Beyond Structured Prediction: Inverse Reinforcement Learning

Hal Daumé III

Computer Science
University of Maryland

me@hal3.name

A Tutorial at ACL 2011
Portland, Oregon

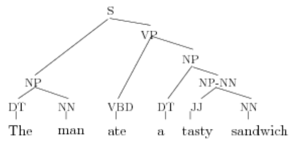
Sunday, 19 June 2011

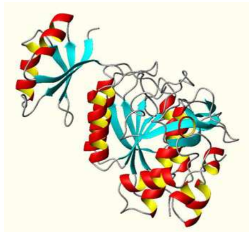
Acknowledgements

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

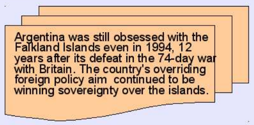
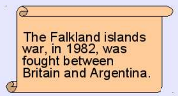
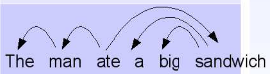
Discussions/Feedback:
MLRG Spring 2010

Ex





NLP as transduction

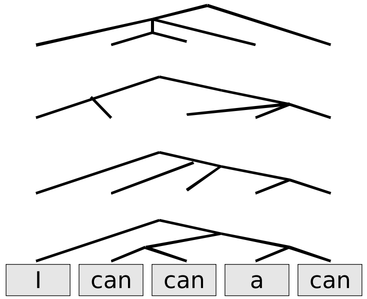
Task	Input	Output
Machine Translation	Ces deux principes se tiennent à la croisée de la philosophie, de la politique, de l'économie, de la sociologie et du droit.	Both principles lie at the crossroads of philosophy, politics, economics, sociology, and law.
Document Summarization		
Syntactic Analysis	The man ate a big sandwich.	
...many more...		

Structured prediction 101

Learn a function mapping inputs to complex outputs:

$$f : X \rightarrow Y$$

Input Space Decoding Output Space



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
  - Stacking
  - Incremental Parsing

Outline: Part II

- Refresher on Reinforcement Learning
  - Markov Decision Processes
  - Q learning
- Inverse Reinforcement Learning
  - Determining rewards given policies
  - Maximum margin planning
- Apprenticeship Learning
  - Searn
  - Dagger
- Discussion

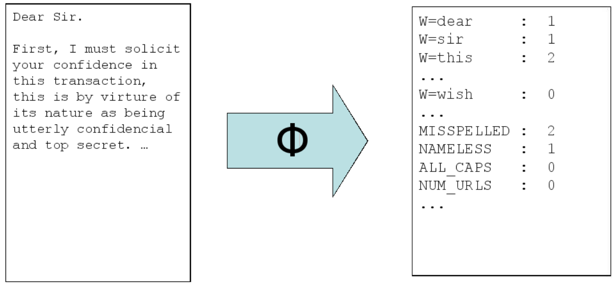
Refresher on Binary Classification

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

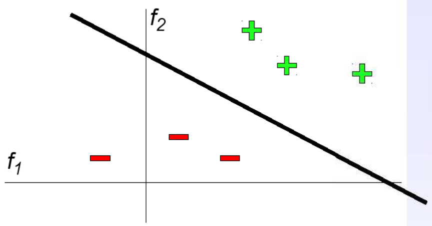
- A feature extractor Φ maps examples to vectors



- Feature vectors in NLP are frequently sparse

Linear models for binary classification

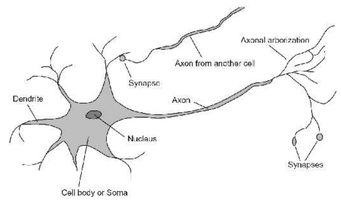
- Decision boundary is the set of "uncertain"points
- Linear decision boundaries are characterized by weight vectors

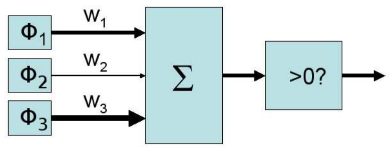


x	$\Phi(x)$	w	$\sum_i w_i \Phi_i(x)$
"free money"	BIAS : 1 free : 1 money : 1 the : 0 ...	BIAS : -3 free : 4 money : 2 the : 0 ...	(1)(−3) + (1)(4) + (1)(2) + (0)(0) + ... = 3

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:

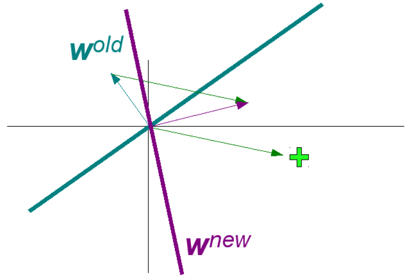
$$w = w + y\,\phi(x)$$

"Error" when:

$$y\, w \cdot \phi(x) \leq 0$$

Why does that update work?

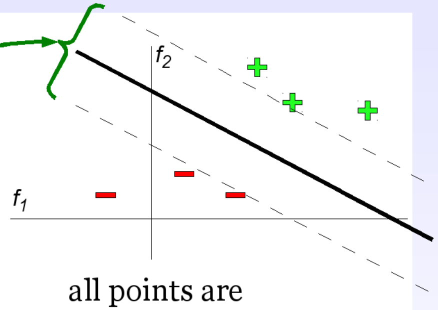
- When $y\, \mathbf{w}^{old} \cdot \phi(x) \le 0$, update $\mathbf{w}^{new} = \mathbf{w}^{old} + y\,\phi(x)$



$$y\,\mathbf{w}^{new}\phi(x) = y\left(\mathbf{w}^{old} + y\,\phi(x)\right)\phi(x)$$
$$= \underbrace{y\,\mathbf{w}^{old}\phi(x)}_{<0} + \underbrace{yy\,\phi(x)\phi(x)}_{>0}$$

Support vector machines

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



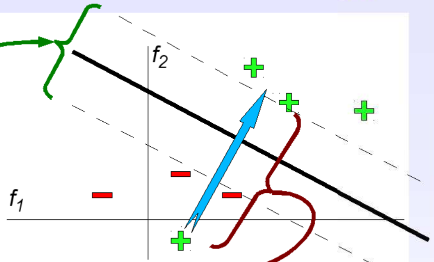
$$\max_{\mathbf{w}} \ \text{margin} \quad s.t. \quad \text{all points are correctly classified}$$

$$\max_{\mathbf{w}} \ \text{margin} \quad s.t. \quad y_n \mathbf{w}\cdot\phi(x_n) \ge 1 \ , \ \forall n$$

$$\min_{\mathbf{w}} \ \|\mathbf{w}\|^2 \quad s.t. \quad y_n \mathbf{w}\cdot\phi(x_n) \ge 1 \ , \ \forall n$$

Support vector machines with *slack*

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



$$\min_{\mathbf{w},\xi} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_n \xi_n$$
$$s.t. \quad y_n \mathbf{w}\cdot\phi(x_n) + \boxed{\xi_n} \ge 1 \ , \ \forall n$$
$$\xi_n \ge 0 \ , \ \forall n$$

Batch versus stochastic optimization

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

$$\min_{\mathbf{w},\xi} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_n \xi_n$$
$$s.t. \quad y_n \mathbf{w}\cdot\phi(x_n) + \xi_n \ge 1 \ , \ \forall n$$
$$\xi_n \ge 0 \ , \ \forall n$$

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

Stochastically optimized SVMs

SVM Objective

SOME MATH

Implementation Note:

Weight shrinkage is *SLOW.* Implement it lazily, at the cost of double storage.

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

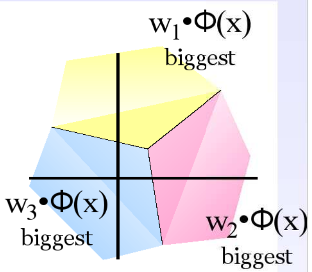
For n=1..N:

- If $y_n \boldsymbol{w} \cdot \phi(x_n) \leq 0$
  - $\boldsymbol{w} = \boldsymbol{w} + y_n \phi(x_n)$

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:

$$\hat{y} = \arg\max_k \boldsymbol{w_k} \cdot \phi(x_n)$$

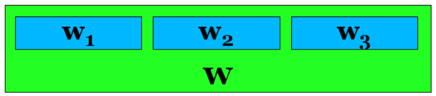
  - If $\hat{y} \neq y_n$:

$$\boldsymbol{w}_{\hat{y}} = \boldsymbol{w}_{\hat{y}} - \phi(x_n)$$
$$\boldsymbol{w}_{\boldsymbol{y_n}} = \boldsymbol{w}_{\boldsymbol{y_n}} + \phi(x_n)$$

?! Why does this do the right thing?

Perceptron with multiple classes v2

- Originally:



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

$$\hat{y} = \arg\max_k \boldsymbol{w_k} \cdot \phi(x_n)$$

  - If $\hat{y} \neq y_n$:

$$\boldsymbol{w}_{\hat{y}} = \boldsymbol{w}_{\hat{y}} - \phi(x_n)$$
$$\boldsymbol{w}_{\boldsymbol{y_n}} = \boldsymbol{w}_{\boldsymbol{y_n}} + \phi(x_n)$$

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

$$\hat{y} = \arg\max_k \boldsymbol{w} \cdot \phi(x_n, k)$$

  - If $\hat{y} \neq y_n$:

$$\boldsymbol{w} = \boldsymbol{w} - \phi(x_n, \hat{y}) + \phi(x_n, y_n)$$

Features for structured prediction

- Allowed to encode *anything* you want

Pro	Md	Vb	Dt	Nn
I	can	can	a	can

$\phi(\boldsymbol{x}, y) =$

```
I_Pro     : 1    <s>-Pro  :  1    has_verb    :  1
can_Md    : 1    Pro-Md   :  1    has_nn_lft  :  0
can_Vb    : 1    Md-Vb    :  1    has_n_lft   :  1
a_Dt      : 1    Vb-Dt    :  1    has_nn_rgt  :  1
can_Nn    : 1    Dt-Nn    :  1    has_n_rgt   :  1
...              Nn-</s>  :  1    ...
                 ...
```

- Output features, Markov features, other features

Structured perceptron

Enumeration over 1..K

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

    $\hat{y} = \arg\max_k \boldsymbol{w} \cdot \phi(x_n, k)$
  - If $\hat{y} \neq y_n$:

    $\boldsymbol{w} = \boldsymbol{w} - \phi(x_n, \hat{y}) + \phi(x_n, y_n)$

Enumeration over all outputs

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

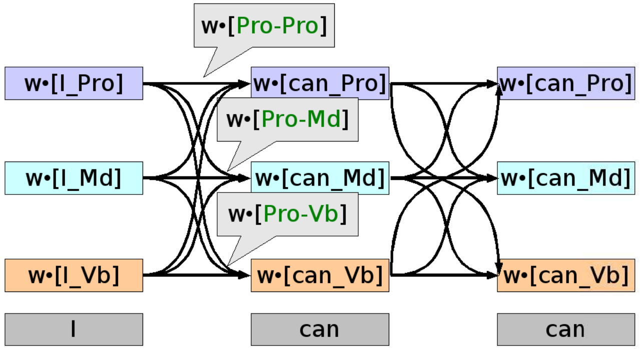
    $\hat{y} = \arg\max_k \boldsymbol{w} \cdot \phi(x_n, k)$
  - If $\hat{y} \neq y_n$:

    $\boldsymbol{w} = \boldsymbol{w} - \phi(x_n, \hat{y}) + \phi(x_n, y_n)$

[Collins, EMNLP02]

Argmax for sequences

- 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: $\hat{y} = \arg\max_k \boldsymbol{w} \cdot \phi(x_n, k)$
  - If $\hat{y} \neq y_n$: $\boldsymbol{w} = \boldsymbol{w} - \phi(x_n, \hat{y}) + \phi(x_n, y_n)$

- When does this make an update?

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

From perceptron to margins

Maximize Margin

Minimize Errors

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

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

Each point is correctly classified, modulo ξ

$$\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_n \xi_{n,\hat{y}}$$

Response for truth

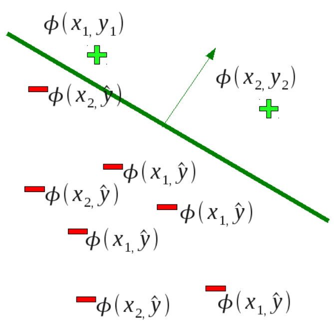
Response for other

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

Each true output is more highly ranked, modulo ξ

[Taskar+al, JMLR05; Tshochandaritis, JMLR05]

From perceptron to margins



$$\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_n \xi_{n,\hat{y}}$$

Response for truth

Response for other

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

Each true output is more highly ranked, modulo ξ

[Taskar+al, JMLR05; Tshochandaritis, JMLR05]

Ranking margins

- Some errors are worse than others...

Pro	Md	Vb	Dt	Nn

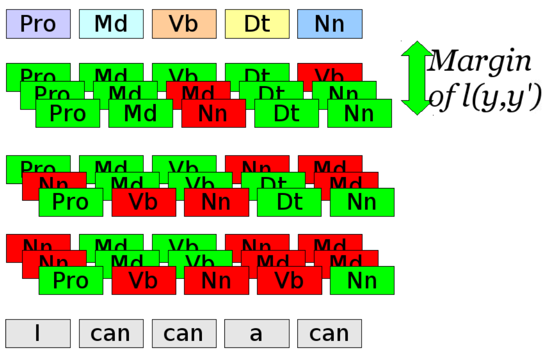
Margin of one

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

[Taskar+al, JMLR05; Tshochandaritis, JMLR05]

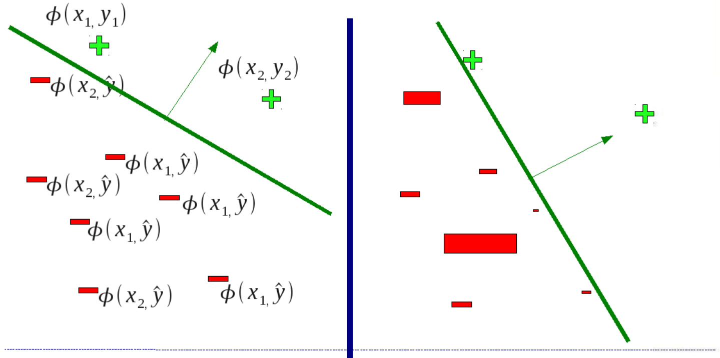
Accounting for a loss function

- Some errors are worse than others...



[Taskar+al, JMLR05; Tshochandaritis, JMLR05]

Accounting for a loss function



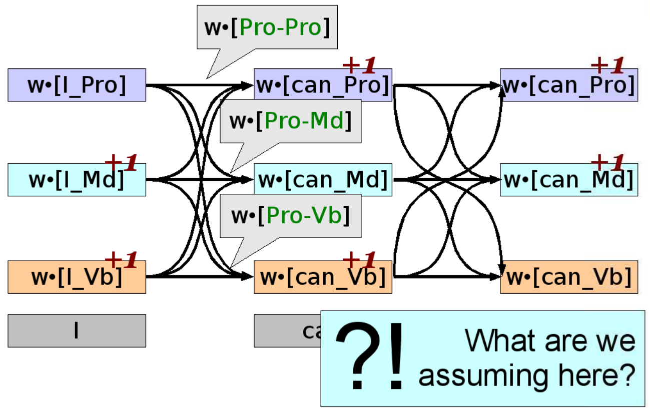
$$w \cdot \phi(x_n, y_n) - w \cdot \phi(x_n, \hat{y}) + \xi_n \geq 1$$

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

[Taskar+al, JMLR05; Tsochandaritis, JMLR05]

Augmented argmax for sequences

- Add "loss" to each wrong node!



[Taskar+al, JMLR05; Tsochandaritis, JMLR05]

Stochastically optimizing Markov nets

M3N Objective

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

[Ratliff+al, AIStats07]

Learning to Search

Argmax is *hard!*

- Classic formulation of structured prediction:

$$score(x,y) = \text{\textit{something we learn to make ``good'' x,y pairs score highly}}$$

- At test time:

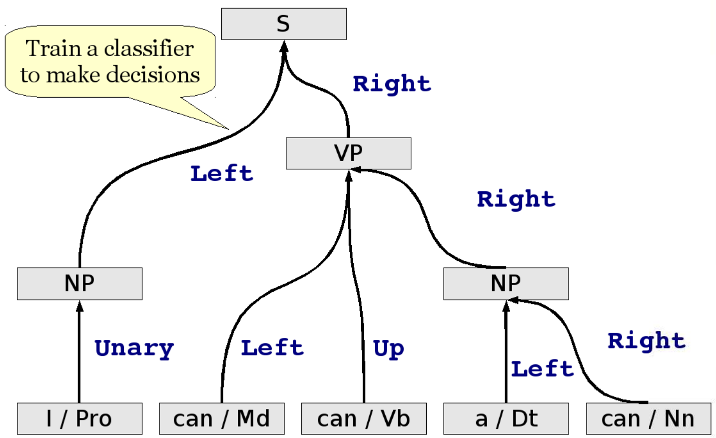
$$f(x) = argmax_{y \in Y}\, score(x,y)$$

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

Stacking

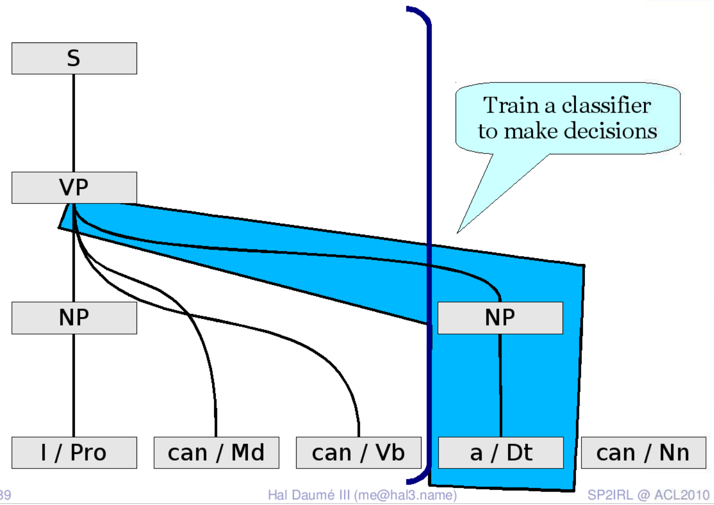
- Click to add an outline

Incremental parsing, early 90s style



[Magerman, ACL95]

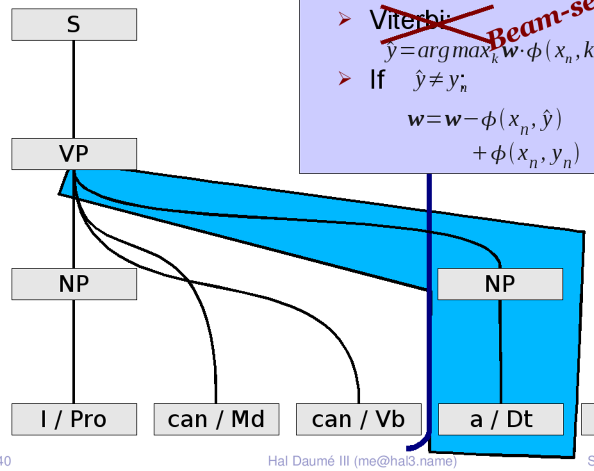
Incremental parsing, mid 2000s style



[Collsns+Roark, ACL04]

Learning to beam-search

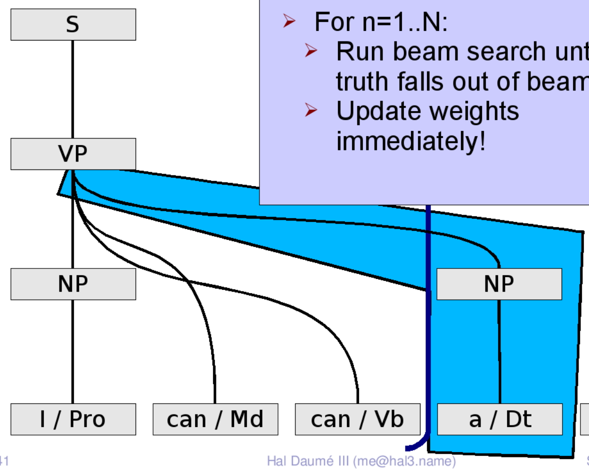
- For n=1..N:
  - ~~Viterbi:~~ Beam-search
    $\hat{y} = \arg\max_k \mathbf{w} \cdot \phi(x_n, k)$
  - If $\hat{y} \neq y_n$:
    $$\mathbf{w} = \mathbf{w} - \phi(x_n, \hat{y}) + \phi(x_n, y_n)$$



[Collins+Roark, ACL04]

Learning to beam-search

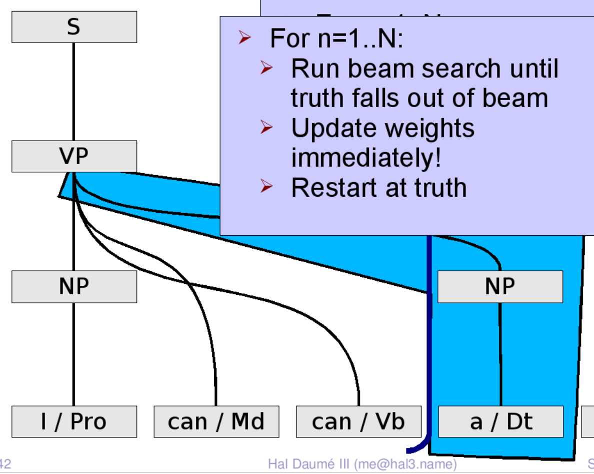
- For n=1..N:
  - Run beam search until truth falls out of beam
  - Update weights immediately!



[Collins+Roark, ACL04]

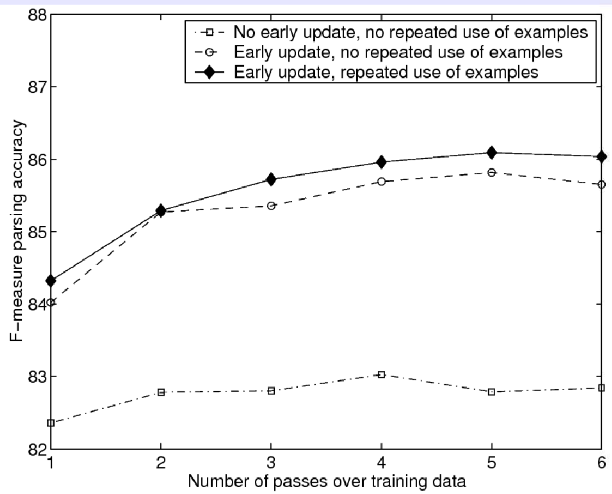
Learning to beam-search

- 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
  - node := RemoveFront(nodes)
  - **if** node is a goal state **return** node
  - next := Operators(node)
  - nodes := Enqueue(nodes, next)
- **fail**

[D+Marcu, ICML05; Xu+al, JMLR09]

Online Learning Framework (LaSO)

- nodes := MakeQueue(S0)
- **while** nodes is not empty
  - node := RemoveFront(nodes)
  - **if** none of {node} ∪ nodes is y-good **or** node is a goal **&** not y-good
    - sibs := siblings(node, y)
    - w := update(w, x, sibs, {node} ∪ nodes)
    - nodes := MakeQueue(sibs)
  - **else**
    - if node is a goal
    - 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?

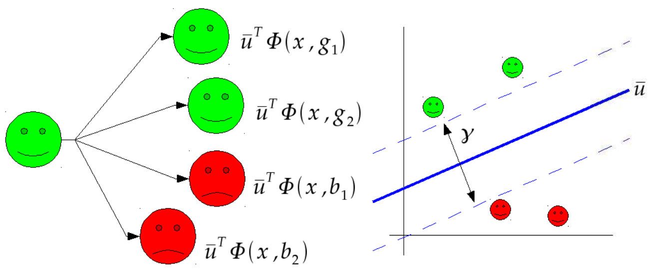
Update our weights based on the good and the bad choices

Continue search...

[D+Marcu, ICML05; Xu+al, JMLR09]

Search-based Margin

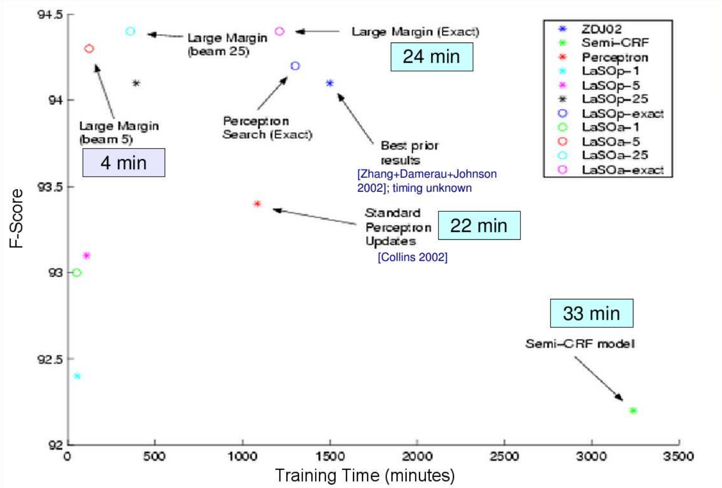
- The *margin* is the amount by which we are correct:



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

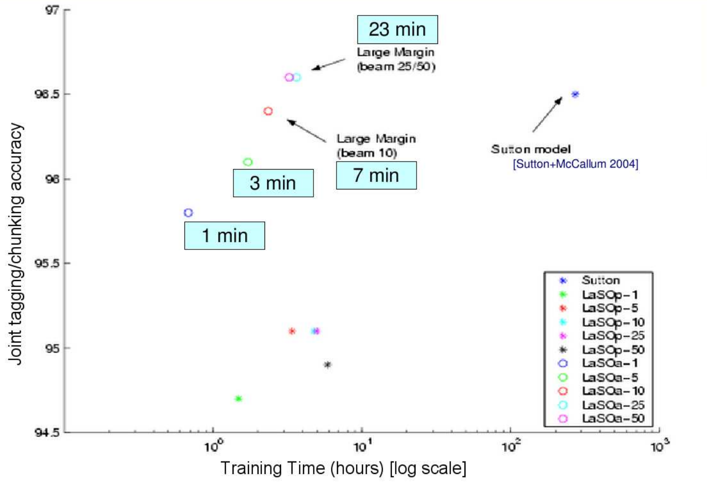
[D+Marcu, ICML05; Xu+al, JMLR09]

Syntactic chunking Results



[D+Marcu, ICML05; Xu+al, JMLR09]

Tagging+chunking results



[D+Marcu, ICML05; Xu+al, JMLR09]

Variations on a beam

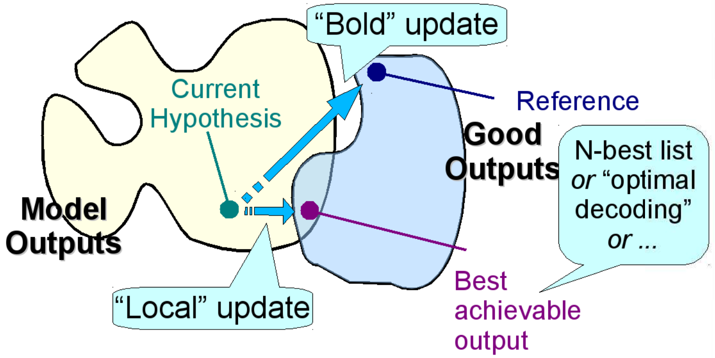
- Observation:
  - We needn't use the same beam size for training and decoding
  - Varying these values independently yields:

Training Beam \ Decoding Beam	1	5	10	25	50
1	93.9	92.8	91.9	91.3	90.9
5	90.5	94.3	94.4	94.1	94.1
10	89.5	94.3	94.4	94.2	94.2
25	88.7	94.2	94.5	94.3	94.3
50	88.4	94.2	94.4	94.2	94.4

[D+Marcu, ICML05; Xu+al, JMLR09]

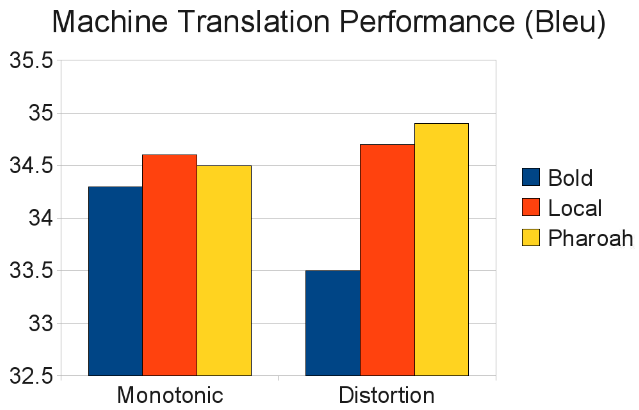
What if our model sucks?

- Sometimes our model *cannot* produce the "correct" output
  - canonical example: machine translation



[Och, ACL03; Liang+al, ACL06]

Local versus bold updating...



Take-home messages

If not, this can be a *really* bad idea! [Kulesza+Pereira, NIPS07]

- 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!!!

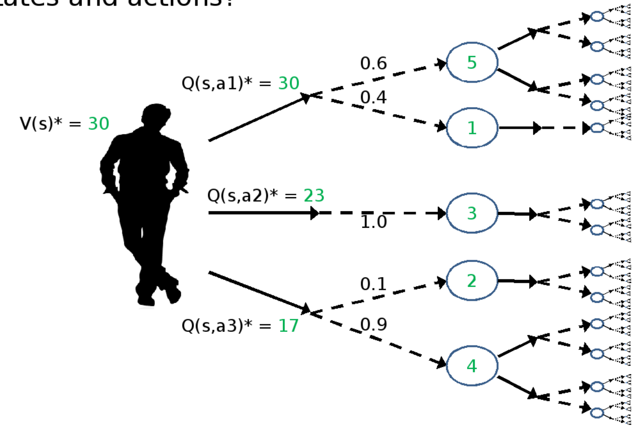
Refresher on Reinforcement Learning

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

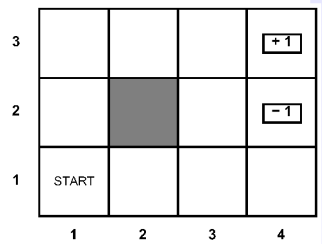
Markov decision processes

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



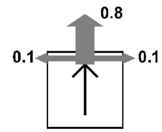
Markov Decision Processes

- 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:

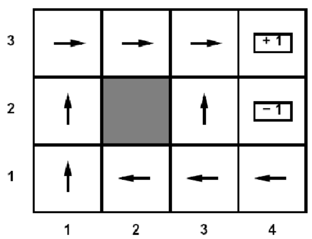
$$\sum_t r_t \text{ or } \sum_t \gamma^t r_t$$



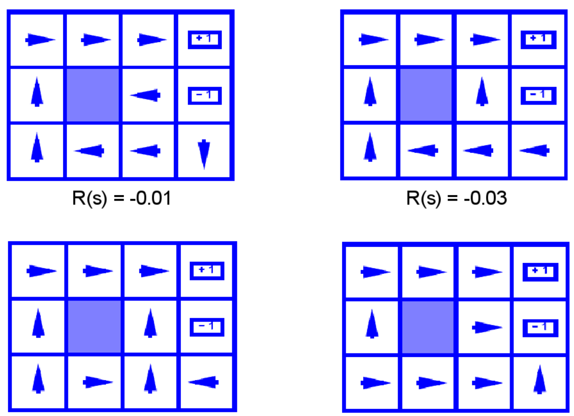
Solving MDPs

- 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 π(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



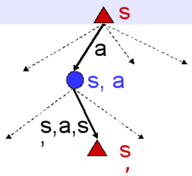
R(s) = -0.01

R(s) = -0.03

R(s) = -0.4

R(s) = -2.0

Optimal Utilities



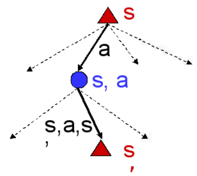
- Fundamental operation: compute the optimal utilities of states s (all at once)
- Why? Optimal values define optimal policies!
- Define the utility of a state s:
  $V^*(s)$ = expected return starting in s and acting optimally
- Define the utility of a q-state (s,a):
  $Q^*(s,a)$ = expected return starting in s, taking action a and thereafter acting optimally
- Define the optimal policy:
  $\pi^*(s)$ = optimal action from state s

3	0.812	0.868	0.912	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

3	→	→	→	+1
2	↑		↑	−1
1	↑	←	←	←
	1	2	3	4

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*
  - $(s_1, a_1, r_1, \quad s_2, a_2, r_2, \quad s_3, a_3, r_3, \quad \ldots)$

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 (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, 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 ε 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:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \, [error]$$

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

Inverse Reinforcement Learning

(aka Inverse Optimal Control)

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 Trajecto

Warning: need to be careful to avoid trivial solutions!

- Optimal policy available through s (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

  and $V_w^{\pi}(s_0)$ be the value of the starting state

$$\max_{\mathbf{w}} \sum_{k=1}^{K} f\left(V_{\mathbf{w}}^{\pi^*}(s_0) - V_{\mathbf{w}}^{\pi_k}(s_0)\right)$$

How good does the "optimal policy" look?

How good does the some other policy look?

[Ng+Russell, ICML00]

Apprenticeship Learning via IRL

- For t = 1,2,…
  - 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 π_t for the estimated reward w

[Abbeel+Ng, ICML04]

Car Driving Experiment

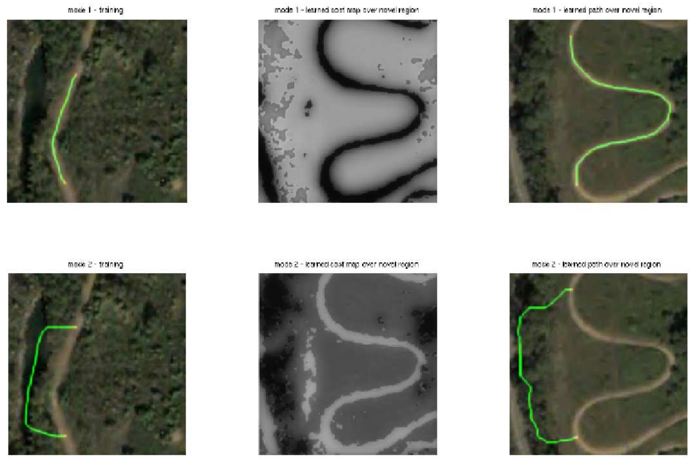
- No explicit reward function at all!
- Expert demonstrates proper policy via 2 min. of driving time on simulator (1200 data points).
- 5 different "driver types" tried.
- Features: which lane the car is in, distance to closest car in current lane.
- Algorithm run for 30 iterations, policy hand-picked.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

[Abbeel+Ng, ICML04]

"Nice" driver

"Evil" driver

Planning as structured prediction



[Ratliff+al, NIPS05]

Maximum margin planning

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

$$\max_{\mathbf{w}} \quad \text{margin} \quad s.t. \quad \text{planner run with w yields human output}$$

Q-state visitation frequency by human

$$\min_{\mathbf{w}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 \quad s.t. \quad \mu(s,a)\boldsymbol{w}\cdot\phi(x_n,s,a) - \hat{\mu}(s,a)\boldsymbol{w}\cdot\phi(x_n,s,a) \geq 1 \quad , \forall n,s,a$$

Q-state visitation frequency by planner

All trajectories, and all q-states

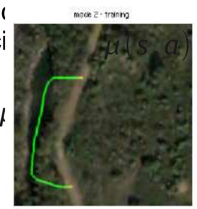
[Ratliff+al, NIPS05]

Optimizing MMP

M3N Objective

SOME MATH

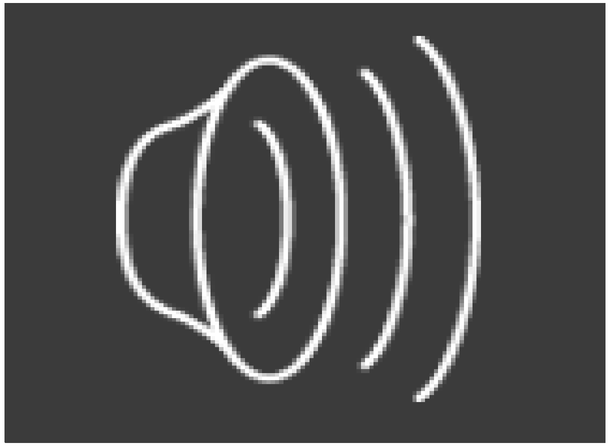
- For n=1..N:
  - Augmented planning:
    Run A* on current (augmented) c
    to get q-state visitation frequenci
  - Update: $w=w+\sum_s\sum_a[\hat{\mu}(s,a)-\mu$
  - Shrink: $w=\left(1-\frac{1}{CN}\right)w$



$\mu(s,a)$

[Ratliff+al, NIPS05]

Maximum margin planning movies



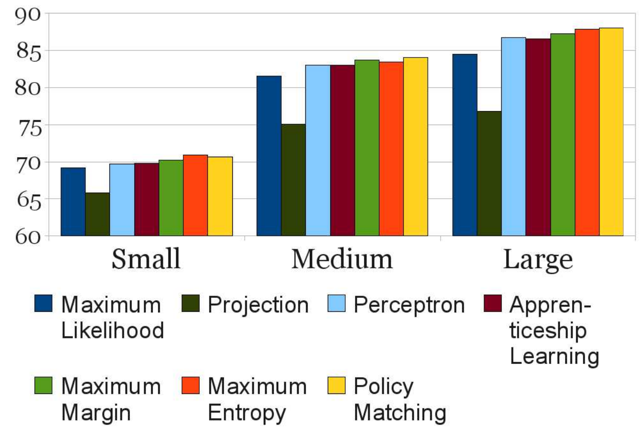
[Ratliff+al, NIPS05]

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

[Neu+Szepevari, MLJ09]

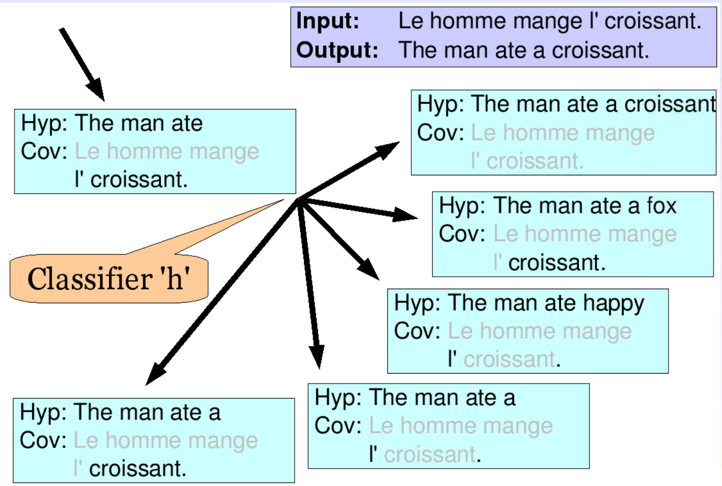
Parsing via inverse optimal control



[Neu+Szepevari, MLJ09]

Apprenticeship Learning

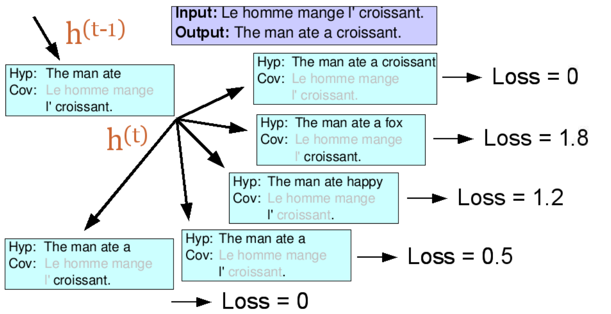
Integrating search and learning



[D+Marcu, ICML05; D+Langford+Marcu, MLJ09]

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



[D+Langford+Marcu, MLJ09]

Theoretical results

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

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

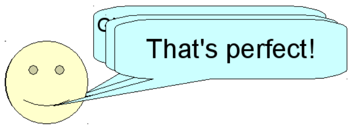
Loss of the optimal policy

Average multiclass classification loss

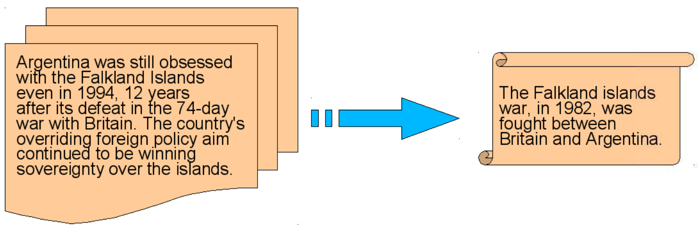
Worst case per-step loss

[D+Langford+Marcu, MLJ09]

Example task: summarization



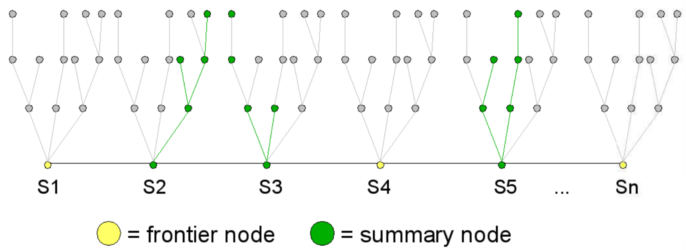
Standard approach is sentence extraction, but that is often deemed to "coarse" to produce good, very short summaries. We wish to also drop words and phrases => document compression



[D+Langford+Marcu, MLJ09]

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



[D+Langford+Marcu, MLJ09]

Example output (40 word limit)

Sentence Extraction + Compression:

+13 Argentina and Britain announced an agreement, nearly eight years after they fought a 74-day war 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.

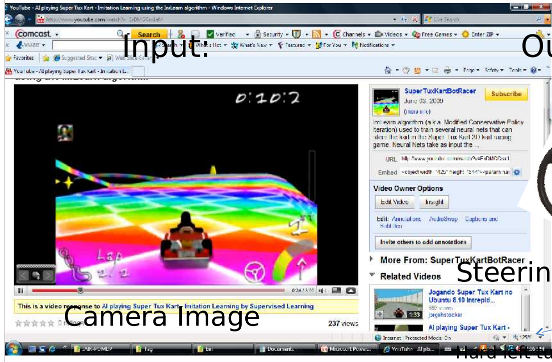
Vine Growth (Searn):

+24 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
- 2 War was 74-days long

[D+Langford+Marcu, MLJ09]

Learning to Drive



Input:

Camera Image

Output:

Steering in [-1,1]

Hard left turn

Hard right turn

DAgger: Dataset Aggregation

Collect trajectories with expert π*

Theoretical Guarantees

Best policy π in sequence π[1:N] guarantees:

$$\mathbf{J}(\pi) \leq \mathbf{T}(\varepsilon_{\mathbf{N}} + \gamma_{\mathbf{N}}) + \mathbf{O}(\mathbf{T}/\mathbf{N})$$

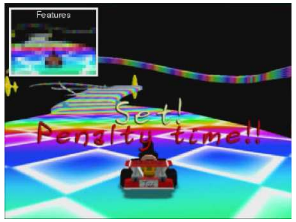
Avg. Loss on Aggregate Dataset

Avg. Regret of π[1:N]

Iterations of DAgger

Experiments: Racing Game

Input:



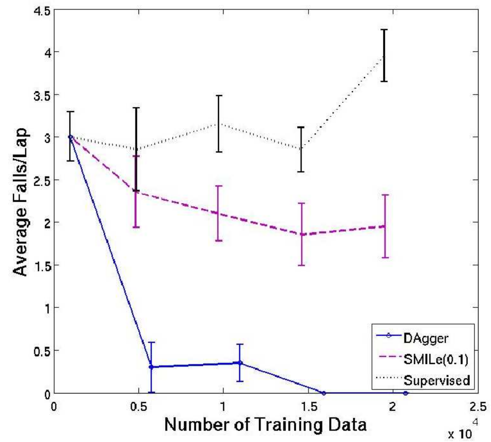
Output:

Steering in [-1,1]

Resized to 25x19 pixels (1425 features)

Average Falls per Lap

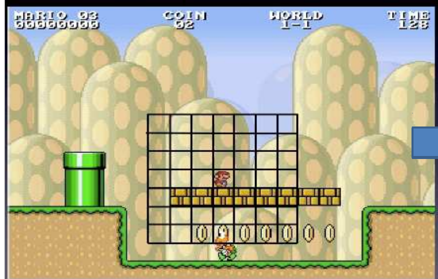
Better



Super Mario Brothers

From Mario AI competition 2009

Interface



tput:

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

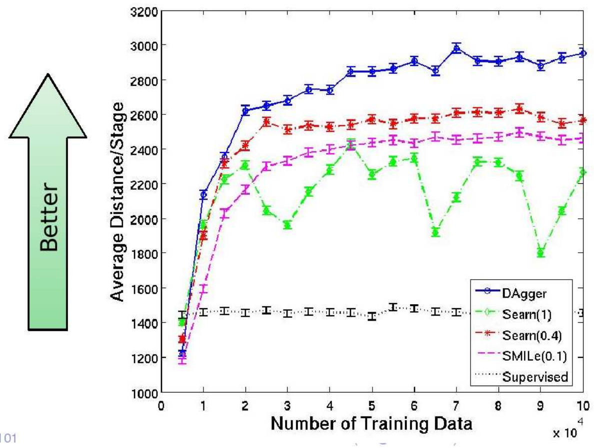
Extracted 27K+ binary features from last 4 observations

(14 binary features for every cell)

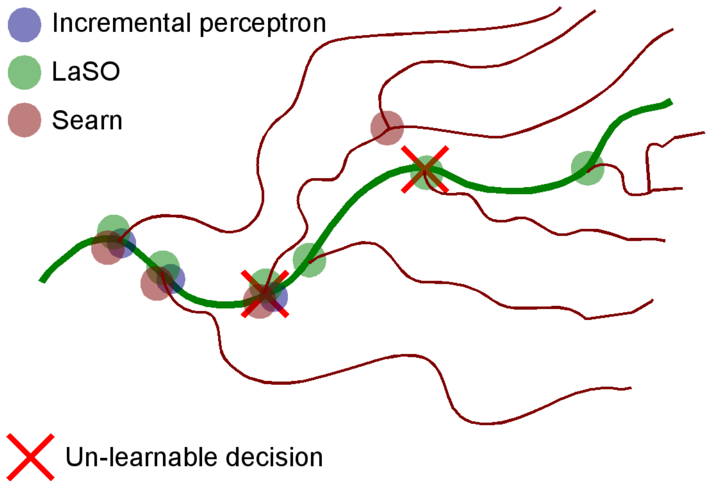
Unknown Zone | Protected Mode: On

Test-time Execution

Average Distance per Stage



Perceptron vs. LaSO vs. Searn



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 stochastiticity 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: ???

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://www.cs.utah.edu/~suresh/mediawiki/index.php/MLRG
http://braque.cc/ShowChannel?handle=P5BVAC34

Stuff we talked about explicitly

- *Apprenticeship learning via inverse reinforcement learning, P. Abbeel and A. Ng. ICML, 2004.*
- *Incremental parsing with the Perceptron algorithm. M. Collins and B. Roark. ACL 2004.*
- *Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms.* M. Collins. EMNLP 2002.
- *Search-based Structured Prediction.* H. Daumé III, J. Langford and D. Marcu. Machine Learning, 2009.
- *Learning as Search Optimization: Approximate Large Margin Methods for Structured Prediction.* H. Daumé III and D. Marcu. ICML, *2005.*
- *An End-to-end Discriminative Approach to Machine Translation.* P. Liang, A. Bouchard-Côté, D. Klein, B. Taskar. ACL 2006.
- *Statistical Decision-Tree Models for Parsing. D. Magerman. ACL 1995.*
- *Training Parsers by Inverse Reinforcement Learning. G. Neu and Cs. Szepesvári. Machine Learning 77, 2009.*
- *Algorithms for inverse reinforcement learning, A. Ng and A. Russell. ICML, 2000.*
- *(Online) Subgradient Methods for Structured Prediction.* N. *Ratliff, J. Bagnell, and M. Zinkevich. AIStats 2007.*
- *Maximum margin planning.* N. Ratliff, J. Bagnell and M. Zinkevich. ICML, 2006.
- *Learning to search: Functional gradient techniques for imitation learning.* N. *Ratliff, D. Silver, and J. Bagnell. Autonomous Robots, Vol. 27, No. 1, July, 2009.*
- *Reduction of Imitation Learning to No-Regret Online Learning.* S. Ross, G. Gordon and J. Bagnell. AIStats 2011.
- *Max-Margin Markov Networks.* B. Taskar, C. Guestrin, V. Chatalbashev and D. Koller. JMLR 2005.
- *Large Margin Methods for Structured and Interdependent Output Variables.* I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. JMLR 2005.
- *Learning Linear Ranking Functions for Beam Search with Application to Planning. Y.* Xu, A. Fern, and S. Yoon. JMLR 2009.
- *Maximum Entropy Inverse Reinforcement Learning.* B. Ziebart, A. Maas, J. Bagnell, and A. Dey. AAAI 2008.

Other good stuff

- *Reinforcement learning for mapping instructions to actions.* S.R.K. Branavan, H. Chen, L. Zettlemoyer and R. Barzilay. ACL, 2009.
- *Driving semantic parsing from the world's response.* J. Clarke, D. Goldwasser, M.-W. Chang, D. Roth. CoNLL 2010.
- *New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. M.*Collins and N. Duffy. *ACL 2002.*
- *Unsupervised Search-based Structured Prediction.* H. Daumé III. ICML 2009.
- *Training structural SVMs when exact inference is intractable.* T. Finley and T. Joachims. ICML, 2008.
- *Structured learning with approximate inference.* A. Kulesza and F. Pereira. NIPS, 2007.
- *Conditional random fields: Probabilistic models for segmenting and labeling sequence data.* J. Lafferty, A. McCallum, F. Pereira. ICML 2001.
- *Structure compilation: trading structure for features.* P. Liang, H. Daume, D. Klein. ICML 2008.
- *Learning semantic correspondences with less supervision.* P. Liang, M. Jordan and D. Klein. ACL, 2009.
- *Generalization Bounds and Consistency for Structured Labeling.* D. McAllester. In *Predicting Structured Data, 2007.*
- *Maximum entropy Markov models for information extraction and segmentation.* A. McCallum, D. Freitag, F. Pereira. ICML 2000.
- *FACTORIE: Efficient Probabilistic Programming for Relational Factor Graphs via Imperative Declarations of Structure, Inference and Learning.* A. McCallum, K. Rohanemanesh, M. Wick, K. Schultz, S. Singh. NIPS Workshop on Probabilistic Programming, 2008
- *Learning efficiently with approximate inference via dual losses.* O. Meshi, D. Sontag, T. Jaakkola, A. Globerson. ICML 2010.
- *Learning and inference over constrained output.* V. Punyakanok, D. Roth, W. Yih, D. Zimak. IJCAI, 2005.
- *Boosting Structured Prediction for Imitation Learning.* N. Ratliff, D. Bradley, J. Bagnell, and J. Chestnutt. NIPS 2007.
- *Efficient Reductions for Imitation Learning.* S. Ross and J. Bagnell. AISTATS, 2010.
- *Kernel Dependency Estimation.* J. Weston, O. Chapelle, A. Elisseeff, B. Schoelkopf and V. Vapnik. NIPS 2002.