CS-587: Assignment 4

RNNs: Formulation and Application in NLP

Issued: 17 May 2023
Deadline: 31 May 2023, 23:59

Description

The goal of this exercise is to implement, train and test a Recurrent Neural Network (RNN) using the TensorFlow 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 a 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 modelling, such 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.

You can use any available function of NumPy, SciPy, TensorFlow, TensorBoard 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. A 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 softmax that is imported in the script.

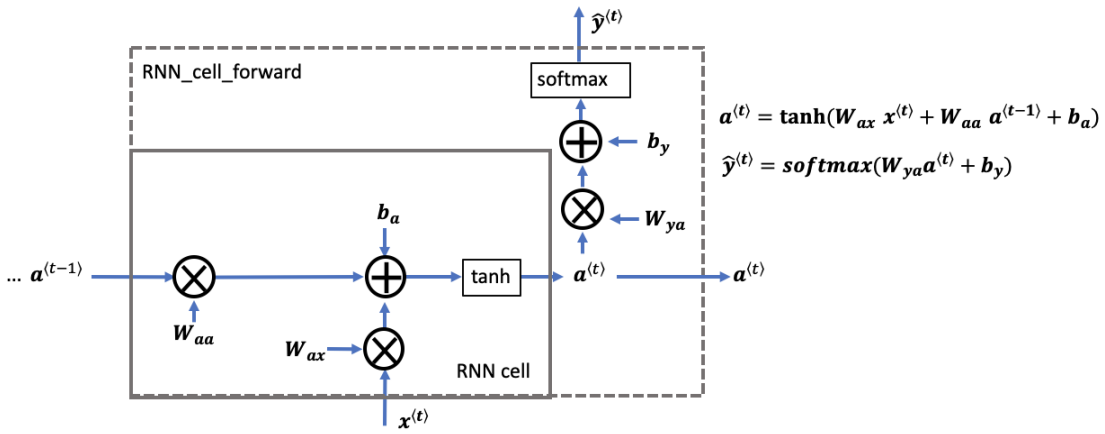


Figure 1: A layout of the RNN cell.

- 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 X, you can observe the following:

   - Each cell at each time step is introduced with two things, (a) the hidden state of the previous cell $\alpha^{\langle t-1 \rangle}$, (b) The current input time-step. $x^{\langle t \rangle}$.
   - At each time-step the model outputs, (a) a hidden state $\alpha^{\langle t \rangle}$, and (b) a prediction $y\langle t-1 \rangle$.
   - 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):

   - Create two arrays of zeros with shapes, (a) (n_a, m, T_x) to store the hidden states, (b) (n_y, m, T_x) that will store the predictions. Name them a, and y.
   - Initialize the 2D hidden state `a_next` by setting it equal to the initial hidden state, a_0.
   - For each time-step t:

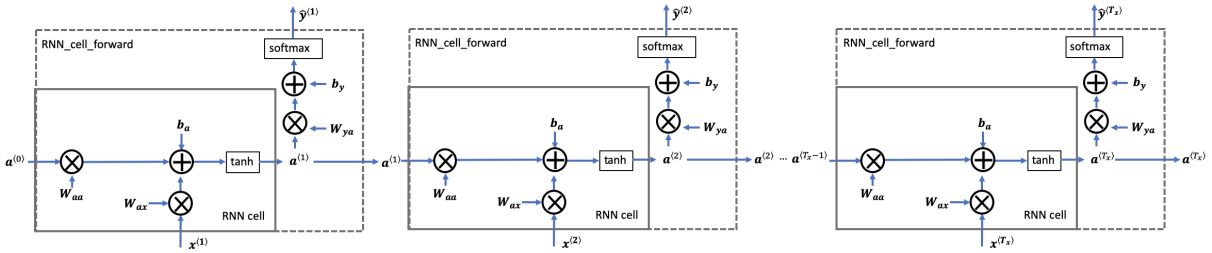


Figure 2: An overview of the RNN model.

- 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 store 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, as well as try to 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. The dataset that we are going to use is a list of the names for the existing chemical elements found in nature! So, our model's task is to create new names for new chemical elements that are likely (not) to be discovered by chemists in the not so near future!

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.

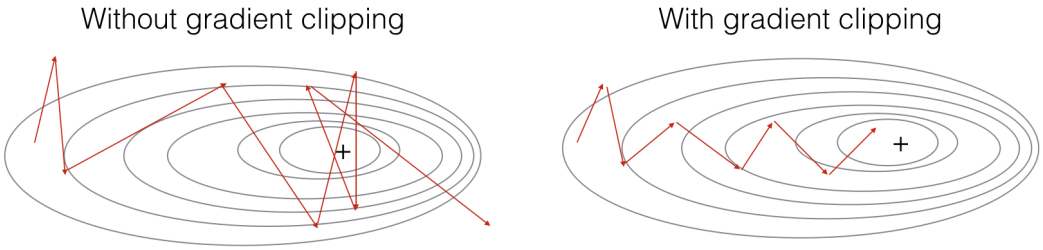


Figure 3: A depiction of the role of clipping in model generalization.

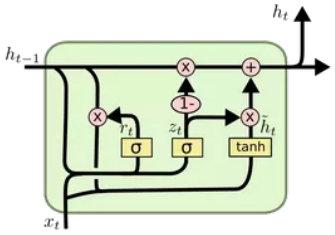
What you need to implement and answer for this part of the assignment:

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 them 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. What is the purpose of gradient clipping? What learning limitation we are aiming to solve?
- B. What do you observe about the new generated chemical names? How do you interpret the format of the new generated chemical element names?
- C. What happens when you train your model for the *movie titles* dataset? what do you observe and how do you interpret this observation?
- D. 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: A +10% will be given to the students that (a) implement the operations of a GRU cell (code), as shown in figure **??**, (b) briefly explain what is different between a GRU cell and a Long-Short Term Memory (LSTM) cell.

Important Notes

Setup your Anaconda, Python, TensorFlow environment

For this assignment, you need to create a new working environment with Python 3.5, TensorFlow 1.15 framework and Scikit-learn library in order to run the provided code.

To create a new environment follow the guidelines mentioned in the first tutorial to setup your Python environment successfully. To install Tensorflow/TensorBoard follow the installation guidelines mentioned in the 2nd Class Tutorial. DO NOT install the tensorflow/tensorboard versions shown in the Anaconda GUI, since these refer to 2.X versions.

Important: Due to major revisions between Tensorflow versions 1.X and the current 2.X, we will be sticking with Tensorflow 1.15 for our assignment. Please DO NOT rely on the documentation of the official Tensorflow page. Instead, you can find Tensorflow 1.15 and TensorBoard documentation in the following links:
`https://devdocs.io/tensorflow~1.15/`
`https://github.com/tensorflow/docs/tree/master/site/en/r1/guide`
`https://itnext.io/how-to-use-tensorboard-5d82f8654496`
`https://github.com/tensorflow/tensorflow/blob/master/tensorflow/tensorboard/README.md`

Also, you can search the web for answers regarding Tensorflow commands, BUT do not forget to specify the version we are using.

Submission info

- Create a .pdf or .doc file to report the resulting scores, images/figures and any other comments or description of your work you may need to submit. Do not forget to include your name, login, ID in the report. Save this file in your working folder.
- Use zip/rar/gz to compress your working folder and rename it to cs587_mylogin_assignment4.xxx in order to submit a single file.
- You will upload your submission in the course's space in the e-learn platform.
- Uploading will not be available after the deadline date-time.

Troubleshooting

In case you find any errors/bugs in the code please send an email to kbacharidis@csd.uoc.gr or the mailing list.