TTIC 31210:
Advanced Natural Language Processing

Kevin Gimpel

Spring 2019

Lecture 8:
Structured Prediction 2

Roadmap

- intro (1 lecture)
- deep learning for NLP (5 lectures)
- **structured prediction (4 lectures)**
  - **introducing/formalizing structured prediction, categories of structures**
  - **inference: dynamic programming, greedy algorithms, beam search**
  - **inference with non-local features**
  - **learning in structured prediction**
- generative models, latent variables, unsupervised learning, variational autoencoders (2 lectures)
- Bayesian methods in NLP (2 lectures)
- Bayesian nonparametrics in NLP (2 lectures)
- review & other topics (1 lecture)

Assignments

- Assignment 2 due in one week
- for the report, please use either pdf format or a Jupyter notebook (no plain text)

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\text{classify}(\boldsymbol{x}, \boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{y}} \ \text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

learning: choose $\boldsymbol{\theta}$

Working definition of structured prediction:

size of output space is exponential in size of input

or is unbounded (e.g., machine translation)

(we can't just enumerate all possible outputs)

What is Structured Prediction?

- when we use a structured scoring function or structured loss function
- we may be predicting a structure, but we might not necessarily be building a "structured predictor"
- we will use the terms "unstructured predictor" or "local predictor" in such cases

Structured Prediction

- a structured score/loss function does not decompose across "minimal parts" of output

- to apply this definition we defined "parts" and "minimal parts"

Sequence Labeling

(e.g., Part-of-Speech Tagging)

determiner	verb (past)	prep.	proper noun	proper noun	poss.	adj.	noun
Some	questioned	if	Tim	Cook	’s	first	product

modal	verb	det.	adjective	noun	prep.	proper noun	punc.
would	be	a	breakaway	hit	for	Apple	.

- parts:

determiner	verb (past)	prep.	proper noun	proper noun	poss.	adj.	noun
Some	questioned	if	Tim	Cook	’s	first	product

modal	verb	det.	adjective	noun	prep.	proper noun	punc.
would	be	a	breakaway	hit	for	Apple	.

- minimal parts:

determiner	verb (past)	prep.	proper noun	proper noun	poss.	adj.	noun
Some	questioned	if	Tim	Cook	’s	first	product

modal	verb	det.	adjective	noun	prep.	proper noun	punc.
would	be	a	breakaway	hit	for	Apple	.

- parts:
  - each “part” is a subcomponent of entire input/output pair
  - “parts function” = decomposition of input/output pair into a set of parts
  - parts functions defined for score/loss function, rather than for task (many parts functions possible for a task)
  - parts may overlap

- minimal parts:
  - smallest possible parts for the task
  - minimal parts function defined for task (structured output space), not for structured score/loss function
  - minimal parts are non-overlapping

Parts and Score Functions

– given a "parts" function

$$\text{parts}(\boldsymbol{x}, \boldsymbol{y})$$

– our score function is then defined:

$$\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \sum_{\langle \boldsymbol{x}_r, \boldsymbol{y}_r \rangle \in \text{parts}(\boldsymbol{x}, \boldsymbol{y})} \text{score}_{\text{part}}(\boldsymbol{x}_r, \boldsymbol{y}_r, \boldsymbol{\theta})$$

– score function decomposes additively across parts

– each part is a subcomponent of input/output pair

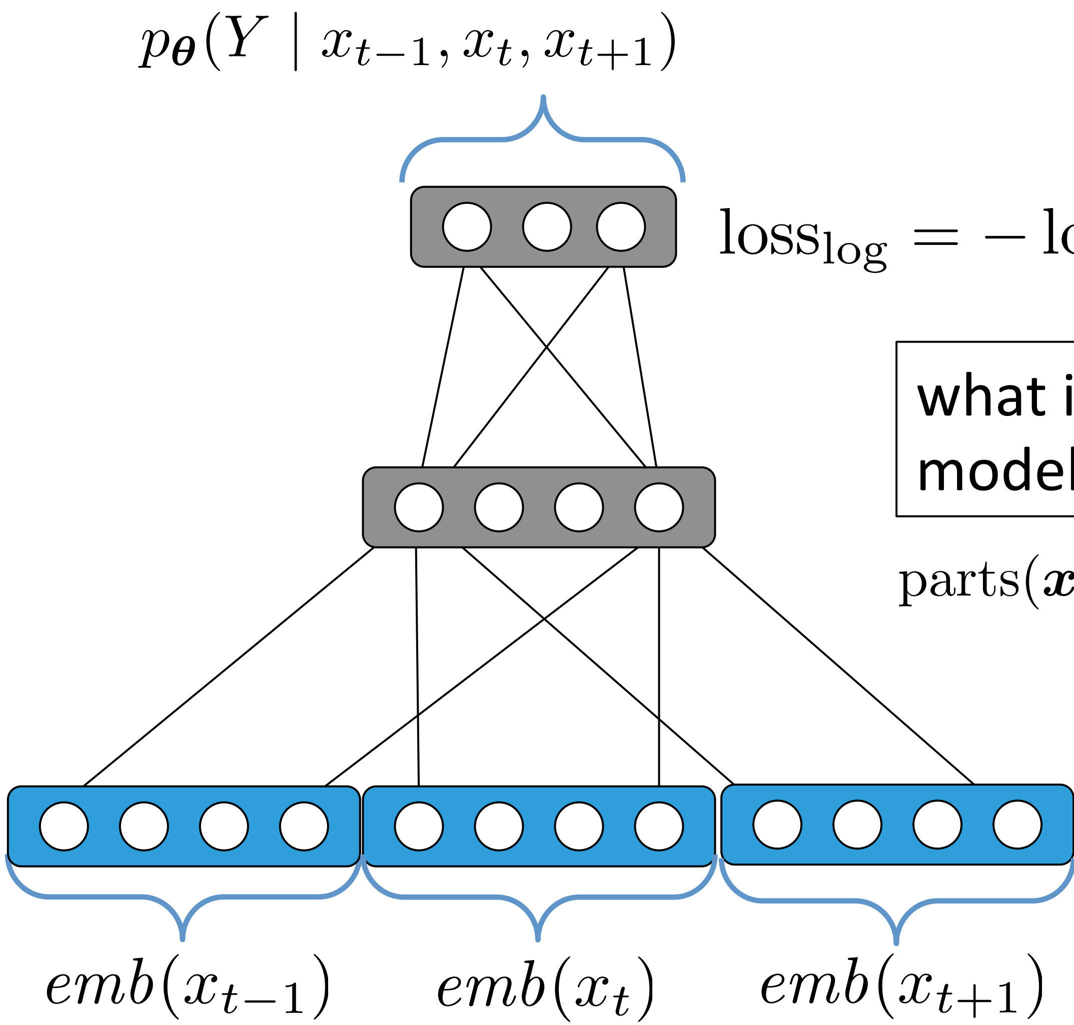
Structured Prediction Tasks

task	output structure	minimal parts	
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set	$\mathrm{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$

Structured Prediction Tasks

task	output structure	minimal parts	
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set	$\mathrm{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length *T* as input sequence; each label is one of *N* possibilities	set containing labels at positions in output sequence	$\mathrm{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$

Feed-Forward POS Tagger



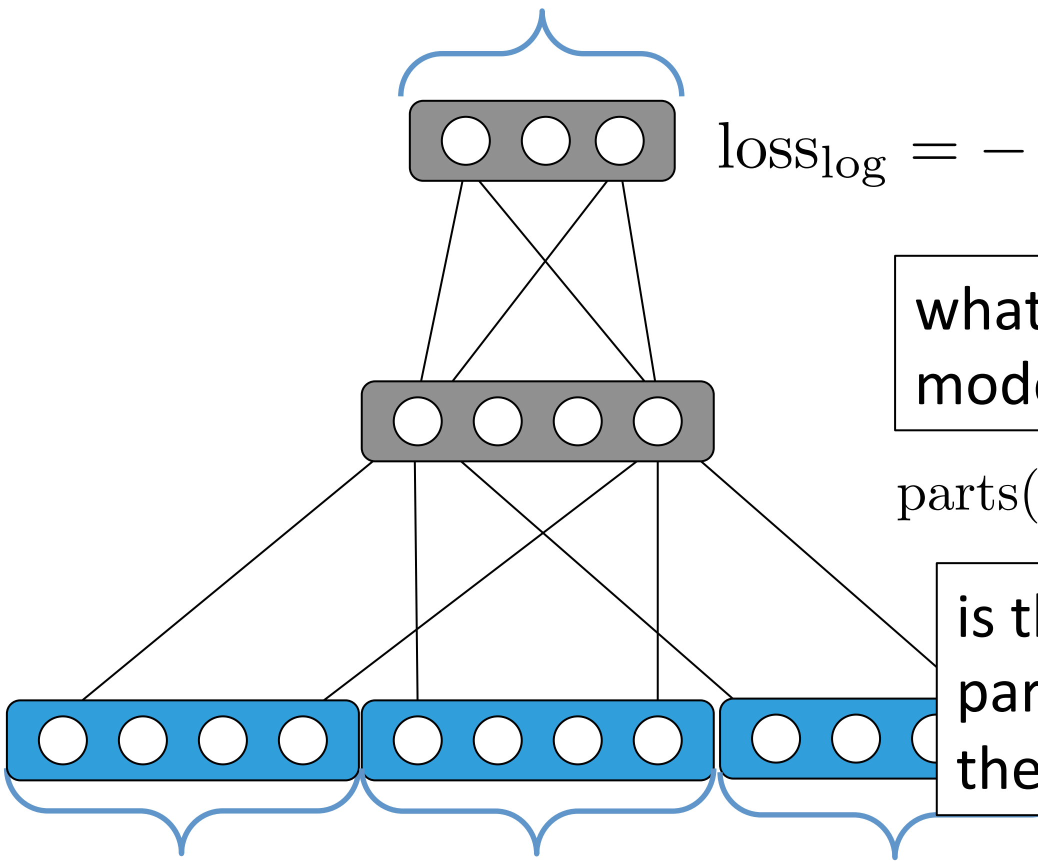
$$\text{loss}_{\text{log}} = -\log p_{\boldsymbol{\theta}}(Y = y_t \mid x_{t-1}, x_t, x_{t+1})$$

what is the parts function for this model & loss?

$$\text{parts}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle \boldsymbol{x}_{t-1:t+1}, y_t \rangle : 1 \leq t \leq T\}$$

Feed-Forward POS Tagger

$$p_{\boldsymbol{\theta}}(Y \mid x_{t-1}, x_t, x_{t+1})$$



$$\text{loss}_{\text{log}} = -\log p_{\boldsymbol{\theta}}(Y = y_t \mid x_{t-1}, x_t, x_{t+1})$$

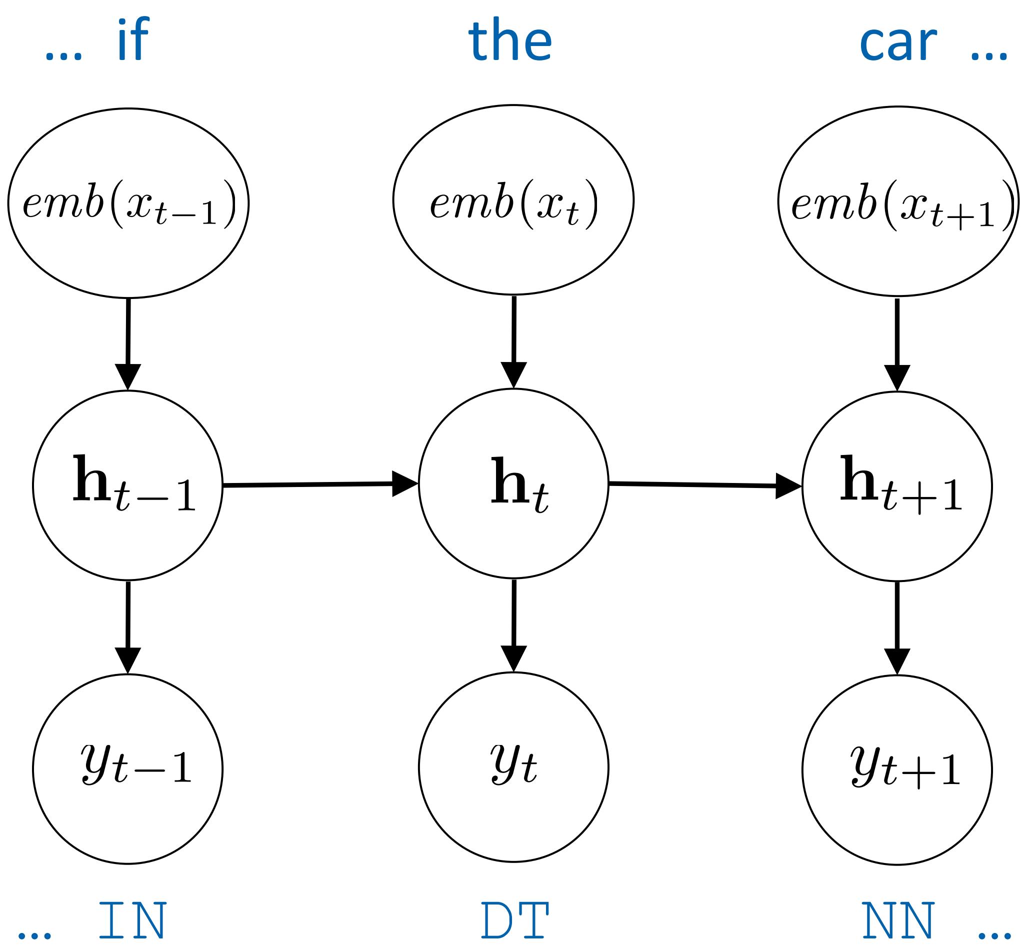
what is the parts function for this model & loss?

$$\text{parts}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle \boldsymbol{x}_{t-1:t+1}, y_t \rangle : 1 \leq t \leq T\}$$

is this a structured predictor? no. parts functions decomposes like the minimal parts (in terms of ***y***)

$emb(x_{t-1})$ $emb(x_t)$ $emb(x_{t+1})$

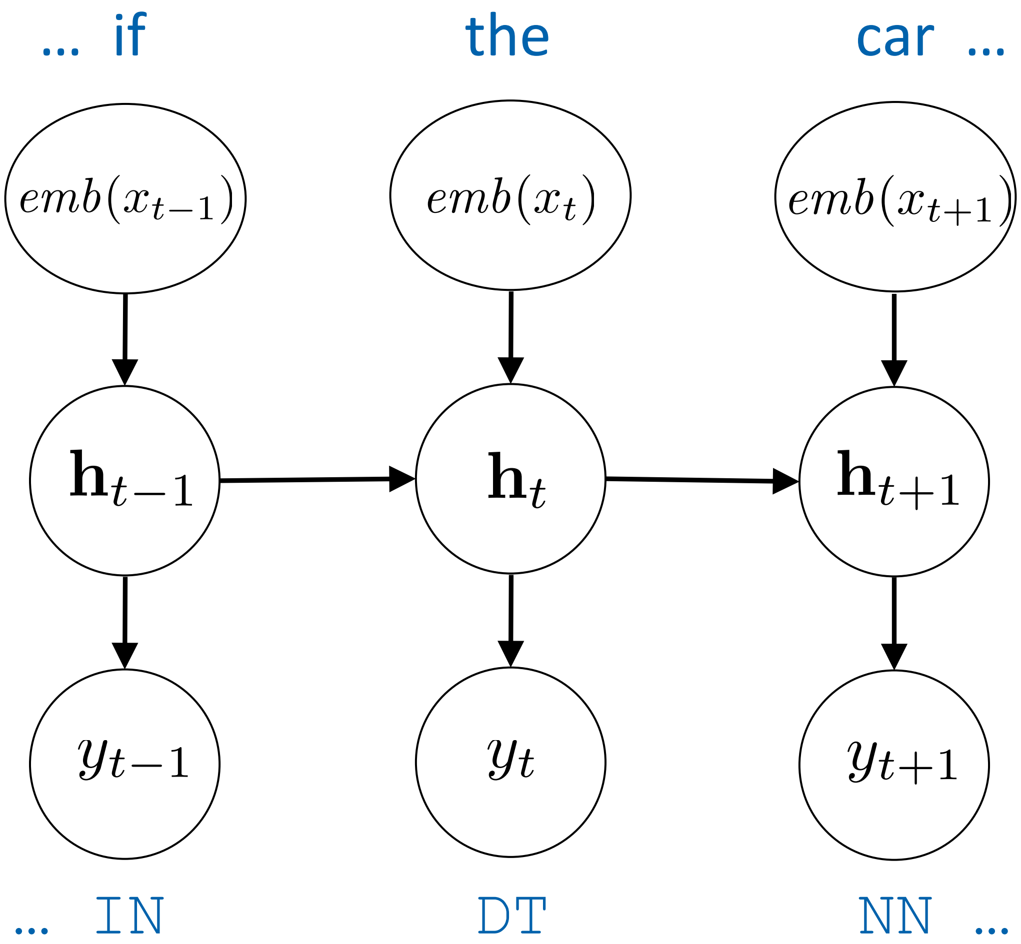
Forward RNN for Part-of-Speech Tagging



hidden vector used to compute probability distribution over tags at each position:

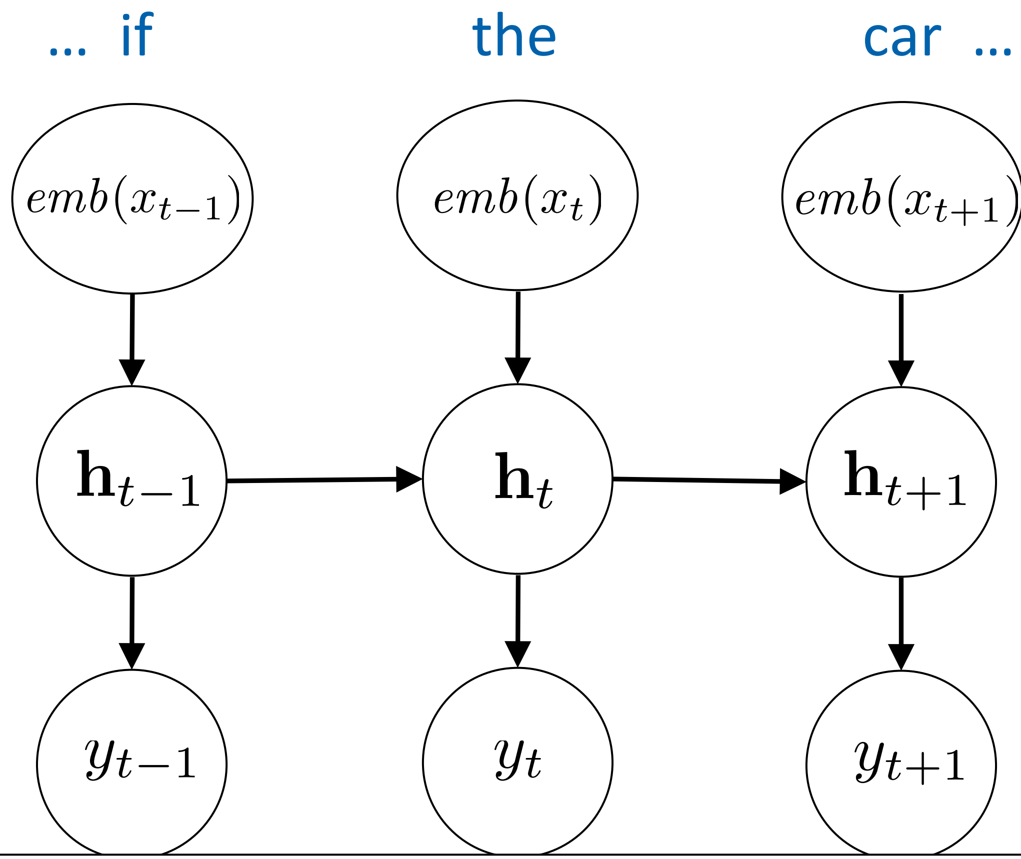
$$p_{\boldsymbol{\theta}}(Y_t \mid \boldsymbol{x}_{1:t})$$

Forward RNN for Part-of-Speech Tagging



loss: $-\sum_t \log p_{\boldsymbol{\theta}}(Y_t = y_t \mid \boldsymbol{x}_{1:t})$

Forward RNN for Part-of-Speech Tagging

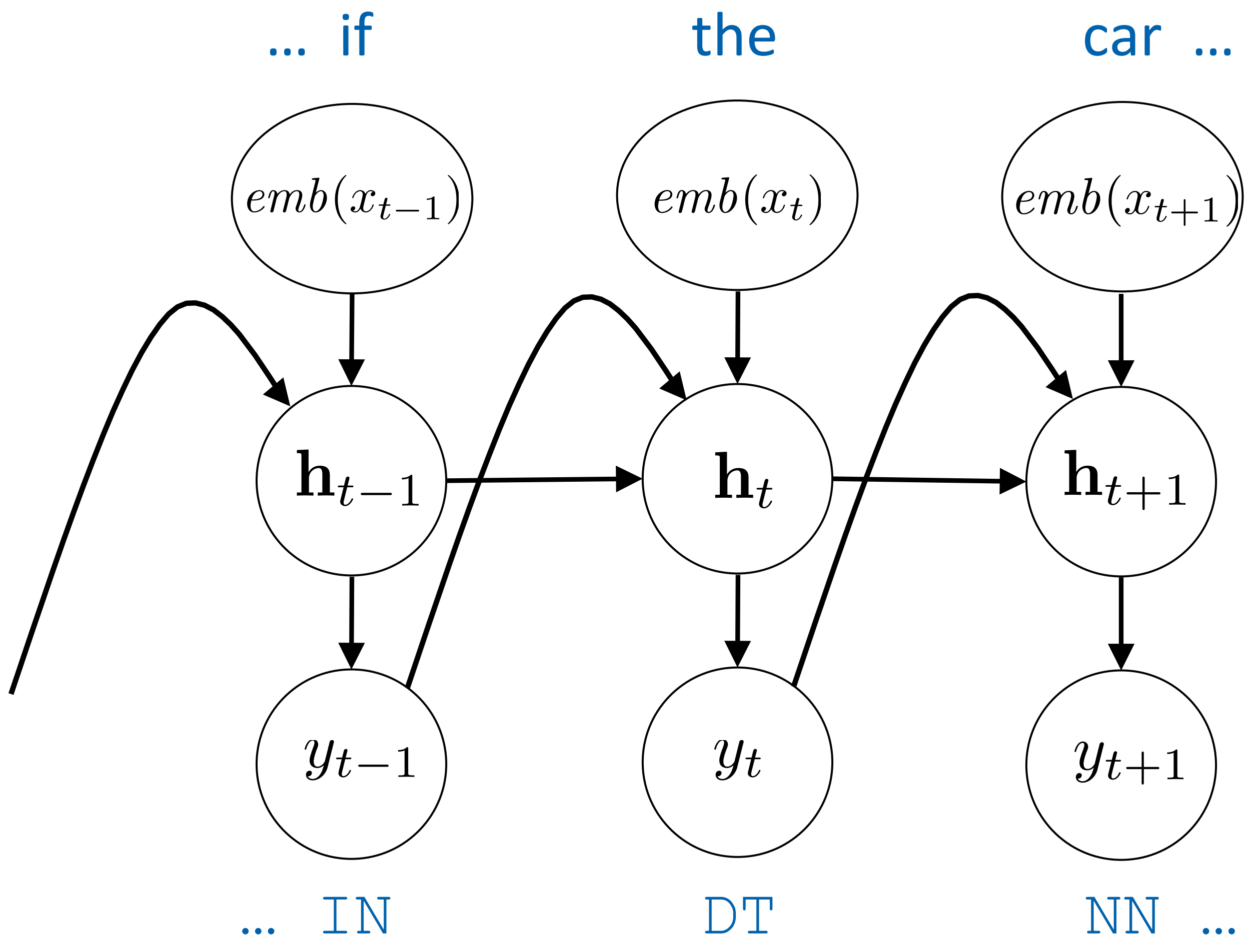


what is the parts function for this model & loss?

$$\text{parts}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle \boldsymbol{x}_{1:t}, y_t \rangle : 1 \leq t \leq T\}$$

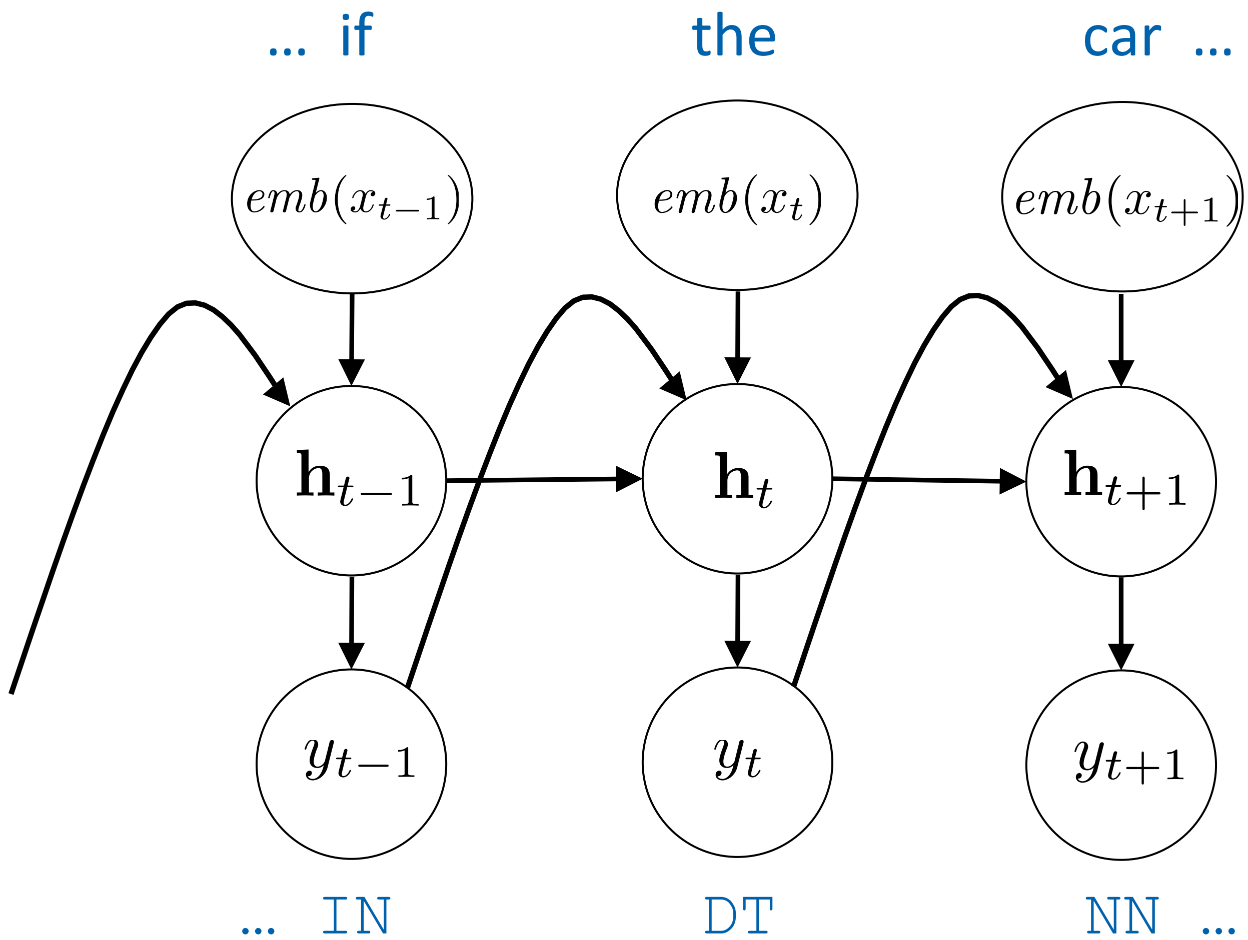
is this a structured predictor? no

Forward RNN for Part-of-Speech Tagging with Previous Label



this model uses the previous *y* to compute a hidden vector

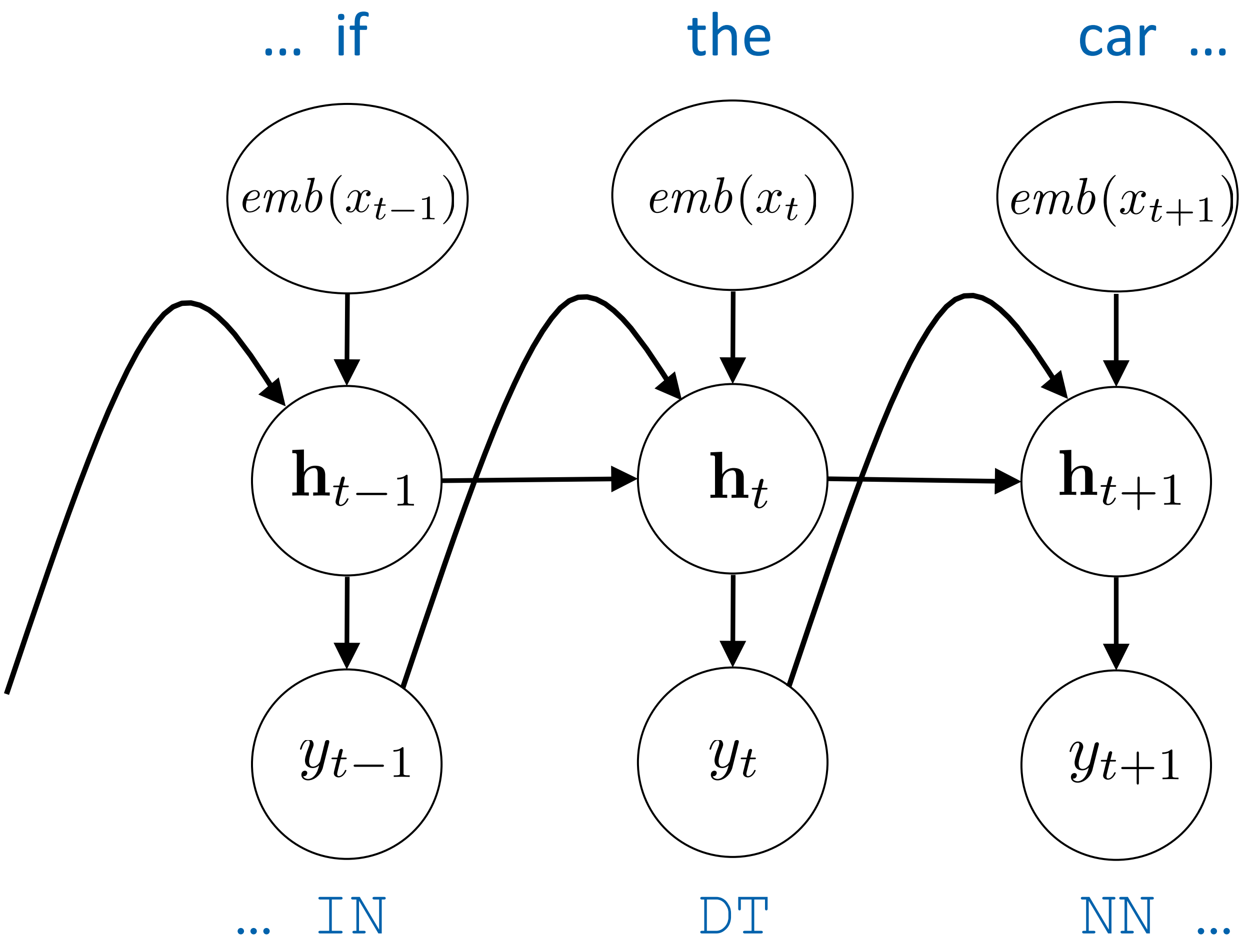
Forward RNN for Part-of-Speech Tagging with Previous Label



hidden vector used to compute probability distribution over tags at each position:

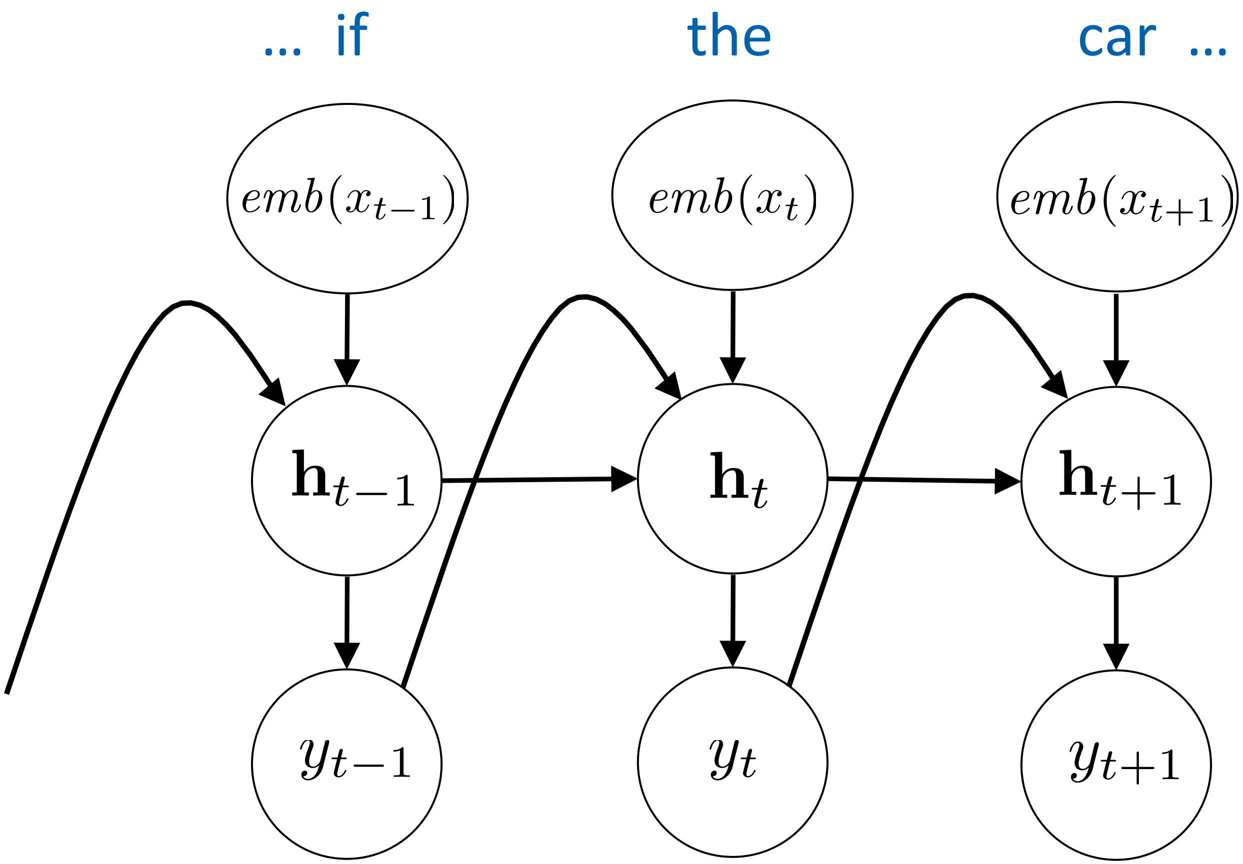
$$p_{\boldsymbol{\theta}}(Y_t \mid \boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t-1})$$

Forward RNN for Part-of-Speech Tagging with Previous Label



loss: $-\sum_t \log p_{\boldsymbol{\theta}}(Y_t = y_t \mid \boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t-1})$

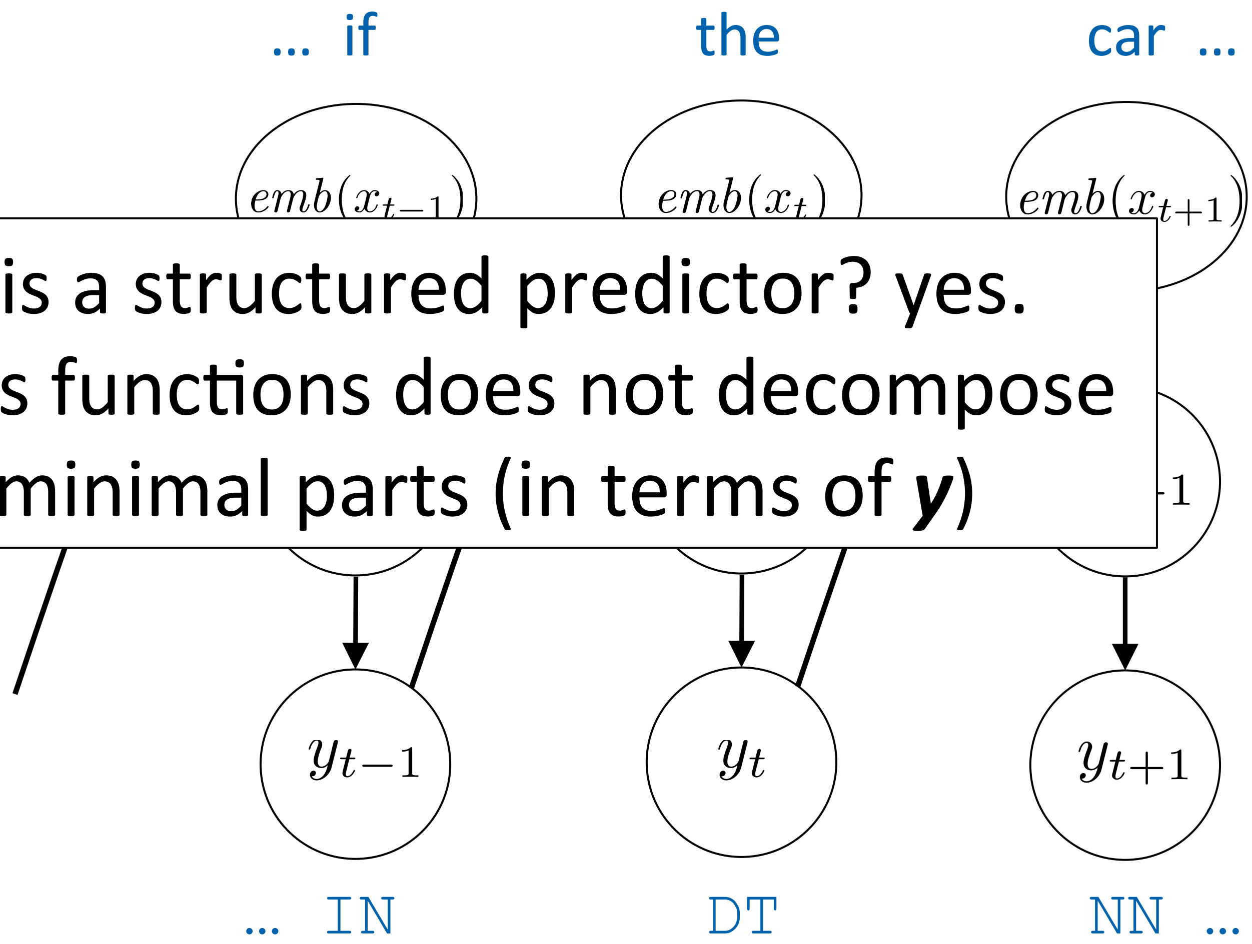
Forward RNN for Part-of-Speech Tagging with Previous Label



what is the parts function for this model & loss?

$$\mathrm{parts}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle \boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t} \rangle : 1 \leq t \leq T\}$$

Forward RNN for Part-of-Speech Tagging with Previous Label



is this a structured predictor? yes. parts functions does not decompose like minimal parts (in terms of ***y***)

$$\text{parts}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle \boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t} \rangle : 1 \leq t \leq T\}$$

Structured Prediction Tasks

task	output structure	minimal parts	
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set	$\text{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length *T* as input sequence; each label is one of *N* possibilities	set containing labels at positions in output sequence	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$

Structured Prediction Tasks

task	output structure	minimal parts
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set $\mathrm{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length *T* as input sequence; each label is one of *N* possibilities	set containing labels at positions in output sequence $\mathrm{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$
labeled segmentation		

Named Entity Recognition

Some questioned if Tim Cook's first product would be a breakaway hit for Apple.

PERSON **ORGANIZATION**

Labeled Segmentation as Sequence Labeling

O	O	O	B-PERSON	I-PERSON	O	O	O
Some	questioned	if	Tim	Cook	’s	first	product

O	O	O	O	O	O	B-ORGANIZATION	O
would	be	a	breakaway	hit	for	Apple	.

B = “begin”
I = “inside”
O = “outside”

Structured Prediction Tasks

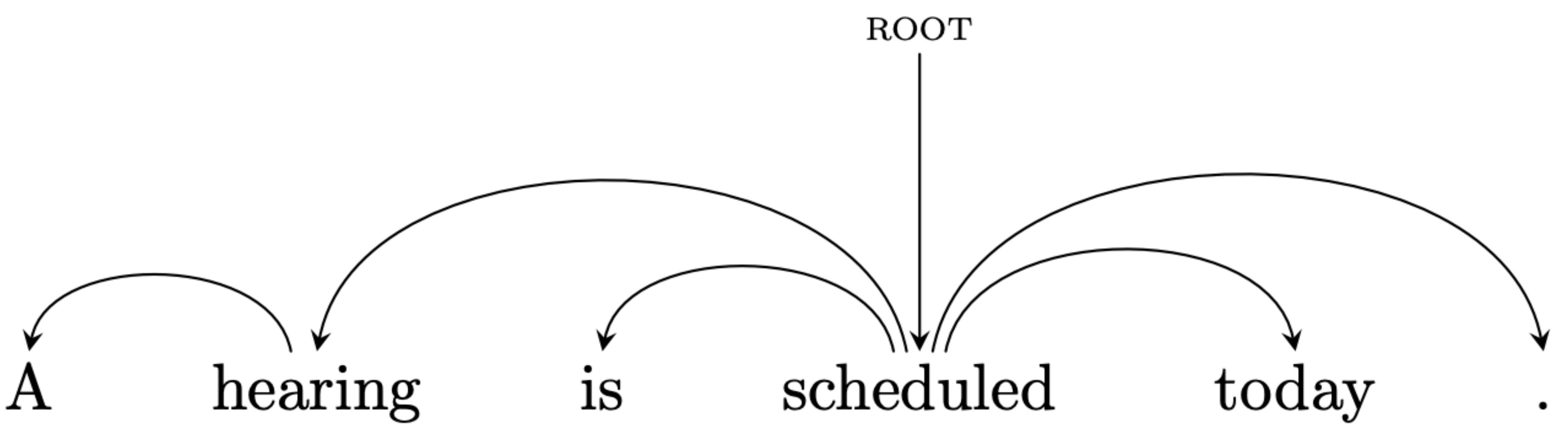
task	output structure	minimal parts	
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set	$\text{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length *T* as input sequence; each label is one of *N* possibilities	set containing labels at positions in output sequence	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$
labeled segmentation			

Structured Prediction Tasks

task	output structure	minimal parts
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set $\mathrm{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length *T* as input sequence; each label is one of *N* possibilities	set containing labels at positions in output sequence $\mathrm{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$
labeled segmentation		
unlabeled dependency parsing		

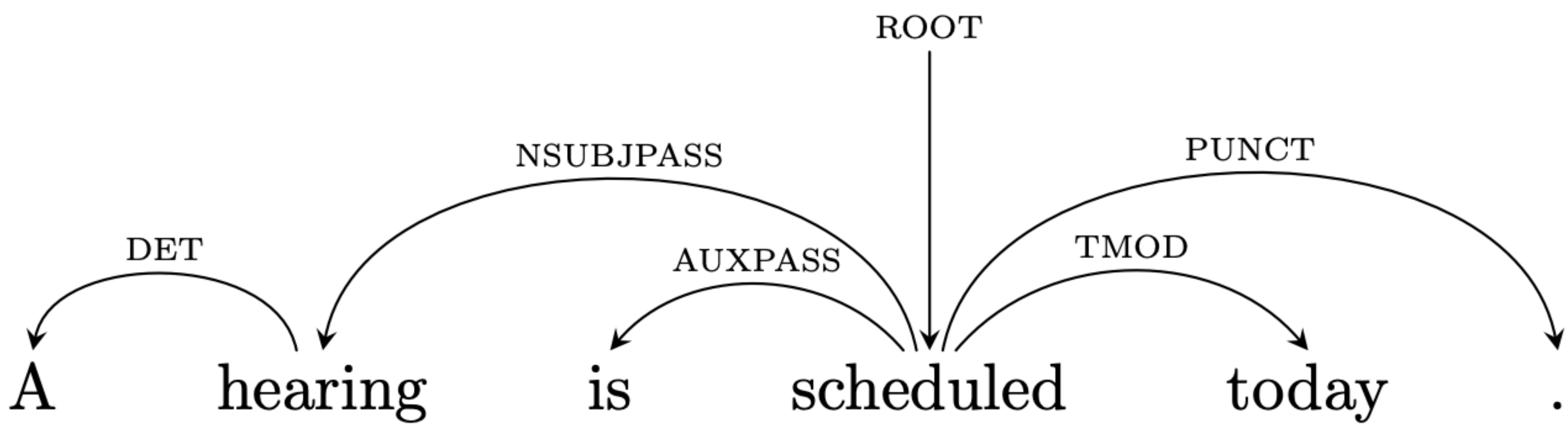
Dependency Trees

- a **dependency** is a relation between a word (a **head** or **parent**) and its dependent (its **modifier** or **child**)
- a dependency tree for a sentence contains a dependency for each word in the sentence
- drawn as a directed tree with parents pointing to children
- one word is the root of the tree



Labeled Dependency Trees

- more common to use labeled dependency trees where each dependency has an associated label:



DET = determiner

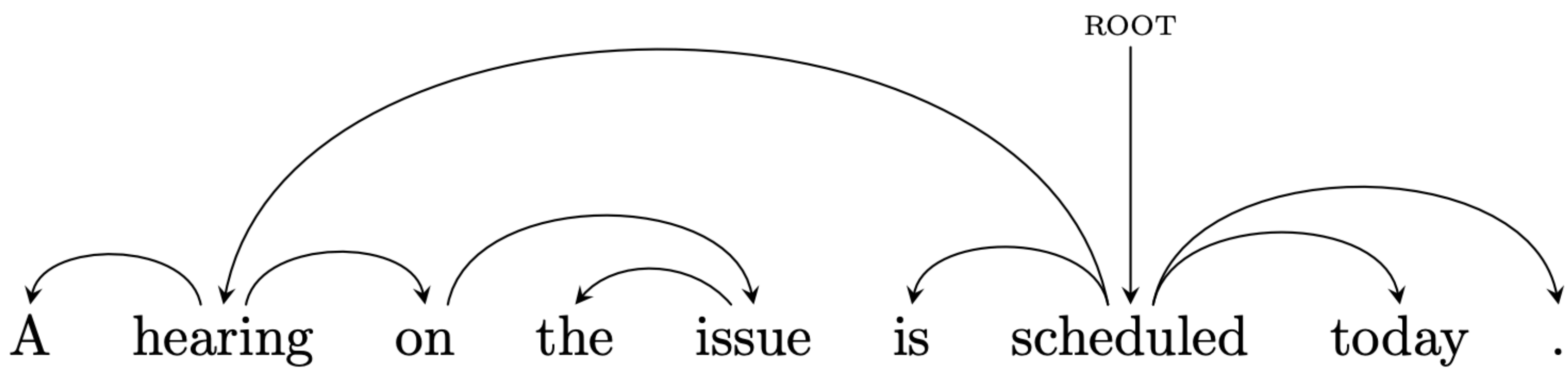
NSUBJPASS = nominal subject in passive construction

AUXPASS = auxiliary verb in passive construction

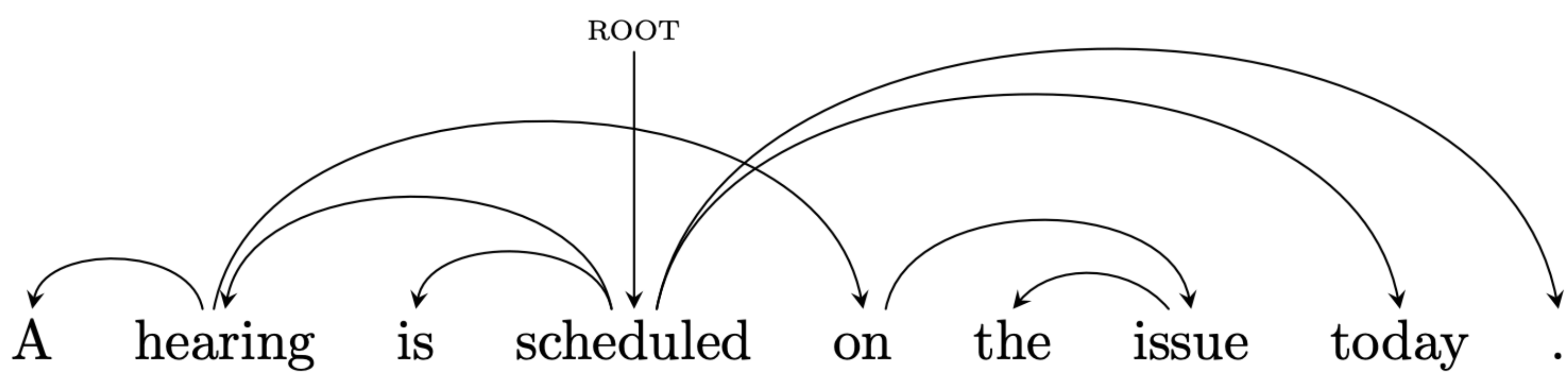
TMOD = temporal modifier

PUNCT = punctuation

- a longer sentence:



- rearranging the words a bit, we now have a **non-projective** dependency tree (i.e., tree with crossing dependencies):



Dependency Parsing

- dependency parsing is the task of predicting a dependency tree for a sentence
- one of the most widely-studied structured prediction problems in NLP
- used for several downstream NLP tasks

Applications of Dependency Parsing

- widely used for NLP tasks because:
  - faster than constituent parsing
  - captures more semantic information
- text classification (features on dependencies)
- syntax-based machine translation
- relation extraction
  - e.g., extract relation between Sam Smith and AITech:

  Sam Smith was named new CEO of AITech.
  - use dependency path between *Sam Smith* and *AITech:*
    - Smith → named, named ← CEO, CEO ← of, of ← AITech

- minimal parts for (unlabeled) dependency parsing:

$$\mathrm{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$$
$$\text{where each } y_t \in \{0, 1, \ldots, T\}$$

- each y_t holds the index in the sentence $\boldsymbol{x}$ of the parent of word x_t
- we use 0 for the ROOT attachment

Unstructured Predictors for Dependency Parsing

- how might you design an unstructured predictor for dependency parsing?
- build a predictor that predicts the index of the head for a word
- can use full sentential context, just can't score multiple dependencies in any single scoring term
- fast, simple, works ok, but doesn't guarantee a tree structure (may have cycles, etc.)

Structured Prediction Tasks

task	output structure	minimal parts	
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set	$\text{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length *T* as input sequence; each label is one of *N* possibilities	set containing labels at positions in output sequence	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$
labeled segmentation			
unlabeled dependency parsing	tree over the words in the input sentence; each word has exactly one parent	set containing indices of parent words for each word in sentence	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{0, 1, \ldots, T\}$

Structured Prediction Tasks

task	output structure	minimal parts	
multi-label classification	set of N labels, each of which can be true or false	set containing individual labels in label set	$\text{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length T as input sequence; each label is one of N possibilities	set containing labels at positions in output sequence	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$
labeled segmentation			
unlabeled dependency parsing	tree over the words in the input sentence; each word has exactly one parent	set containing indices of parent words for each word in sentence	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{0, 1, \ldots, T\}$
conditional generation	sentence (or a paragraph, document, etc.)		

Conditional Generation

- for machine translation and other "conditional generation" tasks, input is a sequence and output is a sequence
- minimal parts for these tasks:

$$\mathrm{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_{|\boldsymbol{y}|}\}$$

$$\text{where each } y_t \in \mathcal{V}$$

- each y_t is a word from the output vocabulary

Unstructured Predictors for Machine Translation

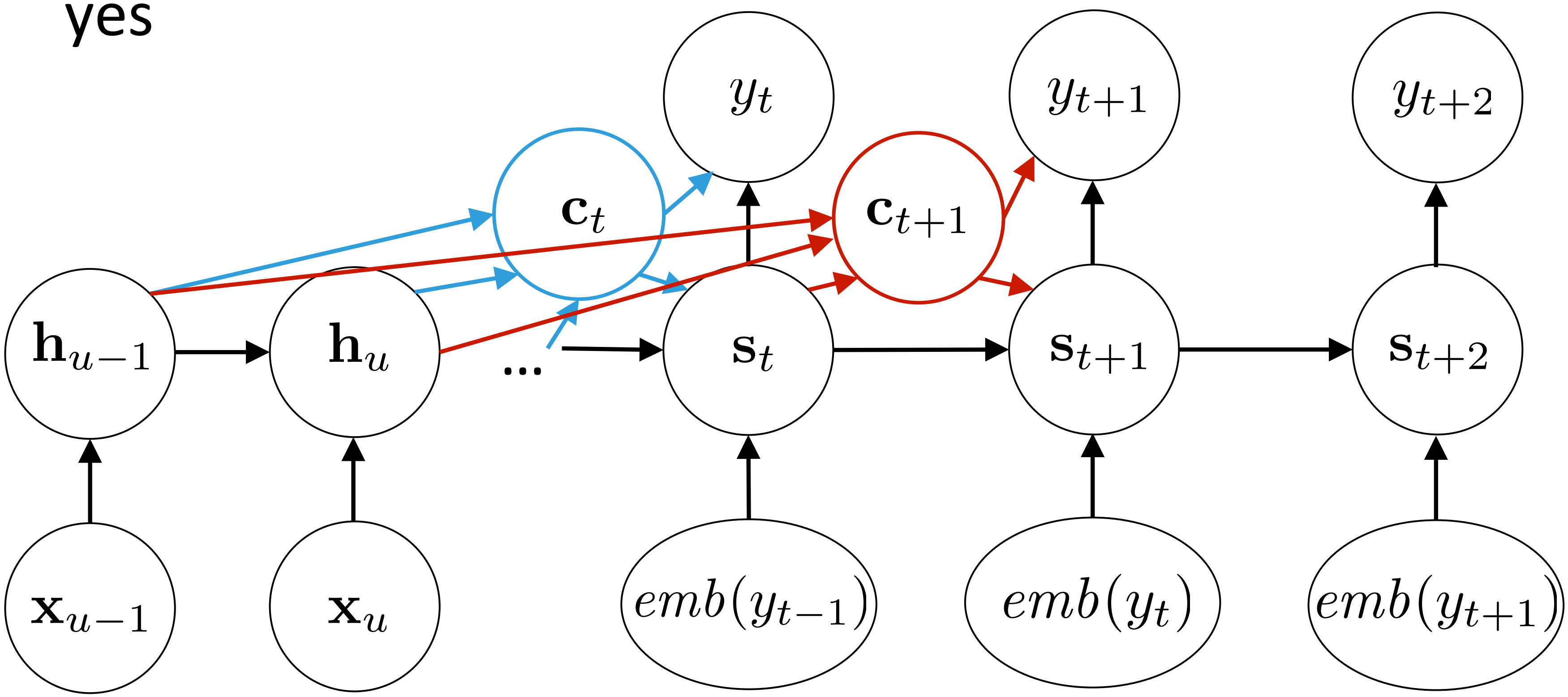
- how might you design an unstructured predictor for machine translation?

- assume a max length of the translation, pad to that length, build predictors that predict the word in position *t* in the translation

- this probably won't work very well, but if we have this model we could maximally parallelize translation across machines

Sequence-to-Sequence Models with Attention

- most common approach for conditional generation
- previously-predicted output symbol used as input for making next prediction ("auto-regressive")
- is this a structured predictor?
- yes



Structured Prediction Tasks

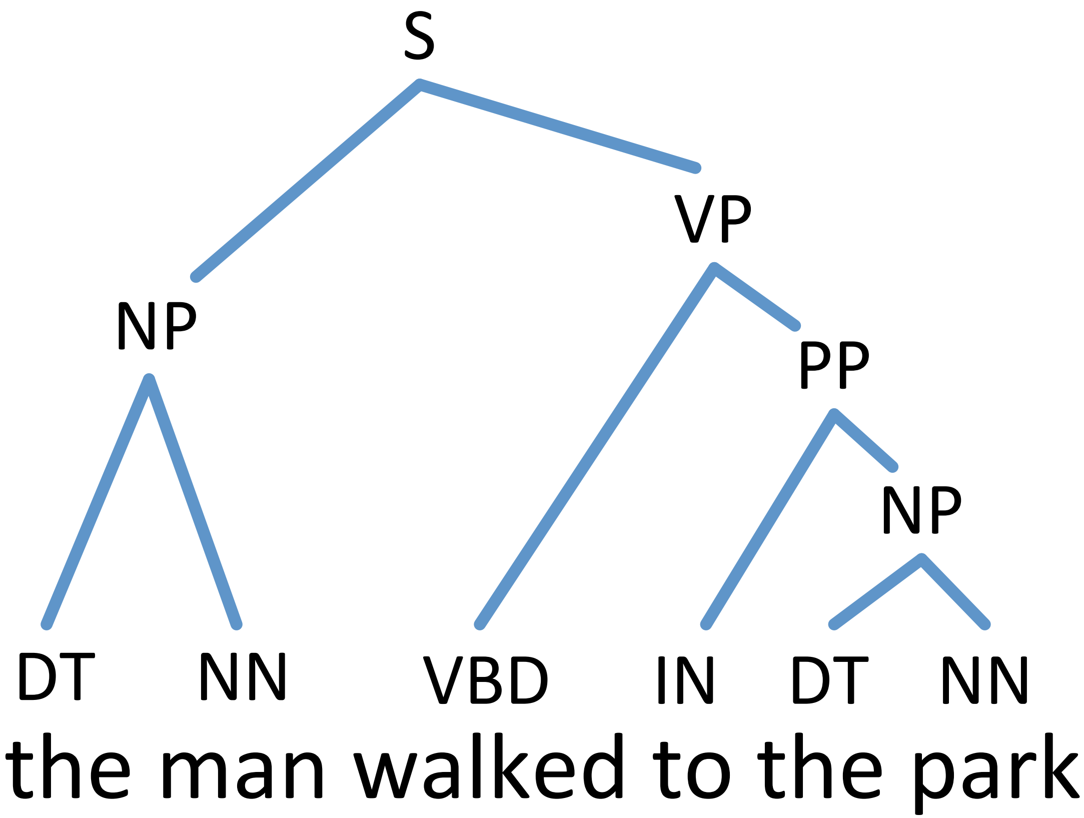
task	output structure	minimal parts	
multi-label classification	set of *N* labels, each of which can be true or false	set containing individual labels in label set	$\text{mp}(\boldsymbol{y}) = \{y_1, ..., y_N\}$ where each $y_i \in \{0, 1\}$
sequence labeling	label sequence with same length *T* as input sequence; each label is one of *N* possibilities	set containing labels at positions in output sequence	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{1, \ldots, N\}$
labeled segmentation			
unlabeled dependency parsing	tree over the words in the input sentence; each word has exactly one parent	set containing indices of parent words for each word in sentence	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_T\}$ where each $y_t \in \{0, 1, \ldots, T\}$
conditional generation	sentence (or a paragraph, document, etc.)	set containing each word in the output	$\text{mp}(\boldsymbol{y}) = \{y_1, \ldots, y_{\lvert\boldsymbol{y}\rvert}\}$ where each $y_t \in \mathcal{V}$

Other Tasks?

- some tasks do not permit an easy definition of minimal parts

Constituency Parsing

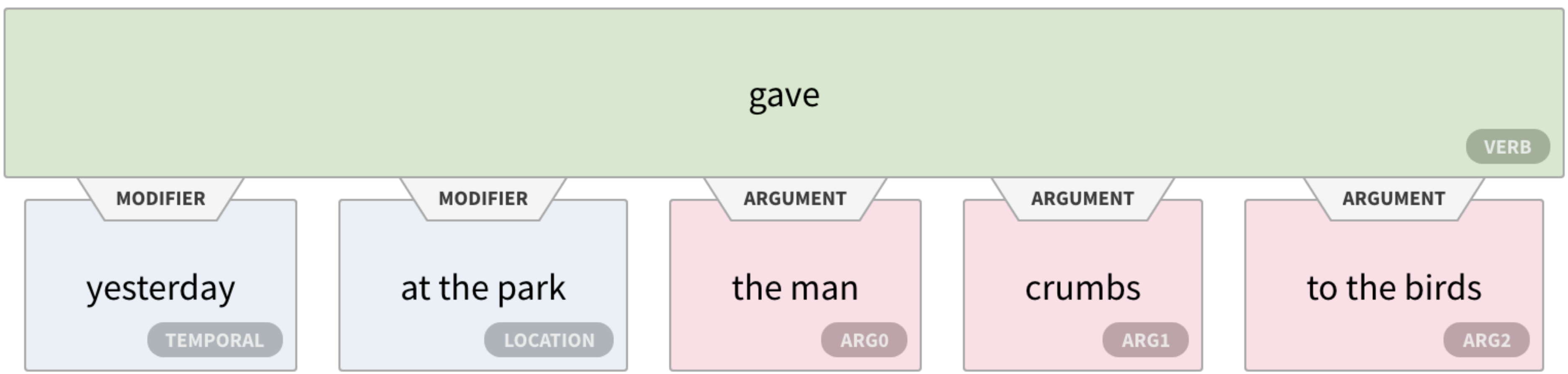
(S (NP the man) (VP walked (PP to (NP the park))))



Key:
S = sentence
NP = noun phrase
VP = verb phrase
PP = prepositional phrase
DT = determiner
NN = noun
VBD = verb (past tense)
IN = preposition

Semantic Role Labeling

yesterday at the park the man gave crumbs to the birds



ARG0 = usually *agent*

ARG1 = typically *patient* or *theme*

ARG2 = often *beneficiary*

AllenNLP

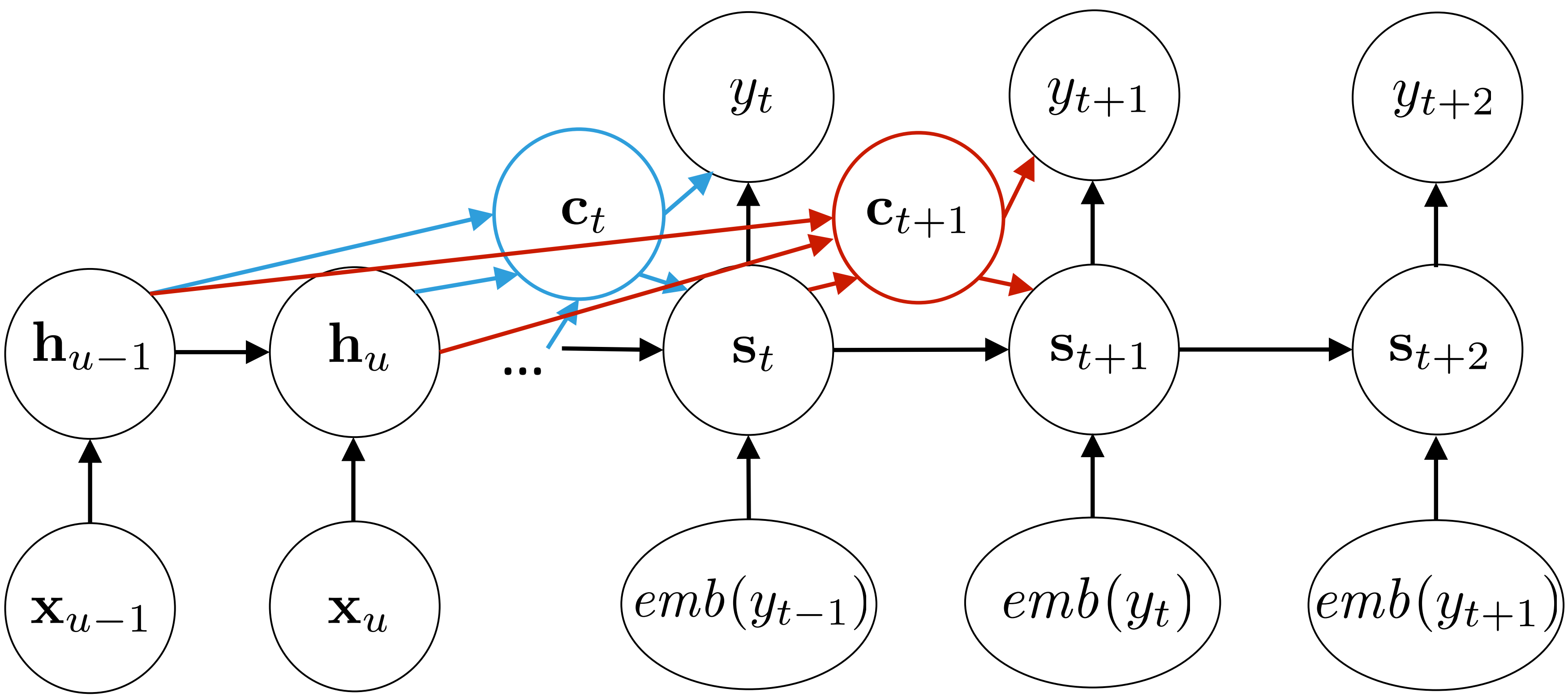
Other Tasks?

- some tasks do not permit an easy definition of minimal parts
  - constituency parsing, semantic role labeling, etc.

- sometimes we can cast these as conditional generation tasks, then inherit the minimal parts definition from conditional generation

Sequence-to-Sequence Models with Attention

- input and output sequences can have different lengths
- we can frame many output structures as sequences



Formulating Constituency Parsing as Conditional Generation

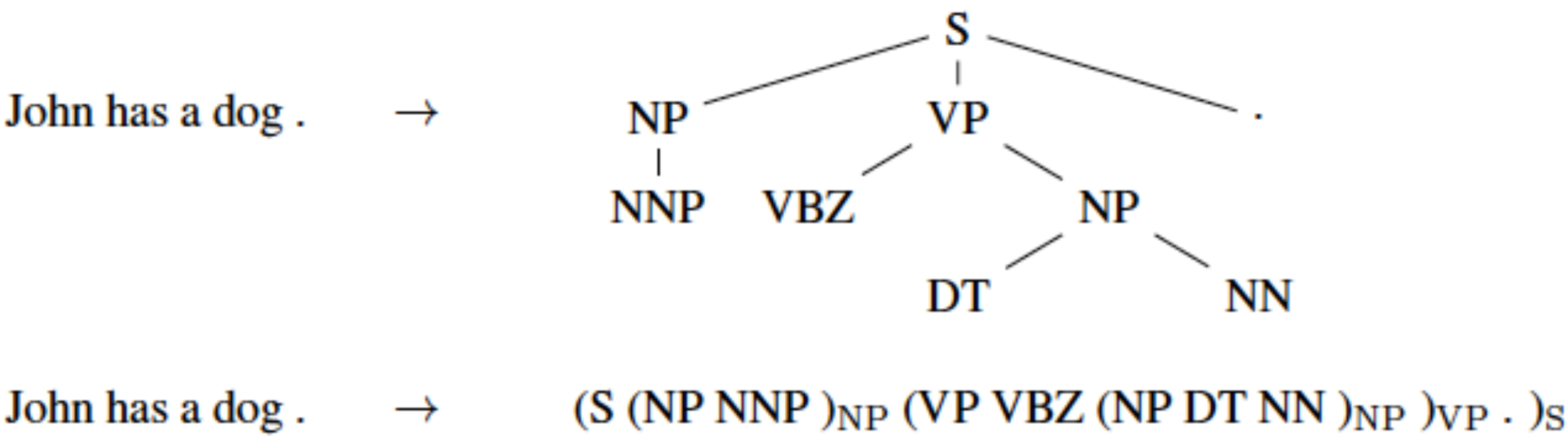


Figure 2: Example parsing task and its linearization.

{Vinyals, Kaiser} et al. (2015): *Grammar as a Foreign Language*

Formulating Constituency Parsing as Conditional Generation

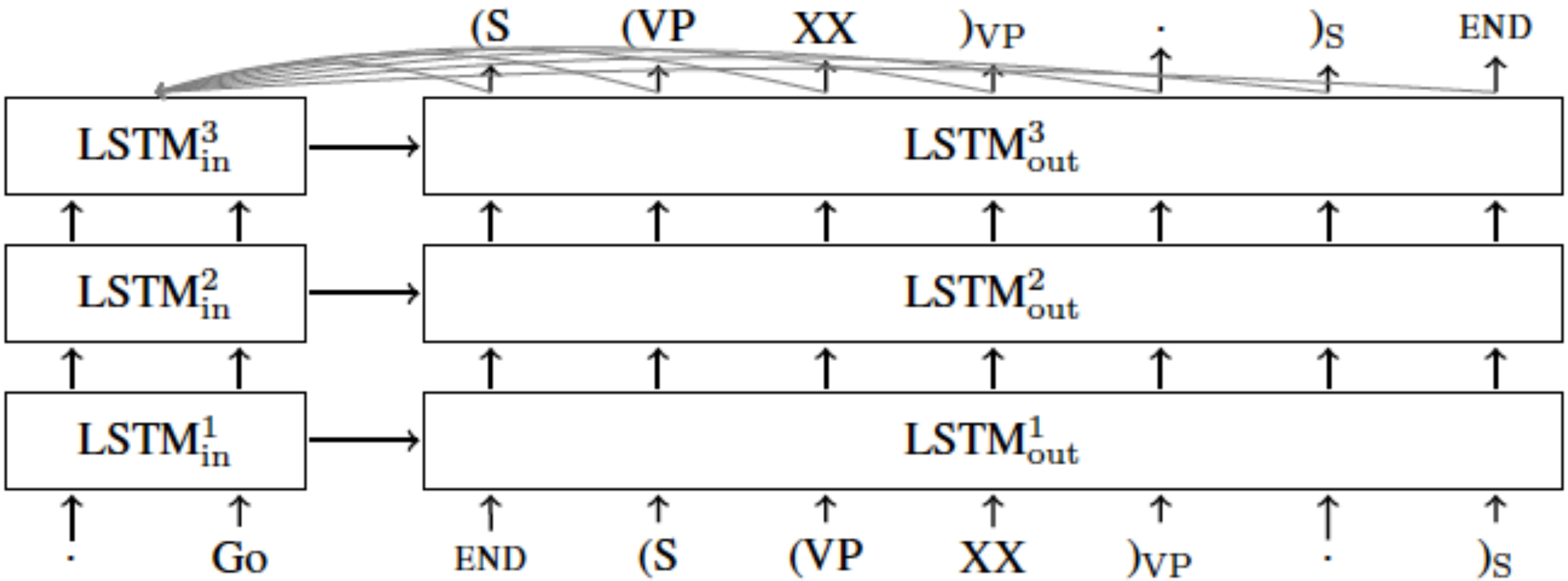


Figure 1: A schematic outline of a run of our LSTM+A model on the sentence "Go.". See text for details.

{Vinyals, Kaiser} et al. (2015): *Grammar as a Foreign Language*

- others have done this for dependency parsing, semantic role labeling, abstract meaning representation parsing, and many other tasks

- sequence-to-sequence models then become a general purpose modeling framework for many structured prediction tasks

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\text{classify}(\boldsymbol{x}, \boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{y}} \ \text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

learning: choose $\boldsymbol{\theta}$

Inference with Structured Predictors

inference: solve argmax

$$\text{classify}(\boldsymbol{x}, \boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{y}} \ \text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

- how do we efficiently search over the space of all structured outputs?
- this space may have size exponential in the size of the input, or be unbounded
- complexity of inference depends on parts function

Hidden Markov Models

- simple, useful, well-known model for sequence labeling: **Hidden Markov Model (HMM)**
- HMMs are used in NLP, speech processing, computational biology, and other areas

Hidden Markov Models

- *n*-gram language models define a probability distribution over word sequences ***x***

- HMMs define a joint probability distribution over input sequences ***x*** and output sequences ***y***

$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{|\boldsymbol{x}|} p(x_i \mid x_1, ..., x_{i-1}, y_1, ..., y_i) p(y_i \mid x_1, ..., x_{i-1}, y_1, ..., y_{i-1})$$

- conditional independence assumptions ("Markov assumption") are used to factorize this joint distribution into small terms

*for now, we are omitting stopping probabilities for clarity

Random Variables for Sequence Labeling

- let's define random variables for observations:
  - observation variable at time step *t*: X_t
  - its possible values: words in vocabulary $\mathcal{V}$
- and we'll define one "hidden" variable for each observation:
  - hidden variable at time *t*: Y_t
  - its possible values: discrete symbols in some set
  - for now, think of the set of possible POS tags

Conditional Independence Assumptions of HMMs

- two *Y*'s are conditionally independent given the *Y*'s between them:

$$Y_{t-1} \perp Y_{t+1} \mid Y_t$$

- an *X* at position *t* is conditionally independent of other *Y*'s given the *Y* at position *t*:

$$X_t \perp Y_{t-1} \mid Y_t$$

$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{|\boldsymbol{x}|} p(x_i \mid x_1, ..., x_{i-1}, y_1, ..., y_i) p(y_i \mid x_1, ..., x_{i-1}, y_1, ..., y_{i-1})$$

$$\Downarrow$$

$$p_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{|\boldsymbol{x}|} p_{\boldsymbol{\tau}}(y_i \mid y_{i-1}) p_{\boldsymbol{\eta}}(x_i \mid y_i)$$

*for now, we are omitting stopping probabilities for clarity

HMMs

$$p_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{|\boldsymbol{x}|} p_{\boldsymbol{\tau}}(y_i \mid y_{i-1}) p_{\boldsymbol{\eta}}(x_i \mid y_i)$$

conditional independence assumptions → we only have to worry about **local distributions:**

transition parameters: $p_{\boldsymbol{\tau}}(y_i \mid y_{i-1})$

emission parameters: $p_{\boldsymbol{\eta}}(x_i \mid y_i)$

*for now, we are omitting stopping probabilities for clarity

Important: Stopping Probabilities

$$p_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{|\boldsymbol{x}|} p_{\boldsymbol{\tau}}(y_i \mid y_{i-1}) p_{\boldsymbol{\eta}}(x_i \mid y_i)$$

$$p_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = p_{\boldsymbol{\tau}}(</\mathrm{s}> \mid y_{|\boldsymbol{x}|}) \prod_{i=1}^{|\boldsymbol{x}|} p_{\boldsymbol{\tau}}(y_i \mid y_{i-1}) p_{\boldsymbol{\eta}}(x_i \mid y_i)$$

special end-of-sequence label

We also assume: $y_0 = <\mathrm{s}>$

special start-of-sequence label

why does this matter?

Parts Function for an HMM

- for a bigram HMM:

$$\text{parts}_{\text{HMM}}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle x_t, y_t \rangle\}_{t=1}^{T} \cup \{\langle \emptyset, y_{t-1:t} \rangle\}_{t=1}^{T}$$

- each word-label pair forms a part, and each label bigram forms a part

Inference in HMMs

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \operatorname*{argmax}_{\boldsymbol{y}} \; p_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y})$$

$$= \operatorname*{argmax}_{\boldsymbol{y}} \; p_{\boldsymbol{\tau}}(</\text{s}> \mid y_{|\boldsymbol{x}|}) \prod_{i=1}^{|\boldsymbol{x}|} p_{\boldsymbol{\tau}}(y_i \mid y_{i-1}) p_{\boldsymbol{\eta}}(x_i \mid y_i)$$

- since the output is a sequence, this argmax requires iterating over an exponentially-large set
- we can use **dynamic programming (DP)** to solve these problems exactly
- for HMMs (and other sequence models), the algorithm for solving this is the **Viterbi algorithm**

Dynamic Programming (DP)

- what is dynamic programming?
  - a family of algorithms that break problems into smaller pieces and reuse solutions for those pieces
  - only applicable when the problem has certain properties (**optimal substructure** and **overlapping sub-problems**)

- we can often use DP to iterate over exponentially-large output spaces in polynomial time
- we focus on a particular type of DP algorithm: **memoization**

Feature Locality

- **feature locality**: how big are the parts?
- for efficient exact inference with DP, we need to be mindful of this
- parts can be arbitrarily big in terms of input, but not in terms of *output*!
- HMM parts are small in both the input and output (only two pieces at a time)

Viterbi Algorithm for HMMs

- recursive equations + memoization:

base case:
returns probability of sequence starting with label *y* for first word

$$V(1, y) = p_{\boldsymbol{\eta}}(x_1 \mid y)\, p_{\boldsymbol{\tau}}(y \mid \text{<s>})$$

$$V(m, y) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y)\, p_{\boldsymbol{\tau}}(y \mid y')\, V(m-1, y') \right)$$

recursive case:
computes probability of max-probability label sequence that ends with label *y* at position *m*

final value is in: $goal(\boldsymbol{x}) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\tau}}(\text{</s>} \mid y')\, V(|\boldsymbol{x}|, y') \right)$

Viterbi Algorithm

- space and time complexity?
- can be read off from the recursive equations:

space complexity:
size of memoization table, which is # of unique indices of recursive equations

length of sentence * **number of labels**

$$V(m, y) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y) \, p_{\boldsymbol{\tau}}(y \mid y') \, V(m-1, y') \right)$$

so, space complexity is O(|*x*| |*L*|)

Viterbi Algorithm

- space and **time** complexity?
- can be read off from the recursive equations:

time complexity:
size of memoization table * complexity of computing each entry

length of sentence * **number of labels** * **each entry requires iterating through the labels**

$$V(m, y) = \max_{y' \in \mathcal{L}} \left(p_{\boldsymbol{\eta}}(x_m \mid y)\, p_{\boldsymbol{\tau}}(y \mid y')\, V(m-1, y') \right)$$

so, time complexity is O(|*x*| |*L*| |*L*|) = O(|*x*| |*L*|^2)

Viterbi Algorithm for Sequence Models

(with tag bigram features)

$$V(1, y) = \text{score}(\boldsymbol{x}, \langle \text{<s>}, y \rangle, 1, \boldsymbol{w})$$

$$V(m, y) = \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, \langle y', y \rangle, m, \boldsymbol{w}) + V(m-1, y'))$$

score function for label bigram <*y*’, *y*> ending at position *m* in ***x***

could be anything!
linear model, feed-forward network, LSTM, etc.