Advanced Reading in Computer Vision (MAT3563)

BÀI THỰC HÀNH SỐ 10 – LONG SHORT TERM MEMORY LSTM

Ví dụ 1. Nội dung:

Trong bài thực hành này chúng ta sẽ thực hiện việc xây dựng một mạng LSTM dựa theo các công thức lý thuyết (chỉ dùng thư viện cơ sở numpy). Code đầy đủ của các phương thức nêu dưới đây đã có trong tệp đính kèm numpy_lstm.ipynb.

Việc xây dựng sẽ bắt đầu từ các cell cơ bản, với hàm kích hoạt tại các cổng được xây dựng như trong phần lý thuyết. Chúng ta cần một số hàm cơ bản như sigmoid, softmax, phương pháp lặp GD Adam giải bài toán tối ưu, cập nhật lại các vector tham số theo phương pháp Adam:

```
import numpy as np

def softmax(x):
# code here


def sigmoid(x):
# code here

def initialize_adam(parameters) :
#code here


def update_parameters_with_adam(parameters, grads, v, s, t, learning_rate = 0.01,
                                beta1 = 0.9, beta2 = 0.999,  epsilon = 1e-8):
# code here
```

Tiếp theo, chúng ta xây dựng phương thức tính cho chiều tiến (forward) tại mỗi cell của mạng LSTM

```
# GRADED FUNCTION: lstm_cell_forward

def lstm_cell_forward(xt, a_prev, c_prev, parameters):
    """
    Implement a single forward step of the LSTM-cell as described in Figure (4)

    Arguments:
    xt -- your input data at timestep "t", numpy array of shape (n_x, m).
    a_prev -- Hidden state at timestep "t-1", numpy array of shape (n_a, m)
    c_prev -- Memory state at timestep "t-1", numpy array of shape (n_a, m)
    parameters -- python dictionary containing:
                        Wf -- Weight matrix of the forget gate, numpy array of shape
(n_a, n_a + n_x)
                        bf -- Bias of the forget gate, numpy array of shape (n_a, 1)
                        Wi -- Weight matrix of the update gate, numpy array of shape
(n_a, n_a + n_x)
                        bi -- Bias of the update gate, numpy array of shape (n_a, 1)
                        Wc -- Weight matrix of the first "tanh", numpy array of shape
(n_a, n_a + n_x)
                        bc --  Bias of the first "tanh", numpy array of shape (n_a, 1)
                        Wo -- Weight matrix of the output gate, numpy array of shape
(n_a, n_a + n_x)
                        bo --  Bias of the output gate, numpy array of shape (n_a, 1)
                        Wy -- Weight matrix relating the hidden-state to the output,
numpy array of shape (n_y, n_a)
                        by -- Bias relating the hidden-state to the output, numpy array
of shape (n_y, 1)

    Returns:
    a_next -- next hidden state, of shape (n_a, m)
```

```
    c_next -- next memory state, of shape (n_a, m)
    yt_pred -- prediction at timestep "t", numpy array of shape (n_y, m)
    cache -- tuple of values needed for the backward pass, contains (a_next, c_next,
a_prev, c_prev, xt, parameters)

    Note: ft/it/ot stand for the forget/update/output gates, cct stands for the
candidate value (c tilde),
          c stands for the memory value
    """
```

Và sau đó là phần tính toán và lưu trữ forward cho toàn bộ các cell trong mạng LSTM

```
# GRADED FUNCTION: lstm_forward

def lstm_forward(x, a0, parameters):
    """
    Implement the forward propagation of the recurrent neural network using an LSTM-
cell described in Figure (3).

    Arguments:
    x -- Input data for every time-step, of shape (n_x, m, T_x).
    a0 -- Initial hidden state, of shape (n_a, m)
    parameters -- python dictionary containing:
                        Wf -- Weight matrix of the forget gate, numpy array of shape
(n_a, n_a + n_x)
                        bf -- Bias of the forget gate, numpy array of shape (n_a, 1)
                        Wi -- Weight matrix of the update gate, numpy array of shape
(n_a, n_a + n_x)
                        bi -- Bias of the update gate, numpy array of shape (n_a, 1)
                        Wc -- Weight matrix of the first "tanh", numpy array of shape
(n_a, n_a + n_x)
                        bc -- Bias of the first "tanh", numpy array of shape (n_a, 1)
                        Wo -- Weight matrix of the output gate, numpy array of shape
(n_a, n_a + n_x)
                        bo -- Bias of the output gate, numpy array of shape (n_a, 1)
                        Wy -- Weight matrix relating the hidden-state to the output,
numpy array of shape (n_y, n_a)
                        by -- Bias relating the hidden-state to the output, numpy array
of shape (n_y, 1)

    Returns:
    a -- Hidden states for every time-step, numpy array of shape (n_a, m, T_x)
    y -- Predictions for every time-step, numpy array of shape (n_y, m, T_x)
    caches -- tuple of values needed for the backward pass, contains (list of all the
caches, x)
    """
```

Sau khi xây dựng các thủ tục cho phần ForWard, chúng ta tiếp tục xây dựng các phương thức cho phần lan truyền ngược (Back Propagation). Tương tự cho phần tính tiến, trước hết ta cần xây dựng tính toán lan truyền ngược tại mỗi cell của LSTM.

```
# Build BackWard at a single cell of lstm

def lstm_cell_backward(da_next, dc_next, cache):
    """
    Implement the backward pass for the LSTM-cell (single time-step).

    Arguments:
    da_next -- Gradients of next hidden state, of shape (n_a, m)
    dc_next -- Gradients of next cell state, of shape (n_a, m)
    cache -- cache storing information from the forward pass
```

```
    Returns:
    gradients -- python dictionary containing:
                        dxt -- Gradient of input data at time-step t, of shape (n_x, m)
                        da_prev -- Gradient w.r.t. the previous hidden state, numpy
array of shape (n_a, m)
                        dc_prev -- Gradient w.r.t. the previous memory state, of shape
(n_a, m, T_x)
                        dWf -- Gradient w.r.t. the weight matrix of the forget gate,
numpy array of shape (n_a, n_a + n_x)
                        dWi -- Gradient w.r.t. the weight matrix of the update gate,
numpy array of shape (n_a, n_a + n_x)
                        dWc -- Gradient w.r.t. the weight matrix of the memory gate,
numpy array of shape (n_a, n_a + n_x)
                        dWo -- Gradient w.r.t. the weight matrix of the output gate,
numpy array of shape (n_a, n_a + n_x)
                        dbf -- Gradient w.r.t. biases of the forget gate, of shape
(n_a, 1)
                        dbi -- Gradient w.r.t. biases of the update gate, of shape
(n_a, 1)
                        dbc -- Gradient w.r.t. biases of the memory gate, of shape
(n_a, 1)
                        dbo -- Gradient w.r.t. biases of the output gate, of shape
(n_a, 1)
    """
```

Cuối cùng, chúng ta xây dựng phần tính toán lan truyền ngược trên toàn bộ mạng LSTM

```
# LSTM pass BackWard
def lstm_backward(da, caches):

    """
    Implement the backward pass for the RNN with LSTM-cell (over a whole sequence).

    Arguments:
    da -- Gradients w.r.t the hidden states, numpy-array of shape (n_a, m, T_x)
    dc -- Gradients w.r.t the memory states, numpy-array of shape (n_a, m, T_x)
    caches -- cache storing information from the forward pass (lstm_forward)

    Returns:
    gradients -- python dictionary containing:
                        dx -- Gradient of inputs, of shape (n_x, m, T_x)
                        da0 -- Gradient w.r.t. the previous hidden state, numpy array
of shape (n_a, m)
                        dWf -- Gradient w.r.t. the weight matrix of the forget gate,
numpy array of shape (n_a, n_a + n_x)
                        dWi -- Gradient w.r.t. the weight matrix of the update gate,
numpy array of shape (n_a, n_a + n_x)
                        dWc -- Gradient w.r.t. the weight matrix of the memory gate,
numpy array of shape (n_a, n_a + n_x)
                        dWo -- Gradient w.r.t. the weight matrix of the save gate,
numpy array of shape (n_a, n_a + n_x)
                        dbf -- Gradient w.r.t. biases of the forget gate, of shape
(n_a, 1)
                        dbi -- Gradient w.r.t. biases of the update gate, of shape
(n_a, 1)
                        dbc -- Gradient w.r.t. biases of the memory gate, of shape
(n_a, 1)
                        dbo -- Gradient w.r.t. biases of the save gate, of shape (n_a,
1)
    """
```

Tự thực hành:

- Hãy sử dụng các phương thức đã có, khởi tạo ngẫu nhiên một bộ dữ liệu vào và một bộ tham số W, sau đó thực hiện các phép tính toán theo phương thức nói trên để cập nhật và cho bộ tham số sau khi đào tạo.

Ví dụ 2: Ví dụ này tham khảo từ mô hình ngôn ngữ ở mức ký tự (Character level language model). Chúng ta dựa vào một tập training là các mẫu tên các loài khủng long đã biết. Sau đó thử dự đoán một số mẫu tên mới với xuất phát từ ký tự nào đó.

Ta có thể thấy rằng đây là ví dụ này là một đơn cử đơn giản cho mô hình ngôn ngữ:

- Các mẫu tên loài sẽ là một dữ liệu dạng chuỗi (được cấu trúc từ các chữ cái và có thứ tự - sequential data).
- Mỗi ký tự sẽ đóng vai trò như đâù vào tại time-step *t* bất kỳ. Dễ thấy nếu thay các ký tự bằng các từ có trong từ điển, thì mô hình sẽ tạo ra các "chuỗi từ" (câu).
- Trong ví dụ này, chúng ta sử dụng mạng RNN cơ sở (xem lại bài thực hành trước).

Tệp dữ liệu training: dinos.txt đính kèm, là danh sách tên các loài đã có.

Chương trình cần thực hiện việc thống kê các từ, lấy các ký tự khác nhau (không phân biệt chữ hoa – chữ thường), thay ký tự bằng "tocken" của nó, bản chất là chỉ số. Giả sử tệp dữ liệu ở cùng thư mục, đoạn lệnh sau thực hiện công việc trên

```
data = open(D:\\Teach_n_Train\\Advanced_Lessons_CV\\LABS\\RNN\\dinos.txt', 'r').read()
data= data.lower()
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print('There are %d total characters and %d unique characters in your data.' %
(data_size, vocab_size))
```

Đoạn lệnh tiếp theo sẽ chuyển các ký tự sang dạng chỉ số (tocken)

```
char_to_ix = { ch:i for i,ch in enumerate(sorted(chars)) }
ix_to_char = { i:ch for i,ch in enumerate(sorted(chars)) }
print(ix_to_char)
```

Toàn bộ phần chương trình tiếp theo có trong tệp nguồn đính kèm Character-level-language-model : Bao gồm phần cơ sở mạng RNNs (đã có trong bài thực hành trước) và ứng dụng vào ví dụ này.