Informatyka, studia dzienne, inż I st. semestr VI

Sztuczna inteligencja i systemy ekspertowe 2019/2020

Prowadzący: dr hab. inż. Piotr Lipiński Poniedziałek, 12:00

Data oddania: _____ Ocena: _____

Maciej Pracucik 216869
Adam Jóźwiak 216786

Zadanie 2: Sieć neuronowa służąca do korygowania pomiaru systemu lokalizacji

1. Opis architektury sieci neuronowej

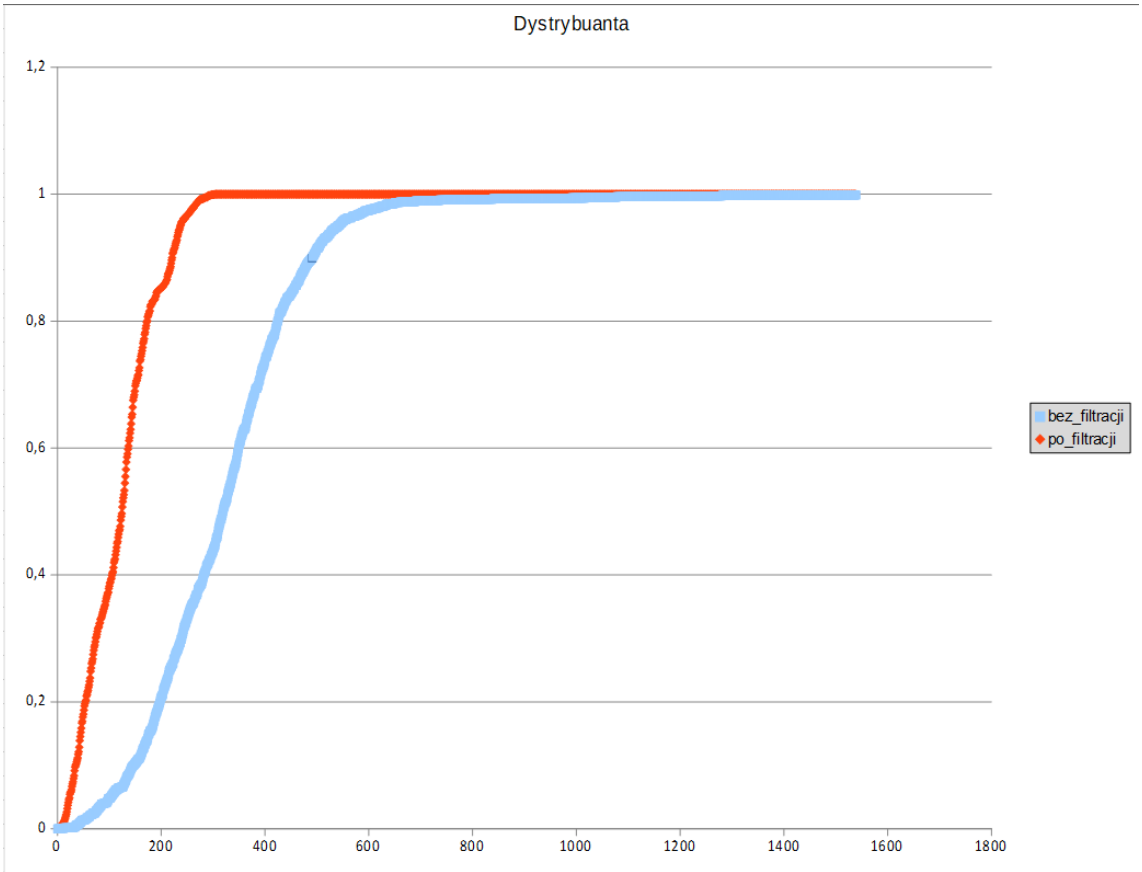
1. Liczba warstw sieci neuronowej - 5 (Jedna warstwa wejscia, trzy warstwy ukryte, jedna warstwa wyjścia)
2. Liczebność neuronów w poszczególnych warstwach - 162, 30, 30, 30, 2
3. Funkcja aktywacji zastosowana w poszczególnych warstwach - RELU (Rectified Linear Unit), która wyrażana jest wzorem y = max(0, x). Dla wszystkich wartości negatywnych przyjmuje wartość 0, natomiast dla wartości pozytywnych jest liniowa
4. Liczba próbek z poprzednich chwil czasowych wykorzystywanych przez sieć neuronową - 80
5. Wagi poszczególnych neuronów w warstwach - zmieniane są ze stałą wartością 0.001, wartość domyślna dla tej biblioteki

2. Opis algorytmu uczenia sieci neuronowej

Algorytm używa klasy MLPRegressor z wysokopoziomowej biblioteki scikit-learn. Implementuje ona Multi-layer perceptron (MLP), który do nauki używa wstecznej propagacji, bez żadnej funkcji aktywacji w warstwie wyjściowej. Do funkcji straty wykorzystuje błąd kwadratowy, natomiast wynik działania sieci składa się z wartości stałych.

Na początku dane z plików xlsx wczytywane są do jednego dużego datasetu, pobieramy z niego dane i dzielimy je na oczekiwane wyniki oraz dane wejściowe. Następnie dzielone są na dane testowe i treningowe. Do danych wejściowych, zarówno testowych jak i treningowych dodawane są punkty z poprzednich chwil czasowych, a w następnej kolejności dane te są skalowane za pomocą zewnętrznej klasy StandardScaler. Tworzymy model MLP z parametrami: ilość neuronów w warstwach ukrytych oraz ilość tych warstw, typ funkcji aktywacji, solver do optymalizacji wag, ilość iteracji oraz wizualne przedstawienie kolejnych iteracji wraz z wartością straty. Do utworzonego modelu przekazywane są dane treningowe wejściowe i wyjściowe. Następnie tworzone są predykcje z wykorzystaniem naszej sieci neuronowej, najpierw z wartościami treningowymi, a następnie dla wartości testowych. Odpowiednio dla każdej z predykcji wyliczany jest błąd średniokwadratowy dla sprawdzenia poprawności i jakości sieci. Dla wizualizacji dane te są przedstawiane na wykresach.

3. Porównanie dystrybuant błędu pomiaru



Rysunek 1. Porównanie dystrybuant błędu pomiaru dla danych ze zbioru testowego oraz dla danych uzyskanych w wyniku filtracji przy użyciu sieci neuronowej.

Dzięki wykresowi dystrybuant błędu, jesteśmy w stanie dowiedzieć się jakie są szanse, że błąd będzie mniejszy niż jego wartość. W naszym przy-

padku, dzięki filtracji za pomocą sieci neuronowej, uzyskaliśmy zdecydowanie mniejsze szanse na uzyskanie błędu o większej wartości.
Weźmy przykładowe wartości błędu np. 400. Przed filtracją mieliśmy 60% szansy, że błąd będzie co najwyżej 400, natomiast po filtracji jesteśmy w 100% pewni, że będzie on poniżej założonej wartości. Biorąc pod uwagę inny przykład, gdy dla błędu 200 nasze wyniki są w 100% mniejsze lub równe 200, to przed filtracją mieliśmy szansę, że błąd będzie wiekszy niż wartość 200. Dzięki filtracji przez sieć neuronową uzyskaliśmy dużo dokładniejsze wyniki, bliższe faktycznej pozycji robota.

4. Kod źródłowy programu do uczenia i testowania sieci neuronowej

```
data = df[['measurement x', 'measurement y', 'reference x', 'reference y']]
input_columns = ['measurement x', 'measurement y']
target_columns = ['reference x', 'reference y']
x = []
y = []
for it in range(data[input_columns[0]].__len__()):
    x.append([data[input_columns[0]][it], data[input_columns[1]][it]])
    y.append([data[target_columns[0]][it], data[target_columns[1]][it]])

# Splitting data into train and test
x_train, x_test = x[:-1540], x[-1540:]
y_train, y_test = y[:-1540], y[-1540:]

# Add previous time steps
prev_steps = 80
for it in range(x_train.__len__()):
    for j in range(prev_steps):
        x_train[it].append(x_train[it - j][0])
        x_train[it].append(x_train[it - j][1])

for it in range(x_test.__len__()):
    for j in range(prev_steps):
        x_test[it].append(x_test[it - j][0])
        x_test[it].append(x_test[it - j][1])

# Scaling input data
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

# Creating MLP model and training on data
mlp = MLPRegressor(hidden_layer_sizes=(30, 30, 30), activation='relu', solver='adam', max_iter=300, verbose=True)
mlp.fit(x_train, y_train)
print(mlp.coefs_)

# Predictions and plotting
predict_train = mlp.predict(x_train)
print(mean_squared_error(y_train, predict_train))
predict_test = mlp.predict(x_test)
print(mean_squared_error(y_test, predict_test))
```

Rysunek 2. Kod źródłowy programu.

Za naukę odpowiada metoda fit wywoływana na modelu sieci neuronowej. Z kolei za testowanie odpowiada drugie wywołanie metody predict, gdzie jako parametr przyjmowany jest zbiór testowy.

Literatura

[1] T. Oetiker, H. Partl, I. Hyna, E. Schlegl. *Nie za krótkie wprowadzenie do systemu LaTeX2e*, 2007, dostępny online.

[2] Dokumentacja biblioteki Scikit - Learn *https://scikit-learn.org/*

[3] Pluralsight - tutorial *https://www.pluralsight.com/guides/machine-learning-neural-networks-scikit-learn*