404 MÉTODO DE DESCENSO DE GRADIENTE ESTOCÁSTICO

October 17, 2025

1 Introducción

El Descenso de Gradiente Estocástico (SGD) es un algoritmo de optimización iterativo diseñado para encontrar el mínimo de una función. Su característica principal, y lo que lo distingue del descenso de gradiente tradicional, es que no calcula el gradiente "exacto" de la función en cada paso. En cambio, utiliza una aproximación ruidosa (estocástica) del gradiente.

En el contexto del aprendizaje automático, este "ruido" proviene de calcular el gradiente usando una única muestra o un pequeño grupo de datos. En un problema de optimización más general, podemos simular este comportamiento introduciendo un término de ruido aleatorio directamente en el cálculo del gradiente.

2 Formulación matemática

Consideremos una función $f(\mathbf{x})$ que queremos minimizar. El objetivo es encontrar los valores $\mathbf{x}^*$ que corresponde al mínimo de la función. El algoritmo de descenso de gradiente tradicional actualizaría los parámetros de la siguiente forma:

$$\mathbf{x}_{k+1} = x_k - \alpha \nabla f(\mathbf{x}_k)$$

donde $\nabla f(\mathbf{x}_k)$ es el gradiente verdadero de la función evaluado en el punto actual $\mathbf{x}_k$.

En el descenso de gradiente estocástico, la actualización se modifica para usar una estimación del gradiente, que llamaremos $\hat{g}(\theta_k)$. Esta estimación es el gradiente verdadero más un término de ruido:

$$\hat{g}(\mathbf{x}_k) = \nabla f(\mathbf{x}_k) + \epsilon$$

donde ϵ es un vector de ruido aleatorio, usualmente con media cero.

La regla de actualización del descenso de gradiente estocástico se convierte en:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \hat{g}(\mathbf{x}_k)$$

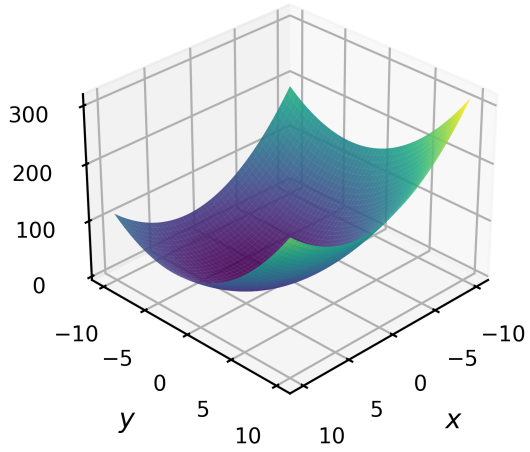
Esta naturaleza ruidosa hace que la trayectoria de convergencia no sea suave, en su lugar, "salta" de forma errática hacia el mínimo. Este comportamiento puede ser ventajoso para evitar quedar atrapado en mínimos locales poco profundos.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
resolucion = 600
```

[2]:
```
# ------------------------------------
# Encontrar el mínimo de la función:
# f(x, y) = (x - 2) ** 2 + (y + 3) ** 2
# ------------------------------------

# Definimos la función objetivo y su gradiente
def paraboloide(x):
    return (x[0] - 2) ** 2 + (x[1] + 3) ** 2

def g_paraboloide(x):
    dx = 2 * (x[0] - 2)
    dy = 2 * (x[1] + 3)
    return np.array([dx, dy])
```

[3]:
```
# Graficar la función
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
Z = paraboloide([X, Y])

fig = plt.figure(figsize = (6, 3), dpi = resolucion)
ax = fig.add_subplot(111, projection = "3d")
ax.plot_surface(X, Y, Z, cmap = "viridis", alpha = 0.9)
ax.view_init(elev = 30, azim = 45)
ax.set_title("$f(x, y) = (x - 2)^2 + (y + 3)^2$")
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
ax.tick_params(labelsize = 8)
plt.show()
```

$$f(x, y) = (x - 2)^2 + (y + 3)^2$$



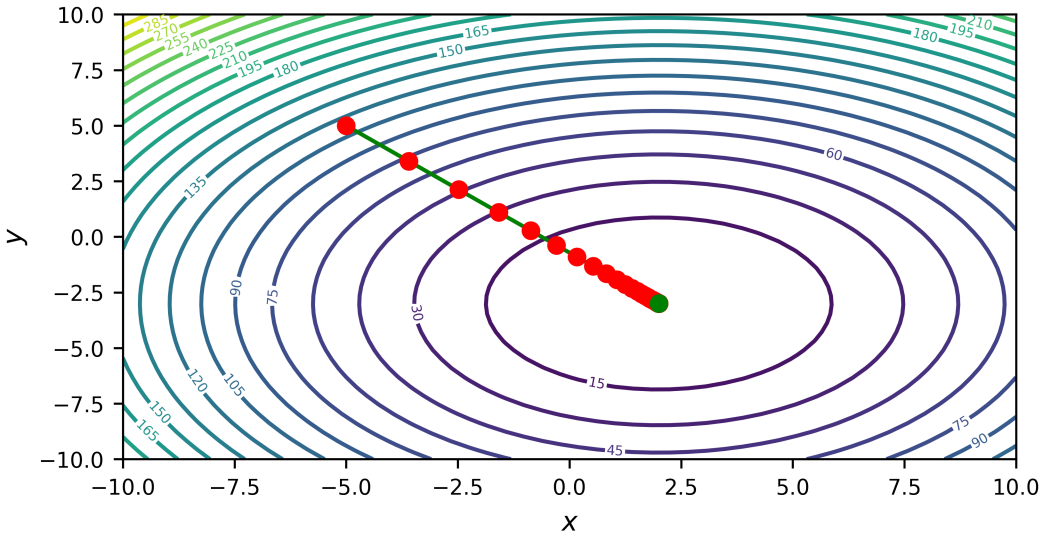
[4]:
```python
def dg_estandar(f, grad_f, x_actual, alpha, max_iter = 1000, tol = 1e-6):
    x_historial = np.array([x_actual])
    for i in range(max_iter):
        x_nuevo = x_actual - alpha * grad_f(x_actual)
        x_historial = np.vstack((x_historial, x_nuevo))
        x_actual = x_nuevo
    f_optimo = f(x_nuevo)
    iteraciones = len(x_historial)
    return x_nuevo, f_optimo, iteraciones, x_historial
```

[5]:
```python
# Ajustes de parámetros del algoritmo
alpha = 0.1
x_actual = np.array([-5, 5])

# Ejecución del algoritmo y resultados
resultado = dg_estandar(paraboloide, g_paraboloide, x_actual, alpha)

print("x =", resultado[0])
print("f(x) =", resultado[1])
print("Iteraciones =", resultado[2])
```

```
x = [ 2. -3.]
f(x) = 9.860761315262648e-31
Iteraciones = 1001
```

[6]:
```python
# Graficar la función
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
```

```
Z = paraboloide([X, Y])

plt.figure(figsize=(6, 3), dpi = resolucion)
contour = plt.contour(X, Y, Z, levels = 20, cmap = "viridis")
plt.clabel(contour, inline = True, fontsize = 5)
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.tick_params(labelsize = 8)

# Puntos de trayectoria
plt.plot(resultado[3][:, 0], resultado[3][:, 1], "g-")
plt.plot(resultado[3][:, 0], resultado[3][:, 1], "ro")
plt.plot(2, -3, 'go')
plt.show()
```

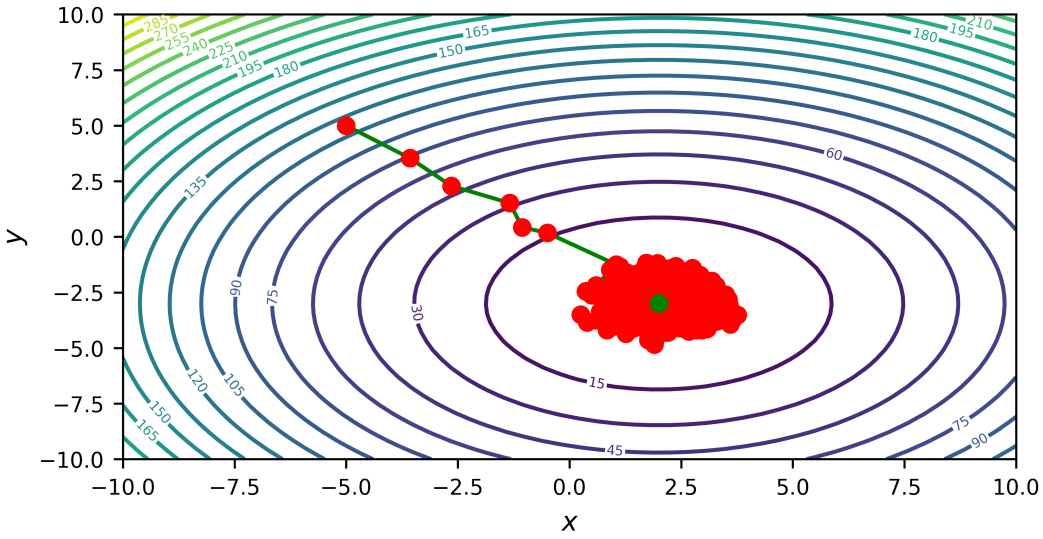


[7]:
```
def dg_estocastico(f, grad_f, x_actual, alpha, max_iter = 1000, tol = 1e-6):
    x_historial = np.array([x_actual])
    for i in range(max_iter):
        gradiente = grad_f(x_actual)
        ruido = np.random.normal(0, 4, size = gradiente.shape[0])
        gradiente_estocastico = gradiente + ruido
        x_nuevo = x_actual - alpha * gradiente_estocastico
        x_historial = np.vstack((x_historial, x_nuevo))
        x_actual = x_nuevo
    f_optimo = f(x_nuevo)
    iteraciones = len(x_historial)
    return x_nuevo, f_optimo, iteraciones, x_historial
```

[8]:
```
# Ajustes de parámetros del algoritmo
alpha = 0.1
x_actual = np.array([-5, 5])

# Ejecución del algoritmo y resultados
resultado = dg_estocastico(paraboloide, g_paraboloide, x_actual, alpha)

print("x =", resultado[0])
print("f(x) =", resultado[1])
print("Iteraciones =", resultado[2])
```

```
x = [ 2.55609125 -2.99064197]
f(x) = 0.3093250495733592
Iteraciones = 1001
```

[9]:
```
# Graficar la función
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
Z = paraboloide([X, Y])

plt.figure(figsize=(6, 3), dpi = resolucion)
contour = plt.contour(X, Y, Z, levels = 20, cmap = "viridis")
plt.clabel(contour, inline = True, fontsize = 5)
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.tick_params(labelsize = 8)

# Puntos de trayectoria
plt.plot(resultado[3][:, 0], resultado[3][:, 1], "g-")
plt.plot(resultado[3][:, 0], resultado[3][:, 1], "ro")
plt.plot(2, -3, 'go')
plt.show()
```



Juan F. Olivares Pacheco (`jfolivar@uda.cl`)

Universidad de Atacama, Facultad de Ingeniería, Departamento de Matemática