

Homework 3

Gated Recurrent Unit Cells and Beam Search Decoder

11-785: INTRODUCTION TO DEEP LEARNING (FALL 2019)

OUT: Sunday October 20th, 2019

DUE: **November 9th, 2019**

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 1 Bonus:** All of the problems in Part 1 Bonus will be graded on Autolab. You can download the starter code from Autolab as well. This assignment has 10 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.

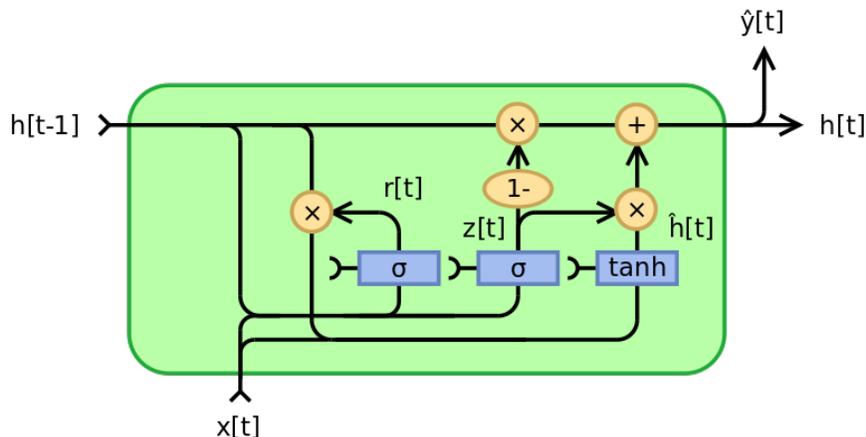
- **Submission:**

- **Part 1:** The compressed handout folder hosted on Autolab contains two python files, gru.py and BeamSearch.py. You need to the classes and 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 implemented files gru.py and BeamSearch.py files. 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 1 Bonus:** TBD
- **Part 2:** See the the competition page for details.

1 Part 1A: GRU

In this part 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 spectrograms 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).

1.1 GRU Formulation



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:

$$\mathbf{z}_t = \sigma(\mathbf{W}_{zh}\mathbf{h}_{t-1} + \mathbf{W}_{zx}\mathbf{x}_t) \quad (1)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{rh}\mathbf{h}_{t-1} + \mathbf{W}_{rx}\mathbf{x}_t) \quad (2)$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h(\mathbf{r}_t \otimes \mathbf{h}_{t-1}) + \mathbf{W}_x\mathbf{x}_t) \quad (3)$$

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z}_t) \otimes \mathbf{h}_{t-1} + \mathbf{z}_t \otimes \tilde{\mathbf{h}}_t \quad (4)$$

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.

1.2 GRU Cell Forward (25 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, \mathbf{x}_t , and the hidden state at the previous time-step \mathbf{h}_{t-1} .

Use Equations 1-4 to implement the forward method, and return the value of \mathbf{h}_t .

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

1.3 GRU Cell Backward (25 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, \mathbf{x}_t and \mathbf{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* $\mathbf{x}_{l+1,t}$ and the derivative of the loss wrt the input hidden-state at the *next time-step* $\mathbf{h}_{l,t+1}$.

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* \mathbf{x}_t , and *the hidden state* \mathbf{h}_t .

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

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

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

1.4 GRU Inference (10 Points)

In this section you will use the GRU Cell implemented in the previous section and a linear layer (provided to you) to compose a neural net. This neural net will unroll over the span of inputs to provide a set of logits per time step of input.

You will compose the neural network with the `CharacterPredictor` class in `gru.py` and use the `inference` function (also in `gru.py`) to use the neural network that you have created to get the outputs.

The `inference` accepts the `net`, an instance of `CharacterPredictor`, and `inputs`, a tensor of shape `seq_len × feature_dim`. You will unwrap the `net` to `seq_len` time steps and return a logits sequence of shape `seq_len × num_classes`.

2 Part 1B: Greedy Search and Beam Search (10 + 30 Points)

In this part you will implement greedy search and beam search. Greedy search greedily picks the label with maximum probability at each time step to compose the output sequence. Beam search is a more effective decoding technique to obtain a sub-optimal result out of sequential decisions, striking a balance between a greedy search and an exponential exhaustive search by keeping a beam of top- k scored sub-sequences at each time step. In the context of CTC, you would also consider a blank symbol and repeated characters, and merge the scores for several equivalent sub-sequences. For details you are referring to P186-195 in Lecture 15 slides.

You need to implement functions `GreedySearch` and `BeamSearch` in file `BeamSearch.py`. For both the functions you will be provided with `SymbolSets` which is a list of symbols that can be predicted **except for the blank symbol**; `y_probs`, an array of shape $(\text{len}(\text{SymbolSets}) + 1, \text{seq_length}, \text{batch_size})$ which is the probability distribution over all symbols **including the blank symbol** at each time step (**note that probability of blank for all time steps is the first row of `y_probs`**). The `batch_size` is `1` for all test cases.

3 Part 1 Bonus: Forward-Backward and BPTT

We will give you a bonus part for this assignment, including forward-backward algorithm for computing posteriori-based divergence, and BPTT. We will release a separate writeup later for details and you could gain up to 10 bonus points for submission.