SPRAWOZDANIE

Zajęcia: Matematyka Konkretna

Prowadzący: prof. dr hab. inż. Vasyl Martsenyuk

Laboratorium Nr 8 Data 28.11.2023 Temat: Liniowe RNN Wariant 6	Rafał Klinowski Informatyka II stopień, stacjonarne, 2 semestr, gr. a

1. Polecenie:

Ćwiczenie polegało na stworzeniu notatnika Jupyter w języku Python do utworzenia rekurencyjnej sieci neuronowej (RNN) oraz przetestowania jej dla odgórnie narzuconych danych wejściowych oraz ustalonej cechy wyjściowej.

Wariant zadania: 6

> 6. 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 0.33, 0.66 lub 1. Cele wyjściowe `t` to średnie odchylenie wartości liczb w sekwencji.

2. Napisany program, uzyskane wyniki

Podczas realizacji laboratorium skorzystano z funkcji, obliczeń i parametrów zaproponowanych w instrukcji laboratoryjnej.

Pierwszym krokiem było utworzenie, zgodnie z poleceniem, 30 sekwencji po 20 kroków czasowych zaokrąglonych do wartości: 0.33, 0.66 i 1.0. Dla wygenerowanych w sposób losowy (z ustalonym ziarnem w celu zwiększenia powtarzalności eksperymentów) obliczono odchylenie standardowe odpowiednią funkcją.

```
np.random.seed(seed=61185)

# Tworzenie danych wejściowych
nb_of_samples = 30
sequence_len = 20

# Tworzenie sekwencji wejściowych z rozkładu jednolitego
X = np.random.uniform(size=(nb_of_samples, sequence_len))
# Zaokrąglanie do 0.33, 0.66 lub 1
X = np.ceil(X * 3) / 3  # Wygenerowano liczby z przedziału [0; 1]; * 3 ->
[0; 3]; ceil -> {1,2,3}; / 3 -> {0.33, 0.66, 1}
X = np.round(X, 2)  # Zaokrąglenie do 2 miejsc po przecinku
# Zamiana 0.67 na 0.66
X[X == 0.67] = 0.66

# Tworzenie celu wyjściowego
t = np.std(X, axis=1)
```

Rysunek 1. Utworzenie sekwencji wejściowych oraz uzyskanie oczekiwanej wartości cechy t jako odchylenie standardowe dla każdej z sekwencji.

Następnie na podstawie instrukcji laboratoryjnej zaimplementowano niezbędne funkcje, w szczególności „forward_states” oraz „backward_gradient”.

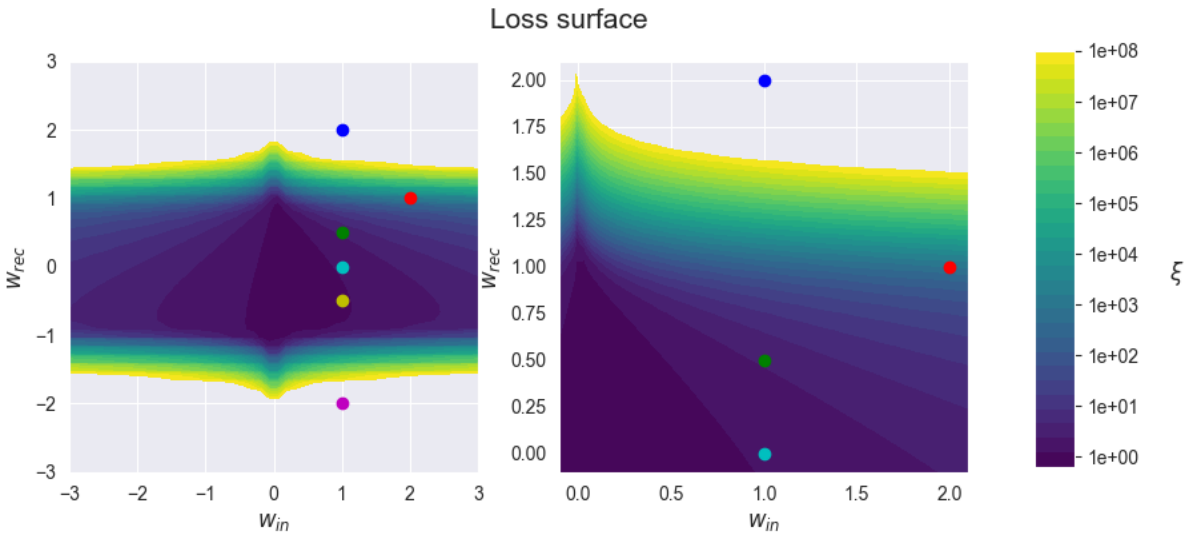
```
def forward_states(X, wx, wRec):
    """
    Unfold the network and compute all state activations
    given the input X, input weights (wx), and recursive weights
    (wRec). Return the state activations in a matrix, the last
    column S[:,-1] contains the final activations.
    """
    # Initialise the matrix that holds all states for all
    #  input sequences. The initial state s0 is set to 0.
    S = np.zeros((X.shape[0], X.shape[1]+1))
    # Use the recurrence relation defined by update_state to update
    #  the states trough time.
    for k in range(0, X.shape[1]):
        # S[k] = S[k-1] * wRec + X[k] * wx
        S[:,k+1] = update_state(X[:,k], S[:,k], wx, wRec)
    return S

def backward_gradient(X, S, grad_out, wRec):
    """
    Backpropagate the gradient computed at the output (grad_out)
    through the network. Accumulate the parameter gradients for
    wX and wRec by for each layer by addition. Return the parameter
    gradients as a tuple, and the gradients at the output of each layer.
    """
    # Initialise the array that stores the gradients of the loss with
    #  respect to the states.
    grad_over_time = np.zeros((X.shape[0], X.shape[1]+1))
    grad_over_time[:,-1] = grad_out
    # Set the gradient accumulations to 0
    wx_grad = 0
    wRec_grad = 0
    for k in range(X.shape[1], 0, -1):
        # Compute the parameter gradients and accumulate the results.
        wx_grad += np.sum(
            np.mean(grad_over_time[:,k] * X[:,k-1], axis=0))
        wRec_grad += np.sum(
            np.mean(grad_over_time[:,k] * S[:,k-1]), axis=0)
        # Compute the gradient at the output of the previous layer
        grad_over_time[:,k-1] = grad_over_time[:,k] * wRec
    return (wx_grad, wRec_grad), grad_over_time
```

Rysunek 2. Zdefiniowane funkcje.

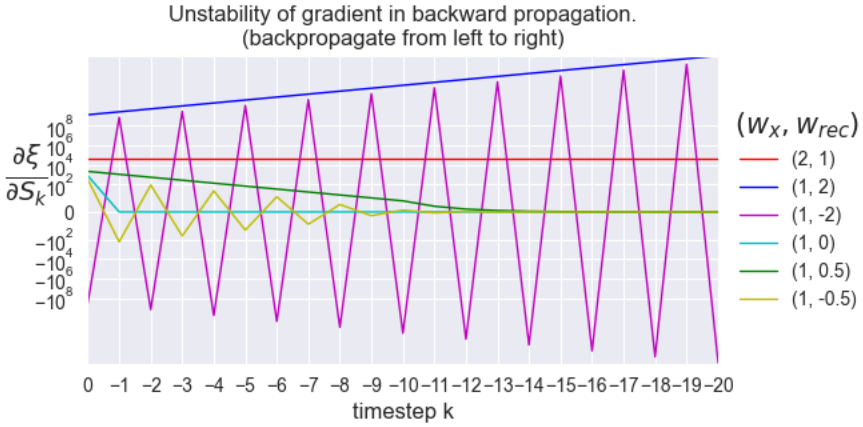
Przeprowadzono analizę gradientu i sprawdzono jego poprawność – nie znaleziono nieprawidłowych wartości.

```
No gradient errors found
```

Następnie utworzono wykresy prezentujące propagację wsteczną oraz gradient.

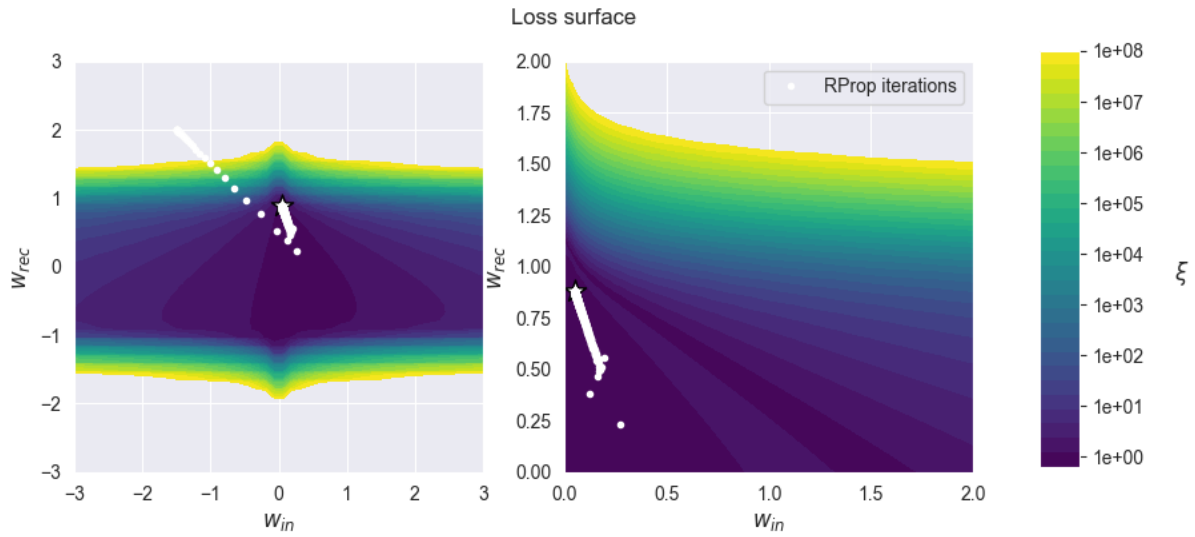


Rysunek 3. Wykresy prezentujące funkcje błędu dla danych wejściowych.



Rysunek 4. Wykres prezentujący gradient propagacji wstecznej w czasie (propagacja od lewej do prawej).

Przeprowadzono optymalizację RProp oraz utworzono wykresy tej optymalizacji.



Rysunek 5. Wykresy przedstawiające przebieg optymalizacji RProp.

Uzyskano następujące wagi modelu:

```
Final weights are: wx = 0.0473,  wRec = 0.8827
```

Rysunek 6. Uzyskane po optymalizacji wagi modelu wx oraz wRec.

Na koniec przetestowano model RNN dla przykładowych danych wejściowych, będących również sekwencją 20 danych przyjmujących wartość 0.33, 0.66 lub 1.0.

```
# Testowanie dla przykładowych danych wejściowych
test_inpt = np.asmatrix([[0.66, 0.33, 0.33, 0.66, 1, 0.66, 0.33, 0.33,
0.66, 1, 0.66, 0.33, 0.33, 0.66, 1, 0.66, 0.33, 0.33, 0.66, 1]])
test_outpt = forward_states(test_inpt, W[0], W[1])[:,-1]
std_test_inpt = test_inpt.std()

print('Input: \n', test_inpt)
print('Output from model: \n', test_outpt)
print('Expected output: \n', std_test_inpt)
```

Rysunek 7. Utworzenie wartości testowych, uzyskanie wyniku z modelu oraz oczekiwanego wyniku – odchylenia standardowego dla tej sekwencji.

```
Input:
 [[0.66 0.33 0.33 0.66 1.   0.66 0.33 0.33 0.66 1.   0.66 0.33 0.33 0.66
  1.   0.66 0.33 0.33 0.66 1.  ]]
Output from model:
 [0.23092597]
Expected output:
 0.25016794358990124
```

Rysunek 8. Uzyskane wyniki; dane testowe, wynik uzyskany z modelu oraz wynik oczekiwany jako odchylenie standardowe dla danych w sekwencji.

Wnioski:

- Uzyskany wynik jest zbliżony do wyniku oczekiwanego, czyli model po optymalizacji i z wykorzystaniem wstecznej propagacji błędu radzi sobie dość dobrze
- W przypadku podania innych danych wejściowych, na przykład mniejszej ilości elementów lub niepoprawnego zakresu, model niepoprawnie obliczy wartość celu
- Implementacja RNN w taki sposób jest dość długa i wymaga sporo pracy, jednak tym samym daje dobre wyniki, dobrze przystosowuje się do różnych wartości celu oraz jest łatwo rozszerzalna i można ją w prosty sposób modyfikować

Repozytorium zawierające uzyskane wyniki wraz z niezbędnymi plikami: https://github.com/Stukeley/MatematykaKonkretna/tree/master/Lab8