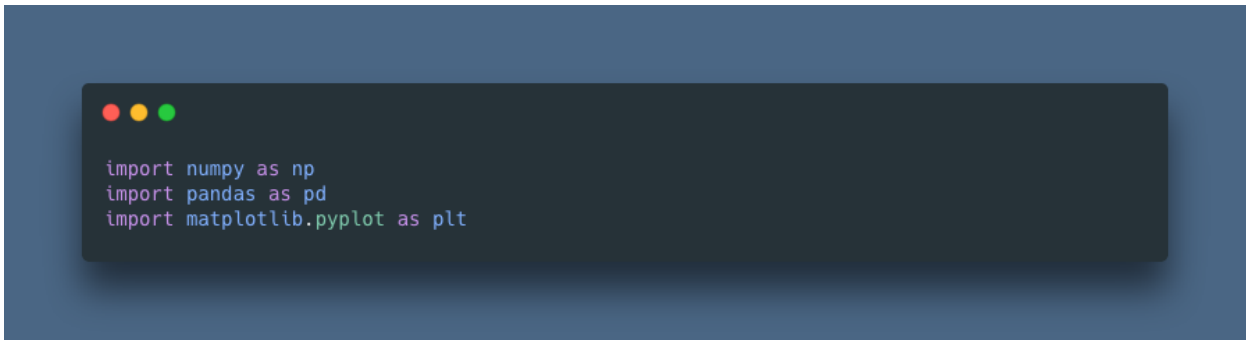
Sieć na własnej implementacji perceptronu

1. Importujemy potrzebne biblioteki:



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

2. Tworzymy klasę realizującą pojedynczy perceptron. Jeden perceptron będzie odpowiedzialny za odgadywanie 1ej z 10ciu liter:

```python
class Perceptron:
    def __init__(self, learning_rate=0.01, learning_iterations=100, random_state=3):
        self.learning_rate = learning_rate
        self.learning_iterations = learning_iterations
        self.random_state = random_state
        self.errors = []
        self.weights = np.array([])

    def predict(self, input_data):
        return np.where((np.dot(input_data, self.weights[1:]) + self.weights[0]) >= 0.0, 1, -1)

    def train_perceptron(self, inputs, targets):
        rgen = np.random.RandomState(self.random_state)
        self.weights = rgen.normal(loc=0.0, scale=0.01, size=inputs.shape[1] + 1)

        for _ in range(self.learning_iterations):
            n_error = 0
            for input_i, target_i in zip(inputs, targets):
                delta_w = self.learning_rate * (target_i - self.predict(input_i))
                self.weights[0] += delta_w

                for i in range(1, len(self.weights)):
                    self.weights[i] += delta_w * input_i[i - 1]

                if delta_w != 0.0:
                    n_error += 1
            self.errors.append(n_error)
        return self
```

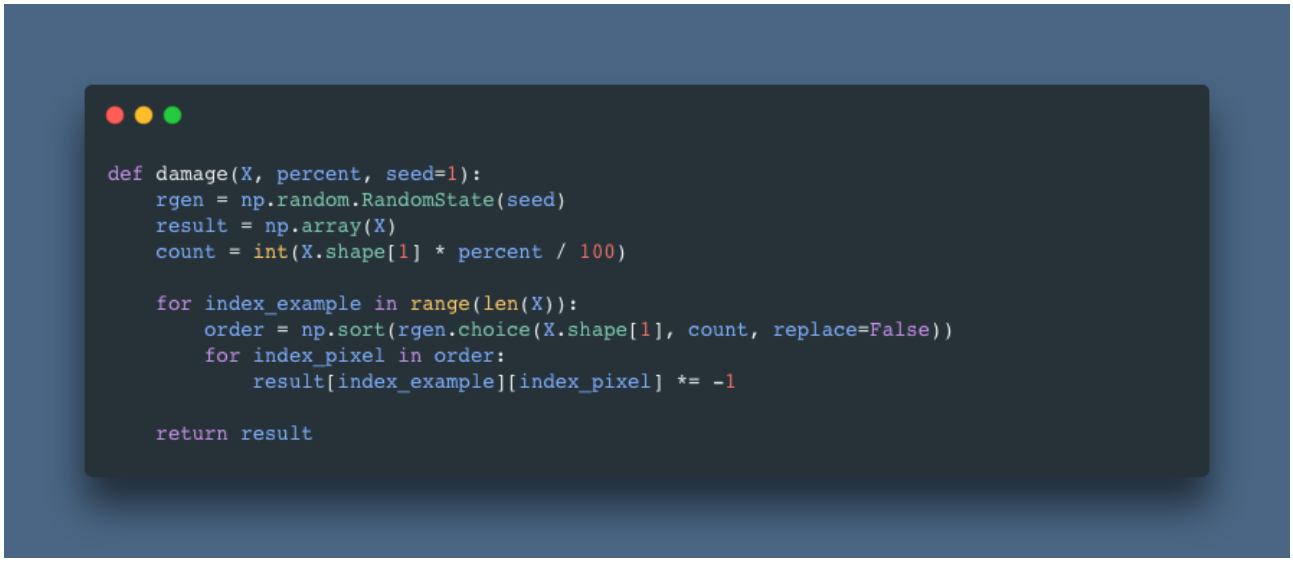
- Konstruktor __init__ - inicjalizuje wartości potrzebne dla procesu uczenia
- Metoda predict - na wejściu otrzymuje kod/y liczby/liczb, na wyjściu zwraca 1 jeśli perceptron się aktywował, -1 w przeciwnym wypadku
- Metoda train_perceptron - przeprowadza uczenie perceptronu

3. Tworzymy klasę Network:

```
class Network:
    def __init__(self, eta=0.05, n_iter=10, random_state=1):
        self.perceptrons = []
        self.predicted = []
        for i in range(10):
            self.perceptrons.append(Perceptron(eta, n_iter, random_state))

    def fit(self, X, Y):
        for i in range(10):
            self.perceptrons[i].train_perceptron(X, Y[i])

    def errors(self):
        total_errors = np.zeros(len(self.perceptrons[0].errors), dtype=int)

        for i in range(10):
            total_errors += np.array(self.perceptrons[i].errors)

        print(total_errors)

        plt.plot(range(1, len(total_errors) + 1), total_errors, marker='x')

        plt.title('errors per learning iteration')
        plt.xlabel('learning iterations')
        plt.ylabel('errors')

        plt.show()

    def show(self, X):
        fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(5.25, 3))

        for i in range(2):
            for j in range(5):

                letter = X[i * 5 + j]

                for y in range(7):
                    for x in range(5):
                        if letter[y * 5 + x] == 1:
                            ax[i, j].scatter(x, y, marker='s', s=90)

                ax[i, j].invert_yaxis()
                ax[i, j].set_xticklabels([])
                ax[i, j].set_yticklabels([])

        plt.show()

    def predict(self, X):
        if len(self.predicted) == 0:
            for i in range(len(self.perceptrons)):
                self.predicted.append(self.perceptrons[i].predict(X))
        else:
            for i in range(len(self.perceptrons)):
                self.predicted[i] = self.perceptrons[i].predict(X)

        print(self.predicted)

    def misclassified(self, Y):
        print("misclassified examples: %d" % (np.array(self.predicted) != Y).sum())
```

- Konstruktor __init__ - tworzy i dodaje do listy perceptrons 10 nowych perceptronów
- Metoda fit - przeprowadza uczenie sieci. W pętli, dla każdego perceptronu z listy perceptrons wywołuje metodę train_perceptron na odpowiednich danych. Teraz każdy perceptron jest w stanie odróżnić kod swojej litery od innych kodów
- Metoda errors - tworzy listę w której jest przechowywana liczba błędów na każdą iterację uczenia. Wyświetla zawartość tej listy, rysuje wykres. Na podstawie tego wykresu możemy łatwo stwierdzić, czy proces uczenia był pomyślny
- Metoda show - na wejściu otrzymuje kody 10ciu liter, a na wyjściu wyświetla je w postaci graficznej.
- Metoda predict - w pętli wywołuje metodę predict dla każdego z perceptronów sieci i dodaje zwróconą listę do listy predicted, jeśli była pusta, w przeciwnym wypadku nadpisuje wartości. Na koniec wyświetla zawartość predicted[]
- Metoda misclassified - wyświetla ile błędów popełniła nasza sieć podczas klasyfikowania

4. Tworzymy funkcję, dzięki której będziemy mogli uszkadzać dane wejściowe(kody liter) o potrzebny nam procent:



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

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

    return result
```

5. Wczytujemy i wybieramy potrzebne dane:

```
[34] df = pd.read_csv('data.csv', header=None)

[26] X = df.iloc[[5, 10, 11, 12, 14, 16, 17, 19, 22, 24], 0:35].values
     Y = df.iloc[0:10, 35:45].values
```

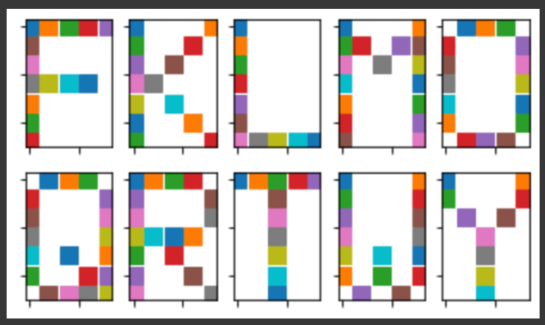
6. Wyświetlamy wybrane dane:

```
[35] X

    array([[ 1,  1,  1,  1,  1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1,
             1,  1,  1, -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1, -1,
            -1, -1, -1],
           [ 1, -1, -1, -1,  1,  1, -1, -1,  1, -1,  1, -1,  1, -1, -1,  1,
             1, -1, -1, -1,  1, -1,  1, -1, -1,  1, -1, -1,  1, -1,  1, -1,
            -1, -1,  1],
           [ 1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1,
            -1, -1, -1, -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1,  1,
             1,  1,  1],
           [ 1, -1, -1, -1,  1,  1,  1, -1,  1,  1,  1, -1,  1, -1,  1,  1,
            -1, -1, -1,  1,  1, -1, -1, -1,  1,  1, -1, -1, -1,  1,  1, -1,
            -1, -1,  1],
           [-1,  1,  1,  1, -1,  1, -1, -1, -1,  1,  1, -1, -1, -1,  1,  1,
            -1, -1, -1,  1,  1, -1, -1, -1,  1,  1, -1, -1, -1,  1, -1,  1,
             1,  1, -1],
           [-1,  1,  1,  1, -1,  1, -1, -1, -1,  1,  1, -1, -1, -1,  1,  1,
            -1, -1, -1,  1,  1, -1, -1, -1,  1,  1, -1, -1, -1,  1, -1,  1,
             1,  1,  1],
           [ 1,  1,  1,  1, -1,  1, -1, -1, -1,  1,  1, -1, -1, -1,  1,  1,
             1,  1,  1, -1,  1, -1,  1, -1, -1,  1, -1, -1,  1, -1,  1, -1,
            -1, -1,  1],
           [ 1,  1,  1,  1,  1, -1, -1,  1, -1, -1, -1, -1,  1, -1, -1, -1,
            -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,
             1, -1, -1],
           [ 1, -1, -1, -1,  1,  1, -1, -1, -1,  1,  1, -1,  1, -1,  1,  1,
            -1, -1, -1,  1,  1, -1,  1, -1,  1,  1, -1,  1, -1,  1, -1,  1,
            -1,  1, -1],
           [ 1, -1, -1, -1,  1,  1, -1, -1, -1,  1, -1,  1, -1,  1, -1, -1,
            -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,  1, -1, -1, -1, -1,
             1, -1, -1]])
```

```
[36] Y

    array([[ 1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
           [-1,  1, -1, -1, -1, -1, -1, -1, -1, -1],
           [-1, -1,  1, -1, -1, -1, -1, -1, -1, -1],
           [-1, -1, -1,  1, -1, -1, -1, -1, -1, -1],
           [-1, -1, -1, -1,  1, -1, -1, -1, -1, -1],
           [-1, -1, -1, -1, -1,  1, -1, -1, -1, -1],
           [-1, -1, -1, -1, -1, -1,  1, -1, -1, -1],
           [-1, -1, -1, -1, -1, -1, -1,  1, -1, -1],
           [-1, -1, -1, -1, -1, -1, -1, -1,  1, -1],
           [-1, -1, -1, -1, -1, -1, -1, -1, -1,  1]])
```

7. Korzystając z metody show() klasy network wyświetlamy dane graficznie:

```
[25] Network().show(X)
```



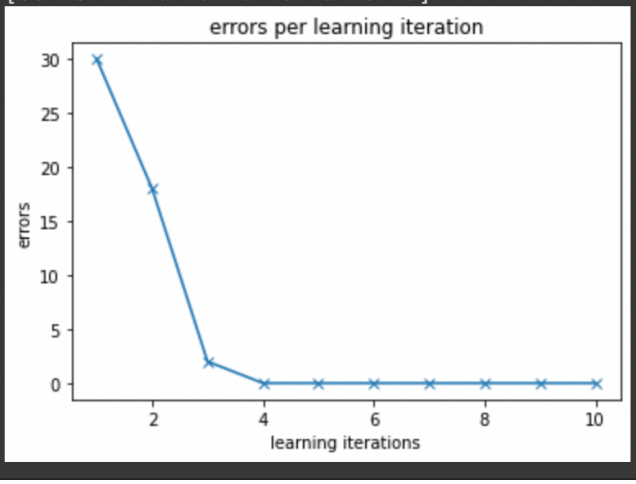
8. Tworzymy obiekt naszej sieci, przeprowadzamy uczenie. Na koniec wyświetlamy wyniki predict na zbiorze uczącym:

```
[138] network = Network()
      network.fit(X, Y)

[139] network.predict(X)
      network.predicted

      [array([ 1, -1, -1, -1, -1, -1, -1, -1, -1, -1]),
       array([-1,  1, -1, -1, -1, -1, -1, -1, -1, -1]),
       array([-1, -1,  1, -1, -1, -1, -1, -1, -1, -1]),
       array([-1, -1, -1,  1, -1, -1, -1, -1, -1, -1]),
       array([-1, -1, -1, -1,  1, -1, -1, -1, -1, -1]),
       array([-1, -1, -1, -1, -1,  1, -1, -1, -1, -1]),
       array([-1, -1, -1, -1, -1, -1,  1, -1, -1, -1]),
       array([-1, -1, -1, -1, -1, -1, -1,  1, -1, -1]),
       array([-1, -1, -1, -1, -1, -1, -1, -1,  1, -1]),
       array([-1, -1, -1, -1, -1, -1, -1, -1, -1,  1])]
```

9. Wyświetlamy zawartość listy errors + rysujemy wykres:

```
network.errors()
```

```
[30 18  2  0  0  0  0  0  0  0]
```



10. Wyświetlamy wynik misclassified na zbiorze uczącym:

```
network.misclassified(Y)
```

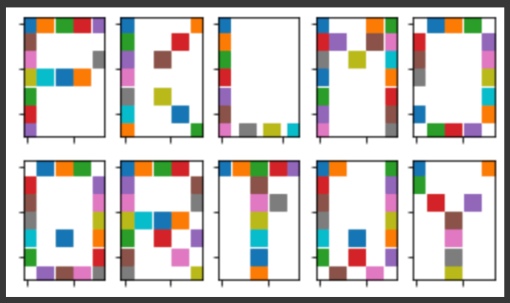
```
misclassified examples: 0
```

11. Korzystając z funkcji damage uszkadzamy kolejno 5%, 15%, 40% danych i zapisujemy te dane do odpowiednich zmiennych:

```
damage5 = damage(X, 5)
damage15 = damage(X, 15)
damage40 = damage(X, 40)
```

12. Wyświetlamy graficznie dane dla zbioru damage5 oraz wyniki predict i misclassified:

```
[143] network.show(damage5)
```



```
[147] network.predict(damage5)
      network.predicted
```

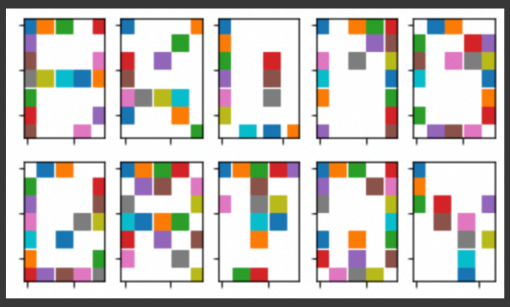
```
[array([ 1, -1, -1, -1, -1, -1, -1,  1, -1, -1]),
 array([-1,  1, -1, -1, -1, -1, -1, -1, -1, -1]),
 array([-1, -1,  1, -1, -1, -1, -1, -1, -1,  1]),
 array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]),
 array([-1, -1, -1, -1,  1, -1, -1, -1, -1, -1]),
 array([-1, -1, -1, -1, -1,  1, -1, -1, -1, -1]),
 array([-1, -1, -1, -1, -1, -1,  1, -1, -1, -1]),
 array([-1, -1, -1, -1, -1, -1, -1,  1, -1, -1]),
 array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]),
 array([-1, -1, -1, -1, -1, -1, -1, -1, -1,  1])]
```

```
[146] network.misclassified(Y)
```

```
misclassified examples: 4
```

13. Wyświetlamy graficznie dane dla zbioru damage15 oraz wyniki predict i misclassified:

```
[197] network.show(damage15)
```



```
[198] network.predict(damage15)
      network.predicted
```

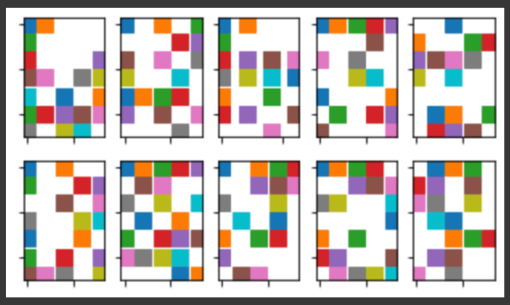
```
[array([ 1, -1, -1, -1, -1, -1, -1,  1, -1, -1]),
 array([-1,  1, -1, -1, -1, -1, -1, -1, -1, -1]),
 array([-1,  1,  1, -1, -1, -1, -1, -1, -1,  1]),
 array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]),
 array([-1, -1, -1, -1,  1, -1, -1, -1, -1, -1]),
 array([-1, -1, -1, -1,  1,  1,  1, -1, -1, -1]),
 array([-1, -1, -1, -1, -1, -1,  1, -1, -1, -1]),
 array([-1, -1, -1, -1, -1, -1, -1,  1, -1, -1]),
 array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]),
 array([-1, -1, -1, -1, -1, -1, -1, -1, -1,  1])]
```

```
[199] network.misclassified(Y)
```

```
misclassified examples: 7
```

14. Wyświetlamy graficznie dane dla zbioru damage40 oraz wyniki predict i misclassified:

```
[239] network.show(damage40)
```



```
[240] network.predict(damage40)
      network.predicted
```

```
      [array([-1, -1, -1,  1, -1, -1,  1,  1, -1,  1]),
       array([-1,  1, -1, -1, -1, -1,  1, -1, -1, -1]),
       array([-1, -1,  1, -1,  1,  1, -1, -1, -1, -1]),
       array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]),
       array([-1, -1, -1, -1,  1,  1, -1, -1,  1, -1]),
       array([-1, -1, -1, -1, -1, -1,  1, -1,  1,  1]),
       array([-1, -1,  1, -1, -1, -1,  1, -1, -1,  1]),
       array([-1, -1, -1, -1, -1, -1,  1, -1, -1,  1]),
       array([-1,  1, -1, -1, -1, -1,  1, -1, -1, -1]),
       array([-1, -1, -1, -1,  1, -1, -1, -1, -1,  1])]
```

```
[241] network.misclassified(Y)
```

```
      misclassified examples: 24
```