Descenso del gradiente

Matemática IV - 2024

September 21, 2024

El siguiente documento introduce el tema de descenso de gradiente, el cual es un método de optimización de funciones iterativo para hallar el mínimo de una función. Este algoritmo es necesario para cosas como redes neuronales cuando se aproximan funciones/entrenan modelos. Aunque también tiene usos en otras ramas de la informática.

1 Introducción

Aunque la determinación de los extremos de una función a través de métodos analíticos, como la búsqueda de puntos estacionarios y el criterio de las derivadas segundas, es bien establecida, existen funciones que son complejas de analizar y optimizar. La optimización de funciones, ya sean de una o múltiples variables, es una aplicación fundamental del análisis matemático en diversas ciencias. Sin embargo, el avance de la informática ha permitido desarrollar soluciones algorítmicas que complementan estos métodos analíticos tradicionales.

La informática, al integrarse con disciplinas como Matemática y Física, facilitó el desarrollo del análisis numérico, una rama que ofrece métodos automatizados para resolver problemas matemáticos complejos, incluyendo la aproximación de extremos de funciones difíciles de manejar. Aunque estos métodos numéricos proporcionan aproximaciones de los valores verdaderos, estas aproximaciones pueden ser tan precisas como se desee, acercándose a los valores analíticos reales.

En esta sección, presentaremos el método numérico conocido como descenso de gradiente, que se utiliza para minimizar una función f(x) mediante un proceso iterativo de aproximación. El descenso de gradiente se basa en la evaluación del gradiente de la función y ajusta los parámetros para encontrar una solución cercana al mínimo verdadero. Este método es una de las múltiples técnicas disponibles para la optimización y tiene aplicaciones significativas en informática.

En aprendizaje automático y redes neuronales, el descenso de gradiente es esencial para entrenar modelos, ajustando los pesos y sesgos para minimizar funciones de pérdida y mejorar la precisión de las predicciones (se detallará más adelante). En procesamiento de imágenes, se utiliza para ajustar parámetros en algoritmos de reconocimiento de patrones y segmentación, facilitando tareas como el reconocimiento de objetos y la mejora de imágenes. Además, en optimización de algoritmos, el descenso de gradiente ayuda a afinar los parámetros de algoritmos complejos para mejorar su rendimiento y eficiencia. También juega un papel en sistemas de recomendación, donde se utiliza para optimizar modelos que predicen preferencias de usuarios en función de datos históricos.

2 Dirección de actualización

La motivación para esta técnica es la optimización de funciones, recordando un método de Matemática 2 para minimizar funciones que es igualar la derivada de una función a cero, identificando así candidatos a mínimos. Es difícil hablar de porque este método siempre llega al mínimo de la función. Uno de los casos en los que se encuentra un mínimo es que la función que estamos aproximando tiene un solo mínimo(esto es que la función sea convexa). Esta estrategia va a generar puntos $x^1, x^2, ..., x^t$ hasta eventualmente llegar al punto que minimiza la función.

Aproximación de la Función

Siguiendo el método de Matemática 2, primero tendremos que derivar la función y luego igualar su derivada a 0 para encontrar un mínimo. Dado que la función puede ser compleja, se utiliza una aproximación de la misma. En este contexto, empleamos como aproximación:

$$f(x) \approx f(x^t) + \nabla f(x^t) \cdot (x - x^t) + \frac{1}{2}(x - x^t)^T H (x - x^t)$$

donde H es la matriz Hessiana, definida como

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

y se aproxima en problemas de optimización con:

$$H \approx \frac{1}{\eta} I$$

Sustituyendo la Hessiana por su aproximación, nuestra función queda:

$$f(x) \approx f(x^t) + \nabla f(x^t) \cdot (x - x^t) + \frac{1}{2\eta} \|x - x^t\|^2$$

Interpretación

- El primer término $f(x^t)$ es constante respecto a x no nos interesa para la minimizacion.
- El segundo término $\nabla f(x^t) \cdot (x - x^t)$ representa la aproximación lineal dé f.
- El tercer término $\frac{1}{2\eta} \|x - x^t\|^2$ es el término cuadrático de corrección.

Formulación del Problema de Optimización

El objetivo es encontrar el punto que minimice nuestra aproximación:

$$\mathbf{x} = \arg\min_x \left[f(x^t) + \nabla f(x^t) \cdot (x - x^t) + \frac{1}{2\eta} \|x - x^t\|^2 \right]$$

Y como dijimos, esto lo hacemos igualando la derivada de la función a 0, tenemos que encontrar$\mathbf{x}$ tal que:

$$\frac{d}{d\mathbf{x}} \left[f(x^t) + \nabla f(x^t) \cdot (\mathbf{x} - x^t) + \frac{1}{2\eta} |\mathbf{x} - x^t|^2 \right] = 0$$

Desarrollo Paso a Paso

Consideramos $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ la entrada de la función, $x^{(t)} = (x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)})$ las coordenadas en el paso t-esimo de la iteración y el gradiente $\nabla f(x^{(t)}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$ evaluado en $x^{(t)}$.

Expansión de cada término:

$$\nabla f(x^{(t)}) \cdot (\mathbf{x} - x^{(t)}) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x^{(t)})(x_i - x_i^{(t)})$$

$f(x^{(t)})$ es un escalar.

$$\|\mathbf{x} - x^{(t)}\|^2 = \sum_{i=1}^{n} (x_i - x_i^{(t)})^2$$

Derivación respecto a cada componente x_k de x:

$$\frac{\partial}{\partial x_k} [\nabla f(x^{(t)}) \cdot (\mathbf{x} - x^{(t)})] = \sum_i [\frac{\partial f}{\partial x_i}(x^{(t)}) \cdot (x_i - x_i^{(t)})] =$$

Con la regla del producto

$$(\frac{\partial}{\partial x_k} [\frac{\partial f}{\partial x_i}(x^{(t)})]) \cdot (x_i - x_i^{(t)}) + \frac{\partial f}{\partial x_i}(x^{(t)}) \cdot (\frac{\partial}{\partial x_k} [x_i - x_i^{(t)}])$$

$\frac{\partial}{\partial x_k} [\frac{\partial f}{\partial x_i}(x^{(t)})] = 0$, porque $x^{(t)}$ es constante respecto a x_k y

$$\frac{\partial}{\partial x_k}[x_i - x_i^{(t)}] = 1 \text{ si } i = k, \text{ y } 0 \text{ si } i \neq k \text{ entonces}$$

$$\frac{\partial}{\partial x_k}[f(x^{(t)})] = 0$$

$$\frac{\partial}{\partial x_k}\left[\frac{1}{2\eta}\|\mathbf{x} - x^{(t)}\|^2\right] = \frac{1}{\eta}(x_k - x_k^{(t)})$$

Remplazamos todas las derivadas parciales:

$$\frac{\partial f}{\partial x_k}(x^{(t)}) + \frac{1}{\eta}(x_k - x_k^{(t)}) = 0$$

Resolvemos para x_k:

$$\frac{1}{\eta}(x_k - x_k^{(t)}) = -\frac{\partial f}{\partial x_k}(x^{(t)})$$

$$x_k - x_k^{(t)} = -\eta\frac{\partial f}{\partial x_k}(x^{(t)})$$

$$x_k = x_k^{(t)} - \eta\frac{\partial f}{\partial x_k}(x^{(t)})$$

Esto se cumple para cada componente k, por lo que en forma vectorial:

$$\mathbf{x} = x^{(t)} - \eta\nabla f(x^{(t)})$$

Este x minimiza la expresión, convirtiéndose en nuestro $x^{(t+1)}$:

$$\mathbf{x} = x^{(t)} - \eta\nabla f(x^{(t)})$$

Así llegamos al criterio de actualización del método de descenso de gradiente .

3 Construcción de un algoritmo

Para construir el algoritmo de descenso del gradiente, seguimos los siguientes pasos:

1. **Seleccionar un punto inicial** x_0.
2. **Seleccionar el valor de tolerancia** ϵ.
3. **Seleccionar el tamaño del paso** η.
4. **Mientras no se cumpla la condición de corte**: $|f(x_{k+1}) - f(x_k)| < \epsilon$:
   - Calcular el siguiente punto de la secuencia según la función de actualización:

$$x_{t+1} = x_t - \eta\nabla f(x_t)$$

Aquí, la condición de corte se define como la diferencia entre los valores de la función en puntos consecutivos siendo menor que la tolerancia ϵ.

3.1 Ejemplo

Veamos un ejemplo del algoritmo paso a paso. Consideremos la función:

$$f(x) = \frac{1}{2}x^2 + \frac{3}{2}$$

Para encontrar el mínimo de esta función, calculamos la derivada:

$$f'(x) = x$$

Igualando $f'(x)$ a 0, obtenemos el punto crítico:

$$x = 0$$

Este punto es donde la función alcanza su mínimo global. Evaluando la función en este punto:

$$f(0) = \frac{1}{2}(0)^2 + \frac{3}{2} = \frac{3}{2}$$

Por lo tanto, el mínimo de la función se encuentra en $x = 0$, y el valor de la función en este punto es $\frac{3}{2}$.

Parámetros

- **Punto inicial**: $x_0 = 1.4$
- **Tolerancia**: $\epsilon = 0.01$
- **Tamaño de paso**: $\eta = 0.4$

3.2 Iteraciones

Iteración 1:

$$\begin{aligned}
\text{Gradiente en } x_0 : \nabla f(x_0) &= x_0 = 1.4 \\
\text{Actualización del punto : } \quad x_1 &= x_0 - \eta \cdot \nabla f(x_0) = 1.4 - 0.4 \cdot 1.4 = 0.84 \\
\text{Evaluación de la función en } x_1 : f(x_1) &= \frac{1}{2}(0.84)^2 + \frac{3}{2} = 1.8528 \\
\text{Condición de corte : } \quad |f(x_1) - f(x_0)| &= |1.8528 - 2.48| = 0.6272
\end{aligned}$$

Como $0.6272 > 0.01$, debemos seguir iterando.

Iteración 2:

$$\begin{aligned}
\text{Gradiente en } x_1 : \nabla f(x_1) &= x_1 = 0.84 \\
\text{Actualización del punto : } \quad x_2 &= x_1 - \eta \cdot \nabla f(x_1) = 0.84 - 0.4 \cdot 0.84 = 0.504 \\
\text{Evaluación de la función en } x_2 : f(x_2) &= \frac{1}{2}(0.504)^2 + \frac{3}{2} = 1.627008 \\
\text{Condición de corte : } \quad |f(x_2) - f(x_1)| &= |1.627008 - 1.8528| = 0.225792
\end{aligned}$$

Como $0.225792 > 0.01$, debemos seguir iterando.

Iteración 3:

$$\begin{aligned}
\text{Gradiente en } x_2 : \nabla f(x_2) &= x_2 = 0.504 \\
\text{Actualización del punto : } \quad x_3 &= x_2 - \eta \cdot \nabla f(x_2) = 0.504 - 0.4 \cdot 0.504 = 0.3024 \\
\text{Evaluación de la función en } x_3 : f(x_3) &= \frac{1}{2}(0.3024)^2 + \frac{3}{2} = 1.545723 \\
\text{Condición de corte : } \quad |f(x_3) - f(x_2)| &= |1.545723 - 1.627008| = 0.081285
\end{aligned}$$

Como $0.081285 > 0.01$, debemos seguir iterando.

Iteración 4:

$$\begin{aligned}
\text{Gradiente en } x_3 : \nabla f(x_3) &= x_3 = 0.3024 \\
\text{Actualización del punto : } \quad x_4 &= x_3 - \eta \cdot \nabla f(x_3) = 0.3024 - 0.4 \cdot 0.3024 = 0.18144 \\
\text{Evaluación de la función en } x_4 : f(x_4) &= \frac{1}{2}(0.18144)^2 + \frac{3}{2} = 1.516460 \\
\text{Condición de corte : } \quad |f(x_4) - f(x_3)| &= |1.516460 - 1.545723| = 0.029263
\end{aligned}$$

Como $0.029263 > 0.01$, debemos seguir iterando.

Iteración 5:

$$\begin{aligned}
\text{Gradiente en } x_4 : \nabla f(x_4) &= x_4 = 0.18144 \\
\text{Actualización del punto : } \quad x_5 &= x_4 - \eta \cdot \nabla f(x_4) = 0.18144 - 0.4 \cdot 0.18144 = 0.108864 \\
\text{Evaluación de la función en } x_5 : f(x_5) &= \frac{1}{2}(0.108864)^2 + \frac{3}{2} = 1.505926 \\
\text{Condición de corte : } \quad |f(x_5) - f(x_4)| &= |1.505926 - 1.516460| = 0.010535
\end{aligned}$$

Como $0.010535 > 0.01$, debemos seguir iterando.

Iteración 6:

$$\begin{aligned}
\text{Gradiente en } x_5 : \nabla f(x_5) &= x_5 = 0.108864 \\
\text{Actualización del punto : } \quad x_6 &= x_5 - \eta \cdot \nabla f(x_5) = 0.108864 - 0.4 \cdot 0.108864 = 0.065318 \\
\text{Evaluación de la función en } x_6 : f(x_6) &= \frac{1}{2}(0.065318)^2 + \frac{3}{2} = 1.502133 \\
\text{Condición de corte : } \quad |f(x_6) - f(x_5)| &= |1.502133 - 1.505926| = 0.003792
\end{aligned}$$

Como $0.003792 < 0.01$, detenemos la ejecución del algoritmo.

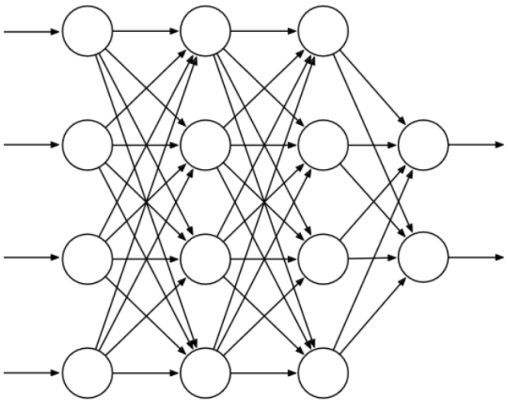
3.3 Resultado

El mínimo aproximado de la función es $x^* = 0.065318$, que es bastante cercano al verdadero mínimo en $x = 0$. Si se quisiera disminuir la tolerancia y encontrar un valor aún más cercano, se debería seguir iterando.

4 Redes Neuronales

Una red neuronal profunda no es más que una composición de funciones y está formada por varias partes:

- Una entrada, valores que ingresan a la red (que puede ser una imagen), estos valores son procesados en la red
- Funciones, por donde pasan estos valores de entradas y son transformados hasta llegar a una
- Salida que describe "lo que el sistema piensa" con respecto a los datos de entrada y luego de eso el paso final, lo que nos permite definir el objetivo
- Medir el Error o *Loss* del sistema



La red no es más que una interconexión de unidades que intentan imitar las neuronas del sistema nervioso.

Estas unidades son típicamente semilineales, computan una suma ponderada de su entrada, luego aplican una función no lineal, a esta función no lineal se la llama **Función de activación**, es la salida de la unidad y conforma una entrada de la siguiente unidad. Existe una propiedad de "aproximación universal" que dice que con la función

de activación indicada y una sola capa oculta, se puede aproximar cualquier función. Pero, la experiencia y la teoría demuestran que la aproximación de funciones complejas se hace más fácil con **abstracciones**. Siendo estas una composición jerárquica de varias abstracciones más bajas. Las capas de una red nos generan una representación más abstracta de la entrada "cruda" de datos, con cada unidad proveyendo una característica que contribuye a la representación. Entrenar las capas ocultas es una forma de generar las características apropiadas sin recaer en características "hechas a mano". Las redes neuronales aprenden las características, típicamente, con **descenso del gradiente estocástico**. Cada peso de la red es ajustado en la dirección que mejora la performance de la red, la mejora se mide por el objetivo de la red, o sea, la función de **Loss**.

4.1 Descenso del gradiente estocástico

Supongamos que queremos aproximar una función cualquiera, de la cual no conocemos su forma. En esta técnica de aproximación de funciones se sustituye el gradiente real por una aproximación de este. En lugar de calcular el gradiente real, este método lo calcula usando una única muestra. Esta estimación local del gradiente aproxima la dirección en la que se debe ajustar los pesos de la función de aproximación para reducir el valor de la función de **Loss** $L(f(x), \mathbf{y})$.Esta función de perdida puede tener diferentes formas, si lo vemos desde el punto de estadística, podríamos usar el error cuadrático medio.

La actualización tiene la misma estructura que en el descenso del gradiente clásico, es menos precisa porque se basa en una sola muestra. Pero como esta aproximación es más rápida, se pueden conseguir más actualizaciones en menos tiempo

4.2 Algoritmo de backpropagation

La idea principal es usar el gradiente para actualizar los parámetros (pesos). Para la actualización calculamos el gradiente del error con respecto a cada parámetro individual, y así obtenemos la dirección de ajuste correspondiente de cada uno de ellos.

Este algoritmo produce buenos resultados para redes poco profundas, pero no funciona tan bien para redes profundas. En generar entrenar una red con $k + 1$ capas ocultas puede resultar en una performance peor que entrenar a una red de k capas ocultas. Muchos factores entran en juego. Primero, la gran cantidad de parámetros los cuales hacen el entrenamiento más largo, pero también dificultan evitar el problema del *sobreajuste* (en el cual la red no generaliza correctamente). Segundo, el backpropagation puede fallar porque a medida que nos acercamos a la capa de entrada el gradiente disminuye rápidamente (gradiente que desaparece) o crece rápidamente(gradiente que explota), haciendo el aprendizaje inestable.

5 Material extra

Bibliografía

[1] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016.

[2] G. Montavon, G. B. Orr, K. R. Müller, *Neural Networks: Tricks of the Trade (2nd ed.)*, Springer, 2012.

[3] S. Ruder, *An Overview of Gradient Descent Optimization Algorithms*, arXiv preprint arXiv:1609.04747, 2016.

[4] J. Simon, H. Dao, et al., *Gradient Descent Finds Global Minima of Deep Neural Networks*, arXiv preprint arXiv:1901.00303, 2019.

Cursos

[1] *Deep Learning Specialization*, Coursera, DeepLearning.AI (Andrew Ng), disponible en: `https://www.coursera.org/specializations/deep-learning`.

[2] *CS231n: Convolutional Neural Networks for Visual Recognition*, Stanford University, disponible en: `http://cs231n.stanford.edu/`.

[3] *Practical Deep Learning for Coders*, FastAI, disponible en: `https://course.fast.ai/`.