

# Language Model Basics

## Artificial Intelligence: Convergence and Application

Seonghyeon Lee

School of Computer Science and Engineering  
Kyungpook National University

February 28, 2025

# Overview

---

1. Language Models
2. Tokenization
3. Model Architecture
4. Training
5. Decoding

# Two Lines of Research with Generative AI

---

1. The first one is the use of generative AI.
  - I don't care how they works.
  - All I want to know is that how to use this new tools for my tasks.
2. The other one is the development of generative AI.
  - I want to understand the underlying mechanism of generative AI.
  - What I want is to develop a model that can beat OpenAI o3.

We will cover both approaches throughout this course. Today, we explore high-level mathematical justification of how to build a large language model, a core component for generative AI.

# What is a Language Model?

---

A language model is any model that outputs a probability distribution over the next token<sup>1</sup> in a sequence given the previous tokens in the sequence.

$$P(y_t | y_{1:t-1})$$

Historically, language models were statistical n-gram models. Instead of taking into account the full history of the sequence, they approximated this history by just looking back a few words.

---

<sup>1</sup>For now, let's assume token is word. We'll come back this.

# Example of Language Modeling

---

Suppose we are building a statistical language model using a text corpus<sup>2</sup>,  $\mathcal{C}$ .

We observe that the word "apple" follows the words "eat the" 2% of the times that "eat the" occurs in  $\mathcal{C}$ .

We can model this observation into the following equation:

$$P(\text{"apple"} | \text{"eat the"}) = 0.02$$

---

<sup>2</sup>Corpus means a bunch of texts

# Computing Sequence Likelihood

---

Language models output the likelihood<sup>3</sup> of the next word.

$$P(y_t | y_{1:t-1})$$

Often we will talk about the likelihood of an entire sequence.

$$P(Y) = P(y_1, \dots, y_T)$$

---

<sup>3</sup>The probability of observing an outcome under a probability distribution.

## Computing Sequence Likelihood (Cont')

---

Sequence likelihood can be computed from an LM using the chain rule.

$$P(\text{"I eat the apple"}) = P(\text{"apple"} | \text{"I eat the"}) P(\text{"I eat the"})$$

$$P(\text{"I eat the"}) = P(\text{"the"} | \text{"I eat"}) P(\text{"I eat"})$$

$$P(\text{"I eat"}) = P(\text{"eat"} | \text{"I"}) P(\text{"I"})$$

Mathematical notation:

$$P(Y) = \prod_{i=1}^T P(y_i | y_{1:i-1})$$

# Conditional Language Modeling

---

Up to this point, we use a random variable  $Y$  to express a text.

From now on, we clarify the role of random variable  $Y$  to **the text generated by the language model**.

We can design the model to predict the next word  $y_i$  given the previous generated ones  $y_{1:i-1}$  and **some additional conditioning text  $X$** .

At each step the LM predicts:  $P(y_t | y_{1:t-1}, x_{1:l})$ .



# Parametrization

---

We introduce parameters  $\theta$  into the model, which represents the probability distribution of next words.

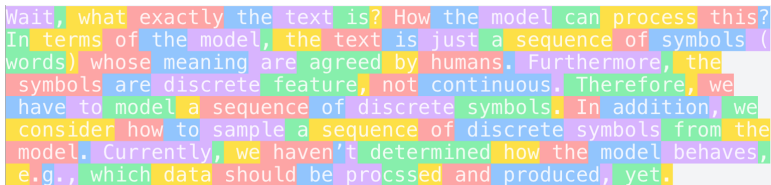
$$P_{\theta}(y_i | y_{1:i-1}, x_{1:l})$$

The remaining part is how to produce the probability of next word using  $\theta$ ,  $x_{1:l}$  and  $y_{1:i-1}$ .

# Language Model Interface

---

$X = x_{1:j}$  and  $Y = y_{1:t}$  are text. Text is just a sequence of words whose meaning are agreed by humans.

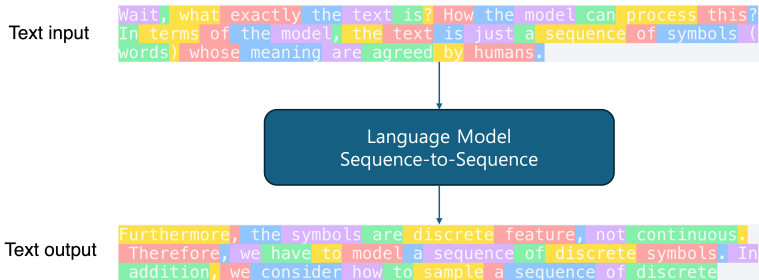
A rectangular text box with a background of a multi-colored grid, where each cell is a different color (e.g., red, green, blue, yellow, purple). The text inside the box is as follows:

Wait, what exactly the text is? How the model can process this?  
In terms of the model, the text is just a sequence of symbols (words) whose meaning are agreed by humans. Furthermore, the symbols are discrete feature, not continuous. Therefore, we have to model a sequence of discrete symbols. In addition, we consider how to sample a sequence of discrete symbols from the model. Currently, we haven't determined how the model behaves, e.g., which data should be processed and produced, yet.

# Sequence-to-Sequence Models

---

The model processes text (a sequence of words) and produces text (a sequence of words).



# Versatility in Sequence-to-Sequence Models

---

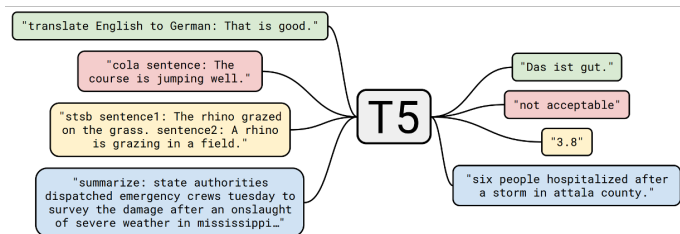
Depending on the semantic of text input and output, the models do different task.

- Translation - Input: English sentence, Output: German sentence.
- CoLA - Input: A sentence, Output: Linguistic acceptability.
- STS - Input: Two sentences, Output: Semantic relevance.
- Summarization - Input: paragraph, Output: summary.

# Text-to-Text Tasks

---

The most fascinating thing is that these tasks could be done in a single model! Raffel et al. (2020)



# Tokenization

---

How to convert a text into a sequence of words?

For instance, splitting text with word boundary works well.

```
"I love you": ["I", "_love", "_you"]
```

Formally, tokenization is the task of taking text (or code or music) and turning it into a sequence of discrete items, called **tokens**.

# Other Tokenization

---

We can split the text into a sequence of characters.

```
"I love you": ["I", " ", "l", "o", "v", "e", " ",  
               "y", "o", "u"]
```

A sequence of characters is too long. Another option is using subword.

```
"I love you": ["I", "_lov", "e", "_y", "ou"]
```

## Tokenization property

Effective tokenization introduces (1) small number of distinct tokens with (2) the short sequence length.

# Vocabulary

---

A vocabulary  $\mathcal{V}$  is the list of all available tokens.

Mostly, tokenization splits text into token sequence where each token is in the vocabulary.

$$y_{1:L} = \text{tokenization}(t) \text{ where } y_{1:L} \in \mathcal{V}^L$$

If we use character-level tokenization, what tokens would make up the vocabulary? Can you predict the size of vocabulary?

If we use word-level tokenization, what tokens would make up the vocabulary? Can you predict the size of vocabulary?

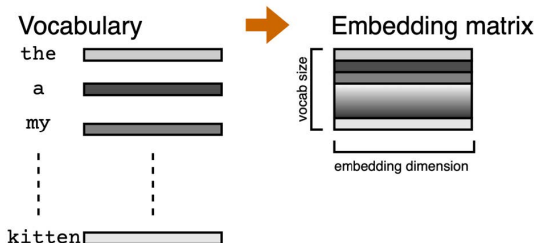


# Discrete Tokens into Continuous Vectors

Neural networks cannot operate on discrete tokens.

Instead, we build an embedding matrix which associates each token in the vocabulary with **a vector embedding**.

$$\mathbf{e}_{1:L} = \text{embedding}(y_{1:L}) \text{ where } \mathbf{e}_{1:L} \in \mathbb{R}^{L \times D}$$



# Transform Embedding Sequences

---

Transform and combine the sequence of embeddings to understand their meaning.

The transformed embedding sequence is called hidden states.

$$h_{1:L} = f(e_{1:L}) \text{ where } h_{1:L} \in \mathbb{R}^{L \times D'}$$

## Dimension configuration

In theory, embedding and hidden dimension size could be different.  
In practice, most language models use the same dimension size.

# Predict Next Token Probability

---

Predict probability distribution of the next token using hidden states.

Note that we need to predict probabilities for all tokens in the vocabulary to get the full probability distribution.

Therefore, the model usually predict scores for each token, called logits.

$$\hat{y}_L = g(h_t) \text{ where } \hat{y}_t \in \mathbb{R}^{|\mathcal{V}|}$$

Then, apply normalization operations, e.g., softmax, to get probability distribution for the next token.

$$P(y_{L+1} = v_i | y_{1:L}) = \frac{\exp(\hat{y}_L[i])}{\sum_{i=1}^{|\mathcal{V}|} \exp(\hat{y}_L[i])}$$

# Training Language Models

---

How to train this model?

In other words, how to make this model produce the probability distribution of next token of our interest?

A fair candidate of probability distribution of our interest is human text distribution.

A simple approach is to use the likelihood of human texts.

# Likelihood of Human Text

---

Given a set of human-written text  $\mathcal{C} = (s_1, \dots, s_{\|\mathcal{C}\|})$ , we write a likelihood of generating human-written texts as follow:

$$P(\mathcal{C}) = \prod_{i=1}^{\|\mathcal{C}\|} P(s_i) = \prod_{i=1}^{\|\mathcal{C}\|} \prod_{j=1}^{N_i} P(x_{ij} | x_{i,<j}) \quad (1)$$

A likelihood function (often simply called the likelihood) measures how well a model explains observed data by **calculating the probability of seeing that data under different parameter values of the model**.<sup>4</sup>

---

<sup>4</sup>[https://en.wikipedia.org/wiki/Likelihood\\_function](https://en.wikipedia.org/wiki/Likelihood_function)

# Negative Log Likelihood Loss

If we want the system to produce high-quality text, a simple movement is to maximize the likelihood of generating human texts.

$$\underset{\theta}{\text{maximize}} P_{\theta}(\mathcal{C})$$

For the computational efficiency, we apply log function and negation.

$$\underset{\theta}{\text{minimize}} -\log \prod_{i=1}^{||\mathcal{C}||} \prod_{j=1}^{N_i} P_{\theta}(x_{ij}|x_{i,<j}) = \sum_{i=1}^{||\mathcal{C}||} \sum_{j=1}^{N_i} -\log P_{\theta}(x_{ij}|x_{i,<j})$$

It is often called **negative log likelihood loss** and we minimize this value to maximize the likelihood.

## Numerical stability

Summation with log scale is numerically more stable than production with normal scale.

# Sampling Text from Language Model

---

Recall chain rule.

$$P(\text{"I eat the apple"}) = P(\text{"apple"} | \text{"I eat the"}) P(\text{"I eat the"})$$

$$P(\text{"I eat the"}) = P(\text{"the"} | \text{"I eat"}) P(\text{"I eat"})$$

$$P(\text{"I eat"}) = P(\text{"eat"} | \text{"I"}) P(\text{"I"})$$

Reverse the equation.

$$P(\text{"I eat"}) = P(\text{"eat"} | \text{"I"}) P(\text{"I"})$$

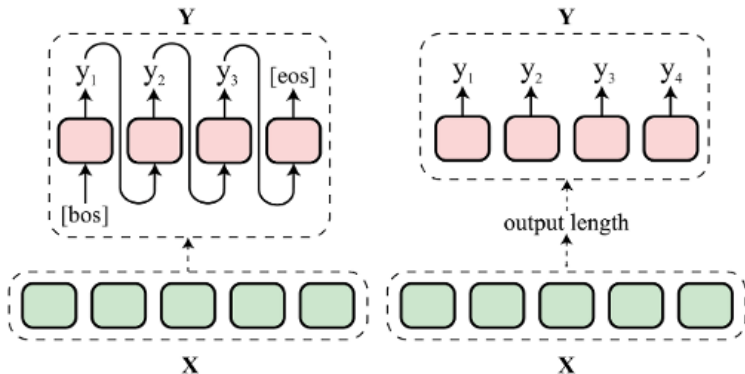
$$P(\text{"I eat the"}) = P(\text{"the"} | \text{"I eat"}) P(\text{"I eat"})$$

$$P(\text{"I eat the apple"}) = P(\text{"apple"} | \text{"I eat the"}) P(\text{"I eat the"})$$

We can complete the text by sampling next tokens iteratively.

# Auto-regressive Sampling

Auto-regressive sampling means we sequentially sample elements conditioning on previously generated elements.





# Mathematical Formulation

---

It can be expressed using chain rule of probability.

$$p(Y) = \prod_{i=1}^N p(y_i | y_{1:i-1}) \quad \text{where } Y = (y_1, \dots, y_N)$$

We sample a sequence of tokens by iteratively sampling next token.

$$v_1 \sim P(y_1)$$

$$v_2 \sim P(y_2 | y_1 = v_1)$$

...

$$v_N \sim P(y_N | y_1 = v_1, \dots, y_{N-1} = v_{N-1})$$

Sampling sequence is  $T = (v_1, \dots, v_N)$ .

# Token Sampling

---

We use several token sampling strategies to increase the likelihood of generated sample.

Normal way to sample sequence is to sample from the predicted probability distribution as is.

$$v \sim P(y_i | y_{1:i-1})$$

# Greedy Sampling

---

One approach is to use the most probable token.

$$v = \arg \max_j P(y_i = v_j | y_{1:i-1})$$

Suppose our vocab consists of 4 words:

$$\mathcal{V} = \{\text{apple, banana, orange, plum}\}$$

We have primed our LM with “apple apple” and want to generate the next word in the sequence. Our language model predicts:

$$P(y_3 = \text{apple} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.05$$

$$P(y_3 = \text{banana} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.65$$

$$P(y_3 = \text{orange} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.2$$

$$P(y_3 = \text{plum} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.1$$

If we sample with argmax, what word would get selected?

(a) apple (b) banana (c) orange (d) plum

# Random Sampling

---

Suppose our vocab consists of 4 words:

$$\mathcal{V} = \{\text{apple, banana, orange, plum}\}$$

We have primed our LM with “apple apple” and want to generate the next word in the sequence. Our language model predicts:

$$P(y_3 = \text{apple} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.05$$

$$P(y_3 = \text{banana} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.65$$

$$P(y_3 = \text{orange} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.2$$

$$P(y_3 = \text{plum} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.1$$

With random sampling, what is the probability we'll pick "banana"?

(a) 0% (b) 5% (c) 65% (d) 100%

# Caveat on Random Sampling

---

## Likelihood of sampling low-probability token

Most tokens in the vocabulary get assigned very low probabilities but cumulatively, **choosing any one of these low-probability tokens is pretty likely**. There is over a 26% chance of choosing a token  $v$  with  $P(y_t = v) \leq 0.01$  within a sequence with length 30.

$$1 - 0.99^{30} = 0.2602$$

Modify the distribution returned by the model to make the tokens in the tail less likely.

# Temperature Sampling

---

We slightly tweak the score before softmax operation to make our distribution sharp or flat.

$$P(y_i = v_j | y_{1:i-1}) = \frac{\exp(\hat{y}_i[j]/K)}{\sum_{x=1}^{\|\mathcal{V}\|} \exp(\hat{y}_i[x]/K)}$$

Calculate on your own.

# Infinite Sampling Temperature

---

Suppose our vocab consists of 4 words:

$$\mathcal{V} = \{\text{apple, banana, orange, plum}\}$$

We have primed our LM with “apple apple” and want to generate the next word in the sequence. Our language model predicts:

$$P(y_3 = \text{apple} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.05$$

$$P(y_3 = \text{banana} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.65$$

$$P(y_3 = \text{orange} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.2$$

$$P(y_3 = \text{plum} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.1$$

What would the probability of selecting “banana” be if we use temperature sampling and set  $T = \infty$ ?

(a) 0% (b) 25% (c) 65% (d) 100%

# Extremely Small Sampling Temperature

---

Suppose our vocab consists of 4 words:

$$\mathcal{V} = \{\text{apple, banana, orange, plum}\}$$

We have primed our LM with “apple apple” and want to generate the next word in the sequence. Our language model predicts:

$$P(y_3 = \text{apple} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.05$$

$$P(y_3 = \text{banana} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.65$$

$$P(y_3 = \text{orange} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.2$$

$$P(y_3 = \text{plum} | y_1 = \text{apple}, y_2 = \text{apple}) = 0.1$$

What would the probability of selecting “banana” be if we use temperature sampling and set  $T = 0.00001$ ?

(a) 0% (b) 25% (c) 65% (d) 100%



# Top-k Sampling & Nucleus Sampling

---

Just sample token from  $k$  most likely tokens.

Usually  $k$  between 10 and 50 is selected.

Or, just sample token from the tokens which form  $p\%$  of the probability mass.

Usually  $p$  between 0.9 and 1.0 is selected.

# Sampling Strategy Tradeoff

---

**Low Sampling Temperature,  
Low Top-k Sampling,  
Low Nucleus Sampling**

**High Sampling Temperature,  
High Top-k Sampling,  
High Nucleus Sampling**

# Other Parameters for Sampling

---

- Frequency penalty: Reduce the likelihood the model generates a token based on how often it has occurred already.
  - The more likely a token has occurred, the less likely it will be to occur in the future.
- Presence penalty: Reduce the likelihood the model generates a token based on whether or not it has occurred already.
  - If a token occurs any number of times, it will be less likely to occur in the future.
- Stopping criteria
  - Stop after generating  $k$  tokens.
  - Stop when a certain token is generated (for example, a period or a newline).

# References

---

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

# **Any Question?**