Accelerated Gradient Temporal Difference Learning

A presentation on the *paper* by Yangchen Pan, Adam White, Martha White

Y Rithvik
21090

K Kalyan Reddy
21361

G Ramesh Babu
20950

Content

- Introduction to the family of Temporal Difference Methods
- Linear TD method
- Least Squares TD method
- Comparison between linear TD and least squares TD methods
- Accelerated gradient TD (ATD) method
- Boyan's Chain environment
- Random Walk environment
- Implementation details
- Results
- Conclusion and Future work
- References

- Temporal Difference (TD) methods are a class of RL algorithms that update the estimated value function of the agent based on the observed rewards and transitions.
- TD methods can be categorized into linear methods like TD(λ), and more sophisticated least squares methods that directly compute the TD solution.

Overview

$$(\mathcal{S}, \mathcal{A}, \mathrm{P}, r)$$

$$\mu : \mathcal{S} \times \mathcal{A} \to [0, \infty)$$

$$R_{t+1} \stackrel{\text{def}}{=} r(S_t, A_t, S_{t+1})$$

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s]$$

$$\begin{aligned} G_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \dots \\ &= R_{t+1} + \gamma_{t+1} G_{t+1} \end{aligned}$$

$$x : \mathcal{S} \to \mathbb{R}^d \qquad \mathbf{x}_t \stackrel{\text{def}}{=} x(S_t)$$

$$v_\pi(S_t) \approx \mathbf{w}^\top \mathbf{x}_t$$

Linear TD(λ) method

- Estimates Value function directly from samples generated while interacting with the environment
- Instead of waiting until the end of a trajectory to update the value of each state, the TD(λ) algorithm adjusts its current estimate of the weights toward the difference between the discounted estimate of the value in the next state and the estimated value of the current state plus the reward along the way:

$$\delta_t \stackrel{\text{def}}{=} \delta(S_t, A_t, S_{t+1}) \stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\hat{v}_{\boldsymbol{w}}(S_{t+1}) - \hat{v}_{\boldsymbol{w}}(S_t)$$

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha\delta_t z_t$$

$$\boldsymbol{z}_t \leftarrow \gamma_t\lambda\boldsymbol{z}_{t-1} + \nabla\hat{v}_{\boldsymbol{w}}(S_t)$$

Least Squares TD method

The idea behind LSTD is to use the Bellman equation to derive a set of linear equations that can be solved using linear regression to obtain the coefficients of the value function approximation

Minimizing MSBPE ($\text{MSPBE}(\theta) = ||V_\theta - \Pi(TV_\theta)||^2_D$) is equivalent to solving $\mathbf{A}\mathbf{w} = \mathbf{b}$

where,

$$\mathbf{A} = \mathbf{X}^\top \mathbf{D}(\mathbf{I} - \gamma\lambda\mathbf{P}^\pi)^{-1}(\mathbf{I} - \gamma\mathbf{P}^\pi)\mathbf{X}$$
$$\mathbf{b} = \mathbf{X}^\top \mathbf{D}(\mathbf{I} - \gamma\lambda\mathbf{P}^\pi)^{-1}\mathbf{r}_\pi$$

Since, stationary distribution is not known, **A** and **b** are approximated as follows

$$\mathbf{A}_T = \frac{1}{T}\sum_{t=0}^{T-1} \mathbf{z}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top \quad \text{and} \quad \mathbf{b}_T = \frac{1}{T}\sum_{t=0}^{T-1} \mathbf{z}_t r_{t+1}$$

for eligibility trace $\mathbf{z}_t = \sum_{i=0}^{t}(\gamma\lambda)^{t-i}\mathbf{x}_i$ and sampled trajectory $s_0, a_0, r_1, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$.

Comparison between linear TD(λ) method and least squares TD method

1. Bias - Variance Trade off : Linear TD(λ) introduces bias in the value function estimates, while Least Squares TD can introduce high variance in the value function (overfitting)
2. Computational Cost: Linear TD(λ) is computationally cheaper than Least Squares TD
3. Memory Requirements: Linear TD(λ) requires less memory than Linear Least Squares TD
4. Parameter Tuning: Least Squares TD do not require tuning a typically highly sensitive learning rate parameter unlike Linear TD(λ)

Motivation

Accelerated TD learning aims to combine the benefits of linear TD(λ) and least squares TD to achieve faster convergence, improved stability, better generalization, and lower memory requirements.

By combining TD(λ) updates with least squares regression updates, accelerated TD learning can achieve a balance between bias and variance, capture both short-term and long-term dependencies in the data, and update the value function incrementally using a single weight vector.

Accelerated TD method

Our Objective is to minimize MSPBE

$$\text{MSPBE}(\mathbf{w}, m) = (\mathbf{b}_m - \mathbf{A}_m \mathbf{w})^\intercal \, \mathbf{C}^{-1} \, (\mathbf{b}_m - \mathbf{A}_m \mathbf{w})$$

where $m : \mathcal{S} \to [0, \infty)$ is a weighting function,

$$\mathbf{A}_m \stackrel{\text{def}}{=} \mathbb{E}_\mu[(\gamma_{t+1}\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \mathbf{w}\, \mathbf{e}_{m,t}]$$

$$\mathbf{b}_m \stackrel{\text{def}}{=} \mathbb{E}_\mu[R_{t+1}\mathbf{e}_{m,t}]$$

$$\mathbf{C} = \mathbb{E}_\mu[\mathbf{x}_t \mathbf{x}_t^\top]$$

$$\mathbf{e}_{m,t} \stackrel{\text{def}}{=} \rho_t(\gamma_{t+1}\lambda \mathbf{e}_{m,t-1} + M_t \mathbf{x}_t)$$

We will consider on policy algorithm, however we can extend this to off-policy with convergence guarantees using Emphatic weighting

To derive the new algorithm, we first take the gradient of the MSPBE

$$-\frac{1}{2}\nabla_{\mathbf{w}}\text{MSPBE}(\mathbf{w}, m) = \mathbf{A}_m^\top \mathbf{C}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$$

Consider a second order update by computing the Hessian: $\mathbf{H} = \mathbf{A}_m^\top \mathbf{C}^{-1}\mathbf{A}_m^\top$

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{\alpha_t}{2}\mathbf{H}^{-1}\nabla_{\mathbf{w}}\text{MSPBE}(\mathbf{w}, m) \\ &= \mathbf{w}_t + \alpha_t \mathbf{A}^{-1}\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]\end{aligned}$$

Computing the Hessian and updating w requires quadratic computation, and in practice quasi-Newton approaches are used that approximate the Hessian

Our objective is to improve on the sample efficiency of linear TD methods, while avoiding both quadratic computation and asymptotic bias

We need an approximation Â to A that provides useful information, but that is also sub-quadratic in storage and computation.

ATD algorithm proposes to achieve this by approximating $\mathbf{A}^{-1}$ and sampling $\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}] = \mathbf{b} - \mathbf{A}\mathbf{w}$ using $\delta_t(\mathbf{w}_t)\mathbf{e}_t$ as an unbiased sample

The proposed ATD(λ) update is:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (\alpha_t \hat{\mathbf{A}}_t^\dagger + \eta \mathbf{I})\delta_t \mathbf{e}_t$$

with expected update

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (\alpha_t \hat{\mathbf{A}}^\dagger + \eta \mathbf{I})\mathbb{E}_\mu[\delta_t(\mathbf{w})\mathbf{e}_{m,t}]$$

How to approximate $\mathbf{A}$?

Diagonal approximation: Storing and using a diagonal approximation would only require $O(d)$ linear time and space

Low Rank Approximation: For a low-rank approximation Â, of rank k, represented with truncated singular value decomposition $\hat{\mathbf{A}} = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$ the storage requirement is $O(dk)$ and the required matrix-vector multiplications are only $O(dk)$

Exploratory experiments revealed that the low-rank approximation approach significantly outperformed the diagonal approximation

Algorithm 1 Accelerated Temporal Difference Learning

▷ where $\mathbf{U}_0 = [], \mathbf{V}_0 = [], \mathbf{\Sigma}_0 = [], \mathbf{b}_0 = \mathbf{0}, \mathbf{e}_0 = \mathbf{0}$, initialized $\mathbf{w}_0$ arbitrarily

function ATD(k, η, λ)

$\mathbf{x}_0 =$ first observation

$\eta =$ a small final stepsize value, e.g., $\eta = 10e-4$

for $t = 0, 1, 2, ...$ **do**

In $\mathbf{x}_t$, select action $A_t \sim \pi$, observe $\mathbf{x}_{t+1}$, reward r_{t+1}, discount γ_{t+1} (could be zero if terminal state)

$\beta_t = 1/(t+1)$

$\delta_t = r_{t+1} + \gamma_{t+1}\mathbf{w}^\top \mathbf{x}_{t+1} - \mathbf{w}^\top \mathbf{x}_t$

$\mathbf{e}_t =$ TRACE_UPDATE($\mathbf{e}_{t-1}, \mathbf{x}_t, \gamma_t, \lambda_t$) ▷ or call EMPAHTIC_TRACE_UPDATE to use emphatic weighting

▷ $\mathbf{U}_t\mathbf{\Sigma}_t\mathbf{V}_t^\top = (1-\beta_t)\mathbf{U}_{t-1}\mathbf{\Sigma}_{t-1}\mathbf{V}_{t-1}^\top + \beta_t\mathbf{e}_t(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})^\top$

$[\mathbf{U}_t, \mathbf{\Sigma}_t, \mathbf{V}_t] =$ SVD-UPDATE($\mathbf{U}_{t-1}, (1-\beta_t)\mathbf{\Sigma}_{t-1}, \mathbf{V}_{t-1}, \sqrt{\beta_t}\mathbf{e}_t, \sqrt{\beta_t}(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1}), k$) ▷ see (Gehring et al. 2016)

▷ Ordering of matrix operations important, first multiply $\mathbf{U}_t^\top \mathbf{e}_t$ in $\mathcal{O}(dk)$ time

▷ to get a new vector, then by $\mathbf{\Sigma}_t^\dagger$ and $\mathbf{V}_t$ to maintain only matrix-vector multiplications

$\mathbf{w}_{t+1} = \mathbf{w}_t + (\frac{1}{t+1}\mathbf{V}_t\mathbf{\Sigma}_t^\dagger\mathbf{U}_t^\top + \eta\mathbf{I})(\delta_t\mathbf{e}_t)$ ▷ where $\mathbf{\Sigma}_t^\dagger = \text{diag}(\hat{\sigma}_1^{-1}, \ldots, \hat{\sigma}_k^{-1}, \mathbf{0})$

Algorithm 2 Conventional trace update for ATD

function TRACE_UPDATE($\mathbf{e}_{t-1}, \mathbf{x}_t, \gamma_t, \lambda_t$)
return $\gamma_t \lambda_t \mathbf{e}_{t-1} + \mathbf{x}_t$

Algorithm 3 Emphatic trace update for ATD

▷ where $F_0 \leftarrow 0, \quad M_0 \leftarrow 0$ is initialized globally, before executing the for loop in ATD(λ)

function EMPHATIC_TRACE_UPDATE($\mathbf{e}_{t-1}, \mathbf{x}_t, \gamma_t, \lambda_t$)
$\rho_t \leftarrow \frac{\pi(s_t, a_t)}{\mu(s_t, a_t)}$ ▷ Where $\rho_t = 1$ in the on-policy case
$F_t \leftarrow \rho_{t-1} \gamma_t F_{t-1} + i_t$ ▷ For uniform interest, $i_t = 1$
$M_t \leftarrow \lambda_t i_t + (1 - \lambda_t) F_t$

return $\rho_t (\gamma_t \lambda_t \mathbf{e}_{t-1} + M_t \mathbf{x}_t)$

Implementation

We have implemented the above-mentioned algorithms in different environments using python

Implementation Details

Environments :

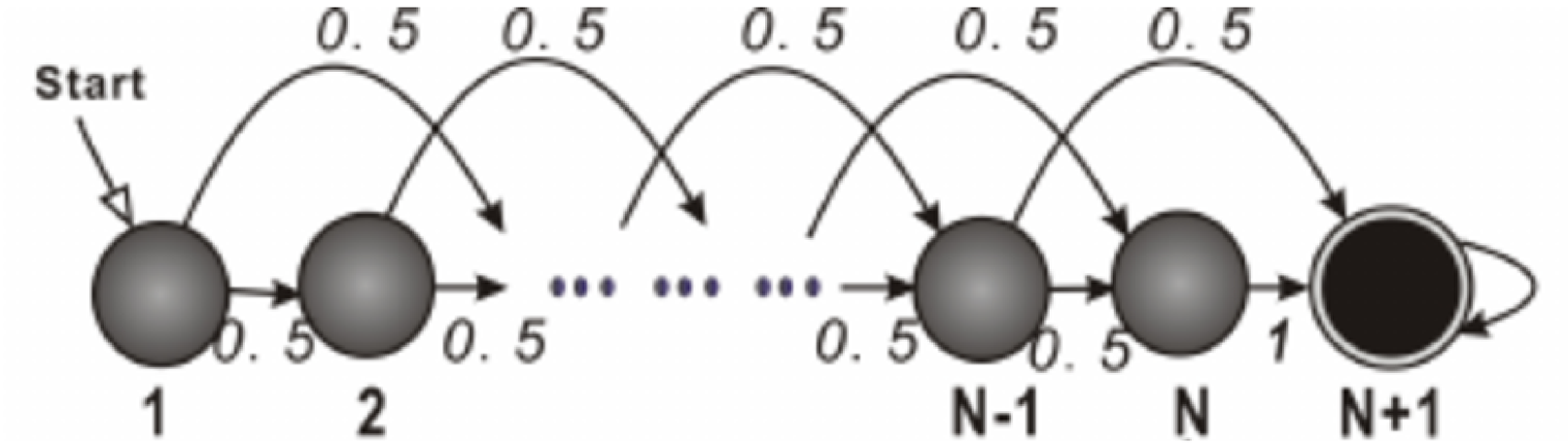
1) Boyan's Chain
2) Random Walk

Agents:

1) TDAgent
2) tLSTDAgent
3) PlainATDAgent
4) SVDATDAgent
5) SVDATDAgent with emphatic trace

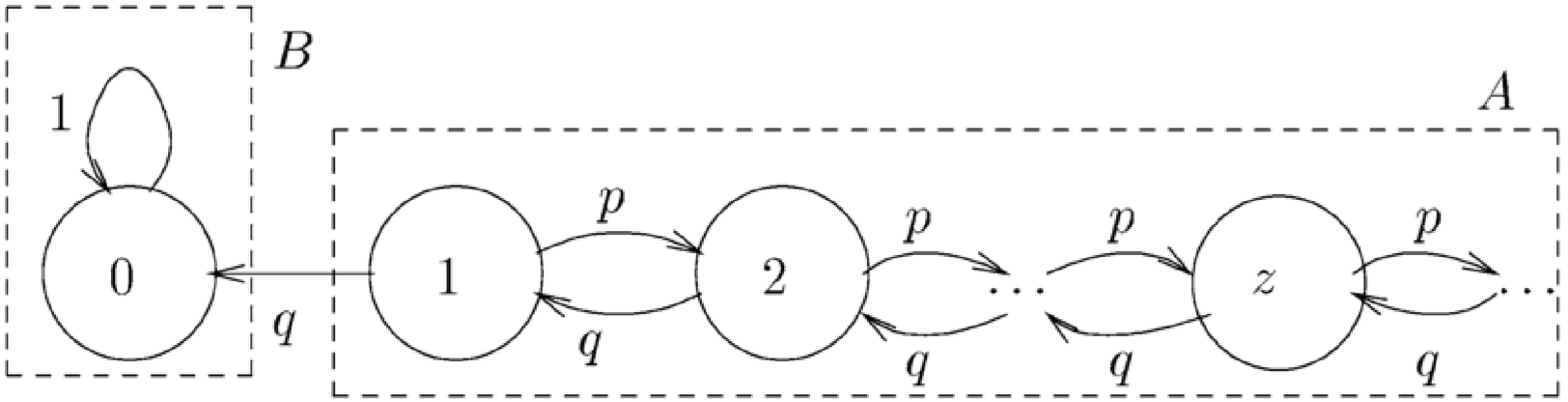
We compared the mentioned agents by computing the percentage error relative to the true value function over the first 1000 steps, averaging over 50 and 10 independent runs in boyan's chain and random walk environment respectively.

Boyans Chain



In out implemenation, we have taken N =13 , feature dimensions =4 for each state

Random Walk



In our implementation, we have used feature dimensions = 7

TD Agent

This is a normal TD(lambda) learner. Weights after each time step is update as follows

```
#Trace Update, lambd = 0 in this case
self.e = self.trace_update(self, observation, discount, self.e, self.lambd)
#trace_update -> discount * lambd * e + observation (return)

# Calculate the TD error
delta = reward + discount * self.w @ next_observation - self.w @ observation

# Updates the weight
self.w += self.lr * delta * self.e
```

SVD ATD Agent

```
beta = 1 / (t + 1)
delta = reward + discount * self.w @ next_observation - self.w @ observation
self.e = self.trace_update(self, observation, discount, self.e, self.lambd)
self.U, self.Sigma, self.V = \
self.svd_update(self.U,
                (1 - beta) * self.Sigma,
                self.V,
                sqrt(beta) * self.e.reshape((self.observation_space_n, 1)),
                sqrt(beta) * (observation - discount *
                              next_observation).reshape((self.observation_space_n, 1)))  # Uses SVD update to reduce the
complexity, enhancing the performance
self.w += (self.lr * Backend.linalg.pinv(self.U @ self.Sigma @ self.V.T, rcond=meta_data["rcond"]) +
                 self.eta * Backend.eye(self.observation_space_n)) @ (delta * self.e)
```

For SVD-Update, we have used the fast low rank approximation method , mentioned in 'Incremental Truncated LSTD' (Gehring et al 2016)

Plain ATD Agent

```
beta = 1 / (t + 1)
delta = reward + discount * self.w @ next_observation - self.w @ observation
self.e = self.trace_update(self, observation, discount, self.e, self.lambd)
self.A = (1 - beta) * self.A + beta * self.e.reshape((self.observation_space_n, 1)) \
            @ (observation - discount * next_observation).reshape((1, self.observation_space_n))
self.w += (self.lr * Backend.linalg.pinv(self.A, rcond=meta_data["rcond"]) + self.eta *
              Backend.eye(self.observation_space_n)) @ (delta * self.e)
```

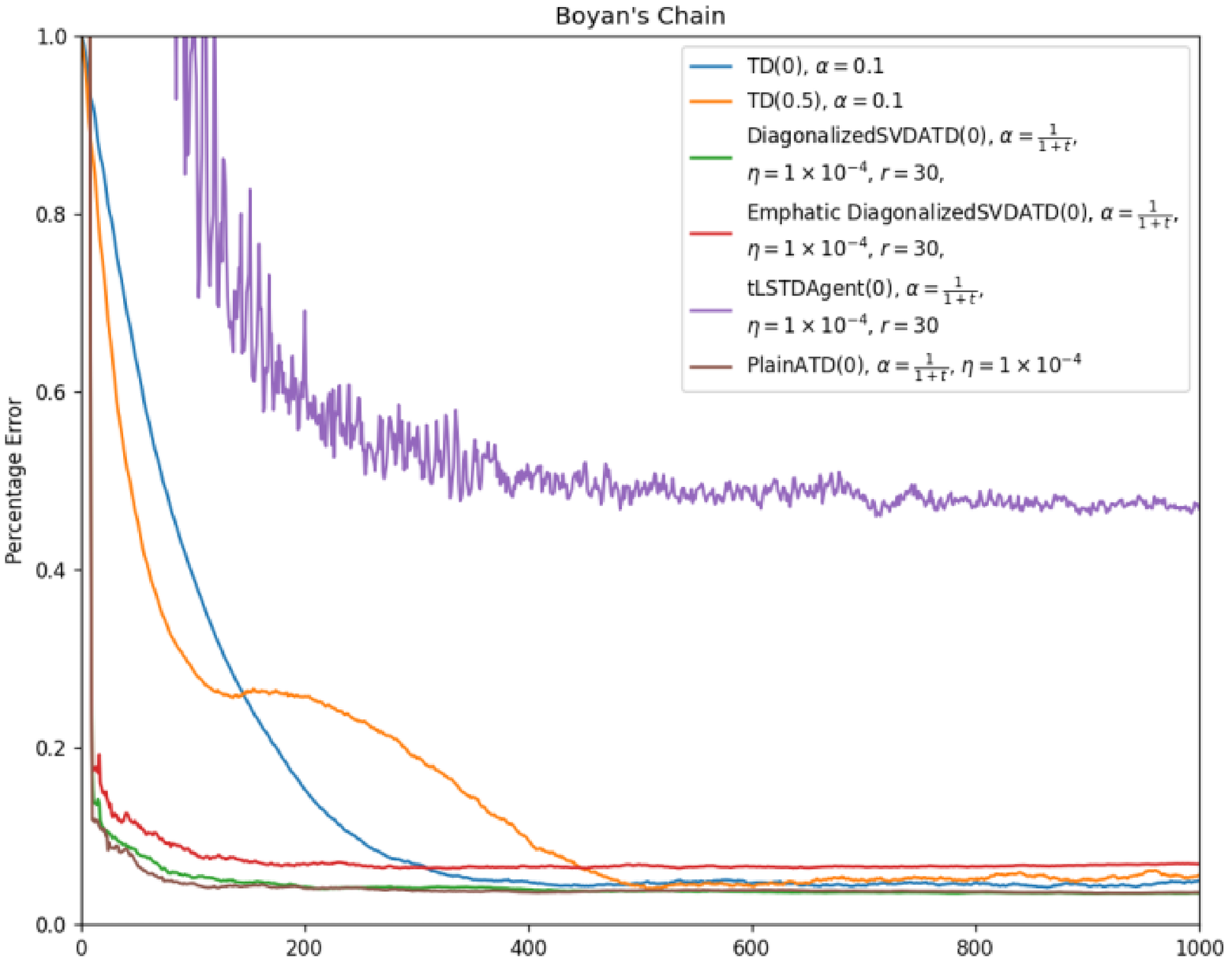
tLSTD Agent

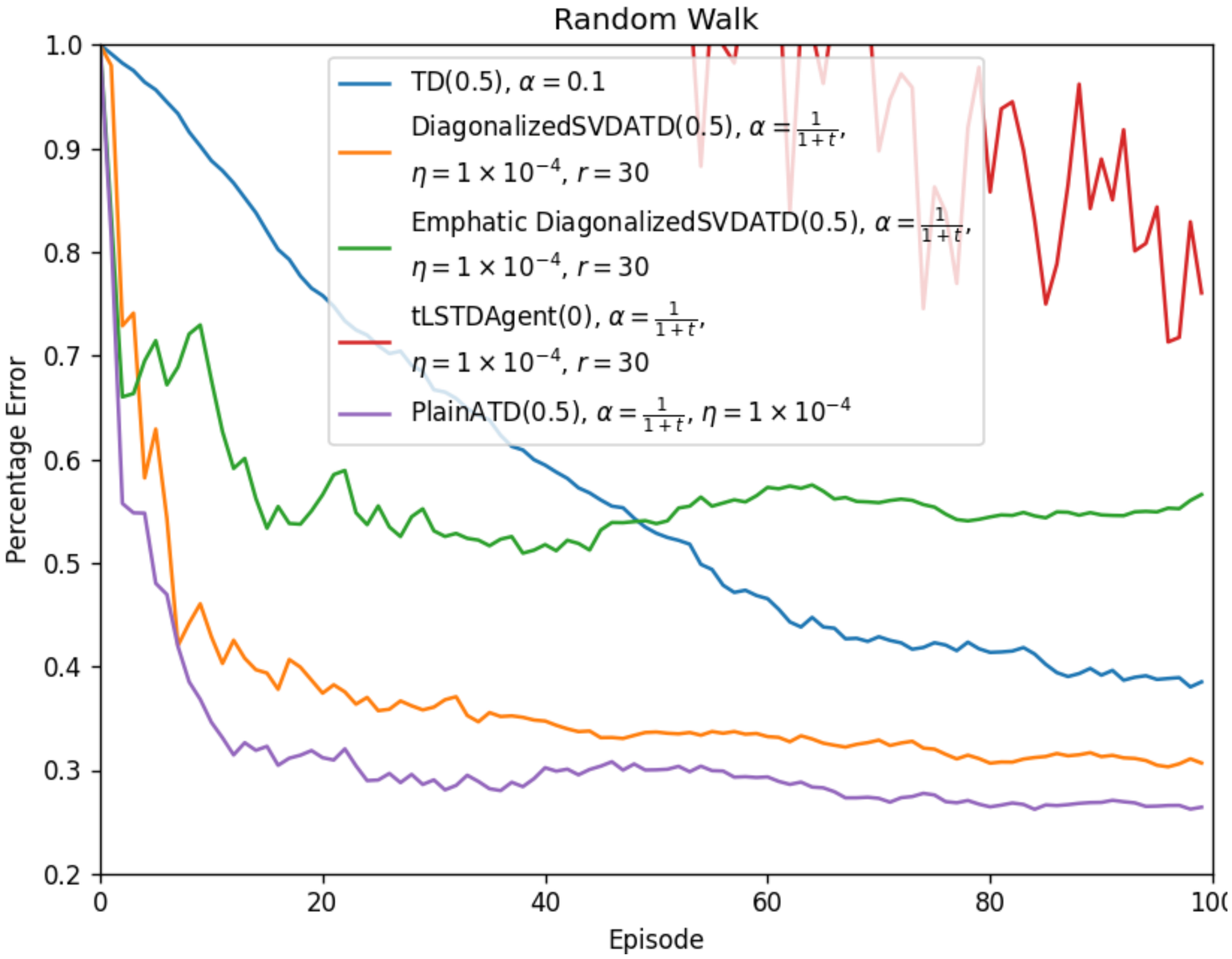
In this method, A matrix is approximated similar to SVD ATD agent, but weights are updated according to the following algorithm:

Algorithm 4 compute-weights($\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}, \mathbf{L}, \mathbf{R}, \mathbf{b}$), O($dr$)

// Solve $\mathbf{A}^{-1}\mathbf{b}$ where $\mathbf{A} = \mathbf{UL\Sigma R}^\top\mathbf{V}^\top$ and so $\mathbf{A}^{-1} = \mathbf{VR\Sigma}^{-1}\mathbf{L}^\top$
// Does not invert any singular values that are below $0.01 * \hat{\sigma}_1$
$\tilde{\mathbf{b}} = \mathbf{L}^\top\mathbf{U}^\top\mathbf{b}$ // O(dr) time, implicit left singular vector is $\mathbf{UL}$
$\hat{\sigma}_1 \leftarrow \mathbf{\Sigma}(1,1)$
$\mathbf{\Sigma}^\dagger \leftarrow \mathbf{0}$ // initialize as zero matrix
for $i \in \{1, \ldots, r\}$ **do**
    if $\mathbf{\Sigma}(i,i) > 0.01\hat{\sigma}_1$ **then**
        $\mathbf{\Sigma}^\dagger(i,i) \leftarrow \mathbf{\Sigma}(i,i)^{-1}$
    else
        break
    end if
end for
$\mathbf{w} = \mathbf{VR\Sigma}^{-1}\tilde{\mathbf{b}}$ // O(dr) time

Results


Boyan's Chain
Percentage Error
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
TD(0), $\alpha = 0.1$
TD(0.5), $\alpha = 0.1$
DiagonalizedSVDATD(0), $\alpha = \frac{1}{1+t}$, $\eta = 1 \times 10^{-4}$, $r = 30$,
Emphatic DiagonalizedSVDATD(0), $\alpha = \frac{1}{1+t}$, $\eta = 1 \times 10^{-4}$, $r = 30$,
tLSTDAgent(0), $\alpha = \frac{1}{1+t}$, $\eta = 1 \times 10^{-4}$, $r = 30$
PlainATD(0), $\alpha = \frac{1}{1+t}$, $\eta = 1 \times 10^{-4}$


Random Walk
Percentage Error
Episode
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
TD(0.5), $\alpha = 0.1$
DiagonalizedSVDATD(0.5), $\alpha = \frac{1}{1+t}$, $\eta = 1 \times 10^{-4}$, $r = 30$
Emphatic DiagonalizedSVDATD(0.5), $\alpha = \frac{1}{1+t}$, $\eta = 1 \times 10^{-4}$, $r = 30$
tLSTDAgent(0), $\alpha = \frac{1}{1+t}$, $\eta = 1 \times 10^{-4}$, $r = 30$
PlainATD(0.5), $\alpha = \frac{1}{1+t}$, $\eta = 1 \times 10^{-4}$

Conclusion and Future work

- The key idea of this Algorithm is to use a preconditioner on the temporal difference update, similar to a quasi-Newton stochastic gradient descent update
- It is proved that the expected update is consistent, and empirically demonstrated improved learning speed and parameter insensitivity, even with significant approximations in the preconditioner.
- This paper only uses a few of the possible preconditioners
- There remains many avenues to explore the utility of other preconditioners, such as diagonal approximations, eigenvalues estimates, other matrix factorizations and approximations to A that are amenable to inversion

References

- *Accelerated Gradient Temporal Difference Learning* by Yangchen Pan, Adam White and Martha White
- *Incremental Truncated LSTD* by Clement Gehring, Yangchen Pan and Martha White
- *A Generalized Projected Bellman Error for Off-policy Value Estimation in Reinforcement Learning* by Andrew Patterson, Adam White and Martha White
- *Linear Least-Squares Algorithms for Temporal Difference Learning* by Steven J Bradtke and Andrew G Barto

Thank You