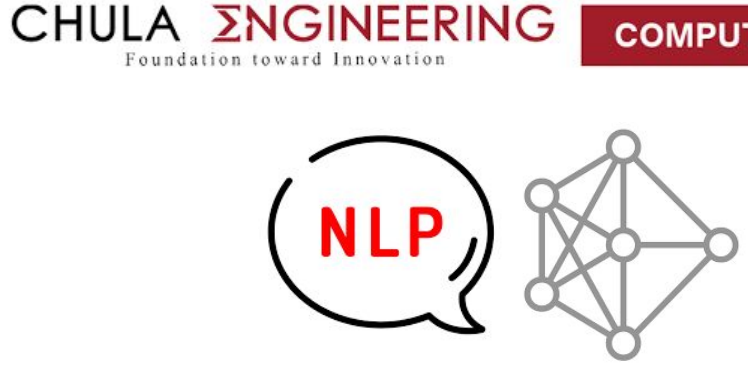


Language Modeling

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

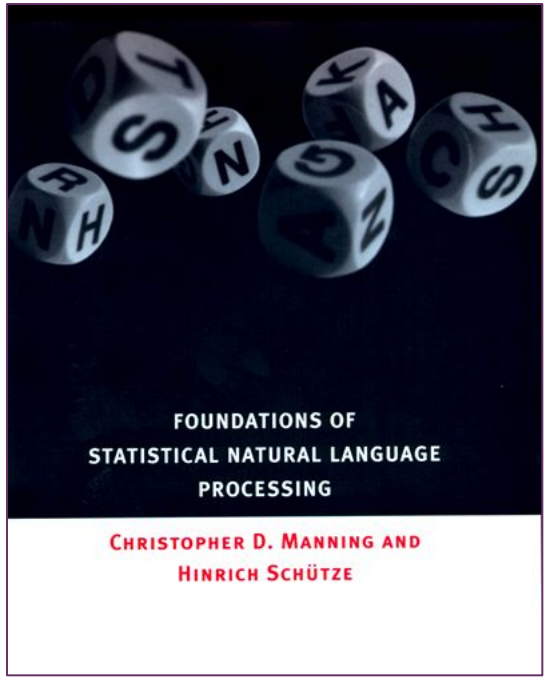
Maximal matching = 3
We need to verify with Language Model (LM)

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.



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

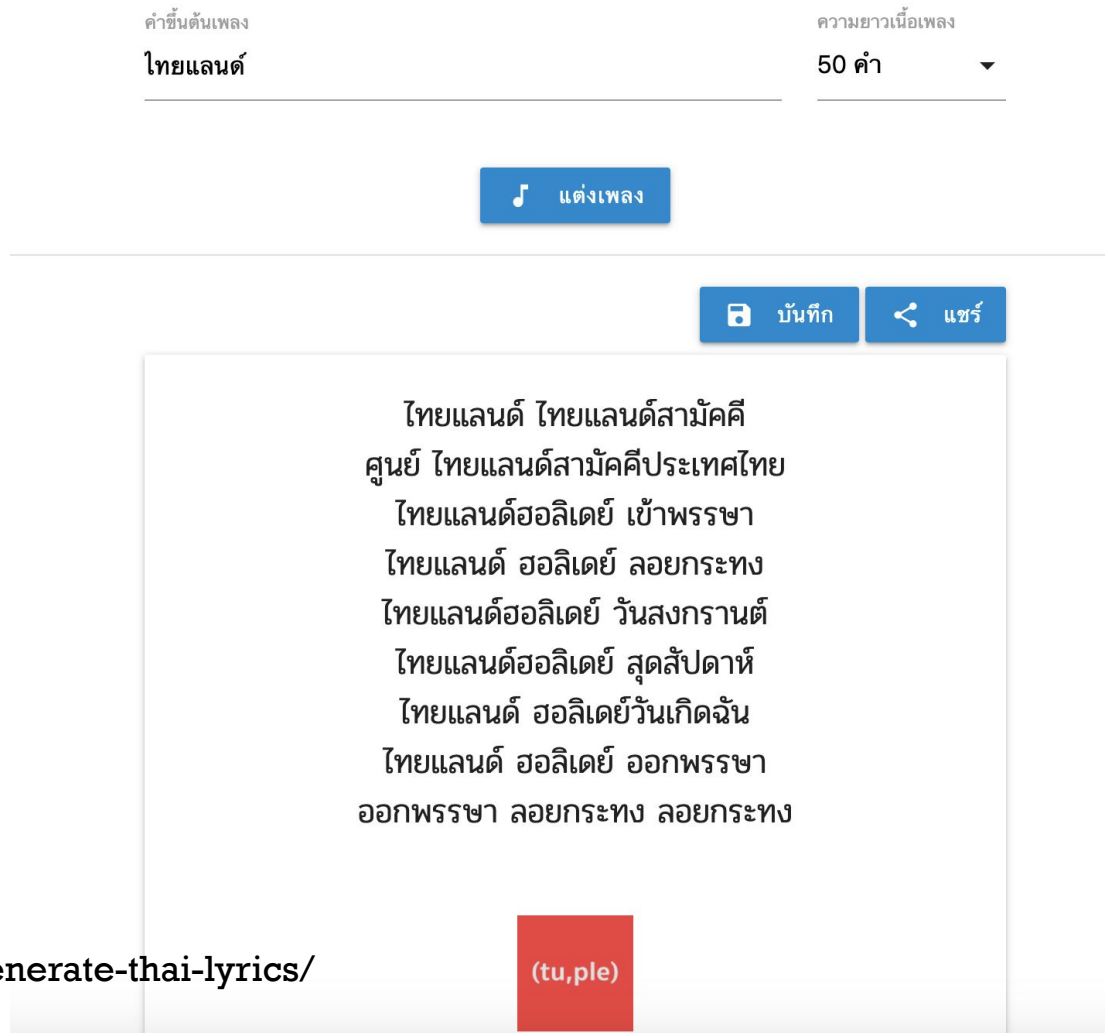
Generating Spatio-Temporal Descriptions in Pollen Forecasts

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)*

There are ten students in the class.

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

    - *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) 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)

  - 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>

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

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

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 (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

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

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

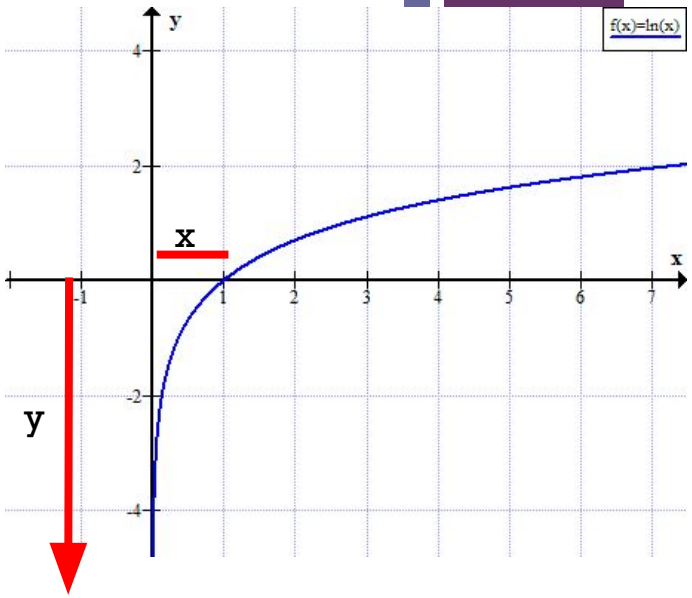
- We do everything in log space (ln(P(S))) to
  - Avoid underflow (numbers too small)
  - Also, adding is faster than multiplying

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

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

- 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?

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

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

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

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 Bigram	60 20
Air Travel Understanding	2,500	Bigram 4-gram	29 22
WSJ Dictation	5,000 20,000	Bigram Trigram Bigram Trigram	80 45 190 120
Switchboard Human-Human	23,000	Bigram Trigram	109 93
NYT Characters	63	Unigram Bigram	20 11
Shannon Letters	27	Human	~ 2

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

Shanon letters

- Guess the next letter until correct.
- How many guesses determine the branching factor
  - Interpret as how many bits to encode letter according to the LM

T	H	E	R	E		I	S		N	O		R	E	V	E	R	S	E
1	1	1	5	1	1	2	1	1	2	1	1	15	1	17	1	1	1	2
	O	N		A		M	O	T	O	R	C	Y	C	L	E		A	…
1	3	2	1	2	2	7	1	1	1	1	4	1	1	1	1	1	3	…

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

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

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)

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

Unknown words (UNK)

However, this still cannot solve the zero issue.

	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

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

  - 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

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

"eat **spend**"

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.

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

discounted bigram; Interpolation weight; unigram

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

	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.)

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

discounted bigram; Interpolation weight; unigram

- Kneser–Ney Smoothing
  - In case of bigram,

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

  - 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 precede w_{i-1}

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

- Kneser–Ney Smoothing
  - In general n-gram

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

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

- 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

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

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

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

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

Other uses of smoothing

Hidden Markov Model

- How to estimate A and B?

A_{ij}	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5

- Counts!

$$P(A_{11}) = \frac{\text{Count(from N to N)}}{\text{Count(N)}}$$

B_{ik}	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

- Counts with interpolation to avoid 0 counts
  - $P(A_{11}) = \rho \frac{\text{Count(from N to N)}}{\text{Count(N)}} + (1-\rho) \frac{\text{Count (N)}}{\text{Count (words)}}$

bigram unigram

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

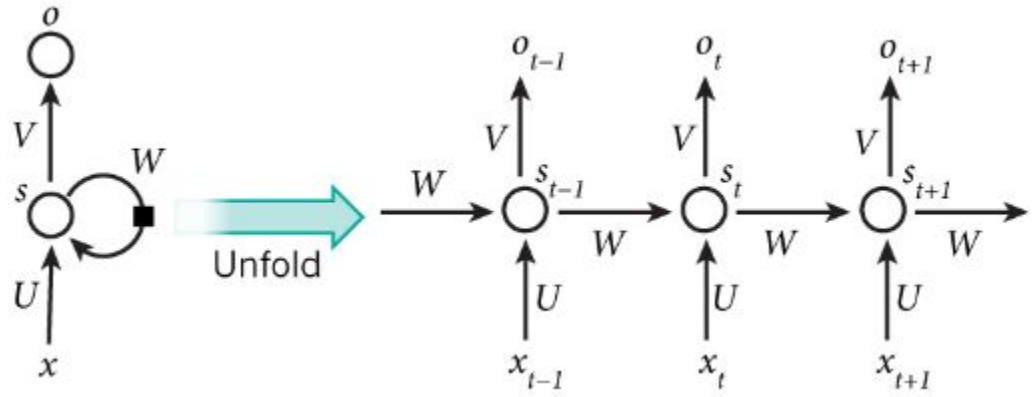
Neural Language Model

- 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 needs 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 models

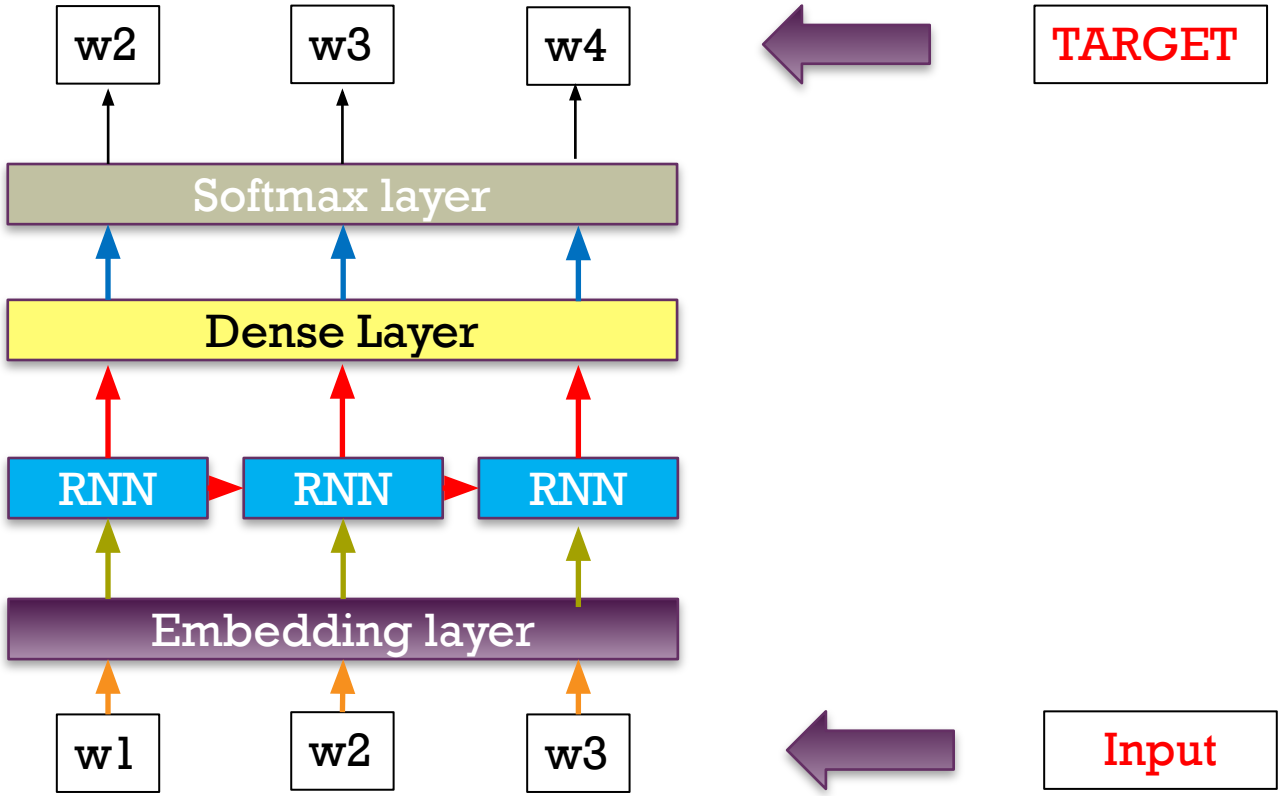


Neural Language Model (cont.)

- 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

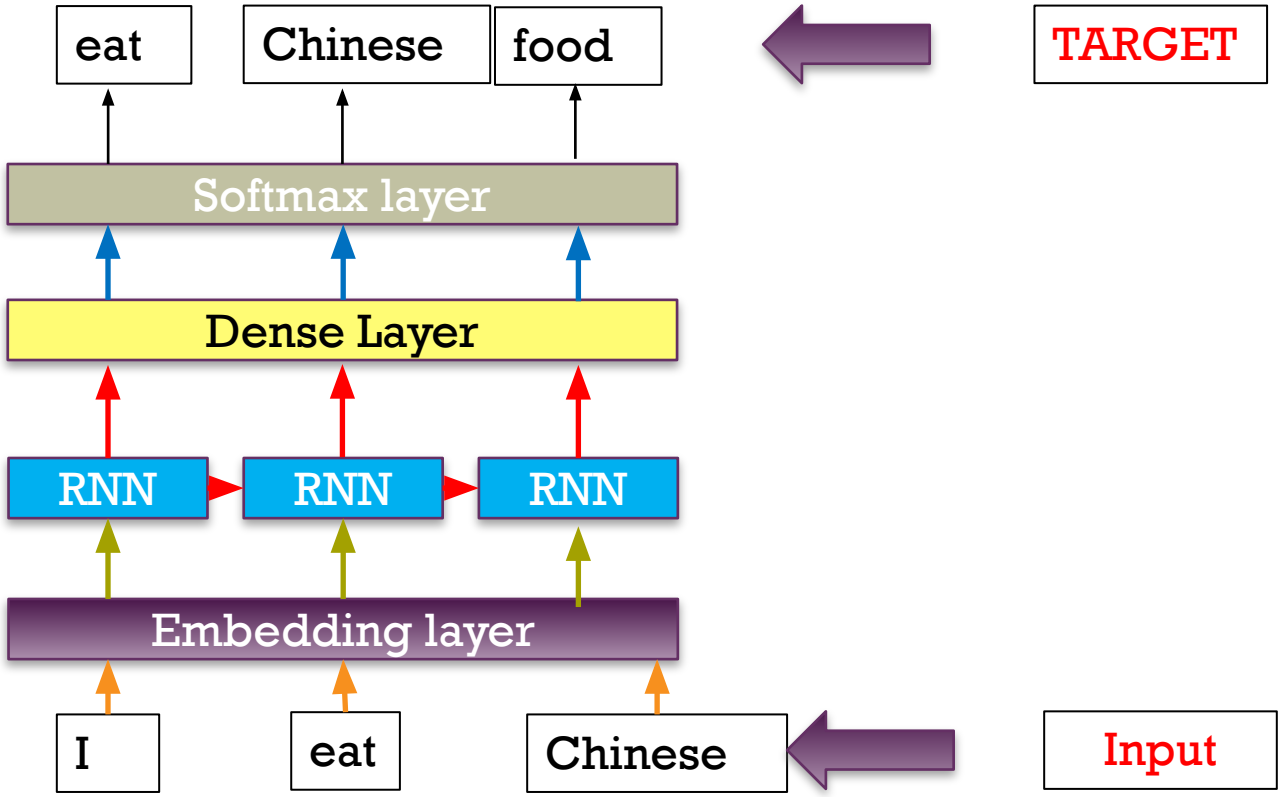


Neural Language Model (cont.)

- 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

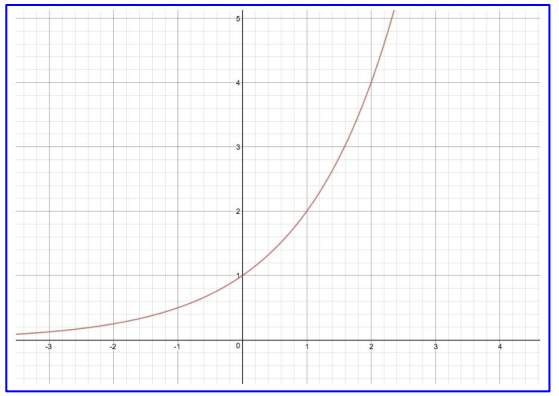


Neural Language Model (cont.)

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

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

Softmax (all classes V)



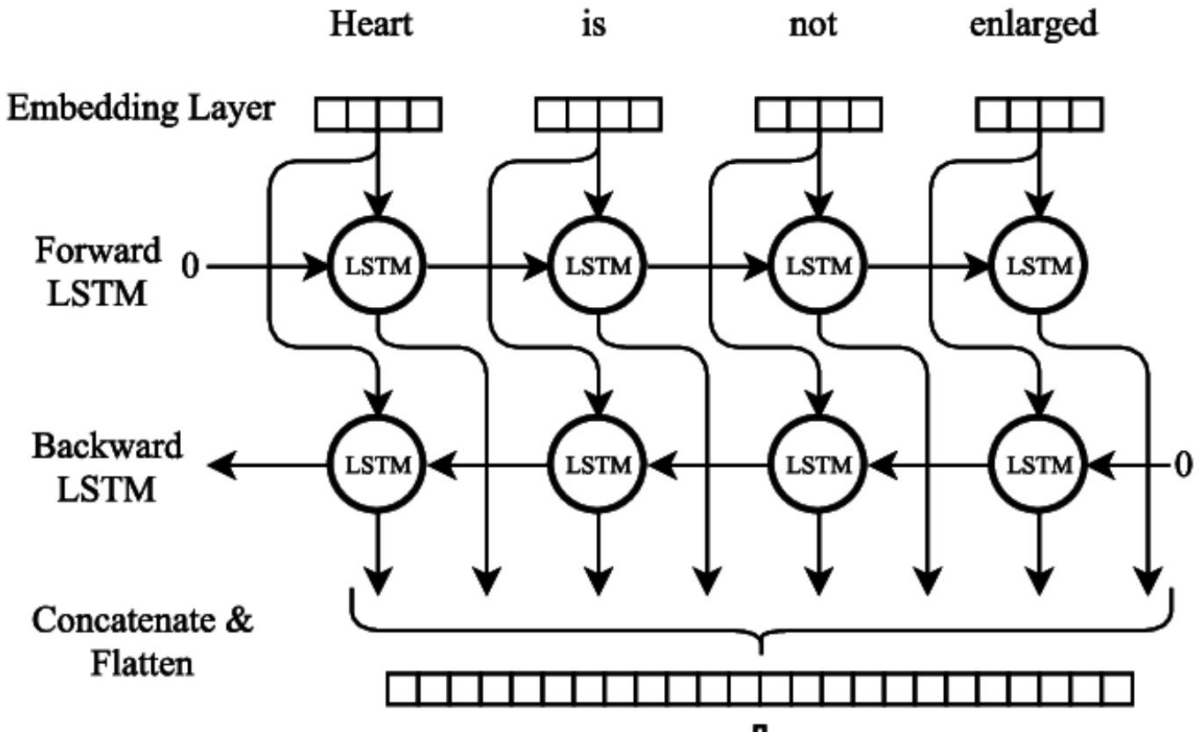
    - Where
      - V = Number of unique words in corpus
      - T = Number of total words in corpus
      - y = Target next word
      - $\hat{y}$ = Distribution of predicted next word
    - Actually, we are calculating perplexity
  - Perplexity = e^J

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

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)
- 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.

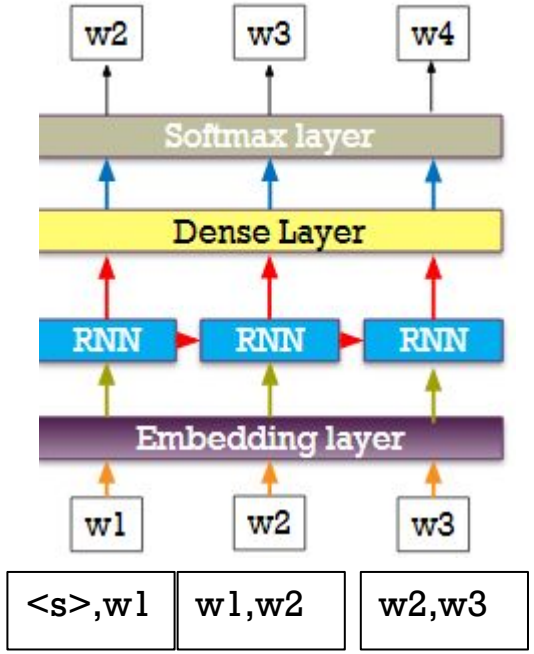


https://paperswithcode.com/method/bilstm#

Neural Language Model (cont.)

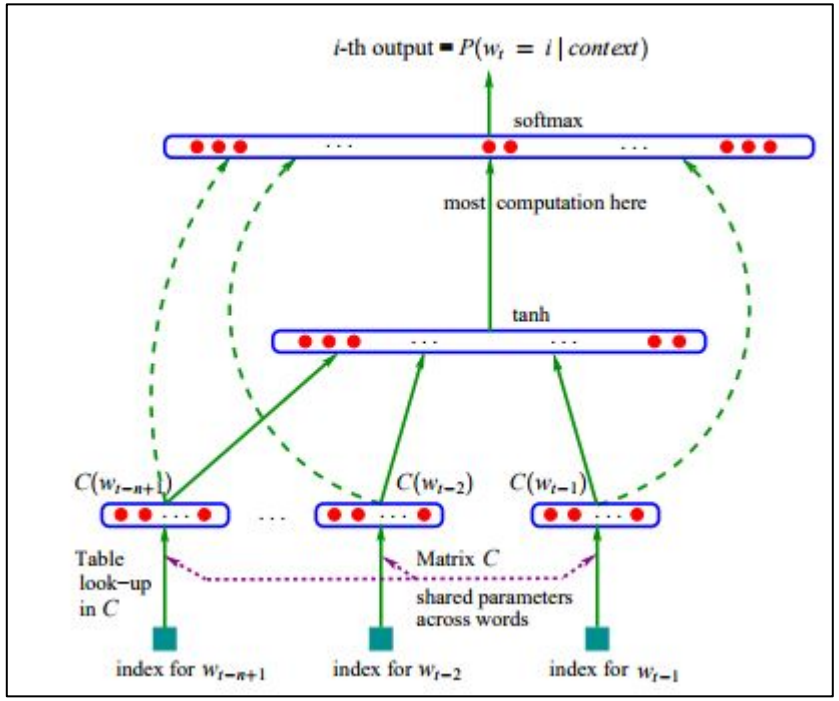
- Conclusion

- Neural Language Model vs. N-grams Model
  - A competitive n-grams model needs 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 subtly processes word semantics via word embeddings
  - 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**

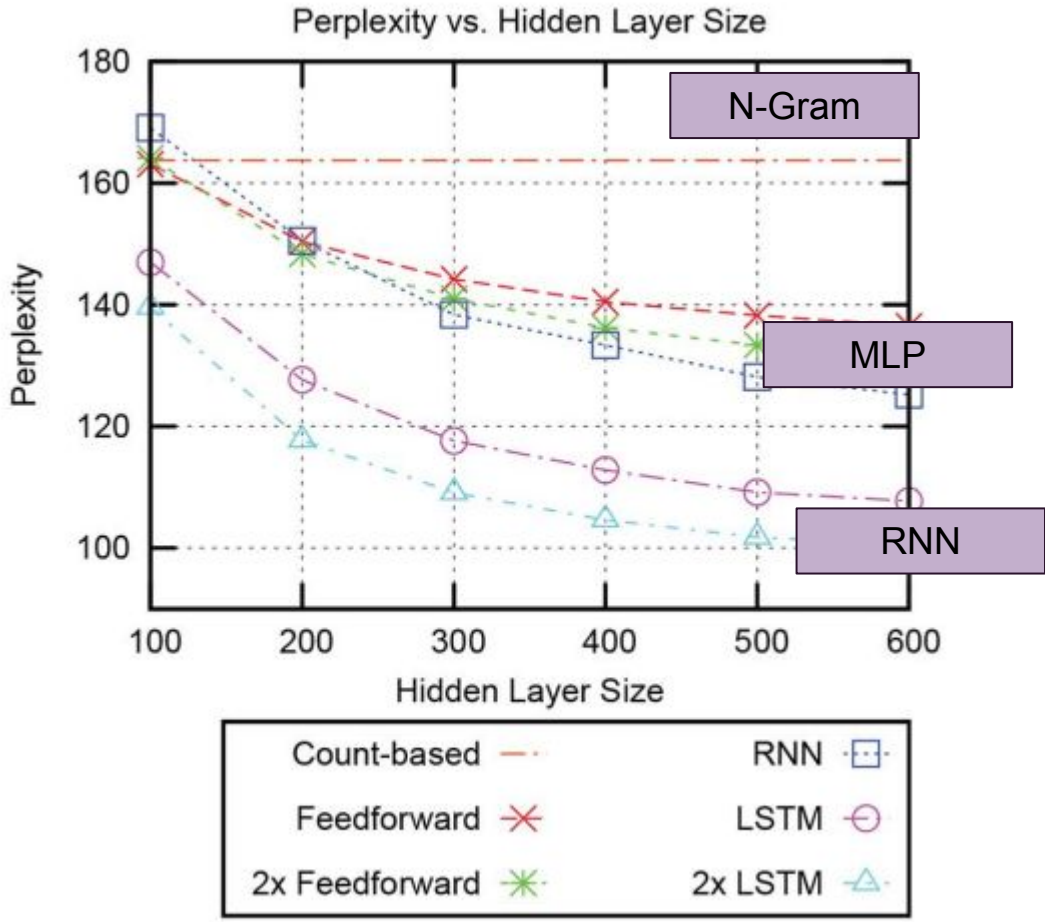


	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5	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

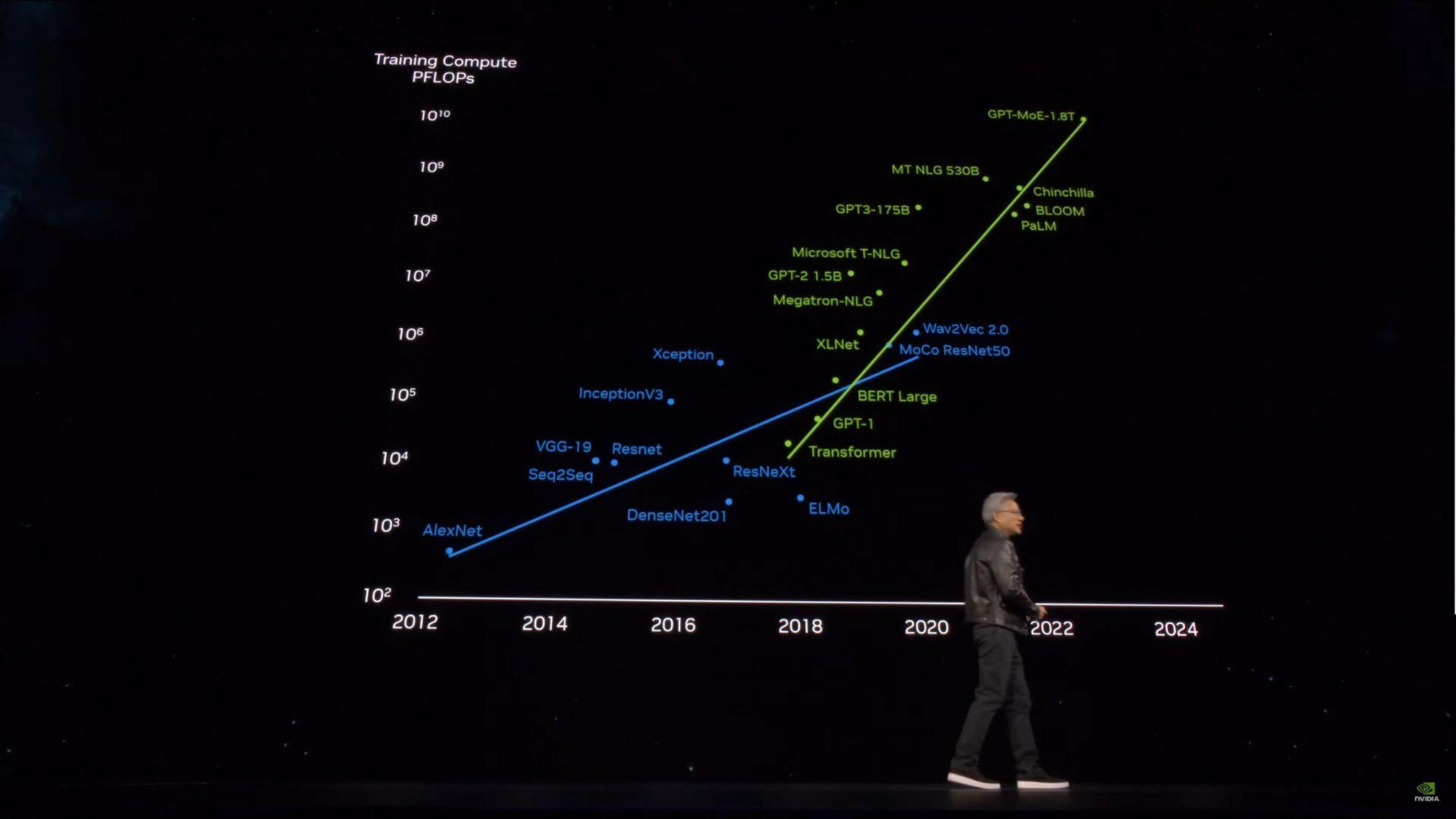
SWITCH TRANSFORMERS: SCALING TO TRILLION PARAMETER MODELS WITH SIMPLE AND EFFICIENT SPARSITY

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


Training Compute
PFLOPs
10^10
10^9
10^8
10^7
10^6
10^5
10^4
10^3
10^2
2012
2014
2016
2018
2020
2022
2024
GPT-MoE-1.8T
MT NLG 530B
Chinchilla
BLOOM
PaLM
GPT3-175B
Microsoft T-NLG
GPT-2 1.5B
Megatron-NLG
Wav2Vec 2.0
XLNet
MoCo ResNet50
Xception
InceptionV3
BERT Large
GPT-1
Transformer
VGG-19
Resnet
Seq2Seq
ResNeXt
ELMo
DenseNet201
AlexNet

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

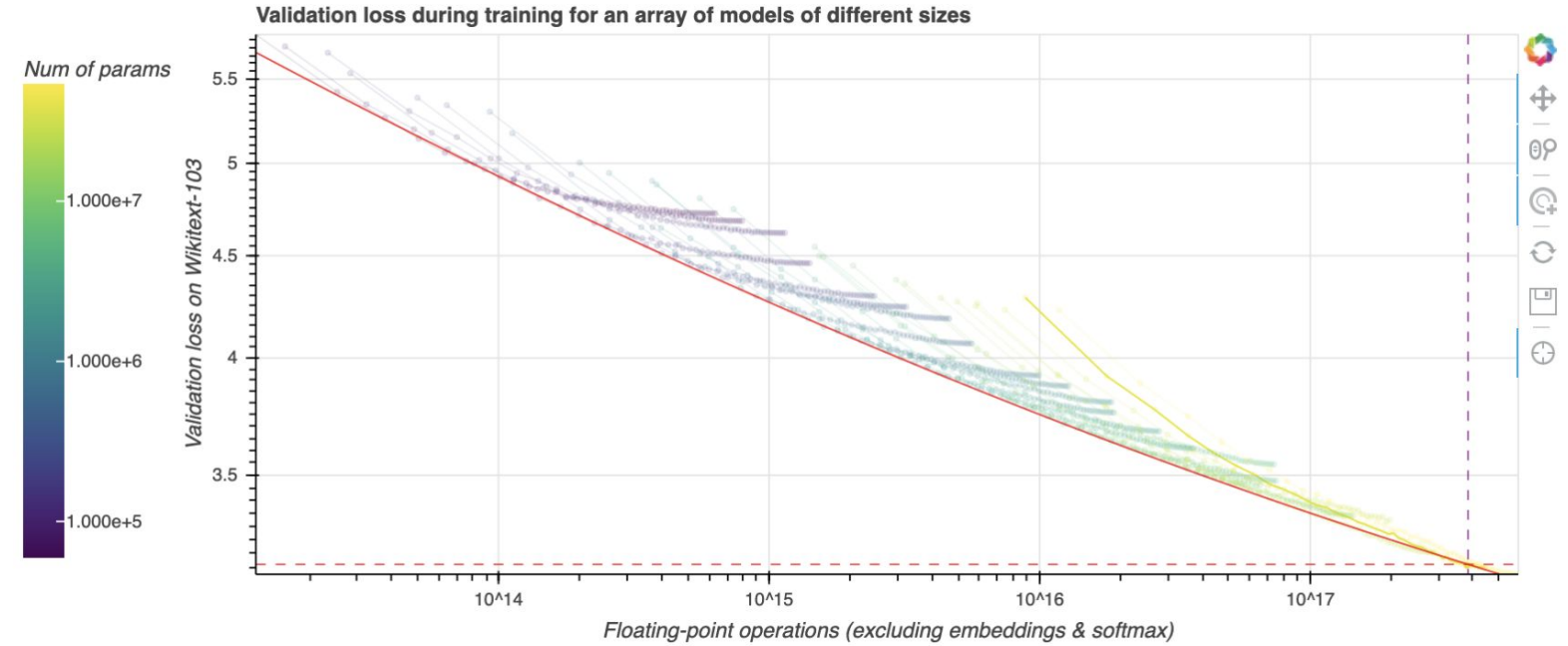
7 layers of 1048 dimensions

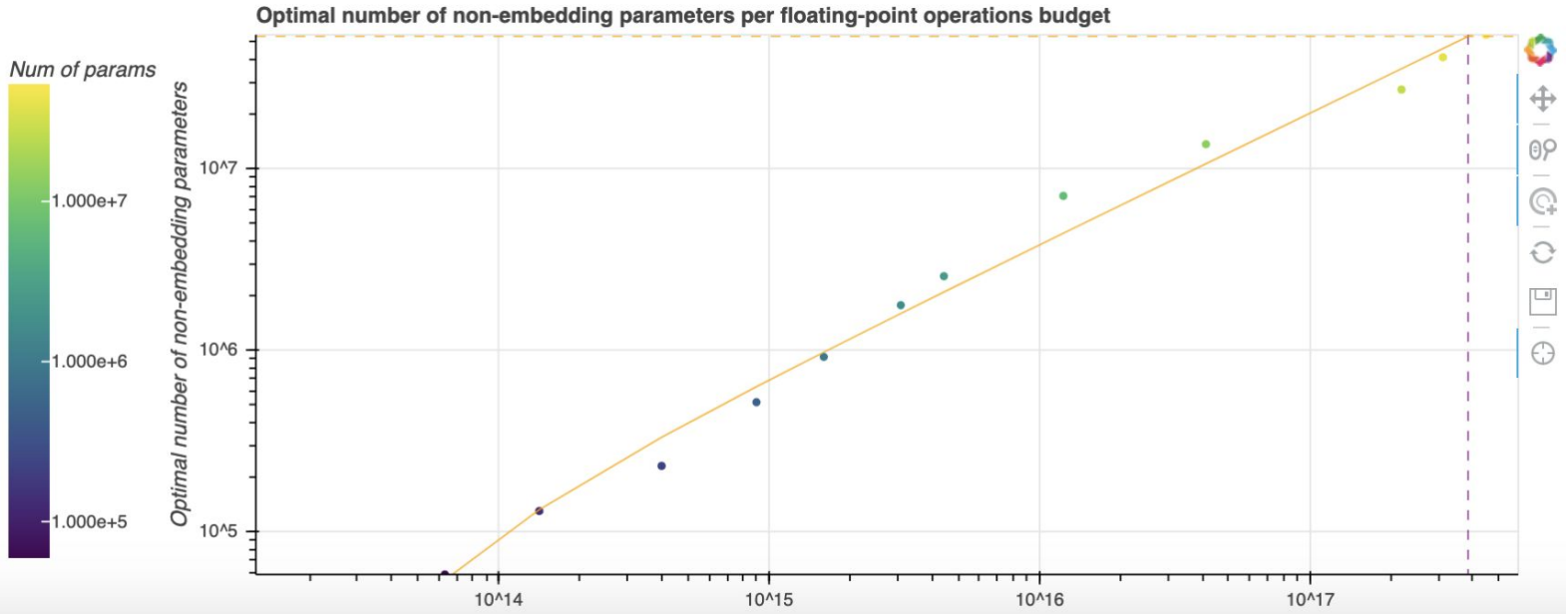
Initialize in 🤗 transformers!

Or a model of

13 layers of 768 dimensions

Initialize in 🤗 transformers!





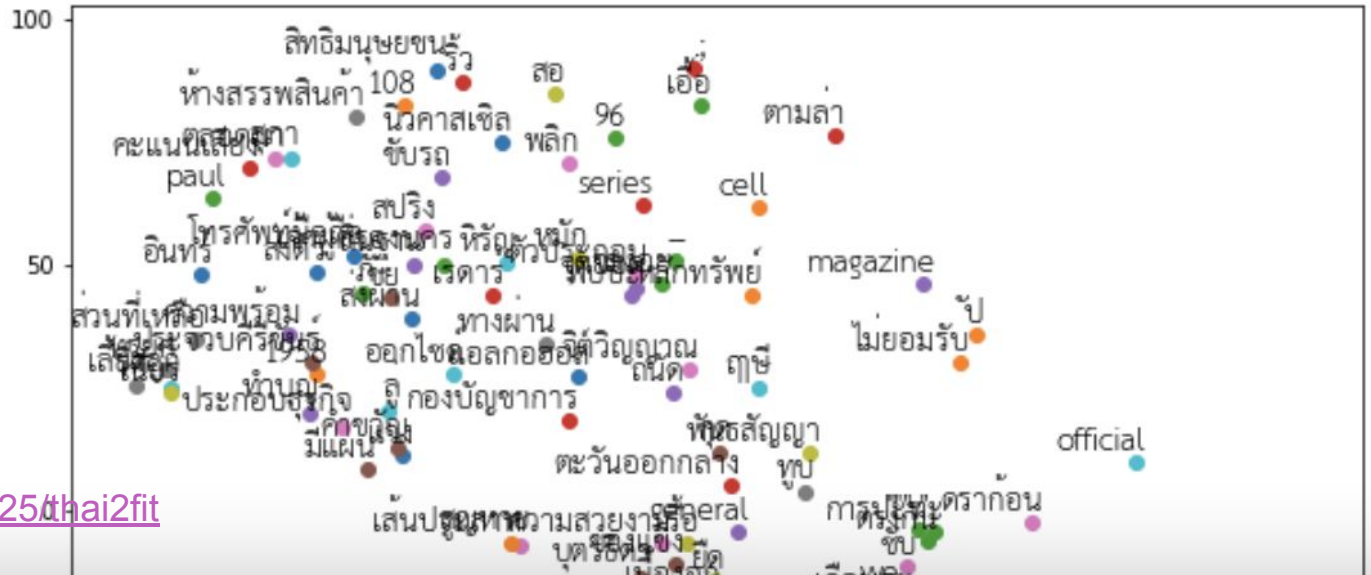
https://huggingface.co/calculator/

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

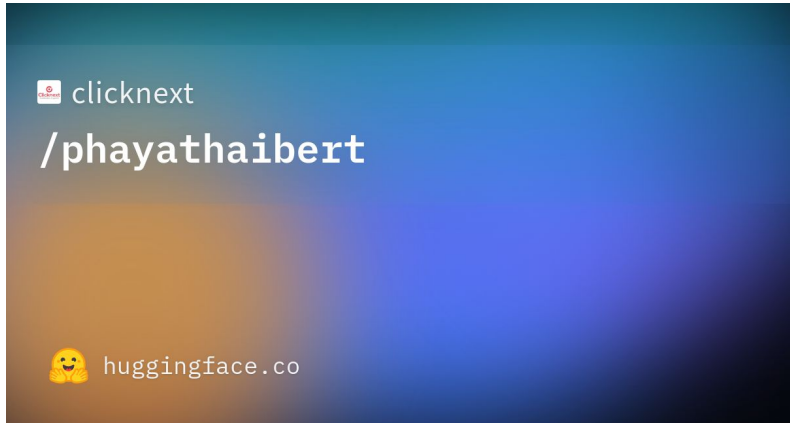
WangchanBERTa โมเดลประมวลผลภาษาไทยที่ใหญ่และก้าวหน้าที่สุดในขณะนี้

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 วันนี้โมเดลก็ยังถูกเทรนอย่างต่อเนื่องในศูนย์วิจัยที่วังจันทร์ จึงเป็นไปได้ว่าเราจะได้โมเดลที่มีประสิทธิภาพดียิ่งกว่ามาใช้ในอนาคต

PhayaThaiBERT



After vocabulary expansion, the model's vocabulary size increases from 25,005 to 249,262, significantly increasing the size of the model from 106M parameters to 278M parameters.

4.4 Train-Validation-Test Splits

After preprocessing and tokenization, our training data has a size of 91,130,926 examples (156.5GB). We then randomly split the data into 91,030,926 examples (156.3GB) for Train set, 50,000 examples (87.9MB) for Validation set, and 50,000 (88MB) examples for Test set.

Dataset	mBERT	XLM-R	WangchanBERTa	PhayaThaiBERT
1. wisesight_sentiment	70.57 / 55.62	71.77 / 58.29	74.35 / 65.23	**76.15 / 66.80**
2. wongnai_reviews	58.08 / 38.67	62.73 / 51.18	63.86 / **53.26**	**64.02** / 52.74
3. yelp_review_full	63.52 / 63.23	**65.19 / 64.80**	54.97 / 54.40	61.69 / 61.26
4. generated_reviews_enth	61.79 / 56.04	**65.06 / 60.28**	64.75 / 59.91	64.85 / 59.44
5. prachathai67k	63.90 / 52.95	66.63 / 58.01	67.51 / 59.05	**69.11 / 61.10**
6. thainer (ner)	79.58 / 69.87	84.95 / 72.20	84.64 / 68.19	**86.42 / 74.77**
7. lst20 (pos)	95.80 / 83.95	95.99 / 85.09	96.74 / **86.59**	**96.79** / 86.26
8. lst20 (ner)	76.48 / 70.27	**78.37** / 72.82	77.99 / **72.86**	78.11 / 72.69
9. thai_nner (layer 1)	61.19 / 23.45	63.88 / 23.28	59.31 / 22.85	**64.26 / 25.70**

Table 3: Fine-tuning results. The reported metrics are micro-average and macro-average F1 score respectively.

Dataset	WangchanBERTa		PhayaThaiBERT	
	<unk> count	Percentage	<unk> count	Percentage
1. wisesight_sentiment	2,900	0.32%	34	∼0%
2. wongnai_reviews	3,855	0.05%	90	∼0%
3. yelp_review_full	15	∼0%	0	0%
4. generated_reviews_enth	99	∼0%	10	∼0%
5. prachathai67k	241	0.02%	61	∼0%
6. thainer	8	∼0%	1	∼0%
7. lst20	19	∼0%	15	∼0%
8. thai_nner	267	0.01%	23	∼0%

Table 5: OOV rates of WangchanBERTa's tokenizer and our expanded tokenizer for each dataset.

Dataset	Proportion of Samples with Unassimilated Loanwords	Performance Gain Compared to WangchanBERTa
1. wisesight_sentiment	27.59%	+1.8 / +1.57
2. wongnai_reviews	37.20%	+0.16 / -0.52
3. yelp_review_full	Entirely English	+6.72 / +9.86
4. generated_reviews_enth	36.68%	+0.1 / -0.47
5. prachathai67k	8.54%	+1.6 / +2.05
6. thainer	10.90%	+1.78 / +6.58
7. lst20 (pos)	2.58%	+0.05 / -0.33
8. lst20 (ner)	2.58%	+0.12 / -0.17
9. thai_nner	38.54%	+4.95 / +2.85

Table 6: Proportion of samples with unassimilated English words compared with performance gain for each dataset.

Thai LLMs

Model	Base Model	Link to HuggingFace
Text		
Typhoon2-1B-Base	Llama-3.2-1B	scb10x/llama3.2-typhoon2-1b
Typhoon2-1B-Instruct		scb10x/llama3.2-typhoon2-1b-instruct
Typhoon2-3B-Base	llama-3.2-3B	scb10x/llama3.2-typhoon2-3b
Typhoon2-3B-Instruct		scb10x/llama3.2-typhoon2-3b-instruct
Typhoon2-7B-Base	Qwen2.5-7B	scb10x/typhoon2-qwen2.5-7b
Typhoon2-7B-Instruct		scb10x/typhoon2-qwen2.5-7b-instruct
Typhoon2-8B-Base	Llama-3.1-8B	scb10x/llama3.1-typhoon2-8b
Typhoon2-8B-Instruct		scb10x/llama3.1-typhoon2-8b-instruct
Typhoon2-70B-Base	Llama-3.1-70B	scb10x/llama3.1-typhoon2-70b
Typhoon2-70B-Instruct		scb10x/llama3.1-typhoon2-70b-instruct
Safety Classifier		
Typhoon2-Safety	mdeberta-v3-base	scb10x/typhoon2-safety-preview
Multimodal		
Typhoon2-Vision	Qwen2-VL-7B-Instruct	scb10x/typhoon2-qwen2vl-7b-vision-instruct
Typhoon2-Audio	Typhoon2-8B-Instruct	scb10x/llama3.1-typhoon2-audio-8b-instruct

~17B tokens

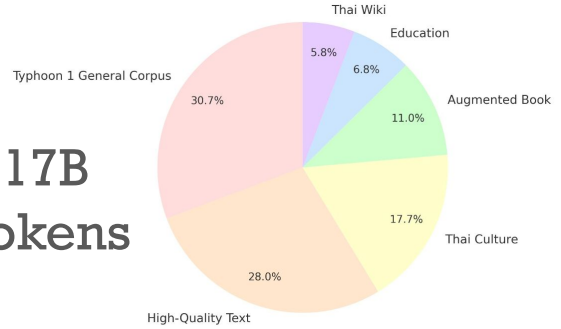


Figure 1: Thai Pretraining Data Mixture

Collection by scb10x

Typhoon 2 Text

Latest Official Text ThaiLLM release by SCB 10X.

huggingface.co

Model	IFEval TH	IFEval EN	MTBench TH	MTBench EN	CS 0.7	CS 1.0	FC TH	FC EN	GSM8K TH	GSM8K EN	Math TH	Math EN	HumanEv TH	HumanEv EN	MBPP TH	MBPP EN
Typhoon2-Q-7B-Instruct	**74.3**	73.3	**6.18**	8.09	**99.2**	**96.8**	**74.2**	**75.4**	**79.0**	**84.2**	**55.4**	66.4	73.2	79.3	78.3	**81.7**
Qwen2.5-7B-Instruct	68.4	**76.8**	6.00	**8.53**	85.8	20.4	66.0	74.8	47.5	81.0	17.4	**73.4**	**77.4**	**81.1**	**80.4**	79.6
OpenThaiGPT 1.5 7B	67.3	75.4	5.69	8.10	93.8	28.0	65.5	73.1	65.7	68.0	24.4	69.6	71.3	78.7	77.5	79.1
Typhoon2-L-8B-Instruct	**72.6**	76.4	**5.74**	7.58	**98.8**	**98.0**	**75.1**	**79.0**	**71.7**	**81.0**	**38.4**	**49.0**	**58.5**	**68.9**	60.8	63.0
Llama3.1-8B-instruct	58.0	**77.6**	5.10	**8.11**	93.0	11.2	36.9	66.0	45.1	62.4	24.4	48.0	51.8	67.7	**64.6**	**66.9**

Table 15: Performance of 7-8B Models: **Q** denotes Qwen2.5, and **L** denotes Llama 3.1.

Model	IFEval TH	IFEval EN	MTBench TH	MTBench EN	CS 0.7	CS 1.0	FC TH	FC EN	GSM8K TH	GSM8K EN	Math TH	Math EN	HumanEv TH	HumanEv EN	MBPP TH	MBPP EN
Typhoon2-70B-Instruct	**81.4**	88.7	**7.36**	8.85	**98.8**	**94.8**	**70.8**	**65.7**	**88.7**	**93.4**	**59.6**	64.9	79.9	83.5	**86.0**	84.9
Llama-3.1-70B-Instruct	64.9	86.3	6.29	**9.10**	90.2	53.0	47.9	53.2	61.1	60.0	40.6	63.6	73.8	79.9	83.6	82.8
Llama-3.3-70B-Instruct	81.0	**91.5**	6.79	8.83	72.6	39.2	50.3	56.3	61.6	87.7	44.3	**73.5**	**81.7**	**84.1**	84.9	**87.3**
Qwen2.5-72B-Instruct	78.6	**86.5**	**7.46**	**9.28**	91.6	48.6	**70.8**	**77.9**	71.7	**94.6**	**47.9**	**83.1**	**84.1**	**87.2**	88.6	**90.5**
OpenThaiGPT 1.5 72B	**80.3**	84.5	7.31	9.08	**95.6**	**50.4**	67.1	74.6	**79.1**	89.9	43.6	81.8	81.7	84.8	**88.9**	89.7

Table 16: 70B Model Performance

OpenThaiGPT

OpenThaiGPT 7b Version 1.5 is an advanced 7-billion-parameter Thai language chat model based on Qwen v2.5 released on September 30, 2024. It has been specifically fine-tuned on over 2,000,000 Thai instruction pairs and is capable of answering Thai-specific domain questions.

14B and 72B models also available

https://arxiv.org/pdf/2412.13702
https://huggingface.co/openthaigpt/openthaigpt1.5-7b-instruct

https://pythainlp.github.io/pythainlp-corpus/ltw2v.html

Large Thai Word2Vec

ltw2v

latest version: 0.1

Description : LTW2V: The Large Thai Word2Vec

Long Description : LTW2V is The large Thai Word2Vec. It built with oxidized-thainlp from OSCAR Corpus (Open Super-large Crawled Aggregated coRpus).

HomePage : https://github.com/PyThaiNLP/large-thaiword2vec

Authors : Wannaphong Phatthiyaphaibun

OSCAR

OSCAR or **O**pen **S**uper-large **C**rawled **A**ggregated co**R**pus is a huge multilingual corpus obtained by language classification and filtering of the Common Crawl corpus using the Ungoliant architecture.

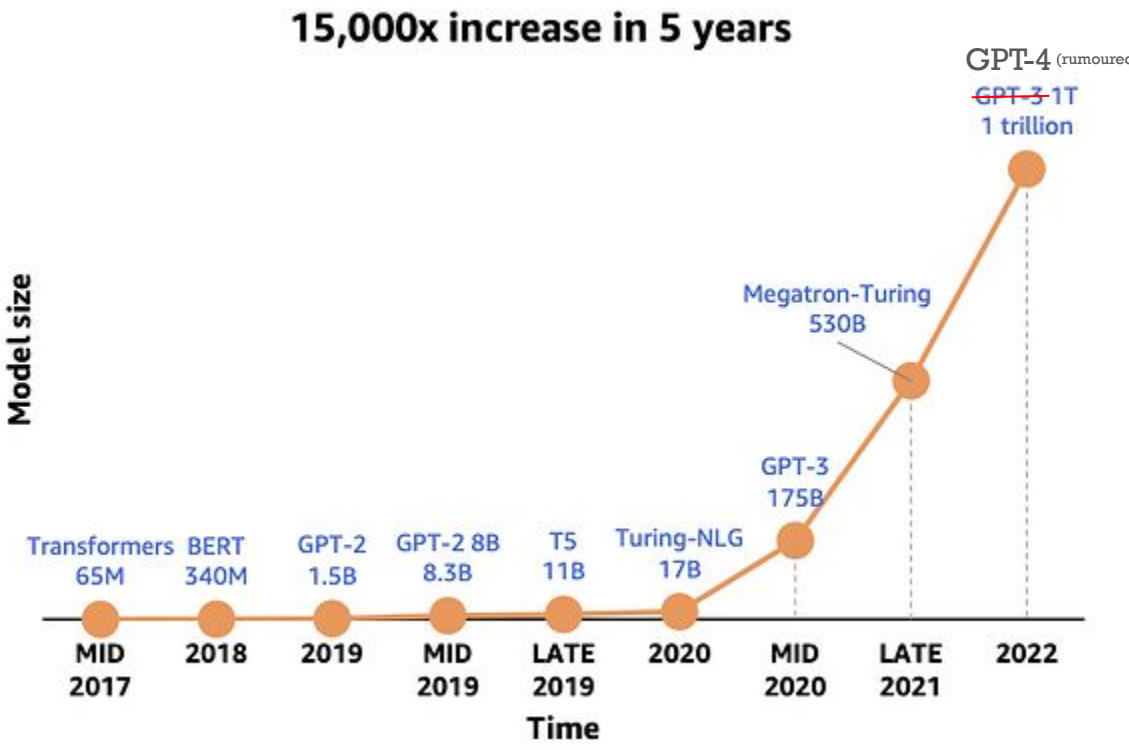
Number

- line: 3,749,826
- Words: 1,739,705,916
- size: 400

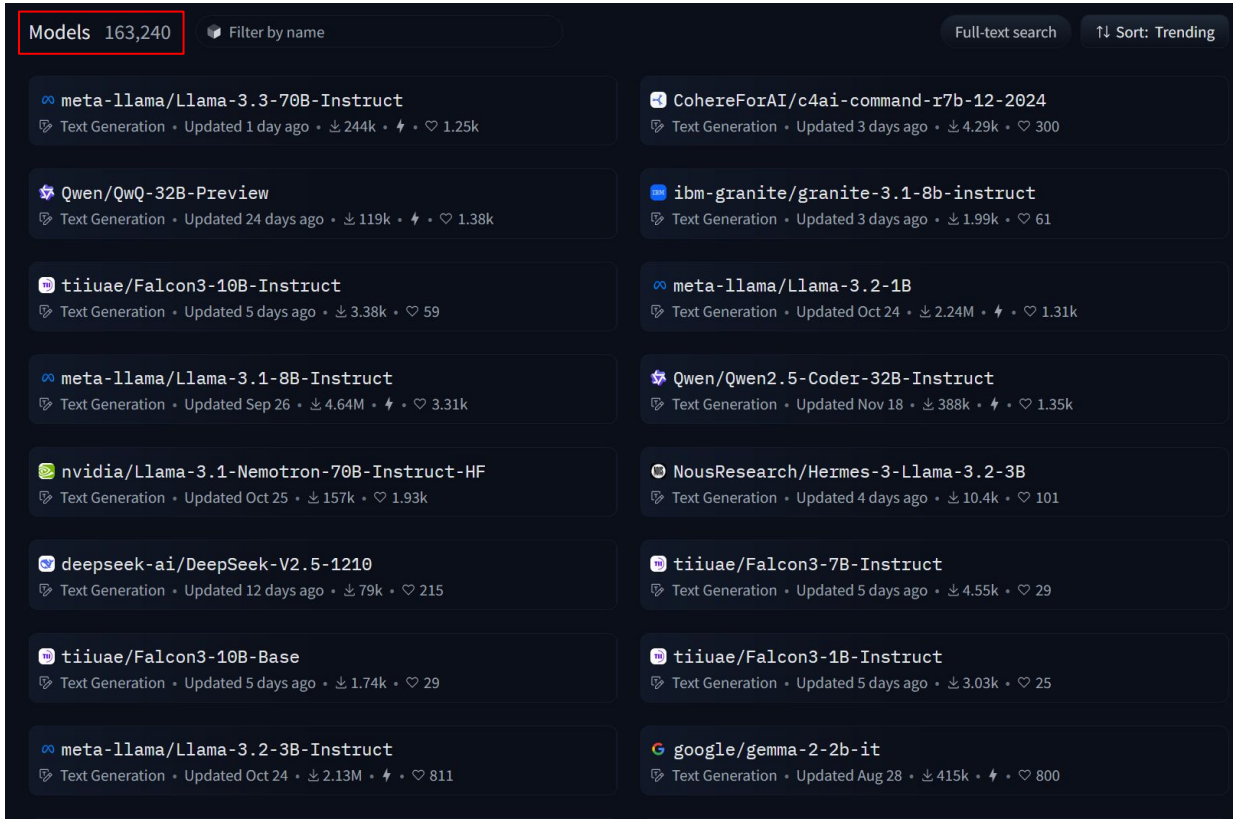
embedding dimensions

https://colab.research.google.com/drive/1XSoEIRWz6l9w7Ebg9vfYEueFLdpZDvdo?usp=sharing

Large Language Models (LLMs)

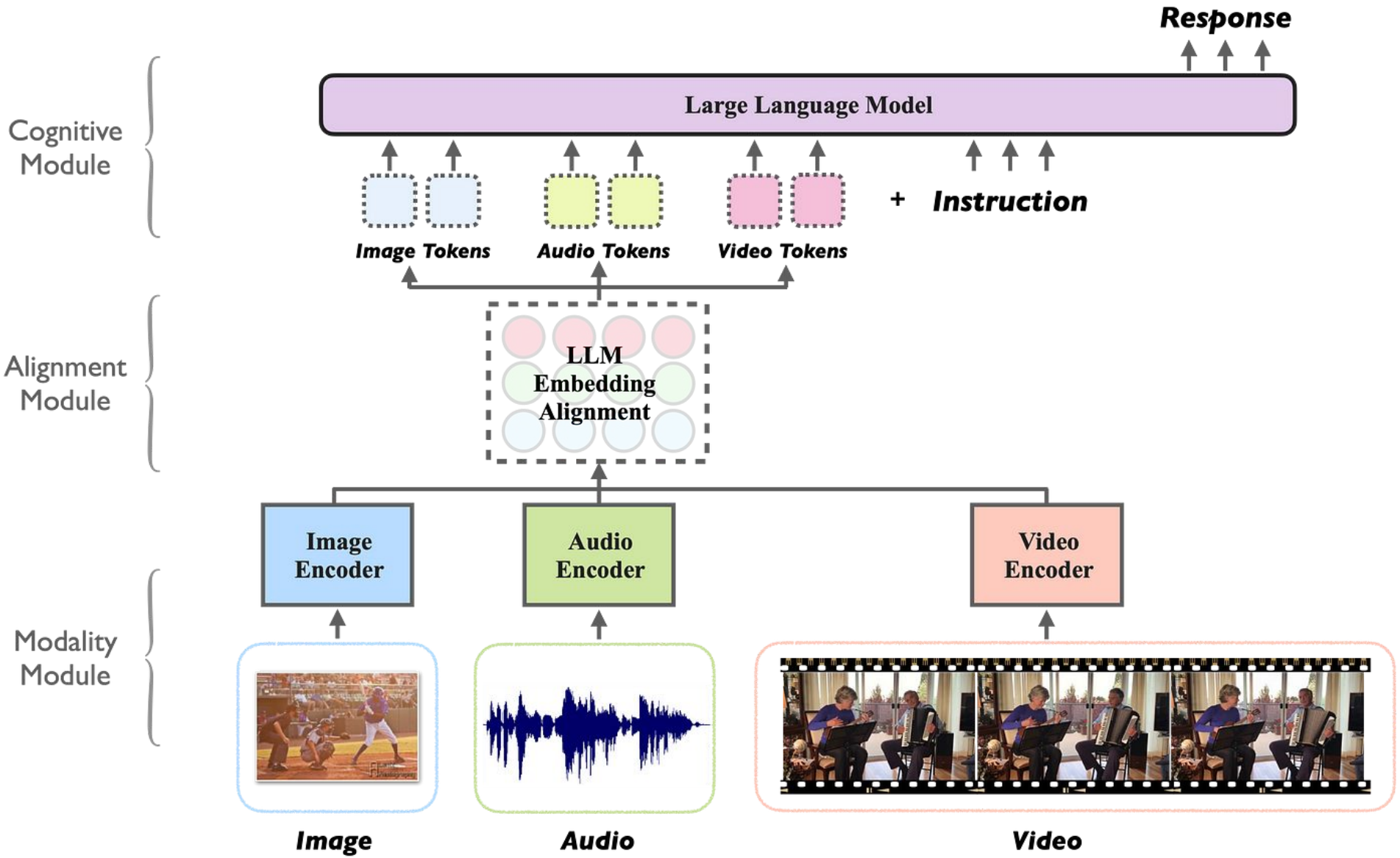


163,240 LLMs and counting!



https://medium.com/@vipra_singh/building-llm-applications-large-language-models-part-6-ea8bd982bdee
https://huggingface.co/models?pipeline_tag=text-generation&sort=trending

Multimodal LLM

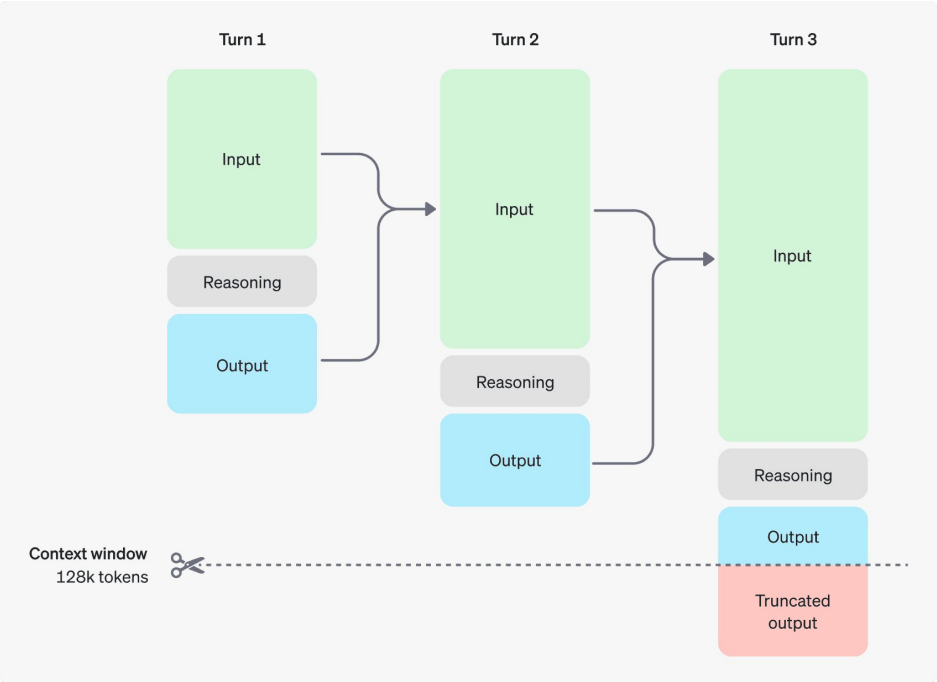


https://medium.com/@cout.shubham/exploring-multimodal-large-language-models-a-step-forward-in-ai-626918c6a3ec

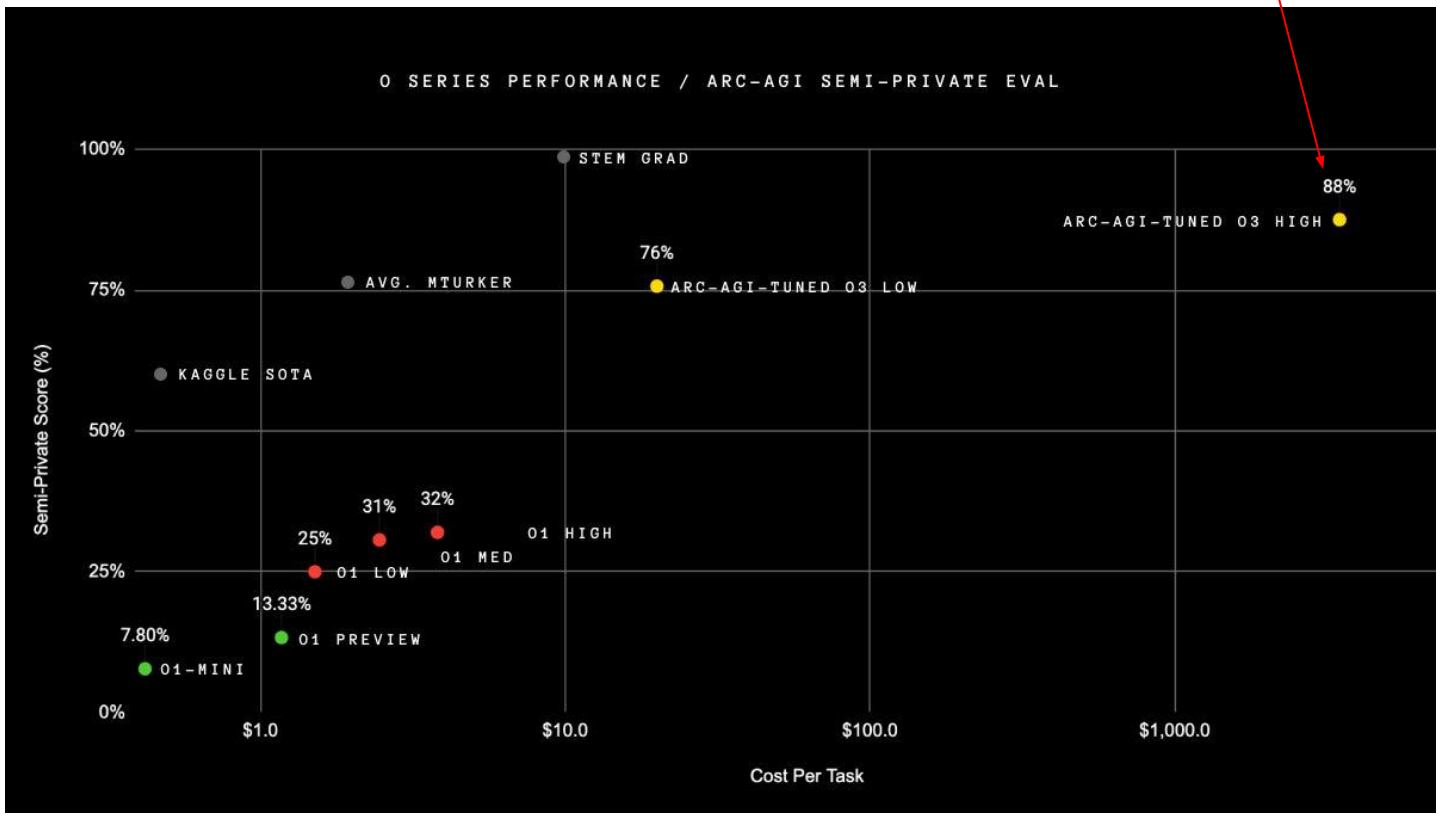
Reasoning Models

GPT-o1 and o3 (Dec 2024)

QwQ (qwen) - open-source!



o3 → ~$3000/task



https://lunary.ai/blog/open-ai-o1-reasoning-models
https://www.thealgorithmicbridge.com/p/openai-o3-model-is-a-message-from

Conclusion

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

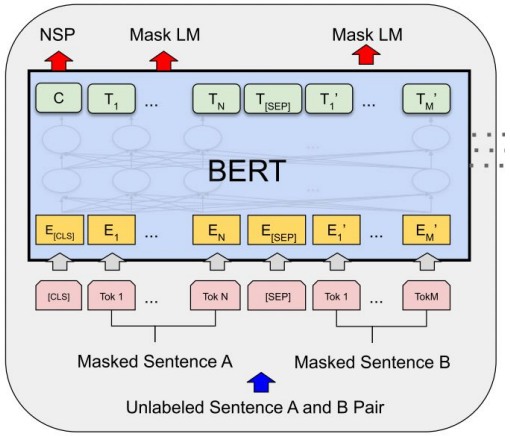
Appendix

BERT (2018)

A model comprises of Transformer encoders only. Its pretraining objective enables the usage of information from both left and right context, i.e. **bidirectional**.

It achieves SOTA results on basically the whole NLP benchmark.

Most importantly, the pretrained weight is available (and you can run it on consumer gpus, e.g. Colab)!



System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

https://arxiv.org/abs/1810.04805

https://huggingface.co/docs/transformers/model_doc/bert

https://openai.com/blog/better-language-models/

GPT2 (2019)

A model comprises of Transformer decoders only. It pretrains on predicting the next word. The model is so fluent in text generation that OpenAI (its developer) claimed that it is "too dangerous" for the pretrained weight (of the largest version) to be released.

February 14, 2019

Release Strategy

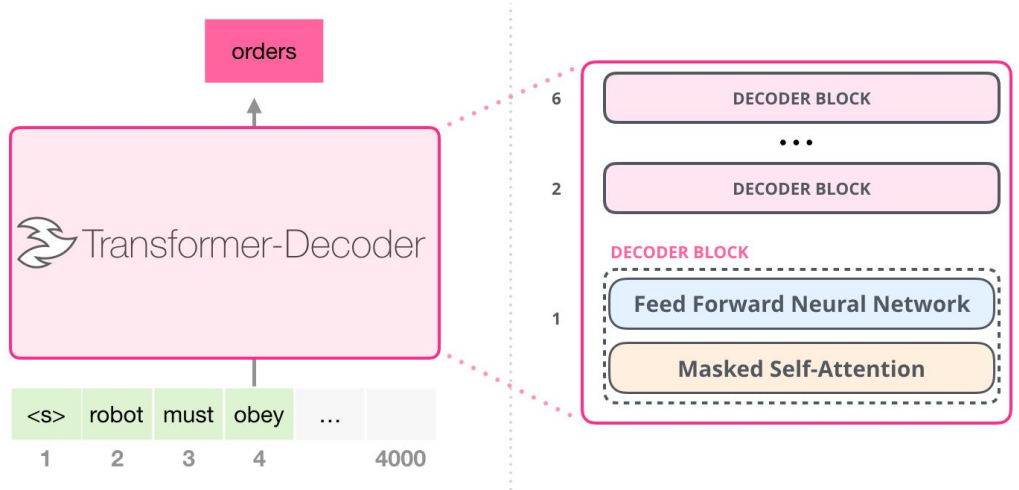
Due to concerns about large language models being used to generate deceptive, biased, or abusive language at scale, we are only releasing a much smaller version of GPT-2 along with sampling code. We are not releasing the dataset, training code,

Though it is now publicly available.

November 5, 2019

GPT-2: 1.5B Release

As the final model release of GPT-2's staged release, we're releasing the largest version (1.5B parameters) of GPT-2 along with code and model weights to facilitate detection of outputs of GPT-2 models. While there have been larger language models released since August, we've continued with our original staged release plan in order to provide the community with a test case of a full staged release process. We hope that this test case will be useful to developers of future powerful models, and we're actively continuing the conversation with the AI community on responsible publication.



https://arxiv.org/pdf/2211.09085.pdf

16 Nov 2022

Galactica: A Large Language Model for Science

Ross Taylor **Marcin Kardas** **Guillem Cucurull**

Thomas Scialom **Anthony Hartshorn** **Elvis Saravia**

Andrew Poulton **Viktor Kerkez** **Robert Stojnic**

Meta AI

Galactica uses a Transformer architecture in a decoder-only setup (like the GPT family)

Question: A needle 35 mm long rests on a water surface at 20°C. What force over and above the needle's weight is required to lift the needle from contact with the water surface? $\sigma = 0.0728$m.

<work>

$$\sigma = 0.0728 \text{ N/m}$$
$$\sigma = F/L$$
$$0.0728 = F/(2 \times 0.035)$$
$$F = 0.0728(2 \times 0.035)$$

calculate.py

```
f = 0.0728*(2*0.035)

with open("output.txt", "w") as file:
    file.write(str(round(f, 5)))
```

«run: "calculate.py">

«read: "output.txt"»

0.0051

</work>

Answer: $F = 0.0051$ N

Write code -> write python command -> read output

Figure 3: Model-Machine Symbiosis. We show an example answer with the <work> working memory token. It performs exact steps for rearranging the equation, and when it reaches a calculation that it cannot solve reliably in a forward-pass, it writes a program, which can then be offloaded to a classical computer.

Model	n_{params}	n_{layers}
GAL 125M	125M	12
GAL 1.3B	1.3B	24
GAL 6.7B	6.7B	32
GAL 30B	30.0B	48
GAL 120B	120.0B	96

Table 5: Details of the models trained

Total dataset size = 106 billion tokens			
Data source	Documents	Tokens	Token %
Papers	48 million	88 billion	83.0%
Code	2 million	7 billion	6.9%
Reference Material	8 million	7 billion	6.5%
Knowledge Bases	2 million	2 billion	2.0%
Filtered CommonCrawl	0.9 million	1 billion	1.0%
Prompts	1.3 million	0.4 billion	0.3%
Other	0.02 million	0.2 billion	0.2%

Table 2: The Galactica Corpus. A full breakdown of these sources is contained in the Appendix.

Here is a sequence for a protein:

[START_AMINO]MEEPQSDPSVEPPLSQETFSDLWKLLPE...[END_AMINO]

And here is an isomeric SMILES for a compound:

[START_I_SMILES]CC(O)(P(=O)(O)O)P(=O)(O)O[END_I_SMILES]

Question: Will the the chemical compound be active against this protein?

Answer: No

Drug Discovery

Figure 15: Tox21 Prompt. We include the protein sequence and the SMILES formula and pose the classification problem in natural language.

Prompt

Citation Prediction

in the BQ literature as, when p is a mixture of Gaussians, the mean element μ_p is analytically tractable (see Appendix C). Some other (p, k) pairs that produce analytic mean elements are discussed in [[START_REF] On the Equivalence between Kernel Quadrature Rules and Random Feature Expansions, Bach [START_REF]].
For this simulation study, we took $p(x)$ to be a 20-component mixture of 2D-Gaussian distributions. Monte Carlo (MC) is often used for such distributions but has a slow convergence rate in $\mathcal{O}_P(n^{-1/2})$. FW and FWLS are known to converge more quickly and are in this sense preferable to MC [[START_REF]

Prediction

On the Equivalence between Herding and Conditional Gradient Algorithms, Bach

Figure 12: Citation Prompt. An example prompt predicting a citation in-context; from Briol et al. (2015).

and a lot more…

Demo : https://galactica.org/explore/

BioBERT: a pre-trained biomedical language representation model for biomedical text mining

Bioinformatics, 2019, 1–7
doi: 10.1093/bioinformatics/btz682
Advance Access Publication Date: 10 September 2019
Original Paper

Jinhyuk Lee [1,†], Wonjin Yoon [1,†], Sungdong Kim [2], Donghyeon Kim [1], Sunkyu Kim [1], Chan Ho So [3] and Jaewoo Kang [1,3,*]

[1]Department of Computer Science and Engineering, Korea University, Seoul 02841, Korea, [2]Clova AI Research, Naver Corp, Seong-Nam 13561, Korea and [3]Interdisciplinary Graduate Program in Bioinformatics, Korea University, Seoul 02841, Korea

Table 1. List of text corpora used for BioBERT

Corpus	Number of words	Domain
English Wikipedia	2.5B	General
BooksCorpus	0.8B	General
PubMed Abstracts	4.5B	Biomedical
PMC Full-text articles	13.5B	Biomedical

BioBERT provides better performance on biomedical data compared to BERT.

Table 9. Prediction samples from BERT and BioBERT on NER and QA datasets

Task	Dataset	Model	Sample
NER	NCBI disease	BERT	WT1 missense mutations, associated with male pseudohermaphroditism in **Denys–Drash syndrome**, fail to . . .
		BioBERT	WT1 missense mutations, associated with **male pseudohermaphroditism** in **Denys–Drash syndrome**, fail to . . .
	BC5CDR (Drug/Chem.)	BERT	. . . a case of oral **penicillin anaphylaxis** is described, and the terminology . . .
		BioBERT	. . . a case of oral **penicillin** anaphylaxis is described, and the terminology . . .
	BC2GM	BERT	Like the DMA, but unlike all other mammalian class II A genes, the zebrafish gene codes for two cysteine residues . . .
		BioBERT	Like the **DMA**, but unlike all other mammalian class II A genes, the zebrafish gene codes for two cysteine residues . . .
QA	BioASQ 6b-factoid		Q: Which type of urinary incontinence is diagnosed with the Q tip test?
		BERT	A total of 25 women affected by clinical **stress** urinary incontinence (SUI) were enrolled. After undergoing (. . .) Q-tip test, . . .
		BioBERT	A total of 25 women affected by clinical **stress urinary incontinence** (SUI) were enrolled. After undergoing (. . .) Q-tip test, . . .
			Q: Which bacteria causes erythrasma?
		BERT	**Corynebacterium** minutissimum is the bacteria that leads to cutaneous eruptions of erythrasma . . .
		BioBERT	**Corynebacterium minutissimum** is the bacteria that leads to cutaneous eruptions of erythrasma . . .

Note: Predicted named entities for NER and predicted answers for QA are in bold.

Answers from BioBERT are more comprehensive

https://arxiv.org/pdf/1908.10063.pdf

FinBERT: Financial Sentiment Analysis with Pre-trained Language Models

Dogu Tan Araci
dogu.araci@student.uva.nl
University of Amsterdam
Amsterdam, The Netherlands

FinBERT is BERT that is further pretrained on a financial corpus.

	# classes	# documents	# sentences	# words	# unique words	avg. # words
TRC2-financial	-	46 143	400K	29M	-	-

Table 4: Performance with different pre-training strategies

Model	Loss	Accuracy	F1 Score
Vanilla BERT	0.38	0.85	0.84
FinBERT-task	0.39	0.86	**0.85**
FinBERT-domain	**0.37**	**0.86**	0.84

Bold face indicates best result in the corresponding metric. Results are reported on 10-fold cross validation.

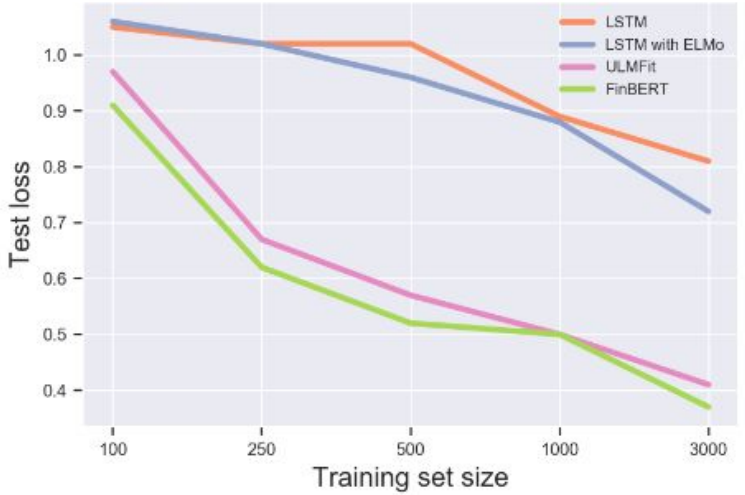


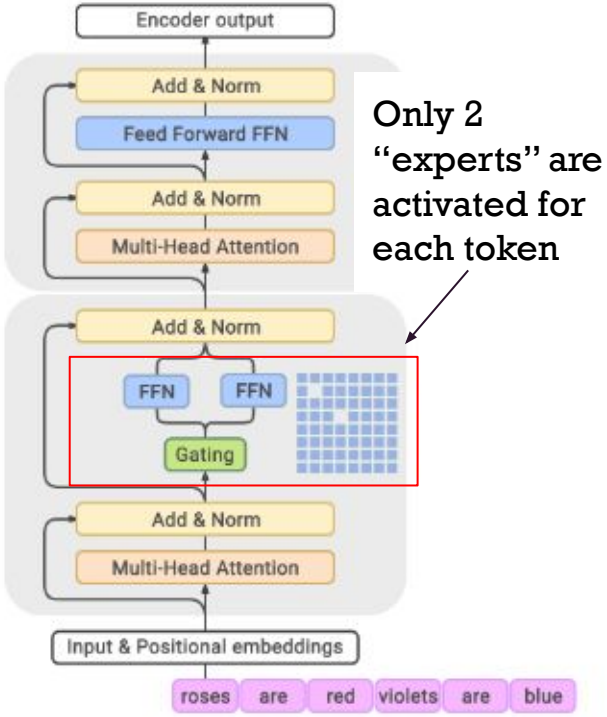
Figure 2: Test loss different training set sizes

https://arxiv.org/pdf/2112.06905.pdf

GLaM: Efficient Scaling of Language Models with Mixture-of-Experts

*Equal contribution [1]Google. Correspondence to: Nan Du, Yanping Huang, and Andrew M. Dai <dunan@google.com, huangyp@google.com, adai@google.com>.

A sparsely activated decoder-only transformer model.



Only 2 "experts" are activated for each token

Over 1 trillion parameters*!

Figure 2. GLaM model architecture. Each MoE layer (the bottom block) is interleaved with a Transformer layer (the upper block). For each input token, *e.g.*, 'roses', the *Gating* module dynamically selects two most relevant experts out of 64, which is represented by the blue grid in the MoE layer. The weighted average of the outputs from these two experts will then be passed to the upper Transformer layer. For the next token in the input sequence, two different experts will be selected.

Table 4. Sizes and architectures of both MoE and dense models that we have trained in our experiments. Models are grouped by the number of activated parameters per token. All trained models share the same learning hyperparameters described in Session 5.1.

GLaM Model	Type	n_{params}	$n_{\text{act-params}}$	L	M	H	n_{heads}	d_{head}	E
0.1B	Dense	130M	130M	12	768	3,072	12	64	–
0.1B/64E	MoE	1.9B	145M						64
1.7B	Dense	1.7B	1.700B						–
1.7B/32E	MoE	20B	1.878B						32
1.7B/64E	MoE	27B	1.879B	24	2,048	8,192	16	128	64
1.7B/128E	MoE	53B	1.881B						128
1.7B/256E	MoE	105B	1.886B						256
8B	Dense	8.7B	8.7B	32	4,096	16,384	32	128	–
8B/64E	MoE	143B	9.8B						64
137B	Dense	137B	137B	64	8,192	65,536	128	128	–
64B/64E	MoE	1.2T	96.6B	64	8,192	32,768	128	128	64

Table 3. Data and mixture weights in GLaM training set.

Dataset	Tokens (B)	Weight in mixture
Filtered Webpages	143	0.42
Wikipedia	3	0.06
Conversations	174	0.28
Forums	247	0.02
Books	390	0.20
News	650	0.02

GLaM was trained on over 1.6 trillion tokens of text,

		GPT-3	GLaM	relative
cost	FLOPs / token (G)	350	**180**	–48.6%
	Train energy (MWh)	1287	**456**	–64.6%
accuracy on average	Zero-shot	56.9	**62.7**	+10.2%
	One-shot	61.6	**65.5**	+6.3%
	Few-shot	65.2	**68.1**	+4.4%

*Technically unfair when compared to dense models since for each token, the number of parameters actually used is only ~96.6B ($n_{\text{act-params}}$)

Lots of GPUs!!!

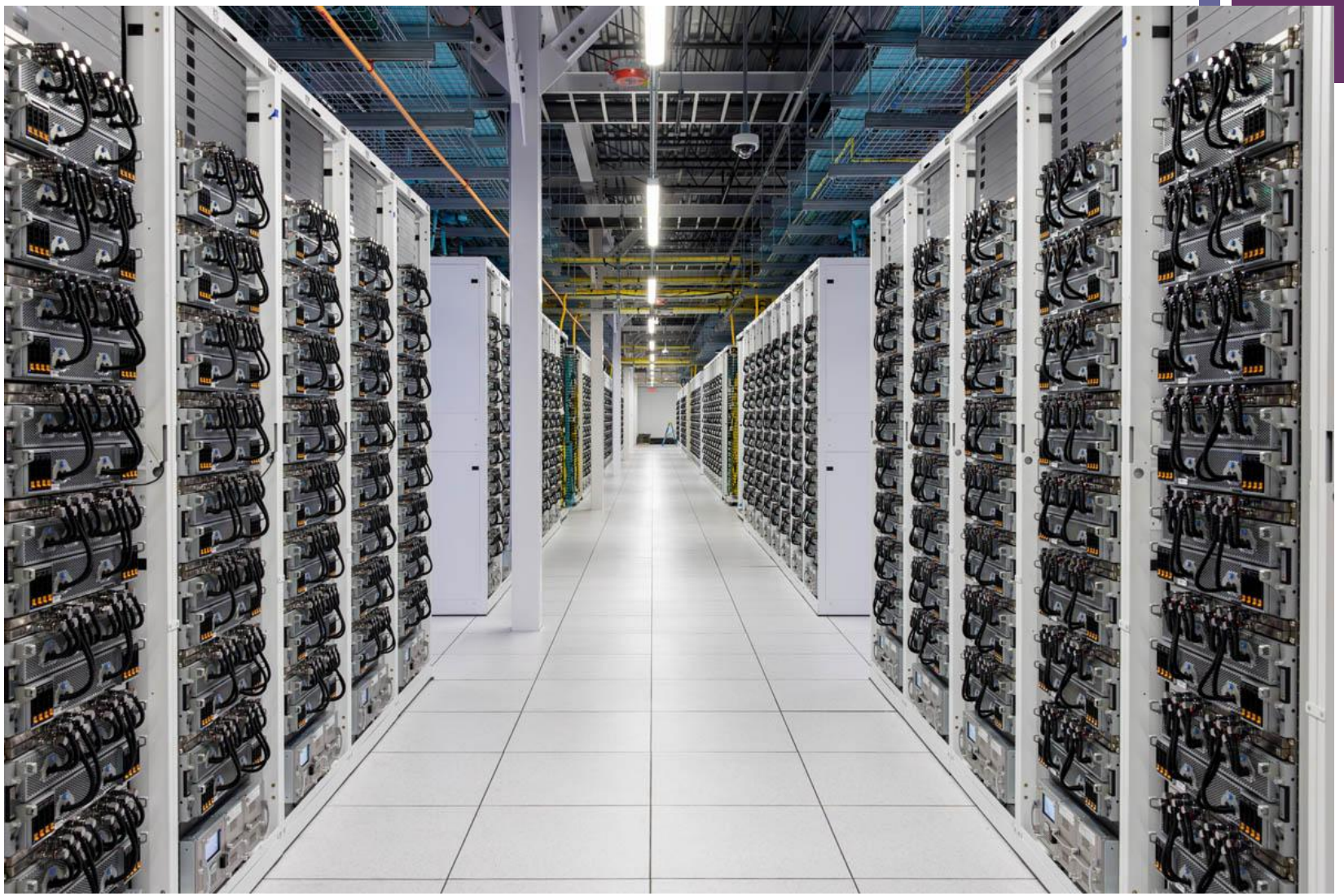
GPT-3 : 175 billion parameters

How many Gpus did gpt3 use?

According to the OpenAI's whitepaper, GPT-3 uses half-precision floating-point variables at 16 bits per parameter. This means the model would require at least 350 GB of VRAM just to load the model and run inference at a decent speed. This is the equivalent of at least 11 Tesla V100 GPUs with 32 GB of memory each. Sep 21, 2563 BE

Inside the 100K GPU xAI Colossus Cluster that Supermicro Helped Build for Elon Musk - ServeTheHome

$3-4 billion of GPUs



Things to think about for next time

What does the ability to predict the next word better gives us?