Uczenie sieci MLP

Joanna Kołodziejczyk

1 Wstęp

Zadanie polegać będzie na implementacji algorytmu uczenia sieci wielowarstwowej (z ograniczeniem do 1 warstwy ukrytej) algorytmem wstecznej propagacji błędów.

Uczenie będzie się odbywać z użyciem zbioru Iris.

2 Problem do rozwiązania

Zadaniem jest implementacja algorytmu uczenia sieci neuronowej typu MLP metodą gradientową klasyfikacji na zbiorze Iris. Sieć ma się składać tylko z jednej warstwy ukrytej co ułatwi obliczenia.

Na początku należy odpowiedzieć na pytania:

1. Ile wejść będzie miała sieć i z czego to wynika?
2. Ile wyjść będzie miała sieć i z czego to wynika?
3. Jaka będzie funkcja aktywacji w neuronach?
4. Jaką funkcję straty wybrać dla wykonywanego zadania?

3 Zadanie programistyczne – krok po kroku

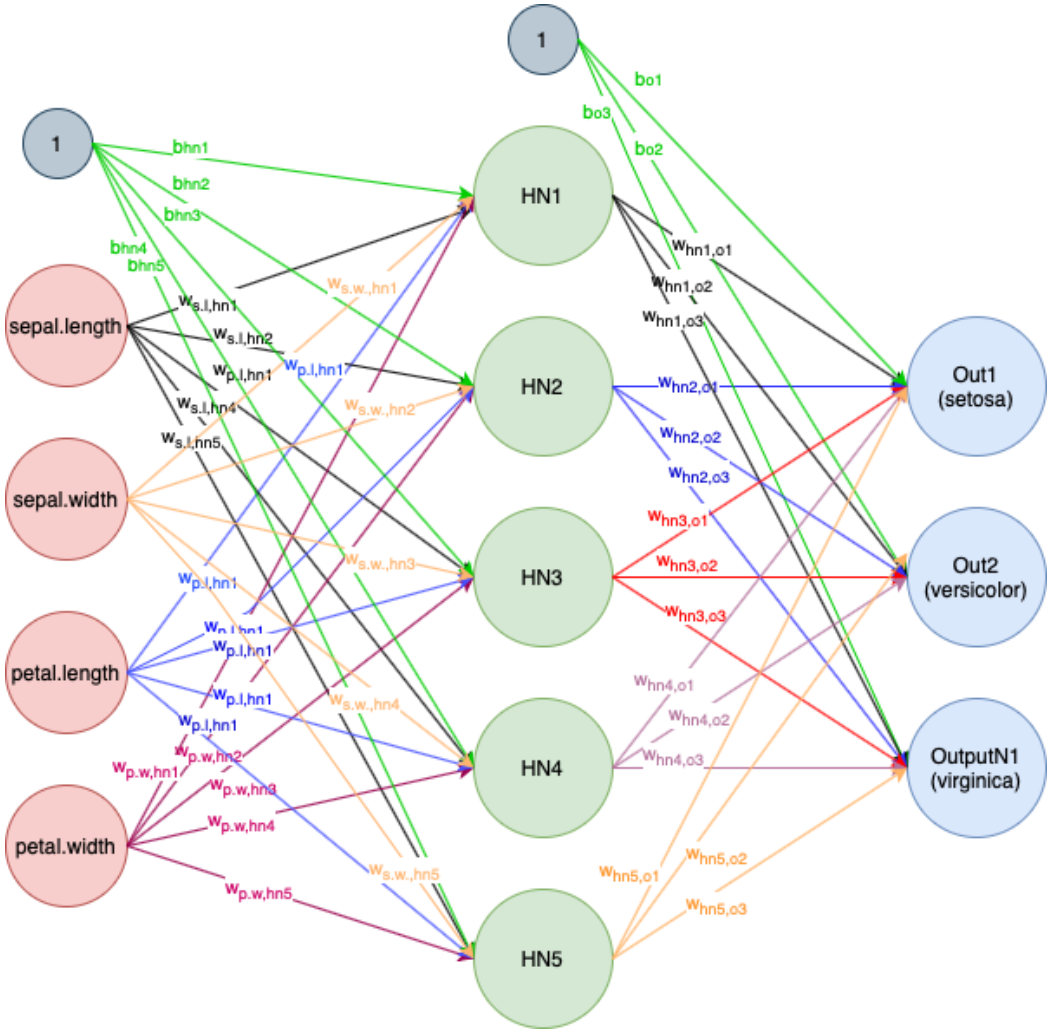
3.1 Wczytanie danych

Zadanie polega na wczytaniu zbioru danych Iris i przygotowaniu go do przetwarzania w sieci neuronowej.

1. Wczytać zbiór danych wykorzystując poniższy kod:

```
from sklearn.datasets import fetch_openml

X_iris, y_iris = fetch_openml(name="iris", version=1, return_X_y=True)
```



Rysunek 1: Architektura sieci do zadania zawierająca 5 neuronów ukrytych.

2. Wczytany zbiór ma na wyjściu wektor zawierający dane typu str o wymiarze [liczba próbek, 1]. Trzeba dokonać jego transformacji na macierz o wielkości [liczba próbek, liczba klas], gdzie

   - 'Iris-setosa' –> [1. 0. 0.],
   - 'Iris-versicolor' –> [0. 1. 0.],
   - 'Iris-virginica'–>[0. 0. 1.].

   W efekcie powstanie macierz y_iris_coded.

3. Dokonać podziału na zbiór uczący i testujący

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
    train_test_split(X_iris, y_iris_coded, random_state=13)
```

3.2 Klasa MLP – klasyfikator neuronowy

Klasa ma opisywać atrybuty i funkcje, działanie modelu neuronowego.

3.2.1 Konstruktor

Zaimplementować klasę MLP

```
class MLP(object):
    def __init__(...
```

z następującymi parametrami, które będą inicjalizowane w konstruktorze.

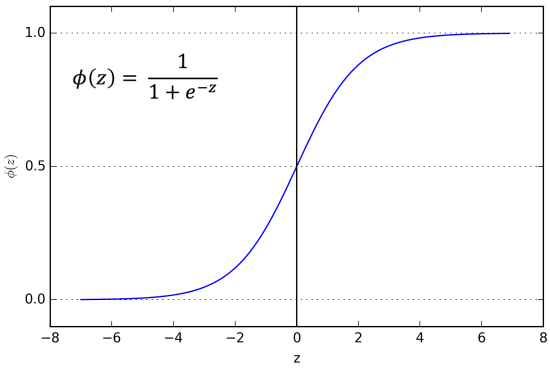
1. hidden : int (domyślnie: 10): Liczba ukrytych neuronów.
2. epochs: int (domyślnie: 100): Liczba epok/kroków prezentacji próbek zestawów treningowych.
3. eta : float (domyślnie: 0.1): Współczynnik uczenia.
4. shuffle: bool (domyślnie: True): Miesza próbki uczące w każdej epoce.

3.2.2 Metoda *_sigmoid*

Zaimplementować w klasie metodę:

```
    def _sigmoid(self, z):
```

obliczającą funkcję logistyczną jako funkcję transferu zgodnie ze wzorem poniżej:



3.2.3 Metoda *_forward*

Zaimplementować w klasie metodę obliczającą wyjście z sieci MLP - krok **forward propagation** dla zadanej macierzy wejść X:

```
def _forward(self, X):
```

w sposób opisany poniżej:

wszystkie parametry sieci (wagi)

- w_h - wagi warstwy ukrytej
- w_o - wagi warstwy wyjściowej
- b_h - biasy warstwy ukrytej
- b_o - biasy warstwy wyjściowej

są atrybutami instancji klasy.

Metoda wykonuje się w następujących krokach:

1. wyjście z sumatorów warstwy ukrytej = suma iloczynów wejść (X o wymiarach [liczba próbek, liczba cech wejściowych]) i wag łączących wejścia z neuronami ukrytymi (w_h o wymiarach [liczba cech wejściowych, liczba neuronów w warstwie ukrytej]) + biasy (b_h o wymiarze [liczba neuronów w warstwie ukrytej]) - wynik ma wymiar [liczba próbek, liczba neuronów ukrytych]

   (Użyć operator numpy.dot do iloczynu)

2. obliczenie funkcji aktywacji (*_sigmoid*) na wszystkich neuronach ukrytych. Działanie na zmiennej obliczonej w kroku 1.

3. wyjście z sumatorów warstwy wyjściowej = suma iloczynów wyjść z warstwy ukrytej o wymiarach [liczba próbek, liczba neuronów ukrytych]) i wag wyjściowych w_o o wymiarach [liczba neuronów w warstwie ukrytej, liczba etykiet klas] + biasy warstwy

wyjściowej b_o [liczba etykiet klas]). Wynik ma wymiar [liczba próbek, liczba etykiet klas].

Działanie z użyciem zmiennej obliczonej w kroku 2.

4. obliczenie funkcji aktywacji $\_sigmoid$ na wszystkich neuronach wyjściowych. Działanie na zmiennej obliczonej w kroku 3.

Metoda zwraca wyjścia z warstwy ukrytej i warstwy wyjściowej.

3.2.4 Metoda $\_compute\_cost$

Zaimplementować w klasie metodę obliczającą funkcję kosztu (cost function) – (Multi-Class Cross-Entropy Loss)

```
def _compute_cost(self, y, out):
```

zgodnie ze wzorem:

$$J(w) = -\sum_{i=1}^{n}\sum_{k=1}^{cl} y_k^{(i)} \log\left(out_k^{(i)}\right) + (1 - y_k^{(i)}) \log\left(1 - out_k^{(i)}\right)$$

gdzie: y_k to wyjście oczekiwane (z próbek) a out_k to wyjście (wartości z warstwy wyjściowej) z sieci neuronowej.

Użyteczny przewodnik: `https://towardsdatascience.com/cross-entropy-for-classification-d98e7f974451`

3.2.5 Metoda fit

Zaimplementować w klasie metodę fit przeprowadzającą uczenie sieci neuronowej metodą wstecznej propagacji błędów:

```
def fit(self, X_train, y_train):
```

1. Inicjalizowanie wag w sieci

   - w_h, w_o - liczbami losowymi z rozkładu normalnego o średniej 0 i odchyleniu std. 0.1
   - b_h, b_o - zerami.

   Gdzie:

   - w_h - wagi warstwy ukrytej mają wymiar: [liczba cech, liczba neuronów ukrytych]
   - b_h - [liczba neuronów ukrytych]
   - w_o - wagi warstwy ukrytej [liczba neuronów ukrytych, liczba klas]
   - b_o - biasy warstwy ukrytej [liczba klas]

2. W pętli indeksowanej liczbą epok (specyficzna nazwa iteracji) wykonywać:

2.1 Pomieszać losowo indeksy w zbiorze uczącym funkcją *shuffle*, jeżeli zmienna $shuffle$ ma wartość $True$.

2.2 Dla każdej próbki ze zbioru uczącego (kolejność określona poprzez losowanie):

2.2.1 Obliczyć wyjścia z sieci metodą $\_forward$ (argumentem metody są wejścia zbioru uczącego) i uzyskać wartości a_o i a_h

2.2.2 Obliczyć pochodną funkcji aktywacji dla warstwy wyjściowej jako: $(a\_o \cdot (1 - a\_o))$

2.2.3 Obliczyć błąd warstwy wyjściowej $delta\_o$ jako: różnicę wyjścia z sieci i wyjścia oczekiwanego (ze zbioru uczącego) * pochodna f. aktywacji warstwy wyjściowej

2.2.4 Obliczyć pochodną funkcji aktywacji dla warstwy ukrytej jako: $(a\_h \cdot (1 - a\_h))$

2.2.5 Obliczyć błąd warstwy ukrytej $delta\_h$ jako iloczyn skalarny $delta\_o$ i w_o (trzeba transponować wektor wag) * pochodna f. aktywacji dla warstwy ukrytej.

2.2.6 Obliczyć gradient dla wag w_h warstwy ukrytej jako iloczyn skalarny wejść do warstwy ukrytej czyli X_train (transponowane) i $delta\_h$.

2.2.7 Gradient dla biasów b_h warstwy ukrytej jest równy $delta\_h$ – wzór się upraszcza, gdyż wejście jest stałą wartością 1.

2.2.8 Obliczyć gradient dla wag w_o warstwy wyjściowej jako iloczyn skalarny wejść do warstwy wyjściowej czyli a_h (transponowane) i $delta\_o$.

2.2.9 Gradient dla biasów b_o warstwy wyjściowej jest równy $delta\_o$ – wzór się upraszcza, gdyż wejście jest stałą wartością 1.

2.2.10 Obliczyć nowe wagi odejmując od poprzednich wartości obliczony gradient pomnożony przez stałą uczenie *eta*. Proces przeprowadzić dla wag: w_h, b_h, w_o i b_o.

2.3 Po nauczeniu wszystkimi próbkami zapisywać (dodawać do listy z wynikami) dla każdej epoki wyniki działania sieci obliczając :

- funkcję kosztu $\_compute\_cost$ dla wyjścia z sieci a_o i wyjścia oczekiwanego ze zbioru uczącego y
- dokładność (accuracy) liczone jako liczba właściwie wskazanych przez sieć klas do liczności zbioru uczącego (uwaga: do obliczenia *accuracy* potrzebne będzie obliczenie *predict*).

2.4 Można po każdej epoce wypisywać wartość funkcji kosztu i dokładność.

3.2.6 Metoda *predict*

Zaimplementować w klasie metodę *predict* zwracającą klasę dla wejścia:

```
def predict(self, X):
```

Obliczyć dla podanego X wyjście z sieci z pomocą $\_forward$. Obliczyć jaka klasa została wskazana i tą wartość zwraca metoda.

3.3 Główna część programu

Główna część programu może być skonstruowana następująco:

1. Wczytanie i przygotowanie danych uczących i testujących.
2. Utworzenie obiektu typu MLP z wybranymi parametrami (domyślne lub inne).
3. Wykonać na modelu uczenie (fit).
4. Wykonać wykresy przebiegu uczenia (oś X - epoki, na osi Y - f. kosztu oraz X - epoki, na osi Y - dokładność)
5. Obliczyć dokładność (*accuracy*) dla zbioru testowego.

4 Zadania

Wszystkie zadania pozwalają na zdobycie max 5 pkt, co należy utożsamić z oceną 5.0.

1. Wykonać implementację zgodnie z opisem - cała sekcja 3 (**4 pkt**) – rozliczenie plik z kodem oraz wykresy.
2. Użyć MLP do klasyfikacji dowolnego zbioru do klasyfikacji binarnej (2 wyjścia) (**1 pkt**) – rozliczenie kod oraz wykresy.
3. Zastosować normalizację wejść w obu zbiorach z użyciem funkcji $MinMaxScaler(0, 1)$. Jaki jest wpływ na uczenie i efektywność MLP? (**zadanie dodatkowe**) – rozliczenie kod oraz wnioski

5 Przekazanie zadań

Kod z rozwiązaniem proszę podpiąć w Teams. Proszę w nazwach plików zawierać swoje nazwisko celem łatwiejszej identyfikacji. Można zadanie w całości przesłać w notatniku Jupiter.