# Continuous Curriculum Learning for Reinforcement Learning

**Andrea Bassich**[1] , **Daniel Kudenko**[2] ,
[1]University of York
[2]University of York, JetBrains Research
{ab1770, daniel.kudenko}@york.ac.uk,

## Abstract

Curriculum Learning for Reinforcement Learning has been an active area of research for over two years. Its principle is to train an agent on a defined sequence of source tasks, called Curriculum, to increase the agent's performance and learning speed. This paper proposes to extend the discrete definition of a Curriculum, to a continuous one. The main concept behind this form of Curriculum is to define a continuous function, called "decay function", that specifies how the difficulty of the environment should change over time, and to adjust the environment accordingly. In this paper, we report the effects of utilising different decay functions, as well as introducing a function that automatically creates a Continuous Curriculum based on the agent's performance. Furthermore, we explore the effects that changing the granularity of the Curriculum (how often the difficulty of the task is updated) has on the performance of the agent. Our experimental results show the benefits of using a Continuous Curriculum in both a single agent and a complex multi-agent task.

## 1 Introduction

"Humans need about two decades to be trained as fully functional adults of our society. That training is highly organised, based on an education system and a Curriculum which introduces different concepts at different times, exploiting previously learned concepts to ease the learning of new abstractions."[Bengio *et al.*, 2009]

Machines, on the other hand, require hours rather than years worth of training, but can still benefit from structure and guidance. This guidance can take various forms, one of which is to organise the learning process in a similar way to the human educational system: progressively introducing more concepts and harder examples by exploiting previous knowledge. This method is called "Curriculum Learning", which has been an active field of research in the past few years, especially regarding its application to Reinforcement Learning (RL). Given RL is used to solve increasingly challenging domains, often it is inefficient, or even not possible for the agent to start learning on the full task. For this reason, the Curriculum Learning approach is used, which consists of defining a set of source tasks and training the agent on each of them individually before progressing to learning on the full task.

Unlike previous work in this area, which focuses on defining discrete sub-tasks for the agent to learn, this paper introduces the concept of changing the difficulty of an environment in a continuous way, to create a Continuous Curriculum. This paper therefore analyses the benefits of using a continuous form of Curriculum and sets out the required mathematical framework to make this possible. This is achieved by extending the notion of Curriculum introduced by Narvekar et al. [Narvekar *et al.*, 2016], and by introducing the concept of granularity - the ability to change the difficulty of the environment up to every episode.

The first section of this paper reviews the relevant literature in Curriculum Learning and then discusses the formal aspects of a Continuous Curriculum. The second section deals with the practical aspects of creating a Continuous Curriculum and introduces different kinds of decay functions, one of which automatically creates continuous curricula based on the agent's performance. Following this, is the section presenting the results obtained on both a single and multi-agent environment, using different decay functions and Curriculum granularities. Finally, we identify areas of future research in this specific field.

## 2 Background

This section of the paper discusses Curriculum Learning, as well as briefly outlining some methods for the automation of the creation of a Curriculum.

### 2.1 Curriculum Learning

Bengio et al. first introduced Curriculum Learning in 2009 [Bengio *et al.*, 2009] as a Machine Learning concept to improve the performance of Supervised Learning. The inspiration for this method came from observing the way humans learn, starting with simple concepts and gradually progressing to harder problems.

Assume there is a family of cost functions $C_\lambda(\theta)$ such that $C_0$ can be easily minimised, and $C_1$ is the target function one actually wants to minimise. The Curriculum Learning approach consists of trying to minimise $C_0(\theta)$ first, and then gradually increasing $\lambda$ until it reaches one while keeping $\theta$ in

a local minimum of $C_\lambda(\theta)$. In order to do this, let $z$ be a variable that represents a training example, then $P(z)$ is defined as the probability of the learner being presented with example $z$, and $Q_\lambda(z)$ is the probability of the learner being presented with example $z$ given a certain $C_\lambda(\theta)$. Given a weight matrix $W_\lambda(z)$, the relationship between $P(z)$ and $Q_\lambda(z)$ is defined by the equation:

$$Q_\lambda(z) \propto W_\lambda(z)P(z) \quad \forall z \tag{2.1}$$

Furthermore, if $W_1(z) = 1$, then the equation becomes:

$$Q_1(z) = P(z) \quad \forall z \tag{2.2}$$

A Curriculum is defined as a sequence of probability distributions $Q_\lambda$ that satisfy the following conditions:

$$H(Q_\lambda) < H(Q_{\lambda+\epsilon}) \quad \forall \epsilon > 0 \tag{2.3}$$

$$W_{\lambda+\epsilon}(z) \geq W_\lambda(z) \quad \forall z, \forall \epsilon > 0 \tag{2.4}$$

The first condition imposes that the entropy of the distributions $Q_\lambda$ has to increase the larger $\lambda$ becomes, whereas the second condition states that $W_\lambda(z)$ has to increase monotonically as $\lambda$ increases [Bengio *et al.*, 2009]. Curriculum Learning, as defined here, progressively increases the difficulty of the training by adjusting the probability of examples of different difficulty being shown to the learner.

### 2.2 Curriculum Learning in RL

Curriculum Learning was first applied to RL by Narvekar et al. in 2016 [Narvekar *et al.*, 2016]. As RL is very different from Supervised Learning, a new Curriculum Learning framework was created. This involved generating a sequence of source tasks, called "Curriculum", and, similarly to what happens in Transfer Learning, training the agent on each one of the source tasks to then train on the target task. Curriculum Learning makes two fundamental assumptions on the domain: that different degrees of freedom exist which serve as parameters of the environment; moreover, it is assumed that there is a known ordering over each parameter that corresponds to task complexity.

One of the most challenging problems to tackle when it comes to Curriculum Learning is the automatic creation of a Curriculum. The problem consists in creating an algorithm capable of automatically generating a Curriculum given: a domain, an agent, and a difficulty ordering over the environment's parameters.

To automate the creation of a Curriculum one approach is to formulate this problem as an MDP, like in Narvekar et al. [Narvekar *et al.*, 2017]. Once the Curriculum generation is defined as an MDP, it is then possible to use an RL technique to learn a solution. In this specific case, the learning algorithm used is a modified version of Monte Carlo Approximation. This method is the first attempt at autonomous Curriculum creation, and as such has its limitations. One of the steps of this algorithm involves learning all the candidate sub-tasks and adding the one that modifies the agent's policy the most to the Curriculum. Which has the potential to be very time consuming especially in large environments where there is a large number of source tasks, or it may take several time-steps for an agent to learn even a single sub-task.

Another approach to the automatic creation of a Curriculum is to represent a Curriculum using a Directed Acyclic Graph (DAG) and use a heuristic to try to determine the transfer potential between different source tasks [Svetlik *et al.*, 2017]. This implies, that instead of learning each task to choose whether or not it was a candidate to be added to the Curriculum, it is sufficient to use a heuristic function to evaluate its transfer potential. This method lays the foundations to have a further layer of automation in the creation of a Curriculum, by using Object-Oriented MDPs (OOMDPs), as in Da Silva et al. [Da Silva and Costa, 2018].

OOMDPs introduce the concept of class, to add a layer of abstraction to the state representation. As in Object-Oriented Programming, a class has different attributes, each one with a different domain. Therefore, a state can be represented as a set of instances of different classes, or as the value of the attributes of each instance of the classes. This makes it possible to automate the creation of source tasks with minimal human supervision. Once the target task is represented using objects, the algorithm should randomly select objects from the target task to build a set of source tasks. It is important to note that this method does not guarantee the creation of solvable examples. As such, human supervision becomes vital, as one can either modify the algorithm to guarantee the solvability of source tasks, or review the set of source tasks and eliminate the ones that cannot be solved. One flaw of both methods above ([Svetlik *et al.*, 2017], [Da Silva and Costa, 2018]) is that they rely on a heuristic to try to determine which source tasks should be added to the Curriculum. However, finding a better way to estimate how adding a source task to a Curriculum would affect the performance of the agent is very challenging, as it is not feasible to learn all of the candidate tasks. Consequently, the application of a new form of Curriculum Learning was designed to avoid these issues, and streamline the creation of a curriculum with minimal human supervision.

## 3 Continuous Curriculum Learning

This chapter first discusses the existing mathematical framework for Curriculum Learning defined in Narvekar et al.[Narvekar *et al.*, 2016] and sets out the necessary framework for a Continuous Curriculum. Later in this section, we introduce decay functions, which specify how the difficulty of the environment should change over time.

### 3.1 Framework Definition

As outlined in Narvekar et al. [Narvekar *et al.*, 2016] the task to solve is represented as a domain $D$, that has a number of degrees of freedom, or parameters, that are contained in a vector $F = [F_0, ..., F_n]$. Furthermore, an ordering $O$ is known over each parameter corresponding to parameter difficulty, and there is a generator $\tau : D * F \Rightarrow M$ that takes a domain $D$ and a parameter vector $F$ as inputs, and creates a Markov Decision Process $M_t$. Finally, a Curriculum is defined as the set of MDPs $M = \{M_1, ..., M_t\}$ used to train the agent.

We formally define the ordering $O$ as a function $O : D * F \Rightarrow [0, 1]$ where the arguments are the domain $D$, and the parameter vector $F$, and the output is a real number with a value representing the difficulty of the MDP $\tau(D, F)$.

### 3.2 Continuous Curriculum

The aim of this paper is to extend the notion of a discrete curriculum to a continuous one; this presents its own set of challenges: in fact in order to achieve this we need a way to calculate the parameter vector $F$ at any given time.

Firstly we define a parameter $\delta \in [0, 1]$ called decay factor, whose value will determine the difficulty of the MDP the agent will be training on. The MDP created when $\delta = 0$ will be the target task, and the bigger the value of $\delta$ the easier the MDP becomes. In our approach, the training of the agent will start with the value of $\delta$ set as one, and decay over time until it reaches zero.

More formally, we need to specify how changing $\delta$ affects the feature vector $F$ while satisfying the constraint of monotonically increasing the difficulty as $\delta$ decreases. The difficulty of a feature vector $F$ is assessed by using the ordering $O$. Consequently, we introduce a Mapping function $\Phi : [0, 1] \rightarrow F$ that maps a certain value of $\delta$ to a set of parameters $F$. The mapping $\Phi$ is bound by the following equation:

$$O(D, \Phi(\delta)) \leq O(D, \Phi(\delta - x)) \tag{3.1}$$
$$\forall \delta \in [0, 1], x \in [0, \delta]$$

which specifies that the difficulty of $F = \Phi(\delta)$, according to the ordering $O$, has to increase monotonically as $\delta$ decreases.

The principle behind Continuous Curriculum Learning is to change the value of $\delta$ up to every episode, so for every time $\delta$ changes, a new source task is added to the Curriculum. This means that given a Mapping function $\Phi$, changing $\delta$ defines a new source task:

$$M_t = \tau(D, \Phi(\delta_t)) \tag{3.2}$$

As $\delta_t$ is the only variable in the creation of $M_t$, specifying the way $\delta_t$ changes over time is enough to define the set of source tasks $M = [M_0, ..., M_t]$, which is the agent's Curriculum.

### 3.3 Decay Functions

As the way $\delta_t$ is modified defines the Curriculum of the agent, decay functions are integral to this paper. They are used to describe how $\delta_t$ should be changed over time and can be considered as the "generators" of a Continuous Curriculum.

Let $P$ be a class of functions, called performance functions, that define how the agent is performing at any given moment. This class includes the reward function and other environment specific functions.

Then the value of $\delta_t$ is modified according to a decay function $\Delta$ that takes a set of arguments (that depend on the decay function used) as input.

$$\Delta : (args) \rightarrow \delta_t \tag{3.3}$$

An important constraint on the value of $\delta_t$ is the fact that it has to be monotonically decreasing, causing the difficulty of the environment the agent is trained on to only increase.

**Fixed decay**

The first class of decay functions used in this paper is called "Fixed decay". It includes all the decay functions that do not take into account any information regarding the performance of the agent to derive the value of $\delta_t$.

The most simple example of this class is the Linear decay function. Once the time when the decay should end, $t_e$, is defined, the equation of this decay function is:

$$\Delta_l(t, t_e) = max(1 - \frac{t}{t_e}, 0) \tag{3.4}$$

This function has the upside of being easily implemented, and having only one parameter makes the parameter selection process somewhat trivial. On the other hand, being a linear function means that it lacks flexibility.

For this reason, another function introduced in this research is the Exponential decay function. It was designed to allow for a parameter to indicate when the decay should end, and also have a parameter $s$ which influences the slope of the decay. If $s$ is positive, the smaller the $s$, the steeper the initial part of the decay, if $s$ is negative, the smaller the absolute value of $s$, the shallower the initial part of the decay. When the absolute value of $s$ is large, the decay converges to a Linear decay. The equation for the Exponential decay function is:

$$\Delta_e(t, t_e, s) = max(\frac{\alpha - \beta}{1 - \beta}, 0) \tag{3.5}$$
$$\alpha = e^{-\frac{t}{t_e * s}}, \beta = e^{-\frac{1}{s}}$$

In the equation above, $\alpha$ is the key factor responsible for the decay, starting at 1 when t = 0 and then decaying as t increases. The reason why $\beta$ is subtracted at the numerator is that given a certain number of time-steps $t_e$, the value of $\alpha - \beta$ will be equal to 0 at $t = t_e$ and it will be clipped to 0 as t keeps increasing. Finally the denominator is necessary to keep the constraint that $\delta_0 = 1$, as without the denominator $\delta_0 = 1 - \beta$.

The functions that belong to the Fixed decay class are generally easy to implement into the environment, and as we will see later, are a quick way to improve the performance of the agent. On the other hand, the agent learns differently on every run, and using the same decay function could sometimes hurt the training performance. Addressing this, and seeking a further layer of automation when creating a Continuous Curriculum, were the motivations behind creating the Adaptive decay class.

**Adaptive decay**

The second class of decay functions defined in this paper as "Adaptive decay" includes all the functions that take into account the performance of the agent when choosing the value of $\delta_t$.

A member of this class is the function called "Friction-Based decay", which uses a simple model from physics, a body sliding on a plane with friction between them, to determine the decay. This provides a variable (the speed of the body) that can be used as the decay factor $\delta_t$. The speed of the body is initialised at 1 and can be modified by changing the friction coefficient $\mu$ between the body and the plane.

**Algorithm 1** Continuous Curriculum Learning

**Input**: Decay function $\Delta$, mapping function $\Phi$, performance function $P$

Let $s$ be the current state
Let $M$ be the MDP the agent trains on
Let $t = 0$
**while** agent learning **do**
  Let $\delta_t = \Delta(\delta_{t-1}, t, P(s))$
  **if** change parameters condition **then**
    Let $F = \Phi(\delta_t)$
    Let $M$ be the MDP with parameter vector $F$
  **end if**
  Let $t = t + 1$
**end while**

The value of $\mu$ is determined by analysing the average gradient of the chosen performance function over a set number of time-steps by using a "Derivative Queue". The length of the Derivative Queue influences how sudden the updates to $\delta_t$ are: in fact the longer the queue, the smoother the updates. A positive $\mu$ results in the object slowing down (and $\delta$ decaying) which occurs when the agent's performance is improving. On the other hand, if the performance drops, the friction coefficient $\mu$ will be negative. In this case, if the object is still moving, a force of the magnitude the friction would have had if $\mu$ was positive is applied to speed up the object.

Under normal circumstances, this would result in one of the constraints for decay functions being broken. We stated that $\delta_t \geq \delta_{t+1}$, so if the speed of the object increases, the value of $\delta$ is kept constant until the speed of the object returns back to $\delta$. This results in $\delta$ decaying when the agent's performance is improving, and staying stationary when the agent's performance drops or returns to the value in which it was previously locked. This particular decay function also has $\delta$ locked at 1 until the Derivative Queue is full, which prevents the derivative from quickly shifting at the start, potentially causing an abrupt increase in difficulty in the initial steps of the training.

This decay function has two parameters: the length of the Derivative Queue and the object's weight. As mentioned above, the former influences the smoothness of the decay; the latter, on the other hand, enables the decay function to adapt to different environments. Its effect is similar to normalising the values of the performance function, as changing the weight of the object influences the extent the decay function is affected by a change in the gradient of the reward function. If the reward of a test run is recorded, this parameter can be derived automatically by simulating the decay and setting a certain number of time-steps where the decay needs to end.

### 3.4 Curriculum Granularity

Granularity is the key feature that distinguishes a Continuous Curriculum from a discrete one. It is defined as the frequency with which the difficulty of an environment is changed during the Curriculum. The more granular the Curriculum, the more frequently the difficulty is changed. Our experimental results show that using a higher granularity results in a higher performance.

However, there are some environments where it is not practical to train with the highest available granularity: all the environments where changing the parameters is time consuming. An example of this is one of our test domains, Half Field Offense; in this domain it is necessary to restart the game server to change the parameters, which can only be done at the end of an episode and takes about two seconds. As the number of episodes can be anywhere from 25 to 40 thousand when training for 3 million time-steps, changing the difficulty after each episode is unfeasible. For this reason in the HFO domain the difficulty is updated every 2000 time-steps.

## 4 Experimental Setting

This section describes the two domains used to test the benefits of a Continuous Curriculum. The first is the Predator-Prey environment implemented on a grid world, while the second is Half Field Offence, a sub-task in RoboCup simulated soccer.

The learning algorithm used in all the experiments was Proximal Policy Optimisation, as implemented in the package "OpenAI Baselines" [Dhariwal *et al.*, 2017]

### 4.1 Predator-Prey Environment

The first test domain used in this paper is the Predator-Prey environment. In this domain, the agent is in a ten by ten grid world, and is tasked with avoiding a predator while eating prey to prevent starvation. The state space is comprised of the relative coordinates of the predator and nearest prey, as well as the distance to each closest wall North, East, South and West. The agent starts with 100 health points and loses one health point per time-step. Eating prey restores 10 health points, and being eaten by the predator or starving results in the episode terminating. The reward function used is the difference between the agent's health at the start and end of the episode, but being eaten by the predator always results in a reward of -100. With an optimal policy it is possible to survive indefinitely in this environment, so the duration of each episode was capped at 1000 time-steps. The way the difficulty was changed in this environment was by modifying the number of food sources available at any given time-step. In a ten by ten grid, the hardest difficulty had one food source, whilst the easiest had fifty.

In this environment, the performance function used was the survival time of the agent rather than the reward function. This is motivated by the fact that in this environment the average collected reward is not a good indication of the quality of the agent's policy.

To better explain this we can show how two similar policies, $A$ and $B$, would be misevaluated by using the cumulative reward as a metric. Let policy $A$ always result in the agent surviving 99 time-steps without gathering food, but getting eaten just before starving; and let policy $B$ always result in the agent surviving 100 time-steps without gathering food and starving as a result. Qualitatively, the two policies are almost equivalent: in fact the only difference between the two is that when following policy $B$, an agent would evade the predator one extra time-step. However, the average reward

policy $A$ gathers during an episode is -2.01, whereas policy $B$ gathers an average reward of -1. Therefore if we use the reward as a metric we would erroneously conclude that policy $B$ is significantly better than policy $A$, but as we stated above, the two policies are qualitatively almost equivalent.

Therefore the metric used in the plots for this environment is the average survival time. Conversely, the metric used in the tables is the integral of the curve representing the average survival time, which gives an indication of the performance of the agent throughout the whole training process. To better compare the performance of various decay functions, the results were obtained through testing the agent on the hardest version of the environment after every batch.

### 4.2 Half Field Offense

The second test domain for our Continuous Curriculum is a classic multi-agent problem: Half Field Offense [Hausknecht *et al.*, 2016]. In this domain, the agents control players belonging to the attacking team on a football pitch, with the objective to score a goal against the defending team. The playing area is restricted to half of the pitch, and the number of players on each team is set to two.

The state space for this environment is the "High level" state space described in [Hausknecht *et al.*, 2016], which is composed of 24 features. The action space, for the tests we conducted, was a subset of the "High-level action set" [Hausknecht *et al.*, 2016], where the actions were restricted to "Move", "Pass", "Shoot", "Go_to_ball". Originally commands for movement were divided into "Move" and "Dribble", however in our experiments, if the agent chooses the action "Move" and has the ball, they will automatically "Dribble". If the agent tries to "Shoot" or "Pass" when they do not have the ball, they will not act for that time-step. The reward function awards the agent a utility of 5 for scoring a goal, and -1 when the ball goes out of bounds, is captured by the defence, or the episode times out. Finally, a reward of -0.005 is awarded at every time-step, to encourage the agent to score as quickly as possible.

The way the difficulty was changed in this environment was by adjusting the distance at which the ball spawns from the goal and whether an agent is initialised on the ball or it has to move to the ball before passing or shooting. The performance function used for the Friction-Based decay on this environment is the reward function with its minimum clipped at -0.005. This was chosen to prevent an agent being perceived as improving if they never shoot, as they would experience fewer negative rewards rather than always losing the ball to the defence or shooting the ball out of bounds.

Note that this version of Half Field Offense differs from the one used in Narvekar et al. [Narvekar *et al.*, 2016]. Here both the agent with the ball and the one without the ball are learning, and have a greater state and action space, which makes the problem more challenging.

The metric used in the plots for this environment is the average scoring probability, while the metric used in the tables is the maximum scoring probability throughout the training. In HFO, the performance in the first 1,000,000 time-steps (before the red line in the plots) should be considered as only an indication when comparing different decay methods. This is because the reported performance was obtained during the training, so the difficulty of the environment at a certain time will vary from method to method.

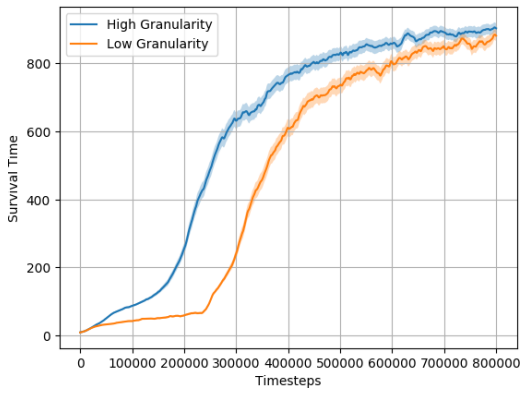


Figure 1: Effect of granularity in Predator-Prey

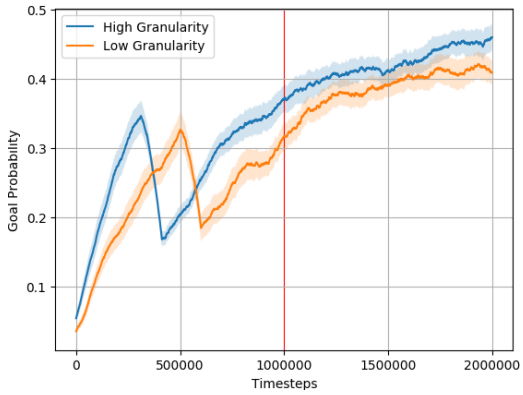


Figure 2: Effect of granularity in HFO

## 5 Results

In this results section we show the performance of the decay functions mentioned above, as well as answer two questions:

1. Does using a Continuous Curriculum improve the agent's performance over a discrete one?
2. What are the benefits of using Friction-Based decay over a Fixed decay function?

### 5.1 Granularity

To answer the first question, we analyse the effects of changing the granularity of the Curriculum and therefore assert why a Continuous Curriculum is needed. Experiments were conducted on both of the test domains outlined above, comparing two curricula generated by the same decay function, but with different granularities. In both test domains, the less granular Curriculum is equivalent to a Discrete Curriculum with five sub-tasks; while the more granular Curriculum is equivalent to a Continuous Curriculum.

The first environment granularity was tested on was the Predator-Prey environment. For this test, the highest performing Curriculum on that environment, Friction-Based decay, was tested in two versions. The version with the high

granularity updates the difficulty at the end of every episode; whereas the version with the lower granularity updates the difficulty five times during the decay. As you can see in Figure 1 and Table 1 the Curriculum with a high granularity clearly outperforms the other in all stages of the training.

For further comparison when testing the effect of changing the granularity of the Curriculum on HFO, the decay function used was the Exponential decay. This is because the effects of changing granularity on the Friction-based decay have already been outlined above, and were consistent with the results in the Predator-Prey environment. As evident in Figure 2 and Table 1, our results show that the agents training with the higher Curriculum granularity again outperformed the ones training with a lower granularity.

Our experiments on both domains demonstrated a more granular Curriculum resulted in a higher survival time in the Predator-Prey environment, and a higher chance of scoring a goal in HFO. Therefore answering the first question posed earlier: it is indeed beneficial to use a Continuous Curriculum over a Discrete one.

| Granularity | Pred. Prey | 2v2 HFO |
|---|---|---|
| High | $(4.774 \pm 0.064) * 10^8$ | $46.0 \pm 1.8$ |
| Low | $(3.797 \pm 0.058) * 10^8$ | $42.0 \pm 1.9$ |

Table 1: The effects of changing granularity

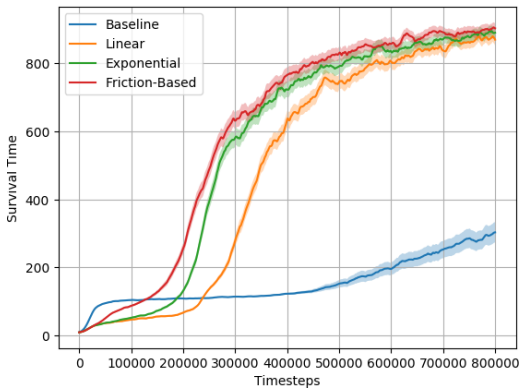


Figure 3: Different Curricula in Predator-Prey

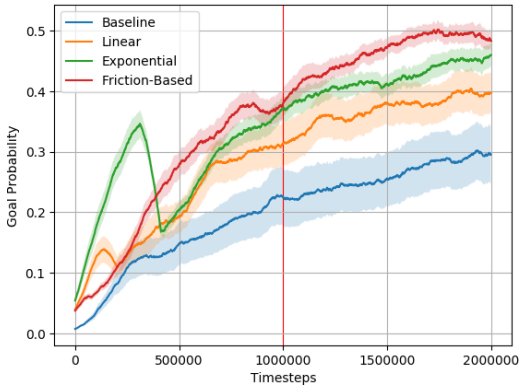


Figure 4: Different Curricula in HFO

## 5.2 Different Continuous Curricula

This section will present the results that each decay function introduced in this paper achieved when tested on the two test environments.

In the Predator-Prey environment, the Adaptive decay clearly performed best overall, followed by the Exponential decay, the Linear decay, and finally, standard RL used as a baseline. The increase in performance over normal RL is expected since using a Curriculum is known to be beneficial to the learning of the agent. Furthermore, the Exponential decay outperforming the Linear decay was unsurprising, as it can perfectly replicate one; therefore, always generating results at least equal to the Linear decay. However, what is interesting is the Adaptive decay outperforming the Exponential decay, which shows the benefits of having a decay function utilising the performance of the agent.

In Half Field Offense the results were mirrored, with Adaptive decay performing best, followed by Exponential, Linear, and finally the baseline. It is interesting to note that a higher performing decay function not only results in a better performance but also a decrease in the standard error.

It is worth noting that finding the parameters for the Adaptive decay is a less time-consuming process, compared to the Exponential decay. In fact, often analysing a previous test run (even from the baseline) is sufficient to correctly estimate the object's weight given a certain length. On the other hand, the only way to find a good parameter for the Exponential decay is through trial and error. The main benefit of using Friction-Based decay over the other decay functions is therefore a simple parameter selection process and a higher performance.

| Decay | Pred. Prey | 2v2 HFO |
|---|---|---|
| Frcition-Based | $(4.774 \pm 0.064) * 10^8$ | $50.2 \pm 1.4$ |
| Exponential | $(4.470 \pm 0.063) * 10^8$ | $46.0 \pm 1.8$ |
| Linear | $(3.880 \pm 0.058) * 10^8$ | $40.4 \pm 3.5$ |
| None | $(1.226 \pm 0.395) * 10^8$ | $30.2 \pm 4.7$ |

Table 2: Comparison of different Curricula

# 6 Conclusions and Outlooks

In this paper we defined the concept of a Continuous Curriculum and provided its mathematical foundation by extending the existing Curriculum Learning framework [Narvekar *et al.*, 2016]. Furthermore, we discussed how a Continuous Curriculum is created, through different decay functions, and provided a way to create a Curriculum with minimal human supervision through the Friction-Based decay (where only a Mapping function has to be provided). Based on our experiments we also found this type of decay to be the best performing one in both our test domains. In the future we intend to expand the current work by:

1. Automatically deriving the Mapping function $\Phi(\lambda)$
2. Automatically deriving the difficulty ordering $O$.

## References


[Bengio *et al.*, 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[Da Silva and Costa, 2018] Felipe Leno Da Silva and Anna Helena Reali Costa. Object-oriented curriculum generation for reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018.

[Dhariwal *et al.*, 2017] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[Hausknecht *et al.*, 2016] Matthew Hausknecht, Prannoy Mupparaju, and Sandeep Subramanian. Half field offense: An environment for multiagent learning and ad hoc teamwork. 2016.

[Narvekar *et al.*, 2016] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[Narvekar *et al.*, 2017] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous task sequencing for customized curriculum design in reinforcement learning. 2017.

[Svetlik *et al.*, 2017] Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. Automatic curriculum graph generation for reinforcement learning agents. 2017.