RL 13: Algorithms for Large State Spaces

Michael Herrmann

University of Edinburgh, School of Informatics

04/03/2014

Overview

- Algorithms for large state spaces
- Basis functions
- Reformulation of algorithms in terms of gradients

Large State Spaces

- Grid-world algorithms: $V(s)$ is a vector, $\mathcal{Q}(s, a)$ a matrix
- In large problems is the complexity often beyond practical limits
  - storage space
  - exploration time
  - convergence time
- Generalisation and flexibility is low

Large State Spaces

- **Alternative:** Represent (e.g.) the value function in the form

$$V_\theta(x) = \theta^\top \varphi(x) = \sum_{i=1}^{N} \theta_i \varphi_i(x)$$

  where $x \in \mathbb{R}^D$ denotes the state of the system, $\theta \in \mathbb{R}^N$, and $\varphi : \mathbb{R}^D \to \mathbb{R}^N$ with $\varphi(x) = (\varphi_1(x), \ldots, \varphi_N(x))^\top$
- Includes the look-up table representation for $\theta_s = V(s)$ and

$$\varphi_s(x) = \begin{cases} 1 & \text{if } \mathrm{int}(x) = s \\ 0 & \text{otherwise} \end{cases}$$

- Many other choices for the basis functions φ are possible.

Feature spaces

$$V_\theta(x) = \theta^\top \varphi(x)$$

- Linear (weighted) sum of non-linear functions
- Can be universal function approximators (RBF network)
- $\theta \in \mathbb{R}^N$
  - parameter vector or weight vector
  - carries the information about the current estimate of the value function
- $\varphi : \mathcal{X} \to \mathbb{R}^N$
  - $\varphi(x) = (\varphi_1(x), \ldots, \varphi_N(x))^\top$
  - $\varphi_i : \mathcal{X} \to \mathbb{R}$ is a basis function
  - $\varphi_i(x)$: a feature of the state x
  - Examples: polynomial, wavelets, RBF, ...
- mathematically convenient: easily differentiable $\Rightarrow$ gradient

Radial basis functions

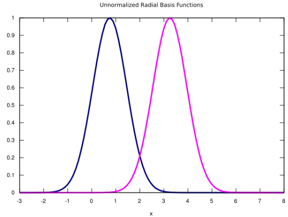
For a function $f : \mathbb{R}^D \to \mathbb{R}$ choose parameters such that

$$f(x) \approx \theta^\top \varphi(x)$$

with $\varphi : \mathbb{R}^D \to \mathbb{R}^N$, e.g.

$$\varphi_i(x) = \exp\left(-\frac{\|x - x^{(i)}\|^2}{2\sigma^2}\right)$$

with $i = 1, \ldots, N$.



example:
$N = 2$, $\theta = (1, 1)$

Determine $\theta \in \mathbb{R}^N$ by

$$\|f - \theta^\top \varphi\| \to \min$$

Solution: see e.g.
http://en.wikipedia.org/wiki/Radial_basis_function_network#Training

Factorial features: Tensor product construction*

Suppose $\mathcal{X} \subset \mathcal{X}_1 \times \mathcal{X}_2 \cdots \times \mathcal{X}_k$ (e.g. input from k sensors)

Let $\varphi^{(m)} : \mathcal{X}_m \to \mathbb{R}^{d_m}$ define d_m features for m-th component of $x \in \mathcal{X}$, $1 \leq m \leq k$

Tensor product $\varphi = \varphi^{(1)} \otimes \varphi^{(2)} \otimes \cdots \otimes \varphi^{(k)}$ defines a feature extractor with $d = d_1 d_2 \cdots d_k$ components indexed by the multi-index $i = (i_1, i_2, \ldots, i_k)$ with $1 \leq i_m \leq d_m$, $m = 1, 2, \ldots, k$

$$\varphi_i = \varphi_{(i_1,\ldots,i_k)}(x) = \varphi^{(1)}_{i_1}(x_1)\, \varphi^{(2)}_{i_2}(x_2) \cdots \varphi^{(k)}_{i_k}(x_k)$$

Assume that for each m, the d_m basis functions are aligned in a row long a one-dimensional $\mathcal{X}_m$ component of the sensor space and only one weight is non-zero: Then the tensor product would (approximately) indicate a position in sensor space.

Tensor product construction: Example

Realisation by radial basis functions (RBF)

$$\varphi^{(m)}(x_m) = \left(G\left(\left|x_m - x_m^{(1)}\right|\right), \ldots, G\left(\left|x_m - x_m^{(d_m)}\right|\right)\right)^\top$$

where the $x_m^{(j)}$ are given (and possibly irregularly spaces) grid points and the basis functions are often chosen as $G(z) = \exp\left(-\frac{z^2}{2\sigma^2}\right)$ with some scale parameter σ. E.g. Gaussian:

$$\varphi_{(i_1,\ldots i_k)}(x) = \exp\left(-\frac{\sum_{m=1}^{k} \|x_m - x_m^{(i_m)}\|_{\mathcal{X}_m}^2}{2\sigma^2}\right)$$

or, symbolically,

$$\varphi_i(x) = \exp\left(-\frac{\|x - x^i\|_{\mathcal{X}}^2}{2\sigma^2}\right)$$

Kernel Smoothing

Similar to previous,

$$V_\theta(x) = \sum_{i=1}^{N} \theta_i \frac{G\left(\|x - x^{(i)}\|\right)}{\sum_{m=1}^{N} G\left(\|x - x^{(m)}\|\right)}$$

More generally,

$$V_\theta(x) = \sum_{i=1}^{N} \theta_i g_i(x)$$

satisfying the conditions $g_i(x) > 0$ and $\sum_{i=1}^{N} g_i(x) = 1 \, \forall x$

V_θ is an "averager", which mixes the values of θ differently at different points in space

Variants of look-up table implementations

- Binary features: $\varphi(x) \in \{0,1\}^N$

$$V_\theta(x) = \sum_{i:\varphi_i(x)=1} \theta_i$$

  Interesting case: only few components of φ are non-zero (sparse) and the relevant indexes can be computed efficiently.
- State aggregation: Indicator function over a certain region in state space
- Tile coding: CMAC (Cerebellar Model Articulation Controller, Albus 1971) uses partially overlapping hyper-rectangles

Curse of dimensionality

- Tile-code spaces are usually huge $\Longrightarrow$ use only cells that are actually visited
- Example: a robot with 6 DoF is characterised by 6 positions and 6 velocities, but e.g. cameras will produce high-dimensional state spaces $\Longrightarrow$ use projection methods (e.g. non-linear PCA)
- Often there are not too many data points $\Longrightarrow$ use non-parametric methods

TD(λ) with function approximation

- Express changes of the value function as changes of parameters
- Changes in parameters are usually small, so δ rule

$$\delta_{t+1} = r_t + \gamma \hat{V}_t(s_{t+1}) - \hat{V}_t(s_t)$$

$$\hat{V}_{t+1}(s_t) := \hat{V}_t(s_t) + \eta \delta_{t+1}$$

$$\Leftrightarrow \qquad \Delta \hat{V}_{t+1}(s_t) = \eta \delta_{t+1}$$

becomes for $V_\theta(x) = \theta^\top \varphi(x)$

$$\Delta\theta = \eta \left(\nabla_\theta V_{\theta_t}(x_t) \right) \delta_{t+1} = \eta \varphi(x) \delta_{t+1}$$

- We assume that the (finite) changes of the value function are linearly reflected in parameter changes and use the chain rule.
- Alternatively, use gradient descent on the fit function.

TD(λ) with function approximation

Given initial values of θ in $V_\theta = \theta^\top \varphi$ and of the eligibility traces $z_0 = (0, \ldots, 0)$ and previous state x_t and next state x_{t+1}

$$\begin{aligned} \delta_{t+1} &= r_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t) \\ z_{t+1} &= \nabla_\theta V_{\theta_t}(x_t) + \lambda z_t \\ \theta_{t+1} &= \theta_t + \alpha_t \delta_{t+1} z_{t+1} \end{aligned}$$

where $\nabla_\theta f(\theta) = \left(\frac{\partial}{\partial \theta_1} f(\theta), \ldots, \frac{\partial}{\partial \theta_N} f(\theta) \right)^\top$ is the gradient of $f(\theta)$

For $V_\theta = \theta^\top \varphi$ we have simply $\nabla_\theta V_\theta(x) = (\varphi_1(x), \ldots, \varphi_N(x))$

Here, eligibility traces measure how much a parameter contributed to V now and, weighted by λ in the past.

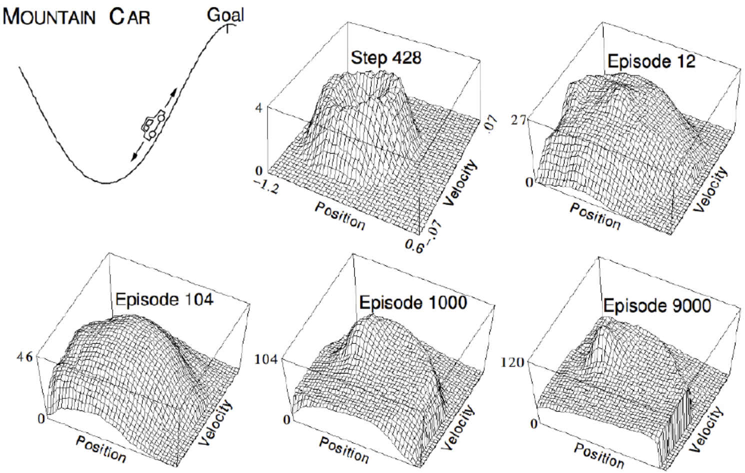
Algorithm: TD(λ) with function approximation

x last state, y next state, r immediate reward, θ parameter vector, z vector of eligibility traces

1. $\delta \leftarrow r + \gamma\theta^\top\varphi[y] - \theta^\top\varphi[x]$
2. $z \leftarrow \varphi[x] + \lambda z$
3. $\theta \leftarrow \theta + \alpha\delta z$
4. return (θ, z)

Note: Supposes linear approximation of V

Linear SARSA for the mountain car problem



Matt Kretchmar, 1995

$\mathcal{Q}$-learning with function approximation

Recall $\mathcal{Q}_{t+1}(x_t, a_t) = \mathcal{Q}_t(x_t, a_t) + \alpha\left(r_r + \gamma V(x_{t+1}) - \mathcal{Q}_t(x_t, a_t)\right)$

Now

$$\begin{aligned} \delta_{t+1} &= r_{t+1} + \gamma V(x_{t+1}) - \mathcal{Q}_t(x_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha_t \delta_{t+1}(\mathcal{Q}_{\theta_t}) \nabla_\theta \mathcal{Q}_{\theta_t}(x_t, a_t) \end{aligned}$$

with $\mathcal{Q}_{\theta_t} = \theta^\top \varphi$ and $\varphi : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}^N$ is a basis function over the state-action space. V is given as a maximum of $\mathcal{Q}$ w.r.t. a.

Algorithm: $\mathcal{Q}$-learning with function approximation

x last state, y next state, r immediate reward, θ parameter vector

1. $\delta \leftarrow r + \gamma \max_{a' \in \mathcal{A}} \theta^\top \varphi\,[y, a'] - \theta^\top \varphi\,[x, a]$
2. $\theta \leftarrow \theta + \alpha \delta \varphi\,[x, a]$
3. return θ

Convergence

- Widely used but convergence can be shown only locally (local optima!)
- Even in the linear case, parameters may diverge (Bertsekas and Tsitsiklis, 1996) due to biased sampling or for non-linear approximations of V or $\mathcal{Q}$.
- Almost sure convergence to a unique parameter vector was shown for linear approximation, ergodic Markov process with well-behaved stationary distribution under the Robbins-Monro conditions and for linearly independent φ.
- If convergent, the best approximation of the true value function among all the linear approximants is found.

Gradient temporal difference learning

Goal: Make sure that divergence does not occur.

For simplicity, assume $\lambda = 0$, and the underlying process (x,r, x') is stationary and the optimal parameter vector θ^* exists.

x last state, y next state, r immediate reward, α, β learning rates, θ parameter vector, w auxiliary weight, minimise δ^2

1. $\delta \leftarrow r + \gamma\theta^\top\varphi[y] - \theta^\top\varphi[x]$
2. $a \leftarrow \varphi(x)^\top w$
3. $\theta \leftarrow \theta + \alpha(\varphi(x) - \gamma\varphi(y))\, a$
4. $w \leftarrow w + \beta(\delta - a)\varphi(x)$
5. return (θ)

Parameter update (step 3) modulated by a which induces normalisation of the projection of δ onto φ, thus avoiding divergence.

The choice of the function space

- In look-up table algorithms averaging happens within the elements of the table and is safe under the RM conditions
- Here, however, approximation and estimation of the value function may interfere
- Target function V and approximation V_θ: Approximation error

$$E = \inf_{\theta} \| V_\theta - V \|^2$$

- Choosing sufficiently many features, the error on a finite number of values (e.g. in an episodic task) can be reduced to zero (in principle) $\Rightarrow$ overfitting for possibly noisy rewards/states
- Trade-off between approximation errors (model) and estimation (values)
- Use regularisation!

Fitted Q-learning: Algorithm

Use all (recent) state-action pairs for the update $\Rightarrow$ Monte Carlo

1. $S \leftarrow [\,]$ // create empty list
2. for $t = 1$ to T // to present
3. $\quad \hat{V} \leftarrow r_{i+1} + \gamma \max_{a' \in \mathcal{A}} \text{predict}\left(\left(y_{t+1}, a'\right), \theta\right)$ //estimate value
4. $\quad S \leftarrow \text{append}\left(S, \left(\left\{x_t, a_t\right\}, \hat{V}\right)\right)$
5. end for
6. $\theta \leftarrow \text{regress}\left(S\right)$ // maximise likelihood of model

Notes: Prediction and regression should be matched. May diverge for unsuitable regressor.

Summary

- Large state space require intelligent representations
- Continuous state spaces require function approximation
- Algorithms do not necessarily become more complex, but lose the property of global convergence
- Choice of function space is an open problem (often not too difficult for practical problems)
- Gradient POMDP?
- Next time: Compatible representations

Acknowledgements

Some material was adapted from web resources associated with Sutton and Barto's Reinforcement Learning book

Today mainly based on C. Szepesvári's book, sections 2.2 and 3.3.2