REINFORCEMENT LEARNING

LU 10 - Representing the Path Costs

Dr. Joschka Bödecker
AG Maschinelles Lernen und Natürlichsprachliche Systeme
Albert-Ludwigs-Universität Freiburg

jboeddeck@informatik.uni-freiburg.de

Acknowledgement
Slides courtesy of Martin Riedmiller and Martin Lauer
LU 10: Representing the path costs

Goals:
- Representation of path costs in very big / continuous state spaces
- Typical forms of function approximation in reinforcement learning

Outline
- Discretization
- Trees
- Linear models
- Nonlinear features for linear models
- Neural nets
**Representation of path costs** $J(\cdot)$

**Until now:** Finite state sets $S = \{0, 1, \ldots, n\}$

- $J$: One-dimensional array of length $n$
- $Q$: $n \times m$ -dimensional table with $m = \text{number of actions}$
Representation of path costs $J(\cdot)$

Until now: Finite state sets $S = \{0, 1, \ldots, n\}$
$\Rightarrow J$: One-dimensional array of length $n$
$\Rightarrow Q$: $n \times m$ -dimensional table with $m =$ number of actions

But: What if $n$ very big (e.g. Backgammon: $10^{15}$) or $S$ has a continuous domain (robotics, control engineering)?
- how can we represent the value function
- how can we visit every state?
Representation of path costs $J(\cdot)$

Until now: Finite state sets $S = \{0, 1, \ldots, n\}$
\Rightarrow J: One-dimensional array of length $n$
\Rightarrow Q: $n \times m$-dimensional table with $m = \text{number of actions}$

But: What if $n$ very big (e.g. Backgammon: $10^{15}$) or $S$ has a continuous domain (robotics, control engineering)?
- how can we represent the value function
- how can we visit every state?
\Rightarrow Approximation of the value function
- Discretization
- Function approximation by regression
Regular discretization

**MountainCar**

Input space is two-dimensional:
- $x$: Position
- $\dot{x}$: Velocity
Output: $J$ (path costs)

State space is regularly discretized with resolution $(m, n)$
⇒ Path costs are piecewise constant.
⇒ Number of parameters: one per cell (exponential increase in dimensions)
Irregular discretization

MountainCar

Input space is two-dimensional:
- $x$: Position
- $\dot{x}$: Velocity

Output: $J$ (path costs)

Individual resolution of separate sections of the state space

$\Rightarrow$ Path costs are piecewise constant

$\Rightarrow$ Number of parameters: one per cell
Discussion (discretization)

Advantages

▶ Efficient data structure (no expensive calculations needed)
▶ Transparent (inspection of the table possible)
▶ Theoretically analysis possible

Disadvantages

▶ High memory demand (curse of dimensionality)
▶ Low generalization performance
▶ For practical problems hardly deployable
Discussion (discretization)

Advantages

▶ Efficient date structure (no expensive calculations needed)
▶ Transparent (inspection of the table possible)
▶ Theoretically analysis possible

Disadvantages

▶ High memory demand (curse of dimensionality)
▶ Low generalisation performance
▶ For practical problems hardly deployable
Adaptive discretization

**MountainCar**

Input space is two-dimensional:
- $x$: Position
- $\dot{x}$: Velocity

Output: $J$ (path costs)

Start with very rough discretization...

Table

<table>
<thead>
<tr>
<th></th>
<th>J(1)</th>
<th>J(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J(0)</td>
<td>J(3)</td>
<td></td>
</tr>
</tbody>
</table>

Prof. Dr. M. Riedmiller, Dr. M. Lauer, Dr. J. Boedecker  
Machine Learning Lab, University of Freiburg  
Reinforcement Learning (7)
Adaptive discretization

MountainCar

Input space is two-dimensional:
- $x$: Position
- $\dot{x}$: Velocity
Output: $J$ (path costs)

Table

... refinement only in important sections of the state space
Adaptive discretization

**MountainCar**

- Input space is two-dimensional:
  - $x$: Position
  - $\dot{x}$: Velocity
- Output: $J$ (path costs)

Problem: Which parts of the state space are important, under which conditions should we refine the grid?
Regression tree

KD-Tree

Leaves are linked to target values ⇒
discretization with tree structure
Regression tree

KD-Tree

Leaves are linked to target values ⇒ discretization with tree structure

Learning regression trees

- Start with root node
- Successive splitting of the leaves along dimensions of the input space
- When do we split a leaf? (e.g. variance)
Discussion (regression trees)

Advantages
- Adapts to complexity of the problem
- Informative (tree structure interpretable)
- Works well with ensemble techniques
- Efficient for hard problems as well
Discussion (regression trees)

Advantages

▶ Adapts to complexity of the problem
▶ Informative (tree structure interpretable)
▶ Works well with ensemble techniques
▶ Efficient for hard problems as well

Disadvantages

▶ Good splitting criteria hard to find
▶ Depth of trees is unbounded
▶ Just a form of discretization ...
▶ Limited generalization in a very local environment
Discussion (regression trees)

Advantages

▶ Adapts to complexity of the problem
▶ Informative (tree structure interpretable)
▶ Works well with ensemble techniques
▶ Efficient for hard problems as well

Disadvantages

▶ Good splitting criteria hard to find
▶ Depth of trees is unbounded
▶ Just a form of discretization ...
▶ Limited generalization in a very local environment

Learning

▶ Requires parameter optimization (one parameter per cell)
▶ Requires update of partitioning structure
Literature about discretization and regression trees

Articles worth reading


Regression

Approximation of data points through a functional relationship given set of patterns

\[ D = \{ (x^1, t^1), (x^2, t^2), \ldots, (x^P, t^P) \} \]

with

\[ x \in \mathbb{R}^n, t \in \mathbb{R}^m \]

Assumption: \( t^i = f(x^i) + \eta^i \), with \( \eta^i \) random variable with zero mean, 'noise'

goal: find a model, i.e. a function \( y(x; w) \) that approximates \( f(.) \) as accurately as possible. Here, \( w \) are the parameters of the model.
Linear regression

Example:
Input space one-dimensional, desired model a linear function

\[ x = x_1; \ w = (w_0, w_1) \]

\[ y(x, w) = w_0 + w_1 x_1 \]
Linear regression

Example:
Input space one-dimensional, desired model a linear function

\[ x = x_1; \ w = (w_0, w_1) \]

\[ y(x, w) = w_0 + w_1 x_1 \]

Extension to \( n \)-dimensional input:

\[ y(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \]

\[ = w_0 + \sum_{i=1}^{n} w_i x_i \]

\[ = w^T x \]

with \( x = (1, x_1, x_2, \ldots, x_n)^T \), \( w = (w_0, w_1, w_2, \ldots, w_n)^T \)
Linear regression

Example:
Input space one-dimensional, desired model a linear function

\[ x = x_1; \quad w = (w_0, w_1) \]

\[ y(x, w) = w_0 + w_1 x_1 \]

Extension to \( n \)-dimensional input:

\[ y(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \]

\[ = w_0 + \sum_{i=1}^{n} w_i x_i \]

\[ = w^T x \]

with \( x = (1, x_1, x_2, \ldots, x_n)^T, \quad w = (w_0, w_1, w_2, \ldots, w_n)^T \)

Learning: Find parameters \( w \) such that model fits best to data
Regression with sum of least squares

How can we determine $w$ from data?

Idea: minimize the sum of squared errors for all data points.
Regression with sum of least squares

How can we determine $w$ from data?

Idea: minimize the sum of squared errors for all data points.

**Error per pattern (residual):** $e^p(w) = (y(x^p, w) - t^p)^2$

**Total error for all patterns:**

$$E(w) := \frac{1}{2} \sum_{p=1}^{P} e^p(w) = \frac{1}{2} \sum_{p=1}^{P} (y(x^p, w) - t^p)^2$$
Regression with sum of least squares

How can we determine \( \mathbf{w} \) from data?

Idea: minimize the sum of squared errors for all data points.

**Error per pattern (residual):**  \[ e^p(\mathbf{w}) = (y(x^p, \mathbf{w}) - t^p)^2 \]

**Total error for all patterns:**

\[
E(\mathbf{w}) := \frac{1}{2} \sum_{p=1}^{P} e^p(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^{P} (y(x^p, \mathbf{w}) - t^p)^2
\]

**task:** find \( \mathbf{w}^* \) with

\[
\mathbf{w}^* = \arg \min_{\mathbf{w}} E(\mathbf{w}) = \arg \min_{\mathbf{w}} \frac{1}{2} \sum_{p=1}^{P} (y(x^p, \mathbf{w}) - t^p)^2
\]
Linear regression with sum of least squares

In the case of linear functions \( y(x, w) = w^T x \) we can solve the task analytically. Approach: Zero the partial derivations of the error term.

\[
\frac{\partial e^i(w)}{\partial w_j} = 2 \cdot x_j^i \cdot (w^T x^i - t^i)
\]

\[
\frac{\partial E(w)}{\partial w_j} = \sum_{i=1}^{P} x_j^i \cdot (w^T x^i - t^i)
\]

\[
= w^T \sum_{i=1}^{P} x_j^i x^i - \sum_{i=1}^{P} x_j^i t^i
\]
Linear regression with sum of least squares

In the case of linear functions \( y(x, w) = w^T x \) we can solve the task analytically. Approach: Zero the partial derivations of the error term.

\[
\begin{align*}
\frac{\partial e^i(w)}{\partial w_j} &= 2 \cdot x^i_j \cdot (w^T x^i - t^i) \\
\frac{\partial E(w)}{\partial w_j} &= \sum_{i=1}^{P} x^i_j \cdot (w^T x^i - t^i) \\
&= w^T \sum_{i=1}^{P} x^i_j x^i - \sum_{i=1}^{P} x^i_j t^i
\end{align*}
\]

Hence,

\[
\text{grad } E = w^T \sum_{i=1}^{P} (x^i)^T x^i - \sum_{i=1}^{P} (x^i)^T t^i
\]
Linear regression with sum of least squares

In the case of linear functions \( y(x, w) = w^T x \) we can solve the task analytically. Approach: Zero the partial derivations of the error term.

\[
\frac{\partial e^i(w)}{\partial w_j} = 2 \cdot x^i_j \cdot (w^T x^i - t^i)
\]

\[
\frac{\partial E(w)}{\partial w_j} = \sum_{i=1}^{P} x^i_j \cdot (w^T x^i - t^i)
\]

\[
= w^T \sum_{i=1}^{P} x^i_j x^i - \sum_{i=1}^{P} x^i_j t^i
\]

Hence,

\[
\text{grad } E = w^T \sum_{i=1}^{P} (x^i)^T x^i - \sum_{i=1}^{P} (x^i)^T t^i
\]

Zeroing gradient yields system of linear equations

\[
\left( \sum_{i=1}^{P} (x^i)^T x^i \right) \cdot w = \sum_{i=1}^{P} t^i x^i \quad \Rightarrow \quad w^* = \left( \sum_{i=1}^{P} (x^i)^T x^i \right)^{-1} \cdot \left( \sum_{i=1}^{P} t^i x^i \right)
\]
Regression with sum of least squares

Alternative way to determine \( w^* \): stochastic gradient descent

If \( w_t \) is our present guess of \( w^* \) and we observe data point \((x^i, t^i)\), we update:

\[
w_{t+1} = w_t - \epsilon \cdot grad e^i(w)
\]

with \( \epsilon > 0 \) a learning rate
Regression with sum of least squares

Alternative way to determine $w^*$: stochastic gradient descent

If $w_t$ is our present guess of $w^*$ and we observe data point $(x^i, t^i)$, we update:

$$w_{t+1} = w_t - \epsilon \cdot \nabla e^i(w)$$

with $\epsilon > 0$ a learning rate

If the learning rate is small and decreasing, the algorithm converges to $w^*$
Stochastic gradient descent can also be applied to nonlinear regression tasks.
Basis functions (features)

**Advantages** of linear models

- closed-form solution
- allow theoretical analysis of convergence behavior
Basis functions (features)

Advantages of linear models
- closed-form solution
- allow theoretical analysis of convergence behavior

Disadvantages of linear models
- poor expressive power
Basis functions (features)

Advantages of linear models

▶ closed-form solution

▶ allow theoretical analysis of convergence behavior

Disadvantages of linear models

▶ poor expressive power

Idea:

▶ combine linear models with nonlinear features (basis functions)

▶ transform data into feature space

▶ learn linear model in feature space
Basis functions (features)

Inputs (states, state-action-pairs): $x \in \mathbb{R}^n$, $t \in \mathbb{R}$
Basis functions (features)

Inputs (states, state-action-pairs): $x \in \mathbb{R}^n$, $t \in \mathbb{R}$

Non-linear basis functions: $\phi_i : \mathbb{R}^n \to \mathbb{R}$

Feature vector: $\Phi(x) = (\phi_1(x), \phi_2(x), ..., \phi_m(x))$
Basis functions (features)

Inputs (states, state-action-pairs): \( x \in \mathbb{R}^n, \ t \in \mathbb{R} \)

Non-linear basis functions: \( \phi_i : \mathbb{R}^n \rightarrow \mathbb{R} \)
Feature vector: \( \Phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_m(x)) \)

Linear model in feature space: \( y(\Phi(x, w)) = \sum_{i=1}^{m} \phi_i(x) \cdot w_i = w^T \cdot \Phi(x) \)
Basis functions (features)

Inputs (states, state-action-pairs): \( x \in \mathbb{R}^n, \ t \in \mathbb{R} \)

Non-linear basis functions: \( \phi_i : \mathbb{R}^n \rightarrow \mathbb{R} \)
Feature vector: \( \Phi(x) = (\phi_1(x), \phi_2(x), ..., \phi_m(x)) \)

Linear model in feature space: \( y(\Phi(x, w)) = \sum_{i=1}^{m} \phi_i(x) \cdot w_i = w^T \cdot \Phi(x) \)

Nonlinear model in input space

Same idea as, e.g., in support vector machines and Gaussian processes
Rectangular basis functions

- 1-d case:
  \[ \phi_i(x) = \begin{cases} 
  1 & \text{if } a_i \leq x < b_i \\
  0 & \text{otherwise}
  \end{cases} \]
Rectangular basis functions

1-d case:
\[
\phi_i(x) = \begin{cases} 
1 & \text{if } a_i \leq x < b_i \\
0 & \text{otherwise}
\end{cases}
\]

n-d case:
\[
\phi_i(\vec{x}) = \begin{cases} 
1 & \text{if } \vec{x} \in A_i \\
0 & \text{otherwise}
\end{cases}
\]

with \( A_i \subseteq \mathbb{R}^n \)
Rectangular basis functions

- **1-d case:**
  \[ \phi_i(x) = \begin{cases} 
  1 & \text{if } a_i \leq x < b_i \\ 
  0 & \text{otherwise} 
  \end{cases} \]

- **n-d case:**
  \[ \phi_i(\vec{x}) = \begin{cases} 
  1 & \text{if } \vec{x} \in A_i \\ 
  0 & \text{otherwise} 
  \end{cases} \]

  with \( A_i \subseteq \mathbb{R}^n \)

  \( \Rightarrow \) step functions
Triangular basis functions

1-d case:
choose $k$ supporting points $z_1, \ldots, z_k$
in increasing order and two boundaries
$z_0 < z_1, z_{k+1} > z_k$

$$
\phi_i(x) = \begin{cases} 
\frac{z_i - x}{z_i - z_{i-1}} & \text{if } z_{i-1} \leq x < z_i \\
\frac{x - z_i}{z_{i+1} - z_i} & \text{if } z_i \leq x < z_{i+1} \\
0 & \text{otherwise}
\end{cases}
$$
Triangular basis functions

- 1-d case:
  choose \( k \) supporting points \( z_1, \ldots, z_k \) in increasing order and two boundaries \( z_0 < z_1, z_{k+1} > z_k \)

\[
\phi_i(x) = \begin{cases} 
\frac{z_i-x}{z_i-z_{i-1}} & \text{if } z_{i-1} \leq x < z_i \\
\frac{x-z_i}{z_{i+1}-z_i} & \text{if } z_i \leq x < z_{i+1} \\
0 & \text{otherwise}
\end{cases}
\]

- \( n \)-d case:
  requires subpartitioning of input space into simplices
  - 1-d simplex = interval
  - 2-d simplex = triangle
  - 3-d simplex = tetrahedron
  - \( n \)-d simplex = area bounded by \( n + 1 \) hyperplanes

one basis function for each vertex of the simplices. Feature functions depend on barycentric coordinates
Triangular basis functions

- **1-d case:**
  choose $k$ supporting points $z_1, \ldots, z_k$ in increasing order and two boundaries $z_0 < z_1, z_{k+1} > z_k$

  $$\phi_i(x) = \begin{cases} 
  \frac{z_i - x}{z_i - z_{i-1}} & \text{if } z_{i-1} \leq x < z_i \\
  \frac{x - z_i}{z_{i+1} - z_i} & \text{if } z_i \leq x < z_{i+1} \\
  0 & \text{otherwise}
  \end{cases}$$

- **$n$-d case:**
  requires subpartitioning of input space into simplices
  - 1-d simplex = interval
  - 2-d simplex = triangle
  - 3-d simplex = tetrahedron
  - $n$-d simplex = area bounded by $n + 1$ hyperplanes

  one basis function for each vertex of the simplices. Feature functions depend on *barycentric coordinates*

  $\Rightarrow$ piecewise linear functions
Monomial features

- 1-d case:
  1, x, x^2, x^3, ...
Monomial features

- 1-d case:
  1, x, x^2, x^3, ...

- 2-d case:
  1, x_1, x_2, x_1^2, x_1 x_2, x_2^2, x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3, ...
Monomial features

- 1-d case:
  1, x, x^2, x^3, ...

- 2-d case:
  1, x_1, x_2, x_1^2, x_1 x_2, x_2^2, x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3, ...

⇒ polynomial functions
Radial basis functions (RBF networks)

Gaussian feature functions

$$\phi_i(x) = \exp\left(\frac{||x-\mu_i||^2}{2\sigma_i^2}\right)$$

⇒ Basis functions represent clusters in the input space

⇒ Idea: Overlap of features achieves good generalisation. Smooth variation of simplex tessalation
CMAC (cerebellar model articulation controller)

Use rectangular feature functions but choose overlapping rectangles.

⇒ better generalization
Neural nets: The Multi-layer perceptron (MLP)

Simple perceptron:

Idea: Cascade multiple neurons
Supervised learning in MLPs

**Task:**
given: Training set $D$ consisting of pairs of patterns $(x^i, t^i)$
task: Find suitable weights $w_{ij}$
Supervised learning in MLPs

**Task:**
given: Training set $D$ consisting of pairs of patterns $(x^i, t^i)$
task: Find suitable weights $w_{ij}$

**Solution:** Minimization of an error function $E$ through **gradient descent**:

Total error function

$$E(w) := \frac{1}{2} \sum_{p=1}^{P} (y(x^p, w) - t^p)^2$$

Weight update

$$w_{ij}^{\text{new}} := w_{ij}^{\text{old}} - \epsilon \frac{\partial E(w)}{\partial w_{ij}}$$

Calculation of $\frac{\partial E}{\partial w_{ij}}$: **Backpropagation**
Setup of single neuron

Terms

- Neuron, unit $i$
- $k$ incoming weights of neuron $j$ to neuron $i$: $w_{i1}, \ldots, w_{ij}, \ldots, w_{ik}$
- 'Net input', 'internal activation':
  \[ net_i := \sum_{j=0}^{n} w_{ij} s_j \]
- Activation or output value of neuron $i$ $s_i := f_{\text{sig}}(net_i)$ with $f_{\text{sig}}$ : activation function

Model:
The activation function

**Motivation:** The threshold function $f_{\text{step}}$ is approximated by a differentiable, monotonically increasing function. Example: sigmoid functions (hyperbolic tangent)

$$f_{\text{sig}}(z) = \frac{1}{1 + e^{-z}}$$
The activation function

**Motivation:** The threshold function \( f_{\text{step}} \) is approximated by a differentiable, monotonically increasing function. Example: sigmoid functions (hyperbolic tangent)

\[
f_{\text{sig}}(z) = \frac{1}{1 + e^{-z}}
\]

Alternatives: hinge function \( z \mapsto \max\{z, 0\} \)
Reinforcement learning with function approximation

This chapter: several generic function approximation methods
This chapter: several generic function approximation methods

Open questions:

▶ how can these approximators be combined with RL-algorithms (Value iteration, Q-learning, TD(0)), policy iteration)?
▶ which phenomena occur?
▶ differences between supervised learning and reinforcement learning