1 Introduction

This report shows a comparison between the results of running several different reinforcement learning algorithms and policies on the three state problem, running on a simulated environment. All the implementation is provided by libPG library. To make the comparison as fair as possible, every algorithm uses a specifically tuned-up set of parameters values (κ, α, λ and ϵ), which were found by exhaustive search.

With softmax policy, the temperature decaying function used was $T(t) = \frac{1}{(1+\kappa t)2}$, where t is the current time step and κ is a constant. Similarly, with ϵ-greedy policy, the ϵ decaying function used was $\epsilon(t) = \frac{1}{(1+\kappa t)2}$. In both cases the constant κ defines how fast the used policy becomes greedier with time, when selecting next actions. The others parameters, α and λ, stands for step size and eligibility traces, respectively.

The results of each algorithm and policy combination were averaged over 100 runs and the temporal difference discount used was $\gamma = 0.6$ in all cases.

2 The Three State Problem

The three state simulator is a common place three-state POMDP employed by Baxter et al and is implemented in libPG.

3 Value Controllers

The main idea of using SARSA and Q-Learning (as defined in sections 7.5 and 7.6 of [Sutton and Barto]) to approach the three state problem is to understand which policy evaluation approach (i.e. on-policy or off-policy) performs better in this case. Also, since action selection influences the results, we tried both softmax and ϵ-greedy policies. Below, it is presented the algorithms and the corresponding resulting graphs.

3.1 SARSA(λ) with Softmax Decaying Temperature Policy

```
double discount  = 0.4;
double step_size = 0.001;
double kapa      = 1e-05;
Controller* controller = new SARSAController(
                         new NeuralNet(nnDims, squash),
               (Policy*) new SoftmaxPolicy(new Temp(kapa)),
                         TD_DISCOUNT);
OLPomdp* learner = new OLPomdp(controller,
                   new ThreeState(), discount, step_size);
```

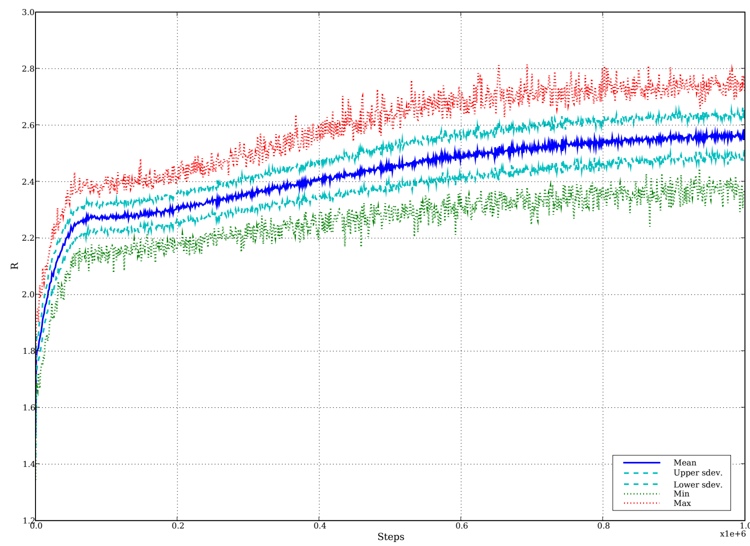

Figure 1: SARSA(λ) with softmax policy (decaying temperature)

3.2 SARSA(λ) with Constant ϵ ϵ-Greedy Policy

```
double discount  = 0.5;
double step_size = 0.001;
double epsilon   = 0.01;
Controller* controller = new SARSAController(
                         new NeuralNet(nnDims, squash)
               (Policy*) new eGreedyPolicy(epsilon),
                         TD_DISCOUNT);
OLPomdp* learner = new OLPomdp(controller,
                   new ThreeState(), discount, step_size);
```

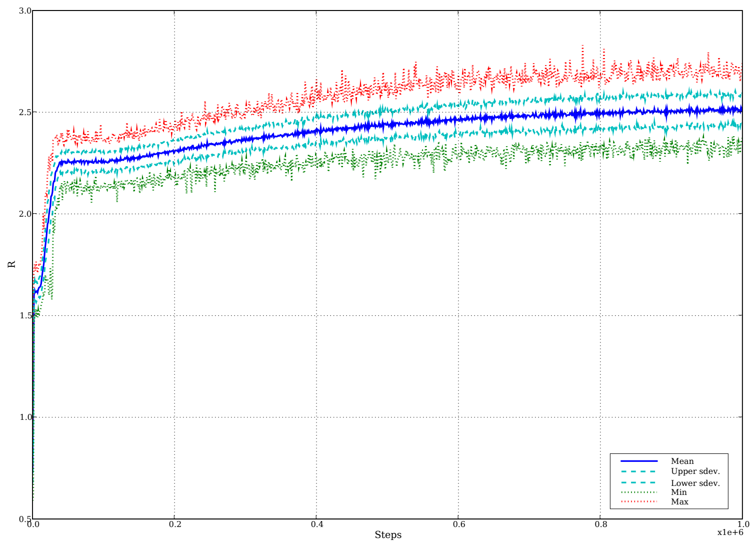

Figure 2: SARSA(λ) with ϵ-greedy policy (constant ϵ)

3.3 SARSA(λ) with Decaying ϵ ϵ-Greedy Policy

```
double discount  = 0.7;
double step_size = 0.001;
double kapa      = 0.001;
Controller* controller = new SARSAController(
                         new NeuralNet(nnDims, squash),
               (Policy*) new eGreedyPolicy(
                         new EpsilonDecay(kapa)),
                         TD_DISCOUNT);
OLPomdp* learner = new OLPomdp(controller,
                   new ThreeState(), discount, step_size);
```

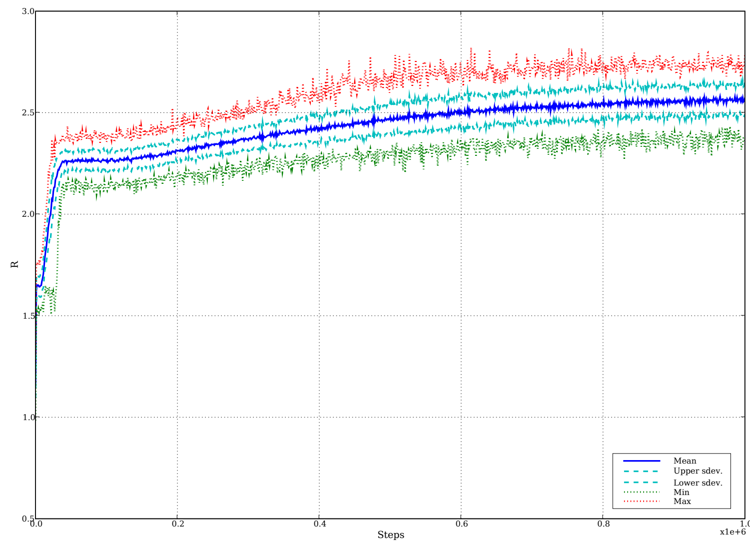

Figure 3: SARSA(λ) with ϵ-greedy policy (decaying ϵ)

3.4 Q-Learning(λ) with Decaying Temperature Softmax Policy

```
double discount  = 0.4;
double step_size = 0.001;
double kapa      = 1e-05;
Controller* controller = new QLearningController(
                         new NeuralNet(nnDims, squash),
               (Policy*) new SoftmaxPolicy(
                         new Temp(kapa)),
                         TD_DISCOUNT);
OLPomdp* learner = new OLPomdp(controller,
                   new ThreeState(), discount, step_size);
```

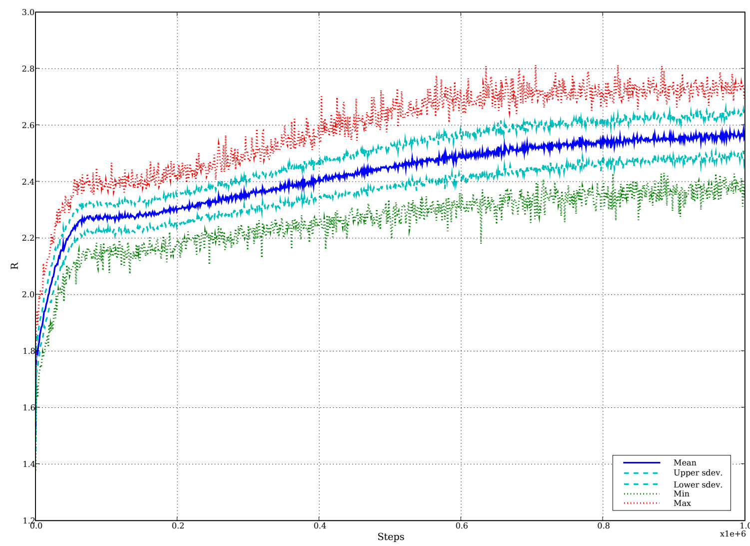

Figure 4: Q-Learning(λ) with softmax policy (decaying temperature)

3.5 Q-Learning(λ) with Constant ϵ ϵ-Greedy Policy

```
double discount =  0.5;
double step_size = 0.001;
double epsilon =   0.01;
Controller* controller = new QLearningController(
                         new NeuralNet(nnDims, squash),
               (Policy*) new eGreedyPolicy(epsilon),
                         TD_DISCOUNT);
OLPomdp* learner = new OLPomdp(controller,
                   new ThreeState(), discount, step_size);
```

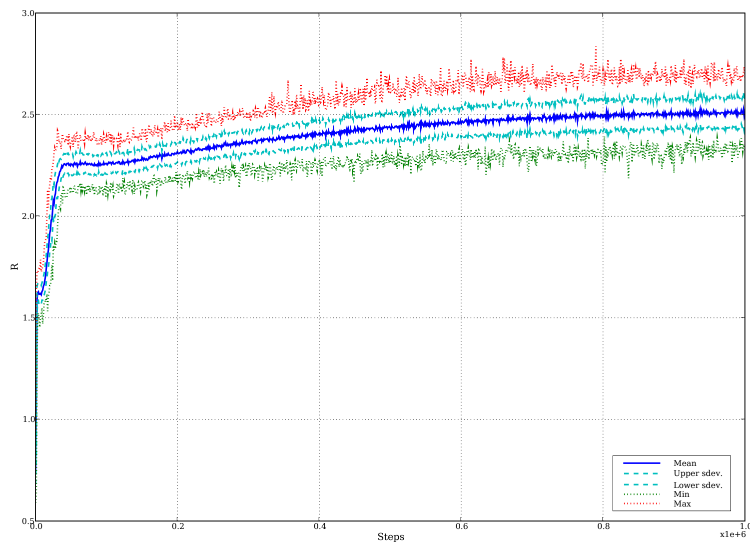

Figure 5: Q-Learning(λ) with ϵ-greedy policy (constant ϵ)

3.6 Q-Learning(λ) with Decaying ϵ ϵ-Greedy Policy

```
double discount  = 0.4;
double step_size = 0.001;
double kapa      = 0.001;
Controller* controller = new QLearningController(
                         new NeuralNet(nnDims, squash),
               (Policy*) new eGreedyPolicy(
                         new EpsilonDecay(kapa)),
                         TD_DISCOUNT);
OLPomdp* learner = new OLPomdp(controller,
                   new ThreeState(), discount, step_size);
```

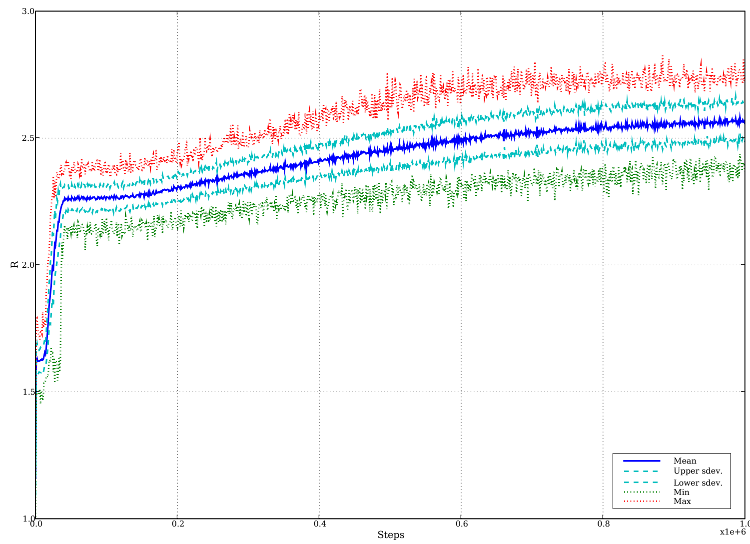

Figure 6: Q-Learning(λ) with ϵ-greedy policy (decaying ϵ)

3.7 Binary Controller

```
double discount  = 0.6;
double step_size = 0.001;
Controller* controller = new BinaryController(
                            new NeuralNet(nnDims, squash));
OLPomdp* learner = new OLPomdp(controller,
                         new ThreeState(), discount, step_size);
```

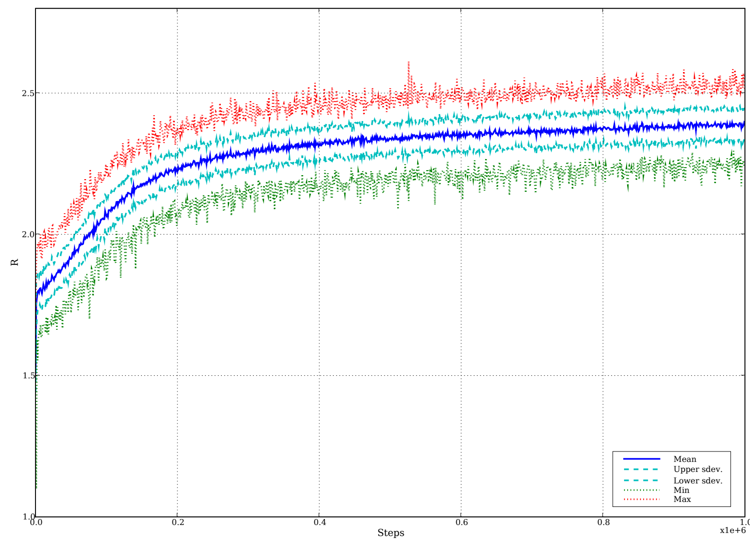

Figure 7: Binary controller

4 Policy Gradient Controllers

Natural Actor-Critics method (as defined in [Peters, Vijayakumar and Schaal]) was also used in comparison with value methods.

4.1 NAC Transform

```
double discount =  0.4;
double step_size = 0.001;
Controller* controller= new NACTransform(
                        new BinaryController(
                        new NeuralNetBatch(nnDims, squash)),
                        TD_DISCOUNT);
OLPomdp* learner = new OLPomdp(controller,
                   new ThreeState(), discount, step_size);
```

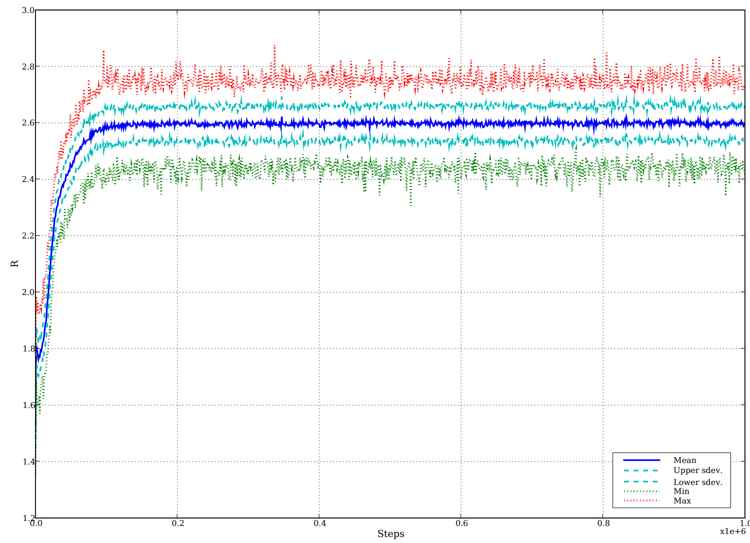

Figure 8: NAC Transform

5 Conclusion

Among value based algorithms, SARSA behaved slightly better than Q-Learning when using the same policy. But, in both algorithms, ϵ-greedy with decaying ϵ performed better than softmax with decaying temperature. The worst result was achieved with ϵ-greedy with constant ϵ, as expected.

Natural Actor-Critics was the best of all methods by far, converging very quickly to the optimum solution, being the most suited algorithm for the three state problem.

References

[Sutton and Barto] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*

[Peters, Vijayakumar and Schaal] Jan Peters, Sethu Vijayakumar, Stefan Schaal. *Natural Actor-Critic*