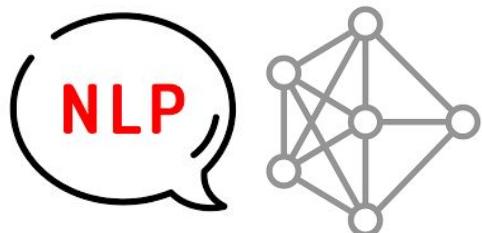




CHULA ENGINEERING
Foundation toward Innovation

COMPUTER



Language Modeling

2110572: Natural Language Processing Systems

Peerapon Vateekul & Ekapol Chuangsawanich

Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

+

Outline

- Introduction
- N-grams
- Evaluation and Perplexity
- Smoothing
- Neural Language Model



Introduction



Introduction

คุณ | อาร์ | กช

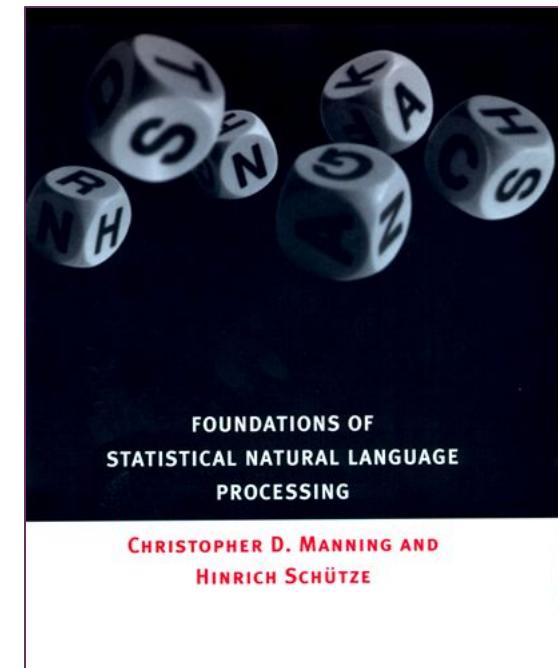
Maximal matching = 3
We need to verify with Language model

คุณ | อาร์ | กช

- **Language Model** (or Probabilistic Language Model for this course) 's goal is
 - (1) to assign probability to a sentence, or
 - (2) to predict the next word
- “Do you live in Bangkok?” and “Live in Bangkok do you?”
 - Which sentence is more likely to occur?

“... the problem is to predict the next word given the previous words. The task is fundamental to speech or optical character recognition, and is also used for spelling correction, handwriting recognition, and statistical machine translation.”

— Page 191, Foundations of Statistical Natural Language Processing, 1999.





Introduction (cont.)

- Application
 - Text Generation
 - Generating new article headlines
 - Generating new sentences, paragraphs, or documents
 - Generating suggested continuation of a sentence
- For example: The Pollen Forecast for Scotland system [Perara R., ECAL2006]
 - Given six numbers of predicted **pollen levels** in different parts of Scotland
 - The system generates **a short textual summary** of pollen levels
 - https://en.wikipedia.org/wiki/Natural_language_generation

- Machine Translation
- Speech Recognition

Generating Spatio-Temporal Descriptions in Pollen Forecasts

Ross Turner, Somayajulu Sripada and Ehud Reiter

Dept of Computing Science,
University of Aberdeen, UK
 {rturner, ssripada, ereiter}@csd.abdn.ac.uk

Ian P Davy

Aerospace and Marine International,
Banchory, Aberdeenshire, UK
 idavy@weather3000.com

Grass pollen levels for Friday have increased from the moderate to high levels of yesterday with values of around 6 to 7 across most parts of the country. However, in Northern areas, pollen levels will be moderate with values of 4. [as of 1-July-2005]



Introduction (cont.)

- How to compute this sentence probability ?
 - $S = \text{"It was raining cat and dog yesterday"}$
 - What is $P(S)$?



Introduction (cont.)

- Conditional Probability and Chain Rule
 - Do you still remember ?

$$P(B|A) = \frac{P(A,B)}{P(B)}$$

$$P(A, B) = P(B|A)P(A)$$

- Chain Rule:

$$P(A, B, C, D) = P(A) \times P(B|A) \times P(C|A, B) \times P(D|A, B, C)$$

- Now, we can write $P(\text{It, was, raining, cat, and, dog, yesterday})$ as :
 - $P(\text{it}) \times P(\text{was} | \text{it}) \times P(\text{raining} | \text{it was}) \times P(\text{cats} | \text{it was raining}) \times P(\text{and} | \text{it was raining cats}) \times P(\text{dogs} | \text{it was raining cats and}) \times P(\text{yesterday} | \text{it was raining cats and dogs})$



Problem with full estimation

- Language is creative.
- New sentences are created all the time.
- ...and we won't be able to count all of them

Training:

<s> I am a student . </s>
<s> I live in Bangkok . </s>
<s> I like to read . </s>

Test:

<s> I am a teacher . </s>

→ $P(\text{teacher}|\text{<s> I am a}) = 0$

→ $P(\text{<s> I am a teacher . </s>}) = 0$

+

N-grams



N-grams: a probability of next word

■ Markov Assumption

- Markov models are the class of probabilistic models that assume we can predict the **probability of some future unit (next word) without looking too far into the past**
- In other word, we can approximate our conditions to unigram, bigrams, trigrams or n-grams
- E.g. Bi-grams
 - $P(F | A, B, C, D, E) \sim P(F | E)$

There are ten students in the **class**.

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- $P(\text{class} | \text{There, are, ten, students, in, the})$
 - *Unigrams* $\sim P(\text{class})$
 - *Bigrams* $\sim P(\text{class} | \text{the})$
 - *Trigrams* $\sim P(\text{class} | \text{in the})$



N-grams (cont.): a probability of the whole sentence

- Now, we can write our sentence probability using **Chain rule (full estimation)**
 $= P(it, was, raining, cats, and, dogs, yesterday)$
 $= P(it) \times P(was | it) \times P(raining | it was) \times P(cats | it was raining) \times P(and | it was raining cats) \times P(dogs | it was raining cats and) \times P(yesterday | it was raining cats and dogs)$
- And, with **Markov assumption (tri-grams)**
 $= P(it, was, raining, cats, and, dogs, yesterday) =$
 $= P(it) \times P(was | it) \times P(raining | it was) \times P(cats | was raining) \times P(and | raining cats) \times P(dogs | cats and) \times P(yesterday | and dogs)$



N-grams (cont.): a probability of the whole sentence – Start & Stop

- And, with **Markov assumption (tri-grams)**

$$= P(it, was, raining, cats, and, dogs, yesterday) =$$

$$= P(it) \times P(was | it) \times P(raining | it was) \times P(cats | was raining) \times P(and | raining cats) \times P(dogs | cats and) \times P(yesterday | and dogs)$$

- And, with **Markov assumption (tri-grams) with start & stop**

$$= P(<\text{s}>, it, was, raining, cats, and, dogs, yesterday, </\text{s}>) =$$

$$= P(<\text{s}>) \times P(it | <\text{s}>) \times P(was | <\text{s}> it) \times P(raining | it was) \times P(cats | was raining) \times P(and | raining cats) \times P(dogs | cats and) \times P(yesterday | and dogs) \times P(</\text{s}> | dogs yesterday)$$



N-grams (cont.): Example

- Estimating Bigrams Probability
 - Assume there are three documents
 - <s> I am Sam </s>
 - <s> Sam I am </s>
 - <s> I am not Sam </s>

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Bigrams Unit	Bigrams Probability
P(I <s>)	= 2/3 = 0.67
P (am I)	= 3/3 = 1.0
P (Sam am)	= 1/3 = 0.33
P (</s> Sam)	= 2/3 = 0.67
P (Sam <s>))	= 1/3 = 0.33
P (I Sam)	= 1/3 = 0.33
P (</s> am)	= 1/3 = 0.33
P (not am)	= 1/3 = 0.33
P (Sam not)	= 1/1 = 1.0

$$P(A, B, C, D, \dots) = P(A) \times P(B|A) \times P(C|A, B) \times P(D|A, B, C)$$

+

N-grams (cont.): Example

■ Estimating Bigrams Probability

- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I am not Sam </s>

Bigrams Unit	Bigrams Probability
P(I <s>)	= 2/3 = 0.67
P (am I)	= 3/3 = 1.0
P (Sam am)	= 1/3 = 0.33
P (</s> Sam)	= 2/3 = 0.67
P (Sam <s>)	= 1/3 = 0.33
P (I Sam)	= 1/3 = 0.33
P (</s> am)	= 1/3 = 0.33
P (not am)	= 1/3 = 0.33
P (Sam not)	= 1/1 = 1.0

Bigrams Unit	Bigrams Probability	
P(I <s>)	= 2/3 = 0.67	14
P (am I)	= 3/3 = 1.0	
P (Sam am)	= 1/3 = 0.33	
P (</s> Sam)	= 2/3 = 0.67	
P(<s>, I, am, Sam, </s>)	= 0.148137	
P (Sam <s>)	= 1/3 = 0.33	
P (I Sam)	= 1/3 = 0.33	
P (am I)	= 3/3 = 1.0	
P (</s> am)	= 1/3 = 0.33	
P(<s>, Sam, I, am , </s>)	= 0.035937	
P(I <s>)	= 2/3 = 0.67	
P (am I)	= 3/3 = 1.0	
P (not am)	= 1/3 = 0.33	
P (Sam not)	= 1/1 = 1.0	
P (</s> Sam)	= 2/3 = 0.67	
P(<s>, I, am, not, Sam, </s>)	= 0.148137	

$$P(A, B, C, D, \dots) = P(A) \times P(B|A) \times P(C|A, B) \times P(D|A, B, C)$$

+

N-grams (cont.): Counting table

15

- Estimating N-grams Probability
- Uni-gram counting

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Bi-grams counting (column given row)
- “i want” → $c(\text{prev, cur}) = c(w_{i-1}, w_i) = c(\text{want, i}) = 827$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

$$P(A, B, C, D, \dots) = P(A) \times P(B|A) \times P(C|A, B) \times P(D|A, B, C)$$

+

N-grams (cont.): Bi-grams probability table

16

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

■ Estimating N-grams Probability

■ Divided by Unigram

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	0	82	1
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

$$P(<\text{s}>, \text{I}, \text{eat}, \text{Chinese}, \text{food}, </\text{s}>) = 1 * 0.0036 * 0.021 * 0.52 * 0.5 = 1.9 \times 10^{-5}$$

$$P(<\text{s}>, \text{I}, \text{spend}, \text{to}, \text{lunch}, </\text{s}>) = 1 * 0.00079 * 0.0036 * 0.0025 * 0.5 = 3.5 \times 10^{-9}$$

Assume $P(\text{I} | <\text{s}>) = 1$, $P(</\text{s}> | \text{food}) = 0.5$, $P(</\text{s}> | \text{lunch}) = 0.5$

From : <https://web.stanford.edu/class/cs124/> by Dan Jurafsky



N-grams (cont.): Log likelihood

- We do everything in log space ($\ln(P(S))$) to
 - Avoid underflow (numbers too small)
 - Also, adding is faster than multiplying

$$\ln(P(A, B, C, D)) = \ln(P(A)) + \ln(P(B|A)) + \ln(P(C|A, B)) + \ln(P(D|A, B, C))$$



Class activity: calculate log likelihood (solution)

Calculate log likelihood of the following sentence:

<s> I eat chinese food </s>

Assume $P(I|<s>) = 1$, $P(</s>|food) = 0.5$, $P(</s>|lunch) = 0.5$

$$\ln(I, \text{eat, Chinese, food}) = \ln(1) + \ln(0.0036) + \ln(0.021) + \ln(0.52) + \ln(0.5) = -10.84$$

$$P(A, B, C, D) = P(A) \times P(B|A) \times P(C|A, B) \times P(D|A, B, C)$$

$$\ln(P(A, B, C, D)) = \ln(P(A)) + \ln(P(B|A)) + \ln(P(C|A, B)) + \ln(P(D|A, B, C))$$



Evaluation

Which model is better?

Evaluation

- We train our model on a **training set**.
- We test the model's performance on data we haven't seen.
 - A **test set** is an unseen dataset that is different from our training set, totally unused.
 - An evaluation metric tells us how well our model does on the test set.
- Sometimes, we allocate some training set to create a **validation set**
 - Which is a pseudo test set, so we can tune performance

Evaluation

- **Extrinsic Evaluation:**
 - Measure the performance of a downstream task (e.g. spelling correction, machine translation, etc.)
 - Cons: Time-consuming
- **Intrinsic Evaluation:**
 - Evaluate the performance of a language model on a hold-out dataset (test set)
 - **Perplexity!**
 - Cons: An intrinsic improvement does not guarantee an improvement of a downstream task, but perplexity often correlates with such improvements
 - Improvement in perplexity should be confirmed by an evaluation of a real task



Perplexity (1)

- **Perplexity** is a quick evaluation metric for language model
- A **better language model** is the one that assigns a higher probability to the test set
 - **Perplexity** can be seen a normalized version of the probability of **the test set**



Perplexity (2)

- Perplexity is the **inverse probability** of the test set, **normalized by the number of words**:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

- **Minimizing** it is the same as maximizing probability
 - **Lower perplexity is better!**



Perplexity (3)

- Perplexity: $\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$

- Logarithmic Version: $b^{-\frac{1}{N} \sum_{i=1}^N \log_b(P(w_i|w_1 \dots w_{i-1}))}$

- Logarithmic Version Intuition:
 - The exponent is number of bits to encode each word

$$2^{-\frac{1}{N} \sum_{i=1}^N \log_2(P(w_i|w_1 \dots w_{i-1}))}$$



Perplexity (4): Intuition of Perplexity

- Perplexity as branching factor:
 - number of possible next words that can follow any word
- Average branching factor:
 - Consider the task of recognizing a string of random digits of length N, given that each of the 10 digits (0-9) occurs with equal probability.
 - How hard is this task?

$$\begin{aligned}
 \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\
 &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\
 &= \frac{1}{10}^{-1} \\
 &= 10
 \end{aligned}$$

Note:
Each of the digits occurs with equal probability: $P = 1/10$

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

10 times

$$P(A, B, C, D) = P(A) \times P(B|A) \times P(C|A, B) \times P(D|A, B, C)$$



Perplexity (5): PP(W) of “I eat chinese food” Bi-grams

- Perplexity: $PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$ or after taking log: $e^{-\frac{1}{N} \sum_{i=1}^N \ln(P(w_i|w_1 \dots w_{i-1}))}$
 - $PP(<\text{s}>, \text{I}, \text{eat}, \text{Chinese}, \text{food}, </\text{s}>)$
 - $= e^{-\frac{1}{5}(\ln(1)+\ln(0.0036)+\ln(0.021)+\ln(0.52)+\ln(0.5))}$
 - $= e^{\frac{1}{5}(10.84)}$
 - $= 8.74$
- Assume $P(\text{I}|<\text{s}>) = 1$, $P(</\text{s}>|\text{food}) = 0.5$, $P(</\text{s}>|\text{lunch}) = 0.5$

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

+

Zeros and Unknown words

Zeros

- **Zeros**

- things that don't occur in the training set
- but occur in the test set

Training set:

... is into health
... is into food
... is into fashion
... is into yoga

Test set:

... is into BNK48
... is into ping-pong

$$P(\text{BNK48} \mid \text{is into}) = 0$$





Zeros (cont.)

- $P(\text{BNK48} \mid \text{is into}) = 0$
- **n-grams with zero probability**
 - mean that we will assign 0 probability to the test set!
- **We cannot compute perplexity**
 - division by zero

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$





Unknown words (UNK)

- Words we have **never seen before in training set**
- Sometimes call **OOV (out of vocabulary)** words
- There are ways to deal with this problem
 - 1) Assign it as a probability of normal word
 - Create a set of vocabulary with **minimum frequency threshold**
 - That is fixed in advanced
 - Or from top n frequency
 - Or words that have frequency more than 1,2,..,v
 - Convert any words in training and testing that is **not in this predefined set**
 - to '**UNK**' token.
 - Simply, deal with UNK word as a normal word
 - 2) Or just define probability of UNK word with constant value

$$p(UNK) = \frac{wc(UNK_{freq=1})}{wc(total)} = \frac{200}{1,000} = 0.2$$

$$p(UNK) = \frac{1}{total\ vocab} = \frac{1}{100} = 0.01$$

+

Smoothing



Smoothing

- Our training data is very sparse, sometimes we **cannot find the n-grams (0)** that we want.
 - In some cases which we do not even have a unigram (a word or OOV) we will use “UNK” token instead

- Notable smoothing techniques
 - Add-one estimation (or Laplace smoothing)
 - Back-off
 - Interpolation
 - Kneser–Ney Smoothing

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

$$\text{Perplexity} = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

ln(0) is undefined!



Smoothing#1: Add-one estimation

■ Add-one estimation (or Laplace smoothing)

- We add one to all the n-grams counts
- For bigram where V is the number of unique word in the corpus:

$$P(S) = \frac{c(w_i, w_{i-1}) + 1}{c(w_{i-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0



	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1



Smoothing#1: Add-one estimation (cont.)

- Add-one estimation (or Laplace smoothing)
 - Pros
 - Easiest to implement
 - Cons
 - Usually perform poorly compare to other techniques
 - The probabilities change a lot if there are too many zeros n-grams
 - useful in domains where the number of zeros isn't so huge

Smoothing#2: Backoff

- Use less context for contexts you don't know about
- Backoff
 - use only the best available n-grams if you have good evidence
 - otherwise backoff!
 - Example:
 - Tri-gram > Bi-grams > Unigram
 - Continue until we get some counts



Smoothing#3: Interpolation

- Interpolation
 - mix unigram, bigram, trigram

$$\hat{P}(w_n | w_{n-2} w_{n-1}) = \lambda_3 P(w_n | w_{n-2} w_{n-1}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_1 P(w_n) + \lambda_0 C$$

- Where C is a constant, often (1/vocabulary) in corpus
- λ is chose from testing on validation data set, and the summation of λ_i is 1 ($\sum \lambda_i = 1$)
- Interpolation is like merging several models



Smoothing#3: Interpolation (cont.)

I	want	to	eat	chinese	food	lunch	spend	Total
2533	927	2417	746	158	1093	341	278	8493
0.2982	0.1091	0.2846	0.0878	0.0186	0.1287	0.0402	0.0327	1.0000

- Interpolation for Bigram

$$\hat{P}(w_n|w_{n-1}) = \lambda_2 P(w_n|w_{n-1}) + \lambda_1 P(w_n) + \lambda_0 C$$

- Where C is a constant,(often =1/vocabulary) in corpus ,and vocabulary size = 1,446

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

$$\begin{aligned}
 P(\text{spend}|\text{eat}) &= \lambda_2 P(\text{spend}|\text{eat}) + \lambda_1 P(\text{spend}) + \lambda_0 C \\
 &= (0.7)(0) + (0.25)(0.0327) + (0.05) (1/1446) \\
 &= 0.00820958
 \end{aligned}$$



Absolute discounting: save some probability mass for the zeros

- Suppose we want to subtract little from a count of 4 to save probability mass for the zeros?
 - How much to subtract?
- Church and Gale (1991)
 - AP newswire dataset
 - 22 million words in training set
 - next 22 million words in validation set
- On average, a bigram that occurred **4 times** in the first 22 million words (training) occurred **3.23 times** in the next 22 million words (validation)
 - So the discrepancy between train & validate of “only this word” is $4 - 3.23 = 0.77$
 - The averaging discrepancy of all words is about 0.75! (called discount, d)

Bigram count in training	Bigram count in validation set
0	0.0000270
1	0.448
2	1.25 (~ -0.75)
3	2.24 (~ -0.75)
4	3.23 (~ -0.75)
5	4.21 (~ -0.75)
6	5.23 (~ -0.75)
7	6.21 (~ -0.75)
8	7.21 (~ -0.75)
9	8.26 (~ -0.75)



Absolute discounting: save some probability mass for the zeros (cont.)

- **Absolute discounting** formalizes this intuition by **subtracting a fixed (absolute) discount d (d=0.75)** from each count.

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1}) P(w)$$

discounted bigram Interpolation weight
 unigram

- **BUT** should we just use the regular unigram?
 - Solution: Kneser–Ney Smoothing

Bigram count in training	Bigram count in validation set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26



Smoothing#4: Kneser–Ney Smoothing

- Kneser–Ney Smoothing
 - Similar to interpolation, but better estimation for probabilities of lower-order grams (like unigram)
 - Ex: *I can't see without my reading ____.*
 - The blank word should be *glasses*, but if we only consider unigram, a word like *Francisco* has higher probability
 - But, *Francisco* always follows *San* (*San Francisco*).
 - We should use continuation probability instead (i.e. how likely a word is a continuation of any word)



Smoothing#4: Kneser–Ney Smoothing (cont.)

- Kneser–Ney Smoothing

- How many word types precede w?
 - $|\{w_i : c(w_i, w) > 0\}|$

- Normalized by total number of word **bigram types**

$$P_{continuation}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$

- If our corpus contains these bigrams

- { San Francisco, San Francisco , San Francisco ,Sun glasses, Reading glasses, Colored glasses }

- $P_{cont}(Francisco) = (1/4) = 0.25$

- $P_{cont}(glasses) = (3/4) = 0.75$

- Now, a word like “Francisco” will have low $P_{continuation}$



Smoothing#4: Kneser–Ney Smoothing (cont.)

$$P_{\text{AbsoluteDiscounting}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1}) P(\overset{\swarrow}{w_i})$$

discounted bigram
 Interpolation weight
 unigram

- Kneser–Ney Smoothing

- In case of bigram,

$$P_{KN}(w_i | w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

- Where
 - d is a constant number, often set to 0.75

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

the normalized discount

a number of word type that can follow w_{i-1}



Smoothing#4: Kneser–Ney Smoothing (cont.)

- Kneser–Ney Smoothing
 - In general n-gram

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(C_{KN}(w_{i-n+1}^i) - d, 0)}{C_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_{i-n+2}^{i-1})$$

$$C_{KN} = \begin{cases} \text{count} & \text{for the highest - order gram} \\ \text{continuation count} & \text{for other lower - order gram} \end{cases}$$

- P_{KN} will continue **recursively** until it reaches unigram.
- Assume tri-grams
 - $P_{KN}(\text{tri-grams}) = \max((C(w_{i-2}, w_{i-1}, w_i) - d), 0) / C(w_{i-2}, w_{i-1}) + \lambda * P_{KN}(\text{bi-grams})$
 - $P_{KN}(\text{bi-grams}) = \max((C_{KN}(w_{i-1}, w_i) - d), 0) / C_{KN}(w_{i-1}) + \lambda * P_{KN}(\text{uni-grams})$
 - $P_{KN}(\text{uni-grams}) = \max((C_{KN}(w_i) - d), 0) / C_{KN}(w_i) + \lambda * (1/V); \quad 1/V = \text{UNK}$

Smoothing Summary

- Summary
 - Add-1 smoothing:
 - OK for text categorization, not for language modeling
 - For very large N-grams like the Web:
 - Backoff
 - The most commonly used method:
 - Interpolation
 - The best method
 - Kneser–Ney smoothing



Reference/Suggested Reading:

Jurafsky, Dan, and James H. Martin. Speech and language processing. Chapter 3.,
<https://web.stanford.edu/~jurafsky/slp3/3.pdf>

+

Neural Language Model



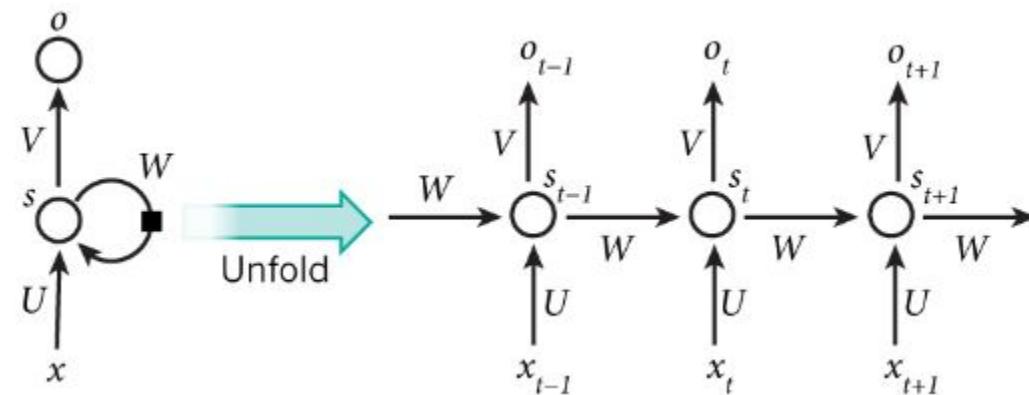
Neural Language Model

- Traditional Language Model
 - Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff (e.g. if 4-gram not found, try 3-gram, etc)
 - However,
 - It need **a lot of memory** to store all those n-grams
 - **It lacks long-term dependency**
 - "Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to __



Neural Language Model (cont.)

- Recurrent Neural Network (RNN)
 - Consider all previous word in the corpus
 - In language modeling,
 - Input (x) is current word in vector form
 - Output (y) is the next word
 - Usually, RNN's performance is better than traditional language model



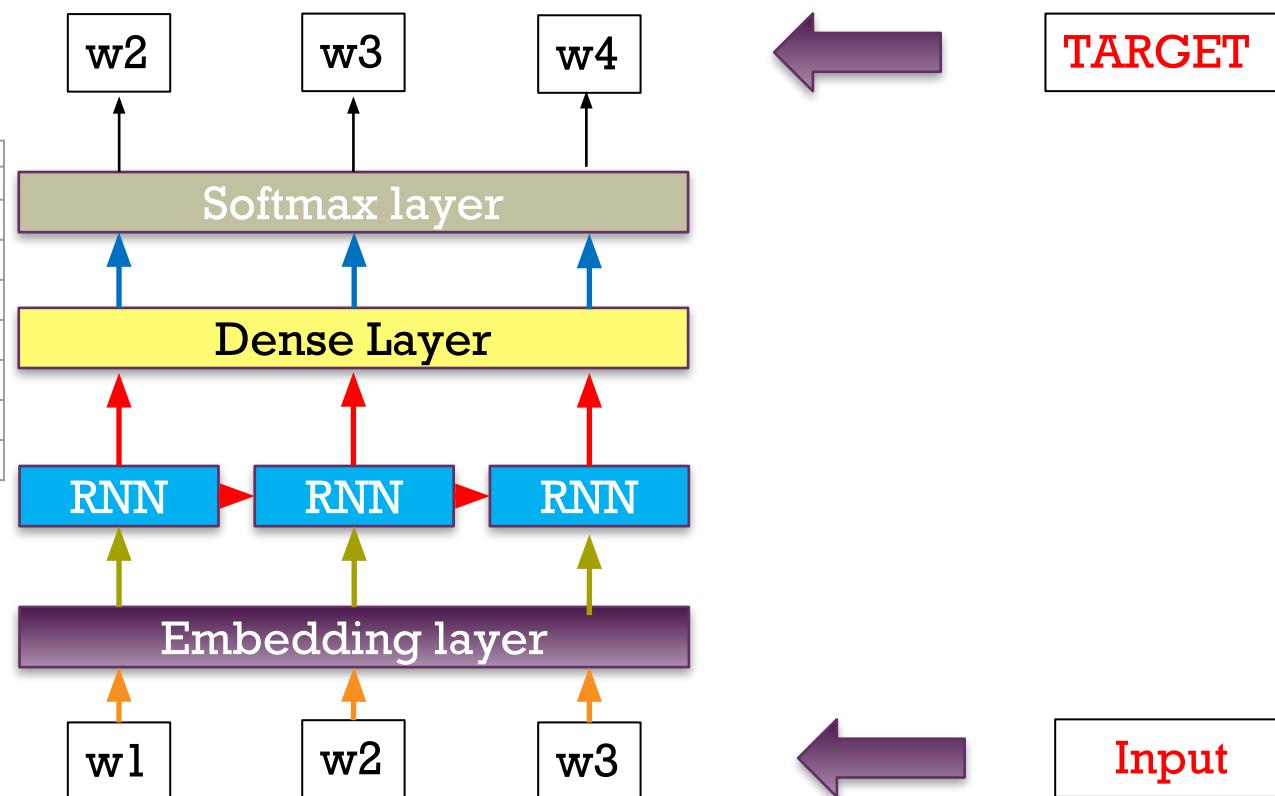


Neural Language Model (cont.)

- Recurrent Neural Network (RNN)
 - A simple language model

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

I eat Chinese food



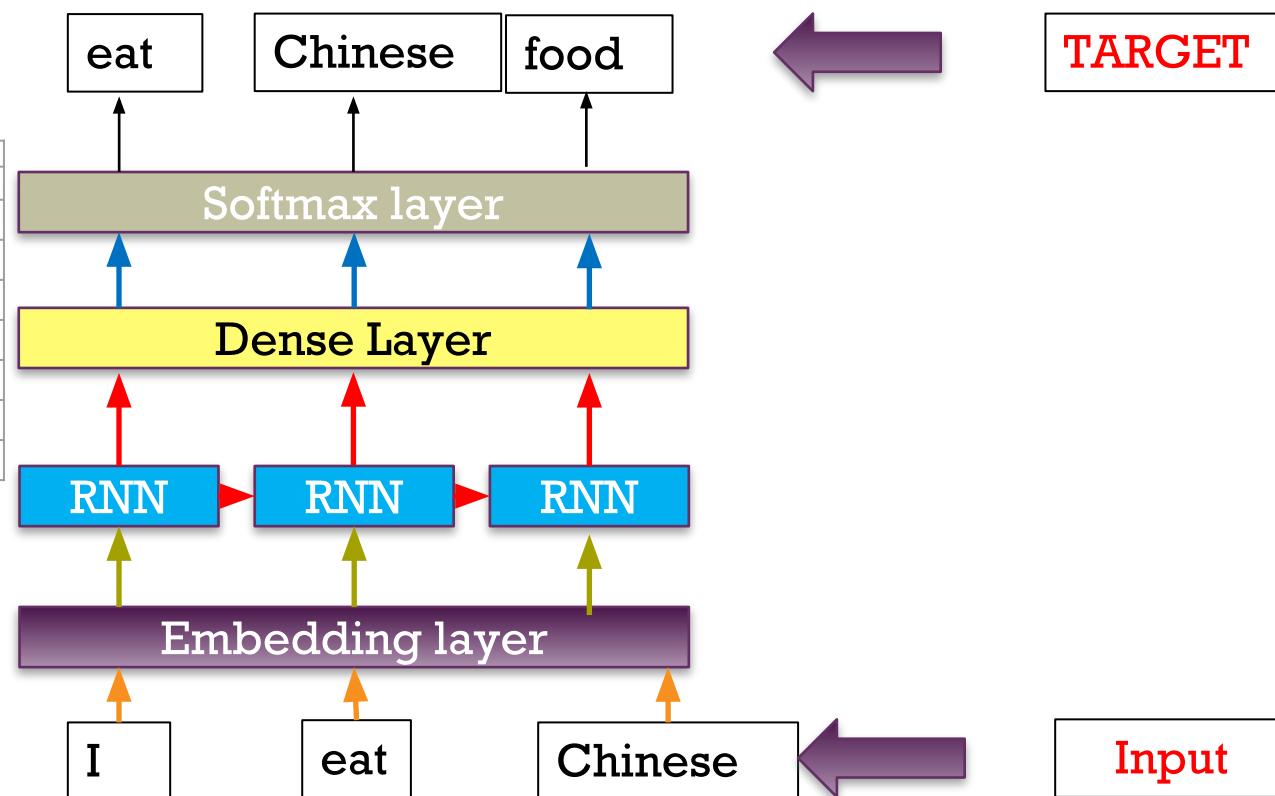


Neural Language Model (cont.)

- Recurrent Neural Network (RNN)
 - A simple language model

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

I eat Chinese food





Neural Language Model (cont.)

■ Recurrent Neural Network (RNN)

- Cost function:

- $$J = -\frac{1}{T} \sum_{t=1}^T \sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

- Where

- V = Number of unique words in corpus
- T = Number of total words in corpus
- y = Target next word
- \hat{y} = Distribution of predicted next word

- Actually, we are calculating perplexity

- Perplexity = e^J

$$\text{Perplexity} = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}},$$

or after taking log : $e^{-\frac{1}{N} \sum_{i=1}^N \ln(P(w_i|w_1 \dots w_{i-1}))}$



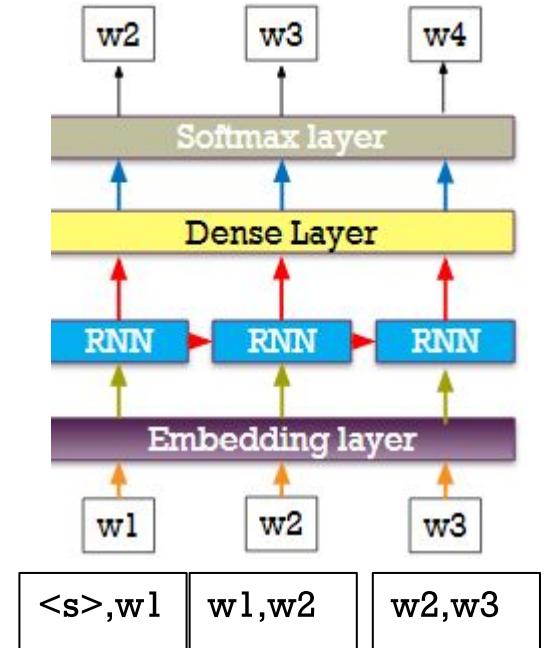
Neural Language Model (cont.)

- RNN suffers from vanishing gradient
 - Use a RNN that has memory unit such as
 - Long Short Term Memory (LSTM)
 - Gate Recurrent Unit (GRU)
- Sometime a future word is important to predict the next word
 - Bidirectional RNN or Bi-RNN can use both past and future words



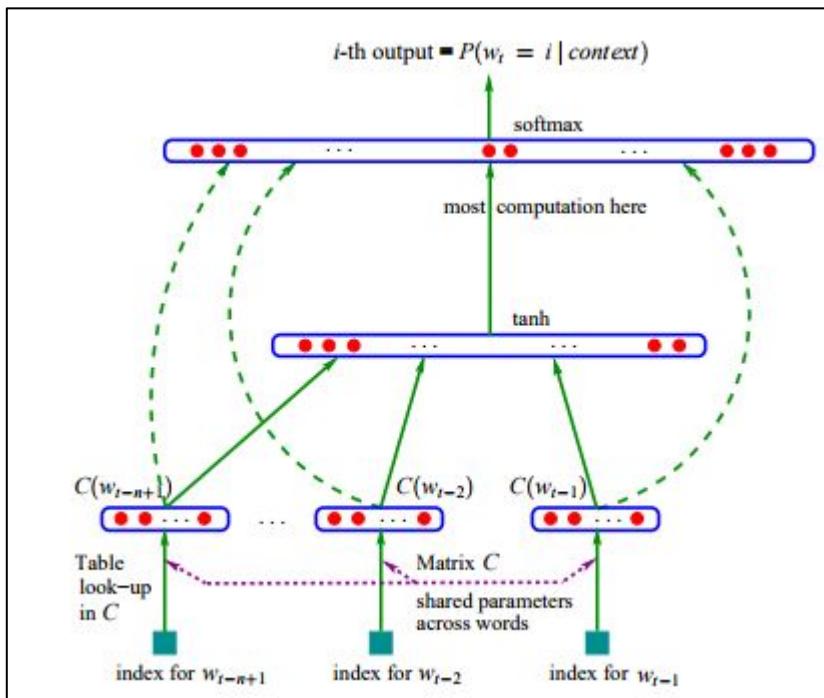
Neural Language Model (cont.)

- Conclusion
- Neural Language Model vs. N-grams Model
 - A competitive n-grams model need huge amount of memory, larger than RNN
 - Neural Language Model usually perform better than n-grams model because
 - it considers long term dependency information
 - It subtlety processes word semantic via word embedding
 - **However**, n-gram is still quite useful and often are incorporated to neural language models



Neural Language Model (cont.)

- [Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. 2003. A neural probabilistic language model. JMLR, 3:1137–1155]
 - This model only use Multilayer Perceptron and Word embedding, **not even RNN**

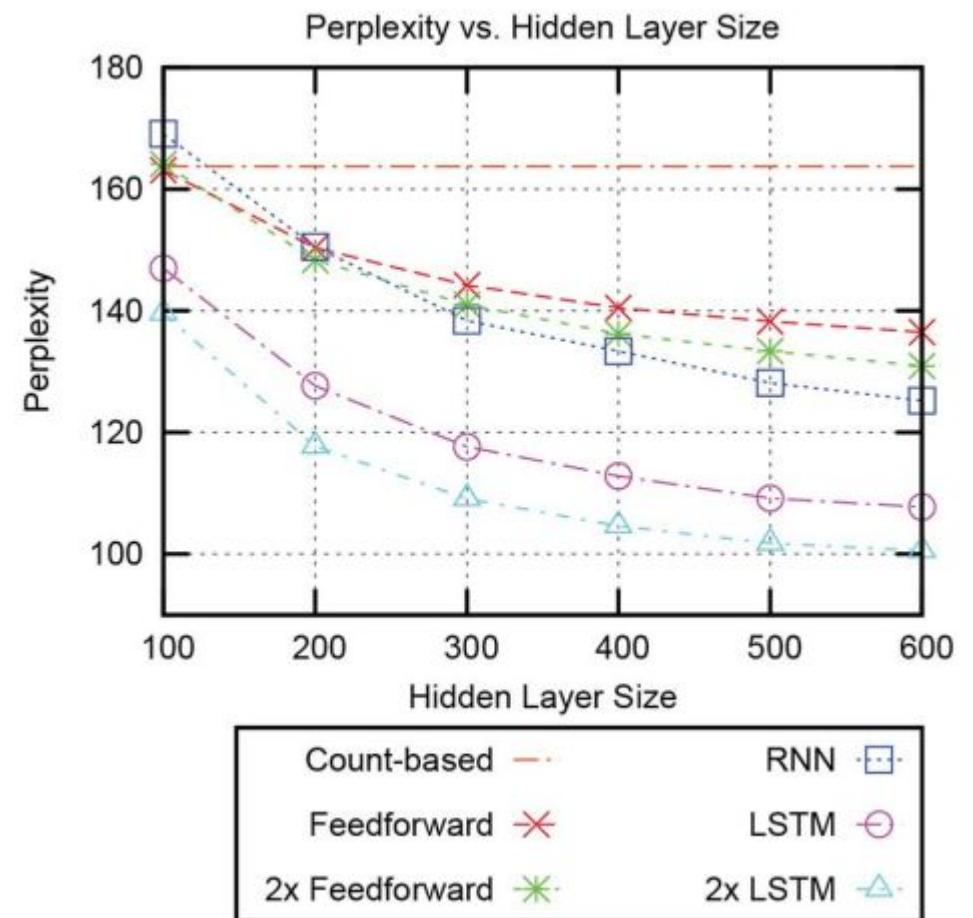




Neural Language Model (cont.)

- [Sundermeyer, Martin, Hermann Ney, and Ralf Schlüter. "From feedforward to recurrent LSTM neural networks for language modeling." *IEEE Transactions on Audio, Speech, and Language Processing* 23.3 (2015): 517-529.]
- LSTM can be used with traditional techniques via interpolation to improve the result

LM	Perplexity	
	Dev	Test
Count-based 4-gram (Reduced)	123.9	144.6
Count-based 4-gram (Full)	102.9	122.0
LSTM	98.6	114.9
+ Count-based 4-gram (Full)	79.9	94.4





Language Model SOTA (2019)

https://github.com/sebastianruder/NLP-progress/blob/master/english/language_modeling.md

1B Words / Google Billion Word benchmark

The One-Billion Word benchmark is a large dataset derived from a news-commentary site. The dataset consists of 829,250,940 tokens over a vocabulary of 793,471 words. Importantly, sentences in this model are shuffled and hence context is limited.

Model	Test perplexity	Number of params	Paper / Source	Code
Transformer-XL Large (Dai et al., 2018) <i>under review</i>	21.8	0.8B	Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context	Official
Transformer-XL Base (Dai et al., 2018) <i>under review</i>	23.5	0.46B	Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context	Official
Transformer with shared adaptive embeddings - Very large (Baevski and Auli, 2018)	23.7	0.8B	Adaptive Input Representations for Neural Language Modeling	Link

