CS 490: NATURAL LANGUAGE PROCESSING

Dan Goldwasser, Abulhair Saparov

Lecture 13: Structured Prediction

Quiz!

Given the sentence: *She ate the cake with candles* , and the mini-grammar, we can derive two parse trees.

Tree 1:

```
S
├── NP
│   └── she
└── VP
    ├── V
    │   └── ate
    └── NP
        ├── NP
        │   ├── Det
        │   │   └── the
        │   └── N
        │       └── cake
        └── PP
            ├── P
            │   └── with
            └── NP
                ├── Det (ø implied)
                └── N
                    └── candles
```

Tree 2:

```
S
├── NP
│   └── she
└── VP
    ├── VP
    │   ├── V
    │   │   └── ate
    │   └── NP
    │       ├── Det
    │       │   └── the
    │       └── N
    │           └── cake
    └── PP
        ├── P
        │   └── with
        └── NP
            └── N
                └── candles
```

Mini Grammar:

S → NP VP
NP → Det N
NP → NP PP
VP → V NP
VP → VP PP
PP → P NP

Lexical rules

Det → she | the|Ø
N → cake | candles
V → ate
P → with

Lecture Overview

- We discussed two parsing algorithms and showed that they are special cases of a family of algorithms called **branch-and-bound.**
- These are algorithms for **discrete optimization.**
  - **E.g.,** find a parse tree for a sentence, …
  - **Or as an optimization problem –** find "the best" parse tree for a sentence assuming several valid options exist.
- In this lecture we will generalize this intuition to many NLP tasks that have a structure, i.e., **require making many interdependent decisions, that together form a structure (such as a tree).**

Lecture Overview

- We will separate between two considerations:
  - Conceptually this is a **modeling problem**: how do we decide how a structure decomposes into individual decisions, and what are the relationships between these decisions?
  - An **inference problem**: given the modeling decision we made, how to find the optimal valid structure?
  - A **learning problem**: how can we learn parameters to use as a scoring function for these decisions?
- We will introduce a general framework to answer these three questions
- We will refer to it as **structure prediction.**

Review from previous lecture

BRANCH AND BOUND

- So it seems like CKY and Earley parsing share a lot of structure.
  - Is there a unifying description?
- Branch-and-bound is a general class of algorithms for discrete optimization/search.
  - Say we want to find a target object x in a large set of objects S that maximizes some priority function $f(x)$.
    - For search, this can be a simple indicator function.
    - $f(x) = 1$ if and only if x is the object we're searching for.
  - First, we partition (i.e., "branch") the set S into subsets:

    $S_1, \ldots, S_n$

  such that their union covers the full set: $S_1 \cup \ldots \cup S_n = S$

BRANCH AND BOUND

- Next, for each subset S_i, create a new state and add it to the priority queue.
  - What should we set its priority to?
  - Ideally, it should be $\max_{x \in S_i} f(x)$.
  - But this quantity can be intractable to compute,
    - Especially if S_i is very large.
  - Instead, we can use an easy-to-compute an upper bound on this quantity:
    $g(S_i) \geq \max_{x \in S_i} f(x)$
  - Just set the priority of the new state to $g(S_i)$.
    (i.e., the "bound" step)

BRANCH AND BOUND

- Then we just repeat:
- For each iteration of the main loop,
  - Pop a state from the priority queue.
  - Partition the set into subsets (branch).
  - Push a new state for each subset with priority given by $g(\cdot)$ (bound).
- Eventually, we will pop a state with a set containing a single element $\{x\}$.
- We can compute $f(x)$ and check if it's larger than the priority of the next state in the queue.
- If so, then x is necessarily the optimal object in S.
  - Since the priority of the next state in the queue is an upper bound on $f(y)$ for all other objects y.

BRANCH AND BOUND

- Then we just repeat:
- For each iteration of the main loop,
  - Pop a state from the priority queue.
  - Partition the set into subsets (branch).
  - Push a new state for each subset with priority given by $g(\cdot)$ (bound).
- Eventually, we will pop a state with a state containing a single element $\{x\}$.
- The first such x may not be strictly more optimal than all other objects y.
  - There may be other objects y such that $f(x) = f(y)$.
- We can continue the branch-and-bound main loop to find the top-k objects.

Review from previous lecture

CKY PARSING AS BRANCH AND BOUND?

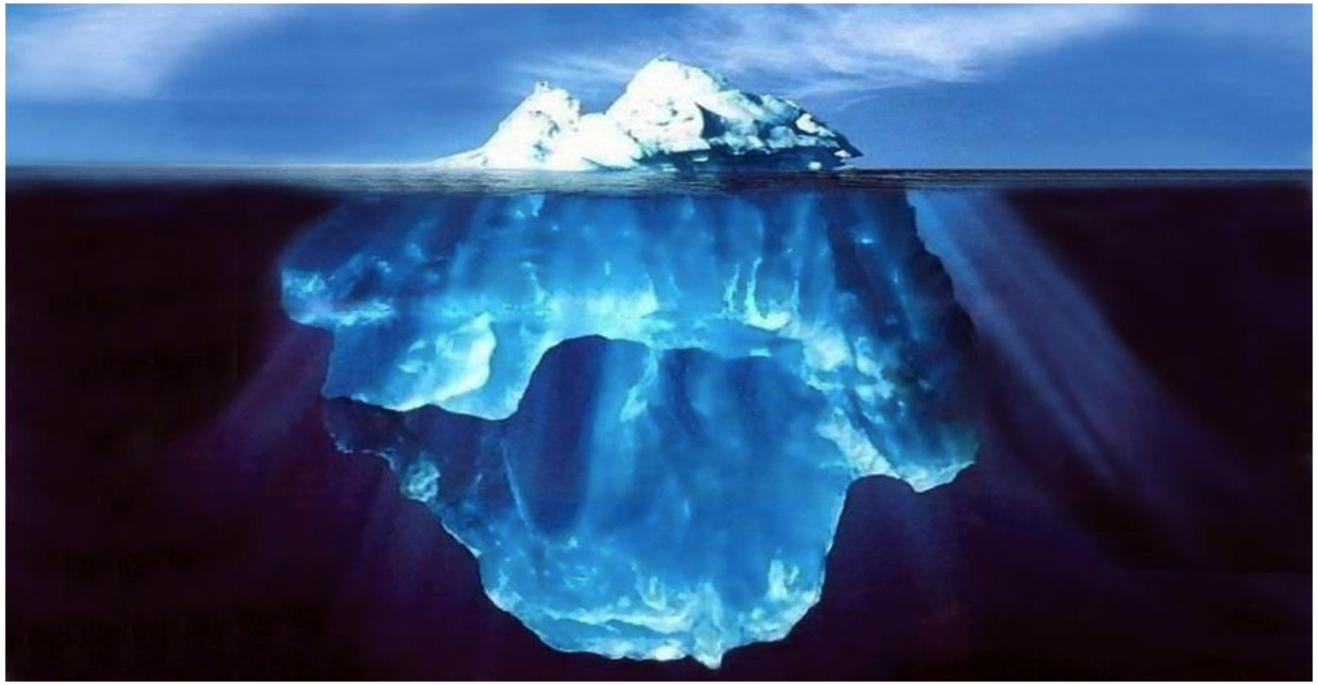
- How can we formulate CKY as a branch-and-bound algorithm?
- S is the set of all syntax trees (both valid and invalid) for a given sentence.
  (an invalid syntax tree would be one containing a rule not in the grammar)
- Recall each state in CKY is a span `(i,j)`.
  - This state represents the set of all syntax trees for the given sentence that contains a valid subtree for the subsequence starting at `i` and ending at `j`.
- Eventually, we reach the span `(0,n)` which represents the set of all valid syntax trees for the full sentence.
- What is the "branch" step?
  - When processing the state `(i,j)`, we add a new state to the queue for each `(i,m)` for `m > j` and for each `(m,j)` for `m < i`.

Review from previous lecture

CKY PARSING AS BRANCH AND BOUND?

- How can we formulate CKY as a branch-and-bound algorithm?
- S is the set of all syntax trees (both valid and invalid) for a given sentence. (an invalid syntax tree would be one containing a rule not in the grammar)
- Recall each state in CKY is a span `(i,j)`.
  - This state represents the set of all syntax trees for the given sentence that contains a valid subtree for the subsequence starting at `i` and ending at `j`.
- What is the "**bound**" $g(x)$?
  - We can set it to $g(x)$ `= i - j` so that shorter spans have higher priority.
  - But we must make sure $g(x)$ is an upper bound of the objective function. (the objective function is 1 iff x is a valid parse of the whole sentence)
  - So we can simply set $g(x)$ `= i - j + n + 1`.

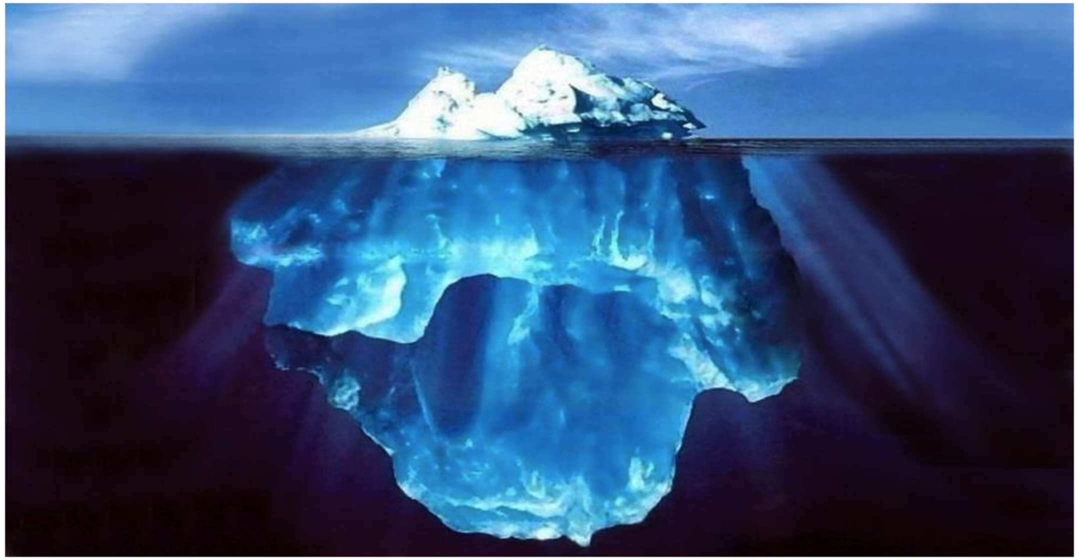
Language is **structured**

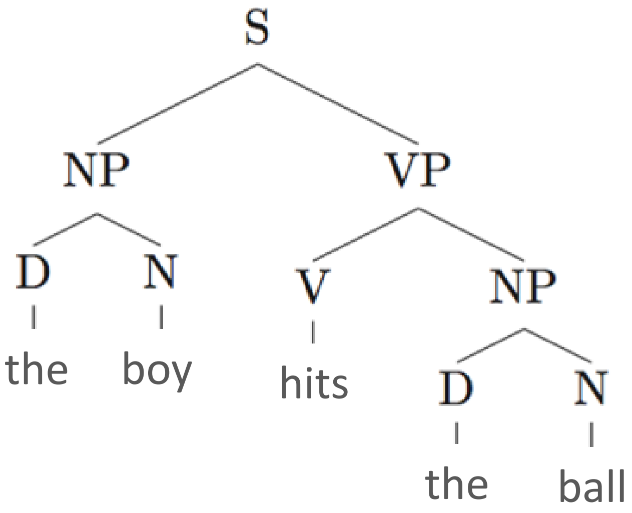


NL Structures: *the 10,000 ft View*

- ***Tip of the iceberg***

Natural language, expressed in words, is just a surface representation underlying a complex structure





*A core concept is **Inference** : generalizing the notion of **classification***

Structured Prediction

- A task in which the output consists of multiple inter-dependent decisions.
- These decisions could form specific structures: graphs, trees, sequences, etc.
- Based on the **connections between decisions**, we need a procedure for exploring the space of possible decisions and finding the optimal valid decision.
- **Branch-and-Bound** is an example of a family of such algorithms.

NL Structures: *Examples*

- Semantic role labeling
- Predict the arguments of verbs
  - “who did what to whom”
  - Large human annotated corpus of semantic relations (*PropBank* [Palmer et. al. 2005])

John was <u>going</u> to New York City

Relation : G o

Arguments
- ***Mover***[**A0**]: John
- ***Destination*** [**A 1**] : *New York City*

Predicting Verb Arguments

- **Input**: pre-processed text (space of candidates)
- **Prediction task:**
  - Identify arguments candidates
  - **Multiclass classification:** argument type, or not-an-argument
    - Assume each classifier gives a probability estimate for output
  - **Inference:** Combine all argument classification decisions
    - Should respect linguistic and structural constraint
    - E.g., no overlapping arguments

Formalizing Structured Output Prediction

- Assume for a given input **x,** a collection of decisions **y**
  - **E.g., y** consists of all derivation rules used to form a tree.

- We assume a scoring function, that scores possible output structure
  - E.g., invalid trees are scored negatively, while valid trees have a positive score.

- The decision can be formulated as solving: $\underset{\boldsymbol{y} \in \boldsymbol{Y}}{\operatorname{argmax}} \boldsymbol{Score}(\boldsymbol{x}, \boldsymbol{y})$

- We note that there are countably many **y** possibilities (e.g., a finite number of possible parse trees for a sentence). Why not just reduce the problem to multiclass prediction?

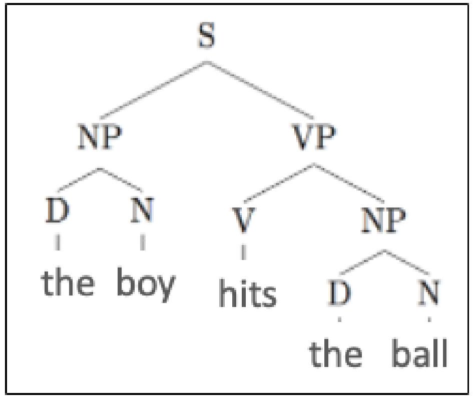
Take 1: *reduce to multiclass*

- Our view of multiclass classification: A set of labels.
  - Define feature functions: $\phi(x,y)$
  - **Prediction**: $\text{argmax}_y \; w_y^T\phi(x,y)$
  - **Training**: find W s.t. $\text{argmax}_y \; w_y^T\phi()$ will return correct label
- **Can we define the structure prediction this ways?**
  - What are the classes for:

    "The boy hits the ball" ?

  - S_0: <D-N-NP-V-D-N-NP-VP-S>
  - S_1: <N-N-NP-V-D-N-NP-VP-S>
  - S_2: <N-D-NP-V-D-N-NP-VP-S>

    …

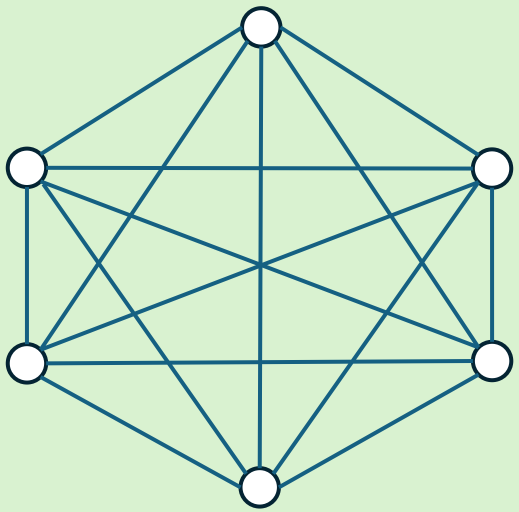


Take 2: *decompose the output*

- **We cannot directly reduce structured prediction to multiclass classification**
  - Enumerating all possible structures is infeasible
  - Maintaining parameters of all possible structures separately
- Instead, decompose the output into parts:
  - **Instead of scoring structures, we can score parts**
  - **Use an Inference algorithm to construct structure from parts**

- **Question**: *how do decisions influence one another?*

Output Decomposition



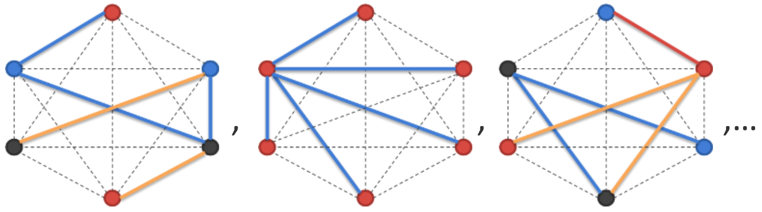
3 possible node labels

3 possible edge labels

Output: Nodes and edges are labeled
The blue and orange edges **form a tree**

Goal: Find the highest scoring tree

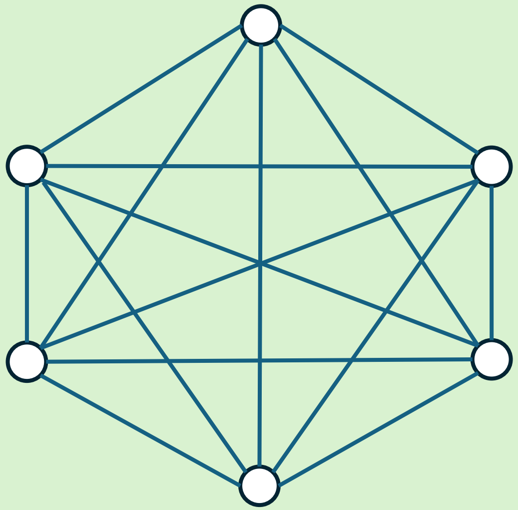
Possible outputs



Assume a scoring function that can score the possible outputs

We would like to break the decision into smaller decisions, and score them independently

Output Decomposition



3 possible node labels

3 possible edge labels

Output: Nodes and edges are labeled
The blue and orange edges **form a tree**

Option 1: Independent decisions

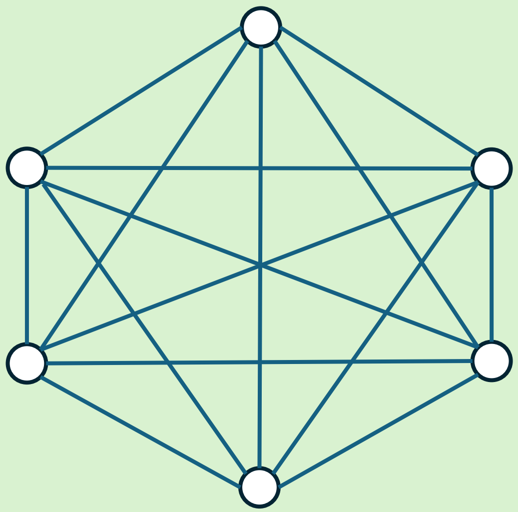
Each node and edge are scored independently.

$$\boldsymbol{Score}(\boldsymbol{x},\boldsymbol{y}) = \sum_{\boldsymbol{n}\in \boldsymbol{N}} \boldsymbol{score}(\boldsymbol{n}) + \sum_{\boldsymbol{e}\in \boldsymbol{E}} \boldsymbol{score}(\boldsymbol{e})$$

$$\operatorname{argmax}_{\boldsymbol{y}} \boldsymbol{score}(\boldsymbol{x},\boldsymbol{y})$$
$$\text{s.t } \boldsymbol{y} \text{ forms a tree}$$

Still need to ensure output legality (tree)

Decomposing the output: Example

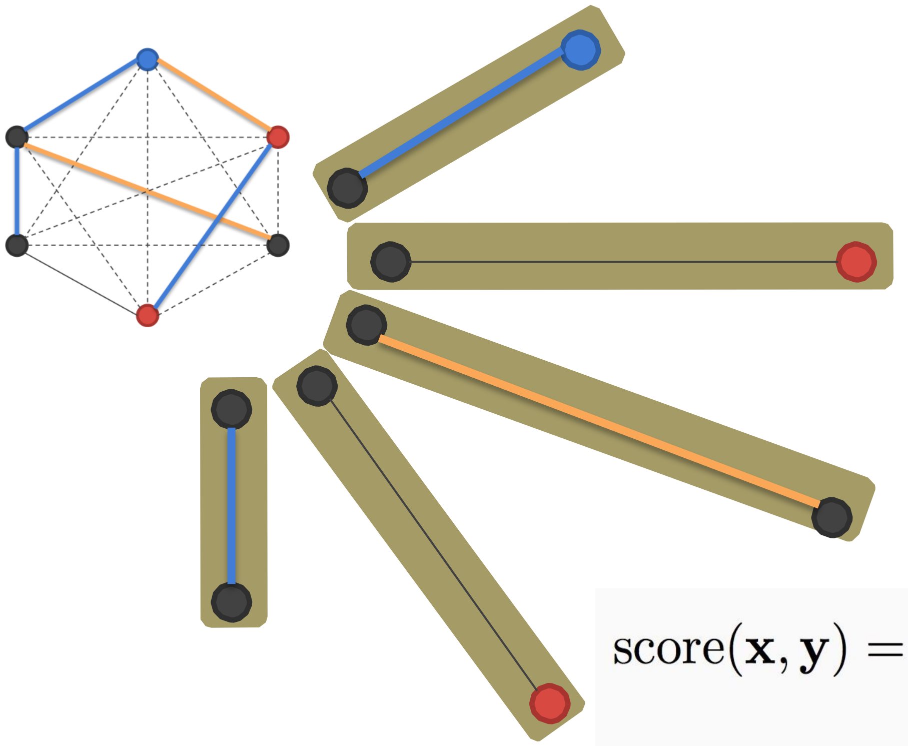


3 possible node labels

3 possible edge labels

Output: Nodes and edges are labeled
The blue and orange edges **form a tree**

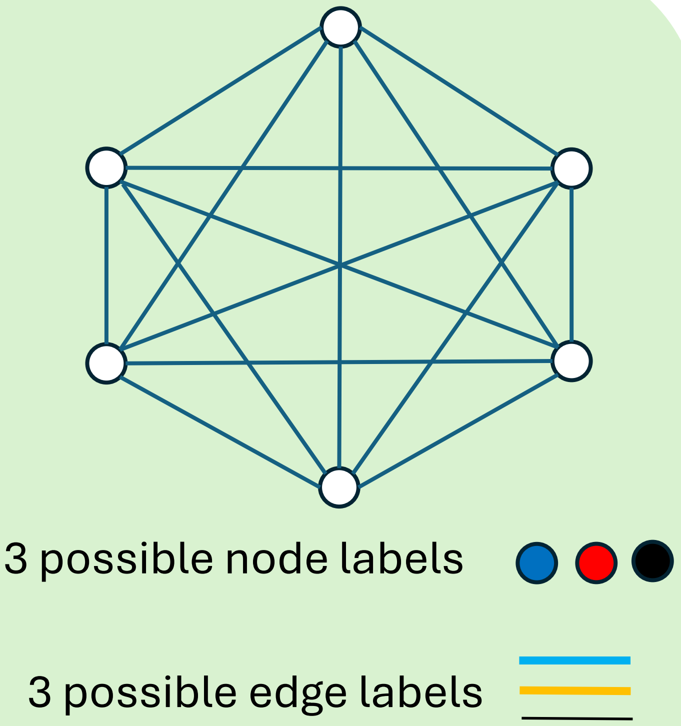
Option 2: **score each edges and its nodes together**



Each edge+ nodes is scored together.
Note: the decision procedure has to ensure consistency between these decisions!

$$\text{score}(\mathbf{x},\mathbf{y}) = \sum_{\substack{n_1,n_2 \in \text{ nodes}(\mathbf{x},\mathbf{y}) \\ e \in \text{ edges}(\mathbf{x},\mathbf{y})}} \text{score}(n_1, n_2, e)$$

Decomposing the output: Example



We have seen two examples of decomposition

Which one is better?

What is the tradeoff?

Training Structured Prediction models

- Decomposition of outputs gives two approaches for training
  - **Decomposed training**
    - Learning **local models, no inference during training**
  - **Global/joint training**
    - Learning algorithm uses the final prediction procedure during training
- Similar to the two strategies we had before with multiclass
- **Inference complexity** is an important consideration in choice of modeling and training

Sequence Models

- Let's start with an easy case: **sequence models.**

- **Many NLP problems can be formulated as a sequence prediction problem:**
  - Part-of-speech tagging, chunking (shallow parsing), NER, etc.
  - Autoregressive language models, translation, etc.

DT	JJ	NN	VBZ	IN	DT	JJ	NN
The	brown	fox	jumps	over	the	lazy	dog

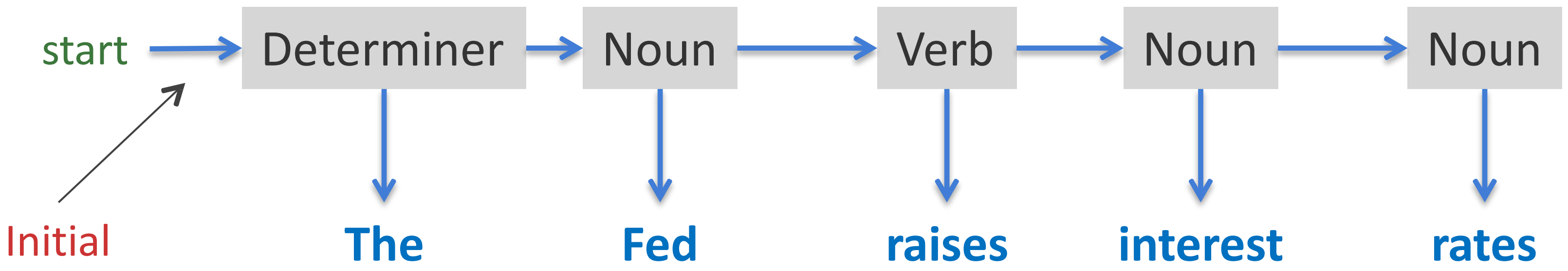
HMM

- Independence assumptions: the probability of an output sequence (e.g., POS tags) is defined by breaking it into a product of **emission** and **transition** probabilities.

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

P(The | DT) = 0.5
P(A | DT) = 0.3
P(An | DT) = 0.1
P(Fed | DT) = 0
…

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
…



Decoding (prediction)

Given an observation sequence and an HMM, we need to find the **optimal state sequence**:

$$\arg\max_{y} p(x_1, .., x_n | y_1, ..., y_n) p(y_1 ..., y_n)$$

- How can we find it?
  - **Combinatorial optimization problem**

Wrong Way: Enumerating all assignments

NNP John NNP ate VBD lunch NNP at NNP the NNP watering NNP hole NNP cafe ti.tititi1

DET John DET ate VBD lunch NNP at NNP the NNP watering NNP hole NNP cafe ti.tititi 2

VBD John VBD ate VBD lunch NNP at NNP the NNP watering NNP hole NNP cafe ti.tititi 2

...

NNP John VBD ate NN lunch IN at DT the NN watering NN hole NN cafe ti.ti1ti2

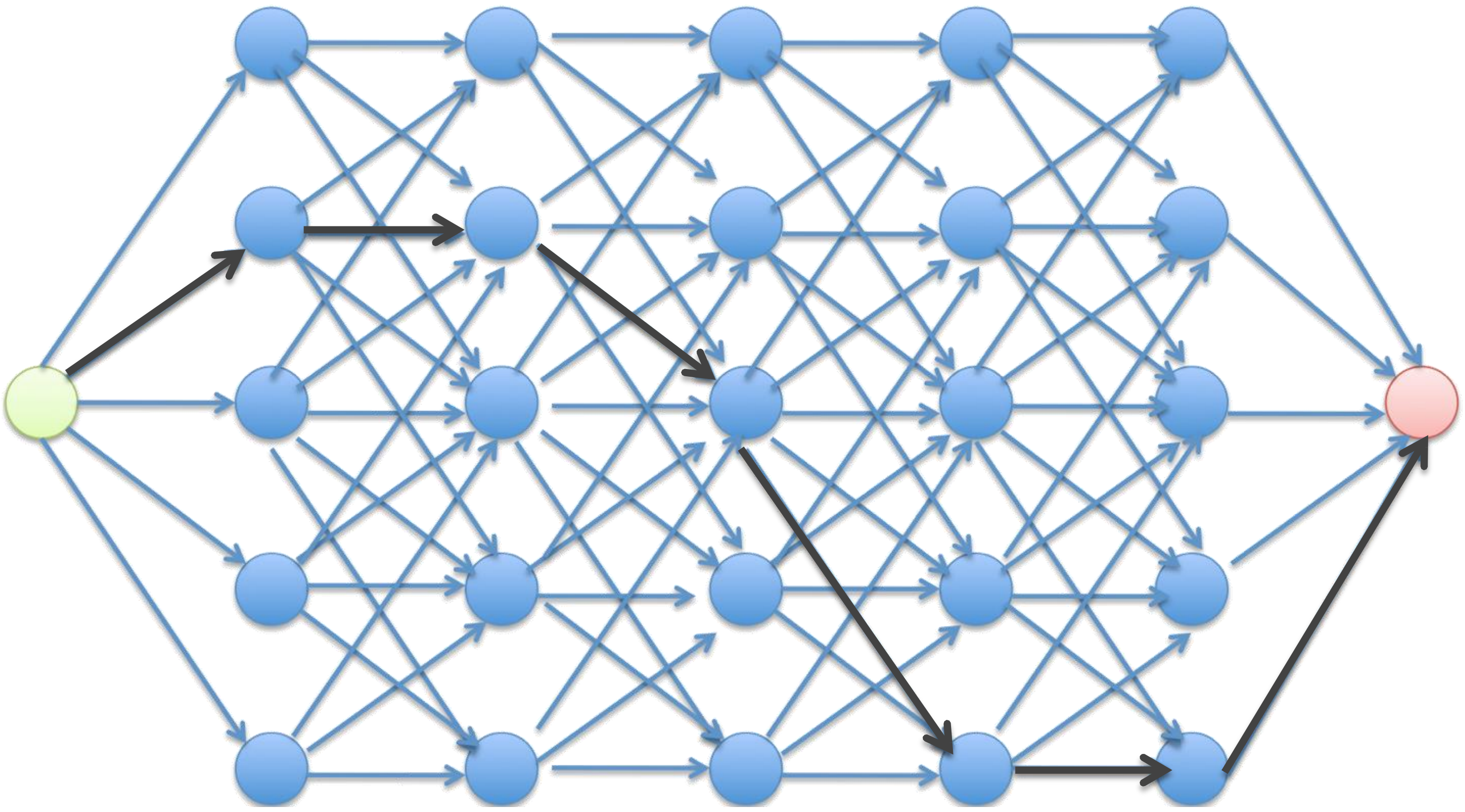
How many possible assignments are there?

Basic idea: $$\arg\max_y \prod_{i=1}^{n} p(x_i|y_i) \prod_{i=1}^{n} p(y_i|y_{i-1})$$

Independence assumptions lead to an algorithmic solution!

Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



Viterbi Algorithm

- **Definitions:**
  - n : length of input, S_k : possible symbols at position *k*

Truncated version of the probability *(defined over k long sequences, k<n)*

$$r(y_1, .., y_k) = \prod_{i=1}^{k} p(y_i|y_{i-1}) \prod_{i=1}^{k} p(x_i|y_i)$$

DP table:

$$\pi(k, v) = max_{(y_1,...,y_k; y_k = v)} r(y_1, ..., y_k)$$
$$\text{max probability tag sequence of size k ending with v}$$

Recursive definition of DP table:

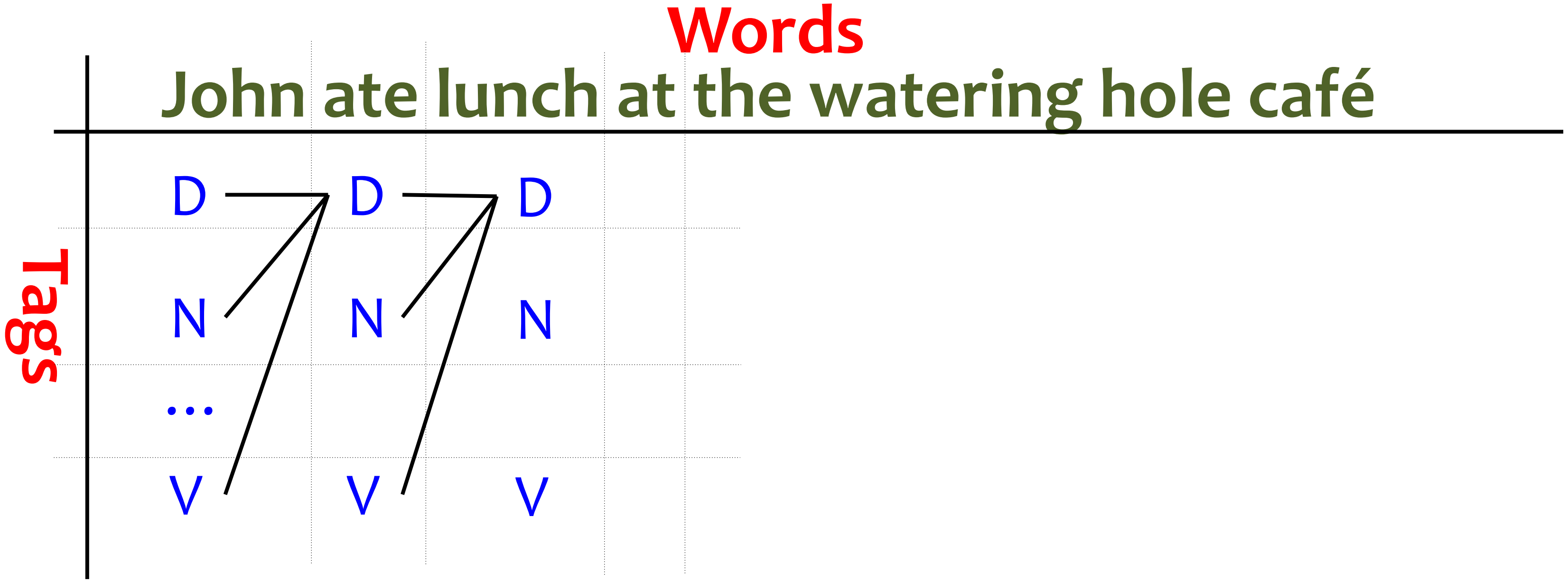
$$\pi(k, v) = \max_{u \in S_k - 1} (\pi(k-1, u) \times p(v|u) \times p(x_k|v))$$

Viterbi: DP Table

$$\pi(k, v) = max_{(y_1,\ldots,y_k;y_k=v)} r(y_1, \ldots, y_k)$$

max probability tag sequence of size k ending with v

$$\pi(k, v) = \max_{u \in S_k - 1} (\pi(k-1, u) \times p(v|u) \times p(x_k|v))$$



$$\pi(3, D) = max_{(y_1,\ldots,y_3;y_3=D)} r(y_1, \ldots, y_3)$$

The Viterbi Algorithm

Input: a sequence $x_1, .., x_n$,

parameters: $p(s|u),\ p(x|s)\ \forall s, u \in S$

Initialization: $\pi(0, \epsilon) = 1$

(Note: ϵ is just a start symbol)

For $k = 1..n$

For $v \in S_k$

$\pi(k, v) = \max_{u \in S_{k-1}} (\pi(k-1, u) \times p(v|u) \times p(x_k|v)$

Return $max_{u \in S_n} (\pi(n, u) \times p(\sigma|u))$

(Note: σ is just an end symbol)

Note:
We augment the set of tags with *start* symbol and compute parameters for these symbols

What is the run time complexity of Viterbi?

What does this algorithm return?

We are interested in the optimal sequence!

Solution: small modification to the algorithm, maintain a list of backpointers

In practice: smoothing is required!

Parameter Estimation

- **Two terms:** $p(y_i|y_{i-1})$ *(transitions probabilities)* $p(x_i|y_i)$ *(emission probabilities)*

$p(y_i|y_{i-1})$ $$p(NN|DET) = \frac{count(NN, DET)}{count(DET)}$$

$$A_{s',s} = \frac{\text{count}\,(s \to s')}{\text{count}\,(s)}$$

$p(x_i|y_i)$ $$p(\text{“}watering\text{”}|NN) = \frac{count(\text{“}watering\text{”}, NN)}{count(NN)}$$

$$B_{s,x} = \frac{\text{count}\begin{pmatrix} s \\ \downarrow \\ x \end{pmatrix}}{\text{count}\,(s)}$$

Initial state probability $$\pi_s = \frac{\text{count}(\text{start} \to s)}{n}$$

Generative vs. Discriminative

- **HMM:** *Model for the joint probability of (x,y)*

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

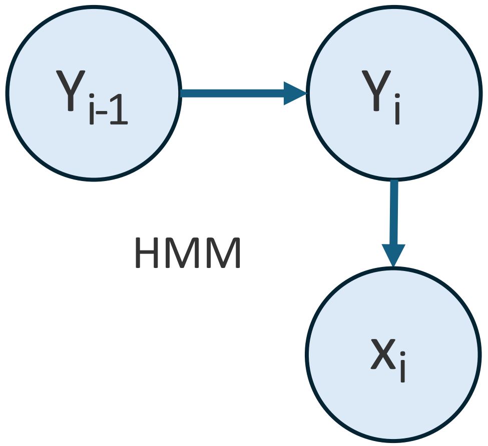
At prediction time we care about the probability of **output given the input**

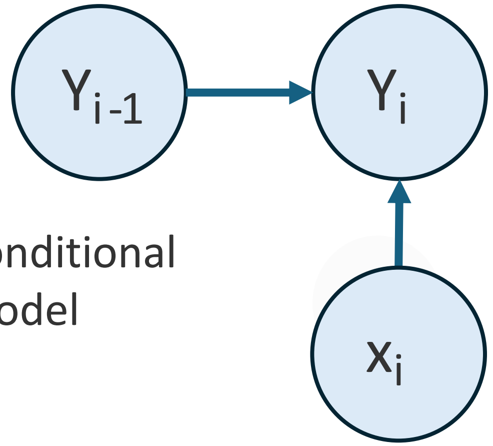
Why not directly optimize this conditional likelihood instead?

- Instead of modeling the joint distribution P(**x**, **y**) only focus on P(**y**|**x**)
  - Maximum Entropy Markov Model (MEMM) [McCallum, et al]
  - Essentially: train a "next state classifier", and use inference to chain the decision together.

Conditional Models

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$





This assumption lets us write the conditional probability of the output as

The probability of the entire structure → $P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|y_{i-1}, x_i)$ We need to learn this function

Note that at decision time, we use inference (e.g., the Viterbi algorithm) to find the max probability sequence: $\underset{\boldsymbol{y}}{\text{argmax}}\, \boldsymbol{score}(\boldsymbol{x}, \boldsymbol{y}) = \underset{\boldsymbol{y}}{\text{argmax}} \prod_{\boldsymbol{i}} \mathrm{P}(y_i|\, y_{i-1}, x_i)$

Using MEMM

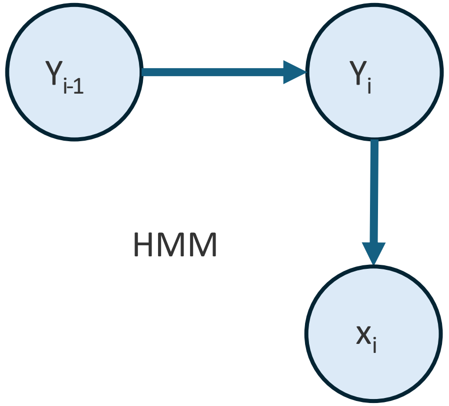
- **Training**
  - ***Local*** *next state classifier, over the previous output and the current input*
  - *You can use* ***any*** *classifier (e.g., Logistic Regression, SVM, NN, …)*

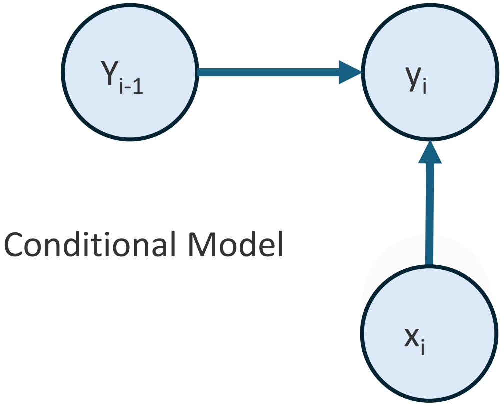
Prediction/decoding

Modify the Viterbi algorithm for the new independence assumptions

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1}, x_i)\text{score}_{i-1}(y_{i-1})$$





Global models

- **Train the predictor globally**
  - Instead of training local decisions independently
- **Normalize globally**
  - Make each edge in the model undirected
  - Not associated with a probability, but just a "score"
- *Recall the difference between local vs. global for multiclass*

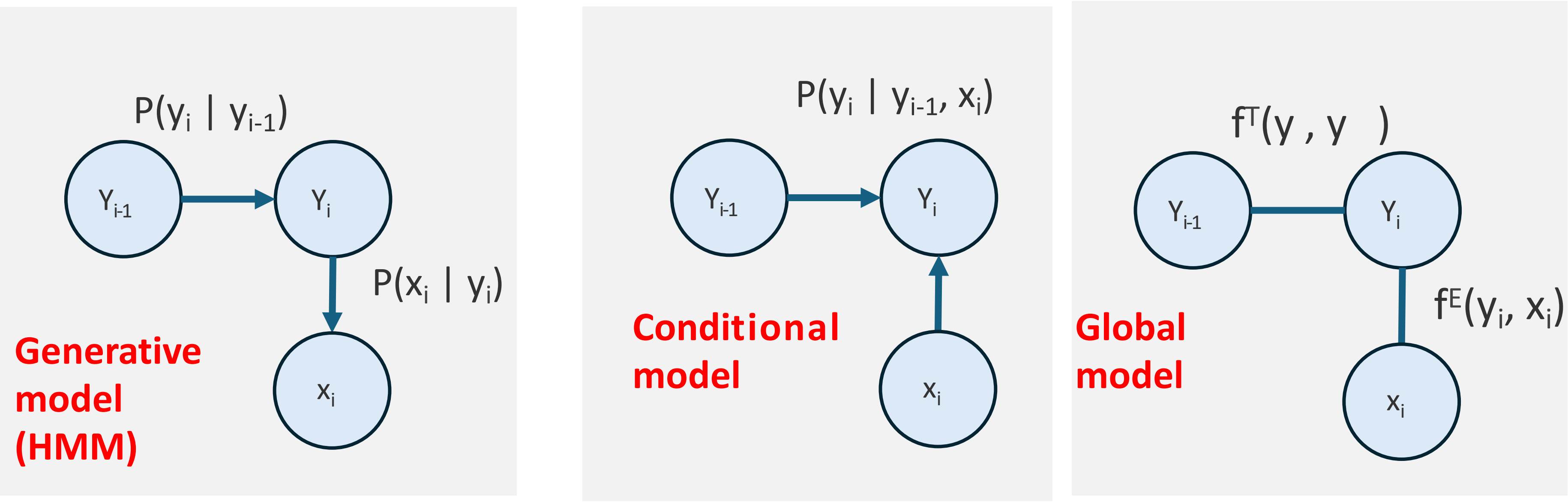
Local vs. Global Models

- So far, we've seen **Local Models,** trained to make local decisions independently

- Instead, we can train the predictor **globally**
  - Instead of a next-state probabilities, **learn a probability distribution over entire structures!**

- **Normalize globally**
  - Make each edge in the model undirected
  - Not associated with a probability, but just a score
  - Normalize these scores to form a distribution.

HMM vs. a **local model** vs. a **global model**



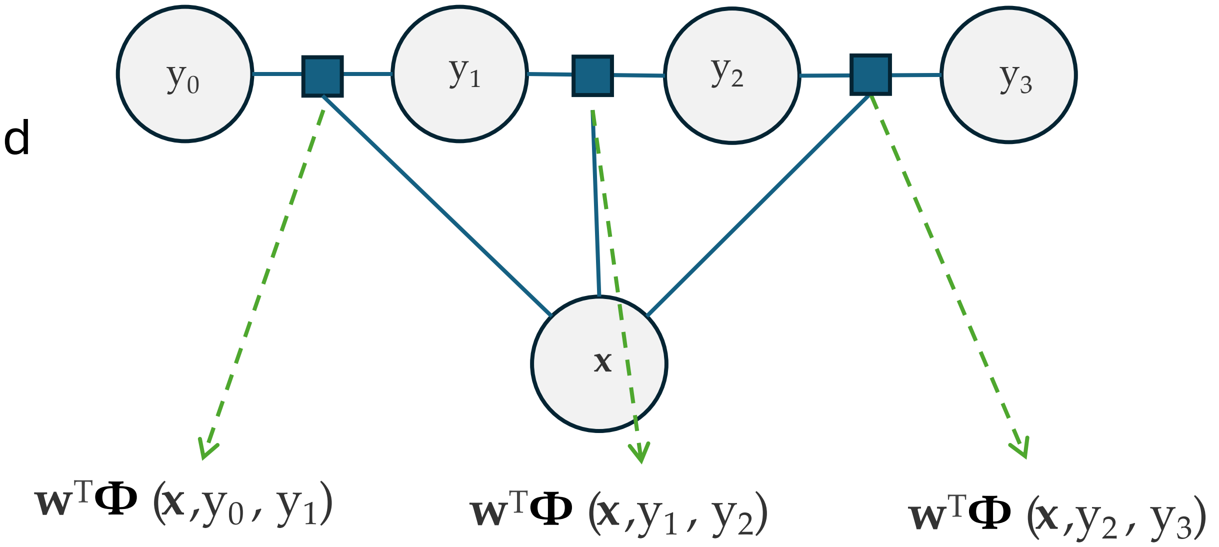
Generative **Discriminative**

Local: P is locally normalized to add up to one for each time step

Global: The functions f^T and f^E are scores that **are not normalized**

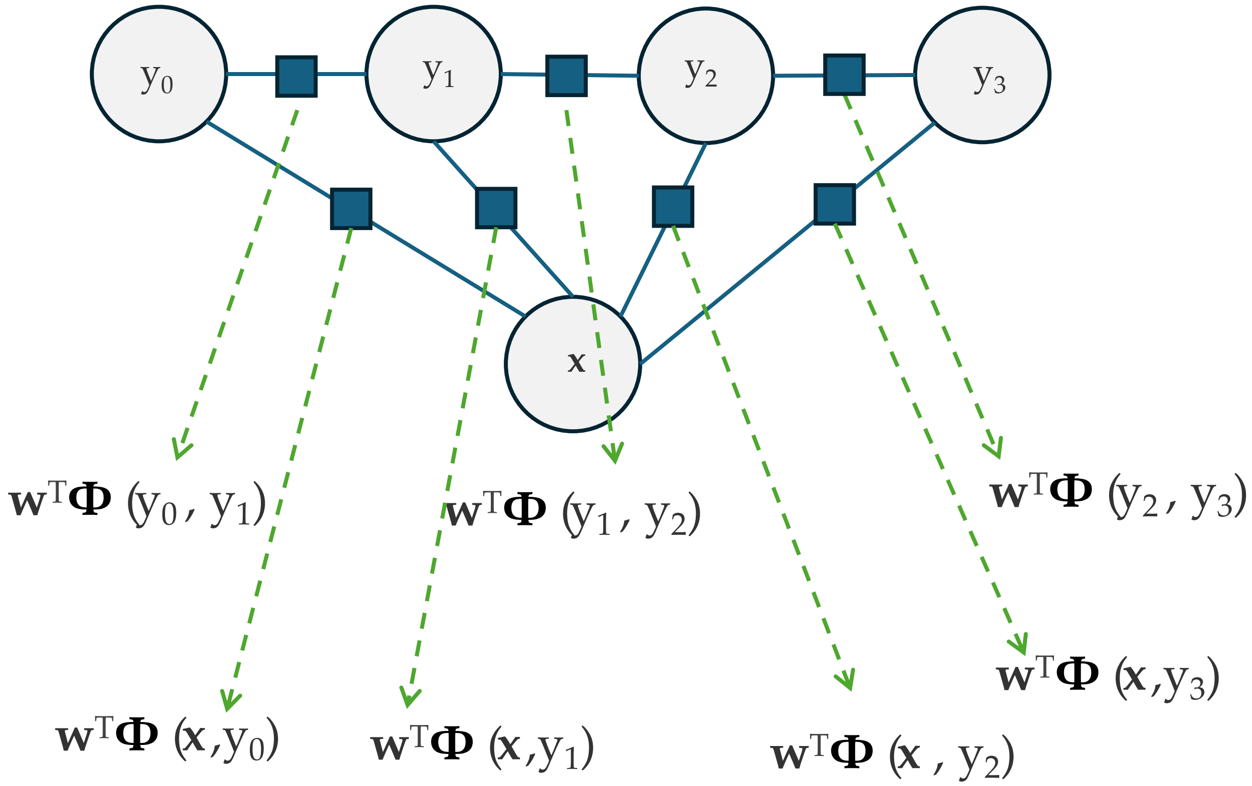
Conditional Random Field

- Factor Graph representation
  - Each node is a random variable
  - The input node is assigned
  - Factor nodes (squares) are associated with a scoring function, based on the nodes it is connected to.

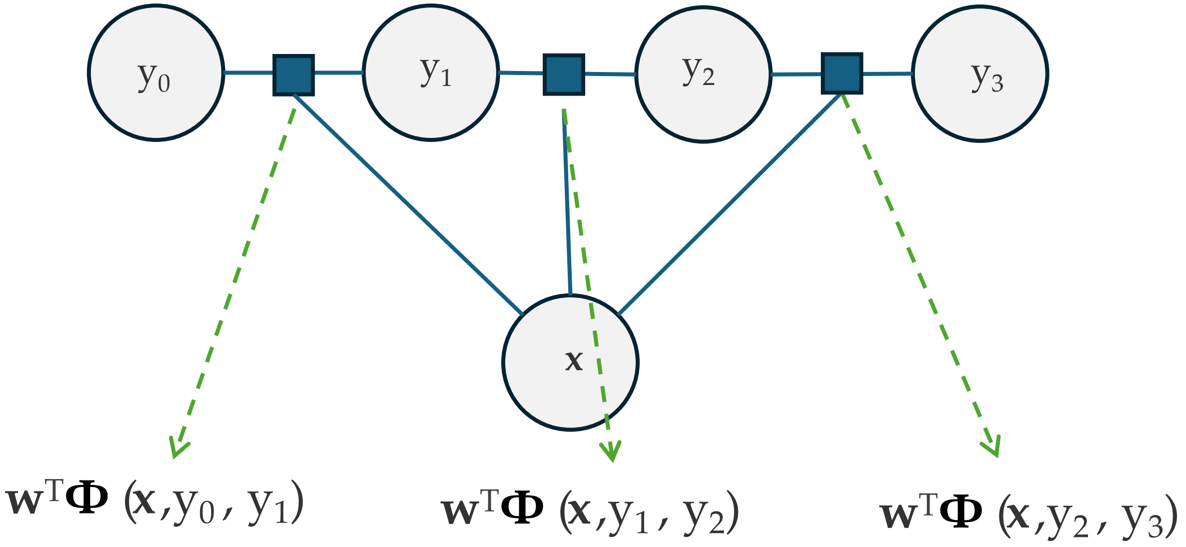


Conditional Random Field

- Factor Graph representation
- The factor graph captures how the problem decomposes into smaller decisions.
- This example shows a different **factorization**



Conditional Random Field for sequences



Assign a (conditional) probability to the entire sequence

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_i \exp\left(\mathbf{w}^T\phi(\mathbf{x}, y_i, y_{i-1})\right)$$

$$Z = \sum_{\hat{\mathbf{y}}}\prod_i \exp\left(\mathbf{w}^T\phi(\mathbf{x}, \hat{y}_i, \hat{y}_{i-1})\right)$$

Z: Normalizing constant, sum over all sequences

Conditional Random Fields

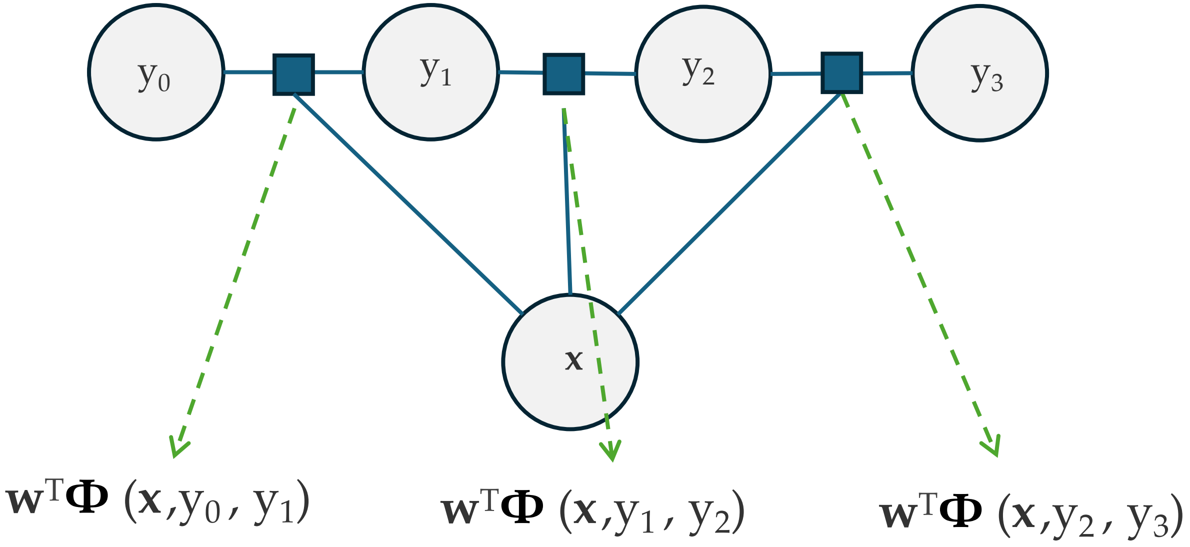
- The model can be viewed as a generalization of a log-linear model
- The joint feature function represents the entire structural decision

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp\left(\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})\right)}{\sum_{\mathbf{y}'} \exp\left(\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}')\right)}$$

Global features

The feature function decomposes over the sequence

→ Aggregates all active features into a global representation



$$\phi(\mathbf{x}, \mathbf{y}) = \sum_i \phi(\mathbf{x}, y_i, y_{i-1})$$

Conditional Random Fields

- The joint feature function represents the entire structural decision

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp\left(\mathbf{w}^T\phi(\mathbf{x},\mathbf{y})\right)}{\sum_{\mathbf{y}'}\exp\left(\mathbf{w}^T\phi(\mathbf{x},\mathbf{y}')\right)}$$

- Prediction: $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} \exp(\mathbf{w}^T\phi(\mathbf{x},\mathbf{y})) = \arg\max_{\mathbf{y}} \mathbf{w}^T\phi(\mathbf{x},\mathbf{y})$

- But since the score decomposes into sequential decisions:

$$\mathbf{w}^T\phi(\mathbf{x},\mathbf{y}) = \sum_i \mathbf{w}^T\phi(\mathbf{x}, y_i, y_{i-1})$$

- Prediction can be done using Viterbi:

$$\text{score}_0(s) = \mathbf{w}^T\phi(\mathbf{x}, y_0, \text{start})$$

$$\text{score}_i(s) = \max_{y_{i-1}}\left(\mathbf{w}^T\phi(\mathbf{x}, y_i, y_{i-1}) + \text{score}_{i-1}(y_{i-1})\right)$$

Training CRF

Maximize the (regularized) log-likelihood

$$\max_{\mathbf{w}} -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})$$

Gradient ascent update:

$$\mathbf{w} \leftarrow \mathbf{w} + \sum_i \left(\phi(\mathbf{x}_i, \mathbf{y}_i) - \sum_{\hat{\mathbf{y}}} P(\hat{\mathbf{y}}|\mathbf{x}_i, \mathbf{w})\phi(\mathbf{x}_i, \hat{\mathbf{y}})\right)$$

This is an instance of a big idea in training global models:
Inference-based Training

Training involves inference!

- A different kind than what we have seen so far
- Summing over all sequences is just like Viterbi
  - *With summation instead of maximization*

Summary

- Global training: learn a probability distribution over entire structure (vs. probability of local transitions).

- Many methods exist, e.g., Perceptron/SVM structural variants
  - Note: computing the gradient step requires **inference**

Structure Learning and Prediction

- Essentially - *"learning the parameters of a combinatorial optimization problem"*

$$\text{argmax}_{y} \ \text{score}(w, \mathbf{x}, \mathbf{y})$$

Algorithmic question: ***how to search the space of possible structures?***

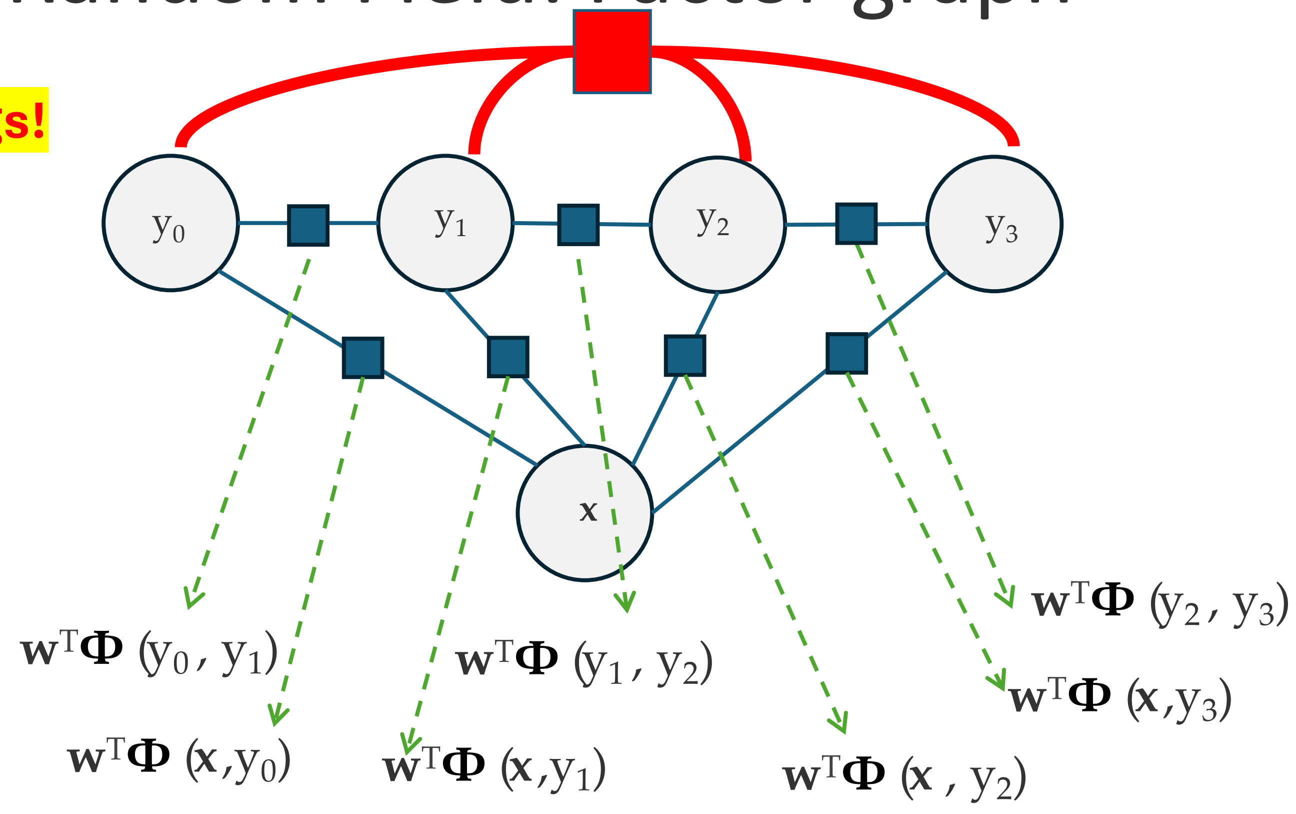
- Approximate or exact?
- Brute-force? Exploit problem structure?

Modeling question: ***how does the structure decompose into parts?***

- What is the scoring function used to represent parts?
- What are the dependencies between parts?
- **Related to algorithmic questions!**

Conditional Random Field: Factor graph

This would change things!
Why?



Adding a global scoring function (e.g., *all POS tags sequences should include at least one verb*), is useful but VERY expansive!

→ Sequence decision is no longer sequential! (i.e., cannot use Viterbi anymore)

Integer Linear Programming

- An expensive, declarative alternative to search algorithms.
- Explicitly states the objective of the search.
- General form:

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0. \end{aligned}$$

- *Solution (assignment to x) has to be an integer!*
  - *We will look at a subset, 0-1 ILP*

Integer Linear Programming

- The CEO problem:
  - We have 8 short wood pieces, and 6 long wood pieces
  - Table requires 2 long pieces, 2 short pieces
  - Chair requires 1 long pieces, 2 short pieces
  - We can sell tables for $20, chairs for $15
- ***We want to maximize our profits!***

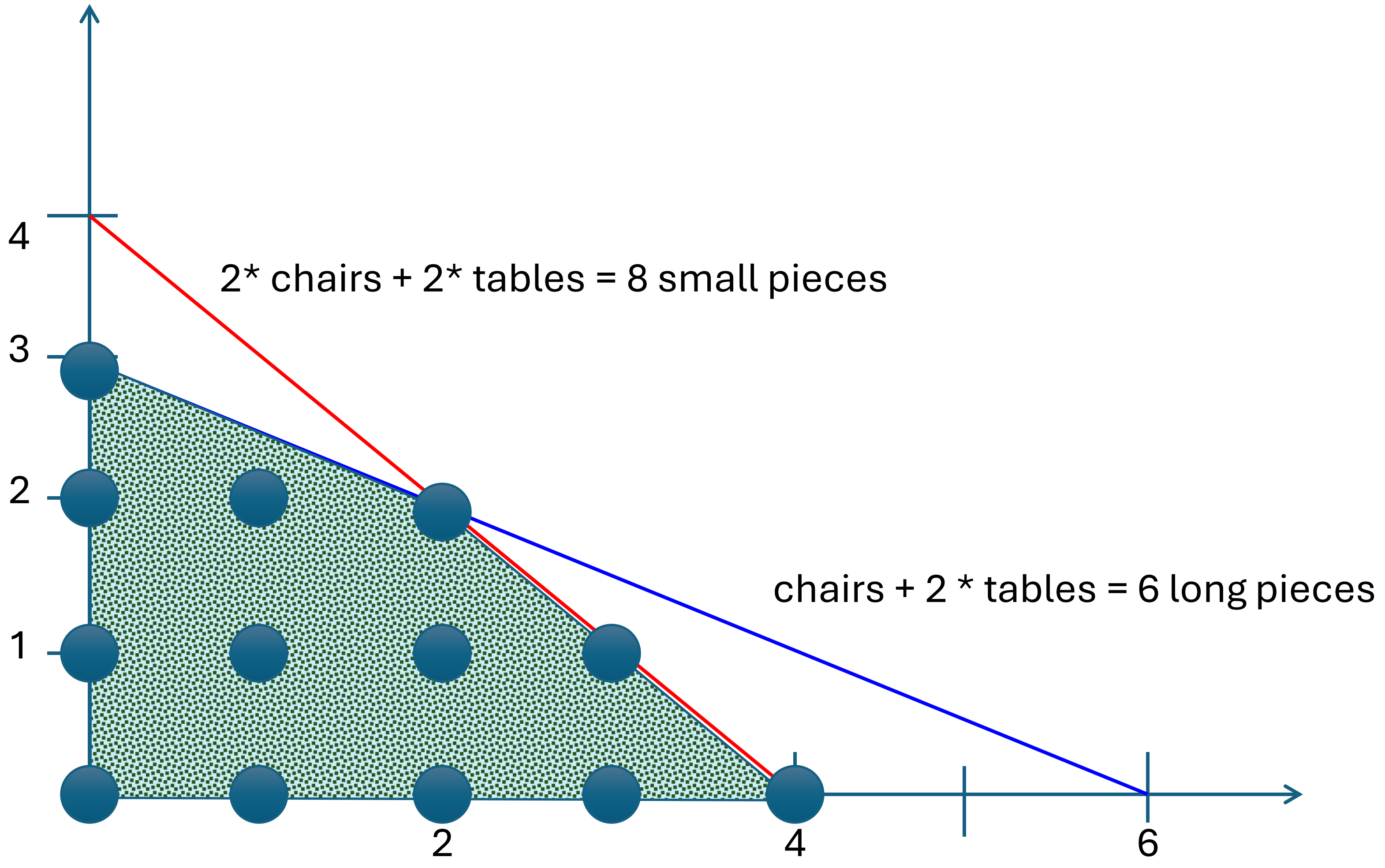
```
Max 15 * chairs + 20 * tables

Subject to

Long pieces:    chairs + 2 tables ≤ 6
Short pieces:   2 chairs + 2 tables ≤ 8

(chairs≥0, tables≥0)
```

Integer Linear Programming



How can we use ILP for inference?

- ILP is a very convenient way to express many combinatorial optimization problems!
- Let's start with an easy example: ***multi class classification***

  - z_A = 1 if output = A, 0 otherwise
  - z_B = 1 if output = B, 0 otherwise
  - z_C = 1 if output = C, 0 otherwise

$$\max_z z_A \cdot c(A) + z_B \cdot c(B) + z_C \cdot c(C) \quad (\textit{maximize the score})$$

$$s.t.$$

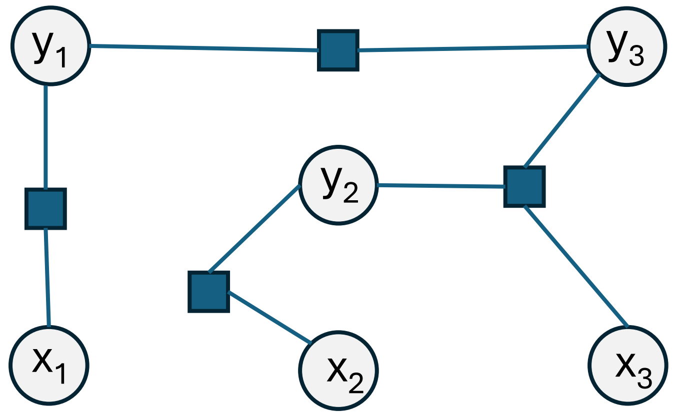
$$z_A, z_B, z_C \in \{0, 1\}$$

$$z_A + z_B + z_C = 1 \quad (\textit{only a single label can be active})$$

How can we use ILP for inference?

- we assign a decision variable for each factor

$$\max_{y} \mathrm{w}^T \phi(x_1, y_1) + \mathrm{w}^T \phi(y_1, y_3) + \mathrm{w}^T \phi(x_3, y_2, y_3) + \mathrm{w}^T \phi(x_1, x_2, y_2)$$

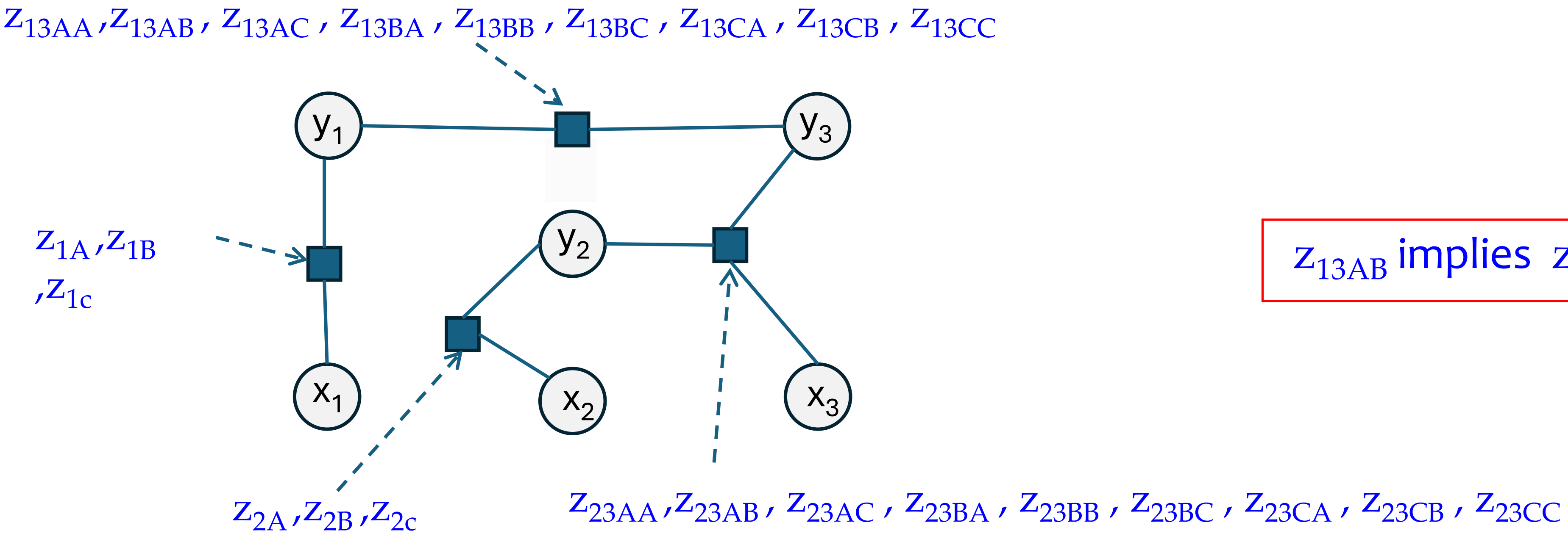


How can we use ILP for inference?

- we assign a decision variable for each factor

$$\max_{\mathbf{z}} \quad \sum_l z_{1l} s_{1l} + \sum_l z_{2l} s_{2l} + \sum_{l,l'} z_{13ll'} s_{13ll'} \sum_{l,l'} z_{23ll'} s_{23ll'}$$

s.t Only valid output allowed



z_{13AB} implies $z_{1A} \wedge z_{3B}$

Slides adapted from V. Srikumar slides

Adding Constraints

- Boolean formulas can be converted into linear constraints
- One out of $z_1,..z_k$

$$z_1+..+z_k = 1$$

- At least m out $z_1,..z_k$

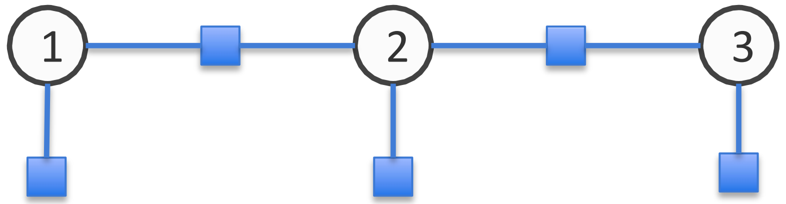
$$z_1+..+z_k \geq m$$

- Implication: $z_1 \rightarrow z_k$

$$z_k \geq z_1$$

Let's practice!

- How would you write a sequence labeling problem as ILP instance?



Connection to Branch-and-Bound

- ILP can be viewed as a general-purpose inference procedure for structured decision.
- Many solvers implement a branch-and-bound approach for solving ILP problems.
- Another, cheaper alternative, relax the inference process to be **approximate,** i.e., optimal solution is not guaranteed.
  - LP relaxation of ILP
  - Search based approaches (e.g., beam search)

Connections to **Branch and Bound**

Key idea: Given $\text{Min}\ \{\boldsymbol{c}^T\boldsymbol{x}: \text{A}\mathbf{x}=\mathbf{b},\ \mathbf{x} \geq \mathbf{0}, \boldsymbol{x} \in \mathbb{Z}\}$, divide the search space into regions (branch) by solving an LP problem (efficient algorithm exists), repeat by solving another LP instance for each region until an integer solution is found (bound)

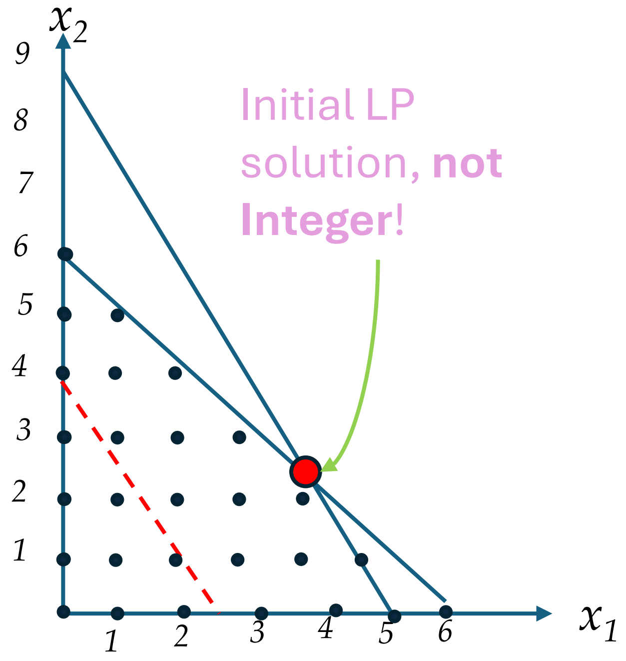
Branch: Partition the feasible region into subregions.

(1) solve a relaxed LP problem (i.e., drop integrality constraints).
(2) Let $\bar{x}$ denote the solution for the LP instance.
(3) if $\bar{x}$ is an integer: Done! (solves ILP problem).
(4) Otherwise, break into two problems, based on the i-th coordinate, $\bar{x}_i$, that is fractional.

$$\text{Min}\ \{\boldsymbol{c}^T\boldsymbol{x}: \text{A}\mathbf{x}=\mathbf{b},\ \boldsymbol{x_i} \leq \lfloor\bar{x}_i\rfloor, \mathbb{Z}, \mathbf{x} \geq \mathbf{0}, \boldsymbol{x} \in \mathbb{Z}\}$$

$$\text{Min}\ \{\boldsymbol{c}^T\boldsymbol{x}: \text{A}\mathbf{x}=\mathbf{b},\ \boldsymbol{x_i} \geq \lfloor\bar{x}_i\rfloor + 1, \mathbb{Z}, \mathbf{x} \geq \mathbf{0}, \boldsymbol{x} \in \mathbb{Z}\}$$

Bound: Determine a lower bound on the optimal value of the subproblems by solving a linear relaxation (LP)



Connections to **Branch and Bound**

Key idea: Given $\mathrm{Min}\ \{\boldsymbol{c}^{\boldsymbol{T}}\boldsymbol{x}\text{: A}\mathbf{x}\text{=}\mathbf{b},\ \mathbf{x} \geq \mathbf{0}, \boldsymbol{x} \in \mathbb{Z}\}$, divide the search space into regions (branch) by solving an LP problem (efficient algorithm exists), repeat by solving another LP instance for each region until an integer solution is found (bound)

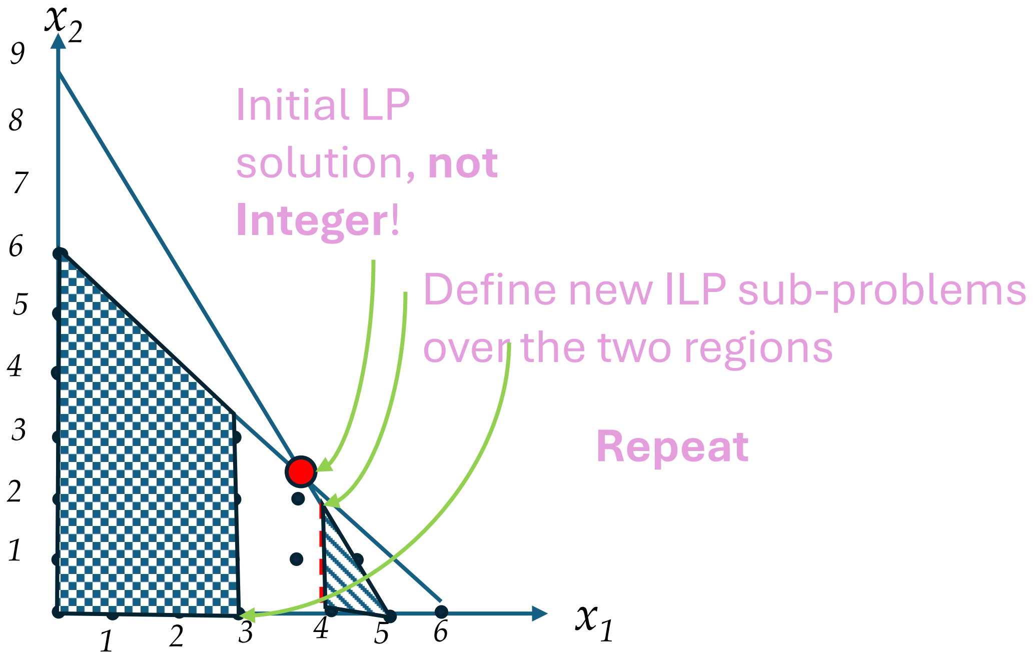
Branch: Partition the feasible region into subregions.

(1) solve a relaxed LP problem (i.e., drop integrality constraints).
(2) Let $\bar{x}$ denote the solution for the LP instance.
(3) if $\bar{x}$ is an integer: Done! (solves ILP problem).
(4) Otherwise, break into two problems, based on the i-th coordinate, $\bar{x}_i$, that is fractional.

$$\mathrm{Min}\ \{\boldsymbol{c}^{\boldsymbol{T}}\boldsymbol{x}\text{: A}\mathbf{x}\text{=}\mathbf{b},\ \boldsymbol{x_i} \leq |\bar{x}_i|, \mathbb{Z}, \mathbf{x} \geq \mathbf{0}, \boldsymbol{x} \in \mathbb{Z}\}$$

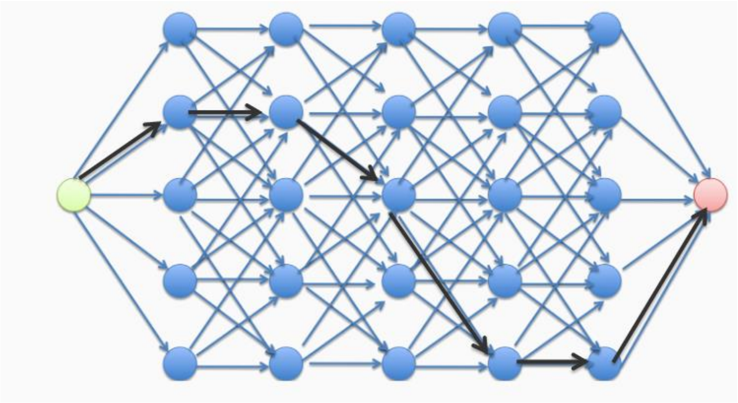
$$\mathrm{Min}\ \{\boldsymbol{c}^{\boldsymbol{T}}\boldsymbol{x}\text{: A}\mathbf{x}\text{=}\mathbf{b},\ \boldsymbol{x_i} \geq |\bar{x}_i| + 1, \mathbb{Z}, \mathbf{x} \geq \mathbf{0}, \boldsymbol{x} \in \mathbb{Z}\}$$

Bound: Determine a lower bound on the optimal value of the subproblems by solving a linear relaxation (LP)



Inference as Search

- Viterbi can be though of as a search problem



- This is a version of **exact search**
- Instead, we can also run cheaper greedy search
  - At each step, take the highest scoring transition
  - **Is this a problem?**
- Greedy algorithms are sometimes optimal!
  - Sub-modular functions

Inference as Search

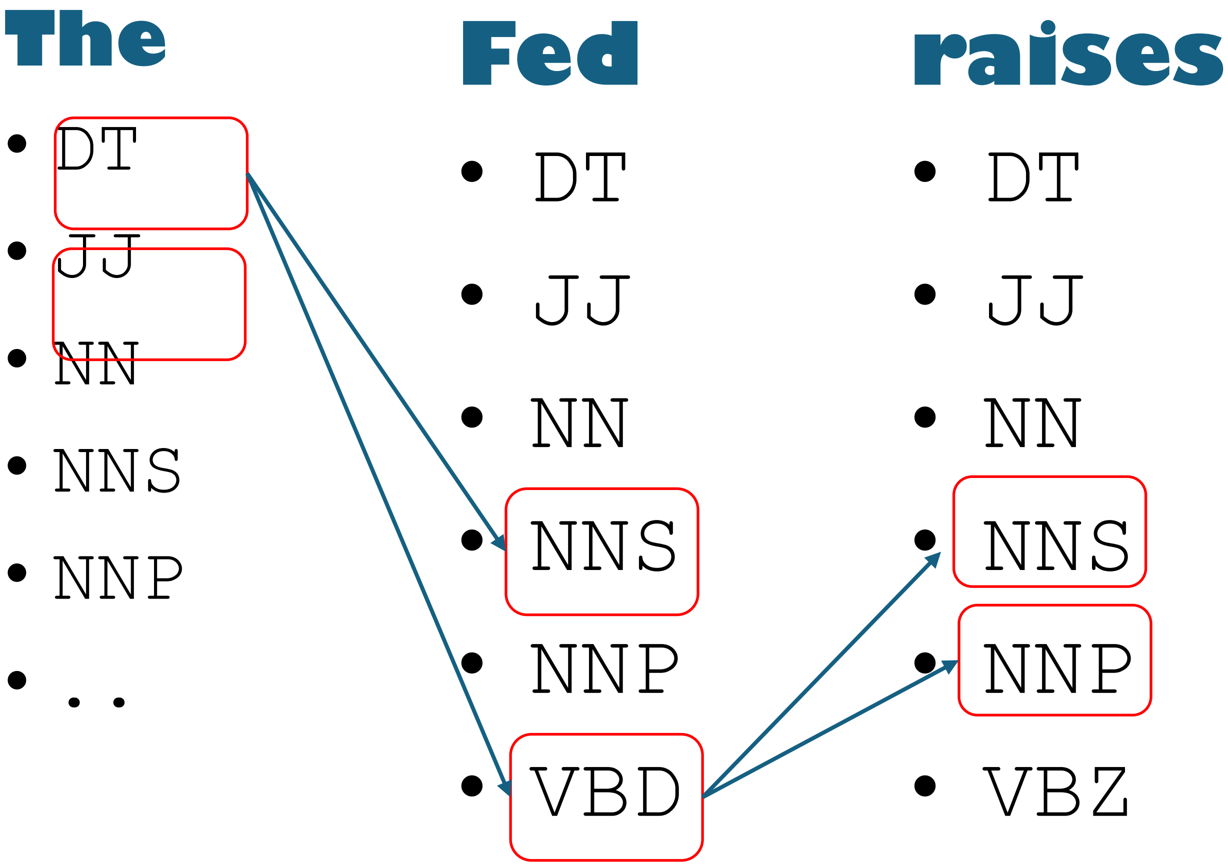
- **Beam search**: mid-point between exact and greedy

- Keep a priority queue of size k, *known as **the beam***
- At each level only explore k next states

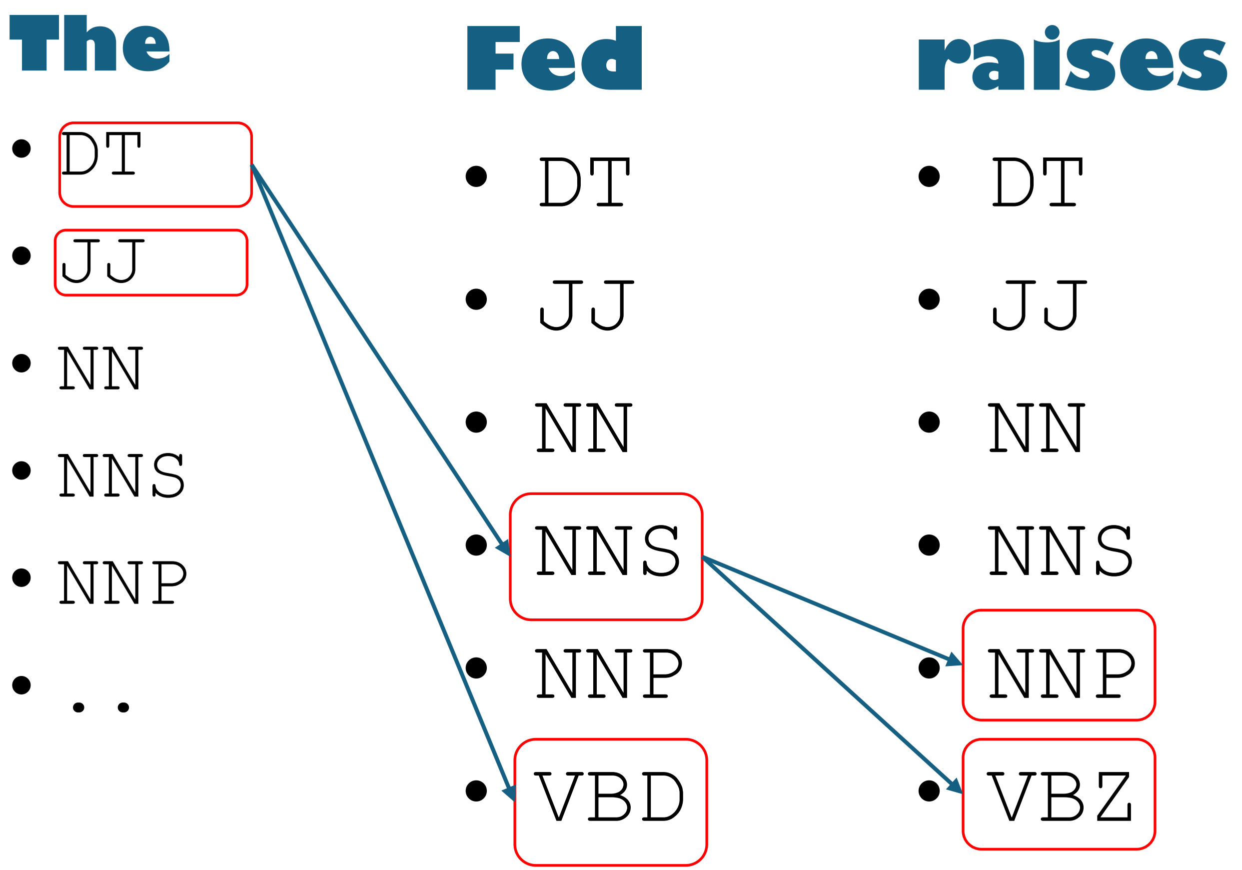
- *If k = infinity:* this is just BFS
- *Otherwise:* greedy search over the top k states

- ***Very popular!***

Beam Search


The
DT
JJ
NN
NNS
NNP
...
Fed
DT
JJ
NN
NNS
NNP
VBD
raises
DT
JJ
NN
NNS
NNP
VBZ

Beam Search (k=2)

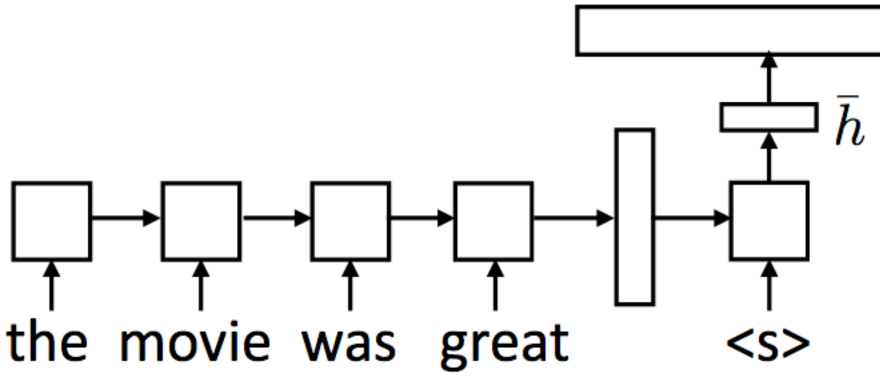


Encoder-Decoder Architecture

- Generate the next word, conditioned on previous words as well as hidden state

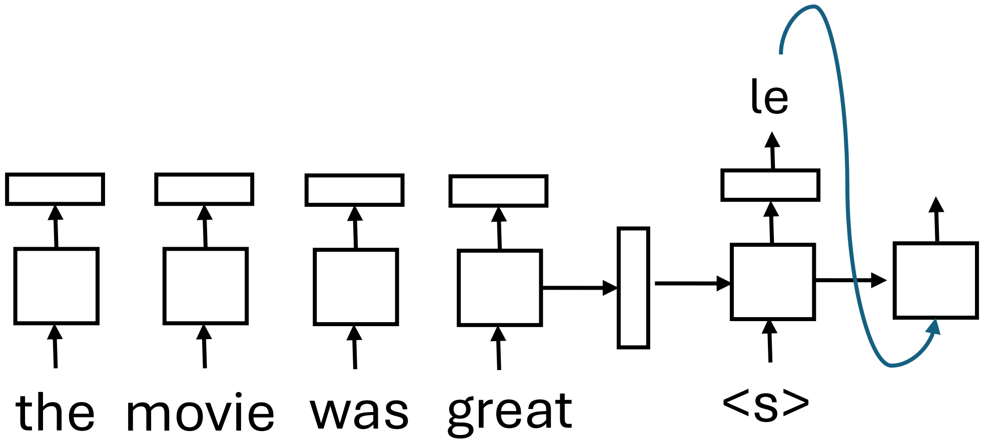
$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h})$$

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1})$$



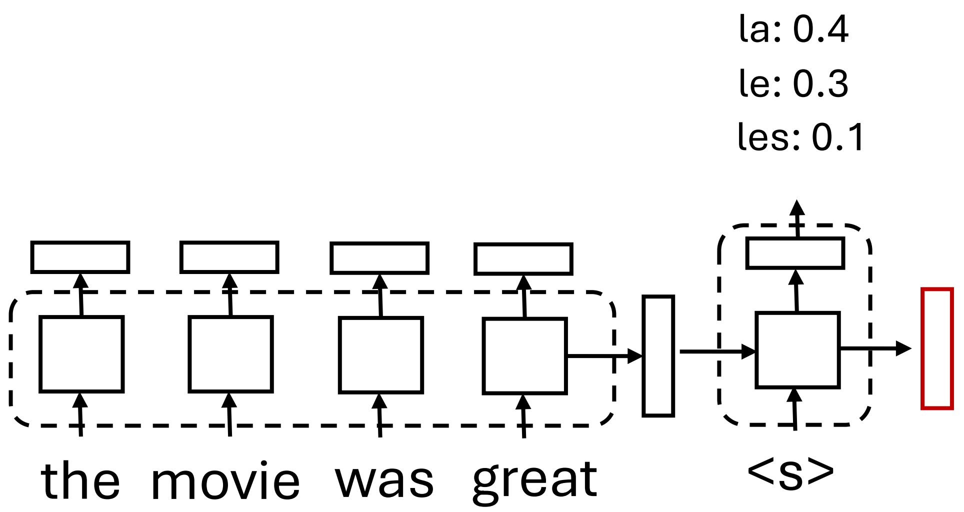
Inference

- At inference time, the predicted word can be used as input to the next word prediction



Inference with Beam Search

- Essentially a structure problem – can include explicit inference using beam search



Inference with Beam Search

The decoder state and token history maintained in the beam