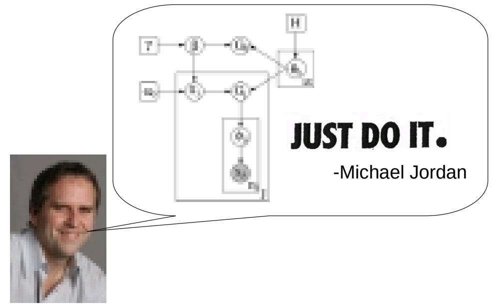
Compiling Comp Ling

Practical weighted dynamic programming and the Dyna language

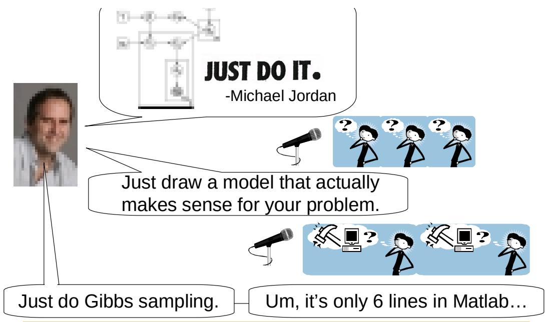
Jason Eisner
Eric Goldlust
Noah A. Smith
Johns Hopkins University

HLT-EMNLP, October 2005

An Anecdote from ACL'05



An Anecdote from ACL'05



Conclusions to draw from that talk

1. Mike & his students are great.
2. Graphical models are great.
   (because they're flexible)
3. Gibbs sampling is great.
   (because it works with nearly any graphical model)
4. Matlab is great.
   (because it frees up Mike and his students to doodle all day and then execute their doodles)

Could NLP be this nice?

1. Mike & his students are great.
2. Graphical models are great.
   (because they're flexible)
3. Gibbs sampling is great.
   (because it works with nearly any graphical model)
4. Matlab is great.
   (because it frees up Mike and his students to doodle all day and then execute their doodles)

Could NLP be this nice?

Parts of it already are …

Language modeling
Binary classification (e.g., SVMs)
Finite-state transductions
Linear-chain graphical models

} Toolkits available; you don't have to be an expert

But other parts aren't …

Context-free and beyond
Machine translation

} Efficient parsers and MT systems are complicated and painful to write

Could NLP be this nice?

This talk: A toolkit that's general enough for <u>these</u> cases.

(stretches from finite-state to Turing machines)

"Dyna"

But other parts aren't …

Context-free and beyond
Machine translation

Efficient parsers and MT systems are complicated and painful to write

7

Warning

- This talk is only an advertisement!
- For more details, please

see the paper

see `http://dyna.org`

(download + documentation)

sign up for updates by email

8

How you build a system ("big picture" slide)

cool model

practical equations

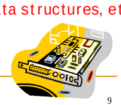
$$\beta_x(i,k) = \sum_{0\le i<j<k\le n} \beta_y(i,j)\beta_z(j,k)\; p(N_x \to N_y N_z \mid N_x)$$

...

pseudocode
(execution order)

```
for width from 2 to n
  for i from 0 to n-width
    k = i+width
    for j from i+1 to k-1
      …
```

tuned C++
implementation
(data structures, etc.)



9

Wait a minute …

Didn't I just implement something like this last month?

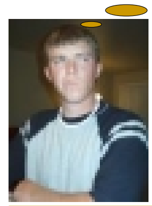


chart management / indexing
cache-conscious data structures
prioritize partial solutions (best-first, pruning)
parameter management
inside-outside formulas
different algorithms for training and decoding
conjugate gradient, annealing, ...
parallelization?

We thought computers were supposed to *automate* drudgery

10

How you build a system ("big picture" slide)

cool model

practical equations

$$\beta_x(i,k) = \sum_{0\le i<j<k\le n} \beta_y(i,j)\beta_z(j,k)\; p(N_x \to N_y N_z \mid N_x)$$

...

Dyna language <u>specifies these equations</u>.

<u>Most</u> programs just need to compute some values from other values. Any order is ok.

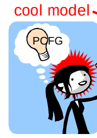
Some programs also need to update the outputs if the inputs change:
- spreadsheets, makefiles, email readers
- dynamic graph algorithms
- EM and other iterative optimization
- leave-one-out training of smoothing params

tion
s, etc.)

11

How you build a system ("big picture" slide)

cool model



practical equations

$$\beta_x(i,k) = \sum_{0\le i<j<k\le n} \beta_y(i,j)\beta_z(j,k)\; p(N_x \to N_y N_z \mid N_x)$$

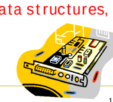
...

Compilation strategies
(we'll come back to this)

pseudocode
(execution order)

```
for width from 2 to n
  for i from 0 to n-width
    k = i+width
    for j from i+1 to k-1
      …
```

tuned C++
implementation
(data structures, etc.)



12

Writing equations in Dyna

- `int a.`
- `a = b * c.`

  a will be kept up to date if **b** or **c** changes.
- `b += x.`
  `b += y.` *equivalent to* `b = x+y.`

  b is a sum of two variables. Also kept up to date.
- ~~`c += z(1).`~~
  ~~`c += z(2).`~~
  ~~`c += z(3).`~~
  ~~`c += z("four").`~~
  ~~`c += z(foo(bar,5)).`~~

  → `c += z(N).`

  a "pattern" — the capitalized N matches anything

  c is a sum of **all** nonzero z(…) values. At compile time, we don't know how many!

13

More interesting use of patterns

- `a = b * c.`
  - scalar multiplication
- `a(I) = b(I) * c(I).`
  - pointwise multiplication
- `a += b(I) * c(I).` *means a =* $\sum_I$ `b(I)*c(I)`
  - dot product; could be sparse

  sparse dot product of query & document

  ```
  ... + b("yetis")*c("yetis")
      + b("zebra")*c("zebra")
  ```
- `a(I,K) += b(I,J) * c(J,K).` $\sum_J$ `b(I,J)*c(J,K)`
  - matrix multiplication; could be sparse
  - **J** is free on the right-hand side, so we sum over it

14

Dyna vs. Prolog

By now you may see what we're up to!

Prolog has Horn clauses:

`a(I,K) :- b(I,J) , c(J,K).`

Dyna has "Horn equations":

`a(I,K) += b(I,J) * c(J,K)`

has a **value** e.g., a real number

definition from other values

Like Prolog:	**Unlike Prolog:**
Allow nested terms	Charts, not backtracking!
Syntactic sugar for lists, etc.	Compile → efficient C++ classes
Turing-complete	Integrates with your C++ code

The CKY inside algorithm in Dyna

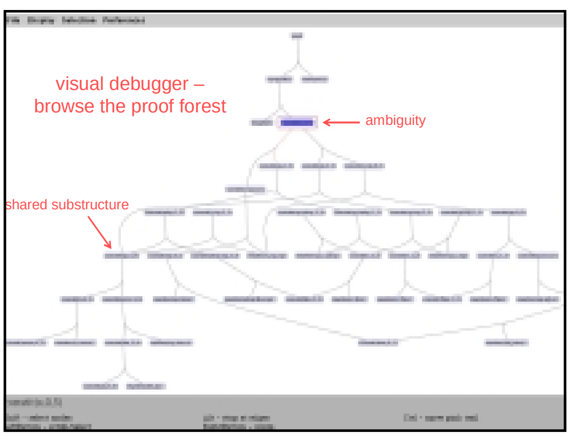
```
:- double item = 0.
:- bool length = false.
constit(X,I,J)  += word(W,I,J)       * rewrite(X,W).
constit(X,I,J)  += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal            += constit("s",0,N) if length(N).
```

```
using namespace cky;
chart c;

c[rewrite("s","np","vp")] = 0.7;
c[word("Pierre",0,1)] = 1;
c[length(30)] = true;     // 30-word sentence
cin >> c;                 // get more axioms from stdin

cout << c[goal];   // print total weight of all parses
```

put in axioms (values **not** defined by the above program)

theorem pops out

16



Related algorithms in Dyna?

```
constit(X,I,J)  += word(W,I,J)       * rewrite(X,W).
constit(X,I,J)  += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal            += constit("s",0,N) if length(N).
```

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

18

Related algorithms in Dyna?

```
constit(X,I,J)  max=  word(W,I,J)       * rewrite(X,W).
constit(X,I,J)  max=  constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal            max=  constit("s",0,N)  if  length(N).
```

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

Related algorithms in Dyna?

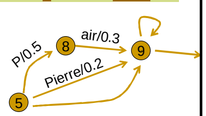
```
constit(X,I,J)  log+=  word(W,I,J)       + rewrite(X,W).
constit(X,I,J)  log+=  constit(Y,I,Mid) + constit(Z,Mid,J) + rewrite(X,Y,Z).
goal            log+=  constit("s",0,N)  if  length(N).
```

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

Related algorithms in Dyna?

```
constit(X,I,J)  +=  word(W,I,J)       * rewrite(X,W).
constit(X,I,J)  +=  constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal            +=  constit("s",0,N)  if  length(N).
```

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing? c[word("Pierre", state(5), state(9))] = 0.2
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?



Related algorithms in Dyna?

```
constit(X,I,J)  +=  word(W,I,J)       * rewrite(X,W).
constit(X,I,J)  +=  constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal            +=  constit("s",0,N)  if  length(N).
```

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

Earley's algorithm in Dyna

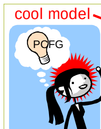
```
constit(X,I,J)  +=  word(W,I,J)       * rewrite(X,W).
constit(X,I,J)  +=  constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal            +=  constit("s",0,N)  if  length(N).
```

magic templates transformation
(as noted by Minnen 1996)

```
need("s",0) = true.
need(Nonterm,J) |= ?constit(_/[Nonterm|_],_,J).
constit(Nonterm/Needed,I,I)
        += rewrite(Nonterm,Needed) if need(Nonterm,I).
constit(Nonterm/Needed,I,K)
        += constit(Nonterm/[W|Needed],I,J) * word(W,J,K).
constit(Nonterm/Needed,I,K)
        += constit(Nonterm/[X|Needed],I,J) * constit(X/[],J,K).
goal += constit("s"/[],0,N) if length(N).
```


Program transformations

cool model



practical equations

$$\beta_x(i,k) = \sum_{0 \le i < j < k \le n} \frac{\beta_y(i,j)\beta_z(j,k)}{p(N_x \to N_y N_z \mid N_x)}$$

…

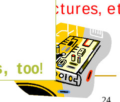
Lots of equivalent ways to write a system of equations!

Transforming from one to another may improve efficiency.

(Or, transform to <u>related</u> equations that compute gradients, upper bounds, etc.)

Many parsing "tricks" can be generalized into <u>automatic transformations</u> that help other programs, too!

d C++
entation
tures, etc.)



Related algorithms in Dyna?

constit(X,I,J) += word(W,I,J) * rewrite(X,W).
constit(X,I,J) += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal += constit("s",0,N) if length(N).

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

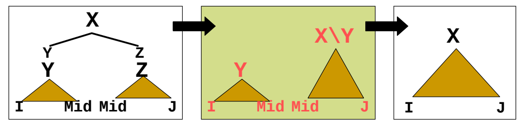
25

Rule binarization

constit(X,I,J) += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).

folding transformation: asymp. speedup!

constit(X\Y,Mid,J) += constit(Z,Mid,J) * rewrite(X,Y,Z).
constit(X,I,J) += constit(Y,I,Mid) * constit(X\Y,Mid,J).



26

Rule binarization

constit(X,I,J) += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).

folding transformation: asymp. speedup!

constit(X\Y,Mid,J) += constit(Z,Mid,J) * rewrite(X,Y,Z).
constit(X,I,J) += constit(Y,I,Mid) * constit(X\Y,Mid,J).

$$\sum_{Y,Z,Mid} \text{constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z)}$$

graphical models
constraint programming
multi-way database join

$$\sum_{Y,Mid} \text{constit(Y,I,Mid)} \sum_{Z} \text{constit(Z,Mid,J) * rewrite(X,Y,Z)}$$

Related algorithms in Dyna?

constit(X,I,J) += word(W,I,J) * rewrite(X,W).
constit(X,I,J) += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal += constit("s",0,N) if length(N).

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

Just add words one at a time to the chart
Check at any time what can be derived from words so far

Similarly, dynamic grammars

28

Related algorithms in Dyna?

constit(X,I,J) += word(W,I,J) * rewrite(X,W).
constit(X,I,J) += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal += constit("s",0,N) if length(N).

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

Again, no change to the Dyna program

29

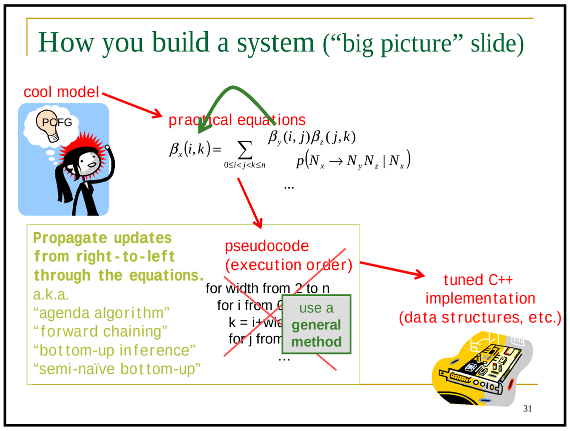
Related algorithms in Dyna?

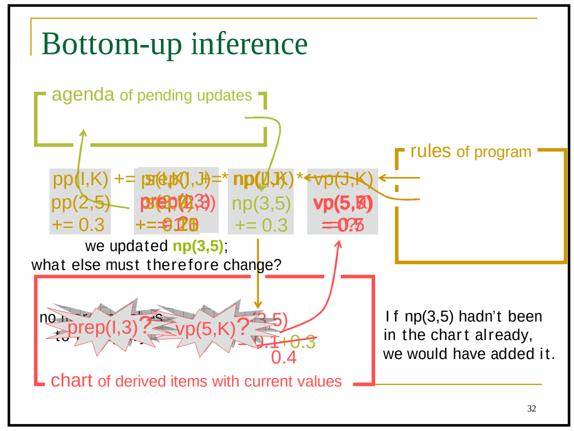
constit(X,I,J) += word(W,I,J) * rewrite(X,W).
constit(X,I,J) += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal += constit("s",0,N) if length(N).

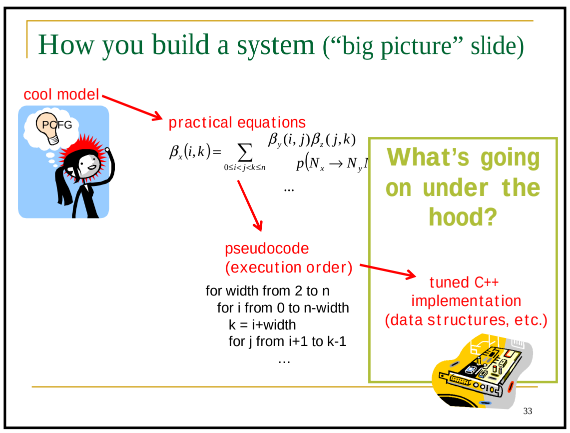
- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley's algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

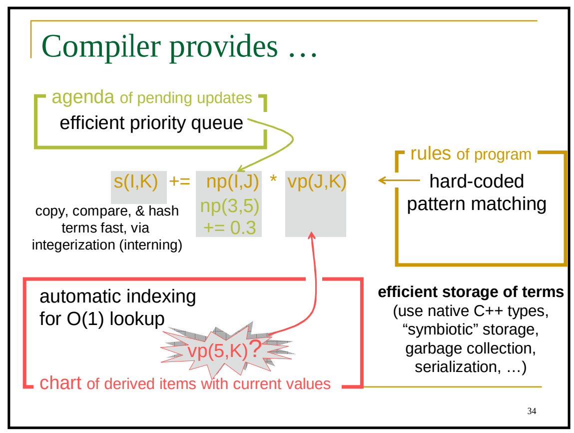
Basically, just add extra arguments to the terms above

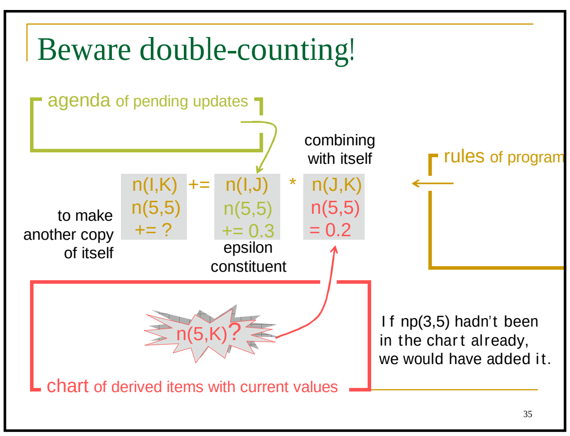
30

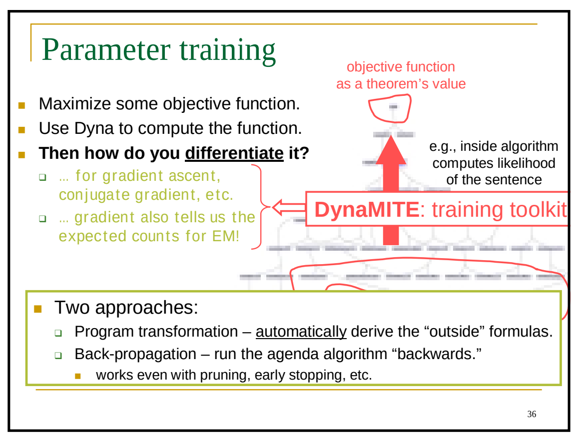

How you build a system ("big picture" slide)
cool model
PCFG
practical equations
$\beta_x(i,k) = \sum_{0 \le i < j < k \le n} \beta_y(i,j)\beta_z(j,k) \; p(N_x \to N_y N_z \mid N_x)$
...
pseudocode
(execution order)
for width from 2 to n
for i from 0
k = i+wid
for j from
...
use a general method
tuned C++ implementation (data structures, etc.)
Propagate updates from right-to-left through the equations.
a.k.a.
"agenda algorithm"
"forward chaining"
"bottom-up inference"
"semi-naïve bottom-up"
31


Bottom-up inference
agenda of pending updates
rules of program
pp(I,K) += prep(I,J) * np(J,K)
np(3,5) += 0.3
we updated np(3,5);
what else must therefore change?
If np(3,5) hadn't been in the chart already, we would have added it.
chart of derived items with current values
32


How you build a system ("big picture" slide)
cool model
PCFG
practical equations
$\beta_x(i,k) = \sum_{0 \le i < j < k \le n} \beta_y(i,j)\beta_z(j,k) \; p(N_x \to N_y N_z \mid N_x)$
...
What's going on under the hood?
pseudocode
(execution order)
for width from 2 to n
for i from 0 to n-width
k = i+width
for j from i+1 to k-1
...
tuned C++ implementation (data structures, etc.)
33


Compiler provides …
agenda of pending updates
efficient priority queue
s(I,K) += np(I,J) * vp(J,K)
np(3,5) += 0.3
copy, compare, & hash terms fast, via integerization (interning)
rules of program
hard-coded pattern matching
automatic indexing for O(1) lookup
vp(5,K)?
chart of derived items with current values
efficient storage of terms (use native C++ types, "symbiotic" storage, garbage collection, serialization, …)
34


Beware double-counting!
agenda of pending updates
rules of program
combining with itself
n(I,K) += n(I,J) * n(J,K)
n(5,5) += ?
n(5,5) += 0.3
n(5,5) = 0.2
to make another copy of itself
epsilon constituent
n(5,K)?
If np(3,5) hadn't been in the chart already, we would have added it.
chart of derived items with current values
35


Parameter training
Maximize some objective function.
Use Dyna to compute the function.
Then how do you differentiate it?
… for gradient ascent, conjugate gradient, etc.
… gradient also tells us the expected counts for EM!
objective function as a theorem's value
e.g., inside algorithm computes likelihood of the sentence
DynaMITE: training toolkit
Two approaches:
Program transformation – automatically derive the "outside" formulas.
Back-propagation – run the agenda algorithm "backwards."
works even with pruning, early stopping, etc.
36

What can Dyna do beyond CKY?

- Context-based morphological disambiguation with random fields (Smith, Smith & Tromble EMNLP'05)
- Parsing with con[...] r & Smith IWPT'05)
- Unsupervised gr[...] ve estimation [...] & Eisner GIA'05)
- Unsupervised lo[...] estimation [...] & Eisner ACL'05)
- Grammar induct[...] & Eisner ACL'04)
- Synchronous cro[...] & Smith EMNLP'04)
- Loosely syntax-based MT … (Smith & Eisner in prep.)
- Partly supervised grammar induction … (Dreyer & Eisner in prep.)
- More finite-state stuff … (Tromble & Eisner in prep.)
- Teaching (Eisner JHU'05; Smith & Tromble JHU'04)
- Most of my own past work on trainable (in)finite-state machines, parsing, MT, phonology …

Easy to try stuff out!

Programs are very short & easy to change!

Can it express everything in NLP? ☺

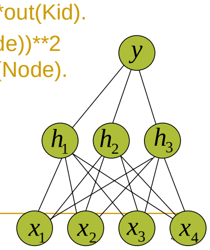
- Remember, it integrates tightly with C++, so you only have to use it where it's helpful, and write the rest in C++. Small is beautiful.
- We're currently extending the class of allowed formulas "beyond the semiring"
  - cf. Goodman (1999)
  - will be able to express smoothing, neural nets, etc.
- Of course, it **is** Turing complete … ☺

Smoothing in Dyna

- mle_prob(X,Y,Z) % context
  = count(X,Y,Z)/count(X,Y).
- smoothed_prob(X,Y,Z)
  = lambda*mle_prob(X,Y,Z)
  + (1-lambda)*mle_prob(Y,Z).
  - % for arbitrary n-grams, can use lists
- count_count(N) += 1 whenever N is count(Anything).
  - % updates automatically during leave-one-out jackknifing

Neural networks in Dyna

- out(Node) = sigmoid(in(Node)).
- in(Node) += input(Node).
- in(Node) += weight(Node,Kid)*out(Kid).
- error += (out(Node)-target(Node))**2
  if ?target(Node).
- Recurrent neural net is ok



Game-tree analysis in Dyna

- goal = best(Board) if start(Board).
- best(Board) max= stop(player1, Board).
- best(Board) max= move(player1, Board, NewBoard) + worst(NewBoard).
- worst(Board) min= stop(player2, Board).
- worst(Board) min= move(player2, Board, NewBoard) + best(NewBoard).

Weighted FST composition in Dyna (epsilon-free case)

- :- bool item=false.
- start (A o B, Q x R) |= start (A, Q) & start (B, R).
- stop (A o B, Q x R) |= stop (A, Q) & stop (B, R).
- arc (A o B, Q1 x R1, Q2 x R2, In, Out)
  |= arc (A, Q1, Q2, In, Match)
  & arc (B, R1, R2, Match, Out).
- Inefficient? How do we fix this?

Constraint programming (arc consistency)

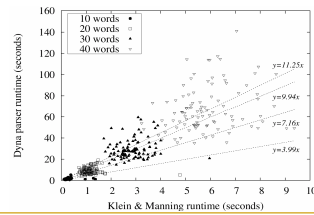
- :- bool item=false.
- :- bool consistent=true. % overrides prev line

- variable(Var) |= in_domain(Var:Val).
- possible(Var:Val) &= in_domain(Var:Val).
- possible(Var:Val) &= support(Var:Val, Var2)
  whenever variable(Var2).
- support(Var:Val, Var2) |= possible(Var2:Val2)
  & consistent(Var:Val, Var2:Val2).

43

Is it fast enough? (sort of)

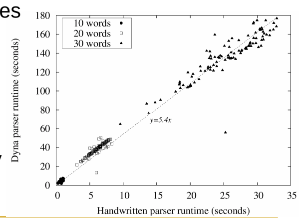
- Asymptotically efficient
- 4 times slower than Mark Johnson's inside-outside
- 4-11 times slower than Klein & Manning's Viterbi parser



44

Are you going to make it faster? (yup!)

- Currently rewriting the term classes
  to match hand-tuned code
- Will support "mix-and-match"
  implementation strategies
  - store X in an array
  - store Y in a hash
  - don't store Z
    (compute on demand)
- Eventually, choose strategies automatically by execution profiling



45

Synopsis:
today's idea → experimental results *fast!*

- Dyna is a language for computation (no I/O).
- Especially good for dynamic programming.
- It tries to encapsulate the black art of NLP.

- Much prior work in this vein …
  - Deductive parsing schemata (preferably **weighted**)
    - Goodman, Nederhof, Pereira, Warren, Shieber, Schabes, Sikkel…
  - Deductive databases (preferably with **aggregation)**
    - Ramakrishnan, Zukowski, Freitag, Specht, Ross, Sagiv, …
  - Probabilistic programming languages (**implemented**)
    - Zhao, Sato, Pfeffer … *(also: efficient Prologish languages)*

46

Contributors!

http://www.dyna.org

- **Jason Eisner**
- **Eric Goldlust**, Eric Northup, Johnny Graettinger (compiler backend)
- **Noah A. Smith** (parameter training)
- Markus Dreyer, David Smith (compiler frontend)
- Mike Kornbluh, George Shafer, Gordon Woodhull (visual debugger)
- John Blatz (program transformations)
- Asheesh Laroia (web services)

47