CS 4650/7650 Fall 2020: Homework 2

September 2, 2020

Instructions

1. This homework has two parts: questions 1–3 are theory questions, and Q4 is a programming assignment with some parts requiring a written answer.

   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 HW2 Writing), make sure to mark which page(s) correspond to each problem or subproblem. For instance, Q1 has 5 subproblems, so the solution to each must start on a new page.

   (b) For the coding problem (Q4), please upload 'hw2_skeleton_char.py', 'hw2_skeleton_word.py', 'ngram.ipynb' and 'rnn.ipynb' under the HW2 Code assignment on Gradescope. Write your solutions for Q4 (b), (c), (d), (e) in your writeup, and attach pdf exports of 'ngram.ipynb' and 'rnn.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. Language Modeling is a technique that allows us to compute the probabilities of word sequences. The probability of a sequence $\mathbf{W} = w_1^N = \{w_1, w_2 ... w_N\}$, with the use of the chain rule, can be estimated as the product of probabilities of each word given the history, as shown (leaving out the exact notation for the $i = 1$ edge case):

$$\begin{aligned} P(\mathbf{W}) &= P(w_1, w_2 ... w_N) \\ &= P(w_1)\ P(w_2|w_1)\ P(w_3|w_1, w_2) ... P(w_N|w_1, w_2 ... w_{N-1}) \\ &= \prod_{i=1}^{N} P(w_i|w_1^{i-1}) \end{aligned}$$

   (a) Using an n-gram model allows us to approximate the above probability using only a subset of of $n-1$ words from the history at each step. Simplify the above expression for the general n-gram case, and the bigram case ($n = 2$). [3 pts]

   (b) A common way to have markers for the start and the end of sentence is to add the [BOS] (beginning of sentence) and [EOS] (end of sentence) tokens at the start and end of each sentence. Consider the following text snippet:

   [BOS] i made cheese at home [EOS]
   [BOS] i like home made cheese [EOS]
   [BOS] cheese made at home is tasty [EOS]
   [BOS] i like cheese that is salty [EOS]

   Using the expression derived in (a), find the probability of the following sequence as per the bigram model: $P($[BOS] I like cheese made at home [EOS]$)$. [5 pts]

   (c) In practice, instead of raw probability, *perplexity* is used as the metric for evaluating a language model. Define perplexity and find the value of perplexity for the sequence in (b) for the bigram case. [2 pts]

   (d) One way to deal with unseen word arrangements in the test set is to use Laplace smoothing, which adds 1 to all bigram counts, before we normalize them into probabilities. An alternative to Laplace smoothing (add-1 smoothing) is add-k smoothing, where k is a fraction that allows assigning a lesser probability mass to unseen word arrangements. Find the probability of the sequence in (b) with add-k smoothing for $k = 0.1$. [5 pts]

   (e) To deal with unseen words in the test set, a common way is to fix a vocabulary by thresholding on the frequency of words, and assigning an [UNK] token to represent all out-of-vocabulary words. In the example from (b), use a threshold of count$(w) > 1$ to fix the vocabulary. Find the probability for the following sequence for an add-0.1 smoothed bigram model:

   $P($[BOS] i like pepperjack cheese [EOS]$)$. [5 pts]

2. Kneser-Ney smoothing is an n-gram smoothing method which addresses cases where we may not have seen an n-gram very often. In this case, we can *back off* to probabilities of n-1 grams. Kneser-Ney allows us to do that while taking into account the diversity of n-1-grams, or their tendency to appear in unique contexts. For instance, if we want to finish the sentence "I want to go to the movie _____" using a bigram model but have not observed "movie theatre" as a bigram before, we may use unigram probability of "theatre" in our corpus. However, another word such as "Francisco" may have a higher unigram probability than "theatre". Since "Francisco" only appears next to "San", we can say it doesn't appear in many diverse contexts. We can use Kneser-Ney smoothing to take the diversity of contexts of each unigram into account.

   The full formula for the bigram version of Kneser-Ney smoothing follows:

$$P(w_i|w_{i-1}) = \underbrace{\frac{\max(C(w_{i-1}, w_i) - d, 0)}{C(w_{i-1})}}_{\text{discounted bigram probability}} + \underbrace{\lambda(w_{i-1}) \cdot P_{CONTINUATION}(w_i)}_{\text{joint unigram probability}} \tag{1}$$

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}, v)} \cdot |w : C(w_{i-1}, w) > 0| \tag{2}$$

$$P_{CONTINUATION}(w_i) = \frac{|v \in V : C(v, w_i) > 0|}{\sum_{w'} |v \in V : C(v, w') > 0|}, \tag{3}$$

   where V is our word vocabulary.

   When combining bigram probability with unigram probability, we need to discount the counts of bigrams to save some probability mass to distribute for unigrams. The discount constant d in equation 2 discounts some probability mass from the high bigram counts, and the normalizing constant λ in equation 3 distributes it across unigram counts. Additionally, $P_{CONTINUATION}$ measures how likely a word w appears as a novel continuation.

   Assume that you have collected the data in the following tables, and assume that all other observed counts are 0. In the bigram table, rows represent w_{i-1}, columns represent w_i: e.g. $C(\text{computer}, \text{keyboard}) = 6$.

$C(w_{i-1}, w_i)$	computer	keyboard	monitor	store
computer	0	6	8	8
keyboard	2	0	0	3
monitor	1	1	0	4
store	1	0	0	0

Table 1: Bigram frequency. Rows = w_{i-1}, columns = w_i.

   Consider the following sentence fragment *S*: "I shopped at the computer _____". You need to determine whether the sentence is more likely to end with "store" or "monitor."

computer	22
keyboard	10
monitor	15
store	15

Table 2: Unigram frequency.

(a) Compute the raw bigram probabilities for the candidate words {*store*, *monitor*} to complete the sentence *S*, i.e. $P(\text{store}|\text{computer})$ and $P(\text{monitor}|\text{computer})$. Is one word more likely than the other, and if so which one? [2 pts]

(b) Compute the Kneser-Ney smoothed bigram probability of the candidate words {*store*, *monitor*} to complete the sentence. Use $d = 0.5$ as the discount term. Is one word more likely than the other, and if so which one? If the result has changed, why do you think it changed? [5 pts]

(c) Change the discount term to $d = 0.1$ and re-compute the Kneser-Ney smoothed bigram probability of the candidate words {*store*, *monitor*} to complete the sentence. Is one word more likely than the other, and if so which one? If the result has changed, why do you think it changed? [3 pts]

3. **Neural Networks** We learned about neural networks in class. The combination of different model architectures, non-linearities through different activation functions, and various ways to avoid overfitting make neural networks a flexible and robust technique to learn from data.

   (a) **Activation functions** Assume a 2-layer feedforward network $\mathbf{y} = f(\mathbf{x})$ such that:

   $$\begin{aligned}\mathbf{a} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\ \mathbf{z} &= \sigma(\mathbf{a}) \\ \mathbf{y} &= \mathbf{W}^{(2)}\mathbf{z} + \mathbf{b}^{(2)}\end{aligned}$$

   where $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ are the weights and biases for the i^{th} linear layer and $\sigma(\cdot)$ is the sigmoid activation function at the hidden layer. Derive an equivalent network f' — by adjusting $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ — such that $f'(\mathbf{x}) = f(\mathbf{x})$ (i.e both networks produce the same output for a given input) but f' uses the $\tanh(\cdot)$ activation function at the hidden layer. [4 pts]

   (b) **Dropout** Among many regularization techniques to avoid overfitting in neural networks, dropout is the simplest. During training, dropout randomly sets units in the hidden layer $\mathbf{h} \in \mathbb{R}^{D_h}$ to zero with probability p_{drop} (dropping different units each minibatch), and then multiplies $\mathbf{h}$ by a constant γ. We can write this as:

   $$\mathbf{h}_{drop} = \gamma d \odot \mathbf{h}$$

   where $d \in \{0, 1\}^{D_h}$ is a mask vector where each entry is 0 with probability p_{drop} and 1 with probability $(1 - p_{drop})$, and $\odot$ is the element-wise product operation (also known as a *Hadamard product*). γ is chosen such that the expected value of $\mathbf{h}_{drop}$ is $\mathbf{h}$ i.e :

   $$\mathbb{E}_{p_{drop}}[h_{drop}]_i = h_i \quad \forall i \in \{1, \ldots, D_h\}.$$

   What should be the value of γ in terms of p_{drop}? Show the calculation to justify your answer. [4 pts]

   (c) Of the types of neural nets covered in class, which would you think is most applicable to the task of language modeling? Explain using explicit elements from the LM formulation and from the network you choose. [2 pts]

4. In the textbook, language modeling was defined as the task of predicting the next word in a sequence given the previous words. In this assignment, you will implement both character-level and word-level n-gram language models. You will also implement a character-level RNN language model. You need to submit your code 'hw2_skeleton_char.py', 'hw2_skeleton_word.py', 'ngram.ipynb' and 'rnn.ipynb' to HW2 Code in Gradescope. Also, you should answer the questions for (b) (c) (d) (e) in your writeup, and export 'ngram.ipynb' and 'rnn.ipynb' with output to a pdf and attach to your writeup. Your writeup should be uploaded to HW2 Writing.

   (a) Complete the scripts 'hw2_skeleton_char.py' and 'hw2_skeleton_word.py'. Detailed instructions can be found in 'ngram.ipynb'. You should also use test cases in 'ngram.ipynb' to get development results for (c) and (d). Submit 'hw2_skeleton_char.py' and 'hw2_skeleton_word.py' to HW2 Code on Gradescope, where you will see the scores for your code. Character-level N-gram language models accounts for 20 points. Word-level n-gram language models accounts for 10 points, which are bonus for CS 4650. [CS 7650: 30pts, CS 4650: 20 pts + bonus 10pts]

   (b) Observe the generation results of your character-level and word-level n-gram language models ($n \geq 1$). The paragraphs which character-level n-gram language models generate all start with *F*. The paragraphs which word-level n-gram language models generate all start with *In*. Did you get such results? Explain what is going on. (CS 4650 only need to explain this phenomenon in character-level N-gram language model.) [2 pts]

   (c) **[Bonus for CS 4650, Required for CS 7650]** Compare the generation results of character-level and word-level n-gram language models. Which do you think is better?

   Compare the perplexity of 'shakespeare_sonnets.txt' when using character-level and word-level n-gram language models. Explain what you found. [2 pts]

   (d) When you compute perplexity, you can play with different sets of hyper-parameters in both character-level and word-level n-gram language models. You can tune n, k and λ. Please report here the best results and the corresponding hyper-parameters in development sets. For character-level n-gram language models **[CS 4650 + CS 7650]**, the development set is 'shakespeare_sonnets.txt' [2 pts]. For word-level n-gram language models **[Bonus for CS 4650, Required for CS 7650]**, the development sets are 'shakespeare_sonnets.txt' [2 pts] and 'val_e.txt' [2 pts].

   (e) For RNN language models, you should complete the forward method of the RNN class in 'rnn.ipynb'. You need to figure out the code and tune the hyperparameters. You should also

      i. Copy a paragraph generated by your model here, [4 pts]

      ii. Report hyperparameters and perplexity on the development set 'shakespeare_sonnets.txt' here, [4 pts] and

      iii. Compare the results of character-level RNN language model and character-level n-gram language model here. [2 pts]

Do not forget to export ‘rnn.ipynb’, including output, to pdf and attach to your writeup.