CS310 Natural Language Processing
自然语言处理
Lecture 04 - Recurrent Neural Networks and Sequence Labeling

Instructor: Yang Xu
主讲人：徐炀
xuyang@sustech.edu.cn

Overview

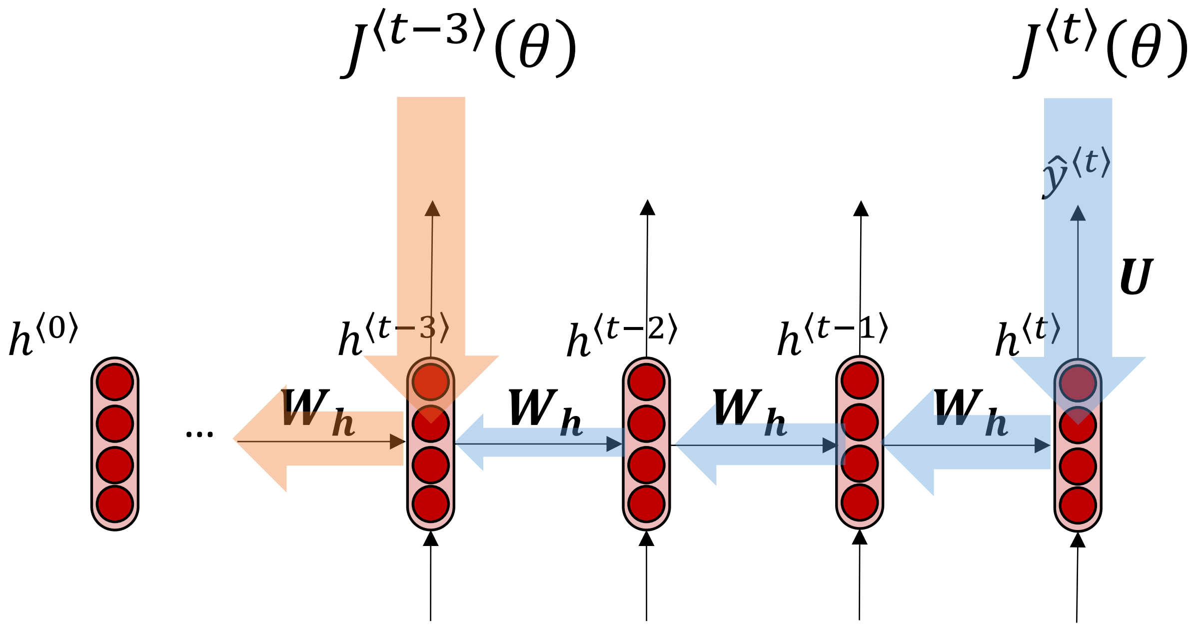
- Recap: Long Short-Term Memory RNNs (LSTMs)
- Bidirectional and multi-layer RNNs
- Sequence Labeling Task

Recap of Previous Lecture

- **Language Model**: Model for predicting next word
- **Recurrent Neural Network**: A family of neural networks that
  - Take sequential input of any length
  - Apply the same weights on each step
- RNNs ≠ Language Model
- RNNs are also useful for much more! (such as the sequence labeling task covered later)
- **Language Modeling** is a traditional subcomponent of many NLP tasks, all those involving generating text or estimating the probability of text

Problems with RNN-LM: Vanishing gradient

- Why is vanishing gradient a problem?



Gradient from far apart is lost because it's much smaller than gradient from close-by

So, model weights are only updated with respect to near effects, not long-term effects.

Effect of vanishing gradient on RNN-LM

step $i = 7$

- **Example:** When she tried to print her *tickets*, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her ______

step $j \gg 7$

- To learn from this training example, the RNN-LM needs to model the **dependency** between "tickets" on the 7th step and the target word "tickets" at the end.

- But if the gradient is small, the model can't learn this dependency
- the model is unable to predict similar ***long-distance dependencies*** at test time

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

How to fix the vanishing gradient problem?

- Main problem: it's too difficult for the RNN to preserve information over many timesteps.
- Because in vanilla RNN the hidden state is constantly being rewritten

$$\boldsymbol{h}^{\langle t\rangle} = g(\boldsymbol{W_h}\boldsymbol{h}^{\langle \boldsymbol{t-1}\rangle} + \boldsymbol{W}_e\boldsymbol{e}^{\langle t\rangle} + b_1)$$

- **Idea**: Design an RNN with separate memory added, besides the constantly updated hidden state

Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997; and a modern version with crucial improvement from Gers etal.(2000)

- Only started to be recognized as promising through the work of S's student Alex Graves in 2006

联结主义 vs. 符号主义

- Hist work: CTC(*connectionist* temporal classification) for speech recognition

- But only really became well-known after Geoffrey Hinton brought it to Google in 2013

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Core Design of LSTMs

- Each step has two states: hidden state $\boldsymbol{h}^{\langle t \rangle}$ and cell state $\boldsymbol{c}^{\langle t \rangle}$
  - They are vectors of same length n
  - The cell $\boldsymbol{c}^{\langle t \rangle}$ stores **long-term** information
  - Can read, erase, and write from/to the cell; like RAM in computer

- The selection of which information is read/erased/written is controlled by three corresponding **gates**:
  - Gates are also vectors of length n
  - At each step, each element in the gates can be open (1) or closed (0), or somewhere in between
  - Gates are dynamically computed based on the current context

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Long Short-Term Memory (LSTM)

$$\boldsymbol{i}^{\langle t\rangle} = \sigma(W_i\boldsymbol{h}^{\langle t-1\rangle} + U_i\boldsymbol{x}^{\langle t\rangle} + b_i)$$

Input gate: determines how much of the input should be added to the current cell

$$\boldsymbol{f}^{\langle t\rangle} = \sigma(W_f\boldsymbol{h}^{\langle t-1\rangle} + U_f\boldsymbol{x}^{\langle t\rangle} + b_f)$$

Forget gate: controls what is kept vs. forgotten from the previous cell state

$$\boldsymbol{o}^{\langle t\rangle} = \sigma(W_o\boldsymbol{h}^{\langle t-1\rangle} + U_o\boldsymbol{x}^{\langle t\rangle} + b_o)$$

Output gate: determines what part of cell should influence the output at current step

$$\tilde{\boldsymbol{c}}^{\langle t\rangle} = \tanh(W_c\boldsymbol{h}^{\langle t-1\rangle} + U_c\boldsymbol{x}^{\langle t\rangle} + b_c)$$

New cell content: new content to be written to cell

$$\boldsymbol{c}^{\langle t\rangle} = \boldsymbol{f}^{\langle t\rangle} \odot \boldsymbol{c}^{\langle t-1\rangle} + \boldsymbol{i}^{\langle t\rangle} \odot \tilde{\boldsymbol{c}}^{\langle t\rangle}$$

Updated cell state: "forget" some content from the previous cell and write some new content

$$\boldsymbol{h}^{\langle t\rangle} = \boldsymbol{o}^{\langle t\rangle} \odot \tanh(\boldsymbol{c}^{\langle t\rangle})$$

Hidden state: read some content from the cell

$\odot$ for element-wise product

LSTM Computational Graph

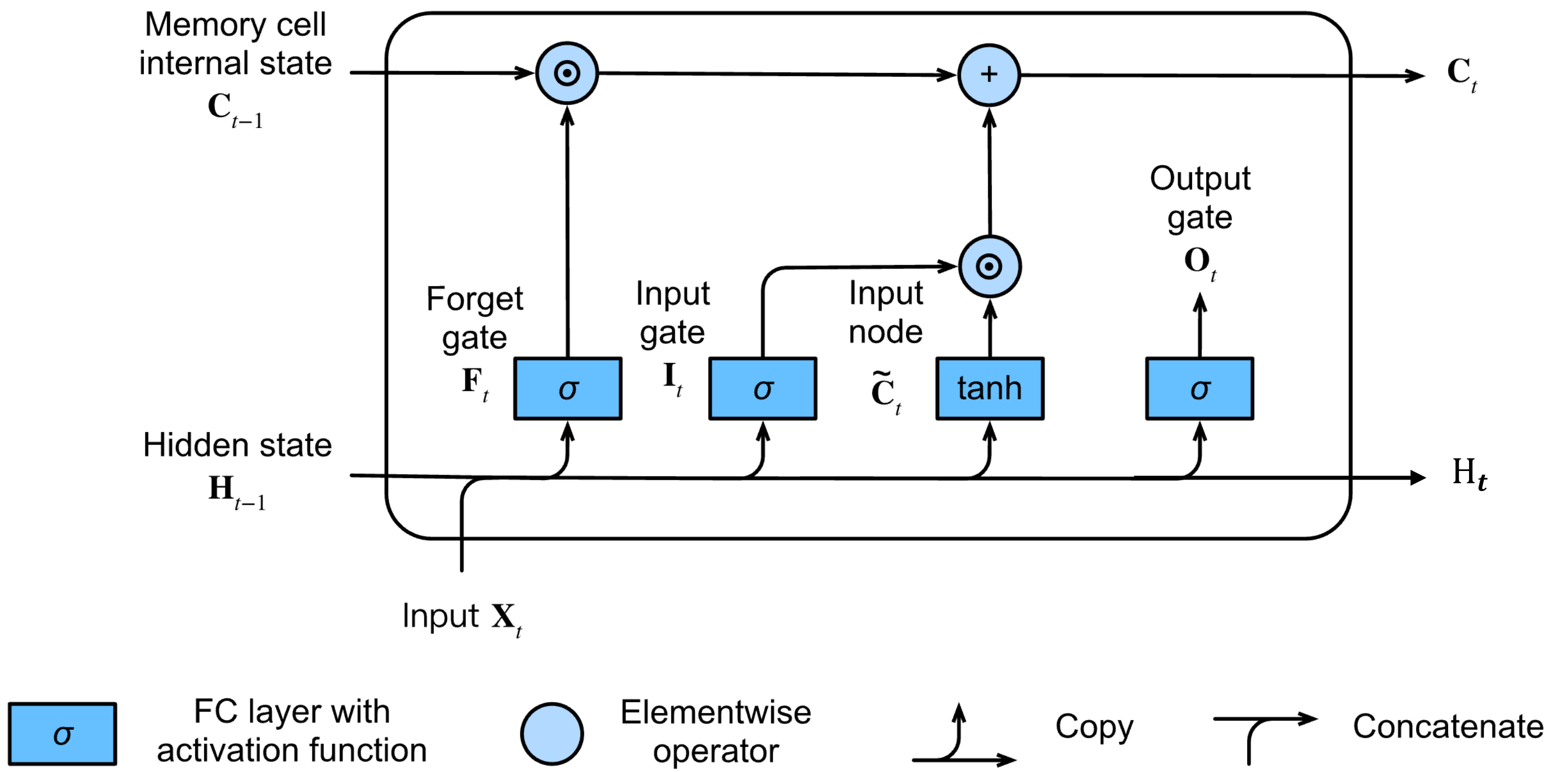


Figure from: https://d2l.ai/chapter_recurrent-modern/lstm.html

LSTM solves vanishing gradients

- LSTM makes it much easier for an RNN to preserve information over many steps
- If the forget gate $\boldsymbol{f}^{\langle t \rangle}$ is set to 1 (for a cell dimension) and the input gate $\boldsymbol{i}^{\langle t \rangle}$ set to 0, then the information (of that cell dimension) is preserved indefinitely.

$$\boldsymbol{c}^{\langle t \rangle} = \boldsymbol{f}^{\langle t \rangle} \odot \boldsymbol{c}^{\langle t-1 \rangle} + \boldsymbol{i}^{\langle t \rangle} \odot \tilde{\boldsymbol{c}}^{\langle t-1 \rangle}$$

LSTMs: History of Success

- In 2013–2015, **LSTMs** started achieving state-of-the-art results
  - Tasks include: language modeling, handwriting recognition, speech recognition, machine translation, parsing, and image captioning
  - LSTMs became the **dominant approach** for most NLP tasks

- For 2019--2023, **Transformers** have become dominant for all tasks
  - For example, in WMT (a Machine Translation conference + competition)
  - WMT2014 0 neural machine translation systems(!)
  - WMT2016 the summary report contains “RNN” 44 times
  - WMT2019: “RNN” 7 times, “Transformer” 105 times

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Overview

- Long Short-Term Memory RNNs (LSTMs)
- **Bidirectional and multi-layer RNNs**
- Sequence Labeling Task

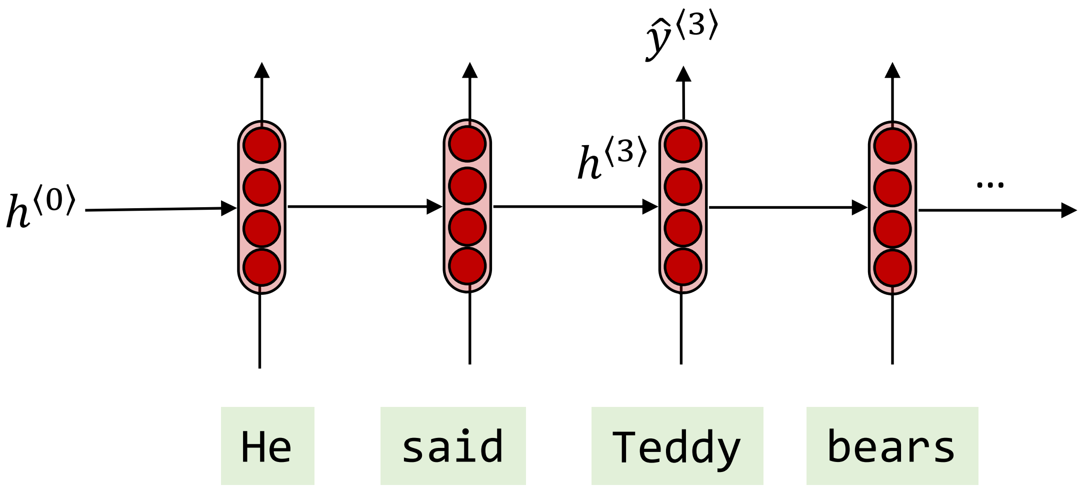
Context: From single or both direction

- Motivation: using only past information is not sufficient

`He said, “`*`Teddy`*` is a great person”`

`He said, “`*`Teddy`*` bears are on sale”`

Task: decide whether a word is a Person’s name

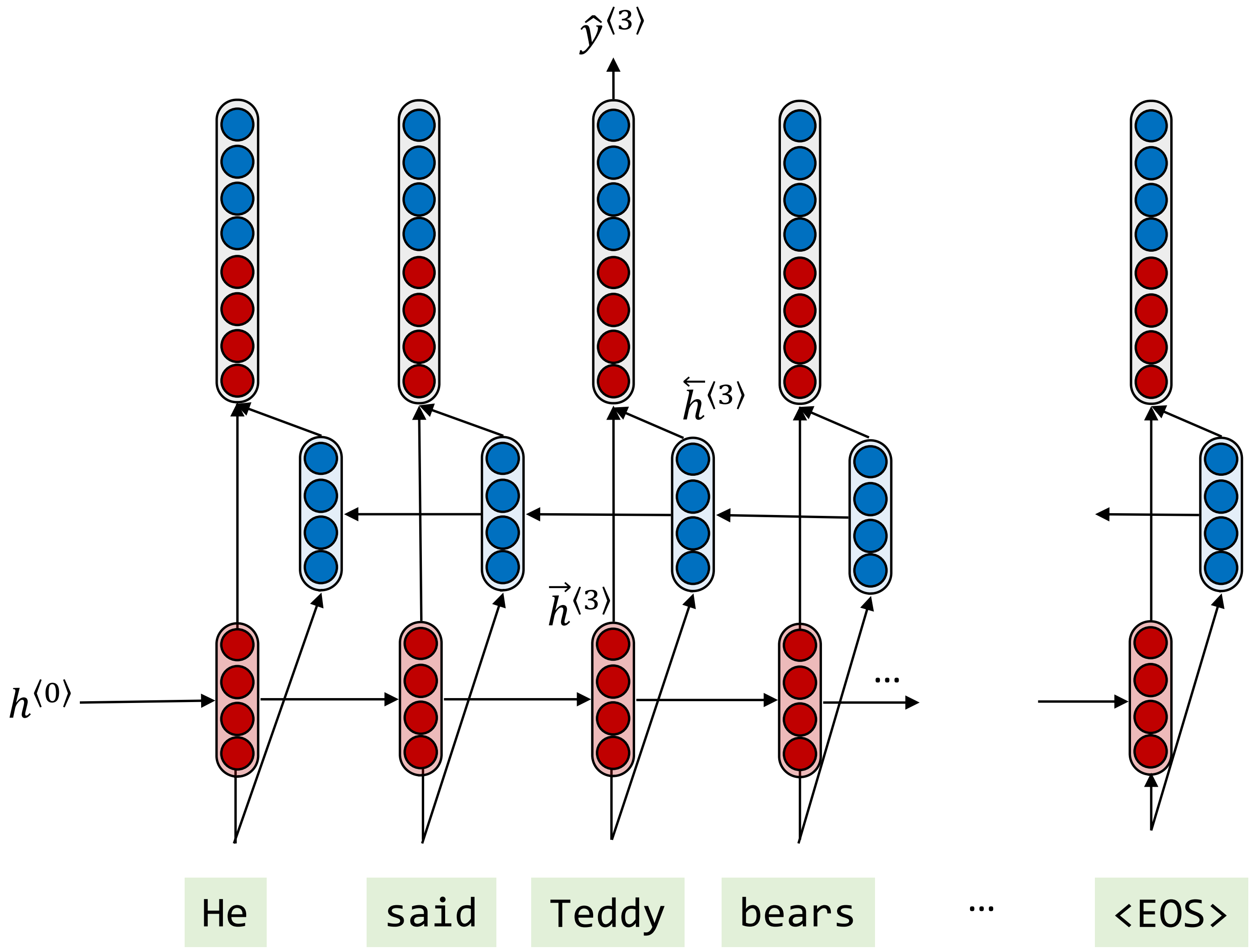


$h^{\langle 3 \rangle}$ is the representation of “Teddy” in the context of sentence

In particular, the **left** context: “He said”

What about **right** context?
“XX” modifies the meaning of “”

Bidirectional RNN



Now the representation of "Teddy" has both left $\overrightarrow{h}^{\langle 3 \rangle}$ and right $\overleftarrow{h}^{\langle 3 \rangle}$ context

Forward RNN:

$$\overrightarrow{h}^{\langle t \rangle} = \mathrm{RNN_F}\left(\overrightarrow{h}^{\langle t-1 \rangle}, x^{\langle t \rangle}\right)$$

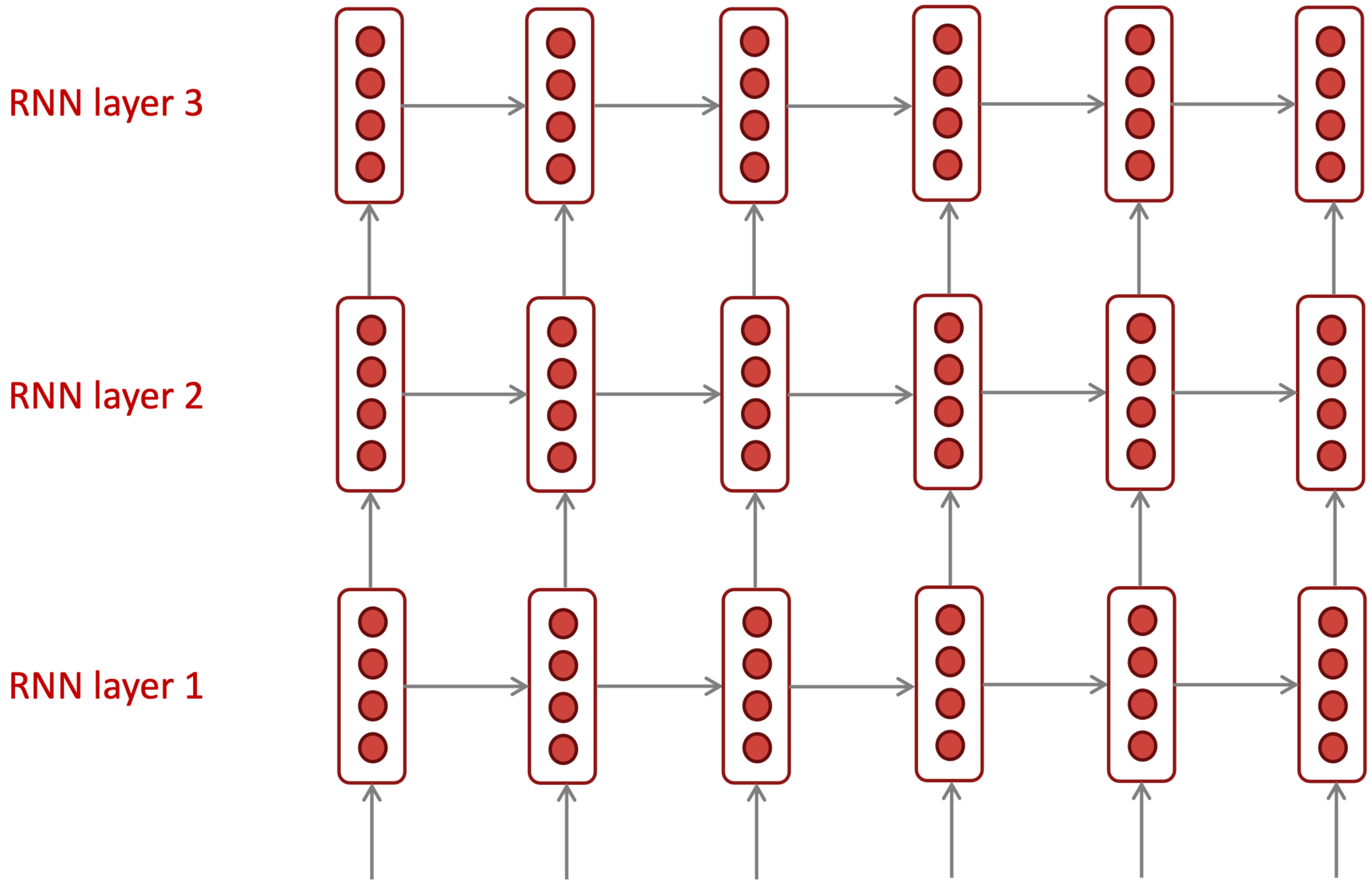
Backward RNN:

$$\overleftarrow{h}^{\langle t \rangle} = \mathrm{RNN_B}\left(\overleftarrow{h}^{\langle t-1 \rangle}, x^{\langle t \rangle}\right)$$

Concatenated hidden state:

$$\boldsymbol{h}^{\langle t \rangle} = \left[\overrightarrow{\boldsymbol{h}}^{\langle t \rangle}; \overleftarrow{\boldsymbol{h}}^{\langle t \rangle}\right]$$

Multi-layer RNNs



The hidden states from RNN layer i are the inputs to RNN layer $i + 1$

Each layer can be a bi-directional RNN layer

Multi-layer RNNs

- Multi-layer RNNs allow a network to compute more complex representations
  - lower layers compute **lower-level features** (lexical etc.) and the higher layers compute **higher-level features** (syntactic etc.)
- High-performing RNNs are usually multi-layer
- Practically, 2 layers is a lot better than 1, and 3 might be a little better than 2
- For deeper RNNs, skip-connections are needed

- Transformer-based networks (e.g., BERT) are usually deeper, like 12 or 24 layers (Will learn in later weeks)

Source: Britz et al, 2017. https://arxiv.org/pdf/1703.03906.pdf

Overview

- Long Short-Term Memory RNNs (LSTMs)
- Bidirectional and multi-layer RNNs
- **Sequence Labeling Task**
  - **Part of Speech Tagging**
  - **Named Entity Recognition**
  - **Algorithms**

Part of Speech

- Words can be classified into grammatical categories

Determinator	Noun	Verb	Determinator	Noun
the	Sun	rises	every	morning

determinator: *n.* 限定词

"n" for Noun ☺

Results returned from: https://corenlp.run/

- These categories are known as **part of speech (POS, POS tags)**, word classes, or simply grammatical categories.
- From the earliest (western) linguistic traditions (Yaska and Panini 5th C. BCE, Aristotle 4th C. BCE)

POS in Chinese Language

Not a complete list

实词 Notional Words **or content words**

名词 Nouns	动词 Verbs	助动词 Auxilliary Verbs	形容词 Adjectives	数词 Numerals	量词 Measure Words: 名量词 Nominal	量词 Measure Words: 动量词 Verbal	代词 Pronouns: 人称代词 Personal	代词 Pronouns: 指示代词 Demonstrative	代词 Pronouns: 疑问代词 Interrogative
国家、妹妹、玫瑰、颜色、月亮	工作、学习、走、吃、打	应该、可以、能、会、想	漂亮、诚实、慢、坏、红	一、二、百、万、亿	个、条、张、公分、只	次、遍、回、趟、轮	我们、你、他、她、咱们	这、那、各、每、该	什么、啥、哪、谁、怎么

unique in Chinese

虚词表 Functional Words

副词 Adverbs	介词 Prepositions	连词 Conjunctions	助词 Particles: 结构助词 Structural	助词 Particles: 动态助词 Aspectual	助词 Particles: 语气助词 Modal	叹词 Interjections	象声词 Onomatope
很、都、就、也、已经	从、向、在、被、把	跟、但是、或者、并且、因为	的、地、得	了、者、过	吗、吧、呢、了	喂、哎呀、嗯、哦	哗哗、乒乓

source: https://chinesenotes.com/grammar_intro.html

Two classes of words: Open vs. Closed

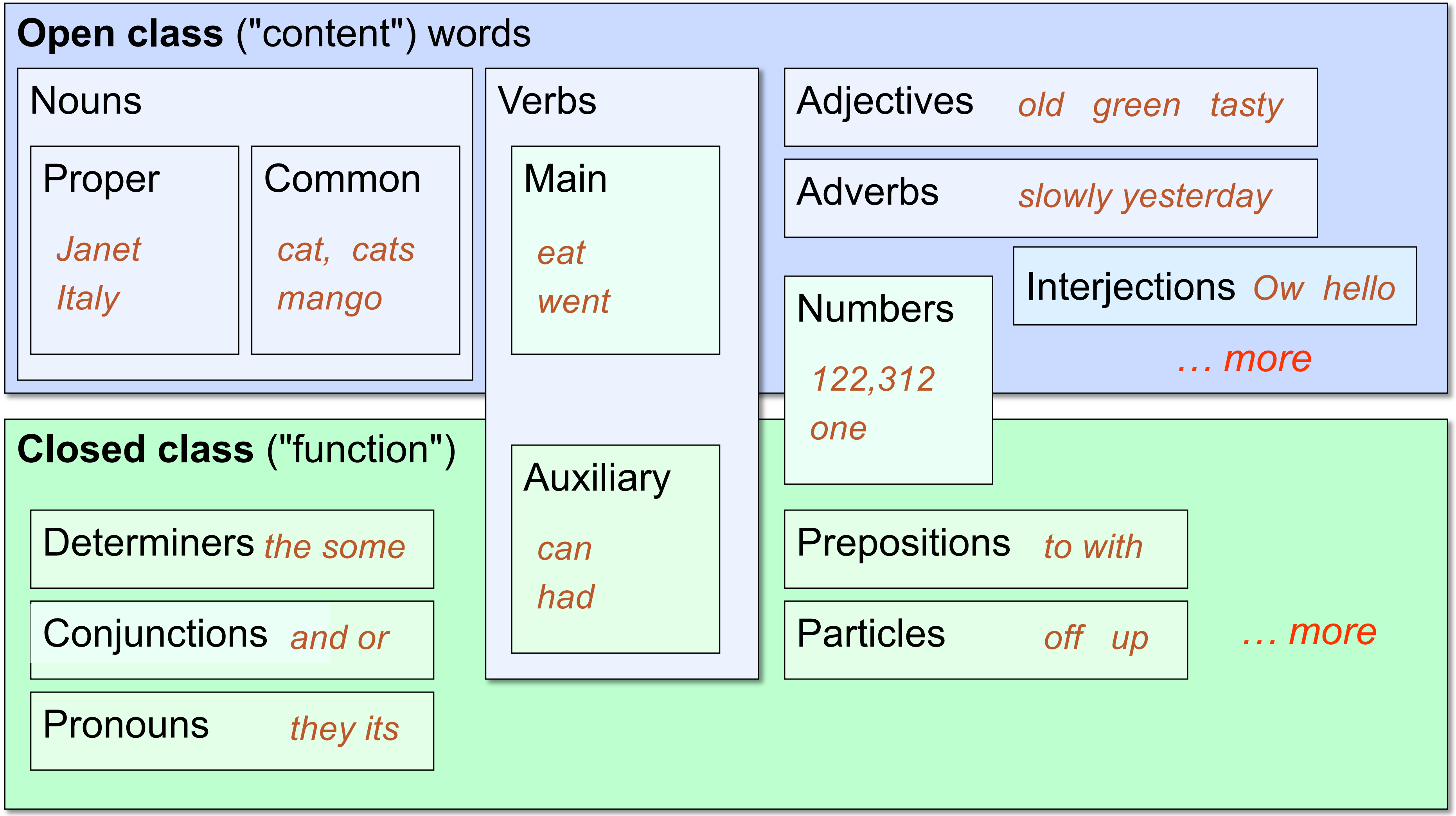
- Open class
  - Usually **content** words: **Nouns, Verbs, Adjectives, Adverbs**
  - New nouns and verbs like *AI* or *GPT* … continuously being created/borrowed

- Closed class
  - Relatively fixed membership
  - Usually **function** words: short, frequent words with grammatical function
    - **determiners** (限定词): *a, an, the*
    - **pronouns** (人称代词): *she, he, I*
    - **prepositions** (介词): *on, under, over, near, by, …*

Adapted from: https://web.stanford.edu/~jurafsky/slp3/slides/8_POSNER_intro_May_6_2021.pptx

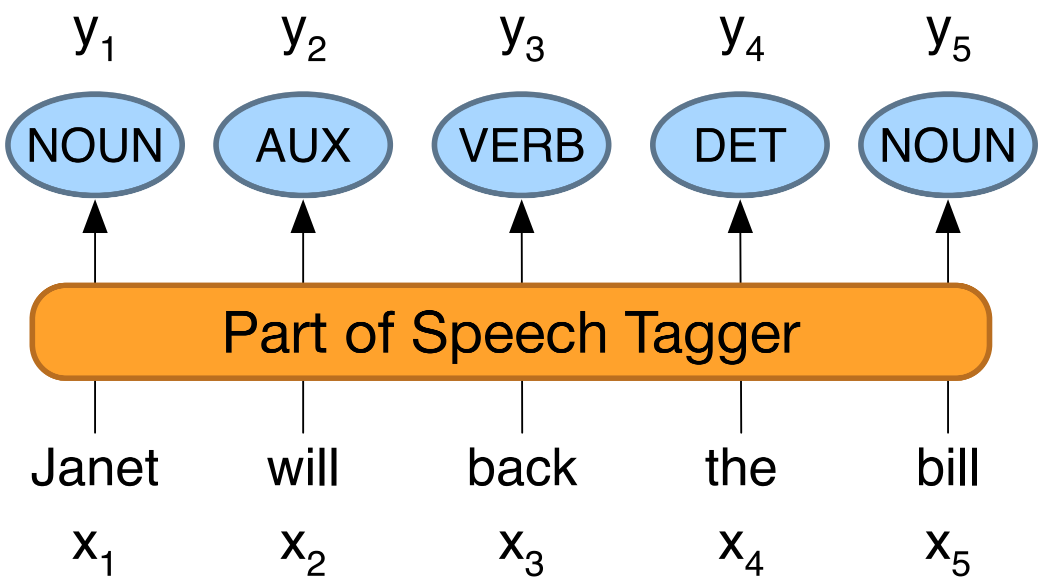
Common POS tags in English

Adapted from: https://web.stanford.edu/~jurafsky/slp3/slides/8_POSNER_intro_May_6_2021.pptx



Part of Speech Tagging Task

- Map from sequence of words $\boldsymbol{x}$ to sequence of POS tags $\boldsymbol{y}$



How difficult is POS tagging in English?

- Roughly 15% of word types are ambiguous
- Hence 85% of word types are unambiguous
- *Janet* is always PROPN, *hesitantly* is always ADV
- But those 15% tend to be very common. So ~60% of word tokens are ambiguous
- E.g., *back*
  - earnings growth took a back/ADJ seat
  - a small building in the back/NOUN
  - a clear majority of senators back/VERB the bill
  - enable the country to buy back/PART debt
  - I was twenty-one back/ADV then

Some ambiguous examples:

$\underbrace{\texttt{Love}}_{\texttt{NP}}\ \underbrace{\texttt{loves}}_{\texttt{VP}}\ \texttt{to}\ \underbrace{\texttt{love}}_{\texttt{VP}}\ \underbrace{\texttt{love}}_{\texttt{NP}}\texttt{.}$ -- *Ulysses* JAMES JOYCE

以其知之所知，以养其知之所不知

Noun Verb

--《庄子·大宗师》

Adapted from: https://web.stanford.edu/~jurafsky/slp3/slides/8_POSNER_intro_May_6_2021.pptx

How difficult is POS Tagging (in English)?

- How many tags are correct? (Tag accuracy)
  - About 97%
    - Hasn't changed in the last 10+ years
    - **HMMs**, **CRFs**, BERT perform similarly
    - Human accuracy about the same
- But baseline is 92%!
  - Baseline is performance of stupidest possible method
    - "Most frequent class baseline" is an important baseline for many tasks
      - Tag every word with its most frequent tag
      - (and tag unknown words as nouns)
  - Partly easy because
    - Many words are unambiguous

Adapted from: https://web.stanford.edu/~jurafsky/slp3/slides/8_POSNER_intro_May_6_2021.pptx

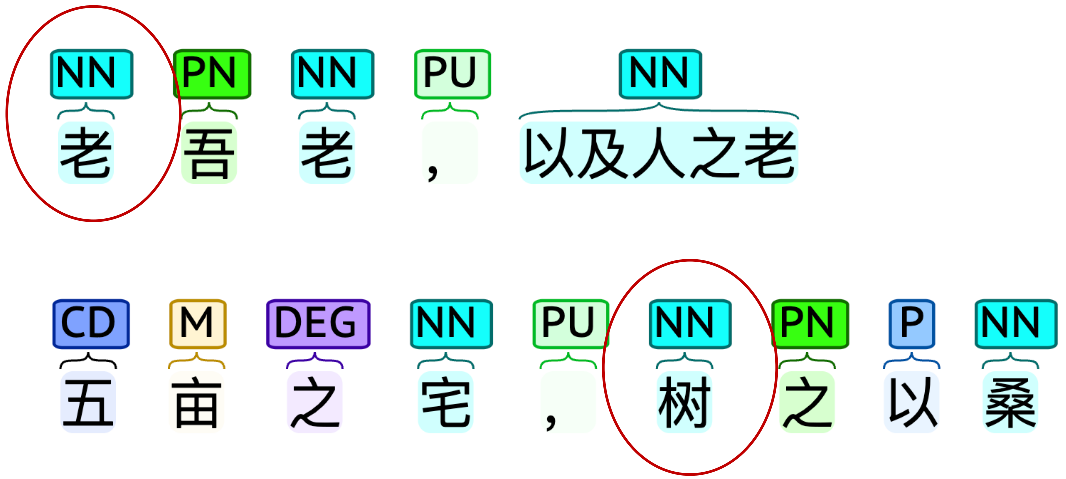
POS Tagging in Chinese

- Similar performance as English

System	F1 score
Tian el. al. (2020)	96.92
Meng et. al. (2019) (Glyce + BERT)	96.61
Meng et. al. (2019) (BERT)	96.06
Shao et. al. 2017	94.38

Data source: https://chinesenlp.xyz/docs/pos_tagging.html

Maybe true for modern Chinese (with better labeled data), but not necessarily true for all cases:



Noun or Verb?

Example:
- 快速 的 棕色 狐狸 跳过 了 懒惰 的 狗
- [快速] VA [的] DEC [棕色] NN [狐狸] NN [跳过] VV [了] AS [懒惰] VA [的] DEC [狗] NN

Named Entity Recognition “命名实体”识别

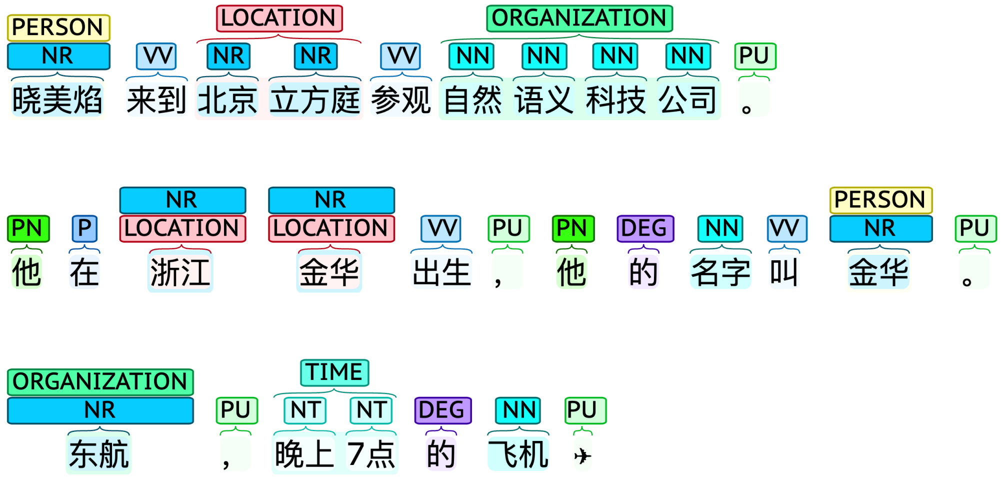
- **Named entity**: means anything that can be referred to with a proper name. Most common 4 tags:
  - PER (Person): “Zhang San”
  - LOC (Location): “Shenzhen City”
  - ORG (Organization): “Southern University of Science and Technology”
  - GPE (Geo-Political Entity): “Beijing, China”
- Often multi-word phrases
- But the term is also extended to things that aren't entities:
  - E.g., dates, times, prices

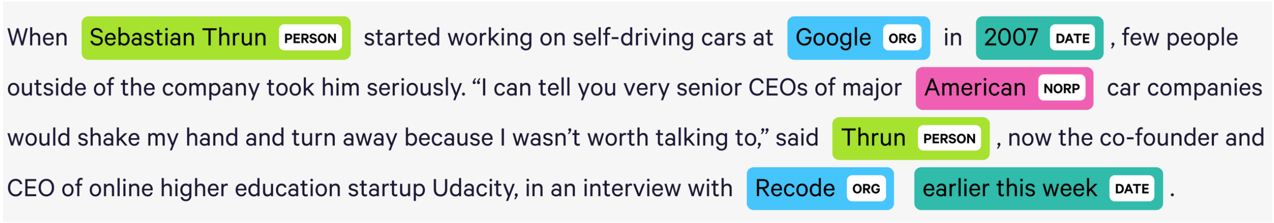
Adapted from: https://web.stanford.edu/~jurafsky/slp3/slides/8_POSNER_intro_May_6_2021.pptx

NER Output Example

- 晓美焰来到北京立方庭参观自然语义科技公司。
- 他在浙江金华出生，他的名字叫金华。
- 东航，晚上7点的飞机✈️

When Sebastian Thrun started working on self-driving cars at Google in 2007, few people outside of the company took him seriously. “I can tell you very senior CEOs of major American car companies would shake my hand and turn away because I wasn’t worth talking to,” said Thrun, now the co-founder and CEO of online higher education startup Udacity, in an interview with Recode earlier this week.





left from: https://hanlp.hankcs.com/demos/
right from: https://demos.explosion.ai/displacy-ent

Why NER?

- Sentiment analysis: consumer's sentiment toward a particular company or person?
- Question Answering: answer questions about an entity?
- Information Extraction: Extracting facts about entities from text.

slide credit: https://web.stanford.edu/~jurafsky/slp3/slides/8_POSNER_intro_May_6_2021.pptx

Why NER is hard?

1. Segmentation

- In POS tagging, no segmentation problem since each word gets one tag.
- In NER we have to find and segment the entities!

2. Type ambiguity

以其知之所知，以养其知之所不知

-- 《庄子·大宗师》

Noun Verb

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.

How: BIO Tagging Method

- Need to turn it into a sequence problem like POS tagging, with **one label per word**
- **Idea**: Use “B-” and “I-” prefixes for entity-words, and “O” for non-entity words

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

Adapted from: https://web.stanford.edu/~jurafsky/slp3/slides/8_POSNER_intro_May_6_2021.pptx

BIO Tagging Method

- **B**: token that *begins* a span
- **I**: tokens *inside* a span
- **O**: tokens outside of any span
- **# of tags** (where n is #entity types):
- 1 O tag,
- *n* B tags,
- *n* I tags
- total of *2n+1*

Words	**BIO Label**
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

Adapted from: https://web.stanford.edu/~jurafsky/slp3/slides/8_POSNER_intro_May_6_2021.pptx

Overview

- Long Short-Term Memory RNNs (LSTMs)
- Bidirectional and multi-layer RNNs
- **Sequence Labeling Task**
  - Part of Speech Tagging
  - Named Entity Recognition
  - **Algorithms**

Standard Algorithms for POS Tagging and NER

- Supervised Machine Learning given a human-labeled training set of text annotated with tags
- Hidden Markov Models
- Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Large Language Models (like BERT), finetuned

Sequential Labeling Task in General

- **Problem statement**: a sequence of input words $\boldsymbol{x} = x_1, \ldots, x_n$, map it to a sequence of labels $\boldsymbol{y} = y_1, \ldots, y_n$, from the label set $\mathcal{L}$
- The goal is to find the labels:

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{L}} p(\boldsymbol{y}|\boldsymbol{x})$$

- where the probability $P(\boldsymbol{y}|\boldsymbol{x})$ can be represented with an abstract **score** function:

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{L}} \text{score}(\boldsymbol{y}, \boldsymbol{x}; \boldsymbol{\theta})$$

- $\boldsymbol{\theta}$ indicates model parameters; different difficulty levels in implementing score

Level 0: Local classifier

- $\mathrm{score}(\boldsymbol{x}, i, y; \boldsymbol{\theta})$: Scoring the label $y \in \mathcal{L}$ for x_i using all words in the sequence:
- For example, we can use the hidden state at step i from an RNN (Bi-LSTM), connect it to softmax:

$$\begin{aligned} \hat{y}_i &= \arg\max_{y \in \mathcal{L}} \mathrm{score}(\boldsymbol{x}, i, y; \boldsymbol{\theta}) \\ &= \arg\max_{y \in \mathcal{L}} \mathrm{softmax}(\boldsymbol{h}^{\langle i \rangle}) \end{aligned}$$

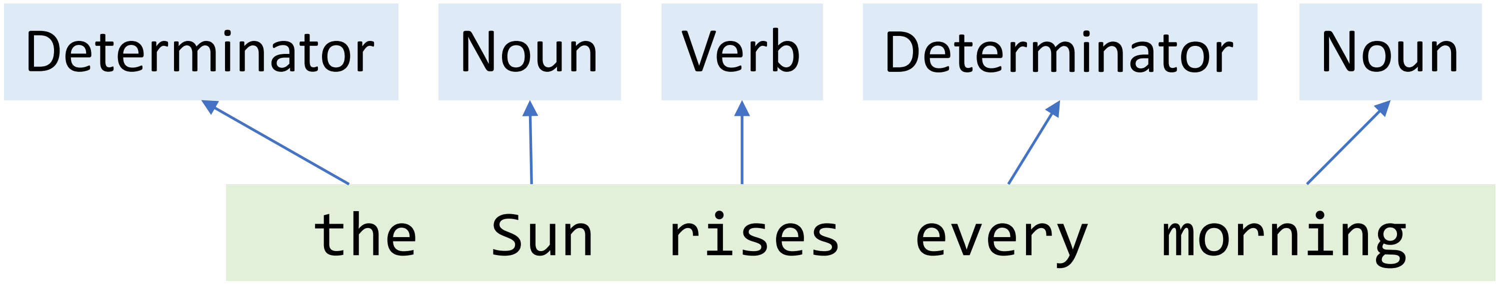
$\left.\begin{array}{l} y_i\text{: one-hot} \\ \hat{y}_i\text{: prob distr} \end{array}\right\} \Rightarrow \text{loss}$

- The classifier model decodes locally, i.e., produce a label to each x_1, x_2 … in turn, with all words made available at each position (via the bi-direction architecture)
- We can do better by using the predictable relationship among labels $\hat{\boldsymbol{y}}$

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Limit of level 0 - local classifier

- Labels cannot affect each other!
- Example: $p(\mathrm{NN}|\mathrm{DET})$ should naturally be higher; but this information is not used



Level 1: Sequential classifier

- $\mathrm{score}(\boldsymbol{x}, i, \widehat{\boldsymbol{y}}_{1:i-1}, y; \boldsymbol{\theta})$: Scoring the label $y \in \mathcal{L}$ for x_i using all words, AND the *previously predicted labels*.

$$\hat{y}_i = \arg\max_{y \in \mathcal{L}} \mathrm{score}(\boldsymbol{x}, i, \widehat{\boldsymbol{y}}_{1:i-1}, y; \boldsymbol{\theta})$$
$$= \arg\max_{y \in \mathcal{L}} \left(\mathrm{softmax}\left(\boldsymbol{h}^{\langle i \rangle}\right)\right) + \mathrm{information}(\widehat{\boldsymbol{y}}_{1:i-1})$$

- The classifier produces a label to each x_1, x_2 ... in turn. Each one uses additional information from the preceding predictions ($\widehat{\boldsymbol{y}}_{1:i-1}$).
- Directly using $\mathrm{information}(\widehat{\boldsymbol{y}}_{1:i-1})$ at training is hard (need a new loss; see Wiseman et al. (2016))
- Testing time is easier ⇒ Classical method: **Beam search**

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

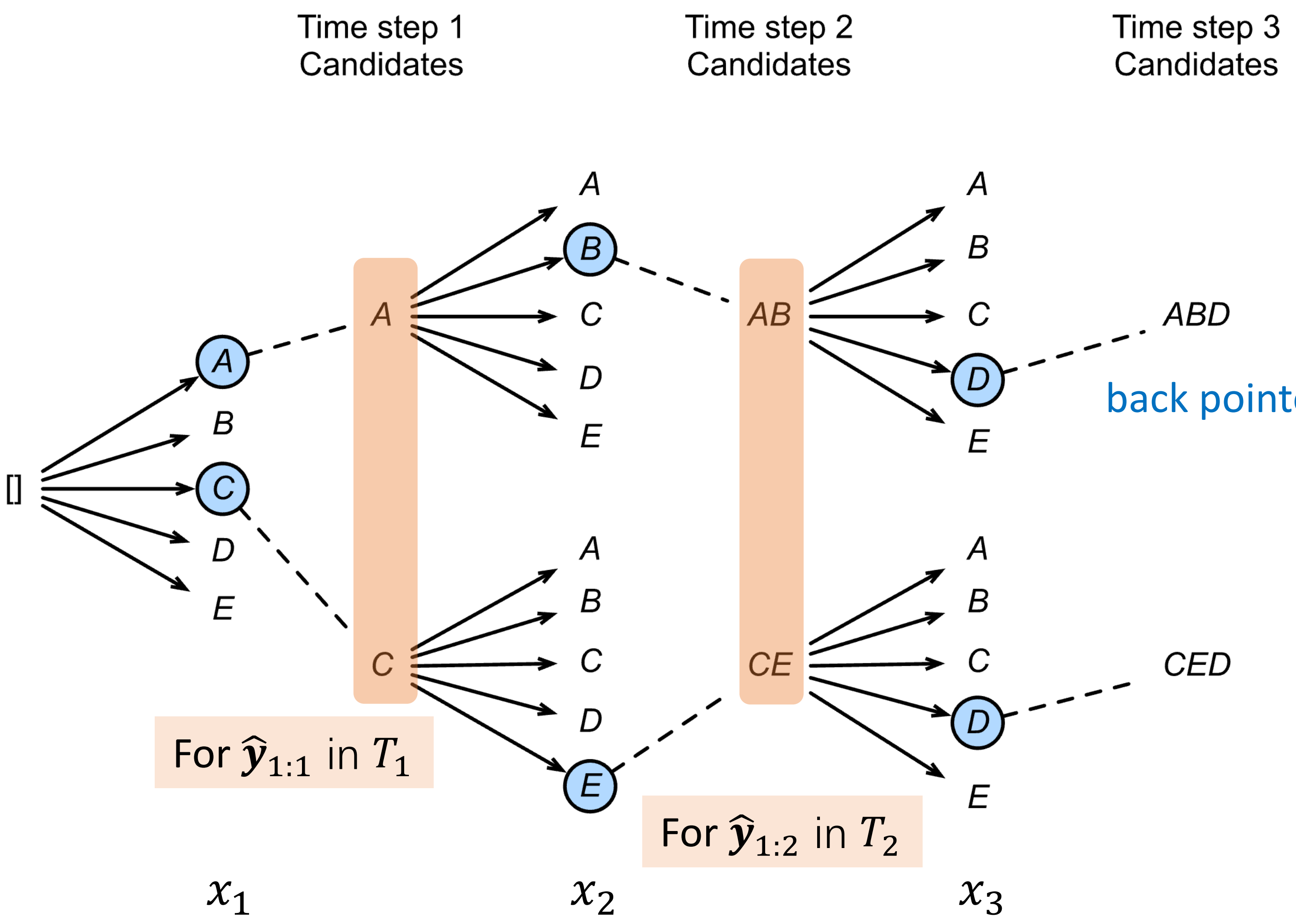
Beam Search for Decoding

"Beam"

- **Input**: sequence $\boldsymbol{x}$, beam width k, and classifier's scoring function $\mathrm{score}(\boldsymbol{x}, i, \hat{\boldsymbol{y}}_{1:i-1}, y)$
- Let T_0 be the top scored labels for step 0 (imaginary; initialized to $\emptyset$)
- For $i \in \{1, \ldots, n\}$:
  - Empty candidate set C
  - For each previous predicted tag sequence $\hat{\boldsymbol{y}}_{1:i-1}$ in T_{i-1}:
    - For each $y \in \mathcal{L}$, insert a new candidate prediction $\hat{\boldsymbol{y}}_{1:i-1}y$ into C whose score is: $T_{i-1}(\hat{\boldsymbol{y}}_{1:i-1}) + \mathrm{score}(\boldsymbol{x}, i, \hat{\boldsymbol{y}}_{1:i-1}, y)$
  - Let T_i be the top-k scored candidates of C
- **Output**: Best scored label in T_n (and its preceding labels in $T_1 \ldots T_{n-1}$ using back pointers)

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Beam Search Example



Beam width $k = 2$, sequence length $n = 3$

$T_0 = \emptyset$

$T_1 = \{A, C\} \Rightarrow$ top 2 candidates for step 1

back pointers

$T_2 = \{B, E\} \Rightarrow$ top 2 candidates for step 2

$T_3 = \{D, D\} \Rightarrow$ top 2 candidates for step 3

The best label sequences are among:
$\{A \to B \to D, C \to E \to D\}$

pick one using some standards

Example from: https://d2l.ai/chapter_recurrent-modern/beam-search.html

Notes on Beam Search for Sequential Classifier

- Time cost is $O(knL)$, let $L = |\mathcal{L}|$, i.e., size of label set, n is sequence length
- Special cases:
  - $k = 1 \Rightarrow$ greedy search
  - $k = O(L^n) \Rightarrow$ brute force exhaustive search
- What if $\hat{\boldsymbol{y}}_{1:i-1}$is wrong? "Downstream" effects of a mistake can be catastrophic.
- No guarantee! Beam search **does not** return global optimal y.

Level 2: Hidden Markov Model

- **Idea**: Just like a language model, there is sequential information between **labels**
- HMM: A **generative** approach ⇒ Labeled sequence is generated according to the following process:

y_1

$y_1 \sim p_{\text{start}}(Y)$

The label for step 1 is drawn from some prior probability of all labels

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Level 2: Hidden Markov Model

- HMM: A generative approach ⇒ Labeled sequence is generated according to the following process:

$$x_1$$

$$\uparrow$$

$$y_1$$

$$x_1 \sim p_{\text{emission}}(X|y_1)$$

The word at step 1 x_1 is "emitted" according to the conditional emission probability distribution of words

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Level 2: Hidden Markov Model

- HMM: A generative approach ⇒ Labeled sequence is generated according to the following process:

$$
\begin{array}{ccc}
x_1 & & \\
\uparrow & & \\
y_1 & \rightarrow & y_2
\end{array}
$$

$$y_1 \sim p_{\text{transition}}(Y|y_1)$$

The label at step 2 y_2 is generated according to the conditional transition probability of labels

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Level 2: Hidden Markov Model

- HMM: A generative approach ⇒ Labeled sequence is generated according to the following process:

$$\begin{array}{ccc} x_1 & & x_2 \\ \uparrow & & \uparrow \\ y_1 & \rightarrow & y_2 \end{array}$$

$$x_1 \sim p_{\text{emission}}(X|y_2)$$

The word at step 2 x_2 is “emitted” according to the conditional emission probability distribution of words

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Level 2: Hidden Markov Model

- HMM: A generative approach ⇒ Labeled sequence is generated according to the following process:

$$
\begin{array}{ccccccccc}
x_1 & & x_2 & & x_3 & & x_4 & & \\
\uparrow & & \uparrow & & \uparrow & & \uparrow & & \\
y_1 & \rightarrow & y_2 & \rightarrow & y_3 & \rightarrow & y_4 & \rightarrow & \text{<EOS>}
\end{array}
$$

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Level 2: Hidden Markov Model

- Based on Markov assumption:

assume $y_{n+1} = \langle \text{EOS} \rangle$

$$P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) = p_{\text{start}}(y_1) \cdot \prod_{i=1}^{n} \left(p_{\text{emission}}(x_i | y_i) \cdot p_{\text{transition}}(y_{i+1} | y_i) \right)$$

Goal of labeling task:

$$\begin{aligned} \hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{Y} \in \mathcal{L}} P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) \\ &= \arg\max_{\boldsymbol{Y} \in \mathcal{L}} P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) \\ &= \arg\max_{\boldsymbol{Y} \in \mathcal{L}} \log P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) \end{aligned}$$

$$= \arg\max_{\boldsymbol{Y} \in \mathcal{L}} \log p_{\text{start}}(y_1) + \sum_{i=1}^{n} (\log p_{\text{emission}}(x_i | y_i) + \log p_{\text{transition}}(y_{i+1} | y_i))$$

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Classical HMM

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{Y} \in \mathcal{L}} \log p_{\text{start}}(y_1) + \sum_{i=1}^{n} \left(\log p_{\text{emission}}(x_i|y_i) + \log p_{\text{transition}}(y_{i+1}|y_i)\right)$$

- Parameters are all interpretable as probabilities
- p_{start} is a distribution over labels $\mathcal{L}$: can be estimated by **counting** how often sequences start with each label in training data (and normalize)
- p_{emission} is a distribution over words for each label. Somewhat counterintuitive! Estimated by **counting** words frequencies that associate with certain labels (and normalize)
    - For instance, for all words associated with label **DET**, "the" occurs 100 times, "a" occurs 75 times and "an"25 times ⇒ $p_{emission}(\text{“the”}|\text{DET}) = \frac{100}{200} = .50$
- $p_{\text{transition}}$ is first-order Markov model (another name for bigram) of all labels

Limits of Classical HMM

- All probabilities have a closed form ($p_{\text{start}}, p_{emission}, p_{\text{transition}}$) with labeled data
- Could suffer from sparsity issue if we simply estimate these probabilities by counting
- Lack of feature from words: $p_{\text{emission}}(x_i|y_i)$ only conditioned on y_i but not on the entire sequence $\boldsymbol{x}$ (thus, not a good language model)
- $p_{\text{transition}}(y_{i+1}|y_i)$ is also not a limited estimation: no $\boldsymbol{y}_{\mathbf{1}:\boldsymbol{i}-\mathbf{1}}$ is used

- If not by counting, then how to estimate the probabilities? Using what **features**?

Level 3: Maximum Entropy Markov Model

- MEMM: An improvement over classical HMM
- Replace the “lookup” probabilities with scoring functions

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{L}} \log p_{\text{start}}(y_1) + \sum_{i=1}^{n} \left(\log p_{\text{emission}}(x_i | y_i) + \log p_{\text{transition}}(y_{i+1} | y_i)\right)$$

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{L}} \phi_{\text{start}}(y_1) + \sum_{i=1}^{n} \left(\phi_{\text{emission}}(x_i, y_i) + \phi_{\text{transition}}(y_i, y_{i+1})\right)$$

Combine and include whole sequence

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{L}} \sum_{i=0}^{n} \phi(\boldsymbol{x}, y_i, y_{i+1})$$

ϕ function should be capable of capturing information from the whole sequence

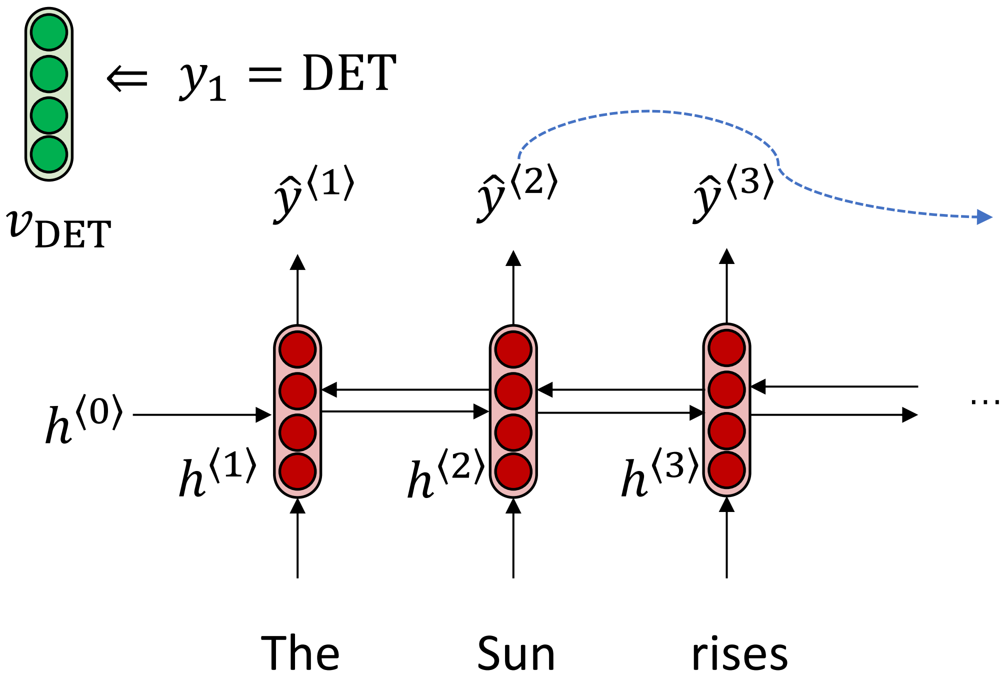
-- RNN!

Level 3: MEMM by RNN

- Choice for $\phi(\boldsymbol{x}, y_i, y_{i+1}) \Rightarrow$ it should measures the likelihood $p(y_{i+1}|\boldsymbol{x}, y_i)$ and be in the form of a valid probability:

$$\phi = \frac{\exp\big(\text{feat}(\boldsymbol{x}, y_i, y_{i+1})\big)}{\sum_{y_{i+1}\in\mathcal{L}} \exp\Big(\text{feat}\big((\boldsymbol{x}, y_i, y_{i+1})\big)\Big)}$$

feat() is a parameterized feature function



$$p(y_2 = \text{NOUN}|\boldsymbol{x}, y_1 = \text{DET}) =$$

$$= \frac{\exp\big(\text{fc}\big(\big[\boldsymbol{h}^{\langle 2\rangle}; v_{\text{DET}}\big]\big)\odot v_{\text{NOUN}}\big)}{\sum_{y\in\mathcal{L}} \exp\big(\text{fc}([\boldsymbol{h}^{\langle 2\rangle}; v_{\text{DET}}])\odot v_y\big)}$$

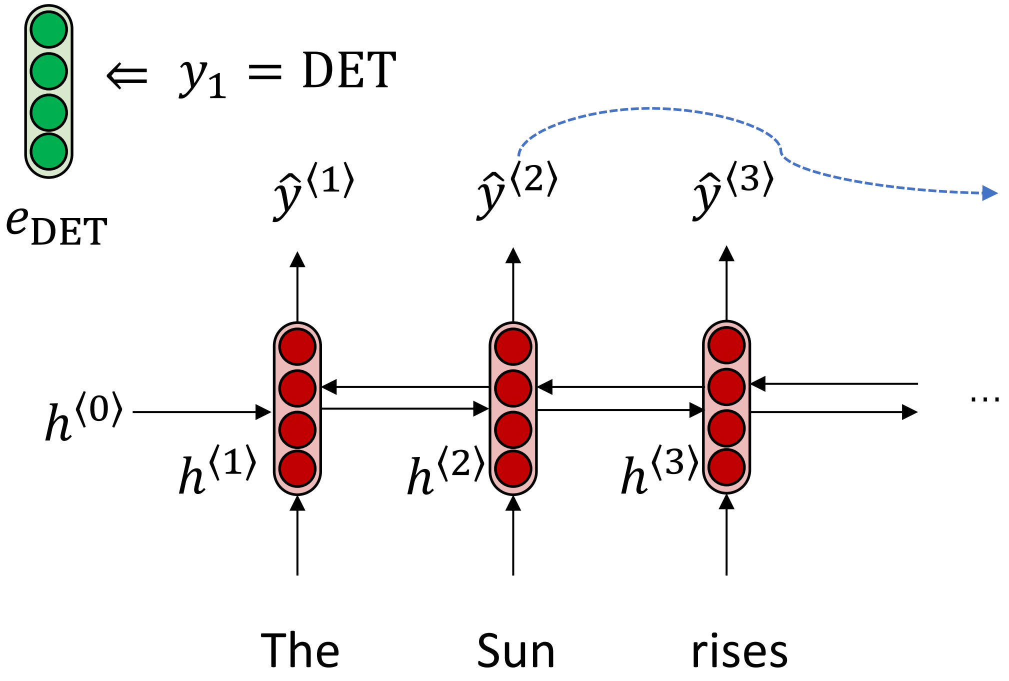
- fc() is a fully-connected layer
- $\big[\boldsymbol{h}^{\langle 2\rangle}; v_{\text{DET}}\big]$ is the concatenation of $\boldsymbol{h}^{\langle 2\rangle}$ and v_{DET}
- $\odot$ is dot product

Level 3: MEMM by RNN

In softmax, we implicitly have a set of parameters $\boldsymbol{v}'_y$ whose dimension is $d(\boldsymbol{h}) + d(v)$

$$\mathrm{softmax}\left(\left[\boldsymbol{h}^{\langle 2\rangle}; v_{\mathrm{DET}}\right]\right)$$

Similar



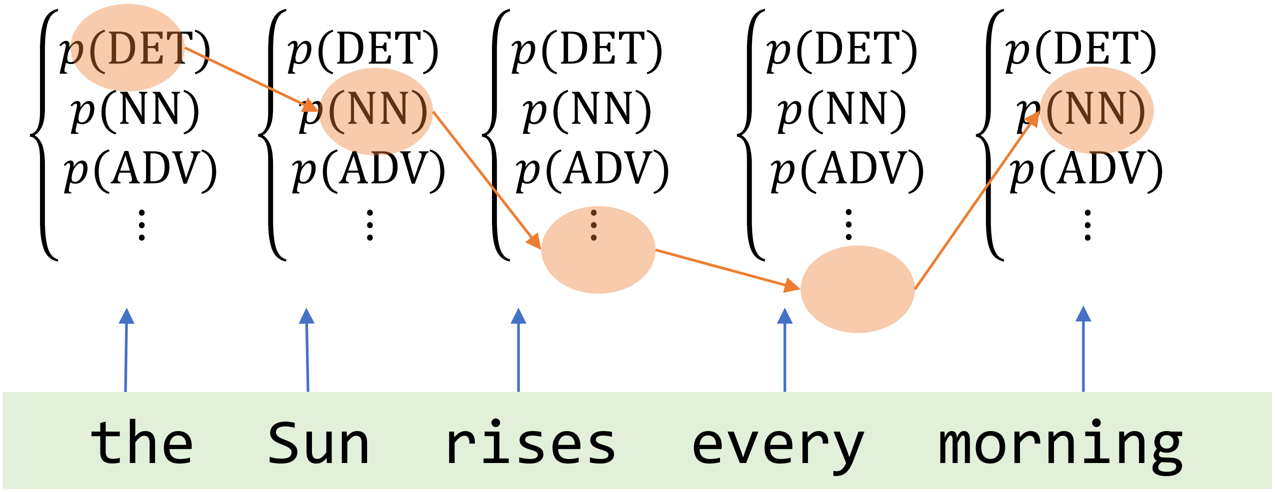
$$p(y_2 = \mathrm{NOUN} | \boldsymbol{x}, y_1 = \mathrm{DET}) =$$

$$= \frac{\exp\left(\mathrm{fc}\left(\left[\boldsymbol{h}^{\langle 2\rangle}; v_{\mathrm{DET}}\right]\right) \odot v_{\mathrm{NOUN}}\right)}{\sum_{y \in \mathcal{L}} \exp\left(\mathrm{fc}\left(\left[\boldsymbol{h}^{\langle 2\rangle}; v_{\mathrm{DET}}\right]\right) \odot v_y\right)}$$

- fc() is a fully-connected layer:
- $\left[\boldsymbol{h}^{\langle 2\rangle}; v_{\mathrm{DET}}\right]$ is the concatenation of $\boldsymbol{h}^{\langle 2\rangle}$ and v_{DET}
- $\odot$ is dot product

Reflection on Level 0 - 3

- **Decoding** is essentially important!
- Core question: How to efficiently get the most likely $\hat{\boldsymbol{y}}$ out of the model's prediction by a step by step decoding?
- Known fact: Searching for the global optima in brute force way is expensive.



Let $L = |\mathcal{L}|$, then there are L^n possible paths

Beam search is not guaranteed to find the global optima

Need a more efficient way

Efficient Decoding needed

- Decode: how to get the most likely $\boldsymbol{y}$ for the $\boldsymbol{x}$ (for both training and testing time)
- **Idea**: The decision for $\hat{y}_i$ is a function of y_{i-1}, $\boldsymbol{x}$, and nothing else.
- If for each value of $y_{i-1} \in \mathcal{L} = \{\ell_1, \ell_2, \dots, \ell_L\}$, we knew the maximum likelihood of $\boldsymbol{x}_{\mathbf{1}:\boldsymbol{i}-\mathbf{1}}$ ends with $\ell_1, \ell_2, \dots, \ell_L$, and the corresponding best labels $\boldsymbol{y}_{\mathbf{1}:\boldsymbol{i}-\mathbf{1}}$, then picking $\hat{y}_i$ would be easy.

- **Solution**: Maintain a $L{\times}n$ table to store the likelihood score of the best label prefix $y_{1:i}$ ending in each label value ℓ for each step i.
- **Viterbi Algorithm**: A dynamic programming algorithm; guaranteed to find the same global optimal solution as brute-force but faster!

Viterbi Algorithm

Recall: ϕ measures likelihood

Table π with entries $\pi[y_i, i]$, $y_i = \ell_1 \dots \ell_L$, $i = 1 \dots n$

$$\pi[y_i, i] = \max_{y_{i-1} \in \mathcal{L}} \left(\pi[y_{i-1}, i-1] + \phi(\boldsymbol{x}, y_{i-1}, y_i)\right)$$

Input sequence

$\mathcal{L}$

$\langle\text{BOS}\rangle$	x_1	x_2	...	x_n	
ℓ_1	$\pi[\ell_1, 1]$	$\pi[\ell_1, 2]$			
ℓ_2	$\pi[\ell_2, 1]$				
...					
ℓ_L	$\pi[\ell_k, 1]$				
$\langle\text{EOS}\rangle$					

$\pi[y_1, 1] = \phi(\boldsymbol{x}, \langle\text{BOS}\rangle, y_1)$ $\langle$BOS$\rangle$ for begin-of-sentence

$$\pi[y_2 = \ell_1, 2] = \max \begin{cases} \pi[\ell_1, 1] + \phi(x, \ell_1, y_2) \\ \pi[\ell_2, 1] + \phi(x, \ell_2, y_2) \\ \pi[\ell_3, 1] + \phi(x, \ell_3, y_2) \\ \quad\dots \end{cases}$$

$$\pi[\langle\text{EOS}\rangle, n+1] = \max_{y_n \in \mathcal{L}} \left(\pi[y_n, n] + \phi(\boldsymbol{x}, y_n, \langle\text{EOS}\rangle)\right)$$

Viterbi Algorithm: Keep back-pointers

Input sequence

$\mathcal{L}$

$\langle\text{BOS}\rangle$	x_1	x_2	…	x_n	
ℓ_1	$\pi[\ell_1, 1]$ $\text{bp}_1(\ell_1)$	$\pi[\ell_1, 2]$ $\text{bp}_2(\ell_1)$			
ℓ_2	$\pi[\ell_2, 1]$ $\text{bp}_1(\ell_2)$				
…					
ℓ_L	$\pi[\ell_k, 1]$ $\text{bp}_1(\ell_L)$				
$\langle\text{EOS}\rangle$					$\pi[\langle\text{EOS}\rangle, n+1]$ $\text{bp}_{\text{n+1}}(\langle\text{EOS}\rangle)$

$$\pi[y_i, i] = \max_{y_{i-1} \in \mathcal{L}} \left(\pi[y_{i-1}, i-1] + \phi(\boldsymbol{x}, y_{i-1}, y_i)\right)$$

$$\text{bp}_i(\ell_i) = \arg\max_{y_{i-1} \in \mathcal{L}} \left(\pi[y_{i-1}, i-1] + \phi(\boldsymbol{x}, y_{i-1}, y_i)\right)$$

$$\text{bp}_1(\ell_i) = \langle\text{BOS}\rangle$$

$$\text{bp}_2(\ell_1) = \arg\max \begin{cases} \pi[\ell_1, 1] + \phi(x, \ell_1, y_2) \\ \pi[\ell_2, 1] + \phi(x, \ell_2, y_2) \\ \pi[\ell_3, 1] + \phi(x, \ell_3, y_2) \\ \cdots \end{cases}$$

Viterbi Algorithm Complexity

- Need to fill in a $n \times L$ table; for each cell, need to iterate over L previous cells $\Rightarrow O(nL^2)$
- Compared to beam search:
- Viterbi calculates max and argmax at each step; beam search is an approximation.
- Viterbi **guaranteed the global optima** while beam search not.

Input sequence

$\mathcal{L}$

$\langle \text{BOS} \rangle$	x_1	x_2	…	x_n	
ℓ_1					
ℓ_2					
…					
ℓ_L					
$\langle \text{EOS} \rangle$					

Level 4: Conditional Random Fields

- Recall MEMM:

$$\hat{\boldsymbol{Y}} = \arg\max_{\boldsymbol{Y} \in \mathcal{L}} \sum_{i=0}^{n} \phi(\boldsymbol{x}, y_i, y_{i+1}) \quad \text{where:} \quad \phi = \frac{\exp\big(\text{feat}(\boldsymbol{x}, y_i, y_{i+1})\big)}{\sum_{y_{i+1} \in \mathcal{L}} \exp\Big(\text{feat}\big((\boldsymbol{x}, y_i, y_{i+1})\big)\Big)}$$

- Extend ϕ to some function over the entire sequence:

$$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \frac{\exp\big(\text{FEAT}(\boldsymbol{x}, \boldsymbol{y})\big)}{\sum_{\boldsymbol{y} \in \mathcal{L}} \exp\big(\text{FEAT}(\boldsymbol{x}, \boldsymbol{y})\big)}$$

The parameterized feature function **FEAT()** is designed over the entire set of **all possible $\boldsymbol{y}$ sequences.**

Reference: Lafferty et al. (2001)

Level 4: Conditional Random Fields

- CRFs generalizes multinomial logistic regression to structured outputs; a tremendously influential model

$$\Phi(\boldsymbol{x},\boldsymbol{y}) = \frac{\exp\big(\mathrm{FEAT}(\boldsymbol{x},\boldsymbol{y})\big)}{\sum_{\boldsymbol{y}\in\mathcal{L}}\exp\big(\mathrm{FEAT}(\boldsymbol{x},\boldsymbol{y})\big)}$$

Let $Z(\boldsymbol{x};\theta) = \sum_{\boldsymbol{y}\in\mathcal{L}}\exp\big(\mathrm{FEAT}(\boldsymbol{x},\boldsymbol{y})\big)$

Then $-\log\Phi(\boldsymbol{x},\boldsymbol{y}) = -\mathrm{FEAT}(\boldsymbol{x},\boldsymbol{y}) + \log Z$

The learning problem of sequence labeling become:

$$\theta = \arg\min_{\theta} \sum_{i=1\ldots|D|} -\mathrm{FEAT}(\boldsymbol{x}_{\boldsymbol{i}},\boldsymbol{y}_{\boldsymbol{i}};\theta) + \log Z(\boldsymbol{x}_{\boldsymbol{i}};\theta)$$

CRFs: Calculating the loss

- Choice for FEAT: naturally, we can think about:

$$\mathrm{FEAT}(\boldsymbol{x}, \boldsymbol{y}; \theta) = \sum_{i=0}^{n} \mathrm{feat}(\boldsymbol{x}, y_i, y_{i+1}; \theta)$$

- Then it holds that

$$\exp\big(\mathrm{FEAT}(\boldsymbol{x}, \boldsymbol{y}; \theta)\big) = \prod_{i=0}^{n} \exp\big(\mathrm{feat}(\boldsymbol{x}, y_i, y_{i+1}; \theta)\big)$$

- and therefore:

$$Z(\boldsymbol{x}; \theta) = \sum_{y \in \mathcal{L}} \prod_{i=0}^{n} \exp\big(\mathrm{feat}(\boldsymbol{x}, y_i, y_{i+1}; \theta)\big)$$

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

How to calculate $Z(\boldsymbol{x}; \theta)$

- Good news! The algorithm that gives us Z is almost exactly like the Viterbi algorithm.
- **Forward algorithm**: sums the $\exp(\text{FEAT})$ values for all label sequences, given x, in the same asymptotic time and space as Viterbi.

- Let $\alpha_i(y)$ be the sum of all $\exp(\text{feat})$ scores of label prefixes of length i, ending in y
- Turns the "scary sum over big product" to "+ × + × + × …"
- Just like Viterbi turns the "scary max over big sum" to "max plus max plus …"

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Forward Algorithm

- **Input**: sequence x, feature function $\text{feat}(\boldsymbol{x}, y_i, y_{i+1}; \theta)$
- **Output**: $Z(\boldsymbol{x}; \theta)$

- Base case: $\alpha_1(y) = \exp\big(\text{feat}(\boldsymbol{x}, \langle \text{BOS} \rangle, y; \theta)\big)$
- For $i \in \{2, \ldots, n+1\}$:
  - Solve for $\alpha_i(*)$ by: at $n+1$, only need to solve $\alpha_i(\langle \text{EOS} \rangle)$

$$\alpha_i(y) = \sum_{y_{i-1} \in \mathcal{L}} \exp\big(\text{feat}(\boldsymbol{x}, y_{i-1} y; \theta)\big) \times \alpha_{i-1}(y_{i-1})$$

- Return $\alpha_{n+1}(\langle \text{EOS} \rangle)$, which is equal to $Z(\boldsymbol{x}; \theta)$

Adapted from: https://nasmith.github.io/NLP-winter23/calendar/

Recap

- LSTM can do a lot work
  - Language modeling, machine translation, sequence labeling etc.
- LSTM is a very power **encoder** and **decoder** of sequences
- Sequence labeling is a classical problem including but not limited to language tasks

To-Do List

- Read Chapter 17 of SLP3 - Context-Free Grammars and Constituency Parsing

References

- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.
- Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10), 2451-2471.
- Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006, June). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning* (pp. 369-376).
- Britz, D., Goldie, A., Luong, M. T., & Le, Q. (2017). Massive exploration of neural machine translation architectures. arXiv preprint arXiv:1703.03906.
- Wiseman, S., & Rush, A. M. (2016). Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*.
- Lafferty, J., McCallum, A., & Pereira, F. (2001, June). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In ICML (Vol. 1, No. 2, p. 3).