CIS 419/519: Applied Machine Learning — Fall 2023

Homework 6

Handed Out: November 8 — *Due: November 22, 8pm*

- You are encouraged to format your solutions using LaTeX. You'll find some pointers to resources for learning LaTeXamong the Canvas primers. Handwritten solutions are permitted, but remember that you bear the risk that we may not be able to read your work and grade it properly — do not count on providing post hoc explanations for illegible work. You will submit your solution manuscript for written HW4 as a single PDF file.

- The homework is **due at 7:59 PM** on the due date. We will be using Gradescope for collecting the homework assignments. Please submit your solution manuscript as a PDF file via Gradescope. Post on Piazza and contact the TAs if you are having technical difficulties in submitting the assignment.

1 Written Questions

Note: You do not need to show work for multiple choice questions. If formatting your answer in LaTeX, use our LaTeX template `hw6_template.tex` (This is a read-only link. You'll need to make a copy before you can edit. Make sure you make only private copies.).

1. [Text Generation/Language Modeling] (10 pts) Text generation is a popular application and area of research in NLP. In this problem, we will look at a specific yet common scenario of text generation, where you want to generate a sentence by sampling words from an autoregressive language model (such as GPT[1]). Given the first k prompt words $\{w_1, w_2, ..., w_k\}$ from left-to-right order in a sentence, an autoregressive language model outputs the probability distribution of the next word conditioned on the prompt words: $P(w_{k+1}|w_1, w_2, ..., w_k)$. A complete sentence can be generated by iteratively sampling words from the next word probability distributions until an end-of-sentence indicator (such as period ".") is reached. But how should we sample the words from $P(w_{k+1}|w_1, w_2, ..., w_k)$?

   In this question, we will compare two different sampling strategies and learn the intuition behind them with a toy example. Suppose you are interested in generating a sentence that starts with the word "`Bob`". You are given an autoregressive language model with only 5 words in vocabulary - {`Bob`, `loves`, `hates`, `cherry`, `cookie`}. You tried the following three prompts, and here are the three conditional probability distributions of the next word you get. For all subquestions, assume that you only want to generate the next two words after "`Bob`".

[1] Free web demo of a GPT-3 like model - https://6b.eleuther.ai/

Next Word	Probability
loves	0.50
hates	0.40
cookie	0.06
cherry	0.03
Bob	0.01

Table 1: $P(w_1|\texttt{Bob})$

Next Word	Probability
cookie	0.40
Bob	0.25
cherry	0.20
hates	0.12
loves	0.03

Table 2: $P(w_2|\texttt{Bob}, \texttt{loves})$

Next Word	Probability
cherry	0.70
cookie	0.20
Bob	0.08
loves	0.01
hates	0.01

Table 3: $P(w_2|\texttt{Bob}, \texttt{hates})$

(a) (1 pts) Suppose we use the greedy sampling strategy, that is, always sample the word with highest conditional probability as the next word. What will be the sentence you generated (i.e. "Bob" plus the next two words)?

(b) (3 pts) Naturally, your goal with text generation is to generate the most probable sentence out of your vocabulary. In other words, you want to sample the sentence which maximizes the joint probability of $P(w_1, w_2|w_0 = \texttt{Bob})$. While deriving the exact probability distribution with RNN models is in most cases NP-hard [2], people commonly use the natural log-sum of the next-word probability as an approximation to the log-likelihood of the sentence; In other words:

$$ln(P(w_1|w_0 = \texttt{Bob})) + ln(P(w_2|w_0 = \texttt{Bob}, w_1)) \quad (1)$$

Use the above formula to estimate the log-likelihood of the following two sentences "Bob loves cookie" and "Bob hates cookie".

(c) (2 pts) From the last question, do you think the greedy sampling strategy will *always* give you the most probable sentence? Why or why not?
(*Hint: Let's take the reasonable assumption that sentences with higher estimated log-likelihood from Eq. 1 are more probable.*)

(d) (4 pts) Let's consider an alternative sampling strategy called beam search. Instead of always taking the highest probability word, let's say we take the top-2 [3] words instead. In our case for w_1, this would give us two beam hypotheses "Bob loves" and "Bob hates".

For the w_2, we sample the top two words for the two beam hypotheses respectively, which gives us the following four hypotheses.

[2] https://aclanthology.org/N18-1205.pdf

[3] k=2 for Top-k here is a tunable parameter, and the correct jargon for this is: beam search with beam size of two

- `Bob loves cookie`
- `Bob loves Bob`
- `Bob hates cherry`
- `Bob hates cookie`

The next step would be estimating the log-likelihood of the four hypotheses, and we will be keeping the top-2 highest probability hypotheses and iteratively generate the next words. Which two hypotheses among the above four should we keep in this case? In other words, which two have the top-2 highest estimated log-likelihood among the above four? Show your computation (for the ones that you haven't computed before).

2. [Attention Mechanism] (10 pts) In this problem, we will walk through how dot-product attention weights we introduced in class are calculated. Suppose we have a Sequence-to-Sequence machine translation (MT) model from English to Dothraki, where the hidden states for the encoder and decoder RNNs have size of 4. We input the English sentence "`Dragons eat apple too`" into the MT model, and below are the values of the hidden states we get from the model in the encoder.

Name	Input Word	Hidden State
h_1	`Dragons`	$[0.7, 0.2, 0.3, 0.1]$
h_2	`eat`	$[0.2, 0.7, 0.3, 0.1]$
h_3	`apple`	$[0.0, 0.6, 0.4, 0.3]$
h_4	`too`	$[0.1, 0.1, 0.0, 0.9]$

Table 4: Encoder hidden state values $h_1, ..., h_4$

Suppose the first word that the MT model generates in the decoder is "`Zhavvorsa`", and the hidden state value for the word is $s_1 = [0.5, 0.2, 0.4, 0.1]$. You are welcome (and encouraged!) to use electronic devices to help with calculations in this question.

(a) (3 pts) Calculate the dot-product attention scores $\mathbf{E}^1$ [4] for the word "`Zhavvorsa`". Recall that the definition of dot-product attention score is

$$\mathbf{E}^t = [s_t^T h_1, ..., s_t^T h_N] \in R^N \tag{2}$$

(b) (4 pts) Use the attention scores derived in (a), derive the attention distribution α^1 for "`Zhavvorsa`". Recall that

$$\alpha^t = softmax(\mathbf{E}^t) = [\frac{e^{s_t^T h_1}}{\sum_{k=1}^N e^{s_t^T h_k}}, ..., \frac{e^{s_t^T h_N}}{\sum_{k=1}^N e^{s_t^T h_k}}] \tag{3}$$

(c) (3 pts) The attention distribution will be used as weights in a weighted summation when computing the attention output. With α^1 you derived in the last sub-question, take the weighted sum of the encoder hidden state to compute the attention output a^1.

[4] Denoted lowercased $\mathbf{e}^1$ in lecture slides. Using uppercase here to avoid confusion with the Euler number e in 3

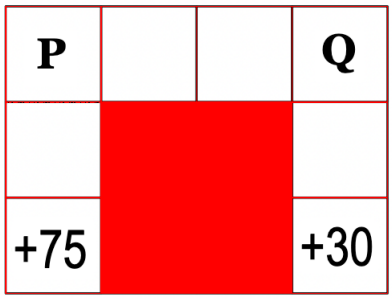

Figure 1: Gridworld for Value Iteration

State s	Condition	i	$V_i(s)$
P	$V_i(P) > 0$		
Q	$V_i(Q) > 0$		
P	$V_i(P) = V^*(P)$		
Q	$V_i(Q) = V^*(Q)$		

Table 5: Value Iteration: Part a

3. [Value Iteration] (12pts) Remember the gridworld environment which we used as a running example throughout the lecture on MDPs and RL. In this question, we will work on a similar gridworld environment, shown in Fig 1. The agent operates in this grid with solid and open cells. The agent remains where it is if it tries to move into a solid cell or outside the world. There are two bigger magnitude rewards at terminal states that end an episode and these terminal rewards are the only rewards that the agent can acquire in this environment. When the agent is at the terminal state, it is forced to execute one further action 'end' (it remains at the same place), thereby collecting the specified reward on the grid. The agent can move North, East, South and West. We decide to perform value iteration (V-value iteration) on this environment to find the optimal value function V^*. Following the value iteration algorithm, we start with the initial value function $V_0(s) = 0$. $V_i(s)$ represents the value function at the end of the i^{th} iteration and $V^*(s)$ represent the optimal value of a state. Consider that the discount γ is 1. Fill the following tables by finding the smallest iteration i for which the given condition holds, along with the value of the state at that smallest iteration i.e., $V_i(s)$.

   (a) (8pts) For this sub-part let us assume that our actions are deterministic, i.e., the chosen action succeeds 100% of the time (for an open cell). Fill table 5. Show/Justify how you arrived at the values.

   (b) (4pts) For this sub-part let us assume that our actions are stochastic, i.e., the chosen action succeeds 80% of the time (for an open cell) and for the remaining 20% of the time the agent remains as is. Fill table 6. Show/Justify how you arrived at the values.

State s	Condition	i	$V_i(s)$
P	$V_i(P) = V^*(P)$		

Table 6: Value Iteration: Part b

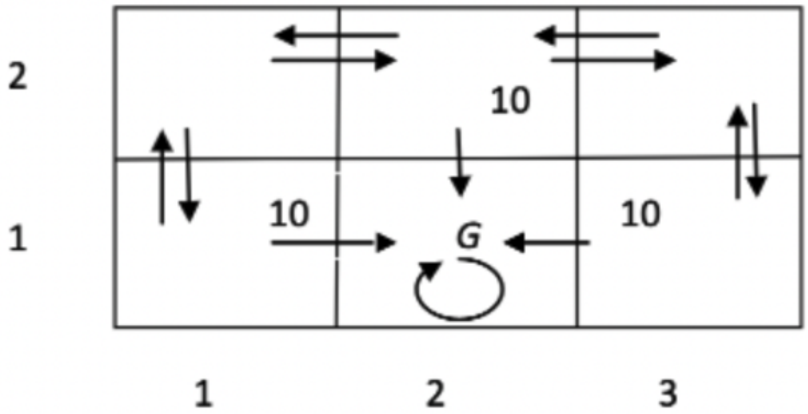

Figure 2: Gridworld

4. [Reinforcement Learning] (7 pts) Consider a deterministic grid world shown in the figure 2 with an "absorbing" state G: any action performed at this state leads back to the same state. The immediate rewards are 10 for the labeled transitions and 0 for the unlabelled transitions. The discount factor $\gamma = 0.8$.

   (a) (1.5 pts) Show the optimal policy by drawing arrows corresponding to optimal actions for each cell in the grid. Note that the optimal policy need not be unique.

   (b) (1.5 pts) Compute the optimal V-value function V^* for the top left state (column 1, row 2) in this grid world.

   (c) (4 pts) Now, consider applying the Q-learning algorithm to this grid world. Assuming the table of Q-values is initialized to zero. Assume the agent begins in the bottom left grid square and then travels clockwise along the perimeter of the grid until it reaches the absorbing goal state, completing the first training episode. Describe which Q-values are modified as a result of this episode, and give their revised values.