AI基础

Lecture 15: Nature Lagrange Processing

Bin Yang
School of Data Science and Engineering
byang@dase.ecnu.edu.cn

Lecture 14 ILOs

- Image Formation
    - Images without/with Lenses
    - Light, shade, color
- Simple Image Features
    - Edges, texture, optical flow, and segmentation
- Classifying Images
    - The images are of objects, scene with multiple objects
    - ImageNet, CNNs
- Detecting Objects
    - Region Proposal Network, Non-maximum suppression (NMS)
    - Bounding box regression
- The 3D World
    - 3D Clues from multiple views, from moving cameras, from a single view.
- Using Computer Vision
    - Human Behavior Understanding, Behavior Classification, Linking pictures with Words, making pictures, style transfer, autonomous driving

Lecture 15 ILOs

- Language Models
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
- The Transformer Architecture
- Pretraining

Outline

- Language Models
    - What is a language model
    - bag-of-words, n-gram
    - Part-of-speech (POS) tagging
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
- The Transformer Architecture
- Pretraining

Natural language processing (NLP)

- Three primary reasons for computers to do NLP
    - To **communicate** with humans. In many situations it is convenient for humans to use speech to interact with computers, and in most situations, it is more convenient to use natural language rather than a formal language such as first-order predicate calculus.
    - To **learn**. Humans have written down a lot of knowledge using natural language. Wikipedia alone has 30 million pages of facts such as "Bush babies are small nocturnal primates," whereas there are hardly any sources of facts like this written in formal logic. If we want our system to know a lot, it had better understand natural language.
    - To **advance the scientific understanding of languages and language use**, using the tools of AI in conjunction with linguistics, cognitive psychology, and neuroscience.

Language Models

- A Language model （LM） is a probability distribution describing the likelihood of any string.
  - “Do I dare disturb the universe?” has a reasonable probability as a string of English.
  - “Universe dare the I disturb do?” is extremely unlikely.
- Language models are at the heart of a broad range of natural language tasks.
  - We can predict what words are likely to come next in a text, and thereby suggest completions for an email or text message.
  - We can compute which alterations to a text would make it more probable, and thereby suggest spelling or grammar corrections.
  - With a pair of models, we can compute the most probable translation of a sentence.
  - With some example question/answer pairs as training data, we can compute the most likely answer to a question.

The bag-of-words model

- Recall the naïve Bayes model
  - Business or weather?

1. Stocks rallied on Monday, with major indexes gaining 1% as optimism persisted over the first quarter earnings season.

2. Heavy rain continued to pound much of the east coast on Monday, with flood warnings issued in New York City and other locations.

$$\mathbf{P}(Class|w_{1:N}) = \alpha\, \mathbf{P}(Class) \prod_j \mathbf{P}(w_j|Class).$$

  - $W_{1:N}$ means a sentence of the words $w_1, w_2, w_3, \cdots, w_N$

The bag-of-words model

- The application of naive Bayes to strings of words is called the bag-of-words model.
- It is a generative model that describes a process for generating a sentence:
  - Imagine that for each category (business, weather, etc.) we have a bag full of words (you can imagine each word written on a slip of paper inside the bag; the more common the word, the more slips it is duplicated on).
  - To generate text, first select one of the bags and discard the others.
  - Reach into that bag and pull out a word at random; this will be the first word of the sentence. Then put the word back and draw a second word.
  - Repeat until an end-of-sentence indicator (e.g., a period) is drawn.
- This model is clearly **wrong**: it falsely assumes that each word is **independent** of the others, and therefore it does not generate coherent English sentences.
  - But it does allow us to do classification with good accuracy using the naive Bayes formula: the words “stocks” and “earnings” are clear evidence for the business section, while “rain” and “cloudy” suggest the weather section.

Computing probabilities using counting

From a corpus we can estimate the prior probability of each category, $\mathbf{P}(Class)$, by counting how common each category is. We can also use counts to estimate the conditional probability of each word given the category, $\mathbf{P}(w_j|Class)$. For example, if we've seen 3000 texts and 300 of them were classified as *business*, then we can estimate

$P(Class = business) \approx 300/3000 = 0.1$. And if within the *business* category we have seen 100,000 words and the word "stocks" appeared 700 times, then we can estimate $P(stocks|Class = business) \approx 700/100,000 = 0.007$. Estimation by counting works well when we have high counts (and low variance), but we will see in Section 23.1.4 a better way to estimate probabilities when the counts are low.

$$\mathbf{P}(Class|w_{1:N}) = \alpha\, \mathbf{P}(Class) \prod_j \mathbf{P}(w_j \,\middle|\, Class).$$

Tokenization

- Note it is not trivial to decide what a word is.
- Is “aren’t” one word, or should it be broken up as “aren/’/t” or “are/n’t,” or something else?
- The process of dividing a text into a sequence of words is called tokenization.

Address the limitations of bag-of-words

- Distinguish words in different contexts
    - The word “quarter” is common in both the Business and Sports categories.
    - Four-word sequence “first quarter earnings report” is common only in Business and “fourth quarter touchdown passes” is common only in Sports.
- Two different approaches
    - Still using the bag-of-words model: treating special phrases like “first quarter earnings report” as if they were single words
    - A more principled approach is to introduce a new model, where each word is dependent on previous words.

$$P\left(w_{1:N}\right) = \prod_{j=1}^{N} P(w_j | w_{1:j-1}).$$

N-gram word models

$$P\left(w_{1:N}\right) = \prod_{j=1}^{N} P(w_j|w_{1:j-1}).$$

- With a vocabulary of 100,000 words and a sentence length of 40, this model would have 10^{200} parameters to estimate.
- Compromise with a Markov chain model that considers only the dependence between n adjacent words.
  - In an n-gram model, the probability of each word is dependent only on the previous n-1 words:

$$\begin{aligned} P(w_j|w_{1:j-1}) &= P(w_j|w_{j-n+1:j-1}) \\ P(w_{1:N}) &= \prod_{j=1}^{N} P(w_j|w_{j-n+1:j-1}). \end{aligned}$$

- P(w1, w2, w3, w4, w5, w6)
  - Unigram, 1-gram: P(w1)*P(w2)*P(w3)*P(w4)*P(w5)*P(w6)
  - Bigram, 2-gram: P(w1)*P(w2|w1)*P(w3|w2)*P(w4|w3)*P(w5|w4)*P(w6|w5)
  - Trigram, 3-gram: P(w1)*P(w2|w1)*P(w3|w1, w2)*P(w4|w2, w3)*P(w5|w3, w4)*P(w6|w4, w5)

Bigram LM

- Unigram language models face challenge in comparing "he ate pizza" and "he drank pizza“
  - P(ate) may be equal to P(drank), which requires knowledge on verb-object relations
- While Bigram language models compute conditional probabilities $P(w_1|w_2)$ for Bigrams $w_1 w_2$
  - P(pizza|ate) > P(pizza|drank)
- Training data: D consisting of a set of sentences
- Given D: MLE for the conditional probabilities:

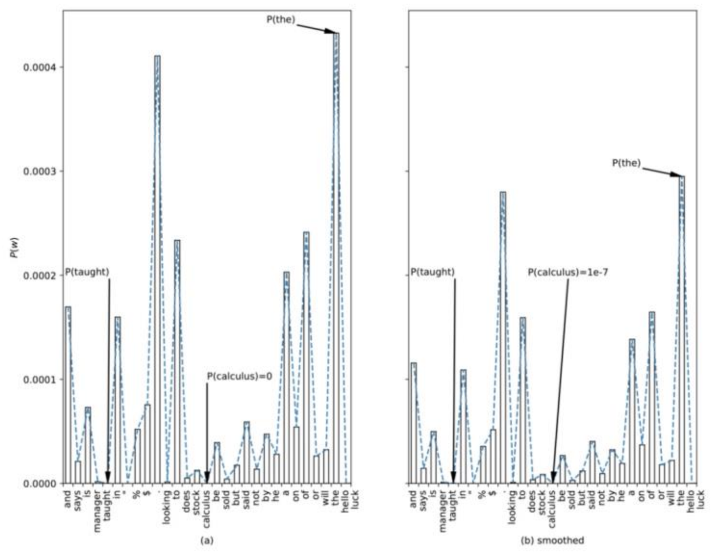
$$P\left(\mathbf{w}_2|\mathbf{w}_1\right) = \frac{(\#\mathbf{w}_1\mathbf{w}_2 \in D)}{\sum_{\mathbf{w}\in V}(\#\mathbf{w}_1\mathbf{w} \in D)}$$

Out-of-vocabulary words

- For out-of-vocabulary (OOV) words in the test data
  - not seen in the training data
  - P(OOV) = 0
  - P(S) = 0, if OOV ∈ S
- Add-one smoothing

$$P(w) = \frac{(\#\mathbf{w} \in D)+1}{\sum_{\mathbf{w}' \in V}((\#\mathbf{w}' \in D)+1)} = \frac{(\#\mathbf{w} \in D)+1}{|V|+\sum_{\mathbf{w}' \in V}(\#\mathbf{w}' \in D)}$$

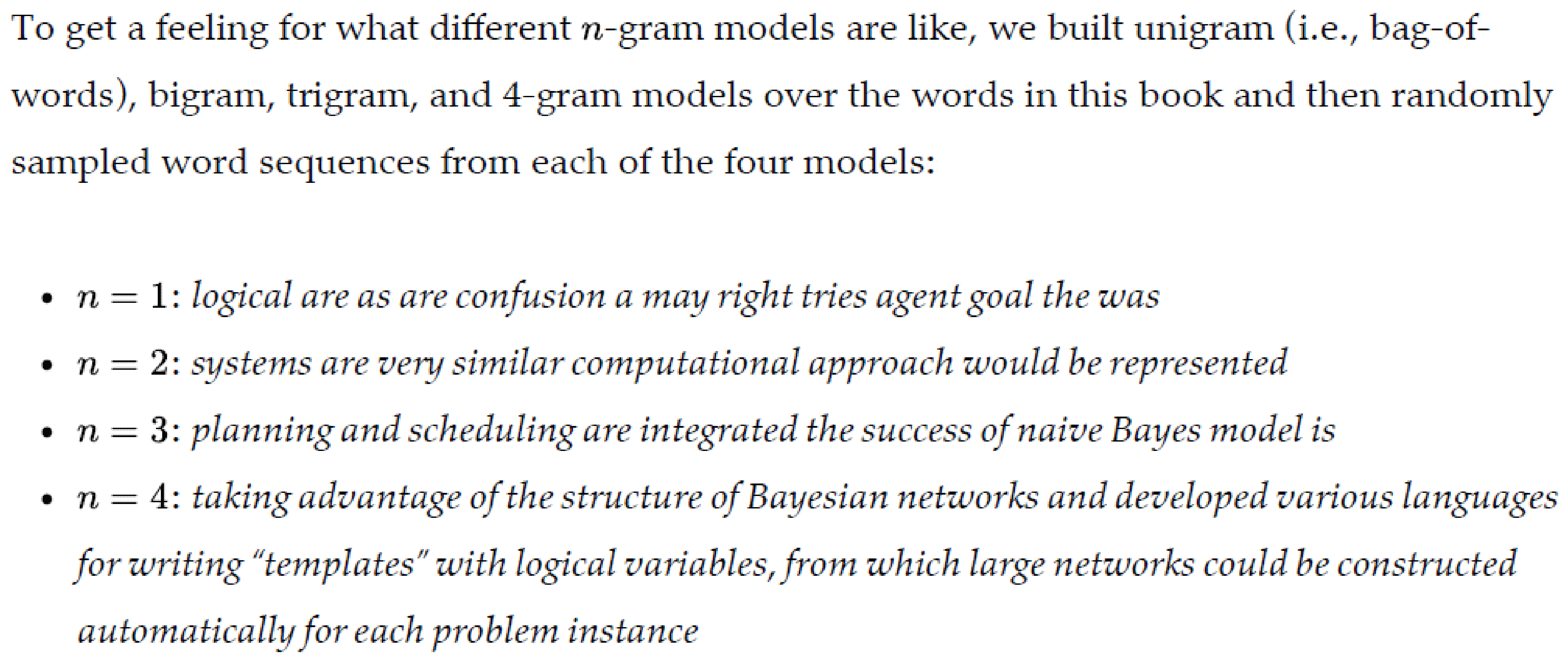
- Backoff model
  - Start by estimating n-gram counts.
  - When having a low (or zero) count, we back off to (n-1)-grams.



(a) Unigram distributions. (b) Unigram distributions with add-10 smoothing.

$$\widehat{P}\left(c_i|c_{i-2:i-1}\right) = \lambda_3 P\left(c_i|c_{i-2:i-1}\right) + \lambda_2 P\left(c_i|c_{i-1}\right) + \lambda_1 P\left(c_i\right),$$

Comparing language models



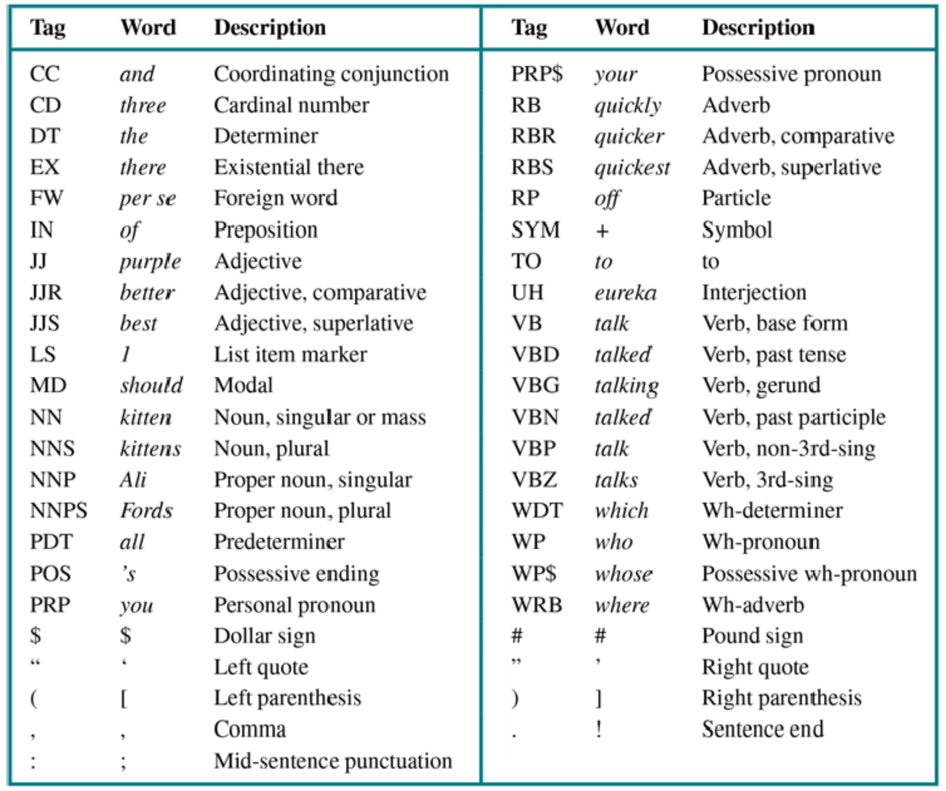
To get a feeling for what different n-gram models are like, we built unigram (i.e., bag-of-words), bigram, trigram, and 4-gram models over the words in this book and then randomly sampled word sequences from each of the four models:

- $n = 1$: *logical are as are confusion a may right tries agent goal the was*
- $n = 2$: *systems are very similar computational approach would be represented*
- $n = 3$: *planning and scheduling are integrated the success of naive Bayes model is*
- $n = 4$: *taking advantage of the structure of Bayesian networks and developed various languages for writing "templates" with logical variables, from which large networks could be constructed automatically for each problem instance*

Part-of-speech (POS) tagging

- The task of assigning a part of speech to each word in a sentence is called part-of-speech tagging.
  - One common model for POS tagging is the hidden Markov model (HMM).

From	the	start	,	it	took	a	person	with	great	qualities	to	succeed
IN	DT	NN	,	PRP	VBD	DT	NN	IN	JJ	NNS	TO	VB



Tag	Word	Description	Tag	Word	Description
CC	*and*	Coordinating conjunction	PRP$	*your*	Possessive pronoun
CD	*three*	Cardinal number	RB	*quickly*	Adverb
DT	*the*	Determiner	RBR	*quicker*	Adverb, comparative
EX	*there*	Existential there	RBS	*quickest*	Adverb, superlative
FW	*per se*	Foreign word	RP	*off*	Particle
IN	*of*	Preposition	SYM	+	Symbol
JJ	*purple*	Adjective	TO	*to*	to
JJR	*better*	Adjective, comparative	UH	*eureka*	Interjection
JJS	*best*	Adjective, superlative	VB	*talk*	Verb, base form
LS	*1*	List item marker	VBD	*talked*	Verb, past tense
MD	*should*	Modal	VBG	*talking*	Verb, gerund
NN	*kitten*	Noun, singular or mass	VBN	*talked*	Verb, past participle
NNS	*kittens*	Noun, plural	VBP	*talk*	Verb, non-3rd-sing
NNP	*Ali*	Proper noun, singular	VBZ	*talks*	Verb, 3rd-sing
NNPS	*Fords*	Proper noun, plural	WDT	*which*	Wh-determiner
PDT	*all*	Predeterminer	WP	*who*	Wh-pronoun
POS	*'s*	Possessive ending	WP$	*whose*	Possessive wh-pronoun
PRP	*you*	Personal pronoun	WRB	*where*	Wh-adverb
$	$	Dollar sign	#	#	Pound sign
“	‘	Left quote	”	’	Right quote
(	[	Left parenthesis	)	]	Right parenthesis
,	,	Comma	.	!	Sentence end
:	;	Mid-sentence punctuation			

Part-of-speech tags (with an example word for each tag) for the Penn Treebank corpus (Marcus *et al.*, 1993). Here “3rd-sing” is an abbreviation for “third person singular present tense.”

Outline

- Language Models
- Grammar and Parsing
    - Grammar, Lexicon, parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
- The Transformer Architecture
- Pretraining

Grammar

- A **grammar** is a set of rules that defines the tree structure of allowable phrases, and a **language** is the set of sentences that follow those rules.

- Probabilistic Context-Free Grammar (PCFG) is an extension of Context-Free Grammar (CFG) that attaches a probability value to each production rule. These probability values reflect the likelihood of using a specific production rule in a given context.

PCFG example for the Wumpus word

$$\begin{array}{lcll} Adjs & \rightarrow & Adjective & [0.80] \\ & | & Adjective\ Adjs & [0.20] \end{array}$$

- The syntactic Adjs category can consist of
  - either a single Adjective, with probability 0.80,
  - or of an Adjective followed by a string that constitutes an Adjs, with probability 0.20.

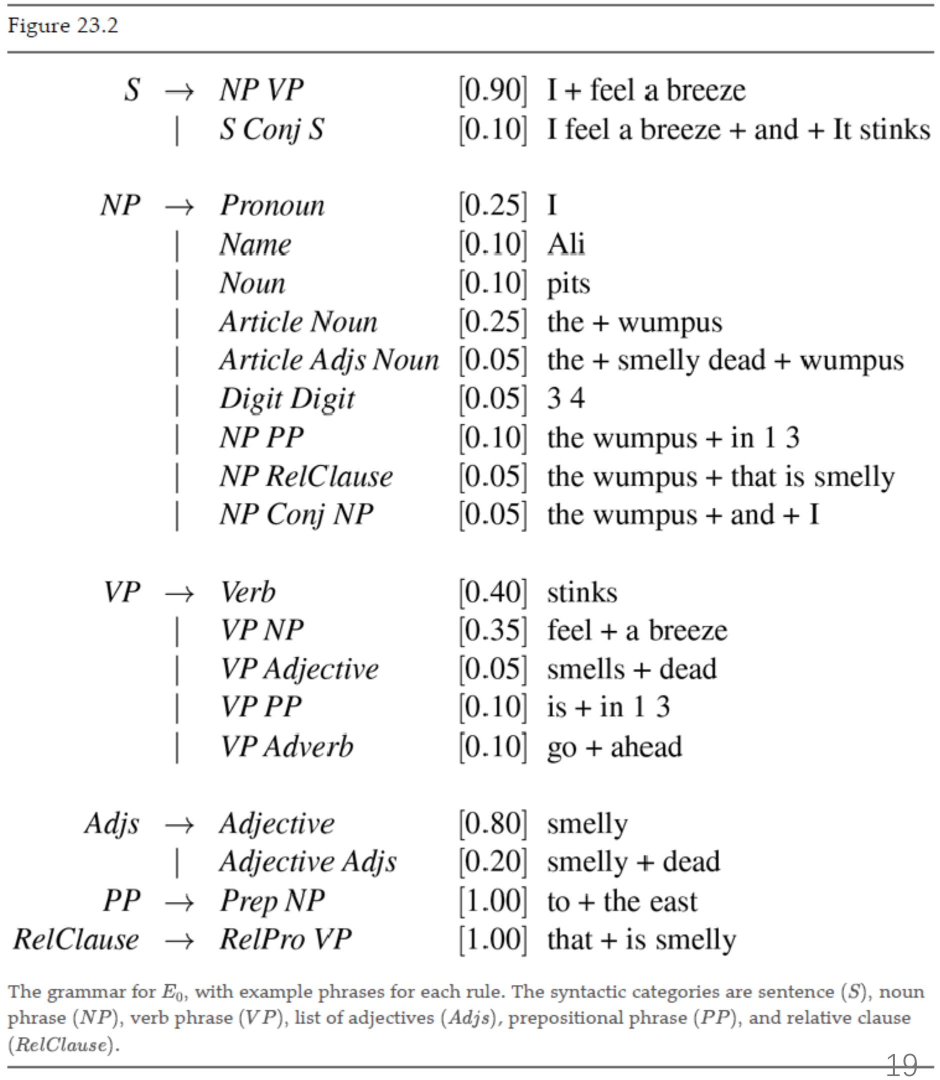


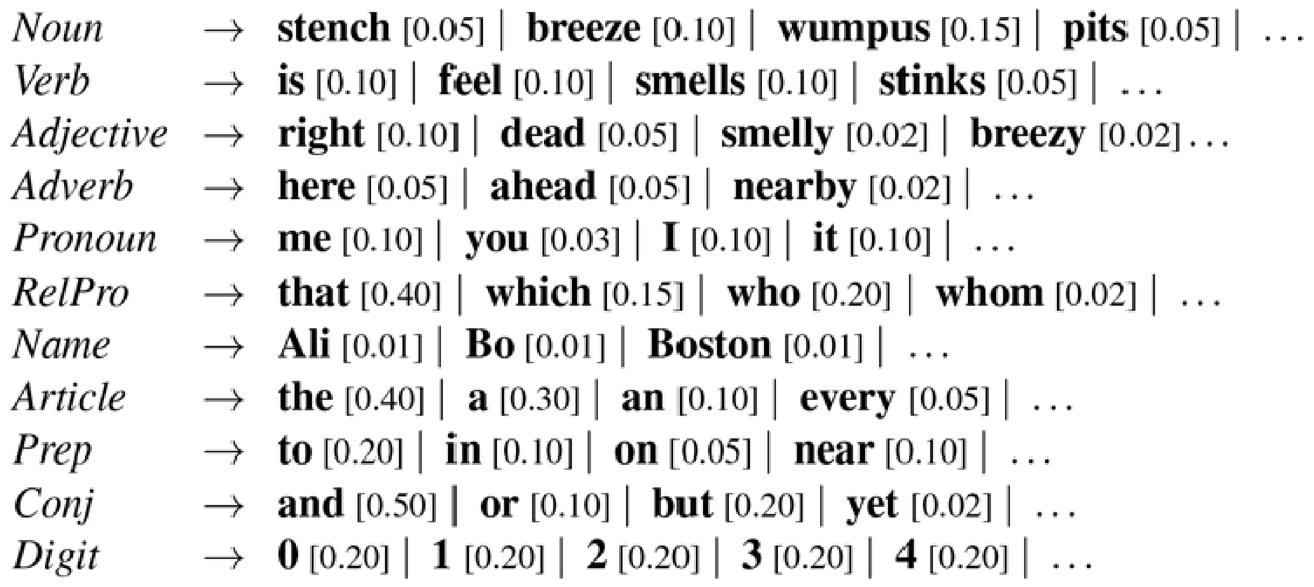
Figure 23.2

S	→	*NP VP*	[0.90]	I + feel a breeze
	|	*S Conj S*	[0.10]	I feel a breeze + and + It stinks
NP	→	*Pronoun*	[0.25]	I
	|	*Name*	[0.10]	Ali
	|	*Noun*	[0.10]	pits
	|	*Article Noun*	[0.25]	the + wumpus
	|	*Article Adjs Noun*	[0.05]	the + smelly dead + wumpus
	|	*Digit Digit*	[0.05]	3 4
	|	*NP PP*	[0.10]	the wumpus + in 1 3
	|	*NP RelClause*	[0.05]	the wumpus + that is smelly
	|	*NP Conj NP*	[0.05]	the wumpus + and + I
VP	→	*Verb*	[0.40]	stinks
	|	*VP NP*	[0.35]	feel + a breeze
	|	*VP Adjective*	[0.05]	smells + dead
	|	*VP PP*	[0.10]	is + in 1 3
	|	*VP Adverb*	[0.10]	go + ahead
Adjs	→	*Adjective*	[0.80]	smelly
	|	*Adjective Adjs*	[0.20]	smelly + dead
PP	→	*Prep NP*	[1.00]	to + the east
RelClause	→	*RelPro VP*	[1.00]	that + is smelly

The grammar for E_0, with example phrases for each rule. The syntactic categories are sentence (S), noun phrase (NP), verb phrase (VP), list of adjectives ($Adjs$), prepositional phrase (PP), and relative clause ($RelClause$).

Lexicon

- Open classes
  - Nouns, names, verbs, adjectives, and adverbs
  - It is infeasible even in principle to list all the words.
  - Not only are there tens of thousands of members in each class.
  - New ones—like humblebrag or microbiome—are being added constantly.
- Closed classes
  - Pronouns, relative pronouns, articles, prepositions, and conjunctions
  - They have a small number of words (dozen or so), and change over the course of centuries, not months.

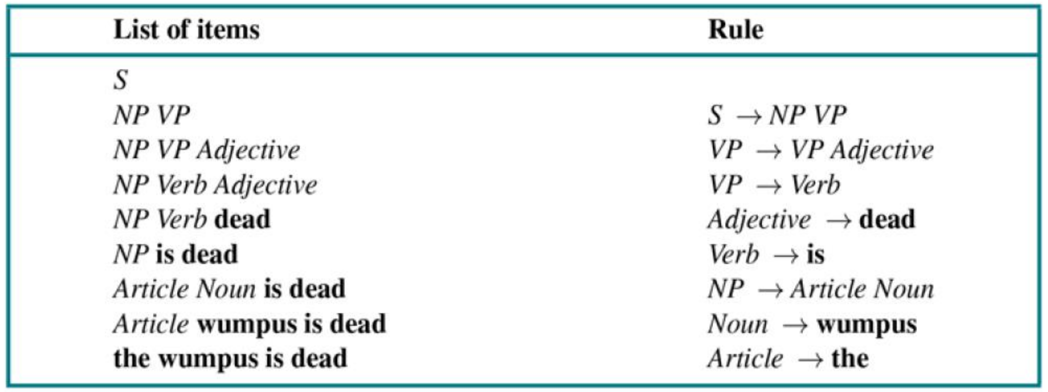


Noun	$\rightarrow$	**stench** [0.05]	**breeze** [0.10]	**wumpus** [0.15]	**pits** [0.05]	…	
Verb	$\rightarrow$	**is** [0.10]	**feel** [0.10]	**smells** [0.10]	**stinks** [0.05]	…	
Adjective	$\rightarrow$	**right** [0.10]	**dead** [0.05]	**smelly** [0.02]	**breezy** [0.02]…		
Adverb	$\rightarrow$	**here** [0.05]	**ahead** [0.05]	**nearby** [0.02]	…		
Pronoun	$\rightarrow$	**me** [0.10]	**you** [0.03]	**I** [0.10]	**it** [0.10]	…	
RelPro	$\rightarrow$	**that** [0.40]	**which** [0.15]	**who** [0.20]	**whom** [0.02]	…	
Name	$\rightarrow$	**Ali** [0.01]	**Bo** [0.01]	**Boston** [0.01]	…		
Article	$\rightarrow$	**the** [0.40]	**a** [0.30]	**an** [0.10]	**every** [0.05]	…	
Prep	$\rightarrow$	**to** [0.20]	**in** [0.10]	**on** [0.05]	**near** [0.10]	…	
Conj	$\rightarrow$	**and** [0.50]	**or** [0.10]	**but** [0.20]	**yet** [0.02]	…	
Digit	$\rightarrow$	**0** [0.20]	**1** [0.20]	**2** [0.20]	**3** [0.20]	**4** [0.20]	…

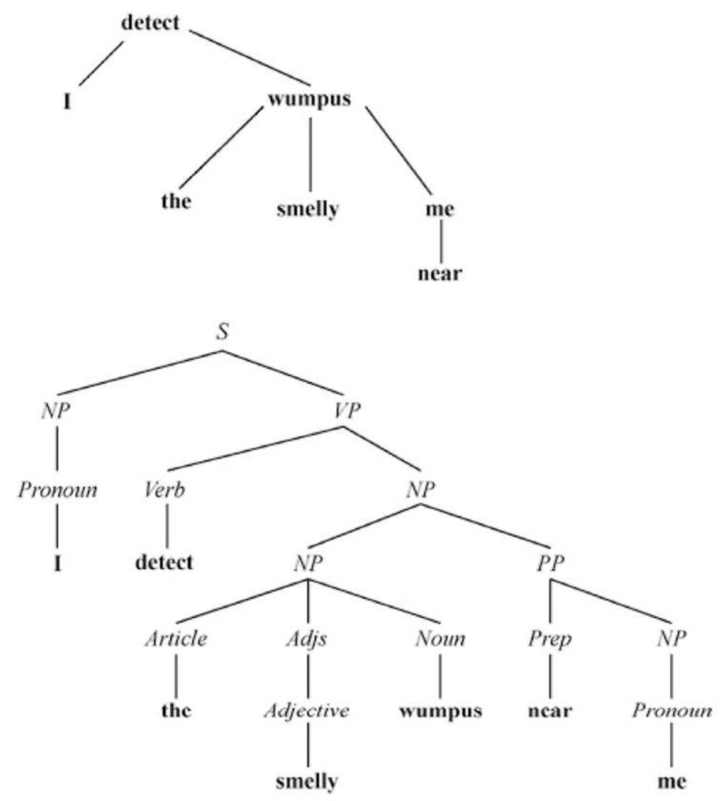
The lexicon for E_0. *RelPro* is short for relative pronoun, *Prep* for preposition, and *Conj* for conjunction. The sum of the probabilities for each category is 1.

Parsing

- Parsing is the process of analyzing a string of words to uncover its phrase structure, according to the rules of a grammar.
- Dependency parsing involves analyzing the syntactic structure of a sentence by identifying the relationships between words.
- It aims to determine the dependency relationships between words in a sentence, where one word is considered the "head" and others are its dependents.



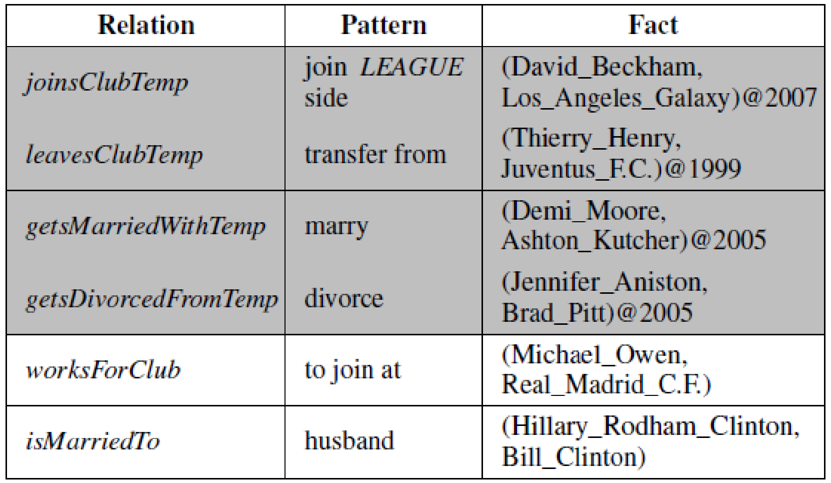
List of items	Rule
S	
NP VP	*S* → *NP VP*
NP VP Adjective	*VP* → *VP Adjective*
NP Verb Adjective	*VP* → *Verb*
NP Verb **dead**	*Adjective* → **dead**
NP **is dead**	*Verb* → **is**
Article Noun **is dead**	*NP* → *Article Noun*
Article **wumpus is dead**	*Noun* → **wumpus**
the wumpus is dead	*Article* → **the**



Outline

- Language Models
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
- The Transformer Architecture
- Pretraining

NLP tasks



Relation	Pattern	Fact
joinsClubTemp	join *LEAGUE* side	(David_Beckham, Los_Angeles_Galaxy)@2007
leavesClubTemp	transfer from	(Thierry_Henry, Juventus_F.C.)@1999
getsMarriedWithTemp	marry	(Demi_Moore, Ashton_Kutcher)@2005
getsDivorcedFromTemp	divorce	(Jennifer_Aniston, Brad_Pitt)@2005
worksForClub	to join at	(Michael_Owen, Real_Madrid_C.F.)
isMarriedTo	husband	(Hillary_Rodham_Clinton, Bill_Clinton)

- **Speech recognition** is the task of transforming spoker
  - We can then perform further tasks (such as question answe
- **Text-to-speech** synthesis is the inverse process—going from text to sound.
  - The challenge is to pronounce each word correctly, and to make the flow of each sentence seem natural, with the right pauses and emphasis.
- **Machine translation** transforms text in one language to another.
  - Systems are usually trained using a bilingual corpus: a set of paired documents, where one member of the pair is in, say, English, and the other is in, say, French.
  - The documents do not need to be annotated in any way; the machine translation system learns to align sentences and phrases and then when presented with a novel sentence in one language, can generate a translation to the other.
- **Information extraction** is the process of acquiring knowledge by skimming a text and looking for occurrences of particular classes of objects and for relationships among them.
  - A typical task is to extract instances of addresses from Web pages, with database fields for street, city, state, and zip code.
  - Relationships among entities in Wikipedia.

Yafang Wang, Bin Yang, Lizhen Qu, Marc Spaniol, Gerhard Weikum: Harvesting facts from textual web sources by constrained label propagation. CIKM 2011: 837-846

NLP tasks

- **Information retrieval** is the task of finding documents that are relevant and important for a given query.
    - Internet search engines such as Google and Baidu perform this task billions of times a day.
- **Question Answering** is a different task, in which the query really is a question, such as “Who founded the U.S. Coast Guard?” and the response is not a ranked list of documents but rather an actual answer: “Alexander Hamilton.”
    - There have been question-answering systems since the 1960s that rely on syntactic parsing as discussed in this chapter, but only since 2001 have such systems used Web information retrieval to radically increase their breadth of coverage.
    - It would search for queries such as [∗ founded the U.S. Coast Guard] and [the U.S. Coast Guard was founded by ∗]

NLP tasks

- Image Caption: describe the given image with vairable-length sequence
- Text Generation: writing a novel
- Text Classification: public opinion analysis
- Machine Translation: Google Translation
- Dialogue System: ChatGPT
- Text Similarity Calculation: find plagiarism

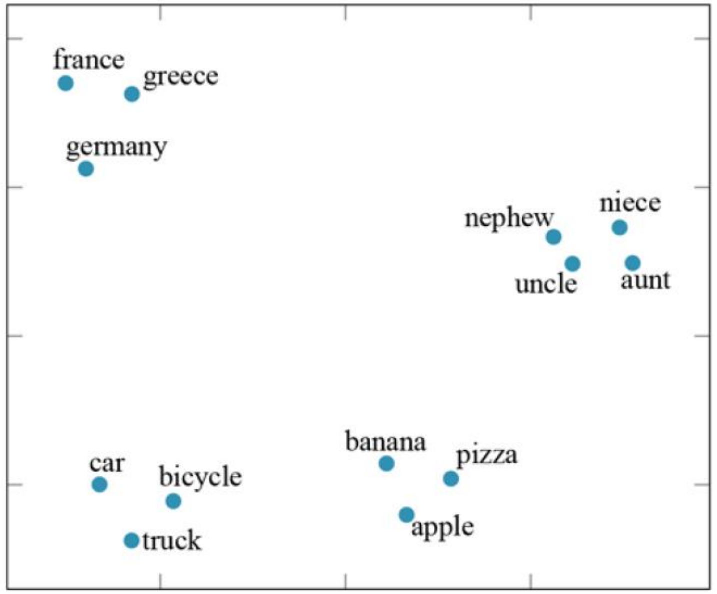
Outline

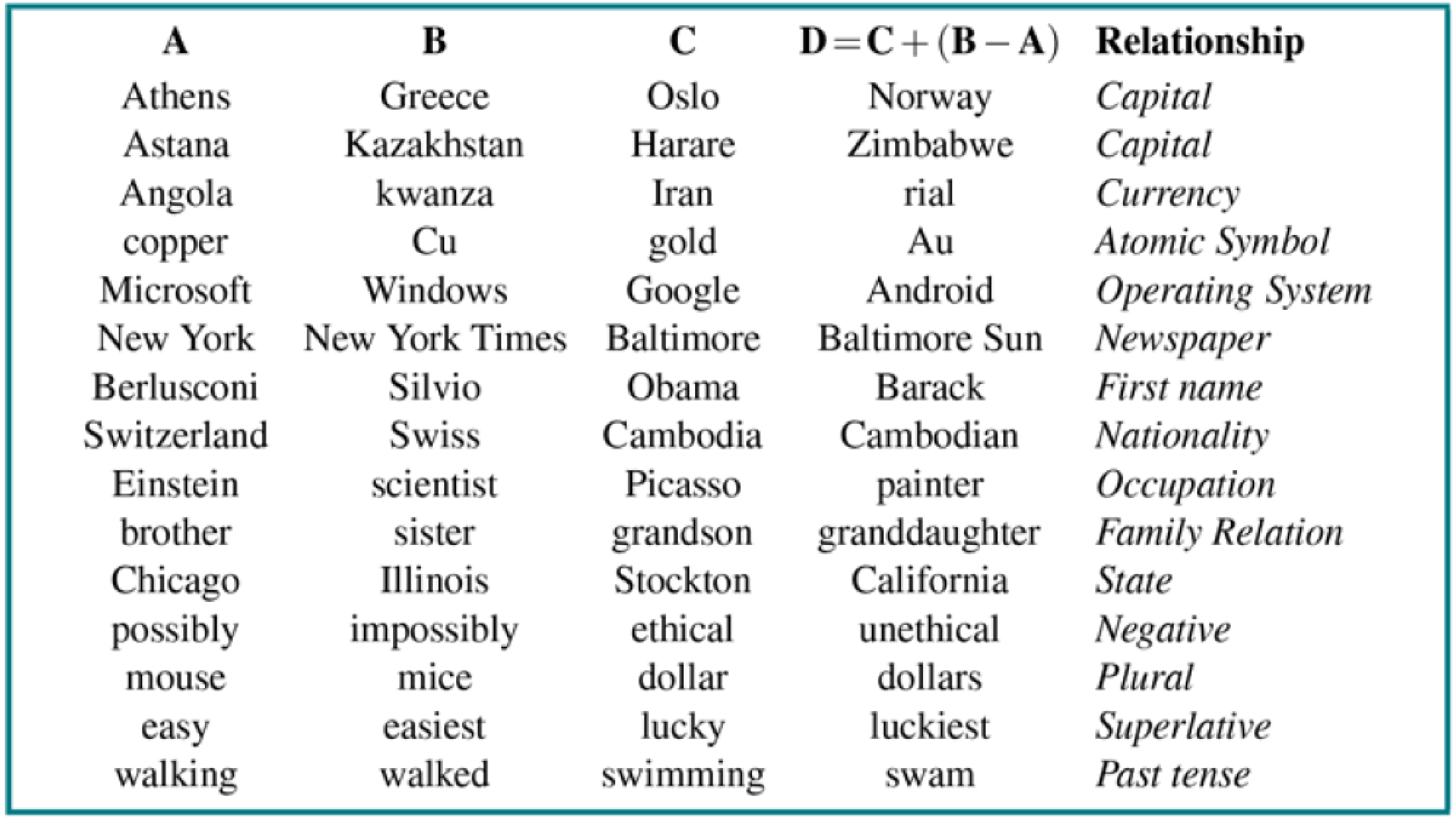
- Language Models
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
    - Why we need word embeddings
    - How we obtain word embeddings, using vs. not using labels.
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
- The Transformer Architecture
- Pretraining

Word Embeddings

- We would like a representation of words that
  - does not require manual feature engineering
  - allows for generalization between related words—words that are related syntactically (“colorless” and “ideal” are both adjectives), semantically (“cat” and “kitten” are both felines), topically (“sunny” and “sleet” are both weather terms), in terms of sentiment (“awesome” has opposite sentiment to “cringeworthy”), or otherwise.
- One-hot vectors
  - each word is represented as a high-dimensional sparse vector with only one position as 1 and all others as 0.
  - Would not capture the similarity between words.
- Word embeddings
  - A lowdimensional vector representing a word
  - Maps words into **a continuous, low-dimensional vector space** to **capture their semantic information**

Examples





A	B	C	D = C + (B − A)	Relationship
Athens	Greece	Oslo	Norway	*Capital*
Astana	Kazakhstan	Harare	Zimbabwe	*Capital*
Angola	kwanza	Iran	rial	*Currency*
copper	Cu	gold	Au	*Atomic Symbol*
Microsoft	Windows	Google	Android	*Operating System*
New York	New York Times	Baltimore	Baltimore Sun	*Newspaper*
Berlusconi	Silvio	Obama	Barack	*First name*
Switzerland	Swiss	Cambodia	Cambodian	*Nationality*
Einstein	scientist	Picasso	painter	*Occupation*
brother	sister	grandson	granddaughter	*Family Relation*
Chicago	Illinois	Stockton	California	*State*
possibly	impossibly	ethical	unethical	*Negative*
mouse	mice	dollar	dollars	*Plural*
easy	easiest	lucky	luckiest	*Superlative*
walking	walked	swimming	swam	*Past tense*

A word embedding model can sometimes answer the question "**A** is to **B** as **C** is to [what]?" with vector arithmetic: given the word embedding vectors for the words **A**, **B**, and **C**, compute the vector **D** = **C** + (**B** − **A**) and look up the word that is closest to **D**. (The answers in column **D** were computed automatically by the model. The descriptions in the "Relationship" column were added by hand.) Adapted from Mikolov *et al.* (2013, 2014).

1. Choose the width w (an odd number of words) for the prediction window to be used to tag each word. A typical value is $w = 5$, meaning that the tag is predicted based on the word plus the two words to the left and the two words to the right. Split every sentence in your corpus into overlapping windows of length w. Each window produces one training example consisting of the w words as input and the POS category of the middle word as output.

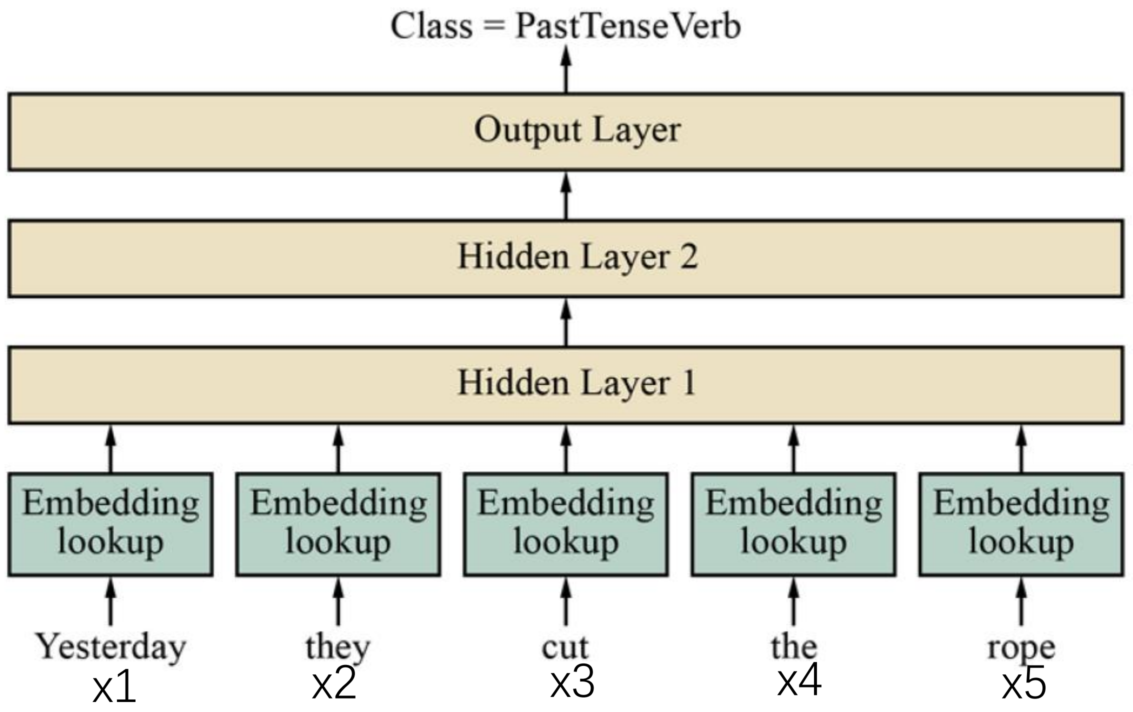
2. Create a vocabulary of all of the unique word tokens that occur more than, say, 5 times in the training data. Denote the total number of words in the vocabulary as v.

3. Sort this vocabulary in any arbitrary order (perhaps alphabetically).

4. Choose a value d as the size of each word embedding vector.

5. Create a new v-by-d weight matrix called $\mathbf{E}$. This is the word embedding matrix. Row i of $\mathbf{E}$ is the word embedding of the ith word in the vocabulary. Initialize $\mathbf{E}$ randomly (or from pretrained vectors).

6. Set up a neural network that outputs a part of speech label, as shown in Figure 24.3. The first layer will consist of w copies of the embedding matrix. We might use two additional hidden layers, $\mathbf{z}_1$ and $\mathbf{z}_2$ (with weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, respectively), followed by a softmax layer yielding an output probability distribution $\hat{\mathbf{y}}$ over the possible part-of-speech categories for the middle word:



$$x = Ex1 + Ex2 + Ex3 + Ex4 + Ex5$$

$$\mathbf{z}_1 = \sigma(\mathbf{W}_1\mathbf{x})$$

$$\mathbf{z}_2 = \sigma(\mathbf{W}_2\mathbf{z}_1)$$

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{W}_{out}\mathbf{z}_2)\,.$$

7. To encode a sequence of w words into an input vector, simply look up the embedding for each word and concatenate the embedding vectors. The result is a real-valued input vector $\mathbf{x}$ of length wd. Even though a given word will have the same embedding vector whether it occurs in the first position, the last, or somewhere in between, each embedding will be multiplied by a different part of the first hidden layer; therefore we are implicitly encoding the relative position of each word.

8. Train the weights $\mathbf{E}$ and the other weight matrices $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{W}_{out}$ using gradient descent. If all goes well, the middle word, *cut*, will be labeled as a past-tense verb, based on the evidence in the window, which includes the temporal past word "yesterday," the third-person subject pronoun "they" immediately before *cut*, and so on.

Word Embedding

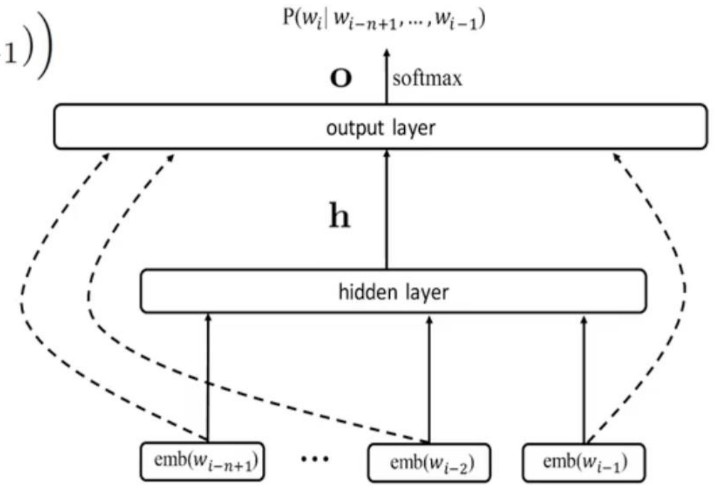
- Advantage of neural language modeling
    - Reduce sparsity
    - “the dog went off” P(went|the dog)
    - “the cat went off” P(went|the cat)
    - The training dataset hasn’t “the cat went”. In n-gram LM, we should use back-off method to estimate P(went|the cat), but under neural n-gram LM, as long as LM see “cat”, it will know “cat” similiar with “dog”, so “the cat” can also “went off”.
- By-product
    - Word embedding vectors
    - Given any word in a dictionary, you can get emb(w)

Word Embedding

- Neural n-gram language modeling
  - model:

$$\mathbf{x} = \left(emb(w_{i-n+1}) \oplus \ldots \oplus emb(w_{i-1})\right)$$

$$\mathbf{h} = tanh(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$$
$$\mathbf{o} = \mathbf{U}\mathbf{x} + \mathbf{W}^o\mathbf{h} + \mathbf{b}^o$$
$$\mathbf{p} = softmax(\mathbf{o}),$$



  - Training:

Given a training corpus $T = \{(w_1^i, w_2^i, \ldots, w_n^i)\}_i^{|T|}$

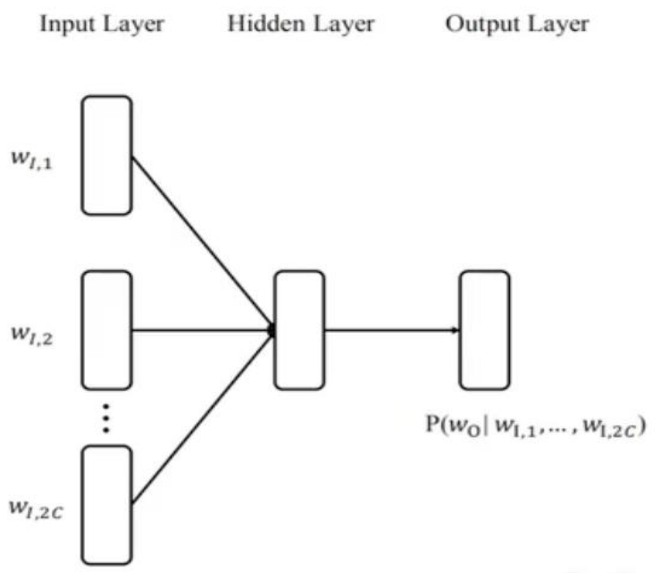
We minimise the loss function(MLE):

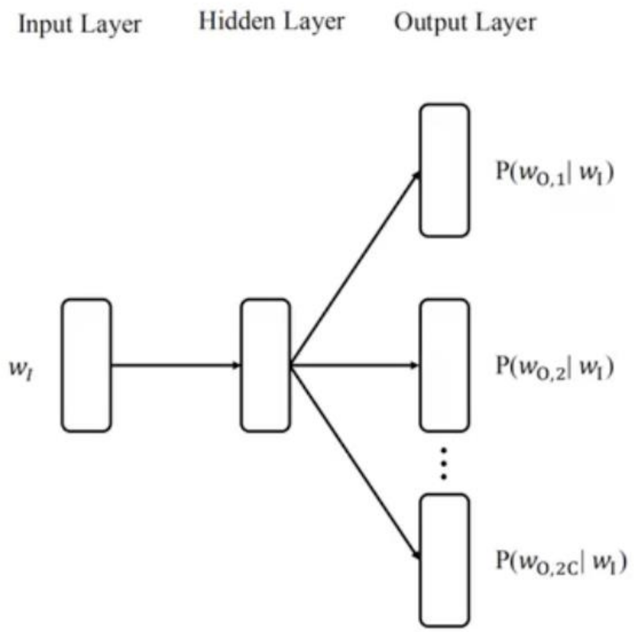
$$L = -\frac{1}{|T|}\sum_{i=1}^{|T|} \log\left(P(w_n^i | w_1^i, ..., w_{n-1}^i)\right)$$

- Model parameters: emb , $\mathbf{W}^h, \mathbf{W}^o, \mathbf{b}^h, \ \mathbf{b}^o, \mathbf{U}$

Word2Vec: CBOW vs. Skip-gram

- Continuous bag-of-words(CBOW): using surrounding c left words and c right words to predict current word.
  - C is the window size for target word w.
  - $W_{I,1}, W_{I,2} \ldots, W_{I,2C}$ can be a surrounding window of w.
- Skip-gram: Use current word to predict each surrounding word





Outline

- Language Models
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
    - Language models with RNNs
    - Classification with RNNs
- Sequence-to-sequence Models
- The Transformer Architecture
- Pretraining

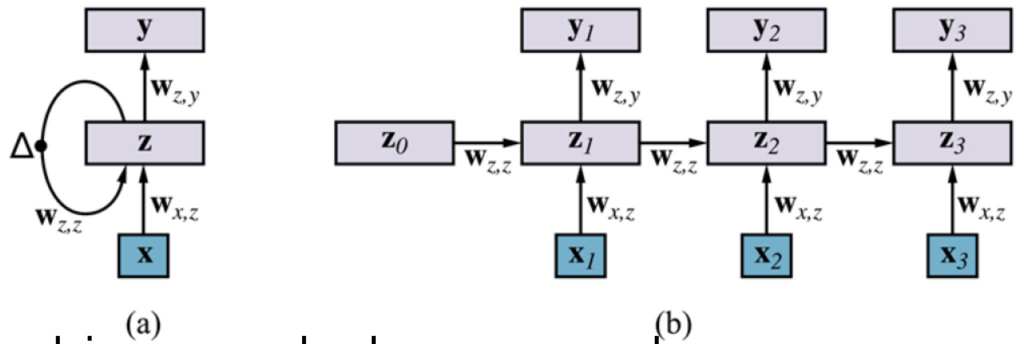
Recurrent Neural Network for NLP

- Word embeddings provide a good representation for **single words in isolation**, but language consists of an **ordered sequence** of words in which the context of surrounding words is important.
- For simple tasks like part of speech tagging, a small, fixed-size window of perhaps five words usually provides enough context.
- More complex tasks such as question answering, or reference resolution may require dozens of words as context.
    - For example, in the sentence “Eduardo told me that Miguel was very sick so I took him to the hospital,” knowing that him refers to Miguel and not Eduardo requires context that spans from the first to the last word of the 14-word sentence.

Language models with RNNs

- Creating a language model with sufficient context.
  - Recall that a language model is a probability distribution over sequences of words. It allows us to predict the next word in a text given all the previous words, and is often used as a building block for more complex tasks.
- Building a language model with either an n-gram model or a feedforward network with a fixed window of words can run into difficulty due to the problem of context:
  - either the required context will exceed the fixed window size.
  - or the model will have too many parameters, or both.

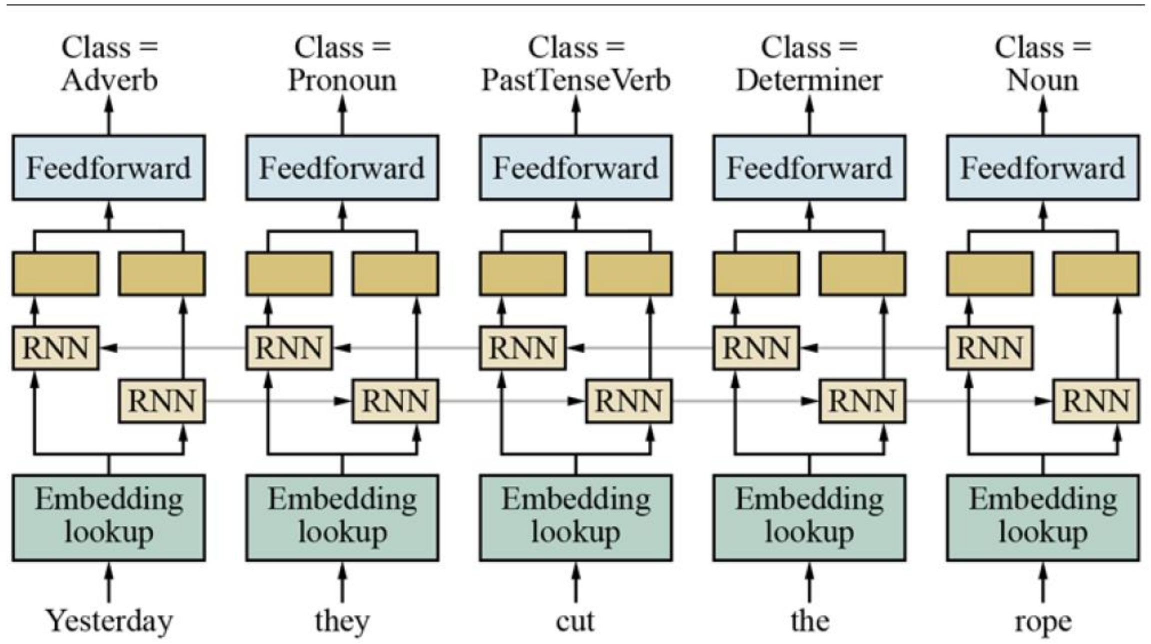
Language models with RNNs



- In an RNN language model each input word is encoded as a word embedding vector x_i.
- There is a hidden layer z_t which gets passed as input from one time step to the next.
- We are interested in doing multiclass classification: the classes are the words of the vocabulary.
    - The output y_t will be a softmax probability distribution over the possible values of the next word in the sentence.
- The RNN architecture can sometimes solve the limited context problem as well. In theory there is no limit to how far back in the input the model can look.
    - Each update of the hidden layer z_t has access to both the current input word x_t and the previous hidden layer z_t, which means that information about any word in the input can be kept in the hidden layer indefinitely, copied over (or modified as appropriate) from one time step to the next.
    - Of course, there is a limited amount of storage in , so it can't remember everything about all the previous words.

Classification with RNNs

- Part of speech tagging or coreference resolution
  - "Eduardo told me that Miguel was very sick so I took **him** to the hospital" it should output a high probability for "Miguel."
- Sentence-level classification
  - "This movie was poorly written and poorly acted" should be classified as Negative.
  - Use the last word's hidden vector.
  - Use pooling of all hidden vectors.



A bidirectional RNN, which concatenates a separate right-to-left model onto the left-to-right model. Bidirectional RNN for POS tagging.

$$\tilde{\mathbf{z}} = \frac{1}{s}\sum_{t=1}^{s} \mathbf{z}_t \,.$$

Outline

- Language Models
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
    - Sequence-to-sequence RNN for machine translation
    - Attention
- The Transformer Architecture
- Pretraining

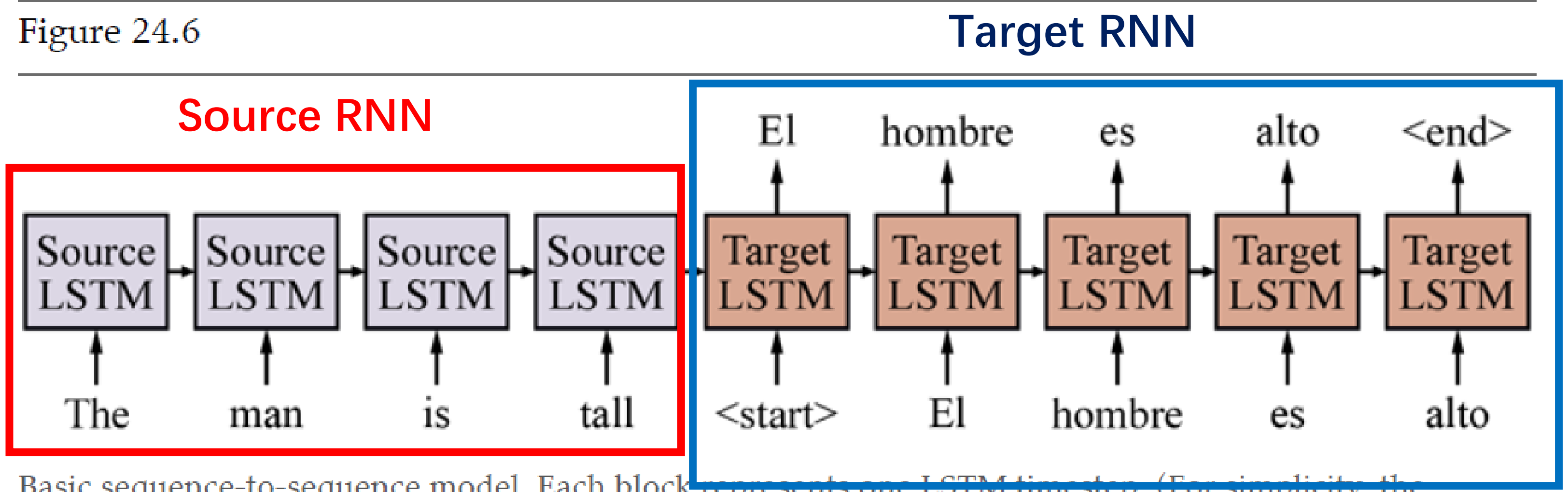
Sequence-to-sequence models

- One of the most widely studied tasks in NLP is **machine translation** (MT), where the goal is to translate a sentence from a source language to a target language.
- If there were a one-to-one correspondence between source words and target words, then we could treat MT as a simple tagging task—given the source word "perro" in Spanish, we tag it as the corresponding English word "dog."
- But in fact, words are not one-to-one: in Spanish the three words "caballo de mar" corresponds to the single English word "seahorse," and the two words "perro grande" translate to "big dog," with the word order reversed.
- We should generate one word at a time, but keep track of the context so that we can remember parts of the source that haven't been translated yet, and keep track of what has been translated so that we don't repeat ourselves.
- The generation of each target word is **conditional on the entire source sentence and on all previously generated target words**.

Seq2seq RNN for MT

- We run the **source RNN** over the source sentence and then use the final hidden state from the source RNN as the initial hidden state for the **target RNN**.
- This way, each target word is implicitly conditioned on both the entire source sentence and the previous target words.

Figure 24.6



Basic sequence-to-sequence model. Each block represents one LSTM timestep. (For simplicity, the embedding and output layers are not shown.) On successive steps we feed the network the words of the source sentence "The man is tall," followed by the <start> tag to indicate that the network should start producing the target sentence. The final hidden state at the end of the source sentence is used as the hidden state for the start of the target sentence. After that, each target sentence word at time t is used as input at time $t+1$, until the network produces the <end> tag to indicate that sentence generation is finished.

Limitations of Seq2Seq Models

- NEARBY CONTEXT BIAS
  - Whatever RNNs want to remember about the past, they have to fit into their hidden state.
  - An RNN is processing word (ortimestep) 57 in a 70-word sequence. The hidden state will likely contain more information about the word at timestep 56 than the word at timestep 5, because eachtime the hidden vector is updated it has to replace some amount of existing information with new information. This behavior is part of the intentional design of the model, and often makes sense for NLP, since nearby context is typically more important. However, far-away context can be crucial as well, and can get lost in an RNN model; even LSTMs have difficulty with this task.
- FIXED CONTEXT SIZE LIMIT
  - In an RNN translation model the entire source sentence is compressed into a single fixed-dimensional hidden state vector. An LSTM used in a state-of-the-art NLP model typically has around 1024 dimensions, and if we have to represent, say, a 64-word sentence in 1024 dimensions, this only gives us 16 dimensions per word—not enough for complex sentences. Increasing the hidden state vector size can lead to slow training and overfitting.
- SLOWER SEQUENTIAL PROCESSING
  - Neural networks realize considerable efficiency gains by processing the training data in batches so as to take advantage of efficient hardware support for matrix arithmetic. RNNs, on the other hand, seem to be constrained to operate on the training data one word at a time.

Attention

- What if the target RNN were conditioned on all of the hidden vectors from the source RNN, rather than just the last one?
  - This would mitigate the shortcomings of nearby context bias and fixed context size limits, allowing the model to access any previous word equally well.
- The target RNN must pay attention to different parts of the source for every word.
  - Suppose a network is trained to translate English to Spanish.
  - Given "The front door is red" followed by an end of sentence marker, which means it is time to start outputting Spanish words.
  - Ideally it should first pay attention to "The" and generate "La," then pay attention to "door" and output "puerta," and so on.

$$\mathbf{h}_i = RNN(\mathbf{h}_{i-1}, \mathbf{x}_i),$$

Attentional Seq2Seq Models

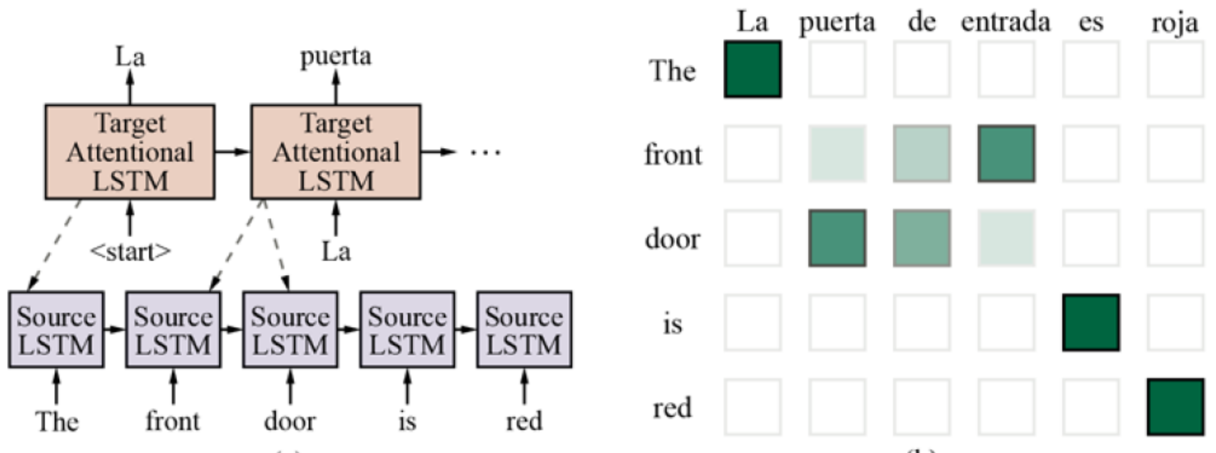
$$\mathbf{h}_i = RNN(\mathbf{h}_{i-1}, [\mathbf{x}_i; \mathbf{c}_i])$$

$$r_{ij} = \mathbf{h}_{i-1} \cdot \mathbf{s}_j$$

$$a_{ij} = e^{r_{ij}} / \left(\sum_k e^{r_{ik}} \right)$$

$$\mathbf{c}_i = \sum_j a_{ij} \cdot \mathbf{s}_j$$

where $\mathbf{h}_{i-1}$ is the target RNN vector that is going to be used for predicting the word at timestep i, and $\mathbf{s}_j$ is the output of the source RNN vector for the source word (or timestep) j. Both $\mathbf{h}_{i-1}$ and $\mathbf{s}_j$ are d-dimensional vectors, where d is the hidden size. The value of r_{ij} is therefore the raw "attention score" between the current target state and the source word j. These scores are then normalized into a probability a_{ij} using a softmax over all source words. Finally, these probabilities are used to generate a weighted average of the source RNN vectors, $\mathbf{c}_i$ (another d-dimensional vector).



(a) (b)

(a) Attentional sequence-to-sequence model for English-to-Spanish translation. The dashed lines represent attention. (b) Example of attention probability matrix for a bilingual sentence pair, with darker boxes representing higher values of a_{ij}. The attention probabilities sum to one over each column.

Decoding

- Once training is complete, we are given a source sentence, and our goal is to generate the corresponding target sentence.
- **Greedy decoding**: the simplest form of decoding is to select the highest probability word at each timestep and then feed this word as input to the next timestep.
  - This is called because after each target word is generated, the system has fully committed to the hypothesis that it has produced so far.
- **Optimal decoding** with beam search: In the context of MT decoding, beam search typically keeps the top hypotheses at each stage, extending each by one word using the top choices of words, then chooses the best of the resulting new hypotheses.

Outline

- Language Models
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
- The Transformer Architecture
    - Self-attention
    - Multi-head attention
    - Transformer
- Pretraining

The Transformer Architecture

- The influential article “Attention is all you need” (Vaswani et al., 2018) introduced the transformer architecture, which uses a self-attention mechanism that can model long distance context without a sequential dependency.
- Previously, in sequence-to-sequence models, attention was applied from the target RNN to the source RNN.
- **Self-attention** extends this mechanism so that each sequence of hidden states also attends to itself—the source to the source, and the target to the target.
- This allows the model to additionally capture long-distance (and nearby) context within each sequence.

Self-attention

- The most straightforward way of applying self-attention
  - The attention matrix is directly formed by the dot product of the input vectors.
  - Problem: The dot product between a vector and itself will always be high, so each hidden state will be biased towards attending to itself.
- Solving the problem: first projecting the input into three different representations using three different weight matrices.

- The **query vector** $\mathbf{q}_i = \mathbf{W}_q\mathbf{x}_i$ is the one being *attended from*, like the target in the standard attention mechanism.
- The **key vector** $\mathbf{k}_i = \mathbf{W}_k\mathbf{x}_i$ is the one being *attended to*, like the source in the basic attention mechanism.
- The **value vector** $\mathbf{v}_i = \mathbf{W}_v\mathbf{x}_i$ is the context that is being generated.

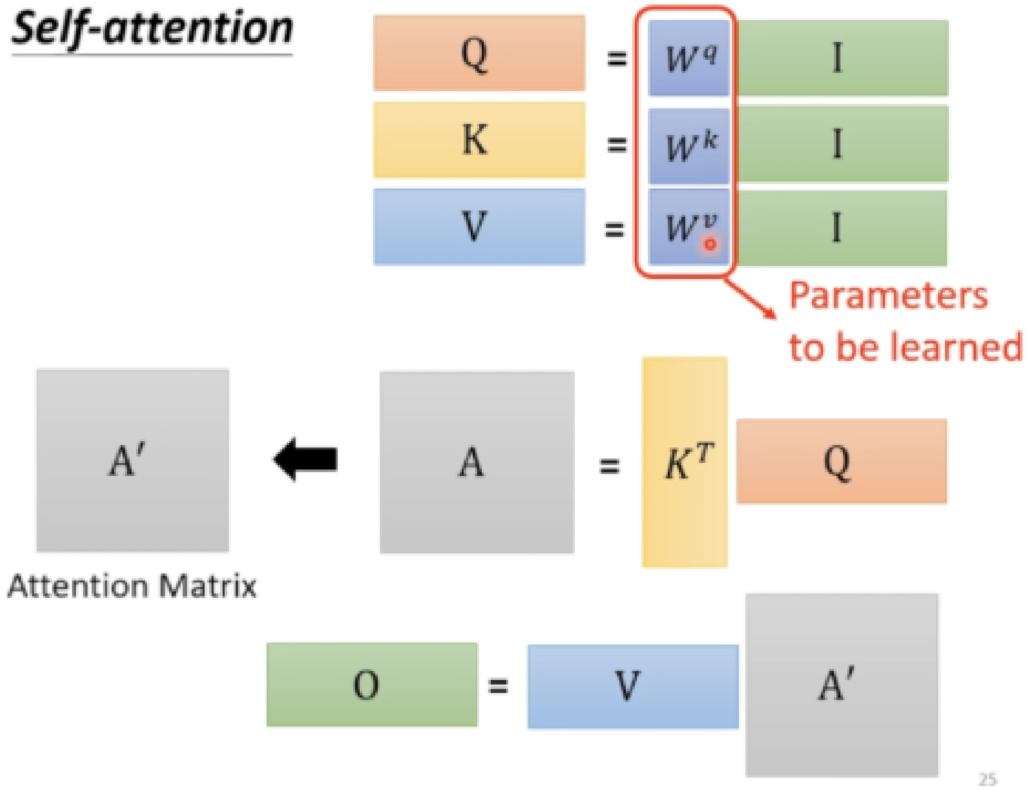
$$r_{ij} = (\mathbf{q}_i \cdot \mathbf{k}_j)/\sqrt{d}$$

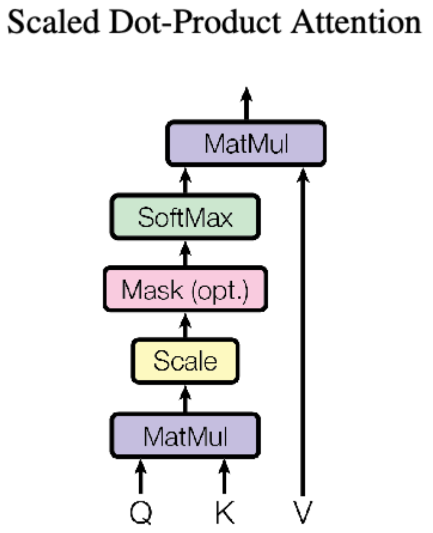
$$a_{ij} = e^{r_{ij}}/(\sum_k e^{r_{ik}})$$

$$\mathbf{c}_i = \sum_j a_{ij} \cdot \mathbf{v}_j \,,$$

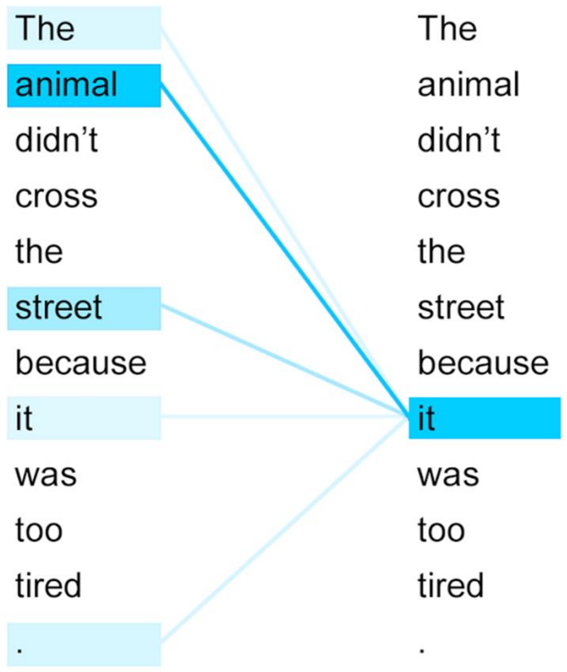
Self-attention

- Self means that Queries, Keys and Values are from the same source.





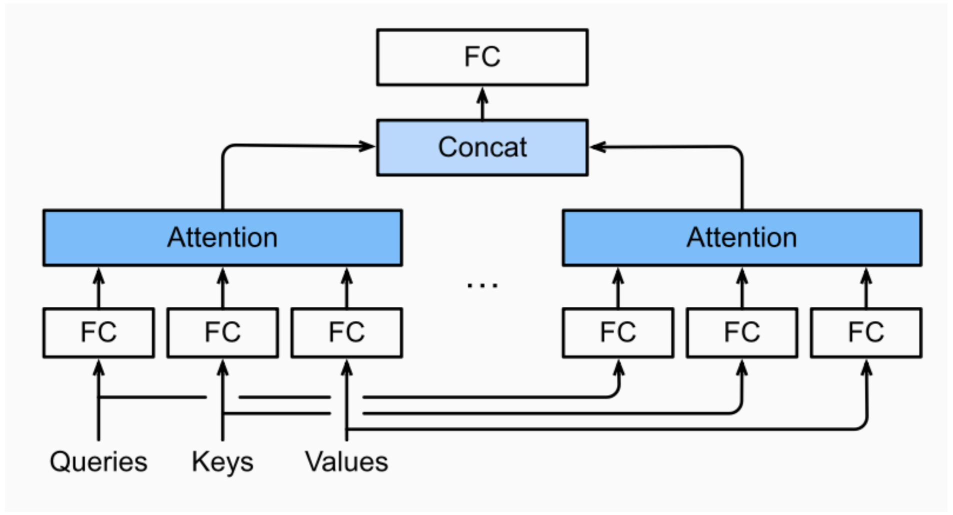
$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

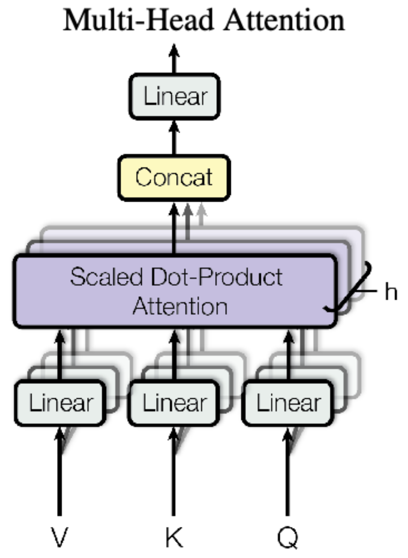


The animal didn't cross the street because it was too tired .
The animal didn't cross the street because it was too tired .

Multiheaded attention

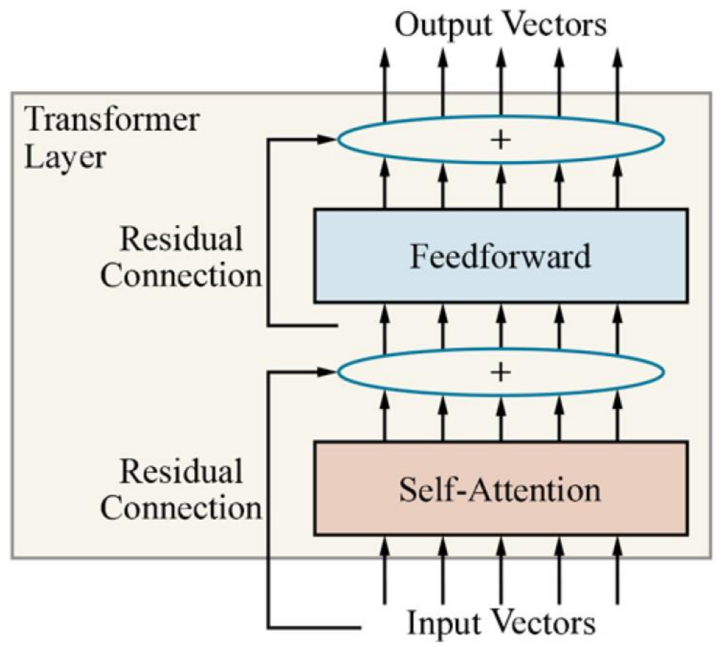
- We may want our model to combine knowledge from different behaviors of the same attention mechanism, such as capturing dependencies of various ranges (e.g., shorter-range vs. longer-range) within a sequence.





From Self-attention to Transformer

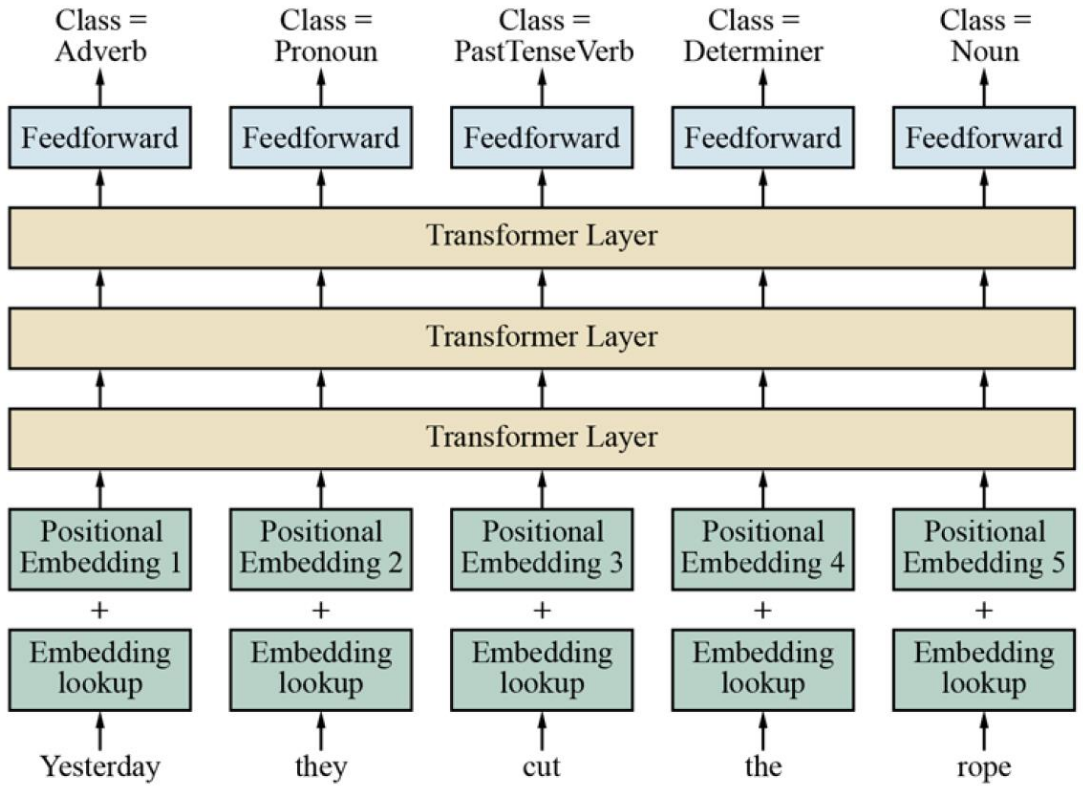
- Self-attention is only one component of the transformer model.
- Each transformer layer consists of several sub-layers.
  - At each transformer layer, self-attention is applied first.
  - The output of the attention module is fed through feedforward layers, where the same feedforward weight matrices are applied independently at each position.
  - A nonlinear activation function, typically ReLU, is applied after the first feedforward layer.
  - To address the potential vanishing gradient problem, two residual connections are added into the transformer layer.



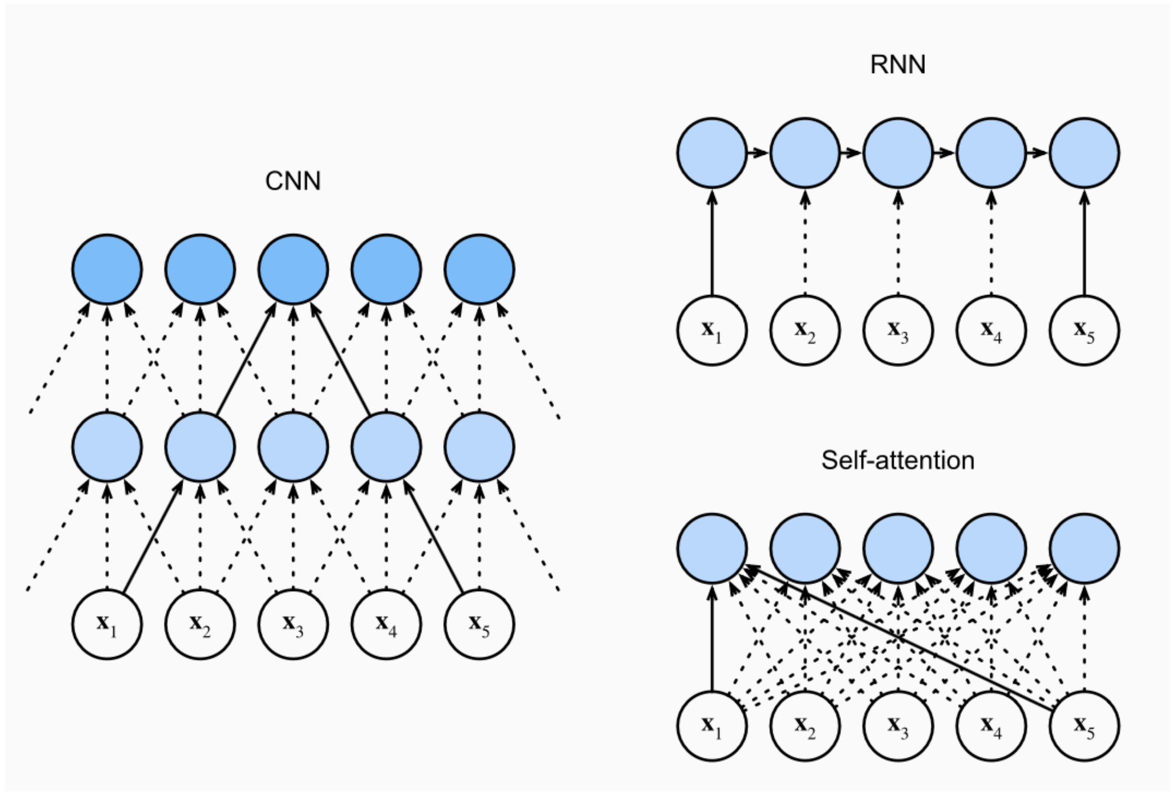
A single-layer transformer consists of self-attention, a feedforward network, and residual connections.

Positional encoding

- The transformer architecture does not explicitly capture the order of words in the sequence, since context is modeled only through self-attention, which is agnostic to word order.
- To capture the ordering of the words, the transformer uses a technique called positional embedding.
    - The input to the first transformer layer is the sum of the word embedding at position t plus the positional embedding corresponding to position t.



Comparing CNNs, RNNs and Self-Attention



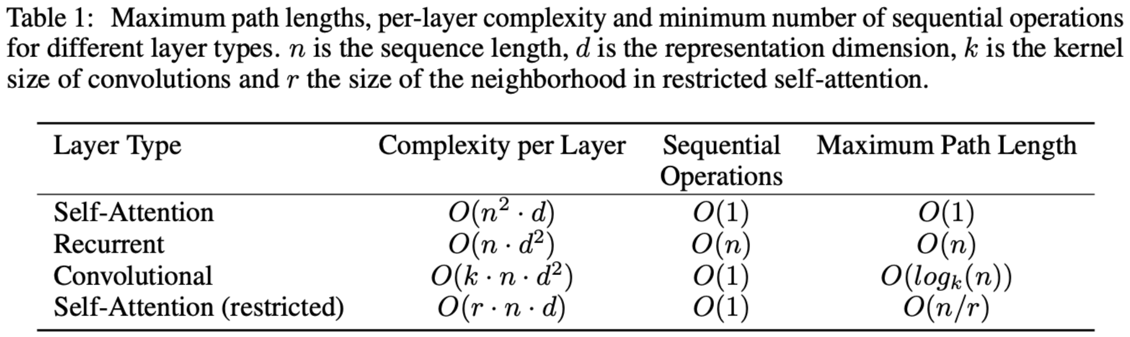


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Outline

- Language Models
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
- The Transformer Architecture
- Pretraining

Pre-training

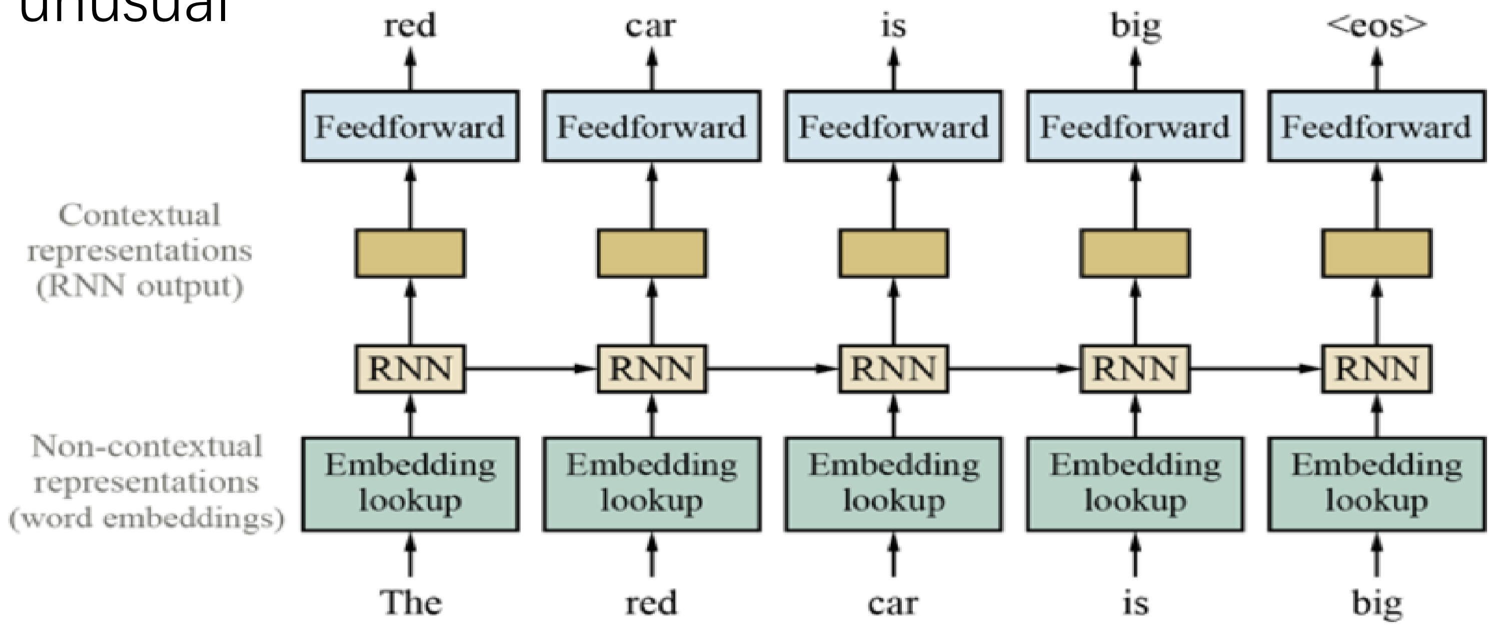
- Getting enough data to build a robust model can be a challenge. In computer vision, that challenge was addressed by assembling large collections of images (such as ImageNet) and hand-labeling them.
- For natural language, it is more common to work with text that is unlabeled.
    - The difference is in part due to the difficulty of labeling: an unskilled worker can easily label an image as “cat” or “sunset,” but it requires extensive training to annotate a sentence with part-of-speech tags or parse trees.
    - The difference is also due to the abundance of text: the Internet adds over 100 billion words of text each day, including digitized books, curated resources such as Wikipedia, and uncurated social media posts.
    - Any running text can be used to build n-gram or word embedding models, and some text comes with structure that can be helpful for a variety of tasks—for example, there are many FAQ sites with question-answer pairs that can be used to train a question-answering system.
    - Similarly, many Web sites publish side-by-side translations of texts, which can be used to train machine translation systems.

Pre-training

- We would prefer not to have to go to the trouble of creating a new data set every time we want a new NLP model.
- **Pretraining**: we use a **large amount** of shared **general domain language data** to train an initial version of an NLP model.
- **Fine-tuning**: From there, we can use **a smaller amount** of **domain-specific data** (perhaps including some labeled data) to refine the model.
  - The refined model can learn the vocabulary, idioms, syntactic structures, and other linguistic phenomena that are specific to the new domain.

Pretrained contextual representations

- Rose
  - The gardener planted a rose bush
  - The cabbage rose had an unusual fragrance
  - River rose five feet.
- First two more similar.
- Different from the 3rd.

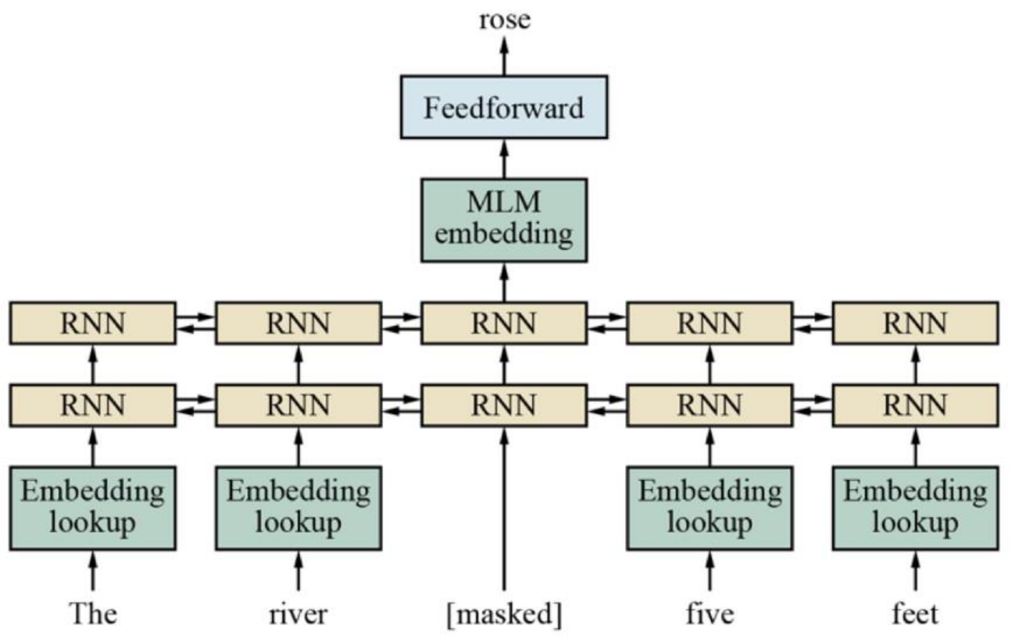

red
car
is
big
<eos>
Feedforward
Contextual representations (RNN output)
RNN
Non-contextual representations (word embeddings)
Embedding lookup
The
red
car
is
big

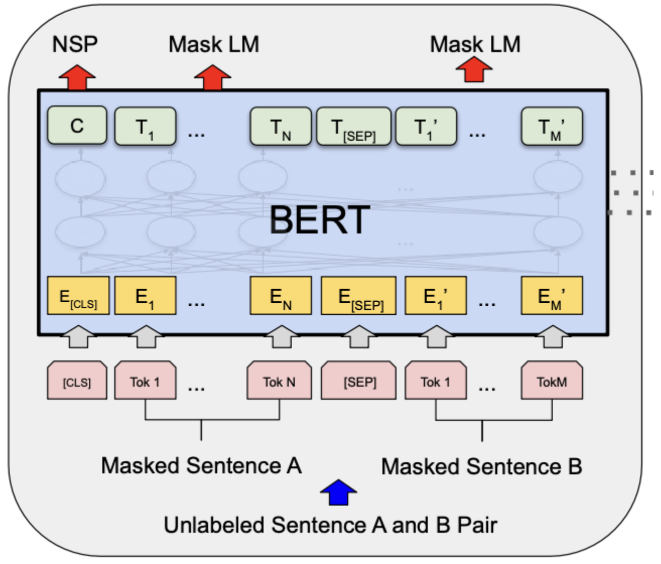
Masked language models

- A weakness of standard language models such as n-gram models is that the contextualization of each word is based only on the previous words of the sentence.
- But sometimes context from later in a sentence—for example, “feet” in the phrase “rose five feet”—helps to clarify earlier words.
- **Masked language model (MLM)** are trained by masking (hiding) individual words in the input and asking the model to predict the masked words.
    - given the input sentence “The river rose five feet” we can mask the middle word to get “The river _ five feet” and ask the model to fill in the blank.
- During training, a single sentence can be used multiple times with different words masked out.
    - The beauty of this approach is that it requires no labeled data; the sentence provides its own label for the masked word.
- If this model is trained on a large corpus of text, it generates pretrained representations that perform well across a wide variety of NLP tasks (machine translation, question answering, summarization, grammaticality judgments, and others).

Masked language models

- LM can learn good representations by training on some simple tasks (e.g. masked LM, Next Sentence Prediction)





Pre-training

Lecture 15 ILOs

- Language Models
- Grammar and Parsing
- Natural Language Processing Tasks
- Word Embeddings
- Recurrent Neural Networks for NLP
- Sequence-to-sequence Models
- The Transformer Architecture
- Pretraining

Next lecture

- Transformer for time series analytics
  - Peng Chen
- Neural Architecture search for time series prediction
  - Xingjian Wu