

Data oddania: \_\_\_\_\_

Ocena: \_\_\_\_\_

Mikołaj Pawłoś 258681

Emilia Szczerba 251643

## Zadanie drugie: Poprawa lokalizacji UWB przy pomocy sieci neuronowych

### 1. Cel

Zaprojektowanie i zaimplementowanie sieci neuronowej, która pozwoli na korygowanie błędów uzyskanych z systemu pomiarowego.

### 2. Wprowadzenie

#### 2.1. Sieć neuronowa

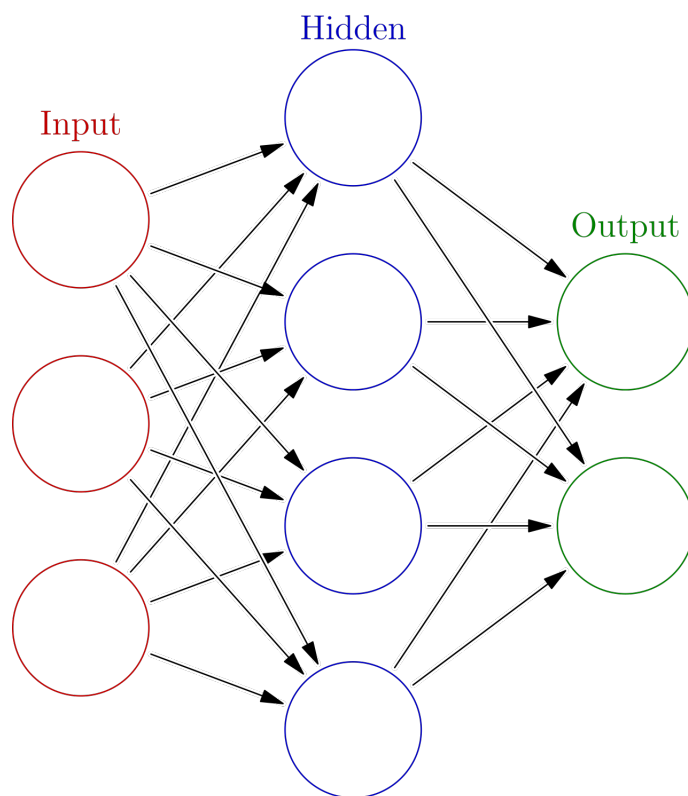
**Sieć neuronowa** (znana również jako *sztuczna sieć neuronowa*, w skrócie **NN** lub **ANN**) to model obliczeniowy inspirowany strukturą i funkcjami biologicznych sieci neuronowych. Sieć neuronowa składa się z połączonych jednostek lub węzłów, zwanych **sztucznymi neuronami**, które luźno odwzorowują neurony w mózgu.

**Neurony** są połączone **krawędziami**, które odwzorowują synapsy w mózgu. Każdy sztuczny neuron odbiera sygnały od połączonych z nim neuronów, przetwarza je, a następnie wysyła sygnał do kolejnych połączonych neuronów.

„**Sygnał**” ma postać liczby rzeczywistej, a wyjście neuronu obliczane jest za pomocą pewnej nieliniowej funkcji sumy jego wejść, zwanej **funkcją aktywacji**.

Siła sygnału w każdym połączeniu jest określana przez **wagę**, która jest dostosowywana podczas procesu uczenia.

Zazwyczaj neurony grupowane są w **warstwy**. Różne warstwy mogą wykonywać różne transformacje danych wejściowych. Sygnały przepływają od pierwszej warstwy (*warstwa wejściowa*) do ostatniej (*warstwa wyjściowa*), przechodząc być może przez kilka warstw pośrednich (*warstw ukrytych*). Sieć nazywa się **głęboką siecią neuronową**, jeśli zawiera co najmniej dwie warstwy ukryte.



Rysunek 1: Schemat przykładowej sieci neuronowej

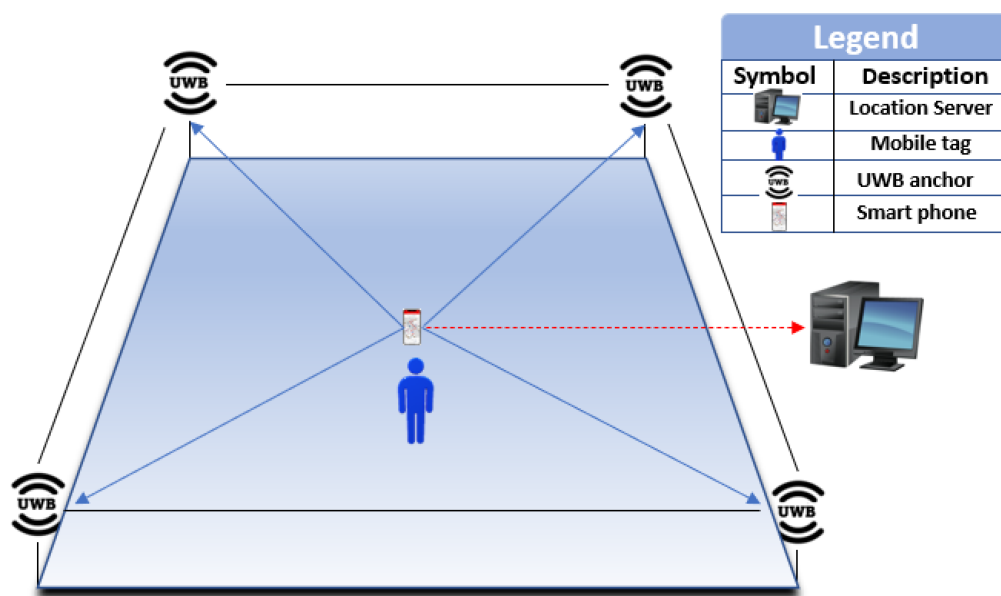
Sztuczne sieci neuronowe są wykorzystywane w różnych zadaniach, takich jak *modelowanie predykcyjne*, *sterowanie adaptacyjne* czy *rozwiązywanie problemów z zakresu sztucznej inteligencji*. Potrafią uczyć się na podstawie doświadczenia i wyciągać wnioski z złożonych i pozornie niepowiązanych danych.

## 2.2. Systemy lokalizacji UWB

Technologia **Ultra-Wideband (UWB)** wykorzystuje krótkie impulsy radiowe o bardzo szerokim paśmie częstotliwości (powyżej 500 MHz), co umożliwia precyzyjny pomiar odległości między nadajnikiem a odbiornikiem. Główną zaletą UWB jest wysoka odporność na zakłócenia wielodrogowe (*multipath*) oraz możliwość osiągnięcia dokładności lokalizacji rzędu **kilku centymetrów**.

Kluczowe metody lokalizacji w UWB obejmują:

- **TOA (Time of Arrival)** – pomiar czasu dotarcia sygnału,
- **TDOA (Time Difference of Arrival)** – różnica czasu dotarcia do wielu odbiorników,
- **RSSI (Received Signal Strength Indication)** – pomiar mocy sygnału (mniej precyzyjny niż TOA/TDOA).



Rysunek 2: przykładowy system lokalizacji UWB

Ograniczenia systemów UWB obejmują:

- Wpływ przeszkód fizycznych (np. ściany) na propagację sygnału,
- Błędy synchronizacji czasowej między urządzeniami,
- Zależność od konfiguracji środowiska (np. rozmieszczenie anchorów).

### 2.3. Uczenie sieci neuronowej

Proces uczenia sieci neuronowej polega na dostosowywaniu wag połączeń między neuronami w celu minimalizacji funkcji straty (*loss function*), która mierzy różnicę między przewidywaniami modelu a rzeczywistymi wartościami. Kluczowe etapy tego procesu obejmują:

- **Propagację wprzód** (*forward propagation*) – obliczenie wyjścia sieci na podstawie danych wejściowych i aktualnych wag.
- **Obliczenie funkcji straty** – np. średni błąd kwadratowy (MSE) dla zadań regresji:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad (1)$$

gdzie  $y_i$  to wartość rzeczywista, a  $\hat{y}_i$  to przewidywana wartość.

- **Propagację wsteczną** (*backpropagation*) – obliczenie gradientów funkcji straty względem wag i ich aktualizacja przy użyciu optymalizatora (np. Adam, SGD).

W projekcie zastosowano dodatkowe techniki poprawiające uczenie, takie jak:

- **Wczesne zatrzymanie** (*early stopping*) – przerwanie uczenia, gdy błąd na zbiorze walidacyjnym przestaje maleć.
- **Normalizacja danych** – przekształcenie cech do podobnego zakresu wartości (np. przy użyciu StandardScaler).

### 3. Opis implementacji

Zadanie zostało wykonane przy użyciu języka **Python3**, z wykorzystaniem następujących bibliotek:

- `Matplotlib`
- `Numpy`
- `PyTorch`
- `scikit-learn`

Projekt został podzielony na następujące pliki:

1. `DataLoader.py` – moduł odpowiedzialny za wczytywanie, filtrowanie i wstępne przetwarzanie danych pomiarowych z plików Excel.
2. `NeuralNetworkModel.py` – główny moduł modelu uczenia maszynowego opartego na sieci neuronowej. Odpowiada za tworzenie, trenowanie, testowanie i zapisywanie modelu, z uwzględnieniem mechanizmów detekcji outlierów i normalizacji danych.
3. `OutlierDetector.py` – moduł odpowiedzialny za identyfikację obserwacji odstających w danych wejściowych. Wspiera różne metody detekcji, m.in. oparte na odległości Mahalanobisa czy odchyleniu standardowym.
4. `main.py` – plik uruchomieniowy, zawierający konfigurację eksperymentów, wczytywanie danych, inicjalizację modelu oraz zapisywanie wyników i metryk. Stanowi punkt wejścia do całego systemu.

### 3.1. Plik `DataLoader.py`

Moduł `DataLoader.py` odpowiada za ładowanie oraz wstępne przetwarzanie danych pomiarowych wykorzystywanych w procesie uczenia i testowania modeli. Główne zadania realizowane przez ten moduł to:

- automatyczne lokalizowanie folderów z danymi wejściowymi (statycznymi i dynamicznymi),
- odczyt danych z plików Excel (`.xlsx`) znajdujących się w podfolderach F8 i F10,
- wstępne czyszczenie danych – usuwanie pustych wierszy,
- selekcja i ekstrakcja cech sensorycznych z kolumn (np. akcelerometr, żyroskop, ciśnienie, kwaterniony),

Dane są dzielone na zbiór treningowy (pochodzący z plików statycznych) oraz testowy (z plików dynamicznych), a następnie przekształcane do postaci numerycznych macierzy przy pomocy biblioteki NumPy. Moduł ten stanowi fundament do dalszych etapów analizy i modelowania, zapewniając jednolite i ustandaryzowane

---

### 3.2. Plik `NeuralNetworkModel.py`

Plik `NeuralNetworkModel.py` zawiera kompletną implementację modelu sieci neuronowej zaprojektowanej z myślą o regresji wielowymiarowej oraz automatycznej detekcji i eliminacji obserwacji odstających (ang. outliers).

Moduł składa się z dwóch głównych klas:

- **NeuralNetwork** – definiuje architekturę wielowarstwowej sieci neuronowej typu *feedforward* z konfigurowalną liczbą warstw ukrytych, funkcją aktywacji i mechanizmem Dropout.
- **EnhancedNeuralNetworkModel** – klasa wyższego poziomu integrująca funkcje uczenia, walidacji, normalizacji danych, predykcji oraz obsługi danych odstających przy użyciu zewnętrznego modułu `OutlierDetector`.

Model wspiera różne optymalizatory (Adam, SGD), posiada wbudowany mechanizm *early stopping*, adaptacyjny harmonogram uczenia (`ReduceLROnPlateau`) oraz możliwość zapisu i odczytu wag modelu (zarówno w formacie binarnym `.pt`, jak i w postaci tabeli CSV).

Trening odbywa się z użyciem biblioteki PyTorch, a dane wejściowe są wstępnie przeskalowywane za pomocą standaryzacji (`StandardScaler`) z biblioteki `scikit-learn`. Proces walidacji może być przeprowadzony na osobnym zbiorze lub automatycznie podzielony z danych treningowych.

Architektura modelu jest w pełni parametryzowana: użytkownik może zdefiniować liczbę neuronów wejściowych, wyjściowych, strukturę warstw ukrytych, liczbę epok, współczynnik uczenia, funkcję aktywacji, metodę detekcji outlierów, a także urządzenie obliczeniowe (CPU lub GPU).

## **Architektura modelu bez eliminacji wartości odstających**

Po wielu eksperymentach, metodą prób i błędów, zdecydowaliśmy się na zastosowanie poniższej architektury sieci neuronowej.

- Liczba warstw ukrytych: 3
- Liczba neuronów w warstwach: 256, 128, 64
- Funkcja aktywacji: ReLU (Rectified Linear Unit)
- Liczba neuronów wyjściowych: 2
- Liczba epok: 300
- Rozmiar partii (batch size): 128
- Współczynnik uczenia (learning rate): 0,001

### 3.3. Plik `main.py`

#### Główna funkcja `main()`

Funkcja `main()` zawiera pełny pipeline systemu korekcji i składa się z następujących etapów:

1. Wczytanie danych uczących i testowych przy pomocy klasy `DataLoader`.
2. Utworzenie i trenowanie dwóch modeli sieci neuronowych:
  - bez usuwania wartości odstających (outlierów),
  - z eliminacją outlierów przy użyciu złożonej metody detekcji.
3. Przewidywanie błędów przez modele, ich odejmowanie od danych wejściowych oraz ocena skuteczności korekcji.
4. Obliczenie metryk błędu dla każdego z podejść oraz zapis wyników do pliku tekstowego i arkusza kalkulacyjnego.
5. Generowanie wykresów i zapis danych dystrybuanty błędów do plików `.xlsx`.
6. Zapis wytrenowanych modeli w formacie PyTorch (`.pth`).

#### Efekty działania i generowane pliki

W wyniku działania programu generowane są:

- Wykresy: `training_history.png`, `error_analysis_*.png`,
- Modele: `model_bez_outlierow.pth`, `model_z_outlierami.pth`,
- Dane wyjściowe: `dystrybuanta_*.xlsx`, `wyniki_szczegolowe.xlsx`, `raport_podsumowanie.txt`.

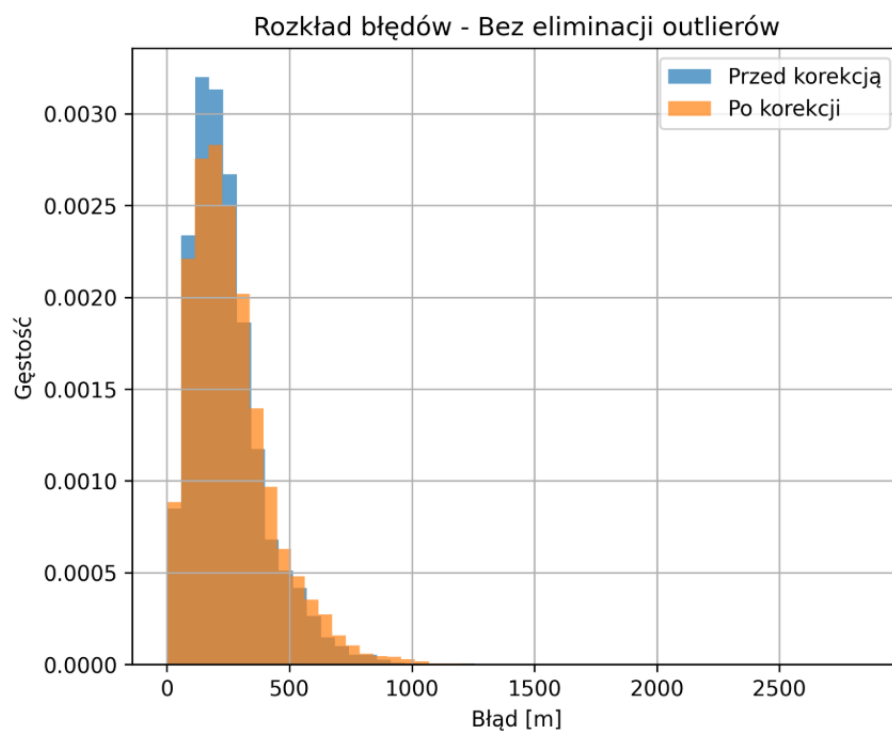
Plik ten stanowi główny komponent praktyczny systemu korekcji błędów i integruje cały proces od wczytania danych po ocenę skuteczności modelu.

## 4. Materiały i metody

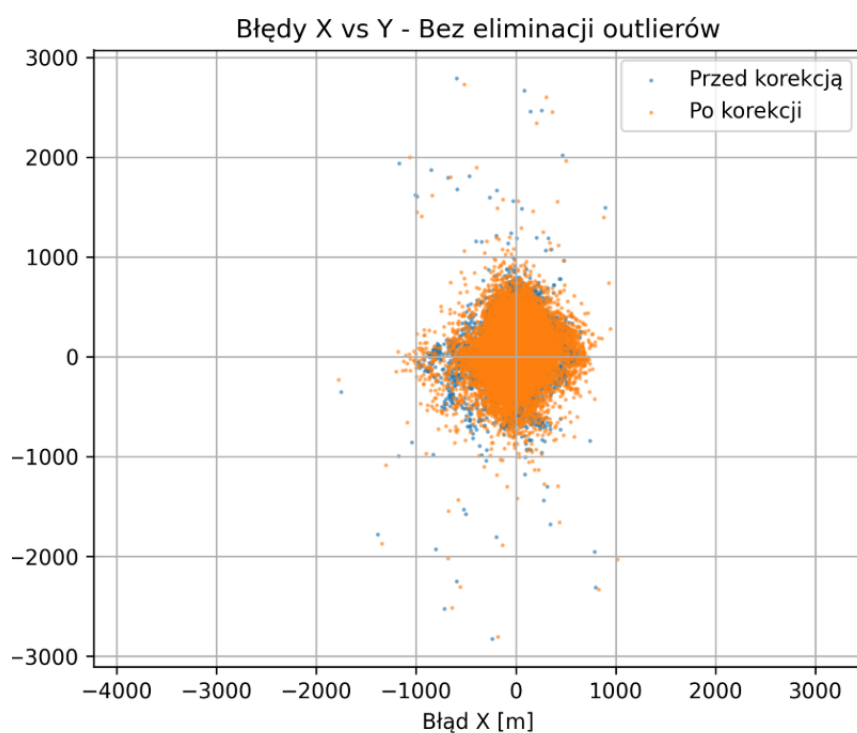
W tym miejscu należy opisać, jak przeprowadzone zostały wszystkie badania, których wyniki i dyskusja zamieszczane są w dalszych sekcjach. Opis ten powinien być na tyle dokładny, aby osoba czytająca go potrafiła wszystkie przeprowadzone badania samodzielnie powtórzyć w celu zweryfikowania ich poprawności. Przy opisie należy odwoływać się i stosować do opisanych w sekcji drugiej wzorów i oznaczeń, a także w jasny sposób opisać cel konkretnego testu. Najlepiej byłoby wyraźnie wyszczególnić (ponumerować) poszczególne eksperymenty tak, aby łatwo było się do nich odwoływać dalej.



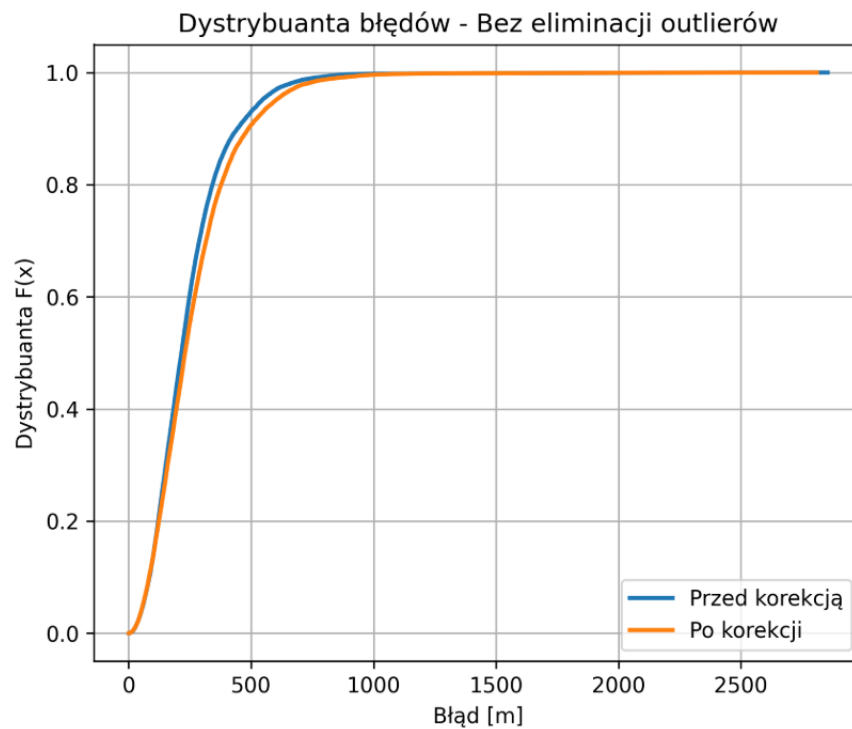
## 5. Wyniki



Rysunek 3: Rozkład błędów po korekcji



Rysunek 4: Rozproszenie błędów x, y



Rysunek 5: zmiana dystrybuanty

Tabela 1: Wagi cech sieci neuronowej

#	cecha	waga
1	data__tagData__linearAcceleration__y	0.159497
2	data__tagData__quaternion__x	0.150790
3	data__tagData__linearAcceleration__x	0.143811
4	data__tagData__quaternion__y	0.143303
5	data__tagData__magnetic__x	0.137812
6	data__tagData__quaternion__z	0.132061
7	data__tagData__magnetic__z	0.129842
8	data__tagData__quaternion__w	0.121032
9	data__tagData__pressure	0.120398
10	data__tagData__magnetic__y	0.111724
11	data__tagData__gyro__z	0.067959
12	data__tagData__linearAcceleration__z	0.067480
13	data__tagData__gyro__y	0.047070
14	data__tagData__gyro__x	0.041198

## 6. Dyskusja

Z uzyskanych wyników widzimy że rozkład błędów stał się bardziej "precyzyjny" (Rys. 4), oraz że zmniejszyła się gęstość występowania błędów w zakresie ok. 0 - 350 (Rysunek 3). Dystrybuanta ogólnie zmieniła się negatywnie z jedynie nieznaczną poprawą na zakresie podanym wyżej.

## 7. Wnioski

W tej, przedostatniej, sekcji należy zamieścić podsumowanie najważniejszych wniosków z sekcji poprzedniej. Najlepiej jest je po prostu wypunktować. Znów, tak jak poprzednio, najistotniejsze są wnioski o charakterze uniwersalnym.

## Literatura

- [1] Wikipedia contributors. "Neural network (machine learning)." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 2025.
- [2] Wikipedia contributors. (2025, May 25). *Ultra-wideband*. In Wikipedia, The Free Encyclopedia. 2025

Na końcu należy obowiązkowo podać cytowaną w sprawozdaniu literaturę, z której grupa korzystała w trakcie prac nad zadaniem.