Filip Kobierski 242336, Piotr Polakowski 247768, Jakub Samek 247781

Sztuczna Inteligencja i Systemy ekspertowe

Zadanie 2: Poprawa lokalizacji UWB przy pomocy sieci neuronowych

7 czerwca 2024

1. Cel

Zaprojektowanie i zaimplementowanie sieci neuronowej, która pozwoli na korygowanie błędów uzyskanych z systemu pomiarowego. Sieć ma być uczona na danych statycznych a testowana na dynamicznych.

2. Opis architektury sieci neuronowej

Naszą sieć zaprojektowaliśmy tak aby miała trzy warstwy. Więcej informacji o niej przedstawia tablica 1. Do tego zadania zdecydowaliśmy się użyć biblioteki Tensorflow, do której będziemy odwoływać się jako `ts`. W naszej sieci używamy `ts.keras` jako interfejsu, znanego nam z Komputerowej Analizy danych `sklearn`, a konkretnie `sklearn.metrics.mean_squared_error()` jako funkcję kosztu. Warto też zaznaczyć, że nasza sieć została skompilowana z optymalizatorem `tf.keras.optimizers.Adam`.

Tablica 1. Warstwy sieci

Nr. warstwy	Liczba neuronów	Funkcja aktywacyjna
1	10	ReLU
2	5	eLU
3	2	SeLU

2.1. Testy i uczenie

Podczas testów ilości epok używaliśmy `keras.src.callbacks.EarlyStopping`. Przy jej pomocy, metodą prób i błędów, ustaliliśmy, że optymalna ilość epok to 29.

Uczenie realizowaliśmy na trzydziestodwuelementowych paczkach zestwu treningowego.

Nasza sieć na wejściu przyjmuje wartości z kolumn:

1. `data_coordinates__x`

2. `data_coordinates__y`
3. `reference__x`
4. `reference__y`

oraz zwraca:

1. wagi neuronów
2. wykres porównujący dystrybuanty błędów
3. plik arkusza kalkulacyjnego Microsoft Excel `xlsx` zawierający pojedyńczą kolumnę z wartościami dystrybuanty błędu pozyskanych w filtracji siecią neuronową.

Pozwolimy sobie tutaj zauważyć, że naszym zdaniem format `xlsx` jest niepotrzebne skomplikowany do tego zadania. Lepiej sprawdziłoby się tu prostsze `csv`, a jeśli z jakiegoś powodu potrzebne są funkcje arkusza kalkulacyjnego, to format `ods` zdaje się nam być lepszym wyborem.

Do trenowania sieci (statycznie) użyliśmy materiałów udostępnionych nam na stronie przedmiotu. Testy były oczywiście przeprowadzane na danych dynamicznych.

2.2. Reszta bibliotek

Poza wspomnianymi wyżej bibliotekami użyliśmy również:

— `numpy`
— `pandas`
— `matplotlib`
— `re`
— `math`
— `glob`

3. Wyniki

Wagi poszczególnych neuronów prezentują rysunki 1 do 3, podczas gdy efektywność naszej sieci podsumowuje rysunek 4. Na nim oś x oznacza błąd pomiaru, a oś y to prawdopodobieństwo wystąpienia zmiennej losowej, której wartości odległości od oczekiwanej (rzeczywistej) jest mniejsza lub równa wartości na osi odcięych.

```
[[-0.00620399 -0.33701584  0.6889706  -0.44631428 -0.06090368  0.57241464
  -0.10857992  0.65317243  0.10749532  0.05042003]
 [-0.7707375  -0.2427195   0.06861603 -0.38940647  0.37916997  0.5108574
   0.7250441  -0.24520046 -1.294854    0.60276437]]
```

Rysunek 1. Wagi neuronów w pierwszej warstwie

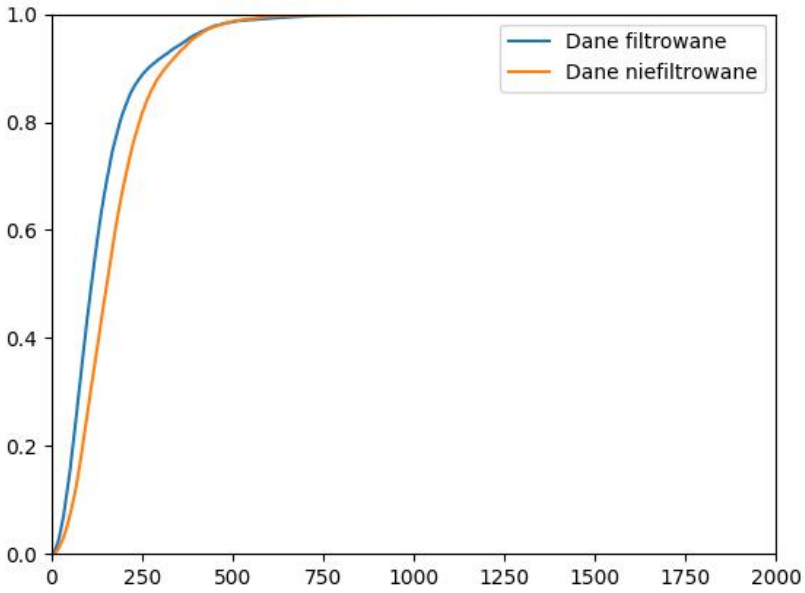
```
[[ 0.7025143   0.37028858 -1.0343028   0.09549365  0.36406264  0.54510254]
 [-0.3788414   0.30427647 -0.57159966  0.08190275 -0.28644514  0.07091594]
 [ 0.41229618  0.6431695   0.37302133 -0.36332688 -0.0436207   0.52235734]
 [-0.07386542 -0.43389687 -0.36127123 -0.08869618 -0.17052473 -0.17362055]
 [-0.12793356 -0.44255143 -0.39562485  0.28131926  0.14053969  0.12881356]
 [ 0.37238884  0.20605299  0.6697696   0.4040027  -0.12035445 -0.41287646]
 [-1.0072443   0.2322274   0.05039882 -0.328088   -0.71282667  0.18791613]]
```

```
 [-0.242283   -0.03349353  0.34129822 -0.16205645 -0.14926694  0.5331037 ]
 [-0.34535378 -0.01738667  0.3936075  -0.85495466  0.08866001 -0.3327119 ]
 [ 0.46039295  0.6313022  -0.801915    0.00364363 -0.27802998 -0.28862533]]
```

Rysunek 2. Wagi neuronów w drugiej warstwie

```
[[-0.25792632 -0.7045216 ]
 [ 0.8623442  -0.6185966 ]
 [ 0.21600294  0.4805853 ]
 [ 0.1858656  -0.4335449 ]
 [-0.2572863   0.02113466]
 [ 0.87135744  0.2554335 ]]
```

Rysunek 3. Wagi neuronów w trzeciej warstwie

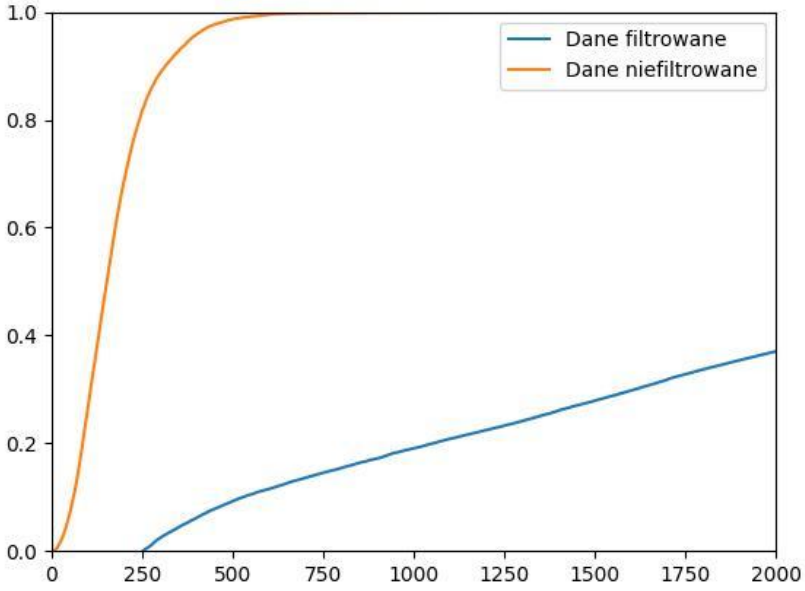


Rysunek 4. Wykres

4. Dyskusja

Przy tworzeniu naszej sieci testowaliśmy jak sprawdzają się różne funkcje aktywacji:

— `sigmoid`
— `elu`
— `relu`
— `selu`
— `tanh`
— `softmax`



Rysunek 5. Działanie funkcji `softmax`

Ta ostatnia okazała się najgorsza do tego zadania, co poazuje obrazek 5.

Pozostałe funkcje aktywacji były ze sobą porównywalnie i znacząco lepsze od softmax'a.

Podczas trenowania sieci zaobserwowaliśmy zjawisko przetrzenowania (*overfitting*) – nasza sieć szkoliła zaczynała sobie lepiej radzić na statycznych danych treningowych, lecz gorzej na dynamicznych danych testowach. Aby sobie z tym poradzić, zgodnie z wiedzą z wykładu, zatrzymywaliśmy wtedy uczenie.

5. Wnioski

Wykres 4 pokazuje, że dane po wyjściu z naszej sieci neuronowej są bardziej dokładne, czyli sieć spełnia swoje zadanie. Kolejnym pytaniem które trzeba sobie zadać jest *Czy to się opłaca?*. My go nie zadajemy bo nie musimy.

Podczas tworzenia sieci neuronowej należy wziąć pod uwagę wiele czynników, takich jak liczba neuronów i warstw, a także liczba epok. Ważne jest również dobranie odpowiedniej funkcji aktywacji, gdyż nie każda funkcja sprawdzi się w każdej sieci.

Ponadto, należy zwrócić uwagę, że funkcja aktywacji używana w sieci neuronowej może mieć duży wpływ na jej działanie. Niektóre typowe funkcje aktywacji obejmują funkcje sigmoid, tanh i ReLU, a najlepszy wybór zależy od konkretnej aplikacji i danych używanych.