SARSA : Trace accumulation vs Trace Replacement

Algorithm

```
Initialize Q(s,a) arbitrarily and e(s,a)=0, for all s,a
Repeat(for each episode)
    Initialize s,a
    Repeat(for each step of episode)
        Take action a, observe r, s’
        Choose a’ from s’ using the behavior policy
        δ = r + γQ(s’,a’)-Q(s,a)
        e(s,a) = 1
        For all s,a:
            Q(s,a) = Q(s,a) +αδe(s,a)
            e(s,a) = γλe(s,a)
        s = s’
        a = a’
    until s is terminal
```

Trace Accumulation

The entire SARSA learning algorithm remains same, but the trace updation step in this case is : `trace[s, a] = trace[s, a] + 1`

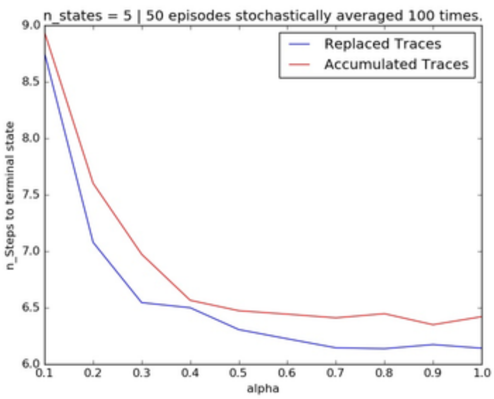
Trace Replacements

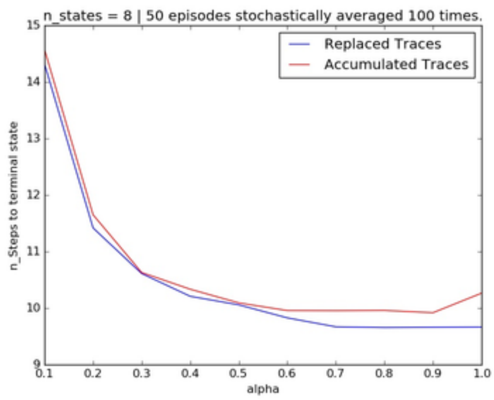
The entire SARSA learning algorithm remains same, but the trace updation step in this case is : `trace[s, a] = 1`

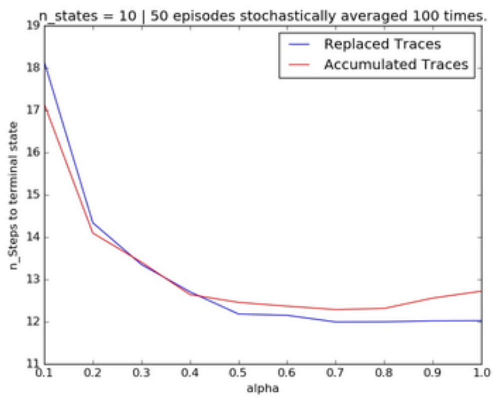
Comparison

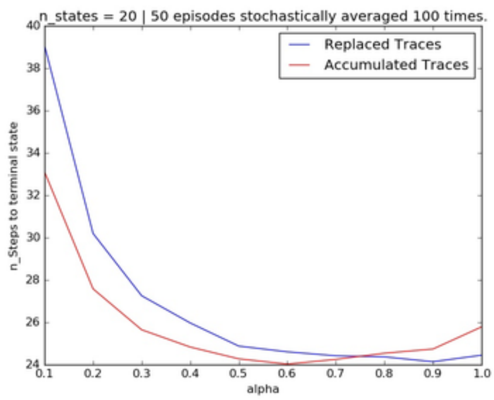
For the given MDP, since we are given the reward only on reaching the terminal state, **accumulating traces** is detrimental to learning as individual step rewards are unknown. **Replacing traces** works well with the **highly stochastic** nature of the MDP and learns in a **prior-free** (unbiased from previous decisions) fashion and thus converges to termination **faster** than the former. We can empirically see using the following figures which clear show that replacing traces **outperforms** accumulating traces for relatively high alphas.

Following plots with all parameters except the `n_states` , we can clearly see that **replacing traces > accumulating traces** for relatively high alphas





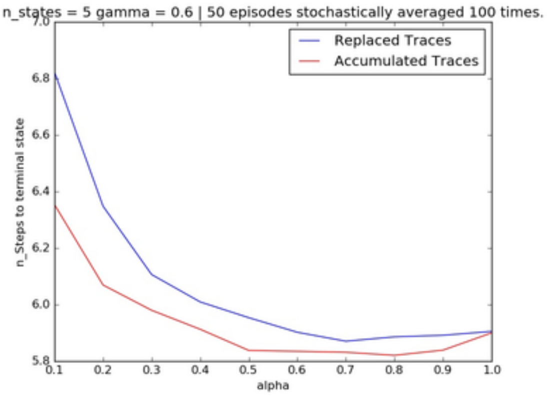


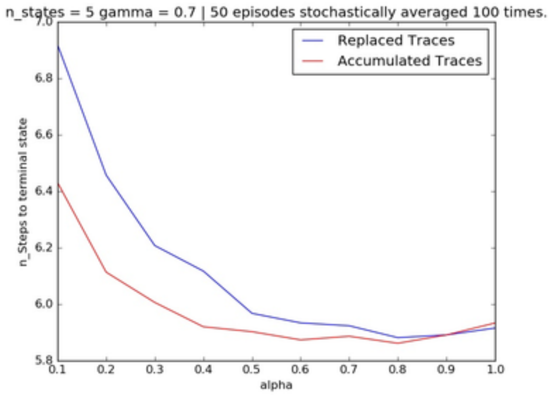


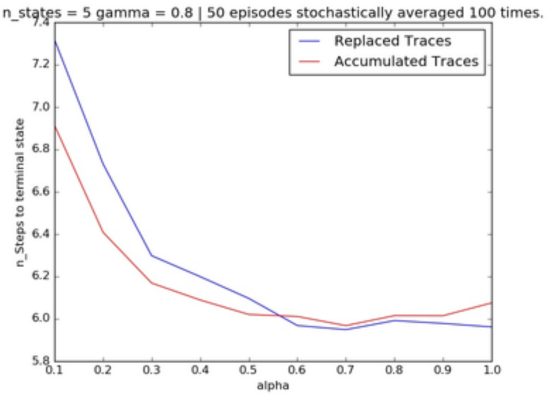
Conclusion

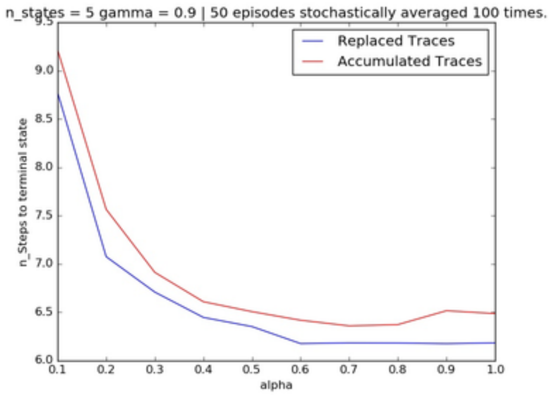
For a given `gamma` , `alpha` and `epsilon` replacing traces outperforms accumulating traces. However we can empirically see this advantage disappears as we increase the number of states / decrease the value of gamma. The intuition behind this can be seen by the fact that with increment in the number of states / decrease in gamma, the reward-delay increases and the MDP becomes more non-Markovian. Since eligibility traces (replacing) are used as a trade-off between TD and MC methods to overcome reward-delays, we see that it **looses its advantages for very long and less discounted** MDPs as the one used in this problem.

In all these plots we increase the value of `gamma` from `0.5` to `0.9` in steps of `0.1` :









References

- http://www-all.cs.umass.edu/pubs/1995_96/singh_s_ML96.pdf
- https://www.tu-chemnitz.de/informatik/KI/scripts/ws0910/ml09_7.pdf (Best explanation for accumulation traces)
- http://www.karanmg.net/Computers/reinforcementLearning/finalProject/KaranComparisonOfSarsaWatkins.pdf

Running code

As suggested in the instructions my code can be run using the files in the `src` directory:

- `python sarsa.py <n_states> <gamma>`
- `./sarsa.sh` to run all the experiments

Plots (similar to the ones I have provided in `imgs` directory) will be generated

Additional Notes

- Due to stochasticity of the experiments, I chose to average the results for first `n_episodes = 50` over a large number (`n_trials = 100`) iterations to retain the credibility of the results obtained.
- I have chosen 10 alpha values ranging from `0.1 to 1 (both inclusive)` with the `alpha[i] = 0.1 * i` .
- The values for `gamma` and `epsilon` were empirically chosen so as to run the experiment quickly.