Uniwersytet Bielsko-Bialski

Liniowe RNN

Sprawozdanie z ćwiczeń
Matematyka Konkretna

Data wykonania:
28.06.2025

Autor:
Bartosz Bieniek 058085

1. Cel ćwiczenia

Dane wejściowe składają się z 30 sekwencji po 20 kroków czasowych każda. Każda sekwencja wejściowa jest generowana z jednolitego rozkładu losowego, który jest zaokrąglany do 0, 0.5 lub 1. Cele wyjściowe t to liczba wystąpień „0.5” w sekwencji.
Zadania umieścić na Github.

2. Przebieg ćwiczenia

1. Import bibliotek
Zaimportowano biblioteki numpy do operacji numerycznych oraz matplotlib.pyplot do wizualizacji wyników. Dzięki temu możliwe było wygodne generowanie danych, obliczenia na tablicach oraz przedstawienie wyników predykcji modelu.

```
import numpy as np
import matplotlib.pyplot as plt
```

Rys 1. Import bibliotek

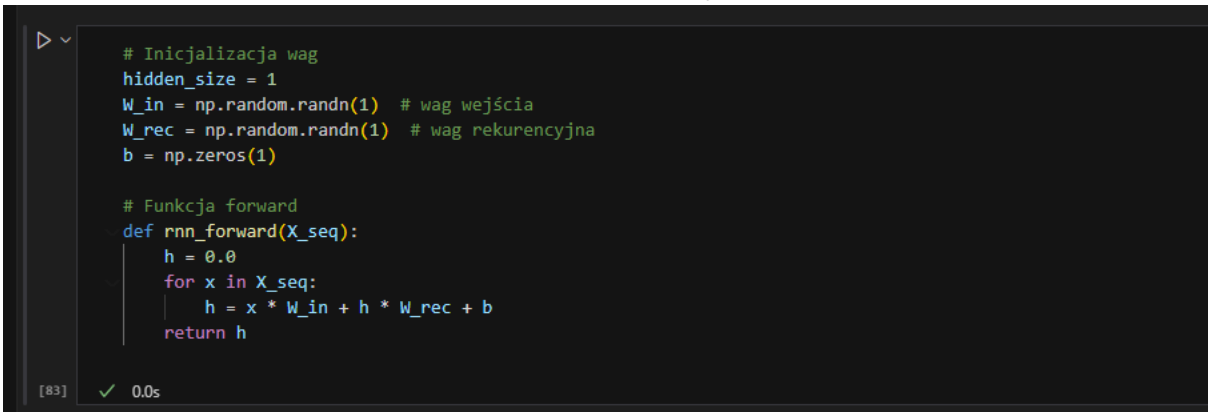
2. Generowanie danych wejściowych i wyjściowych
Wygenerowano 30 sekwencji o długości 20, w których każda wartość pochodziła z rozkładu jednostajnego. Następnie każdą wartość zaokrąglono do najbliższej z trzech wartości: 0, 0.5 lub 1. Cele wyjściowe (targety) obliczono jako liczbę wystąpień wartości 0.5 w każdej sekwencji, co stanowiło problem regresyjny.

```
np.random.seed(42)

# Parametry
num_sequences = 30
sequence_length = 20

# Generowanie danych
X = np.random.uniform(0, 1, (num_sequences, sequence_length))
X = np.round(X * 2) / 2  # Zaokrąglenie do 0, 0.5, 1

# Cele wyjściowe: liczba wystąpień 0.5 w każdej sekwencji
t = np.sum(X == 0.5, axis=1, keepdims=True)

[82] ✓ 0.0s
```

Rys 2. Generowanie danych wejściowych i wyjściowych

3. Definicja prostej liniowej RNN

Zaimplementowano prostą rekurencyjną sieć neuronową bez funkcji aktywacji nieliniowej – każde wyjście obliczano jako liniową kombinację bieżącego wejścia i poprzedniego stanu. Wagi wejściowe i rekurencyjne zainicjalizowano losowo, a stan ukryty aktualizowano iteracyjnie w pętli po krokach czasowych. Tak przygotowana sieć umożliwiała akumulację informacji z całej sekwencji.

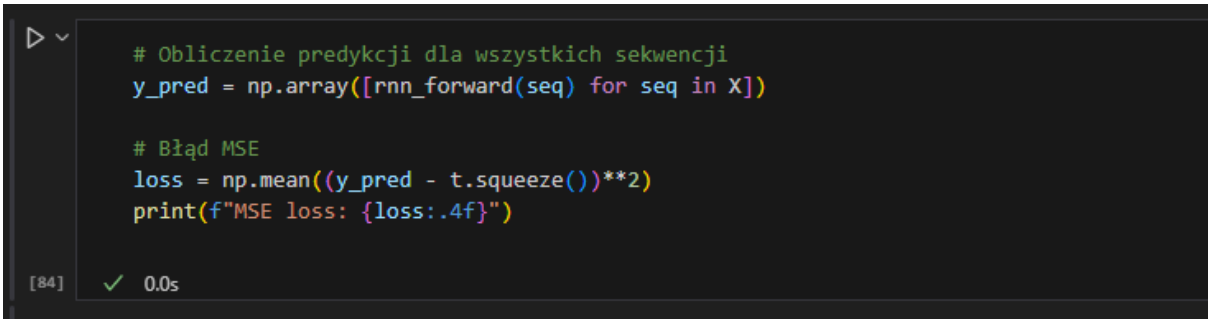



```python
# Inicjalizacja wag
hidden_size = 1
W_in = np.random.randn(1)  # wag wejścia
W_rec = np.random.randn(1)  # wag rekurencyjna
b = np.zeros(1)

# Funkcja forward
def rnn_forward(X_seq):
    h = 0.0
    for x in X_seq:
        h = x * W_in + h * W_rec + b
    return h
```

Rys 3. Definicja prostej liniowej RNN

4. Przejście przez wszystkie sekwencje i obliczenie predykcji

Przeprowadzono propagację sekwencji przez sieć RNN w trybie inference (przewidywania). Dla każdej sekwencji obliczono końcowy stan ukryty, interpretowany jako przewidywana liczba wystąpień wartości 0.5. Na podstawie porównania z wartościami rzeczywistymi obliczono błąd średniokwadratowy (MSE), który posłużył do oceny jakości predykcji.




```python
# Obliczenie predykcji dla wszystkich sekwencji
y_pred = np.array([rnn_forward(seq) for seq in X])

# Błąd MSE
loss = np.mean((y_pred - t.squeeze())**2)
print(f"MSE loss: {loss:.4f}")
```

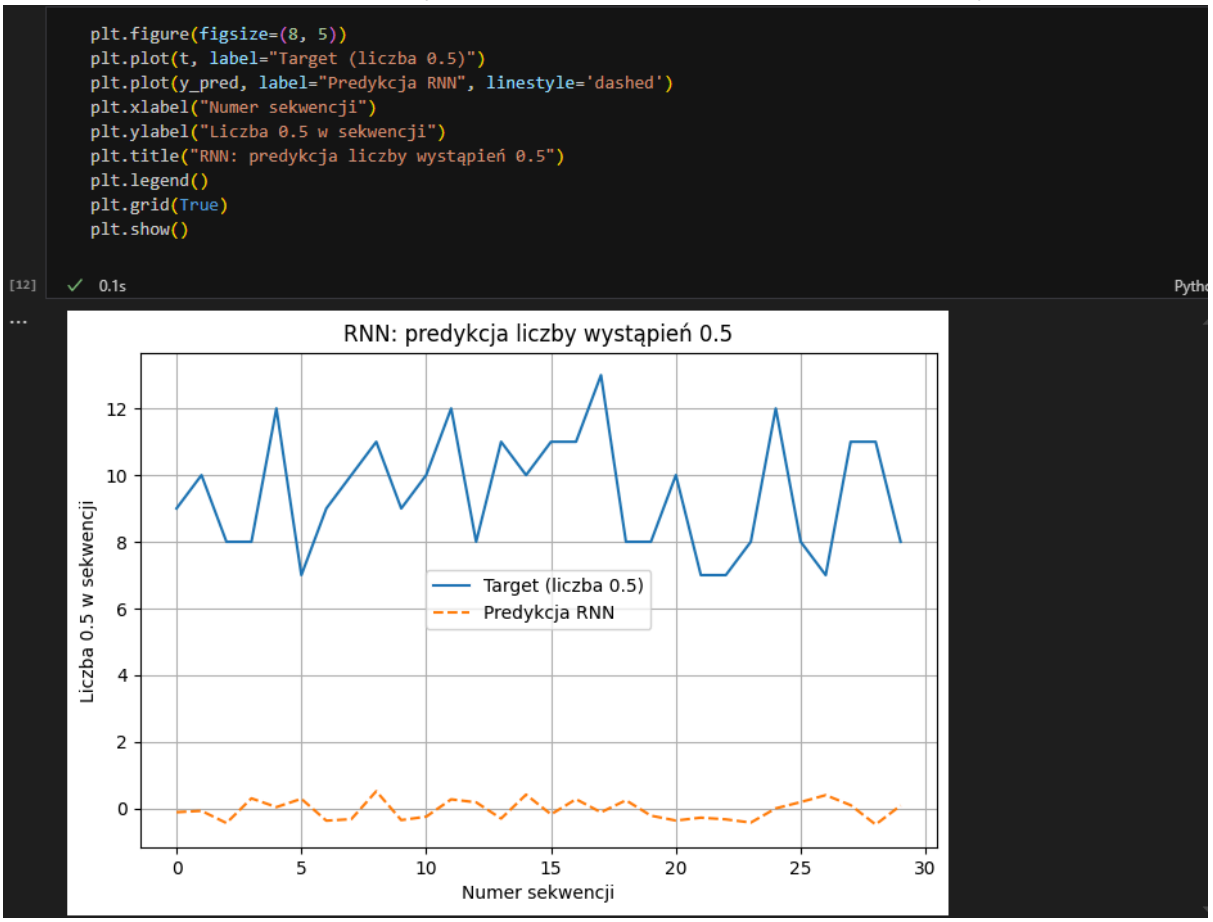
Rys 4. Przejście przez wszystkie sekwencje i obliczenie predykcji

Współczynnik MSE Loss wyniósł 93.4777.

5. Wizualizacja wyników

Na jednym wykresie przedstawiono rzeczywistą liczbę wystąpień wartości 0.5 w każdej sekwencji oraz przewidywaną przez model wartość. Dzięki wizualizacji możliwe było bezpośrednie porównanie trafności predykcji w

kolejnych próbkach. Różnice między wykresami pozwoliły ocenić, na ile skutecznie model nauczył się odwzorowywać zależność czasową.



Rys 5. Wizualizacja wyników

3. Wnioski

Wartość MSE Loss = 93.4777 oznacza, że średni błąd kwadratowy pomiędzy przewidywaniami modelu RNN, a rzeczywistymi liczbami wystąpień wartości 0.5 w sekwencjach wejściowych jest dość wysoki.