Prediction with function approximation

Corrado Possieri

Machine and Reinforcement Learning in Control Applications

Introduction

- If the state space is large, then tabular methods are not implementable.
- Use function approximation
  - introduce weight vector $\mathbf{w} \in \mathbb{R}^d$, with $d \ll |\mathcal{S}|$;
  - approximate $v_\pi(s)$ with $\hat{v}(s, \mathbf{w})$.
- Adjusting $\mathbf{w}$ changes the value to several states.
- Makes RL applicable to partially observable MDPs.

Target update

- Several types of updates
  - MC update
    $$S_t \mapsto G_t;$$
  - TD(0) update
    $$S_t \mapsto R_{t+1} + \gamma V(S_{t+1});$$
  - n-step TD update
    $$S_t \mapsto G_{t:t+n};$$
  - DP update
    $$S_t \mapsto \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})|S_t].$$
- With function approximation $V(S_t)$ is substituted by
  $$\hat{v}(S_t, \mathbf{w}_t).$$
- Use *supervised learning* to mimic input-output relations
  - need to deal with non-stationary relations.

Prediction objective

- In tabular methods, an update at one state affected no other.
- Making a state's estimate better makes other less accurate.
- We need to specify which state we care most about
  - use a distribution μ such that $\mu(s) \geqslant 0$, $\sum_s \mu(s) = 1$;
  - often $\mu(s)$ is the fraction of time spent in s.
- The objective is to minimize the mean square value error

$$\overline{\mathrm{VE}}(\mathbf{w}) = \sum_s \mu(s) \left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)^2 .$$

- Often just a local optimum is achievable.

Gradient descent

- $\mathbf{w}$ is a vector of real valued numbers.
- $\hat{v}(s, \mathbf{w})$ is a real valued differentiable function.
- Assume that
  - states S_t are presented with distribution μ;
  - you can measure the true value $v_\pi(S_t)$.
- Update $\mathbf{w}$ using stochastic (one-sample) gradient descent

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \tfrac{1}{2}\alpha\nabla(v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t))^2 \\ &= \mathbf{w}_t + \alpha(v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t))\nabla\hat{v}(S_t, \mathbf{w}_t),\end{aligned}$$

where α is a small parameter and

$$\nabla\hat{v}(S_t, \mathbf{w}) = \begin{bmatrix} \frac{\partial\hat{v}(S_t,\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial\hat{v}(S_t,\mathbf{w})}{\partial w_d} \end{bmatrix}^\top.$$

Approximation of stochastic gradient descent

- Let U_t be a stochastic approximation of $v_\pi(S_t)$.
- The general SGD is then

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(U_t - \hat{v}(S_t, \mathbf{w}_t))\nabla\hat{v}(S_t, \mathbf{w}_t).$$

- If $\mathbb{E}[U_t|S_t = s] = v_\pi(s)$ and α satisfies the usual conditions
  - $\mathbf{w}_t$ converges to a local optimum.
- In general α has to be small since moving toward a single sample may be detrimental for others.

Gradient Monte Carlo algorithm

Gradient Monte Carlo algorithm

Input: $\alpha > 0$, $\hat{v}$

Output: approximate of v_π

Initialization

$\mathbf{w} \leftarrow$arbitrarily

Loop

generate an episode following π: $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$

$G \leftarrow 0$

for each step $t = T-1, T-2, \ldots, 0$ **do**

$G \leftarrow \gamma G + R_{t+1}$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha(G - \hat{v}(S_t, \mathbf{w}))\nabla\hat{v}(S_t, \mathbf{w})$

Bootstrapping with function approximation

- MC SGD converges since U_t is an unbiased estimate of v_π.
- By bootstrapping the target depends on $\mathbf{w}_t$
  - the target is biased;
  - it is not a true SGD.
- Account for changes on the estimate, but ignore changes on target caused by varied $\mathbf{w}$
  - usually referred to as *semi-gradient* methods.

Semi-gradient TD(0) algorithm

Semi-gradient TD(0) algorithm

Input: $\alpha > 0$, $\hat{v}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$
Output: approximate of v_π

Initialization

$\mathbf{w} \leftarrow$arbitrarily

Loop

initialize S
for each step of the episode **do**
$A \leftarrow \pi(\cdot|S)$
take action A and observe R, S'
$\mathbf{w} \leftarrow \mathbf{w} + \alpha(R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}))\nabla\hat{v}(S, \mathbf{w})$
$S \leftarrow S'$
if S is terminal **then**
reinitialize the episode

Linear methods

- The approximate function is **linear** with respect to $\mathbf{w}$.
- Associate to every $s \in \mathcal{S}$ a *feature vector*

$$\mathbf{x}(s) = \begin{bmatrix} x_1(s) & \cdots & x_d(s) \end{bmatrix}^\top.$$

- The approximate function is linear

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s).$$

- The SGD reduces to a simple form

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(U_t - \hat{v}(S_t, \mathbf{w}_t))\mathbf{x}(S_t).$$

Convergence of linear methods

- The MC algorithm converges to the global optimum of $\overline{\text{VE}}$.
- The TD algorithm update is

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t)\mathbf{x}_t \\ &= \mathbf{w}_t + \alpha(R_{t+1}\mathbf{x}_t - \mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top \mathbf{w}_t).\end{aligned}$$

- The expected next weight is

$$\mathbf{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t),$$

  where $\mathbf{b} = \mathbb{E}[R_{t+1}\mathbf{x}_t]$ and $\mathbf{A} = \mathbb{E}[\mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top]$.

- If TD converges, it converges to

$$\mathbf{w}_{\text{TD}} = \mathbf{A}^{-1}\mathbf{b}.$$

Notes on linear TD methods

- It can be proved that

$$\overline{\mathrm{VE}}(\mathbf{w}_{\mathrm{TD}}) \leqslant \frac{1}{1-\gamma} \min_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}).$$

- Estimate may be biased.
- Variance is usually reduced with respect to MC.
- Can be extended to the n-step case.

Least squares temporal-difference

- Compute estimates of $\mathbf{A}$ and $\mathbf{b}$ and directly compute $\mathbf{w}_{\mathrm{TD}}$

$$\begin{aligned}\hat{\mathbf{A}}_t &= \textstyle\sum_{k=0}^{t-1} \mathbf{x}_k(\mathbf{x}_k - \gamma\mathbf{x}_{k+1})^\top + \epsilon I,\\ \hat{\mathbf{b}}_t &= \textstyle\sum_{k=0}^{t-1} R_{k+1}\mathbf{x}_k,\\ \hat{\mathbf{w}}_t &= \hat{\mathbf{A}}_t^{-1}\hat{\mathbf{b}}_t.\end{aligned}$$

- Note that, letting $\hat{\mathbf{A}}_0^{-1} = \epsilon^{-1}\mathbf{I}$,

$$\hat{\mathbf{A}}_t^{-1} = \hat{\mathbf{A}}_{t-1}^{-1} - \frac{\hat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_{t-1}(\mathbf{x}_{t-1} - \gamma\mathbf{x}_t)^\top \hat{\mathbf{A}}_{t-1}^{-1}}{1 + (\mathbf{x}_{t-1} - \gamma\mathbf{x}_t)^\top \hat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_{t-1}}.$$

- Complexity $O(d^2)$ vs $O(d)$.

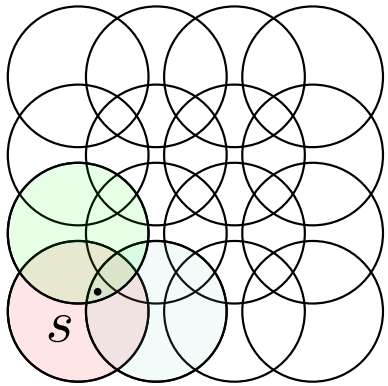
- Does not require a step size.

Feature selection

- Choosing features appropriate to the task is an important way of adding prior domain knowledge.
- The features should correspond to the aspects of the state space along which generalization may be appropriate.
- Linear forms cannot take into account any interactions between features if coded separately.

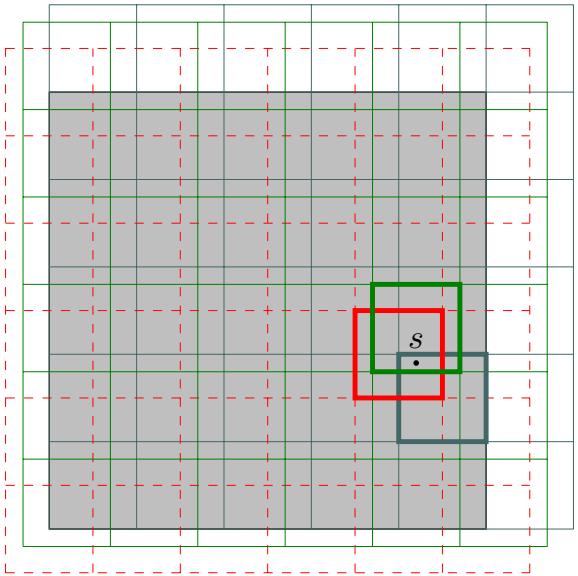
Features

- Polynomials: $s_1, s_2, s_1^2, s_2^2, s_1 s_2, \ldots$
  - grows exponentially with the dimension of the state space.
- Fourier basis: $\cos(i\pi s)$, $s \in [0, 1]$
  - useful for periodic functions or bounded domains;
  - if approximating over $[0, \frac{\tau}{2}]$, you can just use cosines;
  - grows exponentially with the dimension of the state space.
- Coarse coding: features $\mathbf{x} \in \{0, 1\}$
  - each circle has a corresponding weight;
  - the weights of all circles intersecting a state are affected;
  - large feature give broad generalizations;
  - acuity is controlled by the total number of features.



Tile coding

- Use partitions of the state space
  - *e.g.*, grids.
- Overlapping receptive fields
  - tiles of a partition do not overlap.
- Exactly one feature in each tiling.
- Best generalization if offset asymmetrically.
- It may be desirable to use different shaped tiles in different tilings.
- Hashing may reduce the curse of dimensionality.



Radial basis functions

- Generalization of coarse coding.
- Feature can be anything in the interval $[0, 1]$.
- Usually Gaussian shaped functions

$$x_i(s) = \exp\left(-\frac{\|s - c_i\|^2}{\sigma_i^2}\right).$$

- Differentiable approximate functions.
- Greater computational complexity.

Step-size

- Theoretically the step-size should decrease slowly

$$\alpha_t = \frac{1}{t}.$$

- A good rule of thumb for setting the step-size parameter is

$$\alpha = (\tau \mathbb{E}[\mathbf{x}^\top \mathbf{x}])^{-1},$$

where $\mathbf{x} \simeq \mu$.

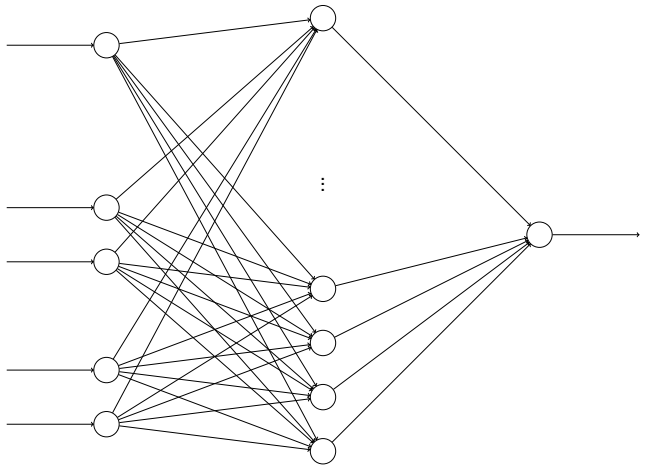
- In tile coding, a reasonable choice is

$$\alpha = \frac{1}{10n},$$

where n is the number of tiles.

Feedforward neural networks

- No loops in the network.
- A real-valued weight is associated with each link.
- The units compute a weighted sum of their inputs and then apply their *activation function*.
- With one hidden layer, universal approximation.
- Deeper architectures are capable of more general abstraction.
- Gradient computed via back-propagation.



Memory-based function approximation

- Non-parametric approach.
- Store samples as they are gathered
  - without updating any parameter.
- Combine samples to output estimated values for query state
  - retrieve samples from memory whose states are judged to be the most relevant to the query state;
  - local approximation discarded after evaluation.
- The simplest forms are
  - nearest neighborhood: return value of closest state;
  - weighted average: locally fits a function.
- Approximation improves as more samples are gathered
  - but requires particular data structures.
- Do not rely on particular functional forms.

Kernel-based methods

- In weighted average, distance is used to measure influence.
- We can use a *kernel* $k(s, s')$ to measure the strength of generalization from s to s', *e.g.*,

$$k(s, s') = \exp\left(-\frac{\|s - s'\|^2}{\sigma^2}\right).$$

- Letting $\mathcal{D}$ be the stored data and g the corresponding value

$$\hat{v}(s, \mathcal{D}) = \sum_{s' \in \mathcal{D}} g(s') k(s, s').$$

- The kernel $k(s, s') = \mathbf{x}^\top(s)\mathbf{x}(s')$ produces the same approximation of the corresponding linear method.

Adapting eligibility traces to function approximation

- Eligibility traces can be used with function approximation.
- In the off-line λ-return algorithm, at the end of the episode, a sequence of off-line updates are made according to

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}_t))\nabla\hat{v}(S_t, \mathbf{w}_t).$$

- Eligibility traces are vectors $\mathbf{z} \in \mathbb{R}^d$ that constitute a short-term memory.
- They affect the weight vector, and then the weight vector determines the estimated value.

TD(λ) with function approximation

- In TD(λ), $\mathbf{z}$ is updated as

$$\begin{aligned}\mathbf{z}_{-1} &= 0,\\ \mathbf{z}_t &= \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{v}(S_t, \mathbf{w}_t).\end{aligned}$$

- The TD error for state-value prediction is

$$\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t).$$

- The weight vector is updated as

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta\mathbf{z}_t.$$

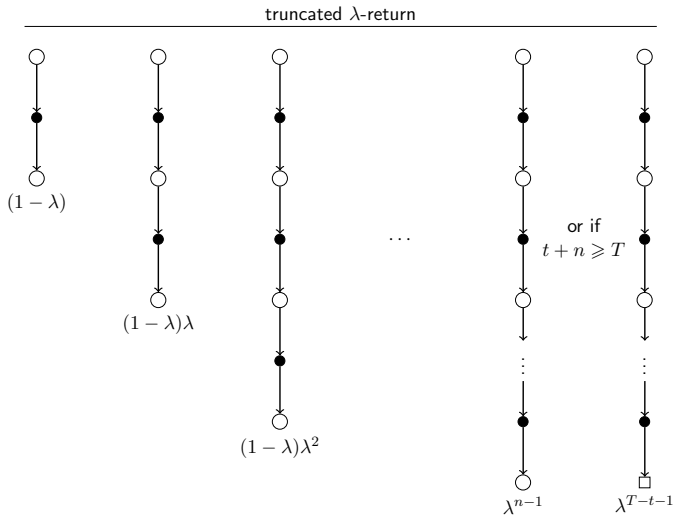
- It can be proved that linear TD(λ) converges to $\mathbf{w}_\infty$ such that

$$\overline{\mathrm{VE}}(\mathbf{w}_\infty) \leqslant \frac{1-\gamma\lambda}{1-\gamma}\min_{\mathbf{w}}\overline{\mathrm{VE}}(\mathbf{w}).$$

Truncated λ-return

- The λ-return is not known until the end of the episode.
- Dependence is weaker for delayed rewards, falling by $\gamma\lambda$.
- Define the *truncated λ-return*

$$G^{\lambda}_{t:h} = (1-\lambda) \sum_{n=1}^{t-h-1} \lambda^{n-1} G_{t:t+n} + \lambda^{t-h-1} G_{t:h}.$$



Notes on the truncated λ-return

- Updates are delayed by n steps and only take into account the first n rewards.
- All the k-step returns are included for $1 \leqslant k \leqslant n$.
- The longest component update is at most n steps.
- TTD(λ) is then defined as

$$\mathbf{w}_{t+n} = \mathbf{w}_{t+n-1} + \alpha \left(G^{\lambda}_{t:t+n} + \hat{v}(S_t, \mathbf{w}_{t+n-1}) \right) \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}),$$

where

$$G^{\lambda}_{t:t+k} = \hat{v}(S_t, \mathbf{w}_{t-1}) + \sum_{i=t}^{t+k-1} (\gamma\lambda)^{i-t} \delta'_i,$$

$$\delta'_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_{t-1}).$$

Redoing updates

- In order to obtain an on-line algorithm, we must redo updates.
- Always use the latest horizon
  - in each pass over the episode, use a slightly longer horizon.
- Given h steps of the episode, the update at $0 \leqslant t \leqslant h \leqslant T$ is

$$\mathbf{w}_{t+1}^h = \mathbf{w}_t^h + \alpha(G_{t:h}^\lambda - \hat{v}(S_t, \mathbf{w}_t^h))\nabla\hat{v}(S_t, \mathbf{w}_t)$$

- This update, together with

$$\mathbf{w}_t = \mathbf{w}_t^t$$

defines the *online λ-return algorithm*.

Updates in the online λ-return algorithm

- $\mathbf{w}_0^h$ are the weights from the end of the previous episode.
- The updates at each step of the episode are

$$
\begin{aligned}
h = 1: \quad & \mathbf{w}_1^1 = \mathbf{w}_0^1 + \alpha(G_{0:1}^\lambda - \hat{v}(S_0, \mathbf{w}_0^1))\nabla\hat{v}(S_0, \mathbf{w}_0^1), \\
h = 2: \quad & \mathbf{w}_1^2 = \mathbf{w}_0^2 + \alpha(G_{0:2}^\lambda - \hat{v}(S_0, \mathbf{w}_0^2))\nabla\hat{v}(S_0, \mathbf{w}_0^2), \\
& \mathbf{w}_2^2 = \mathbf{w}_1^2 + \alpha(G_{1:2}^\lambda - \hat{v}(S_1, \mathbf{w}_1^2))\nabla\hat{v}(S_1, \mathbf{w}_1^2), \\
h = 2: \quad & \mathbf{w}_1^3 = \mathbf{w}_0^3 + \alpha(G_{0:3}^\lambda - \hat{v}(S_0, \mathbf{w}_0^3))\nabla\hat{v}(S_0, \mathbf{w}_0^3), \\
& \mathbf{w}_2^3 = \mathbf{w}_1^3 + \alpha(G_{1:3}^\lambda - \hat{v}(S_1, \mathbf{w}_1^3))\nabla\hat{v}(S_1, \mathbf{w}_1^3) \\
& \mathbf{w}_3^3 = \mathbf{w}_2^3 + \alpha(G_{2:3}^\lambda - \hat{v}(S_2, \mathbf{w}_2^3))\nabla\hat{v}(S_2, \mathbf{w}_2^3).
\end{aligned}
$$

True online TD(λ)

- The online λ-return algorithm generates the weights

$$\begin{array}{lccccc}
h=0 & \mathbf{w}_0^0 & & & & \\
h=1 & \mathbf{w}_0^1 & \mathbf{w}_1^1 & & & \\
h=2 & \mathbf{w}_0^2 & \mathbf{w}_1^2 & \mathbf{w}_2^2 & & \\
h=3 & \mathbf{w}_0^3 & \mathbf{w}_1^3 & \mathbf{w}_2^3 & \mathbf{w}_3^3 & \\
 & \vdots & \vdots & \vdots & \vdots & \ddots \\
h=T & \mathbf{w}_0^T & \mathbf{w}_1^T & \mathbf{w}_2^T & \mathbf{w}_3^T & \cdots \quad \mathbf{w}_T^T
\end{array}$$

- With linear approximation, the *true online TD(λ)* algorithm

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t + \alpha(\mathbf{w}_t^\top\mathbf{x}_t - \mathbf{w}_{t-1}^\top\mathbf{x}_t)(\mathbf{z}_t - \mathbf{x}_t),$$

where

$$\mathbf{z}_t = \gamma\lambda\mathbf{z}_{t-1} + (1 - \alpha\gamma\lambda\mathbf{z}_{t-1}^\top\mathbf{x}_t)\mathbf{x}_t,$$

produce exactly the same sequence of weight vectors.

True online TD(λ) algorithm

True online TD(λ) algorithm

Input: $\alpha > 0$, $\lambda > 0$, basis $\mathbf{x}$ such that $\mathbf{x}(\text{terminal}) = 0$, policy π
Output: approximate of v_π

Initialization

$\mathbf{w} \leftarrow$arbitrarily

Loop

initialize S and obtain initial feature vector $\mathbf{x}$
$\mathbf{z} \leftarrow 0$
$V_{\text{old}} \leftarrow 0$
for each step of the episode **do**
  $A \leftarrow \pi(\cdot|S)$
  take action A and observe $R, \mathbf{x}'$
  $V \leftarrow \mathbf{w}^\top \mathbf{x}$
  $V' \leftarrow \mathbf{w}^\top \mathbf{x}'$
  $\delta \leftarrow R + \gamma V' - V$
  $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + (1 - \alpha\gamma\lambda\mathbf{z}^\top \mathbf{x})\mathbf{x}$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + V - V_{\text{old}})\mathbf{z} - \alpha(V - V_{\text{old}})\mathbf{x}$
  $V_{\text{old}} \leftarrow V'$
  $\mathbf{x} \leftarrow \mathbf{x}'$
  if $\mathbf{x}' = 0$ **then**
    reinitialize the episode