Sampling Strategies for Deep Reinforcement Learning

Bryan Anenberg
Stanford University
Stanford, CA
anenberg@stanford.edu

Bharad Raghavan
Stanford University
Stanford, CA
bharadr@stanford.edu

Abstract

Q-learning algorithms that employ experience replay sample experiences from the replay memory. The quality of the experiences sampled dramatically impacts the training of the algorithm. We extend the Deep-Q-Learning algorithm of Mnih et. al [8, 7] by considering various strategies to better sample experiences when training the agent. We evaluate the impact each sampling strategy has on the learning rate and performance of the agent by training the agent to play five Atari 2600 games.

1. Introduction

The goal of the paper is to train a single neural network agent capable of successfully playing a variety of Atari 2600 games. This is achieved through a variety of extensions to the Deep Q-Learning Network (DQN) presented in [8], [7]. The agent learns to play the Atari games by simply looking at the game screen. The agent is not aware of any game-specific information, hand-crafted features, or the internal state of the game emulator. Rather, the agent extracts high-level visual features directly from the raw video frames using a convolutional neural network. The agent also is aware of the reward and terminal signals, and the set of possible actions for each game. The agent relies on a neural network to predict the Q-values (a global approximation approach), which are used to determine the policy. An experience replay mechanism is used to sample past transitions in order to update the parameters of the neural network through stochastic gradient descent. This paper explores various sampling techniques to better sample past transitions with the goal of expediting the training process and improving the policy.

2. Background

The experiments in this paper are direct extensions to the original [8] paper. In short, at each time step the agent interacts with the Atari emulator by selecting an action a_t, from the set of legal actions. The emulator's internal state is not observed by the agent; the agent only observes the image from the emulator, which is expressed as the vector $x_t \in \mathbb{R}^d$. The reward for the action a_t is expressed as the change in the game score. The agent is also aware of the number of lives it has left in the game. The goal of the agent is to select actions in a way that maximizes the expected future reward.

The optimal action-value function $Q^*(s, a)$ returns the maximum expected value attainable by following policy π after performing an action a from a state s.
$Q^*(s, a) = \max_\pi E[R_t \mid s_t = s, a_t = a, \pi]$. The state s_t is defined as the sequence of actions and observations $s_t = x_1, a_1, x_2, ..., a_{t-1}, x_t$. Bellman's equation is defined as follows:

$$Q^*(s, a) = E_{s' \sim S}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a] \quad (1)$$

The optimal value Q^* can be calculated given the optimal value $Q^*(s', a')$ where s' and a' are the sequence and action at the next time-step. It is not feasible to directly apply value iteration to iteratively update

$$Q_{t+1}(s, a) = E[r + \gamma \max_{a'} Q_t(s', a') \mid s, a] \quad (2)$$

until it converges to the optimal action-value function $Q^*(s, a)$ since we cannot enumerate all possible state-action pairs (s, a) for the Atari simulation. There are too many possible unique pixel inputs to explicitly enumerate all the possible states. Thus, a global function approximator (specifically a convolutional neural network "Q-network") is used to estimate the action-value function $Q(s, a; \theta) \approx Q^*(s, a)$. This model-free approach does not explicitly estimate the reward and transition functions [8].
We can measure how close the Q-network's $Q(s, a; \theta)$ is to the optimal action-value function $Q^*(s, a)$ by calculating the mean-squared error of the Bellman's equation, where the optimal target is
$y^* = r + \gamma \max_{a'} Q^*(s', a')$. The goal is to minimize

$$(y^* - Q(s, a; \theta))^2 \quad (3)$$

Algorithm 1 Deep Q-learning with Experience Replay and Sampling

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for $episode = 1$ **to** M **do**
  begin
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1$ **to** T **do**
    begin
    With probability ϵ select a random action a_t otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    $SAMPLE$ random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equations 5 and 6
    end
  end

We can approximate y^* using the Q-network: $y = r + \gamma \max_{a'} Q(s', a'; \theta_t^-)$. For a set of parameters θ_t at iteration t, the mean-squared error is given by the loss function

$$L_t(\theta_i) = E_{s,a,r}[(y - Q(s, a; \theta_t))^2] \tag{4}$$

As the loss decreases, the Q-network better approximates the optimal action-value function. We can optimize the Q-network by differentiating the loss with respect to the weights θ_t and performing stochastic gradient descent (with mini-batches).

$$\nabla_{\theta_t} L(\theta_t) = E_{s,a,r,s'}\left[(y - Q(s, a; \theta_t))\, \nabla_{\theta_t} Q(s, a; \theta_t)\right] \tag{5}$$

$$\theta_{t+1} := \theta_t + \eta \nabla_{\theta_t} L(\theta_t) \tag{6}$$

η is the learning-rate hyperparameter. The parameters θ_t^- of the target $y = r + \gamma \max_{a'} Q(s', a'; \theta_t^-)$ are held fixed when optimizing the tth loss function $L_t(\theta_t)$. The target Q-network parametrized by θ_t^- is periodically updated after a fixed number of mini-batch updates. It is not feasible to provide the Q-network with an arbitrary long sequence s_t as input. Therefore, the Q-network receives a fixed length input $\phi_t = \phi(s_t)$. In practice, the preprocessing function ϕ transforms the last four raw Atari frames (represented by $\{x_t, x_{t-1}, x_{t-2}, x_{t-3}\}$) from RGB to grey-scale, downscales and crops them from 210×160 pixels to 84×84 regions, and then stacks them into a $84 \times 84 \times 4$ tensor.

2.1. Experience replay

A transition from state s_t to state s_{t+1} by action a_t with reward r_t is referred to as an experience, $e_t = (s_t, a_t, r_t, s_{t+1})$. As the Q-network trains, the experiences are added to an array D . In the experiments, D is initialized to hold 1 million samples. To perform a Q-learning mini-batch update, a fixed set of experiences are drawn from D. The experiments in this paper investigate various strategies to sample the experiences from the experience replay pool D. D is also referred to as the replay memory.

There are many benefits to sampling experiences from replay memory D:

1. Efficient Data Re-utilization: Only a fixed number of experiences are stored, rather than the whole history of experiences. It's also possible for an experience to be sampled multiple times before it's overwritten from the replay memory.

2. Similar Benefits as with Eligibility Traces [2]: With eligibility traces the reward attained upon reaching the terminal state is propagated to all states along the trajectory. With experience replay it is possible to sample experiences at different points in time along the trajectory to a terminal state. In this way it is possible for states along the trajectory to help contribute to the update of the Q-network parameters.

3. Avoids Data Correlation: Consecutive experiences are correlated because consecutive frames are often very similar. Learning from consecutive experiences can often lead to skewed updates. Sampling experiences from D avoids this correlation. [7]

4. Ensures a Smoother Training Distribution: Since consecutive experiences are closely correlated, only sampling from consecutive experiences leads to significant

variance in the parameter updates. Sampling from consecutive experiences makes the training distribution volatile and unstable, which leads to oscillation and divergence among the Q-network parameters. Sampling a wide variety of nonconsecutive experiences from the replay memory ensures that the training distribution is more stable and learning is more consistent. [7].

3. Algorithm

For this paper, we only alter line 12 of Mnih et. al's Deep-Q-Learning algorithm [7, 8, 4] (see Algorithm 1 above). The $SAMPLE$ procedure is responsible for sampling a mini-batch of experiences from the replay memory used to update the parameters of the Q-network via stochastic gradient descent (Equation 6).

It is important that the sampling strategy selects a mini-batch of experiences that the Q-network will learn the most from. Identifying which experiences are the best for learning isn't obvious. We experiment with five different sampling techniques to determine which strategy most improves the learning rate and accuracy of the agent [6, 5, 3].

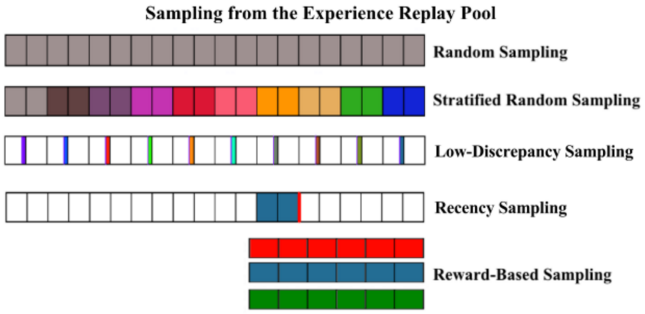

Figure 1. We represent the experience replay pool (replay memory) as an array of experiences stored in temporal order. The shaded portions of the array represent "buckets" to sample experiences from (different colors denote different buckets). For Recency Sampling, the red line indicates the location of the most recent experience and the blue region shows the experiences that precede it.

3.1. Baseline Sampling Strategy

The replay memory is an array that stores the last N experiences (or transitions) in temporal order. Also, we define x to be the number of experiences in a mini-batch update.

3.1.1 Standard Random Sampling

The baseline sampling strategy is uniform random sampling (also referred to as standard random sampling), which is used in [7, 8]. A mini-batch of size x consists of experiences selected with uniform probability across the entire replay memory. On average the mini-batch is representative of the entire replay memory since experiences are randomly sampled with uniform probability; no distinction is made between different experiences.

3.2. Temporally Based Sampling Strategies

As the agent learns to play the game more effectively, it will continue to encounter different (and potentially more rewarding) experiences. The distribution of experiences encountered by the agent evolves over time. It is important to consistently select experiences across the entire temporal history of the replay memory in order to capture the diversity of the replay memory.

3.2.1 Stratified Random Sampling

The first temporally-based sampling strategy is stratified random sampling. In stratified random sampling, the replay memory is divided into x distinct, non-overlapping groups. Each group contains temporally continuous experiences from a distinct period of time. The mini-batch is formed by randomly sampling one experience from each group. Stratified random sampling yields more temporally diverse mini-batches than standard random sampling.

3.2.2 Low-Discrepancy Sampling

Low-discrepancy sampling samples numbers based on a low-discrepancy sequence rather than a pseudo-random number generator. While a pseudo-random number generator samples numbers in a given range with uniform probability, a low-discrepancy sequence deterministically generates numbers that are uniformly spaced out within a given range. Low-discrepancy sampling generates numbers that both uniformly span a given range and are evenly spaced out amongst themselves. [6].

In the experiment we use the 1-dimensional Sobol Sequence [1] as the low-discrepancy sequence. To sample a mini-batch of size x, we generate a Sobol sequence of size x between 0 and 1, scale the numbers in that sequence to the size of our replay memory, and select the experiences found at the indices specified by the numbers in the sequence. This results in the selection of x experiences that both uniformly span and are evenly spaced out among the temporal domain.

3.2.3 Recency Sampling

Recency sampling generates a mini-batch of size x by selecting the x most recent experiences from the replay memory. These samples are all highly correlated since consecutive frames are often similar. We expect recency sampling to perform significantly worse than any sampling technique that takes advantage of the full experience replay memory.

Recency sampling provides a baseline to assess the impact of experience replay when training the Q-network.

3.2.4 Reward-Based Sampling

Every experience in the replay memory possesses a reward. Across all experiments the reward values range from -1 to 1. Reward-based sampling samples experiences uniformly based on their reward value. Specifically, the replay memory is organized into 3 arrays: a low-reward pool (storing experiences with rewards between [-1,-0.33)), a medium-reward pool (storing experiences with rewards between [-0.33,0.33]), and a high-reward pool (storing experiences with rewards between (0.33, 1]). To sample an experience, we randomly select one of the three pools and then uniformly randomly select an experience from that pool. This process is repeated until there are x experiences in the mini-batch.

4. Experiments

Figure 2. Atari 2600 games. From left to right: Assault, Atlantis, Breakout, Gopher, and Kangaroo.

The Q-Network is tested on five Atari 2600 games: Breakout, Assault, Kangaroo, Gopher, and Atlantis (Figure 2). This set of games is diverse in terms of visual input, game length, action space, and game strategy. The same network architecture and hyper-parameter settings are used across all experiments. The only factor changed in each experiment is the sampling strategy.

All experiments are performed with the same conditions as in the original paper, [7, 8]. Nevertheless, it is important to note that all negative rewards are clipped to to -1 and all positive rewards to 1, leaving the 0 rewards unchanged. The behavior policy is ϵ-greedy, with ϵ linearly annealed from 1.0 to 0.1 over the first 1 million frames and remaining at 0.1 thereafter. Additionally the replay memory only tracks the last 1 million experiences. A Q-learning parameter update to the Q-network is performed after every 4 experiences. Each mini-batch consists of 32 experiences sampled from the replay memory. Q-learning updates are only performed after the replay memory has accumulated 50000 experiences.

The results of the experiments are displayed (with 1 plot per game) in Figure 3. The plots illustrate the relative performance of the sampling techniques for each game. It is not possible to directly compare the sampling strategies across games due to the difference in the reward structures for each game.

4.1. Evaluation Metrics

The agent's performance is measured with two metrics:

1. Average Total Reward
2. Average Q-value

The average total reward is the average reward accumulated per episode (episode = one iteration of the game). This metric tends to be quite noisy, as small changes to the weights of the Q-network can significantly alter the policy and lead to drastically different states visited and rewards collected [7].

Another more stable metric is the average Q-value of the function, which estimates the average discounted reward the agent receives by following the current policy from any given state in the state space. This value is estimated by randomly selecting a set of states at the start of training and then tracking the average of the maximum predicted Q-values for this set of states based on the current Q-network. The average Q-value increases more smoothly than the average total reward.

As illustrated in Figure 3, the Average Total Reward and Average Q-value are calculated once every epoch during training, where an epoch equates to 25000 Q-learning parameter updates (mini-batch updates). Since a mini-batch update occurs every 4 experiences, an epoch spans 100000 experiences.

The plots in Figure 3 also allow one to evaluate the rate at which the parameters of the Q-network are optimized for each sampling technique. As evidenced in the figure, the various sampling techniques cause the Q-network to train at different rates.

5. Analysis

The plots in Figure 3 illustrate mixed results across the five games. Stratified random sampling consistently performs as well as, if not better than, the baseline (uniform random sampling). For example, stratified random sampling achieves a greater average reward and average Q-value than uniform random sampling on Breakout and Kangaroo. The consistency of stratified random sampling suggests that having greater temporal diversity among the samples is beneficial for the training of the Q-Network.

Reward-based sampling yields inconsistent results. Reward-based sampling performs worse than the other methods on Breakout, better on Gopher, and around the same level as uniform and stratified random sampling for Assault, Atlantis, and Kangaroo. It is possible that the performance of reward-based sampling is correlated to the reward structure of the game being played, making it a bet-

ter option for certain games, but less consistent overall than uniform and stratified random sampling.

As expected, recency sampling performs worse than the other sampling strategies across all the games. Recency sampling yields a lower average reward and generally a much lower average Q-value (with the exception of Breakout and Atlantis where it performs slightly worse than uniform random sampling). The relatively similar performance to the baseline on Breakout and Atlantis suggests that a long-term replay memory is not required to learn the strategy for simpler games. Breakout and Atlantis are more repetitive and visually simpler than the other games. For more complex games like Kangaroo and Assault, where the game evolves over time, not utilizing the full replay memory impedes the training of the Q-network in the long run.

Low-discrepancy sampling displays inconsistent results on the average Q-value plots. Low-discrepancy sampling results in exponential improvement to the average Q-value in every game except Assault. However, this extraordinary performance is not reflected in the average reward plots for these games. In fact, low-discrepancy sampling generally performs worse than the other sampling strategies across all games with respect to the average reward plots. The exponential increase in the average Q-values shown in the the figure do not correlate with improved performance in the agent. A possible explanation for the exponential divergence of low-discrepancy sampling on the average Q-value is the low diversity of experiences sampled in the Q-learning update. Experiences are consistently sampled from the same indices (with a small degree of variation), which leads to near identical mini-batch updates for long periods during training.

5.1. Learning Rate

The average Q-values for stratified and uniform random sampling generally increase at approximately the same rate. Reward-based sampling generally has a more inconsistent learning rate, initially outpacing stratified and uniform random sampling in Gopher but falling behind early on in Breakout and Kangaroo. Reward-based sampling often begins learning more slowly than the other methods, but achieves comparable Q-values after many epochs (as evidenced in Kangaroo).

An explanation for the slower initial learning rate of reward-based sampling is that positive and negative reward states are relatively scarce compared to 0 reward states during the beginning of training. The sampled mini-batch will on average contain an equal number of experiences from each reward state, regardless of how many experiences are in each bucket. This means the agent can potentially sample the same experience multiple times if there are very few experiences in a particular reward bucket. Until the agent accumulates more experiences, the samples might not be very diverse, which could inhibit learning.

As previously noted, recency sampling performs worse than the other methods and has an erratic and inconsistent learning rate. This observation confirms the claim that the Q-learning algorithm learns best from a stationary distribution of experiences. The full experience replay memory provides a stable source of experiences, and experiences sampled from the replay memory result in more stable training.

6. Conclusion

This paper built upon the work of Mnih et. al [7, 8] by extending their Deep-Q-Learning algorithm to utilize a variety of sampling strategies. Various experiments were performed to determine which strategy, if any, could outperform the baseline (Uniform Random Sampling). After testing the sampling strategies on five games, it was discovered that on the whole, no sampling strategy greatly improved upon the baseline, and the only sampling strategy found to perform consistently on par with or slightly better than Uniform Random Sampling was Stratified Random Sampling.

References

[1] Sobol sequence. `https://en.wikipedia.org/wiki/Sobol_sequence`, 2015. Sobol sequence strategies background reference.

[2] S. Adam, L. Busoniu, and R. Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 42(2):201–212, 2012.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012.

[4] I. Kuzovkin. Deepmind atari deep q-learner. `https://github.com/kuz/DeepMind-Atari-Deep-Q-Learner`, 2015.

[5] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.

[6] A. McDougall. Useful information on sampling strategies. `http://pages.csam.montclair.edu/~mcdougal/SCP/Sampling.htm`, 2015. Sampling strategies background reference.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

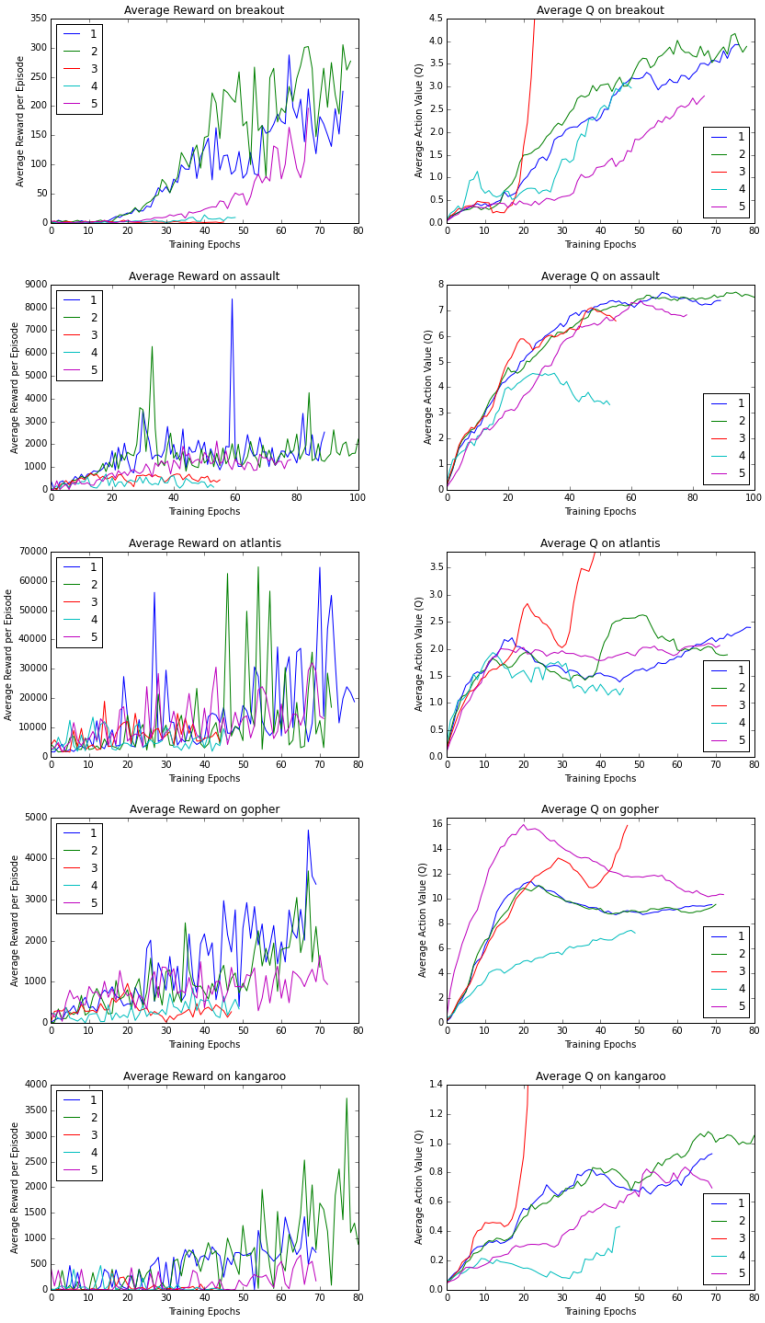

Figure 3. The left column shows the average reward per episode on Breakout, Assault, Atlantis, Gopher, and Kangaroo during training. The column on the right shows the average of the maximum predicted Q-values of a fixed set of states on the same five games. One epoch corresponds to 25,000 mini-batch weight updates. The blue line (1) corresponds to an algorithm using Uniform Random Sampling (baseline), the green line (2) denotes Stratified Random Sampling, the red line (3) denotes Low-Discrepancy Sampling, the light-blue line (4) denotes Recency Sampling, and the purple line (5) denotes Reward-Based Sampling.