CIS 4190/5190: Applied Machine Learning Fall 2025

Homework 4

Handed Out: Nov 12th *Due: 7:59pm Nov 26th*

- You are encouraged to format your solutions using LaTeX. 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 — we will not accept post hoc explanations for illegible work. You will submit your solution manuscript for written HW 3 as a single PDF file.
- Familiarize yourself with the "Human and AI Assistance in Homework" Policy included in the administrivia slides from the first lecture. https://www.seas.upenn.edu/~cis5190/fall2025/schedule.html
- The homework is **due at 8 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 Ed Discussion and contact the TAs if you are having technical difficulties in submitting the assignment.
- Make sure to assign pages to each question when submitting homework to Gradescope. The TA may deduct 0.2 points per sub-question if a page is not assigned to a question.
- Items marked [**5190 Only**] are mandatory for students enrolled in CIS 5190, and optional for CIS 4190. More information on the administrivia slides.

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 hw_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] (6 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

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

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".

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 $P(w_1, w_2|w_0 = \texttt{Bob})$. When deriving the exact probability distribution with RNN models, 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:

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

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.*)

2. [Attention Mechanism] (11 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) (4 pts) Calculate the dot-product attention scores $\mathbf{E}^1$ [2] 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.

3. [Policy Gradient] (10 pts) Consider a deterministic graph navigation MDP with the following structure:

$$\underline{a} \leftarrow \underline{b} \leftrightarrow \underline{c} \leftrightarrow \underline{d} \leftrightarrow \underline{e} \leftrightarrow \underline{f} \leftrightarrow \underline{g} \rightarrow \underline{h}$$

where each node is a state, and the arrows represent the possible actions an agent can take to traverse the graph (the transition function obeys these actions). States $\underline{a}, \underline{h}$ are terminal states, so the episode ends upon reaching them.

The agent gets 100 reward for reaching $\underline{a}$, 30 reward for reaching $\underline{h}$, and 0 everywhere else.

$$R(s_t, a_t, s_{t+1}) = \begin{cases} 100 & \text{if } s_{t+1} = \underline{a} \\ 30 & \text{if } s_{t+1} = \underline{h} \\ 0 & \text{otherwise} \end{cases}$$

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

(a) (1 pts) The agent starts at state f, and the discount factor is 1.0. What is the optimal action at state $\underline{f}$?

(b) (1 pts) The agent starts at state f, and the discount factor is 0.5. What is the optimal action at state $\underline{f}$?

(c) (2 pts) The agent starts at state f. What discount factor would result in both actions being equally likely? Show your work and give the answer rounded to 2 decimal places (e.g. 0.98).

(d) (6 pts) We are learning a stochastic policy π_θ using the policy gradient algorithm. Let $S = \{\underline{a}, \ldots, \underline{h}\}$ denote the state space and $A = \{L = \text{“go left”}, R = \text{“go right”}\}$ denote the action space. The policy is parameterized by $\theta = [\theta_L, \theta_R]^\top \in \mathbb{R}^2$ and defined by $\pi_\theta(a|s) = \text{softmax}(\theta)_a$ for each $a \in A\}$ and $s \in S$. We have the following hyperparameters:

- $M = 1$ sample trajectories per policy gradient update
- discount rate $\gamma = 0.5$
- learning rate $\alpha = 0.01$
- initial parameter values $\theta^{(0)} = [0, 0]^\top$.

You are also given that:

$$\nabla_\theta \log \pi_\theta(L|\cdot) = \begin{bmatrix} \text{softmax}(\theta)_R \\ -\text{softmax}(\theta)_L \end{bmatrix} \quad \text{and} \quad \nabla_\theta \log \pi_\theta(R|\cdot) = \begin{bmatrix} -\text{softmax}(\theta)_R \\ \text{softmax}(\theta)_L \end{bmatrix}$$

(i) (1 pt) We collect the trajectory $\tau = (\underline{b} \xrightarrow{R} \underline{c} \xrightarrow{L} \underline{b} \xrightarrow{L} \underline{a})$. Compute returns $G_t(\tau)$ for each timestep t in the trajectory.

(ii) (4 pts) Compute the policy gradient $\nabla_\theta J(\theta^{(0)})$ for the first update step using the trajectory from part (i).

(iii) (1 pt) Compute the new parameters $\theta^{(1)}$ after applying the gradient update, and the corresponding probabilities $\pi_\theta(a|s)$ of each action.

4. [Reinforcement Learning] (8 pts) Consider a deterministic grid world, shown in Figure 1, 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) (2 pts) Can you guess an optimal policy? Show it by drawing arrows corresponding to optimal actions for each cell in the grid.

Note: The optimal policy need not be unique, and you should not need to solve for it algorithmically.

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

(c) (4 pts) Show that the Bellman optimality equation for state-value functions holds for V^* at the top left state, using the policy you described in part (a).

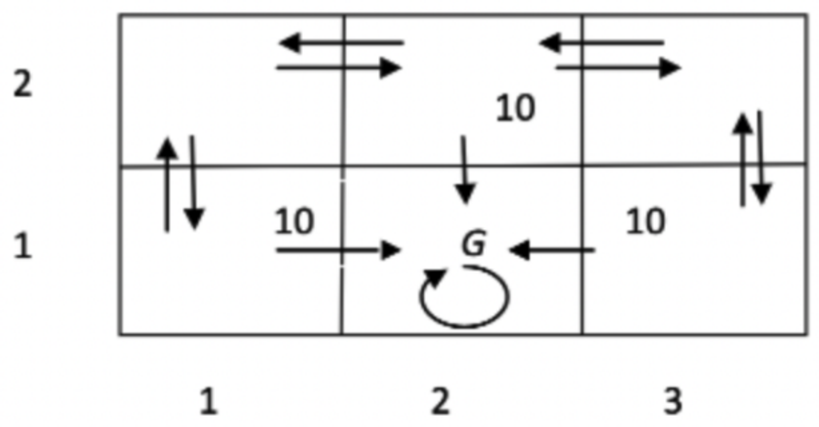

Figure 1: Gridworld

5. [Bellman equations] (15 pts) While we do not typically test you on proofs, this problem is intended to help you understand why the Bellman equations hold, and represent a kind of "law of conservation of rewards," as we have discussed in class. Recall the Bellman equation for action-value functions in the setting of stochastic transitions $P : S \times A \times S \to [0,1]$ and deterministic policies $\pi : S \to A$:

$$Q^\pi(s,a) = \sum_{s' \in S} P(s'|s,a)\left[R(s,a,s') + \gamma Q^\pi\left(s', \pi(s')\right)\right]$$

where Q^π is defined as

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t\geq 0} \gamma^t r_{t+1} \,\middle|\, s_0 = s, a_0 = a\right].$$

(a) (2 pt) State the analogous Bellman equation for deterministic transition functions $P : S \times A \to S$ (and deterministic policies).

(b) (1 pt) Unravel the return $G_0 = \sum_{t\geq 0} \gamma^t r_{t+1}$ by one step to obtain an expression in terms of r_1, γ, and the return G_1. Recall that for trajectory

$$(s_0, a_0, r_1, s_1, \ldots, s_t, a_t, r_{t+1}, \ldots),$$

we define the return from timestep t as:

$$G_t = \sum_{k\geq 0} \gamma^k r_{t+k+1}$$

(c) (2 pts) In 1-2 sentences, cite specific properties of MDPs to explain why

$$\mathbb{E}_\pi\left[G_0 \mid s_0 = s, a_0 = a\right] = \mathbb{E}_\pi\left[G_1 \mid s_1 = s, a_1 = a\right].$$

Remember that G_t includes rewards obtained at all times after time t (until the agent reaches a terminal state or $t \to \infty$).

(d) (3 pts) Use the definition of Q^π, and what we have learned from parts (b) and (c), to prove your stated Bellman equation in part (a) for deterministic transitions and policies.

(e) (6 pts) Now we return to the setting of stochastic transitions (still with deterministic policies) we used in class. The Law of Total Expectation tells us that:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_{\tau\sim\overrightrightarrows{\pi}}\left[G_0 \;\middle|\; s_0 = s, a_0 = a\right] \\ &= \mathbb{E}_{\tau\sim\overrightrightarrows{\pi}}\left[\mathbb{E}_{s'\sim P(\cdot|s,a)}\left[G_0 \;\middle|\; s_1 = s'\right] \;\middle|\; s_0 = s, a_0 = a\right] \end{aligned}$$

(i) (2 pts) Use this fact to rewrite $Q^\pi(s,a)$ as a sum over states $s' \in S$.

(ii) (4 pts) [**5190 Only**] Further rearrange your answer to be expressed in terms of $V^\pi(s') = \mathbb{E}_\pi\left[G_0 \;\middle|\; s_0 = s'\right]$.

(f) (1 pts) [**5190 Only**] Use your result from part (e) to derive the Bellman equation for action-value functions that we presented in class (stochastic transitions and deterministic policies).