Informatyka, studia dzienne, inż I st. semestr VI

Sztuczna inteligencja i systemy ekspertowe 2019/2020

Prowadzący: dr inż. Krzysztof Lichy wtorek, 10:30

Data oddania: _____ Ocena: _____

Radosław Grela 216769
Jakub Wąchała 216914

Zadanie 2: Sieć neuronowa

1. Cel

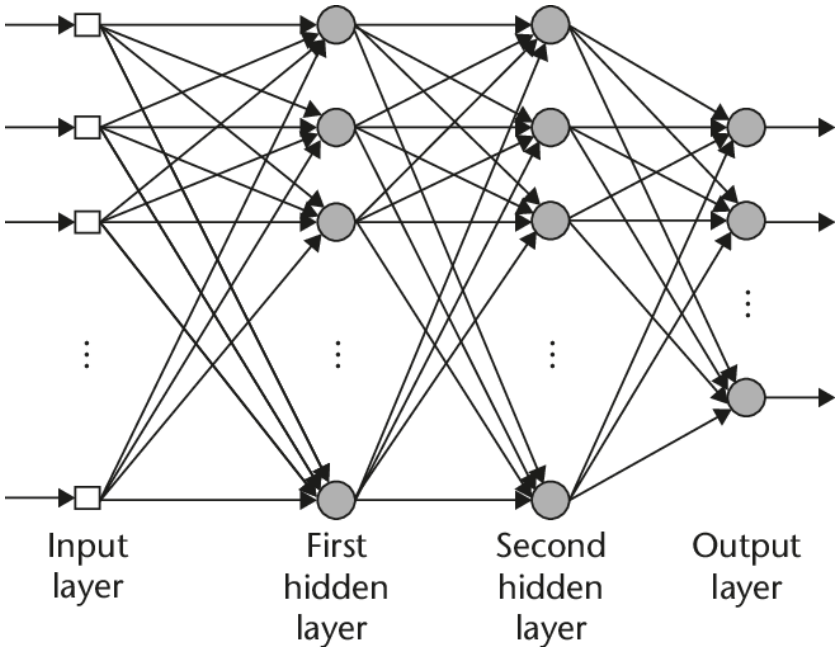
Zadanie 2 polega na zaprojektowaniu i zaimplementowaniu sieci neuronowej, która pozwoli na korygowanie błędów uzyskanych z systemu pomiarowego. Do nauki sieci neuronowej należało wykorzystać dane pomiarowe z 12 przejazdów testowych zawarte w plikach z rozszerzeniem .xlsx. Następnie, za pomocą pliku testowego porównaliśmy uzyskane wyniki. [1]

2. Wprowadzenie

Wielowarstwowy perceptron to (MLP - MultiLayer Perceptron) to jeden z najpopularniejszych typów sieci neuronowej. Składa się zazwyczaj z jednej warstwy wejściowej i wyjściowej oraz kilku warstw ukrytych. Ilości neuronów w poszczególnych warstwach musi zostać ustalona przez twórcę. [2]

Neuron w rozumowaniu matematycznym / informatycznym to pewnego rodzaju sumator, który oblicza sumę ważoną, a następnie podstawia ją do funkcji aktywacji. Wynikiem tej operacji jest wyjście neuronu. [4]

Nauka takiej sieci neuronowej polega na nieustannym zmienianiu wag neuronu w taki sposób, aby błąd był jak najmniejszy. Realizuje się to za pomocą wstecznej propagacji błędów.



Rysunek 1. Przykładowy schemat sieci neuronowej MLP. [3]

2.1. Opis architektury sieci neuronowej

Nasza sieć neuronowa wykorzystana do rozwiązania problemu korygowania błędów systemu pomiarowego posiada następujące właściwości:

1. Liczba warstw sieci neuronowej: 5
2. Liczebność neuronów w poszczególnych warstwach: 50
3. Funkcje aktywacji zastosowanych w poszczególnych warstwach: „Relu", czyli *rectified linear unit*, f(x) = max(0, x)
4. Liczba próbek z poprzednich chwil czasowych wykorzystywanych przez sieć neuronową:
5. Wagi poszczególnych neuronów w warstwach:

2.2. Opis algorytmu uczenia sieci neuronowej

Sieć neuronowa została nauczona za pomocą algorytmu „Adam". Jest to stochastyczny optymalizator gradientowy. Wybraliśmy go, ponieważ dobrze nadaje się do badania dużych zbiorów danych (co najmniej kilka tysięcy danych treningowych). [6]

Algorytm „Adam" został zaproponowany przez Jimmy Lei Ba oraz Diederik P. Kingma. Jest algorytmem optymalizującym pewną zadaną funkcję kosztu opierającym się na *stochastic gradient descent*. Główna różnica między zwykłym stochastic gradient descent a Adamem polega na liczeniu przesunięcia wartości nie tylko na podstawie aktualnego gradientu, ale również poprzednich. Wpływ gradientu na kolejne przesunięcia spada jednak wykładniczo. Pamiętanie poprzednich gradientów nie tylko pozwala na szybszą naukę, ale nawet osiąganie lepszych wyników przy tej samej architekturze. [7]

Algorytm przebiega w następujący sposób [8]:

1. Obliczenie wykładniczo średniej ważonej przeszłych gradientów i zapisanie ich do zmiennych VdW i Vdb (przed korektą odchylenia) oraz VdW-corrected i Vdbcorrected (z korekcją odchylenia).
2. Obliczenie wykładniczo ważoną średnią kwadratów poprzednich gradientów i zapisanie ich w zmiennych SdW i Sdb (przed korektą odchylenia) oraz SdWcorrected i Sdbcorrected (z korekcją odchylenia).
3. Aktualizacja parametrów w kierunku opartym na łączeniu informacji z „1" i „2".

3. Opis implementacji i kod źródłowy

Napisany przez nas program jest aplikacją w języku Python. Składa się z dwóch plików: FileReader.py, zawierający klasę odpowiedzialną za wczytanie plików z danymi treningowymi oraz testowymi. Drugi plik, Main.py, odpowiada za sterowanie programem oraz siecią neuronową.

3.1. FileReader.py

```
from pathlib import Path

import openpyxl

prefix = 'dane\\pozyxAPI_only_localization'
main_file = prefix + "_measurement"
test_file = prefix + '_dane_testowe_i_dystrybuanta'
suffix = ".xlsx"
nr_of_records = 1540


class FileReader:

    def __init__(self):
        self.train_ref = []
        self.train = []
        self.test = []
        self.test_ref = []

    def read_train_files(self, nr: int):
        xlsx_file = Path(main_file + str(nr) + suffix)
        wb_obj = openpyxl.load_workbook(xlsx_file)
        sheet = wb_obj.active
        for row in sheet.iter_rows(2, nr_of_records + 1):
            self.train.append([row[4].value, row[5].value])
            self.train_ref.append([row[6].value, row[7].value])

    def read_test_file(self):
        xlsx_file = Path(test_file + suffix)
```

```
        wb_obj = openpyxl.load_workbook(xlsx_file)
        sheet = wb_obj.active
        for row in sheet.iter_rows(2, nr_of_records + 1):
            self.test.append([row[4].value, row[5].value])
            self.test_ref.append([row[4].value, row[5].value])

    def read_files(self):
        for i in range(1, 13):
            self.read_train_files(i)
        self.read_test_file()
        return self.get_all_data()

    def get_all_data(self):
        return self.train, self.train_ref, self.test, self.test_ref
```

3.2. Main.py

```
import pandas as pd
import warnings
from FileReader import FileReader
from math import sqrt
from sklearn.neural_network import MLPRegressor
from sklearn.exceptions import ConvergenceWarning

warnings.filterwarnings(action='ignore', category=ConvergenceWarning)
mlp = MLPRegressor(max_iter=200, tol=1, activation='relu',
                   solver='adam', shuffle=False,
                   random_state=3, hidden_layer_sizes=(3, 3),
                   alpha=0.001, momentum=1)


def count_errors(predicted_: [], reference: []):
    for i in range(len(predicted_)):
        tmp_x = reference[i][0] - predicted_[i][0]
        tmp_y = reference[i][1] - predicted_[i][1]
        errors.append(sqrt(tmp_x ** 2 + tmp_y ** 2))


def count_distribution():
    distribution = []
    for i in range(int(max(errors) + 2)):
        distribution.append(0)
        count_elements_less_than_i(distribution, i)
    save_to_xlsx(distribution)


def count_elements_less_than_i(distribution, i):
```

```
    for j in range(len(errors)):
        if errors[j] < i:
            distribution[i] += 1 / 1540


def save_to_xlsx(tab):
    df = pd.DataFrame({"dystrybuanta": tab})
    df.to_excel('test.xlsx', sheet_name='sheet1', index=False, header=


def print_results():
    for i in range(len(predicted)):
        print(i + 1, "predicted:", predicted[i],
              "reference:", test_ref[i],
              "␣error:", errors[i])
    print("average␣error:", sum(errors) / len(errors))


if __name__ == "__main__":
    train, train_ref, test, test_ref = FileReader().read_files()
    mlp.fit(train, train_ref)
    predicted = list(mlp.predict(test))
    errors = []
    count_errors(predicted, test_ref)
    print_results()
    count_distribution()

    # print(mlp.coefs_[0])  # warstwa wyjsciowa
    # print(mlp.coefs_[1])  # warstwa ukryta
    # print(mlp.coefs_[2])  # warstwa ukryta
```

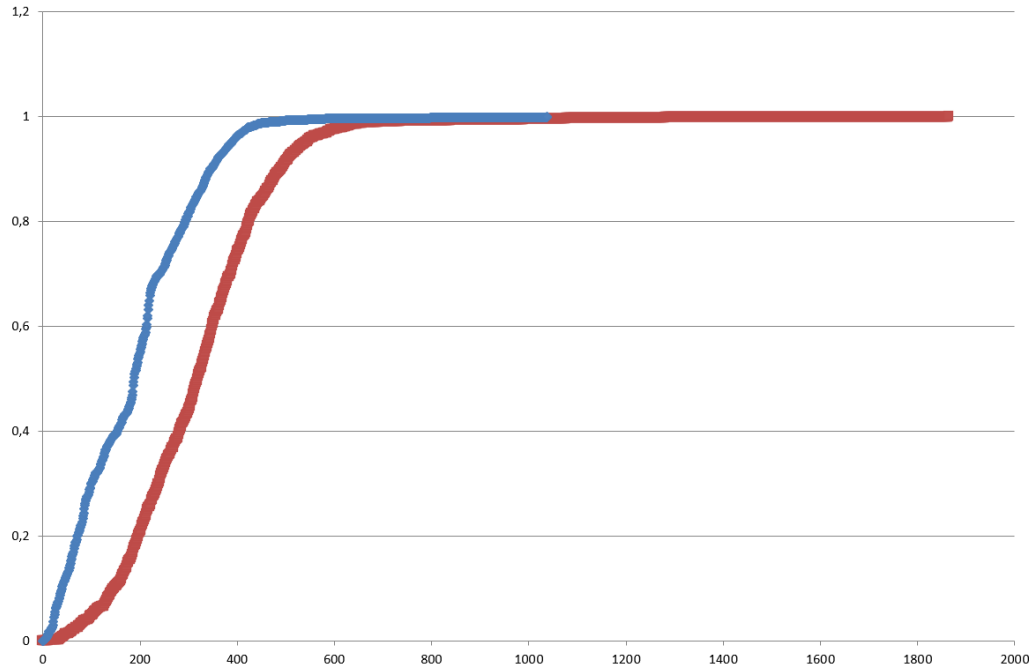
4. Materiały i metody

Do przeprowadzenia nauki sieci neuronowej użyliśmy 12 plików z danymi pomiarowymi przejazdu robota. Następnie, do sprawdzenia wyników nauczenia się sieci neuronowej wykorzystaliśmy plik z danymi testowymi. [9]

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

6. Dyskusja

dyskusja



Rysunek 2. Porównanie dystrybuant błędu pomiaru.

7. Wnioski

wnioski

Literatura

[1] *https://ftims.edu.p.lodz.pl/mod/page/view.php?id=73137* [dostęp 08.05.2020]
[2] *https://pl.wikipedia.org/wiki/Perceptron_wielowarstwowy* [dostęp 08.05.2020]
[3] *https://www.researchgate.net/publication/244858164/figure/fig2/AS:670028080902154@153675855* [dostęp 08.05.2020]
[4] *https://pl.wikipedia.org/wiki/Neuron_McCullocha-Pittsa* [dostęp 08.05.2020]
[5] *https://arxiv.org/pdf/1412.6980.pdf* [dostęp 08.05.2020]
[6] *https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html* [dostęp 08.05.2020]
[7] *https://piotrmicek.staff.tcs.uj.edu.pl/machine-learning/docs/szymon-stankiewicz-lincencjat.pdf* [dostęp 08.05.2020]
[8] *https://engmrk.com/adam-optimization-algorithm/* [dostęp 08.05.2020]
[9] *https://ftims.edu.p.lodz.pl/mod/resource/view.php?id=73138* [dostęp 08.05.2020]