# Part II: Approximate Solution Methods

- Issues with tabular methods for arbitrarily large state spaces:
  - Memory requirements
  - No generalization
- Solution = function approximation from supervised learning with the additional problems:
  - non-stationarity (target function changes over time)
  - bootstrapping (require incremental updates)
  - delayed targets

# On-policy Prediction with Approximation

- Parameterized state-value function $v(s, \boldsymbol{w}) \approx v_\pi(s)$
  - Where $\boldsymbol{w}$ could be vector feature weights for a linear function, decision tree split points and leaf values or ANN weights
  - Each update is a training example to produce an estimate value function with the aim to reduce the approximation error
- Typically, $\boldsymbol{w} \ll |S|$ so changing one weight changes the estimated value of many states = generalisation
- Also makes RL applicable to partially observable problems

# Prediction objection: mean squared value error

- Mean error for each state (difference between the approximate value and the true value) weighted by the state distribution $\mu(s)$

$$\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \Big[ v_\pi(s) - \hat{v}(s,\mathbf{w}) \Big]^2 .$$

- $\mu(s)$ often chosen to be the fraction of time spent in state $s$, aka the on-policy distribution
  - Continuous tasks: stationary distribution under $\pi$
  - Episodic tasks: fraction of time spent in each state $\eta(s)$ normalized to sum to one

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}, \quad \text{for all } s \in \mathcal{S}.$$

- Goal is to minimise $\overline{VE}$, ideally globally but typically local optimum but no guarantees of convergence (some even diverge)

# Stochastic-gradient decent

- SGD for learning value function which is differentiable w.r.t $\boldsymbol{w}$
- Assuming states appear in examples with distribution $\mu(s)$ then minimise error on the observed examples, i.e. SGD

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]^2 = \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t),$$

  - SGD takes a small step in the direction of the gradient so as to balance the error in all states. Local convergence with decreasing step size

# Semi-gradient Methods

- Now consider the case where the target output is not the true value but an approximation $U_t$ (i.e. bootstrapping target)

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

- If $U_t$ is an unbiased (e.g. Monte Carlo target $G_t$) then guaranteed to converge to a local optimum
- However, if a bootstrapping target is used which is biased then not true gradient decent (as the target ignores the effect of changing $\boldsymbol{w}$) but known as semi-gradient methods
  - Same convergence guarantees are not obtained but do get benefits of bootstrapping (faster learning and online learning)

## Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using $\pi$
    Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

**Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$**

Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action $A$, observe $R, S'$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until $S$ is terminal

# Linear methods

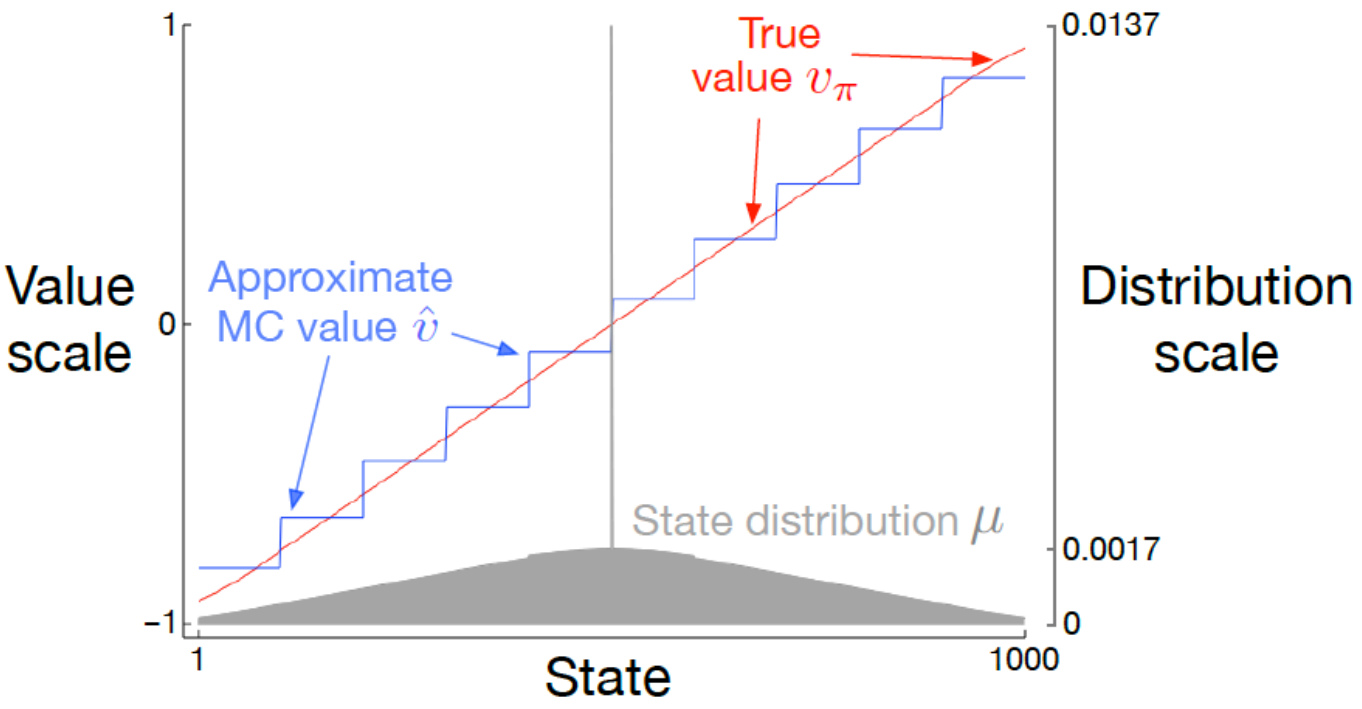
- Value function is a linear combination of weights and feature vector (basis functions)

$$\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s).$$

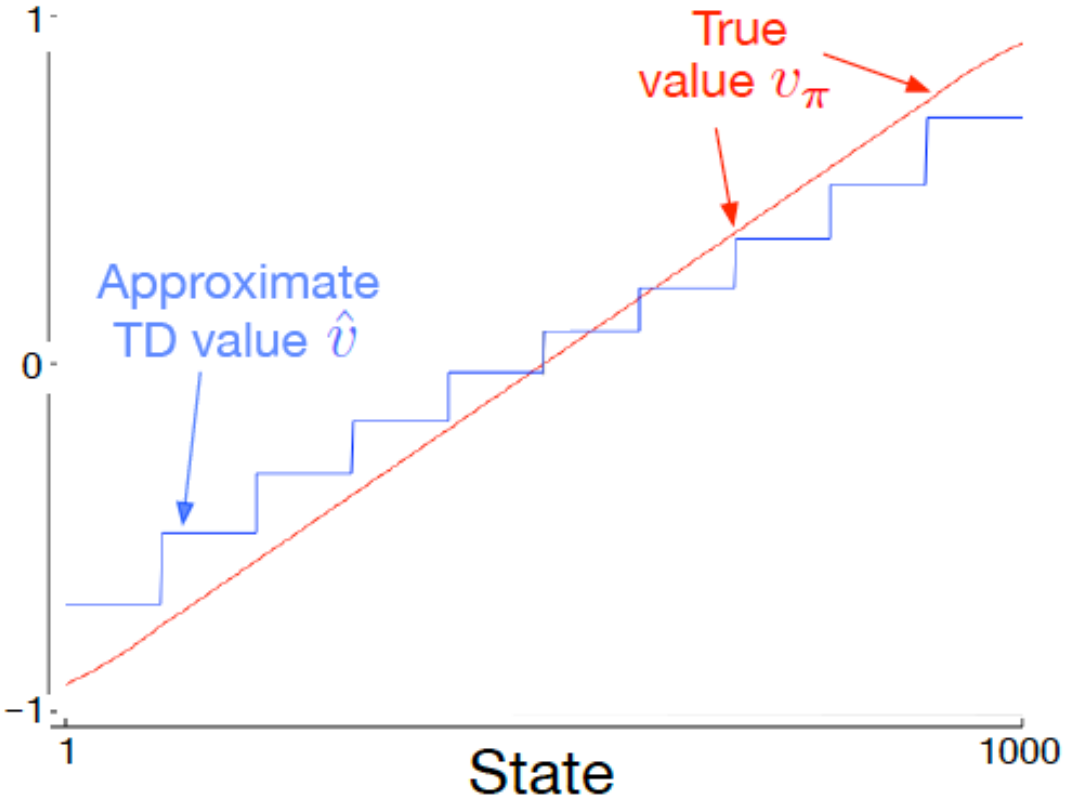
- SGD update: $\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[ U_t - \hat{v}(S_t,\mathbf{w}_t) \Big] \mathbf{x}(S_t).$
- For on-policy updates MC converges to global optimum (SGD with decaying $\alpha$) whilst TD(0) converges to near a local optimum (TD fixed point which has a bounded error)

# Random walk with state aggregation
# 1000 states, 10 groups

## Gradient Monte-Carlo

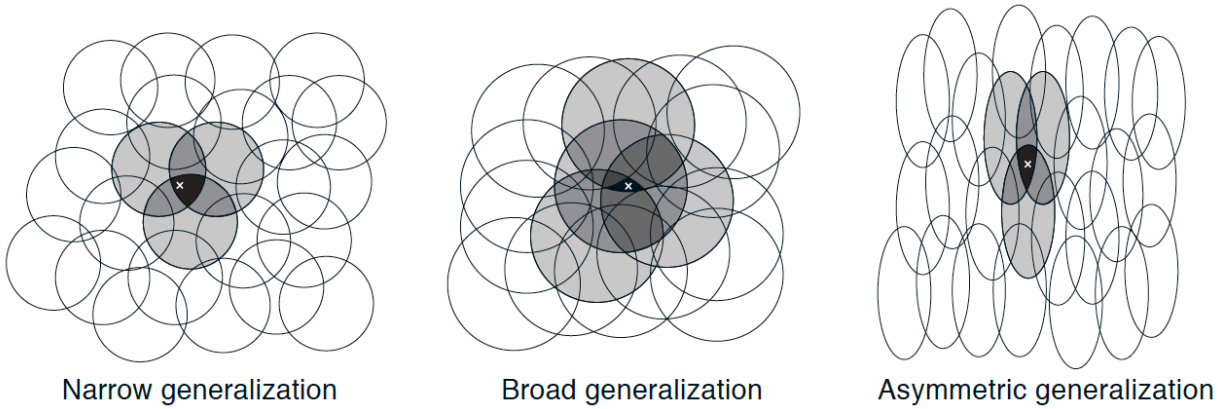


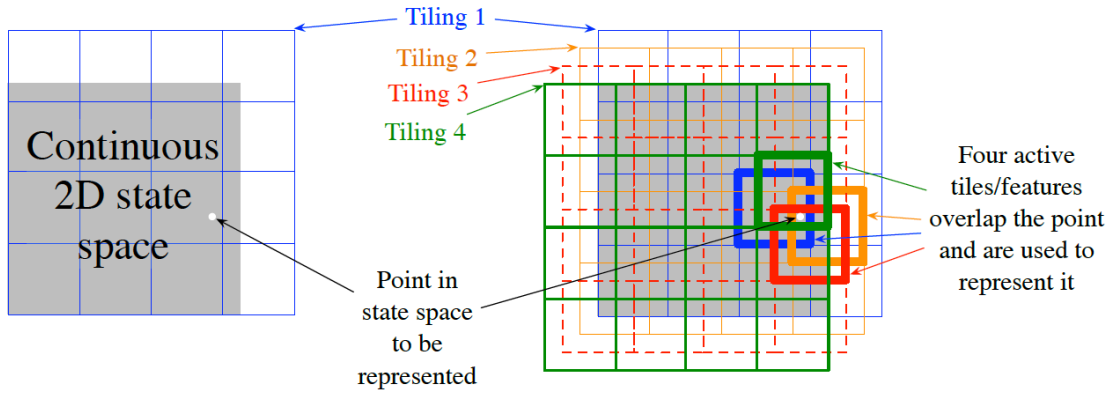

## Semi-gradient TD(0)



# Feature Construction

- Choosing features that represent the states is a way to add domain knowledge and allows appropriate generalization
- Potential linear basis functions:
  - Polynomials
  - Fourier basis
  - Coarse coding
  - Tile coding
  - Radial basis functions
- Nonlinear FA using ANN
  - Automatic production of hierarchical representations



# Least-squares TD

- TD(0) with linear FA converges asymptotically, LSTD estimates A and b

$$\mathbf{w}_{\mathrm{TD}} = \mathbf{A}^{-1}\mathbf{b}, \quad \widehat{\mathbf{A}}_t \doteq \sum_{k=0}^{t-1} \mathbf{x}_k(\mathbf{x}_k - \gamma \mathbf{x}_{k+1})^\top + \varepsilon \mathbf{I} \quad \text{and} \quad \widehat{\mathbf{b}}_t \doteq \sum_{k=0}^{t-1} R_{k+1}\mathbf{x}_k,$$

- Data efficient but computationally expensive
  - $O(d^2)$ compared to semi-gradient TD $O(d)$
- Does not require a step size parameter (however $\varepsilon$ is required) but this means it never forgets which is problematic if the target policy changes

# Memory-based Function Approximation

- As opposed to parametric approaches, these nonparametric approaches save training examples in memory which are recalled to make a query
- Local learning approximates a value function in the neighborhood of the query state, e.g. nearest neighbor, weighted average, locally weighted regression
  - Useful for trajectory sampling focusing on local states addressing curse of dimensionality but speed of queries degrades with memory size (solutions include k-d trees and forgetting entries)
- Kernel function is a measure of similarity between states (distances, strength of generalization)