RAPORT -ZADANIE STAŻOWE

Bartosz Lachowicz

16 KWIETNIA 2020

1. Wstęp

Swój raport chciałbym zacząć od poinformowania Państwa, że biorąc pod uwagę e-mail, który otrzymałem (ten który informował nas, iż cenią państwo sobie prostotę) zarówno jak i ograniczenia czasowe ze względu na końcówkę okresu projektów semestralnych postanowiłem zbytnio nie kombinować. Nie zmienia to faktu, że gdybym miał więcej czasu to te kombinacje miałby miejsce i wprowadziłbym do mojego rozwiązania konkretne ulepszenia. Do alternatywnych, lub bardziej zaawansowanych rozwiązań mojej pracy odniosę się w dalszej części tego dokumentu.

2. Plan pracy

1. Implementacja regresji logistycznej
2. Implementacja BGD vanilla
3. Implementacja SGD zwykłego
4. Implementacja SGD z momentum
5. Dobieranie hiper parametrów
6. Ewaluacja modelu

3. Podejście do danych

Moje podejście do danych było takie aby je wczytać i zamienić liczby pierwsze na 0 a złożone na 1. Jedną z kombinacji o której wspominałem wcześniej byłoby podejście do tego problemu nieco inaczej. Nie zamieniałbym klas na 0 i jedynki, ale spróbował uczyć model regularnego rozpoznawania cyfr a potem starałbym się przekształcić je na 0 i 1.

4. Normalizacja/Skalowanie

Ze względu na funkcję kosztu gdzie mamy logarytm (1 - y_pred) lub (y_pred), jeśli dane nie są znormalizowane pojawia się problem, gdyż **y_pred** może równać się 0 lub 1 i wtedy mamy log(0) czyli błąd. Ten problem próbowałem najpierw rozwiązać korzystając ze wzoru na skalowanie danych $z = \frac{x - średnia(x)}{std(x)}$ gdzie **std** to odchylenie standardowe, w tym samym czasie inicjalizując parametry **theta** używając rozkładu gaussa w przedziale <0,1> (numpy.random.normal). Niestety nie zdało to sprawdzianu. Skończyłem używając wzoru $z = (x - \frac{x - \min(x)}{max(x) - \min(x)}$ co w przypadku datasetu MNIST sprowadza się do podzielenie każdego samplu przez 255. Ja podzieliłem przez 510, i na pewno w przyszłości chciałbym znaleźć jakąś bardziej stosowną metodę obróbki tych danych.

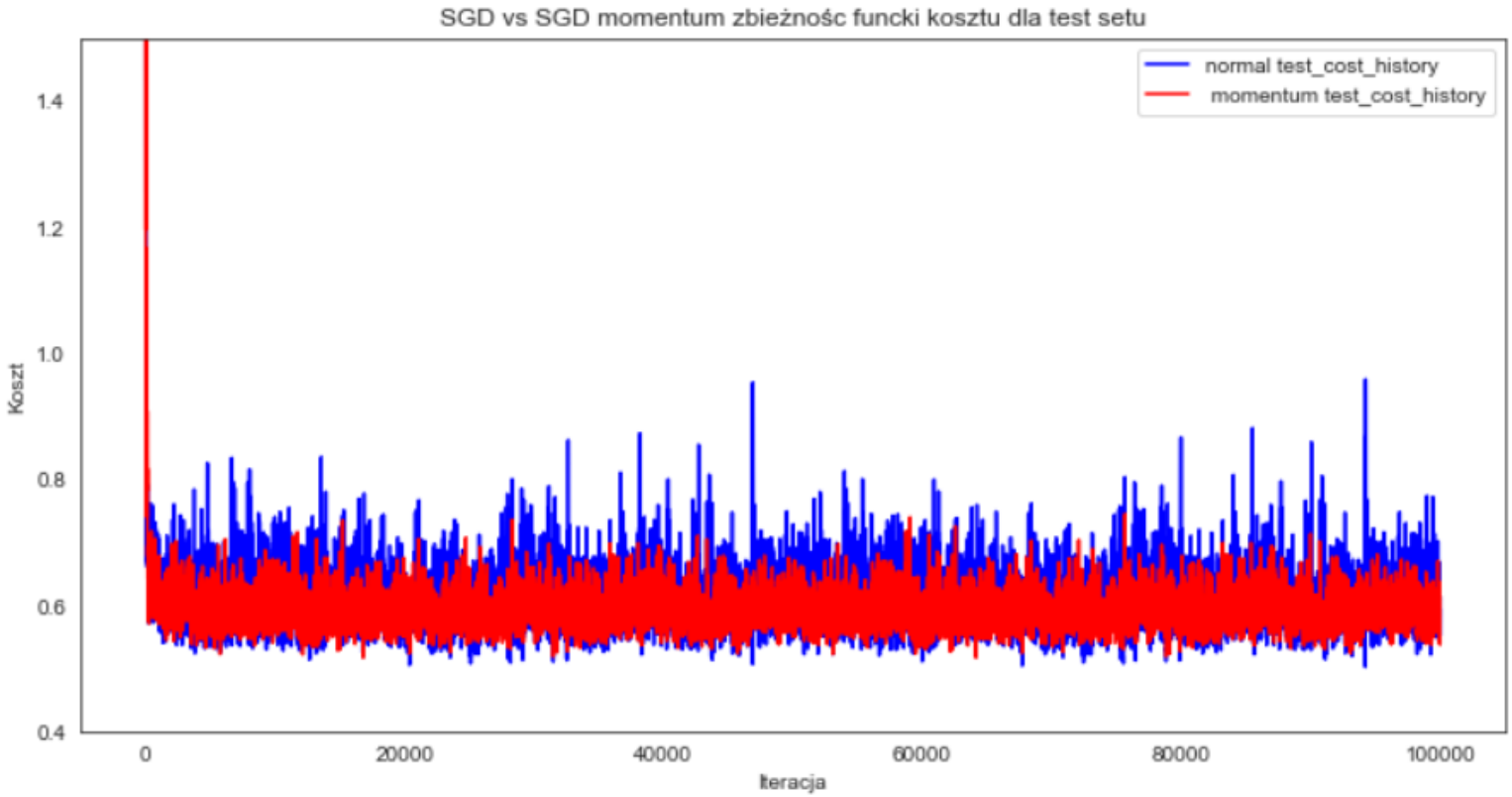
5. Dobieranie hiperparametrów

Ze względu na złożoność obliczeniową tej czynności zdecydowałem się ograniczyć ilość danych zarówno testowych jak i treningowych.
W celu dobrania hiperparametrów dla SGD zwykłego zrobiłem pętlę **FOR** która stopniowo zmieniała wartości **wskaźnika uczenia się gradientów (learning rate**) o 0.05 na każdą iteracji pętli. Kiedy znalazłem najlepszą wartość – **0.055**, zmniejszyłem przyrost w pętli for z 0.05 na 0.01 i próbowałem szukać lepszego wskaźnika bliżej **0.055**, ale ten pozostał bez zmian.
Aby zoptymalizować SGD + momentum zrobiłem dwie pętle **FOR** jedna w drugiej.
Od jednej z nich zależna była wartość **wskaźnika uczenia się gradientów**, a od drugiego **wartość wskaźnika momentu**. Wartości które znalazłem to:

- Wskaźnik uczenia się gradientów - **0.065**
- Wskaźnik momentum - **0.774(9)**

Niestety kiedy spróbowałem użyć tych wartości dla całego datasetu moja funkcja zbiegła się do kosztu **nieco powyżej 0.61**. Dlatego też zacząłem stopniowo szukać wokoło wskaźnika uczenia się gradientów. I tak idąc w dół Porównując zwykły SGD do SGD z momentum użyłem całych dostępnych danych i wykonałem 3 epoki. widać jak wprowadzenie ważone średniej wykładniczej do modelu nadaję mu „pędu” w stronę minima funkcji i przechodząc dane mniej razy można osiągnąć lepsze wyniki.

Wybrany algorytm	Ilość danych testowych	Ilość danych treningowych	Liczba epok/iteracje	Koszt test setu 12000 próbek	Koszt dla test setu całość danych
SGD	12000	2000	4 / 48000	0.56905	0.61364
SGD+ momentum	12000	2000	1 / 12000	0.54879	0.57251



6. Ewaluacja modelu dla SGD/SGD+M

In [21]:
```
iterations = 100000
learning_rate = 0.045
momentum_rate = 0.7749999999999999
theta = np.ones((n,1))/4

(momentum_test_cost_history, momentum_train_cost_history,params_optimal) = sgd_momentum(
                                                                                    lear

test_final_cost = compute_cost(test_X, test_y, params_optimal)
train_final_cost = compute_cost(X, y, params_optimal)
print("Train set Final Cost is: {} \n".format(train_final_cost))
print("Test set Final Cost is: {} \n".format(test_final_cost))
```

```
Train set Final Cost is: 0.5893510999279149

Test set Final Cost is: 0.5866384710754672
```