CHULA ΣNGINEERING
Foundation toward Innovation
COMPUTER

NLP

Language Modeling

2110572: Natural Language Processing Systems

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

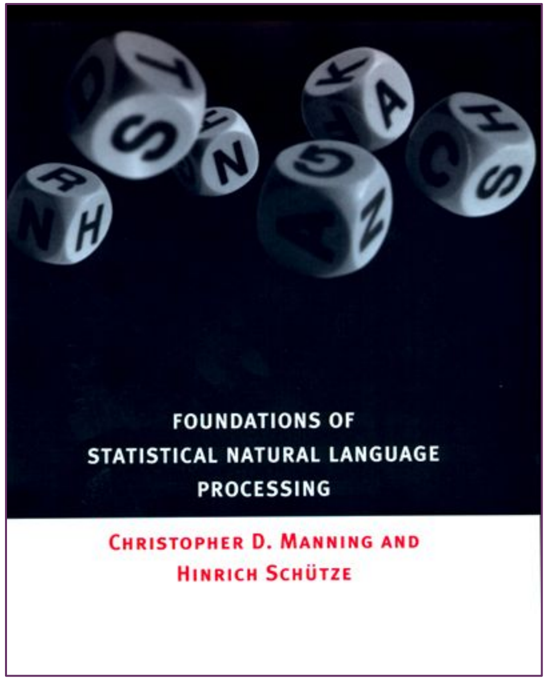
คุณ | อากร | กซ

Maximal matching = 3
We need to verify with Language Model (LM)

คุณ | อา | กรกซ

- 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

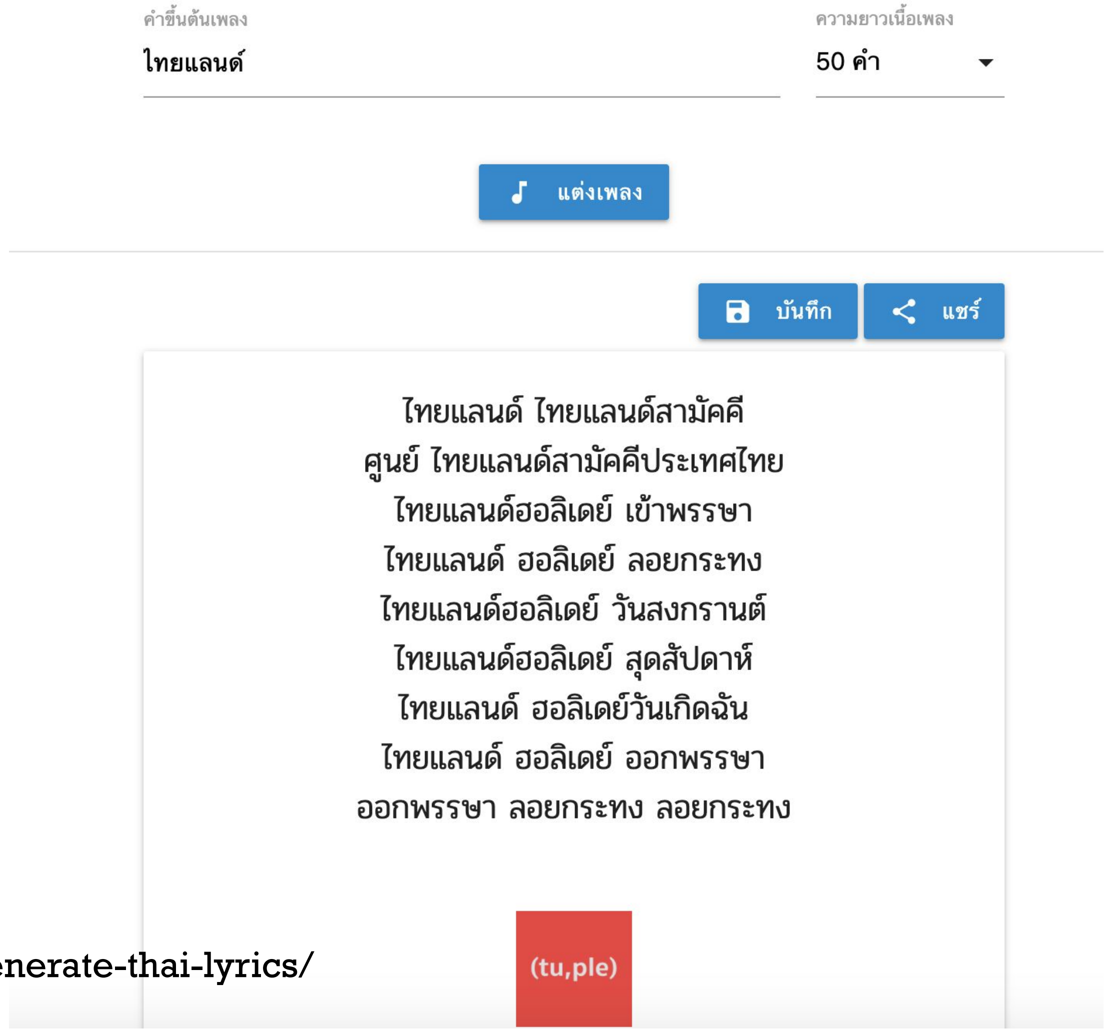
Generating Spatio-Temporal Descriptions in Pollen Forecasts

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]

AI generate Thai songs



https://tupleblog.github.io/generate-thai-lyrics/

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 ?

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

- 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(It, was, raining, cat, and, dog, yesterday) as :
  - P(it) × P(was | it) × P(raining | it was) × P(cats | it was raining) × P(and | it was raining cats) × P(dogs | it was raining cats and) × P(yesterday | 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(teacher|<s> I am a) = 0
→ P(<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) ~ 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 (full estimation)

  = 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) <u>with start & stop</u>

  = *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)*

  - Start tokens give context for start of the sentence
  - End token give an end to the sentence for language generation (sample till end token)
  - P(<s>) is always 1.

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

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

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.): 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

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

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 (column given row)
    - "i want" → c(prev, cur) = $c(w_{i-1}, w_i)$ = c(want, i) = 827

curr

prev

	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

from counting to prob tables

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

- Estimating N-grams Probability
  - Divided by Unigram

i	want	to	eat	chinese	food	lunch	spend
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

Sentence = “i want” & curr = “want”, prev = “i”
p(want | i) = p(i, want) / p(i) = 827 / 2533 = 0.33

	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(<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

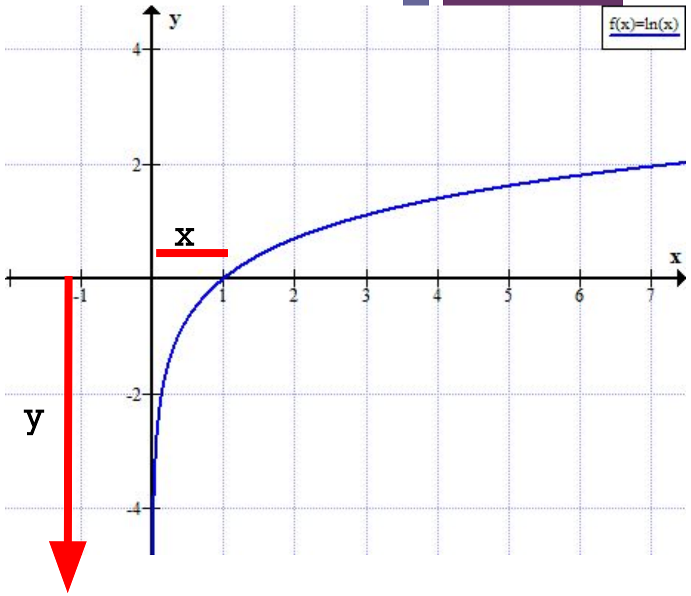
From : https://web.stanford.edu/class/cs124/ by Dan Jurafsky

+ 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

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

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(P(I, 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**:

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

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

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}

Perplexity (3)

- Perplexity: $$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$
- Logarithmic Version: $$b^{-\frac{1}{N}\sum_{i=1}^{N} log_b(P(w_i|w_1 \ldots w_{i-1}))}$$
- Logarithmic Version Intuition:
  - The exponent is number of bits to encode each word

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

+ 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?

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

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

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

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 example

Domain	Size	Type	Perplexity
Digits	11	All word	11
Resource Management	1,000	Word-pair	60
		Bigram	20
Air Travel Understanding	2,500	Bigram	29
		4-gram	22
WSJ Dictation	5,000	Bigram	80
		Trigram	45
	20,000	Bigram	190
		Trigram	120
Switchboard Human-Human	23,000	Bigram	109
		Trigram	93
NYT Characters	63	Unigram	20
		Bigram	11
Shannon Letters	27	Human	~ 2

Perplexity is related to vocabulary size.
Comparing perplexity between different vocabulary size is unfair!

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

- Perplexity: $PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$ or after taking log: $e^{-\frac{1}{N}\sum_{i=1}^{N} ln(P(w_i|w_1 \ldots w_{i-1}))}$
- PP(<s>,I,eat,Chinese,food,</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(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

Zeros and Unknown words

Zeros

- Zeros
  - things that don’t occur in the training set
  - but occur in the test set
  - **and it is still in vocab lists.**

Training set:
… is into health
… is into food
… is into fashion
… is into yoga

Test set:
… is into BNK48
… is into ping-pong

P(BNK48 | is into) = 0

Zeros (cont.)

- P(BNK48| 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 (/0)

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

However, this still cannot solve the zero issue.

+ Unknown words (UNK)

	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

- Words we have never seen before in training set and **not in vocab list**
- Sometimes call OOV (out of vocabulary) words
- There are ways to deal with this problem
  - 1) Assign it as a probability of normal word
    - Step1) 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
    - Step2) 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

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

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

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

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

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

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

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

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

- 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 ($\Sigma\lambda_i=1$)
- 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

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

  - 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

P (spend|eat) $= \lambda_2 P(\text{spend}|\text{eat}) + \lambda_1 P(\text{spend}) + \lambda_0 C$

“eat **spend**” $= (0.7)(0) + (0.25)(0.0327) + (0.05)(1/1446)$

$= 0.00820958$

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 and give to zero counts.

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \underbrace{\frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})}}_{\text{discounted bigram}} + \underbrace{\lambda(w_{i-1})}_{\text{Interpolation weight}} \underbrace{P(w)}_{\text{unigram}}$$

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

	a	b	c
a	10	0	0
b			
c			

	a	b	c
a	9/10	?	?
b			
c			

P(b) = 0.1, P(c) = 0.3
P(b|a) = 0 + xP(b)
P(c|a) = 0 + xP(c)
xP(b) + xP(c) = 0.1

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.)

discounted bigram

Interpolation weight

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

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

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

the normalized discount

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

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

- Kneser–Ney Smoothing
  - In general n-gram

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

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

- P_{KN} will continue **recursively** until it reaches unigram.

- Assume tri-grams
  - Pkn(tri-grams) = max((C(wi-2, wi-1, wi)-d),0)/C(wi-2, wi-1) + lambda*Pkn(bi-grams)
  - Pkn(bi-grams) = max((Ckn(wi-1, wi)-d),0)/Ckn(wi-1) + lambda*Pkn(uni-grams)
  - Pkn(uni-grams) = max((Ckn(wi)-d),0)/Ckn(w) + lambda*(1/V); 1/V=UNK

+ Example: a bigram Kneser-ney

Imagine we have the following training corpus:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I am Sam </s>

<s> I like green eggs </s>

Train a bigram Kneser-ney model using the corpus above

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

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

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

Example: a bigram Kneser-ney (cont.)

training corpus:
<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I like green eggs </s>

Create a unigram counting table

<s>	I	am	Sam	like	green	eggs	</s>
4	4	3	3	1	1	1	4

Example: a bigram Kneser-ney (cont.)

Create a bigram counting table

training corpus:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I am Sam </s>

<s> I like green eggs </s>

	<s>	I	am	Sam	like	green	eggs	</s>
<s>	0	3	0	1	0	0	0	0
I	0	0	3	0	1	0	0	0
am	0	0	0	2	0	0	0	1
Sam	0	1	0	0	0	0	0	2
like	0	0	0	0	0	1	0	0
green	0	0	0	0	0	0	1	0
eggs	0	0	0	0	0	0	0	1
</s>	0	0	0	0	0	0	0	0

<s>	I	am	Sam	like	green	eggs	</s>
4	4	3	3	1	1	1	4

+ Example: a bigram Kneser-ney (cont.)

Compute the log-likelihood of the sentence "<s> am Sam </s>"

Pkn2(am|<s>)=(max(0-0.75,0)/4)+(0.75*2/4)*(1/11) =0.03409

Pkn2(Sam|am)=(max(2-0.75,0)/3)+(0.75*2/3)*(2/11) =0.5076

Pkn2(</s>|Sam)=(max(2-0.75,0)/3)+(0.75*2/3)*(3/11)=0.5530

LL = ln(0.03409) + ln(0.5076) + ln(0.5530) = -4.6492

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

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

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

	<s>	I	am	Sam	like	green	eggs	</s>
<s>	0	3	0	1	0	0	0	0
I	0	0	3	0	1	0	0	0
am	0	0	0	2	0	0	0	1
Sam	0	1	0	0	0	0	0	2
like	0	0	0	0	0	1	0	0
green	0	0	0	0	0	0	1	0
eggs	0	0	0	0	0	0	0	1
</s>	0	0	0	0	0	0	0	0

training corpus:
<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I like green eggs </s>

<s>	I	am	Sam	like	green	eggs	</s>
4	4	3	3	1	1	1	4

Example: a bigram Kneser-ney (cont.)

Compute the perplexity of the sentence "<s> am Sam </s>"

Perplexity = exp(-LL/n) = exp(-(-4.6492) / 3) = 4.7

Smoothing Summary

- Summary
  - 1) Add-1 smoothing:
    - OK for text categorization, not for language modeling
  - For very large N-grams like the Web:
    - 2) Backoff
  - The most commonly used method:
    - 3) Interpolation
  - The best method
    - 4) 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

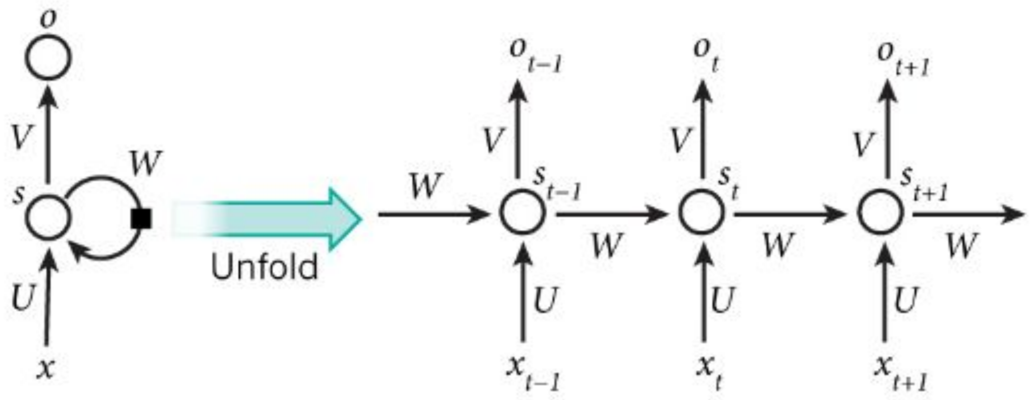
+ Neural Language Model

- 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 ___

	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

+ 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

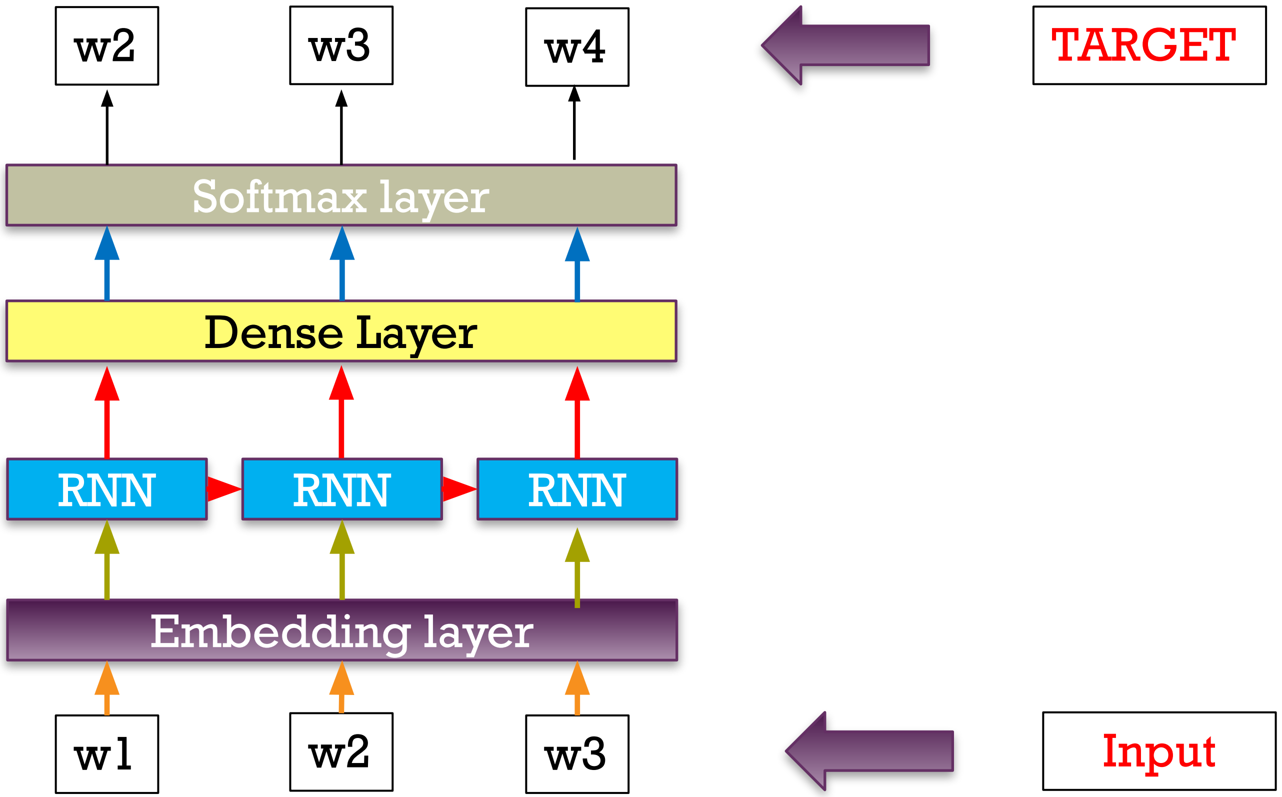


Neural Language Model (cont.)

- 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

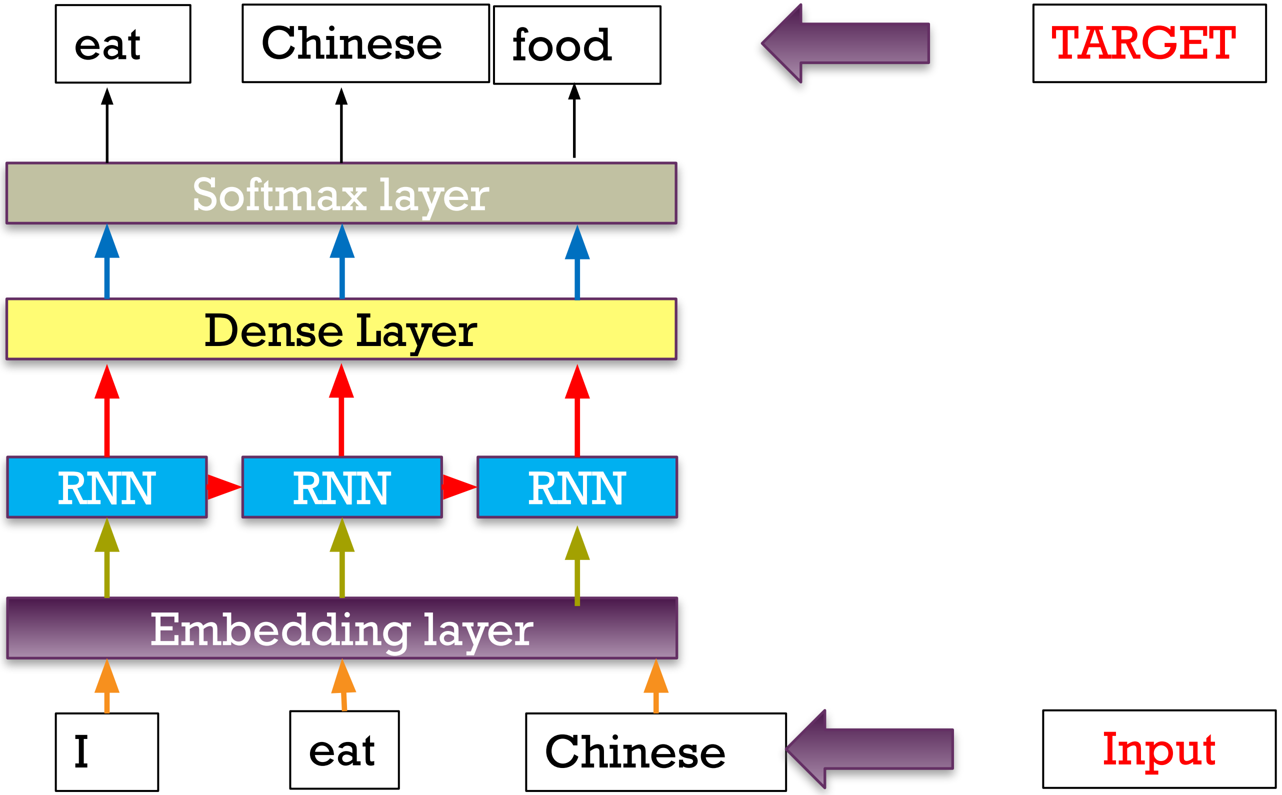


+ Neural Language Model (cont.)

- 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



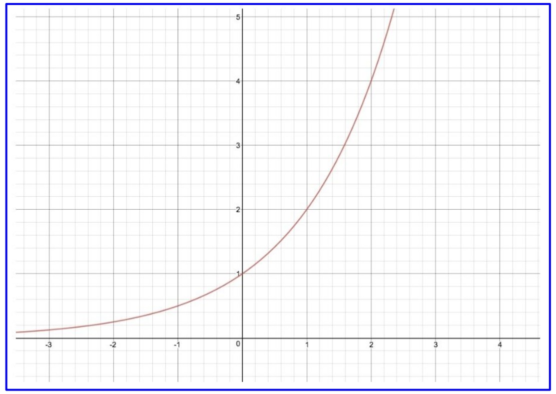
Neural Language Model (cont.)

- Recurrent Neural Network (RNN)
  - Cost function:
    - For each training example, Whole training data (T)

      Softmax (all classes V)

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

      - 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}

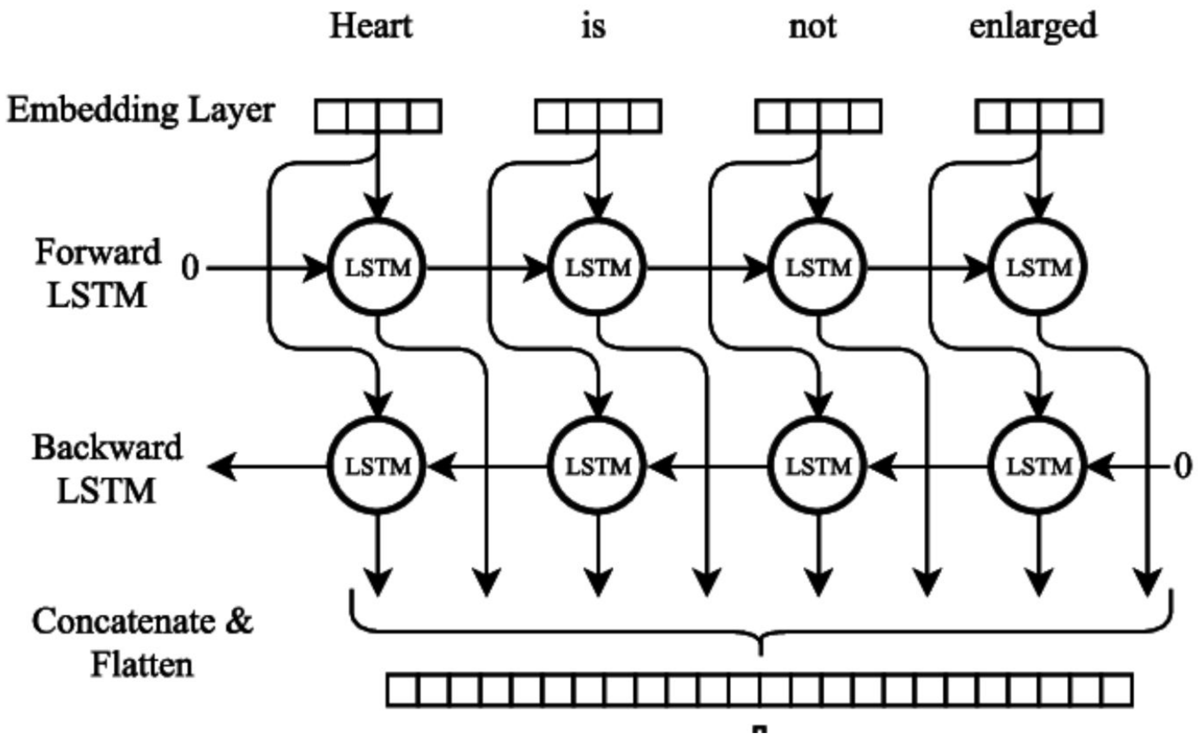


$$Perplexity = \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P(w_i|w_1 ... w_{i-1})}},$$

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

+ 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)
- Bidirectional RNN?
  - Bidirectional RNN cannot apply here since we predict the next word and cannot use future information (violating assumption).
  - However, special types of Bi-RNN (ELMO) or special networks (Transformer: BERT) can be applied without violating assumption.

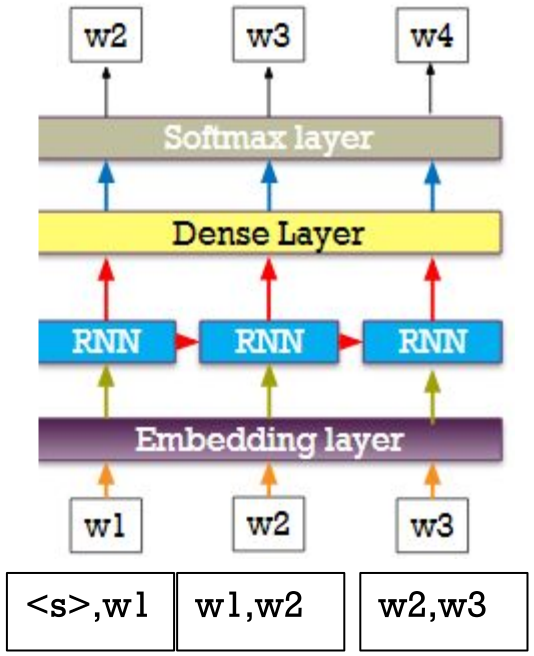


https://paperswithcode.com/method/bilstm#

Neural Language Model (cont.)

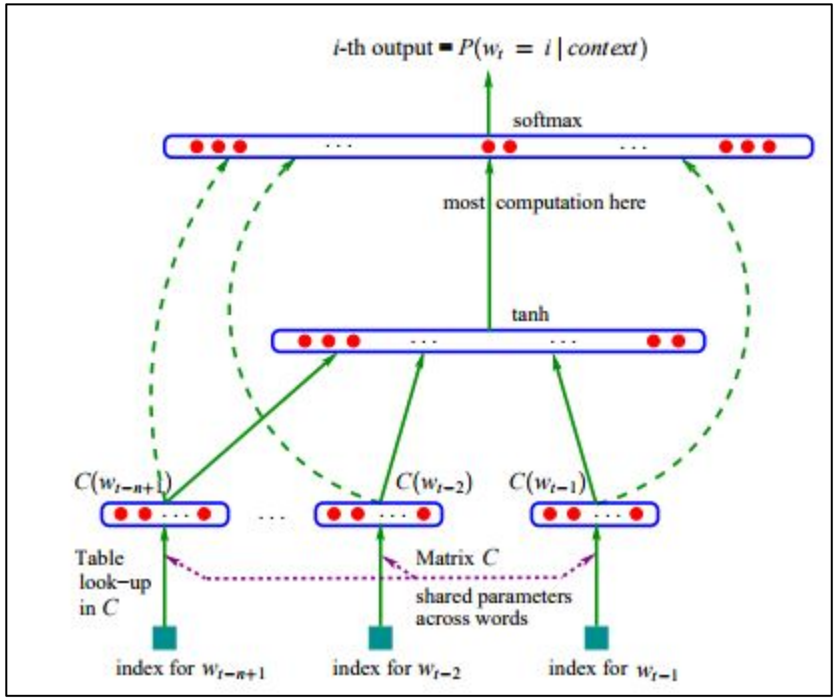
- Conclusion

- 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 as features or for beamsearch pruning.



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**

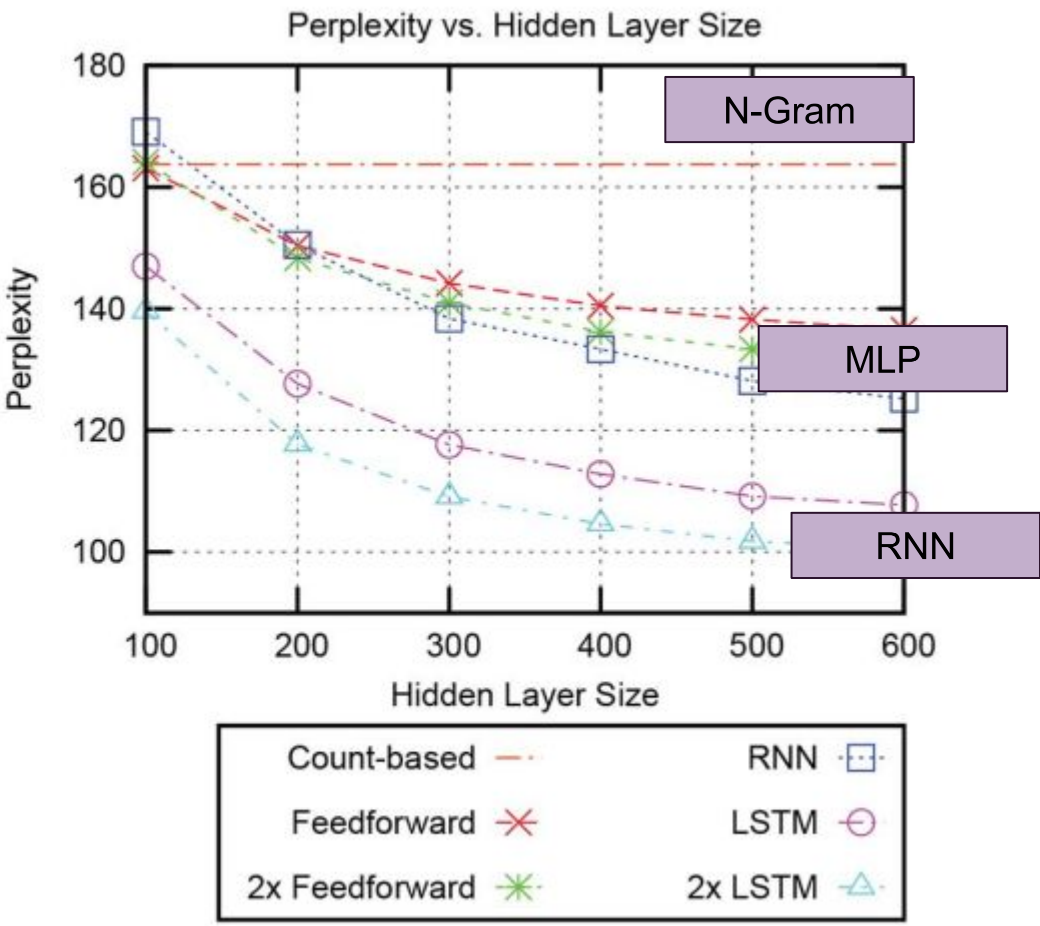


	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		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	Perplexity 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; outdated)

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) ***under review***	21.8	0.8B	Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context	Official
Transformer-XL Base (Dai et al., 2018) ***under review***	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

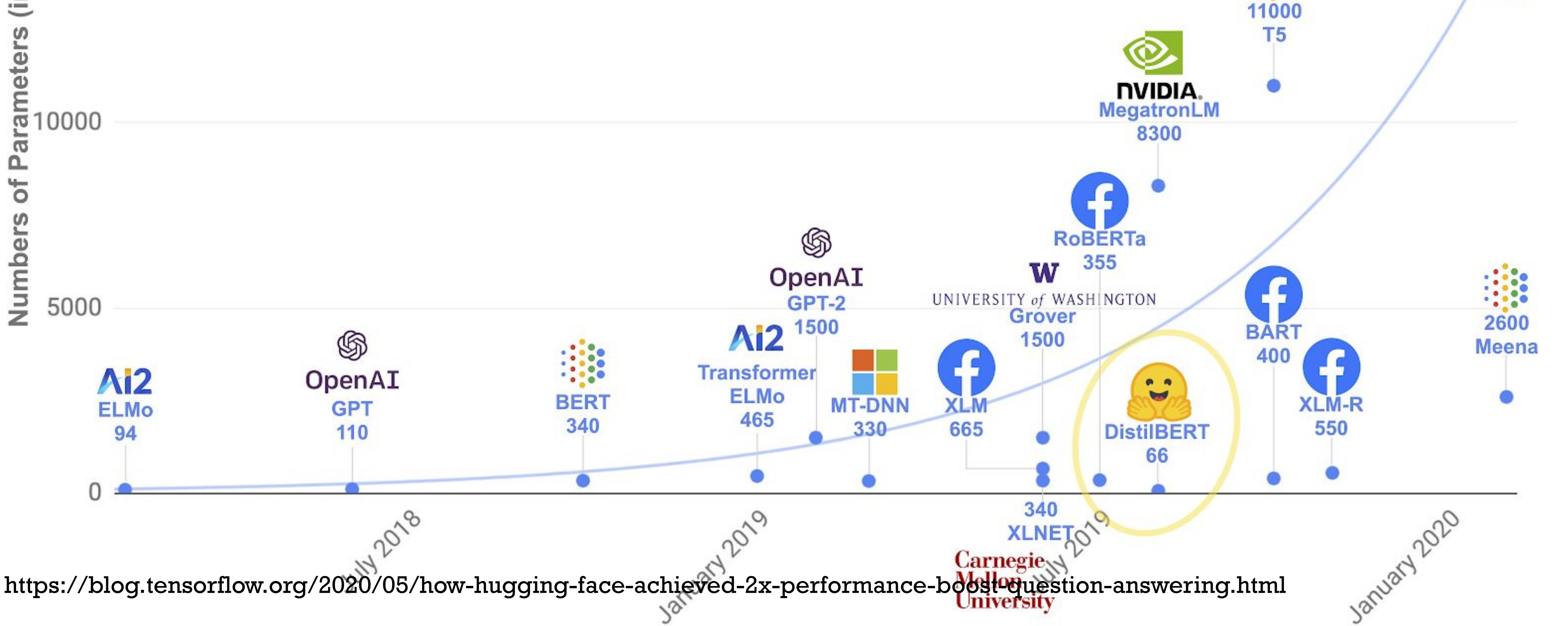
Outdated

SWITCH TRANSFORMERS: SCALING TO TRILLION PARAMETER MODELS WITH SIMPLE AND EFFICIENT SPARSITY

William Fedus* Google Brain liamfedus@google.com

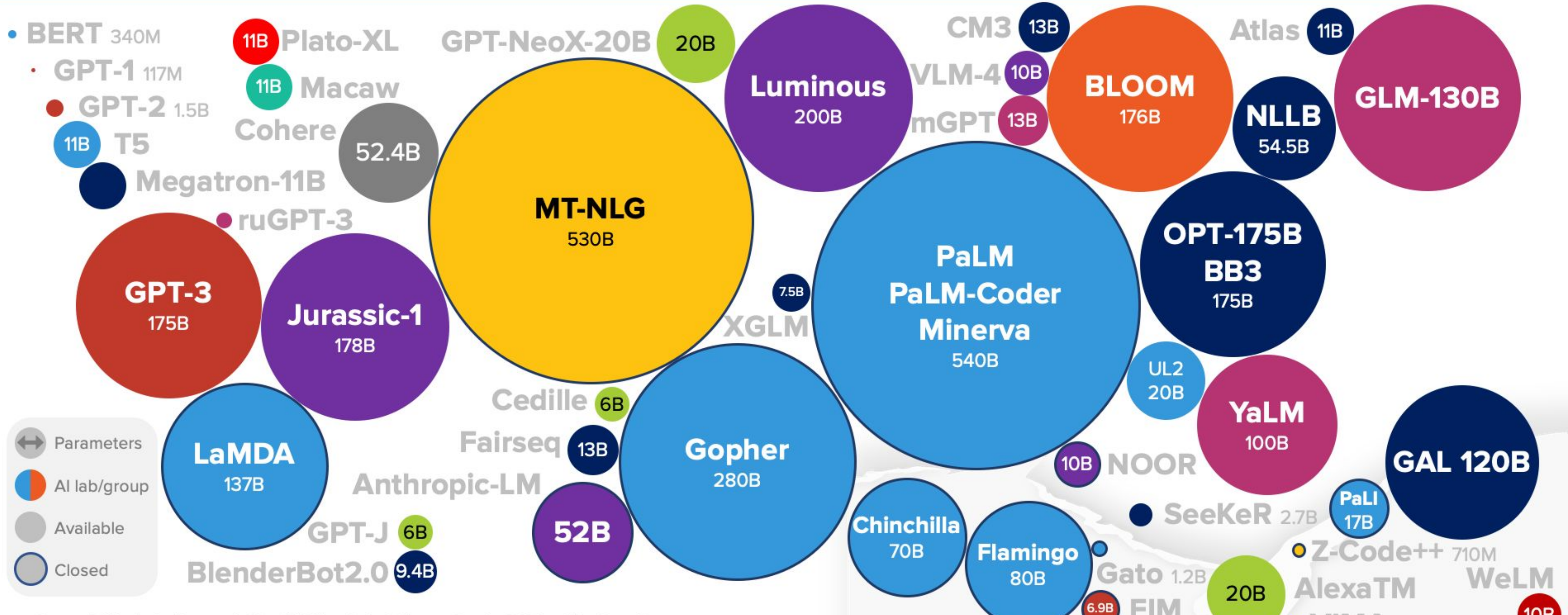
Barret Zoph* Google Brain barretzoph@google.com

Noam Shazeer Google Brain noam@google.com



https://blog.tensorflow.org/2020/05/how-hugging-face-achieved-2x-performance-boost-question-answering.html https://lifearchitect.ai/models/

LANGUAGE MODEL SIZES TO DEC/2022



Beeswarm/bubble plot, sizes linear to scale. Selected highlights only. Alan D. Thompson. December 2022. https://lifearchitect.ai/

LifeArchitect.ai/models

Choose a GPU, AMP mode, and budget:

GPU

V100 ▼

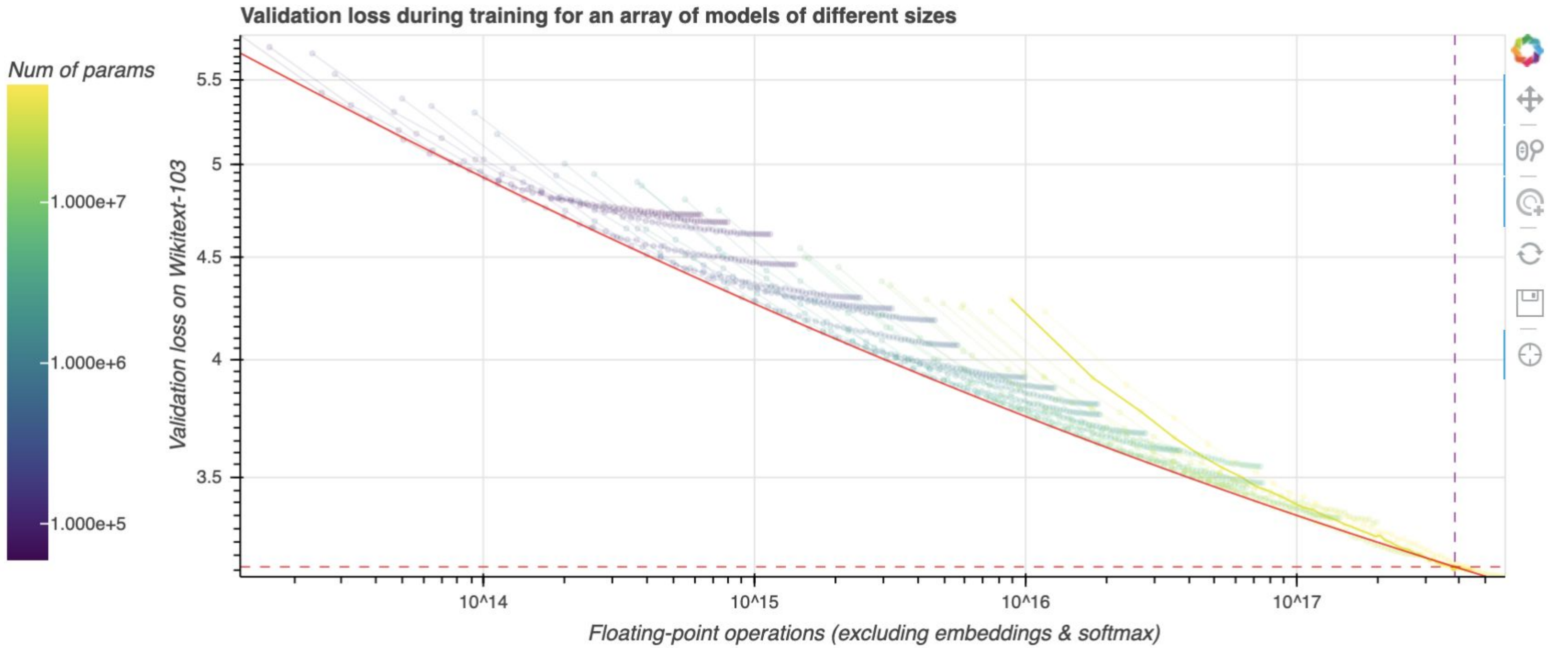
AMP mode

O0 ▼

Wall time (hours): **13.51**

Budget (dollars): **27.08**

This will consume about **1.61** kWh, releasing **0.86** kgs of CO2. That is equivalent to **3.44** kms with an average American passenger car and could be offset by growing a tree for **52.36** days.[1]



Expected wt-103 validation loss:

3.16

Optimal number of non-embedding parameters:

5.35e+07

For example, this could be a model of

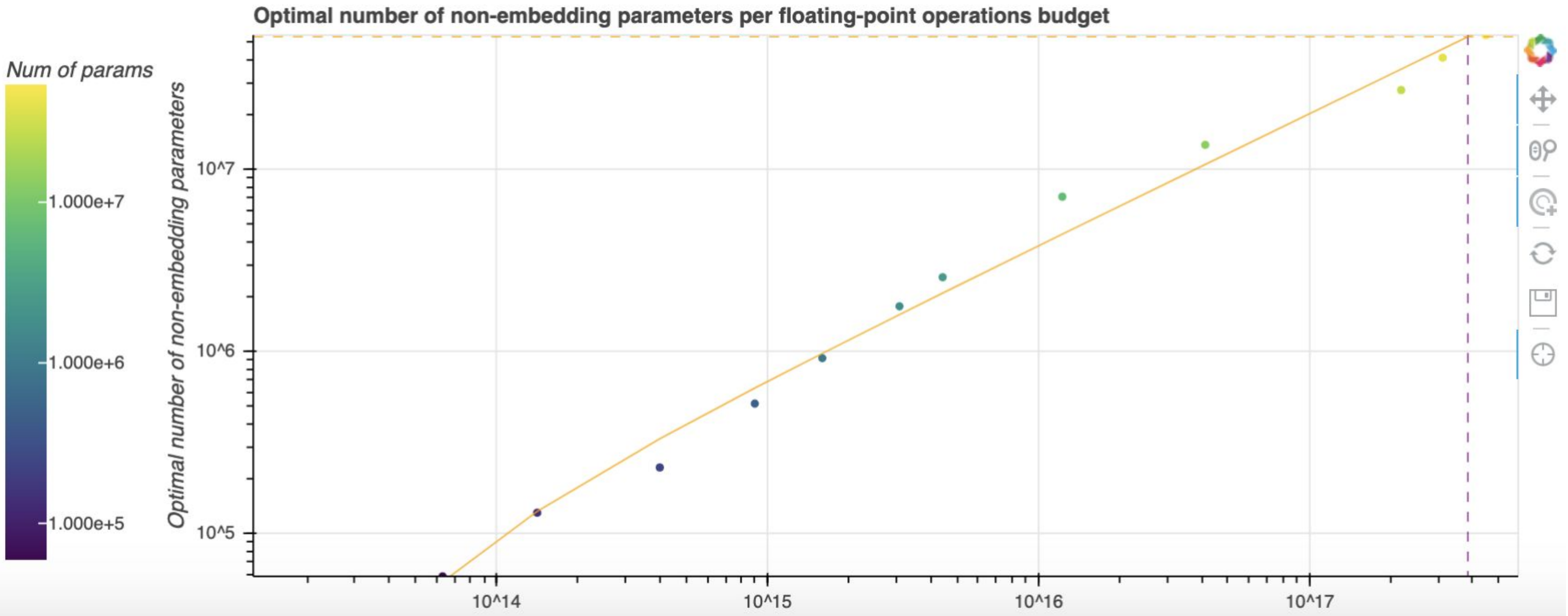
7 layers of 1048 dimensions

Initialize in 🤗 transformers!

Or a model of

13 layers of 768 dimensions

Initialize in 🤗 transformers!



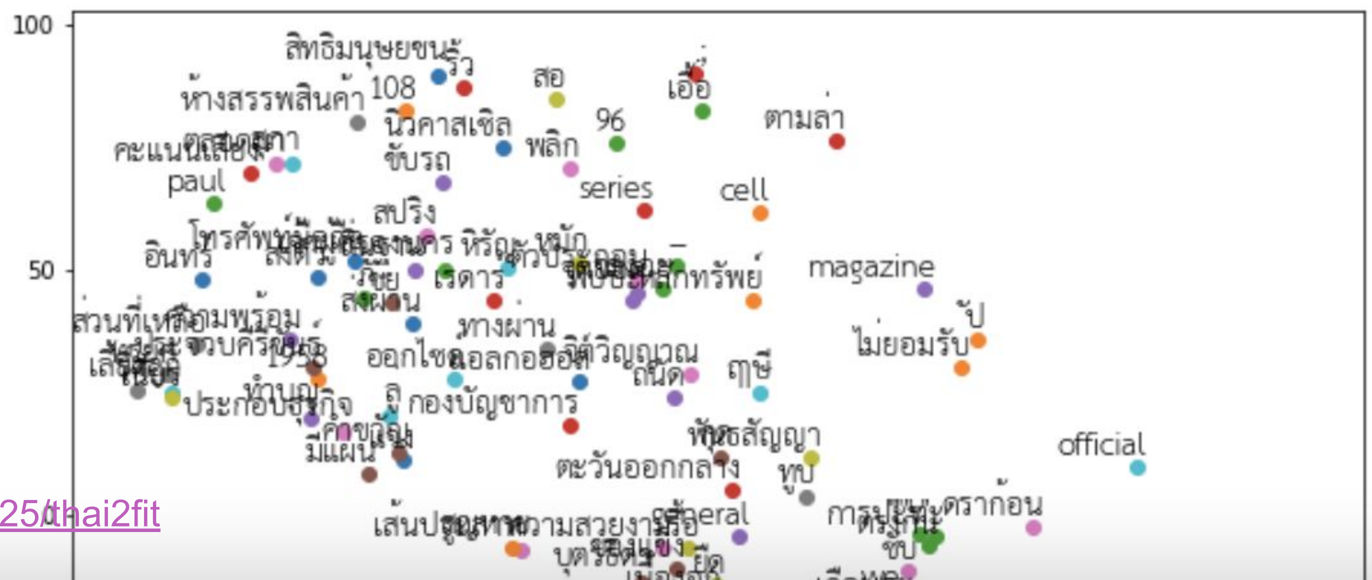
https://huggingface.co/calculator/

thai2fit (formerly thai2vec)

ULMFit Language Modeling, Text Feature Extraction and Text Classification in Thai Language. Created as part of pyThaiNLP with ULMFit implementation from fast.ai

Models and word embeddings can also be downloaded via Dropbox.

We pretrained a language model with 60,005 embeddings on Thai Wikipedia Dump (perplexity of 28.71067) and text classification (micro-averaged F-1 score of 0.60322 on 5-label classification problem. Benchmarked to 0.5109 by fastText and 0.4976 by LinearSVC on Wongnai Challenge: Review Rating Prediction. The language model can also be used to extract text features for other downstream tasks.



https://github.com/cstorm125/thai2fit

https://medium.com/airesearch-in-th/wangchanberta-%E0%B9%82%E0%B8%A1%E0%B9%80%E0%B8%94%E0%B8%A5%E0%B8%9B%E0%B8%A3%E0%B8%B0%E0%B8%A1%E0%B8%A7%E0%B8%A5%E0%B8%9C%E0%B8%A5%E0%B8%A0%E0%B8%B2%E0%B8%A9%E0%B8%B2%E0%B9%84%E0%B8%97%E0%B8%A2%E0%B8%97%E0%B8%B5%E0%B9%88%E0%B9%83%E0%B8%AB%E0%B8%8D%E0%B9%88%E0%B9%81%E0%B8%A5%E0%B8%B0%E0%B8%81%E0%B9%89%E0%B8%B2%E0%B8%A7%E0%B8%AB%E0%B8%99%E0%B9%89%E0%B8%B2%E0%B8%97%E0%B8%B5%E0%B9%88%E0%B8%AA%E0%B8%B8%E0%B8%94%E0%B9%83%E0%B8%99%E0%B8%82%E0%B8%93%E0%B8%B0%E0%B8%99%E0%B8%B5%E0%B9%89-d920c27cd433

Image by Phannisa Nirattiwongsakorn

WangchanBERTa โมเดลประมวลผลภาษาไทยที่ใหญ่และก้าวหน้าที่สุดในขณะนี้

VISTEC-depa AI Research Institute of Thailand Follow

Jan 24 · 5 min read

เราใช้เวลากว่า 3 เดือนในการเทรนโมเดลให้ loss ลดลงมาในระดับที่ 2.592 (perplexity = 13.356) ณ step ที่ 360,000 จากทั้งหมด 500,000 steps ณ วันนี้โมเดลก็ยังถูกเทรนอย่างต่อเนื่อง ในศูนย์วิจัยที่วังจันทร์ จึงเป็นไปได้ว่าเราจะได้โมเดลที่มีประสิทธิภาพดียิ่งกว่ามาใช้ในอนาคต

Conclusion

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

Appendix