Gradient Descent Algorithm

Opis i implementacja

Wstęp

Metoda gradientu prostego (ang. *Gradient Descent*) jest klasycznym algorytmem iteracyjnym służącym do znajdowania minimum lokalnego funkcji wielu zmiennych rzeczywistych. Idea metody polega na przemieszczaniu się w kolejnych iteracjach w kierunku przeciwnym do gradientu funkcji, który wskazuje kierunek najszybszego wzrostu. Sama polska nazwa *gradient prosty* oznacza bazowy, klasyczny gradient i odróżnia go od bardziej złożonych metod[1]. Przy czym czasami w literaturze pojawia się bezpośrednie tłumaczenie: metoda spadku gradientu[2].

Gradient prosty jest jedną z podstawowych metod optymalizacji. Algorytm ten oraz jego rozszerzenie w postaci stochastycznego spadku gradientu znajdują szerokie zastosowanie m.in. w uczeniu sieci neuronowych.

1. Opis algorytmu

Rozważmy problem znalezienia minimum lokalnego funkcji

$$f : D \mapsto \mathbb{R}, \quad D \subset \mathbb{R}^n,$$

która jest ciągła i różniczkowalna w D. Celem algorytmu gradientu prostego jest więc znalezienie punktu $x_k \in D$, w którym funkcja osiąga minimum lokalne. Metoda opiera się na iteracyjnym przemieszczaniu się w kierunku przeciwnym do gradientu funkcji, który z definicji wskazuje kierunek największego wzrostu wartości funkcji.

Podstawowy schemat iteracyjny metody gradientu prostego jest następujący:

1. Wybierz punkt startowy $x_0 \in D$.
2. Oblicz gradient funkcji $\nabla f(x_k)$ w aktualnym punkcie x_k.
3. Wyznacz nowy punkt:
$$x_{k+1} = x_k - \alpha_k \nabla f(x_k),$$
gdzie $\alpha_k > 0$ może być stałe lub dobierane indywidualnie w każdej iteracji np. metodą maksymalnego spadku.
4. Powtarzaj kroki 2-3 aż do spełnienia kryterium stopu, np.:
$$\|\nabla f(x_k)\| < \varepsilon,$$
gdzie $\varepsilon > 0$ jest ustaloną dokładnością.

[1] Wikipedia - Metoda gradientu prostego
[2] Np. w Quo vAIdis A. Dragana.

2. Implementacja algorytmu

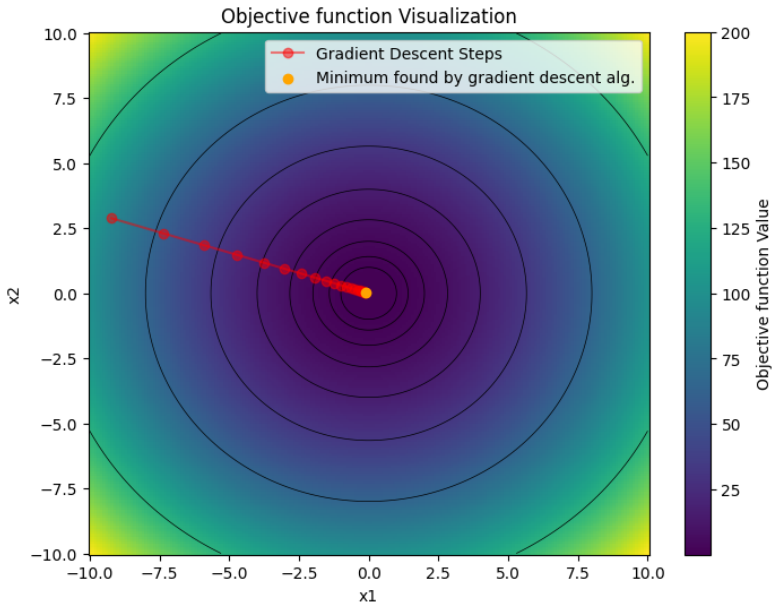
Algorytm zaimplementowałem w języku Python z wykorzystaniem biblioteki `autograd` do automatycznego wyznaczania gradientu oraz `numpy` do operacji wektorowych.

```
def descent_gradient_alg(
        function: callable, start: list,
        steps: int, learning_rate: float) -> list:

    trace = [start.copy()]
    grad_fun = grad(function)
    point = np.array(start, dtype=float)

    for _ in range(steps):
        grad_vec = grad_fun(point)

        if vec_length(grad_vec) < 1e-3:
            break

        point = point - learning_rate * grad_vec
        trace.append(point.copy())

    return trace
```

Funkcja przyjmuje argumenty:

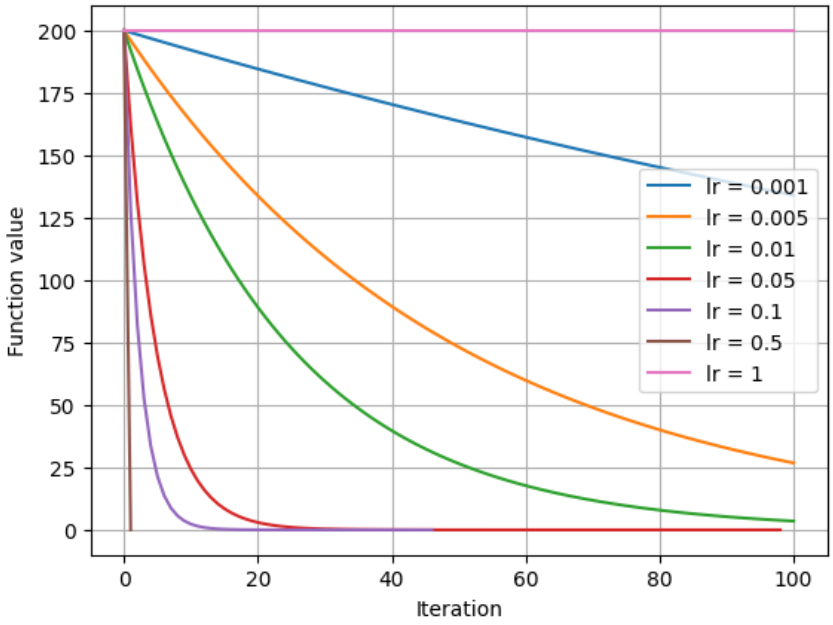
- `function` - minimalizowana funkcja f
- `start` - punkt startowy $[x_0, x_1, \ldots, x_n]$
- `steps` - maksymalna liczba iteracji (dodatkowe kryterium stopu)
- `learning_rate` - stały parametr skoku α_k

Po wykonaniu algorytmu zwracana jest ścieżka punktów odwiedzonych, gdzie ostatni punkt jest docelowo minimum lokalnym. Trajektorię można zwizualizować za pomocą biblioteki `matplotlib`, co przedstawiono na przykładzie minimalizacji paraboloidy na rysunku poniżej.



3. Wpływ parametru kroku na zbieżność gradientu

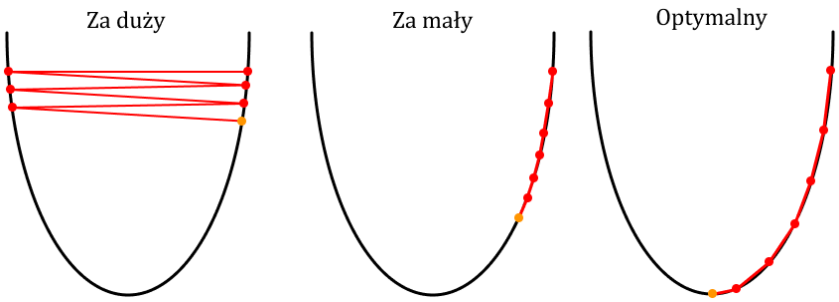
Jeżeli w algorytmie gradientu prostego stosujemy stały parametr kroku $\alpha_k = \alpha$, to jego wartość ma kluczowe znaczenie dla zbieżności algorytmu. Poniższy wykres przedstawia wpływ różnych wartości α (`learning_rate`) na skuteczność zbieżności do minimum lokalnego funkcji kwadratowej $f(x) = x_0^2 + x_1^2$, zaczynając od punktu startowego $[10; 10]$.



Na podstawie powyższego wykresu można zauważyć następujące zależności:

- Zbyt mała wartość, np. $\alpha = 0,01$, powoduje bardzo powolną zbieżność, co wymaga wielu kroków przed osiągnięciem minimum.
- Zbyt duża wartość, np. $\alpha = 1,0$, może powodować niestabilność i oscylacje wokół minimum.
- Optymalna wartość, np. $\alpha = 0,1$, pozwala na szybką i stabilną zbieżność do minimum lokalnego.
- Może również wystąpić szczególny przypadek, np. dla $\alpha = 0,5$, w którym algorytm trafia w minimum już po jednym kroku. Wynika to jednak ze specyficznego wyboru punktu startowego i dla innego punktu początkowego prawdopodobnie nie będzie zachodzić.

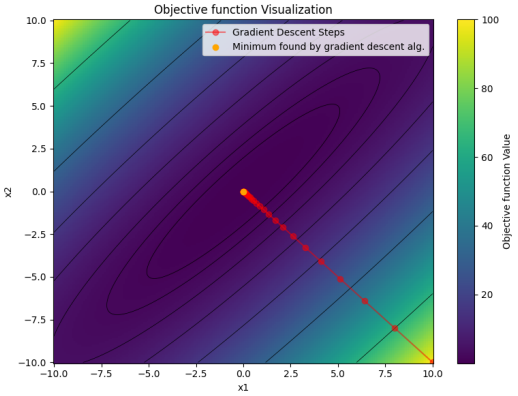
Powyższe wnioski o doborze parametru kroku można łatwo zilustrować:



4. Punkt startowy a zachowanie algorytmu

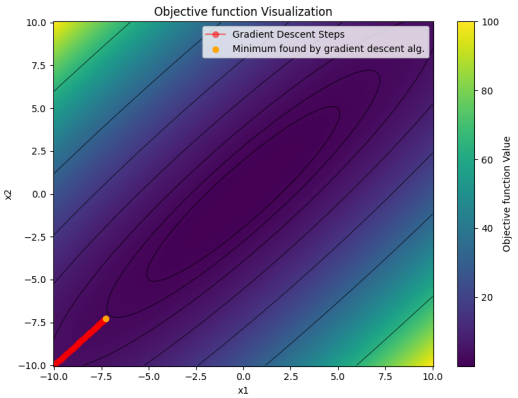
Wybór punktu startowego jest istotny, ponieważ w przypadku funkcji posiadających wiele minimów lokalnych algorytm może zbiegać do różnych rozwiązań. Dodatkowo punkt startowy wpływa na szybkość zbieżności metody oraz stabilność procesu optymalizacji. Dlatego w praktyce warto testować kilka różnych punktów początkowych.

Weźmy funkcję Matyasa: $f(x) = 0.26(x_0^2 + x_1^2) - 0.48x_0x_1$, która ma globalne minimum w punkcie $[0; 0]$, i wykonajmy algorytm z parametrami: `steps=40, learning_rate=0.2` oraz różnymi punktami startowymi. W zależności od wyboru punktów startowych otrzymujemy różne trajektorie zbieżności, co widać na poniższych wykresach.



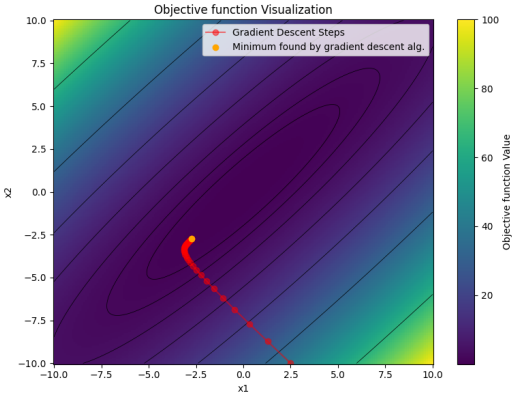
Punkt startowy $[10, -10]$.

Dla tego punktu startowego algorytm wykazuje stabilną i monotoniczną zbieżność w kierunku minimum globalnego. Trajektoria optymalizacji jest płynna, a wartość funkcji celu konsekwentnie maleje z każdą iteracją, co potwierdza poprawny dobór parametrów.



Punkt startowy $[-10, -10]$.

Algorytm szybko osiąga dolinę funkcji, w której gradient jest niewielki. W rezultacie kolejne kroki stają się coraz mniejsze, a proces optymalizacji ulega znacznemu spowolnieniu - algorytm praktycznie zatrzymuje się, nie osiągając dokładnego minimum.



Punkt startowy $[2{,}5, -10]$.

Algorytm początkowo szybko redukuje wartość funkcji celu, skutecznie zbliżając się do obszaru minimum. Po osiągnięciu doliny dalsza poprawa staje się jednak bardzo powolna ze względu na niewielkie wartości gradientu. Obserwacja ta sugeruje, że w tego typu sytuacjach korzystniejsze może być zastosowanie innych metod np. metody Newtona.

5. Metoda gradientu prostego z momentem

ang. Gradient Descent with Momentum

Jak zaobserwowaliśmy w poprzednich eksperymentach, algorytm gradientu prostego ma swoje ograniczenia - zatrzymuje się w dolinach funkcji przy niewielkim gradiencie oraz potrafi utknąć w niekoniecznie najlepszym minimum lokalnym. Jedną z metod zwiększenia jego skuteczności jest uwzględnienie poprzednich gradientów, które budują *moment* pozwalający pokonywać płaskie obszary i płytkie minima lokalne.

Moment w punkcie x_k wyznaczany jest jako wykładnicza średnia krocząca gradientów:

$$v_k = \beta\, v_{k-1} + (1 - \beta)\, \nabla f(x_k),$$

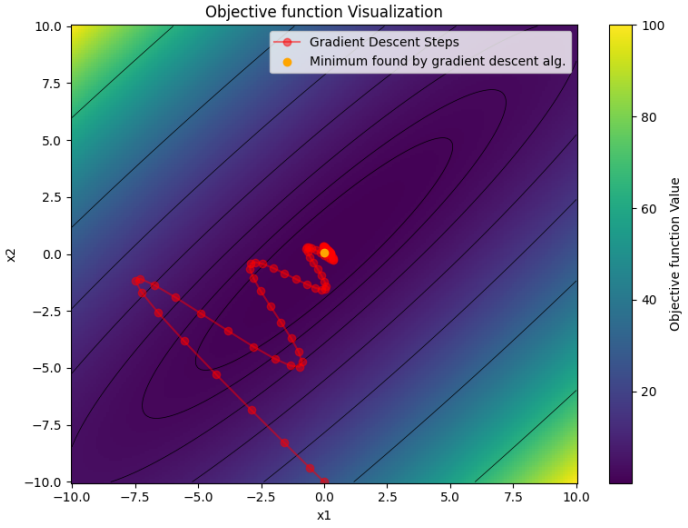
gdzie $\beta \in [0, 1)$ jest współczynnikiem momentu określającym, jak silnie uwzględniana jest historia wcześniejszych gradientów. Dla $\beta = 0$ metoda redukuje się do klasycznego gradientu prostego. Typowa wartość β wynosi 0,9.

Aktualizacja punktu odbywa się z użyciem zakumulowanego momentu zamiast samego gradientu:

$$x_{k+1} = x_k - \alpha\, v_k,$$

gdzie $\alpha > 0$ jest parametrem kroku. Dzięki temu w obszarach, gdzie gradient konsekwentnie wskazuje ten sam kierunek, moment narasta i algorytm wykonuje większe kroki. Natomiast w kierunkach, w których gradient oscyluje, składowe momentu wzajemnie się znoszą, co stabilizuje trajektorię.

Jak widzimy poniżej na przykładzie wcześniej wspomnianej funkcji Matyasa, punkt x_k toczy się po funkcji niczym kula - dzięki zakumulowanemu momentowi algorytm nie zatrzymuje się w dolinie o niewielkim gradiencie, lecz kontynuuje ruch w kierunku minimum globalnego. W porównaniu z klasycznym gradientem prostym, który w tej samej konfiguracji ulegał spowolnieniu, metoda z momentem osiąga dokładniejsze rozwiązanie.



6. Podsumowanie

W niniejszym raporcie przedstawiono algorytm gradientu prostego - jedną z podstawowych metod optymalizacji iteracyjnej. Omówiono jego schemat działania, implementację w języku Python oraz przeprowadzono eksperymenty badające wpływ kluczowych parametrów na zbieżność metody.

Przeprowadzone eksperymenty wykazały, że dobór parametru kroku α ma decydujący wpływ na zachowanie algorytmu. Zbyt mała wartość prowadzi do bardzo powolnej zbieżności, natomiast zbyt duża - do oscylacji i niestabilności. Optymalna wartość parametru kroku zapewnia szybkie i stabilne osiągnięcie minimum.

Analiza wpływu punktu startowego na przykładzie funkcji Matyasa potwierdziła, że wybór punktu początkowego istotnie wpływa na trajektorię i efektywność optymalizacji. W szczególności zaobserwowano, że w obszarach o niewielkim gradiencie algorytm może ulegać znacznemu spowolnieniu, co stanowi ograniczenie metody ze stałym krokiem.

Przedstawiono również rozszerzenie algorytmu o mechanizm momentu, który poprzez akumulację wcześniejszych gradientów pozwala pokonywać płaskie obszary oraz płytkie minima lokalne. Eksperymenty na funkcji Matyasa potwierdziły, że metoda z momentem osiąga dokładniejsze rozwiązanie w porównaniu z klasycznym gradientem prostym.

Podsumowując, metoda gradientu prostego jest skutecznym narzędziem optymalizacji dla odpowiednio dobranych parametrów, lecz wymaga starannego ich doboru zarówno parametru kroku, jak i punktu startowego. Wprowadzenie momentu znacząco poprawia zachowanie algorytmu w trudnych obszarach przestrzeni funkcji, stanowiąc proste, a zarazem skuteczne rozszerzenie klasycznej metody.