CS310 Natural Language Processing
自然语言处理
Lecture 14 - Course Review

Instructor: Yang Xu

主讲人：徐炀

xuyang@sustech.edu.cn

A Historic View of NLP

Stage 1 （196X ~ 8X）

Early NLP

Rule-based

Stage 2 （199X）

Statistical NLP

Statistics-based
Manual feature extraction
Annotated data
Machine learning

Stage 3 (2008 ~ 2022)

Neural-Network-based NLP

Deep neural network-based
Automatic feature extraction
Deep learning

2022 onward: LLMs

Table of Content   Lecture 02 - Word Vectors

- Motivation
- Documents and Counts-based Method
- Neural Network-based Method -- word2vec
- Evaluation and Applications

Build Word-Document Matrix (term-document matrix)[1]

- Build matrix $\mathbf{A} \in \mathbb{R}^{V \times C}$, which contains the count of each word in each document
- **Example**:

x_1: 学 而 时 习 之
x_2: 学 而 不 思 则 罔
x_3: 思 而 不 学 则 殆

Entry $\mathbf{A}_{v,c} = \text{count}_{x_c}(v)$, count of word v in the cth document

C (columns x_1–x_3), V (rows)

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

[1] https://en.wikipedia.org/wiki/Term-document_matrix

Pointwise Mutual Information

$$PMI = [\mathbf{A}]_{v,c} = \left[\log \frac{\text{count}_{\boldsymbol{x}_c}(v)}{\frac{\text{count}_{\boldsymbol{x}}(v)}{N} \cdot \ell_c}\right]_+$$

- If a word v has nearly same frequency in every document, then its row $[\mathbf{A}]_{v,*}$ will be nearly all zeros
- If a word v only occurs in one document c, then its PMI will be large and positive
- Thus, PMI is sensitive to rare words; usually need to smooth the frequencies by filtering rare words

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

Adapted from: https://nasmith.github.io/NLP-winter23/assets/slides/vectors.pdf

Improvement: Latent Semantic Analysis

(Deerwester et al., 1990)

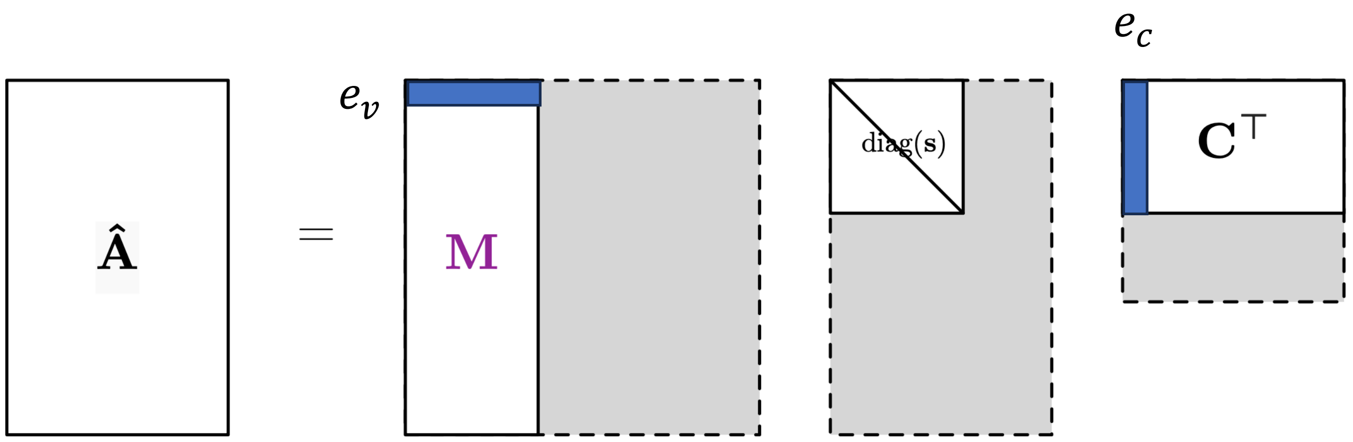
- LSA seeks to find a more compact (low rank) representation of document-word matrix $\mathbf{A}$

$$\underset{V \times C}{\mathbf{A}} \approx \hat{\mathbf{A}} = \underset{V \times d}{\mathbf{M}} \times \underset{d \times d}{\operatorname{diag}(\mathbf{s})} \times \underset{d \times C}{\mathbf{C}^{\top}}$$

- Can be solved by applying singular value decomposition to $\mathbf{A}$, and then truncating to d dimensions ($\hat{\mathbf{A}}$)
- $\mathbf{M}$ contains left singular vectors of $\mathbf{A}$
- $\mathbf{C}$ contains right singular vectors of $\mathbf{A}$
- $\mathbf{s}$ are singular values of $\mathbf{A}$

Truncated SVD => word vectors

$$\mathbf{A} \approx \widehat{\mathbf{A}} = \mathbf{M} \times \mathrm{diag}(\mathbf{s}) \times \mathbf{C}^\top$$

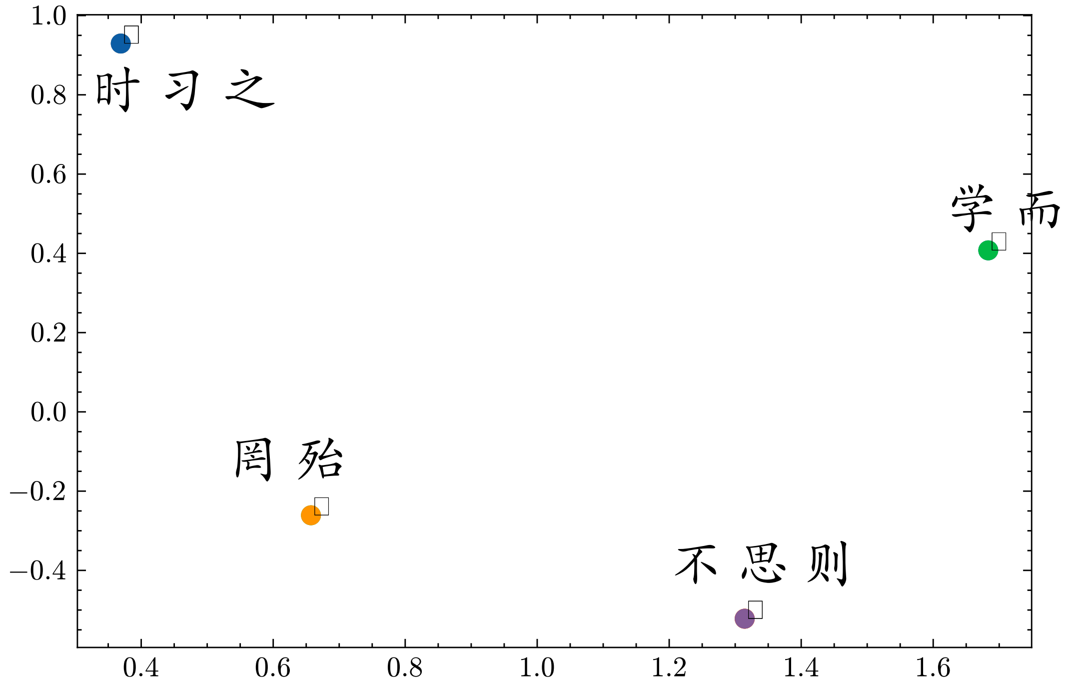


- vth column in $\mathbf{M}$ is the embedding vector for word v
- cth column in $\mathbf{C}$ is the embedding vector for document c

- $\mathbf{M}$ contains useful word vectors ("embeddings") of d dimensions
- $\mathbf{C}$ contains document vectors

LSA Example $d = 2$

- Word vectors $\mathbf{M}$ plotted
- Note that some words are in the same spot. Why?



$\mathbf{M} =$

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

$\mathbf{A} =$

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

Overview

Lecture 12 - Question Answering

- **Question Answering (QA)**
  - What is QA?
  - Information Retrieval; Tf-idf
  - Retriever-based QA; Datasets
    - Answer Span Extraction
    - Retrieval-Augmented Generation

Brief Overview of Information Retrieval (IR)

- **Information retrieval, IR**: Retrieval of all kinds of media based on user information needs. IR system ≈ **search engine**
- We focus on **ad hoc (临时) retrieval**: a user poses a query to an IR system, which then returns an ordered set of documents from some collection

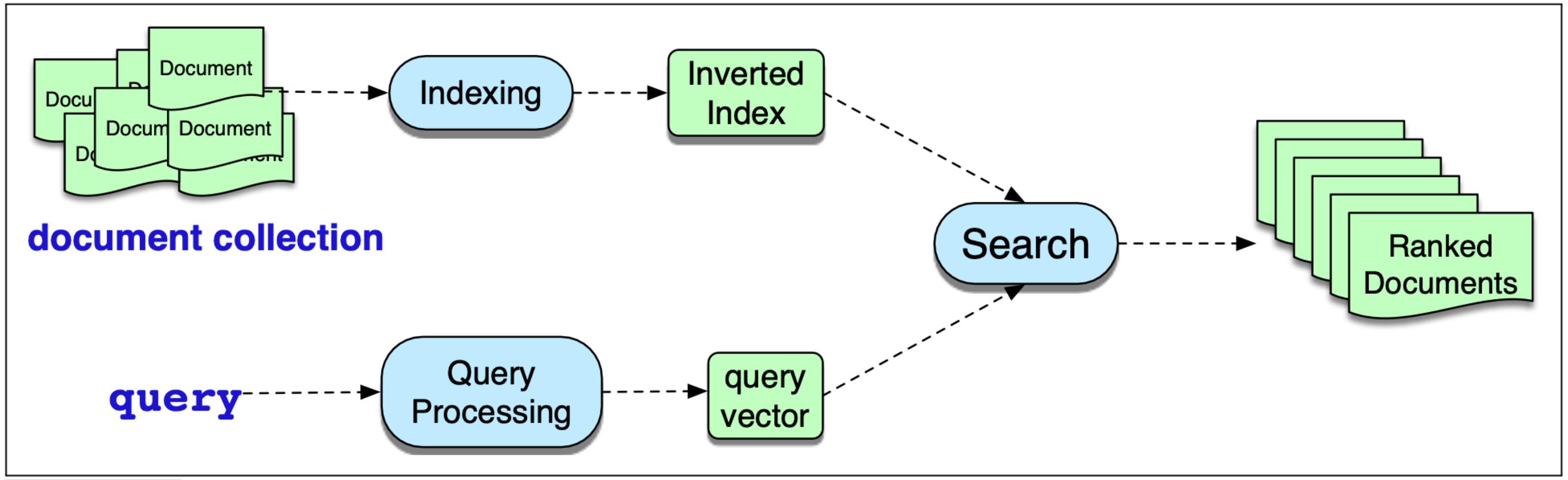


Figure 14.1 The architecture of an ad hoc IR system.

Query: a user's information need expressed as a set of **terms**

Term refers to a word/phrase in a collection of documents

Figure from SLP3, Ch 14

How to match a document a query?

- Compute a term weight for each document term
- Common method: **tf-idf** and BM25
  - **tf**: term frequency
  - **idf**: inverse document frequency
- tf-idf $\triangleq$ tf $\times$ idf (product of the two)

term t; document d

$$\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **tf**: words that occur more often in a document are likely to be informative about the document's content
- Use the $\log_{10}$ of word frequency count rather than raw count
- Why? A word appearing 100 times doesn't make it 100 times more likely

Tf-idf

$$\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

term t; document d

term occurs 0 times in document: tf = 0
term occurs 1 times in document: tf = 1
term occurs 10 times in document: tf = 2, …

- **document frequency** df_t of a term t is the number of documents it occurs in
- Terms that occur in only **a few** documents are useful for discriminating those documents from the rest of the collection;
- terms that occur across the entire collection aren't as helpful (*the, a, an, …*)
- **inverse document frequency** or **idf** is defined as:

$$\text{idf}_t = \log_{10} \frac{N}{\text{df}_t}$$

N: total number of documents
The fewer documents in which t occurs, the higher idf_t

Inverse document frequency example

- Some idf values for some words in the corpus of Shakespeare plays

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Extremely informative words that occur in only one play like *Romeo*

good or *sweet* tare completely non-discriminative since they occur in all 37 plays

Scoring with tf-idf

- We can score document d by the cosine of its vector $\vec{\boldsymbol{d}}$ with the query vector $\vec{\boldsymbol{q}}$:

$$\text{score}(q, d) = \cos\left(\vec{\boldsymbol{q}}, \vec{\boldsymbol{d}}\right) = \frac{\vec{\boldsymbol{q}} \cdot \vec{\boldsymbol{d}}}{|\vec{\boldsymbol{q}}| \cdot |\vec{\boldsymbol{d}}|}$$

- in which $\vec{\boldsymbol{q}}$ and $\vec{\boldsymbol{d}}$ are vectors of query length n, whose values are the **tf-idf** values (normalized):

$$\vec{\boldsymbol{q}} = \frac{[\text{tf} - \text{idf}(t_1, q), \dots, \text{tf} - \text{idf}(t_n, q)]}{\sqrt{\sum_{t \in q} \text{tf} - \text{idf}^2(t, q)}}$$

$$\vec{\boldsymbol{d}} = \frac{[\text{tf} - \text{idf}(t_1, d), \dots, \text{tf} - \text{idf}(t_n, d)]}{\sqrt{\sum_{t \in d} \text{tf} - \text{idf}^2(t, d)}}$$

$$\text{score}(q, d) = \sum_{t_i \in q} \frac{\text{tf} - \text{idf}(t_i, q)}{\sqrt{\sum_{t \in q} \text{tf} - \text{idf}^2(t, q)}} \cdot \frac{\text{tf} - \text{idf}(t_i, d)}{\sqrt{\sum_{t \in q} \text{tf} - \text{idf}^2(t, q)}}$$

Table of Content

Lecture 03 - Recurrent Neural Networks and Language Modeling

How to Learn an LM?

- Pre- Neural network solution: ***n*-gram Language Model**
- **Def.** An *n*-gram is a chunk of *n* consecutive words

```
the Sun rises every ______
```

- **Uni**grams (*n*=1): “the”, “Sun”, “rises”, “every”
- **Bi**grams (*n*=2): “the Sun”, “Sun rises”, “rises every”
- **Tri**grams (*n*=3): “the Sun rises”, “Sun rises every”
- **Four**-grams (*n*=4): “the Sun rises every”

- **Idea**: Count the frequencies of different n-grams and use these to predict the next word

n-grams LM: Markov assumption

Andrey Andreyevich Markov
(14 June 1856 – 20 July 1922)

- **Markov assumption**: a word at only depends on its preceding $n-1$ words

$$P\left(x^{\langle t+1\rangle} \middle| x^{\langle 1\rangle}, \dots, x^{\langle t\rangle}\right) = P\left(x^{\langle t+1\rangle} \middle| \underbrace{x^{\langle t-n+2\rangle}, \dots, x^{\langle t\rangle}}_{n-1 \text{ words}}\right)$$

Probability of a *n*-gram

Probability of a (*n*-1)-gram

$$= \frac{P(x^{\langle t-n+2\rangle}, \dots x^{\langle t\rangle}, x^{\langle t+1\rangle})}{P(x^{\langle t-n+2\rangle}, \dots x^{\langle t\rangle})}$$

By the definition of conditional probability

- **Question**: How to obtain the probabilities?
- **Answer**: By counting them from some large enough corpora (statistical approximation)

$$\approx \frac{\text{count}(x^{\langle t-n+2\rangle}, \dots x^{\langle t\rangle}, x^{\langle t+1\rangle})}{\text{count}(x^{\langle t-n+2\rangle}, \dots x^{\langle t\rangle})}$$

A Historic View of NLP

Stage 1 （196X ~ 8X）

Early NLP

Rule-based

Stage 2 （199X）

Statistical NLP

Statistics-based
Manual feature extraction
Annotated data
Machine learning

Stage 3 (2008 ~ 2022)

Neural-Network-based NLP

Deep neural network-based
Automatic feature extraction
Deep learning

2022 onward: LLMs

Overview

- Context Free Grammars (CFG)
- Constituency Parsing
- Neural Constituency Parsing

Lecture 05 - Context-Free Grammars and Constituency Parsing

Context-Free Grammars (CFG)

Wundt, German psychologist
Chomsky, American linguist
Backus & Naur, American computer scientists (FORTRAN, ALGO)

- Also called Phrase-Structure Grammar
  - Wilhelm Wundt (1900), Chomsky (1956), Backus (1959), Naur (1963)
- A set of **rules** or **productions** that express the ways symbols can be grouped and ordered together.

Ex.

```
     NP → Det Nominal
     NP → ProperNoun
Nominal → Noun | Nominal Noun
```

A **NP** (**noun phrase**) can be composed of

- either a *ProperNoun* (专有名词)
- or a determiner (*Det* 冠词) followed by a *Nominal* (名词性)

A Nominal can be one or more *Nouns* (名词)

Rules can be hierarchically embedded

Ex.

```
     NP → Det Nominal                          Det → a
     NP → ProperNoun                           Det → the
Nominal → Noun | Nominal Noun                 Noun → | flight
```

Two classes of symbols: **terminal** and **non-terminal**

- **terminal:** symbols that are actual words in language (“a”, “the”, …)
- **non-terminal**: symbols that express abstractions over terminals

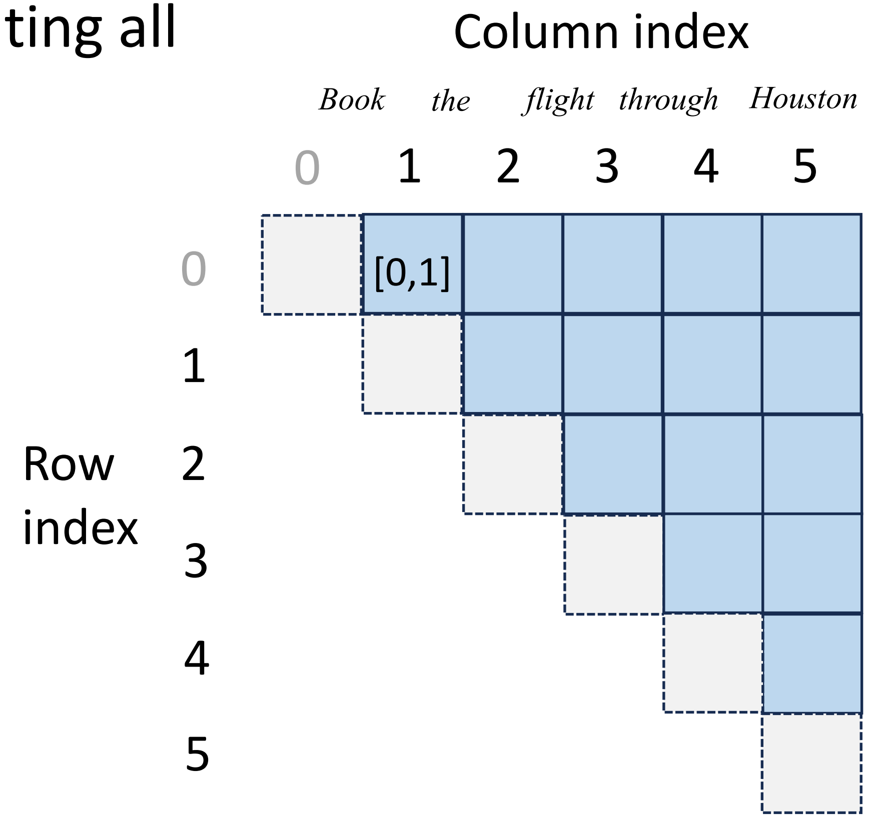
CKY Recognition

- For a sentence of length n, work with an $(\boldsymbol{n}+\mathbf{1})\times(\boldsymbol{n}+\mathbf{1})$ **matrix** $\boldsymbol{m}$, using the upper triangular portion ⇒ **parse table**
- Cell $m[i,j]$ contains the set of non-terminals representing all constituents that span position i through j.

Index: $_0$ *Book* $_1$ *the* $_2$ *flight* $_3$ *through* $_4$ *Houston* $_5$

Ex. Cell $m[0,3]$ contains the set of non-terminals for all constituents spanning 0 to 3

Task: Fill in the parse table correctly in a bottom-up way



CKY Recognition: Task breakdown

- Because grammar is in **CNF** ⇒ each non-terminal entry has exactly **two children**.
- For each constituent represented by entry $[i, j]$, there must be a position in the input, k, where it can be split into two parts $(i < k < j)$
- The first component $[i, k]$ is to the left of $[i, j]$, at the same row i
- The second component $[k, j]$ is beneath $[i, j]$, at the same column j

The smallest problems are $[j-1, j]$, i.e., spans of single word

- Then, $[i, j]$ is the **combination** of $[i, k]$ and $[k, j]$ for all possible k

In order to solve the **problem**, we need to solve **smaller problems** first

- Very typical dynamic programming approach - *optimal substructure*

CKY Recognition: Example

Index: 0 *Book* 1 *the* 2 *flight* 3

Solve span [0, 1]

	Book 1	*the* 2	*flight* 3
0	S, VP, Verb, Nominal, Noun		
1			
2			
3			

5 production rules match with ***"Book"***

Store them in **cell [0,1]**

$$m[0,1] = \{S, Nominal, VP, Noun, Verb\}$$

$\mathscr{L}_1$ in CNF

- $S \rightarrow NP\ VP$
- $S \rightarrow X1\ VP$
- $X1 \rightarrow Aux\ NP$
- $S \rightarrow book \mid include \mid prefer$
- $S \rightarrow Verb\ NP$
- $S \rightarrow X2\ PP$
- $S \rightarrow Verb\ PP$
- $S \rightarrow VP\ PP$
- $NP \rightarrow I \mid she \mid me$
- $NP \rightarrow TWA \mid Houston$
- $NP \rightarrow Det\ Nominal$
- $Nominal \rightarrow book \mid flight \mid meal \mid money$
- $Nominal \rightarrow Nominal\ Noun$
- $Nominal \rightarrow Nominal\ PP$
- $VP \rightarrow book \mid include \mid prefer$
- $VP \rightarrow Verb\ NP$
- $VP \rightarrow X2\ PP$
- $X2 \rightarrow Verb\ NP$
- $VP \rightarrow Verb\ PP$
- $VP \rightarrow VP\ PP$
- $PP \rightarrow Preposition\ NP$
- $Det \rightarrow that \mid this \mid the \mid a$
- $Noun \rightarrow book \mid flight \mid meal \mid money$
- $Verb \rightarrow book \mid include \mid prefer$

Overview

Lecture 06 - Dependency Parsing

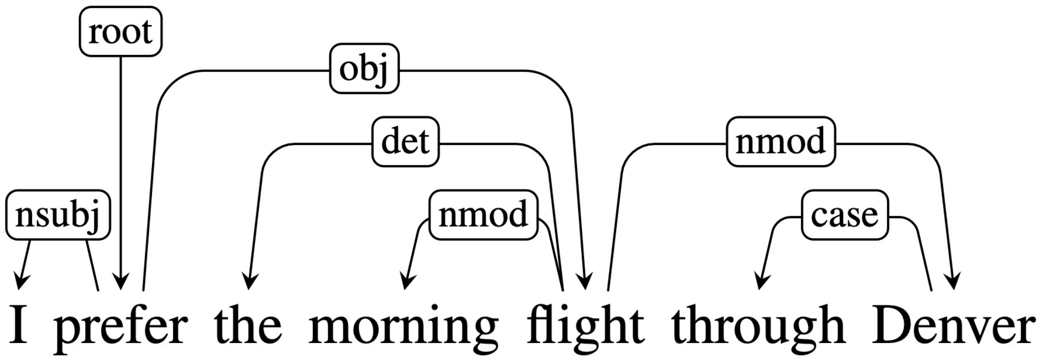
- Dependency Grammars
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluation

Dependency Grammars

- Different from context-free grammars and constituency-based representations
- **Dependency Grammars** describe syntactic structure of a sentence solely in terms of directed grammatical *relations between words*

arc: *n.* 弧线

Labeled arcs from **heads** to **dependents**



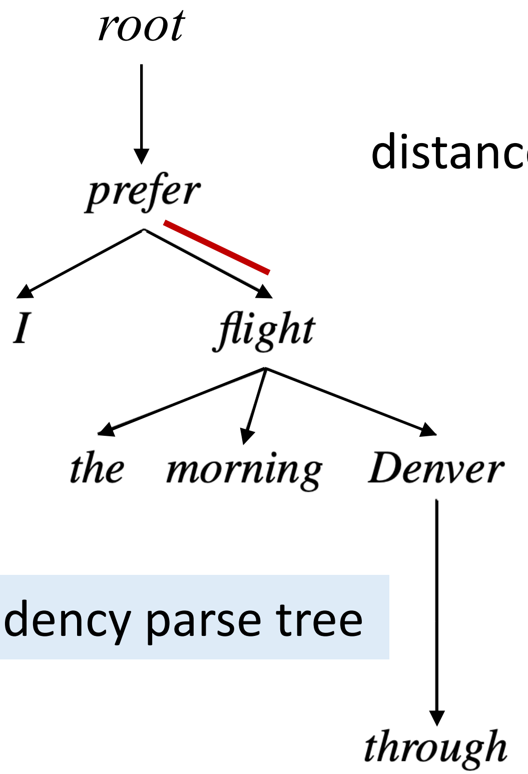
$prefer \xrightarrow{\text{nsubj}} I$ — *prefer* is the head of *I*

$prefer \xrightarrow{\text{obj}} flight$ — *flight* is the dependent of *prefer*

$root \rightarrow prefer$ — *root* is the head of *prefer* and also the head of the entire structure (root of the tree)

Compare to Context-Free Grammars

- Head-dependent relations **directly** encode important information that is often **buried** in the more complex constituency parses (by CFG)

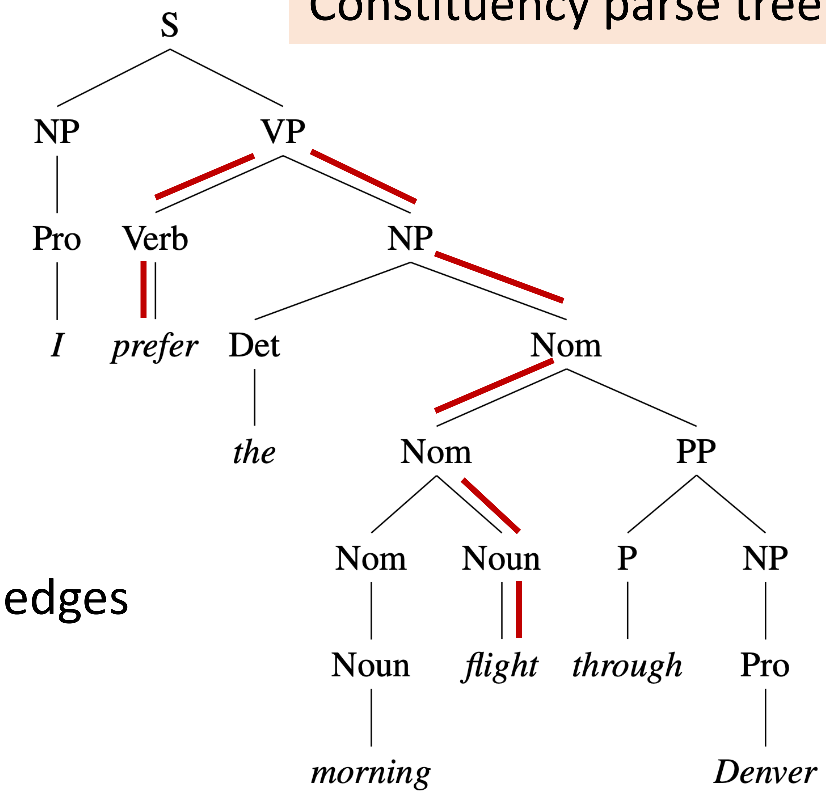


dependency parse tree

prefer flight

distance(Verb, Obj) = 1 edge

Constituency parse tree

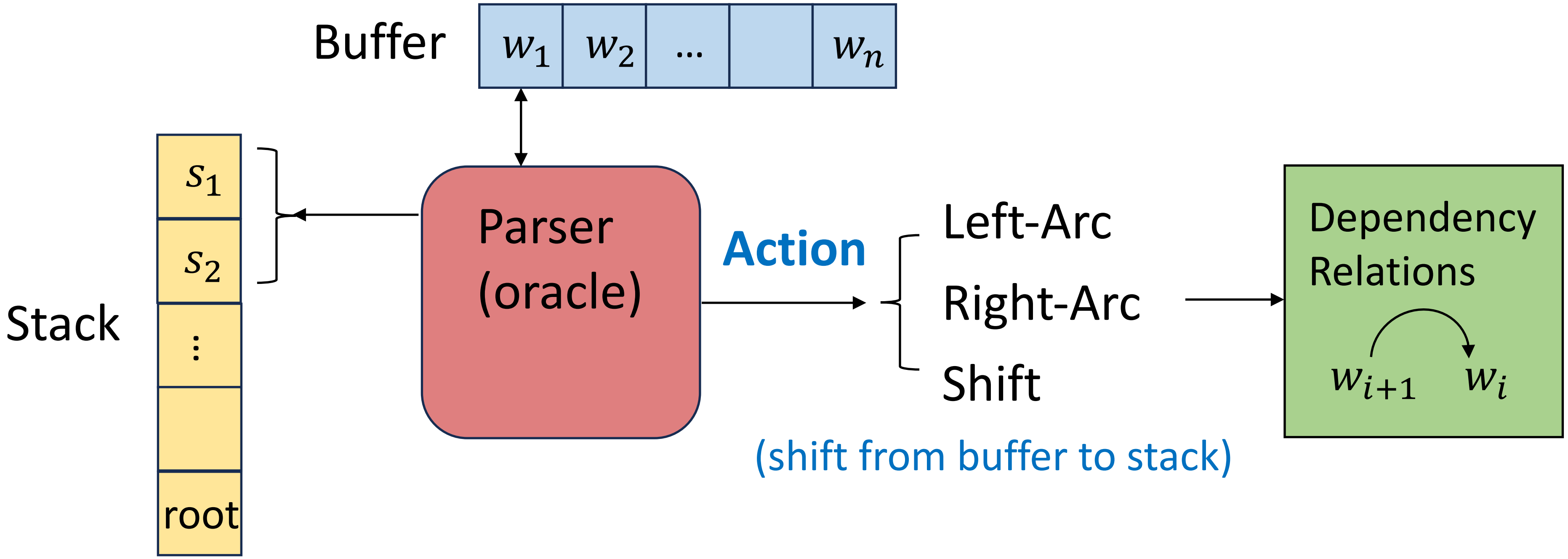


prefer flight

distance(Verb, Obj) = 7 edges

Transition-Based Dependency Parsing

- An architecture that draws on shift-reduce parsing (a paradigm for analyzing programming languages)
- Key components: **stack**, **buffer**, and **oracle**.



A Historic View of NLP

Stage 1 （196X ~ 8X）

Early NLP

Rule-based

Stage 2 （199X）

Statistical NLP

Statistics-based
Manual feature extraction
Annotated data
Machine learning

Stage 3 (2008 ~ 2022)

Neural-Network-based NLP

Deep neural network-based
Automatic feature extraction
Deep learning

2022 onward: LLMs

Table of Content

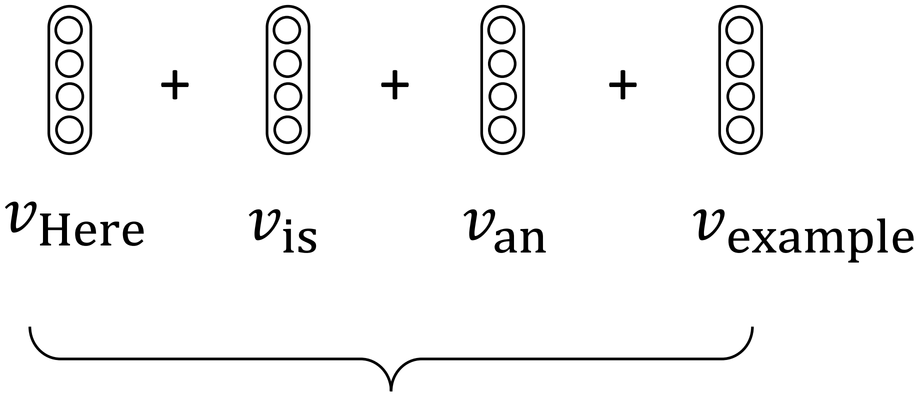
Lecture 01 - Word Vectors and Neural Networks

- **Motivation**
- Word Vectors
- Neural Networks
- Neural Text Classification

Bag-of-Words Neural Net

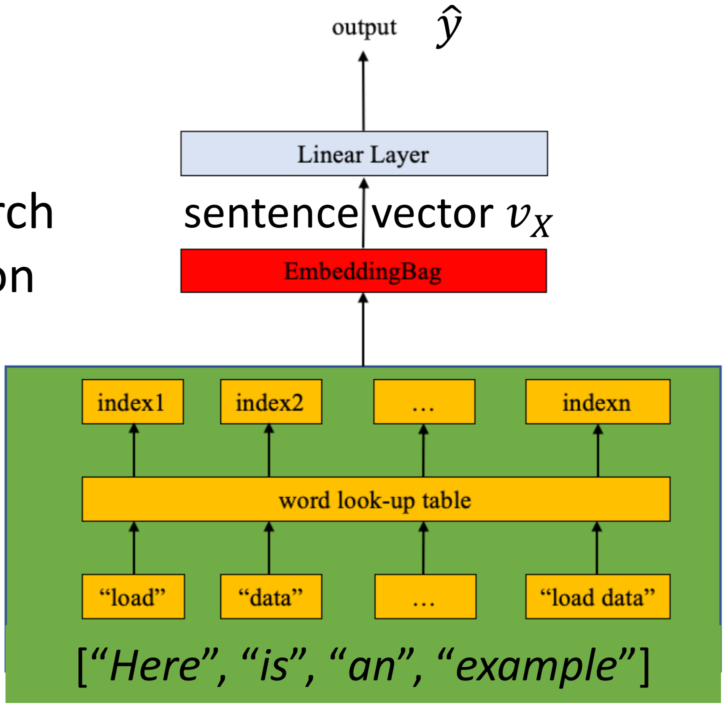
Task: News text classification

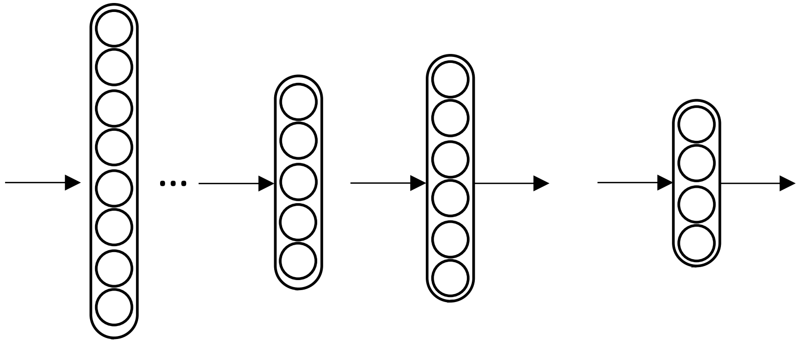
X: [*“Here”*, *“is”*, *“an”*, *“example”*]



sentence vector v_X

A typical PyTorch implementation





$$\hat{y} = \begin{bmatrix} P(\text{World}|X) \\ P(\text{Sports}|X) \\ P(\text{Business}|X) \\ P(\text{Sci/Tec}|X) \end{bmatrix}$$

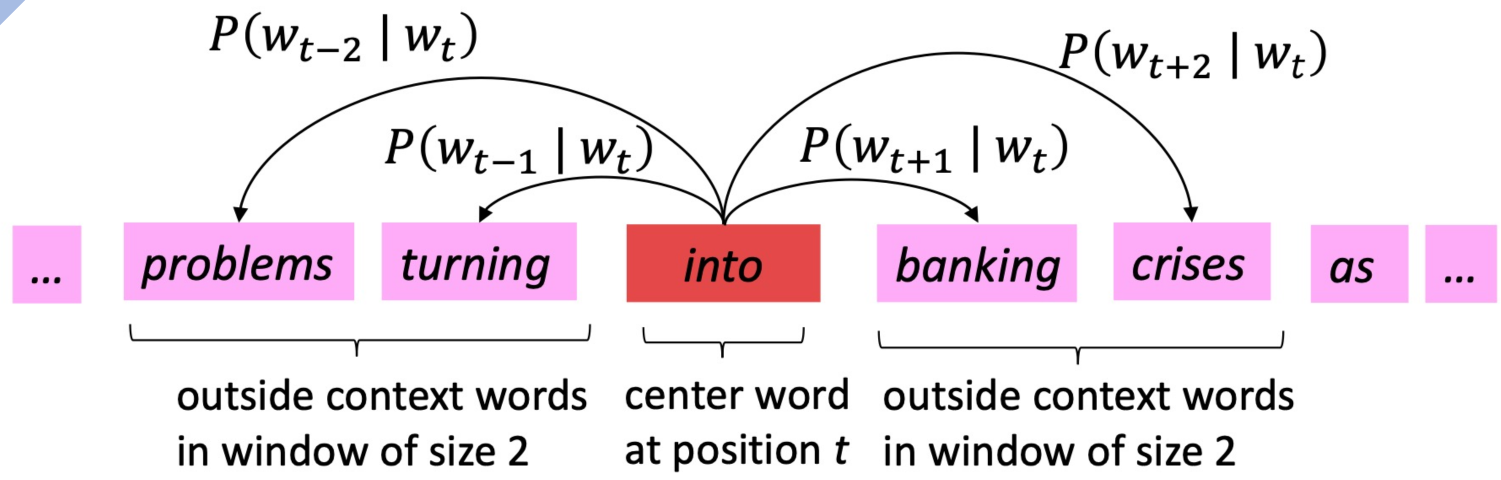
Table of Content Lecture 02 - Word Vectors

Two architectures of Word2vec

Word2vec:
- **Skip-gram**: Maximize $P(o|c)$
- Continuous Bag-of-words (**CBOW**): Maximize $P(c|o)$

- "o" for outside (context) words
- "c" for center word

Compute probability $P(w_{t+j}|w_t)$, for $j \in \{-2,-1,1,2\}$ when window size is 2



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

Negative Sampling: Objective Function

- For token at position t, maximize the log-likelihood:

Word o is the positive sample

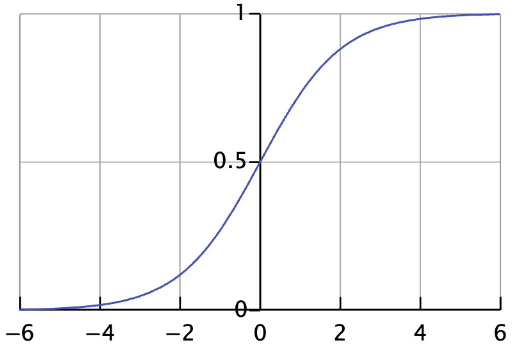
$$J_t(\theta) = \log \sigma(u_o^\top v_c) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P(w)} \left[\log \sigma(-u_{w_i}^\top v_c) \right]$$

The k words w_i $(i = 1 \dots k)$ are the negative samples

- Sigmoid function $\sigma(u_o^\top v_c)$ outputs the probability of o in the context window of c

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

a monotone increasing function



Maximizing this term will push the dot product $u_o^\top v_c$ to **larger** values, i.e., making o and c closer in semantic space

Maximizing this term will push the dot product $u_{w_i}^\top v_c$ to **smaller** values, i.e., making w_i and c farther apart in semantic space

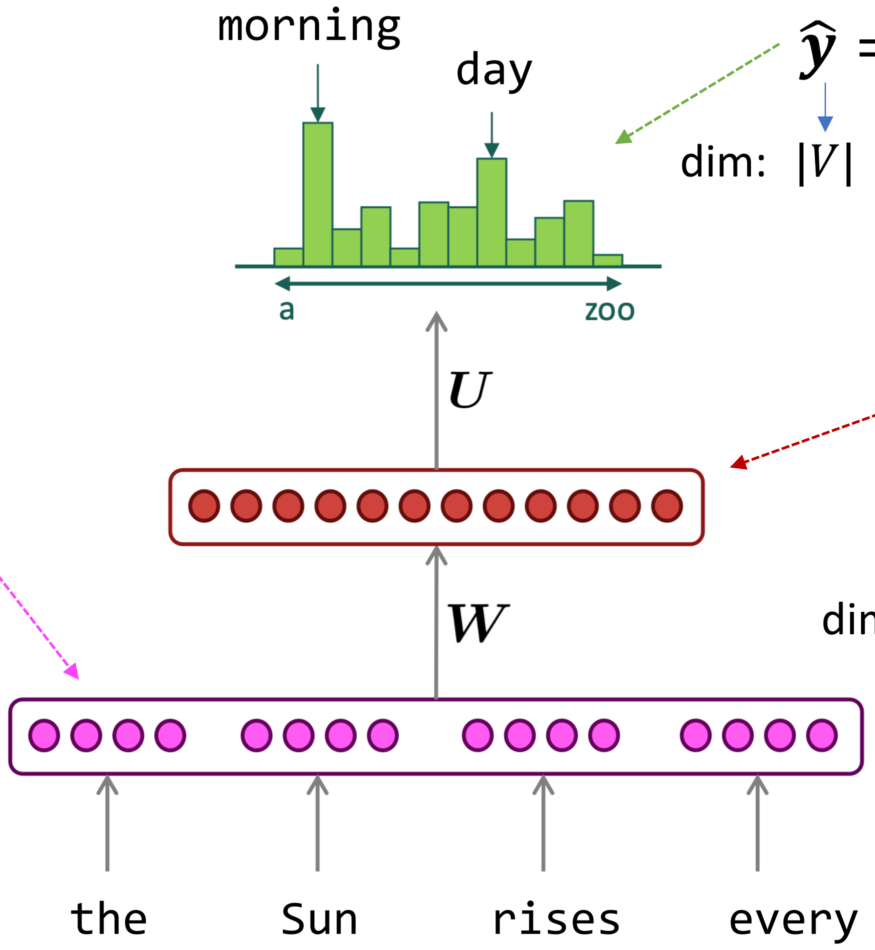
Table of Content

- **Language Modeling**
- Neural Language Models
- Recurrent Neural Networks for LM
- Evaluate LMs

Lecture 03 - Recurrent Neural Networks and Language Modeling

Fixed-Window Neural Language Model

- **Idea**: Represent words with embedding vectors; predict the next word using the concatenated embeddings from a fixed context window



output

$$\hat{\boldsymbol{y}} = \mathrm{softmax}(U\boldsymbol{h}^\top + b_2)$$

dim: $|V|$ $\qquad$ $|V| \times d'$

hidden layer

$$\boldsymbol{h} = g(W\boldsymbol{e}^\top + b_1)$$

dim: d' $\qquad$ $d' \times 4d$

concatenated word embeddings

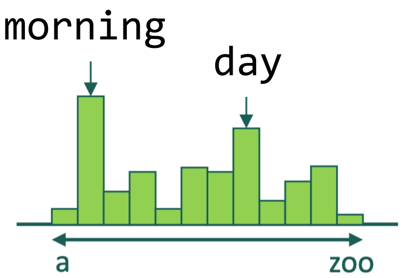
$$\boldsymbol{e} = \underbrace{\left[\boldsymbol{e}^{\langle 1\rangle}; \boldsymbol{e}^{\langle 2\rangle}; \boldsymbol{e}^{\langle 3\rangle}; \boldsymbol{e}^{\langle 4\rangle}\right]}_{4\times d} \Big\} \, d$$

Input tokens: $x^{\langle 1\rangle}, x^{\langle 2\rangle}, x^{\langle 3\rangle}, x^{\langle 4\rangle}$

(window size = 4)

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

A Simple RNN Language Model



$\hat{y}^{\langle 4 \rangle} = P(x^{\langle 5 \rangle}|\texttt{the Sun rises every})$

output (optional)

$$\hat{\boldsymbol{y}}^{\langle t \rangle} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h}^{\langle t \rangle} + b_2)$$

hidden state

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

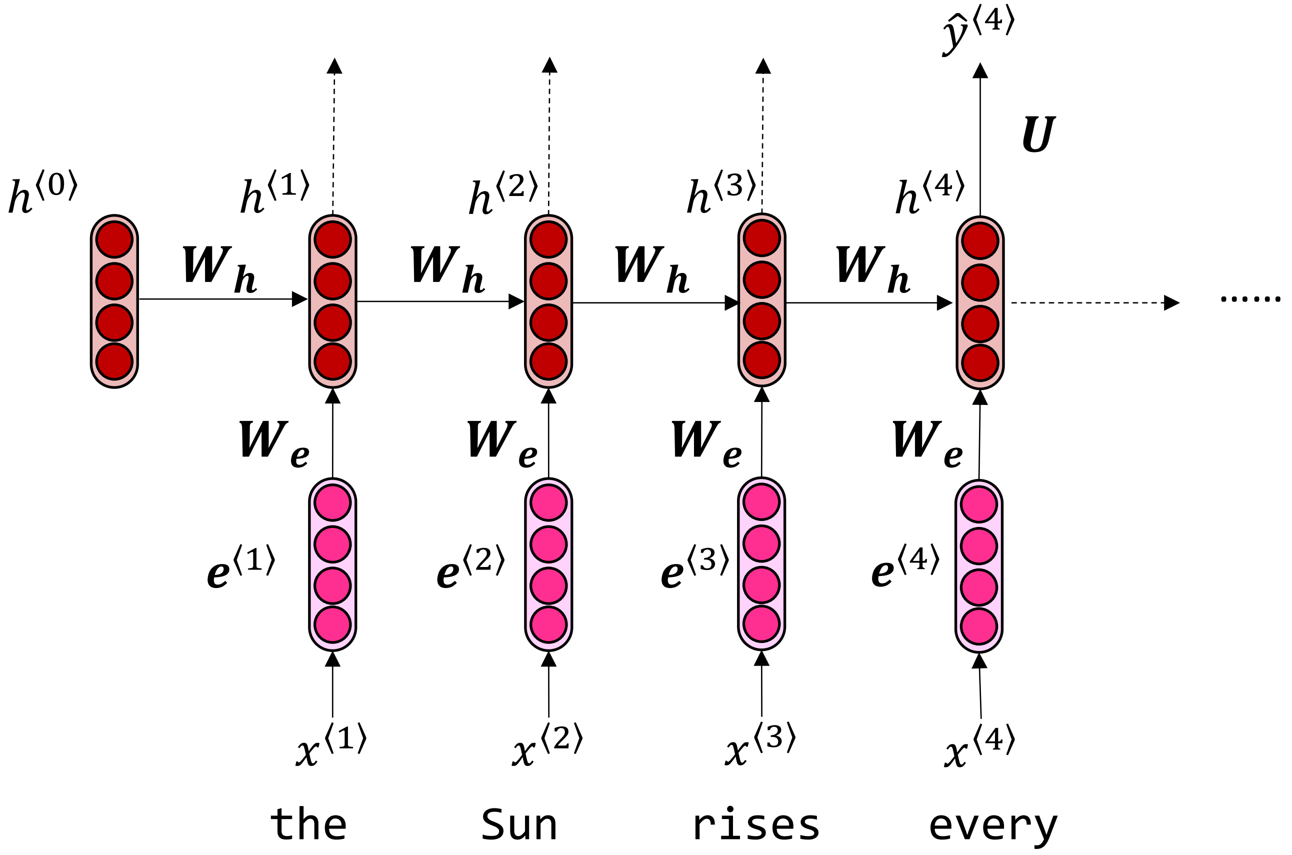
($\boldsymbol{h}^{\langle 0 \rangle}$ is the initial hidden state)

input embedding

$$\boldsymbol{e}^{\langle t \rangle} \in \mathbb{R}^d$$

input sequence

$$\boldsymbol{x}^{\langle t \rangle}$$



Overview

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

Lecture 04 - Recurrent Neural Networks and Sequence Labeling

LSTM Computational Graph

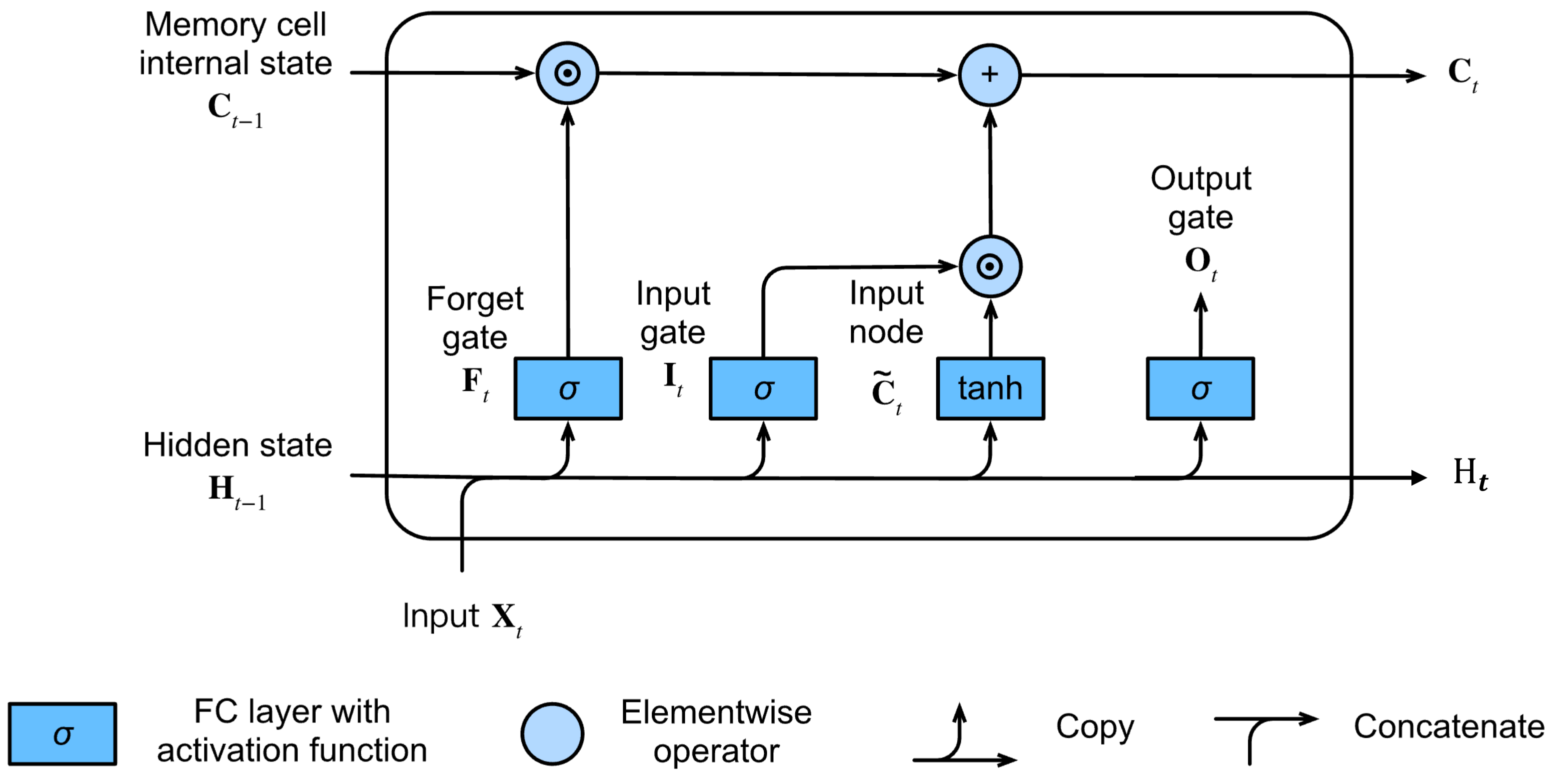


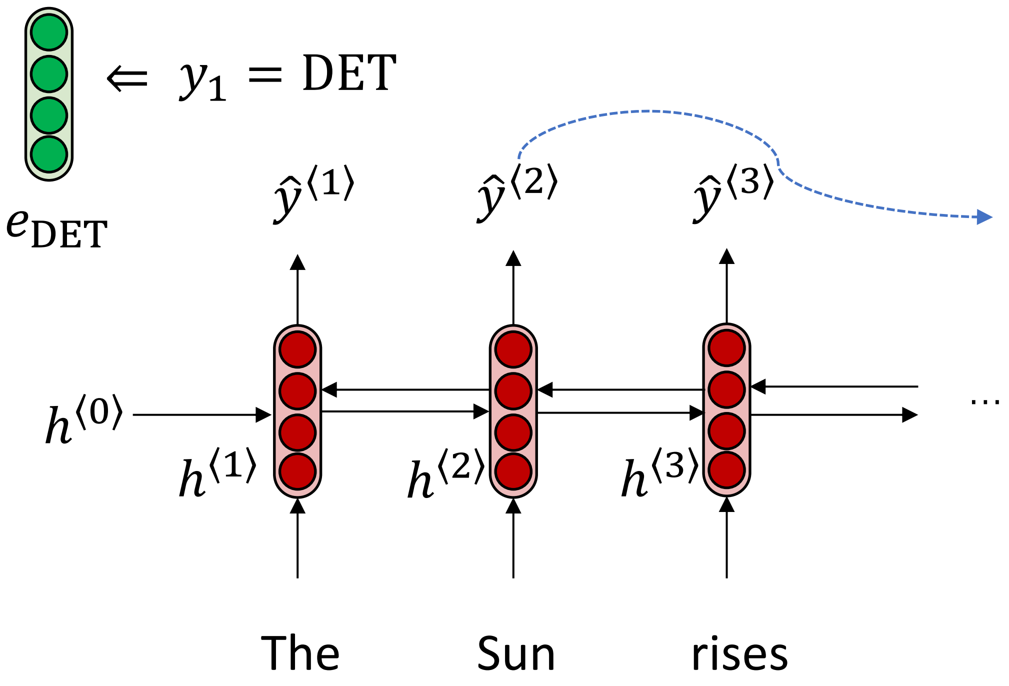
Figure from: https://d2l.ai/chapter_recurrent-modern/lstm.html

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:

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

feat() is a parameterized feature function



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

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

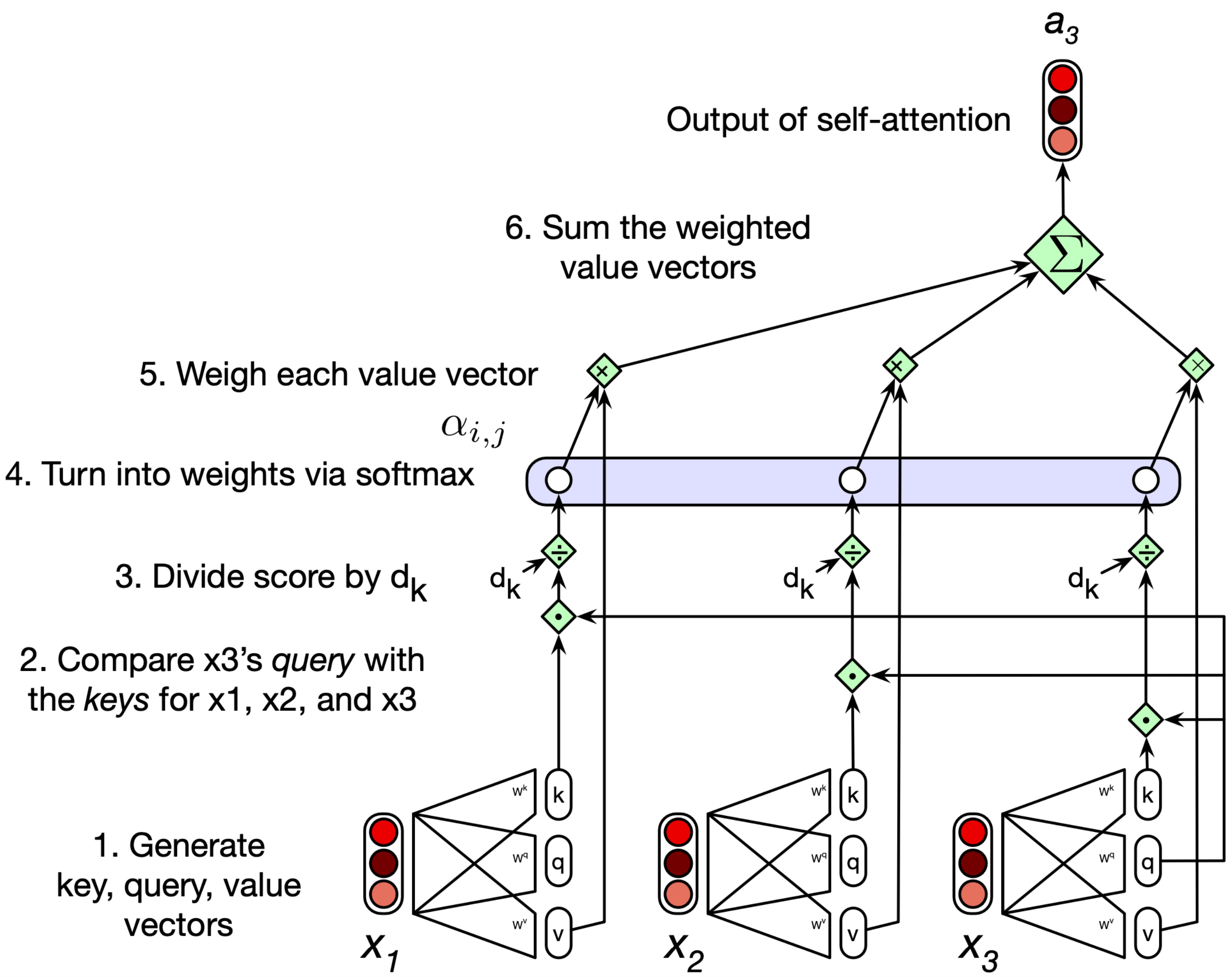
- fc() is a fully-connected layer
- $\left[\boldsymbol{h}^{\langle 2\rangle}; e_{\text{DET}}\right]$ is the concatenation of $\boldsymbol{h}^{\langle 2\rangle}$ and e_{DET}
- $\odot$ is dot product

Overview

Lecture 07 - Transformer

- Motivation
- Transformer Model
- Achievements and Drawbacks

Self-Attention Final Version



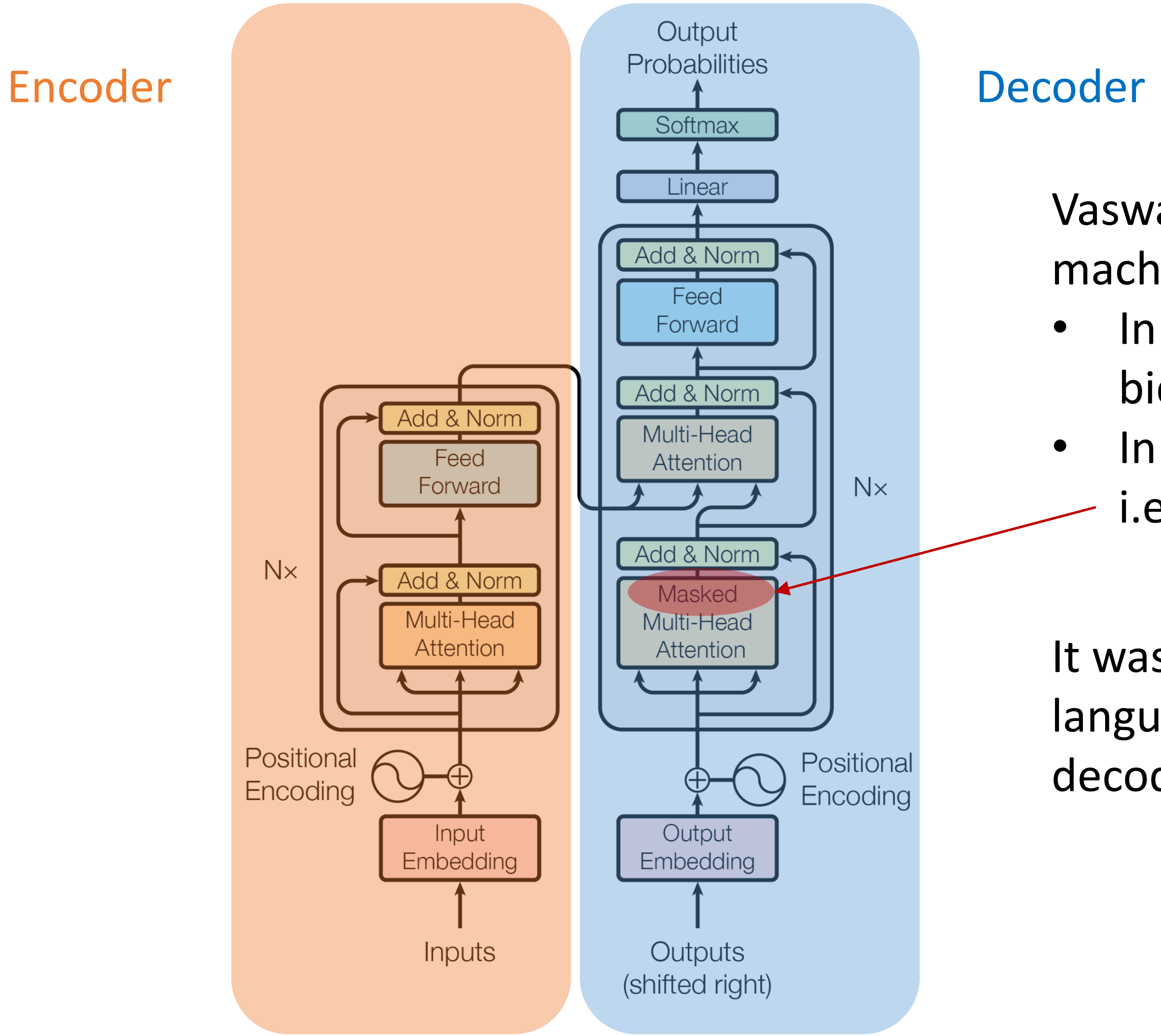
$$q_i = x_i W^Q$$
$$k_i = x_i W^K$$
$$v_i = x_i W^V$$

$$\text{score}(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}\left(\text{score}(x_i, x_j)\right)$$
$$\forall j \leq i$$

$$a_i = \sum_{j \leq i} \alpha_{ij} v_j$$

Transformer Model (encoder-decoder)



Vaswani et al. (2017)'s original work is for machine translation task

- In their encoder, the self-attention is bidirectional;
- In the decoder, the self-attention is causal, i.e., future words are masked out

It was only later that the paradigm for causal language model was defined using only the decoder part

A Historic View of NLP

Stage 1 （196X ~ 8X）

Early NLP

Rule-based

Stage 2 （199X）

Statistical NLP

Statistics-based
Manual feature extraction
Annotated data
Machine learning

Stage 3 (2008 ~ 2022)

Neural-Network-based NLP

Deep neural network-based
Automatic feature extraction
Deep learning

2022 onward: LLMs

Overview

Lecture 09 - Natural Language Generation

- Motivation: What is NLG?
- Review of Language Models
- Decoding from NLG models
- Evaluating NLG Systems

Overview

Lecture 10 - Instruction Tuning and Human Feedbacks

- Motivation
- Instruction Tuning
- Reinforcement Learning from Human Feedback (RLHF)
- Evaluation of RLHF
- What’s Next?

Overview

- Prompting
- Parameter Efficient Fine-Tuning

Lecture 11 - Prompting and Parameter Efficient Fine-Tuning