The Impact of Determinism on Learning Atari 2600 Games

Matthew Hausknecht and Peter Stone
{mhauskn,pstone}@cs.utexas.edu
University of Texas at Austin

Pseudo-random number generation on the Atari 2600 was commonly accomplished using a Linear Feedback Shift Register (LFSR). One drawback was that the initial seed for the LFSR had to be hard-coded into the ROM. To overcome this constraint, programmers sampled from the LFSR once per frame, including title and end screens. Since a human player will have some random amount of delay between seeing the title screen and starting to play, the LFSR state was effectively randomized at the beginning of the game despite the hard-coded seed. Other games used the player's actions as a source of randomness. Notable pseudo-random games include Adventure in which a bat randomly steals and hides items around the game world and River Raid which used randomness to make enemy movements less predictable.

Relying on the player to provide a source of randomness is not sufficient for computer controlled agents which are capable of memorizing and repeating pre-determined sequences of actions. Ideally, the games themselves would provide stochasticity generated from an external source such as the CPU clock. In practice, this was not an option presented by the hardware. **Atari games are deterministic given a fixed policy leading to a set sequence of actions.** This article discusses different approaches for adding stochasticity to Atari games and examines how effective each approach is at derailing an agent known to memorize action sequences. Additionally it is the authors' hope that this article will spark discussion in the community over the following questions:

- How important is stochasticity in Atari 2600 games?
- Can an agent be considered general if it is memorizing multiple deterministic games?
- Should a distinction be made between agents that memorize sequences of actions and those that do not?
- Given that Atari games are deterministic, what is the right way to inject randomness?
- How well do the current methods for adding stochasticity perform?

This article examines three ways to add stochasticity: **1)** Randomly skipping frames at the beginning of an episode **2)** ϵ-greedy action selection, and **3)** randomly repeating actions throughout an episode.

Appeared in AAAI-2015 workshop on Learning for General Competency in Video Games; Jan 26, 2015; Austin, TX, USA

Case Study

In order to test the efficacy of different methods for adding randomness, we re-use an agent from (Hausknecht et al. 2013), *memorizing-NEAT*, a memorizing agent evolved on fully-deterministic Atari games. Its counterpart, *randomized-NEAT*, is an agent that was evolved on the same set of games with ϵ-greedy action selection ($\epsilon = .005$). Randomized-NEAT exhibits lower performance, but increased tolerance for randomness.

The following experiments compare the performance of memorizing-NEAT and randomized-NEAT under different types of stochasticity. In all of the following figures, solid, rectangular boxplots correspond to memorizing-NEAT while hollow, pinched boxplots correspond to randomized-NEAT. Each boxplot represents a single evaluation of all 61 Atari games currently supported by ALE. Z-Score normalization is then applied to normalize the per-game scores of all results in that figure.

An ideal type of randomness should decrease the scores of memorizing-NEAT without degrading the scores of randomized-NEAT.

Random Initialization

Random initialization is currently implemented in ALE and adds a random number of NOOP actions to the beginning of each game. The number of NOOPs is selected uniformly between 0 and 499. This option is enabled in ALE by adding the command line flag `-use_environment_distribution`. Random initialization is minimally invasive in the sense that it allows the agent to play exactly as it wants provided it can handle the randomized starting state. Furthermore, random initialization resulted in a change in memorizing-NEAT's scores for all 61 games (except Pitfall which had zero score in both cases). This indicates that random initialization forced memorizing-NEAT to start playing in an unfamiliar game state for all 61 games.

While effective at adding stochasticity, random initialization is quite detrimental to the performance of randomized-NEAT, indicating that it may be too harsh of a randomization method (Figure 1).

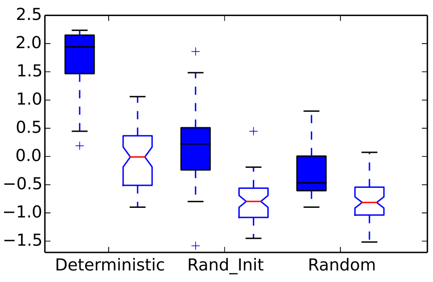

Figure 1: Effects of **random initialization** on memorizing-NEAT (solid rectangular boxplots) and randomized-NEAT (pinched hollow boxplots). Reference scores of each agent are provided in a fully deterministic environment and a fully random environment (enforced $\epsilon = 1$ greedy action selection). Higher aggregate Z-Scores are better.

Epsilon-Greedy Action Selection

ϵ-greedy action selection chooses a random legal action at each frame with probability ϵ. (Bellemare et al. 2013; Mnih et al. 2013) used ϵ-greedy action selection with $\epsilon = .05$.

Enforcing an ϵ-greedy action selection step in ALE would be difficult to implement in an algorithm-friendly manner. Two main factors to consider: 1) should ALE overwrite a requested action with a random one or simply insert a random action after a requested action? and 2) Should ALE report back to the algorithm that it overwrote/inserted a random action or should it silently take the random action and report the resulting reward and next state as if nothing special happened? The former would require a more complex agent/ALE interface, while the latter would hide potentially important information from the agent. Given the dissatisfying qualities of both options, perhaps the least of all evils is to encourage some standard value of ϵ and rely on practitioners to implement and self-report.

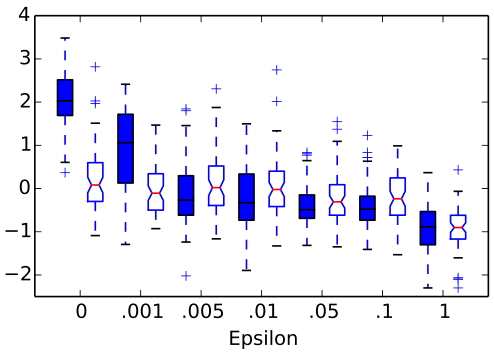

Figure 2: **ϵ-Greedy Action Selection**: Even small values of ϵ drastically reduce memorizing-NEAT's performance. $\epsilon = 0$ corresponds to the entirely deterministic agent while $\epsilon = 1$ is a completely random agent.

Figure 2 indicates that ϵ-greedy action selection is effective at derailing memorizing-NEAT even at small values of ϵ such as 0.005. Perhaps the prior practice of using $\epsilon = .05$ could be relaxed, leading to increased agent performance.

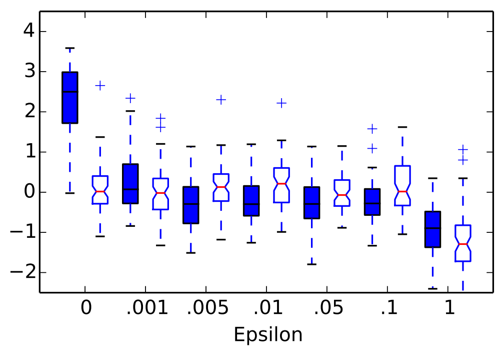

Figure 3: **ϵ-Repeat Action Selection** has the most detrimental effects towards memorizing-NEAT and the least effect on randomized-NEAT.

Epsilon-Repeat Action Selection

Rather than choosing an entirely random action with probability ϵ, ALE could instead repeat the last requested action for an extra frame. This would have a randomizing effect for all but the most degenerate of policies.[1] Additionally, as Figure 3 shows, repeating a selected action is less detrimental than selecting an action entirely at random. Implementation-wise, enforcing randomized action repeats in ALE would have the same complications as enforcing ϵ-greedy action selection. Figure 3 confirms that ϵ-repeat action selection is just as effective as ϵ-greedy action selection at degrading memorizing-NEAT's performance but has very little effect on randomized-NEAT.

Discussion

Any form of forced randomness that does not come from the environment will necessarily degrade the performance of a learning agent. Of the different methods for adding stochasticity to Atari 2600 games, ϵ-repeat action selection best fits the desired criteria: it has the most detrimental effects towards memorizing agents and is the least detrimental to already randomized agents.

In the future, perhaps the best way to overcome the Atari 2600's determinism is through two-player games (or competitions) in which randomness stems from the other player.

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res. (JAIR)* 47:253–279.

Hausknecht, M.; Lehman, J.; Miikkulainen, R.; and Stone, P. 2013. A neuroevolution approach to general atari game playing. In *IEEE Transactions on Computational Intelligence and AI in Games*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *CoRR* abs/1312.5602.

[1] A policy that only selects a single action would be unaffected.