CS4740: Intro to NLP
Project 3: Neural Networks

Due **Friday, November 8, 11:59PM**

Overall Goal

In this project we will consider **neural networks**: first a *Feedforward Neural Network* (FFNN) and second a *Recurrent Neural Network* (RNN), for performing a 5-class **Sentiment Analysis** task.

The project is divided into parts. In **Part 1**, you will be given an implementation for a FFNN and be asked to debug it in a specific way. In **Part 2**, you will then implement an RNN model for performing the same task. In **Part 3**, you will analyze these two models in two types of comparative studies and in **Part 4** you will answer questions describing what you have learned through this project. You also will be required to submit a description of libraries used, how your group divided up the work, and your feedback regarding the assignment (**Part 5**).

We **do not** provide a grading rubric (a mapping from submitted components to point values) like in past assignments (since this assignment is quite new) but we are quite **explicit about various requirements on a per-section basis**. We believe this solution resolves many concerns about the clarity in assignments for Projects 1 and 2 while providing us with flexibility for handling issues that we may not have entirely foreseen, e.g. how easy/difficult the new project components are. The assignment comes with a reasonably strong **guarantee** that we will not deviate from what we ask for in each section. We reserve the right to provide opportunities for **extra credit** should they arise.

Logistics: You should work in groups of 2 or 3 students. Students in the same group will get the same grade. Thus, you should make sure that everyone in your group contributes to the project. Also, **remember to form groups on BOTH CMSX and Gradescope** or not all group members will receive grades!!! If you fail to do this, you *will* lose points. Similarly, ensure that pages are selected correctly on Gradescope or you *will* lose points on the assignment.

Advice: As always, the report is important! The report is where you get to show that you understand not only *what* you are doing but also *why* and *how* you are doing it. So be clear, organized and concise; avoid vagueness and excess verbiage. Spend time doing error analysis for the models. This is how you understand the advantages and drawbacks of the systems you build. The reports should read more like the re-

search papers we have been reading for the critiques, not a lab report or list of facts or specifications.

This assignment, at its core, involves training and experimenting with neural models. We have made a very committed effort towards making these models run quickly but this is not easy to guarantee/maintain with neural models. Procrastinating on this assignment is likely to be **substantially more detrimental** than on past assignments.

Policy: This assignment, while in some ways similar to the third assignment created in Fall 2018, is a new assignment. It requires interfacing with `PyTorch` and involves added complexities that did not arise in the previous assignments. As such, we may need to make changes. You are **responsible for all pinned posts on Piazza**. We will not change this document (and if we do, it will be referenced on Piazza as well).

Reference Sources

For the project, you will be dealing with neural network models for NLP. Both of the course texts we have used throughout the semester thus far, i.e. Jurafsky & Martin and Eisenstein, provide introductions to neural models. The lectures will most directly reflect what is assigned in the readings from Jurafsky & Martin. The Eisenstein text will provide some more nuanced points, however it is worth remembering that there is a **third course text specifically dedicated to neural networks in NLP**, i.e. the book written by Yoav Goldberg. In particular, the sections on Feedforward Neural Networks (**Section 4**), Practical Implementation Advice (**Section 5.2**), and Recurrent Neural Networks (primarily **Section 14.1 - 14.3**) may be quite useful for this assignment.
Alongside the course text, this assignment will introduce you to the `PyTorch` framework and the well-written `PyTorch` documentation is likely also to be useful[1]. As is reiterated below, these are **the only sources you may consult** for the assignment (in addition to the lectures and associated slide decks).

Setup

In NLP and other fields that apply deep learning methods, the prevalence of deep learning frameworks has greatly advanced the ease and consistency with which neural models are implemented. 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.

While there is a great deal of debate regarding which framework (`TensorFlow`, `PyTorch`, `DyNet`, etc.) is optimal, we will use `PyTorch` for this assignment[2]. One thing to note is that all of these frameworks are written in Python and most 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).

[1] `https://pytorch.org/docs/stable/index.html`
[2] Recently, Horace He, a CS undergrad here at Cornell, wrote this excellent blog post.

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) if there are any problems.

Dataset

You are given (via Piazza) access to a set of Yelp reviews. These reviews are associated with a rating $y \in \mathcal{Y} := \{1, 2, 3, 4, 5\}$[3]. For this project, given the review text, you will need to predict the associated rating, y. This is sometimes called *fine-grained* sentiment analysis in the literature; we will simply refer to it as sentiment analysis in this project.

We will minimally preprocess the reviews and handle tokenization in what we release. For this assignment, we do not anticipate any further preprocessing to be done by you. Should you choose to do so, it would be interesting to hear about in the report (along with whether or not it helped performance), but it is not a required aspect of the assignment.

Part 1: Feedforward Neural Network

In the released code (via Piazza), you will find three files relevant to **Part 1**.

1. `data_loader.py`
   As the name suggests, this file loads the data from the dataset files and handles other preprocessing. You will **not** need to change this file and should **not** change this file throughout the assignment.

2. `ffnn.py`
   This contains the model, and code that uses the model, for **Part 1**. This will be discussed subsequently[4].

3. `main.py`
   This is mostly a convenience file for running the model. This file allows you to specify configurations such as whether you are using the RNN and/or FFNN as well as some parameters. You will need to change these configurations for different experiments but there is no error/bug in this file for the purposes of **Part 1**.

In `ffnn.py`, you will find a Feedforward Neural Net serving as the underlying model for performing sentiment analysis. Assume that an omniscient oracle has told you there are **4 fundamental errors** in this implementation. They may be anywhere in the file (unless otherwise indicated). Your objective is to *find* and *fix* each of these errors and to include in the report a description of the original error along with the fix.
To help your efforts, the oracle has provided you with additional information about the properties of the errors as follows:

[3] In the assignment code, we use $\{0, 1, 2, 3, 4\}$ to more naturally align with Python's zero-indexing.
[4] **Remark:** We have written the code in a fairly monolithic style to maintain simplicity for the assignment.

- CORRECTNESS
  Each error causes the code to be strictly incorrect. There is absolutely no ambiguity that the errant code (or missing code) is incorrect. This means errors are not due to the code being inefficient (in run-time or in memory).

- LOCALIZED
  Each error can be judged to be erroneous by strictly looking at the code (along with your knowledge of machine learning as taught through this course). The errors therefore are not due to the model being uncompetitive in terms of performance with state-of-the-art performance for this task nor are they due to the amount of data being insufficient for this task in general.

- GENERAL
  Each error is general in nature. They will not be triggered by the model receiving a pathological input, i.e. they will not be something that is triggered specifically when NLP is referenced with negative sentiment.

- FUNDAMENTAL
  Each error is a fundamental failure in terms of doing what is intended. This means that errors do not hinge on nuanced understanding of specific `PyTorch` functionality. This also means they will not exploit properties of the dataset in a subtle way that could only be realized by someone who has comprehensively studied the data.

The bottom line: the errors should be fairly obvious. The oracle further reminds you that **performance/accuracy of the (resulting) model should not be how you ensure you have debugged successfully**. For example, if you correct some, but not all, of the errors, the remaining errors may mask the impact of your fixes. Further, performance is not guaranteed to improve by fixing any particular error. Consider the case where the training set is also employed as the test set; performance will be very high but there is something very wrong. And fixing the problem will reduce performance.

In fixing each error, the oracle provides some further insight about the fixes:

- MINIMAL
  A reasonable fix for each error can be achieved in < 5 lines of code being changed. We do not require you to make fixes of 4 or fewer lines, but it should be a cause for caution if your fixes are far more elaborate.

- ILL-POSED
  While the errors are unambiguous, the method for fixing them is under-specified: you are free to implement any reasonable fix and all such fixes will equally receive full credit.

Part 1: Report

For **Part 1**, your report should have a description of the error, a description of your fix, and a code snippet indicating the fix for each of the **4** errors. Absolutely nothing else is needed for this part. We anticipate this part should take at most 1 page in the report and does not entail submitting anything to CMSX.

Part 1: Tips

We do not assume you have any experience working with neural networks and/or debugging them. As such, you may discover this process, while topically similar, is quite different from debugging in general software engineering and from debugging in other domains such as algorithms and systems.
What we suggest is you systematically step through the code and simultaneously (perhaps by physically drawing it out) describe what the computations *mean*. What you are looking for is where code differs from what is expected.

Part 1: Rules

For **Part 1**, you will not be able to ask any questions on Piazza and we will be unable to provide any meaningful advice in office hours. This is the nature of debugging, it is unlikely anyone can give you specific advice for most problems you encounter and we have already provided general tips in the preceding section. If you absolutely must ask a question or you believe there is some kind of issue with the assignment for this part, please submit a private Piazza post and we will respond swiftly.
As a reminder, communication about the assignment *between* distinct groups is not permitted and is a violation of the Academic Integrity policy. For this assignment, we will be extremely stringent about this, given that debugging is entirely pointless if someone else in a different group tells you where the error is.

Part 2: Recurrent Neural Networks

Recurrent neural networks have been the workhorse of NLP for a number of years. A fundamental reason for this success is they can inherently deal with *variable* length sequences. This is axiomatically important for natural language; words are formed from a variable number of characters, sentences from a variable number of words, paragraphs from a variable number of sentences, and so forth. This differs from a field like Computer Vision where images are (generally) of a fixed size.
This is also very different scenario than that of the classifiers we have studied (e.g. Naive Bayes, Perceptron Learning, Feedforward Neural Networks), which take in a fixed-length vector.
To clarify this, we can think of the *types* of the mathematical functions described by a FFNN and an RNN. What is pivotal in what follows is that k need not be constant across examples.
FFNN.
INPUT: $\vec{x} \in \mathbb{R}^d$
MODEL OUTPUT: $\vec{z} \in \mathbb{R}^{|\mathcal{Y}|}$
FINAL OUTPUT: $\vec{y} \in \mathbb{R}^{|\mathcal{Y}|}$
$\vec{y}$ satisfies the further constraint of being a probability distribution, i.e. $\sum_{i \in |\mathcal{Y}|} \vec{y}[i] = 1$ and $\min_{i \in |\mathcal{Y}|} \vec{y}[i] \leq 1$, which is achieved via `Softmax` applied to $\vec{z}$.

RNN.

INPUT: $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_k; \vec{x}_i \in \mathbb{R}^d$
MODEL OUTPUT: $\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_k; \vec{z}_i \in \mathbb{R}^h$
FINAL OUTPUT: $\vec{y} \in \mathbb{R}^{|\mathcal{Y}|}$
$\vec{y}$ satisfies the further constraint of being a probability distribution, i.e. $\sum_{i \in |\mathcal{Y}|} \vec{y}[i] = 1$ and $\min_{i \in |\mathcal{Y}|} \vec{y}[i] \geq 0$, which is achieved by the process described later in this report and as you have seen in class.

Intuitively, an RNN takes in a sequence of vectors and computes a new vector corresponding to each vector in the original sequence. It achieves this by processing the input sequence one vector at a time to (a) compute an updated representation of the entire sequence (which is then re-used when processing the next vector in the input sequence), and (b) produce an output for the current position. The vector computed in (a) therefore not only contains information about the current input vector but also about the previous input vectors. Hence, $\vec{z}_j$ is computed after having observed $\vec{x}_1, \ldots, \vec{x}_j$. As such, a simple observation is we can treat the last vector computed by the RNN, i.e. $\vec{z}_k$, as a representation of the entire sequence. Accordingly, we can use this as the input to a single-layer linear classifier (Equation 1) to compute a vector $\vec{y}$ as we will need for classification.

$$\vec{y} = \texttt{Softmax}(W\vec{z}_k); W \in \mathbb{R}^{|\mathcal{Y}| \times h} \tag{1}$$

Part 2: Implementation

Similar to **Part 1**, you will find three files relevant in **Part 2**.

1. `data_loader.py`
   We do not envision that it will be useful to change this for **Part 2**.

2. `rnn.py`
   As you will see, we have included some stubs to help give you a place to start.

3. `main.py`
   Perhaps more so than in **Part 1**, you will need to change the parameters you specify.

Additionally, we remind you that **Part 1** furnishes a near-functional implementation of a similar neural model for the same task. If you successfully do **Part 1** correctly, it will be wholely functional. Using it as a template for **Part 2** is both prudent and suggested.

Part 2: Report

For **Part 2**, your report should have a description of each major step of implementing the RNN accompanied by the associated code-snippet. Each step should have an explanation for why you decided to do something (when one could reasonably do the same step in a different way); your justification will not be based on empirical results in this section but should relate to something we said in class, something mentioned in any of the course texts, or some other source (i.e. literature in NLP or official `PyTorch`

documentation). **Unjustified, vague, and/or under-substantiated explanations will not receive credit.**

Some steps that should definitely be included:

1. REPRESENTATION
   Each $\vec{x}_i$ needs to be produced in some way and should correspond to word i in the review. This is different from the text classification approaches we have studied, where the entire document/review is represented with a single vector. Where and how is this being done for the RNN?

2. INITIALIZATION
   There will be weights that you will update in training the RNN. Where and how are these initialized?

3. TRAINING
   You are given the entire training set of N examples. How do you make use of this training set? How does the model modify its weights in training (this likely entails somewhere where gradients are computed and somewhere else where these gradients are used to update the model)?

4. MODEL
   This is the core model code, i.e. where and how you apply the RNN to the $\vec{x}_i$.

5. LINEAR CLASSIFIER
   Given the outputs of the RNN, how do you consume these to actually compute $\vec{y}$?

6. STOPPING
   How does your training procedure terminate?

7. HYPERPARAMETERS
   To run your model, you must fix some hyperparameters, such as h (the hidden dimensionality of the $\vec{z}_i$ referenced above). Be sure to exhaustively describe these hyperparameters and why you set them as you did (this almost certainly will require some brief exploration: we highly suggest the course text by Yoav Goldberg as well as possibly the `PyTorch` official documentation). Be sure to accurately cite either source.

You will also submit the entire code on CMSX. We will primarily grade the code via the snippets in the report; if the snippets are insufficient (especially for describing the aforementioned and enumerated steps of the implementation), we will then consult your CMSX code submission and **may deduct points** for the insufficiency of the code snippets. We anticipate **Part 2** will take at most 3 pages.

Part 2: Rules

Part 2 requires implementing a rudimentary RNN in `PyTorch` for text classification. Countless blog posts, Internet tutorials, and other implementations available publicly (and privately) do precisely this. In fact, almost every student in Cornell NLP likely has

some code for doing this on their Github. You **cannot** use any such code (though you may use anything you find in the course notes and course texts) irrespective of whether you cite it or do not.

Submissions will be passed through the MOSS system, which is a sophisticated system for detecting plagiarism in code and is robust in the sense that it tries to find alignments in the underlying semantics of the code and not just the surface level syntax. Similarly, the course staff are also quite astute with respect to programming neural models for NLP and we will strenuously look at your code. We flagged multiple groups for this last year, so we strongly suggest you resist any such temptation (if the Academic Integrity policy alone is insufficient at dissuading you).

Part 3: Analysis

From **Part 1** and **Part 2**, you will have two different models in hand for performing the same 5-way sentiment analysis task. In **Part 3**, you will conduct a comprehensive analysis of these models, focusing on two comparative settings.

Part 3: Across-Model Comparison

In this section, you will report results detailing the comparison of the two models. Specifically, we will consider the issue of *fair comparison*[5], which is a fundamental notion in NLP and ML research and practice. In particular, given model A, it is likely the case we can make a model B that is computationally more complex and, hence, more costly and achieves superior performance. However, this makes for an unfair comparison. For our purposes, we want to study how the FFNN and RNN compare when we try to control for hyperparameters and other configurable values being of similar computational cost[6]. That said, it is impossible to have identical configurations as these are different models, i.e. the RNN simply has hyperparameters for which there are no analogues in the FFNN.

In the report you will need to begin by describing 3 pairs of configurations, with each pair being comprised of a FFNN configuration and a RNN configuration that constitute a *fair comparison*. You will need to argue for why the two parts of each pair are a fair comparison. Across the pairs, you should try different types of configurations (e.g. trying to resolve like questions of the form: *Does the FFNN perform better or worse when the hidden dimensionality is small as opposed to when it is large?*) and justify what you are trying to study by having the results across the pairs.
Next, you will report the quantitative accuracy of the 6 resulting models. You will analyze these results and then move on to a more descriptive analysis.
The descriptive analysis can take one of two forms[7]:

[5]This term takes on different meanings in different settings. Here we simply mean that we are trying to compare different models while controlling for similar "complexity"/computational cost.

[6]We have not taught you how to do this rigorously and the theory for doing this is still underdeveloped. We only expect a reasonable attempt.

[7]This is the minimal requirement, if you provide other, more elaborate, analyses, we certainly welcome this.

1. NUANCED QUANTITATIVE ANALYSIS
   If you choose this option, you will need to further break down the quantitative statistics you reported initially. We provide some initial strategies to prime you for what you should think about in doing this: one possible starting point is to consider: if model X achieves greater accuracy than model Y, to what extent is X getting everything correct that Y gets correct? Alternatively, how is model performance affected if you measure performance on a specific strata/subset of the reviews?

2. NUANCED QUALITATIVE ANALYSIS
   If you choose this option, you will need to select individual examples and try to explain or reason about why one model may be getting them right whereas the other isn't. Are there any examples that all 6 models get right or wrong and, if so, can you hypothesize a reason why this occurs?

Part 3: Within-Model Comparison

To complement **Part 3: Across-Model Comparison**, in **Part 3: Within-Model Comparison**, you will need to study what happens when you change parameters within a model. To limit your workload, you need only do this for the RNN; and you may use at most one RNN model from the prior section.
In the prior section, we discussed *fair comparison*. Another aspect of rigorous experimentation in NLP (and other domains), is the *ablation study*. In this, we *ablate* or remove aspects of a more complex model, making it less complex, to evaluate whether each aspect was necessary. To be concrete, for this part, you should train 4 variants of the RNN model, and describe them as we do below:

1. Baseline model

2. Baseline model made more complex by modification A (e.g. changing the hidden dimensionality from h to $2h$).

3. Baseline model made more complex by modification B (where B is an entirely distinct/different update from A).

4. Baseline model with both modifications A and B applied

Under the framing of an ablation study, you would describe this as beginning with model 4 and then ablating (i.e. removing) each of the two modifications, in turn; and then removing both to see if they were genuinely necessary for the performance you observe.
Once you describe each of the four models, report the quantitative accuracy as in the previous section. Conclude by performing the **opposite** nuanced analysis from the one you did in the previous section (i.e. if in **Part 3: Across-Model Comparison** you did a NUANCED QUANTITATIVE ANALYSIS, for **Part 3: Within-Model Comparison** perform a NUANCED QUALITATIVE ANALYSIS and vice versa).

Part 3: Report

To reiterate, for **Part 3**, you will need to describe the 6 models (i.e. 3 pairs) of configurations, report the results, and then perform a nuanced analysis for **Part 3: Across-Model Comparison**. You will need to then to describe 4 RNN model variants in the ablation style described, report the results, and then perform a nuanced analysis of the opposite type as before. If you are efficient, you will only need to train 9 models instead of 10; you may make use of our suggestions for specific nuanced analyses and variants but we would prefer (but will not explicitly mandate) you consider other ideas of your own creation.

In addition, you will be required to submit the code used in finding these results on CMSX. This code should be legible and we will consult it if we find issues in the results. It is worth noting that in **Part 1** and **Part 2**, we primarily are considering the correctness of the code-snippets in the report. If your model is flawed in a way that isn't exposed by those snippets, this will likely surface in your results for **Part 3**. We will deduct points for correctness in this section to reflect this and we will try to localize where the error is (or think it is, if it is opaque from your code). That said, we will be lenient about absolute performance (within reason) in this section.[8] As a result, there is no Kaggle competition for this assignment nor is there any provided set of baseline scores from instructor implementations.

Part 4: Questions

In **Part 4**, you will need to answer the three questions below. We expect answers to be to-the-point; answers that are vague, meandering, or imprecise **will receive fewer points** than a precise but partially correct answer.

1. Earlier in the course, we studied models that make use of *Markov* assumptions. Recurrent neural networks do not make any such assumption. That said, RNNs are known to struggle with long-distance dependencies. What is a fundamental reason for why this is the case?

2. In applying RNNs to tasks in NLP, we have discovered that (at least for tasks in English) feeding a sentence into an RNN backwards (i.e. inputting the sequence of vectors corresponding to (*course*, *great*, *a*, *is*, *NLP*) instead of (*NLP*, *is*, *a*, *great*, *course*)) tends to improve performance. Why might this be the case?

3. In using RNNs and word embeddings for NLP tasks, we are no longer required to engineer specific features that are useful for the task; the model discovers them automatically. Stated differently, it seems that neural models tend to discover better features than human researchers can directly specify. This comes at the cost of systems having to consume tremendous amounts of data to learn these kinds of patterns from the data. Beyond concerns of dataset size (and the computational resources required to process and train using this data as well as

[8] Teaching good heuristics and policies for setting hyperparameters in neural models could constitute an entirely separate course just for NLP alone.

the further environmental harm that results from this process), why might we disfavor RNN models?

Part 5: Miscellaneous

List the libraries you used and sources you referenced and cited (labelled with the section in which you referred to them). Include a description of how your group split up the work. Include brief feedback on this asignment.
Each section must be clearly labelled, complete, and the corresponding page should be correctly assigned to the corresponding Gradescope rubric item. If you follow these steps for each of the 4 components requested, you are **guaranteed** full credit for this section. Otherwise, you will receive **no credit** for this section.