From Structured Prediction to Inverse Reinforcement Learning

Hal Daumé III
Computer Science
University of Maryland
me@hal3.name
A Tutorial at AAAI 2011
San Francisco, California
Monday, 8 August 2011

Examples of structured problems

Acknowledgements

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

Discussions/Feedback:
MLRG Spring 2010

Examples of demonstrations
# 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 croisade 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>
</tbody>
</table>

...many more...

## Structured prediction 101

Learn a function mapping inputs to complex outputs:

\[
f : X \rightarrow Y
\]

### 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
- Structure without Structure
  - Stacking
  - Structure compilation

Outline: Part II
- Learning to Search
  - Incremental parsing
  - Learning to queue
- Refresher on Markov Decision Processes
- Inverse Reinforcement Learning
  - Determining rewards given policies
  - Maximum margin planning
- Learning by Demonstration
  - Search
  - Dagger
- Discussion

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 $X \times Y$
    - A loss function over $Y \times Y$
    - 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

\[
\begin{align*}
W= & \text{dear} : 1 \\
W= & \text{sir} : 1 \\
W= & \text{this} : 2 \\
\ldots & \\
W= & \text{wish} : 0 \\
\ldots & \\
\text{MISSPELLED} & : 2 \\
\text{NAMELESS} & : 1 \\
\text{ALL\_CAPS} & : 0 \\
\text{NUM\_URLS} & : 0 \\
\ldots & 
\end{align*}
\]

- 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

\[
\begin{align*}
\Sigma_i w_i \Phi_i(x) \\
\text{“free money”}
\end{align*}
\]

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:
  \[
w = w + y \phi(x)
\]

Why does that update work?

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

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

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

\[
\begin{align*}
\max_{\mathbf{w}} & \quad \text{margin} \quad s.t. \quad \text{all points are correctly classified} \\
\max_{\mathbf{w}} & \quad \text{margin} \quad s.t. \quad y_n \mathbf{w} \cdot \phi(x_n) \geq 1 \quad \forall n \\
\min_{\mathbf{w}} & \quad \|\mathbf{w}\|^2 \quad s.t. \quad y_n \mathbf{w} \cdot \phi(x_n) \geq 1 \quad \forall n
\end{align*}
\]

Support vector machines with slack

- Explicitly optimize the margin
- Allow some “noisy” points to be misclassified

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

Batch versus stochastic optimization

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

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

Stochastically optimized SVMs

SVM Objective

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

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

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

Perceptron with multiple classes

- Store separate weight vector for each class $w_1, w_2, ..., w_K$
- For $n=1..N$:
  - Predict:
    \[ \hat{y} = \arg \max_k w_k \cdot \phi(x_n) \]
  - If $\hat{y} \neq y_n$
    \[ w_\hat{y} = w_\hat{y} - \phi(x_n) \quad w_y = w_y + \phi(x_n) \]

Perceptron with multiple classes v2

- Originally:
  - $w_1$, $w_2$, $w_3$
  - $w$ (unstructured)
- For $n=1..N$:
  - Predict:
    \[ \hat{y} = \arg \max_k w_k \cdot \phi(x_n) \]
    \[ \hat{y} = \arg \max_k w \cdot \phi(x_n, k) \]
    - If $\hat{y} \neq y_n$
      \[ w_\hat{y} = w_\hat{y} - \phi(x_n) \quad w = w - \phi(x_n, \hat{y}) \]
      \[ w_y = w_y + \phi(x_n) + \phi(x_n, y_n) \]

Perceptron with multiple classes v2: Example

- Originally:
  - $w_1$, $w_2$, $w_3$
  - $w$ (unstructured)
- For $n=1..N$:
  - Predict:
    \[ \hat{y} = \arg \max_k w_\text{spam} \cdot \phi(x_n, \text{spam}) \]
    \[ \hat{y} = \arg \max_k w_\text{ham} \cdot \phi(x_n, \text{ham}) \]
    - If $\hat{y} \neq y_n$
      \[ w_\hat{y} = w_\hat{y} - \phi(x_n, \text{spam}) \quad w = w - \phi(x_n, \hat{y}) \]
      \[ w_y = w_y + \phi(x_n, \text{spam}) + \phi(x_n, y_n) \]
Features for structured prediction

- Allowed to encode *anything* you want

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

\[ \begin{align*}
I_{\text{Pro}} : 1 & \quad <s>-\text{Pro} : 1 & \quad \text{has_verb} : 1 \\
\text{can}_\text{Md} : 1 & \quad \text{Pro-Md} : 1 & \quad \text{has_nn_lft} : 0 \\
\text{can}_\text{Vb} : 1 & \quad \text{Md-Vb} : 1 & \quad \text{has_n_lft} : 1 \\
a_{\text{Dt}} : 1 & \quad \text{Vb-Dt} : 1 & \quad \text{has_n_rgt} : 1 \\
\text{can}_\text{Nn} : 1 & \quad \text{Dt-Nn} : 1 & \quad \text{has_n_rgt} : 1 \\
\ldots & & \ldots
\end{align*} \]

- Output features, Markov features, other features

**Argmax for sequences**

- If we only have output and Markov features, we can use Viterbi algorithm:

\[ w[\text{Pro-Pro}] \]

\[ w[I_{\text{Pro}}] \]

\[ w[I_{\text{Md}}] \]

\[ w[I_{\text{Vb}}] \]

(plus some work to account for boundary conditions)

**Structured perceptron**

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

**Structured perceptron as ranking**

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

- When does this make an update?
From perceptron to margins

Maximize Margin
Minimize Errors

\[
\min_{\mathbf{w}, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \xi_n, \quad s.t. \quad y_n \mathbf{w} \cdot \phi(x_n) + \xi_n \geq 1, \quad \forall n
\]

Response for truth
Response for other

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

From perceptron to margins

\[
\min_{\mathbf{w}, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \xi_n, \quad s.t. \quad \mathbf{w} \cdot \phi(x_n, y_n) - \mathbf{w} \cdot \phi(x_n, \hat{y}) + \xi_n \geq 1, \quad \forall n, \hat{y} \neq y_n
\]

Response for truth
Response for other

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

Ranking margins

- Some errors are worse than others...

<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>Nn</td>
</tr>
</tbody>
</table>

Margin of one

Accounting for a loss function

- Some errors are worse than others...

<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>Vb</td>
<td>Nn</td>
<td>Dt</td>
</tr>
<tr>
<td>Pro</td>
<td>Md</td>
<td>Vb</td>
<td>Nn</td>
<td>Dt</td>
</tr>
<tr>
<td>Pro</td>
<td>Md</td>
<td>Vb</td>
<td>Nn</td>
<td>Nn</td>
</tr>
<tr>
<td>Nn</td>
<td>Md</td>
<td>Vb</td>
<td>Nn</td>
<td>Dt</td>
</tr>
</tbody>
</table>

Margin of \(l(y, \hat{y})\)
Accounting for a loss function

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

is equivalent to

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

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

Augmented argmax for sequences

- Add “loss” to each wrong node!

Stochastically optimizing Markov nets

M3N Objective

\[ \text{SOME MATH} \]

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

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

Stacking

- Structured models: accurate but slow
- Independent models: less accurate but fast
- Stacking: multiple independent models
Training a stacked model

- Train independent classifier $f_1$ on input features
- Train independent classifier $f_2$ on input features + $f_1$’s output
- Danger: overfitting!
- Solution: cross-validation

Do we really need structure?

- **Structured models**: accurate but slow
- **Independent models**: less accurate but fast
- **Goal**: transfer power to get fast+accurate
- **Questions**: are independent models...
  - … expressive enough? (approximation error)
  - … easy to learn? (estimation error)

“Compiling” structure out

```
CRF($f_1$) POS: 95.0% NER: 75.3%
IRL($f_1$) POS: 91.7% NER: 69.1%

$f_1$ = words/prefixes/suffixes/forms
```

“Compiling” structure out

```
CRF($f_1$) POS: 95.0% NER: 75.3%
IRL($f_1$) POS: 91.7% NER: 69.1%

IRL($f_2$) POS: 94.4% NER: 66.2%
CompIRL($f_2$) POS: 95.0% NER: 72.7%
```

$Liang+D+Klein, ICM^L08$
Decomposition of errors

\text{CRF}(f_1): p_C

- coherence
- marginalized CR
- nonlinearity

\text{IRL}(f_\infty): p_{A^*}

- global information

\text{IRL}(f_2): p_{1^*}

\text{Theorem:}
\text{KL}(p_C \parallel p_{1^*}) = \text{KL}(p_C \parallel p_{MC}) + \text{KL}(p_{MC} \parallel p_{A^*}) + \text{KL}(p_{A^*} \parallel p_{1^*})

Structure compilation results

\begin{itemize}
\item Part of speech
\item Named Entity
\itemParsing
\end{itemize}

- **Structured**
- **Independent**

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
- Structure without Structure
  - Stacking
  - Structure compilation
Learning to Search

Argmax is hard!

- Classic formulation of structured prediction:
  \[ \text{score}(x, y) = \text{something we learn to make “good” } x, y \text{ pairs score highly} \]

- 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


Order these words: bart better I madonna say than ,
Argmax is hard!

- Classical score functions
- At test time,
f(x, y)

Combinatorial optimization problem
- Efficient only in very limiting cases
- Solved by heuristic search: beam + A* + local search

[Order these words: bart better I madonna say than, best search (32.3): I say better than bart madonna, original (41.6): better bart than madonna, I say]


Argmax is hard!

- Classical score functions
- At test time,
f(x, y)

Combinatorial optimization problem
- Efficient only in very limiting cases
- Solved by heuristic search: beam + A* + local search

[Order these words: bart better I madonna say than, best search (32.3): I say better than bart madonna, original (41.6): better bart than madonna, I say]

[Order these words: bart better I madonna say than, best search (51.6): and so could really be a neural apparently thought things as dissimilar firing two identical]


Incremental parsing, early 90s style

Train a classifier to make decisions

[Sagerman, ACL95]
Incremental parsing, mid 2000s style

Learning to beam-search

For \( n = 1 \ldots N \):
- Run beam search until truth falls out of beam
- Update weights immediately!

Learning to beam-search

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

Train a classifier to make decisions
Incremental parsing results

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

Search-based Margin

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

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

- Update our weights based on the good and the bad choices

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

Tagging+chunking results

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>Decoding Beam</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>25</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Beam</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>93.9</td>
<td>92.8</td>
<td>91.9</td>
<td>91.3</td>
<td>90.9</td>
</tr>
<tr>
<td>5</td>
<td>90.5</td>
<td>94.3</td>
<td>94.4</td>
<td>94.1</td>
<td>94.4</td>
</tr>
<tr>
<td>10</td>
<td>89.5</td>
<td>94.3</td>
<td>94.4</td>
<td>94.2</td>
<td>94.2</td>
</tr>
<tr>
<td>25</td>
<td>88.7</td>
<td>94.2</td>
<td>94.5</td>
<td>94.3</td>
<td>94.3</td>
</tr>
<tr>
<td>50</td>
<td>88.4</td>
<td>94.2</td>
<td>94.4</td>
<td>94.2</td>
<td>94.4</td>
</tr>
</tbody>
</table>

What if our model sucks?

> Sometimes our model cannot produce the “correct” output
> - canonical example: machine translation

“Bold” update

“Local” update

Reference

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

Best achievable output
**Local versus bold updating...**

![Graph showing machine translation performance (Bleu) for Monotonic and Distortion categories, with bars for Bold, Local, and Pharaoah]

**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?

![Markov decision process diagram with states and transitions]

\[ V(s) = 30 \]

\[ Q(s,a1) = 30, \quad Q(s,a2) = 23, \quad Q(s,a3) = 17 \]
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:
  $$\sum_t r_t$$
  $$\sum_t y^t r_t$$

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

Example Optimal Policies

- Fundamental operation: compute the optimal utilities of states $s$ (all at once)
  - Why? Optimal values define optimal policies!

Optimal Utilities

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

- Formally:

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
  - 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
    
    \[ (s_1, a_1, r_1, \ s_2, a_2, r_2, \ s_3, a_3, r_3, \ \ldots) \]

- Many algorithms exist for this problem; see Sutton+Barto's excellent book!
### 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

### Inverse Reinforcement 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:
  
  $Q(s,a) \leftarrow Q(s,a) + \alpha [error]$

  $w_i \leftarrow w_i + \alpha [error] f_i(s,a)$
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 Trajectory

- Optimal policy available through sample trajectory (eg., driving a car)
- Want to find Reward function that makes this policy look as good as possible
- Write $R_w(s) = w \phi(s)$ so the reward is linear
- $V^\pi_w(s_0)$ be the value of the starting state

$$\max_w \sum_{k=1}^K f \left( V_w^\pi(s_0) - V_{w_k}(s_0) \right)$$

Warning: need to be careful to avoid trivial solutions!

Apprenticeship Learning via IRL

- 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)

“Ice” 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 causal entropy over trajectories given stochastic outcomes:

\[ \max \mathbb{H}(P(\zeta) || O) \]

(Condition on random uncontrolled outcomes, but only after they happen)
Data collection

Predicting destinations....

Planning as structured prediction

Maximum margin planning

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

\[
\text{max} \quad \frac{1}{2} ||w||^2 \\
\text{s.t.} \quad \mu(s,a)w \cdot \phi(x_n,s,a) \geq 1, \quad \forall n,s,a
\]

\[
\text{min} \quad \frac{1}{2} ||w||^2 \\
\text{s.t.} \quad -\hat{\mu}(s,a)w \cdot \phi(x_n,s,a) \geq 1, \quad \forall n,s,a
\]
Optimizing MMP

**MMP Objective**

For n=1..N:

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

- Update: $w = w + \sum s \sum a \left[ \hat{\mu}(s,a) - \mu(s,a) \right] \phi(x_n, s,a)$

- 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
Learning by Demonstration

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

```
\text{Input: Le homme mange l' croissant.}
\text{Output: The man ate a croissant.}
```

<table>
<thead>
<tr>
<th>Hyp: The man ate a fox</th>
<th>Cov: Le homme mange l' croissant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss = 1.2</td>
<td>Loss = 0</td>
</tr>
<tr>
<td>Hyp: The man ate happy</td>
<td>Cov: Le homme mange l' croissant.</td>
</tr>
<tr>
<td>Loss = 1.8</td>
<td>Loss = 0</td>
</tr>
</tbody>
</table>

Reduction for Structured Prediction

- Idea: view structured prediction in light of search

```
\text{Loss function:}
L([N V R], [N V R]) = 0
L([N V R], [N V V]) = 1/3
```

Each step here looks like it could be represented as a weighted multi-class problem.

Can we formalize this idea?
Reducing Structured Prediction

Desired: good policy on test data (i.e., given only input string)

Key Assumption: Optimal Policy for training data
Given: input, true output and state;
Return: best successor state
Weak!

How to Learn in Search

I sing merrily

I sing merrily

I sing merrily

I sing merrily

How to Learn in Search

I sing merrily

I sing merrily

I sing merrily

I sing merrily

Searn

Algorithm: Searn-Learn(A, D^SP, loss, π*, β)
1: Initialize: π = π*
2: while not converged do
3: Sample: D ~ Searn(D^SP, loss, π)
4: Learn: h ← A(D)
5: Update: π ← (1 - β) π + β h
6: end while
7: return π without π*

Ingredients for Searn:
Input space (X) and output space (Y), data from X
Loss function (loss(y, y')) and features
“Optimal” policy π*(x, y_0)

Theorem: \( L(\pi_0) = L(\pi) + \text{loss}(T \ln T + c(1 + \ln T) / T) \)
But what about demonstrations?

- What did we assume before?

**Key Assumption:**
*Optimal Policy for training data*

**Given:** input, true output and state;
**Return:** best successor state

- We can have a *human* (or system) demonstrate, thus giving us an *optimal policy*

---

### DAgger: Dataset Aggregation

- Collect trajectories from expert $\pi^*$
- Dataset $D_0 = \{ (s, \pi^*(s)) \mid s \sim \pi^* \}$
- Train $\pi_1$ on $D_0$
- Collect new trajectories from $\pi_1$
  - But let the *expert* steer!
- Dataset $D_1 = \{ (s, \pi^*(s)) \mid s \sim \pi_1 \}$
- Train $\pi_2$ on $D_0 \cup D_1$

- In general:
  - $D_n = \{ (s, \pi^*(s)) \mid s \sim \pi_n \}$
  - Train $\pi_n$ on $\cup_{i<n} D_i$

---

### Experiments: Racing Game

**Input:**

- Resized to 25x19 pixels (1425 features)

**Output:**

- Steering in [-1,1]
Average falls per lap

Super Mario Bros.

From Mario AI competition 2009

Input:

Output:

Jump in \{0,1\}
Right in \{0,1\}
Left in \{0,1\}
Speed in \{0,1\}

Extracted 27K+ binary features from last 4 observations (14 binary features for every cell)

Training (expert)

Test-time execution (classifier)
Test-time execution (Dagger)

Average distance per stage

Perceptron vs. LaSO vs. Searn

Incremental perceptron
LaSO
Searn / DAgger

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

**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]

- Role of experts?
  - what if your expert isn't actually optimal?
  - what if you have more than one expert?
  - what if you only have trajectories, not the expert?

**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!

**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
Things I have no idea how to solve...

all : (a -> Bool) -> [a] -> 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 sadd =
  %rec
  (main:MyPrelude.myzuall :: forall tada . (tada ->
  ghczprmin:GHCziBool.Bool) ->
  (main:MyPrelude.MyList tada) ->
  ghczprmin:GHCziBool.Bool =
  \@ tada
    (padk::tada -> ghczprmin:GHCziBool.Bool)
    {dddE::(main:MyPrelude.MyList tada)} ->
    %case ghczprmin:GHCziBool.Bool {dddK}
      {wildB1::(main:MyPrelude.MyList tada)}
    {main:MyPrelude.Nil} ->
    ghczprmin:GHCziBool.True;
    main:MyPrelude.Cons
    {xadm::(main:MyPrelude.MyList tada)} ->
    %case ghczprmin:GHCziBool.Bool {padk xadm}
      {wildX::ghczprmin:GHCziBool.Bool}
    {false} ->
    ghczprmin:GHCziBool.False;
    ghczprmin:GHCziBool.Bool {true} ->
    main:MyPrelude.myzuall \@ tada padk xsadm};
```

Things I have no idea how to solve...

(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. The boys, using all their strength, and were able to do so, but when the faggot, took them again put them in pieces easily. (s6) "My sons, if you all stick to one another, you will be able to overcome all of your enemies; if you will be broken

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/Dagger)
- 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