Adaptive Curriculum Learning in the Google Research Football Environment

Ritwik Dixit RITWIK@BERKELEY.EDU
UC Berkeley

Avik Jain AVIKJ@BERKELEY.EDU
UC Berkeley

1. Extended Abstract

We studied the value of having access to multiple non-optimal rule-based experts during training in a competitive environment, in the context of the Google Research Football environment, and used results from our curriculum learning experiments to inform how to re-design the algorithm to better fit our problem.

The environment provides rule-based experts with a tune-able difficulty parameter. Since none of these experts were optimal, we chose to investigate training while using these bots as opponents, rather than for imitation learning.

Training against multiple opponents lends itself naturally to curriculum learning. Competitive Reinforcement Learning (RL) agents are often trained with self-play, rather than training against strong opponents from the start because it allows more efficient learning than training against an opponent of fixed difficulty. Using these rule-based bots, we suspected we could achieve this property on our opponent's skill level without actually playing against our own policy through an organized curriculum.

Our first proposed algorithm used a fixed, pre-sorted curriculum based on the natural ordering of tasks, and moved on to the next task once mean reward over a recent window exceeded a low threshold, which would indicate we are now more skilled than that opponent.

Results from these experiments challenged our assumption regarding the difficulty of tasks, as we saw clear reward trends against different difficulties, but without distinct correlation to the numerical difficulty value; while there is a natural ordering on bot difficulty defined for human users, partially based on reaction time, these characteristics do not necessarily translate into a difficulty ordering for RL training.

We then re-framed our problem as having a set of N very similar tasks, without any natural ordering. Due to the problems with an irreversible curriculum, and to mitigate "forgetting" of tasks, we proposed a curriculum learning framework where a new task is sampled for each training update. Hyper-parameters are set to ensure that all tasks always have some non-negligible probability of being sampled for training. Furthermore, this framework uses assumptions more suited to our problem: we assume we have very similar tasks with identical reward scales, and thus can use learning-curve based heuristics to define our curriculum's adaptive task distribution.

We study heuristics based on recent changes in reward for a given task, mean of recent rewards, and variance of recent rewards. We found that compared to a 0-heuristic, which yields a uniform distribu-

tion over tasks, weighting the distribution towards tasks with higher recent mean reward enables significantly faster learning.

1.1. Preliminaries

1.1.1. Curriculum Learning Background

In standard reinforcement learning, the objective is to maximize expected reward for a task, formalized as an MDP. In recent years, the strategy of curriculum learning has been increasingly studied to improve learning on complex tasks, across domains in machine learning. Curriculum learning is based on the idea that it may be easier to learn a complex task by first tackling a simpler task and then applying this knowledge to solve harder tasks; this is natural for humans, and has shown success in recent years in the domain of reinforcement learning, in solving tasks which are not otherwise tractable such as learning to interpret nested Python expressions (2). In curriculum learning, we have a set of N related tasks $T_1, ..., T_N$, and hope to learn to optimize our reward on one or more of them by collecting experience on all tasks through a curriculum.

Bengio et. al (1) showed that a curriculum that gradually trained on tasks of increasing difficulty would be more efficient than a randomized one as long as there was a reliable way of quantifying task difficulty. Zaremba Sutskever showed that when training LSTMs to predict the output of short Python programs, a curriculum was necessary. Difficulty of tasks was quantified as a function of 2 parameters of input code: length and depth, depth defined by cumulative sum of depth of nested function definitions. One interesting result they found was that a mixed strategy which, in addition to linearly scaling up complexity of training examples, occasionally randomly included easier tasks, outperformed deterministic scaling because it mitigated the problem of forgetting easy skills. This result suggests that a curriculum based on a probabilistic distribution over tasks may lend itself well to training as it generalizes well over many curriculum difficulties.

1.1.2. Environment

We use our algorithms in the Google Research Football environment. The environment is based on the Football Engine, an advanced simulator which comes complete with all aspects of the game, including goals, fouls, corner kicks, penalty kicks, and offsides. The length of the game is measured as the number of frames, where a real-time simulation would run at 10 frames/second. It supports running games with variable numbers of players per team, and running in multiplayer settings or against a rule-based bot.

Environment Details The state of the environment is defined by all relevant information about the game at a given frame. The state encodes information such as the locations and velocities of players and the ball, penalty counts, score, yellow cards, activate player, and other physics-based attributes. We use a provided grid-based state representation, enabling use of convolutional networks over the field of play. The discrete set of actions includes standard 8-directional moving (sticky), shooting, crossing (long upward kicks), sprinting, slide tackling, and dribbling. We frame this as a single-agent RL problem, where a rule-based opponents policy is part of the environment dynamics.

Rule-based opponent The Engine comes predefined with rule-based bots that were created in the original GameplayFootball simulator. Upon environment creation the opposing teams rule-based bots are given a difficulty parameter θ, to be configured smoothly between 0 (easiest) and 1 (hardest). The difficulty parameter determines the bots reaction time and quality of decision making; note that these are difficulties designed for human play, and we simply begin with the assumption that this natural ordering is valuable.

We define our set of tasks according to discrete values of θ from 0 to 1. We chose $N = 10$ because it was a tractable number of tasks that were far apart enough in difficulty to present unique challenges. More precisely, we define tasks T_j as simulator environments with difficulty $\theta_j = 0.1j$ respectively, for $j \in \{1, 2, ..., N\}$.

Reward Shaping The environment comes packaged with two reward functions to use for training: a sparse reward (Scoring reward) each time a goal is scored, and a shaped reward function which also rewards moving the ball closer to the goal. The shaped reward function yields 0.1 reward for moving the ball 10% of the distance to the goal from the max distance, up to 0.9, resetting progress when a goal is scored. This gives a much less sparse reward signal, and in benchmarks provided alongside the environment, results show that the Checkpoint reward enables simpler policies to train faster. We opted to use the checkpoint reward in our experiments, once we realized that learning would be nearly impossible given our computational resources if we simply used the sparse reward.

2. Method

2.1. Sorted Curriculum with Reward Thresholds

Since we were provided an environment where difficulty was directly encoded by a scalar parameter, we could assume a natural ordering over task difficulties. Considering the success of self-play methods in competitive domains, we wondered whether curriculum learning could provide a similar benefit, since it is hypothesized that self-play works well in-part because we always train against an opponent of similar skill to our agent; since this environment enables us to set difficulty of an opponent, we could simply use an adaptive curriculum which attempts to match opponent difficulty to our difficulty.

We leveraged the zero-sum nature of this competitive task to choose a reward threshold to progress to the next task. Given N tasks with a natural ordering on difficulty, we simply trained on each task in succession. Rather than training for a fixed number of steps, we opted to increase difficulty once a moving average of reward passes a fixed threshold. We refer to this as Iteration I of the algorithm.

We found that irreversibly increasing difficulty after crossing a reward threshold suffered from variance in the reward moving average, causing irreversible increases in difficulty due to random clusters of high reward episodes.

More importantly, our evaluation results did not show any clear correlation between difficulties and rewards, but showed distinctly varied performance across these tasks, challenging our assumption that the scalar human-defined ”difficulty” parameter provided by the environment corresponds to difficulty of learning a policy, or that it corresponds to pre-requisite skills which need to be learned.

2.2. Per-episode task-sampling

We recognized that any curriculum which irreversibly progresses through tasks would be prone to stepping too early or too late, and wondered if this could be mitigated with randomness. Results by Zaremba and Sutskever (2014) suggest that it is important to occasionally train on easy tasks later in training, to avoid forgetting of basic skills and overfitting to harder tasks. Thus, in our second iteration, we opted to use an algorithm which would sample a task to train on every time a new episode is sampled.

Furthermore, evaluation results on various difficulties from the previous method also challenged our assumption that there is a natural ordering on our task set Figure 6. Thus, in redesigning the algorithm, we framed the problem as having N closely related tasks with identical API's, all of which we are trying to optimize simultaneously, with differences in skills required to accumulate reward and timelines for learning, but no consistent natural ordering on difficulty.

In the following algorithms, the training data batch for any given training update is always accumulated by running a task sampled for that update, from a softmax distribution, with hyperparameters set to ensure the distribution always has a non-negligible likelihood of sampling any given task.

This distribution is computed based on a hand-designed heuristic, inspired by an insight in Bengio's paper (2009), which suggests curriculum learning could be beneficial by training on tasks which are not too easy, but not too hard, so we are always training on "interesting" data.

2.2.1. Reward Delta Heuristic

The first heuristic we proposed for whether a task would be "interesting" to train on was recent change in reward. Intuitively, if the reward has been decreasing significantly for a given task, we may be forgetting how to accumulate reward this easier task, and if reward has been increasing significantly, our policy may be in a region where collecting data and training is working effectively; however, if the reward has not changed in recent episodes of training, we may benefit from training on other tasks requiring more immediately learnable skills. In our third and final iteration, we formalized this by using the absolute value of difference moving average of reward over an interval of length W in the softmax distribution.

The probability of sampling task d, with a reward difference window size W, smoothing factor k, and buffer of past $W+k$ rewards for this task R_d, is computed as follows:

$$z_d = \frac{1}{k}\sum_{i=0}^{k} R_d(N-k+i) - \frac{1}{k}\sum_{i=0}^{k} R_d(N-W-k+i)$$

$$\sigma(z) = \frac{e^{\alpha z_d}}{\sum_{d=0}^{N} e^{\alpha z_d}}$$

Here, α is an energy coefficient, which should be tuned based on the possible range of heuristic values to ensure there is always non-negligible chance of sampling any task.

2.2.2. Reward Mean/Variance Heuristic

We noticed our previous method had multiple interacting window sizes (difference window and smoothing window) as hyperparameters, and wondered if we could simplify these expressions and capture similar intuition. We realized that to distinguish between changing rewards and stable rewards, we could simply use variance over the window of length W. With this simplification, we noticed we could also study whether training on higher- or lower-scoring tasks is preferred, by also including the mean when computing probabilities. We update our definition of z_d as follows, using a linear combination of mean and variance in the exponent. Here, let $R_d[W:]$ represent the last W entries of R_d.

The probabilities are now computed as follows.

$$\mu_d = \frac{1}{W}\sum_{i=0}^{W}(R_d(N-i)$$

$$\sigma_d^2 = \frac{1}{W-1}\sum_{i=0}^{W}(R_d(N-i) - \mu_d)^2$$

$$z_d = \alpha * \mu_d + \beta * \sigma_d^2$$

$$\sigma(z) = \frac{e^{\alpha z_d}}{\sum_{d=0}^{N} e^{\alpha z_d}}$$

3. Experimental Details

We set up our experiments in the 1v1 single-player game scenario of the Football environment. Although the engine allowed for more complex, multi-agents scenarios such as 5v5 and 11v11 full games, we decided to fix the environment to a tractable one in order to ensure that we could focus on the curriculum itself, rather than weighing computational limitations such as Colab's limitation to 24 hr experiments. Most experiments were run for 200,000 to 1 million timesteps on Google Colab Pro at a rate of approximately 50,000 timesteps per hour, so many of these were full-day or multi-day experiments (including debugging). We trained on a modified version of OpenAI's PPO implementation in this scenario and utilized Google Research Football's custom IMPALA Convolutional Neural Network (CNN) architecture.

3.0.1. Evaluation

Because of the high variability in the algorithms we ran, we created an evaluation metric to ensure consistent results that could be compared. The evaluation setup was a recurring sweep through of difficulties $d\, \forall d \in \{0.1, 0.2, \ldots 0.9\}$ done every 20 episodes. The environment was duplicated and a rollout of 1024 timesteps was conducted. The evaluation strategy could generalize to any set of N tasks as a simple iteration through the tasks.

3.1. Training Details

Training was done in a series of episodes, in which each episode would train the model in 16 environments in parallel. An episode would constitute 1024 timesteps, and after every episode, the rewards from each environment would be added to a replay buffer. In iterations II III of the algorithm, each separate task d had its own replay buffer in order to determine its probability of being sampled next.

4. Results and Discussion

Our results guided our iterations of the algorithm.

4.1. Baseline for Fixed Difficulty

We ran against a fixed difficulty of $theta = 0.6$ for 1 million timesteps initially to collect a baseline for performance. We also ran evaluation against 5 different difficulties uniformly distributed from $0 to 1$.

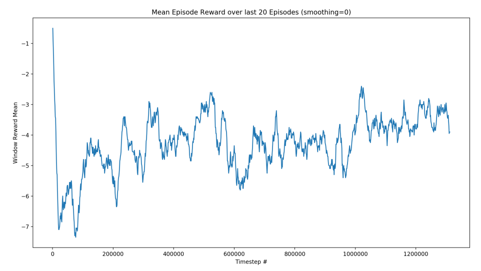

Figure 1: Fixed Difficulty Training Rewards

4.2. Sorted Curriculum with Reward Threshold

The results of the first experiment conducted with a window size $W = 2$ episodes over 1.2 million timesteps. It seems that the agent, once able to progress past the few first difficulties, can quickly learn the greater difficulties Figure 3. However, this difficulty evolution when increasing the window size to $W = 20$ is nonexistent.

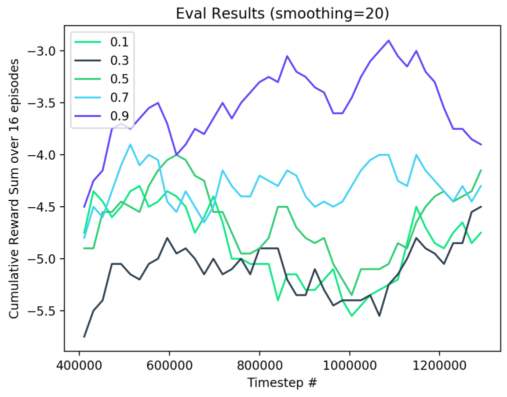

Figure 2: Fixed Difficulty Evaluation Results for various difficulties

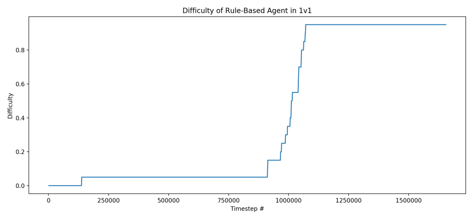

Figure 3: Evolution of Difficulty while Training with Reward Threshold for W=2

Upon closer inspection of the rewards in trial 1 of this iteration, it seems likely that the jumps in difficulty were due to noise. Smoothed over the last 100 episodes there were very few instances where the cumulative mean was greater than zero, which would indicate that the agent can consistently beat the current θ and is ready to progress to the next level.

Although reward was not able to remain consistently positive, it was significantly higher than the baseline, in which evaluation rewards were in the interval $(-5.5, -3.0)$.

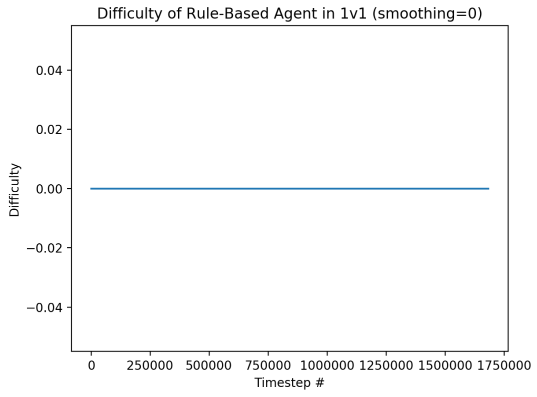

Figure 4: Evolution of Difficulty while Training with Reward Threshold for W=20

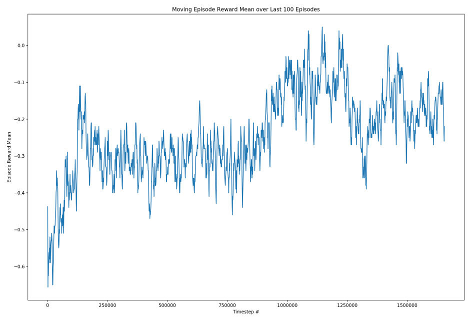

Figure 5: Episode Reward Mean, Smoothed over Last 100 Episodes

4.3. Per-episode task-sampling with Reward Delta-based Probabilities

Results for the Per-episode task-sampling were far more fruitful, perhaps because the variability in the curriculum made the model more robust at learning. By this iteration of the algorithm, other hyperparameters had already been tweaked and optimized for performance. The probabilistic curriculum was

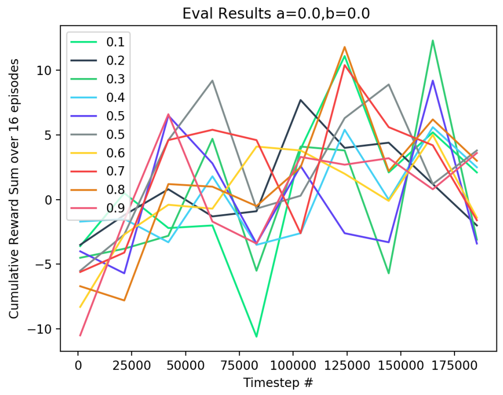

Figure 6: Evaluation Results for Uniform Sampling. These results indicate that there is not a clear ordering in our task set, despite the linearity of difficulty

a step in the right direction as there was a consistent upward trend in reward over all difficulties as indicated by Figure 7.

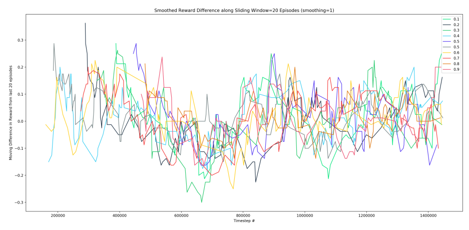

Figure 7: Difference in Reward over W=20 Episode Interval by Difficulty

Tuning the parameter α was paramount to maintaining relative uniformity during training. We found that for values of $\alpha > 0.2$, the distribution would too easily skew toward the more recently successful tasks, which would create a negative cycle in which those tasks would not be sampled for training in subsequent episodes. Ultimately, the probabilities associated with those tasks would tend to 0. Although it was possible to mitigate this phenomenon by tuning the α parameter, as the formulation of z_d was somewhat arbitrary, a more general formulation was preferred.

4.4. Per-episode task-sampling with Reward Mean and Variance-based Probabilities

We experimented with various values for scalar multipliers a and b, for mean and variance, respectively. For the magnitudes of the values, we finalized on $a = 0.1, b = 0.1$ after many experiments as to preserve relative uniformity and ensure that no task had a probability an order of magnitude over another. Many of the results were poor relative to uniform sampling indicated in Figure 6.

The mean values of evaluation results after 200,000 timesteps as indicated by the equation below are shown in table T. These values are summed over 16 environments run in parallel per episode.

$$\mu_d = 1/N \sum_{d=0}^{N} (R_{eval,d}(T = 200,000)$$

Table 1: Sum of Evaluation Reward for values of (a, b) over 16 environments

(a, b)	**Reward**
(Uniform)	0.35
(0, -0.1)	-1.95
(0.1, 0)	20.21
(0.1, -0.1)	0.07
(0, 0.1)	-0.81
(-0.1, 0)	2.73

Notable results included the substantially better performance of $(0.1, 0)$ than the baseline of uniform sampling and almost all other trial runs. The only 2 parameter pairs that performed better than uniform sampling weighted the mean. Additionally, the parameter pairings with the worst performance both gave nonzero weight to sample variance in reward and zero weight to the mean. We conclude that per-episode task sampling with a softmax as a function of recent reward in a given task is the most viable curriculum when the each of the N tasks give similar amount of rewards. We also conclude that recent sample variance in reward for a given task is a poor indicator of an efficient task on which to train.

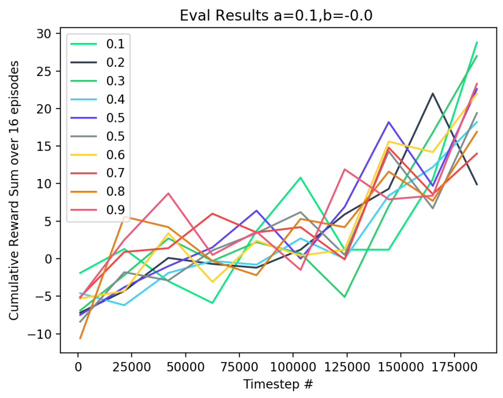

Figure 8: Evaluation Results for (0.1, 0)

As can be seen in Figure 8 there is a consistent upward trend in evaluation reward across all difficulties when this curriculum is employed. Note that the magnitude of the mean evaluation reward is high because each step of evaluation sums rewards over 16 environments run in parallel.

Another interesting result was that the more successful strategies tend toward the ends, the least and most difficult tasks in the pool Figure 9, whereas the less successful curricula seem to form more of a normal distribution around the set of tasks, when ordered by difficulty Figure 10.

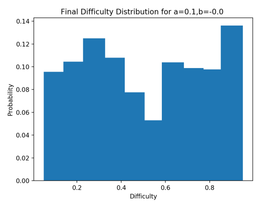

Figure 9: Final Task Difficulty Distribution for most successful pair (0.1, 0)

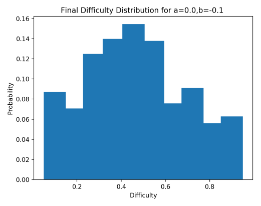

Figure 10: Final Task Difficulty Distribution for least successful pair (0, -0.1)

5. Conclusion and Future Work

Through our experiments, we found that irreversibly progressing through a curriculum can be very susceptible to variance in reward, and requires aggressive smoothing in many scenarios to be useful. We further discovered that accepted, human-defined difficulty values for bots may not necessarily provide a natural ordering on task difficulty for RL agents. This enabled us to define a more general framework for applying adaptive curriculum learning to sets of closely related tasks, with better assumptions for the problem we tackled; we were able to explore heuristics and find the good heuristic of weighting towards high reward tasks, by defining a simple framework for stochastic curricula based on arbitrary data at runtime.

Future directions for this work includes doing a finer search over hyperparameters for our mean/variance heuristic, to get a better sense of whether these are useful. Furthermore, a broader survey of existing curriculum learning and multi-agent research could suggest intuitive ideas which can be easily translated to heuristics to plug into our framework.

6. Citations and Bibliography

(1) Yoshua Bengio, et al. “Curriculum learning.” ICML 2009.

(2) Wojciech Zaremba and Ilya Sutskever. “Learning to execute.” arXiv preprint arXiv:1410.4615 (2014).

(3) Karol Kurach et al. ”Google Research Football: A Novel Reinforcement Learning Environment.” arXiv:1907.11180