CS 4650/7650 Fall 2020: Homework 5

October 21, 2020

Instructions

1. This homework has two parts: questions 1–4 are theory questions, and Q5 is a programming assignment with some written components within a Jupyter Notebook.

   We will be using Gradescope to collect your assignments. Please read the following instructions for submitting to Gradescope carefully!

   (a) Each subproblem must be submitted on a separate page. When submitting to Gradescope (under HW5 Writing), make sure to mark which page(s) correspond to each problem or subproblem. For instance, Q2 has two subproblems, so the solution to each must start on a new page.

   (b) For the coding problem (Q5), please upload 'MyQA.py' and 'HW5.ipynb' under the HW5 programming assignment on Gradescope. Write your solutions for Q5 (a) and (b) in your writeup, and Attach a pdf export of 'HW5.ipynb', including outputs, to your writeup.

   (c) Note: This is a large class and Gradescope's assignment segmentation features are essential. Failure to follow these instructions may result in parts of your assignment not being graded. We will not entertain regrading requests for failure to follow instructions.

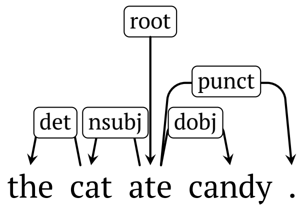
2. LaTeX solutions are strongly encouraged (a solution template is available on the class website), but scanned handwritten copies are also acceptable. Hard copies are not accepted.

3. We generally encourage collaboration with other students. You may discuss the questions and potential directions for solving them with another student. However, you need to write your own solutions and code separately, and not as a group activity. Please list the students you collaborated with on the submission site.

Questions

1. By now you have extensive knowledge of constituency parsing, which represents natural language sentences using a hierarchical tree structure which groups words together into phrases. Another common representation of syntax in NLP is **dependency parsing**, which overlays the sentence with a set of edges over the word nodes. In a dependency graph (also called a *dependency tree*), there are no phrases, or nodes which are not words, and syntactic hierarchy is determined by relating a *head* word with an edge towards its *dependent* word, annotated with a syntactic *relation*. An example is presented below. The submission template includes tikz-dependency code for generating it, so you can draw your own trees in case you're submitting the assigment in LaTeX.

   Please read up on dependency syntax in Eisenstein's textbook, chapter 11 until the end of section 11.1.

   (a) The following two sentences are syntactically ambiguous. Draw both possible trees for each sentence. Refer to the documentation of relations in the following url: https://universaldependencies.org/u/dep/. You're only going to need the following: **root, nsubj, compound, aux, amod, det, dobj, punct.** [8 points]

      i. I saw the funny giraffe movie.

      ii. We can fish.

(b) A recent version of the *Universal Dependencies* treebank project has changed the convention for preposition phrases: instead of the preposition being a `prep` dependent of the parent and the `pobj` head of the subordinate, it is now the `prep` dependent of the subordinate, which is a `pobj` dependent of the parent. Eisenstein gives an example (*scratch with claws*) in §11.1.1.

Is this change expected to change the average **depth** of trees (calculated as the length of the longest path from root to leaf) in English treebanks? If so, in which direction? [2 points]

(c) A *projective* parse tree is one in which no edges cross when drawn on a plane above the text. Think of a sentence in English whose parse tree is not projective. Draw the tree. [**Bonus** 5 points]

2. **Open-domain Question Answering** aims to find the answers to questions expressed in natural language from a large collection of documents.

   Traditional methods for Open-domain Question Answering are composed of two stages. In the first stage, a Document Retriever retrieves articles that are likely to be relevant to the question as candidate documents. In the second stage, a Document Reader predicts a text span in the candidate documents as the final answer.

   (a) In the Document Reader, we consider candidate documents as a large paragraph composed of N words $\{w_1, w_2, \cdots, w_N\}$. We use a paragraph encoder to encode all words in the paragraph into vectors: $\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_N\}$. We also use a question encoder to encode the question into a vector $\mathbf{q}$. Finally, we compute the probabilities of each token in the paragraph being start and end of the correct span: $P_{start}(i) = \mathbf{p}_i \mathbf{W}_s \mathbf{q}$, $P_{end}(i) = \mathbf{p}_i \mathbf{W}_e \mathbf{q}$.

      i. In the paragraph encoder, the basic input are word embeddings. Besides word embeddings, we want to design a binary feature as additional input for each word. What would you choose as a binary feature to improve the performance most. [2 pt]

      ii. Suppose parts of $\mathbf{p}_i$ is constructed from aligned question embedding. Assume we have a column vector to represent the word "America" in the paragraph: $\mathbf{e} \in \mathbb{R}^d$. We have a matrix composed of three vectors to represent the question "Where is Seattle": $\mathbf{Q} \in \mathbb{R}^{3 \times d}$. $f(\cdot)$ denotes a softmax layer. Use dot-product attention mechanism to calculate the aligned question embedding for the word "America". Your answer should be in terms of $\{\mathbf{Q}, \mathbf{e}, f(\cdot)\}$ [3 points].

   (b) In the Document Retriever, articles and questions are compared as TF-IDF weighted bag-of-word vectors in order to return the k most relevant articles. But such term-based sparse representations have limited capabilities in matching questions and passages, e.g., there could be many relevant articles where there is no exact term match.

      Instead, we want a model to learn to retrieve documents. Suppose we have a large collection of documents $\{z_1, z_2, \cdots, z_M\}$. We use a model to compute $p(z_i|x)$, where x is the question. Document Reader can compute the probability of answer a: $p(a|x, z_i)$. To train Document Retriever and Document Reader jointly, we maximize the probability $p(a|x)$.

      i. Write down the probability $p(a|x)$ in terms of $p(z_i|x)$ and $p(a|x, z_i)$. [2 points]

      ii. Computing $p(a|x)$ exactly in such formula is very time-consuming. Explain why [1 point], and propose a solution to approximate $p(a|x)$ [2 points].

3. (a) The **Bi**Lingual **E**valuation **U**nderstudy (**BLEU**) is a method for automatic evaluation of machine translation that is quick, inexpensive and language-independent, and correlates highly with human evaluation.
The formal definition of BLEU is

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right),$$

where

$$\text{BP} = \begin{cases} 1 & \text{if } c > r, \\ \exp(1 - r/c) & \text{otherwise.} \end{cases}$$

p_n represents the n-gram precision, associated with its positive weight w_n.
c is the length of the candidate translation and r is the length of the reference translation.
Suppose there are a reference translation and a candidate translation:

i. reference: "Tom likes to study natural language processing at night";
ii. candidate: "Tom loves to study language processing at night".

Assume $N = 4$ and $w_1 = w_2 = w_3 = w_4 = \frac{1}{4}$.
Compute p_1, p_2, p_3, p_4, and then compute BP and BLEU. [6 points]

(b) Suppose you are given a set of sentence pairs

$$\begin{aligned}(\mathbf{e}, \mathbf{f}) = \Big\{ &([\text{'the'}, \text{'house'}], [\text{'das'}, \text{'haus'}]), \\ &([\text{'the'}, \text{'book'}], [\text{'das'}, \text{'buch'}]), \\ &([\text{'a'}, \text{'book'}], [\text{'ein'}, \text{'buch'}]) \Big\}.\end{aligned}$$

Use IBM Model 1 to compute the translation probabilities. The translation probabilities $t(e|f)$ are initialized as the column "initial" in Table 1.

e	f	initial	1st it.	2nd it.
the	das	0.3		
book	das	0.3		
house	das	0.3		
the	buch	0.3		
book	buch	0.3		
a	buch	0.3		
book	ein	0.4	0.5	0.4222
a	ein	0.4	0.5	0.5778
the	haus	0.4	0.5	0.4222
house	haus	0.4	0.5	0.5778

Table 1: Translation probabilities $t(e|f)$.

Fill in all the blanks in Table 1. Please show the intermediate steps of your calculation as well. [9 points]

h_j / p_i	**a**	**soccer**	**game**	**with**	**multiple**	**men**	**playing**
some	0.1	0.2	0.2	0.1	0.8	0.4	0.1
men	0.2	0.4	0.3	0.2	0.1	1	0.2
are	0.2	0.2	0.4	0.2	0.3	0.8	0.7
playing	0.1	0.5	0.6	0.2	0.4	0.3	1
a	1	0.2	0.2	0.1	0.2	0.3	0.2
sport	0.1	0.9	0.7	0.1	0.2	0.3	0.5
few	0.5	0.1	0.2	0.3	0.3	0.2	0.1
people	0.2	0.3	0.2	0.1	0.5	0.7	0.4
basketball	0.1	0.5	0.6	0.2	0.2	0.4	0.7

Table 2: Lexical scores for ordered word pairs- $P(h_j|p_i)$.

4. **Natural language inference** (NLI) is the problem of determining whether a natural language hypothesis h can reasonably be inferred from a natural language premise p. This task is also referred to as Recognizing Textual Entailment (RTE) and we say that p entails h if, typically, a human reading p would infer that h is most likely true. The task is usually defined with labels {entails, contradicts, neutral}. Let us look at a few examples:

   p: The flight crash landed into the Atlantic Ocean.
   h_1: The flight's journey was not successful. (ENTAILS)
   h_2: There has been an increase in crash landings lately. (NEUTRAL)
   h_3: A flight crash landed into the Indian Ocean. (CONTRADICTS)

   (a) A very simple approach to the NLI task is with a bag-of-words entailment model. Let $P(h|p)$ denote the probability that a premise p supports an inference to a hypothesis h. This probability is decomposed to independently account for the probability that each individual word $h_j \in h$ is entailed by p. Assuming that each word $h_j \in h$ derives its support mainly from a single word in p, $P(h_j|p)$ can be identified with the max over the probability of its support by the individual words $p_i \in p$.

   $$P(h|p) = \prod_j P(h_j|p) = \prod_j \max_i P(h_j|p_i)$$

   Use table 2 to find values of $P(h_j|p_i)$ and find the probabilities that hypotheses h_1 and h_2 are entailed by premise p. [4 points]

   p: a soccer game with multiple men playing
   h_1: some men are playing a sport
   h_2: a few people are playing basketball

   (b) The most crucial part of the simple model above is the lexical scoring function $P(h_j|p_i)$ which maps ordered pairs of words (h_j, p_i) to real values in the interval

[0,1]. Mention one choice of scoring function that can be used for this task. [2 points]

(c) A robust NLI system can be used for several applications. One example of this is for evaluating Machine Translation, where a candidate translation x can be compared with a ground truth y translation to check for semantic equivalence (x entails y and y entails x). Mention and describe another application where NLI can be used. [2 points]

(d) Describe in brief how a neural network technique can be used for the NLI task. [2 points]

5. In this assignment, you will implement a slight variant of the Bi-Directional Attention Flow For Machine Comprehension or BiDAF method. You should refer to the original paper as your implement the methods. We will learn the BiDAF on The Stanford Question Answering Dataset or SQuAD 2.0. The dataset is composed question-answer pairs that pertain to a particular passage from a Wikipedia article. Every passage has several questions, and each question has several answers. Furthermore, each answer is also annotated with its start position in the passage. We will be using answer's start and end position as labels to provide supervision. Feel free to explore and familiarize yourself with the data and check out *Background, Challenges, Progress* to learn more.

   You will write your code in `HW5.ipynb` and `MyQA.py`. "**`MyQA.py`**" should be submitted to **HW5 programming**; make sure the file doesn't use or import any packages not included in the `requirements.txt` file. You can download the HW5 package from the course website. Furthermore, **`HW5.ipynb`** should be converted to PDF and attached at the end of your written report and submitted to **HW5 Writing**.

   We have already downloaded and preprocessed all SQuAD data for you and they are in the `.pickle` files in the HW5 folder you downloaded from course website. The purpose of this assignment is not to build state-of-the-art question answering system. You can probably do that by fine-tuning a pretrained model (which you can do for Bonus in part d). For parts a and b, your main goal is to build and partially train BiDAF in PyTorch directly from the original paper. Don't worry, we have provided plenty of structured code to help guide you through.

   (a) For this part **implement** the Encoder for our BiDAF model. This will look very similar to the POSTagger code you wrote in HW4. Instead of encoding characters using a CNN, like they do in the Seo et. al (2017), we will just use another LSTM for simplicity. We will also ignore the "Highway Network" used in Seo et. al (2017) for simplicity. However, these are the only two deviations from the original paper. The rest of the architecture will stay the same. Implement the `LSTMEncoder` in `MyQA.py` and test whether the output dimensions are correct by running the notebook cell in *Part 1: Encoder*. Your implementation should work correctly with various hyperparameters, this will be tested by the autograder. **What** are some pros and cons of using LSTM to incorporate character information as opposed to CNN as done by original authors? **What** is the purpose of the Highway Network that was used by the authors? [6 points code + 4 points written]

   (b) Next, **implement** the Attention Flow Layer of the network. More guidance is provided in the python file. You may also need to refer to the original paper to better understand the approach. Check your work with the appropriate notebook cell. **Reflect** on the pros and cons of this approach. [10 points code + 5 points written]

   (c) Finally, **implement** the rest of the modules: `ModelingLayer`, `OutputLayer` and `BiDAF`. Once you have verified that the implementation using checks provided in the notebook, **implement** items in the notebook under "2. Implement Training and Eval" section. You do not need to train the network for optimal accuracy, however, you should run the train and eval for about 2 epochs to verify that your implementation works. Please also modify the hyperparameters so that your loss is decreasing. Make sure that that outputs are preserved when you attach PDF of your notebook in your write-up. Finally, feel free to modify other functions in the notebook (such as `get_eval_scores`) or add new ones for your convenience. **Please explain** what hyperparameters you modified and how it influences the loss and F1 scores. [15 points]

(d) Build any network and submit to SQuAD 2.0 to earn extra credit. The full SQuAD 2.0 data and instructions on submission can be found here. You can obtain a pretrained network like BERT and fine-tune on SQuAD as discussed in lecture or train one from scratch. You could even try out some feature based methods or knowledge based methods discussed in class. Whichever approach you use, you must submit all the code files in separate folder to gradescope, explain your approach in the write-up, attach a screenshot of your submission to SQuAD and your performance on their test set. You do not need to upload any model or data files. Your performance must be above 75% to get the full 15 points. However, you can still get partial bonus if you submit to SQuAD and submit everything you worked on. [**BONUS** 15 points]