Informatyka, studia dzienne, inż I st. semestr IV
Sztuczna inteligencja i systemy ekspertowe 2022/2023
Prowadzący: Krzysztof Lichy poniedziałek, 12:15

Data oddania: _____ Ocena: _____

Amadeusz Sitnicki 242524
Artur Surosz 242542

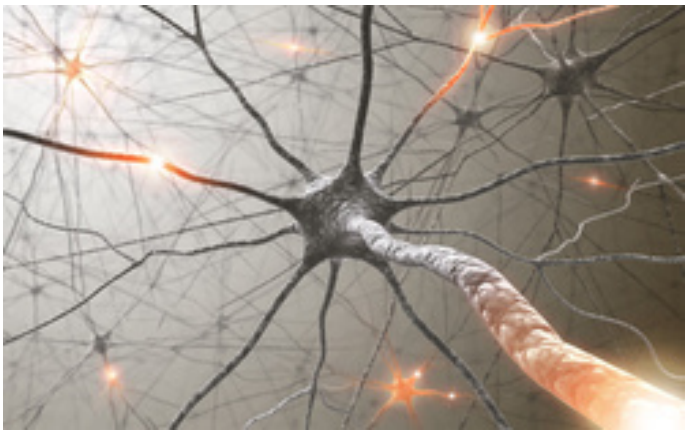
Zadanie 2: Poprawa lokalizacji UWB przy pomocy sieci neuronowej

1. Cel

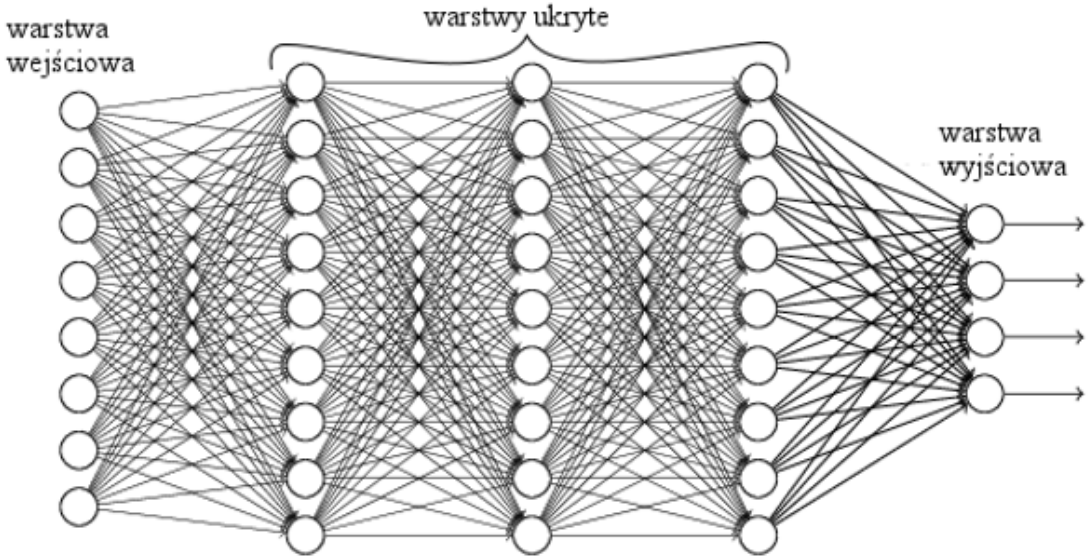
Celem wykonania zadania jest zapoznanie się z działaniem sieci neuronowych i technik uczenia maszynowego poprzez stworzenie programu, który dokonuje poprawy współrzędnych wyznaczonych przez zewnętrzny system, tak aby zmniejszyć błędy w wyznaczanej przez niego lokalizacji. W celu weryfikacji działania sieci konieczna jest wizualizacja danych w postaci wykresów dystrybuanty rozkładu błędu zarówno poprawianej lokalizacji wyznaczonej przez zewnętrzny system – jak i poprawionej lokalizacji wyznaczonej przez wytrenowany model uczenia maszynowego.

2. Wprowadzenie

Przy rozwijaniu algorytmów sztucznej inteligencji, twórca modelu matematycznego często wzoruje się na zjawiskach zachodzących naturalnie w biologii. W przypadku sieci neuronowych – model ten próbuje naśladować – i w pewnym stopniu odwierciedla reakcje mające miejsce w układzie nerwowym organizmów żywych. Jako model matematyczny sieć neuronowa jest kolekcją składającą się z :
— Warstwy danych wejściowych
— Pewnej liczby warstw neuronów pośredniczących - tzw. warstwy ukryte
— Warstwy neuronów wyjściowych
Graficznie można przedstawić przykładową sieć neuronową przy pomocy diagramu zamieszczonego na ilustracji. Każde połączenie posiada przyporząd-



Rysunek 1. Ilustracyjny obraz układu nerwowego organizmów żywych



Rysunek 2. Diagram ilustrujący model sieci neuronowej

kowaną wagę. Każdy neuron przetwarza dane w następujący sposób:
— Dla każdego połączenia przychodzącego odbierana jest wartość wejściowa
— Każda z wartości wejściowych wymnażana jest przez wagę danego połączenia
— Obliczana jest ważona suma wartości wejściowych
— Obliczana jest wartość nieliniowej funkcji aktywacji neuronu dla ważonej sumy wartości wejściowych
— Wartość funkcji aktywacji jest przekazywana dalej jako wartość wyjściowa neuronu

Jeśli sieć neuronowa posiada k_{we} wejść x_i, k_{wy} neuronów wyjściowych o wagach w_{nij}, funkcji aktywacji $f_n(x)$ i wartościach wyjściowych o_i oraz n warstw ukrytych o liczebnościach l_i z neuronami o wagach w_{ijk} oraz funkcjach aktywacji f_i o wartościach wyjściowych h_{ij} , to:
— $h_{0i} = f_0(\sum_{j=0}^{k_{we}} w_{0ij}x_j)$
— $h_{i+1,k} = f_i(\sum_{k=0}^{l_i} w_{ijk}h_{ik})$
— $o_i = f_n(\sum_{j=0}^{k_{wy}} w_{nij}h_{n-1,j})$

Aby sieć realizowała powierzone jej zadania, należy tak dobrać wartości wag sieci, aby określona funkcja straty dla sieci osiągnęła jak najmniejszą wartość. Funkcja straty może mieć następującą postać:

$$L(\vec{w}) = \frac{1}{m}\sum_{i=1}^{m}(o_i - d_i)^2,$$

gdzie d_i są pożądanymi wartościami wyjść sieci neuronowej z kolekcji m przykładowych danych dla sieci. Takie przykładowe dane dla sieci, które służą ustalaniu wag połączeń w sieci neuronowej noszą nazwę zbioru treningowego.

Dobór wag połączeń w sieci neuronowej jest równoważny zagadnieniu optymalizacji funkcji straty $L(\vec{w})$, która jest określona na przestrzeni o tylu wymiarach, ile jest połączeń w sieci neuronowej. Zatem funkcja straty jest odwzorowaniem:

$$L : \mathbb{R}^{k_{we}l_0+\sum_{i=0}^{n-2} l_i l_{i+1}+l_{n-1}k_{wy}} \mapsto \mathbb{R}.$$

W naszym rozwiązaniu użyjemy metody stochastycznego zejścia gradientowego (SGD) w celu stopniowego aktualizowania wartości wag i obniżania funkcji straty. Ciąg aktualizacji wag jest określony przez wzór rekurencyjny:

$$\vec{w}_{i+1} = \vec{w}_i + \vec{\nabla}L(\vec{w}_i)$$

Wartości pochodnych cząstkowych $\frac{\partial L}{\partial w_i}$ można obliczać dzięki zdefiniowaniu pewnego zbioru liczbowego: Uogólniony zbiór liczb dualnych rzędu r określamy jako zbiór liczb rzeczywistych rozszerzony o obiekty ε_i, $i = 1, 2, ..., r$. Obiekty ε_i spełniać muszą dwie zależności:
— $\varepsilon_i\varepsilon_j = 0$
— $\varepsilon_i \notin \mathbb{R} \wedge \varepsilon_j \notin \mathbb{R}$

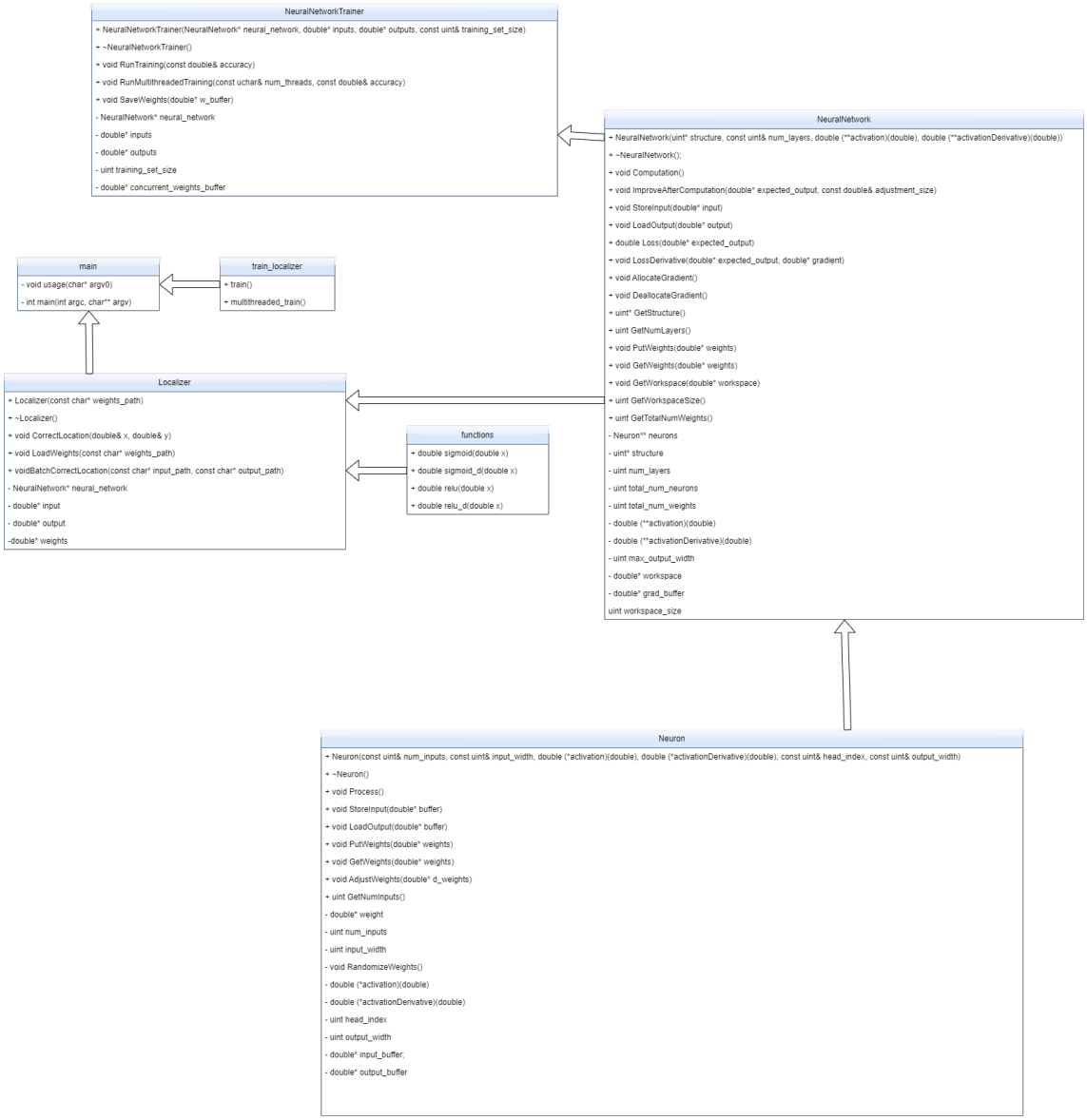
Na podstawie tej definicji da się udowodnić, że:

$$L(\vec{w} + \vec{\varepsilon}) = L(\vec{w}) + \sum_{i=1}^{r} \varepsilon_i \frac{\partial L}{\partial w_i} = L(\vec{w}) + \vec{\varepsilon} \cdot \vec{\nabla}L(\vec{w})$$

Powyższa równość pozwala na bezpośrednie wyznaczanie gradientu funkcji straty wprost z definicji tej funkcji i nietrudnej w implementacji arytmetyki na uogólnionych liczbach dualnych.

3. Opis implementacji

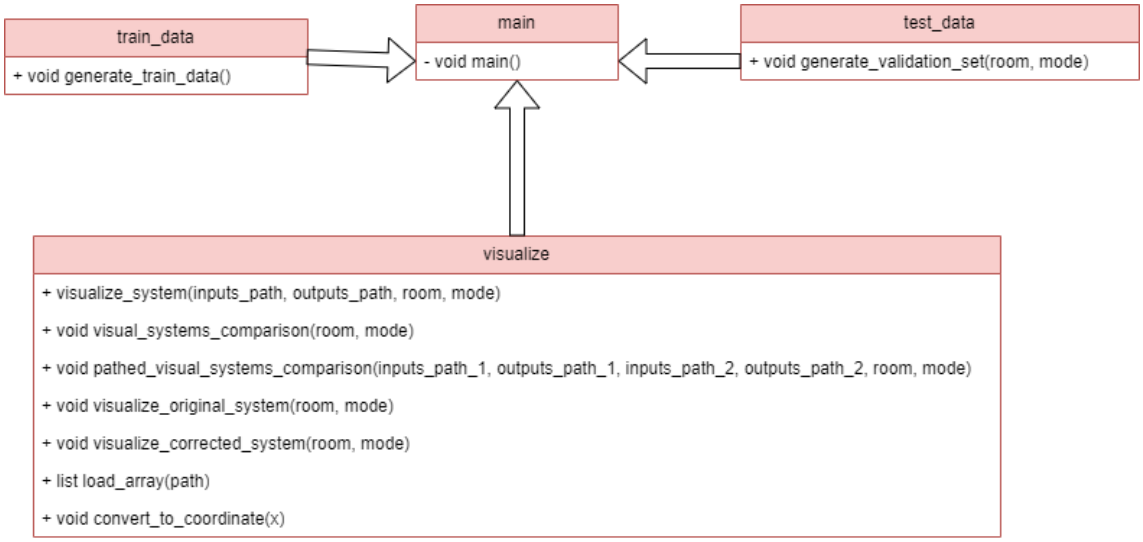
Program realizujący oraz trenujący sieć neuronową został stworzony na potrzeby zajęć laboratoryjnych przy pomocy języka C++ oraz kompilatora MinGW-W64 g++ 12.2.0 jako aplikacja konsolowa uruchamiana z poziomu wiersza poleceń powłoki systemowej. Implementacja sieci neuronowej od podstaw z trenowaniem w oparciu o uogólnione liczby dualne.



Rysunek 3. Schemat powiązań plików i klas rozwiązania C++

Program pomocniczy wizualizujący dystrybuanty rozkładów błędu lokalizacji wyznaczonej przez system a także służący do ekstrakcji danych m. in.

z plików xlsx, napisany został przy użyciu języka Python oraz ogólnodostępnych bibliotek.



Rysunek 4. Schemat powiązań plików rozwiązania Python

4. Materiały i metody

W celu wytrenowania sieci neuronowej:

— Ekstrakcja danych z plików .xlsx – wykonanie poniższych poleceń języka Python, odwołując się do pomocniczych skryptów test_data.py oraz train_data.py :

generate_train_data()
generate_validation_set(”f8”, ”1p”)
generate_validation_set(”f8”, ”1z”)
generate_validation_set(”f8”, ”2p”)
generate_validation_set(”f8”, ”2z”)
generate_validation_set(”f8”, ”3p”)
generate_validation_set(”f8”, ”3z”)
generate_validation_set(”f10”, ”1p”)
generate_validation_set(”f10”, ”1z”)
generate_validation_set(”f10”, ”2p”)
generate_validation_set(”f10”, ”2z”)
generate_validation_set(”f10”, ”3p”)
generate_validation_set(”f10”, ”3z”)
generate_validation_set(”f8”, ”random_1p”)
generate_validation_set(”f8”, ”random_2p”)
generate_validation_set(”f10”, ”random_1p”)
generate_validation_set(”f10”, ”random_2p”)

— Uruchomienie procesu trenującego sieć (wagi zapisywane są w katalogu weights):

...\localizer\ >loca train

— Po zakończeniu trenowania uruchomienie procesu wykonawczego sieci neuronowej dla wybranego pliku z wagami:

...\localizer\ >loca run weights\weights_0_00428953.bin

— Wizualizacja dystrybuant obu rozkładów błędów (przed i po filtrowaniu SN) – wykonanie poniższych poleceń języka Python, odwołując się do pomocniczego skryptu visualize.py:

visual_systems_comparison("f8", "1p")
visual_systems_comparison("f8", "1z")
visual_systems_comparison("f8", "2p")
visual_systems_comparison("f8", "2z")
visual_systems_comparison("f8", "3p")
visual_systems_comparison("f8", "3z")
visual_systems_comparison("f10", "1p")
visual_systems_comparison("f10", "1z")
visual_systems_comparison("f10", "2p")
visual_systems_comparison("f10", "2z")
visual_systems_comparison("f10", "3p")
visual_systems_comparison("f10", "3z")
visual_systems_comparison("f8", "random_1p")
visual_systems_comparison("f8", "random_2p")
visual_systems_comparison("f10", "random_1p")
visual_systems_comparison("f10", "random_2p")

5. Wyniki

Zaprojektowana i zrealizowana sieć neuronowa posiada:

— 2 wejścia
— 4 warstwy neuronowe
— liczebności warstw neuronowych to odpowiednio 20, 10, 5, 2
— sieć posiada 2 neurony wyjściowe
— dla każdego neuronu funkcja aktywacji ustalona jest jako funkcja sigmoidalna $\sigma(x) = -1 + \frac{2}{1+e^{-x}}$

```
uint* structure = new uint[5]{2, 20, 10, 5, 2};
uint num_layers = 4;

double (**activation)(double) = new RealFunction[4]{sigmoid, sigmoid, sigmoid, sigmoid};
double (**activationDerivative)(double) = new RealFunction[4/*2*/]{sigmoid_d, sigmoid_d, sigmoid_d, sigmoid_d};
this->neural_network = new NeuralNetwork(structure, num_layers, activation, activationDerivative);
```

Rysunek 5. Konfiguracja sieci neuronowej w kodzie źródłowym

Dodatkowo przed wejściem do sieci, dane podlegają transformacji funkcją:

$$T(x) = -1 + \frac{2}{1 + e^{-0.001x}}$$

Zaś po wyjściu z sieci, dane są przekształcane funkcją odwrotną:

$$T^{-1}(x) = -1000 \ln(\frac{2}{1+x} - 1)$$

Przypomnijmy, że w naszym rozwiązaniu używamy metody stochastycznego zejścia gradientowego (SGD) w celu stopniowego aktualizowania wartości wag i obniżania funkcji straty. Ciąg aktualizacji wag jest określony przez wzór rekurencyjny:

$$\vec{w}_{i+1} = \vec{w}_i + \vec{\nabla} L(\vec{w}_i)$$

Wartości pochodnych cząstkowych $\frac{\partial L}{\partial w_i}$ można obliczać dzięki zdefiniowaniu pewnego zbioru liczbowego: Uogólniony zbiór liczb dualnych rzędu r określamy jako zbiór liczb rzeczywistych rozszerzony o obiekty ε_i, $i = 1, 2, ..., r$. Obiekty ε_i spełniać muszą dwie zależności:

— $\varepsilon_i \varepsilon_j = 0$

— $\varepsilon_i \notin \mathbb{R} \wedge \varepsilon_j \notin \mathbb{R}$

Na podstawie tej definicji da się udowodnić, że:

$$L(\vec{w} + \vec{\varepsilon}) = L(\vec{w}) + \sum_{i=1}^{r} \varepsilon_i \frac{\partial L}{\partial w_i} = L(\vec{w}) + \vec{\varepsilon} \cdot \vec{\nabla} L(\vec{w})$$

Powyższa równość pozwala na bezpośrednie wyznaczanie gradientu funkcji straty wprost z definicji tej funkcji i nietrudnej w implementacji arytmetyki na uogólnionych liczbach dualnych.

W wyniku trenowania sieci, minimalna wartość funkcji straty jaką osiągnięto wynosi $L = 0.00428953$

Na ilustracjach widoczne jest porównanie dystrybuant rozkładu (CDF) błędu lokalizacji – porównanie dla systemu UWB oraz dla systemu wspomaganego siecią neuronową.

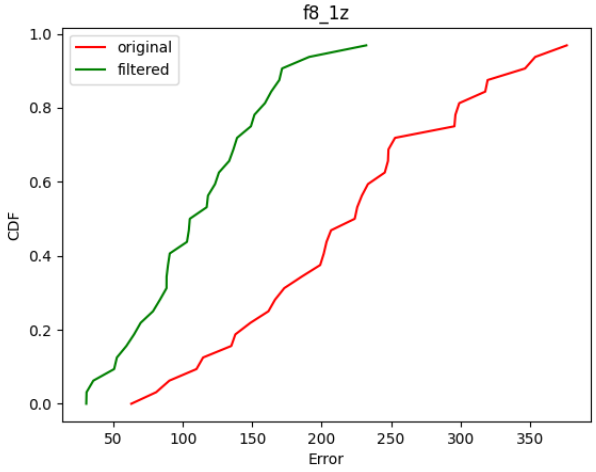
6. Dyskusja

6.1. Wykorzystana metoda uczenia

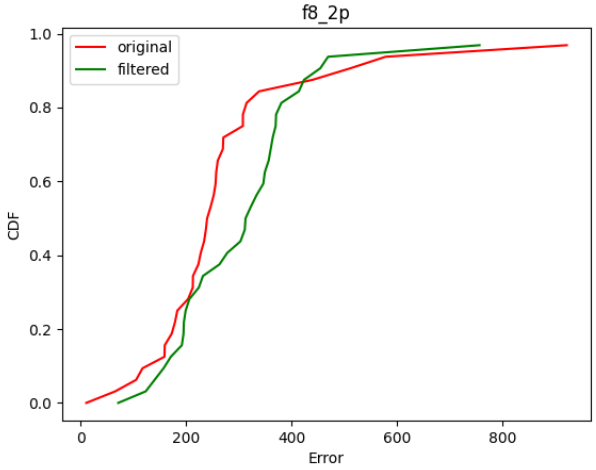
Wykorzystana metoda uczenia pozwala wykorzystać własności arytmetyki na uogólnionym zbiorze liczb dualnych. Dzięki temu nie trzeba implementować metody propagacji wstecznej. W ten sposób realizacja zadania jest ułatwiona, a kod – krótszy, mimo że sieć neuronowa została zaimplementowana od podstaw z wykorzystaniem języka C++.

6.2. Zrównoleglanie obliczeń

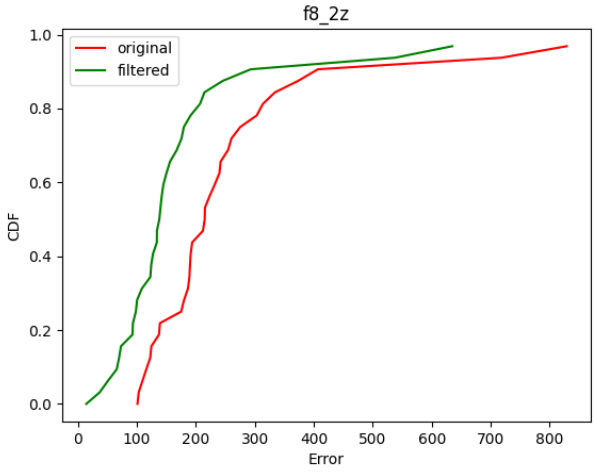
Implementacja algorytmu uczenia sieci w ten sposób ma jednak pewien problem, który polega na gorszym przystosowaniu algorytmu do zrównoleglania. Nie jest jednak wykluczone, że istnieje optymalizacja metody, pozwalająca na zrównoleglenie trenowania modelu – zbadanie takiego zoptymalizowanego wariantu tej metody pozwoliłoby przywrócić szybkości znajdowania



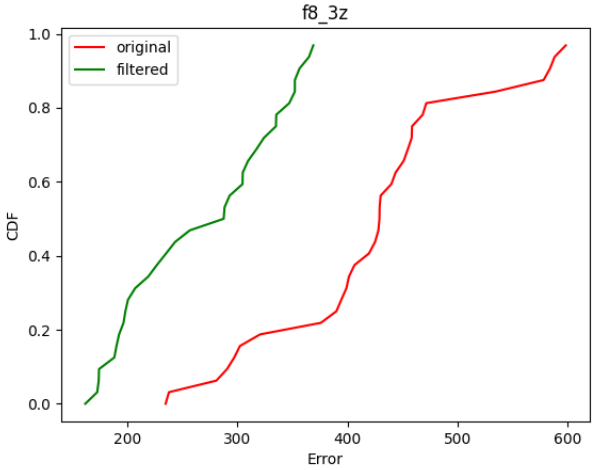
Rysunek 6. Porównanie dystrybuant dla f8_1z



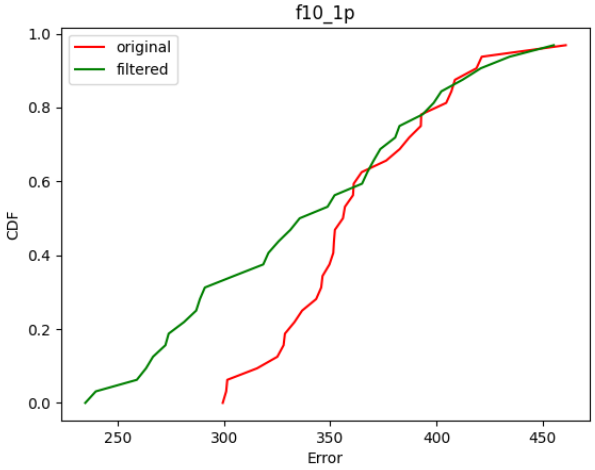
Rysunek 7. Porównanie dystrybuant dla f8_2p



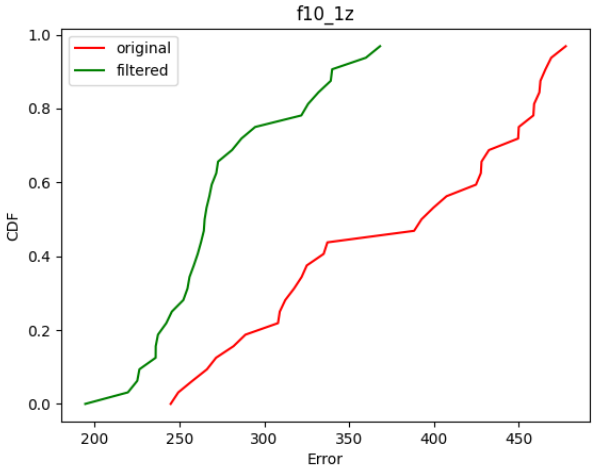
Rysunek 8. Porównanie dystrybuant dla f8_2z



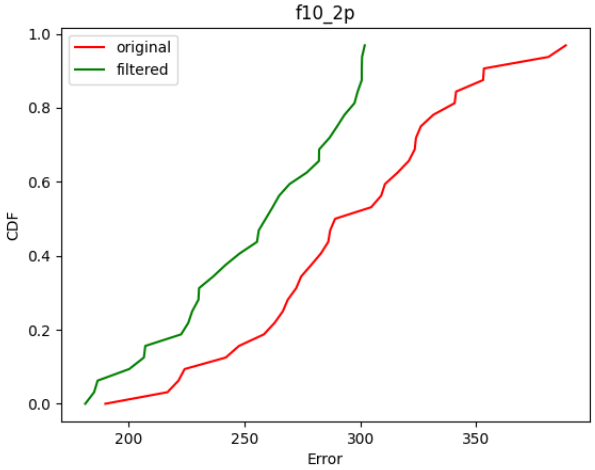
Rysunek 9. Porównanie dystrybuant dla f8_3z



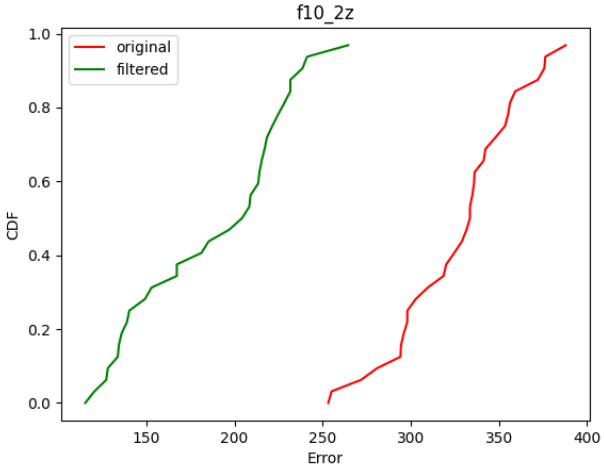
Rysunek 10. Porównanie dystrybuant dla f10_1p



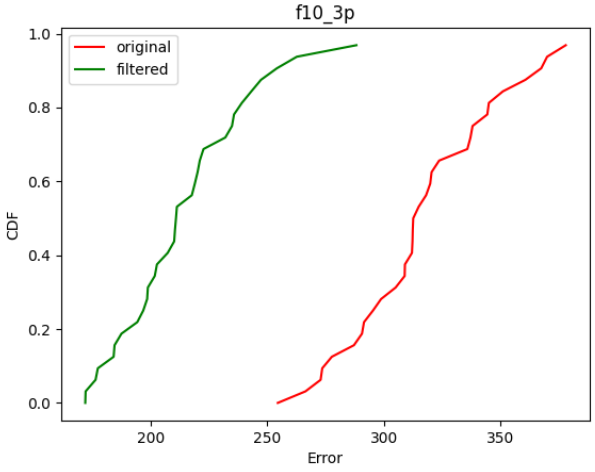
Rysunek 11. Porównanie dystrybuant dla f10_1z



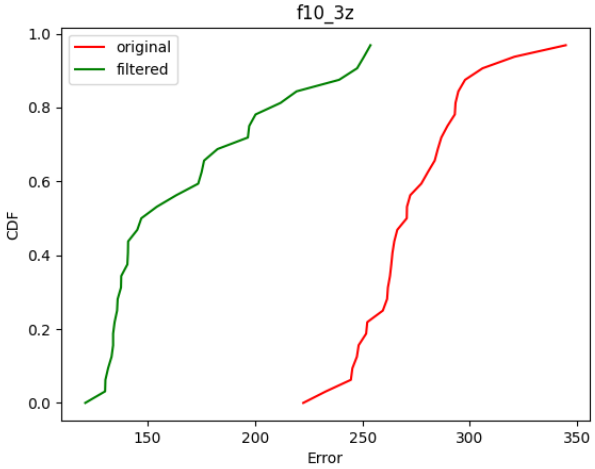
Rysunek 12. Porównanie dystrybuant dla f10_2p



Rysunek 13. Porównanie dystrybuant dla f10_2z



Rysunek 14. Porównanie dystrybuant dla f10_3p



Rysunek 15. Porównanie dystrybuant dla f10_3z

optymalnych wag sieci neuronowej, znane z popularnych bibliotek deep learning wykorzystujących przetwarzanie współbieżne.

6.3. Skuteczność sieci neuronowej

Dla pewnych torów (np. f8_2p) obliczenia dokonane przez sieć mają mniejszą skuteczność. Możliwe, że taki stan rzeczy jest spowodowany doborem funkcji transformującej dane przed wejściem i po wyjściu z sieci neuronowej - zmienność wartości $T(x)$ (por. rozdział 5) jest w przypadku znajdowania się systemu pomiarowego skrajnie z prawej strony znacząco ograniczona i wykonuje jedynie niewielkie wahania w okolicach wartości 1. Taki efekt może być przyczyna zjawiska i powodować mniejszą skuteczność sieci dla torów ruchu o środku geometrycznym skrajnie z prawej strony obszaru, na którym odbywało się uczenie sieci.

7. Wnioski

— Sieci neuronowe i uczenie maszynowe są bardzo dobrym narzędziem ogólnego przeznaczenia do przewidywania i redukcji błędów pomiarowych
— Skuteczność sieci zależy od systematyczności błędu – jeśli błąd będzie nosił znamiona losowości osobliwej – wykrywanie błędów będzie utrudnione
— Przekształcenia wstępne i końcowe mogą znacząco zaburzyć lub poprawić skuteczność obliczeń przeprowadzanych przez sieć neuronową
— Manipulowanie architekturą sieci, daje szansę na dalsze zmniejszanie funkcji straty, gdy zawiodła wcześniej testowana architektura.
— Nadzorowane uczenie sieci neuronowych jest zadaniem stosunkowo wymagającym obliczeniowo dla sprzętu co wiąże się z wyjątkowo długim czasem oczekiwania przez konstruktora sieci na znalezienie optymalnych parametrów sieci. Pomocą w tej kwestii może być zastosowanie przetwarzania współbieżnego – najlepiej z użyciem dedykowanych GPU lub jednostek przetwarzających TPU – szczególnie w przypadku, gdy ma się do

czynienia z dużo większymi zbiorami danych niż te obecne w tym projekcie

ppauza na przykład przetwarzanie obrazów przy użyciu sieci neuronowych.

.

Literatura

[1] *Sieci neuronowe - Wikipedia*, dostępny online.

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