SPRAWOZDANIE

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

Zadanie 9
Temat: Nieliniowe sieci RNN w oparciu o tensory
Wariant 13

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

1. Polecenie:

Zadaniem do zrealizowania jest opracowanie rekurencyjnej sieci neuronowej, która implementuje operacji na dwóch liczbach binarnych zgodnie z wariantem zadania.

2. Wprowadzane dane:

Wariant 13 – Różnica dwóch liczb 28 - bitowych

3. Wykorzystane komendy:

Tworzenie zestawu danych:
W pierwszym etapie kodu tworzony jest dataset, który będzie wykorzystywany do nauki sieci neuronowej. Funkcja create_dataset generuje dane do trenowania sieci neuronowej.
create_dataset(nb_samples, sequence_len) tworzy zbiór danych do trenowania sieci neuronowej, generując losowe liczby binarne do dodawania. Parametr nb_samples określa liczbę próbek w zbiorze danych, natomiast sequence_len określa długość sekwencji binarnej.
max_int: Maksymalna wartość liczby, którą można odjąć. Jest to liczba wynikająca z długości sekwencji binarnej.
format_str: Formatuje liczbę całkowitą na postać binarną.
X: Macierz przechowująca dane wejściowe (liczby do odjęcia).
T: Macierz przechowująca dane wyjściowe (wyniki odejmowania).
for i in range(nb_samples): Pętla generująca losowe liczby do odjęcia i obliczająca ich różnicę, czyli wynik odejmowania. Dodatkowo przekształca liczby całkowite na ich binarną postać i odwraca kolejność bitów, aby pasowała do konwencji czytania sekwencji przez sieć neuronową.

Tworzenie i definiowanie sieci neuronowej
W drugim etapie kodu definiowana jest struktura sieci neuronowej, która będzie używana do nauki na wcześniej wygenerowanych danych. Poniżej znajdują się kluczowe funkcje tego etapu:
TensorLinear: Klasa reprezentująca warstwę liniową dla tensorów.
LogisticClassifier: Klasa reprezentująca warstwę klasyfikatora logistycznego.
TanH: Klasa reprezentująca warstwę Tangens hiperboliczny (tanh).
RecurrentStateUpdate: Klasa reprezentująca warstwę aktualizacji stanu rekurencyjnego.
RecurrentStateUnfold: Klasa reprezentująca warstwę rozwinięcia stanu rekurencyjnego w czasie.

RnnBinaryAdder: Klasa reprezentująca pełną sieć neuronową do dodawania binarnego.

Trenowanie sieci neuronowej

W trzecim etapie kodu sieć neuronowa jest trenowana na wcześniej wygenerowanych danych. Używane są algorytmy optymalizacji gradientowej do dostosowania wag sieci neuronowej. Kluczowe parametry i elementy dla tej części to:

Set hyper-parameters: Ustawienie hiperparametrów, takich jak lambda dla Rmsprop, współczynnik uczenia, momentum i epsilona.

RNN = RnnBinaryAdder: Tworzenie instancji sieci neuronowej do nauki.

getParamGrads: Obliczanie gradientów parametrów za pomocą propagacji wstecznej.

ls_of_loss: Lista przechowująca wartości straty (loss) w kolejnych iteracjach trenowania.

for i in range(5): Pętla trenująca sieć neuronową przez określoną liczbę iteracji.

for mb in range(nb_train // mb_size): Pętla iterująca po mini-batchach danych treningowych.

RNN.getOutput(X_mb): Obliczanie wyjścia sieci neuronowej dla danej mini-batch.

RNN.loss(RNN.getOutput(X_mb), T_mb): Obliczanie wartości funkcji straty dla danego mini-batch.

Ocena na zbiorze testowym:

W ostatnim etapie kodu sieć neuronowa jest oceniana na zbiorze testowym. Wyniki są wypisywane dla każdego przypadku testowego. Kluczowa elementami w tej części kodu są:

getBinaryOutput: Metoda zwracająca binarne wyjście dla danych wejściowych.

getOutput: Metoda zwracająca wyjście dla danych wejściowych.

printSample: Funkcja wyświetlająca przykłady danych treningowych w czytelny sposób.

Link do repozytorium:

https://github.com/denniak/MK/tree/main/MK_9

4. Wynik działania:

```
# Create training samples
X_train, T_train = create_dataset(nb_train, sequence_len)
print(f'X_train tensor shape: {X_train.shape}')
print(f'T_train tensor shape: {T_train.shape}')
#
```

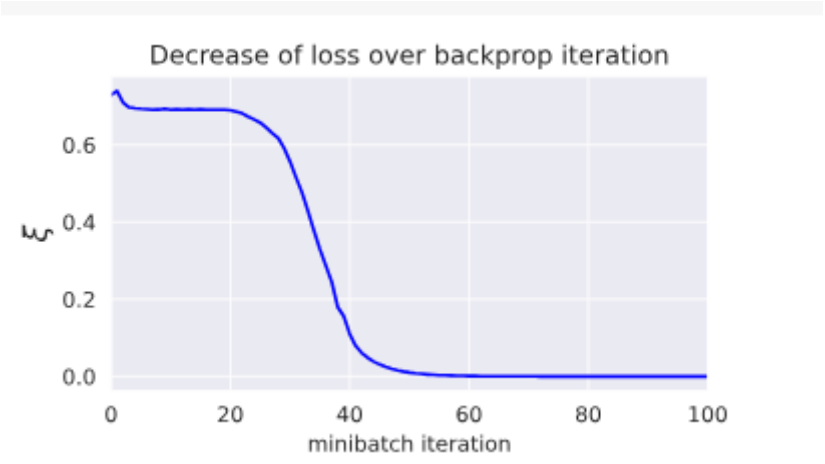
```
X_train tensor shape: (2000, 28, 2)
T_train tensor shape: (2000, 28, 1)
```

Rysunek 1: Wyświetlenie parametrów zbiorów treningowych

```
# Print the first sample
printSample(X_train[0,:,0], X_train[0,:,1], T_train[0,:,:])
#
```

```
x1:   01100001101111010000101110000   15252870
x2: - 10010000100001100111000010000   9330953
      -------   --
t:  = 10111110001110100101101000000   5921917
```

Rysunek 2: Przykładowe wywołanie funkcji odejmującej liczby



Rysunek 3: Funkcja straty w zależności od iteracji propagacji wstecznej

```
x1:   10111111010001000011000000110   101458685
x2: - 01010100010100111011111111000   33409578
      -------   --
t:  = 11001011000110100111000000010   68049107
y:  = 11001011000110100111000000010

x1:   10000111000111101100111001110   108230881
x2: - 10111111110110100001110111010   95968253
      -------   --
t:  = 00100111001110001101110100000   12262628
y:  = 00100111001110001101110100000

x1:   11110111100010100011010011110   120345071
x2: - 01111011110100010000111001000   40930270
      -------   --
t:  = 10001000011000111101110100010   79414801
y:  = 10001000011000111101110100010

x1:   10111110110100000010011001110   107219837
x2: - 11110010000010111101001100010   80465999
      -------   --
t:  = 01110100110111000001100110000   26753838
y:  = 01110100110111000001100110000

x1:   10011011101101000010001001110   105131481
x2: - 10001010101110100101101000000   5922129
      -------   --
t:  = 00010001000010111001011110100   99209352
y:  = 00010001000010111001011110100
```

Rysunek 4: Uzyskany wynik dla przykładowych pięciu próbek

5. Wnioski:

Na podstawie otrzymanego wyniku można stwierdzić, że nieliniowa sieć rekurencyjna neuronów zdolna jest efektywnie uczyć się operacji różnicy dwóch liczb binarnych o długości 28 bitów. Dzięki zastosowaniu odpowiednio zdefiniowanej struktury sieci i odpowiednich funkcji aktywacji, sieć ta może dokładnie odwzorować złożoną nieliniową relację pomiędzy danymi wejściowymi a ich różnicą, co widać w uzyskanych wynikach dla przykładowych próbek. Ponadto, analiza funkcji straty wskazuje na skuteczność procesu trenowania sieci, co świadczy o potencjalnej użyteczności tego podejścia w rozwiązywaniu problemów związanych z operacjami na danych binarnych.