Load balancing problem with Least-Square Policy Iteration

Vincent Antaki

McGill University

LSPI

Least-Square Policy Iteration

Michail G. Lagoudakis & Ronal Parr [2003]. *Least-Square Policy Iteration*

LSTDQ

LSTDQ $(D,\ k,\ \phi,\ \gamma,\ \pi)$ // Learns $\widehat{Q}^{\pi}$ from samples

// D : Source of samples (s, a, r, s')
// k : Number of basis functions
// ϕ : Basis functions
// γ : Discount factor
// π : Policy whose value function is sought

$\widetilde{\mathbf{A}} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$\widetilde{b} \leftarrow \mathbf{0}$ // $(k \times 1)$ vector

for each $(s, a, r, s') \in D$

$$\widetilde{\mathbf{A}} \leftarrow \widetilde{\mathbf{A}} + \phi(s,a)\Big(\phi(s,a) - \gamma\phi(s', \pi(s'))\Big)^{\intercal}$$

$$\widetilde{b} \leftarrow \widetilde{b} + \phi(s,a)r$$

$$\widetilde{w}^{\pi} \leftarrow \widetilde{\mathbf{A}}^{-1}\widetilde{b}$$

return $\widetilde{w}^{\pi}$

Figure: The LSTDQ algorithm

- Linear function approximation.
- Need to use basis functions.
- Requires pseudo-matrix inversions.

* Image from Lagoudakis & Parr, *Least-Squares Policy Iteration*

Load balancing problem - variant 1

- The agent has 3 servers at its disposition and needs to dispatch them tasks it receives. Tasks arrive randomly following a Poisson distribution with $\lambda = 2$.
- Tasks requires a certain amount of work to be completed. This amount of work required to complete a task is equal to $1 + T$ where T is a random poisson variable with $\lambda = 5$. The agent never knows the associated workload with a task.
- All server queues have a maximum length of 10. If the agent tries to add a task to an already full queue, the task is discarded and the agents receive a -5 points reward.
- At every timestep, all servers accomplish one unit of work on the first task in their queue.
- Upon the completion a task by a server, the agent receives a reward equal to $\frac{5}{\#\text{ of iteration to complete task}}$.

Load balancing problem - variant 2

- We now have just 2 servers
- The amount of work generated by the servers is now different for every server and stochastic.
- Distributions : $\mathcal{N}(0.8, 0.1), \mathcal{N}(1.2, 0.1)$

Load balancing problem - state definition

For simplicity, we consider the state to be defined as follows :

- Current timestep
- Queue length of all servers
- Time since the current task has been added to the queue for all servers
- Number of timesteps spent on current task for all servers.

To notice :

- Multiples tasks can arrive at the same timestep.
- Decisions are not spread at a regular interval through time.

Considerations

Difficulties inherent to that problem :

- Delayed reward
- A bit of stochasticity
- Partially observable state
- Would be better formulated as an average reward problem?

Considerations

The optimal policy is however very simple.

- Give the task to the server which is expected to complete it the soonest.

For the first variant, this means :

- Give the task to the server with the shortest queue.
- If multiples servers have the shortest length of queue, give the task to the one which been running his task for the longest amount of time

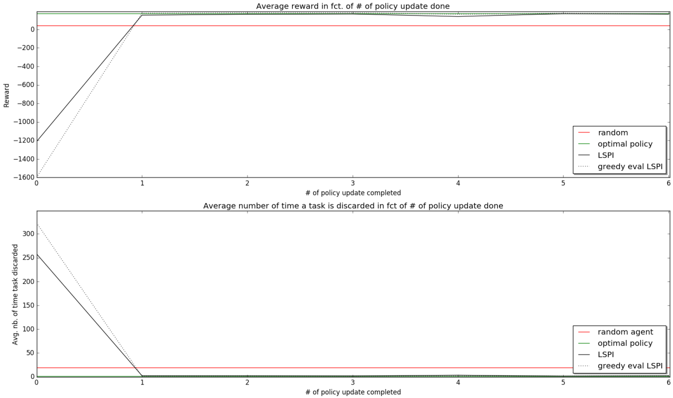
Choice of features - basis function dilemma

- Our qvalues are probably not a linear combination of the state as previously expressed. We need to consider basis functions to use for preprocessing.
- We do not wish to spend too much time on designing features; we would prefer the algorithm to sort it out.
- However, adding new features means we need to increases the amount examples to look at (to avoid overfitting).
- Increasing the amount of examples is a costly option since LSTDQ does a pseudo-inverse operation.

LSPI settings

- 100 episodes for policy evaluation
- $\gamma = 0.95$
- Our behavior policy is ϵ-greedy with $\epsilon = 0.2$
- Initially, we attempted $\phi(s, a) =$concat($[\vec{0}$ if $a' \neq a$ else $s \forall\ a']$) where $\vec{0}$ is a null vector the same size as s. Results were unsucessful.
- $\phi(s, a)$: [timestep, $l_{a'} - l_a\ \forall a' \neq a,\ t_{a'} - t_a\ \forall a' \neq a$] where l_a is the length of the queue corresponding to action a and t_a the number of time since the first task of queue a was added to the queue.

Results - variant 1



- Although these results are impressive, they can be mostly be credited to the use of the appropriate set of features.

Results - variant 2

Unfortunatly, our set of features did not achieve decent results on the second variant. Here are some features considered :

- the state
- the indicators $l_{a'} = 0 \;\forall\; a'$ and, if $l_{a'} \neq 0$, ratios between queues length $\frac{l_a}{l_{a'}}, \frac{l_{a'}}{l_a}$ else $\vec{0}$.
- Indicators over $\frac{l_{a'}}{l_a} > r$ for $r \in [0.1, 0.2, 0.25, 0.33, 0.5, 1, 2, 3, 4, 5, 10]$
- a onehot encoding of the action.
- Appliying tanh, log out of despair.
- Action-based sparse encoding of input : $\phi(x, a) =$concat($[\vec{0}$ if $a' \neq a$ else $x \;\forall\; a']$)
- For binary variables, we also considered the transformation $\phi(x) = [x, 1 - x]$.

In the end, none worked. We remain cursed with a linear model in a very non-linear environment.

Conclusion

What have has been learn through those experiments?

- Methods like LSPI generate a new set of samples at each iteration and process it in one batch.
- Basis function dilemma.
- The load balancing problem (variant 2) is hard. Non-linear method should be considered.

The End

Thank you!