

▼ Perceptron + letters

Autor: Adam Polański

▼ Klasa Perceptron

Klasa została utworzona podczas poprzednich zajęć zgodnie z instrukcją z pliku perceptron.pdf.

```
import numpy as np
class Perceptron(object):
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
        self.errors_ = []
        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

▼ Klasa SLP

Tworzenie klasy SLP krok po kroku:

1. Na początku stworzyłem konstruktor klasy. Nie ma tu nic szczególnego.

2. Stworzyłem metodę `fit(self, X, y)`, a w niej:

- listę `self.errors_ = []`, która w przyszłości będzie przechowywała sumę błędów wszystkich perceptronów w każdej kolejnej iteracji (epochu).
- listę `self.perceptrons_ = [0] * len(X)`, która będzie przechowywała tyle obiektów klasy `Perceptron`, aby na każdą literę przypadła dokładnie jeden z nich.

Następnie dla każdej litery tworzę obiekt typu `Perceptron` i wywołuję metodę `fit()` tego obiektu, przekazując w argumentach wszystkie 10 liter oraz odpowiedni wektor oczekiwanej odpowiedzi.

Następne linijki tej funkcji to sumowanie błędów do listy `self.errors_`.

3. Tworzę metodę `predict(self, X)`, która ma za zadanie generować wektory odpowiedzi.

Tworzę w niej 2-wymiarową tablicę zer, każdy wiersz odpowiada za odpowiedzi jednego perceptrona, a każda kolumna za jedną literę przekazaną w postaci elementu listy `X`.

Dla każdego perceptrona wywołuję jego metodę `predict()` i jej wynik zapisuje do odpowiedniego wiersza. Na koniec zwracam uzyskany zestaw odpowiedzi.

4. Tworzę metodę `misclassified(self, X, y)`, która korzystając z faktu, że odpowiedzi są zapisane w postaci liczb `-1` lub `1` zwraca sumę błędów ostatniego wywołania metody `predict(X)` (wynik wywołania metody zapisany jest w zmiennej `self.predictions`) w stosunku do zestawu odpowiedzi `y`.

Aby uzyskać sumę błędów, najpierw 'odwracam' tablicę `self.predictions` (dla każdego `i` w tablicy `self.predictions` wykonaj `i = i * (-1)`). Aby nie modyfikować oryginalnej tablicy odpowiedzi, wynik takiego działania zapisuje do zmiennej lokalnej.

Następnie sumuje każdy element z `X` i `y`, a wynik:

- `'0'`: oznacza że w danej komórce nie ma błędu
- `'-2'`: oznacza, że dany perceptron zareagował pozytywnie, mimo że nie powinien.
- `'2'`: oznacza, że dany perceptron nie zareagował pozytywnie, mimo że powinien.

Z każdego elementu takiej 'zsumowanej' tablicy, biorę wartość bezwzględną (nie interesują nas typy błędów, a ich ilość) oraz wykonuje dzielenie całkowite przez 2, a następnie sumuje wszystkie elementy do jednej zmiennej.

5. Ostatnia funkcja, to funkcja `show(self, X)`, która ma za zadanie wyświetlić zbiór liter 'X' w postaci graficznej.

```
import numpy as np
import matplotlib.pyplot as plt
class SLP(object):
    def __init__(self, eta=0.05, n_iter=10, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        self.errors_ = []
        self.perceptrons_ = [0] * len(X)
        for i in range(0, len(y)):
            self.perceptrons_[i] = Perceptron(eta=self.eta, n_iter=self.n_iter, random_state=self.random_state)
            self.perceptrons_[i].fit(X, y[i])
            if len(self.errors_) == 0:
                self.errors_ = self.perceptrons_[i].errors_
            else:
                self.errors_ = [sum(j) for j in zip(self.errors_, self.perceptrons_[i].errors_)]

    def predict(self, X):
        self.predictions = np.zeros((len(self.perceptrons_), len(self.perceptrons_)))
        for i in range(0, len(self.perceptrons_)):
            self.predictions[i, :] = self.perceptrons_[i].predict(X)
        return self.predictions

    def misclassified(self, X, y):
        result = 0
        predictions = self.predictions
        predictions = [list(map(lambda x: int(x * (-1)), i)) for i in predictions]
        for i in range(0, len(y)):
            result = result + (sum(map(lambda x: abs(x) // 2, map(sum, zip(predictions[i], y[i]))))))
            #print(list(map(lambda x: abs(x) // 2, map(sum, zip(predictions[i], y[i]))))))
        return result

    def show(self, X):
        fig, axs = plt.subplots(2, 5, figsize=(14, 8), sharex='col', sharey='row')

        for i, x in enumerate(X):
            ax = axs[i // 5, i % 5]
            ax.imshow(x.reshape(7, 5), cmap='binary')

        plt.show()
```

▼ Przygotowywanie danych.

Wczytuje plik z danymi z dysku.

```
import pandas as pd
from google.colab import files
uploaded = files.upload()
```

Wybierz pliki | letters.data

- letters.data(n/a) - 4361 bytes, last modified: 1.06.2023 - 100% done
- Saving letters.data to letters.data

Tworzę obiekt perceptron. Następnie przekazuje dane z pliku do zmiennej `df` typu `DataFrame` i ucinam niepotrzebne wiersze (zostawiam tylko moje przypadki). Aby sprawdzić poprawność danych, wyświetlam 10 ostatnich wierszy.

```
net = SLP()

import io
df = pd.read_csv(io.BytesIO(uploaded['letters.data']), header = None)
#my_cases = '10 11 12 13 14 15 16 17 18 19' # dane z przykładu
my_cases = '2 4 8 9 12 13 14 17 20 23'
my_cases = list(map(int, my_cases.split()))
print(my_cases)
df = df.iloc[my_cases]
df.head(10)
```

```
[2, 4, 8, 9, 12, 13, 14, 17, 20, 23]
```

```
0 1 2 3 4 5 6 7 8 9 ... 51 52 53 54 55 56 57 58 59 60
```

2	-1	1	1	1	-1	1	-1	-1	-1	1	...	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	1	1	1	-1	-1	-1	-1	...	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	-1	1	1	1	-1	-1	-1	1	-1	-1	...	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	1	1	1	1	1	-1	-1	-1	-1	1	...	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
12	1	-1	-1	-1	1	1	1	-1	1	1	...	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
13	1	-1	-1	-1	1	1	-1	-1	-1	1	...	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
14	-1	1	1	1	1	-1	1	-1	-1	-1	1	...	-1	-1	-1	-1	-1	-1	-1	-1	-1
17	1	1	1	1	-1	1	-1	-1	-1	1	...	-1	1	-1	-1	-1	-1	-1	-1	-1	-1
20	1	-1	-1	-1	1	1	-1	-1	-1	1	...	-1	-1	-1	-1	1	-1	-1	-1	-1	-1

Następnie rozdzielałam każdy wiersz na dane litery przechowywane w 'X' oraz dane odpowiedzi przechowywane w 'y'. Dodatkowo, ucinam wszystkie niepotrzebne kolumny ze zmiennej 'y' (nie ma sensu trzymać np. kolumny o indeksie 3, jeżeli wiersz 3 i tak został ucięty). Na końcu wyświetlałam zawartość obu zmiennych.

```
X = df.iloc[:, :35].values
y = df.iloc[:, 35:].values
y = y[:, my_cases]

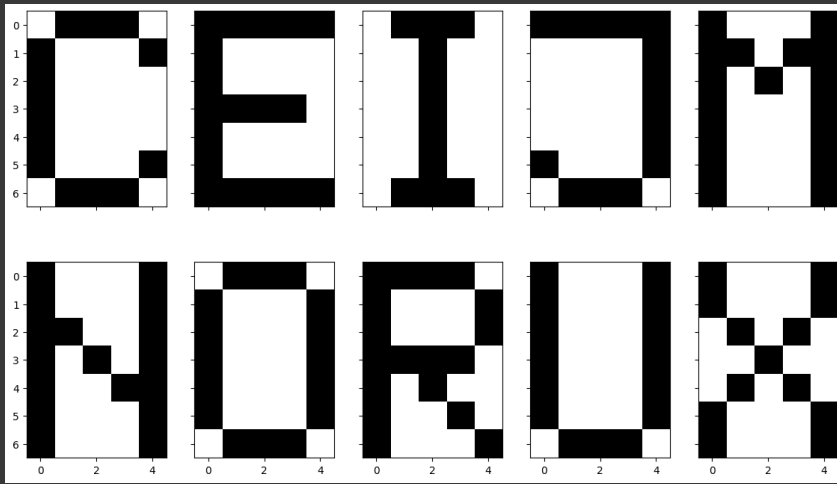
print(X)
print('\n\n')
print(y)

[[-1 1 1 1 -1 1 -1 -1 -1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1
-1 1 -1 -1 -1 1 -1 1 1 1 -1]
[ 1 1 1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 1 -1 1 -1 -1 -1
-1 1 -1 -1 -1 -1 1 1 1 1 1]
[-1 1 1 1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1
-1 -1 -1 1 -1 -1 -1 1 1 1 -1]
[ 1 1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1
1 1 -1 -1 -1 1 -1 1 1 1 -1]
[ 1 -1 -1 -1 1 1 1 -1 1 1 1 -1 1 -1 1 1 -1 -1 -1 1 1 -1 -1 -1
1 1 -1 -1 -1 1 1 -1 -1 -1 1]
[ 1 -1 -1 -1 1 1 -1 -1 -1 1 1 1 -1 -1 1 1 1 -1 -1 1 1 -1 -1 1
1 1 -1 -1 -1 1 1 -1 -1 -1 1]
[-1 1 1 1 -1 1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 -1 -1
1 1 -1 -1 -1 1 -1 1 1 1 -1]
[ 1 1 1 1 -1 1 -1 -1 -1 1 1 -1 -1 -1 1 1 1 1 1 -1 1 -1 1 -1
-1 1 -1 -1 1 -1 1 -1 -1 -1 1]
[ 1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 -1 -1
1 1 -1 -1 -1 1 -1 1 1 1 -1]
[ 1 -1 -1 -1 1 1 -1 -1 -1 1 -1 1 -1 1 -1 -1 -1 -1 -1 1 -1 1
-1 1 -1 -1 -1 1 1 -1 -1 -1 1]]

[[ 1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
[-1 1 -1 -1 -1 -1 -1 -1 -1 -1]
[-1 -1 1 -1 -1 -1 -1 -1 -1 -1]
[-1 -1 -1 1 -1 -1 -1 -1 -1 -1]
[-1 -1 -1 -1 1 -1 -1 -1 -1 -1]
[-1 -1 -1 -1 -1 1 -1 -1 -1 -1]
[-1 -1 -1 -1 -1 1 -1 -1 -1 -1]
[-1 -1 -1 -1 -1 -1 1 -1 -1 -1]
[-1 -1 -1 -1 -1 -1 -1 1 -1 -1]
[-1 -1 -1 -1 -1 -1 -1 1 -1 -1]
[-1 -1 -1 -1 -1 -1 -1 -1 1 -1]]
```

Wyświetlałam wylosowane dla mnie litery.

```
net.show(X)
```



▼ Uczenie modelu

Cały proces uczenia odbywa się w komórce poniżej, lecz w tej sekcji uwzględnij dodatkowo rezultat uczenia się modelu oraz wykres zmian wag modelu.

```
net.fit(X, y)
```

Wywołuje metodę `net.predict(X)`, aby uzyskać od modelu odpowiedzi.

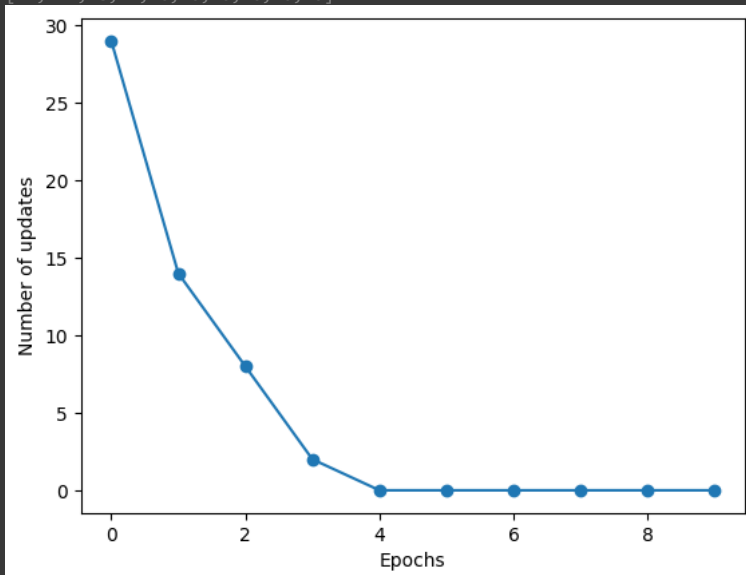
```
print(net.predict(X))
```

```
[[ 1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
 [-1.  1. -1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1.  1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1.  1. -1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1.  1. -1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1.  1. -1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1.  1. -1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1.  1. -1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1.  1. -1.]
 [-1. -1. -1. -1. -1. -1. -1. -1. -1.  1.]]
```

Sprawdzam także, ile zmian nastąpiło podczas każdej iteracji. Liczba '0' w kilku ostatnich iteracjach najczęściej oznacza (nie zawsze), że wagi mają już prawidłowe wartości, a model działa prawidłowo.

```
print(net.errors_)
plt.plot(range(0, len(net.errors_)), net.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
```

```
[29, 14, 8, 2, 0, 0, 0, 0, 0, 0]
```



Sprawdzam, czy model popełnił jakieś błędy.

```
print(net.misclassified(X, y))
```

0

▼ Testowanie możliwości modelu

Definiuje funkcję `damage(X, percent, seed=1)`.

```
def damage(X, percent, seed=1):
    rgen = np.random.RandomState(seed)
    result = np.array(X)
    count = int(X.shape[1]*percent/100)

    for indeks_example in range(len(X)):
        order = np.sort(rgen.choice(X.shape[1], count, replace=False))
        for indeks_pixel in order:
            result[indeks_example][indeks_pixel] *= -1

    return result
```

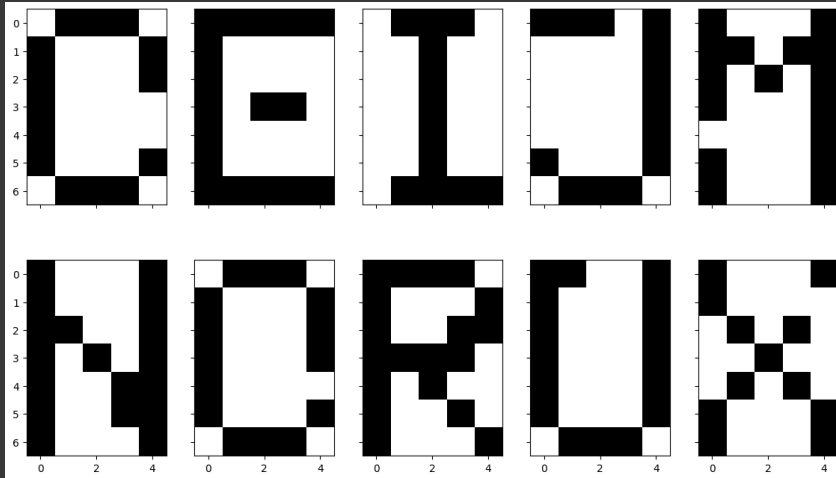
Następnie tworze 3 dodatkowe zbiory, uszkodzone o kolejno:

- 5%
- 15%
- 40%

```
damaged5 = damage(X, 5)
damaged15 = damage(X, 15)
damaged40 = damage(X, 40)
```

Wyświetlam graficznie pierwszy zbiór.

```
net.show(damaged5)
```



Wywołuje metodę `predict` przekazując uszkodzony zbiór.

```
print(net.predict(damaged5))

[[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
 [-1.  1. -1. -1. -1. -1. -1. -1. -1. -1.]
 [-1. -1.  1. -1. -1. -1. -1. -1. -1. -1.]
```

```
[-1. -1. -1.  1. -1. -1. -1. -1. -1. -1.]
[-1. -1. -1. -1.  1. -1. -1. -1. -1. -1.]
[-1. -1. -1. -1. -1.  1. -1. -1. -1. -1.]
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
[-1. -1. -1. -1. -1. -1. -1.  1. -1. -1.]
[-1. -1. -1. -1. -1. -1. -1. -1.  1. -1.]
[-1. -1. -1. -1. -1. -1. -1. -1.  1.  1.]
```

Wywołuje metode `misclassified`, aby sprawdzić ile błędów zostało popełnionych podczas ostatniej predykcji.

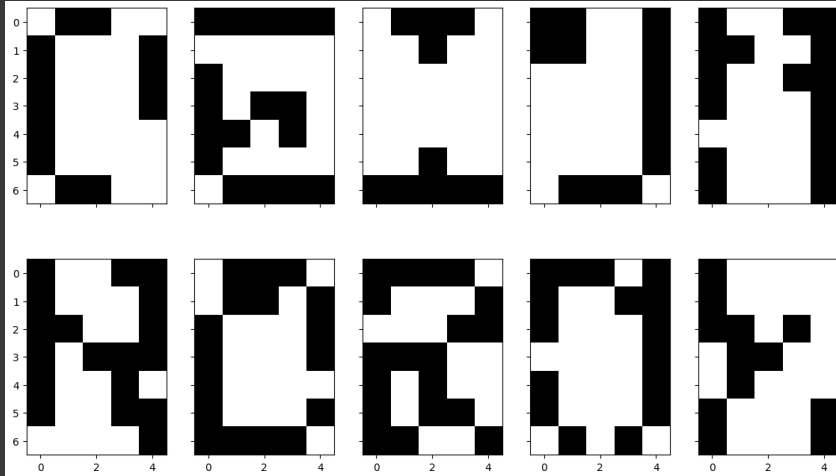
Jak widać 2 razy litera została źle zinterpretowana przez jakiś perceptron.

```
print(net.misclassified(damaged5, y))
```

2

W następnych liniijkach powtarzam powyższe kroki dla zestawów `damage15` oraz `damage40`.

```
net.show(damaged15)
```



```
print(net.predict(damaged15))
```

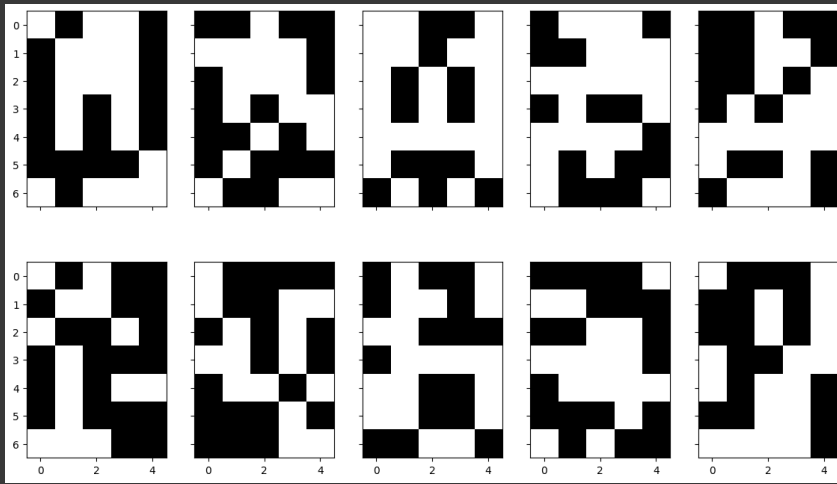
```
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
[-1.  1. -1. -1. -1. -1. -1. -1. -1. -1.]
[-1. -1.  1. -1. -1. -1. -1. -1. -1. -1.]
[-1. -1. -1.  1. -1. -1. -1. -1. -1. -1.]
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
[-1. -1. -1. -1. -1.  1. -1. -1. -1. -1.]
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
[-1. -1. -1. -1. -1. -1. -1.  1. -1. -1.]
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
```

Tym razem 5 błędów.

```
print(net.misclassified(damaged15, y))
```

5

```
net.show(damaged40)
```



```
print(net.predict(damaged40))
```

```
[[-1. -1.  1. -1. -1. -1. -1. -1.  1.]  
 [-1. -1.  1.  1. -1. -1. -1. -1.  1.]  
 [-1. -1.  1. -1. -1. -1.  1.  1. -1.]  
 [-1. -1.  1. -1. -1. -1. -1. -1. -1.]  
 [-1. -1.  1. -1. -1. -1. -1.  1. -1.]  
 [ 1. -1. -1. -1. -1. -1.  1. -1. -1.]  
 [-1. -1.  1. -1. -1. -1.  1. -1.  1.]  
 [ 1. -1.  1. -1. -1.  1. -1.  1. -1.]  
 [ 1. -1. -1.  1. -1. -1. -1. -1. -1.]  
 [-1. -1.  1. -1. -1. -1. -1.  1. -1.]]
```

Aż 29 błędów (większość perceptronów reaguje pozytywnie na więcej niż jedną literę z zestawu damaged40).

```
print(net.misclassified(damaged40, y))
```

29