Informatyka, studia dzienne, inż I st. semestr VI

Sztuczna inteligencja i systemy ekspertowe 2019/2020

Prowadzący: dr inż. Krzysztof Lichy wtorek, 12:15

Data oddania: _____ Ocena: _____

Konrad Jaworski 216782
Bartłomiej Jencz 216783

Zadanie 2: Sieć neuronowa służąca do korygowania pomiaru systemu lokalizacji

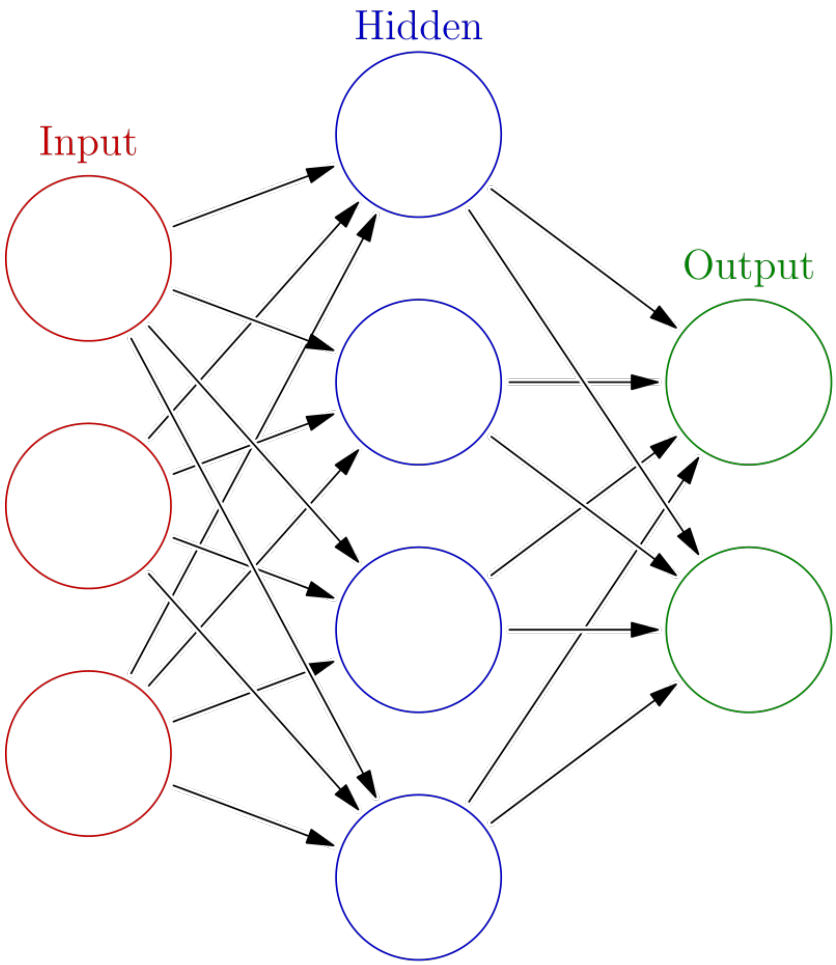
1. Cel

Celem zadania 2 jest zaprojektowanie i zaimplementowaniu sieci neuronowej, która pozwoli na korygowanie błędów uzyskanych z systemu pomiarowego robota jeżdżącego po torze.

2. Wprowadzenie

2.1. Sieć neuronowa

Sieć neuronowa [6] (sztuczna sieć neuronowa) – ogólna nazwa struktur matematycznych i ich programowych lub sprzętowych modeli, realizujących obliczenia lub przetwarzanie sygnałów poprzez rzędy elementów przetwarzających, zwanych sztucznymi neuronami, wykonujących pewną podstawową operację na swoim wejściu. Oryginalną inspiracją takiej struktury była budowa naturalnych neuronów, łączących je synaps, oraz układów nerwowych, w szczególności mózgu.



Rysunek 1. Sztuczna sieć neuronowa to połączona grupa węzłów, zainspirowana uproszczeniem neuronów w mózgu. Tutaj każdy okrągły węzeł reprezentuje sztuczny neuron, a strzałka reprezentuje połączenie między wyjściem jednego sztucznego neuronu a wejściem innego.

2.1.1. Sieci jednokierunkowe

Sieci jednokierunkowe [6] to sieci neuronowe, w których nie występuje sprzężenie zwrotne, czyli pojedynczy wzorzec lub sygnał przechodzi przez każdy neuron dokładnie raz w swoim cyklu.

W bardziej zaawansowanych rozwiązaniach stosuje się funkcje przejścia. Najpopularniejszą klasę funkcji stosowanych w sieciach neuronowych stanowią funkcje sigmoidalne, np. tangens hiperboliczny.

Sieci jednokierunkowe dzielą się na jednowarstwowe, dwuwarstwowe i wielowarstwowe. Sieci jednowarstwowe mogą rozwiązać jedynie wąską klasę problemów. Sieci dwu i wielowarstwowe mogą rozwiązać znacznie szerszą klasę i są pod tym względem równoważne, jednak stosuje się do nich inne algorytmy uczenia (dla wielowarstwowych są one prostsze).

3. Opis implementacji

Sieć neuronowa została napisana w języku Python z wykorzystaniem wysokopoziomowej biblioteki do tworzenia sieci neuronowych jaką jest TensorFlow. W niej skorzystaliśmy z Functional API oferowaną przez bibliotekę KERAS.
W programie została zaimplementowana klasa **NEURAL_NETWORK_SETTINGS** [**config.py**] pozwalająca w łatwy sposób manipulować parametrami sieci neuronowej oraz innych parametrów programu (między innymi czy program ma wykonać normalizację danych)

3.1. Budowa modelu

Główna funkcja definująca budowę modelu sieci neuronowej z wykorzystaniem **Functional API** z pakietu KERAS[1]

```
1  def build_model(train_dataset):
2      _model = keras.Sequential([
3          layers.Dense(NEURAL_NETWORK_SETTINGS.NUMBER_OF_INPUTS, activation='relu',
4                       input_shape=[2,], name='InputLayer'),
5          layers.Dense(NEURAL_NETWORK_SETTINGS.NEURONS_FIRST_LAYER, activation='relu'),
6          layers.Dense(NEURAL_NETWORK_SETTINGS.NEURONS_SECOND_LAYER, activation='relu'),
7          layers.Dense(NEURAL_NETWORK_SETTINGS.NUMBER_OF_OUTPUTS, name='OutputLayer')
8      ])
9
10     optimizer = tf.keras.optimizers.RMSprop(0.001)
11
12     _model.compile(loss='mse',
13                    optimizer=optimizer,
14                    metrics=['mae', 'mse'])
15     return _model
```

Optymalizator **RMSprop** [3] polega na dzieleniu współczynniku uczenia się dla wagi według bieżącej średniej z wielkości ostatnich gradientów dla tej wagi.

$$MeanSquare(w,t) = 0.9MeanSquare(w,t-1) + 0.1(\frac{\partial E}{\partial w}(t))^2 \qquad (1)$$

Jako funkcją straty wykorzystujemy średni błąd kwadratowy (*z ang. MSE*) definiowana w KERAS jako

```
loss = square(y_true - y_pred)
```

Do oceny osiągów sieci neuronowej wykorzystaliśmy dwie metryki **średni błąd bezwzględny** (MAE) i **średni błąd kwadratowy** (MSE)

3.1.1. Średni błąd bezwzględny [5]

Średni błąd bezwzględny (MAE) jest miarą błędów między sparowanymi obserwacjami wyrażającymi to samo zjawisko. Przykłady Y w porównaniu

z X obejmują porównania przewidywanego i obserwowanego, czasu późniejszego w stosunku do czasu początkowego oraz jednej techniki pomiaru w porównaniu z alternatywną techniką pomiaru. MAE oblicza się jako:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}. \tag{2}$$

3.1.2. Średni błąd kwadratowy [4]

Jeśli wektor prognozy n jest generowany z próbki n punktów danych dla wszystkich zmiennych, a Y jest wektorem obserwowanych wartości przewidywanej zmiennej, z $\tilde{Y}_i$ będący przewidywanymi wartościami (np. z metody najmniejszych kwadratów), wtedy średni błąd kwadratowy w próbce predyktora jest obliczany jako

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{3}$$

3.2. Opis architektury sieci neuronowej

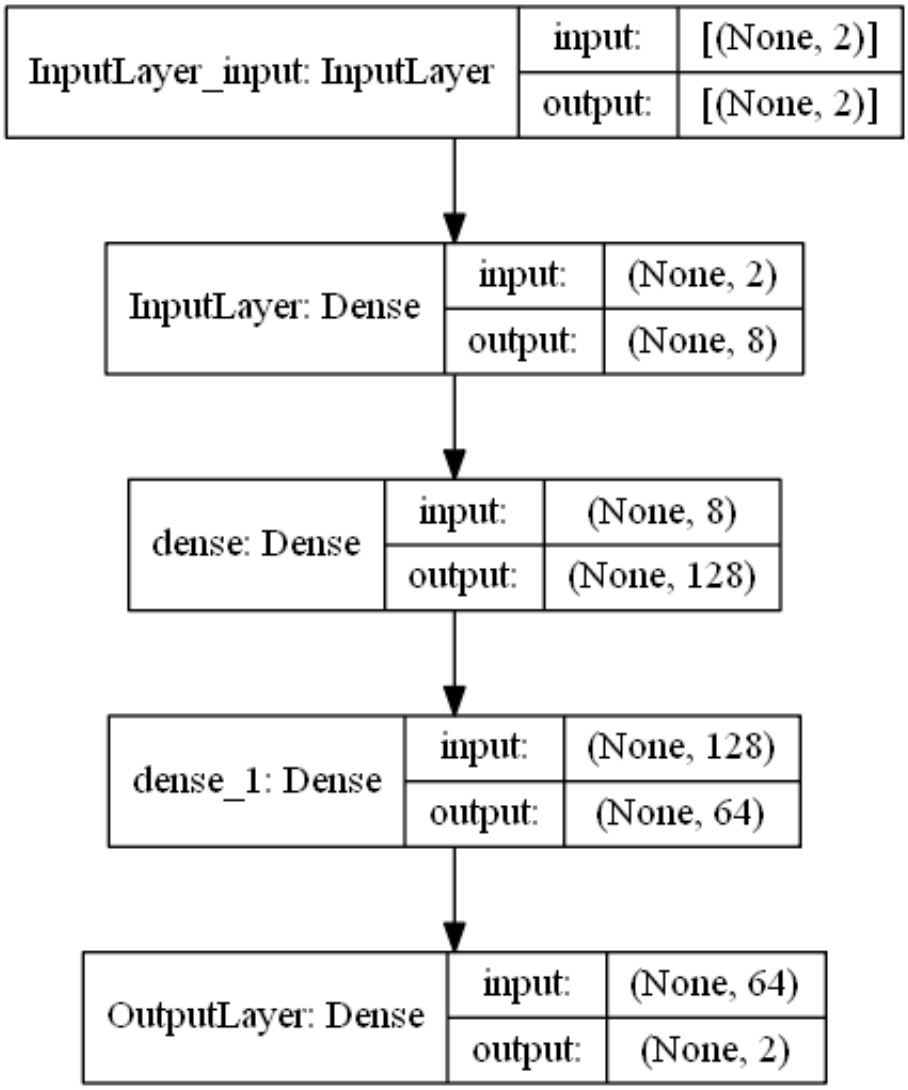
Podsumowanie jednego z lepszych modeli wywołanego poleceniem

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
InputLayer (Dense)           (None, 8)                 24
_________________________________________________________________
dense (Dense)                (None, 128)               1152
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
OutputLayer (Dense)          (None, 2)                 130
=================================================================
Total params: 9,562
Trainable params: 9,562
Non-trainable params: 0
_________________________________________________________________
```

3.2.1. Liczba warstw sieci neuronowej

Autorzy uznali że najlepszą liczbą warstw dla tego eksperymentu jest ilość 2 warstw ukrytych. Oznaczone są one jako **dense** oraz **dense_1**



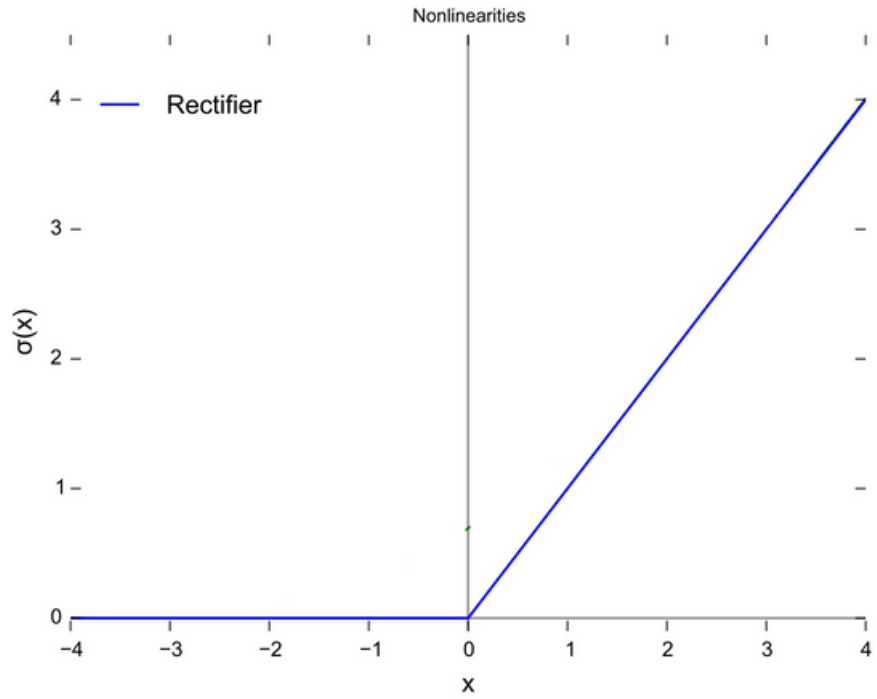
Rysunek 2. Model jako graf

3.2.2. Liczebność neuronów w poszczególnych warstwach

Autorzy uznali że odpowiednią ilość próbek z poprzednich chwil czasowych jest liczba próbek równa 8. Następnie na kolejnej warstwie została zwiększona ilość neronów do liczby 128 która potem maleje do liczby 64 neuronów na kolejnej.

3.2.3. Funkcje aktywacji zastosowane w poszczególnych warstwach

Funkcją aktywacyjną z której skorzystali autorzy jest reLU (rectifier Linear Unit) [2] nazywana również potocznie funkcją rampy.



Rysunek 3. Funkcja reLU

Możemy to zapisać jako

$$f(x) = x^+ = \max(0, x) \tag{4}$$

gdzie x jest wejściem do neurona.
Autorzy testowali również inne funkcje takie jak **SoftPlus**, **SoftMax** jednakże stwierdzone zostało że najlepiej działa wspomniana wcześniej funkcja **reLU**.
W programie można dobrać funkcję aktywacyjną na poszczególnej warstwie widać to na poniższym fragmencie kodu jako argument **'activation'**. Możliwe parametry są opisane w dokumentacji KERAS [1]

```
1 def build_model(train_dataset):
2     _model = keras.Sequential([
3         layers.Dense(NEURAL_NETWORK_SETTINGS.NUMBER_OF_INPUTS, activation='relu',
4                      input_shape=[2,]),
5         layers.Dense(NEURAL_NETWORK_SETTINGS.NEURONS_FIRST_LAYER, activation='relu'),
6         layers.Dense(NEURAL_NETWORK_SETTINGS.NEURONS_SECOND_LAYER, activation='relu'),
```

4. Opis algorytmu uczenia sieci neuronowej

W sekcji 3.1 opisaliśmy algorytm **RMSprop** który został użyty do uczenia sieci neuronowej. Więcej szczegółów dotyczących tego algorytmu można

uzyskać w [3]. Można wykorzystać również inne algorytmy uczące które są opisane w API [1]
— SGD
— RMSprop
— Adam
— Adadelta
— Adagrad
— Adamax
— Nadam
— Ftrl

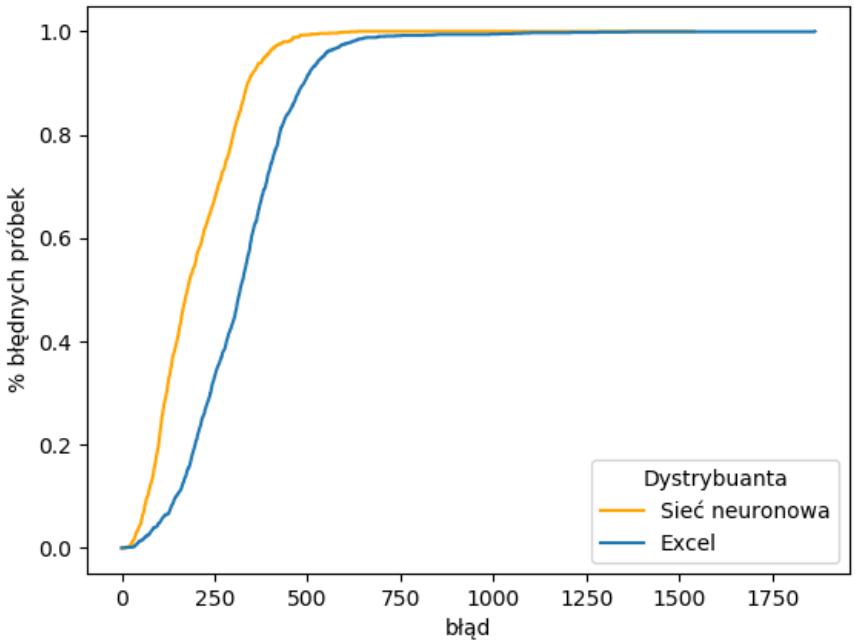
5. Porównanie dystrybuant błędu pomiaru dla danych ze zbioru testowego oraz dla danych uzyskanych w wyniku filtracji przy użyciu sieci neuronowej

W tej sekcji zostanie zaprezentowany wynik jednej sieci neuronowej (sieć **20200504-094744**). Program został uruchomiony z następującymi ustawieniami


```
"settings":[
      {
         "NUMBER_OF_INPUTS":8,
         "NEURONS_FIRST_LAYER":128,
         "NEURONS_SECOND_LAYER":128,
         "NUMBER_OF_OUTPUTS":2,
         "EPOCHS":100,
         "NORMALIZATION":true,
         "NORMALIZATION_DIVIDER":7081,
      }
```

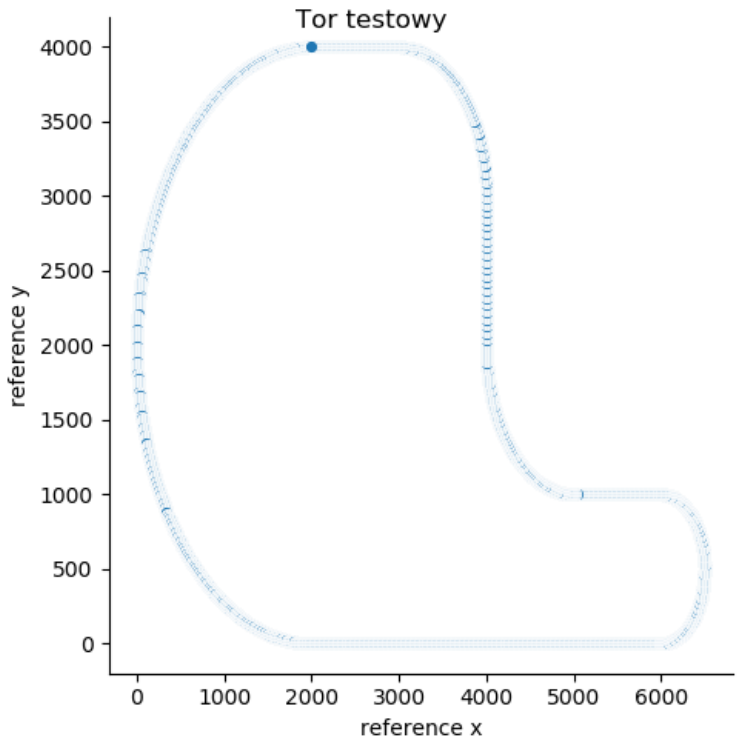

5.1. Porównanie dystrybuant

Na podstawie pliku **pozyxAPI_only_localization_dane_testowe_i_dystrybuanta.xlsx** została obliczona dystrybuanta błędu pomiaru.
Na rysunku 4 autorzy porównują rozkład dystrybuanty dla danych uzyskanych w wyniku filtracji przy użyciu sieci neuronowej z dystrybuantą błędu pomiaru dla danych ze zbioru testowego

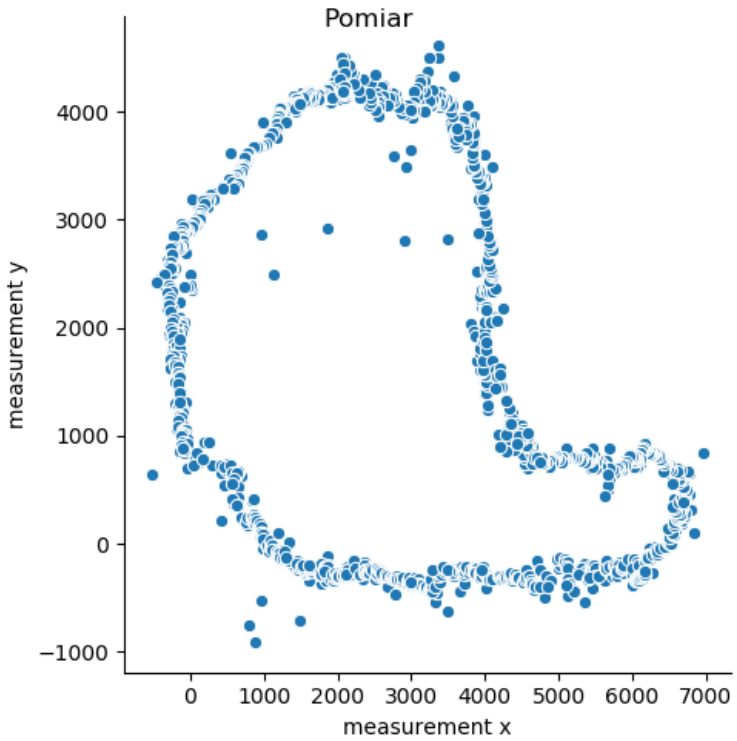


Rysunek 4. Porównanie dystrybuant

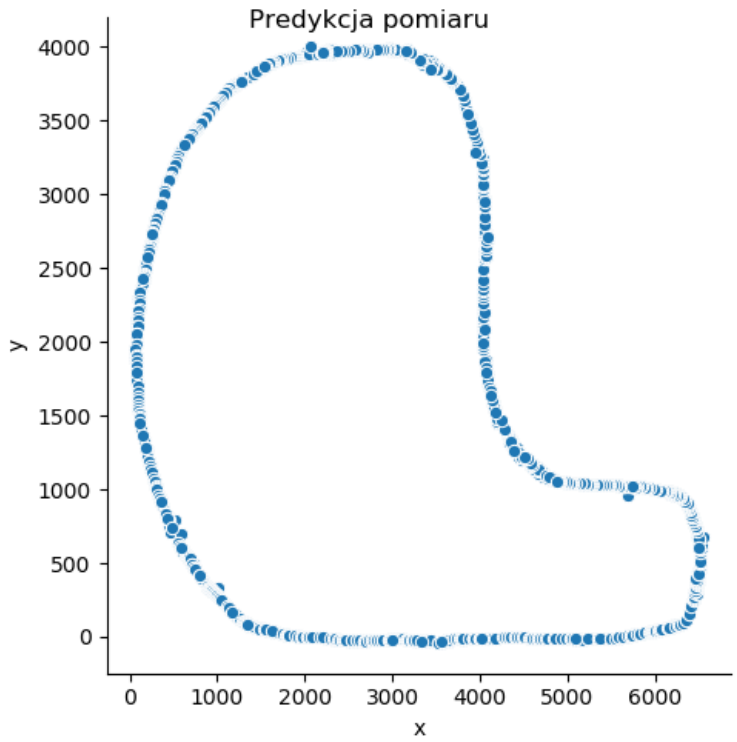
5.2. Wyniki



Rysunek 5. Tor testowy



Rysunek 6. Pomiar



Rysunek 7. Predykcja pomiaru

Porównując rysunki (5, 6, 7) autorzy stwierdzili że zaproponowany przez nich model sieci neuronowej działa poprawnie i pozwoli na korygowanie błędów uzyskanych z systemu pomiarowego robota jeżdżącego po torze.

6. Kod źródłowy programu do uczenia i testowania sieci neuronowej

Kod źródłowy programu został dołączony do tego sprawozdania

7. Wnioski

1. Zaproponowana sieć neuronowa poprawnie koryguje system pomiarowy robota jeżdżącego po torze.
2. Duża ilość neuronów w poszczególnych warstwach nie zawsze oznacza poprawę wyniku, zawsze jednak wydłuża czas działania programu. (**Przy zachowaniu tej samej ilości epok**).
3. Sieć neronowa posiadająca **mało neronów a dużo epok** osiągała podobne wyniki do sieci która posiada **dużo neronów a mało epok**
4. Funkcja aktywacji **reLU** jest w testowanych przypadkach wydajniesza w porównaniu do innych funkcji aktywacyjnych.

Literatura

[1] Dokumentacja online KERAS *https://keras.io/api/*

[2] Opis funkcji rampy *https://en.wikipedia.org/wiki/Rectifier_(neural_networks)*

[3] Lecture 6a Overview of mini-batch gradient descent, Geoffrey Hinton *http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf*

[4] Średni błąd kwadratowy *https://en.wikipedia.org/wiki/Mean_squared_error*

[5] Średni błąd bezwzględny *https://en.wikipedia.org/wiki/Mean_absolute_error*

[6] Sieci neuronowe definicje *https://pl.wikipedia.org/wiki/Sieć_neuronowa*