Informatyka, studia dzienne, inż I st. semestr V

Sztuczna inteligencja i systemy ekspertowe 2021/2022

Prowadzący: dr. hab. inż. Piotr Lipiński czwartek, 15:45

Data oddania: _____ Ocena: _____

Wiktor Bechciński 229840
Kamil Budzyn 229850

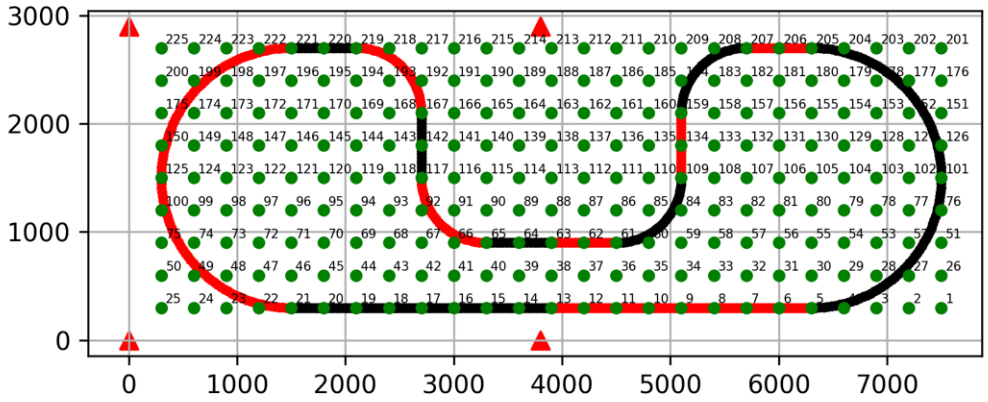
Zadanie 2: Poprawa lokalizacji UWB przy pomocy sieci neuronowych

1. Cel

Zadanie polega na zaprojektowaniu i zaimplementowaniu sieci neuronowej, która pozwoli na korygowanie błędów uzyskanych z systemu pomiarowego. Do nauki sieci neuronowej należy wykorzystać dane pomiarowe.

2. Wprowadzenie

Nazwa sieci neuronowych oraz struktura są wzorowane na działaniu ludzkiego mózgu. Wzorują się na działaniu biologicznych neuronów, które się komunikują między sobą. Sztuczne sieci neuronowe składają się z warstw węzłów, które obejmują warstwę wejściowa (input layer), warstwę wyjściową (output layer) oraz zazwyczaj wiele warstw ukrytych (hidden layers). Każdy z węzłów jest połączony z innymi za pomocą wag, które nieustannie modyfikuje w procesie nauki. Do szkolenia symulowanych sieci neuronowych należało wykorzystać dane pomiarowe statyczne, natomiast do weryfikacji poprawności uzyskanych wyników należało wykorzystać dane dynamiczne.



Rys. 1. Tor ruchu robota wraz z punktami statycznymi.

Zaimplementowaliśmy wielowarstwowy perceptron (MLP) który uczy sieć przy pomocy propagacji wstecznej, dzięki czemu jest on w stanie wygenerować dane w dowolnym zakresie. Wykorzystuje on błąd kwadratowy jako funkcję straty, a wyjściem jest zestaw wartości ciągłych.

3. Opis implementacji

Program został napisany w języku Python przy wykorzystaniu środowiska Jupyer Notebook. Zaimplementowanie sieci neuronowej umożliwiły biblioteki Tensorflow oraz Keras (który używa Tensorflow jako swojego backendu). Biblioteka Tensorflow została zbudowana ze wsparciem procesora graficznego (NVIDIA CUDA) w celu optymalizacji działania programu. Do treningu naszej sieci wykorzystaliśmy wszystkie pliki pomiarów statystycznych, a konkretniej dane z kolumn "data_coordinates_x" oraz "data_coordinates_y", oraz ich kolukmny referencyjne. Do weryfikacji poprawności uzyskanych wyników wykorzystaliśmy dane dynamiczne. Weryfikację skuteczności działania naszej

sieci porównaliśmy dystrybuanty błędu danych testowych oraz dystrybuanty błędu lokalizacji. Liczbę neuronów w sieci oszacowaliśmy na podstawie prób i błędów wspomagając się własnymi doświadczeniami oraz różnymi artykułami, lecz niestety nie ma żadnego precyzującego ich ilość wzoru. Projekt podzieliliśmy na dwa główne pliki: READandPREP.jpynb oraz LEARNandRESULT.jpynb. Gdzie pierwszy odpowiada za odczytanie i przygotowanie danych testowych, a drugi z nich za naukę sieci neuronowej oraz przygotowanie wyników. Stworzona sieć wykorzysuje do korekcji błędu pomiarowego dwa bloki.

Pierwszy:

- Liczba warstw sieci - 8
- warstwa 1 - 64
- warstwa 2 - 128
- warstwa 3 - 256
- warstwa 4 - 512
- warstwa 5 - 128
- warstwa 6 - 128
- warstwa 7 - 128
- warstwa 8 - 64

Drugi:

- Liczba warstw sieci - 9
- warstwa 1 - 128
- warstwa 2 - 256
- warstwa 3 - 512
- warstwa 4 - 512
- warstwa 5 - 512
- warstwa 6 - 256
- warstwa 7 - 256
- warstwa 8 - 128
- warstwa 9 - 64

Model posiada 1,208,582 parametrów. Wszędzie zastosowaliśmy funkcję aktywacji typu ELU. Jako inicjalizację neuronów użyliśmy inicjalizera He_normal który dobiera próbki z obciętego rozkładu normalnego wyśrodkowanego na 0. Liczba wag jest na tyle duża że postanowiliśmy nie umieszczać jej w sprawozdaniu, za to są one zapisane w pliku trained_weights.

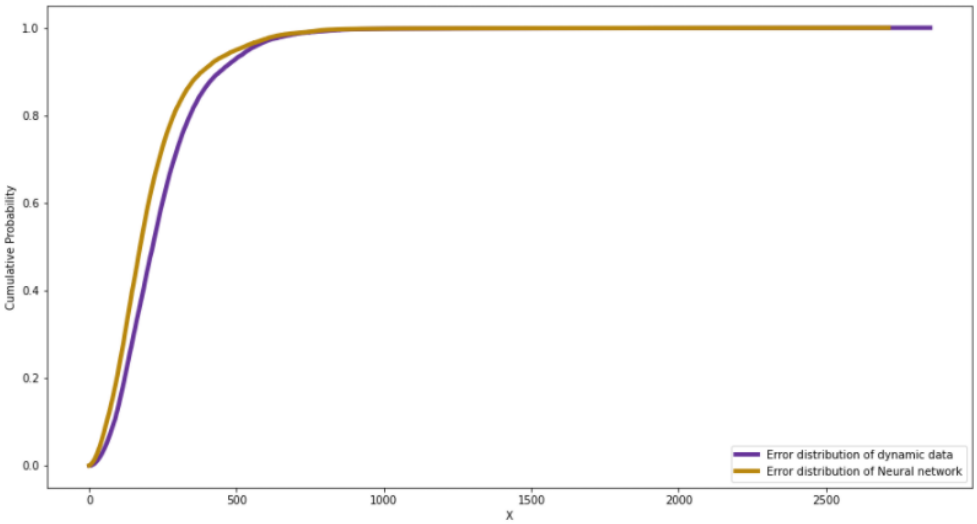
4. Materiały i metody

Zastosowana przez nas funkcja ELU (Exponential Linear Unit) to funkcja, która ma tendencję do szybszego zbiegania do zera i uzyskiwania dokładniejszych wyników. W procesie uczenia sieci wykorzystaliśmy algorytm ADAM (Adaptive moment estimation). Algorytm ten jest prosty w implementacji, wydajny obliczeniowo, wymaga niewielkiej ilości pamięci i jest odpowiedni w sytuacjach z dużymi zestawami danych i parametrami. Adam to rozszerzona wersja stochastycznego zejścia gradientowego. W tym algorytmie optymalizacji wykorzystywane są średnie bieżące zarówno aktualnych gradientów, ale również jego poprzednich gradientów.

Jako algorytm uczenia sieci neuronowej wykorzystaliśmy propagację wsteczną. Na ogól przy regresji jako funkcję straty używa się błędu średniokwadratowego, jednakże w zadaniu otrzymane dane mają bardzo dużo elementów odstających od siebie, więc w tym przypadku znacznie lepszym rozwiązaniem będzie funkcja Hubera, z biblioteki keras.optimizers, która jest funkcją straty mniej wrażliwą na wartości odstające w danych niż kwadratowa strata błędu.

5. Wyniki

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



Rys. 2 Wykres dystrybuant błędów pomiaru.

6. Dyskusja

Powyższy wykres przedstawia porównanie dystrybuanty błędu danych testowych oraz dystrybuanty błędu lokalizacji uzyskanej w wyniku zastosowania sieci neuronowej. Widzimy tutaj niewielką, ale zdecydowanie zauważalną poprawę, więcej pomiarów ma mniejsze błędy co może świadczyć o tym że zastosowana przez nas sieć neuronowa spełnia swoją funkcję. Błąd średniokwadratowy zmniejszył się z 43635 do 32878.

7. Wnioski

Zastosowana sieć neuronowa pozwala uzyskać widocznie lepsze wyniki dystrybuanty błędu. Zastosowanie sieci neuronowej pozwoliło zmniejszyć średni błąd kwadratowy do wartości 32878.

Literatura

[1] Model Construction `https://d2l.ai/chapter_deep-learning-computation/model-construction.html`
[2] Actavation Functions `https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html`
[3] Layers Initializers `https://keras.io/api/layers/initializers/`
[4] Keras Dokumentacja `https://www.tensorflow.org/api_docs/python/tf/keras`