CHULA ENGINEERING
Foundation toward Innovation
COMPUTER

Language Modeling

2110594: Natural Language Processing (NLP)

Peerapon Vateekul & Ekapol Chuangsuwanich
Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

Outline

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

Introduction

Introduction

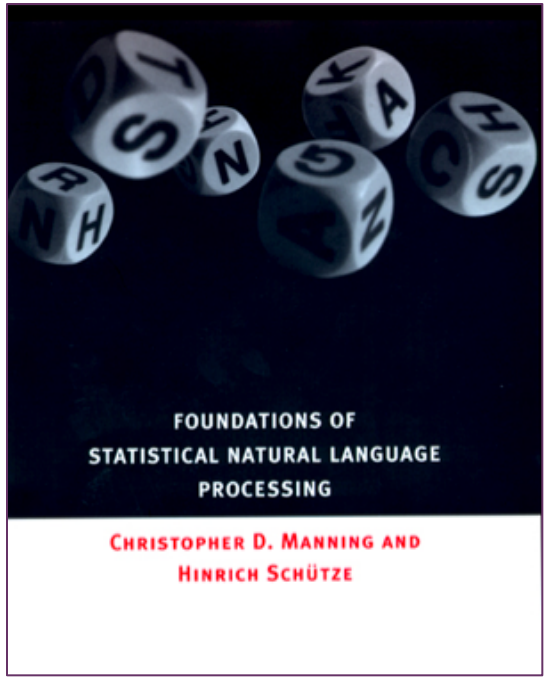
Maximal matching = 3
We need to verify with Language model

คุณ | อากร | กช

คุณ | อา | กรกช

- 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.
> https://machinelearningmastery.com/applications-of-deep-learning-for-natural-language-processing/



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]

QA (after midterm)

Introduction (cont.)

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

Introduction (cont.)

- Conditional Probability and Chain Rule
  - Do you still remember ?
    - $P\ (B \mid A) = \frac{P(A,B)}{P(B)}$
    - $P(A,\ B)\ = P\ (B \mid A)\ P(B)$
  - Chain Rule:
    - $P(A, B, C, D, \dots) = P(A)\ x\ P(B|A)\ x\ P(C|A, B) x\ P(D|A\ , B\ , C)$
  - Now, we can write P(It, was, raining, cat, and, dog, yesterday) as :
    - *P(it) x P(was | it) x P(raining | it was) x P(cats | it was raining) x P(and | it was raining cats) x P(dogs | it was raining cats and) x P(yesterday | it was raining cats and dogs)*

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) ~ P(F| E)*

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

There are ten students in the class.

- *P(class | There, are, ten, students, in, the)*
  - *Unigrams ~ P(class)*
  - *Bigrams ~ P(class | the)*
  - *Trigrams ~ P(class | in the)*

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

- Now, we can write our sentence probability using Chain rule

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

  = *P(it) x P(was | it) x P(raining | it was) x P(cats | it was raining) x P(and | it was raining cats) x P(dogs | it was raining cats and) x P(yesterday | it was raining cats and dogs)*

- And, with Markov assumption (tri-grams)

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

  = *P(it) x P(was | it) x P(raining | it was) x P(cats |was raining) x P(and | raining cats) x P(dogs | cats and) x 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) x P(was | it) x P(raining | it was) x P(cats |was raining) x P(and | raining cats) x P(dogs | cats and) x P(yesterday | and dogs)

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

  = P(<s>, it, was, raining, cats, and, dogs, yesterday, </s>) =

  = P(<s>) x P(it|<s>) x P(was | <s> it) x P(raining | it was) x P(cats |was raining) x P(and | raining cats) x P(dogs | cats and) x P(yesterday | and dogs) x P(</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>

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

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

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

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
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

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

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

N-grams (cont.): Counting table

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

- Estimating N-grams Probability
  - Uni-gram counting

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

  - Bi-grams counting (row given column)

	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

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

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

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

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

- 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
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	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(<s>,I, eat, Chinese, food,</s>) = 1*0.0036 * 0.021 * 0.52*0.5 = 1.9 x 10^{-5}
P(<s>,I, spend, to, lunch,</s>) = 1*0.00079*0.0036*0.0025*0.5 = 3.5 x 10^{-9}

Assume P(I|<s>)=1, P(</s>|food)=0.5, P(</s>|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(I, eat, Chinese, food) = ln(1)+ln(0.0036) + ln(0.021) + ln(0.52)+ln(0.5) = -10.84

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

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

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

Evaluation

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
- **Perplexity** is a quick evaluation metric for language model
  - The testing data should look like the training data

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

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

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

Perplexity

- Intuition of Perplexity
  - "Model A is better than B, if it give higher probability on the target words"
- Perplexity is the inverse probability of the test set, normalized by the number of words
  - $Perplexity = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$, or after taking log : $e^{-\frac{1}{N}\sum_{i=1}^{N} \ln(P(w_i|w_1 \ldots w_{i-1}))}$
    - N is the number of total word in our corpus
    - $P(w_i|w_1 \ldots w_{i-1})$ is the n-grams probability of w_i
  - **Minimizing** it is the same as maximizing probability

Perplexity (cont.)

- Intuition of Perplexity
  - "Model A is better than B, if it give higher probability on the target words"
  - $Perplexity = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$, or after taking log : $e^{-\frac{1}{N}\sum_{i=1}^{N} \ln(P(w_i|w_1 \ldots w_{i-1}))}$

- With the same sentence "I eat Chinese food"
  - Perplexity(<s>, I, eat, Chinese, food, </s>)
    - = $e^{(-(\ln(1) + \ln(0.0036) + \ln(0.021) + \ln(0.52) + \ln(0.5))/ 5)}$
    - = $e^{(-(-10.84) /5)}$
    - = 8.74

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

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)
  - Interpolation
  - Back-off
  - 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

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

ln(0) is undefined!

Smoothing1: Add-one estimation

- Add-one estimation (or Laplace smoothing)
  - We add one to all the n-grams counts
  - $P(S) = \frac{c(w_i, w_{i-1}) + 1}{c(w_{i-1}) + V}$ for bi-gram where V is the number of unique word in corpus

	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

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

Smoothing1: 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

Smoothing2: Interpolation

- Interpolation
  - If we are trying to compute a tri-grams
    - but we have no examples of that particular trigram
      - use lower grams probability
    - $\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_3 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_1 P(w_n) + \lambda_0 C$
      - Where C is a constant, often $1/_{vocabuary}$ 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

Smoothing2: 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
  - $\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_2 P(w_n|w_{n-1}) + \lambda_1 P(w_n) + \lambda_0 C$
    - Where C is a constant, often $^1/_{vocabuary}$ 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

P (spend|eat) $= \lambda_2$P(spend|eat) + λ_1P(spend) + $\lambda_0 C$
= (0.7)(0) + (0.25)(0.0327) +(0.05) (1/1446)
= 0.00820958

Smoothing3: Backoff

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

82

	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

- Backoff
  - Similar to interpolation
  - However, it use only the best available n-grams
    - Tri-gram > Bi-grams > Unigram
    - Continue until we get some counts

	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

$P(spend|eat) = P(spend|eat) \rightarrow \beta P(spend)$
$= 0 \rightarrow (0.0275)(0.0327)$
$= 0.00065175$

Smoothing4: 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) .

Smoothing4: Kneser–Ney Smoothing (cont.)

- Kneser–Ney Smoothing
  - How many word types occur before w_i?
    - $|\{w_i : c(w_i, w) > 0\}|$
  - Normalized by total number of word bigram types
    - $P_{continuation} = \frac{|\{w_i : c(w_i, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$
- If our corpus contains these bigrams
  - { San Francisco, San Francisco , San Francisco ,Sun glasses, Reading glasses, Colored glasses }
  - $P_{continuation}$ (Francisco) = (1/4) = 0.25
  - $P_{continuation}$ (glasses) = (3/4) = 0.75
  - Now, a word like "Francisco" will have low $P_{continuation}$

Smoothing4: Kneser–Ney Smoothing (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

82

	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

- Kneser–Ney Smoothing
  - In case of bigram,
    - $P_{KN}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1},w_i)-d,0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{continuation}(w_{i-1})$
      - Where
        - d is a constant number, often set to 0.25
        - $\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}|\{w: c(w_{i-1},w) > 0\}|$
          - $|\{w: c(w_{i-1},w) > 0\}|$ = a number of word type that can follow w_{i-1}
  - In general n-gram
    - $P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(C_{KN}(w_{i-n+1}^{i})-d,0)}{C_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_{i-n+2}^{i-1})$
      - $C_{KN} = \begin{cases} count \text{ for the highest} - \text{order gram} \\ continuation\ count \text{ for other lower} - \text{order gram} \end{cases}$
      - P_{KN} will continue recursively until it reaches unigram

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

+

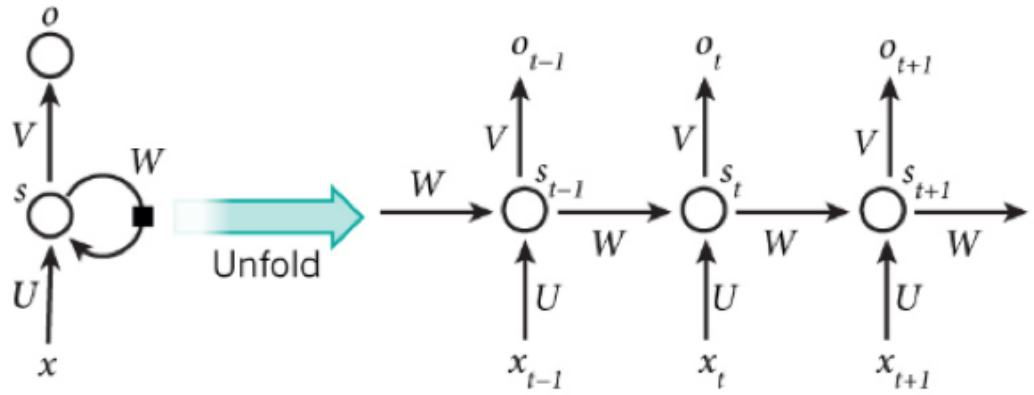
Neural Language Model

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

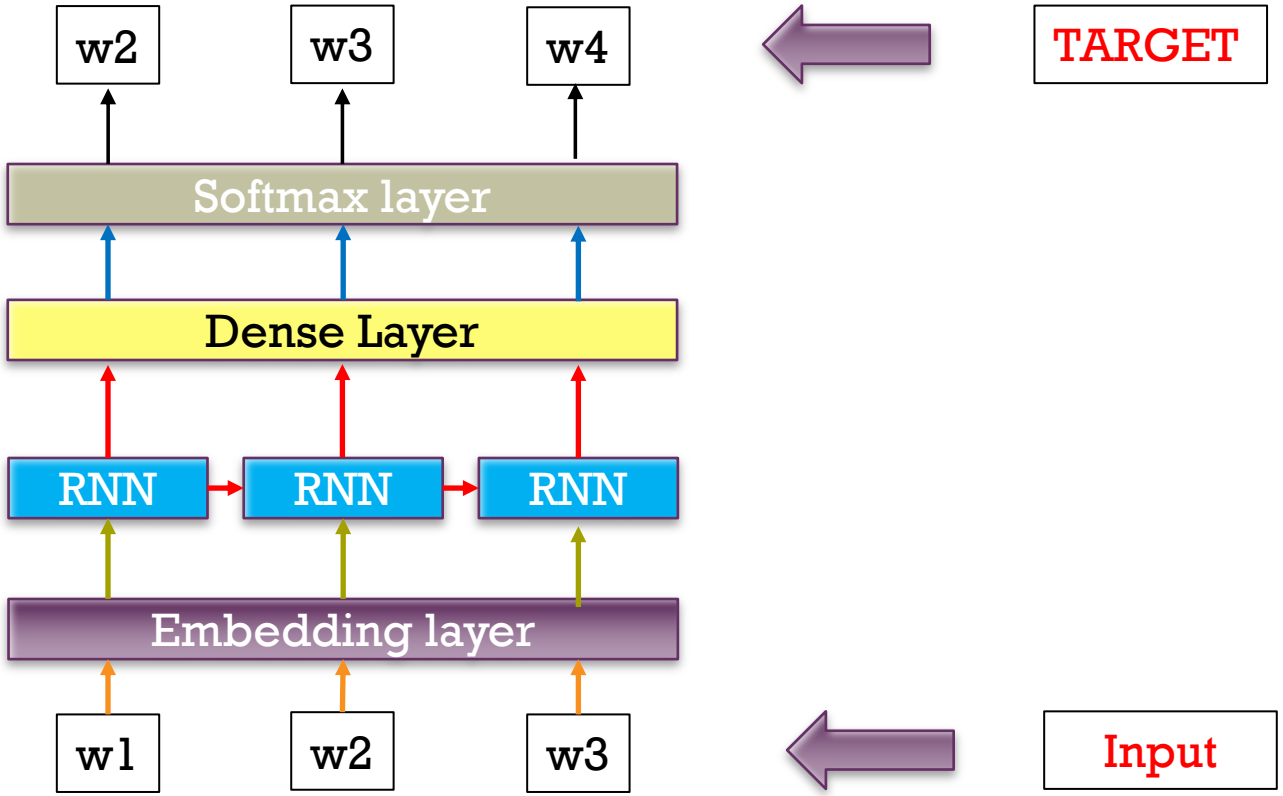


Neural Language Model (cont.)

- 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)
  - 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

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

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

Neural Language Model (cont.)

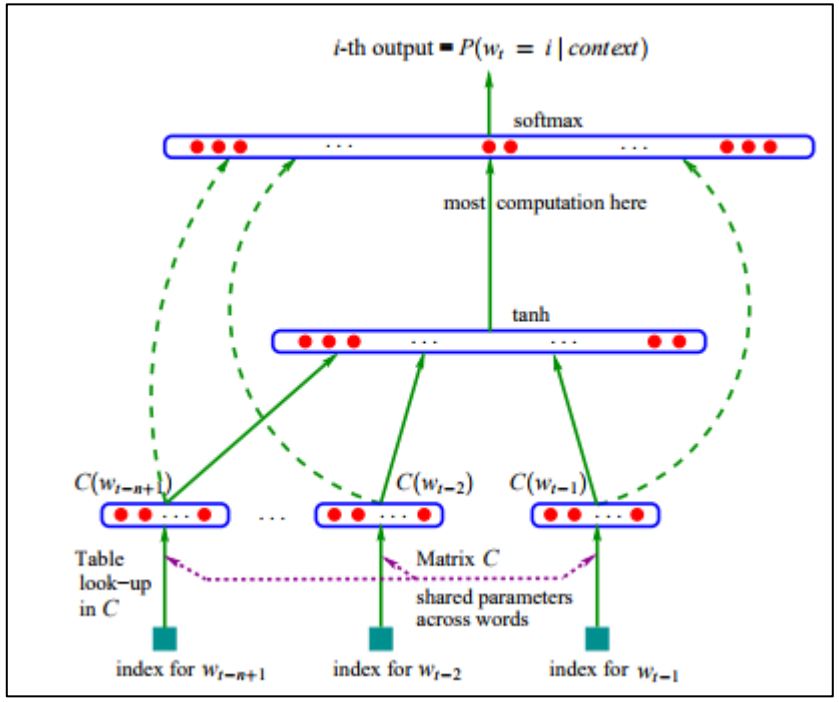
- RNN suffers from vanishing gradient
  - Use a RNN that has memory unit such as
    - Long Short Term Memory (LSTM)
    - Gate Recurrent Unit (GRU)
- Sometime a future word is important to predict the next word
  - Bidirectional RNN or Bi-RNN can use both past and future words

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



	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

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 use 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