Krzysztof Terlecki 318454 ,

Politechnika Warszawska

Wprowadzenie do sztucznej inteligencji - ćwiczenie 1

17 marca 2023

1. Wstęp

Ćwiczenie obejmuje implementacje i analizy algorytmów związanych z problemem plecakowym i najszybszym spadkiem gradientu. Do tego raportu zawierającego zestawienia wyników doświadczeń na algorytmach został załączony projekt wraz z kodem źródłowym w pythonie. Zawiera on moduły:

— knapsackProblem- zawiera implementtacje rozwiązań problemu plecakowego.

— findMinimum- zawiera trzy algorytmy minimalizujące funkcje metodą najszybszego spadku gradientu oraz trzy skrypty pomocnicze: example 1 i 2 to przykłady z instrukcji ćwiczeniowej, zmodyfikowane tak aby działały w moim projekcie; example4 jest przykładem z repozytorium cec2017functions z githuba.

— cec2017functions- sklonowany z githuba zbiór implementacji funkcji cec2017 w pythonie.

Ponadto załączony jest Jupyter Notebook "Doświadczenia Lab1" zawierający omawiane w tym raporcie doświadczenia, bazujące na algorytmach z powyższych modułów.

2. Problem plecakowy

2.1. Zadania

1. Rozwiązać problem przy użyciu heurystyki: do plecaka pakujemy przedmioty według kolejności wynikającej ze stosunku p/w. Uwaga: heurystyka to nie funkcja heurystyczna. Nie używamy tu jeszcze funkcji heurystycznej i algorytmu A*.
2. Znaleźć rozwiązanie optymalne przez przegląd wyczerpujący.

2.2. Algorytmy

Pozwoliłem sobie rozszerzyć zadanie i zaimplementowałem 4 algorytmy:

— algorithm1- algotytm stosujący prostą heurystykę: sortuje przedmioty względem parametru wartość/waga i pakuje od najkorzystniejszego do wyczerpania miejsca. Przewidujemy, że takie podejście będzie niezwykle szybkie, ale nie zawsze optymalne.

— algorithm2- jest to bardziej złożony algorytm heurystyczny i stanowi rozwinięcie poprzedniego. Heurystyka jest analogiczna, ale w momencie gdy plecak jest pełen, algorytm zapamiętuje jego stan i szuka lepszego rozwiązania. Robi to w następujący sposób: wypakowuje najmniej korzystny przedmiot i pakuje kolejne, aż do zapełnienia miejsca. Algorytm ten ma szereg zalet w stosunku do poprzednika. Po pierwsze w nieograniczonym czasie przejdzie przez wszystkie możliwe dozwolone stany (w tym sensie jest algorytmem przeglądu wyczerpującego). Po drugie dzięki zastosowaniu heurystyki, jeśli w pewnym momencie sumaryczna waga spakowanych przedmiotów zrówna się z maksymalną dopuszczaną wagą to mamy gwarancje, że algorytm

znalazł stan optymalny (albo obecny, albo poprzedni optymalny zapisany w pamięci); tym ten algorytm przoduje nad nie heurystycznym algorytmem bruteforce.

— algorithm3- identyczny jak poprzedni, z tą różnicą, że zadajemy mu limit kroków. W ten sposób możemy ograniczyć czas jego działania. Ponieważ dzięki stosowanej heurestyce algorytm przeszukuje stany od tych, które moją największe prawdopodobieństwo na bycie stanami optymalnymi lub bliskimi optymalnych znacznie zwiększamy szanse, że w zadanym czasie znajdziemy "wystarczająco dobre" rozwiązanie.

— algorithm4- przegląd wyczerpujący. Ma złożoność $O(n^2)$ więc nie jest optymalny dla tego problemu. Zdecydowałem się na niego ze względu na łatwość implementacji w wybranej strukturze danych i dlatego iż jest on odpowiedni aby wyciągnąć z tego ćwiczenia właściwe wnioski.

2.3. Pytania

— 1. Jakie rozwiązania i jaką wartość funkcji oceny uzyskano? Czy uzyskano takie same rozwiązania?

— 2. Jak dużą instancję problemu (liczba przedmiotów) da się rozwiązać w około minutę metodą przeglądu wyczerpującego?

— 3. Jak dużą instancję problemu da się rozwiązać w około minutę metodą zachłanną (używając heurystyki)?

— 4. Jak bardzo wydłuży obliczenia dodanie jeszcze jednego przedmiotu?

— 5. Jakie wnioski można wyciągnąć na podstawie wyników tego ćwiczenia?

2.4. Doświadczenia

2.4.1. Ad 1

	Ocena	Czas
Algorytm Heurestyczny	14	5,269050598144531e-05
Przegląd Wyczerpujący	17	1,7404556274414062e-04

Tabela przedstawia rezultat działania algorytmów dla przedmiotów o wagach: 8, 3, 5, 2 i wartościach 16, 8, 9, 6. Przykład ten dobrze ukazuje czym różnią się te algorytmy. Heurystyczny przoduje szybkością, podczas gdy przegląd wyczerpujący gwarantuje optymalne rozwiązanie.

2.5. Dalsze pomiary

Tabele prezentują wyniki testowania algorytmów: 1. Heurystycznego (algorithm1) 2. Ulepszonego heurystycznego (algortihm2) 3. Przeglądy wyczerpującego (algorithm4).

Tabela prezentuje czas w sekundach w zależoności od liczby przedmiotów i algorytmu:

Liczba przedmiotów	8	10	16	20	25	26	27
Algorytm Heurestyczny	7,77e-05	3,62e-05	4,17e-05	4,05e-05	4,29e-05	4,48e-05	4,41e-05
Przegląd Wyczerpujący	2,05e-03	2,51e-03	5,19e-02	9,04e-01	22,88	48,00	96,69
Heurystyczny- Wyczerpujący	1,59e-03	1,18e-03	1,26e-02	4,84e-03	2,7e-03	3,61e-03	2,89e-03

Tabela prezentuje wycenę w zależoności od liczby przedmiotów i algorytmu:

Liczba przedmiotów	8	10	16	20	25	26	27
Algorytm Heurestyczny	33	42	64	79	107	115	101
Przegląd Wyczerpujący	36	46	68	81	107	117	105
Heurystyczny- Wyczerpujący	36	46	68	81	107	117	105

2.5.1. Ad 2

W minutę przeglądu wyczerpującego udało mi się rozwiązać problem 26-przedmiotowy.

2.5.2. Ad 3

Największy problem jaki udało się rozwiązać algorytmem heurystycznym liczył 10 001 000. Zajmowało to 3,61 sekundy.

2.5.3. Ad 4

Przy próbie zwiększenia liczby przedmiotów, mimo licznych prób, kernel Jupytera się crashował.

2.5.4. Ad 5- wnioski

W rozważanym tutaj problemie plecakowym, niezależnie od doboru parametrów było widać dwie prawidłowości:

— 1. Algorytm heurestyczny był wielokrotnie szybszy niż przegląd wyczerpujący. Różnica była tym bardziej widoczna im więcej przedmiotów było do rozważenia.

— 2. Kosztem wspomnianej wyżej szybkości algorytmu heurystycznego była utrata na jakości rozwiązań. Podczas gdy wolniejszy algorytm przeglądu wyczerpującego gwarantował poprawność rozwiązania, algorytm heurestyczny takiej gwarancji niezapewniał. Mimo to, dzięki zastosowanej heurestyce, we wszystkich przeprowadzonych próbach proponowane przez owy algorytm rozwiązania nie były dużo mniejsze od optymalnych, a czasami nawet były optymalnymi. Nie jest to jednak regułą, gdyż łatwo wyobrazić sobie doświadczenie, w którym algorytm heurestyczny będzie swoim rozwiązaniem znacznie odbiegał od rozwiązania optymalnego.

Otrzymane wnioski można ekstrapolować na inne problemy. Z reguły jest tak, że algorytmy opierające się na heurystyce są dużo szybsze, ale nie gwarantują optymalnych rozwiązań. Natomiast algorytmy gwarantujące optymalne rozwiązanie są dużo wolniejsze (przez co mogą wystąpić problemy, w których wykorzystanie takich algorytmów w rozsądnym czasie nie jest możliwe).

3. Gradient descent

3.1. Zadania

— 1. Zaimplementować metodę najszybszego wzrostu/spadku (minimalizacja, spodziewam się stałego współczynnika kroku, jeśli jednak ktoś chce zrobić więcej i zastosować zmienny współczynnik to ma taką możliwość). Gradient wyliczamy numerycznie.

— 2. Narysować zachowanie algorytmu (kolejne kroki algorytmu jako strzałki na tle poziomic funkcji celu). Uwaga: w praktycznych zadaniach optymalizacji nie da się narysować funkcji celu ponieważ zadania mają wiele wymiarów (np. 100), oraz koszt wyznaczenia oceny jednego punktu jest duży.

— 3. Zastosować metodę do znalezienia optimum funkcji booth w 2 wymiarach, po czym do znalezienia optimum funkcji o numerach od 1 do 3 z CEC 2017 w 10 wymiarach (na wykresie narysować kroki w wybranych 2 wymiarach z 10). Ograniczenia kostkowe przestrzeni to -100, 100.

3.2. Algorytmy

Zaimplementowałem 3 algorytmy najszybszego spadku gradientu. Dwie z tych funkcji wyliczają gradient analitycznie dla funkcji Rosenbrocka (findMinimumRosenbrock) i Bootha (findMinimumBooth) i jedna numerycznie, działając dla funkcji więcej niż dwuwymiarowych (findMinimumCec2017).

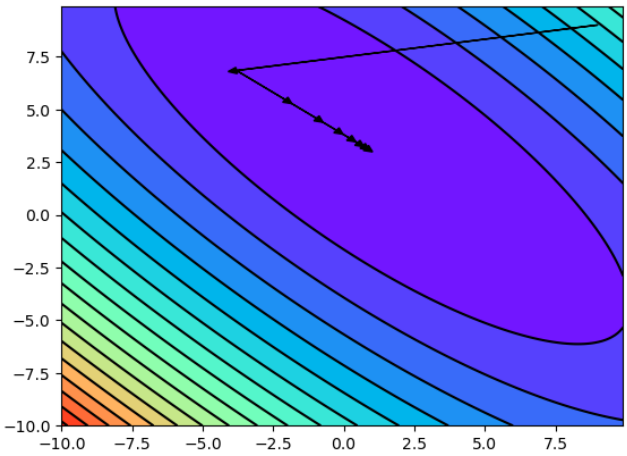
Dla funckji Bootha i Cec2017 (tylko dwa wymiary) istnieją metody ilustrujące na rysunku krok po kroku kroki algorytmu. Metody te, dla czytelności, działają w zakresie argumentów od -10 do 10. Same funkcje optymalizujące dobrze radzą sobie ze znajdowaniem minimów w zakresie szerszym od -100 do 100.

3.3. Prezentacja działania

3.3.1. Funkcja Bootha 1

Koordynaty startowe (9;9), $\beta = 0.1$, limit kroków 10 000, $\epsilon = 0.1$ (po zbliżeniu się o tyle do minimum lokalnego = 0 algorytm kończy działanie).

Wynik działania: (0,79; 3,17) wartość 0,08. Znalezienie zajęło 7 kroków.

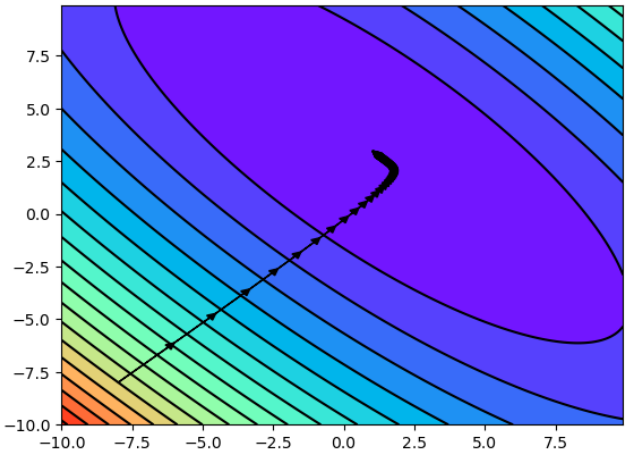


Rys. 1.

Jak widać algorytm zwrócił wynik wystarczająco dobry, zgodny z parametrami. W kolejnym doświadczeniu zdecydowałem się zmniejszyć parametr β do wartości 0.01.

3.3.2. Funkcja Bootha 2

Koordynaty startowe (-8; -8), $\beta = 0.01$, limit kroków 10 000, $\epsilon = 0.1$

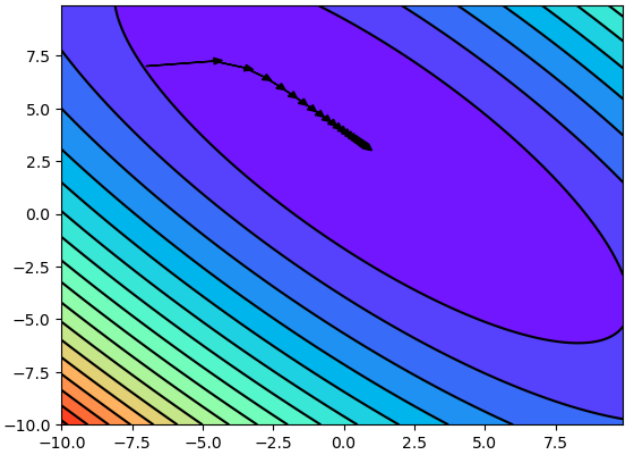


Rys. 2.

Dla parametru $\beta = 0.01$ widać powolniejsze i "łagodniejsze" zbieganie do minimum.

3.3.3. Funkcja Bootha 2

Koordynaty startowe (-7; 7), $\beta = 0.05$, limit kroków 10 000, $\epsilon = 0.1$



Rys. 3.

Pośredni paramter β i pośrednie tempo zbliżania się do minimum. Wynik działania: (0,79; 3,21) wartość 0,09. Znalezienie zajęło 23 kroki.

3.3.4. Funkcja Bootha dla większych liczb

startowy X	startowy Y	β	liczba kroków	ϵ	wynik X	wynik Y	wartość minimum
21	37	0,1	11	0,1	0,80	3,16	0,07
42	0	0,01	270	0,01	1,07	2,93	0,0098
-99	99	0,01	398	0,001	0,98	3,02	0,00097

3.3.5. Funkcja f1 cec2017- aproksymacje minimów

Ustalamy parametr $\beta = 1e-08$. Jest to parametr optymalny, ponieważ jest to największa wartość dla której algorytm sprawnie "zbiega" do minimum funkcji, zamiast nad nim "skakać". Sprawdziłem to testując algorytm cec2017.gradientDescent w trybie verbose.

Algorytm znajduje w 10 000 kroków znajduje minima:

— Dla parametrów: [60, 60, 60, 60, 60, 60, 60, 60, 60, 60]: [-30.76,-70.42,-29.61,-58.32,22.08,59.93,55.73,18.57, 76.68,-52.83] wartość f1=1761.10.

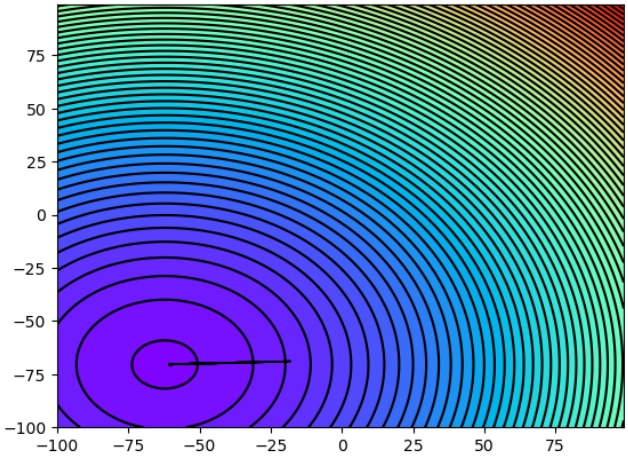
— Dla parametrów: [50, 50, 50, 50, 50, 50, 50, 50, 50, 50]: [-35.05,-70.42,-29.61,-58.32,22.08,59.93,51.33,18.55,76.68,-49.21] wartość f1=1231.28.

— Dla parametrów: [-50, 50, -50, 50, -50, 50, -50, 50, -50, 50]: [-100.49, -57.47, -16.92,-40.15,9.99,58.27,-28.29,23.84,66.47,18.85] wartość f1=1.05886535e+10.

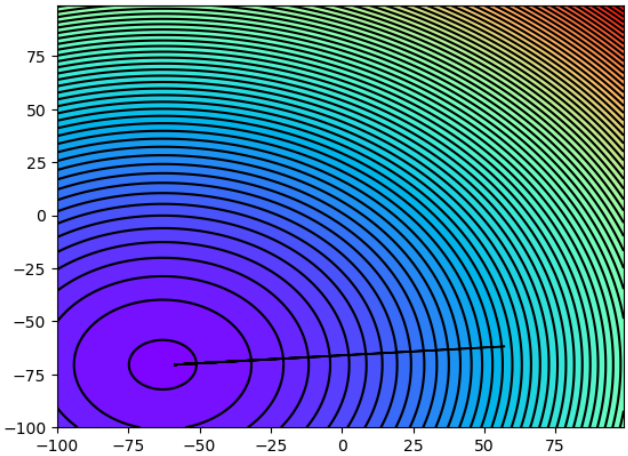
Aproksymacje minimów nie są idealne. Nie można jednak zwiększyć β ponieważ jest ona największą możliwą. Można by zwiększyć liczbę kroków i poczekać dłużej, co dało by lepszy efekt, lub odpalić algorytm ze znalezionego punktu, który jest bliżej minimum i wtrdy dobrać nową β.

3.3.6. Funkcja f1 cec2017- wizualizacja

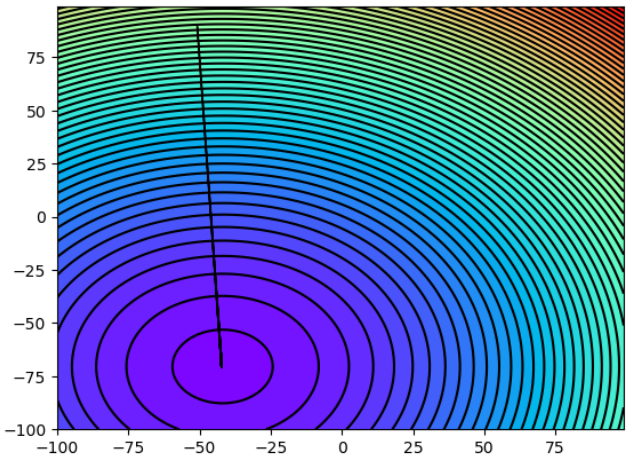
Poniżej, dla funkcji f1, tym razem w 20 wymiarach, zwizualizowano algorytm dla wybranych 2 wymiarów. Punkty startowe losowe:



Rys. 4.



Rys. 5.



Rys. 6.

3.3.7. Funkcja f2 cec2017

Dla funkcji f2 najelpszy okazał się parametr $\beta = 1e - 16$. Dla $\beta = 1e - 15$ przynajmniej jeden z parametrów szybko szybował poza zakres. Dla obranej β jednak każdy krok algorytmu był bardzo niewielki.
Przykładowo znaleziono takią aproksymacje minimum: [-49.78,-9.31,-49.97, 55.24,-20.53,49.40,-50.02,78.52,-65.25,64.75], wartość funkcji: 9.97566006e+12.

3.3.8. Funkcja f3 cec2017

Dla f3 dobrano betę analogicznie jak wcześniej. $\beta = 1e-8$. Przykładowe znalezione minimum: [-42.34,36.80,-46.99,51.60,-51
o wartości: 68165.1449202.

Dla funkcji f1 i f2, przez naturę wybranych X1 i X2 nie znaleziono czytelnych i mówiących cokolwiek wizualizacji.

3.4. Podsumowanie

Algorytm najszybszego spadku gradientu świetnie sprawdził się w przypadku dwuwymiarowej funkcji Bootha. Również poradził sobie dobrze w przypadku funkcji f1. Dla funkcji f2 i f3 pojawiły się problemy z odpowiednim dobraniem współczynnika β. Zbyt duży powodował szybkie wychodzenie argumentów poza zakres, a funkcja rozbiegała się do nieskończoności. Mniejszy natomiast był zbyt mały, żeby w rozsądnym czasie na moim komuterze znaleźć minima 10 wymiarowych funkcji.
Algorytm ten jest odpowiedni dla wielu funkcji. Jest prosty w swoich założeniach i skutecznie znajduje minima. Jednak dla funkcji wielowymiarowych, gdzie koszt obliczenia kroku jest większy, wykonanie algorytmu zawiera zdecydowanie więcej czasu.
Szczególne problemy pojawiają się przy funkcjach "bardzo nierównych", gdzie w jednym momencie współczynnik β w bardzo powolny sposób wpływa na dążenie do minimum, by chwilę później, gdy funkcja "gwałtownie zaczyna się wspinać/maleć" potrafi "wystrzelić" do bardzo odległych wartości.
W przypadku takich funkcji dobranie odpowiedniego współczynnika β jest niemożliwe i stosowanie niezmodyfikowanego algorytmu gradient descent staje się niepraktyczne.