CS-587: Assignment 5

RNNs: Formulation and Application in NLP

Issued: Wednesday, 15 May 2024
Deadline: Saturday, 15 June 2024, 23:59

Description

The goal of this exercise is to implement, train and test a Recurrent Neural Network (RNN) using the *Numpy* framework.

This assignment consists of two parts. In the first part, you will implement step-by-step the structure and functionalities of RNN cells. In the second part, you will implement the key functionalities of an RNN model, which will be used in a specific task of Natural Language Processing, called Character-level Text Generation. The aim of this exercise is to teach you how RNN models are defined at core-level and to allow you to gain intuition about the functionality, limitations and advantages of RNNs in problems that require sequence modeling, such as text generation and query answering, video analysis, music/sound processing and synthesis, etc.

Part A: RNN cells

Each RNN model (shown in Figure 2) can be considered as the repeated use of a single cell (a recurrence learning cell), known as the RNN cell. The goal of this part of the assignment is to implement the RNN-cell described in Figure 1. Moreover, in order to get an understanding of the purpose of the use of these functions to form a memorization cell (such as the RNN), as an additional task for this part of the assignment, you are required to answer questions regarding the overall functionality, limitations, and advantages, as well as the potential solutions to the limitations of RNN cells.

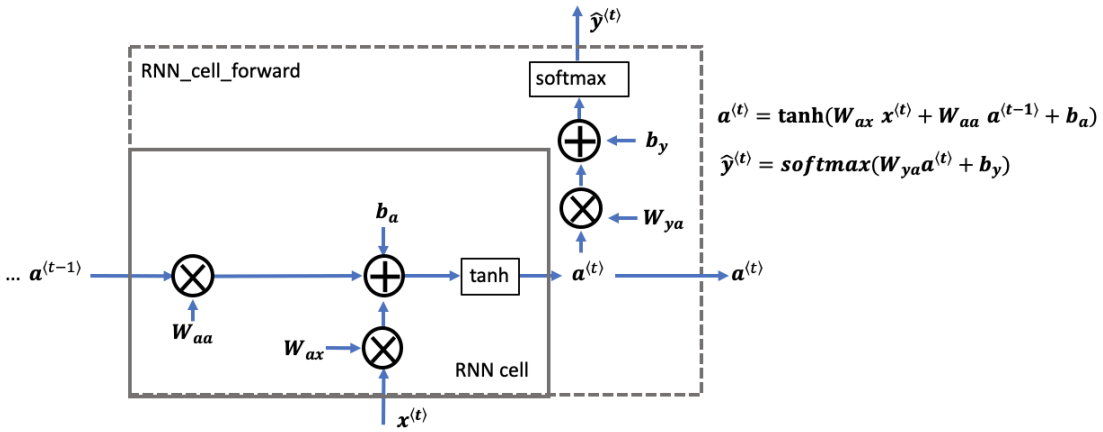


Figure 1: A layout of the RNN cell.

You can use any available function of NumPy or SciPy required for the following tasks.

1. Implement the forward step of the RNN-cell as described in figure 1. Define the activation states and the output of the cell. An RNN cell outputs the hidden state $\alpha^{\langle t \rangle}$. In this task you will also need to implement the prediction $\hat{y}^{\langle t \rangle}$, shown in the figure as the outer box that has dashed lines.

   The implementation that you need to do in the *rnn_cell_forward* function is summarized in the following steps:

   - Compute the hidden state (output of RNN cell), $\alpha^{\langle t \rangle}$.
   - Use the hidden state to compute the prediction, $\hat{y}^{\langle t \rangle}$. Use the soft-max that is imported in the script.
   - Store $(a^{\langle t \rangle}, a^{\langle t-1 \rangle}, x^{\langle t \rangle}, parameters)$ in a python tuple, and return $a^{\langle t \rangle}$, $\hat{y}^{\langle t \rangle}$ and the tuple.

2. An RNN model simply repeats the RNN cell that you've just built. If your input data is a sequence of N time steps long, then you will re-use the RNN cell N times.

   Based on the overview of the model in Figure 2, you can observe the following:

   - Each cell at each time step takes as input two things
     (a) the hidden state of the previous cell, $\alpha^{\langle t-1 \rangle}$
     (b) the input at the current time-step, $x^{\langle t \rangle}$
   - At each time-step the model outputs
     (a) a hidden state $\alpha^{\langle t \rangle}$
     (b) a prediction $\hat{y}^{\langle t \rangle}$
   - The weights and biases $(W_{aa}, b_a, W_{ax}, b_x)$ are re-used each time step (stored in a dictionary called *parameters*).

   In this task, you will implement the learning flow of the forward pass of an RNN model. Specifically, you will fill in the *rnn_forward* function with the appropriate code to perform the following steps (in the order that they are mentioned):

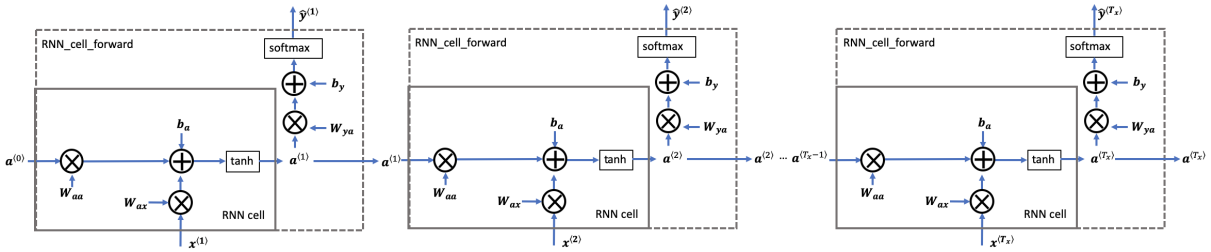


Figure 2: An overview of the RNN model.

   - Create an array of zeros with shape (n_a, m, T_x) to store the hidden states. Name it `a`.
   - Create an array of zeros with shape (n_y, m, T_x) to store the predictions. Name it `y_pred`.
   - Initialize the 2D hidden state `a_next` by setting it equal to the initial hidden state, `a0`.
   - For each time-step t:
     - Get the corresponding data to be fed to the RNN cell. This will be a slice of `x`, with `x` having a shape of (n_x, m, T_x).
     - Update the hidden state $a^{\langle t \rangle}$ using the `a_next`, the prediction $\hat{y}^{\langle t \rangle}$ and the tuple that stores the values you need for backward propagation, by running `rnn_cell_forward`. Note that `a` has shape (n_a, m, T_x)
     - Store the current hidden state in the 3D tensor `a`.
     - Store the current $\hat{y}^{\langle t \rangle}$ prediction (named `yt_pred` in your code) in the tensor $\hat{y}_{pred}$.
     - Add the current values you need for backward propagation (that are stored in your tuple), to the list `caches`.

3. As a final task of this part of the assignment, you are required to answer the following **questions** in your report:

   A. Explain briefly the general functionality of RNN cells (how do we achieve memorization, the purpose of each function on the memorization task).

   B. List the limitations of RNNs, and briefly mention why these issues arise.

   C. Consider the two following problems, (a) Recognizing images between 10 different species of birds, and (b) Recognizing whether a movie review says that the movie is worth watching or not. For which of the two problems can we use a RNN-based model? Justify your answer.

   D. While training an RNN, you observe an increasing loss trend. Upon looking you find out that at some point the weight values increase suddenly very much and finally, take the value NaN. What kind of learning problem does this indicate? What can be a solution? (HINT: answer this after you get a glimpse of Part B).

   E. Gated Recurrent Units (GRUs) have been proposed as a solution on a specific problem of RNNs. What is the problem they solve? And how?

Part B: Text Generation using a RNN model

As an application of the RNN model that you have implemented in the previous part of the assignment, we will use it to build a character-level language model to generate new text. The algorithm will learn the different text patterns that are present in your training dataset, and randomly generate new text. As datasets, we are going to use a list that contains the names of existing chemical elements found in nature, and a list with movie titles. So, our model's task is to create names for new chemical elements that are likely (not) to be discovered by chemists, or new movie titles.

The pipeline of the algorithm that we will be using in our model can be summarized in the following stages:

- Load the names, split them up to a single-character level and find the unique characters (i.e. we define the alphabet of the letters that are used to form a chemical element name).
- Define our model, i.e. the forward and backward propagation operations that lead to the definition of a loss function and the gradients that will be used to update the model's parameters.

In order for our model to better learn to generalize, in our algorithm we will define an additional routine that **clips** the gradients into a specific range. This process facilitates learning leading our model to achieve generalization, as shown in figure 3.

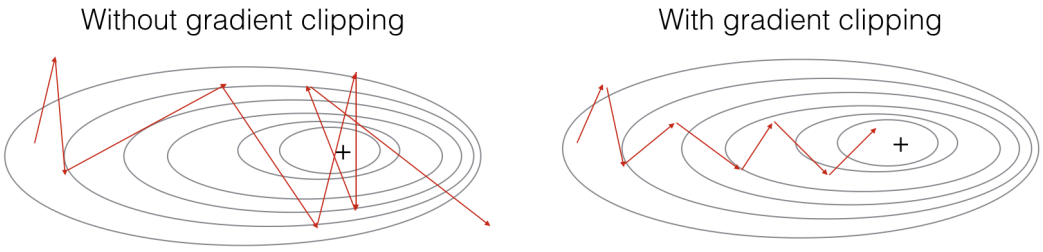


Figure 3: A depiction of the role of clipping in model generalization.

For this part of the assignment, you need to implement and answer the following:

1. **Coding**: For the coding part of the assignment, you need to fill in the following parts of the scripts.
   A. `rrn_step_forward`: define the operations for the hidden state `a_next`, and the prediction for each time-step, $y\langle t \rangle$.
   B. `rrn_step_backward`: compute the gradients (partial derivatives) for each of the parameters. Name them as `dWya, dby, da_next, db, dWax, dWaa, da_next`. Store the gradients in the python dictionary `gradients`, that is provided in the function.
   C. `update_parameters`: use the gradients that you have computed in the previous step to define the updates for the parameters of the RNN cell. Store the updated values in the dictionary `parameters`, that is provided.
   D. `clip`: implement gradient clipping on the gradients of each parameter.
   E. `optimize`: implement the optimization process for the model by calling at the appropriate place the functions `rnn_forward, rnn_backward, clip, and update_parameters`.
   F. Instead of chemical names, you will now generate new movie titles. Replace the dataset and change the appropriate parts of the code in order for your model to executed. Add in your report the new movie titles that your model generated in the last epoch (iteration).

2. **Questions**:

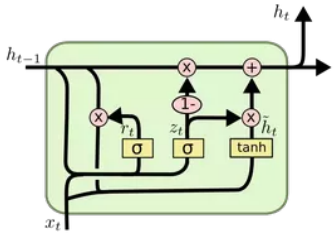
   A. Include the computational graph of the RNN cell in your report, along with the extensive mathematical derivation process you used to get the gradients.

   B. What is the purpose of gradient clipping? What learning limitation we are aiming to solve?

   C. What do you observe about the new generated chemical names? How do you interpret the format of the new generated chemical element names?

   D. What happens when you train your model for the *movie_titles* dataset? what do you observe and how do you interpret this observation?

   E. Can you think of ways to improve the model for the case of the *movie_title* dataset?



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 4: A depiction of a GRU cell and its mechanisms.

BONUS: Extra points (up to +15%) will be distributed to the students that
(a) implement the operations of a GRU cell (code), as shown in figure 4 (+10%).
(b) briefly explain the difference between GRU and Long-Short Term Memory (LSTM) cells (+5%).

Important Notes

Submission info

- Create a .pdf (preferred) or .doc file to report the resulting generated words, and any other comments or description of your work you may need to submit. Do not forget to include your name and ID in the report. Save this file in your working folder.
- After you have finalized your coding and report, remove the `datasets` folder from the working directory to be submitted.
- Use zip/rar/gz to compress your working folder and rename it to cs587_mylogin_assignment5.xxx in order to submit a single file.
- The submission of your implementation will be via the e-learn platform. Submissions via e-mail will not be accepted.
- You can upload your submission as many times as you need, keeping the same filename.
- Late submissions will be accepted **within two days** of the original deadline. However, **a penalty of 25% per day** will be applied to the final grade for each late day. Assignments submitted more than two days late will not be accepted.

Academic integrity

The assignment is individual. You may discuss with each other in general terms, but the code and the report should be written individually. If you use existing material, be sure to cite the sources in your report. The use of AI language models to produce the report or your implementation is strictly prohibited, and submissions will be verified with an AI detection tool. You may be asked to take an additional short oral examination.

Troubleshooting & Contact

If you encounter any errors or bugs in the provided code, you can report them by sending an email to kaziales@csd.uoc.gr or the course mailing list hy578-list@csd.uoc.gr. Please note that the course mailing list is the preferred channel for general questions about the assignment, as it allows everyone in the class to benefit from the answer.

For any questions of a more personal nature that are not relevant to the entire class, you can reach out to the teaching assistant directly at his email address kaziales@csd.uoc.gr.