Homework 3
Gated Recurrent Unit Cells

11-785: Introduction to Deep Learning (Spring 2019)

OUT: March 12, 2019

DUE: **March 31, 2019, 11:59 PM**

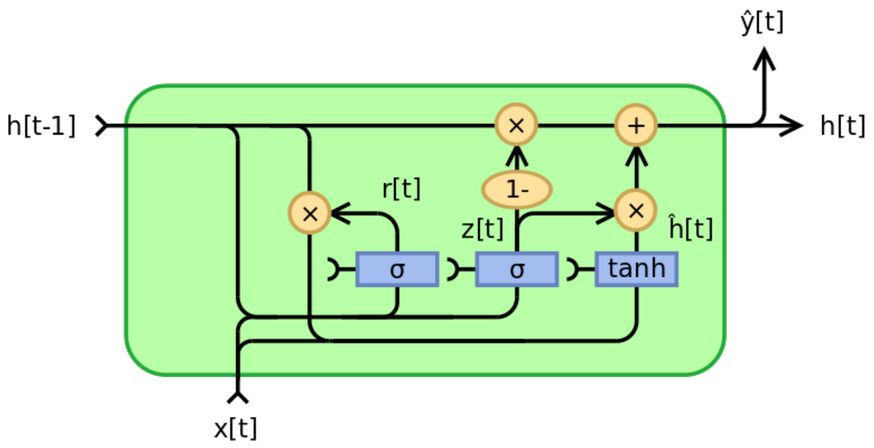
Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism.
  - You are allowed to talk with / work with other students on homework assignments
  - You can share ideas but not code, you must submit your own code. All submitted code will be compared against all code submitted this semester and in previous semesters using MOSS.
- **Overview:**
  - **Part 1**: All of the problems in Part 1 will be graded on Autolab. You can download the starter code from Autolab as well. This assignment has 100 points, total.
  - **Part 2**: This section of the homework is an open ended competition hosted on Kaggle.com, a popular service for hosting predictive modeling and data analytics competitions. All of the details for completing Part 2 can be found on the competition page.
- **Submission:**
  - **Part 1**: The compressed handout folder hosted on Autolab contains one python file, gru.py. File gru.py contains outline class of GRU_Cell and helper classes of Sigmoid and Tanh activations. You need to implement the GRU_Cell class's forward and backward functions according to the specification provided in this write-up. Your submission must be titled handin.tar (gzip format) and it is minimally required to contain a directory called hw3, which contains the gru.py file implementing the functions specified in the write-up. Please do not import any other external libraries other than NumPy and the default python packages, as extra packages that do not exist in the autograder image will cause a submission failure.
  - **Part 2**: See the the competition page for details.

1 Introduction

In part one of this assignment you will make a recurrent neural network, specifically you will replicate a portion of the torch.nn.GRUCell interface. GRUs are used for a number of tasks such as Optical Character Recognition and Speech Recognition on spectograms using transcripts of the dialog. This homework is to develop your basic understanding of Backpropagating through a GRUCell, which can potentially be used for GRU networks to grasp the concept of Backpropagation through time (BPTT).

2 GRU: Gated Recurrent Unit



You will be implementing the forward pass and backward pass for a GRUCell using python and numpy in this assignment, analogous to the Pytorch equivalent nn.GRUCell. The equations for a GRU cell looks like the following:

$$z_t = \sigma(W_{zh}h_{t-1} + W_{zx}x_t) \tag{1}$$
$$r_t = \sigma(W_{rh}h_{t-1} + W_{rx}x_t) \tag{2}$$
$$\tilde{h}_t = tanh(W_h(r_t \otimes h_{t-1}) + W_x x_t) \tag{3}$$
$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \tilde{h}_t \tag{4}$$

Where x_t is the input vector at time t, and h_t the output. **There are other possible implementations, you need to follow the equations for the forward pass as shown above.** If you do not, you might end up with a working GRU and zero points on autolab. Do not modify the `init` method, if you do, it might result in lost points.

Similar to previous assignments, you will be implementing a Python class, `GRU_Cell`, found in `gru.py`. Specifically, you will be implementing the `forward` and the `backward` methods.

2.1 GRU Cell Forward (30 Points)

In this section, you will implement the `forward` method of the `GRU_Cell`. This method takes **2 inputs**: the observation at the current time-step, x_t, and the hidden state at the previous time-step h_{t-1}.

Use Equations 1-4 to implement the forward method, and return the value of h_t.

Hint: Store all relevant intermediary values in the forward pass.

2.2 GRU Cell Backward (70 Points)

The `backward` method of the `GRU_Cell`, is the most time-consuming task of this homework.

This method takes as input delta, and must calculate the gradients wrt the parameters and returns the derivative wrt the inputs, x_t and h_t, to the cell.

The partial derivative input you are given, `delta`, is the summation of the derivative of the loss wrt the *input* of the *next layer* $x(l+1,t)$ and the derivative of the loss wrt the input hidden-state at the *next time-step* $h(l,t+1)$.

Using these partials, you will need to compute the partial derivative of the loss wrt each of the *six weight matrices* (see Equations 1-4), and the partial derivative of the loss wrt *the input* x_t, and *the hidden state* h_t.

Specifically, there are **eight gradients that need to be computed**:

1. $\frac{\partial L}{\partial W_{rx}}$, stored in `self.dWrx`
2. $\frac{\partial L}{\partial W_{rh}}$, stored in `self.dWrh`
3. $\frac{\partial L}{\partial W_{zx}}$, stored in `self.dWzx`
4. $\frac{\partial L}{\partial W_{zh}}$, stored in `self.dWzh`
5. $\frac{\partial L}{\partial W_{x}}$, stored in `self.dWx`
6. $\frac{\partial L}{\partial W_{h}}$, stored in `self.dWh`
7. $\frac{\partial L}{\partial x_t}$, returned by the method
8. $\frac{\partial L}{\partial h_t}$, returned by the method

You **will** need to derive the formulae for the back-propagation in order to complete this section of the assignment.