Project 3 - CS 4740 Fall 2018
Neural Models in NLP

Due Date: Sunday, 11/11/2018 11:59 PM

1 Introduction

In this project, you will explore neural models for NLP tasks. Our goal is to introduce you to the challenges in implementing neural models and the variety of decisions one has to make when dealing with how to build these kinds of model. In addition, we hope to demonstrate that working with neural networks in NLP is challenging for reasons that go beyond the model and that unlike domains such as ML/CV, NLP neural models tend to require different optimization techniques/tuning methods.
For this assignment, you will work on two independent components related to neural models. The first will be a set of **debugging** tasks, where you will be provided with a neural model that has some error and be asked to identify and fix the model. The second part will involve the **seq2seq** architecture, a standard architecture for approaching sequence-to-sequence tasks (hence the name) in NLP, which makes use of 2 recurrent neural networks (often described as the encoder-RNN and decoder-RNN). In addition, you will be introduced to the concept of attention, which is in an innovation that has led to dramatic performance improvements across a wide range of neural models for NLP.
Beyond the course texts, we recommend the following literature on neural models for NLP:
Yoav Goldberg's Neural Network Methods for Natural Language Processing:
Jacob Eisenstein's NLP notes
Sebastian Ruder's Notes on Best Practices for NLP

2 Setup

In NLP and other fields that apply deep learning methods, the prevalence of deep learning/machine learning/scientific computing frameworks has become apparent. These frameworks allow for the standardization of core operations and models to promote modular programming practices as well as for allowing better optimization of computational resources (in research, this often refers

to preparing computations to be executed quickly on a GPU, though for this assignment all tasks should be able to be done on your local CPU).
While there is a great deal of debate regarding which framework (TensorFlow, PyTorch, DyNet, MXNET, etc) is optimal, we will use PyTorch for this assignment. One thing to note is that all of these frameworks are written in Python and almost all of modern deep learning (and its applications to NLP) is done in Python. If you have questions regarding setting up PyTorch or using Python, feel free to come to TA office hours (please don't trouble Professor Cardie regarding this).
We have released the code with a specific directory structure. You should not need to modify the directory structure to run the code and please let us know (via a private Piazza post/in OH) if there are any problems.

3 Debugging

In this part of the assignment, you will be given a near-functional implementations of a neural model for various core for a sentence-level text classification task.
The goal of the task is to identify whether a sentence is objective (which is defined by the dataset creators as appearing in a quote from a movie review) or subjective (which is defined by the dataset creators as appearing in the description of the plot). We did not cover this task in class but it is fairly straightforward and you do not need to concern yourself with the intricacies of this task (as it will not be needed for this assignment).
If you are interested, you can read more here, which is a paper by Bo Pang (a former Cornell PhD) and Professor Lillian Lee.
The implementation we provide has 4 errors! You are asked to do the following:

1. Identify all errors in the implementation (details on how to describe this in the report are below).
2. Fix these errors in the implementation in a reasonable way.
3. Provide a description of how the solution you implement fixes the original problem.

In the context of the report, this will mean having one subsection for each error with these three pieces. The fixed files will be submitted on CMS whereas the written description should be included in the report. In addition, please include the snippet of text you changed in the report (but do not include the entirety of the file, only the parts you change).
Please remove all debugging statements/print statements you introduced in your final submissions (on CMS and in the report).
All of the errors can be fixed using a few lines (< 5) but you are allowed to modify more if you like.
To provide some guidance, the errors can be in any stage of the model pipeline, which we enumerate as follows:

1. Data
2. Model
3. Training/Learning/Optimization
4. Inference
5. Evaluation

In addition, we guarantee that the hints provided are intended to be helpful and that the errors are not pathological (i.e. the error won't be invoked by having a sequence that includes specifically "42" as a token or that describes "Marseille" with negative sentiment). All of the errors can also be guaranteed to be strictly about the correctness of the implementation (i.e. the problem will not be that a model is too slow or too memory intensive) and the errors are local to the implementation itself (i.e. the error won't be that the implementation is not competitive with state-of-the-art methods/the input dataset is very small for the task being done). One final observation is you should not necessarily rely on the model performance as the sole indication that you have found and fixed the errors correctly.
Due to the nature of this section of the assignment, we remind you that there can be no collaboration between teams (and we also want to emphasize that you will not learn anything if someone else tells you what the bug is). In addition, by its nature, we cannot provide meaningful assistance in office hours for this section, so please realize that our advice will be extremely general for this section. We also ask that you do not discuss this section on Piazza. If you find it absolutely necessary, please only make private posts (and do not do so excessively).

4 Seq-2-Seq

The seq-2-seq framework is a popular and effective neural framework for handling sequence-to-sequence tasks in NLP. It has been successively applied to many such tasks, including neural machine translation (NMT) and automatic summarization. In this part of the project, we will work through 2 parts to familiarize you with the behavior of NLP neural pipelines and the encoder/decoders RNN's involved in this specific pipeline. In addition, you will be introduced to **attention** which is a core innovation in making these models competitive.

4.1 Toy Problems

You will begin by considering some toy tasks (sequence copying, sequence reversing, sequence sorting) that can be understood as sequence-to-sequence tasks. These tasks are fairly intuitive but to formalize them, we give a simple formalism below (all inputs can be assumed to be at least one token long):

1. Sequence Copying:
   Given an input sequence $\vec{x}$ which corresponds to the tokens $w_1, \ldots, w_n$, the correct output of the model should be $\vec{y}$ where $\vec{y} = \vec{x} = w_1, \ldots, w_n$. In other words, you can think of the function $(\vec{x} \mapsto \vec{y})$ as the identity function.

2. Sequence Reversing:
   Given an input sequence $\vec{x}$ which corresponds to the tokens $w_1, \ldots, w_n$, the correct output of the model should be $\vec{y}$ where $\vec{y} = w_n, w_{n-1}, \ldots, w_2, w_1$.

3. Sequence Sorting:
   Given an input sequence $\vec{x}$ which corresponds to the tokens $w_1, \ldots, w_n$, the correct output of the model should be $\vec{y}$ where $\vec{y}$ is a permutation of $\vec{x}$ that is the "sorted" result for $\vec{x}$. In the training dataset we provide, the inputs are guaranteed to be natural numbers (i.e. $w_i \in \mathbb{N}$) and the sorted output is the sequence $\vec{y} = w'_1, \ldots, w'_n$ such that $w'_i \leq w'_{i+1} \forall i \in [1, n-1]$ (i.e. you take in a sequence of numbers and output the numbers sorted).

4.1.1 Experiments

We have already implemented a fully-functional seq-2-seq architecture for you. In this part of the assignment, you must construct 3 data-related experiments. 2 of the experiments must be conducted using both the sequence-copying tasks and sequence reversing tasks and 1 of the experiments must be conducted using the sequence sorting task. An explanation of what a "data-related experiment" is is provided below. You may not repeat the "same experiment" as is described below.
A data-related experiment is this assignment will mean producing a test dataset and evaluating model performance (trained on the datasets we provide) on the test dataset we release as well as the test dataset you produce (for the appropriate tasks). In your report, you should include a description of how you constructed the dataset, how the model performs quantitatively on both datasets, examples to qualitatively demonstrate differences in model performance, and a justification for the behavior you observe. As an example, one experiment you might try is seeing how the model performs as a function of sequence length. Hence, you will construct a dataset with sequences of different length from the sequence lengths you see in the test dataset we provide (and then perform the analysis described above).

4.1.2 Questions

In addition to the experiments, you will need to provide answers in your report to the following five questions:

1. Think about the three toy tasks. In some sense, they are very similar, as they take in a sequence as an input and produce a sequence as an output

that is some kind of permutation of the input. Rank the task difficulties for:
a) A human
b) A classical/deterministic algorithm
c) The neural model we implement
Explain why the neural model either is in agreement with the difficulty judgments for humans/deterministic algorithms or not.

2. Why are the toy tasks we describe above *easier* than sequence-to-sequence tasks in NLP?
   Your answer should address both of the following points:

   (a) Problems in natural language understanding and how this is different between these toy tasks and sequence-to-sequence tasks in NLP

   (b) Inherent differences about the specifics of these tasks/explicit properties of these tasks as machine learning problems (Hint: Look at the code we release, we provided some hints about this)

3. Why are some/all of the toy tasks we describe *harder* than sequence-to-sequence tasks in NLP?
   Hint: Think about inputs to the RNN decoder and how these inputs are relevant for the toy tasks as compared to sequence-to-sequence tasks in NLP.

4. How does a neural approach to these toy tasks differ from classical algorithms for these tasks?
   You answer should address all of the following points:

   (a) Run-time. How does the asymptotic runtime for these neural models for toy tasks (and what is the asymptotic $\mathcal{O}(\cdot)$ (big-O) runtime for these models) differ from the algorithms for other tasks. Sequence-copying and sequence-reversing have $\mathcal{O}(n)$ algorithms for completing them and sequence-sorting have standard $\mathcal{O}(n\log(n))$ algorithms (you can use quicksort/mergesort as a good reference frame). In addition, talk about the practical runtimes.

   (b) Guarantees. How do the guarantees we have on model performance differ?

   (c) Information. In the case of sequence-copying, what information on the inputs is provided directly to the neural model? How does this information differ from the information provided to a standard sorting algorithm (quicksort, mergesort).

5. If you look at the code in *model.py*, you will see there is a binary flag for "teacher forcing" (you may find it useful to read about teacher forcing in RNN models, we provide a link in the code itself). Train all three models with and without teacher forcing (turning the flag to $True$ or $False$). Record the changes in quantitative performance on the test set we release

and explain the behavior you observe.

4.2 Attention

Attention is a highly popular component in modern neural systems in NLP. Attention is a beneficial and often necessary augment to achieve competitive results using seq2seq architectures.

4.2.1 Introduction

In the seq2seq architecture, for each decoder step (so each step of producing the output summary), we use some representation of the input sequence (this is an output from the encoder model). In the seq2seq implementation we provide, we use the last encoder state as the initialization of the decoder. This is a conventional approach but this implies that representation of the input is invariant during all decoder steps. One could argue that instead we want a dynamic representation of the input for each step in decoding. As you will see, attention is a natural approach for doing this.

4.2.2 Implementation

As discussed in the previous section, there may be problems in how we represent the input sequence within the decoder. To reconcile this, we introduce the concept of attention, which is learning a sequence of weights corresponding to each input token. Specifically, the goal of an attention mechanism is to produce a vector $\vec{a}$ such that $\sum a_i = 1$ and $\vec{a} \in \mathbb{R}^n$ where n is the number of input tokens. We will then use this to produce an input to the decoder. Instead of using the representation produced by the encoder, we will instead take a weighted-average over the encoder hidden states. That is, the input to the decoder will be $\vec{e} = \sum_{i=1}^{n} a_i \vec{e}_i$ where $\vec{e}_i$ is the encoder state corresponding to position i. We can think of the original approach that was used (using the final encoder state) as being a degenerate form of attention where all the attention is allocated to the final state (i.e. an attention vector $\vec{a} = [0, 0 \ldots, 0, 1]^T$).

There are two standard approaches for computing attention of this form, known as multiplicative and additive attention:

1. Multiplicative
2. Additive

You will need to implement both types of attention. In addition, carry out an experiment that demonstrates the performance difference of the model without attention, with multiplicative attention, and with additive attention (and conduct the analysis in the same way as the earlier section on experiments). Finally, compute how the asymptotic performance of the model changes with each of the two types of attention as well as how it changes from not having

attention.
Absolutely no external code may be included in this assignment. You may not use existing implementations for these attention methods and must write everything from scratch (relying on the primitives provided by PyTorch, NumPy, and Python). All submissions will be run through MOSS and will also be strenuously looked at by the TAs.

5 Report

Your report (6-7 pages should suffice) should be divided into sections as follows:

1. **Debugging**
   For each error, describe the error. Then, describe how you fix it and include the code snippet of your fix/what you changed.
2. **seq2seq**
   (a) Toy Problems:
   For each of the three data experiments, do the following:
      i. Description of your test datasets (how it was constructed and what you are trying to evaluate)
      ii. Quantitative performance on both datasets
      iii. Examples of differences in performance
      iv. Justification for observed behavior

   (b) Questions:
   For each of the five questions, provide an answer (that addresses all of the referenced points if points are provided)

   (c) Attention:
      i. Implementation:
      The code should be submitted on CMS and should not be in the report.
      ii. Experiment:
      Include the same points as done in the first set of experiments (under Toy Problems)
      iii. Asymptotic Performance:
      Provide the asymptotic complexity (using big-O notation) and justify it. Appropriately identify what each of the parameters is (i.e. if you use the variable n, identify what n corresponds to).
3. **Workflow**
   Briefly describe how you distributed the work and report if the workload was unfairly distributed.
4. **Feedback (optional)**
   This is a new assignment that was designed from scratch for this semester.

Please briefly describe what you thought of the assignment, what you learned, and what you would like for us to change/retain for future semesters. We may offer extra credit for teams that complete this section thoughtfully.

6 Grading Guide

Unlike the previous assignments, we do not intend to release a tentative grading guide for this assignment as it is a new assignment. Instead, we will provide you a reasonable guarantee that the grading criterion we use in grading the reports will correspond to explicit points we mention in the "Report" section (as well as your implementation correctness/overall style and clarity)

6.1 Things to avoid

As this is a new assignment, we are not sure what common errors will be. That being said, we ask that you monitor all *pinned* posts on Piazza and we will try to provide helpful hints/clarify common misconceptions on Piazza as the assignment carries on. You are responsible to read all changes introduced on Piazza in *pinned* posts (and we recommend being familiar with *all* of the content on Piazza, as it is there for your benefit!). In addition, we do not intend to change this document once the assignment has been released. All changes will instead be announced on Piazza.

7 What to Submit

7.1 Report/Gradescope

Refer to the section on "Report"

7.2 Code/CMS

1. **Debugging**
   Submit the modified files we released with the changes you made. Only submit files you changed.
   Do not submit files you did not change!

2. **seq2seq**
   Submit the modified files with the implementations of additive and multiplicative attention.
   Please also ensure that one of the following two points is met:

   (a) PREFERRED!
   The attention method used should be able to set as a command line flag. Ideally, this is done in the following formats:
   No attention: *python main.py*

Additive attention: *python main.py add*
Multiplicative attention: *python main.py mul*

(b) NOT PREFERRED!
If you have trouble setting the attention methods to be toggled by command line arguments, please provide a README.

We will deduct points if there is no README and we cannot set the attention method used using the command line pattern we provide above.