Ćwiczenie-4

1 Gradientowe zejście

Metoda opadania gradientu jest jednym z najczęściej stosowanych algorytmów optymalizacji parametrów w uczeniu maszynowym. Jest to iteracyjny algorytm optymalizacji, który służy do znalezienia lokalnego minimum/maksimum danej funkcji - szczególności w uczeniu maszynowym jest często używany do minimalizacji funkcji koszt/strata (np. w regresji liniowej). W związku z tym najpierw dowiedzmy się, czym jest spadek gradientu funkcji.

Aby znaleźć spadek gradientu funkcji w danym punkcie, co intuicyjnie oznacza znalezienie nachylenia krzywej reprezentującej funkcję w tym punkcie, funkcja musi być różniczkowalna (po ang. differentiable) w każdym punkcie swojej dziedziny oraz wypukła (po ang. convex). Nie wszystkie funkcje są różniczkowalne w każdym punkcie swojej dziedziny. Zwykle funkcje, które mają nieciągłość w swojej dziedzinie (tj. krzywa nieciągła) nie są różniczkowalne. Np. każda funkcja wielomianu jest różniczkowalna w swojej dziedzinie, ale $f(x) = \frac{1}{x}$ nie jest różniczkowalna. Z drugiej strony, dla funkcji wypukłej, jeśli dowolne dwa punkty krzywej reprezentującej funkcję są połączone linią prostą, to prosta powinna leżeć wewnątrz krzywej (tzn. prosta nie powinna przecinać krzywej)

W przypadku funkcji jednowymiarowej zejście gardientowe funkcji w danym punkcie jest po prostu pierwszą pochodną (po ang. first derivative) w tym punkcie. W przypadku funkcji wielowymiarowej spadek gradientu funkcji w danym punkcie jest wektorem pochodnych cząstkowych (po ang. partial derivatives) wzdłuż osi zmiennych. Zatem dla funkcji $f(x)$ z pojedynczą zmienną x gradient $f(x)$, oznaczony przez $\nabla f(x)$, jest zdefiniowany jako $\nabla f(x) = \frac{df(x)}{dx}$. Dla funkcji $f(x_1, x_2, \ldots, x_n)$ ze zmiennymi $x_1, \ldots, x_n$, $\nabla f(x_1, x_2, \ldots, x_n) = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}]$. Należy tu zauważyć, że różniczkując f względem x_i $(i = 1, \ldots, n)$, wyrażenia zawierające inne zmienne x_j, gdzie $j \neq i$, traktuje się jako stałe.

Aby zapoznać się z różnymi rodzajami funkcji jednej lub wielu zmiennych, które są różniczkowalne i wypukłe, proszę o przeczytanie ten artykuł pod tym linkiem.
`https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21`

1.1 Wykorzystanie spadku gradientu w algorytmach uczenia maszynowego

Zobaczmy teraz, jak w algorytmie opadania gradientu, używając pojęcia opadania gradientu funkcji, możemy iteracyjnie zminimalizować koszt/stratę aproksymacji funkcji. W algorytmie podajemy punkt startowy p_0, funkcję f, dla której należy obliczyć gradient, szybkość uczenia μ (po ang. learning rate), o którą iteracyjnie będzie korygowany początkowy wynik. Algorytm iteracyjnie oblicza następny punkt na podstawie gradientu w bieżącym punkcie, skaluje go (z szybkością uczenia) i odejmuje uzyskaną wartość od bieżącego punktu. Odejmuje wartość, ponieważ chcemy zminimalizować funkcję (aby zmaksymalizować byłoby

to dodanie). Tak więc, w pierwszej iteracji opartej na punkcie początkowym, możemy obliczyć następne punkty w następujący sposób.

$$\begin{aligned} p_1 &= p_0 - \mu * \nabla f(p_0) \\ p_2 &= p_1 - \mu * \nabla f(p_1) \\ &\vdots \\ p_n &= p_{n-1} - \mu * \nabla f(p_{n-1}) \end{aligned}$$

Zwykle wybierany jest mały dodatni μ, a liczba iteracji potrzebnych do uzyskania zbieżności procesu zależy od wartości μ. W celu bezpiecznego zakończenia algorytm przyjmuje również maksymalną liczbę iteracji jako dane wejściowe. Jeśli μ jest zbyt duża, algorytm może nie dojść do optymalnego punktu (przeskakiwać) lub nawet całkowicie się rozejść, więc ustalenie maksymalnej liczby iteracji staje się tutaj pomocne. Pamiętajmy też, że jako dane wejściowe algorytm uwzględnia również próg wskazujący, jak duża jest dopuszczalna różnica między rzeczywistą wartością funkcji a przewidywaną wartością funkcji f. Próg ten zasadniczo wskazuje poziom tolerancji koszt/strat ponoszonych w procesie optymalizacji. Powyższy link przedstawia prosty przykład zastosowania algorytmu gradientowego opadania w pythonie.

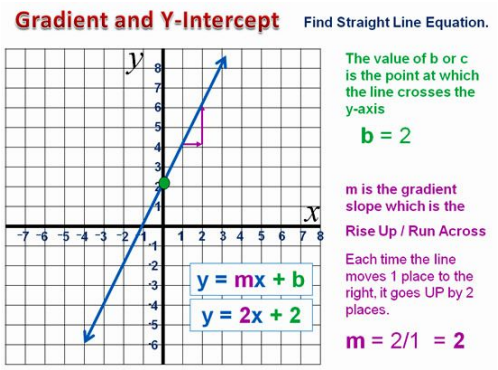
1.2 Wykorzystanie spadku gradientu w regresji liniowej

Rozważmy regresję liniową, która jest podstawowym i powszechnie stosowanym rodzajem analizy predykcyjnej. W szczególności za pomocą regresji liniowej możemy oszacować wartość funkcji $f(x)$. Regresja liniowa dopasowuje linię prostą, która minimalizuje rozbieżności między przewidywanymi a rzeczywistymi wartościami wyjściowymi.

Rozważmy prosty problem z jedną zmienną wejściową x i zmienną wyjściową y parą wartości podaną w postaci tabeli danych. Chcemy znaleźć funkcję, która najlepiej pasuje do tych par wartości wejścia-wyjścia.

dane wejściowe (x)	dane wyjściowe (y)
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_n	y_n

Linia prosta w przestrzeni dwuwymiarowej jest reprezentowana przez następującą funkcję: $\hat{y} = m * x + b$, gdzie m reprezentuje nachylenie, a b reprezentuje punkt przecięcia z osią $\hat{y}$ (lub odchylenie) linia. Poniższy rysunek daje wyobrażenie o współczynnikach m i b w linii.



Istnieje wiele metod określania optymalnych wartości dla m i b, biorąc pod uwagę nasze n punktów danych. Algorytm spadku gradientu ma na celu zminimalizowanie błędu średniokwadratowego między obserwowanymi punktami danych (y) a punktami, które przewidzieliśmy za pomocą naszej linii regresji ($\hat{y}$). Błąd średniokwadratowy (po ang. mean squarred error lub MSE) jest również określany jako funkcja kosztu, zwykle oznaczany jako J, która zależy od wartości m i b, ponieważ przy odpowiednim doborze tych dwóch parametrów przewidywana wartość modelu regresji liniowej (czyli $\hat{y} = m*x+b$) może być bliższa rzeczywistej wartości dla y, a tak więc koszt/strata będzie mniejsza. Więc mamy poniższe równanie na podstawie n par wejście-wyjście $(x_1, y_1), (x_2, y_2) \ldots, (x_n, y_n)$.

$$J(m,b) = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - (m * x_i + b))^2$$

Naszym celem jest dostosowanie naszych parametrów m i b, aż funkcja kosztu (czyli $J(w,b)$) osiągnie swoje minimum i w tym celu musimy obliczyć spadek gradientu dla $J(m,b)$.

$$\nabla J(m,b) = [\frac{\partial J(m,b)}{\partial m}, \frac{\partial J(m,b)}{\partial b}] = \\ [\frac{2}{n}\Sigma_{i=1}^{n}(-x_i)(y_i - (m * x_i + b)), \frac{2}{n}\Sigma_{i=1}^{n} - (y_i - (m * x_i + b))]$$

Teraz możemy krok po kroku obniżyć koszt $J(m,b)$, dostosowując poprzednie wartości m i b za pomocą metody opadania gradientu.

$$m = m - \mu * \frac{\partial J(m,b)}{\partial m}$$
$$b = b - \mu * \frac{\partial J(m,b)}{\partial b}$$

Powyższy krok musimy kontynuować w kilku iteracjach, aż $J(w,b)$ zostanie zminimalizowane do wymaganego stopnia.

Proszę zapoznać się z poniższym linkiem, aby zrozumieć, jak za pomocą regresji liniowej dla danej tabeli danych można przewidzieć funkcję najlepiej dopasowaną do danych i odpowiednio wartości wyjściowe dla odpowiednich danych wejściowych. Można również zobaczyć, jak za pomocą bibliotek pythona można przedstawić wizualną reprezentację zmiany wartości parametrów i linii regresji podczas iteracji.
`https://towardsdatascience.com/gradient-descent-animation-1-simple-linear-regression-e49315`

1.3 Zadania do zrobienia

1. Rozważ poniższe dane, które pokazują procent bezrobotnych w wieku 25 lat (lub starszych) w mieście w pewnym okresie czasu, podanym w latach.

Year	2000	2002	2005	2007	2010
Percentage	6.5	7.0	7.4	8.2	9.0

   Napisz program, który znajdzie model regresji liniowej do przewidywania procentu bezrobotnych w danym roku z dokładnością do trzech miejsc po przecinku.
2. Korzystając z otrzymanego modelu określ, w którym roku procent ten przekroczy 12%.
3. Przedstawić proces znajdowania regresji liniowej przy użyciu niektórych technik animacji z biblioteki (e.g. matplotlib.pyplot) Pythona.

2 Uczenie się z danych

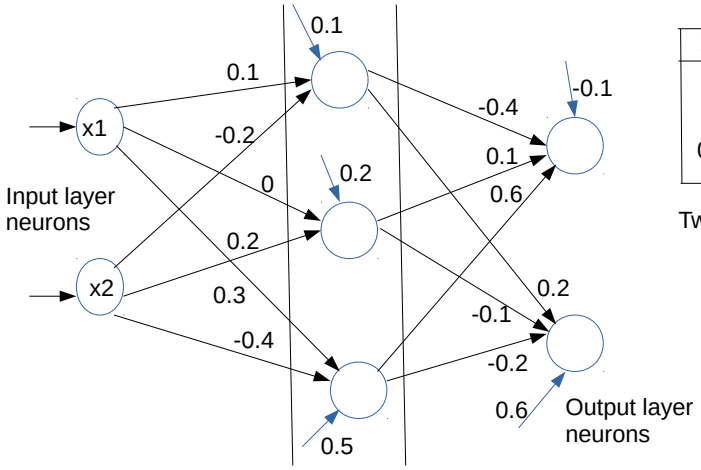
1. Część (i) i (ii) można napisać na papierze.

   (i) Zaprojektuj perceptron z dwoma wejściami reprezentujący funkcję boolowską $x_1 \wedge \neg x_2$.

   (ii) Zaprojektuj dwuwarstwowa sieć perceptronów implementująca $x_1\, XOR\, x_2$.

   (iii) Napisz Perceptron-Learn Program dla funkcji $x_1 \wedge \neg x_2$ (zobacz pseudokod podanym w pliku SI-W11). Jako początkowy wektor wagi można rozważyć $\langle w_0, w_1, w_2\rangle = \langle 0.5,\ 0.5,\ 0.5\rangle$. Współczynnik uczenia się można również ustawić na 0.5.

   (iv) Biorąc pod uwagę następującą sieć neuronową z zainicjowanymi wagami, jak na rysunku, wyjaśnij sieć architektura. Wynik sieci powinien być w stanie poprawnie sklasyfikować wyniki tam, gdzie na rysunku podano przykłady treningowe. Niech współczynnik uczenia się α będzie równy 0.1, a wagi będą takie, jak pokazano na poniższym rysunku. Zrób przód propagację sygnałów w sieci przy użyciu pierwszy przykład jako wejścia, a następnie wykonać propagację wsteczną błąd. Pokaż zmiany wag. (Pomocne będzie najpierw obliczenie kroków na papierze, a następnie, jeśli to możliwe, wykonanie ich za pomocą programu.)



x1	x2	output
0.6	0.1	1
0.2	0.3	0

Two training examples

3 Drzewo decyzyjne

1. Jeśli wybrano Patrona jako atrybut początkowy, znajdź dla pełnej ścieżki drzewa krok po kroku wybór atrybutów, które mają być przetestowane w oparciu o pojęcie entropii (można rozwiązać na papierze, obliczając uzyskanie informacji dla atrybutów)