SPRAWOZDANIE

Zajęcia: Matematyka Konkretna
Prowadzący: prof. dr hab. Vasyl Martsenyuk

Zadanie 8
Temat: Liniowe RNN
Wariant 13

Łukasz Pindel
Informatyka II stopień,
stacjonarne,
2 semestr,
Gr. 1B

1. Polecenie:

Zadaniem do zrealizowania jest zaimplementowanie sieci neuronowej RNN tak, aby rozwiązać problem z parametrami, przypisanymi zgodnie z wariantem zadania.

2. Wprowadzane dane:

Wariant 13 (odpowiednio wariant 1) - parametryzacja

1. 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.

Rysunek 1: Zestaw parametrów do utworzenia RNN

3. Wykorzystane komendy:

Tworzenie danych:
Na początku generowane są losowe sekwencje liczb rzeczywistych za pomocą funkcji **np.random.uniform()**, które są zaokrąglane do wartości 0.5 lub 1 za pomocą funkcji **np.ceil()**. Następnie, na podstawie tych sekwencji, obliczane są etykiety (**targets**) za pomocą funkcji **np.where()**.

Definicja funkcji krokowych:
W tym etapie zdefiniowane są funkcje **update_state() i forward_states()**, które realizują krok do przodu (**forward step**) RNN, obliczając stan RNN na podstawie poprzedniego stanu i bieżącego wejścia.

Funkcje straty i gradientu:
Podobnie jak wcześniej, zdefiniowane są funkcje **loss** obliczająca błąd średniokwadratowy (MSE) oraz **output_gradient()** obliczająca gradient funkcji straty względem wyjścia sieci.

Algorytm propagacji wstecznej:
Implementowana jest funkcja **backward_gradient()**, która realizuje propagację wsteczną (**backpropagation**) przez sieć, obliczając gradienty funkcji straty względem wag.

Sprawdzanie gradientu:
Przeprowadzane jest sprawdzanie gradientu w celu zweryfikowania poprawności implementacji propagacji wstecznej, wykorzystując m.in. komendy **np.isclose()**.

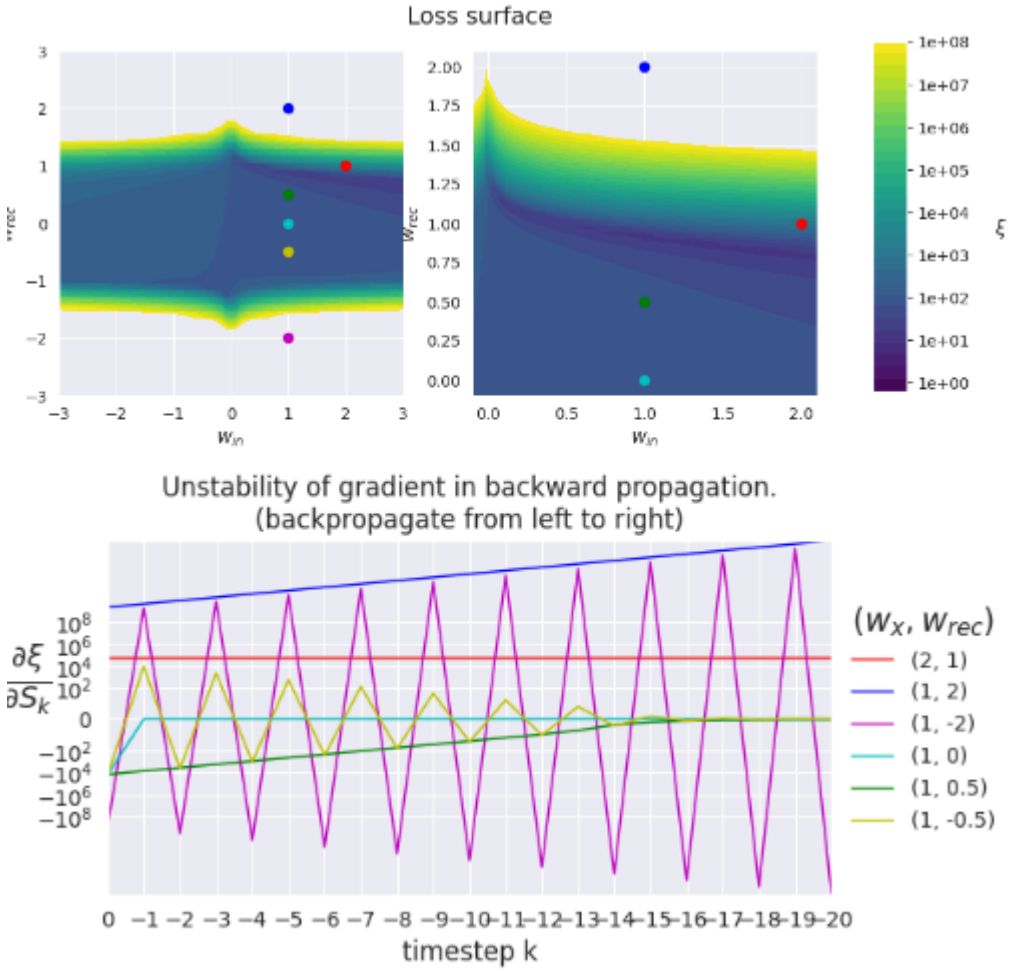
Wizualizacja powierzchni straty:
Dla wizualizacji danych, generowana jest wizualizacja powierzchni straty (**loss surface**) oraz trajektorii optymalizacji wag na tej powierzchni, korzystając z komend związanych z modułem matplotlib.

Algorytm optymalizacji Rprop:
Implementowana jest procedura optymalizacji wag za pomocą algorytmu **Rprop**, korzystając z komend do aktualizacji wag na podstawie znaku gradientu i jego poprzedniego znaku.

Testowanie na nowych danych:
Przeprowadzane jest testowanie modelu na nowych danych, aby sprawdzić jego działanie, wykorzystując m.in. komendy **np.array() i forward_states()**.

Link do repozytorium:
https://github.com/denniak/MK/tree/main/MK_8

4. Wynik działania:

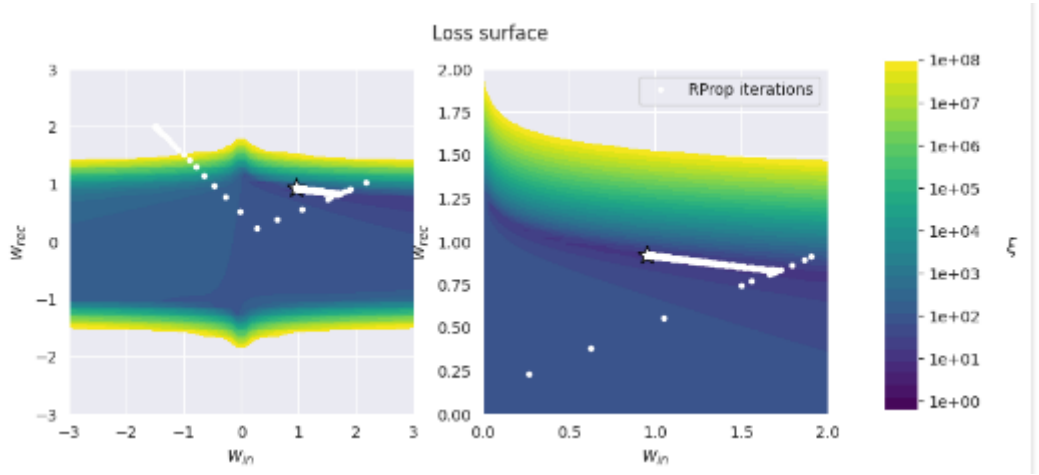


Rysunek 2: Wykresy funkcji błędu

```
print(f'Final weights are: wx = {W[0]:.4f},  wRec = {W[1]:.4f}')
```

```
Final weights are: wx = 0.9549,  wRec = 0.9223
```

Rysunek 3: Uzyskane wagi dla modelu



Rysunek 4: Przedstawienie przebiegu optymalizacji RProp

```
test_input = np.array([[0.5, 1, 1, 0.5, 0.5, 0, 0, 1, 0, 0, 1, 1, 0.5, 1, 1, 0.5, 0.5, 0.5, 1, 1]])
test_output = forward_states(test_input, W[0], W[1])[:, -1]
expected_output = np.sum(test_input == 0.5)

print('Input: \n', test_input)
print('Output from model: \n', test_output)
print('Expected output: \n', expected_output)
```

```
Input:
 [[0.5 1.  1.  0.5 0.5 0.  0.  1.  0.  0.  1.  1.  0.5 1.  1.  0.5 0.5 0.5
  1.  1. ]]
Output from model:
 [6.61848073]
Expected output:
 7
```

Rysunek 5: Wprowadzenie przykładowych danych wejściowych oraz rezultat

5. Wnioski:

Na podstawie otrzymanego wyniku można stwierdzić, że implementacja sieci neuronowej RNN w oparciu o zadane parametry przyniosła zadowalające rezultaty. Dla 20 przykładowych danych, wartość oczekiwana jest zbliżona do wartości zwracanej przez model (6,6 – model, 7 - oczekiwana). Wykorzystane algorytmy propagacji wstecznej i optymalizacji Rprop pozwoliły na skuteczne uczenie się modelu i dostosowanie wag do danych. Wizualizacje powierzchni straty oraz trajektorii optymalizacji wag pomogły w lepszym zrozumieniu procesu uczenia się sieci.