Práctica 1
Aprendizaje Automático

Ignacio Aguilera Martos

25 de Marzo de 2019

Índice

1. Ejercicio 1

1.1. Apartado 1

Se nos pide implementar el algoritmo de gradiente descendente, veamos como funciona para justificar la implementación.

El algoritmo de gradiente descendente aplicado a minimización de funciones toma la idea del ajuste de una función lineal que hemos estudiado en teoría. Ahora en vez de actualizar w_j de forma proporcional a la derivada parcial j-ésima de el error dentro de la muestra vamos a actualizarla mediante la derivada parcial j-ésima de la función que queremos minimizar.

El algoritmo parte de un punto w inicial, que se irá actualizando hasta aproximar el mínimo de la función. El valor de w en la iteración i+1-ésima será el valor de w en la iteración i-ésima menos una constante de proporcionalidad que llamamos tasa de aprendizaje por el gradiente de la función a minimizar, esto es:

$$w_{i+1} = w_i - \eta \cdot \nabla f(x_1, ..., x_n)$$

Donde $f(x_1, ..., x_n)$ es la función que queremos minimizar.

De esta forma lo que vamos a hacer en la práctica es comprobar lo siguiente: vamos a actualizar de esta forma el valor de w hasta que agotemos un número de iteraciones máximas fijadas o hasta que el valor absoluto de la diferencia de $f(w_i))$ y $f(w_{i+1})$ sea menor que una cierta tolerancia, esto es que la imagen de los w_i y w_{i+1} no hayan experimentado un cambio apreciable entre dos iteraciones consecutivas.

Cabe destacar que en mi caso he generalizado la implementación del algoritmo mediante el uso de la librería SymPy. Mediante esta librería he implementado una función que generaliza el cálculo del gradiente en un punto, de tal forma que no tengo que programar explícitamente la función que calcula el gradiente de la función que queremos minimizar. Esto se plasma en las funciones "evaluate" y "gradiente" de mi código.

1.2. Apartado 2

1.2.1. Apartado a

En este apartado se nos proporciona la siguiente función a minimizar:

$$E(u, v) = (u^2 e^v - 2v^2 e^{-u})^2$$

Vamos a calcular analíticamente la expresión del gradiente como se nos pide en el ejercicio. Lo primero que tenemos que hacer es ver cómo sería la expresión del gradiente en nuestro caso:

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

Calculemos pues cada una de las dos parciales.

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

$$\frac{\partial E(u,v)}{\partial v} = 2(u^2e^v - 2v^2e^{-u})(u^2e^v - 4ve^{-u})$$

Por tanto:

$$\nabla E(u,v) = (\frac{\partial E(u,v)}{\partial u}, \frac{\partial E(u,v)}{\partial v}) = 2(u^2e^v - 2v^2e^{-u}) \cdot [(2ue^v + 2v^2e^{-u}), (u^2e^v - 4ve^{-u})](u,v)$$

Cabe decir que esta es una función $\nabla E(u,v) : \mathbb{R}^2 \to \mathbb{R}^2$, es decir que toma dos variables y que obtiene un vector de dimensión 2.

1.2.2. Apartado b y c

En este apartado se nos pide dar el punto en el que el método de gradiente descendente ha alcanzado por primera vez un valor de la función menor que 10^{-14} y el número de iteraciones que ha consumido hasta obtener dicho valor.

En este caso el punto en el que se ha encontrado un valor que cumple esta restricción es $w = (0{,}619207678450638, 0{,}968448269010049)$ y para ello ha consumido 33 iteraciones.

1.3. Apartado 3

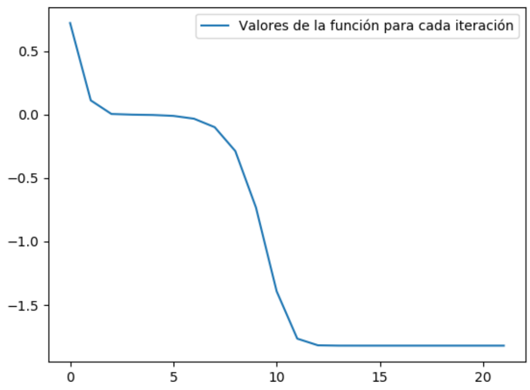
Consideramos en este apartado la función:

$$f(x,y) = x^2 + 2y^2 + 2\sin(2\pi x)\sin(2\pi y)$$

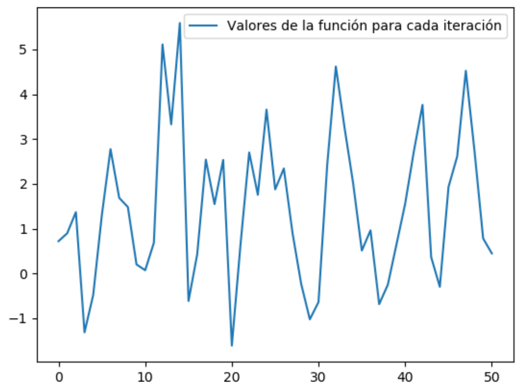
1.3.1. Apartado a

Se nos pide aplicar el algoritmo de gradiente descendente a la función anteriormente definida empezando en el punto $(x_0 = 0{,}1, y_0 = 0{,}1)$ con tasa de aprendizaje $\eta = 0{,}01$ y como máximo 50 iteraciones.

Tras esto se nos pide elaborar un gráfico con los valores de la función obtenidos para cada iteración. Veamos los gráficos para poder analizarlos.



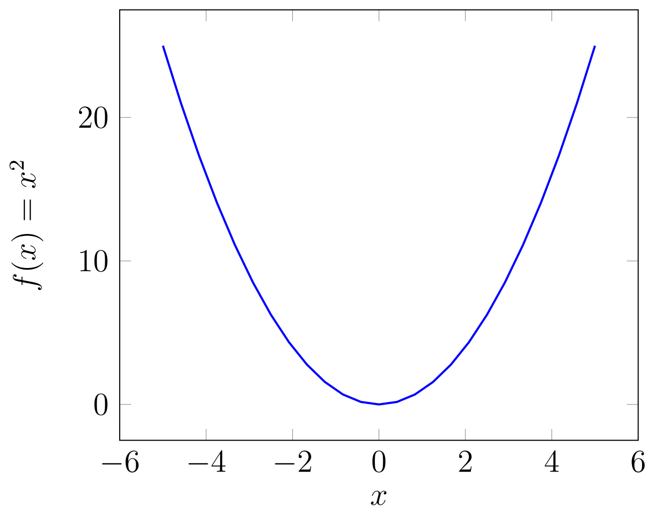
(a) Tasa de aprendizaje $\eta = 0{,}01$



(b) Tasa de aprendizaje $\eta = 0{,}1$

Como podemos observar el resultado usando $\eta = 0{,}01$ siempre mejora en cada iteración, o a lo sumo permanece en un valor igual, en cambio con $\eta = 0{,}1$ obtenemos una oscilación pronunciada entre iteraciones, esto es, entre dos iteraciones consecutivas no siempre pasamos a un punto con menor valor de la función, si no que mejoramos y empeoramos el resultado de forma cíclica.

Cabe explicar este comportamiento de forma más detallada para razonarlo. La tasa de aprendizaje nos da el factor de salto por el que va multiplicado el gradiente de la función, es decir, a mayor tasa de aprendizaje mayor será el salto que hagamos para obtener el siguiente valor de w. Supongamos que tenemos una función como $f(x) = x^2$:



Según el valor de la tasa de aprendizaje η podemos encontrarnos con los dos escenarios que nos están ocurriendo:

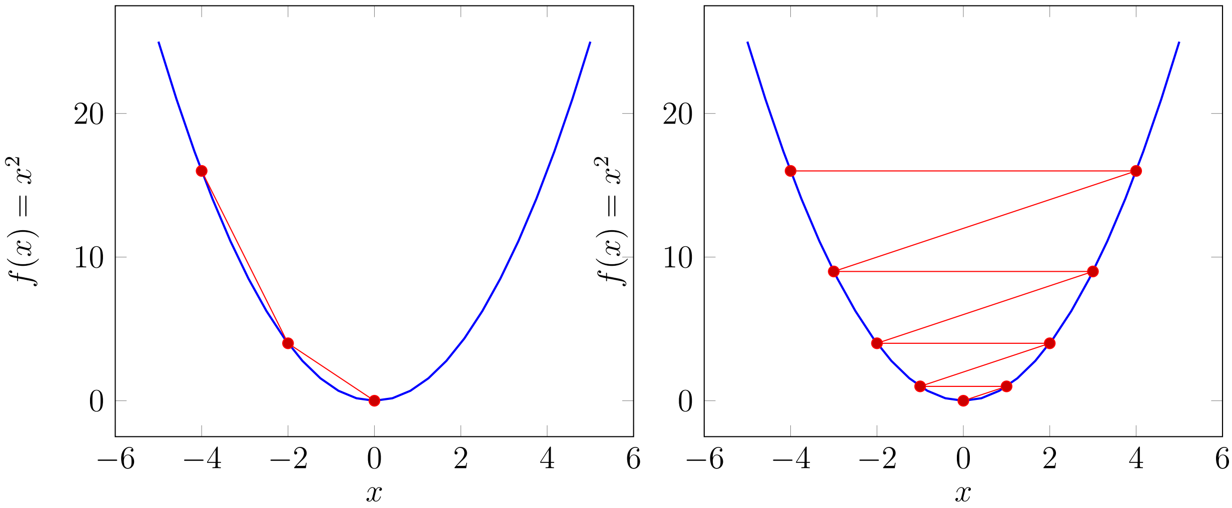


Figura 2: Ejemplo de comportamiento del gradiente descendente

Como podemos ver una aproximación no realiza "saltos" alrededor del mínimo, si no que va de forma directa al mínimo, mientras que en el otro caso tenemos una oscilación alrededor del mínimo.

Si estudiamos la dependencia del comportamiento del algoritmo en función de la tasa de aprendizaje η vemos que la magnitud del salto del punto actual al siguiente viene dada exactamente por η en la dirección del gradiente de la función.

En nuestro caso, con $\eta = 0{,}01$ podemos osbervar en la gráfica **1a** que la convergencia es de forma gradual y sin saltos alrededor del mínimo, pues siempre que se progresa en cada iteración se hace obteniendo un valor menor de la función, es decir estaríamos en el primero de los casos expuestos de la figura **2**.

En el segundo caso con $\eta = 0{,}1$ podemos observar en la gráfica **1b** que el comportamiento es errático. Si observamos la evolución entre iteraciones podemos observar que no mejora si no que oscila alrededor del mínimo sin llegar a él. Esto es debido a que la longitud del salto, es decir, el valor de η es demasiado grande y hace que avance demasiado entre iteraciones. Esto comparado con nuestro ejemplo de la figura **2** se correspondería con el segundo de los casos, en el que el camino hacia el mínimo de la función se hace oscilando entorno a él.

En la figura con $\eta = 0{,}1$ también se aprecia que la amplitud de valores que toma va disminuyendo, es decir, poco a poco se está dirigiendo hacia el mínimo. El algoritmo gradiente descendente llegará eventualmente al mínimo con este valor de η pero consumirá un número mucho mayor de iteraciones que si escogemos de forma más ajustada la tasa de aprendizaje.

Esto último mencionado no quiere decir que el algoritmo partiendo de un punto inicial fijo siempre converja hacia el mismo mínimo de la función, ya que puede que en función del valor de η las iteraciones vayan saltando hasta dar con otro mínimo local o global de la función.

1.3.2. Apartado b

Se nos pide obtener el valor mínimo con 50 iteraciones fijando los puntos de inicio $w_1 = (0{,}1, 0{,}1)$, $w_2 = (1, 1)$, $w_3 = (-0{,}5, -0{,}5)$, $w_4 = (-1, -1)$.

Veamos los valores mínimos obtenidos con estos puntos:

Punto inicial	Mínimo encontrado	Valor en el mínimo
[0.1 0.1]	[0.243804966858502 -0.237925820555452]	-1.82008
[1 1]	[1.21807029968290 0.712811949619077]	0.593269
[-0.5 -0.5]	[-0.731377457654812 -0.237855363313347]	-1.33248
[-1 -1]	[-1.21807029968290 -0.712811949619077]	0.593269

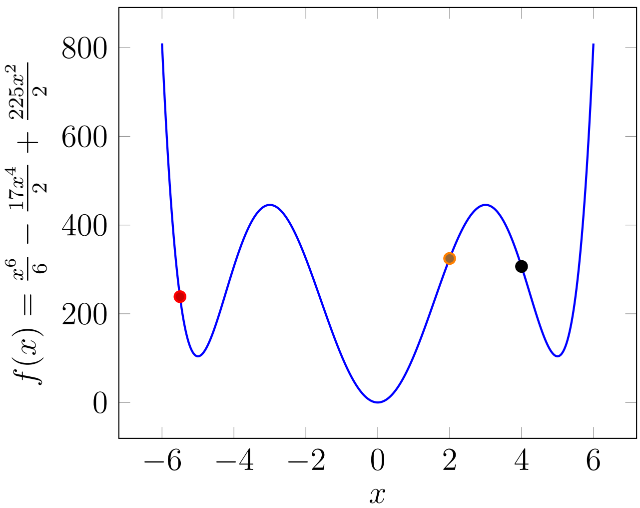
Como podemos observar el punto de inicio es altamente determinante para obtener el mínimo de la función ya que dependiendo del valor inicial podemos acabar obteniendo uno u otro mínimo local o incluso el mínimo global de la función. Como podemos ver inclusive en esta función (por ciertas razones de simetría de los senos) hemos acabado obteniendo como mínimo un punto y su opuesto como se puede ver en las filas 2 y 4 de la tabla.

1.4. Apartado 4

Si observamos todos los resultados que hemos obtenido hasta ahora podemos obtener dos conclusiones acerca de la dificultad de hallar un mínimo:

- La elección de la tasa de aprendizaje: si escogemos una tasa de aprendizaje muy grande podemos incluso forzar que la convergencia se produzca en un número de iteraciones excesivamente grande no obteniendo por tanto un mínimo satisfactorio de la función. Esto no es realmente un problema, puesto que siempre podemos escoger una tasa de aprendizaje muy pequeña para avanzar en pasos pequeños y no saltarnos el mínimo más próximo al punto de inicio. Esto nos obligaría probablemente a usar un número mayor de iteraciones, pero no influiría en el mínimo que obtenemos.

- El punto de inicio: esto es totalmente determinante. Supongamos que tenemos una función con una serie de mínimos locales y un mínimo global de la siguiente forma:



Si consideramos como punto de partida el punto rojo con una tasa de aprendizaje adecuada acabaremos llegando al mínimo local que tiene justo a su derecha, gracias a que el gradiente de la función “empuja” al algoritmo hacia ese punto.

Si tomásemos como punto inicial el punto negro iríamos también hacia el mínimo local que tiene a su derecha.

En cambio estos dos mínimos no son los mínimos globales de la función ya que de los tres puntos tomados sólo el naranja tendería hacia el mínimo real de la función. En este caso es fácil poder escoger el punto inicial del que partimos, incluso en una función de dos variables sería sencillo tomar este punto inicial pero imaginemos una función de 10 variables con valores en $\mathbb{R}$. Esta función ya no la podemos dibujar, por lo que no podemos tomar el punto de forma visual aproximada. Si no tenemos más información sobre la función este es el paso más complejo del algoritmo pues según nuestra elección del punto inicial obtendremos un verdadero mínimo global (si es que existe) o un mínimo local de la función.

2. Ejercicio 2

2.1. Apartado 1

Vamos en primer lugar a estudiar los algoritmos que se nos proponen para implementar.

El primero de los algoritmos implementados es el algoritmo de la pseudoinversa. Este algoritmo se basa en la resolución del método de mínimos cuadrados lineal, es decir, resolver el problema de ajustar una serie de datos mediante una función lineal minimizando el cuadrado de los errores a dicha función. Esto se puede visualizar con ejemplos clásicos como:

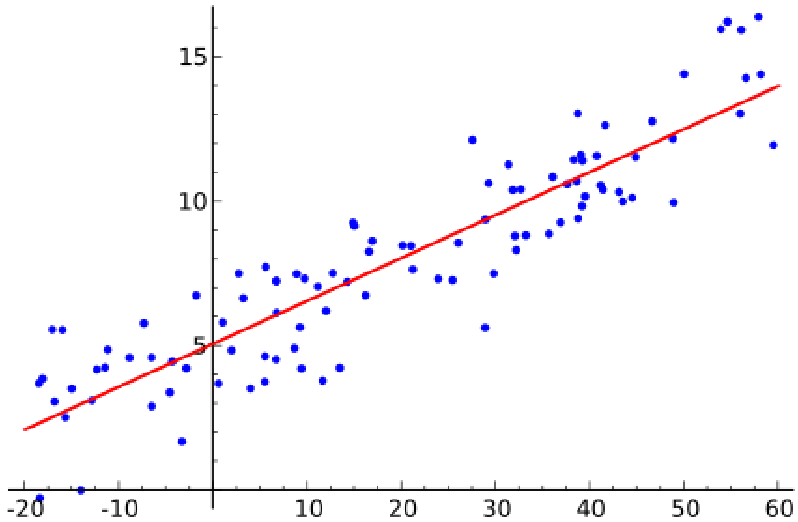


Figura 3: Ejemplo de la función lineal con mínimos cuadrados.

en la cual podemos intuir que la recta minimiza los errores entre los valores de la propia recta y los valores reales de los puntos del conjunto de datos.

El fundamento teórico parte de que podemos expresar el gradiente de $E_{in}(w)$ como:

$$\nabla E_{in}(w) = \frac{2}{N} X^T (Xw - y) = 0$$

De donde podemos obtener la aproximación de igualdad $X^T X w = X^T y$

Entonces podemos expresar w como $w = X^\dagger y$ donde $X^\dagger = (X^T X)^{-1} X^T$

Pero no tenemos la certeza de que podamos calcular siempre $(X^T X)^{-1}$, por lo que acudimos a lo que se define como la pseudo inversa de Moore-Penrose.

Partimos de la descomposición en valores singulares de la matrix $X = UDV^T$, entonces podemos obtener que $X^T X = VDDV^T$.

Si intentamos obtener la inversa de aquí entonces obtenemos $(X^T X)^{-1} = (V^T)^{-1}(D^2)^{-1}V^T$. Observemos que en general tampoco podemos considerar la inversa de la matriz diagonal D, pero en nuestra implementación de Python, al obtener la descomposición en valores singulares con NumPy se nos devuelve únicamente los elementos de D que no son cero. Con ellos podemos crear una matriz diagonal D que sí es cuadrada y que es la que emplearemos y a la que sí le podemos encontrar la inversa.

Visto ya todo esto sólo resta volver al punto anterior a la disertación sobre la inversa y ver que podemos definir $X^\dagger = (X^T X)^{-1} X^T$ calculado como se ha descrito previamente y observar que podemos por tanto expresar $w = X^\dagger y$.

Con este comportamiento descrito es como se ha implementado el algoritmo de la pseudo inversa.

Para gradiente descendente estocástico partimos de la base conocida del algoritmo gradiente descendente aplicado a regresión lineal.

La idea es la misma que en el algoritmo gradiente descendente. Queremos actualizar el valor de w en función de un factor η que llamamos tasa de aprendizaje y en función de $\frac{\partial E_{in}(w)}{\partial w_j}$, pero en este caso no tenemos en cuenta toda la muestra completa, si no que vamos a crear subconjuntos de la muestra que llamamos "minibatches". Estos subconjuntos tienen un tamaño finito, típicamente entre 32 y 128 (en mi implementación particular 64 por recomendación del profesor del prácticas).

Por tanto el valor de w lo vamos a actualizar tomando, como se muestra en las diapositivas de teoría:

$$\frac{\partial E_{in}(w)}{\partial w_j} = \frac{2}{M} \sum_{n=1}^{M} x_{nj}(h(x_n) - y_n)$$

Donde M es el tamaño del minibatch y tomamos los índices correspondientes al mismo para el cómputo del cálculo. Es decir, tomando la filosofía de gradiente descendente tendríamos:

$$w_j = w_j - \eta \frac{\partial E_{in}(w)}{\partial w_j}$$