Prioritized Experience Replay via Learnability Approximation

Nomi Ringach and Megumi Sano

1. Introduction

Experience replay is a key technique in reinforcement learning that increases sample efficiency by having the agent repeatedly learn on previous experiences stored in a replay buffer (Lin, 1992). Schaul et al. introduced **Prioritized Experience Replay (PER)**, which weights the probability of replaying an experience (a tuple containing state s_{t-1}, action a_{t-1}, reward r_t, discount γ_t, next state s_t) in the buffer with its temporal difference (TD) error (Schaul et al., 2016). While there are clear gains with this now widely used approach, we propose a potential weakness: **the existing prioritization algorithm gives experiences in states with noisy reward values high prioritization values due to high TD error** in these states. We expect that this will lead the agent to train more on experiences in these noisy states despite there being other informative examples to learn on, resulting in a potentially suboptimal data sampling scheme. We believe that a truly helpful prioritization scheme would prioritize experiences based on their **learnability**. In fact, the authors of the paper note in Appendix A that absolute TD error is only one of the many possible proxies for the ideal priority measure of expected learning progress and suggest that the first derivative of the TD error may be a better approximation of learning progress (i.e. the more improvement the agent can make on a certain experience, the more it should focus on learning from that experience). In our project, we will propose a new prioritization algorithm inspired by this idea, which takes as input new experiences the agent encountered and outputs prioritization values used for the sampling probabilities.

2. Related works

Experience replay increases sample efficiency by having the agent repeatedly learn on previous experiences stored in a replay buffer (Lin, 1992). In online learning, the agent must learn how to collect the data it is trained on, as it trains, unlike traditional supervised learning in which the model is fed independent data points. Therefore, a fundamental problem that arises is that the data the agent trains on is not independently and identically distributed. Experience replay (learning on randomly sampled experiences from the buffer) allows decorrelation of data, leading to more stable training. Experience replay is also considered to prevent overfitting as well as catastrophic forgetting by allowing learning on data generated by previous versions of the agent's policy as well as rare and old transitions.

Prioritized Experience Replay (PER) weights the probability of replaying an experience in the buffer with the magnitude of its temporal difference (TD) error (Schaul et al., 2016). Temporal difference error is a proxy for how "surprising" or unexpected the transition is, or more mathematically, the difference between the current value estimate and the TD target: $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$, where R is the reward and V denotes the estimated value (sum of future discounted rewards) of a given state. While the authors use TD error to approximate learnability, they discuss that it is a poor estimate when rewards are noisy, since unlearnable states may have high TD error.

Distributed Prioritized Experience Replay was more recently introduced as an extension of prioritized experience replay, in which there exist multiple actors that interact with individual instances of the environment and push their own experiences into a shared replay buffer. Then, a single learner samples experiences from the buffer (Horgan et al., 2018). Distributing generation of data across multiple workers allows more diverse data, and the use of PER allows the agent to only train on important samples among this diverse set of transitions.

Hindsight Experience Replay (HER) A slightly different method of experience replay motivated by human learning is hindsight experience replay (HER) (Andrychowicz et al., 2017). In HER, the agent has a goal state. When an agent fails to reach the goal state from some initial state, the idea is to pretend that the state the agent ended up in instead had been the goal state from the start and learn from the experience regardless. The trajectory is stored in the buffer along with pretend rewards that the agent would have gotten if that state had been the goal. In this way, independent of the agent's currently learned policy, it is able to learn from positive rewards.

3. Environment and experimental set up

In this section, we describe the environment in which we train our reinforcement learning agent, which is the equivalent of "dataset and features" in online learning. The primary goal of our experiments is to test whether our proposed method of prioritization will enable **lower sample complexity** (i.e. number of environment interactions needed to master a task) than traditional PER in settings where we expect PER to fail (e.g. environments with noisy states or rewards). Therefore, one necessary step before running the experiments is creating such environments. Fortunately, existing environments from OpenAI Gym (Brockman et al., 2016), where RL benchmarking is often performed, are easily modifiable. In particular, we use CartPole-v0 and PongNoFrameskip-v4 from OpenAI Gym. During training, we alternate between a "noisy" and a "non-noisy" environment, where in the "noisy" environment, we control the extent to which some states are noisy by randomizing the reward with a small probability.

One trial consists of 50,000 frames of training. Every time an episode is finished, a new episode is started with the "noisy" environment if the previous environment was "non-noisy", and with "non-noisy" otherwise. We use standard Q-learning with a discount factor of 0.99 and a simple epsilon-greedy exploration strategy. For CartPole, our Q-function architecture is a two-layer fully connected network with 128 units. For Pong, since the input is an image, we use three convolutional layers. Our buffer size is 1,000 to allow for faster computation, although typically buffers can have sizes around 100,000 when training in Gym environments. Each experiment consists of four trials.

For evaluation, at every 200 training frames, we test our agent in a test environment (that is "non-noisy") across 10 episodes and calculate the mean reward. Plotting this across the number of frames of training passed allows us to compare the sample complexity (number of environment interactions required to master a task or more colloquially, *how quickly the RL agent is able to learn in that environment*) between our methods and the baselines. Our codebase for the environment and DQN is based primarily on the higgsfield (2017) repository with a significant portion of our changes focused on the prioritization algorithm and artificial creation of noisy environments. We use PyTorch (Paszke et al., 2017) for the DQN and Scikit-learn (Pedregosa et al., 2011) for clustering method implementation.

4. Methods

We propose several methods with the goal of improving upon traditional PER. The overarching goal of each method is to cluster experiences in the replay buffer (RB), approximate how quickly the RL agent can learn on each cluster (i.e. how much progress can the agent make with the fewest number of environment interactions = **sample complexity**), and then give experiences that are the most learnable the highest priorities. This is in contrast to PER, which assigns the highest priorities to experiences that have high TD error (i.e. the most unexpected). Below, we describe our three distinct learning methods for assigning priority values to experiences in the RB.

Cluster Mean Comparison

This method begins by clustering the experiences in the RB using one of the following clustering methods: k-means clustering (Lloyd, 1982) and mean-shift clustering (Fukunaga and Hostetler, 1975). Both of these clustering methods require the number of clusters as a hyperparameter. As we discuss in the experimental results section, we found the optimal number of clusters for both methods to be 5. Regardless of clustering method, we clustered on the state vector of our environment. For example, in CartPole this would simply be a vector $s \in \mathbb{R}^4$ where $s = [x, \dot{x}, \theta, \dot{\theta}]$. The goal of clustering experiences in the RB is to form classes of similar environment states, which are then used to classify a new experience. Consequently, when a new experience is formed, it is first assigned a cluster based on the method described and then given a priority value equivalent to the sigmoid of the difference of its TD error and the mean TD error of its assigned cluster. If we let m be a vector where m_i represents the mean of the i-th cluster, $TD : \mathbb{R}^4 \to \mathbb{R}$ be a function that gives the TD error of a state, $C : \mathbb{R}^4 \to \mathbb{N}$ be a function that classifies a state to a cluster, and $P : \mathbb{R}^4 \to \mathbb{R}$ be a function that returns the priority (as a probability) of an experience in the RB, then

we can succinctly write this method as:

$$P(s) := \sigma\left(m_{C(s)} - TD(s)\right), \tag{1}$$

for $s \in \mathbb{R}^4$. In this way, environment states that are difficult or impossible to learn will have clusters with high mean TD error values, causing new experiences with similarly high TD error values to be tagged with a low priority.

Continuous Naive Linear Regression on Clusters

For this method, clusters are formed in an identical matter to **Cluster Mean Comparison** and with the same goal of forming classes of environment states. These methods diverge after the clustering step. Imagine chronologically[1] sorting the experiences within each cluster and plotting their individual TD error values against the chronological order (i.e. a TD error v. time graph). Without making any assumptions on the distribution or underlying structure of the experiences, we use linear regression to fit a line to this data or to some pre-set number of most recent experiences. Finally, a new experience is classified into a cluster and receives the sigmoid of the negative slope of the resulting fitted model from linear regression as its priority value. We must use the negative slope because a decreasing TD error indicates that we are improving in the cluster and that it is learnable, meaning that we want to give our new experience a higher priority. This method therefore prioritizes sampling from environment states that are both not yet mastered and possible to learn. We can use similar notation as in Eq. (1), with δ being a vector of the slopes from linear regression of each cluster, to write

$$P(s) := \sigma\left(-\delta_{C(s)}\right). \tag{2}$$

Observe that different sizes of subsets of most recent data to regress on can affect the accuracy of our learnability approximation. The size of the subset can be intuitively explained as the time scale over which we wish to measure learning progress. Generally, one might hypothesize that regressing on a subset of experiences based on their recency will result in higher performance, because this gives a more local approximation of the derivative. For example, imagine a TD error v. time graph that resembles a $y = \frac{1}{x}$ curve. This means the initial and recent rates of decrease in TD error are different, the latter being smaller than the former. Subsetting recent experiences to regress on will lead to a less negative slope of TD error than regressing on every experience, which means we assign a lower priority value on the new experience, whereas not subsetting would assign a higher priority value despite the lack of recent progress.

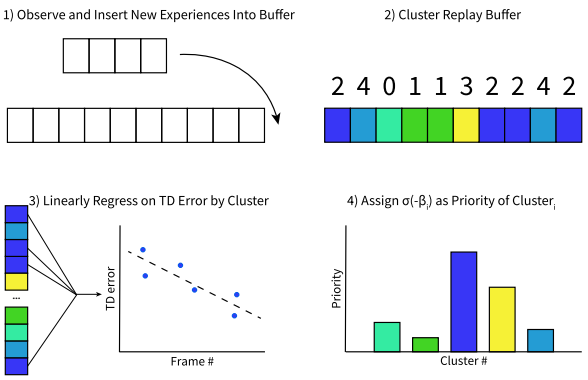

Figure 1: Implementation of DNLRC

Degenerate Naive Linear Regression on Clusters

Our final proposed method is identical to **Continuous Naive Linear Regression on Clusters** up to the point of the assignment of prioritization values to new experiences. The key difference here is that after a new experience is formed and classified to a cluster, every experience in the cluster, including the new experience, has its priority value set to sigmoid of the negative slope of the fitted line (Fig. 1). Furthermore, the number of recent experiences that are regressed on are limited to 100 in order to prevent very old experiments from significantly affecting training. Following this, experiences to replay are similarly sampled based on their priority value distribution. The goal here

1. Here, "chronological" is measured with respect to environment time steps. An experience encountered at the t-th time step is chronologically earlier than an experience encountered at the $t+1$-th time step.

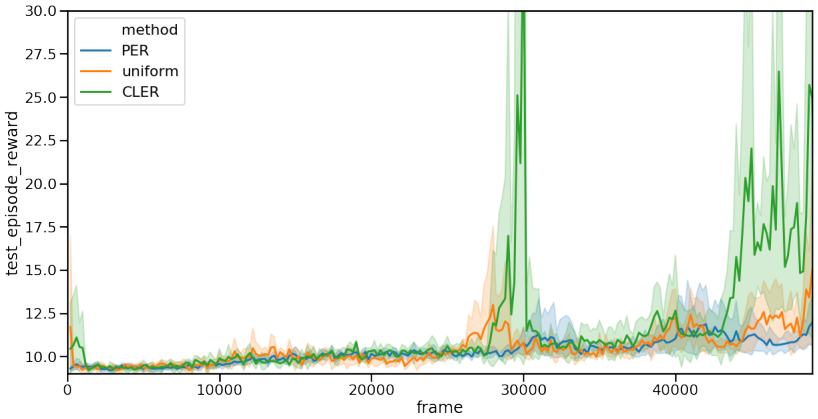

Figure 2: CLER achieves higher test episode rewards in fewer frames than PER and uniform

is to decrease variance in our model by restricting the priorities of each cluster to be identical. Almost identically to Eq. (2), letting $\mathscr{C}$ be our collection of clusters where $\mathscr{C}_i$ contains all experiences in the same cluster, we can write

$$P\left(\mathscr{C}_{C(s)}\right) := \sigma\left(-\delta_{C(s)}\right). \tag{3}$$

We will evaluate our prioritization method against the following baselines:

1. **Uniform sampling** (Original implementation of experience replay before PER was published, where experiences are uniformly sampled from the buffer (Lin, 1992))
2. **PER** (Prioritized Experience Replay; We expect this to be the best performing baseline and therefore the most important target of comparison to our proposed method (Schaul et al., 2016))

5. Experimental results and analysis

We test our three proposed prioritization methods (CMC, CNLRC, DNLRC), and tune clustering method (k-means, mean-shift) and number of clusters (3 to 7) as hyperparameters. In order to quantify how good each combination of prioritization/clustering method with hyperparameters was, we used the test episode reward (how well our agent performed in an episode) over time to both observe the final ability of an agent and how quickly it was able to learn. A summary of our results for each experiment we ran in comparison to PER and uniform sampling based on the aforementioned metric is in Table 1 in our Appendix. Throughout the rest of this section, we will focus on the best performing combination: DNLRC using mean-shift with 5 clusters, which we call **Cluster-wise Learnability Experience Replay (CLER)**.

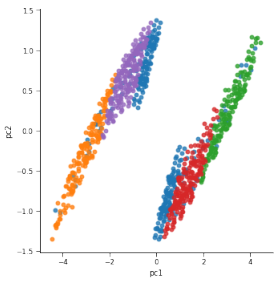

Figure 3: CLER-identified clusters

Sample complexity. Plotting the test episode reward of CLER against that of PER clearly demonstrates that, while CLER does have a higher variance than PER, it is able to achieve higher test episode reward in approximately 11% fewer frames (See Fig. 2).

Cluster formation. Because our prioritization algorithm depends on the buffer clustering process, one worthy analysis is to interpret the formed clusters. We performed PCA on the input state vectors stored in the buffer (which

are in the space that the clustering is performed in) and colored the points according to cluster assignment. (See Fig 3), the PCA plot for clustered states used in DNLRC utilizing k-means and 5 clusters[2]). Since the explained variance ratio for the second principal component was low, it is difficult to interpret this two-dimensional projection of the originally five-dimensional state space. However, this suggests that our clustering method results in the number of points in a cluster roughly being equal between clusters and that the distribution of points is close to uniform, which is a nice property to have for our proposed prioritization method which assumes that each experience belongs to a cluster containing enough number of experiences to be able to approximate the prioritization value well.

Sampled state distribution. Lastly, we also analyzed the distribution of states that were sampled using the weighted probabilities assigned by each prioritization algorithm. Specifically, we computed the proportion of experiences in the sampled set that were from the "noisy" environment, as opposed to the "non-noisy" environment, and compared this ratio to PER's ratio. CLER is able to sample significantly less states from the "noisy" environment than PER, meaning that CLER is better at identifying states that are either noisy or cannot be learned. This property was our main goal in creating CLER, and therefore this analysis indicates that our high-level theoretical goal was translated effectively into an algorithm.

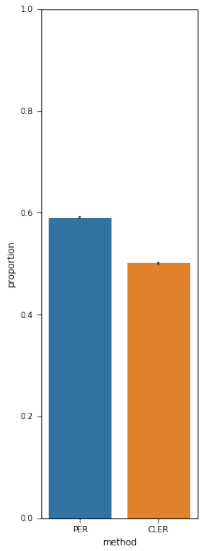

Figure 4: Proportion of states in the sampled set that were from the "noisy" environment

6. Conclusion

We observed that our learnability-based CLER algorithm obtains lower sample complexity (achieves higher reward at test time earlier in training) compared to the original prioritized experience replay (PER) algorithm. We believe that this is because CLER's prioritization scheme allows the model to ignore states with high TD error and low learnability. We also saw that out of the learnability-based prioritization algorithms we designed, DNLRC achieved the lowest sample complexity. It is perhaps not so surprising that this algorithm performed better than the other main learnability-based algorithm we implemented, CNLRC, because DNLRC assigns the same prioritization value, $\sigma(-\delta)$, to all experiences in the same cluster, which results in lower variance in the sampling scheme compared to CNLRC, which only updates the prioritization value of the experience that was newly assigned to a cluster. High variance of gradients is already a substantial challenge in RL, and we believe that DNLRC helps decrease the variance in the prioritization method, which in turn helps more stable training of our DQN. Finally, we believe the mean-shift clustering algorithm performed better than k-means as it allows a fewer number of clusters to be formed than specified, which makes buffer clustering more dependent on actual data observed.

For future work, we would like to think of better ways to approximate learnability. One idea is to use two Q-networks, $\mathcal{M}_{old}$ and $\mathcal{M}_{new}$, and update $\mathcal{M}_{new}$ with gradient descent and $\mathcal{M}_{old}$ with weight mixing with $\mathcal{M}_{new}$ such that $\mathcal{M}_{old}$ always lags behind. We can approximate the derivative of the TD error by computing the TD error using both of these networks and taking the difference between them. We would also like to explore probabilistic and theoretical guarantees on performance and perform rigorous hyperparameter tuning for parameters such as number of clusters. Lastly, the largest disadvantage of our method is that it is computationally more expensive than PER due to the constant need to re-cluster the RB. Thus, more efficient methods and different clustering algorithms are also areas that we would like to delve into.

All code and materials available at:
`https://github.com/megsano/progress_replay`

2. We would have liked to do the same analysis in CLER, but we unfortunately had an error in the saved data and we did not have time to re-run the experiments.

Contributions

Nomi was mainly responsible for implementing the k-means and mean-shift clustering methods, as well as both prioritization methods.

Megumi was mainly responsible for implementing the environment set up, performance metrics, and data analysis pipeline, as well as the Pong environment.

Both contributed to design of prioritization methods and monitored experiment runs. Most of our time was spent on developing the algorithm.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, jan 1975. ISSN 0018-9448. doi: 10.1109/TIT.1975.1055330. URL `http://ieeexplore.ieee.org/document/1055330/`.

higgsfield. Rl adventure. `https://github.com/higgsfield/RL-Adventure`, 2017.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, mar 1982. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489. URL `http://ieeexplore.ieee.org/document/1056489/`.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Tom Schaul, John Quan, Ioannis Antonogloy, and David Silver. Prioritized experience replay. *ICLR*, 2016.

Appendix

Prioritization Method	Clustering Method	# Clusters	Results
CMC	k-means	3	Significantly worse than PER
CMC	k-means	5	Significantly worse than PER
CMC	k-means	7	Significantly worse than PER
CMC	mean-shift	5	Significantly worse than PER
CNLRC	k-means	3	About as good as PER, but with much higher variance
CNLRC	k-means	4	About as good as PER, but with higher variance
CNLRC	k-means	5	About as good as PER, but with higher variance
CNLRC	k-means	6	About as good as PER, but with higher variance
CNLRC	k-means	7	About as good as PER, but with higher variance
CNLRC	mean-shift	3	About as good as PER, but with much higher variance
CNLRC	mean-shift	4	About as good as PER, but with higher variance
CNLRC	mean-shift	5	About as good as PER, but with higher variance
CNLRC	mean-shift	6	About as good as PER, but with higher variance
CNLRC	mean-shift	7	About as good as PER, but with higher variance
DNLRC	k-means	3	About as good as PER, but with much higher variance
DNLRC	k-means	4	About as good as PER, but with higher variance
DNLRC	k-means	5	About as good as PER, but with higher variance
DNLRC	k-means	6	About as good as PER, but with higher variance
DNLRC	k-means	7	About as good as PER, but with higher variance
DNLRC	mean-shift	3	About as good as PER, but with much higher variance
DNLRC	mean-shift	4	About as good as PER, but with higher variance
DNLRC	mean-shift	5	Better than PER, but with higher variance
DNLRC	mean-shift	7	About as good as PER, but with higher variance

Table 1: Summary of Results