a que estos problemas requieren procedimientos de solución iterativos, la robustez es una propiedad deseable debido a que predecir el comportamiento de funciones no lineales y de varias variables es imposible. Estos métodos tienen como objetivo determinar puntos $x^* = [x1, x2, x3 \ldots xn]$ que minimicen a $f(x1, x2, x3\ldots) \equiv f(x)$ Prácticamente para todos los métodos existe una estructura fundamental. El algoritmo comienza desde un punto inicial $x0$, a continuación, se determina, mediante una regla, una dirección de movimiento, y se sigue en esa dirección hasta llegar a un mínimo (relativo) de la función objetivo sobre esa recta. En ese nuevo punto se determina una nueva dirección utilizando la misma regla anterior y se repite el proceso, la búsqueda finaliza cuando se cumple con un criterio de convergencia. Como se puede suponer la diferencia entre los métodos radica en la regla mediante la cual se selecciona la dirección de movimiento en cada paso del algoritmo. [1]

El método del descenso máximo o descenso del gradiente es uno de los procedimientos más utilizados para minimizar una función diferenciable de varias variables. Un vector $\mathbf{d}$ se dice que es una dirección de descenso de una función f en un punto $\mathbf{x}$ si existe $\delta > 0$ tal que $f(\mathbf{x} + \lambda\mathbf{d}) < f(x)$ para todo $\lambda \in (0, \delta)$. [2] Si

$$f'(\mathbf{x}; \mathbf{d}) = \lim_{\lambda \to 0^+} \frac{f(\mathbf{x} + \lambda\mathbf{d}) - f(\mathbf{x})}{\lambda} < 0$$

entonces d es una dirección de descenso de f en x. La dirección de descenso máximo es a la dirección $\mathbf{d}$, con $||\mathbf{d}|| = 1$, que minimiza el límite anterior. Se demuestra que, si f es diferenciable en $\mathbf{x}$ con gradiente no nulo, entonces

$$-\frac{\nabla f(\mathbf{x})}{||\nabla f(\mathbf{x})||}$$

es la dirección de descenso máximo. El método del descenso máximo se mueve a lo largo de esta dirección, o, equivalentemente, a lo largo de la dirección $-\nabla f(\mathbf{x})$ hasta localizar un punto con gradiente nulo. Por esta razón, a este método se le conoce habitualmente como el método del gradiente. [2]

En redes neuronales, el Descenso del Gradiente es el algoritmo de entrenamiento más simple y también el más extendido y conocido. Solo hace uso del vector gradiente, y por ello se dice que es un método de primer orden. [3]

Aunque es muy sencillo, este algoritmo tiene el gran inconveniente de que, para funciones de error con estructuras con valles largos y estrechos, requiere muchas iteraciones. Se debe a que, aunque la dirección elegida es en la que la función de error disminuye más rápidamente, esto no significa que necesariamente produzca la convergencia más rápida. [3]

MATERIAL Y EQUIPO.

- Equipo de cómputo (PC o laptop).
- Python2 o Python3
- Matlplotlib

DESARROLLO DE LA PRÁCTICA.

Para esta práctica comenzamos por hacer un script de Python que se encargara de hacer el cálculo del gradiente para la función $f(x)=x^2$ y la función en sí, el cual se muestra en la figura 1 con su respectivo resultado para $x=0.5$:

```
1  dx = 0.0001
2
3  def grad1D(x):
4      return (f1D(x+dx)-f1D(x))/dx
5
6  def f1D(x):
7      return x*x
8
9  print(f1D(0.5))
10
11 print(grad1D(0.5))
```

```
0.25
1.0000999999998372
```

Figura 1. Script de Python ejecutado en Jupyter Notebook para el cálculo del gradiente de $f(x)=x^2$ con sus respectivos resultados.

De la imagen anterior vemos que el cálculo del gradiente no es el exacto, ya que se utilizó una secante para aproximar la derivada y si hacemos cada vez más pequeño el valor de dx esta se acercará cada vez más a 1, que es el valor exacto.

Así mismo, elaboramos un script de Python para calcular el gradiente de la función $f(x,y)= x^2 +y^2$, que se muestra a continuación en la figura 2 con sus respectivas salidas evaluando en el punto (0.5,0.5):

```python
dx = 0.00001
dy = 0.00001

def grad2D(x,y):
    return (f2D(x + dx,y) - f2D(x,y))/dx,(f2D(x,y + dy) - f2D(x,y))/dy

def f2D(x,y):
    return x*x + y*y

print(f2D(0.5,0.5))

print(grad2D(0.5,0.5))
```

```
0.5
(1.0000099999962764, 1.0000099999962764)
```

Figura 2. Script de Python ejecutado en Jupyter Notebook para el cálculo del gradiente de $f(x,y)=x^2+ y^2$ con sus respectivos resultados.

Igual al caso anterior, en la sesión de laboratorio observamos que a medida que hagamos más pequeños los valores de dx y dy, el valor del gradiente evaluado en un punto será más cercano al valor exacto.

Nuestro siguiente trabajo, fue hacer un script que ejecutara el algoritmo del gradiente descendiente, esto se hizo mediante un loop, en el que se avanzara hacia la dirección en la que la función decrecía, ya que el objetivo es encontrar un mínimo. El script resultante se muestra en la figura 3 con su respectivo resultado, en la cual podemos apreciar que, por cada iteración, el algoritmo se aproxima a el punto (0,0) donde ya se sabe con anticipación que se encuentra el mínimo de esa función. Además, el avance depende en gran medida del valor de gamma, pues este determina si se darán pasos más grandes en el avance o más pequeños. Se recomiendan valores pequeños (no mucho), cosa que también sucede en el caso del algoritmo backpropagation para el entrenamiento de redes neuronales multicapa, pues si los pasos son muy grandes o muy pequeños se puede causar un sobre entrenamiento de la red, y es necesario implementar algoritmos para evitarlo como es el caso del conocido “early-stopping”.

```python
dx = 0.0001
gamma = 0.1

def grad1D(x):
    return (f1D(x+dx)-f1D(x))/dx

def f1D(x):
    return x*x

def gradDescent1D(x):
    for i in range(10):
        x -= gamma*grad1D(x)
        print(x,f1D(x),grad1D(x))

print(f1D(0.5))

print(grad1D(0.5))

print(gradDescent1D(0.5))
```

```
0.25
1.0000999999998372
0.3999900000000163 0.15999200010001302 0.8000799999999253
0.31998200000002375 0.1023884803240152 0.640063999999968
0.25597560000002695 0.06552350779537379 0.5120512000000854
0.2047704800000184 0.041930949479437936 0.4096409600000406
0.16380638400001435 0.026832531439160156 0.3277127680000047
0.13103510720001388 0.017170199318919128 0.2621702144000107
0.10481808576001281 0.0109868311023934 0.20973617152003632
0.08384446860800918 0.007029894916159437 0.16778893721602212
0.06706557488640696 0.004497791334844261 0.13423114977281422
0.05364245990912554 0.002877513505102141 0.10738491981825224
```

Figura 3. Script de Python ejecutado en Jupyter Notebook el algoritmo del gradiente descendiente para $f(x)=x^2$ con sus respectivos resultados.

Así mismo, se realizó el mismo algoritmo pero para la función de dos variables $f(x,y)=x^2+y^2$, en el que se muestran los resultados y el código en la figura 4:

```python
dx = 0.0001
dy = 0.0001
gamma = 0.1

def grad2D(x,y):
    return (f2D(x + dx,y) - f2D(x,y))/dx, (f2D(x,y + dy) - f2D(x,y))/dy

def f2D(x,y):
    return x*x + y*y

def gradDescent2D(x,y):
    for i in range(10):
        g = grad2D(x,y)
        x -= gamma*g[0]
        y -= gamma*g[1]
        print(x,y,f2D(x,y),grad2D(x,y))

print(f2D(0.5,0.5))
print(grad2D(0.5,0.5))
print (gradDescent2D(0.5,0.5))
```

```
0.5
(1.0000999999992821, 1.0000999999992821)
0.3999900000000718 0.3999900000000718 0.31998400020011486 (0.8000799999996477, 0.8000799999996477)
0.319982000000107 0.319982000000107 0.20477696064813697 (0.6400640000001068, 0.6400640000001068)
0.25597560000009634 0.25597560000009634 0.13104701559081863 (0.512051200000363, 0.512051200000363)
0.20477048000006004 0.20477048000006004 0.08386189895890998 (0.4096409600000406, 0.4096409600000406)
0.16380638400005598 0.16380638400005598 0.053665062878347596 (0.3277127680000741, 0.3277127680000741)
0.13103510720004857 0.13103510720004857 0.03434039863785644 (0.2621702144001148, 0.2621702144001148)
0.1048180857600371 0.1048180857600371 0.021973662204796983 (0.209736171520071, 0.209736171520071)
0.08384446860802999 0.08384446860802999 0.014059789832325854 (0.1677889372160568, 0.1677889372160568)
0.06706557488642431 0.06706557488642431 0.008995582669693174 (0.13423114977286627, 0.13423114977286627)
0.05364245990913685 0.05364245990913685 0.005755027010206888 (0.10738491981827393, 0.10738491981827393)
```

Figura 4. Script de Python ejecutado en Jupyter Notebook el algoritmo del gradiente descendiente para $f(x,y)=x^2+y^2$ con sus respectivos resultados.

De la figura anterior, al igual que el caso para una variable, se pudo ver que, por cada iteración, el algoritmo se acercaba cada vez más al punto (0,0,0), que se sabía que era el mínimo de dicha función.

Para visualizar estos datos, graficamos los resultados utilizando Matlplotlib, para el caso de una variable basta con una gráfica en dos dimensiones, mientras que para el caso de 2 variables se hará uso de curvas de nivel. El caso de una variable se muestra en la figura 5 y el de 2 variables en la figura 6. Nota: Los códigos además de anexarse al envío con este archivo de práctica se encontrarán en mi repositorio de GitHub para una mejor visualización del código: https://github.com/IvanovskyOrtega/Evolutionary-Computing/tree/master/Practica-2

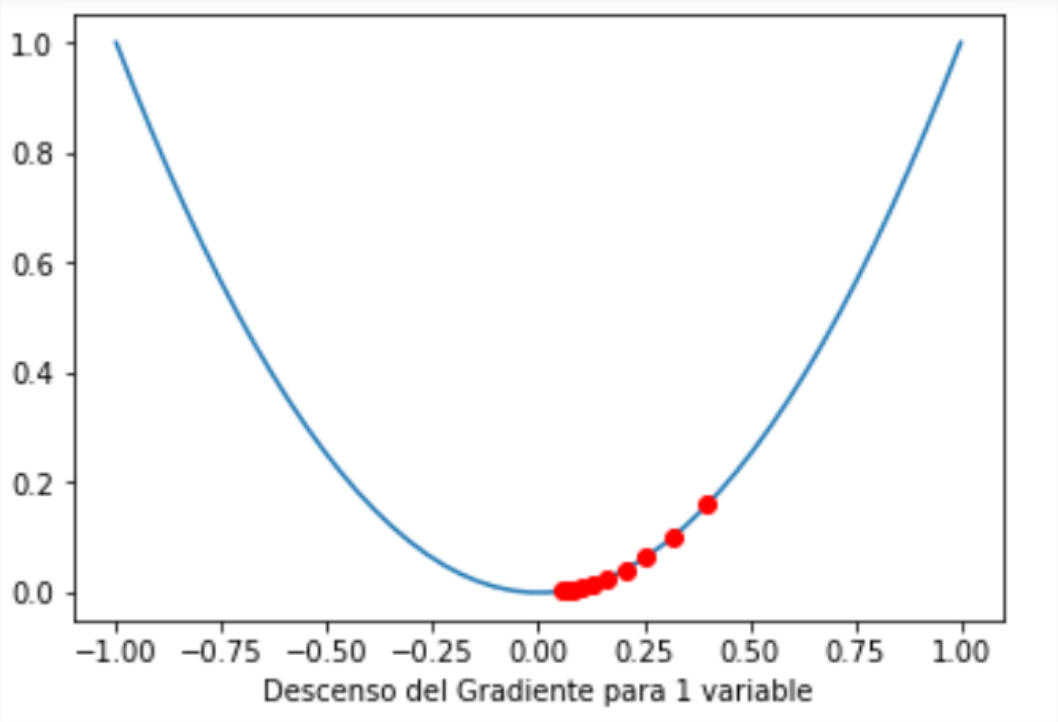

Figura 5. Descenso del gradiente para 1 variable.

Como se puede apreciar en los puntos rojos, estos se van acercando cada vez más al mínimo global en cada iteración, lo ideal sería hacer más iteraciones sin llegar a un punto en el que la concavidad de la curva cambie pase a ser creciente, o hacer muchas iteraciones, pero hacer más pequeño el valor de gamma para tener avances más pequeños.

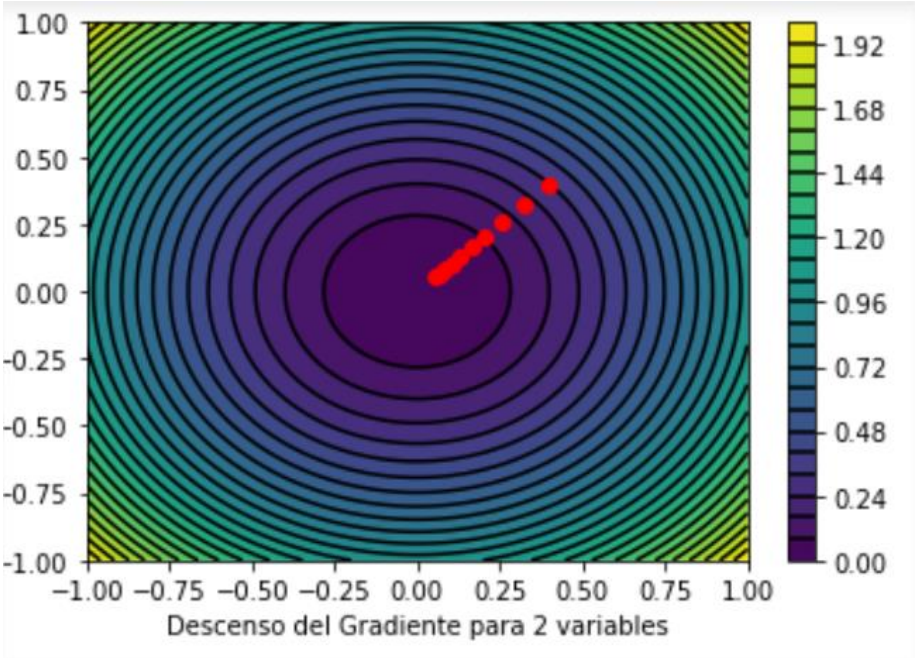

Figura 6. Descenso del gradiente para 2 variables.

Al igual que en el caso de 1 variable vemos que por cada iteración del algoritmo, este se acerca cada vez más al mínimo global.

Por último, se nos pidió realizar la tabla para el KP 0-1, es decir, resolviendo el problema mediante un algoritmo de programación dinámica (solución incompleta, pues falta reconstruir la solución mediante los datos de la tabla). El resultado se muestra a continuación en la figura 7 (en la misma figura se muestran los valores de w y v utilizados):

```
w = [2,3,4,4,5]
v = [3,3,3,5,6]
```

```
[0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3]
[0, 0, 3, 3, 3, 6, 6, 6, 6, 6, 6]
[0, 0, 3, 3, 3, 6, 6, 6, 6, 9, 9]
[0, 0, 3, 3, 5, 6, 8, 8, 8, 11, 11]
[0, 0, 3, 3, 5, 6, 8, 9, 9, 11, 12]
```

Figura 7. Resultados del algoritmo para resolver el *0-1 Knapsack Problem* utilizando programación dinámica.

CONCLUSIONES.

Con esta práctica pude entender un poco más acerca de la teoría sobre el método del descenso del gradiente, ya lo había visto previamente en la materia optativa de Redes Neuronales para su uso en el algoritmo backpropagation, sin embargo, no se nos dio mucha información sobre cómo funcionaba. Además, practiqué un poco más el uso de Pyplot en Python, en la práctica anterior me resultó más complicado, pero en esta fue un poco más sencillo. En cuanto a la implementación del algoritmo para hacer la tabla del *0-1 Knapsack Problem*, había algunos errores con el propuesto en clase en su mayoría con los tamaños de la matriz y el manejo de índices, por lo que tuve que corregirlos para tener el resultado adecuado.

Referencias

[1] «Métodos matemáticos de optimización no restringida. Búsqueda multivariable.,» [En línea]. Available: http://www.fio.unicen.edu.ar/usuario/cgely/q13-0/Apuntes/unidad4b.pdf.

[2] «Método del descenso máximo. Optimización de funciones diferenciables de varias variables.,» [En línea]. Available: https://webs.um.es/mpulido/miwiki/lib/exe/fetch.php?id=pnl&cache=cache&media=wiki:prpnl4.pdf.

[3] F. S. Caparrini, «Entrenamiento de Redes Neuronales: mejorando el Gradiente Descendiente,» [En línea]. Available: http://www.cs.us.es/~fsancho/?e=165.