Dwuwarstwowy model rozpoznawania kwiatów irysa.

Komentarze w kodzie z przyzwyczajenia będą w j. ang. 😊

Model będzie się uczyć na danych z: <https://www.kaggle.com/datasets/uciml/iris>

Zacznę od spojrzenia na dane. Nasze dane prezentują się w ten sposób:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Mamy 4 wartości float, które są opisem naszego kwiata, następnie mamy kolumnę variety, która mówi nam z jakim rodzajem kwiata mamy doczynienia. Mamy 3 rodzaje kwiatów, są to: Setosa, Virginica i Versicolor.

Musimy więc nasze kolory zmapować na liczby, aby nasz model mógł je przyjąć.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Wykonałem od razu szybki podział na dane oraz klasy kwiatów które będziemy klasyfikować – X i y odpowiednio.

Problem nie jest zbyt skomplikowany i model w łatwy sposób jest w stanie się go nauczyć. Mamy tutaj około 150 rekordów kwiatów, więc dobrym pomysłem walidacji naszego modelu byłoby podzielenie setu na dane treningowe oraz testowe do walidacji. Podzieliłem set 70/30, metodą train\_test\_split(). Stratify jest tylko po to aby w każdym z setów było więcej niż 0 każdego rodzaju kwiata.



Teraz zajmę się wczytaniem danych do Loaderów, którymi łatwo będę mógł iterować po rekordach, przy okazji wdrażając podział na ‘batche’.

Na początek zamienie sobie dane na tensory i wczytam do CPU lub GPU.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Teraz stworzę sobie zmodyfikowaną klasę Dataset, na potrzeby naszych danych w której umieszczę nasze dane. Z tej pozycji banalnym będzie zamienienie danych na loadery.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, Czcionka, zrzut ekranu, Grafika

Opis wygenerowany automatycznie

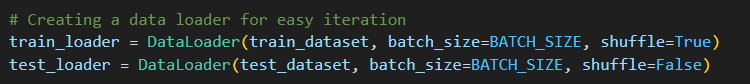
Metody \_\_len\_\_ oraz \_\_getitem\_\_ są to metody wymagane przy tworzeniu customowego Datasetu.

W międzyczasie zdefiniuje sobie dwa hiperparametry których będę używał:

Obraz zawierający tekst, Czcionka, zrzut ekranu, numer

Opis wygenerowany automatycznie

Teraz mogę bezpośrednio stworzyć dataloadery na danych:



Dane przygotowane, więc mogę się teraz zabrać za tworzenie modelu.

Model będzie mało skomplikowany – warstwa wejścia, warstwa ukryta oraz wyjścia, z funkcją ReLU jako funkcją aktywacji. Wszystkie warstwy będą liniowe:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Definiuje sobie funkcje straty oraz optymalizator.

W tym przypadku użyję CrossEntropyLoss, który jest powszechnie stosowany w problemach klasyfikacji. Jest to funkcja logarytmiczna, która mierzy jak daleko jest przewidywanie od prawdy.

Co do optymalizatora, użyje do tego problemu Adam (Adaptive Moment Estimation), który automatycznie dostosowuje współczynniki uczenia dla każdego parametru w modelu na podstawie gradientów oraz ich średnich kwadratów.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Możemy przejść do właściwego treningu modelu.

Do treningu użyje funkcji train:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Dzięki data loaderom mogę po prostu iterować po danych, wydzielając input i label z każdego rekordu. Liczymy dla każdego rekordu wartość straty oraz używam optymalizatora aby dostosować wagi po treningu przez propagacje wsteczną.

Po każdej epoce wyświetlam jaką wartość ma strata, aby sprawdzić czy model się uczy

Teraz mogę wykorzystać ewaluację, aby zobaczyć jak sprawuje się model na danych testowych.

Obliczam sobie przy tym średnią wartość straty oraz celność modelu.

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Model po wytrenowaniu, mogę bez problemu zapisać w pliku .pth, który zapisuje lokalnie.