RL 10: Algorithms for Large State Spaces

Michael Herrmann

University of Edinburgh, School of Informatics

13/02/2015
Overview

- Algorithms for spatially continuous problems
- Basis functions
- Reformulation of algorithms in terms of gradients
Large State Spaces

- Grid-world algorithms: $V(s)$ is a vector, $Q(s, a)$ a matrix (look-up tables)
- In large problems, in particular in continuous domains where discretisation is not obvious, the complexity is often beyond practical limits
  - storage space
  - exploration time
  - convergence time
- Generalisation and flexibility is low
- Possible approaches:
  - Hierarchical representations
  - Vector quantisation for efficient state space representation,
  - Function approximation
Function approximation

- Use methods that are known to generalise well.
- Learning from examples: Reconstruct the underlying function by supervised learning
  - E.g. neural networks
    - given samples \( \{x_i, y_i\}_{i=1,...,M} \)
    - initialise weight vector \( \theta \)
    - realised function: \( f(\cdot, \theta) \), i.e. \( y = f(x, \theta) \)
    - error measure: \( E = \frac{1}{2} \sum_{i=1}^{M} \| y_i - f(x_i, \theta) \|^2 \)
    - update weights, e.g. simply by: \( \theta_t = \theta_{t-1} - \eta \frac{\partial E}{\partial \theta} \), i.e. for single datum:
      \[
      \Delta \theta_t = \eta (y_i - f(x_i, \theta)) \nabla_\theta f(x_i, \theta)
      \]
    - try to control generalisation properties
in RL: $x$ encodes state, $f$ represents value function or policy

Problems:

- Distribution of samples depends on policy

$$E = \frac{1}{2} \sum_{x \in S} d(x) \| y(x) - f(x, \theta) \|^2$$

where $d$ is a distribution over states, $\sum_{x \in S} d(x) = 1$

- Initially no samples available!

Possible solution:

1. Agent produces samples
2. Estimate value function on samples
3. Use estimate to train the function approximator
Use $\delta$-error

$$\delta_t = r_{t+1} + \gamma \hat{V}(x_{t+1}, \theta) - \hat{V}(x_t, \theta)$$

for parameter update (learning rate $\eta$)

$$\Delta \theta = \eta \delta_t \zeta_t$$

with error

$$\zeta_t = \nabla_{\theta} \hat{V}(x_t, \theta)$$

Initialisation: $\theta = (0, \ldots, 0)\top$

Termination: $\max \delta$ sufficiently small
Function approximation for TD(\(\lambda\))

Use \(\delta\)-error

\[
\delta_{t,1} = r_{t+1} + \gamma \hat{V}(x_{t+1}, \theta) - \hat{V}(x_t, \theta)
\]

for parameter update (learning rate \(\eta\))

\[
\Delta \theta = \eta \delta_t \zeta_t
\]

including eligibility trace \(\zeta_t = (\zeta_{t,1}, \ldots, \zeta_{t,n})^\top\) with \(n \gg (1 - \lambda)^{-1}\),

\[
\zeta_t = \gamma \lambda \zeta_{t-1} + \nabla_\theta \hat{V}(x_t, \theta)
\]

\(\delta_i = (\delta_{t,1}, \ldots, \delta_{t,n})\) is also a vector containing the past \(\delta\) values.

Initialisation: \(\zeta_t = (0, \ldots, 0)^\top\), \(\theta = (0, \ldots, 0)^\top\)

Termination: \(\max \delta\) sufficiently small
The “true value” \( r_{t+1} + \gamma \hat{V}(x_{t+1}, \theta) \) in the error term contains the current approximation of the value function and is updated only in the next step.

For \( \lambda = 0 \), the algorithm become a standard gradient descent (but with respect to an estimate), compare slide 4.

For \( \lambda > 0 \) a trace over previous errors is assumed to improve the current error estimate.

The general-\( \lambda \) case is very appropriate here, since we are assuming that the values function is smooth. Furthermore, the information from previous states does not enter the value for a specific state but via the parameters at least some region in state space.

The gradient w.r.t. \( \theta \) must be known. We are using parametric function approximation: The parametrisation should be expressive and convenient.
Represent the value function e.g. in this 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 \( \varphi : \mathbb{R}^D \rightarrow \mathbb{R}^N \) with \( \varphi(x) = (\varphi_1(x), \ldots, \varphi_N(x))^\top \)

Many choices for the basis functions \( \varphi \) are possible.

Including the trivial case of the look-up table representation for \( \theta_s = V(s) \) and

\[ \varphi_s(x) = \begin{cases} 
1 & \text{if } \text{int}(x) = s \\
0 & \text{otherwise}
\end{cases} \]
Feature spaces

- $V_\theta(x) = \theta^\top \varphi(x)$ is a 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

- $\varphi : \mathcal{X} \rightarrow \mathbb{R}^N$
  - $\varphi(x) = (\varphi_1(x), \ldots, \varphi_N(x))^\top$
  - $\varphi_i : \mathcal{X} \rightarrow \mathbb{R}$ is a basis function
  - $\varphi_i(x)$: a feature of the state $x$
  - Examples: polynomial, wavelets, RBF, ... 

- mathematically convenient: easily differentiable $\Rightarrow$ gradient

- Other parametrisations may be more effective. Currently, deep networks are of interest in RL.
Radial basis functions

For a function \( f : \mathbb{R}^D \rightarrow \mathbb{R} \) choose parameters such that

\[
f(x) \approx \theta^\top \varphi(x)
\]

with \( \varphi : \mathbb{R}^D \rightarrow \mathbb{R}^N \), e.g.

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

with \( i = 1, \ldots, N \).

Determine \( \theta \in \mathbb{R}^N \) by

\[
\|f - \theta^\top \varphi\| \rightarrow \min
\]

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

example:
\( N = 2, \theta = (1, 1) \)
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 $1 \leq i_m \leq d_m$, $m = 1, 2, \ldots, k$.

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

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.
Realisation by radial basis functions (RBF)

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

where the \( x_m^{(j)} \) are given (and possibly irregularly spaced) 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 \( \sigma \). E.g. Gaussian:

\[ \varphi(i_1, \ldots i_k)(x) = \exp \left( -\sum_{m=1}^{k} \left\| x_m - x_m^{(i_m)} \right\| x_m^{2\sigma^2} \right) \]

or, symbolically,

\[ \varphi_i(x) = \exp \left( -\frac{\|x - x^i\|^2}{2\sigma^2} \right) \]
Kernel Smoothing

Similar to previous,

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

More generally,

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

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

\( V_{\theta} \) is an “averager”, which mixes the values of \( \theta \) differently at different points in space
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 \( \varphi \) 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 $\implies$ 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 $\implies$ use projection methods (e.g. non-linear PCA)
- Often there are not too many data points $\implies$ use non-parametric methods
Express changes of the value function as changes of parameters

Changes in parameters are usually small, so $\delta$ rule

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

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

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

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

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

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

\[
\delta_{t+1} = r_{t+1} + \gamma V_{\theta_t} (x_{t+1}) - V_{\theta_t} (x_t)
\]
\[
z_{t+1} = \nabla_{\theta} V_{\theta_t} (x_t) + \lambda z_t
\]
\[
\theta_{t+1} = \theta_t + \eta_t \delta_{t+1} z_{t+1}
\]

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 $\lambda$ in the past.
Algorithm: TD(\(\lambda\)) with function approximation

\(x\) last state, \(y\) next state, \(r\) immediate reward, \(\theta\) parameter vector, 
\(z\) vector of eligibility traces

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

Note: Supposes linear approximation of \(V\)
Linear SARSA (see next slide) for the mountain car problem

Matt Kretchmar, 1995
Linear SARSA(\(\lambda\)) (see Fig 9.8 in S&B 2nd Ed.)

Let \(w\) and \(z\) be vectors with one component for each possible feature.
Let \(\mathcal{F}_a\), for every possible action \(a\), be a set of feature indices, initially empty.
Initialize \(w\) as appropriate for the problem, e.g., \(w = 0\).
Repeat (for each episode):

\[
\begin{align*}
&z = 0 \\
&S, A \leftarrow \text{initial state and action of episode} \\
&\mathcal{F}_A \leftarrow \text{set of features present in } S, A \\
\end{align*}
\]

Repeat (for each step of episode):

For all \(i \in \mathcal{F}_A:\)

\[
\begin{align*}
z_i &\leftarrow z_i + 1 & \text{(accumulating traces)} \\
&\text{or } z_i &\leftarrow 1 & \text{(replacing traces)}
\end{align*}
\]

Take action \(A\), observe reward, \(R\), and next state, \(S\).
\[
\delta \leftarrow R - \sum_{i \in \mathcal{F}_A} w_i
\]
(1 - \(\varepsilon\):

With probability 1 - \(\varepsilon\):

For all \(a \in \mathcal{A}(S)\):

\[
\begin{align*}
&\mathcal{F}_a \leftarrow \text{set of features present in } S, a \\
&\hat{q}_a \leftarrow \sum_{i \in \mathcal{F}_a} w_i \\
&A \leftarrow \arg \max_{a \in \mathcal{A}(S)} \hat{q}_a
\end{align*}
\]

else

\[
A \leftarrow \text{a random action } \in \mathcal{A}(S)
\]

\[
\mathcal{F}_A \leftarrow \text{set of features present in } S, A
\]

\[
\hat{q}_A \leftarrow \sum_{i \in \mathcal{F}_A} w_i
\]

\[
\delta \leftarrow \delta + \gamma \hat{q}_A
\]

\[
w \leftarrow w + \alpha \delta z
\]

\[
z \leftarrow \gamma \lambda z
\]

13/02/2015 Michael Herrmann RL 10
Recall $Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha (r_t + \gamma V(x_{t+1}) - Q_t(x_t, a_t))$

Now

$$
\delta_{t+1} = r_{t+1} + \gamma V(x_{t+1}) - Q_t(x_t, a_t)
$$

$$
\theta_{t+1} = \theta_t + \alpha_t \delta_{t+1} (Q_{\theta_t}) \nabla_{\theta} Q_{\theta_t}(x_t, a_t)
$$

with $Q_{\theta_t} = \theta^\top \varphi$ 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 $Q$ w.r.t. $a$. 

13/02/2015 Michael Herrmann RL 10
Algorithm: Q-learning with function approximation

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

1. δ ← r + γ \( \max_{a' \in A} \theta^T \varphi[y, a'] - \theta^T \varphi[x, a] \)
2. θ ← θ + αδ \( \varphi[x, a] \)
3. return θ
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 $Q$.

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 $\varphi$.

If convergent, the best approximation of the true value function among all the linear approximants is found.
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 $V_\theta$: 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 ⇒ Monte Carlo

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