# REVERSE CURRICULUM REINFORCEMENT LEARNING IN A BALLOONING ENVIRONMENT

**Martin Zuber**
martin_zuber@bluewin.ch

June 15, 2023

## ABSTRACT

Curriculum reinforcement learning has been successfully applied to many different tasks in the past. Especially in goal-oriented problems with sparse rewards, curriculum learning helps to overcome high exploration costs or in some cases enables learning at all. This paper presents a reinforcement learning environment which can serve as a basis for the implementation of different goal-oriented ballooning tasks. Furthermore, it demonstrates that a non-trivial ballooning task with sparse rewards can be solved by applying curriculum learning approaches. Two different approaches are evaluated. First, a fixed curriculum schedule, where tasks are defined ahead of training and presented to the agent in ascending difficulty level. Second, an adaptive curriculum schedule is implemented. With the adaptive curriculum schedule new tasks are generated during training and their difficulty level is adjusted based on the current skills level of the agent. This paper concludes, that an adaptive schedule can lead to curriculums with an optimized number of tasks without losing the ability to solve the final problem.

***Keywords*** Curriculum Learning · Reinforcement Learning · Adaptive Curriculum

## 1 Introduction

Reinforcement learning (RL) has produced groundbreaking results in the past. Famous examples include agents learning to play Atari video games [Mnih et al., 2015], beating humans in chess, shogi and Go [Silver et al., 2018] or developing strategies by manipulating objects in a multi-agent environment for hide-and-seek [Baker et al., 2019]. A promising and highly researched approach to train agents in such challenging environments is curriculum learning (CL). In CL the agent is presented with a sequence of tasks ordered by ascending difficulty level [Narvekar et al., 2020]. An agent learns easy tasks first and transfers gained knowledge to more difficult tasks, which the agent would be unable to solve by learning from scratch. Such curriculum based RL approaches have recently been used to learn complex video games [Hao et al., 2020, Song et al., 2021, Lin et al., 2023] or robotics tasks [Karpathy and van de Panne, 2012, Xie et al., 2020, Zhang et al., 2023].

CL can also help to tackle goal-oriented tasks with sparse rewards [Florensa et al., 2017]. In such environments, an agent only receives a reward upon reaching a target without getting any intermediate rewards. Sparse rewards make it difficult or even impossible for the agent to receive a strong reward signal during exploration. One technique to deal with sparse rewards is called reward shaping. With reward shaping, the reward function is altered to produce meaningful intermediate rewards (e.g. rewards based on distance). Shaping reward functions is often not trivial and requires expert domain knowledge and time consuming experimenting [Graesser and Keng, 2019]. Furthermore, it can lead to unpredicted and unwanted behavior or restricts the agent to find creative solutions [Riedmiller et al., 2018]. As this work will show, CL is a technique which can solve environments with sparse rewards without using reward shaping.

Looking at a real-world example, the problem of sparse rewards can be found in hot air ballooning competitions. The simplest task in such ballooning competitions is Judge Declared Goal (JDG). In JDG, a pilot has to fly his balloon from a starting area to a goal previously defined by the judge or competition organizers. To finish the task, the pilot has to drop a marker and the distance to the goal determines his ranking compared to the other competitors. The pilot does not receive any intermediate rewards during flight and the time needed to complete the task is not relevant, although time limits may apply [24th FAI World Hot Air Balloon Championship, 2022].

In this work, a goal-oriented environment is designed and implemented to form a basis for ballooning competition tasks such as JDG. Furthermore, CL approaches are implemented to solve a non-trivial JDG task. It is not in the scope of this work to implement an environment with all the detailed physics of ballooning, but to show if an agent can learn the main patterns needed to successfully solve a JDG task with sparse rewards using CL.

## 2 Related Work

The term curriculum learning has first been introduced by Bengio et al. [2009] in the context of supervised learning. An excellent overview of CL in general is given in Soviany et al. [2021]. The survey shows applications not only in the domain of RL but also in the domain of supervised learning. A recent overview of CL solely dedicated to the domain of RL is given by Narvekar et al. [2020].

Already before CL was introduced as a term, the method "Learning from Easy Missions" (LEM) was published by Asada et al. [1996]. Their method already resembled the idea of creating a curriculum to solve RL problems. Their work applies LEM on a vision-based problem in which a robot learns to shoot a ball into a goal with the help of a camera mounted on the robot. The camera image is the only source of information for the robot from the environment. Substates (or subtasks) are manually defined for the dimensions goal size, goal position and ball size. In total, 319 different substates are created and ordered by difficulty level (e.g. a larger goal size is considered easier, since the robot is closer to the goal). A reward of +1 is given when the ball goes into the goal, otherwise the reward is 0. With LEM, the robot is first trained in easy situations and later in ever more difficult scenarios. Asada et al. [1996] conducted experiments with LEM and Q-Learning in a computer simulation as well as with a real-world robot. LEM accelerated training in both setups.

Florensa et al. [2017] presented a more generalized approach for goal-oriented problems by iteratively altering the start state distributions from which the agent learns to reach a goal. Initially, a start state close to the goal is selected from which the agent is likely to reach the goal and to receive a reward signal. Once this initial task is mastered, a set of new promising start states farther away from the goal is selected iteratively until the desired final start state distribution is reached. Since the start states are moving ever farther away, they call their approach "Reverse Curriculum Generation".

The selection of promising new start states is done in three steps. First, for each mastered start state a subset of states is uniformly sampled. In a second step, a number of random actions is taken backwards from each of these start states. Every state reached during this "Brownian Motion" reverse expansion is a possible new start state. In a third and last step, the expected return for each of these possible new start states is estimated based on the current policy. If the expected return is within a certain predefined range of $R_{min}$ and $R_{max}$, the start is considered a "good" start. These selected "good" starts are used to train the agent in the next iteration. Using such a selection based on expected returns should prevent selecting start states which are "too easy" or "too hard" for the current agent to learn.

Their experiments in classical maze environments and with robotic manipulation tasks show the positive effects of "Reverse Curriculum Generation" by modifying the start state distributions in each iteration. Not only does conducting a "Brownian Motion" have a positive impact on the learning performance, but also filtering of new states based on estimated returns accelerates training significantly.

LEM and "Reverse Curriculum Generation" specifically tackle problems of sparse rewards in goal-oriented problems. Both approaches work by modifying the start state distributions from easy to more difficult problems measured as distance to the goal. They also do not change the environment dynamics, action space or the reward function when making the problems more difficult. An obvious difference is, that LEM relies on a domain expert who manually defines the subtasks, whereas the approach of Florensa et al. [2017] automatically generates and samples them during training.

## 3 Proximal Policy Optimization (PPO)

In this section, the RL algorithm used to perform the experiments is briefly introduced. RL algorithms can broadly be classified into value-based, policy-based and combined actor-critic algorithms. Value-based algorithms learn a value function (either for a state or a state-action-pair) and their policy can simply be based on selecting the action with the highest value for a certain state. Deep Q-Learning (DQN) and its variants is a popular value-based algorithm introduced by Mnih et al. [2013]. Policy-based algorithms instead learn a policy function directly by maximizing a goal (e.g. the cumulated discounted reward of an agent). The learned policy function is stochastic, meaning that it gives the action probabilities for a given state and actions are sampled from the given distribution [Graesser and Keng, 2019].

Actor-critic algorithms combine the value- and policy-based approaches. The actor is policy-based, but instead of receiving its reinforcement signal directly as rewards from the environment, the critic is providing the actor with such a signal. Usually, the critic uses an advantage function instead of a classical value function for a state-action-pair in order to calculate the reinforcement signal for the actor. The advantage hereby quantifies how much better an action for a given state is relative to all other available actions. The objective in an actor-critic setup using an advantage is defined in Equation 1 (see Graesser and Keng [2019]).

$$J(\theta) = \mathbb{E}_t\big[A_t^{\pi_\theta} \log \pi_\theta(a_t \mid s_t)\big] \tag{1}$$

For the calculation of the advantage this work uses Generalized Advantage Estimation (GAE) introduced by Schulman et al. [2015]. In GAE, the advantage is calculated as defined by Equations 2 and 3.

$$A_{GAE}^{\pi}(s_t, a_t) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{2}$$

$$\delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \tag{3}$$

In order to calculate the advantage $A_{GAE}$, a value-function $V^\pi$ is learned, providing the value for a certain state. The values of the current state and the discounted value of the next state are used together with the current reward $r_t$ to calculate $\delta_t$. The advantage $A_{GAE}$ for a state-action-pair is then calculated as the exponentially weighted average of all future $\delta$ using a discount factor $\gamma$ and $\lambda$. By adjusting $\lambda$, the bias-variance tradeoff inherent in advantage estimations can be controlled [Schulman et al., 2015, Graesser and Keng, 2019].

This work uses the popular and widely used algorithm Proximal Policy Optimization (PPO) with clipping in an actor-critic setup. The algorithm has been introduced by Schulman et al. [2017]. PPO with clipping replaces the original policy-based objective in Equation 1 by a clipped surrogate objective as defined in Equations 4 and 5.

$$J^{CLIP}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)A_t^{\pi_{\theta_{old}}}, \operatorname{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)A_t^{\pi_{\theta_{old}}}\Big)\Big] \tag{4}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \tag{5}$$

The ratio $r_t(\theta)$ quantifies the difference of the new and the old policy. The range within policy updates are tolerated is defined by the hyperparameter $\epsilon$ and the ratio is bound to the interval $[1-\epsilon, 1+\epsilon]$. Furthermore, by taking the minimum of the clipped objective $\operatorname{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)A_t^{\pi_{\theta_{old}}}$ and the unclipped objective $r_t(\theta)A_t^{\pi_{\theta_{old}}}$, a pessimistic lower bound is added [Schulman et al., 2017]. Large policy updates are avoided and the risk of performance collapse is reduced. Thanks to such conservative policy updates, PPO allows to perform multiple updates within a training iteration and to reuse samples. This mechanism is reflected in the PPO Algorithm 1 by the use of minibatches.

The PPO Algorithm 1 implemented as part of this work largely follows the implementation of Graesser and Keng [2019]. The actor (policy function) and critic (value function) are both implemented with neural networks. Two separate networks are used without any weight sharing. Furthermore, no policy entropy regularization and multi-actor parallelization are used. Both techniques were implemented by Graesser and Keng [2019] but have been left out in this work.

## 4 Ballooning Competition Environment

In the next sections, the design and implementation of the ballooning environment are described, with a special focus on the configuration of wind conditions.

### 4.1 General Environment Description

The environment consists of a balloon, a goal and wind measuring stations. It is implemented as a 3D environment with a continuous space from 0 to 1 on each axis. This space forms the competition area. The balloon, the goal and the wind stations can be assigned every value in this continuous 3D space. As an observation, the environment returns the positions of the balloon and the goal, the wind vector components (quantifying wind speed and wind direction) and the coordinates for each wind station as well as the vertical speed with which the balloon ascends and descends.

**Algorithm 1** PPO with clipping and actor-critic

```
1: Set $I$, number of iterations
2: Set $T$, number of timesteps
3: Set $K$, number of epochs
4: Set $M$, minibatch size
5: Set $B$, number of batches ($\frac{T}{M}$)
6: Initialize actor $\theta_A$ and critic $\theta_C$
7: for $iteration$ = 1, 2, . . . , $I$ do
8:     Set $\theta_{A_{old}} = \theta_A$
9:     Collect trajectories with policy $\theta_{A_{old}}$ for $T$ timesteps
10:    Calculate advantages $A_{GAE}$ for each timestep using $\theta_C$
11:    for $epoch$ = 1, 2, . . . , $K$ do
12:        for $batch$ = 1, 2, . . . , $B$ do
13:            Sample minibatch
14:            Calculate clipped policy loss for the minibatch and update $\theta_A$
15:            Calculate value loss for the minibatch and update $\theta_C$
16:        end for
17:    end for
18: end for
```

The agent has only two actions available: (1) ascend the balloon (i.e. using the burner to heat the air in the balloon) or (2) descend the balloon (i.e. not using the burner and therefore cooling down the air in the balloon and descend). For simplicity, ascending and descending is done with the same vertical speed. This vertical speed can be configured, as it is done for the experiments in Section 5.

As a first ballooning competition task, JDG has been implemented. In this task, the agent only receives a reward of +1 upon reaching the goal, otherwise the reward is 0. Averaging such a binary reward over multiple episodes gives the probability of the agent solving the task. A time limit can be set in terms of maximum timesteps for an episode. When the maximum number of timesteps is reached, the episode ends with reward 0. The episode also terminates with reward 0 if the balloon is flying out of bounds. For simplicity, the task is implemented as a single agent task. No ranking of results compared to other competitors is done, in contrast to the real-world task setup. Furthermore, a range around the goal coordinates can be defined to form a landing area. When the agent lands within this area, the goal is reached. The agent doesn't need to perform a separate "Marker drop" action to indicate his final position.

The environment is implemented according to the guidelines and examples for custom OpenAI Gym environments [OpenAI, 2023]. This offers RL practitioners a familiar interface and integration into their training and test loops. In addition, it allows the use of existing OpenAI Gym wrappers such as FlattenObservation or RecordVideo.

## 4.2 Wind Scenario Configuration

Wind conditions can be flexibly configured in the environment. The configuration is done during the initialization but can also be changed during an episode to simulate dynamic wind conditions. For each axis the number of wind stations can be specified and the stations are then evenly distributed on the axis. The wind direction (given in degrees) and wind speed (given in m/s) can be set individually for each wind station, enabling the implementation of complex wind scenarios. The environment expects that wind direction degrees are given in the mathematical wind direction convention, as opposed to the meteorology convention. Wind directions and wind speeds are converted to wind vector components in order to calculate the new position of the balloon in each timestep and to provide the wind information to the agent.

The wind speed and direction affecting the balloon is the one of the nearest wind station to the balloon. No extra interpolation of wind conditions between wind stations is done, since this would only refine but not change the overall characteristics of the tasks to solve and it would furthermore require to choose one of many different interpolation techniques. Reinhardt and Samimi [2018] provide a good overview of different interpolation techniques for wind data used in practice.

The wind scenario configuration used for the experiments is shown in Figure 1. In total, twelve wind stations (blue) are defined in this configuration (four on each of three wind layers). The first wind layer has a constant west wind (left to right), whereas the second and third wind layers have constant east winds (right to left). The starting position of the

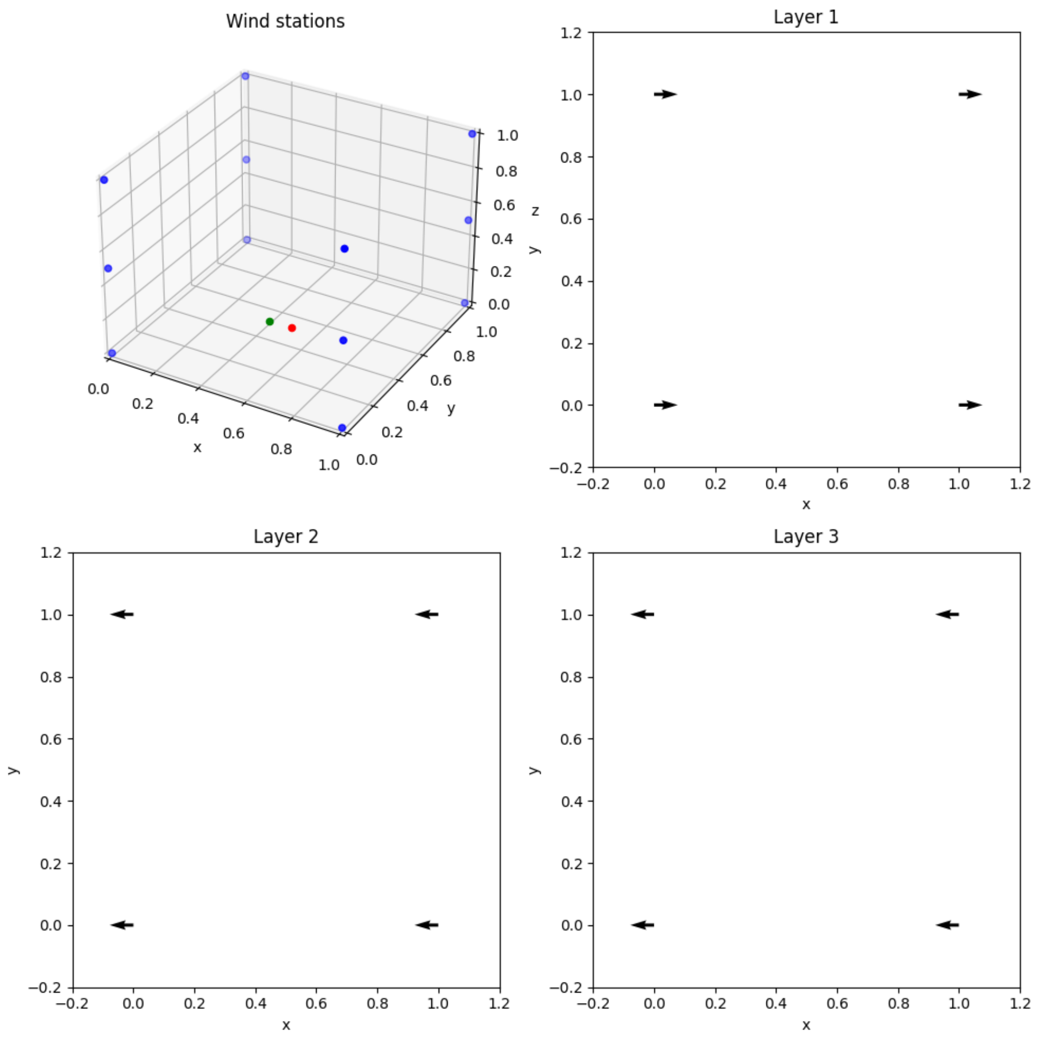


Figure 1: Wind Configuration with multiple Wind Stations and Layers

balloon (red) is placed on the right side of the goal (green). For the experiments, wind conditions are kept static during an episode.

## 5 Experimental Setup and Results

The curriculum design and the main results of the experiments are presented in the following sections. Details about the environment configurations, the training parameters and neural network designs can be found in the code repository `https://github.com/mazub/reversecurriculumlearning`.

### 5.1 Curriculum Design

The task designed for the experiments challenges the agent flying the balloon from its starting position to the goal position under headwind conditions in the first wind layer and tailwind conditions on the upper wind layers (see Figure 1). Therefore, in order to reach the goal the agent must first reach the upper wind layers, since otherwise he would drift ever farther away from the goal. Distance-driven reward shaping is especially hard in such a problem because the agent has to sacrifice his initial closer position to the goal in order to reach more favorable wind conditions. This design was chosen to make the problem non-trivial and to serve as an example for designing curriculum tasks where distance-driven reward shaping is hard to achieve.

Two approaches were used in constructing the curriculum. Building a fixed curriculum before training and an adaptive curriculum generated during training.

### 5.1.1 Fixed Curriculum

This work follows a reverse curriculum approach (see Related Work Section 2). This implies, that an agent starts at a position close to the goal. The distance is then increased step by step to make the problem more difficult until the final starting position is reached. In order to move the agent close to the goal for the proposed task, the vertical ascending/descending speed of the agent will be increased. By using an artificially high vertical speed the number of minimum steps to reach the upper wind layers and to finally reach the goal will be reduced and the agent has a higher chance to receive a positive reward signal. During training, continually decreasing the vertical speed to the predefined final speed on a fixed schedule makes the problem more difficult since the minimum steps to reach the goal are increased. In such a setup, the starting position of the balloon remains always on the ground, but the distance to the goal nevertheless increases. Algorithm 2 details the steps to train an agent using a fixed curriculum.

An important aspect is the reward threshold. This threshold sets the condition by when a task is considered mastered. It also serves as an early stopping condition, since there is no need to continue training on a task which the agent is able solve. The threshold for all experiments was set to 0.75. In case the agent doesn't reach the threshold after the maximum training iterations for a task, a task switch is nevertheless performed. Furthermore, the policy network (actor) and the value network (critic) of the agent are both continuously trained. The final parameters after training on a previous task are directly used as a starting point to continue training on the next task. Since the environment dynamics, observation space, action space and reward function are stable across all tasks, there is no need of a separate transfer learning mechanism (e.g. mapping of action spaces) before continuing training of the agent on new tasks.

**Algorithm 2** Fixed Curriculum

```
1: Generate $curriculum$ with a set of tasks ordered by ascending difficulty level
2: for $task$ in $curriculum$ do
3:     Initialize environment with task configuration
4:     for $iteration$ = 1, 2, . . . , $I$ do
5:         Collect trajectories for $T$ timesteps
6:         Train agent (actor and critic) for $K$ epochs with batch size $M$
7:         Initilize test environment with task configuration
8:         Collect test trajectories for $V$ timesteps with current agent
9:         Calculate average test reward $R$
10:        if $R >= threshold$ then                          ▷ Early stopping condition
11:            Continue with next $task$
12:        end if
13:    end for
14: end for
```

### 5.1.2 Adaptive Curriculum

The steps to train an agent with an adaptive curriculum are outlined in Algorithm 3. In Algorithm 3, the initial state $s_0$ refers to the task configuration from which the agent is highly likely to reach the goal, but not always. The vertical speed for this initial state is set to 0.25 in the experiments. On the other hand, $s_G$ refers to the final task configuration from which the agent ultimately should be able to reach the goal. This final state corresponds to a vertical speed of 0.025 ($\frac{1}{10}$ of the initial vertical speed).

The agent is trained iteratively on ever more difficult task configurations. This is achieved by sampling a set of new possible task configurations between the recently mastered task and the final task (i.e. between the mastered vertical speed and the final vertical speed). From this set, the most promising task (or vertical speed) is selected and used to continue training the agent. The most promising task is defined by testing the current agent on the sampled tasks and selecting the one with the lowest vertical speed, but still lying within an average reward range of $R_{min}$ and $R_{max}$. By applying such a reward range, tasks which are too easy or yet too difficult are filtered out, as done by Florensa et al. [2017]. By using the lowest vertical speed across all tasks lying within the range, training is accelerated the most.

After each training, the agent is tested against the final task configuration. If the agent masters the final task, training is completed, otherwise the next sampling iteration will start. As in the fixed curriculum design, the agent is also continuously trained in the adaptive curriculum approach. There is no need for an additional transfer learning mechanism.

**Algorithm 3** Adaptive Curriculum

```
1: Set start state $s_0$
2: Set goal state $s_G$
3: $s_{new} \leftarrow s_0$
4: Train agent in environment with $s_{new}$    ▷ Perform initial training on start state
5: Test agent in environment with $s_G$ and calculate average $R_{new}$
6: if $R_{new}$<$threshold$ then    ▷ Only start sampling when initial agent can't solve final task
7:     for $sampleiteration$ = 1, 2, . . . , $S$ do
8:         $S_{samples} \leftarrow Uniform(s_{new}, s_G, N)$
9:         for $s_{sample}$ in $S_{samples}$ do
10:            Test agent in environment with $s_{sample}$ and calculate average $R_{sample}$
11:        end for
12:        $s_{new} \leftarrow Select(s_{sample}, R_{sample}, R_{min}, R_{max})$    ▷ Select most promising next task
13:        Train agent in environment with $s_{new}$
14:        Test agent in environment with $s_G$ and calculate average $R_{new}$    ▷ Check if new agent can solve final task
15:        if $R_{new}$>=$threshold$ then
16:            Break
17:        end if
18:    end for
19: end if
```

This adaptive approach has some major differences to the approach of Florensa et al. [2017]. First, no "Brownian Motion" in the action space is performed to determine new start states farther away from the goal. Instead, the reverse expansion is achieved by solely decreasing the vertical speed of the balloon. This increases the minimum steps required to reach the goal and therefore makes the task more difficult. Second, the selection of new states (or tasks) based on the expected return is done by conducting test runs with the current agent on sampled states. Whereas Florensa et al. [2017] use the returns of previous training iterations to estimate the returns. Third, only one newly selected state (task) is used for training in the next iteration, in contrast to Florensa et al. [2017], where a set of new states is used (also including already mastered states).

## 5.2 Experimental Results

The experiments focused on the following three questions:

- What is the effect of different fixed curriculums (task schedules) compared to training from scratch?
- Is training suffering from catastrophic forgetting?
- Does training benefit from an adaptive curriculum design?

All experiments (for fixed curriculum and adaptive curriculum) where conducted with the same training parameters (e.g. batch size, learning rate, maximum iterations, reward threshold). The goal was not to perform parameter tuning to create an optimal agent for each task configuration, but to be able to compare the performance of different task schedules and curriculum designs given the same training parameters.

### 5.2.1 Effect of Fixed Curriculums

Figure 2 shows the three manually constructed fixed task schedules. The fast schedule reduces the vertical speed by dividing it with the task difficulty level $T$. This produces a schedule where the initial vertical speed is halved in task 2 and then reduced in ever smaller steps for the next task difficulty levels $T$ until the final vertical speed of 0.025 is reached. The slow schedule is the opposite of the fast schedule and the linear schedule reduces the vertical speed from 0.25 to 0.025 with a fixed steps size. All three schedules contain ten tasks. Additionally, a two steps task schedule consisting of the initial and final vertical speed was also used in the experiments.

The training progress measured by the moving reward average over 100 training episodes is shown in Figure 3 for the different task schedules as well as training from scratch (i.e. the agent is trained directly with the final vertical speed of 0.025). Ten different training trials are visualized. The different number of timesteps across trials is a result of the early stopping condition, where task level switches occur when the agent masters a certain task level. Task level switches also occur when the agent doesn't master the task, but the maximum number of timesteps per task is reached. For each task schedule, task switches for a representative example trial are visualized by red vertical lines. Since the early stopping

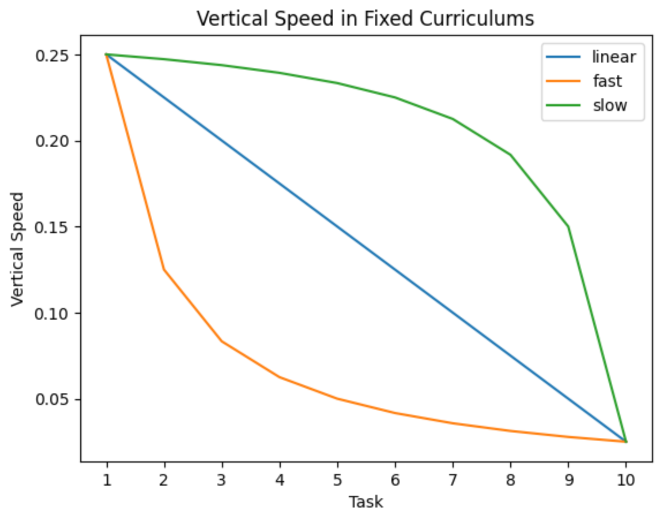


Figure 2: Vertical Speed in Fixed Curriculums

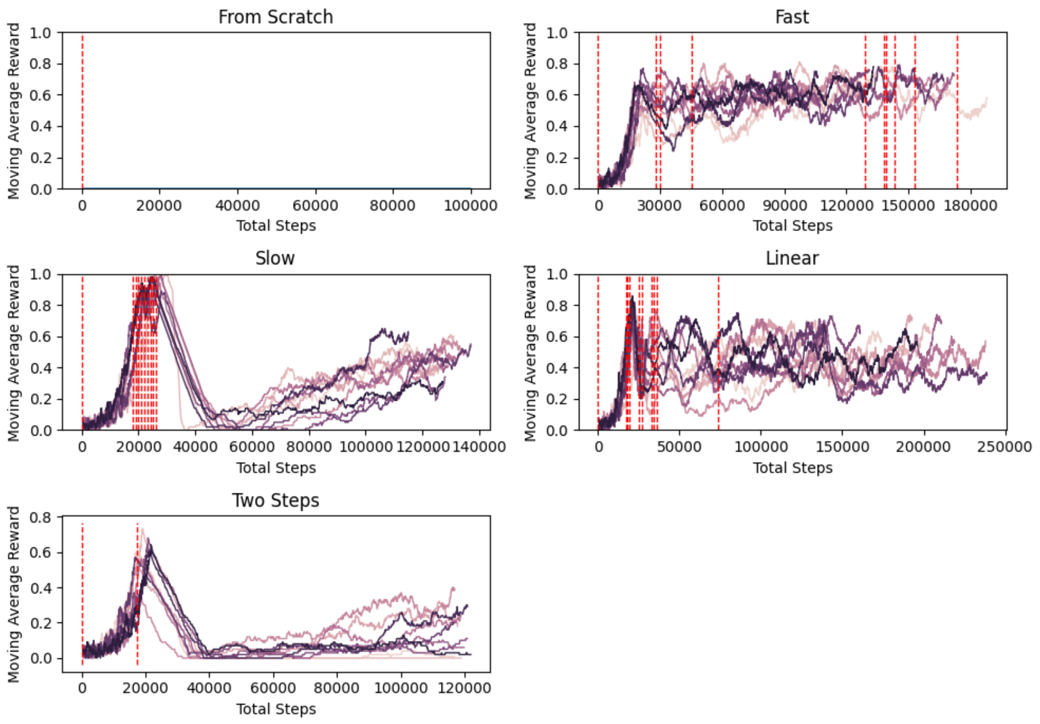
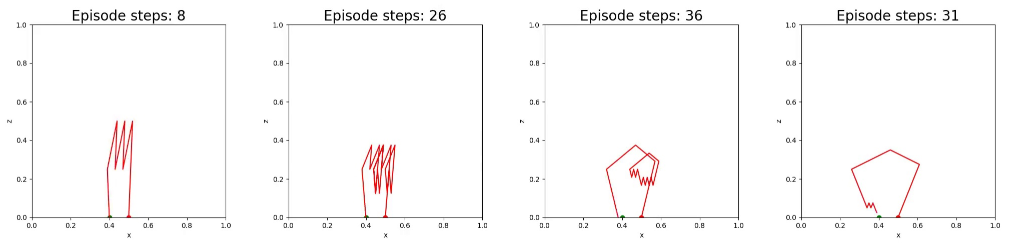


Figure 3: Training Progress with Fixed Curriculums

condition is set by the reward threshold of 0.75, training on a task is not continued to reach a higher average reward. This is the reason why the average reward plateaus around 0.75 (for the fast curriculum).

As expected, training the agent from scratch doesn't show any "signs of life" after the maximum training iterations. The slow and the two steps curriculum show a similar training progress pattern. In both curriculums the average reward collapses after the agent starts training on the final task. This indicates that the switch from the previous to the final task is too hard for the agent. Nevertheless, the intermediate tasks in the slow curriculum still seem to yield some benefits, since the recovery of the average reward is stronger for the slow schedule than for the two steps curriculum.

The fast curriculum performs best. This is shown by the stable average reward which remains close to the reward threshold of 0.75. Task switches also don't always occur close to each other (in contrast to the slow or linear curriculum),



(a) Vertical Speed=0.25 (b) Vertical Speed=0.125 (c) Vertical Speed=0.042 (d) Vertical Speed=0.025

Figure 4: Trajectories for different Tasks

but with some timesteps in between. Together with the stable average reward, this indicates, that the difficulty levels of the tasks are not too easy or too hard for the agent and the schedule allows a smooth and undisrupted learning.

Successful example trajectories for the fast curriculum tasks 1, 2, 6 and 10 are visualized on the XZ-plane in Figure 4 (from left to right). It can be observed how the agent is first pushed to the right away from the goal. After reaching a higher wind layer the agent moves to the left of the goal and is able to perform the final descent in order to successfully finish the task. As seen in trajectories (b) and (c), the balloon doesn't always reach the goal in the first descent attempt. Luckily, the agent learned that the balloon needs to perform another small loop to try another descent to the goal, instead of just descending and missing the goal.

Comparing trajectories (a) and (d) reveals how the movement of the balloon gets refined with the help of a curriculum. In the initial easy task (a), the balloon only needs eight timesteps to finish the task due to the high vertical speed. Within two steps, the balloon reaches the upper wind layer and then starts moving to the left. After six steps, it starts the final descend and reaches the goal within two more steps. In contrast, the final task in (d) is solved in 31 timesteps with a much lower vertical speed. Performing such a trajectory in 31 timesteps by training an agent from scratch is impossible (as seen in Figure 3). The agent never receives a strong reward signal because it is highly unlikely that he reaches the goal during exploration. In that sense, the initial task teaches the main pattern needed to finish the task and facilitates learning. With an appropriate curriculum design, this broad pattern gets refined during training by the presented task sequence until the final task is mastered.

### 5.2.2 Investigating Catastrophic Forgetting

The term "catastrophic forgetting" refers to the phenomenon of an agent forgetting how to solve a simple task after being trained on a more difficult task [Narvekar et al., 2020]. Although we are generally only interested in the performance of our agent on the final task, it is still worth investigating if catastrophic forgetting occurs. In order to do that, a subset of agent versions is tested against a subset of tasks for a corresponding task schedule. Agent version in this context refers to the agent trained on a particular task difficulty level (e.g. agent version 5 was trained on task 5). By testing different agent versions and task combinations, the opposite phenomenon is also investigated, when an agent trained on an easy task can already solve a more difficult task.

The test results averaged for all models of ten different trials are shown in Figure 5. For completeness, the results for a random agent (agent version 0) are also included. Looking at the results of agent version 10 of the fast curriculum schedule, it can be concluded that the agent performs equally well on the final task 10 as well as on the easier task levels 1, 3, 6 and 8. The agent doesn't suffer from catastrophic forgetting. The same conclusion can be drawn for the linear schedule and the slow schedule, although the final agent trained with a slow schedule performs considerably worse throughout all task levels compared to the other curriculum schedules.

For the fast curriculum schedule, also agent versions 6 and 8 are able to gain an average reward of around 0.7 for task level 10. This is because the difference of the vertical speed in tasks 6, 8 and 10 is very small due to the curriculum design. In the other curriculum schedules only the final agent is able to perform better than a random agent on the final task 10. Furthermore, the results for the two steps curriculum reflect the findings already seen in the section for the training progress. The gap between the initial task and the final task in this setup is too large and the performance of the agent trained on the final task 10 is very low.

### 5.2.3 Effect of an Adaptive Curriculum

The experiments with adaptive curriculums where conducted with $R_{min} = 0.2$ and $R_{max} = 0.5$. These values provide a good range of sampling promising task configurations and accelerate training of the agent until the final task can be

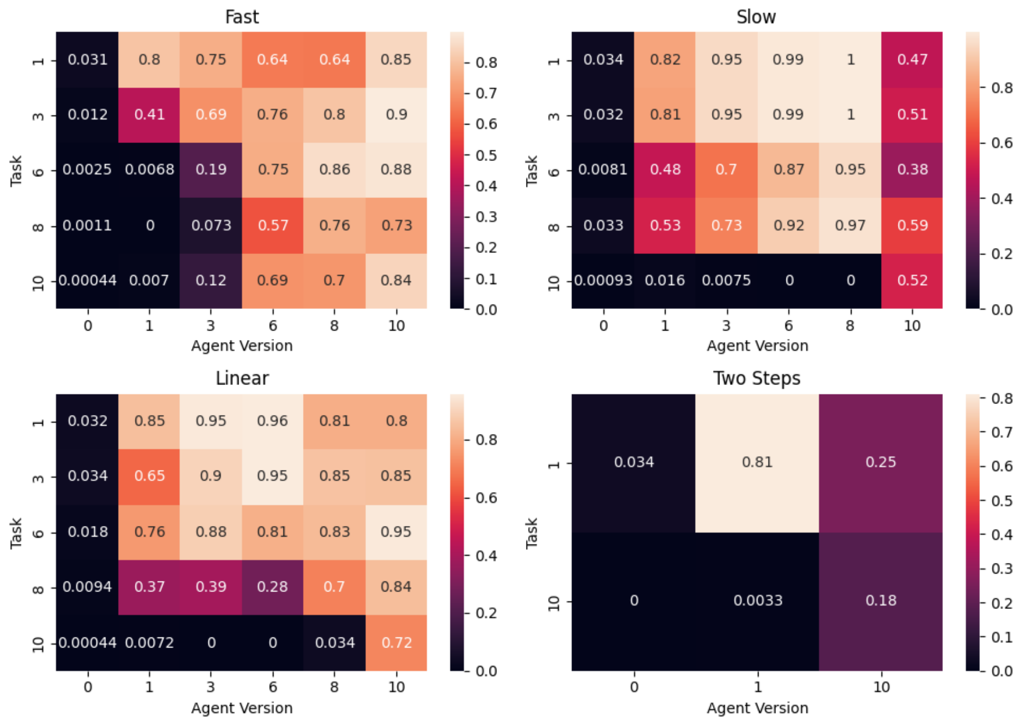


Figure 5: Testing of Task/Agent Combinations (incl. Random Agent)

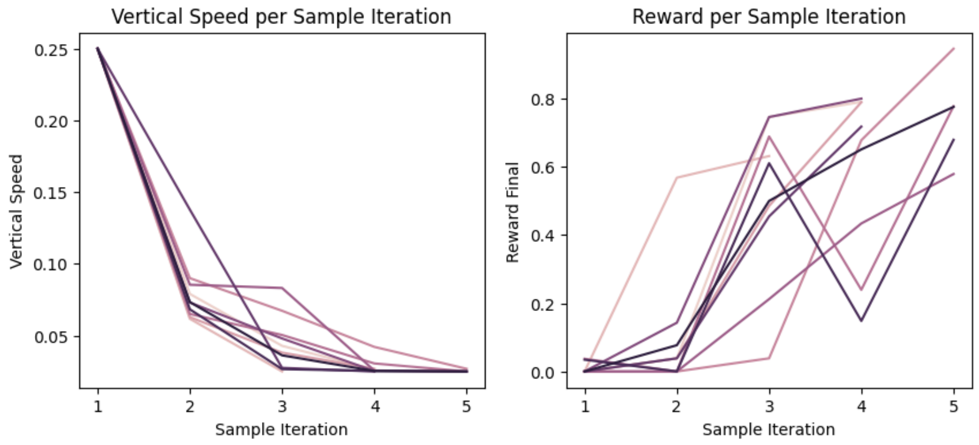


Figure 6: Vertical Speed and Reward in Adaptive Curriculums

solved. Figure 6 shows the results for ten trials. On the left side, the vertical speed per sampling iteration is visualized. The average test reward on the final task after each sampling iteration is shown on the right.

The adaptive curriculum algorithm tends to generate the same curriculum as the fast fixed curriculum, but with fewer tasks. During the second iteration, the initial vertical speed of 0.25 is heavily reduced below 0.1. Subsequently, it is reduced further and approaching the final vertical speed of 0.025 within a maximum of three iterations resulting in a total of maximum five iterations (or tasks). This is half the number of tasks compared to the fixed curriculum setup. As listed in Table 1, the number of tasks across all trials is reduced to an average of 4.4, but the average number of training steps per trial is the second highest of all experiments. This points to a possible conflict in the presented algorithm, where the hyperparameters could either be tuned to reduce the number of tasks or the number of total timesteps within a curriculum. Nevertheless, the adaptive approach reaches an average test reward on the final task of 0.75, which is equal to the defined reward threshold. Although the adaptive approach uses on average more training steps than the fixed curriculums, it avoids the costly process of designing a fixed curriculum suitable for a specific problem.

Table 1: Experiments Overview

| Curriculum | Training Steps | Tasks | Test Reward |
|---|---|---|---|
| Fast | 142’274.4 | 10.0 | 0.84 |
| Slow | 128’851.9 | 10.0 | 0.52 |
| Linear | 197’475.6 | 10.0 | 0.72 |
| Two Steps | 119’039.3 | 2.0 | 0.18 |
| Adaptive | 177’922.9 | 4.4 | 0.75 |

## 6 Conclusion and Future Work

This paper presented an implementation of a 3D ballooning environment. Thanks to its flexible wind scenario configuration capabilities, the environment can serve as foundation for different hot air ballooning competition tasks. As a first task, Judge Declared Goal (JDG) has been implemented. This task is a classical goal-oriented problem with sparse rewards. The experiments in this papers demonstrated, that the task can be solved with curriculum reinforcement learning approaches without using reward shaping. A fixed as well as an adaptive curriculum approach have been applied. Not all fixed curriculum schedules performed equally well, which leads to the conclusion that designing the right schedule for a given problem is time consuming and requires a lot of experimenting. The adaptive curriculum approach offers a better alternative since it finds the optimal schedule by itself during training. Nevertheless, also the adaptive approach can be subject to time consuming hyperparameter tuning and experimenting.

As future work, new and more complex hot air ballooning competition tasks could be implemented. The task difficulty of JDG could also be increased by adding obstacles or prohibited flying zones. This would change the observation space and would required transfer learning mechanism when the agent switches from tasks without obstacles to tasks with obstacles. Furthermore, changing wind conditions during episodes would be an extension worth investigating.

## References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. URL `http://science.sciencemag.org/content/362/6419/1140/tab-pdf`.

Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. 2019. URL `http://arxiv.org/abs/1909.07528`.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. 2020. URL `https://arxiv.org/abs/2003.04960`.

Daniel Hao, Penny Sweetser, and Matthew Aitchison. *Designing Curriculum for Deep Reinforcement Learning in StarCraft II*, pages 243–255. 11 2020. ISBN 978-3-030-64983-8. doi:10.1007/978-3-030-64984-5_19.

Yunlong Song, HaoChih Lin, Elia Kaufmann, Peter Dürr, and Davide Scaramuzza. Autonomous overtaking in gran turismo sport using curriculum reinforcement learning. 2021. URL `https://arxiv.org/abs/2103.14666`.

Fanqi Lin, Shiyu Huang, Tim Pearce, Wenze Chen, and Wei-Wei Tu. Tizero: Mastering multi-agent football with curriculum learning and self-play. 2023. URL `https://arxiv.org/abs/2302.07515`.

Andrej Karpathy and Michiel van de Panne. Curriculum learning for motor skills. In Leila Kosseim and Diana Inkpen, editors, *Advances in Artificial Intelligence*, pages 325–330, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-30353-1.

Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. ALLSTEPS: curriculum-driven learning of stepping stone skills. 2020. URL `https://arxiv.org/abs/2005.04323`.

Yunbo Zhang, Alexander Clegg, Sehoon Ha, Greg Turk, and Yuting Ye. Learning to transfer in-hand manipulations using a greedy shape curriculum. 2023. URL `https://arxiv.org/abs/2303.12726`.

Carlos Florensa, David Held, Markus Wulfmeier, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. 2017. URL `http://arxiv.org/abs/1707.05300`.

Laura Graesser and Wah Loon Keng. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Addison-Wesley, 2019.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom van de Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4344–4353. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/riedmiller18a.html`.

24th FAI World Hot Air Balloon Championship. Official rules whabc 2022, 2022. URL `https://watchmefly.net/assets/uploads/enb/ruleswhabc2022finalversion.pdf`. [Online; accessed 22-April-2023].

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi:10.1145/1553374.1553380. URL `https://doi.org/10.1145/1553374.1553380`.

Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. 2021. URL `https://arxiv.org/abs/2101.10382`.

Minoru Asada, Shoichi Noda, Tawaratsumida Sukoya, and Hosoda Koh. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 1996.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. 2013. URL `http://arxiv.org/abs/1312.5602`.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. 2015. URL `https://arxiv.org/abs/1506.02438`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017. URL `http://arxiv.org/abs/1707.06347`.

OpenAI. Gym library documentation, 2023. URL `https://www.gymlibrary.dev/`. [Online; accessed 24-April-2023].

Katja Reinhardt and Cyrus Samimi. Comparison of different wind data interpolation methods for a region with complex terrain in Central Asia. *Climate Dynamics*, 51(9-10):3635–3652, November 2018. doi:10.1007/s00382-018-4101-y.