Addition of PPO, Twin Delayed DDPG, Hindsight Experience Replay to RL Codebase

Basic Information :

Name : Eshaan Agarwal
University : Indian Institute of technology (BHU) Varanasi
Field of Study : Electrical Engineering
Time study was started: Nov 2020
Expected Graduation date: July 2024
Degree: Bachelor of Technology
Year: Sophomore
Github: eshaanagarwal
LinkedIn: eshaanagarwal
IRC nick: eshaanagarwal
Email Address: eshaan060202@gmail.com
Phone number: (+91) 8989113000
CV:https://drive.google.com/file/d/1tznD38Cz-ayNb_FhOAu1AoO5qZ8hnCD2/view?usp=sharing
Timezone: Indian Standard Time (UTC +5:30)

Table of Contents :

The Project Proposal

Objective

While going through the Reinforcement Learning codebase of Mlpack, I noticed that a lot of state-of-the-art algorithms are missing. So after comparing various algorithms and brainstorming about their methods of implementation, I came up with the **PPO** (Schulman et al., 2017), **TD3** (Fujimoto et. al 2018) and **Hindsight Experience Replay (HER)** (Andrychowicz et al., 2018) as the most in-demand and must-have algorithms, whose implementation in mlpack would be crucial.

So here are the details of what I expect to have accomplished at the end of the summer.

- ➔ Addition of Hindsight Experience Replay along with thorough testing
- ➔ Implementing Proximal Policy Gradient (PPO) along with its tests
- ➔ Implementing TD3 along with test cases.
- ➔ Creating detailed docs for all the above implementations
- ➔ Benchmarking of Current RL Algorithms on various environments and comparing performances between algorithms - for example : TD3+HER vs TD3 or DQN+HER vs DQN
- ➔ Addition of simple tutorials demonstrating usability of implemented algorithm
- ➔ Creating necessary environments, for proper testing of algorithms above (after discussion with mentor)

Background Info :

This includes descriptions of the algorithms / data structures / ideas I plan to implement. I made sure to detail background information, such that a person who is reasonably familiar with machine learning and mlpack will be able to understand the description without needing to consult other references.

Part 1: Hindsight Experience Replay : Learn from Failure

Sparse Binary Reward Environments :
Often real-life situations are challenging and have sparse binary rewards i.e. either we win or lose the game. Having no intermediate rewards during the episodes makes

learning extremely difficult in most cases, as the agent might never actually win, and therefore have no feedback on how to improve its performance.

A traditional approach to tackle the above problem has been to augment the reward using domain knowledge, in what is known as Reward Engineering but this is not always easy to do. Often we are not able to engineer the reward properly. Another danger is that once we engineer the reward, we are no longer directly optimizing for the metric we are really interested in, but instead optimizing a proxy that we hope will make the learning process easier. This could cause a compromise in performance relative to the true objective, and sometimes even lead to unexpected and unwanted behavior that might necessitate a frequent fine tuning of the engineered reward in order to get it right.

Multi Goal RL :
Often existing problems that we have attempted to solve with Reinforcement Learning have a specific objective, such as “score as many points as possible in a game like Breakout”.

But in reality, many real-world problems are not like that, we have multiple goals / tasks instead of a global task. Therefore we expect that our agent will be able to achieve many different goals, such as “fetch the red ball”, or “fetch the green cube” i.e.any of these tasks upon request.

In that case we can express our policy as:

$$\pi(a|s, g)$$

Where ‘g’ is the desired goal. This is a multi-goal learning problem.

We know data in the experience replay buffer can originate from an exploration policy in off policy RL algorithms. HER deals with a very important question in regard with experience replay :

What if we could add fictitious data, by imagining what would happen had the circumstance been different?

In HER, the authors suggest the following strategy: suppose our agent performs an episode of trying to reach goal state G from initial state S, but fails to do so and ends up in some state S’ at the end of the episode. We cache the trajectory into our replay buffer:

$$\{(S_0, G, a_0, r_0, S_1), (S_1, G, a_1, r_1, S_2), \dots, (S_n, G, a_n, r_n, S')\}$$

The idea in HER is to imagine that our goal has actually been S’ all along, and that in this alternative reality our agent has reached the goal successfully and got the positive reward for doing so. So, in addition to caching the real trajectory as seen before, we also cache the following trajectory:

$$\{(S_0, S', a_0, r_0, S_1), (S_1, S', a_1, r_1, S_2), \dots, (S_n, S', a_n, r_n, S')\}$$

This trajectory is the imagined one, and is motivated by the human ability to learn useful things from failed attempts. **By introducing the imagined trajectories to our replay buffer, we ensure that no matter how bad our policy is, it will always have some positive rewards to learn from.**

The idea is this will help the agent explore better and also learn intermediate goals that build up to the actual desired goal.

HER allows sample-efficient learning from rewards which are sparse and binary and therefore avoid the need for complicated reward engineering.
It can be combined with an arbitrary off-policy RL algorithm like (DQN, SAC, DDPG, TD3) and may be seen as a form of implicit curriculum.

Graphs and Result :

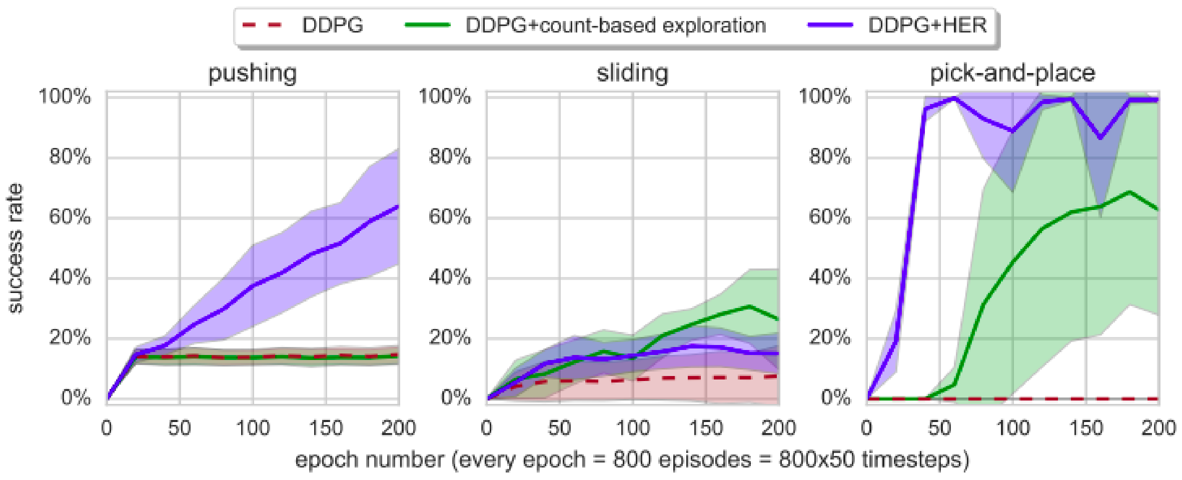
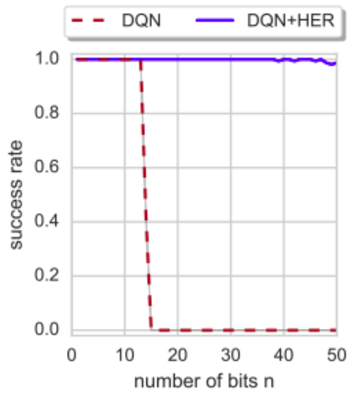
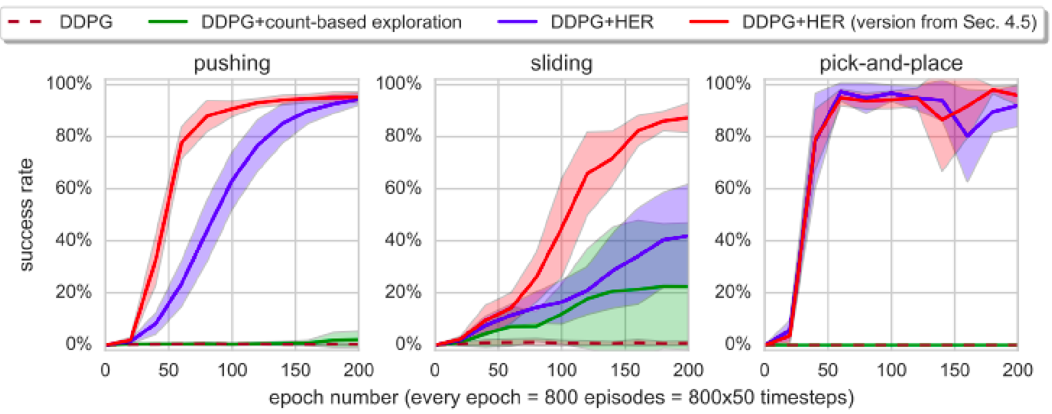


Figure 4: Learning curves for the single-goal case.





Part 2 : Proximal Policy Gradient - Clipped PPO

Overview :

Policy gradient methods are very popular class of algorithms and have been precursor to recent breakthroughs of using RL in control, locomotion etc; Having said that, these algorithms are sensitive to hyperparameters like stepsize — too small, and progress is hopelessly slow; too large and the signal is overwhelmed by the noise, or one might see catastrophic drops in performance. They also often have very poor sample efficiency, taking millions (or billions) of timesteps to learn simple tasks.

Researchers have sought to eliminate these flaws with approaches like TRPO and ACER, by constraining or otherwise optimizing the size of a policy update.

These methods have their own trade-offs —

ACER is far more complicated than PPO, requiring the addition of code for off-policy corrections and a replay buffer, while only doing marginally better than PPO on the Atari benchmark;

TRPO — though useful for continuous control tasks — isn't easily compatible with algorithms that share parameters between a policy and value function or auxiliary losses, like those used to solve problems in Atari and other domains where the visual input is significant. A major disadvantage of TRPO is that it's computationally expensive, Schulman et al. proposed proximal policy optimization (PPO) to simplify TRPO by using a clipped surrogate objective while retaining similar performance.

To summarize, **PPO is simpler, faster, and more sample efficient then most of its competitors**

Algorithmic Description :

The variant I would be implementing has novel objective function not typically found in other algorithms:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\ ]$$

- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\hat{A}_t$is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

Clipping of PPO serves as a regularizer by removing incentives for the policy to change dramatically, and the hyperparameter epsilon corresponds to how far away the new policy can go from the old while still profiting the objective. The objective function takes the minimum between the clipped and unclipped objective, so the final objective is a pessimistic bound on the unclipped one.

This objective implements a way to do a Trust Region update which is compatible with Stochastic Gradient Descent, and simplifies the algorithm by removing the KL penalty and need to make adaptive updates. In tests, this algorithm has displayed the best performance on continuous control tasks and almost matches ACER's performance on Atari, despite being far simpler to implement.

You can check out this link to understand PPO's performance.

Graphs and Flow Chart :

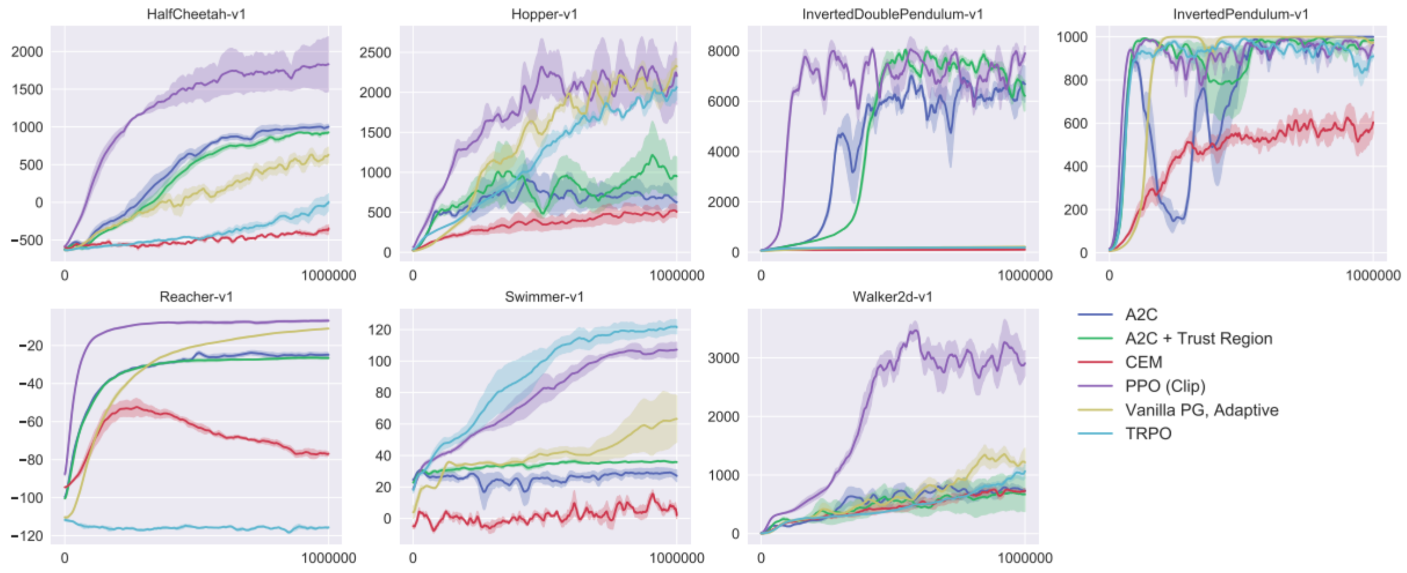


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

Part 3 : Twin Delayed Deep Deterministic Policy Gradients

Overview :

While DDPG can achieve great performance sometimes, it is frequently brittle with respect to hyperparameters and other kinds of tuning. A common failure pattern for DDPG is that the learned Q-function begins to dramatically overestimate Q-values, which then leads to the policy breaking, because it exploits the errors in the Q-function. **Twin Delayed DDPG (TD3) is an algorithm that addresses this issue by introducing three critical tricks.**
Together, these three tricks result in substantially improved performance over baseline DDPG.

What is DDPGs :

Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. DDPG can only be used for environments with continuous action spaces and is often thought of as being deep Q-learning for continuous action spaces.

Because the action space is continuous, the function Q^*(s,a)(optimal state value function) is presumed to be differentiable with respect to the action argument. This allows us to set up an efficient, gradient-based learning rule for a policy mu(s) which exploits that fact. Then, instead of running an expensive optimization subroutine each time we can approximate it with $\max_a Q(s,a) \approx Q(s,\mu(s))$.

Why TD3 ?

Three critical tricks that help in removing overestimation are -

Trick One: Clipped Double-Q Learning - TD3 learns two Q-functions instead of one (hence “twin”), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

$$y(r, s', d) = r + \gamma(1-d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')),$$

and then both are learned by regressing to this target:

$$L(\phi_1, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_{\phi_1}(s,a) - y(r,s',d) \right)^2 \right],$$

$$L(\phi_2, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_{\phi_2}(s,a) - y(r,s',d) \right)^2 \right].$$

Using the smaller Q-value for the target, and regressing towards that, helps fend off overestimation in the Q-function.

Trick Two: "Delayed" Policy Updates - TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.

Policy is learned just by maximizing Q_{ϕ_1}:

$$\max_{\theta} \underset{s\sim\mathcal{D}}{\mathrm{E}} \left[Q_{\phi_1}(s, \mu_\theta(s)) \right],$$

This is pretty much unchanged from DDPG. However, in TD3, the policy is updated less frequently than the Q-functions are. This helps damp the volatility that normally arises in DDPG because of how a policy update changes the target.

Trick Three: Target Policy Smoothing -TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Actions used to form the Q-learning target are based on the target policy, $\mu_{\theta_{\text{targ}}},$ but with clipped noise added on each dimension of the action. After adding the clipped noise, the target action is then clipped to lie in the valid action range (all valid actions, a, satisfy $a_{Low} \leq a \leq a_{High}$). The target actions are thus:

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High} \right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

Target policy smoothing essentially serves as a regularizer for the algorithm. It addresses a particular failure mode that can happen in DDPG: if the Q-function approximator develops an incorrect sharp peak for some actions, the policy will quickly exploit that peak and then have brittle or incorrect behavior. This can be averted by smoothing out the Q-function over similar actions, which target policy smoothing is designed to do.

Difference and Similarities in TD3 and SAC : (Keeping SAC as reference since it has been already implemented in a GSoC 2020 Project)

The Q-functions are learned in a similar way in both TD3 and SAC, but with a few key differences.

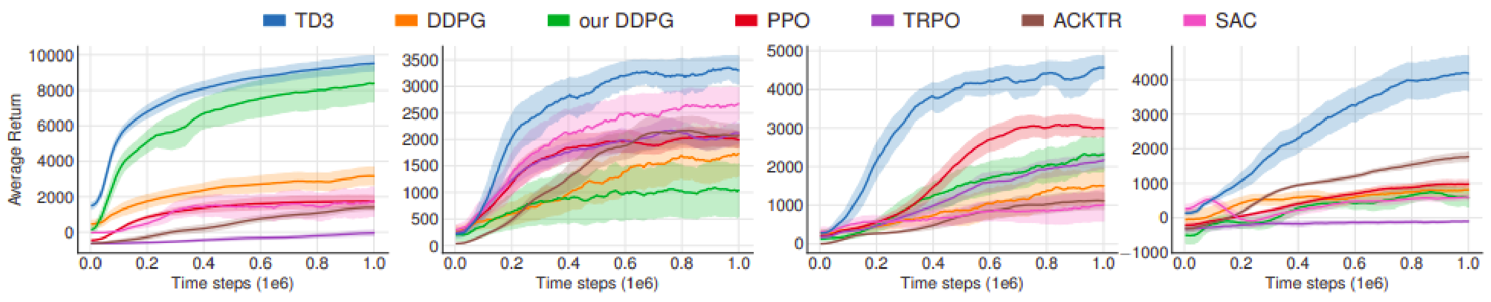
First, what's similar?

1. Like in SAC, both Q-functions are learned with MSBE minimization, by regressing to a single shared target.
2. Like in SAC, the shared target is computed using target Q-networks, and the target Q-networks are obtained by polyak averaging the Q-network parameters over the course of training.
3. Like in SAC, the shared target makes use of the clipped double-Q trick.

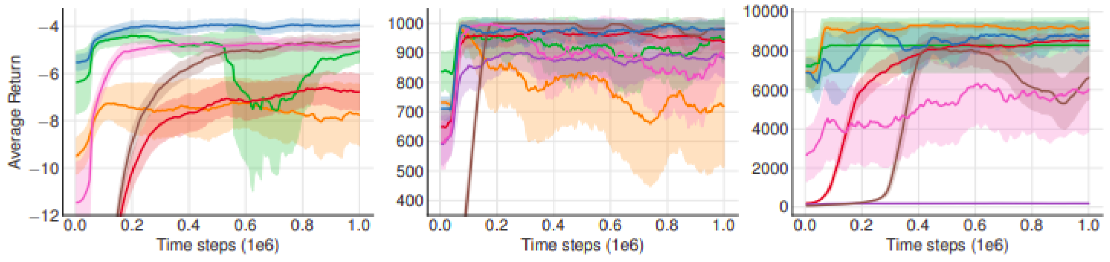
What's different?

1. Unlike in TD3, the target also includes a term that comes from SAC's use of entropy regularization.
2. Unlike in TD3, the next-state actions used in the target come from the current policy instead of a target policy in SAC.
3. Unlike in TD3, there is no explicit target policy smoothing. TD3 trains a deterministic policy, and so it accomplishes smoothing by adding random noise to the next-state actions. SAC trains a stochastic policy, and so the noise from that stochasticity is sufficient to get a similar effects

Graphs :



(a) HalfCheetah-v1 (b) Hopper-v1 (c) Walker2d-v1 (d) Ant-v1



(e) Reacher-v1 (f) InvertedPendulum-v1 (g) InvertedDoublePendulum-v1

TD3 matches or outperforms all other algorithms in both final performance and learning speed across all tasks.

Part 1 : Implementation of Hindsight Experience Replay (HER)

Abstract

For hindsight experience replay :

The simple idea is that after experiencing some episode s0, s1, . . . , sT we store in the replay buffer every transition st → st+1 not only with the original goal used for this episode but also with a subset of other goals.

Note : We can easily notice that the goal being pursued influences the agent's actions but not the environment dynamics and therefore we can replay each trajectory with an arbitrary goal to any off policy RL algorithm.

Goal Selection Strategy for HER :

How can goals be selected with which we can replay trajectories ?

In the most basic version of HER, we replay each trajectory with the goal m(sT), i.e. the goal which is achieved in the final state of the episode. This strategy is called **final.**

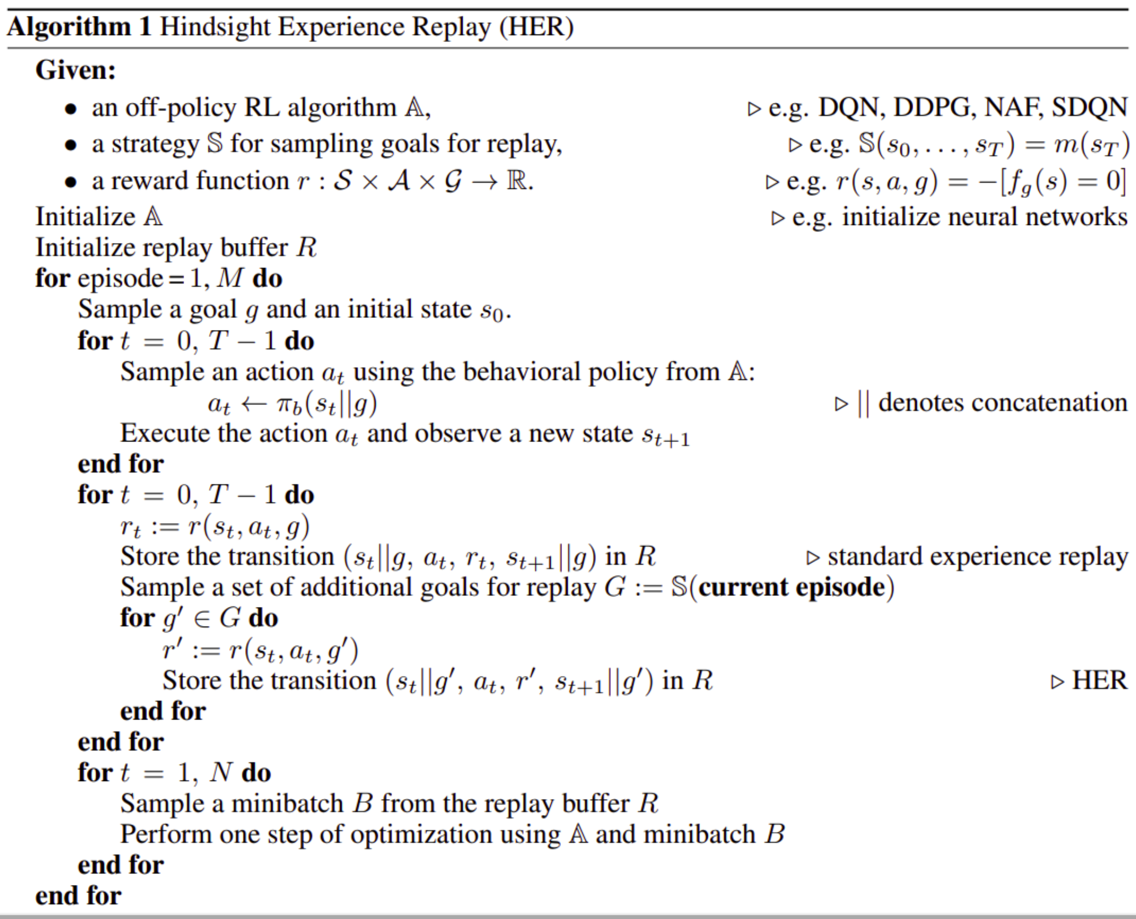
There could be other strategies for goal selection. Authors of HER in their paper also experimented between 4 strategies :

- **final :** replay whole trajectory with final state as targeted goal (i.e. give positive reward for success)
- **future** — replay with k random states which come from the same episode as the transition being replayed and were observed after it.
- **episode** — replay with k random states coming from the same episode as the transition being replayed.
- **random** — replay with k random states encountered so far in the whole training procedure.

k (hyperparameter) - 3 of these strategies have a hyperparameter k which controls the ratio of HER data to data coming from normal experience replay in the replay buffer.

Implementation Details

Pseudocode of HER :



Algorithm 1 Hindsight Experience Replay (HER)

```
Given:
  • an off-policy RL algorithm $\mathbb{A}$,                        ▷ e.g. DQN, DDPG, NAF, SDQN
  • a strategy $\mathbb{S}$ for sampling goals for replay,          ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
  • a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$.   ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$
Initialize $\mathbb{A}$                                              ▷ e.g. initialize neural networks
Initialize replay buffer $R$
for episode = 1, $M$ do
    Sample a goal $g$ and an initial state $s_0$.
    for $t = 0, T - 1$ do
        Sample an action $a_t$ using the behavioral policy from $\mathbb{A}$:
                $a_t \leftarrow \pi_b(s_t||g)$                        ▷ || denotes concatenation
        Execute the action $a_t$ and observe a new state $s_{t+1}$
    end for
    for $t = 0, T - 1$ do
        $r_t := r(s_t, a_t, g)$
        Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in $R$     ▷ standard experience replay
        Sample a set of additional goals for replay $G := \mathbb{S}(\textbf{current episode})$
        for $g' \in G$ do
            $r' := r(s_t, a_t, g')$
            Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in $R$     ▷ HER
        end for
    end for
    for $t = 1, N$ do
        Sample a minibatch $B$ from the replay buffer $R$
        Perform one step of optimization using $\mathbb{A}$ and minibatch $B$
    end for
end for
```

For reference, I plan to take inspiration from these two quality python based implementations.

These are :

1) HER Wrapper implementation of **Coach Library** of **Intel Labs** - Link
2) HER Replay Buffer implementation of **stable-baselines3** library of **DLR-RM** which provides pytorch implementation of SOTA algorithms in RL. - Link

I have chosen these implementations as they completely cover the original algorithm (and its flavors) and are in line with our current design choices and existing codebase.

Because of their resemblance to codebase and with few modifications, I would be easily able to add HER to the codebase.

Proposed Class Constructor and Methods

Constructor to instance HER Class :

```
/**
   * Construct an instance of hindsight experience replay class.
   *
   * @param batchSize Number of examples returned at each sample.
   * @param capacity Total memory size in terms of number of
examples.
   * @param herRatio ratio of HER data to data coming from normal
experience replay in the replay buffer.
   * @param nSteps Number of steps to look at in the future.
   * @param dimension The dimension of an encoded state.
   * @param strategy goal selection strategy for following instance
   */
HindsightExperienceReplay(const size_t batchSize,
                    const size_t capacity,
                    const size_t herRatio = 4,
                    goalStrategy strategy = goalStrategy::FUTURE
                    const size_t nSteps = 1,
                    const size_t dimension = StateType::dimension) :
      batchSize(0),
      capacity(0),
      position(0),
      full(false),
      herRatio(herRatio),
      goalSelectionStrategy(FUTURE)
      states(dimension, capacity),
      actions(capacity),
      rewards(capacity),
      goals(dimension, capacity*(herRatio+1)),
      nextStates(dimension, capacity),
      isTerminal(capacity)
      nSteps(nSteps)
  { /* Nothing to do here. */ }
```

Class Enum for GoalSelectionStrategy:

```
enum goalStrategy{
      FINAL,
      FUTURE,
      RANDOM,
      EPISODE,
}
```

Method Description

Following methods would be required in the implementation of HER :

In the Store() function, the following need to be added:

➔ This function would store explored transition with the initial given goal

➔ Calls **StoreStrategicGoal(),** which would store experiences with custom goal sampled

Arguments: state, action, nextState, reward, isEnd, strategy, herRatio

Return : void

In the StoreStrategicGoal() function, the following need to be added**:**

➔ Store experience/transition in an episode with custom goals sample using the goalSelectionStrategy and herRatio

Return : void

In the Sample() function, the following need to be added**:**

➔ This function would sample some experiences with the given goal.

Arguments : state, action, nextState, reward, goal, isTerminal

Return : void

In the GetNStepInfo() function, the following need to be added**:**

➔ Get the reward, next state and terminal boolean for nth step

Arguments : reward, goal, isEnd and Discount

Return : void

In the Update() function, the following need to be added**:**

➔ Update the gradients

Arguments : state, action, nextState, reward, goal, isEnd

Return : void

Proposed changes in the existing codebase

➔ **Addition of the file** *methods/reinforcement_learning/replay/her.hpp*
  - ◆ This would contain the definition and implementation for the HindsightExperienceReplay class

➔ **Addition of tests in** *tests/q_learning_test.cpp***:**
  - ◆ Test for DQN+HER on CartPole Task , TD3+HER Pendulum Task and SAC+HER Pendulum Task

➔ **Adding new environments in** *methods/reinforcement_learning/environment***:**
  - ◆ New continuous and discrete action space environments could be added for the testing of HER

➔ **Editing replayMethod in** *methods/reinforcement_learning/sac_impl.hpp and methods/reinforcement_learning/dqn_impl.hpp* **:**
  - ◆ Editing call and use of methods of replayMethod for example like here

```
replayMethod.Sample(sampledStates, sampledActions, sampledRewards,
     sampledNextStates, isTerminal);
```

to accommodate methods in Hindsight Experience Replay

Testing and Benchmarking :

Describes how I will test and benchmark my project

Since I also plan to add TD3 this summer, I endorse a thorough testing and benchmarking of HER with DQN, TD3 and possibly SAC (If time permits).

I plan to follow two-tier based approach for testing :

1. **Testing using Open AI's Gym TCP-API** : I would like to train and test it using a gym TCP-API (link) project, which will allow us to communicate with the Open AI's gym environment via an IP address. But this would only allow us to train and test the agent on our local systems.

2. **Testing using Implemented Environments in Mlpack** : For the test suitcase, we need a faster and easier approach. So we could reuse already implemented environments in mlpack for testing. A good number of environments both continuous and discrete are already implemented for smooth testing.

Continuous Environments :
In the paper, it is mentioned that HER+ DDPG improved performance on the Robotics related tasks. Most of these robotics tasks simulate continuous sparse binary reward situations therefore we can efficiently evaluate HER performance.

- **For testing and training through Open AI's gym environment**, we could choose robotic tasks like **FetchPickAndPlace-v1 and FetchPush-v1 or BipedalWalker-v2 and LunarLander** etc;

(I would love to discuss further specific for testing using gym environment with mentor)

- **For Testing using Implemented Environments in Mlpack:**
  Currently, Mlpack has the following environments with continuous action space:
  ➔ MountainCar Continuous
  ➔ DoublePoleCart Continuous
  ➔ Pendulum
  We could use the following to test HER in Continuous environments.

I will get the appropriate thresholds for rewards for each environment, after running the same environment and comparing results.

Discrete Environments :
In the paper, it is mentioned that HER+ DQN significantly improved performance on bit flipping environments. DQN without HER can only solve the task for $n \leq 13$ while DQN with HER easily solves the task for n up to 50.

- **For Testing using Implemented Environments in Mlpack:**
  Currently, Mlpack has the following environments with continuous action space:
  ➔ CartPole
  ➔ DoublePoleCart
  ➔ MountainCar
  ➔ Acrobot

We could use the following to test HER in discrete environments .We would just need to add higher thresholds for DQN +HER.
And as for the exact Reward threshold, I would individually check and accordingly set reward thresholds.

On the basis of further needs and requirements of the project, I would love to add more environments with prior discussion with mentor and community.

Part 2 : Implementation of Proximal Policy Optimization Algorithm (PPO)

Abstract

PPO-clip updates policies via

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a \sim \pi_{\theta_k}}{\mathrm{E}} \left[L(s,a,\theta_k,\theta) \right],$$

typically taking multiple steps of (usually minibatch) SGD to maximize the objective. Here L is given by :

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s,a)) \right),$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

When

Advantage is positive: Suppose the advantage for that state-action pair is positive, in which case its contribution to the objective reduces to -

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon) \right) A^{\pi_{\theta_k}}(s,a).$$

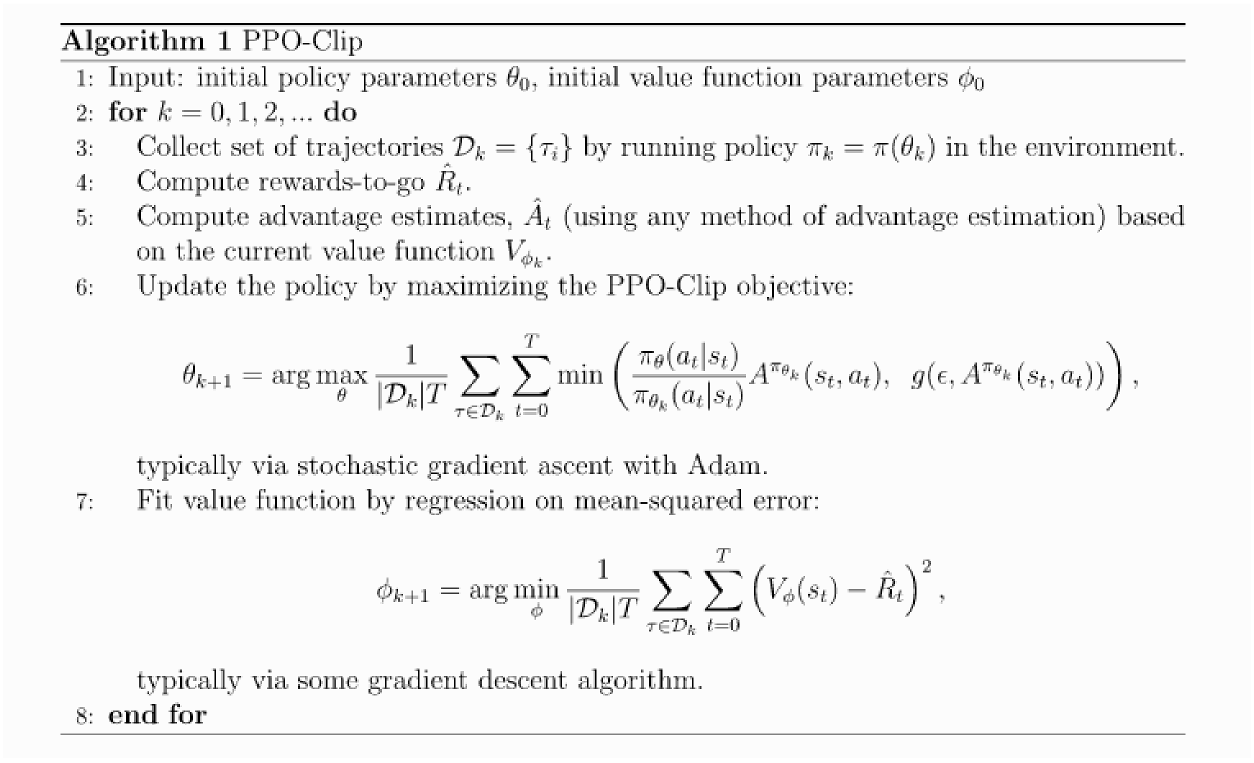
Advantage is negative: Suppose the advantage for that state-action pair is negative, in which case its contribution to the objective reduces to -

$$L(s,a,\theta_k,\theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon) \right) A^{\pi_{\theta_k}}(s,a).$$

Plan of Action

Pseudocode:

For the implementation of PPO, I plan to follow the following pseudocode which has been used in Open AI's baseline - LINK



```
Algorithm 1 PPO-Clip
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, \ldots$ do
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
       on the current value function $V_{\phi_k}$.
6:     Update the policy by maximizing the PPO-Clip objective:
         $$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$
       typically via stochastic gradient ascent with Adam.
7:     Fit value function by regression on mean-squared error:
         $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$
       typically via some gradient descent algorithm.
8: end for
```

Reference for Implementation : Various pull request like #1912 and #2788 have attempted to add PPO in the past to mlpack codebase but they were unfortunately unsuccessful. Nevertheless, they are an excellent source for reference for constructors and initialization of classes and writing methods and their structure.
Link of Summary from Previous Attempt to add PPO.

Some other standard implementations that I plan to take reference include :
1) https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail - Pytorch
2) Stable Baselines (PPO2 from https://github.com/hill-a/stable-baselines)
3) Spinning Up - https://github.com/openai/spinningup/blob/master/spinup/algos/pytorch/ppo
4) Stable Baselines 3 https://github.com/DLR-RM/stable-baselines3/tree/master/stable_baselines3/ppo

Proposed Class Constructor and Methods

Proposed Constructor for PPO :

```
template<
  typename EnvironmentType,
  typename ActorNetworkType,
  typename CriticNetworkType,
  typename UpdaterType
>
PPO<
  EnvironmentType,
  ActorNetworkType,
  CriticNetworkType,
  UpdaterType
>::PPO(TrainingConfig& config,
       ActorNetworkType& actor,
       CriticNetworkType& critic,
       UpdaterType updater,
       EnvironmentType environment):
  config(config),
  actorNetwork(actor),
  criticNetwork(critic),
  actorUpdater(std::move(updater)),
  #if ENS_VERSION_MAJOR >= 2
  actorUpdatePolicy(NULL),
  #endif
  criticUpdater(std::move(updater)),
  #if ENS_VERSION_MAJOR >= 2
  criticUpdatePolicy(NULL),
  #endif
  environment(std::move(environment)),
  totalSteps(0),
  deterministic(false)
```

Proposed Methods in PPO Class :

```
/**
    * Selects action from distribution.
    * @return Action for given state.
    */
  void SelectAction();

  /**
    * Execute an episode.
    * @return Return of the episode.
    */
  double Episode();

  /**
   * Update the actor and critic model
   * */
  void Update();
```

Method Description

Following methods would be required in the implementation of PPO :

- **In the Episode() function**, the following need to be added:
  ➔ Get an initial state from the environment
  ➔ Create variable for the counting of total steps and return
  ➔ Till the environment ends, do:
    - ◆ totalReturn += Steps()
    - ◆ If Deterministic, end the loop
    - ◆ Execute Update()

- **In the SelectAction() function**, the following need to be added:
  ➔ Select Action based on current policy and distribution.

- **In the Update() function,** the following need to be added :
    - ◆ This function will contain code for the update of all networks.
    - ◆ In terms of pseudo code:

  ➔ Forward pass the state into the policy network to get the action
  ➔ Sample reward and next state from environment by passing in the state and action
  ➔ Calculate Discounted Rewards and then calculate Advantage
  ➔ Update Policy using Clipped Objective Function via Stochastic Gradient Descent

Proposed changes in training_config.hpp

```
//! Get the discount rate for future reward.
  double Epsilon() const { return epsilon; }
  //! Modify the discount rate for future reward.
  double& Epsilon() { return epsilon; }

/**
   * the discount rate for future reward.
   */
  double epsilon;
```

Proposed changes in the existing codebase

- ➔ **Addition of the file** *methods/reinforcement_learning/ppo.hpp*
  - ◆ This would contain the definition for the PPO class
- ➔ **Addition of the file** *methods/reinforcement_learning/ppo_impl.hpp*
  - ◆ This would contain the implementation for the PPO class
- ➔ **Addition of tests** in *tests/ppo_test.cpp:*
  - ◆ For testing the PPO environment.
- ➔ **Adding hyperparameters for PPO** in *methods/reinforcement_learning/training_config.hpp:*
  - ◆ Addition of hyperparameters like epsilon for PPO
- ➔ **Addition of new environments in** *methods/reinforcement_learning/environment*:

◆ New continuous action space environments could be added for the testing of PPO

Testing

Describes how I will test my project

Currently, Mlpack has the following environments with continuous action space:

➔ MountainCar Continuous
➔ DoublePoleCart Continuous
➔ Pendulum

These could be used for testing out PPO. More importantly as per discussion in #1912, we could evaluate PPO specifically on **DoublePoleCart Continuous** as it is relatively tougher than other environments.

I will get the appropriate thresholds for rewards for each environment with experimentation and comparing results.

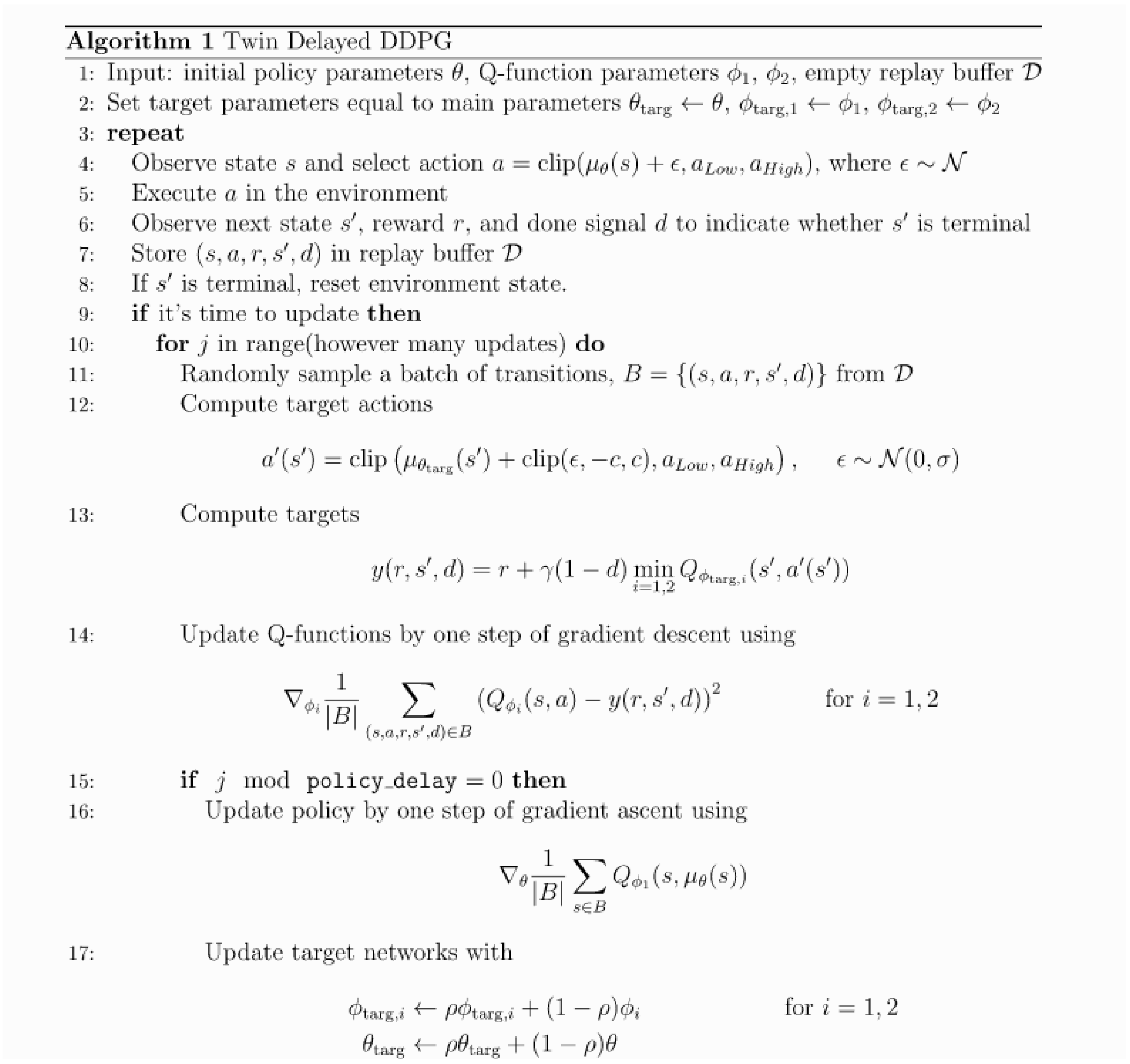
Moreover, there is a need for a more complex and tough environment to appropriately gauge performance of PPO like ***LunarLandarContinuous****. Unfortunately #1912 and #2788 could never get merged and LunarLandarContinuous is not part of mlpack as of now.*

If required and time permits, I would like to add them after discussing its feasibility with my mentor. If given chance to work on it, I aim to pick up from the already done work in ***#1912*** *which made considerable efforts to implement it and build upon it new suggestions and ideas from community and mentor.*

Part 3 : Implementation of Twin Delayed DDPG (TD3) :

Abstract

Pseudocode :



```
Algorithm 1 Twin Delayed DDPG
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3: repeat
4:    Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5:    Execute $a$ in the environment
6:    Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:    Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:    If $s'$ is terminal, reset environment state.
9:    if it's time to update then
10:      for $j$ in range(however many updates) do
11:         Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:         Compute target actions
               $a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$
13:         Compute targets
               $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$
14:         Update Q-functions by one step of gradient descent using
               $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$
15:         if $j$ mod policy_delay $= 0$ then
16:            Update policy by one step of gradient ascent using
                  $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$
17:            Update target networks with
                  $\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$
                  $\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$
```

Plan of Action

Reference Implementation :

Official Implementation **-** https://github.com/sfujim/TD3
Spinning Up -
https://github.com/openai/spinningup/tree/master/spinup/algos/pytorch/td3

Proposed Class Constructor and Methods

Proposed Constructor :

```
template <
  typename EnvironmentType,
  typename QNetworkType,
  typename PolicyNetworkType,
  typename UpdaterType,
  typename ReplayType
>
TDDDPG<
  EnvironmentType,
  QNetworkType,
  PolicyNetworkType,
  UpdaterType,
  ReplayType
>::TDDPG(TrainingConfig& config,
       QNetworkType& learningQ1Network,
       PolicyNetworkType& policyNetwork,
       ReplayType& replayMethod,
       UpdaterType qNetworkUpdater,
       UpdaterType policyNetworkUpdater,
       EnvironmentType environment):
  config(config),
  learningQ1Network(learningQ1Network),
  policyNetwork(policyNetwork),
  replayMethod(replayMethod),
  qNetworkUpdater(std::move(qNetworkUpdater)),
  #if ENS_VERSION_MAJOR >= 2
  qNetworkUpdatePolicy(NULL),
  #endif
  policyNetworkUpdater(std::move(policyNetworkUpdater)),
  #if ENS_VERSION_MAJOR >= 2
  policyNetworkUpdatePolicy(NULL),
  #endif
  environment(std::move(environment)),
  totalSteps(0),
  deterministic(false)
```

Proposed Methods in TDDPG:

```
/**
   * Update the learning Q network.
   * */
  void Update();

/**
   * Perform delay updates on target Q and policy networks.
   * */
  void delayUpdate();


  /**
   * Select an action, given an agent.
   */
  void SelectAction();

  /**
   * Execute an episode.
   * @return Return of the episode.
   */
  double Episode();
```

Variables and Networks Present In Class :

```
private:
//! Locally-stored hyper-parameters.
  TrainingConfig& config;

```

```
//! Locally-stored learning Q1 and Q2 network.
QNetworkType& learningQ1Network;
QNetworkType learningQ2Network;

//! Locally-stored target Q1 and Q2 network.
QNetworkType targetQ1Network;
QNetworkType targetQ2Network;

//! Locally-stored policy network.
PolicyNetworkType& policyNetwork;

//! Locally-stored experience method.
ReplayType& replayMethod;

//! Locally-stored updater.
UpdaterType qNetworkUpdater;
#if ENS_VERSION_MAJOR >= 2
typename UpdaterType::template Policy<arma::mat, arma::mat>*
    qNetworkUpdatePolicy;
#endif

//! Locally-stored updater.
UpdaterType policyNetworkUpdater;
#if ENS_VERSION_MAJOR >= 2
typename UpdaterType::template Policy<arma::mat, arma::mat>*
    policyNetworkUpdatePolicy;
#endif

//! Locally-stored reinforcement learning task.
EnvironmentType environment;

//! Total steps from the beginning of the task.
size_t totalSteps;

//! Locally-stored current state of the agent.
StateType state;

//! Locally-stored action of the agent.
ActionType action;
```

```
//! Locally-stored flag indicating training mode or test mode.
bool deterministic;

//! Locally-stored loss function.
mlpack::ann::MeanSquaredError<> lossFunction;
```

Method Description

- **In the Episode() function**, the following need to be added:
  ➔ Get an initial state from the environment
  ➔ Create variable for the counting of total steps and return
  ➔ Till the environment ends, do:
    - ◆ totalReturn += Steps()
    - ◆ end the loop when it's time update
    - ◆ Execute Update()

- **In the SelectAction() function**, the following need to be added:
  ➔ Select Action based on current policy and distribution.

- **In the Update() function,** the following need to be added :
    - ◆ This function will contain code for the update of q learning network.
    - ◆ In terms of pseudo code:

      - Sample a batch from the replay buffer.
      - Compute target actions **(Target Policy Smoothing correction)**

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

      - Compute targets for Q_learning functions (Q1 and Q2) (**clipped double-Q learning correction**)

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

```
policyNetwork.Predict(sampledNextStates, nextStateActions);
arma::mat targetQInput = arma::join_vert(nextStateActions,
      sampledNextStates);
arma::rowvec Q1, Q2;
targetQ1Network.Predict(targetQInput, Q1);
targetQ2Network.Predict(targetQInput, Q2);
arma::rowvec nextQ = sampledRewards + config.Discount() * ((1 -
isTerminal) % arma::min(Q1, Q2));
```

- Update Q_learning network by one step of gradient descent

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s,a) - y(r,s',d) \right)^2 \qquad \text{for } i = 1, 2$$

- **In the delayUpdate() function,** the following need to be added: **(Delay Update)**
    - This function will contain code for the update of target q network and policy network.
    - In terms of pseudo code:
        - Update policy network by one step of Gradient Ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

        - Update Target Networks

$$\begin{aligned} \phi_{\text{targ},i} &\leftarrow \rho \phi_{\text{targ},i} + (1-\rho)\phi_i \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1-\rho)\theta \end{aligned}$$

Proposed changes in the existing codebase

➔ **Addition of the file** *methods/reinforcement_learning/td3.hpp*
- This would contain the definition for the TD3 class

➔ **Addition of the file** *methods/reinforcement_learning/td3_impl.hpp*
- This would contain the implementation for the TD3 class

➔ **Addition of tests** in *tests/td3_tests.cpp:*

  - ◆ For testing the PPO environment.

➔ **Adding hyperparameters for TD3** in *methods/reinforcement_learning/training_config.hpp:*

  - ◆ Addition of hyperparameters like epsilon for TD3

➔ **Addition of new environments in** *methods/reinforcement_learning/environment*:

◆ New continuous action space environments could be added for the testing of TD3

Testing

Describes how I will test my project

Since, Mlpack has the following environments with continuous action space:

➔ MountainCar Continuous
➔ DoublePoleCart Continuous
➔ Pendulum

These could be used for testing out TD3. I will get the appropriate thresholds for rewards for each environment, after running the same environment in the original paper's implementation(link), and comparing results.

If required and time permits, I would like to add other continuous environments after discussing its feasibility with my mentor.

Part 4 : Documentation / Tutorial / Examples

In our bid to be the fastest and most reliable machine learning library, proper documentation, self-explainable examples and simple tutorials are important components.
I have personally followed a lot of tutorials (including method specific tutorials like K-Means tutorial (link) and Reinforcement Learning Tutorial (link) of mlpack to get started with the library.

I also went through examples repository under mlpack and experimented with various examples like - lunarlander_dqn, bipedal-walker_sac etc;

During my exploration and experimentation, I have gotten myself familiar with running C++ notebooks on jupyter using xeus-cling and have already gone through the jupyter-conda-setup.sh file.

Plan of action

Along with readable implementation and tests, I intend to work on these tutorial and examples for all of my proposed additions and algorithms i.e. PPO, DDPG.

Examples under examples repository :

I have prepared a brief idea for the examples we could create to showcase simple use of our efficient algorithmic implementations.

Design and Implementation :

Examples already present inside the **reinforcement_learning_gym** folder of examples repository provide a very decent idea to create further examples for different learning algorithms.

We could use the same structure of implementation for our examples. With some small changes and using different API calls, we would be able to create future examples.

Proposed API

◆ **Use of proposed PPO API can be done as following for Pendulum Environment**

```
PPO<ContinuousActionEnv,
      decltype(actorNetwork),
      decltype(criticNetwork),
      AdamUpdate>
      agent(config, qNetwork, policyNetwork);
```

◆ **Use of proposed TDDPG API can be done as following for BipedalWalker Environment:**

```
RandomReplay<ContinuousActionEnv> replayMethod(32, 10000);

TDDDPG<ContinuousActionEnv,
      decltype(qNetwork),
      decltype(policyNetwork),
      AdamUpdate>
      agent(config, qNetwork, policyNetwork, replayMethod);
```

Proposed changes in the existing codebase of examples repository

---

➔ **PPO with Pendulum Environment** :
- ◆ **Addition of the file** *pendulum_ppo.cpp and pendulum_ppo.ipynb*
- ◆ These files would contain demonstrations of PPO agents on Pendulum Environment.

➔ **PPO with LunarLander Environment** :
- ◆ **Addition of the file** *lunar-lander_ppo.cpp and lunar-lander_ppo.ipynb*
- ◆ These files would contain demonstrations of PPO agents on the Lunar Lander Environment.

➔ **PPO with BipedalWalker Environment** :
- ◆ **Addition of the file** *bipedal-walker_ppo.cpp and bipedal-walker_ppo.ipynb*
- ◆ These files would contain demonstrations of PPO agents on the Bipedal Walker Environment.

➔ **TD3 with LunarLander Environment** :
- ◆ **Addition of the file** *lunar-lander_td3.cpp and lunar-lander_td3.ipynb*
- ◆ These files would contain demonstrations of TD3 agents on the Lunar Lander Environment.

➔ **TD3 with BipedalWalker Environment** :
- ◆ **Addition of the file** *bipedal-walker_td3.cpp and bipedal-walker_td3.ipynb*
- ◆ These files would contain demonstrations of TD3 agents on the Bipedal Walker Environment.

➔ **DQN+HER with LunarLander Environment** :
- ◆ **Addition of the file** *pendulum_her-dqn.cpp and pendulum_her-dqn.ipynb*
- ◆ These files would contain demonstrations of DQN+HER agents on LunarLander Environment.

➔ **TD3+HER with BipedalWalker Environment** :
- ◆ **Addition of the file** *bipedal-walker_her-td3.cpp and bipedal-walker_her-sac.ipynb*
- ◆ These files would contain demonstrations of TD3 agents on the Lunar Lander Environment.

➔ **SAC+HER with BipedalWalker Environment** :

- ◆ **Addition of the file** *bipedal-walker_her-sac.cpp and bipedal-walker_her-sac.ipynb*
- ◆ These files would contain demonstrations of SAC+HER agents on the Bipedal Walker Environment.

I also noticed that SAC and other components do not have any tutorials at Reinforcement Learning Tutorial (link).

If time permits, I have following sections in my mind :

- **PPO in Mlpack :**
    - Initialize and set up the actor and critic networks.
    - Setup the other components like hyper parameters
    - Declaring PPO agent
    - Training of PPO agent under any simple environment
- **TD3 in Mlpack :**
    - Initialize and set up the actor , critic and q networks.
    - Setup the other components like replay method, policy, hyper parameters
    - Declaring TD3 agent
    - Training of TD3 agent under any simple environment

I would love to discuss more specifics about preparing tutorials with my mentor so that the final content is exhaustive and readily understandable to a first time user of the library.

Expected Timeline with Proposed Deliverables

I don't have any other major commitments during the coding period.
Because of the extended time period of my project, I will have exams around September - October. Exact dates and details will be communicated to my mentor inorder to avoid any deadline or progress issues.
While making the expected timeline, I made sure to keep sufficient buffer periods including a buffer week to deal with various unforeseeable situations.

This makes my timeline more realistic and shock proof to future events.

Community Bonding Period May 20 - June 12

➔ I will discuss project in more detail with the mentor.
➔ I will get more acquainted with the codebase, especially Reinforcement Learning and FFN, along with that I plan on getting to know the armadillo library as it will be used extensively.
➔ I'll try to fix some issues (if any) related to reinforcement learning or mlpack in general. Along with this, I plan to complete all of my ongoing pull requests so that they can be successfully merged.
➔ I will also try and experiment more with the currently proposed idea, with inputs from the other developers will discuss requirement of any new environment to be implemented

Week 1 June 13 - 20

➔ This period would be spent on writing a skeleton layout for the PPO that i will implement
➔ I would also spend more time on reading related research papers to get a more robust view of the problem, along with articles from Spinning Up (OpenAI) and various other accepted courses like UC Berkeley RL Bootcamp.

Weeks 2 & 3 June 21 - July 4

➔ Once the skeleton is ready, I'll finish writing functions inside PPO so that it could be completed.
➔ I also plan to implement LunarLanderContinunous or other decided continuous environments after discussion with mentor so that PPO's performance can be assessed properly.
➔ Along with this, I plan to read reference implementations of PPO to improve code structure and quality.

Week 4 & 5 July 5 - July 18

➔ I'll dedicate these 2 week to fix emerged issues and try to get PPO merged.
➔ Buffer time allotted for completing implementation of new environments if any.
➔ Along with this I will plan to write tests for components inside PPO and tests about its performance on environments in general.

Week 6 & 7 **July 19 - July 31**

➔ I plan to finish PPO in totality by the end of these two weeks therefore, buffer time has been allocated to deal with unforeseeable issues and difficulties with PPO.

➔ By end of this time period, I'll make sure to complete documentation PPO, describing each function/method in detail.

Week 8 & 9 **Aug 1 - August 15**

➔ These two weeks will be spent working on TD3. I will try to complete the basic skeleton for the implementation of class.

➔ Along with this, I plan to go through research papers and referenced implementation of TD3 again in order to rule out any design inconsistencies and ensure correctness of future implementation.

➔ I'll also test the code along the way to check if it converges for relatively easy environments like Pendulum

Week 10 & 11 **Aug 16 - Sept 31**

➔ During these two weeks, I will work on completing all the necessary functions and methods of TD3

➔ I'll provide a simple tutorial with detailed documentation for PPO demonstrating its use case and applicability, if time remains.

Week 12-13 **Sept 1 - Sept 14**

➔ I will work extensively writing tests for TD3.

➔ Along with that, I will make efforts to resolve issues encountered with implementation of TD3

➔ I will try to get PR for TD3 which will include all tests and implementation merge.

➔ I'll provide a simple tutorial with detailed documentation for TD3 demonstrating its use case and applicability, if time remains.

Week 14 - (Buffer Week) **Sept 15 - Sept 22**

➔ This week is dedicated to act as buffer. My mid-term exams will happen around the same time so this week could absorb that time. This will ensure that deliverables remain achievable and the proposed timeline or project does not experience any shocks.

Week 15 **Sept 23 - Sept 29**

➔ I will try to start with HER and complete basic layout for it

➔ Along with that, I will try to read reference implementations and research papers to understand HER. This will help in extensive testing and benchmarking of HER's performance

➔ Along with that, this time period is dedicated as to wrap up TD3, If time permit.

Week 16-17 **Sept 29 - Oct 12**

➔ Once the skeleton is ready, I'll finish writing functions like Sample(), Update() and Store() etc inside HER so that it could be completed.

➔ After completing HER, I will update and edit call and use of methods of replayMethod to accommodate HER in implementations like SAC, DQN and TD3.

Week 18-19 **Oct 12 - Nov 4**

➔ I'll get HER merged as a separate PR by writing tests, and getting issues fixed.

➔ Create examples of different off-policy algorithms with and without HER to understand effectiveness and improved performance under various environments.

➔ Detailed Documentation of HER will be completed.

Week 19-20 **Nov 5 - Nov 21**

➔ I would wrap up my implementations by completing all pending work (if any) from before, including implementation, tests and documentation.

➔ I would try to get all PRs merged, after thorough review from mentors and fellow developers.

➔ This time will be spent writing report for my work done under GSoC period which would include all the intricate details.

➔ Finally, I will discuss with the mentor regarding future improvements.

Personal Details

I am Eshaan Agarwal, pursuing graduation in Electrical Engineering at Indian Institute of Technology (BHU), Varanasi, India.

I had my first hands-on experience with coding 5 years back when I first heard about MIT APP Inventor. From dragging and dropping code pieces for fun, to complex machine learning algorithms, I have come a long way. During this journey, I have grown to appreciate self learning and curiosity. This curiosity has led me to mind-boggling things like machine learning, edge computing, cryptography and finally open source. Following my brief introduction with the open source community in Hacktoberfest I have started appreciating the vibrance and efforts of open source and its community.

Besides that my current interests include : *reinforcement learning, federated learning and cryptography (zero knowledge proofs still blows my mind!)* etc;

I have extensively used deep learning frameworks like pytorch and keras with tensorflow for my projects. Creating such a massive library which made machine learning so easy is not a task and I have always wondered what goes into making such easy to use APIs. Since my engagement with mlpack for the last couple months, I have had the opportunity to observe it. Now with GSoC 22, I would like to be a part of this amazing team.

I have used Ubuntu 18.04 and been using Ubuntu 20.04. Generally, I use Sublime Text for small codes and Visual Studio Code for working on development projects.

Technical Proficiency and coding skills

1) What languages do you know? Rate your experience level (1-5: rookie-guru) for each.

➢ Python : 4
➢ C++ : 4
➢ JavaScript : 4
➢ Go : 2
➢ Rust : 1

2) How long have you been coding in those languages?
C++ - 3 years : It is one of the first languages that I started with. I learned most of it from youtube and internet articles. Most of my exposure to C++ has been related to competitive programming due to its rich libraries and incredible speed. I also took a course CS101 which was in C++ which was my formal introduction to programming. I also took a course on data structures and algorithms which was taught in C++ I was surprised but also amazed at the capabilities of C++ when I heard that it is used to make softwares like Photoshop or Gaming Engines. Recently I also took a session for my college students on advanced concepts of C++ (link to youtube video) where I taught them file handling, advanced data structures and OOPs in C++.

Python - 3 years : I was interested in machine learning and that motivated me to learn python. It is an amazing language and provides a fast prototyping platform. I have read a lot of articles and have great command of it as I built numerous projects. I also have developed a decent understanding of the pythonic way of writing codes and class based design in python. Built complex applications like referral, promo-code systems. I also improved the payment architectures of startup Ureify during my internship.

JavaScript - 1 year : I got introduced to javascript when I ventured into the realm of web development. Recently I was shocked when I got to know about tensorflow.js. I would definitely love to implement a small project with it someday.

I have started learning Go and Rust in my free time. I have a lot to learn but I am equally excited because of their amazing capabilities. *Especially when a lot of people are comparing Rust as a potential competitor of C++.*

3) Are you a contributor to other open-source projects?
I started contributing to open source projects through hacktoberfest where I made 4 PRs in different open source organizations. I am also developing our college's programming clubs website with a fellow team of developers.(link)

I also contributed to a federated recommendation system library called “Envisedge” in a couple of issues like this (link). I have been maintainer of the website of our institute’s machine learning group for 1+ years now.(Link of site)

4) Do you have a link to any of your work (i.e. github profile)?
➢ DQN on Google Chrome’s Dino Game: (Code)
➢ Futuristic Car Design Generation Engine : (Code) - Team Project - Came 2nd in competition sponsored by Mercedes.
➢ U-net based ensemble model for Road Semantic Segmentation: (Code)

➢ Implementing Word2Vec word embeddings from scratch : (Link)
➢ Swimming pools detection from Satellite Images using yolo-v5; (link)
➢ Automatic Captcha with Emoji Solver - (link)
➢ SMS Spam Detection App: (Code) - Team Project
➢ Students' Programming Club Website: Code ➢ Code
➢ Other Projects can be found here: https://github.com/eshaanagarwal

5) What areas of machine learning are you familiar with?
For 15 year old kid, to be told that machine can differentiate between cats and dogs, perform tasks just like humans was remarkable on its own, The curiosity of to know its working led me to the world of machine learning, where I explored all my interests:

➢ **Natural Language Processing** : I tried to construct my own **Word2Vec** word embeddings model. This was my introduction to the field of Natural Language Processing. Thereafter I experimented with numerous things like *sentiment analysis and text summarization, bitcoin prediction using tweets* etc.

➢ **Reinforcement Learning** : After hearing about machine learning algorithms that had great performances on Atari Games, I tried to implement an agent which used **DQN** on Google Chrome's Offline Dinosaur Game. This introduced me to concepts on Reinforcement Learning. From there I worked on numerous projects around it. Recently I was working on using '**Model Based Reinforcement Learning Algorithms on Feedback Control System**' under my institute's professor.

➢ I have in depth knowledge about the implementation of some model-free off-policy algorithms like **DQNs, SAC, A2C, REINFORCE, along with PPO and DDPG, HER etc**;

➢ To dive deeper, I started working with **CNNs**, and building projects while also participating in some supervised learning competitions mentioned in my resume. I also worked on *Real Time Road Semantic Segmentation* using **FCN, Unet, MobileVnet2** algorithms.

➢ When I came across **Variational Autoencoders and GANs**, I found it amazing and hence learnt about them, by hacking through their codes and tweaking parameters.
I also used **GANs** to create '*A Car Design Generation Engine*' for a competition sponsored by Mercedes. (link)

➢ Apart from that, I have used **XGBoost** and **CatBoost** and stacking ensemble of various tree based algorithms like **Random Forest** etc for various data science competitions for which we bagged prizes. I have also used **PCA** for dimensionality reduction

6) Have you taken any coursework relevant to machine learning?
➢ In order to understand working behind word embeddings and other NLP foundations, I took '**Stanford'sCS224n: Natural Language Processing with Deep Learning**'
➢To understand basic of CNNS for computer vision,, I completed '**Stanford's CS231N CNNs for Computer Vision**' course.
➢ When i was working on making my DQN Agent, For RL, I started following the **"UCL's course on RL by David Silver."**
➢ I have also taken **Stanford's Lectures on Deep Learning CS230** and **Andrew Ng course on Machine Learning on Coursera**.
➢ As for other parts of my work on machine learning, I rely on Medium blogs, reddit, StackOverflow discussions and Research Papers. I learnt a lot by interacting with the fellow researchers and community out there.
➢ In addition to that, I have studied course on **Probability and Statistics and Multivariate Calculus** at my institute.

Other open-ended questions

1) What are your long-term plans, if you have figured those out yet? Where do you hope to see yourself in 10 years?

➢ I plan on working at the junction of research-oriented machine learning and development. Because I believe that machine learning problems can only be properly tested when applied to real-world problems. So, I hope to see myself in 10 years, working with an eminent research facility or organization, implementing frontier research ideas for real-life use-cases. With the growing demand and data awareness, people now care about use and visibility of their data, federated learning or machine learning on edge devices is gaining traction. I plan on working on privacy preserving federated learning or applications of AI in cyber physical systems. If I am lucky enough, I would love to work in places like UC Berkeley, CMU, ETH Zurich or corporate research arms like FAIR if I get a chance.

➢ I am really enjoying my journey into open source. My experience across organizations has been amazing. I really like the idea of developers across the world bonding over their love to solve problems with occasional fun. Now that I am getting familiar with the workflow, I have no plans on quitting. So, I would like to contribute to open-source projects whenever I get time from my work, for fun, learning, and to have a sense of belonging, as I have been experiencing for quite some days now I think OpenSource development will occupy a long portion of my life

2) Describe the most interesting application of machine learning you can think of, and then describe how you might implement it.

➢ There are countless applications of machine learning almost everywhere around us, which have made our lives easier. But all these are uses of supervised and clustering algorithms, along with a few applications of Reinforcement Learning here and there.

➢ One interesting application I worked on recently was to generate “Futuristic Car Prototype Designs”. More specifically, we created an engine that could generate custom new car designs with few user-specified parameters and design choices. This particularly had applications to car designers all over the world and demonstrates real life application of Machine Learning in the best way possible.

➢ I believe this is achievable with the the following approach. Idea was based on the assumption that upcoming new car designs will be similar to existing car designs in their basic structures so that designs don't lose their feasibility of implementation.

➢ **Proposed Solution :** Formally, we can perform this by training a Style-GAN2 on a car dataset (any car dataset which has images representing modern patterns of car) with differential augmentation and LeCam regularizer. This could be followed by fine-tuning of the trained GAN on a futuristic cars dataset (concept cars photos scraped from google) using FreezeD technique. Further we could use a Res-Net18 model to extract latent codes of common and futuristic cars which can be then combined using weighted average and fed into StyleGAN.

➢ By freezing those early layers and manipulating the last few layers. We will be able to control the rendering style separately and thus generate images of a specific style. To obtain the real latent codes corresponding to our desired car images, we can apply backpropagation on perceptual loss between this inaccurate image (obtained from our model) and desired image with only latent code elements as trainable parameters.

➢ Weighted average gives us the flexibility to decide the intensity of futuristic flavor which can be added to existing designs to obtain novel ones. Thus, generate custom new car designs with few user-specified parameters and design choices.

3) Both algorithm implementation and API design are important parts of mlpack. Which is more difficult? Which is more important? Why?

➢ Although both are important parts of mlpack, For implementation, there are numerous micro-decisions required when implementing a machine learning algorithm and these decisions are often missing from the formal algorithm descriptions.I personally find API design to be more challenging and important at the same time. This is because implementation of an algorithm might be tricky, but for a proper API design, the developer has to consider a lot of things, including the ease of usage and understandability both for fellow/future developers and for end users. Also, the codebase needs to be structured in such a way that it is easy for adding new features in the future.

Communication

I'm flexible with my schedule and have inculcated the habit of working at night, so time zone difference shouldn't be an issue. I'm comfortable with any of the communication mediums I mentioned above.
I can work full-time on weekdays and am usually available between 11 AM IST to 2 AM IST. On weekends, I would love to spend time communicating with the team to learn from them, while working on whatever issues occur at that time.

I'll responsibly keep my mentor updated in case of any emergency that occurs with suitable details.

Contributions to mlpack :

I have been trying to engage with the mlpack community since I participated in my first issue at mlpack in January. Here are my contributions till now. I would love to keep on contributing to mlpack.
Interaction with community on issue and pull request :
Some of them include #3172 and #3119

Open Pull Requests:

➔ **In mlpack repository :**
  - **PR #3164** -
    Name - Added Size checks for Matrix Completion, Kmeans and Linear

Description - Adding size checks and Editing size check functionalities for wider scope -
Status - Approved (need to be merged)

- **PR #3186** -
  Name - Fix Naive Bayes Classifier
  Description - Bugfix related to Classify() of Naive Bayes. Also going to add test for check the same
  Status - Ongoing

➔ **In examples repository**

- **PR #192 -**
  Name - Switched to Internal split for LSTM examples
  Description - Using Data::Split instead of manually splitting in lstm example
  Status - Approved (need to be merged)

Post GSoC / Future Work

If there are things left unimplemented, I'll try to complete them post GSoC and will keep contributing to Mlpack, by adding other algorithms, which are yet to be implemented like: Neural Episodic Control (NEC), Intrinsic Curiosity Module (ICM), ACER, Random Network Distillation (RND), ACKTR, A2C etc.

I think Graph Neural Networks is also an interesting direction. Mlpack would have a great addition with implementation of GNNs. I would love to be part of the team of contributors who would add GNNs to mlpack.