EECS 230 Deep Learning
Lecture 10: Language Model

Many slides from Julia Hockenmaier and Christopher Manning

ChatGPT (Generative Pretrained Transformer)

HU **You**

What are top 10 things to do in Merced?

ChatGPT

Merced, California, offers a range of activities and attractions for visitors to enjoy. Here are the top 10 things to do in Merced:

1. **Merced National Wildlife Refuge**: Explore the refuge and observe diverse bird species, including migratory birds, waterfowl, and endangered species like the Aleutian Canada goose.
2. **Castle Air Museum**: Discover the history of aviation at this museum, which features a vast collection of aircraft, including military planes, helicopters, and memorabilia.
3. **Yosemite National Park**: While not directly in Merced, the park is nearby and offers breathtaking scenery, hiking trails, waterfalls, and opportunities for camping and wildlife viewing.
4. **Applegate Park and Zoo**: Enjoy a day outdoors at this park, which features playgrounds, picnic areas, a small zoo with animals like monkeys, birds, and reptiles, and a miniature train ride.
5. **Downtown Merced**: Explore the historic downtown area with its charming shops, cafes, restaurants, and galleries. Don't miss the Merced County Courthouse Museum to learn about the region's history.
6. **Merced County Fairgrounds**: Check out events and festivals held at the fairgrounds throughout the year, including the Merced County Fair, concerts, rodeos, and more.
7. **Lake Yosemite**: Spend a relaxing day by the lake, where you can enjoy fishing, boating, kayaking, picnicking, and walking along the scenic trails.

UCMERCED

What is Natural Language Processing?

- Natural language processing is the set of methods for making human language accessible to computers. (Jacob Eisenstein)
- Natural language processing is the field at the intersection of Computer science (Artificial intelligence) and linguistics. (Christopher Manning)

NLP application: Machine translation

爷爷心疼孙女让娃睡懒觉 没想到引发了家庭矛盾

2020-08-25 08:06:03 来源：钱江晚报

70岁的钟大伯（化名）陷入了“暑期焦虑”：这个暑假，他每周都要接送孙女上下培训班。高温、酷暑，每天回来，都像脚踩棉花般没力气。

除了身体上的不适，还有精神上的紧张。

觉得儿子儿媳给孩子报班太多，钟大伯还和他们产生了冲突：“大热天的，大人孩子都遭罪。”

这段时间，钟大伯因为容易激动发火，失眠，胃口差，血压一直不稳定，来到了浙江省人民医院精神卫生科就诊。

Google Translate

Grandpa feels sorry for his granddaughter and let the baby sleep in

2020-08-25 08:06:03 Source: Qianjiang Evening News

Uncle Zhong (a pseudonym), 70, fell into "summer anxiety": This summer, he would shuttle his granddaughter to and from training classes every week. With high temperatures and scorching heat, every day I come back, I feel as weak as stepping on cotton.

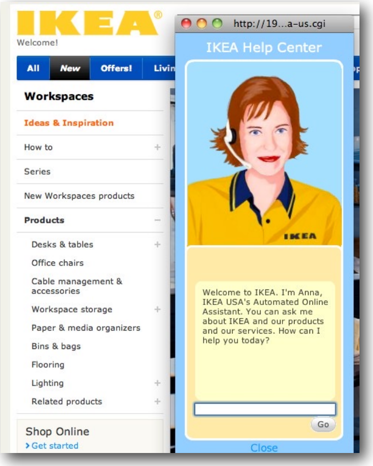
In addition to physical discomfort, there is also mental tension.

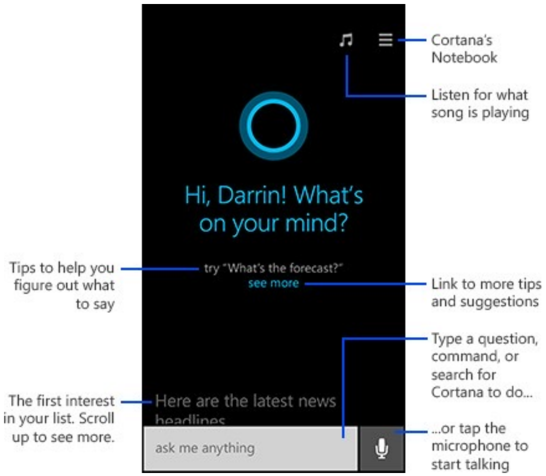
Feeling that his son and daughter-in-law were reporting too much for their children, Uncle Zhong also had a conflict with them: "It's a hot day, adults and children suffer."

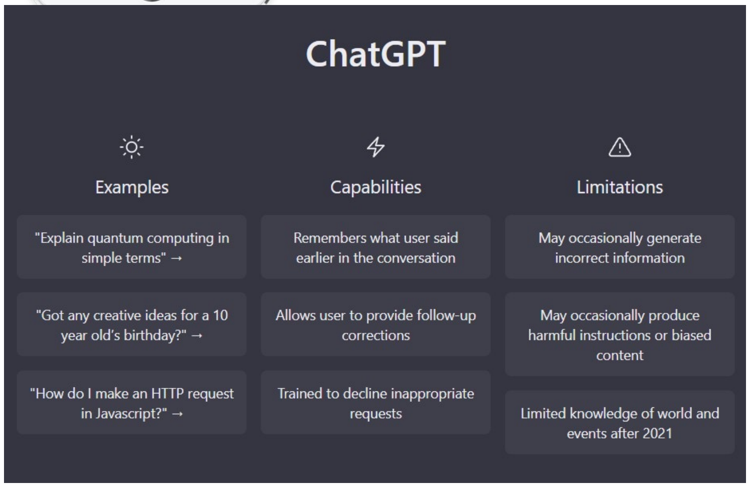
During this period of time, Uncle Zhong came to the Mental Health Department of Zhejiang Provincial People's Hospital because he was prone to get angry, insomnia, poor appetite, and unstable blood pressure.

http://education.news.cn/2020-08/25/c_1210768533.htm

NLP application: Dialog systems, chatbots, assistants



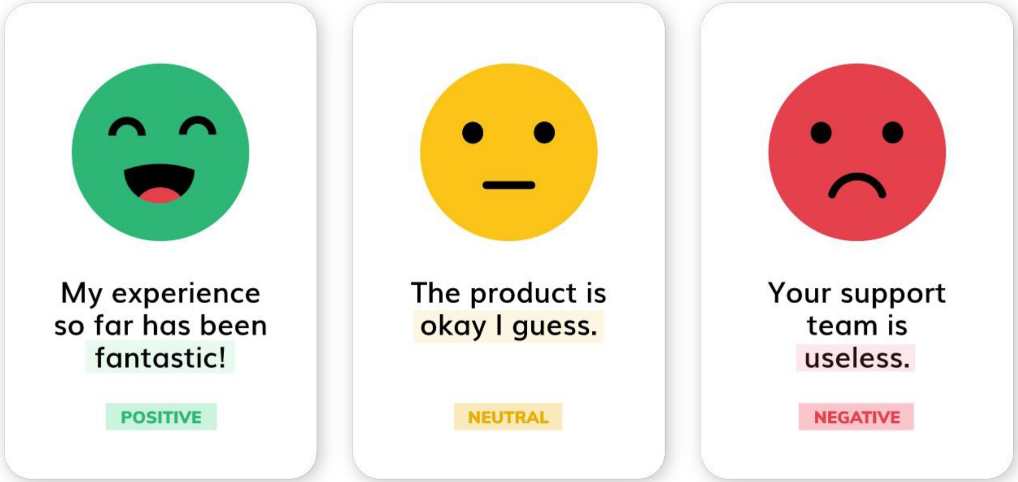




UCMERCED

NLP application: Sentiment analysis

- ❑Determine the meaning behind is positive, negative, or neutral



Outline

- ❑ Language Model
  - ❑ What is language model?
  - ❑ N-gram language model
- ❑ Word Embedding
  - ❑ Word2Vec
- ❑ Basic Neural Network
  - ❑ Feedforward Neural Network for language modeling
  - ❑ Convolutional Neural Network for NLP

Part I: Language Modeling

English Vocabulary

How large is the **vocabulary** of English (or any other language)?

Vocabulary size = the number of distinct word types

If you count words in text, you will find that…

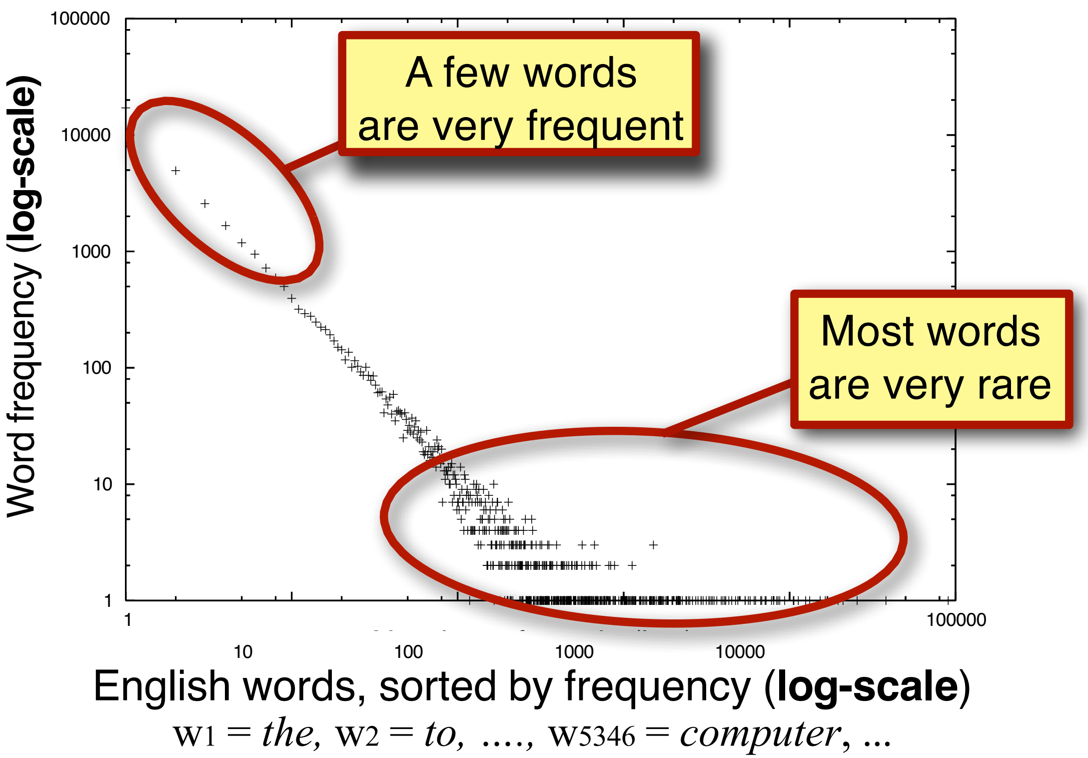
…a few words are very frequent
(the, be, to, of, and, a, in, that,…)

… most words are very rare.

… even if you've read a lot of text,
you will keep finding words you haven't seen before.

Word frequency: the number of occurrences of a word type in a text (or in a collection of texts)

Long-tailed word distribution



Implications of Long-tailed distribution for NLP

The good:

Any text will contain a number of words that are very **common**. We have seen these words often enough that we know (almost) everything about them.

The bad:

Any text will contain a number of words that are **rare**. We know *something* about these words, but haven't seen them often enough to know everything about them. They may occur with a meaning or a part of speech we haven't seen before.

The ugly:

Any text will contain a number of words that are **unknown** to us. We have *never* seen them before, but we still need to get at the structure (and meaning) of these texts.

How do we represent unknown words?

Many NLP systems assume a fixed vocabulary, but still have to handle **out-of-vocabulary (OOV)** words.

the **UNK** token

Replace all rare words (with a frequency at or below a given threshold, e.g. 2, 3, or 5) in your training data with an UNK token (UNK = “Unknown word”).

Replace *all* unknown words that you come across after training (including rare training words) with the same UNK token

UCMERCED

Why do we need language models?

Many NLP tasks require **natural language output**:

- **Machine translation**: return text in the target language
- **Speech recognition**: return a transcript of what was spoken
- **Natural language generation**: return natural language text
- **Spell-checking**: return corrected spelling of input

Language models define **probability distributions over** (natural language) **strings or sentences**.

➔ We can use a language model to generate strings

➔ We can use a language model to score/rank candidate strings so that we can choose the best (i.e. most likely) one:
if $P_{LM}(A) > P_{LM}(B)$, return A, not B

Hmmm, but…

… what does it mean for a language model to "*define a probability distribution*"?

… *why* would we want to define probability distributions over languages?

… how can we construct a language model such that it *actually* defines a probability distribution?

You should be able to answer these questions after this lecture

Key concepts

- N-gram language models
  - Independence assumptions
  - Getting from n-grams to a distribution over a language
  - Relative frequency (maximum likelihood) estimation
  - Smoothing

Now let's look at natural language

Text as a bag of words

```
Alice was beginning to get very tired of
sitting by her sister on the bank, and of
having nothing to do: once or twice she
had peeped into the book her sister was
reading, but it had no pictures or
conversations in it, 'and what is the use
of a book,' thought Alice 'without
pictures or conversation?'
```

$P(\texttt{of}) = 3/66$

$P(\texttt{Alice}) = 2/66$

$P(\texttt{was}) = 2/66$

$P(\texttt{to}) = 2/66$

$P(\texttt{her}) = 2/66$

$P(\texttt{sister}) = 2/66$

$P(\texttt{,}) = 4/66$

$P(\texttt{'}) = 4/66$

UCMERCED

Sampling with replacement

A sampled sequence of words

beginning by, very Alice but was and? reading no tired of to into sitting sister the, bank, and thought of without her nothing: having conversations Alice once do or on she it get the book her had peeped was conversation it pictures or sister in, 'what is the use had twice of a book''pictures or' to

$P(\texttt{of}) = 3/66$

$P(\texttt{Alice}) = 2/66$

$P(\texttt{was}) = 2/66$

$P(\texttt{to}) = 2/66$

$P(\texttt{her}) = 2/66$

$P(\texttt{sister}) = 2/66$

$P(\texttt{,}) = 4/66$

$P(\texttt{'}) = 4/66$

In this model, $P(English\ sentence) = P(word\ salad)$

Language modeling with N-grams

A **language model** over a vocabulary V assigns probabilities to strings drawn from V^*.

How do we compute the **probability of a string** $w^{(1)} \dots w^{(i)}$?

Recall the **chain rule**:

$$P(w^{(1)} \dots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} | w^{(1)}) \cdot \dots \cdot P(w^{(i)} | w^{(i-1)}, \dots, w^{(1)})$$

An **n-gram** language model assumes each word $w^{(i)}$ depends only on the **last n–1 words** $w^{(i-1)}, \dots, w^{(i-(n+1))}$

$$P_{ngram}(w^{(1)} \dots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} | w^{(1)}) \cdot \dots \cdot P(w^{(i)} | w^{(i-1)}, \dots, w^{(i-(n+1))})$$

N-gram models

N-gram models *assume* each word (event) depends only on the previous n–1 words (events):

Unigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$

Bigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)})$

Trigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)}, w^{(i-2)})$

Independence assumptions where the n-th event in a sequence depends only on the last n-1 events are called Markov assumptions (of order n–1).

How many parameters do n-gram models have?

Given a vocabulary V of |V| word types: so, for $|V| = 10^4$:

Unigram model: |V| parameters — 10^4 parameters

(one distribution $P(w^{(i)})$ with |V| outcomes
[each $w \in V$ is one outcome])

Bigram model: $|V|^2$ parameters — 10^8 parameters

Trigram model: $|V|^3$ parameters — 10^{12} parameters

A bigram model for Alice

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

$P(w^{(i)} = \texttt{of} \mid w^{(i-1)} = \texttt{tired}) = 1$

$P(w^{(i)} = \texttt{of} \mid w^{(i-1)} = \texttt{use}) = 1$

$P(w^{(i)} = \texttt{sister} \mid w^{(i-1)} = \texttt{her}) = 1$

$P(w^{(i)} = \texttt{beginning} \mid w^{(i-1)} = \texttt{was}) = 1/2$

$P(w^{(i)} = \texttt{reading} \mid w^{(i-1)} = \texttt{was}) = 1/2$

$P(w^{(i)} = \texttt{bank} \mid w^{(i-1)} = \texttt{the}) = 1/3$

$P(w^{(i)} = \texttt{book} \mid w^{(i-1)} = \texttt{the}) = 1/3$

$P(w^{(i)} = \texttt{use} \mid w^{(i-1)} = \texttt{the}) = 1/3$

UC MERCED

Using a bigram model for Alice

English

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

Word Salad

beginning by, very Alice but was and? reading no tired of to into sitting sister the, bank, and thought of without her nothing: having conversations Alice once do or on she it get the book her had peeped was conversation it pictures or sister in, 'what is the use had twice of a book''pictures or' to

Now, $P(English) \gg P(word\ salad)$

$P(w^{(i)} = \texttt{of} \mid w^{(i-1)} = \texttt{tired}) = 1$

$P(w^{(i)} = \texttt{of} \mid w^{(i-1)} = \texttt{use}) = 1$

$P(w^{(i)} = \texttt{sister} \mid w^{(i-1)} = \texttt{her}) = 1$

$P(w^{(i)} = \texttt{beginning} \mid w^{(i-1)} = \texttt{was}) = 1/2$

$P(w^{(i)} = \texttt{reading} \mid w^{(i-1)} = \texttt{was}) = 1/2$

$P(w^{(i)} = \texttt{bank} \mid w^{(i-1)} = \texttt{the}) = 1/3$

$P(w^{(i)} = \texttt{book} \mid w^{(i-1)} = \texttt{the}) = 1/3$

$P(w^{(i)} = \texttt{use} \mid w^{(i-1)} = \texttt{the}) = 1/3$

UCMERCED

From n-gram probabilities to language models

Recall: a language $L \subseteq V^*$ is a (possibly infinite) set of strings over a (finite) vocabulary V.

$P(w^{(i)} \mid w^{(i-1)})$ defines a distribution over the words in V:

$$\forall w \in V : \left[\sum_{w' \in V} P\left(w^{(i)} = w' \mid w^{(i-1)} = w\right) \right] = 1$$

By multiplying this distribution N times, we get one distribution over all strings of the same length N (V^N):

Prob. of one N-word string: $P\left(w_1 \dots w_N\right) = \prod_{i=1 \dots N} P\left(w^{(i)} = w_i \mid w^{(i-1)} = w_{i-1}\right)$

UCMERCED

From n-gram probabilities to language models

We have just seen how to use n-gram probabilities to define *one* distribution $P(V^N)$ *for each string length* N.

But a language model $P(L)=P(V^*)$ should define one distribution $P(V^*)$ that sums to one over *all* strings in $L \subseteq V^*$, *regardless of their length*:

$$P(L) = P(V) + P(V^2) + P(V^3) + \dots P(V^n) + \dots = 1$$

Solution:

Add an End-of-Sentence (EOS) token to V

Assume a) that each string ends in EOS and

b) that EOS can only appear at the end of a string.

From n-gram probabilities to language models with EOS

Think of a language model as a **stochastic process:**

— At each time step, randomly pick one more word.

— **Stop** generating more words when the word you pick is a special end-of-sentence (EOS) token.

To be able to pick the EOS token, we have to **modify our training data** so that each sentence ends in EOS.

This means our vocabulary is now $V_{EOS} = V \cup \{EOS\}$

We then get an actual language model,
i.e. a distribution over strings of *any* length

Technically, this is only true because P(EOS | …) will be high enough that we are always guaranteed to stop after having generated a finite number of words
A leaky or inconsistent language model would have $P(L) < 1$. That could happen if EOS had a very small probability (but doesn't really happen in practice).

UCMERCED

Why do we want *one* distribution over L?

Why do we care about having ***one probability distribution for all lengths?***

This allows us to *compare the probabilities* of strings of different lengths, because they're computed by the same distribution.

This allows us to *generate* strings of arbitrary length with one model.

Learning (estimating) a language model

Where do we get the parameters of our model (its actual probabilities) from?

$$P(w^{(i)} = \text{'}the\text{'} \mid w^{(i-1)} = \text{'}on\text{'}) = ???$$

We need (a large amount of) text as training data to estimate the parameters of a language model.

The most basic parameter estimation technique: relative frequency estimation (frequency = counts)

$$P(w^{(i)} = \text{'}the\text{'} \mid w^{(i-1)} = \text{'}on\text{'}) = C(\text{'}on\ the\text{'}) \,/\, C(\text{'}on\text{'})$$

Also called Maximum Likelihood Estimation (MLE)

$C(\text{'}on\ the\text{'})$ [or f('on the') for frequency]:

How often does 'on the' appear in the training data?

NB: $C(\text{'}on\text{'}) = \sum_{w \in V} C(\text{'}on\text{'}\ w)$

Handling unknown words: UNK

Training:

— Define a fixed vocabulary V such that all words in V appear at least *n* times in the training data

(e.g. all words that occur at least *5* times in the training corpus, or the most common 10,000 words in training)

— Add a new token UNK to V, and replace all other words in the corpus that are not in V by this token UNK

— Estimate the model on this modified training corpus.

Testing (when computing the probability of a string):

Replace any words not in the vocabulary by UNK

What about the beginning of the sentence?

In a trigram model

$$P(w^{(1)}w^{(2)}w^{(3)}) = P(w^{(1)})P(w^{(2)} \mid w^{(1)})P(w^{(3)} \mid w^{(2)}, w^{(1)})$$

only the third term $P(w^{(3)} \mid w^{(2)}, w^{(1)})$ is an actual trigram probability. What about $P(w^{(1)})$ and $P(w^{(2)} \mid w^{(1)})$?

If this bothers you:

Add n–1 **beginning-of-sentence** (BOS) symbols to each sentence for an n–gram model:

BOS_1 BOS_2 Alice was ...

Now the unigram and bigram probabilities involve only BOS symbols.

Summary: Estimating a bigram model with BOS (<s>), EOS (</s>) and UNK using MLE

1. Replace all words not in V in the training corpus with UNK
2. Bracket each sentence by special start and end symbols:

   <s> Alice was beginning to get very tired …</s>
3. Define the Vocabulary V' = all tokens in modified training corpus (all common words, UNK, <s>, </s>)
4. Count the frequency of each bigram….

   $C(\text{<s> Alice}) = 1$, $C(\text{Alice was}) = 1$, …
5. .... and normalize these frequencies to get probabilities:

$$P(\textit{was} \mid \textit{Alice}) = \frac{C(\textit{Alice was})}{\sum_{w_i \in V'} C(\textit{Alice } w_i)}$$

UCMERCED

How do we use language models?

Independently of any application, we could use a language model as a random sentence *generator*
(we sample sentences according to their language model probability)

We can use a language model as a sentence *ranker*.

We prefer output sentences S_{Out} that have a higher language model probability. We can use a language model $P(S_{Out})$ to score and rank these different candidate output sentences, e.g. as follows:

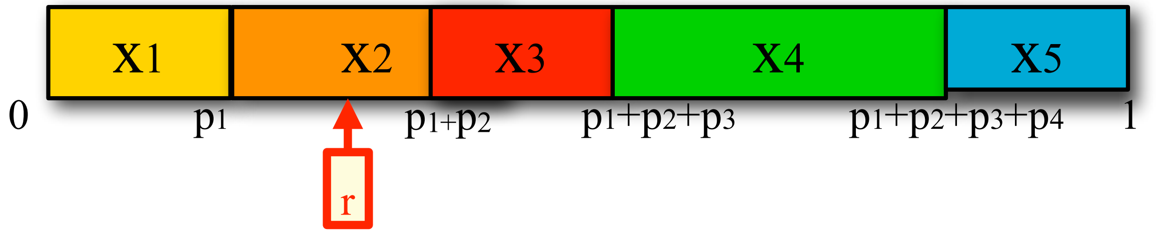
$$\mathrm{argmax}_{S_{Out}}\, P(S_{Out} \mid \mathrm{Input}) = \mathrm{argmax}_{S_{Out}}\, P(\mathrm{Input} \mid S_{Out}) P(S_{Out})$$

Generating from a distribution

How do you generate text from an *n*-gram model?

That is, how do you sample from a distribution $P(X \mid Y=y)$?

- Assume X has N possible outcomes (values): $\{x_1, \ldots, x_N\}$ and $P(X=x_i \mid Y=y) = p_i$
- Divide the interval [0,1] into N smaller intervals according to the probabilities of the outcomes
- Generate a random number r between 0 and 1.
- Return the x_1 whose interval the number is in.



Generating the Wall Street Journal

unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

bigram: Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

trigram: They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Generating Shakespeare

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

Shakespeare as corpus

The Shakespeare corpus has N=884,647 word tokens for a vocabulary of V=29,066 word types

Shakespeare used 300,000 bigram types out of V^2= 844 million possible bigram types.
99.96% of possible bigrams don't occur in this corpus.

Corollary: A relative frequency estimate based on this corpus assigns non-zero probability to only 0.04% of possible bigrams

- That percentage is even lower for trigrams, 4-grams, etc.
- 4-grams *look* like Shakespeare because they *are* Shakespeare!

The UNK token

What would happen if we used an UNK token on a corpus the size of Shakespeare's?

1. If we set the frequency threshold for which words to replace too high, a very large fraction of tokens become UNK.

2. Even with a low threshold, UNK will have a very high probability, because in such a small corpus, many words appear only once.

3. But we would still only observe a small fraction of possible bigrams (or trigrams, quadrigrams, etc.)

MLE doesn't capture unseen events

We estimated a model on 884K word tokens, but:

Only 30,000 word types occur in the training data
Any word that does not occur in the training data has zero probability!

Only 0.04% of all possible bigrams (for 30K word types) occur in the training data
Any bigram that does not occur in the training data has *zero* probability (even if we have seen both words in the bigram by themselves)

Part II: Word Embedding

How to represent the meaning of a word usable for a computer?

- Common NLP solution: Use, e.g., WordNet, a thesaurus containing lists of synonym sets and hypernyms ("is a" relationships).

e.g., synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g., hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Problems with WordNet

- Great as a resource but missing nuance
  - e.g., "proficient" is listed as a synonym for "good" This is only correct in some contexts
- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can't compute accurate word similarity

Representing words as discrete symbols

- In traditional NLP, we regard words as discrete symbols: hotel, conference, motel
- Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- Vector dimension = number of words in vocabulary (e.g., 500,000)

Problem with words as discrete symbols

- **Example:** in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"
- But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- These two vectors are orthogonal
- There is no natural notion of **similarity** for one-hot vectors!

- **Solution:**

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- Instead: learn to encode similarity in the vectors themselves

Representing words by their context

- Distributional semantics: A word's meaning is given by the words that frequently appear close-by
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	banking	crises as happened in 2009…
…saying that Europe needs unified	banking	regulation to replace the hodgepodge…
…India has just given its	banking	system a shot in the arm…

These context words will represent ***banking***

Word Vector

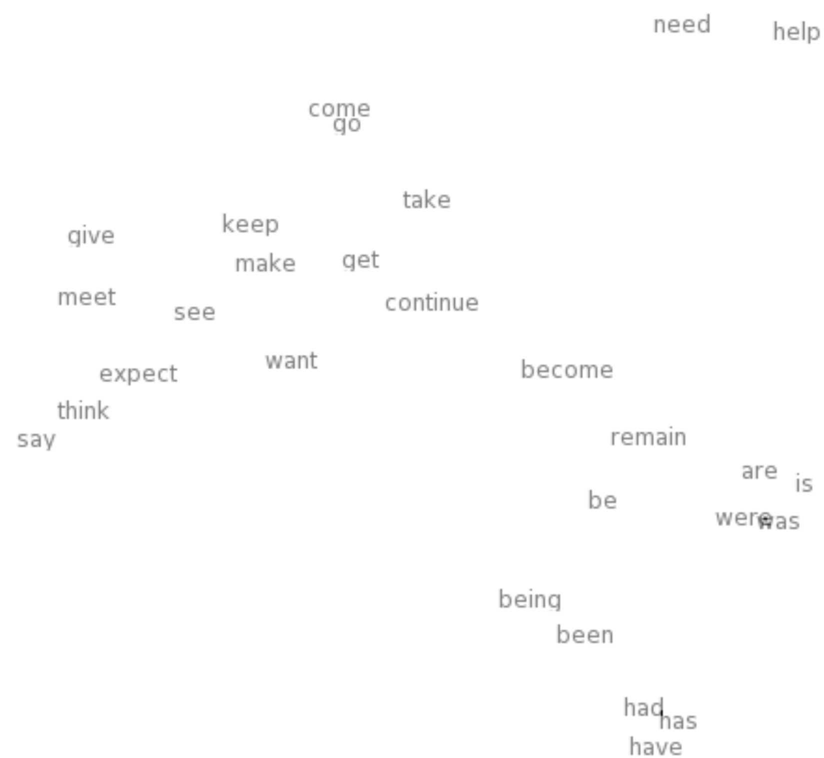
- We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

$$banking = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

- Note: word vectors are also called word embeddings or (neural) word representations

Visualize Word Vectors

$$expect = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



Word2Vec: Overview

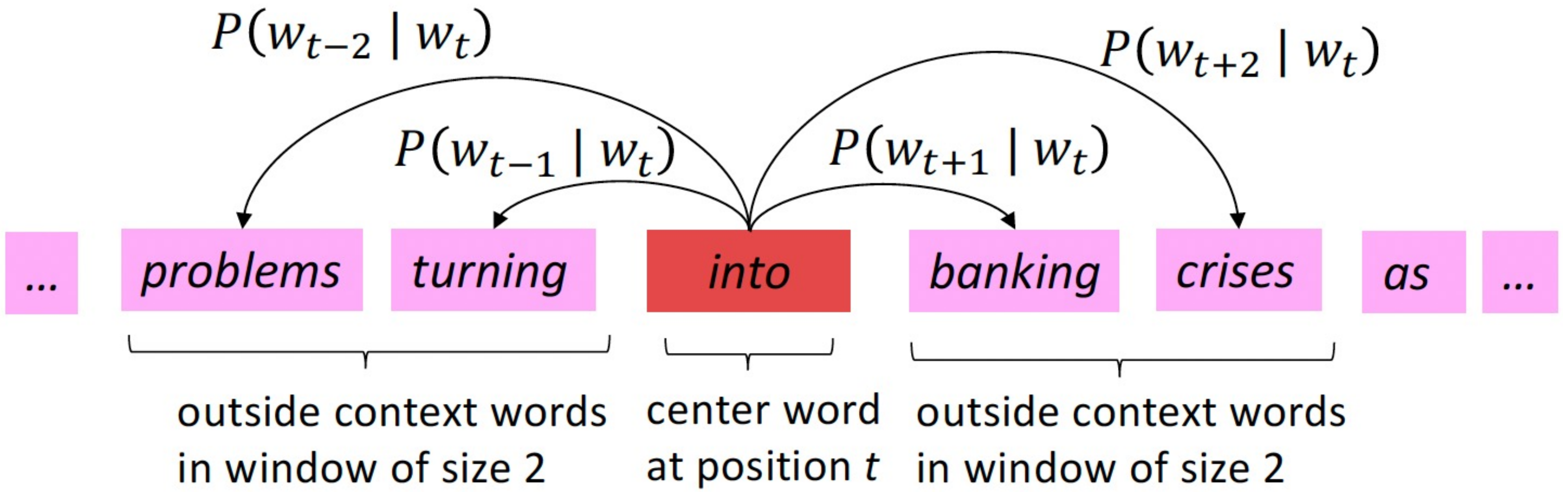
- Word2vec *(Mikolov et al. 2013)* is a framework for learning word vectors

Idea:

- We have a large corpus (“body”) of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context (“outside”) words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

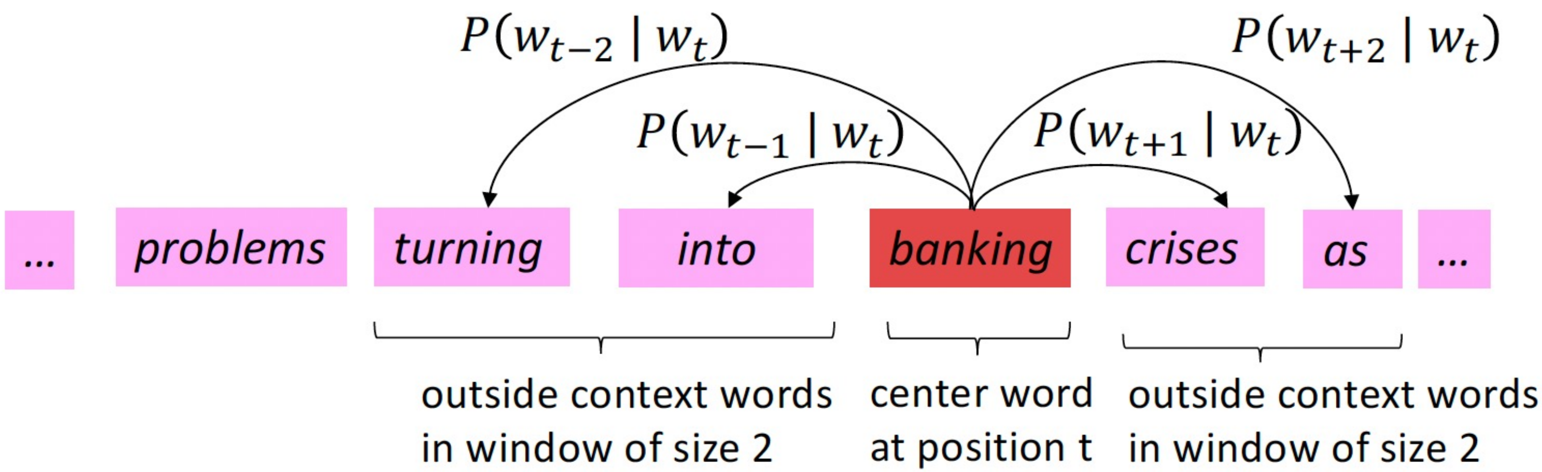
Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Objective Function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word w_j. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2Vec Objective Function

- We want to minimize the objective function:

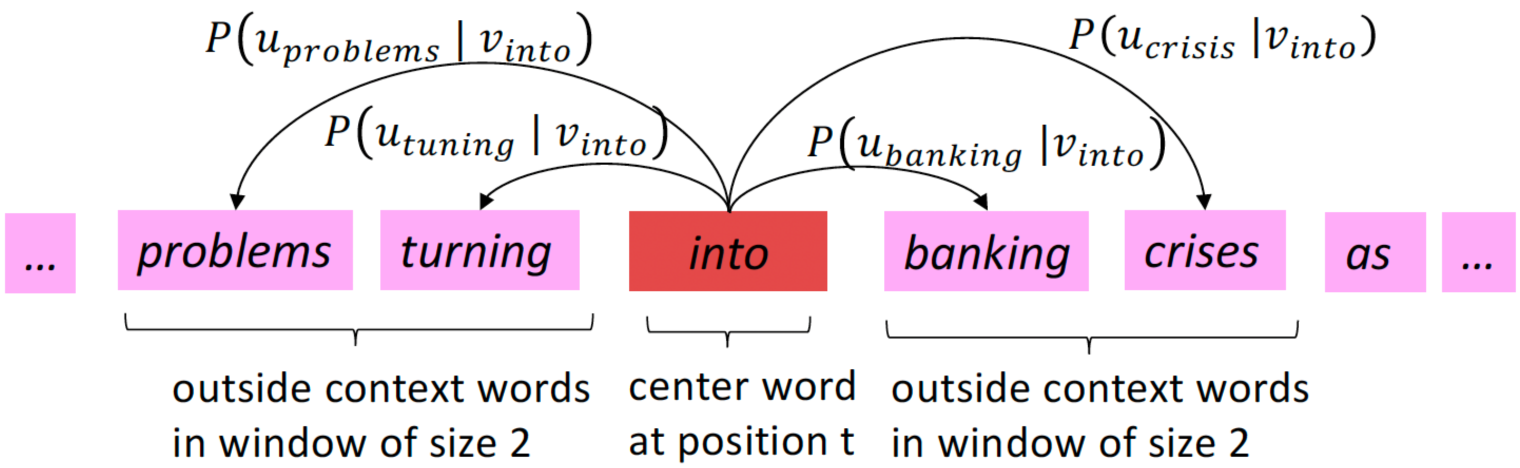
$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- **Question:** How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Word2Vec Overview with vectors

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $\mathrm{P}(problems \mid into \, ; u_{problems}, v_{into}, \theta)$



Word2Vec prediction function

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

① Dot product compares similarity of *o* and *c*.

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

- This is a soft-max function!

Word2Vec: Optimization

- To train a model, we gradually adjust parameters to minimize a loss
- Recall: θ represents all the model parameters, in one long vector
- In our case, with d-dimensional vectors and V-many words, we have:
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- We optimize these parameters by gradient descent

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

UCMERCED

Word2Vec: Gradient

- Need gradients for all center words and outside words

$$\log p(o|c) = \log \frac{\exp({u_o}^T v_c)}{\sum_{w=1}^{V} \exp({u_w}^T v_c)}$$

- Why two vectors? Easier optimization. Average both at the end.

Part III: Basic Neural Networks for NLP

What have we covered so far?

We have covered a broad overview of some basic techniques in NLP:

– N-gram language models
– Word embeddings

Let's create a (much better) neural language model!

Our first neural net for NLP:

A neural n-gram model

Given a fixed-size vocabulary V, an *n*-gram model predicts the probability of the *n*-th word following the preceding *n*–1 words:

$$P(w^{(i)} \mid w^{(i-1)}, w^{(i-2)}, \ldots, w^{i-(n-1)})$$

How can we model this with a neural net?

— **Input layer:** concatenate *n*–1 word vectors

— **Output layer:** a softmax over |V| units

An n-gram model $P(w \mid w_1 \ldots w_k)$ as a feedforward net (naively)

Assumptions:

The **vocabulary** V contains V types (incl. UNK, BOS, EOS)
We want to condition each word on k preceding words

Our (naive) model:

— [Naive]
Each **input word** $w_i \in \mathrm{V}$ is a ***V*-dimensional one-hot vector** $v(w)$
$\rightarrow$ The **input layer** $\mathbf{x} = [v(w_1),\ldots,v(w_k)]$ has $V \times k$ elements

— We assume **one hidden layer h**

— The **output layer** is a softmax over V elements
$$P(w \mid w_1 \ldots w_k) = \mathrm{softmax}(\mathbf{h}\mathbf{W}^2 + \mathbf{b}^2)$$

An n-gram model $P(w \mid w_1 \ldots w_k)$ as a feedforward net (better)

Assumptions:

The **vocabulary** V contains *V* types (incl. UNK, BOS, EOS)
We want to condition each word on *k* preceding words

Our (better) model:

— [Better]
Each **input word** $w_i \in \mathrm{V}$ is an ***n*-dimensional dense embedding vector** $v(w)$ (with $n \ll V$)
→ The **input layer** $\mathbf{x} = [v(w_1),\ldots,v(w_k)]$ has $n \times k$ elements

— We assume **one hidden layer h**

— The **output layer** is a softmax over V elements

$$P(w \mid w_1 \ldots w_k) = \mathrm{softmax}(\mathbf{h}\mathbf{W}^2 + \mathbf{b}^2)$$

UC MERCED

Our neural n-gram models

Architecture:

Input Layer: $\mathbf{x} = [v(w_1)\ldots.v(w_k)]$

Hidden Layer: $\mathbf{h} = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)$

Output Layer: $P(w \mid w_1 \ldots w_k) = \text{softmax}(\mathbf{h}\mathbf{W}^2 + \mathbf{b}^2)$

How many parameters do we need? [# of weights and biases]:

Hidden layer with one-hot inputs: $\mathbf{W}^1 \in \mathbb{R}^{(k \cdot V) \times \dim(\mathbf{h})}$ $\quad \mathbf{b}^1 \in \mathbb{R}^{\dim(\mathbf{h})}$

Hidden layer with dense inputs: $\mathbf{W}^1 \in \mathbb{R}^{(k \cdot n) \times \dim(\mathbf{h})}$ $\quad \mathbf{b}^1 \in \mathbb{R}^{\dim(\mathbf{h})}$

Output layer (any inputs): $\mathbf{W}^2 \in \mathbb{R}^{\dim(\mathbf{h}) \times V}$ $\quad \mathbf{b}^2 \in \mathbb{R}^{V}$

With V = 10K, n = 300 (word2vec), dim(h) = 300

k = 2 (trigram): $\mathbf{W}^1 \in \mathbb{R}^{20{,}000 \times 300}$ or $\mathbf{W}^1 \in \mathbb{R}^{600 \times 300}$ and $\mathbf{b}_1 \in \mathbb{R}^{300}$

k = 5 (six-gram): $\mathbf{W}^1 \in \mathbb{R}^{50{,}000 \times 300}$ or $\mathbf{W}^1 \in \mathbb{R}^{1500 \times 300}$ and $\mathbf{b}_1 \in \mathbb{R}^{300}$

$\mathbf{W}^2 \in \mathbb{R}^{300 \times 10{,}000}$ $\mathbf{b}^2 \in \mathbb{R}^{10{,}000}$

Six-gram model with one-hot inputs: 27,000,460,000 parameters,
with dense inputs: 3,460,000 parameters

Traditional six-gram model: $10^{4 \times 6} = 10^{24}$ parameters

Naive (one-hot input) neural n-gram model

Advantage over non-neural n-gram model:

— The hidden layer captures **interactions** among context words

— **Increasing the order** of the n-gram requires only a small **linear increase** in the number of parameters.

$\dim(\mathbf{W}^1)$ goes from $(k \cdot \dim(V)) \cdot \dim(\mathbf{h})$ to $((k+1) \cdot \dim(V)) \cdot \dim(\mathbf{h})$

— **Increasing the vocabulary** also leads only to a linear increase in the number of parameters

But: With a one-hot encoding and $\dim(V) \approx 10K$ or so, this model still requires a LOT of parameters to learn.
And: The Markov assumption still holds

Better (dense embeddings input) neural n-gram model

Advantage over non-neural n-gram model:
— Same as naive neural model, plus:

Advantages over naive neural n-gram model:
— We have far **fewer parameters** to learn
— **Better generalizations:** If similar input words have similar embeddings, the model will predict similar probabilities in similar contexts:

$P(w \mid \text{the doctor saw the}) \approx P(w \mid \text{a nurse sees her})$

But: This generalization only works if the contexts have similar words in the *same* position.

And: The Markov assumption still holds.

Neural n-gram models

Naive neural n-gram models (one-hot inputs) have similar shortcomings to standard n-gram models

- Models get very large (and sparse) as n increases
- We can't generalize across similar contexts
- Markov (independence) assumptions are too strict

Better neural n-gram models can be obtained with dense word embeddings:

- Models remain much smaller
- Embeddings may provide some (limited) generalization across similar contexts

1D CNNs for text

Text is a (variable-length) **sequence** of words (word vectors)

[#channels = dimensionality of word vectors]

We can use a **1D CNN** to slide a window of *n* tokens across:

— Filter size n = 3, stride = 1, no padding

The quick brown fox jumps over the lazy dog

The **quick brown fox** jumps over the lazy dog

The quick **brown fox jumps** over the lazy dog

The quick brown **fox jumps over** the lazy dog

The quick brown fox **jumps over the** lazy dog

The quick brown fox jumps **over the lazy** dog

— Filter size n = 2, stride = 2, no padding:

The quick brown fox jumps over the lazy dog

The quick **brown fox** jumps over the lazy dog

The quick brown fox **jumps over** the lazy dog

The quick brown fox jumps over **the lazy** dog

What we have learned today

- ❑Language Modeling
- ❑N-gram is a simple language model
- ❑Word2Vec gives embeddings of a word based on its context
- ❑Neural N-gram Language Model
- ❑Feedforward network and CNN for NLP
- ❑Next lecture: Recurrent Neural Network, Sequence-t-sequence model, and transformer network