معرفی الگوریتم‌های گرادیان نزولی

گرادیان نزولی یکی از متداول‌ترین روش‌های بهینه‌سازی در یادگیری ماشین است. هدف این روش‌ها، یافتن مقادیر بهینه برای پارامترهای مدل با کمینه کردن تابع هزینه است.

انواع الگوریتم‌های گرادیان نزولی

1. Batch Gradient Descent: در این روش، از کل مجموعه داده برای محاسبه‌ی گرادیان و به‌روزرسانی پارامترها استفاده می‌شود. این روش پایدار است، اما در صورت بزرگ بودن مجموعه داده، بسیار زمان‌بر خواهد بود.
   این روش برای مجموعه داده‌های کوچک یا زمانی که به همگرایی دقیق‌تر نیاز داریم، ایده‌آل است. با این حال، برای داده‌های بزرگ می‌توان زمان‌بر و از نظر مصرف حافظه مشکل‌ساز باشد.
2. Stochastic Gradient Descent (SGD): در این روش، در هر تکرار، تنها از یک نمونه از داده‌ها برای محاسبه‌ی گرادیان استفاده می‌شود. این باعث می‌شود که بهینه‌سازی سریع‌تر انجام شود، اما ممکن است دارای نوسانات باشد.
   این نوسانات به الگوریتم کمک می‌کند تا از نقاط بهینه نسبی عبور کرده و بهینه‌ای کلی را پیدا کند، اما ممکن است به اندازه‌های دسته‌ای پایدار نباشد. برای داده های بزرگ مناسب است، این روش نیازی به پردازش کل داده‌ها نیست و به این ترتیب سرعت همگرایی آن افزایش می‌یابدو از آنجا که نوسان دارد، به عنوان یک روش برای نزدیک شدن به جواب‌های بهینه در داده‌های پیچیده استفاده می‌شود.
3. Mini-Batch Gradient Descent: این روش ترکیبی از دو روش قبل است؛ به‌طوری که در هر تکرار از یک زیرمجموعه کوچک (Mini-Batch) از داده‌ها برای محاسبه‌ی گرادیان استفاده می‌شود. این روش تعادل مناسبی بین سرعت و دقت بهینه‌سازی ایجاد می‌کند. در داده های بزرگ و زمانی که به تعادل بین سرعت و پایداری نیاز دارید، مینی بچ بهترین انتخاب است. این روش در شبکه های عصبی و مدل های پیچیده با داده های حجیم به کار می رود.

پیاده‌سازی هر یک از روش‌ها

برای پیاده‌سازی، از مجموعه داده **"California Housing"** که از کتابخانه scikit-learn قابل بارگذاری است استفاده شده. این مجموعه اطلاعات مربوط به قیمت خانه ها در مناطق مختلف کالیفرنیا را شامل می شود. ویژگی‌های مانند درآمد، تعداد اتاق‌ها، و جمعیت منطقه را دارد و هدف آن پیش‌بینی قیمت متوسط خانه‌ها در هر منطقه است.

بارگذاری داده‌ها و آماده‌سازی اولیه

```
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler
import numpy as np
data = fetch_california_housing()
```

```
X = data.data  # ویژگی‌ها
y = data.target.reshape(-1, 1)  # تبدیل هدف به ماتریس
# استانداردسازی داده‌ها
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_b = np.c_[np.ones((X_scaled.shape[0], 1)), X_scaled]  # افزودن بایاس
m = X_b.shape[0]  # تعداد نمونه‌هاy = data.target.reshape(1 ,1-)
# افزودن ستون بایاس
X_b = np.c_[np.ones((X.shape[0], 1)), X]
```

Batch Gradient Descent

```
# تنظیمات اولیه
learning_rate = 0.01
n_iterations = 1000
np.random.seed(42)
theta = np.random.randn(X_b.shape[1], 1)
theta_path_bgd = []
for iteration in range(n_iterations):
    gradients = 2 / m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - learning_rate * gradients
    theta_path_bgd.append(theta.copy())
print("Theta (Batch Gradient Descent):", theta)
theta_path_bgd = np.array(theta_path_bgd)
plt.plot(theta_path_bgd[:, 0], theta_path_bgd[:, 1], "b-", label="Batch Gradient Descent")
plt.xlabel(r"$\theta_0$")
plt.ylabel(r"$\theta_1$", rotation=0)
plt.legend()
plt.title("Path of Theta in Batch Gradient Descent")
plt.show()
```

Theta (Batch Gradient Descent): [[2.06855817e+00]

[7.40904659e-01]

[1.53569295e-01]

[-1.68169024e-03]

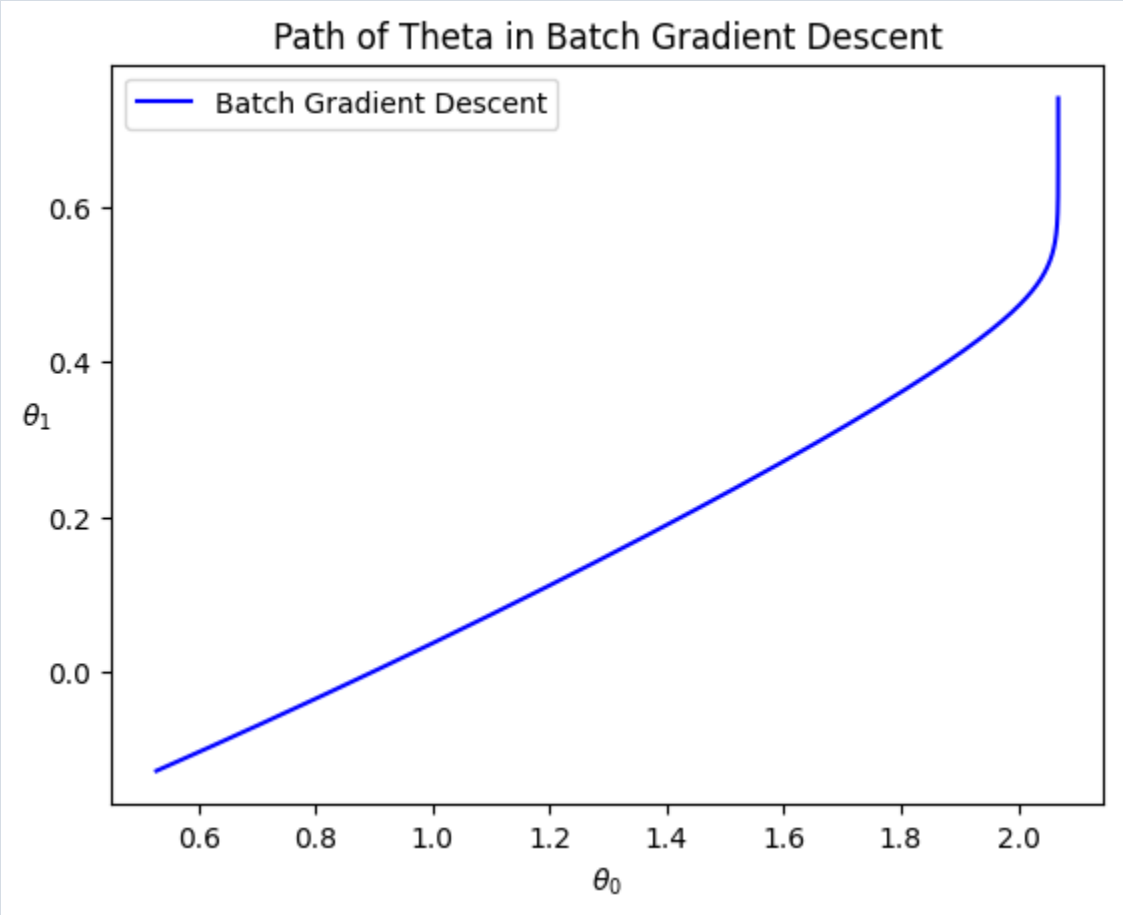
[4.51982176e-02]

[9.31456754e-03]

[-4.03799788e-02]

[-7.29380857e-01]

[-6.85055444e-01]]



نزول گرادیان تصادفی

```
n_epochs = 50
t0, t1 = 5, 50
def learning_schedule(t):
   return t0 / (t + t1)
np.random.seed(42)
theta = np.random.randn(X_b.shape[1], 1)
theta_path_sgd = []
for epoch in range(n_epochs):
   for i in range(m):
      random_index = np.random.randint(m)
      xi = X_b[random_index:random_index + 1]
      yi = y[random_index:random_index + 1]
      gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
      eta = learning_schedule(epoch * m + i)
      theta = theta - eta * gradients
      theta_path_sgd.append(theta.copy())
print("Theta (Stochastic Gradient Descent):", theta)
theta_path_sgd = np.array(theta_path_sgd)
plt.plot(theta_path_sgd[:, 0], theta_path_sgd[:, 1], "r-", label="Stochastic Gradient Descent")
plt.xlabel(r"$\theta_0$")
plt.ylabel(r"$\theta_1$", rotation=0)
plt.legend()
plt.title("Path of Theta in Stochastic Gradient Descent")
plt.show()
```

Theta (Stochastic Gradient Descent): [[2.07074491]

[0.86206258]

[0.09616402]

[-0.3783594]

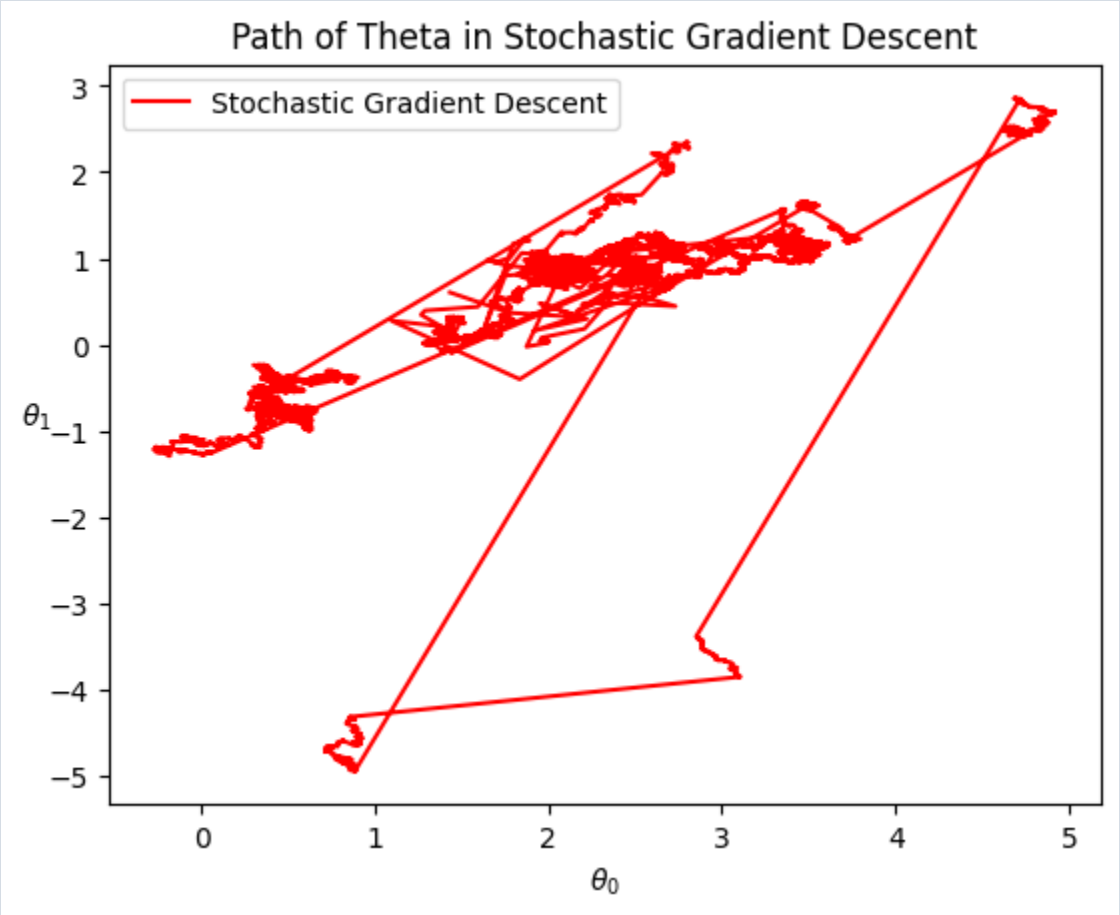
[0.42297977]

[-0.01251968]

[-0.03977074]

[-1.03621604]

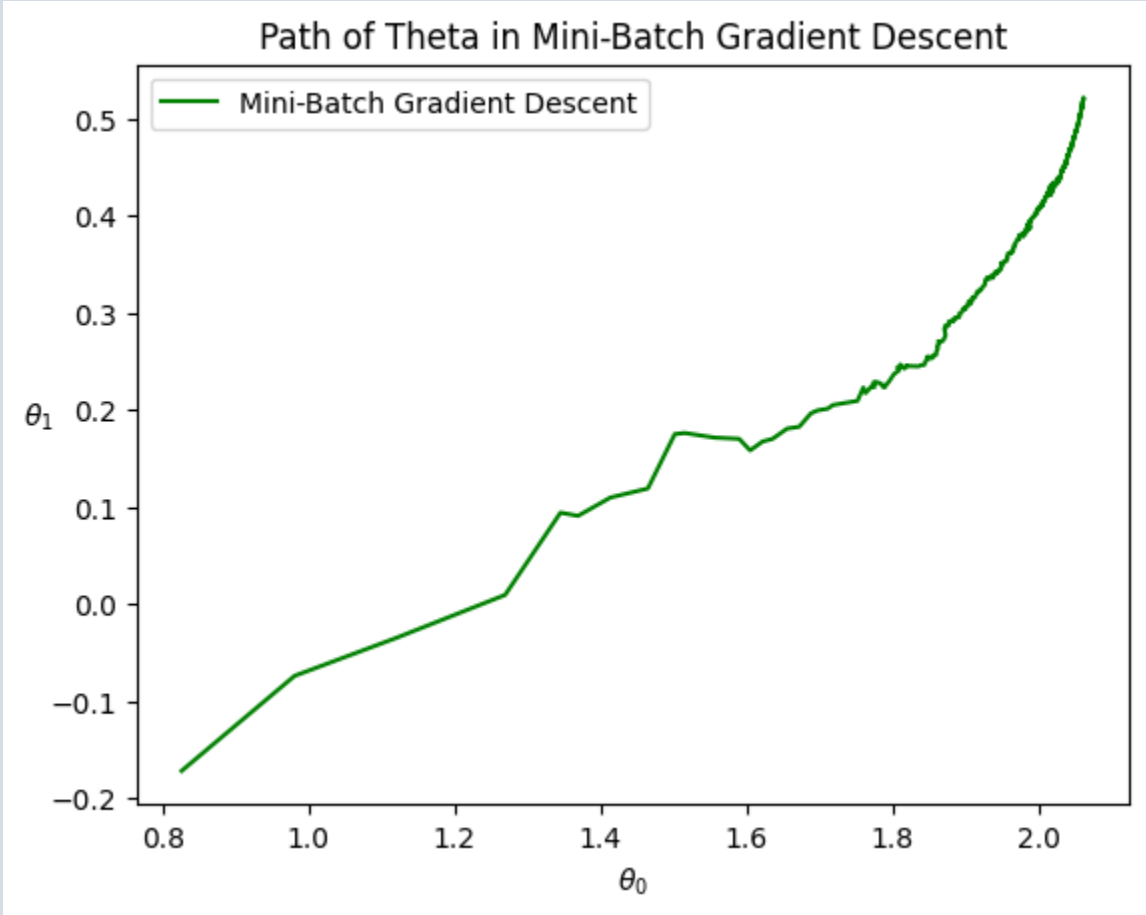
[-1.01753973]]



مینی دسته ای نزول گرادیان

```
mini_batch_size = 20
np.random.seed(42)
theta = np.random.randn(X_b.shape[1], 1)
theta_path_mgd = []
for epoch in range(n_epochs):
    shuffled_indices = np.random.permutation(m)
    X_b_shuffled = X_b[shuffled_indices]
    y_shuffled = y[shuffled_indices]
    for i in range(0, m, mini_batch_size):
        xi = X_b_shuffled[i:i + mini_batch_size]
        yi = y_shuffled[i:i + mini_batch_size]
        gradients = 2 / mini_batch_size * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
        theta_path_mgd.append(theta.copy())
print("Theta (Mini-Batch Gradient Descent):", theta)
theta_path_mgd = np.array(theta_path_mgd)
plt.plot(theta_path_mgd[:, 0], theta_path_mgd[:, 1], "g-", label="Mini-Batch Gradient Descent")
plt.xlabel(r"$\theta_0$")
plt.ylabel(r"$\theta_1$", rotation=0)
plt.legend()
plt.title("Path of Theta in Mini-Batch Gradient Descent")
plt.show()
```

```
Theta (Mini-Batch Gradient Descent): [[ 2.06142523]
 [ 0.52134207]
 [ 0.22717854]
 [ 0.61993608]
 [-0.56883761]
 [ 0.02041465]
 [ 0.05445352]
 [-0.30341598]
 [-0.22037649]]
```



مقایسه‌ی الگوریتم‌ها

الگوریتم	سرعت همگرایی	پایداری	حافظه مورد نیاز
نزول گرادیان دسته ای	کند	پایدار	زیاد
نزول گرادیان تصادفی	سریع	دارای نوسان	کم
مینی دسته ای نزول گرادیان	متوسط	نسبتا پایدار	متوسط

- نـزول رنجیـان دسـته‌ای (Batch Gradient Descent)بـه دلیـل پـردازش تمـام داده‌هـا در هـر گـام، پایـدار اسـت و بـه حافظه‌های زیادی نیاز دارد.
- نـزول رنجیـان تصـادفی (Stochastic Gradient Descent)سـریع اسـت، امـا بـه دلیـل پـردازش یـک داده در هـر گـام دارای نوسان است و حافظه‌ی کمی نیاز دارد.
- نـزول پیشـرفتیان مینی‌دسـته‌ای (Mini-Batch Gradient Descent)بـین دو روش قبلـی و بـه صـورت متوسـط در هـر سه بخش عمل می‌کند.

تکنیک های جلوگیری از بهینه نسبی

برای جلوگیری از گرفتار شدن در بهینه‌های نسبی، تکنیک‌های زیر وجود دارد:

- کاهش نرخ به مرور زمان: استفاده از نرخ‌هایی که با افزایش تعداد تکرارها کاهش می‌یابند، می‌توانند از بهینه‌های محلی کمک کند. معمولاً از تکنیک‌هایی مانند کاهش سرعت نمایش (Exponential Decay) یا کاهش با دوره‌های خاص (Step Decay) استفاده می‌شود. در این روش، میزان به کاهش و تعداد تکرارها تنظیم می شود. از مزایای این تکنیک می‌توان به جلوگیری از نوسان زیاد در نزدیکی بهینه و همگرایی پایدارتر اشاره کرد.
- افزودن مومنتوم (Momentum): مومنتوم با ایجاد بخشی از تغییرات ذخیره شده قبلی در فرآیند بهینه سازی در عبور از نقاط بهینه نسبی کمک می کند. با به خاطر سپردن مقداری از جهت حرکت قبلی، از نوسانات زیاد در نزدیکی بهینه‌های محلی جلوگیری و الگوریتم به سمت دامنه‌های بهتر هدایت می‌شود. در این روش، یک اصطلاح جدید به فرمول به‌روزرسانی اضافه می‌شود که شامل جهت حرکت قبلی است. از مزایای آن می توان به افزایش سرعت همگرایی و جلوگیری از نوسانات بیش از در جهت های مختلف اشاره کرد.
- استفاده از الگوریتم‌های پیشرفته‌تر (مانند آدام): الگوریتم‌های بهینه‌سازی پیشرفته‌تر مانند Adam و RMSprop می‌توانند به حل مشکلات بهینه‌سازی کمک کنند. Adam و RMSprop دو الگوریتم پیشرفته برای بهینه‌سازی هستند که هر دو از تکنیک‌های تنظیم خودکار استفاده می‌کنند.
- Adam (Adaptive Moment Estimation): این الگوریتم ترکیبی از مومنتوم و RMSprop است و از دو اصطلاح اول و دوم (میانگین متحرک) برای تنظیم نرخ استفاده می‌شود.
- RMSprop (Root Mean Square Propagation): این الگوریتم از مقادیر متحرک مربعات برای تنظیم میزان استفاده می‌کند. به این ترتیب، سرعت همگرایی در جهات مختلف بهتر تنظیم می شود.

این الگوریتم‌ها به بهینه‌سازی در هر جهت کمک می‌کنند، نوسانات را کاهش می‌دهند و معمولاً به بهترین نتایج دقیق‌تری می‌رسند.

نتیجه گیری

در این گزارش انواع الگوریتم‌های نزولی را مقایسه کردیم. هر کدام از این الگوریتم‌ها و معایب خاص خود را دارند و بسته به نیاز و منابع سیستم می‌توانند مورد استفاده قرار گیرند. تکنیک‌های بهینه‌سازی مانند استفاده از ارزیابی‌ها و مومنتوم به بهبود عملکرد الگوریتم‌ها و جلوگیری از گرفتار شدن در بهینه‌ای نسبی کمک می‌کنند.

در حالت کلی، بهترین روش بستگی به مجموعه داده و نیاز پروژه دارد:

اگر دقت بالا و مجموعه کوچکی داشته باشید، Batch Gradient Descentمناسب است.

اگر با داده‌های بزرگ و نیاز به اجرای سریع، SGDمناسب‌تر است، اما برای همگرایی بهتر می‌توان نوسانات را با استفاده از تکنیک‌های کاهش‌دهنده (مانند افزایش نرخ‌گذاری) کنترل کرد.

Mini-Batch Gradient Descent بهترین انتخاب برای بسیاری از شرایط عملی است، چون عملکرد بین سرعت و دقت ایجاد می‌کند و نتایج نسبتاً پایدار و دقیق‌تری دارد.