CS 4122: Reinforcement Learning

Programming Assignment #1 (for Module 1)

Release date: 15th February, 2025, 12:00 pm (IST)

Due date: 8th March, 2025, 11:59 pm (IST)

Maximum Score: 100 marks (this assignment is graded)

Read the following before you move forward:

1. This programming assignment is worth **10% of the total marks** of this course.

2. **Late policy:** You can be late by a maximum of 3 days. Your assignment will not be accepted more than 3 days after the due date. The **actual score** and **received score** (received score is actual score minus penalty for late submission) are related as follows:

$$\text{received score} = \begin{cases} \text{actual score} & ;\text{late by 0 days} \\ 0.9 \cdot \text{actual score} & ;\text{late by 1 day} \\ 0.8 \cdot \text{actual score} & ;\text{late by 2 days} \\ 0.7 \cdot \text{actual score} & ;\text{late by 3 days} \\ 0 & ;\text{late by more than 3 days} \end{cases}$$

3. This is a **team project**. You have to do the project with the team that was finally selected for you. **Only one submission per team**. To submit, go to your team's Google drive folder. The Google drive link was emailed to you. Then go to the sub-folder titled **Programming Assignment 1** and drop the following FOUR files (DON'T zip/compress these files, DON'T submit any other files):
   (a) report.pdf (b) lin_greedy.py (c) lin_ucb.py (d) policy_gradient.py
   You must read section 7 carefully to understand what needs to be included in these four files.

4. Apart from what is mention in section 7, the **first page of report.pdf should contain a detailed description of the contribution made my individual team members**. NOTE: Documenting the work is NOT a valid contribution for this programming assignment.

5. Even though each team members may contribute to a specific section of the assignment, every team member should have the technical understanding of the entirety of the project. I will be conducting random assessment to check student's understanding of the assignment. Assignment marks will depend on the random assessment as well.

6. Plagiarism, if detected, will lead to a score of ZERO for all the team member; no exceptions!
   a. It is ok to take help from AI tools. It is down right stupid to copy-paste from AI tools and think that is acceptable.
   b. It is ok to take help from other teams. But, it is a bad idea to keep open another teams work in front of you while you do the assignment. While you may think you are just taking help, your reports and your codes may get heavily influenced by the other team to the point that it will qualify as being plagiarized.

Server Allocation in Data Centers for Batch Processing Tasks Using Contextual Bandits

Section 1: Installation

1. I assume that you already have a setup to code in Python. Typically, this would mean that you have **Anaconda** installed in your system. Anaconda has both **Spyder** and **Jupyter Notebook** that can be used to code in Python. NOTE: Even if you are using Jupyter notebook for coding, the final codes that you submit must be Python scripts (.py files) as mentioned in the 1st page of this document.
2. If you are using **Anaconda**, execute the following pip commands in **Anaconda prompt** to install the libraries required for this assignment (for **Colab**, pip should be replaced with !pip):
   pip install gymnasium
   pip install swig
   Please Note: You need to install **OpenAI Gymnasium** and NOT OpenAI Gym *(**Gym** and **Gymnasium** are two different libraries that contains environments for reinforcement learning. Gymnasium is the newer version of Gym and supports more environments than Gym. Hence, install Gymnasium)*.
3. You also have to install a **Deep Learning library** like **Keras, Tensorflow, or Pytorch**. This will be required to implement policy gradient algorithm.

Section 2: Sharing Workload

The way I see it there are four major tasks: *(i)* Implementing ε-greedy policy for linear bandits, *(ii)* Implementing UCB policy for linear bandits, *(iii)* Implementing policy gradient algorithm, and *(iv)* one theory question. If there are four members, four of you can divide these four tasks among yourself. If there are three members, the one who is coding the ε-greedy policy can do the theory question as well. **It is to be clearly noted that these are just suggestions; you are free to divide the task any way you see fit. That said, read points 4 and 5 in the first page of this assignment before dividing the tasks.**

Section 3: Useful Resources

1. All the required pseudocodes are there in lectures 8 to 12.
2. You also need to know **how to use OpenAI Gym environment**. For this, refer to either:
   a. The video lecture of **lecture 0, part 3**. The first 52 minutes is enough.
   b. The following YouTube video by Nicholas Renotte. The first 8 minutes are enough.
   https://www.youtube.com/watch?v=cO5g5qLrLSo&t=581s&ab_channel=NicholasRenotte
3. In the following tutorial I explain how to use OpenAI Gymnasium and also **show how to implement ε-greedy policy for linear bandits** in Python: https://youtu.be/LfH0DzDBtgs. NOTE: The first 36 minutes of this video has huge overlap with lecture 0, part 3. So, you may choose to skip it.

4. You also need to know deep learning and how to code in either Keras, Tensorflow, or Pytorch. In the beginning of the course, I emailed you a bunch of resources in this regard. You can also refer to lecture 0, part 2.

Section 4: Problem Setup

A data center is a physical facility that provides access to computing resources. Among other things, data centers have servers (basically a computer). Users can remotely connect with these servers and use it for their computational needs. Whenever we are using cloud computing websites like Amazon AWS, Google Colab, ChatGPT etc., some server housed in some data center around the world is being used.

Batch processing is a commonly used approach to schedule jobs in data centers. In batch processing, a data center schedules a bunch of jobs/tasks in a periodic manner. In some ways batch processing is better than other commonly used approaches like serving jobs on first-come-first-served basis. This can be explained using the following example. Say that a low priority job arrives first followed by a high priority job. In this example, first-come-first-served approach will process the low priority job first which is likely to be sub-optimal. Batch processing on the other hand will wait for both the jobs to arrive before deciding which one to process first.

Before even batch processing begins, the data centers have to make another critical decision, i.e. how many servers to allocate in order to do a certain batch of jobs? If too many servers are allocated, the jobs are likely to be processed faster but the downside is that it will cost the data center a lot of energy to power these servers. Your task in this programming assignment is to use contextual bandits to decide how many servers to allocate for a batch of jobs so as to minimize a weighted sum of the time required to complete the jobs (also called latency) and the energy cost to power the servers.

Is contextual bandits the right approach for this problem?

A problem can be solved using multiple approaches. Just because you know an approach does not mean it is best suited for the problem under consideration. It is a good problem-solving practice to ask the question: "Why to use a particular approach for a given problem?" Keeping this is mind, let's ask the question: "Why to use contextual bandit for deciding the number of servers for batch processing in data centers?"

Answer: Contextual bandit is useful for this problem for the following reasons:

1. Contextual bandits deal with stochastic setups. In this problem, stochasticity arises because of the randomness in processing time of jobs and energy cost to power servers.
2. Contextual bandits need "context". The context for this problem are the metadata associated with a job (those who took a course on operating systems will know that this metadata is include in the process control block of a job). Metadata includes information like job priority, required computational resources, I/O requirements, etc.
3. Contextual bandits deal with "learning" setup. Learning is required for this problem because the data center does not know the probability distribution governing the processing time of the jobs. Also, the processing time changes depends on the type of jobs, I/O requirements etc.

Section 5: The Gym Environment

NOTE: Even though we will keep saying "Gym" for convenience, we actually mean "Gymnasium".

You are given a Python file *CloudComputing.py* that has a class *ServerAllocationEnv*. This class implements a simplified version of the problem setup mentioned in section 4. More specifically, *ServerAllocationEnv* implements an OpenAI Gym environment for server allocation in data centers for batch processing. In this section, we give a detailed description of *ServerAllocationEnv*.

Any environment for RL/bandits is associated with the concept of **episode**. An episode is a sequence of consecutive time slots associated with a certain task. An episode starts from time slot $t = 0$ and continues for a maximum of say H time slots. After $t = H - 1$, the episode ends. After an episode ends, we have to **reset the environment** in order to start a new episode back from time slot $t = 0$. In our setup, a time slot is of 60 seconds duration, and an episode is an entire day, i.e. $H = 60 \cdot 24 = 1440$.

Consider time slot t. In this time slot, n_t jobs arrive at data center for processing. n_t is a random integer between 1 and 8. The jobs are non-preemptable. The number of jobs, n_t, changes from one time to the next. The i^{th} job of time slot t is associated with the following information/metadata:

1. $w_{i,t}$: The priority of the i^{th} job of time slot t. $w_{i,t}$ is an integer between 1 to 3. The lower the value of $w_{i,t}$, the higher its' priority.
2. $\phi_{i,t}$: The type of the i^{th} job of time slot t. The jobs can be of three types: A, B, and C. Hence, $\phi_{i,t} \in \{A, B, C\}\ \forall i, t$. Practically speaking, types of jobs includes: machine learning jobs that requires huge computational resources, real-time jobs that requires high speed processing, and I/O intensive jobs that requires a lot of network usage.
3. $\rho_{i,t}$: The network usage of the i^{th} job of time slot t. $\rho_{i,t}$ is a real number between 0 to 1. A higher value of $\rho_{i,t}$ for a job implies that it will use more internet time for data transfers and I/O operations.
4. $\delta_{i,t}$: The estimated processing time of the i^{th} job of time slot t . The unit of $\delta_{i,t}$ is seconds. It is a real number between 0 to 60 (remember that one time slot is maximum of 60 seconds and hence the processing time of a job can't be more than 60 seconds).

The datacenter knows the above four information of every job that arrives in a given time slot. This information is the context based on which the data center makes its decision. So the context, c_t, for time slot t is a tuple-of-tuples as shown below,

$$c_t = \Big((w_{1,t}, \phi_{1,t}, \rho_{1,t}, \delta_{1,t}), (w_{2,t}, \phi_{2,t}, \rho_{2,t}, \delta_{2,t}), \cdots, (w_{n_t,t}, \phi_{n_t,t}, \rho_{n_t,t}, \delta_{n_t,t})\Big)$$

It is important to repeat that the number of jobs, n_t, varies with time slot. Hence, the size of the context is itself a variable.

After the data center learns the context c_t of all the jobs, it has to decide how many servers it wants to allocate for the batch of n_t jobs that arrived in time slot t. Data center can allocate a minimum of 1 and a maximum of 8 servers. From the perspective of contextual bandit, the numbers of servers that the data center allocates in a time slot is the **action**. Let a_t denote the number of servers allocated by the data center at time slot t. The data center has to allocate at least one server and at most ten servers. After

data center allocates the servers, it also has to schedule the batch of jobs. As such, scheduling the jobs is also an action[1]. However, we will NOT consider job scheduling as an action. Instead, we use the following hardcoded rule to decide job scheduling: highest priority job first. An example of highest priority jobs first is as follows:

Consider time slot t*. Five jobs arrive in this time slot, i.e.* $n_t = 5$*. The priorities if these jobs indexed* 1 *to* 5 *are* 2*,* 1*,* 3*,* 2*, and* 1 *respectively. Say that the data center allocated* 3 *servers. Then,*

1) *Job indexed* 1*,* 2*, and* 5 *are scheduled first on the three servers.*
    a. *Remember lower value of priority implies a higher priority.*
    b. *We could have scheduled job* 4 *instead of* 1 *because they both have the same priority level).*
2) *Whichever server completes processing one of these three jobs first, will be allocated job* 4*.*
3) *Finally, whichever server gets free next will be allocated job* 3*.*

The data center incurs a cost in every time slot based on the jobs that arrives and the number of servers allocated by the data center. This cost is a weighted sum of:

1. Wait time of all the jobs in the batch. To elaborate, wait time of a job is the amount of the time a job has to wait while the allocated server is processing the other jobs.
2. A huge penalty if wait time of a job is more than 60 seconds (60 seconds is the duration of one time slot).
3. Energy cost of running the allocated servers. More the servers, more is the energy cost.

The weights are according to the priority of the jobs. The **reward** is the negative of this cost. It is to be noted that even though the data center knows the estimated processing time, $\delta_{i,t}$, of each of the jobs, it does not know the true processing time. Hence, latency of the jobs is a random variable as far as the data center is concerned. That said, estimated processing time is a good indicator of latency. The type of job, $\phi_{i,t}$, and the network usage, $\rho_{i,t}$, are also indicators of true processing time and hence latency for the following reasons:

1. For some type of jobs, estimated processing time is a better indicator of the true processing time as compared to other type of jobs.
2. A higher network usage implies a higher true processing time.

Section 6: Using the Gym Environment

You don't have to understand the code in *CloudComputing.py*. You just have to know how to use a Gym environment. In section 3, I have listed a few resources to learn about Gym environments.

VERY IMPORTANT: Now, there is a "STRICT DON'T" as far as using *ServerAllocationEnv* is concerned. As you know, contextual bandit is a learning setup. So, while coding the agent/policy, you are not supposed to know the internal working of *ServerAllocationEnv* other than those topics mentioned in the sections 4 and 5. Now, I can't stop you from looking into the code of *ServerAllocationEnv* and understanding it. But, if I find that any part of your implementation used some internal working of *ServerAllocationEnv* that you

[1] In fact, it can be formulated as a combinatorial bandit problem which is beyond the scope of our syllabus.

are not supposed to know, then it will negatively affect your score. To be on the safe side, limit yourself to using only the following attributes/functions of *ServerAllocationEnv* when implementing the policies:

1. env.Horizon. This is H; the total number of time slots in an episode. In our case, it is 1440.
2. env.MaxServers. This is the maximum number of servers that can be allocated in one time slot. In our case, it is 8.
3. env.MaxJobs. The is the maximum number of jobs that can arrive in one time slot. In our case, it is 8.
4. env.MinPriority. This is the minimum priority level. In our case, it is 3.
5. env.NTypes. This is the number of types of jobs. In our case, it is 3.
6. env.observation_space. The is the context space associated with this environment.
7. env.action_space. This is the action space associated with this environment.
8. env.reset(). This function is required to start a new episode.
9. env.step(action). This function takes an action and returns (i) the **context for the next time slot**, (ii) the **reward** associated with the current context and current action, (iii) a value called **terminated** which is NOT USEFUL for this setup, (iv) a value called **truncated** this is **False** when an episode is not over and **True** when an episode is over, and (v) a dictionary called **information** that is NOT USEFUL for this setup.

Section 7: The Deliverables

1. (5 marks) In every contextual bandits, there must be randomness in the reward associated with an action. Based on your reading of what is discussed above, what are the sources of this noise? The answer to this question must be given in **report.pdf**. The answer not more than half a page.

2. (10 marks) Based on your reading of sections 4 and 5, find features from the context that will be useful for training RL agent in the subsequent parts. Your answer should address:
   - How to deal with the fact that the context size is varying? This is important because most ML/DL models can't deal with varying input size.
   - Can we reduce the number of features? This will eventually help you in training agents in the subsequent parts.

   The answer to this question must be given in **report.pdf**. IMPORTANT: Just some "bare-basic" answer to this question will lead to bare-basic points in this part and the subsequent parts.

3. (15 marks) In the blank Python script *lin_greedy.py* that is provided to you, implement an RL agent for the *ServerAllocationEnv* environment that uses **ε-greedy algorithm for linear bandit**. This Python script should also plot a **receding window time-averaged reward** incurred by the RL agent (window size of 500 time slots in recommended). You MUST NOT include anything in **report.pdf** for this part other than the plot of the receding window time averaged reward.
   - What is receding window time averaged reward? Check this small 9 minutes video.

- The number of episodes you train your RL agent is up to you. The only condition is that the plot of the receding window time averaged reward should "plateau out".
- You may choose to use a time varying exploration rate. How you vary the exploration rate is completely up to you but it must satisfy basic rules of exploration-exploitation tradeoff.

4. (15 marks) In the blank Python script *lin_ucb.py* that is provided to you, implement an RL agent for the *ServerAllocationEnv* environment that uses **UCB algorithm for linear bandit**. This Python script should also plot a **receding window time-averaged reward** incurred by the RL agent (window size of 500 time slots in recommended). You MUST NOT include anything in **report.pdf** for this part.
    - The first two sub-points of *lin_greedy.py* applies here as well.

5. (25 marks) In the blank Python script *policy_gradient.py* that is provided to you, implement an RL agent for the *ServerAllocationEnv* environment that uses **policy gradient algorithm for contextual bandits**. This Python script should also plot a **receding window time-averaged reward** incurred by the RL agent (window size of 500 time slots in recommended). You MUST NOT include anything in **report.pdf** for this part other than the plot of the receding window time averaged reward.
    - The first two sub-points of *lin_greedy.py* applies here as well.
    - During lecture we discussed several variants of the policy gradient pseudocode. All those variants are allowed.
    - The structure of the neural network model is up to you. My advice is to keep it simple.
    - Normalization/standardization of the input to the neural network is important (just like in supervised learning). For standardization, you need to know the mean and standard deviation of the input. In case of supervised learning, finding the mean and standard deviation of the input is straightforward because you have the entire data. Here you don't have all the data in advance; you have to interact with the environment to collect data. So here, you have to compute the mean and standard deviation using **incremental approach**. During lecture, we already discussed how to calculate mean incrementally. Similar approach is possible for standard deviation; you can find it online.

6. (10 marks) It is always a good idea to test whether the policies that you trained is giving logical output. In this regard, test the following:
    (a) Does the number of allocated servers increase as priority increases?
    (b) Does the number of allocated servers increase as the estimated processing time increases?
    (c) Does the number of allocated servers increase as the number of jobs increases?

    You have to do these three tests for all the three policies that you trained above. There is a lot of details about how you will do these tests that I am leaving up to you. The answer to this part should be there in **report.pdf** only. Don't submit any code for this part; not even in **report.pdf**. Just the final answer which can be in form of graphs and/or tables along with necessary explanation.

7. All the policies that you have learned for contextual bandit is for discrete action space. In this question, you will develop a policy gradient algorithm for continuous action space. To simplify, we assume that the **action is a scalar quantity** from $-\infty$ to ∞.

   Recall that $\pi(a|x,\theta)$ is the policy. More specifically, $\pi(a|x,\theta)$ is a conditional probability distribution of action a conditioned on the fact that the current state is x and the parameter of the policy is θ. For continuous action space, we assume that the policy is a Gaussian distribution with mean $\theta_1^T x$ and standard deviation $exp(\theta_2^T x)$,

$$\pi(a|x,\theta) = \frac{1}{exp(\theta_2^T x)\sqrt{2\pi}} \cdot exp\left(-\frac{(a-\theta_1^T x)^2}{2\left(exp(\theta_2^T x)\right)^2}\right)$$

   We have used $exp(\theta_2^T x)$ for standard deviation because standard deviation is always a positive quantity which can be guaranteed by using an exponential function.

   Answer the following questions in **report.pdf only**:

   (a) (4 marks) Write a Python code (just like 1-3 lines) to sample an action from a Gaussian distribution of mean $\theta_1^T x$ and standard deviation $exp(\theta_2^T x)$.

   (b) (10 marks) θ_1 and θ_2 are the parameters of this policy that needs to be updated using gradient ascent. To do this, follows these steps:

   i. STEP 1: Go through the derivation in slides 66 to 85 of lectures 8 to 12. Everything about this derivation remains the same for continuous action space as well. Just that, a summation will get converted into an integral. You don't have to do much here; just convince yourself that the steps of the derivation will hold true for continuous action space as well.
   ii. STEP 2: Now consider equation (P.5) in slide 85. Plug in the Gaussian distribution in place of $\pi(a_t|\theta,x_t)$. Then, simplify the resulting equation.
   iii. STEP 3: Partially differentiate the resulting equation from step 2 with respect to θ_1 to get a **closed-form expression** of an estimate of $\nabla_{\theta_1} J$.
   iv. STEP 4: Partially differentiate the resulting equation from step 2 with respect to θ_2 to get a **closed-form expression** of an estimate of $\nabla_{\theta_2} J$.
   v. STEP 5: Using your answers from steps 3 and 4, write two questions showing how to update θ_1 and θ_2 using gradient ascent.

   Document steps 2 to 5 in **report.pdf**. DON'T document step 1.

   (c) (6 marks) Finally, write a pseudocode of the policy gradient algorithm that we discussed here.