From Structured Prediction to Inverse Reinforcement Learning

Hal Daumé III

Computer Science
University of Maryland

me@hal3.name

A Tutorial at ACL
Uppsala Universitet

Sunday, 11 July 2010

Acknowledgements

Some slides:
Stuart Russell
Dan Klein
J. Drew Bagnell
Nathan Ratliff

Discussions/Feedback:
MLRG Spring 2010
## NLP as transduction

<table>
<thead>
<tr>
<th>Task</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Translation</td>
<td>Ces deux principes se tiennent à la croisée de la philosophie, de la politique, de l’économie, de la sociologie et du droit.</td>
<td>Both principles lie at the crossroads of philosophy, politics, economics, sociology, and law.</td>
</tr>
<tr>
<td>Document Summarization</td>
<td>Argentina was still obsessed with the Falkland Islands even in 1994, 12 years after its defeat in the 74-day war with Britain. The country's overriding foreign policy aim continued to be winning sovereignty over the islands.</td>
<td>The Falkland islands war, in 1982, was fought between Britain and Argentina.</td>
</tr>
<tr>
<td>Syntactic Analysis</td>
<td>The man ate a big sandwich.</td>
<td>The man ate a big sandwich</td>
</tr>
<tr>
<td>...many more...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Structured prediction 101

Learn a function mapping inputs to complex outputs:

\[ f : X \rightarrow Y \]

Input Space  Decoding  Output Space

I  can  can  a  can
Why is structure important?

- Correlations among outputs
  - Determiners often precede nouns
  - Sentences usually have verbs

- Global coherence
  - It just *doesn't make sense* to have three determiners next to each other

- My objective (aka “loss function”) forces it
  - Translations should have good sequences of words
  - Summaries should be coherent
Outline: Part I

- What is Structured Prediction?
- Refresher on Binary Classification
  - What does it mean to learn?
  - Linear models for classification
  - Batch versus stochastic optimization
- From Perceptron to Structured Perceptron
  - Linear models for Structured Prediction
  - The “argmax” problem
  - From Perceptron to margins
- Learning to Search
  - Incremental Parsing
  - Search-based Structured Prediction
Outline: Part II

- Refresher on Reinforcement Learning
  - Markov Decision Processes
  - Q learning
- Apprenticeship Learning
  - Inverse RL
  - Apprenticeship Learning via IRL
- Inverse Optimal Control and A* Search
  - Maximum Margin Planning
  - Learning to Search

- Discussion
Refresher on Binary Classification
What does it mean to learn?

- Informally:
  - to predict the future based on the past

- Slightly-less-informally:
  - to take *labeled examples* and construct a function that will label them as a human would

- Formally:
  - Given:
    - A fixed unknown distribution $D$ over $X*Y$
    - A loss function over $Y*Y$
    - A finite sample of $(x,y)$ pairs drawn i.i.d. from $D$
  - Construct a function $f$ that has low expected loss with respect to $D
Feature extractors

- A feature extractor $\Phi$ maps examples to vectors

**Example:**

Dear Sir.

First, I must solicit your confidence in this transaction, this is by virtue of its nature as being utterly confidencial and top secret. ...

$\Phi$

- $W=\text{dear} : 1$
- $W=\text{sir} : 1$
- $W=\text{this} : 2$
- $W=\text{wish} : 0$
- $\text{MISSPELLED} : 2$
- $\text{NAMELESS} : 1$
- $\text{ALL\_CAPS} : 0$
- $\text{NUM\_URLS} : 0$

- Feature vectors in NLP are frequently sparse
Linear models for binary classification

- Decision boundary is the set of “uncertain” points

- Linear decision boundaries are characterized by weight vectors

<table>
<thead>
<tr>
<th>x</th>
<th>Φ(x)</th>
<th>w</th>
<th>∑ᵢ wᵢ Φᵢ(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>“free money”</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIAS : 1</td>
<td>BIAS : -3</td>
<td>(1)(-3) +</td>
<td></td>
</tr>
<tr>
<td>free : 1</td>
<td>free : 4</td>
<td>(1)(4) +</td>
<td></td>
</tr>
<tr>
<td>money : 1</td>
<td>money : 2</td>
<td>(1)(2) +</td>
<td></td>
</tr>
<tr>
<td>the : 0</td>
<td>the : 0</td>
<td>(0)(0) +</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 3</td>
<td></td>
</tr>
</tbody>
</table>
The perceptron

- Inputs = feature values
- Params = weights
- Sum is the response

- If the response is:
  - Positive, output +1
  - Negative, output -1

- When training, update on errors:
  \[ w = w + y \phi(x) \]

"Error" when:
\[ y w \cdot \phi(x) \leq 0 \]
Why does that update work?

- When \( y w^{\text{old}} \cdot \phi(x) \leq 0 \), update: \( w^{\text{new}} = w^{\text{old}} + y \phi(x) \)

\[
y w^{\text{new}} \phi(x) = y \left( w^{\text{old}} + y \phi(x) \right) \phi(x) \\
= y w^{\text{old}} \phi(x) + yy \phi(x) \phi(x)
\]

\[
\begin{align*}
&< 0 \\
&> 0
\end{align*}
\]
Support vector machines

- Explicitly optimize the **margin**
- Enforce that all training points are correctly classified

\[
\begin{align*}
\max_w & \quad \text{margin} \quad s.t. \quad \text{all points are correctly classified} \\
\max_w & \quad \text{margin} \quad s.t. \quad y_n w \cdot \phi(x_n) \geq 1, \quad \forall n \\
\min_w & \quad \|w\|^2 \quad s.t. \quad y_n w \cdot \phi(x_n) \geq 1, \quad \forall n
\end{align*}
\]
Support vector machines with slack

- Explicitly optimize the **margin**
- Allow some "noisy" points to be misclassified

\[
\begin{align*}
\min_{\mathbf{w}, \xi} & \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \xi_n \\
\text{s.t.} & \quad y_n \mathbf{w} \cdot \phi(x_n) + \xi_n \geq 1, \quad \forall n \\
& \quad \xi_n \geq 0, \quad \forall n
\end{align*}
\]
Batch versus stochastic optimization

- Batch = read in all the data, then process it
- Stochastic = (roughly) process a bit at a time

\[
\min_{\mathbf{w}, \xi} \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_n \xi_n
\]

s.t. \[ y_n \mathbf{w} \cdot \phi(x_n) + \xi_n \geq 1 \]
\[ , \quad \forall n \]
\[ \xi_n \geq 0 \quad , \quad \forall n \]

- For \( n=1..N \):
  - If \( y_n \mathbf{w} \cdot \phi(x_n) \leq 0 \)
  - \( \mathbf{w} = \mathbf{w} + y_n \phi(x_n) \)
Stochastically optimized SVMs

SVM Objective

For n=1..N:
- If \( y_n w \cdot \phi(x_n) \leq 1 \)
  - \( w = w + y_n \phi(x_n) \)
- \( w = \left(1 - \frac{1}{CN}\right)w \)

Implementation Note:
Weight shrinkage is SLOW. Implement it lazily, at the cost of double storage.

For n=1..N:
- If \( y_n w \cdot \phi(x_n) \leq 0 \)
  - \( w = w + y_n \phi(x_n) \)
From Perceptron to Structured Perceptron
Perceptron with multiple classes

- Store separate weight vector for each class $w_1, w_2, \ldots, w_K$

- For n=1..N:
  - Predict:
    $$\hat{y} = \arg \max_k w_k \cdot \phi(x_n)$$
  - If $\hat{y} \neq y_n$:
    $$w_{\hat{y}} = w_{\hat{y}} - \phi(x_n)$$
    $$w_{y_n} = w_{y_n} + \phi(x_n)$$

?! Why does this do the right thing?
Perceptron

- Originally:

\[
\begin{array}{c|c|c}
\text{x} & \Phi(x,1) & \Phi(x,2) \\
\hline
\text{“free money”} & \begin{array}{l}
\text{spam\_BIAS : 1} \\
\text{spam\_free : 1} \\
\text{spam\_money : 1} \\
\text{spam\_the : 0} \\
\text{...}
\end{array} & \begin{array}{l}
\text{ham\_BIAS : 1} \\
\text{ham\_free : 1} \\
\text{ham\_money : 1} \\
\text{ham\_the : 0} \\
\text{...}
\end{array}
\end{array}
\]

- For n=1..N:
  - Predict:

\[
\hat{y} = \text{arg} \max_k w_k \cdot \phi(x_n)
\]

  - If \( \hat{y} \neq y_n \):

\[
\begin{align*}
w_{\hat{y}} &= w_{\hat{y}} - \phi(x_n) \\
w_{y_n} &= w_{y_n} + \phi(x_n)
\end{align*}
\]

- For n=1..N:
  - Predict:

\[
\hat{y} = \text{arg} \max_k w \cdot \phi(x_n, k)
\]

  - If \( \hat{y} \neq y_n \):

\[
\begin{align*}
w &= w - \phi(x_n, \hat{y}) + \phi(x_n, y_n)
\end{align*}
\]
Features for structured prediction

- Allowed to encode *anything* you want

\[ \phi(x, y) = \]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I_Pro</td>
<td>1</td>
<td>&lt;s&gt;-Pro</td>
<td>1</td>
<td>has_verb</td>
</tr>
<tr>
<td>can_Md</td>
<td>1</td>
<td>Pro-Md</td>
<td>1</td>
<td>has_nn_lft</td>
</tr>
<tr>
<td>can_Vb</td>
<td>1</td>
<td>Md-Vb</td>
<td>1</td>
<td>has_n_lft</td>
</tr>
<tr>
<td>a Dt</td>
<td>1</td>
<td>Vb-Dt</td>
<td>1</td>
<td>has_nn_rgt</td>
</tr>
<tr>
<td>can_Nn</td>
<td>1</td>
<td>Dt-Nn</td>
<td>1</td>
<td>has_n_rgt</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>Nn-/s&gt;</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>

- Output features, *Markov features*, other features
Structured perceptron

- For n=1..N:
  - Predict:
    \[ \hat{y} = \text{arg max}_k \mathbf{w} \cdot \phi(x_n, k) \]
  - If \( \hat{y} \neq y_n \):
    \[ \mathbf{w} = \mathbf{w} - \phi(x_n, \hat{y}) + \phi(x_n, y_n) \]

- For n=1..N:
  - Predict:
    \[ \hat{y} = \text{arg max}_k \mathbf{w} \cdot \phi(x_n, k) \]
  - If \( \hat{y} \neq y_n \):
    \[ \mathbf{w} = \mathbf{w} - \phi(x_n, \hat{y}) + \phi(x_n, y_n) \]
If we only have output and Markov features, we can use Viterbi algorithm:

(plus some work to account for boundary conditions)
Structured perceptron as ranking

- For n=1..N:
  - Run Viterbi: \( \hat{y} = \text{arg max}_k w \cdot \phi(x_n, k) \)
  - If \( \hat{y} \neq y_n \): \( w = w - \phi(x_n, \hat{y}) + \phi(x_n, y_n) \)

- When does this make an update?

<table>
<thead>
<tr>
<th>Pro</th>
<th>Md</th>
<th>Vb</th>
<th>Dt</th>
<th>Nn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pro</td>
<td>Md</td>
<td>Md</td>
<td>Dt</td>
<td>Vb</td>
</tr>
<tr>
<td>Pro</td>
<td>Md</td>
<td>Md</td>
<td>Dt</td>
<td>Nn</td>
</tr>
<tr>
<td>Pro</td>
<td>Md</td>
<td>Nn</td>
<td>Dt</td>
<td>Md</td>
</tr>
<tr>
<td>Pro</td>
<td>Md</td>
<td>Nn</td>
<td>Dt</td>
<td>Nn</td>
</tr>
<tr>
<td>Pro</td>
<td>Md</td>
<td>Vb</td>
<td>Dt</td>
<td>Md</td>
</tr>
<tr>
<td>Pro</td>
<td>Md</td>
<td>Vb</td>
<td>Dt</td>
<td>Vb</td>
</tr>
<tr>
<td>I</td>
<td>can</td>
<td>can</td>
<td>a</td>
<td>can</td>
</tr>
</tbody>
</table>
From perceptron to margins

Maximize Margin

\[
\min_{\mathbf{w}, \xi} \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_n \xi_n
\]

\[s.t.\quad y_n \mathbf{w} \cdot \phi(x_n) + \xi_n \geq 1, \quad \forall n\]

Each point is correctly classified, modulo \( \xi \)

Minimize Errors

\[
\min_{\mathbf{w}, \xi} \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_n \xi_n
\]

\[s.t.\quad y_n \mathbf{w} \cdot \phi(x_n, y_n)
\]

\[
-\mathbf{w} \cdot \phi(x_n, \hat{y}) + \xi_n \geq 1, \quad \forall n, \hat{y} \neq y_n
\]

Each true output is more highly ranked, modulo \( \xi \)

Response for truth

Response for other

[Taskar+al. JMLR05; Tshochandaritis, JMLR05]
From perceptron to margins

\[ \min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_n \xi_n, \hat{y} \]

Response for truth:

\[ s.t. \ w \cdot \phi(x_n, y_n) - w \cdot \phi(x_n, \hat{y}) + \xi_n \geq 1, \forall n, \hat{y} \neq y_n \]

Response for other:

Each true output is more highly ranked, modulo \( \xi \)
Ranking margins

- Some errors are worse than others...

```
Pro  Md  Vb  Dt  Nn
Pro  Md  Md  Dt  Vb
Pro  Md  Md  Dt  Nn
Pro  Md  Nn  Dt  Md
Pro  Md  Nn  Dt  Nn
Pro  Md  Vb  Dt  Md
Pro  Md  Vb  Dt  Vb
I    can can a  can
```
Accounting for a loss function

- Some errors are worse than others...

![Diagram showing the concept of margin in a loss function]

Margin of $l(y, y')$

I can can a can
Accounting for a loss function

\[ \phi(x_1, y_1) \]

\[ \phi(x_2, \hat{y}) \]

\[ \phi(x_1, \hat{y}) \]

\[ \phi(x_2, \hat{y}) \]

\[ \phi(x_1, y_2) \]

\[ w \cdot \phi(x_n, y_n) - w \cdot \phi(x_n, \hat{y}) + \xi_n \geq 1 \]

\[ w \cdot \phi(x_n, y_n) - w \cdot \phi(x_n, \hat{y}) + \xi_n \geq l(y_n, \hat{y}) \]
Augmented argmax for sequences

- Add “loss” to each wrong node!
Stochastically optimizing Markov nets

For n=1..N:

- Augmented Viterbi:
  \[
  \hat{y} = \arg \max_k w \cdot \phi(x_n, k) + l(y_n, k)
  \]

- If \( \hat{y} \neq y_n \):
  \[
  w = w - \phi(x_n, \hat{y}) + \phi(x_n, y_n)
  \]

- \[
  w = \left(1 - \frac{1}{CN}\right)w
  \]
Learning to Search
Argmax is hard!

- Classic formulation of structured prediction:

\[
\text{score}(x, y) = \text{something we learn to make “good” \(x, y\) pairs score highly}
\]

- At test time:

\[
f(x) = \arg\max_{y \in Y} \text{score}(x, y)
\]

- Combinatorial optimization problem
  - Efficient only in very limiting cases
  - Solved by heuristic search: beam + A* + local search
Incremental parsing, early 90s style

Train a classifier to make decisions

S

Right

VP

Left

Up

Right

Left

Right

Unary

I / Pro

can / Md

can / Vb

a / Dt

can / Nn

NP

NP

NP
Incremental parsing, mid 2000s style

Train a classifier to make decisions

S
VP
NP
I / Pro
can / Md
can / Vb

NP
a / Dt
can / Nn

[Collins+Roark, ACL04]
Learning to beam-search

- For n=1..N:
  - Viterbi:
    \[ \hat{y} = \arg\max_k w \cdot \phi(x_n, k) \]
  - If \( \hat{y} \neq y_n \)
    \[ w = w - \phi(x_n, \hat{y}) + \phi(x_n, y_n) \]

[Collins+Roark, ACL04]
Learning to beam-search

For n=1..N:
- Run beam search until truth falls out of beam
- Update weights immediately!
Learning to beam-search

For n=1..N:
- Run beam search until truth falls out of beam
- Update weights immediately!
- Restart at truth

[D+Marcu, ICML05; Xu+al, JMLR09]
Incremental parsing results

[Collins+Roark, ACL04]
Generic Search Formulation

- **Search Problem:**
  - Search space
  - Operators
  - Goal-test function
  - Path-cost function

- **Search Variable:**
  - Enqueue function

Varying the **Enqueue** function can give us DFS, BFS, beam search, A* search, etc...

- **nodes :=**
  - `MakeQueue(S0)`

- **while** nodes is not empty
  - **if** node is a goal state
    - **return** node
  - next := Operators(node)
  - nodes :=
    - Enqueue(nodes, next)

- **fail**
Online Learning Framework (LaSO)

- nodes := MakeQueue(S0)
- while nodes is not empty
  - node := RemoveFront(nodes)
  - if none of \( \{\text{node}\} \cup \text{nodes} \) is y-good or node is a goal & not y-good
    - sibs := siblings(node, y)
    - w := update(w, x, sibs, \( \{\text{node}\} \cup \text{nodes} \))
    - nodes := MakeQueue(sibs)
  - else
    - if node is a goal state return w
    - next := Operators(node)
    - nodes := Enqueue(nodes, next)

Monotonicity: for any node, we can tell if it can lead to the correct solution or not

If we erred...

Where should we have gone?

Update our weights based on the good and the bad choices

Continue search...
Search-based Margin

- The margin is the amount by which we are correct:

\[ \bar{u}^T \Phi(x, g_1) \]

\[ \bar{u}^T \Phi(x, g_2) \]

\[ \bar{u}^T \Phi(x, b_1) \]

\[ \bar{u}^T \Phi(x, b_2) \]

- Note that the margin and hence linear separability is also a function of the search algorithm!
Syntactic chunking Results

F-Score

Training Time (minutes)

[Collins 2002]

[Zhang+Damerau+Johnson 2002]; timing unknown

4 min

24 min

22 min

33 min
Tagging+chunking results

Joint tagging/chunking accuracy vs. Training Time (hours) [log scale]

- Large Margin (beam 25/50): 23 min
- Large Margin (beam 10): 3 min
- Sutton model: 7 min
- Sutton: 1 min

[Sutton+McCallum, ICML05; Xu+al, JMLR09]
Variations on a beam

- **Observation:**
  - We needn't use the same beam size for training and decoding
  - Varying these values independently yields:

<table>
<thead>
<tr>
<th>Training Beam</th>
<th>Decoding Beam</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>93.9</td>
</tr>
<tr>
<td>5</td>
<td>90.5</td>
</tr>
<tr>
<td>10</td>
<td>89.5</td>
</tr>
<tr>
<td>25</td>
<td>88.7</td>
</tr>
<tr>
<td>50</td>
<td>88.4</td>
</tr>
</tbody>
</table>
What if our model sucks?

- Sometimes our model \textit{cannot} produce the “correct” output
  - canonical example: machine translation

```
Model Outputs
```

```
Current Hypothesis
```

```
Good Outputs
```

```
Best achievable output
```

```
“Local” update
```

```
“Bold” update
```

```
N-best list or “optimal decoding” or ...
```

[Och, ACL03; Liang+al, ACL06]
Local versus bold updating...

Machine Translation Performance (Bleu)

- Monotonic
  - Bold: 34.5
  - Local: 34.5
  - Pharoah: 35.5

- Distortion
  - Bold: 33.5
  - Local: 34.5
  - Pharoah: 35.5
Integrating search and learning

Input: Le homme mange l' croissant.
Output: The man ate a croissant.

Hyp: The man ate a croissant.
Cov: Le homme mange l' croissant.

Hyp: The man ate a fox.
Cov: Le homme mange l' croissant.

Hyp: The man ate happy.
Cov: Le homme mange l' croissant.

Hyp: The man ate happy.
Cov: Le homme mange l' croissant.

Hyp: The man ate a croissant.
Cov: Le homme mange l' croissant.

Classifier 'h'
Reducing search to classification

- **Natural chicken and egg problem:**
  - Want $h$ to get low expected future loss
  - ... on future decisions made by $h$
  - ... and starting from states visited by $h$

- **Iterative solution**

  ![Diagram showing iterative solution with hypotheses and losses](image)

  - Hyp: The man ate
    Cov: Le homme mange l’croissant.
  - Hyp: The man ate a croissant
    Cov: Le homme mange l’croissant.
  - Hyp: The man ate a fox
    Cov: Le homme mange l’croissant.
  - Hyp: The man ate happy
    Cov: Le homme mange l’croissant.
  - Hyp: The man ate a croissant
    Cov: Le homme mange l’croissant.
  - Loss = 0
  - Loss = 1.8
  - Loss = 1.2
  - Loss = 0.5
  - Loss = 0

[D+Langford+Marcu, MLJ09]
Theoretical results

**Theorem:** After $2T^3 \ln T$ iterations, the loss of the learned policy is bounded as follows:

$$L(h) \leq L(h_0) + 2T \ln T l_{avg} + (1 + \ln T) \frac{c_{max}}{T}$$

- **Loss of the optimal policy**
- **Average multiclass classification loss**
- **Worst case per-step loss**
Example task: summarization

That's perfect!

Standard approach is sentence extraction, but that is often deemed to "coarse" to produce good, very short summaries. We wish to also drop words and phrases => document compression

Argentina was still obsessed with the Falkland Islands even in 1994, 12 years after its defeat in the 74-day war with Britain. The country's overriding foreign policy aim continued to be winning sovereignty over the islands.

The Falkland islands war, in 1982, was fought between Britain and Argentina.
Structure of search

- Lay sentences out sequentially
- Generate a dependency parse of each sentence
- Mark each root as a frontier node
- Repeat:
  - Choose a frontier node node to add to the summary
  - Add all its children to the frontier
  - Finish when we have enough words

Argentina was still obsessed with the Falkland Islands even in 1994, 12 years after its defeat in the 74-day war with Britain. The country's overriding foreign policy aim continued to be winning sovereignty over the islands.
Sentence Extraction + Compression:
Argentina and Britain announced an agreement, nearly eight years after they fought a 74-day war over a populated archipelago off Argentina's coast. Argentina gets out the red carpet, official royal visitor since the end of the Falklands war in 1982.

Vine Growth (Searn):
Argentina and Britain announced to restore full ties, eight years after they fought a 74-day war over the Falkland islands. Britain invited Argentina's minister Cavallo to London in 1992 in the first official visit since the Falklands war in 1982.

6 Diplomatic ties restored
5 Major cabinet member visits
5 Exchanges were in 1992
3 War between Britain and Argentina
3 Falkland war was in 1982
3 Cavallo visited UK
3 War was 74-days long
Perceptron vs. LaSO vs. Searn

- Incremental perceptron
- LaSO
- Searn

Un-learnable decision
Take-home messages

- If you can predict (ie., solve argmax) you can learn (use structured perceptron)

- If you can do loss-augmented search, you can do max margin (add two lines of code to perceptron)

- If you can do beam search, you can learn using LaSO (with no loss function)

- If you can do beam search, you can learn using Searn (with any loss function)
Coffee Break!!!
Refresher on Reinforcement Learning
Reinforcement learning

- Basic idea:
  - Receive feedback in the form of rewards
  - Agent’s utility is defined by the reward function
  - Must learn to act to maximize expected rewards
  - Change the rewards, change the learned behavior

- Examples:
  - Playing a game, reward at the end for outcome
  - Vacuuming, reward for each piece of dirt picked up
  - Driving a taxi, reward for each passenger delivered
Markov decision processes

What are the values (expected future rewards) of states and actions?

\[ V(s)^* = 30 \]

\[ Q(s,a1)^* = 30 \]

\[ Q(s,a2)^* = 23 \]

\[ Q(s,a3)^* = 17 \]
Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s, a, s')$
  - Prob that $a$ from $s$ leads to $s$
  - i.e., $P(s' | s, a)$
  - Also called the model
  - A reward function $R(s, a, s')$
  - Sometimes just $R(s)$ or $R(s')$
  - A start state (or distribution)
  - Maybe a terminal state

- MDPs are a family of non-deterministic search problems
- Total utility is one of:
  \[
  \sum_{t} r_t \quad \text{or} \quad \sum_{t} \gamma^t r_t
  \]
Solving MDPs

- In deterministic single-agent search problem, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi(s)$
  - A policy gives an action for each state
  - Optimal policy maximizes expected if followed
  - Defines a reflex agent

Optimal policy when $R(s, a, s') = -0.04$ for all non-terminals $s$
Example Optimal Policies

\[ \begin{align*}
\text{R}(s) &= -0.01 \\
\text{R}(s) &= -0.03 \\
\text{R}(s) &= -0.4 \\
\text{R}(s) &= -2.0
\end{align*} \]
Optimal Utilities

- Fundamental operation: compute the optimal utilities of states \( s \) (all at once)

- Why? Optimal values define optimal policies!

- Define the utility of a state \( s \): 
  \[ V^*(s) = \text{expected return starting in } s \text{ and acting optimally} \]

- Define the utility of a q-state \((s,a)\): 
  \[ Q^*(s,a) = \text{expected return starting in } s, \text{ taking action } a \text{ and thereafter acting optimally} \]

- Define the optimal policy: 
  \[ \pi^*(s) = \text{optimal action from state } s \]
The Bellman Equations

- Definition of utility leads to a simple one-step lookahead relationship amongst optimal utility values:

  Optimal rewards = maximize over first action and then follow optimal policy

- Formally:

  \[ V^*(s) = \max_a Q^*(s, a) \]

  \[ Q^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]

  \[ V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]
Solving MDPs / memoized recursion

- Recurrences:

\[ V_0^*(s) = 0 \]

\[ V_i^*(s) = \max_a Q_i^*(s, a) \]

\[ Q_i^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_{i-1}^*(s') \right] \]

\[ \pi_i(s) = \arg \max_a Q_i^*(s, a) \]

- Cache all function call results so you never repeat work

- What happened to the evaluation function?
Q-Value Iteration

- Value iteration: iterate approx optimal values
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given $V_i^*$, calculate the values for all states for depth $i+1$:

  $V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left( R(s, a, s') + \gamma V_i(s') \right)$

- But Q-values are more useful!
  - Start with $Q_0^*(s,a) = 0$, which we know is right (why?)
  - Given $Q_i^*$, calculate the q-values for all q-states for depth $i+1$:

  $Q_{i+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left( R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right)$
RL = Unknown MDPs

- If we knew the MDP (i.e., the reward function and transition function):
  - Value iteration leads to optimal values
  - Q-value iteration leads to optimal Q-values
  - Will always converge to the truth

- Reinforcement learning is what we do when we do not know the MDP
  - All we observe is a trajectory
  - \((s_1,a_1,r_1, s_2,a_2,r_2, s_3,a_3,r_3, \ldots)\)
Q-Learning

- Learn $Q^*(s, a)$ values
  - Receive a sample $(s, a, s', r)$
  - Consider your old estimate: $Q(s, a)$
  - Consider your new sample estimate:
    $$Q^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$

- Incorporate the new estimate into a running average:
  $$sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$
  $$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) [sample]$$
Exploration / Exploitation

- Several schemes for forcing exploration
  - Simplest: random actions ($\epsilon$ greedy)
    - Every time step, flip a coin
    - With probability $\epsilon$, act randomly
    - With probability $1-\epsilon$, act according to current policy

- Problems with random actions?
  - You do explore the space, but keep thrashing around once learning is done
  - One solution: lower $\epsilon$ over time
  - Another solution: exploration functions
Q-Learning

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states:
    \[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]

- Very simple stochastic updates:
  \[ Q(s, a) \leftarrow Q(s, a) + \alpha [\text{error}] \]
  \[ w_i \leftarrow w_i + \alpha [\text{error}] f_i(s, a) \]
Inverse RL and Apprenticeship Learning
Inverse RL: Task

- Given:
  - measurements of an agent's behavior over time, in a variety of circumstances
  - if needed, measurements of the sensory inputs to that agent
  - if available, a model of the environment.

- Determine: the reward function being optimized

- Proposed by [Kalman68]
- First solution, by [Boyd94]
Why inverse RL?

- Computational models for animal learning
  - “In examining animal and human behavior we must consider the reward function as an unknown to be ascertained through empirical investigation.”

- Agent construction
  - “An agent designer [...] may only have a very rough idea of the reward function whose optimization would generate 'desirable' behavior.”
  - eg., “Driving well”

- Multi-agent systems and mechanism design
  - learning opponents’ reward functions that guide their actions to devise strategies against them
IRL from Sample Trajectories

- Optimal policy available through trajectories (e.g., driving a car)
- Want to find \( R_w(s) = w \phi(s) \) so the reward is linear
- \( V^\pi_w(s_0) \) be the value of the starting state

\[
\max_w \sum_{k=1}^{K} f \left( V^\pi^*_w(s_0) - V^\pi_k_w(s_0) \right)
\]

Warning: need to be careful to avoid trivial solutions!

How good does the “optimal policy” look?
How good does the some other policy look?
For $t = 1, 2, \ldots$

- **Inverse RL step:**
  Estimate expert’s reward function $R(s) = w^T \phi(s)$ such that under $R(s)$ the expert performs better than all previously found policies $\{\pi_i\}$.

- **RL step:**
  Compute optimal policy $\pi_t$ for the estimated reward $w$
Car Driving Experiment

- No explicit reward function at all!
- Expert demonstrates proper policy via 2 min. of driving time on simulator (1200 data points).
- 5 different “driver types” tried.
- Features: which lane the car is in, distance to closest car in current lane.
- Algorithm run for 30 iterations, policy hand-picked.
- Movie Time! (Expert left, IRL right)
“Nice” driver
“Evil” driver
Maxent IRL

Distribution over trajectories:

\[ P(\zeta) \]

Match the reward of observed behavior:

\[ \sum_\zeta P(\zeta) f_\zeta = f_{\text{dem}} \]

Maximizing the \textit{causal entropy} over trajectories given stochastic outcomes:

\[ \max H(P(\zeta)||O) \]

(Condition on random uncontrolled outcomes, but only \textbf{after} they happen)

\textbf{As uniform as possible}
Data collection

25 Taxi Drivers

Over 100,000 miles

Length
Speed
Road
Type
Lanes

Accidents
Construction
Congestion
Time of day
Predicting destinations....
Inverse Optimal Control
Planning as structured prediction
Maximum margin planning

- Let $\mu(s,a)$ denote the probability of reaching q-state $(s,a)$ under current model $w$

\[
\begin{align*}
\max_{w} & \quad \text{margin} & \quad \text{s.t.} & \quad \text{planner run with } w \text{ yields human output} \\
\min_{w} & \quad \frac{1}{2} \|w\|^2 & \quad \text{s.t.} & \quad \text{Q-state visitation frequency by human} \\
& & & \mu(s,a)w \cdot \phi(x_n,s,a) \\
& & & -\hat{\mu}(s,a)w \cdot \phi(x_n,s,a) \geq 1 \\
& & & , \quad \forall n,s,a
\end{align*}
\]

- Q-state visitation frequency by planner

- All trajectories, and all q-states

[1] Ratliff+al. NIPS05
Optimizing MMP

M³N Objective

SOME MATH

For n=1..N:

- Augmented planning:
  Run A* on current (augmented) cost map to get q-state visitation frequencies \( \mu(s,a) \)

- Update: \( w = w + \sum_s \sum_a [\hat{\mu}(s,a) - \mu(s,a)] \phi(x_n, s, a) \)

- Shrink: \( w = \left(1 - \frac{1}{CN}\right)w \)
Maximum margin planning movies
Parsing via inverse optimal control

- State space = all partial parse trees over the full sentence labeled “S”
- Actions: take a partial parse and split it anywhere in the middle
- Transitions: obvious
- Terminal states: when there are no actions left
- Reward: parse score at completion
Parsing via inverse optimal control

[Diagram showing bar charts for Small, Medium, and Large categories. Colors represent different learning methods: Maximum Likelihood (blue), Projection (brown), Perceptron (light blue), Apprenticeship Learning (purple), Maximum Margin (green), Maximum Entropy (red), Policy Matching (yellow).]
Learning to Search
Learning to search
**Learch**

Until converged do:
- Initialize modification set to empty
- For each example, add cost function modifications:
  - Make **loss-augmented prediction** using current cost
  - Update data set: Label desired feature vector as -1 and predicted feature vector as +1
- Generalize using a least-squares regression
- Add it to the current cost function

[Ratliff+al, AutRobots09]
Discussion
Relationship between SP and IRL

- Formally, they're (nearly) the same problem
  - See humans performing some task
  - Define some loss function
  - Try to mimic the humans

- Difference is in philosophy:
  - (I)RL has little notion of beam search or dynamic programming
  - SP doesn't think about separating reward estimation from solving the prediction problem
  - (I)RL has to deal with stochasticity in MDPs
Important Concepts

- Search and loss-augmented search for margin-based methods
- Bold versus local updates for approximate search
- Training on-path versus off-path
- Stochastic versus deterministic worlds
- Q-states / values
- Learning reward functions vs. matching behavior
Hal's Wager

- Give me a structured prediction problem where:
  - Annotations are at the lexical level
  - Humans can do the annotation with reasonable agreement
  - You give me a few thousand labeled sentences

- Then I can learn reasonably well...
  - ...using one of the algorithms we talked about

- Why do I say this?
  - Lots of positive experience
  - I'm an optimist
  - I want your counter-examples!
Open problems

- How to do SP when argmax is intractable….
  - Bad: simple algorithms diverge  [Kulesza+Pereira, NIPS07]
  - Good: some work well  [Finley+Joachims, ICML08]
  - And you can make it fast!  [Meshi+al, ICML10]

- How to do SP with delayed feedback (credit assignment)
  - Kinda just works sometimes  [D, ICML09; Chang+al, ICML10]
  - Generic RL also works  [Branavan+al, ACL09; Liang+al, ACL09]

- What role does structure actually play?
  - Little: only constraints outputs  [Punyakanok+al, IJCAI05]
  - Little: only introduces non-linearities  [Liang+al, ICML08]
  - Lots: ???
**Things I have no idea how to solve...**

\[
\textbf{all} : (a \rightarrow \text{Bool}) \rightarrow [a] \rightarrow \text{Bool}
\]

Applied to a predicate and a list, returns `True` if all elements of the list satisfy the predicate, and `False` otherwise.

```haskell
%module main:MyPrelude
%data main:MyPrelude.MyList aadj =
    {main:MyPrelude.Nil;
     main:MyPrelude.Cons aadj ((main:MyPrelude.MyList aadj))};
%rec
{main:MyPrelude.myzuall :: %forall tadA . (tadA ->
    ghczmprim:GHCziBool.Bool)
->
(main:MyPrelude.MyList tadA) ->
    ghczmprim:GHCziBool.Bool =

\ @ tadA
 (padk::tadA -> ghczmprim:GHCziBool.Bool)
 (dsddE::(main:MyPrelude.MyList tadA)) ->
 %case ghczmprim:GHCziBool.Bool dsddE
 %of (wildBl::(main:MyPrelude.MyList tadA))
 {main:MyPrelude.Nil ->
    ghczmprim:GHCziBool.True;
 main:MyPrelude.Cons
 (xadm::tadA) (xsadn::(main:MyPrelude.MyList tadA)) ->
 %case ghczmprim:GHCziBool.Bool (padk xadm)
 %of (wild1Xc::ghczmprim:GHCziBool.Bool)
 {ghczmprim:GHCziBool.False ->
   ghczmprim:GHCziBool.False;
 ghczmprim:GHCziBool.True ->
   main:MyPrelude.myzuall @ tadA padk xsadn})};
```
(s1) A father had a family of sons who were perpetually quarreling among themselves. (s2) When he failed to heal their disputes by his exhortations, he determined to give them a practical illustration of the evils of disunion; and for this purpose he one day told them to bring him a bundle of sticks. (s3) When they had done so, he placed the faggot into the hands of them to break it in pieces. (s4) They could not do it, for they were weak in strength, and were unable to break the faggot, took the sticks again put them in their hands, and then they could break them easily. (s6) "My sons, if you will live together, you will be as strong as ten men; but if you will be divided, you will be broken as by the hand of a stranger."
Software

- Sequence labeling
  - Mallet  http://mallet.cs.umass.edu
  - CRF++  http://crfpp.sourceforge.net

- Search-based structured prediction
  - LaSO  http://hal3.name/TagChunk
  - Searn  http://hal3.name/searn

- Higher-level “feature template” approaches
  - Alchemy  http://alchemy.cs.washington.edu
  - Factorie  http://code.google.com/p/factorie
Summary

- Structured prediction is easy if you can do argmax search (esp. loss-augmented!)
- Label-bias can kill you, so iterate (Searn)
- Stochastic worlds modeled by MDPs
- IRL is all about learning reward functions
- IRL has fewer assumptions
  - More general
  - Less likely to work on easy problems
- We're a long way from a complete solution
- Hal's wager: we can learn pretty much anything

Thanks! Questions?
References

See also:

http://braque.cc/ShowChannel?handle=P5BVAC34
Stuff we talked about explicitly

Other good stuff