A Row-rank properties of $\mathbf{SA}$

To ensure the right pseudo-inverse is well-defined in Section 5, we show that the projected matrix $\mathbf{SA}$ is full row-rank with high probability, if $\mathbf{A}$ has sufficiently high rank. We know that the probability measure of row-rank deficient matrices for $\mathbf{S}$ has zero mass. However in the following, we prove a stronger and practically more useful claim that $\mathbf{SA}$ is far from being row-rank deficient. Formally, we define a matrix to be δ-*full row-rank* if there is no row that can be replaced by another row with distance at most δ to make that matrix row-rank deficient.

Proposition 2. *Let $\mathbf{S} \in \mathbb{R}^{k\times d}$ be any Gaussian matrix with 0 mean and unit variance. For $r_A = rank(\mathbf{A})$ and for any $\delta > 0$, $\mathbf{SA}$ is δ-full row-rank with probability at least $1 - \exp(-2\frac{(r_A(1-0.8\delta)-k)^2}{r_A})$.*

Proof. Let $\mathbf{A} = \mathbf{U\Sigma V}^\top$ be the SVD for $\mathbf{A}$. Since $\mathbf{U}$ is an orthonormal matrix, $\mathbf{S}' = \mathbf{SU}$ has the same distribution as $\mathbf{S}$ and the rank of $\mathbf{SA}$ is the same as $\mathbf{S}'\mathbf{\Sigma}$. Moreover notice that the last $d - r_A$ columns of $\mathbf{S}'$ get multiplied by all-zero rows of $\mathbf{\Sigma}$. Therefore, in what follows, we assume we draw a random matrix $\mathbf{S}' \in \mathbb{R}^{k\times r_A}$(similar to how $\mathbf{S}$ is drawn), and that $\mathbf{\Sigma} \in \mathbb{R}^{r_A\times r_A}$ is a full rank diagonal matrix. We study the rank of $\mathbf{S}'\mathbf{\Sigma}$.

Consider iterating over the rows of $\mathbf{S}'$, the probability that any new row is δ-far from being a linear combination of the previous ones is at least $1 - 0.8\delta$. To see why, assume that you currently have i rows and sample another vector $\mathbf{v}$ with entries sampled i.i.d. from a standard Gaussian as the candidate for the next row in $\mathbf{S}'$. The length corresponding to the projection of any row $\mathbf{S}'_{j:}$ onto $\mathbf{v}$, i.e., $\mathbf{S}'_{j:}\mathbf{v} \in \mathbb{R}$, is a Gaussian random variable. Thus, the probability of the $\mathbf{S}'_{j:}\mathbf{v}$ being within δ is at most 0.8δ. This follows from the fact that the area under probability density function of a standard Gaussian random variable over $[0, x]$ is at most $0.4x$, for any $x > 0$.

This stochastic process is a Bernoulli trial with success probability of at least $1-0.8\delta$. The trial stops when there are k successes or when the number of iterations reaches r_A. The Hoeffding inequality bounds the probability of failure by $\exp(-2\frac{(r_A(1-0.8\delta)-k)^2}{r_A})$. □

B Alternative iterative updates

In addition to the proposed iterative algorithm using a left-sided sketch of $\mathbf{A}$, we experimented with a variety of alternative updates that proved ineffective. We list them here for completeness.

We experimented with a variety of iterative updates. For a linear system, $\mathbf{Aw} = \mathbf{b}$, one can iteratively update using $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t)$ and $\mathbf{w}_t$ will converge to a solution of the system (under some conditions). We tested the following ways to use sketched linear systems.

First, for the two-sided sketched $\mathbf{A}$, we want to solve for $\mathbf{S}_L\mathbf{A}\mathbf{S}_R^\top\mathbf{w} = \mathbf{S}_L\mathbf{b}$. If $\tilde{\mathbf{A}}_t = \mathbf{S}_L\mathbf{A}_t\mathbf{S}_R^\top$ is square, we can use the iterative update

$$\begin{aligned}
\tilde{\mathbf{A}}_{t+1} &= \tilde{\mathbf{A}}_t + \frac{1}{t+1}\left(\mathbf{S}_L\mathbf{e}_t(\mathbf{S}_R\mathbf{d}_t)^\top - \tilde{\mathbf{A}}_t\right)\\
\tilde{\mathbf{b}}_{t+1} &= \tilde{\mathbf{b}}_t + \frac{1}{t+1}\left(r_{t+1}\mathbf{S}_L\mathbf{e}_t - \tilde{\mathbf{b}}_t\right)\\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha_t(\tilde{\mathbf{b}}_{t+1} - \tilde{\mathbf{A}}_{t+1}\mathbf{w}_t)\\
&= \mathbf{w}_t + \alpha_t(\mathbf{S}_L\mathbf{b}_{t+1} - \mathbf{S}_L\mathbf{A}_{t+1}\mathbf{S}_R^\top\mathbf{w}_t)
\end{aligned}$$

and use $\mathbf{w}$ for prediction on the sketched features. Another option is to maintain the inverse incrementally, using Sherman-Morrison

$$\begin{aligned}
\mathbf{a}_d &= \mathbf{d}_t^\top\mathbf{S}_R^\top\tilde{\mathbf{A}}_t^{-1}\\
\mathbf{a}_u &= \tilde{\mathbf{A}}_t^{-1}\mathbf{S}_L\mathbf{e}_t\\
\tilde{\mathbf{A}}_t^{-1} &= \tilde{\mathbf{A}}_t^{-1} - \frac{\mathbf{a}_u\mathbf{a}_d}{1+\mathbf{d}_t^\top\mathbf{a}_u}\\
\tilde{\mathbf{b}}_t &= \tilde{\mathbf{b}}_t + \frac{r_{t+1}\mathbf{S}_L\mathbf{e}_t - \tilde{\mathbf{b}}_t}{t}\\
\mathbf{w} &= \tilde{\mathbf{A}}_t^{-1}\tilde{\mathbf{b}}_t
\end{aligned}$$

If $\mathbf{S}_L\mathbf{A}\mathbf{S}_R^\top$ is not square (e.g., $\mathbf{S}_R = \mathbf{I}$), we instead solve for $\mathbf{S}_L^\top\mathbf{S}_L\mathbf{A}\mathbf{S}_R^\top\mathbf{S}_R\mathbf{w} = \mathbf{S}_L^\top\mathbf{S}_L\mathbf{b}$, where applying $\mathbf{S}_L^\top$ provides the recovery from the left and $\mathbf{S}_R$ the recovery from the right.

Second, with the same sketching, we also experimented with $\mathbf{S}_L^\dagger$, instead of $\mathbf{S}_L^\top$ for the recovery, and similarly for $\mathbf{S}_R$, but this provided no improvement.

For this square system, the iterative update is

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha_t\mathbf{S}_L^\dagger(\tilde{\mathbf{b}}_{t+1} - \tilde{\mathbf{A}}_{t+1}\mathbf{S}_R\mathbf{w}_t)\\
&= \mathbf{w}_{t+1} + \alpha_t\mathbf{S}_L^\dagger(\mathbf{S}_L\mathbf{b}_{t+1} - \mathbf{S}_L\mathbf{A}_{t+1}\mathbf{S}_R^\top\mathbf{S}_R\mathbf{w}_t)
\end{aligned}$$

for the same $\tilde{\mathbf{b}}_t$ and $\tilde{\mathbf{A}}_t$ which can be efficiently kept incrementally, while the pseudoinverse of $\mathbf{S}_L$ only needs to be computed once at the beginning.

Third, we tried to solve the system $\mathbf{S}_L^\top\mathbf{S}_L\mathbf{A}\mathbf{w} = \mathbf{b}$, using the updating rule $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t(\mathbf{b}_{t+1} - \mathbf{S}_L^\top\mathbf{S}_L\mathbf{A}_{t+1}\mathbf{w}_t)$, where the matrix $\mathbf{S}_L\mathbf{A}_{t+1}$ can be incrementally maintained at each step by using a simple rank-one update.

Fourth, we tried to explicitly regularize these iterative updates by adding a small step in the direction of $\delta_t\mathbf{e}_t$.

In general, none of these iterative methods performed well. We hypothesize this may be due to difficulties in

choosing stepsize parameters. Ultimately, we found the sketched updated within ATD to be the most effective.

C Experimental details

Mountain Car is a classical episodic task with the goal of driving the car to the top of mountain. The state is 2-dimensional, consisting of the (position, velocity) of the car. We used the specification from (Sutton & Barto, 1998). We compute the true values of 2000 states, where each testing state is sampled from a trajectory generated by the given policy. From each test state, we estimate the value—the expected return—by computing the average over 1000 returns, generated by rollouts. The policy for Mountain Car is the energy pumping policy with 20% randomness starting from slightly random initial states. The discount rate is 1.0, and is 0 at the end of the episode, and the reward is always -1.

Puddle World Boyan & Moore (1995) is an episodic task, where the goal is for a robot in a continuous gridworld to reach a goal state within as fewest steps as possible. The state is 2-dimensional, consisting of (x, y) positions. We use the same setting as described in (Sutton & Barto, 1998), with a discount of 1.0 and -1 per step, except when going through a puddle that gives higher magnitude negative reward. We compute the true values from 2000 states in the same way as Mountain Car. A simple heuristic policy choosing the action leading to shortest Euclidean distance with 10% randomness is used.

Acrobot is a four-dimensional episodic task, where the goal is to raise an arm to certain level. The reward is -1 for non-terminal states and 0 for goal state, again with discount set to 1.0. We use the same tile coding as described in (Sutton & Barto, 1998), except that we use memory size $2^{15} = 32,768$. To get a reasonable policy, we used true-online Sarsa(λ) to go through 15000 episodes with stepsize $\alpha = 0.1/48$ and bootstrap parameter $\lambda = 0.9$. Each episode starts with a slight randomness. The policy is $\epsilon-$greedy with respect to state value and $\epsilon = 0.05$. The way we compute true values and generate training trajectories are the same as we described for the above two domains.

Energy allocation (Salas & Powell, 2013) is a continuing task with a five-dimensional state, where we use the same settings as in Pan et al. (2017). The matrix $\mathbf{A}$ was shown to have a low-rank structure (Pan et al., 2017) and hence matrix approximation methods are expected to perform well.

For radial basis functions, we used format $k(\mathbf{x}, \mathbf{c}) = \exp(-\frac{||\mathbf{x}-\mathbf{c}||_2^2}{2\sigma^2})$ where σ is called RBF width and $\mathbf{c}$ is a feature. On Mountain Car, because the position and velocity have different ranges, we set the bandwidth separately for each feature using $k(\mathbf{x}, \mathbf{c}) = \exp(-((\frac{\mathbf{x}_1-\mathbf{c}_1}{0.12r_1})^2 + (\frac{\mathbf{x}_2-\mathbf{c}_2}{0.12r_2})^2))$, where r_1 is the range of the first state variable and r_2 is the range of second state variable.

In Figure 4, we used a relatively rarely used representation which we call spline feature. For sample $\mathbf{x}$, the ith spline feature is set to 1 if $||\mathbf{x} - \mathbf{c}_i|| < \delta$ and otherwise set as 0. The centers are selected in exactly the same way as for the RBFs.

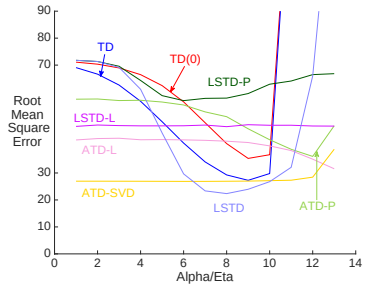
Parameter optimization. We swept the following ranges for stepsize (α), bootstrap parameter (λ), regularization parameter (η_t), and initialization parameter ξ for all domains:

1. $\alpha \in \{0.1 \times 2.0^j | j = -7, -6, ..., 4, 5\}$ divided by l_1 norm of feature representation, 13 values in total.
2. $\lambda \in \{0.0, 0.1, ..., 0.9, 0.93, 0.95, 0.97, 0.99, 1.0\}$, 15 values in total.
3. $\eta \in \{0.01 \times 2.0^j | j = -7, -6, ..., 4, 5\}$ divided by l_1 norm of feature representation, 13 values in total.
4. $\xi \in \{10^j | j = -5, -4.25, -3.5, ..., 2.5, 3.25, 4.0\}$, 13 values in total.

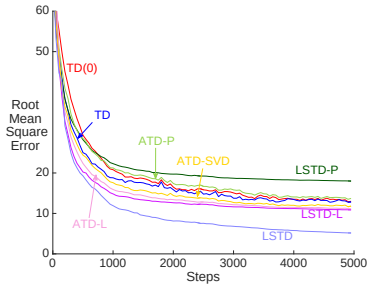
To choose the best parameter setting for each algorithm, we used the sum of RMSE across all steps for all the domains Energy allocation. For this domain, optimizing based on the whole range causes TD to pick an aggressive step-size to improve early learning at the expense of later learning. Therefore, for Energy allocation, we instead select the best parameters based on the sum of the RMSE for the second half of the steps.

For the ATD algorithms, as done in the original paper, we set $\alpha_t = \frac{1}{t}$ and only swept the regularization parameter η, which can also be thought of a (smaller) final stepsize. For this reason, the range of η is set to 0.1 times the range of α, to adjust this final stepsize range to an order of magnitude lower.

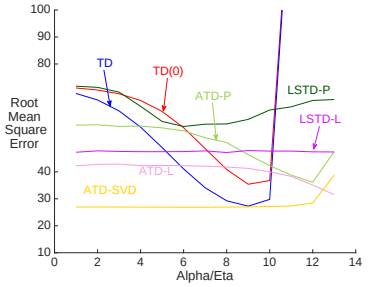
Additional experimental results. In the main paper, we demonstrated a subset of the results to highlight conclusions. For example, we showed the learning curves and parameter sensitivity in Mountain Car, for RBFs and tile coding. Due to space, we did not show the corresponding results for Puddle World in the main paper; we include these experiments here. Similarly, we only showed Acrobot with RBFs in the main text, and include results with tile coding here.



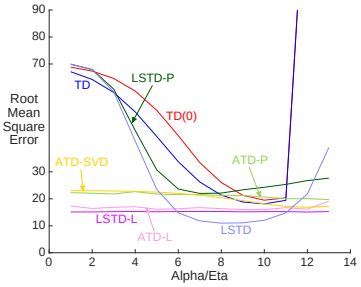
(a) Puddle World, tile coding, $k = 50$



(b) Puddle World, RBF, $k = 50$

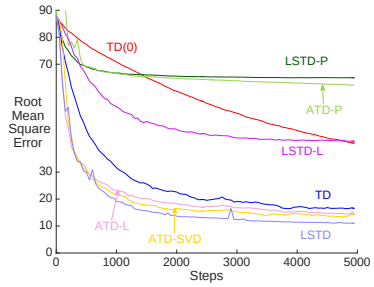


(c) Puddle World, tile coding, $k = 50$

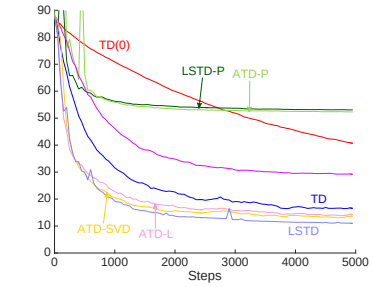


(d) Puddle World, RBF, $k = 50$

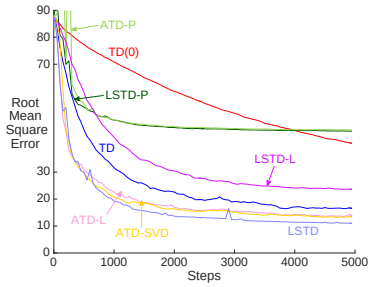
Figure 6: The two sensitivity figures are corresponding to the above two learning curves on Puddle World domain. Note that we sweep initialization for LSTD-P, but keep initialization parameter fixed across all other settings. The one-side projection is almost insensitive to initialization and the corresponding ATD version is insensitive to regularization. Though ATD-SVD also shows insensitivity, performance of ATD-SVD is much worse than sketching methods for the RBF representation. And, one should note that ATD-SVD is also much slower as shown in the below figures.



(a) Mountain Car, Tile coding, $k = 25$

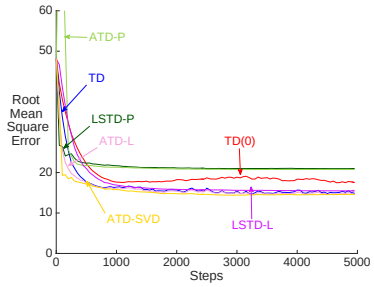


(b) Mountain Car, Tile coding, $k = 50$

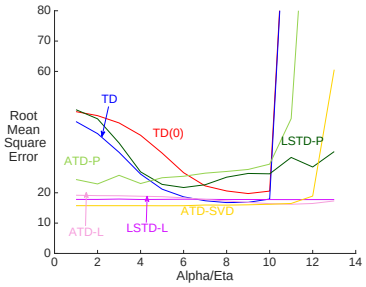


(c) Mountain Car, Tile coding, $k = 75$

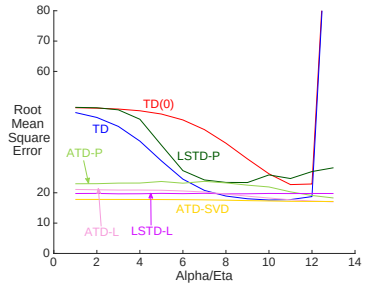
Figure 7: Change in performance when increasing k, from 25 to 75. We can draw similar conclusions to the same experiments in Puddle World in the main text. Here, the unbiased of ATD-L is even more evident; even with as low a dimension as 25, it performs similarly to LSTD.



(a) Acrobot, tile coding, $k = 50$



(b) Acrobot, tile coding, $k = 50$



(c) Acrobot, RBF, $k = 75$

Figure 8: Additional experiments in Acrobot, for tile coding with $k = 50$ and for RBFs with $k = 75$.