Zmh6339
AI & Machine Learning, CS-UH 3260
Professor Keith Ross

Ziad Hassan
January 31, 2024

Assignment 1

Part 1: Continuous Bandit Algorithm

Link to Google Colab

Here is the link to the Google Colab notebook for the code.

Pseudocode

Algorithm 1 Neural Network Algorithm for Continuous Bandit Problem

Initialize parameters: dimension d, matrix R, learning rate, initial samples, iterations, samples per iteration
Define $cost\_function(a, R)$
Define $generate\_random\_actions(num\_samples, d)$
Define MLPWithGrad with $input\_dim$
Define $train\_model(model, optimizer, loss\_function, actions, costs, epochs)$
Define $gradient\_descent\_on\_action(model, action, learning\_rate, steps)$
Define $select\_actions\_based\_on\_prediction(model, num\_actions, d, num\_candidates)$
$initial\_actions, initial\_costs \leftarrow$ Generate and calculate costs
$actions\_tensor, costs\_tensor \leftarrow$ Convert to tensors
Set up MLP model and optimizer
Train MLP model on initial samples
for each iteration **do**
    $selected\_actions \leftarrow$ Select actions based on MLP predictions
    for each action in $selected\_actions$ **do**
        $refined\_action \leftarrow$ Apply gradient descent on action
        Update actions and costs tensors with $refined\_action$
    end for
    Retrain MLP model with updated data
end for
$best\_action, best\_cost \leftarrow$ Find action with minimum cost

Algorithm Description

This algorithm combines a neural network-based prediction model with an optimized action selection strategy. Initially, it uses random sampling to gather a diverse set of actions from the space $[-2, 2]^d$. These actions and their associated costs, calculated using a predetermined cost function, form the training data for a Multi-Layer Perceptron (MLP). The MLP, designed to approximate the cost function, learns from this data, enabling it to make informed predictions about the costs of new actions.

The main part of the algorithm lies in its iterative refinement process, where it selects new actions based on the MLP's cost predictions, rather than random sampling all $10,000$ actions like the baseline algorithm. This method represents a shift from exploration to exploitation by using the MLP's learned patterns to guide the search towards areas in the action space predicted to yield lower costs. Selected actions are further refined through gradient descent, adjusting them in the direction that minimizes the predicted cost. This approach ensures that the algorithm not only learns from its past experiences (initial random samples) but also actively makes it explore in a more focused manner to quickly find the optimal action.

Algorithm Results

Baseline Algorithm

- $R = \mathbb{I}_{30}$

```
# Run the baseline algorithm
best_action, min_cost = random_sampling_baseline(d, R)

print("Minimum Cost:", min_cost)
print("Action with Minimum Cost:", best_action)
```

```
Minimum Cost: 19.161972152934936
Action with Minimum Cost: [-0.83450898  1.21780109 -0.20325143  0.93514716 -0.27157209 -0.36833186
  0.10116192 -0.93066379 -0.04002379  0.95269479  0.48564846 -0.90046208
  0.60194963  0.48598302 -0.01915248  1.18892618 -0.48960774  0.88727396
  1.4295038   1.30346536 -0.74762967  0.6291577   0.38000298 -0.51304459
 -1.38953527 -0.93947524 -0.29698969 -0.7566904  -1.10758602 -0.42396255]
```

- $R =$ random matrix

```
[65] # Run the baseline algorithm
best_action, min_cost = random_sampling_baseline(d, R)

print("Minimum Cost:", min_cost)
print("Action with Minimum Cost:", best_action)
```

```
Minimum Cost: 308.8819186918877
Action with Minimum Cost: [-0.83174832  1.83659625  0.08314279 -1.28687975  0.07376016 -0.14733009
  0.41859857  0.43693495 -0.94471631  0.84217628 -0.88258377  1.77234832
 -0.86903243 -0.1862842  -0.47025847 -0.26705909 -0.77614907 -0.2472248
 -0.75598564  0.59788623  0.67978326  0.01486975 -0.76467882 -0.9213135
 -0.61208655 -1.31603461 -0.34872563  0.5385937  -0.65602394  0.38394475]
```

Neural Network Algorithm

Hyperparameters:

```
num_initial_samples = 500
num_iterations = 95
num_samples_per_iter = 100
Total number of samples = 500 + (95 * 100) = 10,000
learning_rate = 0.01
```

- $R = \mathbb{I}_{30}$

Ziad Hassan


```
# Find the best action
best_cost, best_action_index = torch.min(costs_tensor, 0)
best_action = actions_tensor[best_action_index].numpy()

print("Minimum Cost:", best_cost.item())
print("Action with Minimum Cost:", best_action)
```

```
Minimum Cost: 2.8079819679260254
Action with Minimum Cost: [[ 0.3546725  -0.11318836  0.11688691 -0.7857074   0.37984022  0.03942293
   0.47019467 -0.06927031 -0.03652958  0.30821317 -0.51663136 -0.08281938
  -0.13382208  0.12202585 -0.37522838 -0.5282538   0.45708972 -0.2091363
  -0.12895055  0.31381622 -0.44711357  0.03460172  0.21378729  0.21422534
  -0.06124394  0.02385355  0.10785682 -0.30157408  0.2536847   0.11037226]]
```

- R = random matrix

```
# Find the best action
best_cost, best_action_index = torch.min(costs_tensor, 0)
best_action = actions_tensor[best_action_index].numpy()

print("Minimum Cost:", best_cost.item())
print("Action with Minimum Cost:", best_action)
```

```
Minimum Cost: 25.04450225830078
Action with Minimum Cost: [[-0.10271379 -0.06647134  0.15941732 -0.07121614 -0.05333747  0.01652632
   0.1118411   0.32495314  0.44559485 -0.5347324  -0.22345197  0.05995404
  -0.06193    -0.15328518  0.1145125  -0.10895859  0.05912644  0.07123166
  -0.20409694  0.16479968 -0.15978023 -0.10759465 -0.38867533  0.12673546
   0.46392092 -0.30364084 -0.28328708 -0.0343338   0.3023249  -0.12673491]]
```

Part 2: Theory

a) *Proof.*

$$
\begin{aligned}
P(\text{greedy action}) &= 1 - P(\text{not greedy action}) \\
&= 1 - P(\text{not greedy selection } AND \text{ not greedy action}) \\
&= 1 - P(\text{not greedy selection}) \cdot P(\text{not greedy action} \mid \text{not greedy selection}) \\
&= 1 - \epsilon \cdot \left(\frac{k-1}{k}\right) \qquad {}^{*}\textit{ Since there is only } \textbf{\textit{one}} \textit{ greedy action} \\
&= 1 - \epsilon \cdot \left(1 - \frac{1}{k}\right) \\
&= 1 - \epsilon + \frac{\epsilon}{k}
\end{aligned}
$$

■

b) i) *Proof.*

To determine the probability that the greedy action was chosen for the first time at time T, we need to consider that it was not chosen at any time before T, and that it was chosen at time T.
Thus, the following equation should be quantified:

$$P(\text{greedy at } T) = P(\text{not greedy before } T) \cdot P(\text{greedy at } T)$$

Therefore, the probability that the greedy action was chosen for the first time at time T is:

$$\begin{aligned} P(\text{greedy at } T) &= P(\text{not greedy before } T) \cdot P(\text{greedy at } T) \\ &= \left(1 - 1 + \epsilon - \frac{\epsilon}{k}\right)^{T-1} \cdot \left(1 - \epsilon + \frac{\epsilon}{k}\right) \\ &= \left(\epsilon - \frac{\epsilon}{k}\right)^{T-1} \cdot \left(1 - \epsilon + \frac{\epsilon}{k}\right) \end{aligned}$$

■

ii) *Proof.*

To get the expected number of steps, $\mathbb{E}[T]$, until a greedy action is chosen, we can consider $P(\text{greedy action})$ as a geometric series. Therefore, the expected number of steps until a greedy action is chosen is:

$$\mathbb{E}[T] = \frac{1}{1 - \epsilon + \frac{\epsilon}{k}}$$

■

c) i) *Proof.*

Firstly, denote $\max(q(1), q(2), \ldots, q(10))$ as q_*.
The algorithm will choose the greedy selection, q_* with probability $1 - \epsilon$, and a non-greedy selection with probability ϵ.
Moreover, the expected value of the an action during non-greedy selection is the average of all the action values, which is $\frac{\sum_{i=1}^{10} q(i)}{10}$.
Therefore, the long-run reward is

$$R = (1 - \epsilon) \cdot q_* + \epsilon \cdot \frac{\sum_{i=1}^{10} q(i)}{10}$$

■

ii) *Proof.*

Since $q(1), q(2), \ldots, q(10)$ are i.i.d. random variables, the expected value of a greedy action becomes $\mathbb{E}[q_*] = b$.
Moreover, since $q(a)$ is $\mathcal{N}(0, 1)$, the expected average of all the action values becomes $\mathbb{E}\left[\frac{\sum_{i=1}^{10} q(i)}{10}\right] = 0$.
Therefore, the long-run reward is

$$\begin{aligned} R &= (1 - \epsilon) \cdot \mathbb{E}[q_*] + \epsilon \cdot \mathbb{E}\left[\frac{\sum_{i=1}^{10} q(i)}{10}\right] \\ &= (1 - \epsilon) \cdot b + \epsilon \cdot 0 \\ &= (1 - \epsilon) \cdot b \end{aligned}$$

■

d) • Restatement In any stochastic bandit problem, denoted as ν, where you have a set of actions A that is either finite or can be counted (like a list), and a fixed number of rounds or steps $n \in \mathbb{N}$, the regret R_n of following a certain strategy or policy π in this environment can be calculated as follows:

The total regret R_n is the sum of the regrets for each individual action in A. The regret

for each action a is the product of two things: the suboptimality gap Δ_a of action a (which is how much less reward you expect from action a compared to the best possible action), and the expected number of times $E[T_a(n)]$ that action a is chosen within the first n rounds.
Mathematically, this is represented as:

$$R_n = \sum_{a \in A} \Delta_a E[T_a(n)]$$

This means you add up the products of the suboptimality gap and the expected number of selections for each action to find the total regret.

- Proof

Proof.

Before the proof, we need to redefine the above terms to be consistent with S&B's notation.
Firstly, the set of actions, A, is the set of arms, k.
Secondly, the suboptimality gap, Δ_a, is the difference between the expected reward of the optimal action, $\max_{1 \le a \le k} q_*(a)$, and the expected reward of action $q_*(a)$:

$$\Delta_a = \max_{1 \le a \le k} q_*(a) - q_*(a)$$

Thirdly, the expected number of times $E[T_a(n)]$ that action a is chosen within the first n rounds is the expected number of times that action a is chosen within the first n rounds:

$$\mathbb{E}[T_a(n)] = \mathbb{E}[N_n(a)]$$

Since the regret is the difference in choosing the optimal action and the action chosen, it can be expressed as follows:

$$\text{Reg}_n = n \cdot \max_{1 \le a \le k} q_*(a) - \mathbb{E}[R_n]$$

where n is the number of rounds, $\max_{1 \le a \le k} q_*(a)$ is the expected reward of the optimal action, and $\mathbb{E}[R_n]$ is the expected reward of the action chosen up to round n.
To further break down the term $\mathbb{E}[R_n]$, we can transform it into a term that descirbes actions, rather than rewards.
Thus, consider the following representation:

$$R_n = \sum_{t=1}^{n} \sum_{a=1}^{k} R_t(a) \cdot \mathbb{1}_{A_t = a}$$

Now, the regret can be expressed as follows:

$$\begin{aligned} \text{Reg}_n &= n \cdot \max_{1 \le a \le k} q_*(a) - \mathbb{E}[R_n] \\ &= n \cdot \max_{1 \le a \le k} q_*(a) - \mathbb{E}\left[\sum_{t=1}^{n} \sum_{a=1}^{k} R_t(a) \cdot \mathbb{1}_{A_t = a}\right] \end{aligned}$$

Using properties of the expected value and that we are summing over n time steps, the above equation can be simplified to the following:

$$\begin{aligned} \text{Reg}_n &= n \cdot \max_{1 \le a \le k} q_*(a) - \mathbb{E}\left[\sum_{t=1}^{n} \sum_{a=1}^{k} R_t(a) \cdot \mathbb{1}_{A_t = a}\right] \\ &= \sum_{a=1}^{k} \sum_{t=1}^{n} \mathbb{E}\left[\max_{1 \le a \le k} q_*(a) - R_t(a) \cdot \mathbb{1}_{A_t = a}\right] \end{aligned}$$

The expected reward on round t is conditional on the action chosen on round t, A_t.
Thus, the expected value term can be expressed as follows:

$$\begin{aligned}\mathbb{E}\left[\max_{1\le a\le k} q_*(a) - R_t(a)\cdot \mathbb{1}_{A_t=a}|A_t\right] &= \mathbb{1}_{A_t=a}\mathbb{E}\left[\max_{1\le a\le k} q_*(a) - R_t(a)|A_t\right] \\ &= \mathbb{1}_{A_t=a}\left(\max_{1\le a\le k} q_*(a) - q(a)\right) \\ &= \mathbb{1}_{A_t=a}\cdot\Delta_a\end{aligned}$$

The term $\sum_{t=1}^{n}\mathbb{1}_{A_t=a}$ is the number of times that action a is chosen within the first n rounds, $N_n(a)$.
Therefore, we get the final result that the formula for the regret is

$$\text{Reg}_n = \sum_{a=1}^{k}\Delta_a\cdot\mathbb{E}[N_n(a)]$$

■

e)
- Algorithm Restatement in S&B Terms
  (a) **Initialization:**
  - For each action a (in $1, 2, ..., k$):
    - Initialize action-value estimate $Q_t(a) = 0$.
    - Initialize the number of times action a has been chosen, $N_t(a) = 0$.

  (b) **Loop for each step $t = 1, 2, 3, ...$:**
  - Select action A_t using the UCB criterion:

$$A_t = \arg\max_a\left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}\right]$$

    where c is a confidence level parameter.
  - Take action A_t, observe reward R_t.
  - Update the action-value estimate for A_t:

$$Q_{t+1}(A_t) = Q_t(A_t) + \frac{1}{N_t(A_t)}(R_t - Q_t(A_t))$$

  - Increment $N_t(A_t)$.

- Proof Restatement
  In a k-armed bandit problem using the UCB algorithm, for any time step t, the expected regret R_t is bounded by:

$$R_t \le 3\sum_{i=1}^{k}\Delta_i + \sum_{i:\Delta_i>0}\frac{16\log(t)}{\Delta_i}$$

  where $\Delta_i = q_*(a^*) - q_*(a_i)$ is the regret associated with choosing action a_i instead of the optimal action a^*.

  (a) **Optimal Action Assumption:**
  Assume that the first arm a_1 is optimal, so $q_*(a_1) = q_*(a^*)$, where $q_*(a)$ represents the true value of action a.

(b) **Regret Decomposition:**
The total regret after t steps is:

$$R_t = \sum_{i=1}^{k} \Delta_i E[N_t(a_i)]$$

where $E[N_t(a_i)]$ is the expected number of selections of action a_i up to time t.

(c) **Defining "Good" Event G_i:**
Define the good event G_i for a suboptimal arm a_i as:

$$G_i = \left\{ q_*(a_1) < \min_{\tau \leq t} UCB(a_1, \tau) \right\} \cap \left\{ \hat{q}_\tau(a_i) + \sqrt{\frac{2\log(1/\delta)}{N_\tau(a_i)}} < q_*(a_1) \, \forall \, \tau \leq t \right\}$$

where $\hat{q}_\tau(a_i)$ is the estimated value of action a_i at time τ, $N_\tau(a_i)$ is the number of times action a_i has been selected up to time τ, and δ is a confidence level parameter.

(d) **Bounding $E[N_t(a_i)]$ under G_i:**
Show that under G_i, a_i is selected at most u_i times, where u_i is a function of Δ_i and δ. This leads to:

$$E[N_t(a_i)|G_i] \leq u_i$$

(e) **Bounding Probability of G_i^c:**
The probability of the complement event G_i^c (where the good event does not occur) is small. It is bounded using concentration inequalities.

(f) **Expected Selection Bound:**
Combine these results to express $E[N_t(a_i)]$:

$$E[N_t(a_i)] = E[N_t(a_i)|G_i]P(G_i) + E[N_t(a_i)|G_i^c]P(G_i^c) \leq u_i + tP(G_i^c)$$

(g) **Choosing u_i and c:**
The choice of u_i is made to ensure that the upper confidence bound for suboptimal arm a_i is properly controlled. A natural choice for u_i, considering the balance between exploration and exploitation, is given by:

$$u_i = \left\lceil \frac{2\log(1/\delta)}{(1-c)^2\Delta_i^2} \right\rceil$$

where δ is the confidence level parameter, and c is a constant.
In the proof, c is chosen to be $1/2$ somewhat arbitrarily but in a way that balances the two terms in the regret bound. This choice leads to a simplification of the regret bound while maintaining a balance between exploration and exploitation.

(h) **Finalizing the Bound on R_t:**
With u_i and c chosen, the final bound on R_t is derived by substituting these values into the regret decomposition:

$$R_t \leq \sum_{i:\Delta_i>0} \Delta_i \left(\left\lceil \frac{2\log(1/\delta)}{(1-c)^2\Delta_i^2} \right\rceil + tP(G_i^c) \right)$$

Applying the choice of $c = 1/2$, the final bound simplifies to:

$$R_t \leq 3\sum_{i=1}^{k} \Delta_i + \sum_{i:\Delta_i>0} \frac{16\log(t)}{\Delta_i}$$

f) We consider a stochastic k-armed bandit problem where the goal is to maximize cumulative rewards over time.

We use the Upper Confidence Bound (UCB) algorithm for arm selection. Let Q_n be the average reward at time n for a chosen arm, and let a^* be the optimal arm.
We aim to prove that:

$$\lim_{n\to\infty} \mathbb{E}[Q_n] = q_*(a^*)$$

where $q_*(a^*)$ is the expected reward of the optimal arm.

Proof.

- Theorem 7.1 Condition
  Theorem 7.1 provides an upper bound on the regret R_n for the UCB algorithm: Under the conditions of Theorem 7.1 and using Algorithm 3, the expected estimated value of the action selected at a large time step n converges to the true value of the best action.
- Definition of Q_n
  Q_n is defined as the average of rewards received from taking action a up to time t:

  $$Q_n = \frac{\sum_{i=1}^{t} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t} \mathbb{1}_{A_i=a}}$$

  where R_i is the reward received at time i, A_i is the action taken at time i, and $\mathbb{1}_{A_i=a}$ is the indicator function.

By the law of large numbers, for each arm a, the average reward converges to the expected reward $q_*(a)$ as n increases.

Therefore,

$$\frac{\sum_{i=1}^{n} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{n} \mathbb{1}_{A_i=a}} \to q_*(a)$$

as $n \to \infty$.

The UCB algorithm selects arms based on the estimated reward and the uncertainty in that estimate. Over time, it favors arms with higher estimated rewards.

For the optimal arm a^*, as $n \to \infty$, the UCB algorithm will increasingly favor this arm, assuming it correctly identifies it. Therefore, the frequency of selecting arm a^*, denoted by $N_{a^*}(n)$, will dominate over other arms, leading to:

$$\frac{N_{a^*}(n)}{n} \to 1$$

and for each suboptimal arm a,

$$\frac{N_a(n)}{n} \to 0$$

Given the dominance of the optimal arm in selections, the limit of the expected average reward $\mathbb{E}[Q_n]$ converges to the expected reward of the optimal arm:

$$\lim_{n\to\infty} \mathbb{E}[Q_n] = q_*(a^*)$$

Under the UCB algorithm, the expected average reward for the arm selected by the algorithm converges to the expected reward of the optimal arm as the number of trials n goes to infinity.

■