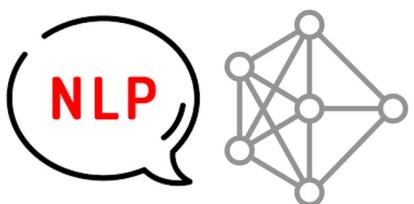


CHULA ΣENGINEERING  
Foundation toward Innovation

COMPUTER



# Language Modeling

2110572: Natural Language Processing Systems

Assoc. Prof. Peerapon Vateekul, Ph.D.

Department of Computer Engineering,  
Faculty of Engineering, Chulalongkorn University  
[Peerapon.v@chula.ac.th](mailto:Peerapon.v@chula.ac.th)

Credits to: Aj.Ekapol & TA team (TA.Pluem, TA.Knight, and all TA alumni)

# + Outline

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

+

## Introduction



# Introduction

คุณ | อาการ | กช

Maximal matching = 3

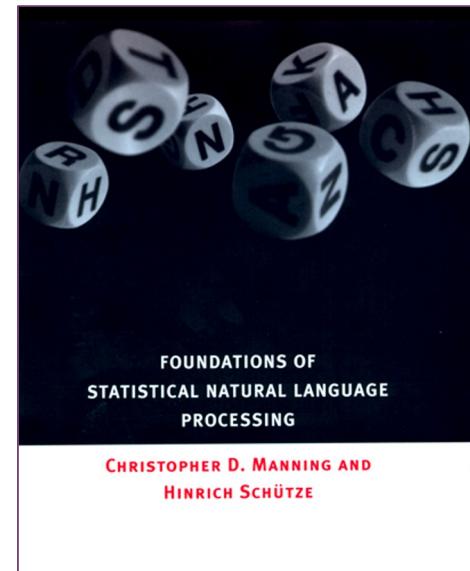
We need to verify with Language Model (LM)

คุณ | อาการ | กช

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

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

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





# Introduction (cont.)

- Application
  - 1) 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](https://en.wikipedia.org/wiki/Natural_language_generation)

- 2) Machine Translation
- 3) Speech Recognition

## Generating Spatio-Temporal Descriptions in Pollen Forecasts

Ross Turner, Somayajulu Sripada and Ehud Reiter

Ian P Davy

Dept of Computing Science,

Aerospace and Marine International,

University of Aberdeen, UK

Banchory, Aberdeenshire, UK

{rturner, ssripada, ereiter}@csd.abdn.ac.uk

idavy@weather3000.com

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



## Introduction (cont.)

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



## Introduction (cont.)

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

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

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

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

## + Problem with full estimation

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

Training:

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

Test:

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

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

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

+

N-grams



# N-grams: a probability of next word

- **Markov Assumption**

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

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

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

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



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

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



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

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

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

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

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

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

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

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



## N-grams (cont.): Example of Bigrams Prob.

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

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

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

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



## N-grams (cont.): Example

### ■ Estimating Bigrams Probability

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

| <b>Bigrams Unit</b> | <b>Bigrams Probability</b> |
|---------------------|----------------------------|
| 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                |

| <b>Bigrams Unit</b>           | <b>Bigrams Probability</b> |
|-------------------------------|----------------------------|
| 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                 |

14

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

+

## N-grams (cont.): Counting table

*assume on real data*

- 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” →  $c(\text{prev}, \text{cur}) = c(w_{i-1}, w_i) = c(\text{want}, i) = 827$

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

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

curr

prev

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

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

+

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

16

- 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     |



|         | 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       |

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

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

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

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

## + N-grams (cont.): Log likelihood

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

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

+

## 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 models.
- A **better language model** is one that assigns a higher probability to the test set
  - **Perplexity** can be seen a normalized version of the probability of **the test set**

## + Perplexity (2)

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

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

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

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



## Perplexity (3)

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

---

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

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

---


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



## Perplexity (4): Intuition of Perplexity

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

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

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

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

**10 times**

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

# Perplexity 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): $\text{PP}(W)$ of “I eat chinese food” Bi-grams

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

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

+ Smoothing  
Motivation: 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.)

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

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





## 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**
      - This is fixed in advance.
      - 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

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

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

+

## Smoothing Techniques

# +

## Smoothing

- Our training data is very sparse, sometimes we **cannot find the n-grams (0)** that we want.
  - In some cases where we do not even have a **unigram** (a word or OOV), we will use “UNK” token instead
- Notable smoothing techniques
  - Add-one estimation (or Laplace smoothing)
  - Back-off
  - Interpolation
  - Kneser–Ney Smoothing

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

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

ln(0) is undefined!

## + Smoothing#1: Add-one estimation

- Add-one estimation (or Laplace smoothing)
  - We add one to all the n-grams counts
  - For bigram, where V is the number of unique words in the corpus:

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

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



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



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

- Add-one estimation (or Laplace smoothing)
  - Pros
    - Easiest to implement
  - Cons
    - Usually perform poorly compared 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
- $\lambda$  is chosen from testing on validation data set, and the summation of  $\lambda_i$  is 1 ( $\sum \lambda_i = 1$ )
- Interpolation is like merging several models



## Smoothing#3: Interpolation (cont.)

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

- Interpolation

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

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

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

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



## Absolute discounting: save some probability mass for the zeros

- Suppose we want to subtract a 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 average 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  $d$**

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

discounted bigram                          Interpolation weight  
 ↙                                                    ↖  
 unigram                                            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 |      |   |   |

$$\begin{aligned}
 P(b) &= 0.1, P(c) = 0.3 \\
 P(b|a) &= 0 + xP(b) \\
 P(c|a) &= 0 + xP(c) \\
 xP(b) + xP(c) &= 0.1
 \end{aligned}$$

| 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 (all possible combinations)
$$P_{continuation}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$
- If our corpus contains these bigrams
  - { San Francisco, San Francisco, San Francisco, Sun glasses, Reading glasses, Colored glasses }
- $P_{cont}(Francisco) = (1/4) = 0.25$
- $P_{cont}(glasses) = (3/4) = 0.75$
- Now, a word like “Francisco” will have low  $P_{continuation}$



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

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

discounted bigram  
 interpolation weight  
 ↙  
 unigram

- Kneser–Ney Smoothing

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

- 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  $w_{i-1}$



## 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

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

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

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



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

Create a unigram 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> |
|-----|---|----|-----|------|-------|------|------|
| 4   | 4 | 3  | 3   | 1    | 1     | 1    | 4    |



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

Create a bigram counting table

|       | <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    |

training corpus:

<s> I am Sam </s>  
<s> Sam I am </s>  
<s> I am Sam </s>  
<s> I like green eggs </s>

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

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

$$P_{\text{kn2}}(\text{am} | \text{<s>}) = (\max(0 - 0.75, 0) / 4) + (0.75 * 2 / 4) * (1 / 11) = 0.03409$$

$$P_{\text{kn2}}(\text{Sam} | \text{am}) = (\max(2 - 0.75, 0) / 3) + (0.75 * 2 / 3) * (2 / 11) = 0.5076$$

$$P_{\text{kn2}}(\text{</s>} | \text{Sam}) = (\max(2 - 0.75, 0) / 3) + (0.75 * 2 / 3) * (3 / 11) = 0.5530$$

$$\text{LL} = \ln(0.03409) + \ln(0.5076) + \ln(0.5530) = -4.6492$$

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

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

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

|       | <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>”

$$\text{Perplexity} = \exp(-\text{LL}/n) = \exp(-(-4.6492)/3) = 4.7$$

# +

## 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

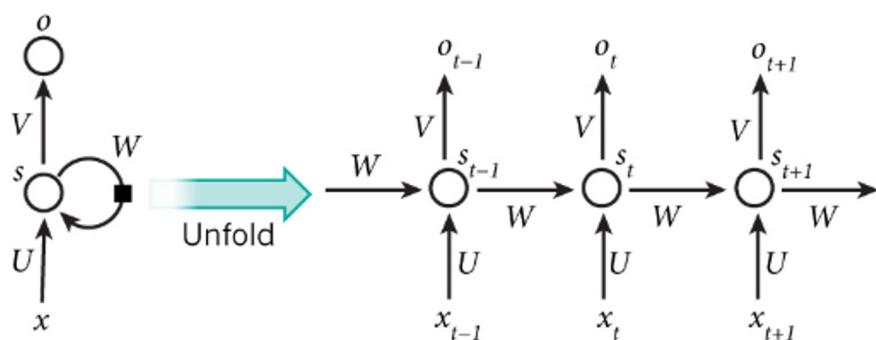
- Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff (e.g. if 4-gram not found, try 3-gram, etc)
- However,
  - It needs **a lot of memory** to store all those n-grams
  - **It lacks long-term dependency**
  - "Jane walked into the room. John walked 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

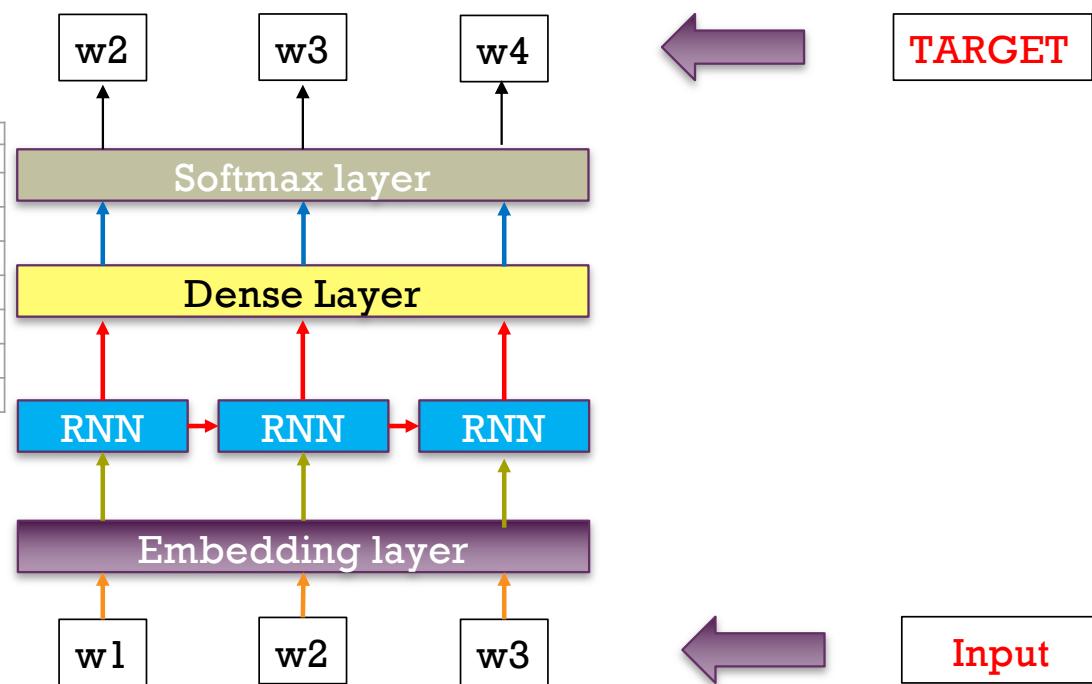


# + Neural Language Model (cont.)

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

| i       | i  | want | to  | eat | chinese | food | lunch | spend |
|---------|----|------|-----|-----|---------|------|-------|-------|
| want    | 5  | 827  | 0   | 9   | 0       | 0    | 0     | 2     |
| to      | 2  | 0    | 608 | 1   | 6       | 6    | 5     | 1     |
| 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



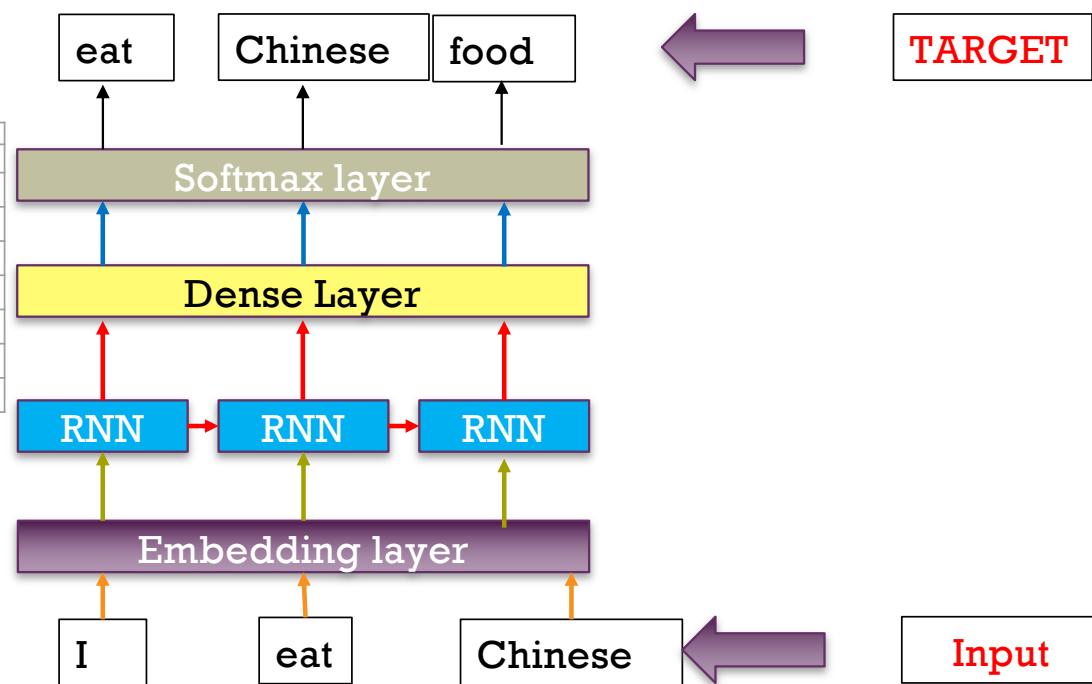
# +

## 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:

- 

- 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

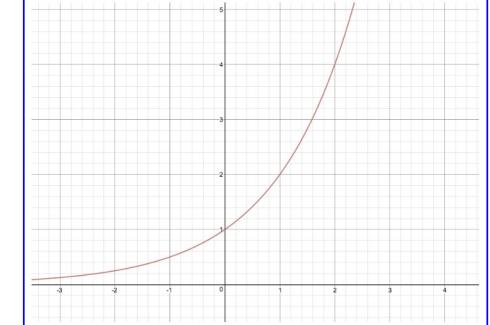
- Actually, we are calculating perplexity

- Perplexity =  $e^J$

For each training example,  
Whole training data (T)

Softmax (all classes V)

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



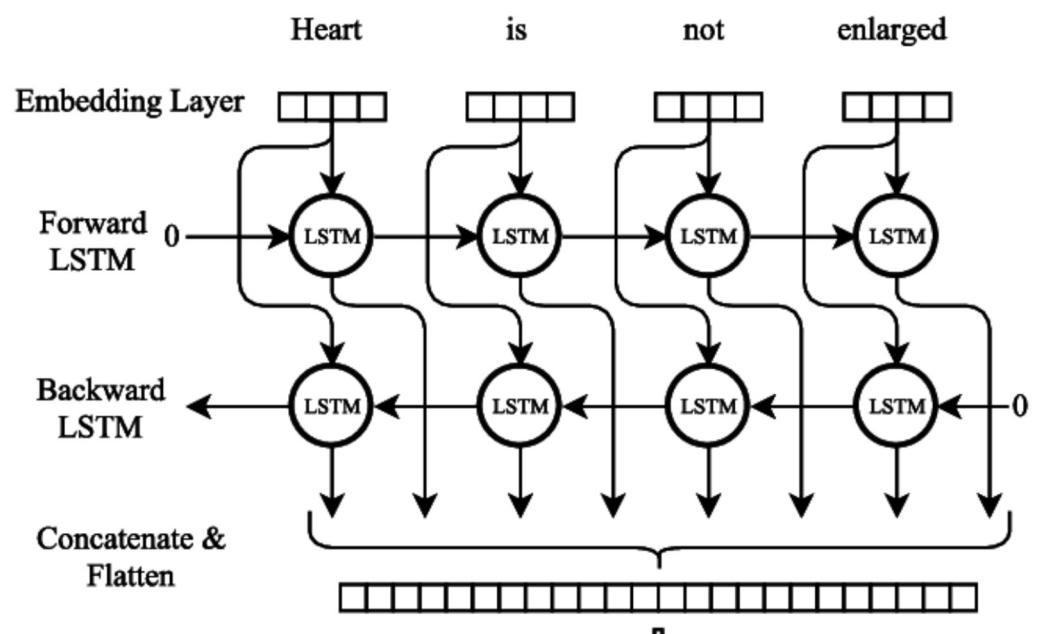
$$\text{Perplexity} = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}},$$

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

# +

## Neural Language Model (cont.)

- RNN suffers from vanishing gradient
  - Use a RNN that has memory unit such as
    - Long Short Term Memory (LSTM)
    - Gate Recurrent Unit (GRU)
- Bidirectional RNN?
  - Bidirectional RNN **cannot** apply here since we predict the next word and cannot use future information (violating assumption).
  - However, special types of special networks (Transformer: **BERT**) can be applied without violating assumptions.



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

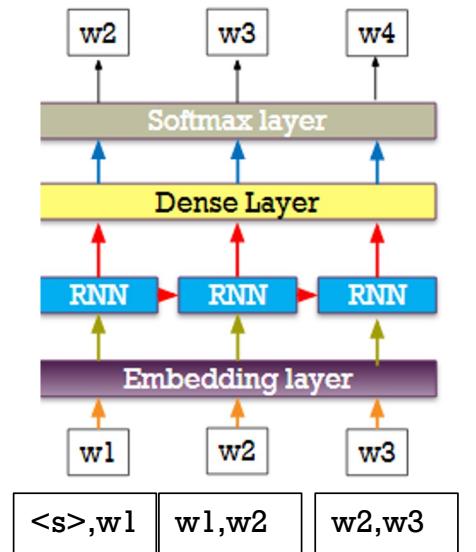


# Neural Language Model (cont.)

## ■ Conclusion

### ■ Neural Language Model vs. N-grams Model

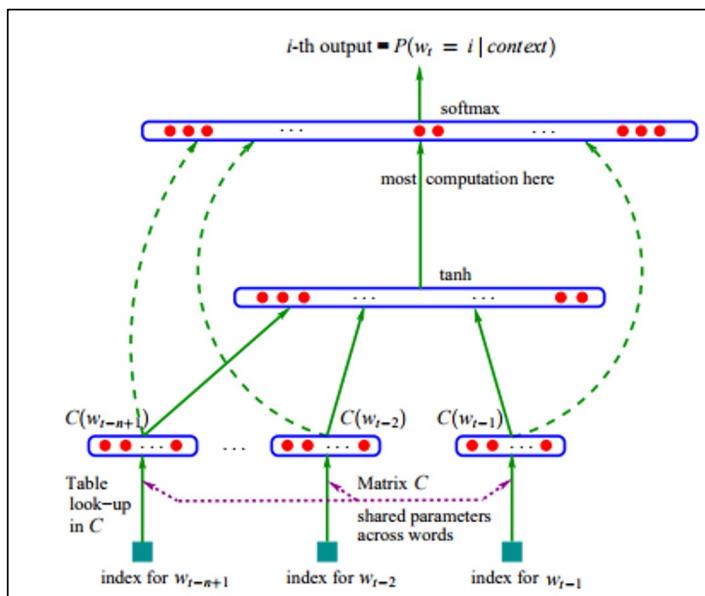
- A competitive n-grams model needs **huge amount of memory**, larger than RNN
- Neural Language Model usually **perform better** than n-grams model because
  - it considers **long-term** dependency information
  - It subtly processes word semantics via **word embeddings**
- **However**, n-grams are still quite useful and often are incorporated into neural language models as features or for beamsearch pruning.  
**(ngrams → NN)**



# +

## Neural Language Model (cont.)

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

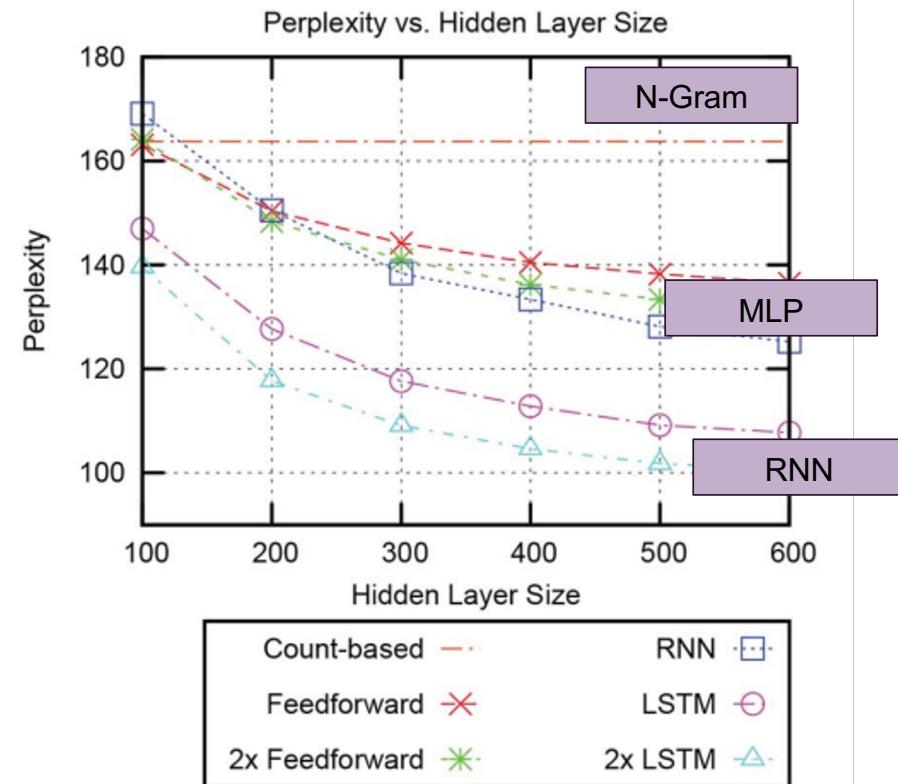




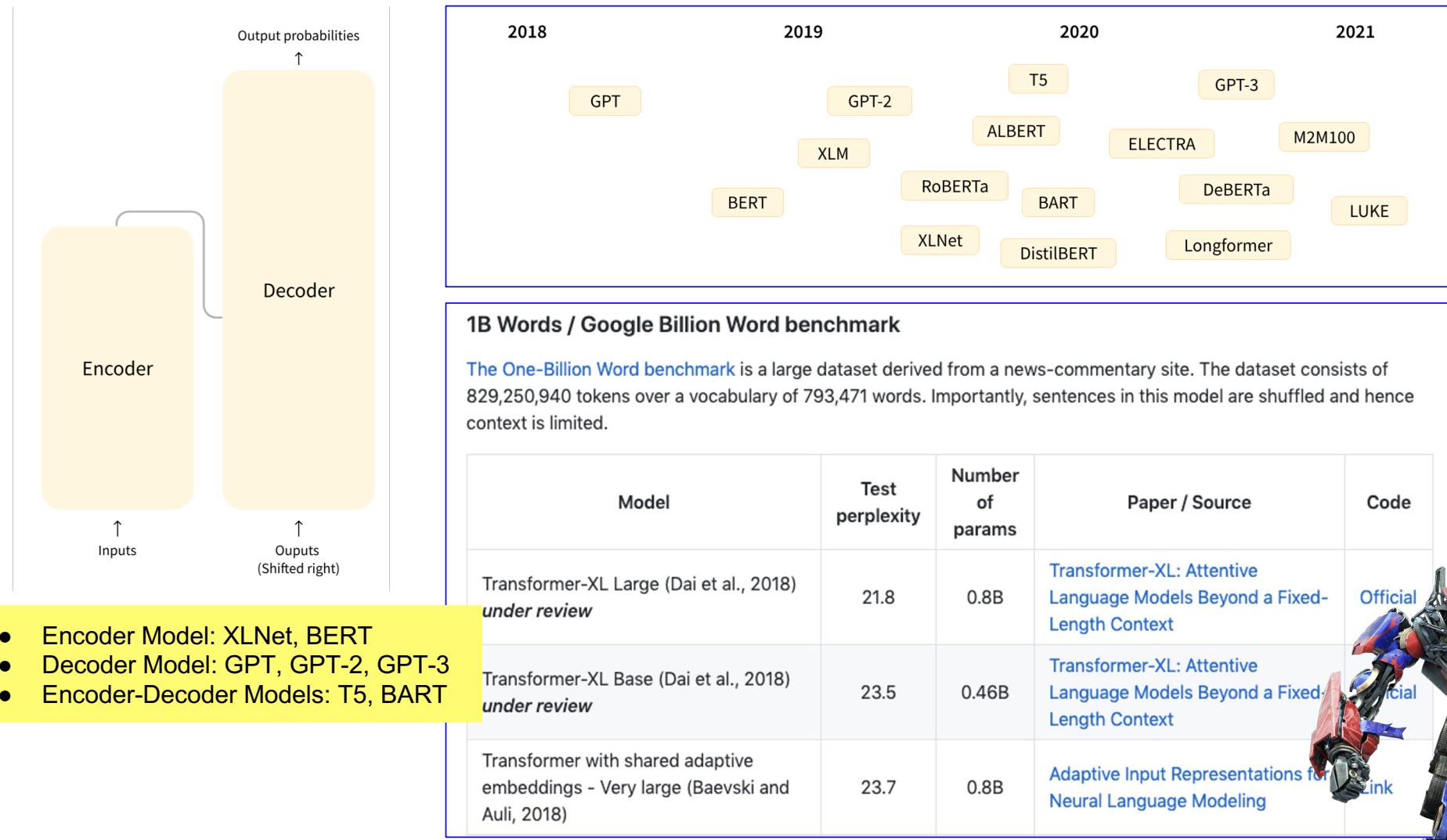
## Neural Language Model (cont.)

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

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



## + Language Model SOTA (2019; outdated)



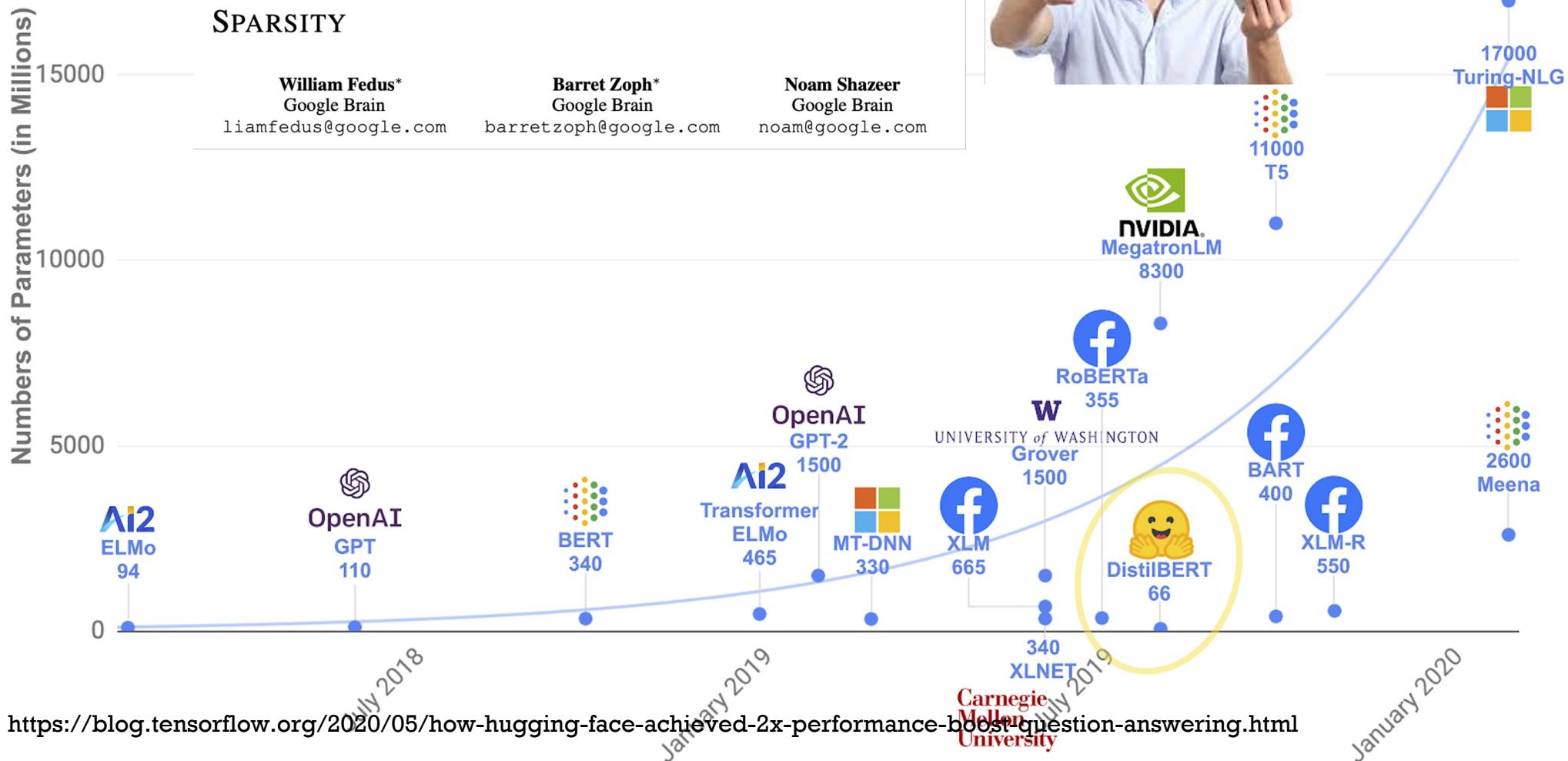
# Outdated

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

William Fedus\*  
Google Brain  
liamfedus@google.com

Barret Zoph\*  
Google Brain  
barrettzoph@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://medium.com/airesearch-in-th/wangchanberta-%E0%B9%62%E0%B8%A1%OpenDataForEveryone-%E0%B8%94%E0%B8%A5%E0%B8%9B%E0%B8%A3%E0%B8%9C%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%85%E0%B9%88%E0%B8%83%E0%B8%AB%E0%B8%8D%E0%B9%88%E0%B9%81%E0%B8%A5%E0%B8%80%E0%B8%81%E0%B9%89%E0%B8%82%E0%B8%A7%E0%B8%AB%E0%B8%99%E0%B9%89%E0%B8%97%E0%B8%88%E0%B8%AA%E0%B8%83%E0%B8%94%E0%B9%83%E0%B8%99%E0%B8%82%E0%B8%93%E0%B8%80%E0%B8%99%E0%B8%85%E0%B9%89-d920c27cd433](https://medium.com/airesearch-in-th/wangchanberta-%E0%B9%62%E0%B8%A1%OpenDataForEveryone-%E0%B8%94%E0%B8%A5%E0%B8%9B%E0%B8%A3%E0%B8%9C%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%85%E0%B9%88%E0%B8%83%E0%B8%AB%E0%B8%8D%E0%B9%88%E0%B9%81%E0%B8%A5%E0%B8%81%E0%B9%89%E0%B8%82%E0%B8%A7%E0%B8%AB%E0%B8%99%E0%B9%89%E0%B8%97%E0%B8%88%E0%B8%AA%E0%B8%83%E0%B8%94%E0%B9%83%E0%B8%99%E0%B8%82%E0%B8%93%E0%B8%80%E0%B8%99%E0%B8%85%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 เดือนในการเทรน โมเดลให้ loss ลดลงมาในระดับที่ 2.592 (perplexity = 13.356) ณ step ที่ 360,000 จากทั้งหมด 500,000 steps ณ วันนี้ โมเดลก็ยังถูกเทรนอย่างต่อเนื่อง ในศูนย์วิจัยที่วังจันทร์ จึงเป็นไปได้ว่าเราจะได้ โมเดลที่มีประสิทธิภาพดียิ่งกว่ามาใช้ในอนาคต

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

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%        |

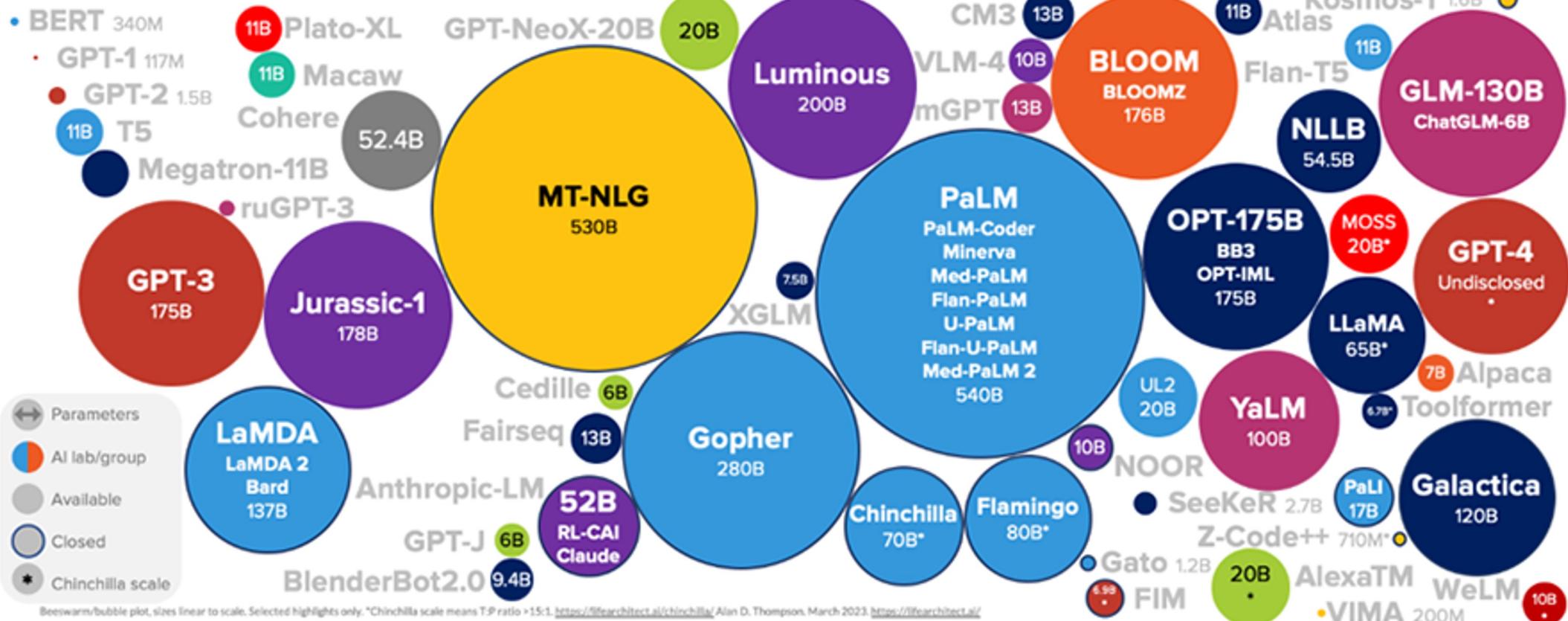
Table 5: OOV rates of WangchanBERTa's tokenizer and our expanded tokenizer for each dataset.

| Dataset                   | Proportion of Samples with Unassimilated Loanwords | Performance Gain Compared to WangchanBERTa |
|---------------------------|----------------------------------------------------|--------------------------------------------|
| 1. wisestight_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                              |

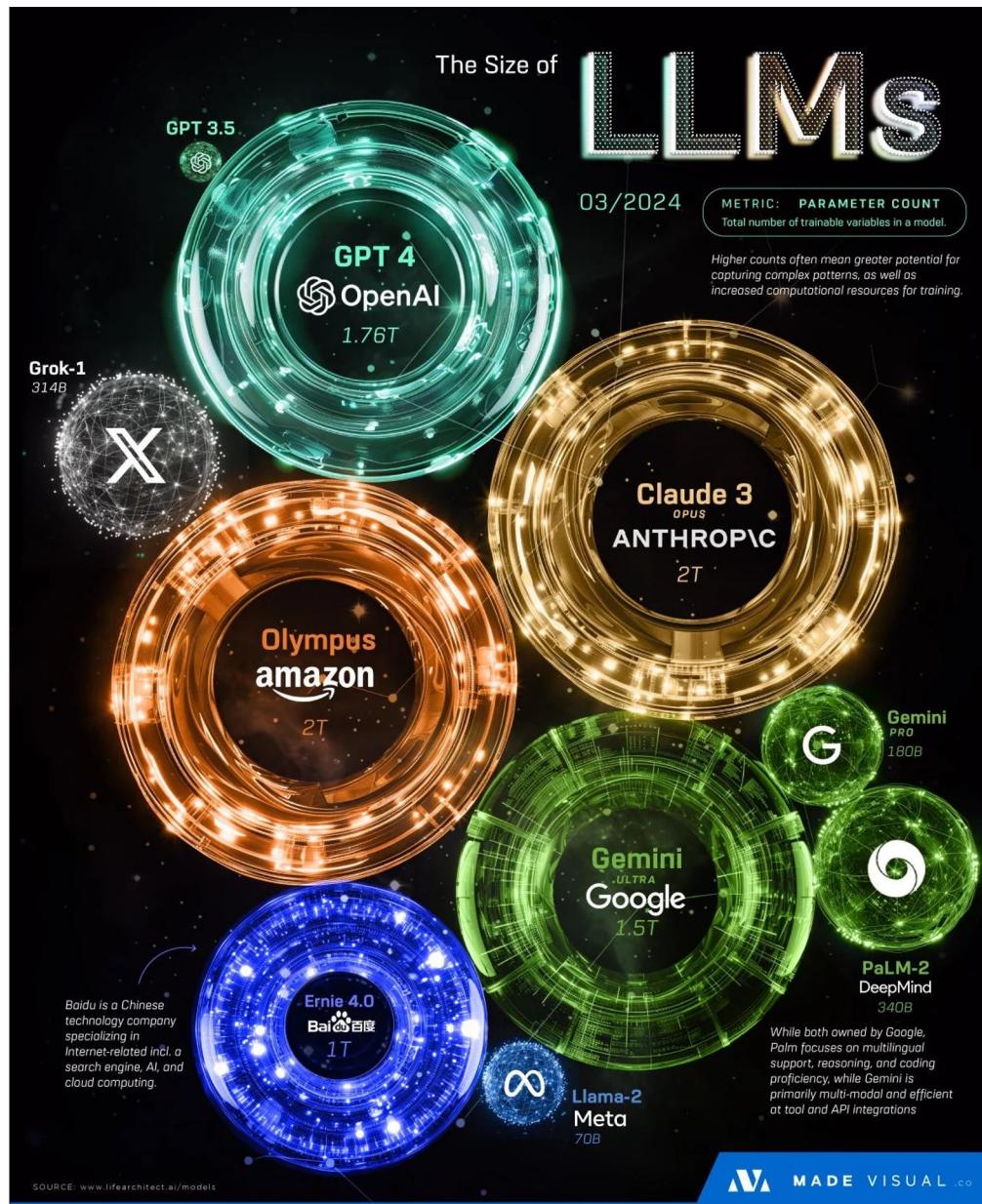
Table 6: Proportion of samples with unassimilated English words compared with performance gain for each dataset.

# LANGUAGE MODEL SIZES TO MAR/2023

<https://blogs.cfainstitute.org/investor/2023/05/26/chatgpt-and-large-language-models-six-evolutionary-steps/>



LifeArchitect.ai/models



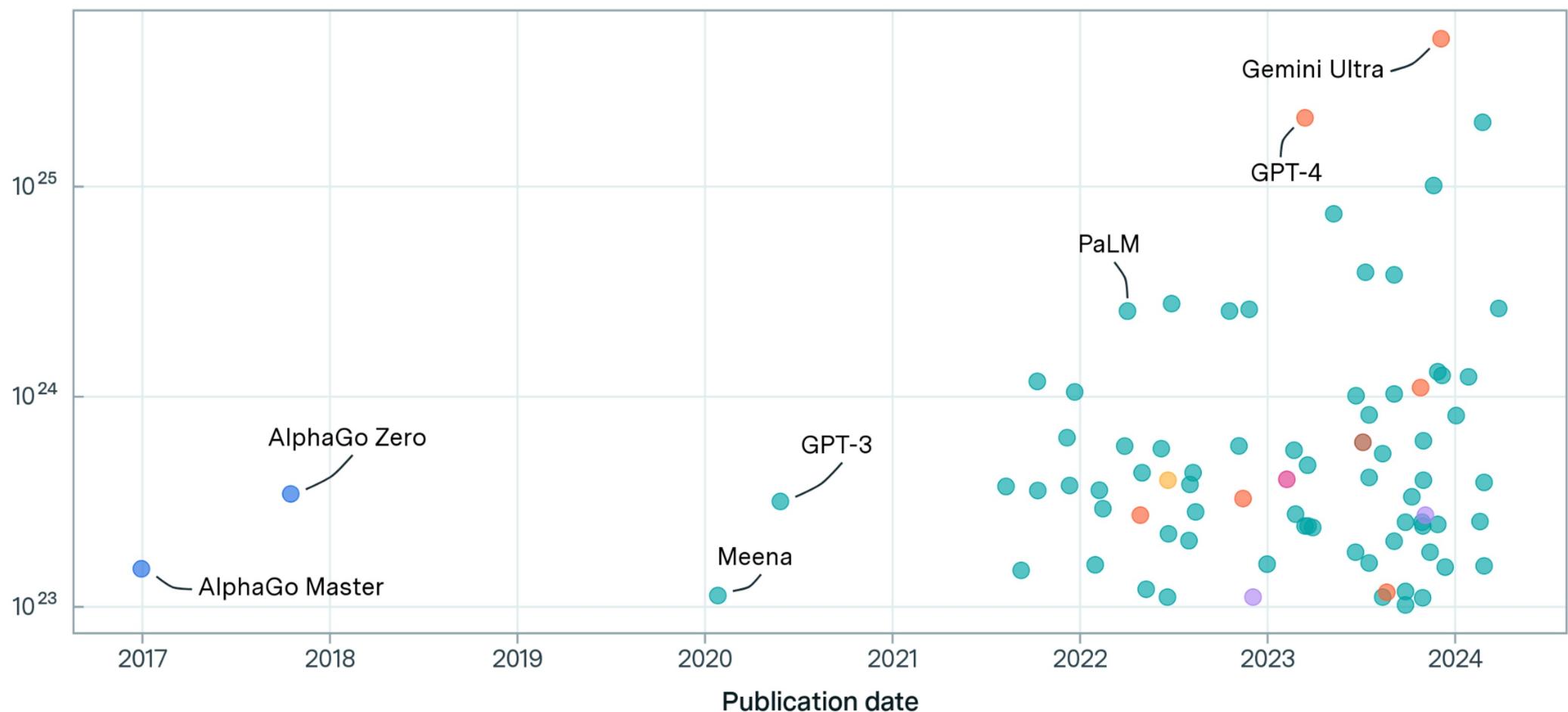
<https://epoch.ai/blog/tracking-large-scale-ai-models>



## Large-scale models by domain and publication date

Training compute (FLOP)

● Language ● Multimodal ● Speech ● Games ● Drawing ● Biology ● Vision





# Thai LLMs

by scb10x

## Typhoon 2 Text

Latest Official Text ThaiLLM release by SCB 10X.

huggingface.co

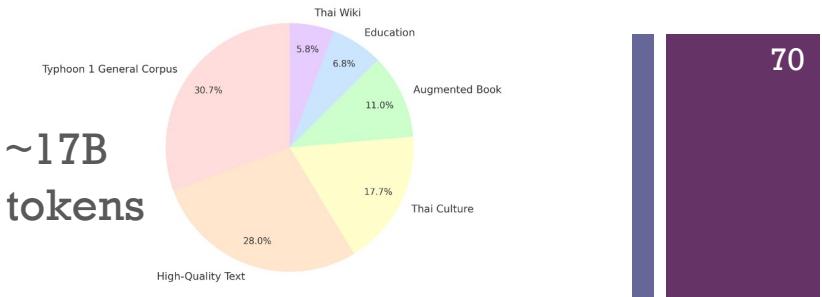


OpenThaiGPT

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

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



70

| Model                  | IFEval<br>TH | IFEval<br>EN | MTBench<br>TH | MTBench<br>EN | CS<br>0.7 | CS<br>1.0 | FC<br>TH | FC<br>EN | GSM8K<br>TH | GSM8K<br>EN | Math<br>TH | Math<br>EN | HumanEv<br>TH | HumanEv<br>EN | MBPP<br>TH | MBPP<br>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<br>TH | IFEval<br>EN | MTBench<br>TH | MTBench<br>EN | CS<br>0.7 | CS<br>1.0 | FC<br>TH | FC<br>EN | GSM8K<br>TH | GSM8K<br>EN | Math<br>TH | Math<br>EN | HumanEv<br>TH | HumanEv<br>EN | MBPP<br>TH | MBPP<br>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

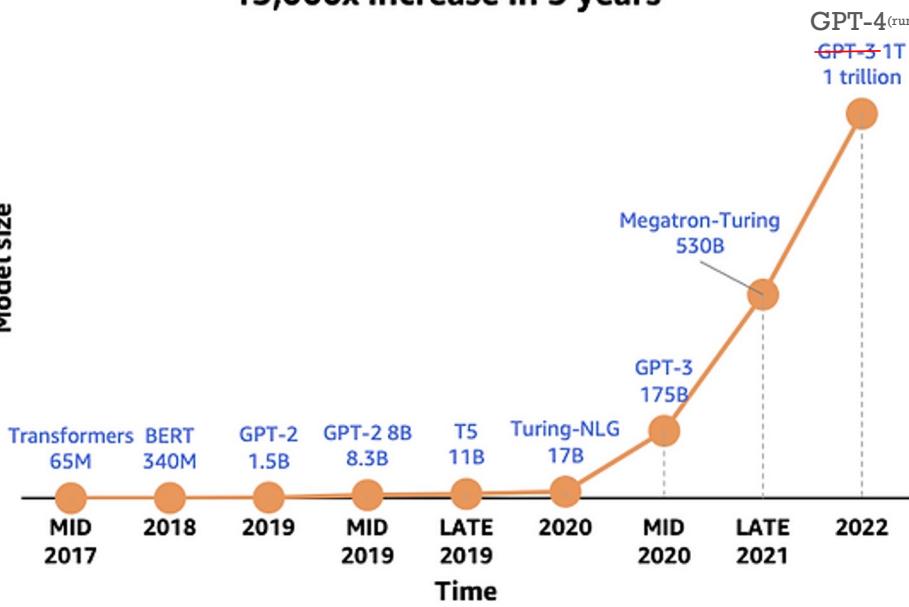
<https://arxiv.org/pdf/2412.13702.pdf>  
<https://huggingface.co/openthaigpt/openthaigpt1.5-7b-instruct>



# Large Language Models (LLMs)

*As of Jan-2025, there are 163,240 LLMs in HF!*

15,000x increase in 5 years



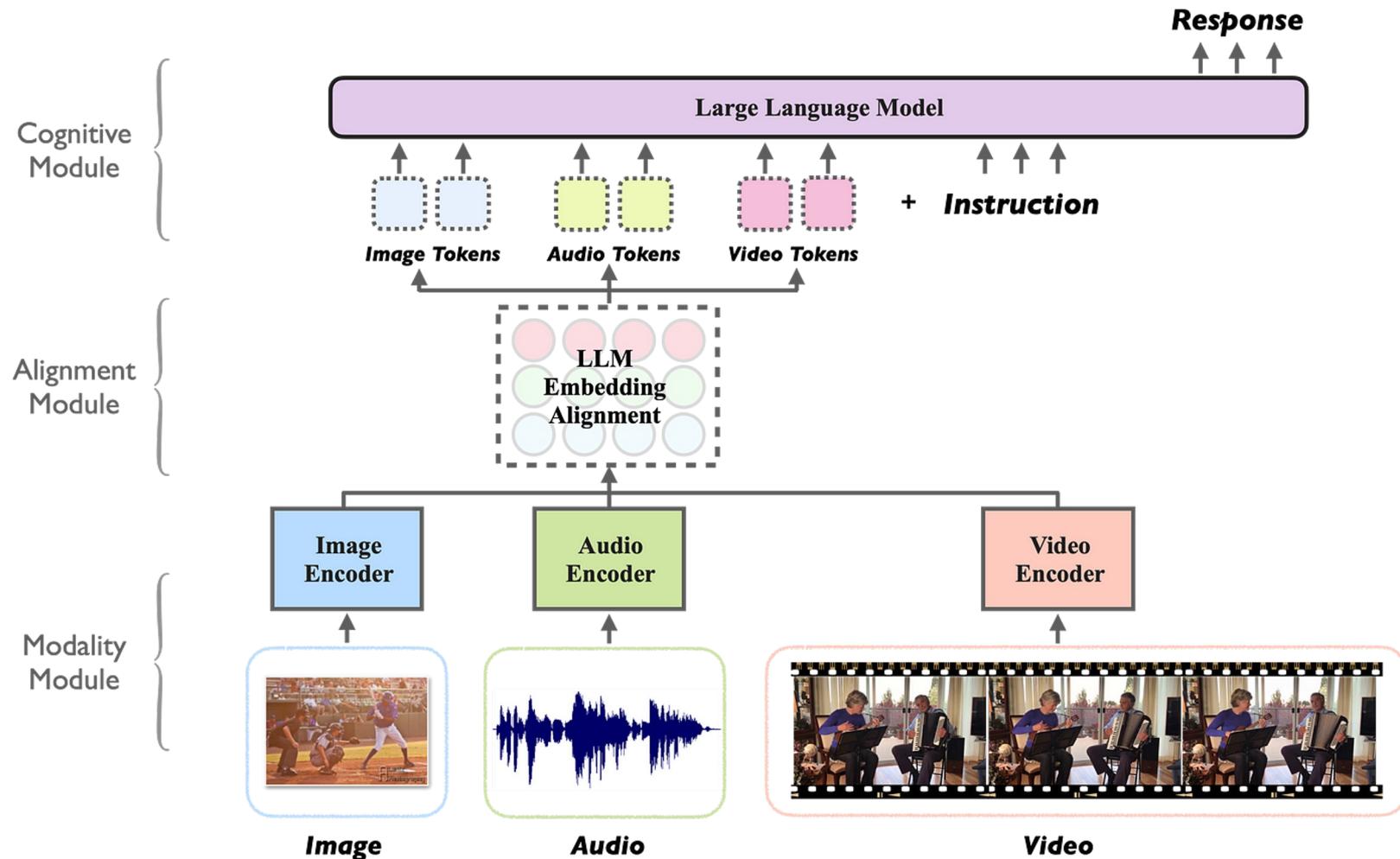
Models 163,240 Filter by name Full-text search Sort: Trending

163,240 LLMs and counting!

|                                           |                                      |
|-------------------------------------------|--------------------------------------|
| meta-llama/Llama-3.3-70B-Instruct         | CohereForAI/c4ai-command-r7b-12-2024 |
| Qwen/QwQ-32B-Preview                      | ibm-granite/granite-3.1-8b-instruct  |
| tiiuae/Falcon3-10B-Instruct               | meta-llama/Llama-3.2-1B              |
| meta-llama/Llama-3.1-8B-Instruct          | Qwen/Qwen2.5-Coder-32B-Instruct      |
| nvidia/Llama-3.1-Nemotron-70B-Instruct-HF | NousResearch/Hermes-3-Llama-3.2-3B   |
| deepseek-ai/DeepSeek-V2.5-1210            | tiiuae/Falcon3-7B-Instruct           |
| tiiuae/Falcon3-10B-Base                   | tiiuae/Falcon3-1B-Instruct           |
| meta-llama/Llama-3.2-3B-Instruct          | google/gemma-2-2b-it                 |



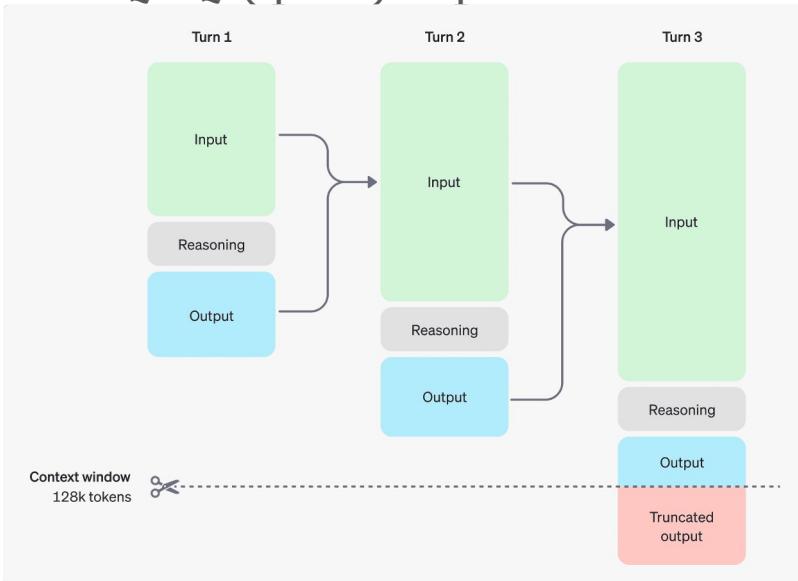
# Multimodal LLM



# + Reasoning Models

GPT-o1 and o3 (Dec 2024)

QwQ (qwen) - open-source!



<https://lunary.ai/blog/open-ai-o1-reasoning-models>

<https://www.thealgorithmicbridge.com/p/openai-o3-model-is-a-message-from>

**Self-taught Reasoner (STaR): Get better data by fine-tuning with reasoning traces**

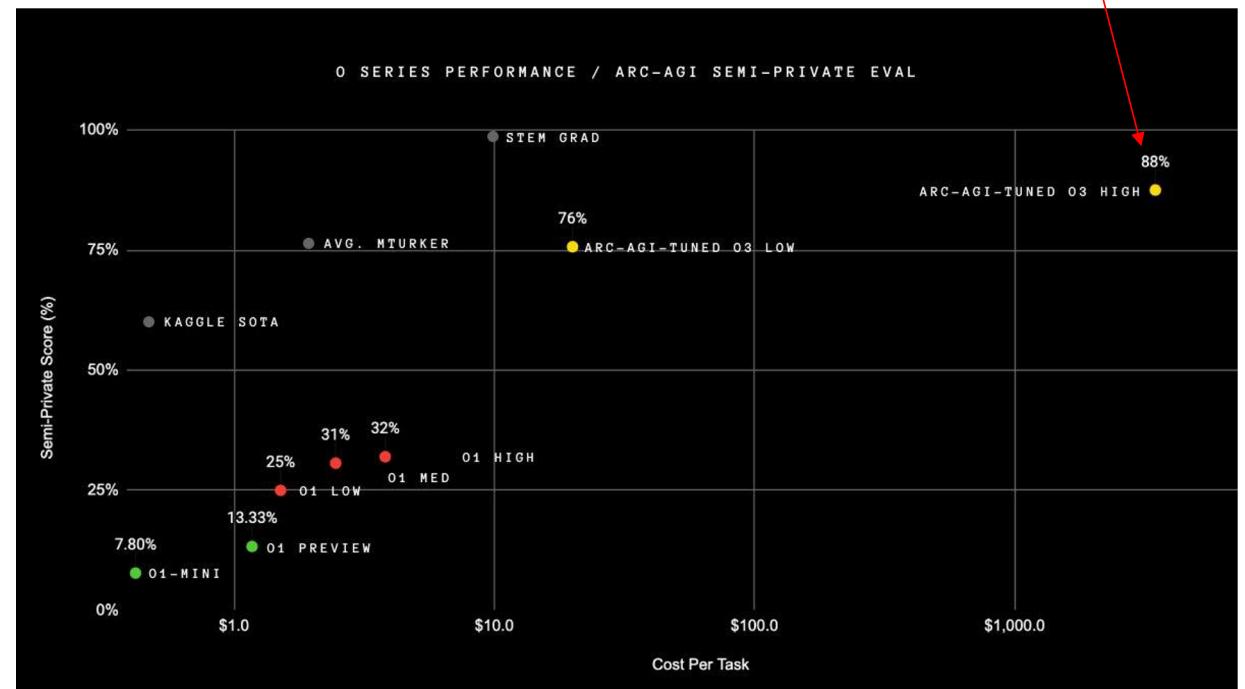
- LLMs get the right sequence more often with Chain of Thought to condition the next few tokens with previous reasoning/context
- Why not imbue such reasoning **into the expert data?**

In the fine-tuned data, put in more distribution of data (simple to difficult problems).

Figure 1: An overview of STaR and a STaR-generated rationale on CommonsenseQA. We indicate the fine-tuning outer loop with a dashed line. The **questions** and ground truth **answers** are expected to be present in the dataset, while the **rationales** are generated using STaR.

Credit Aj.Piyalitt Ittichaiwong's post on 2 Jan 2025

o3 → ~\$3000/task



# Conclusion

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

+

Thank you ☺