AIE1902: Exploring Language Models with AI

Xiaopeng Li
CUHK(SZ)
SAI

January 5, 2026

Outline

Introduction to Myself

My name: Xiaopeng Li

Position: Assistant Professor, School of Artificial Intelligence, CUHK(SZ)

Website: `https://williampixell.github.io`

Education background:

- Ph.D. in Operations Research, Columbia University.
- B.S. in Applied Mathematics, The Chinese University of Hong Kong, Shenzhen.

Research focus: nonconvex optimization theory and algorithms with applications in machine learning; large language model reasoning.

How to address me: You can call me 'Xiaopeng' or 'Dr. Li'.

Contact and Office Hours

Instructor: Xiaopeng Li
Email: xiaopengli1@cuhk.edu.cn
Lecture: 3:30pm–5:20pm, Monday, TA313
Office hour: 3pm–4pm, Wednesday, TA409c

Teaching Assistant: Jiarong Lian
Email: 225080010@link.cuhk.edu.cn
Office hour: 10am–11am, Thursday, TA414
Role: The TA will grade assignments and projects, host office hours, delivers lab sessions, and provide support on questions related to course content.

Communication:
Please email with a clear subject line and include AIE1902 in the subject. We aim to respond within 48 hours on weekdays.

Selected Learning Objectives

By the end of this course, you will be able to:

1. Understand core principles: the main ideas behind statistical and neural language models.

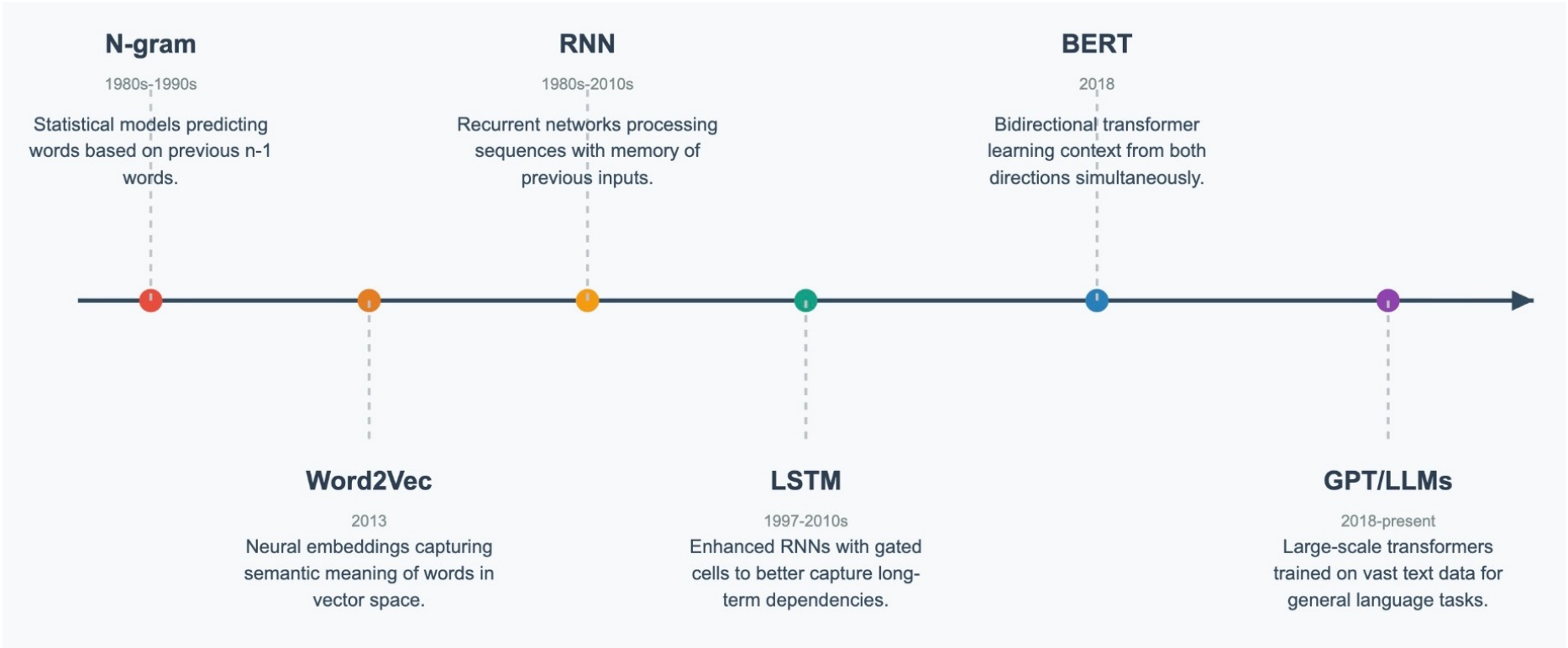
2. Explore the development workflow: the tools and infrastructure used to implement, train, and evaluate language models.

3. Build language representations: use embeddings and sequence modeling to support language understanding and generation.

4. Apply pretrained models in practice: use prompting and efficient adaptation to solve downstream tasks.

5. Deliver an end-to-end system: collaborate in teams to design and present a language-model agent with a complete pipeline.

Course Organization



Course Organization

This course combines basic concepts with hands-on practice for building language models and LLM-based agents.

- **Lectures:** From n-gram language models to embeddings, RNN/LSTM, transformers, and pretrained LMs (BERT/GPT).
- **Labs & Assignments:** GPU-enabled implementation, training, and evaluation of models; decoding control and efficient adaptation.
- **Project:** Teams (4–5 people) design an end-to-end language-model agent pipeline (dataset, embeddings, retrieval, fine-tuning, inference).
- **Checkpoints:** Mid-term progress presentation (Week 7), and final showcase (Week 12).

Readings

Essential Materials: Lecture slides & lab notebooks

Optional Readings:

- NLP Modules for High School (Greg Durrett) https://www.ifml.institute/node/368
- Practical tooling (datasets, training, fine-tuning): Hugging Face NLP Course: https://huggingface.co/learn/nlp-course/
- CS224N: Natural Language Processing with Deep Learning (Stanford) https://web.stanford.edu/class/cs224n/
- *Speech and Language Processing* (Jurafsky & Martin, draft): https://web.stanford.edu/~jurafsky/slp3/

Assessment Scheme

Component	Weight
Assignment	40%
Project midterm presentation	30%
Project final presentation	30%
Total	100%

Notes:

- Assignments emphasize on understanding basic concepts of language models and hands-on implementation.
- The project emphasizes on end-to-end system building: dataset, embeddings, retrieval, fine-tuning, and inference.
- You are welcome to finish your work with the help of LLM (e.g., ChatGPT, DeepSeek), but you should understand the output.

Assignments

Individual Assignments: Four individual assignments will be released during the semester to reinforce the lecture and lab materials. They are to be completed independently.

Late Policy: An assignment submitted n hours late will have its score reduced by n percent, unless an extension is approved by the instructor before the deadline. If you anticipate conflicts, communicate early.

Academic Integrity: You may discuss high-level ideas with classmates, but all submitted solutions (write-ups and codes) must be your own work. Copying codes or write-ups without proper attribution is not permitted. Suspected violations will be handled according to university policies and may result in a zero grade and further disciplinary action.

Project

Team size: Teams should have 4–5 members (formed in Week 1–2).

Team lead: Each team should designate a team leader.

Regular check-ins: Each team is expected to meet the instructor/TA at key milestones to discuss the progress and plan.

Clear division of labor: Tasks should be allocated explicitly properly, and every member's responsibilities should be transparent.

Work log: Maintain a work log that records individual contributions to support fair assessment.

Late policy: Same as assignment.

Project Deliverables

Scope: Build an end-to-end language-model agent with a measurable objective. Your project should define a clear task, implement at least one baseline with real data set, and include a non-trivial technical component from this course.

Midterm: State the task and motivation, describe the dataset, present at least one working baseline, and provide a concrete plan for what will be completed by the final presentation.

Final: Demonstrate a working end-to-end system, including a complete pipeline (data $\rightarrow$ modeling $\rightarrow$ inference), performance measurement with appropriate metrics and baselines, and a short demo that showcases the agent's capability.

Support: It's impossible to finish this project by yourself. You have to collaborate with your teammates and use AI tools and instructor/TA to help you finish the task.

Outline

Roadmap

What is a language model (LM) and why we care?

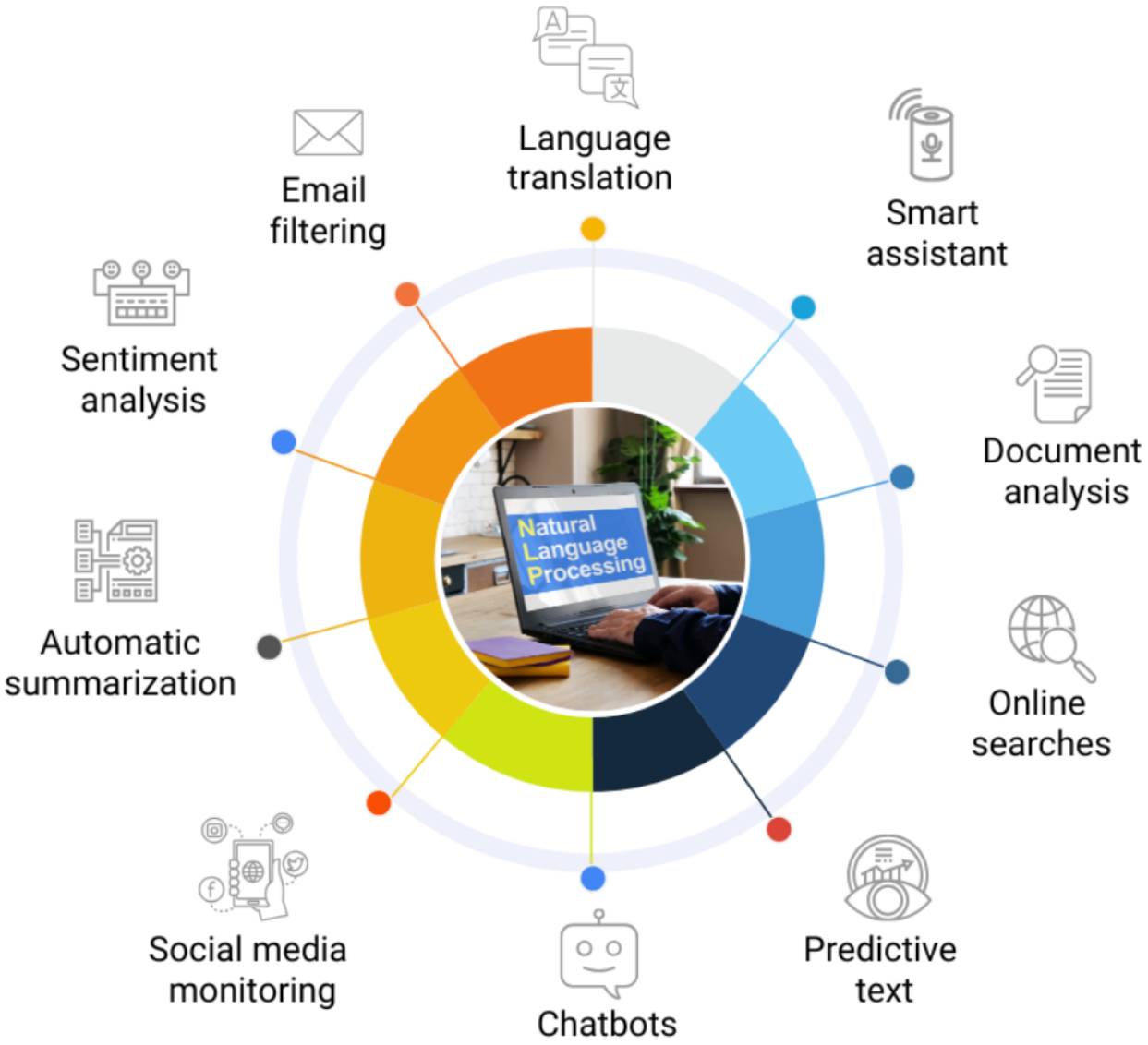
How to define probabilities over sentences?

N-gram models: unigram/bigram/trigram and the Markov assumption

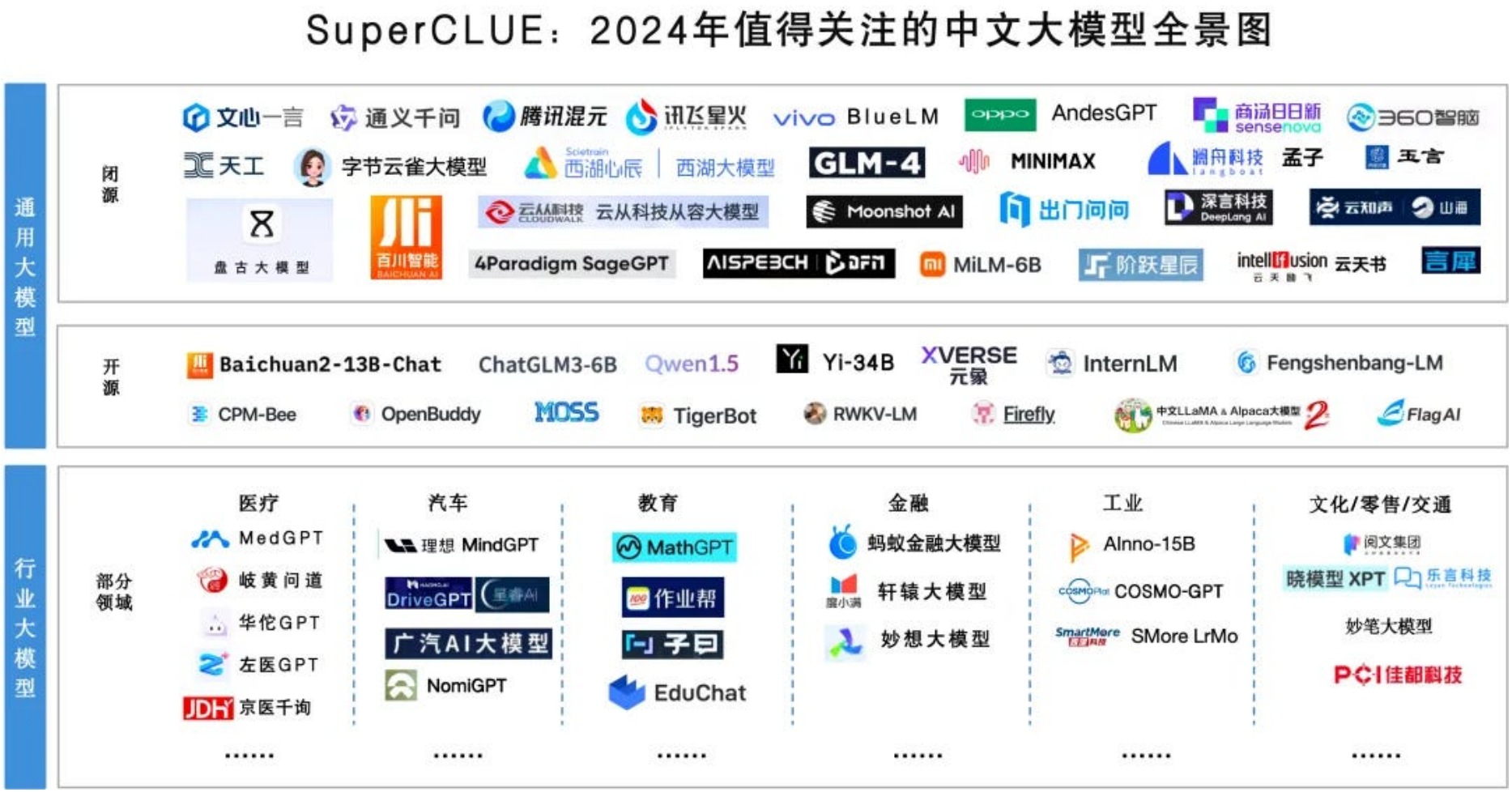
Training and evaluation: MLE counts and perplexity

Practical issues: sparsity, zeros, smoothing, interpolation/backoff

Language Model in Our Life



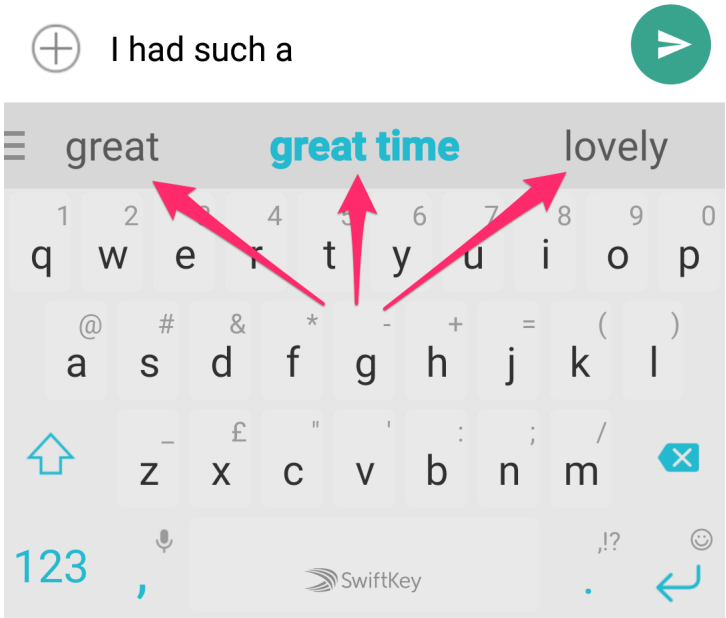
Language Model in Our Life



Previous Language Model

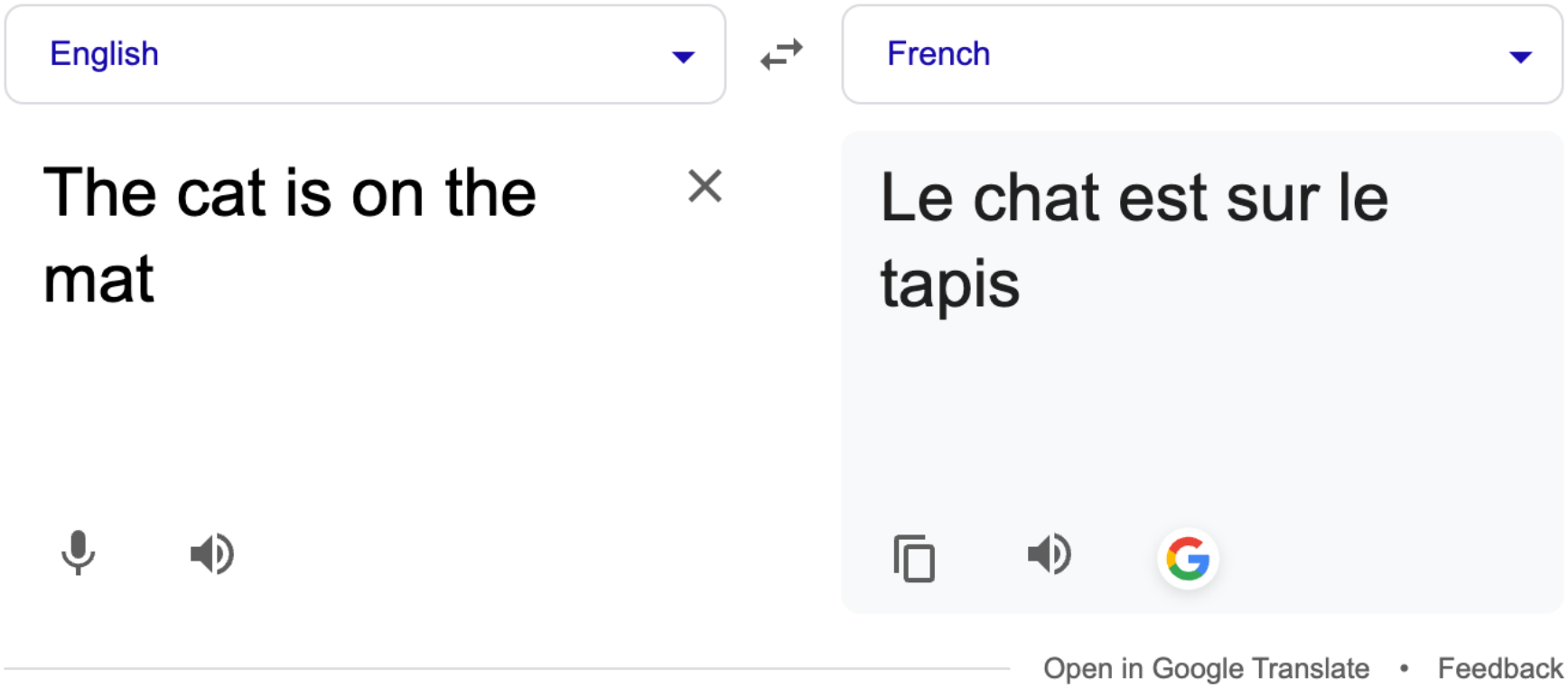
15 years ago, you are probably using this kind of language model:

Predictive text; tap a suggestion to apply.



Previous Language Model

7 years ago, you are probably using this kind of language model:



Recent Language Model

Now, you can do this:

What is the exact value of integral of sin(x^2) over (0,+inf) ?

Thought for 25 seconds ›

The integral $\int_0^\infty \sin(x^2)\,dx$ is a Fresnel integral. Its exact value can be derived using complex analysis or properties of the Gamma function.

Consider the Gaussian integral with a complex parameter:

$$\int_0^\infty e^{-ix^2}\,dx = \frac{1}{2}\sqrt{\frac{\pi}{i}} = \frac{1}{2}\sqrt{\pi}e^{-i\pi/4} = \frac{1}{2}\sqrt{\pi}\left(\cos\frac{\pi}{4} - i\sin\frac{\pi}{4}\right) = \frac{1}{2}\sqrt{\pi}\left(\frac{\sqrt{2}}{2} - i\frac{\sqrt{2}}{2}\right) = \sqrt{\frac{\pi}{8}}$$

Expanding the left-hand side:

$$\int_0^\infty e^{-ix^2}\,dx = \int_0^\infty \cos(x^2)\,dx - i\int_0^\infty \sin(x^2)\,dx.$$

Equating real and imaginary parts:

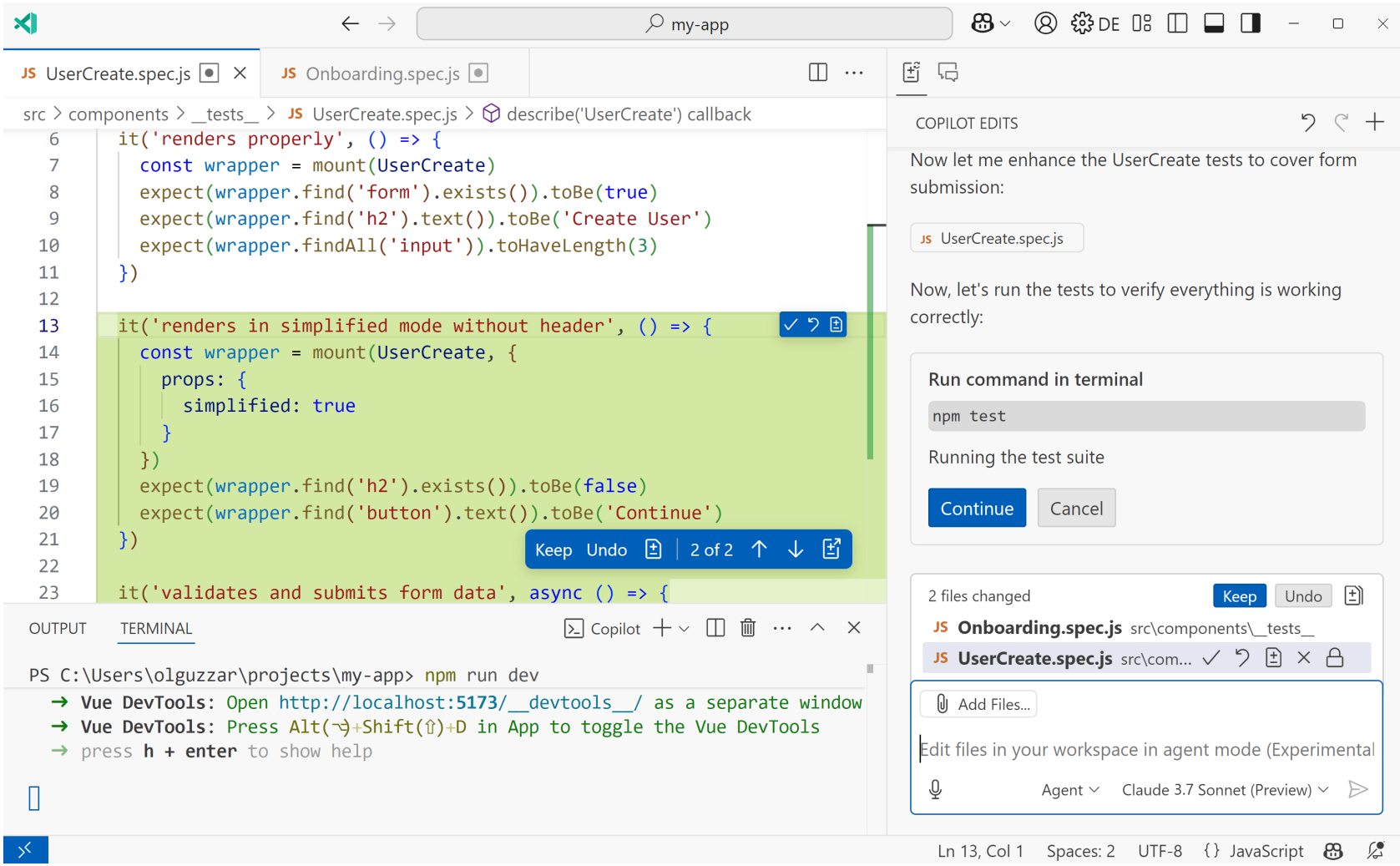
$$\int_0^\infty \cos(x^2)\,dx = \sqrt{\frac{\pi}{8}}, \quad \int_0^\infty \sin(x^2)\,dx = \sqrt{\frac{\pi}{8}}.$$

Thus, the exact value is $\sqrt{\frac{\pi}{8}}$, which can also be written as $\frac{1}{2}\sqrt{\frac{\pi}{2}}$ or $\frac{\sqrt{2\pi}}{4}$.

$$\boxed{\sqrt{\frac{\pi}{8}}}$$

Recent Language Model

Now, you can do this:



Motivating Questions

From single-purpose to general-purpose:
How did we move from task-specific language model (e.g., mobile auto-completion) to general assistants like ChatGPT or DeepSeek?

The core question:
What is the key core technology behind ChatGPT or DeepSeek?

Course answer:
Perhaps unexpectedly, we will use modern language models (ChatGPT, DeepSeek, etc.) to help us learn language models.

Outline

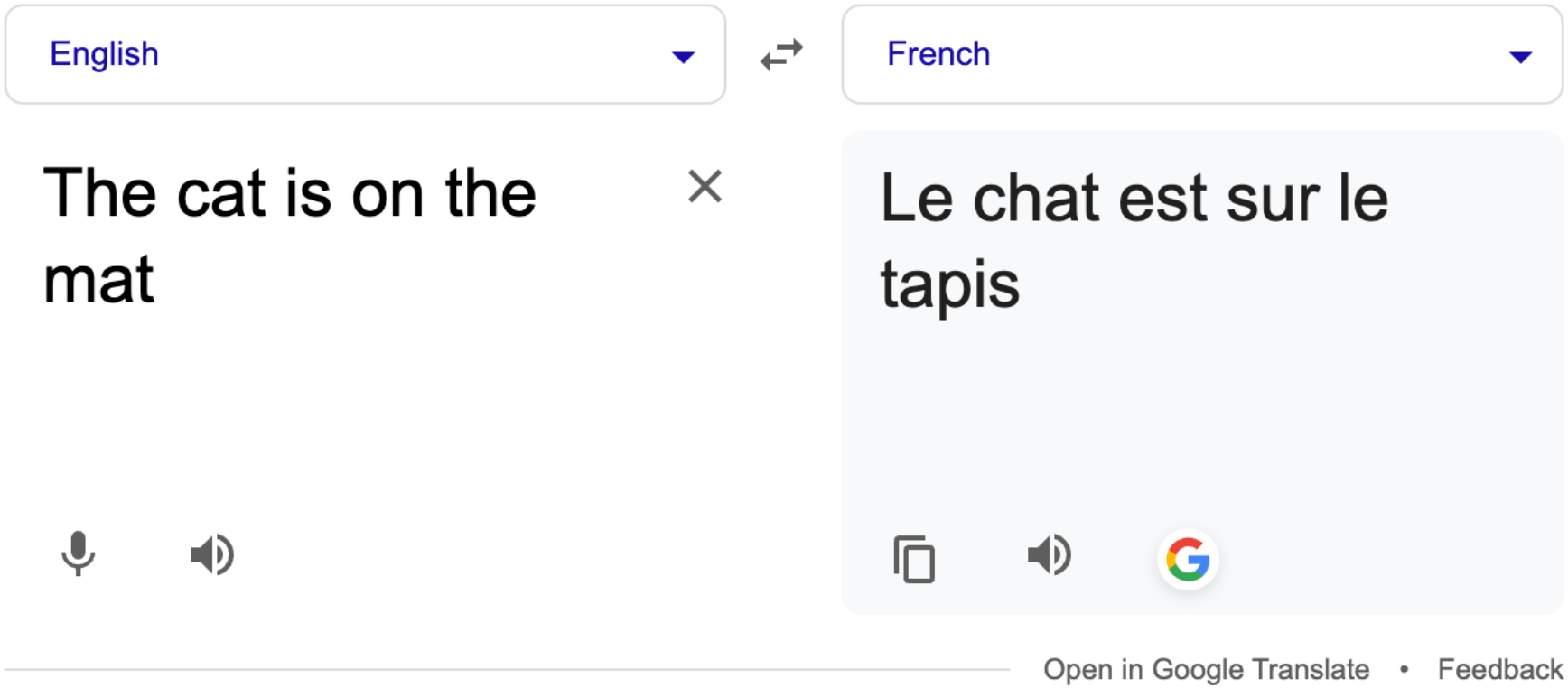
Recall Your Python Course

Write a function to uppercase every other letter in a string, starting with the second one.

```
input = "The cat is on the mat"
i = 0
result = ""
for letter in input:
  if i % 2 == 1:
    result += letter.upper()
  else:
    result += letter
  i += 1
```

$\Longrightarrow$ `"ThE CaT Is oN ThE MaT"`

How to Write Programs?



Have A Try

Translate "The cat is on the mat." with hand-written rules.

```
words = ["The","cat","is","on","the","mat","."]
result = []
for w in words:
   if w == "cat":    result.append("chat")
   elif w == "mat":   result.append("tapis")
   elif w == "on":    result.append("sur")
   elif w == "is":    result.append("est")
   elif w == "the":   result.append("le")
   else:          result.append(w)
```

$\Longrightarrow$ `le chat est sur le tapis.`

Question: Is this a good solution? What if we change the sentence?

Context

```
if w == "the" and next == "cat":  result.append("le")
if w == "the" and next == "table":
result.append("la")
if w == "the" and next is plural:
result.append("les")
```

the cat → le chat

the table → la table

the cats → les chats

The cat is on the mat.

The cats are on the mats. ?

The cat was under the table. ?

Takeaway: A fixed rule list cannot scale; we need a model that learns patterns from data.

Learning from Examples

We look at correct examples (data):

The cat is on the mat.
Le chat est sur le tapis.

The cats are on the mats.
Les chats sont sur les tapis.

The cat was under the table.
Le chat était sous la table.

Idea: By seeing ground truth data for how to translate, a machine can learn the process.

Training

Predict: the model outputs a translation.

Check: compare with the correct answer.

Update: adjust the model to reduce mistakes.

`The cat is on the mat.`

`Le chat est sur la tapis.` **(wrong)**

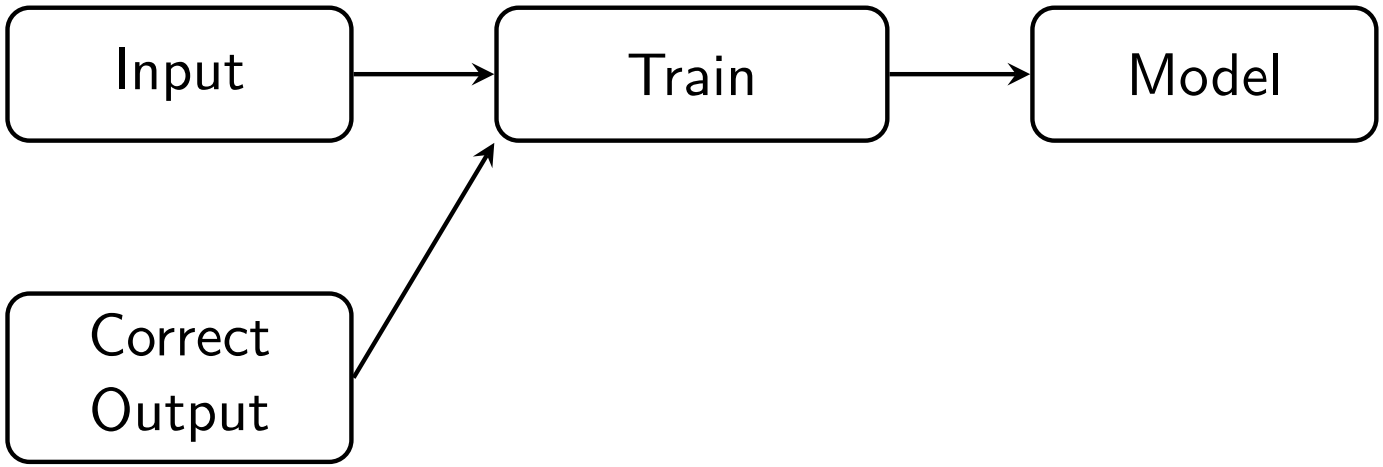
`Le chat est sur le tapis.` **(correct)**

Key point: Feedback replaces manual rule-writing.

Programming vs. Machine Learning

Programming

Machine Learning



From Translation to Language Model

Translation is hard to code by rules: even simple sentences require context (gender, number, and tense).

Machine learning changes the workflow: instead of writing rules, we learn from many examples of correct language behavior.

What should the model learn? A simple, powerful goal is predictive text: given the words so far (the context), predict what comes next.

Language model: a system that learns patterns of language so it can make good next-word predictions.

A Concrete Example

Task: Predict the next word in a sentence.

```
input = "I want to drink a cup of"
Model prediction (probabilities):
                      coffee  0.42
                         tea  0.31
                       water  0.08
                        milk  0.06
                           ⋮
output = "I want to drink a cup of coffee."
```

Key idea: A language model is a machine learning model that assigns probabilities to text.

Vocabulary

Language is made of words.

A corpus is a large collection of text (a set of sentences), such as news articles, books, or webpages.

Let $\mathcal{V}$ be the vocabulary: the set of all words that appear in the corpus.

- $\mathcal{V}$ can be large (thousands or more),
- but we assume it is finite.

A sentence is a sequence of words:

$$x_1, x_2, \ldots, x_{T-1}, \qquad x_t \in \mathcal{V}.$$

Language Model

We introduce two special symbols:

- `START`: marks the beginning of a sentence,
- `STOP`: marks the end of a sentence.

These symbols are not in the vocabulary $\mathcal{V}$.

With padding, a sentence looks like:

$$(\texttt{START}, x_1, x_2, \ldots, x_{T-1}, \texttt{STOP}).$$

Example sentences:

```
START the dog barks STOP
START the cat saw the dog STOP
```

Probability as a Score

```
START the dog barks STOP
START the STOP
START START cat cat cat STOP
```

All of these are valid sequences of tokens. But some sound much more natural than others.

Goal: we want a model that assigns higher scores to more natural sentences.

- Given a vocabulary $\mathcal{V}$, there are infinitely many possible sentences.
- A language model assigns each sentence a probability

$$p(x_0, x_1, \ldots, x_T), \quad \text{s.t.} \quad \sum_{x \in \mathcal{S}} p(x) = 1.$$

- The above defines a Language Model.

Probability as the Score

A naive example:

- Assume we have a training corpus of sentences.
- For each sentence x, define

$$p(x) = \frac{\text{count}(x)}{|\mathcal{S}|},$$

where $\text{count}(x)$ is the number of times sentence x occurs in the corpus $\mathcal{S}$.

Obviously, many sentences will have probability 0, but that doesn't mean we shall never predict those sentences. Thus, this is a bad model.

Question: How can we do better? How can we say one language model is better than another?

Outline

Markov Language Models

Our goal is to model the probability of a sentence

$$p(x_0, x_1, \ldots, x_T).$$

By the probability chain rule:

$$\begin{aligned} p(x_0, x_1, \ldots, x_T) &= p(x_0)p(x_1, \ldots, x_T \mid x_0) \\ &= p(x_0)p(x_1 \mid x_0)p(x_2, \ldots, x_T \mid x_0, x_1) \\ &= \cdots \\ &= p(x_0) \prod_{t=1}^{T} p(x_t \mid x_{t-1}, \ldots, x_0). \end{aligned}$$

This decomposition simplifies the original probability, but is still difficult: conditioning on the entire past is expensive.

Markov Language Models

Assumption: the future depends only on a finite window of the past.

Instead of

$$p(x_t \mid x_{t-1}, \ldots, x_0),$$

we approximate it by conditioning on recent words only.

This gives n-gram models:

- **Unigram:** $p(x_t)$
- **Bigram:** $p(x_t \mid x_{t-1})$
- **Trigram:** $p(x_t \mid x_{t-1}, x_{t-2})$

This simplifying assumption is called the Markov Assumption.

START Tokens

Using the chain rule, the sentence probability includes $p(x_0)$:

$$p(x_0, x_1, \ldots, x_T) = p(x_0) \prod_{t=1}^{T} p(x_t \mid x_{t-1}, \ldots, x_0).$$

We handle this by introducing a special START symbol:

$$x_0 = \texttt{START}.$$

Since all sentences begin with the same padding, $p(x_0)$ is a constant across sentences.

Therefore, when comparing or scoring sentences, we can safely drop this constant and renormalize if needed.

Example: Trigram Model

Assumption: the next token depends on the previous two tokens.

Parameters: $\theta(w \mid u, v)$, interpreted as

$$\theta(w \mid u, v) = \mathbb{P}(x_t = w \mid x_{t-2} = u,\ x_{t-1} = v).$$

Token sets: $w \in \mathcal{V} \cup \{\texttt{STOP}\}$ and $u, v \in \mathcal{V} \cup \{\texttt{START}\}$.

Sentence probability (with padding): let $x_{-1} = x_0 = \texttt{START}$ and $x_T = \texttt{STOP}$, then

$$p(x) = \prod_{t=1}^{T} \theta(x_t \mid x_{t-2}, x_{t-1}).$$

Learning the Parameters from Data

Data: a corpus of sentences $\{x^{(i)}\}_{i=1}^{N}$, each with lengths T_i.

Training principle: choose θ that makes the corpus likely.

- Objective (minimizing average negative log-likelihood):

$$\mathcal{L}(\theta) = -\frac{\sum_{i=1}^{N} \log p(x^{(i)})}{N} = -\frac{\sum_{i=1}^{N} \sum_{t=1}^{T_i} \log \theta(x_t^{(i)} \mid x_{t-2}^{(i)}, x_{t-1}^{(i)})}{N}.$$

- Constraints (probabilities must be valid):

$$\forall (u, v): \quad \theta(w \mid u, v) \geq 0, \qquad \sum_{w} \theta(w \mid u, v) = 1.$$

You don't need to know how to solve it rigorously because it requires method of Lagrange multipliers, which will be introduced in MAT1002/1012 or MAT3007 later.

Best Parameters: Relative Frequency

Best Solution:

$$\theta^\star(w \mid u, v) = \frac{\text{count}(u, v, w)}{\text{count}(u, v)}.$$

- count(u, v, w) is $\#$ of times we see (u, v, w) in the training corpus;
- count (u, v) is $\#$ of times we see (u, v) in the training corpus.

Intuition: given the context (u, v), the best estimate of "how likely w comes next" is the fraction of times we saw w after (u, v) in the data. The denominator makes probabilities sum to 1.

Example: if count$(\texttt{the}, \texttt{cat}) = 50$ and count$(\texttt{the}, \texttt{cat}, \texttt{saw}) = 5$, then

$$\theta(\texttt{saw} \mid \texttt{the}, \texttt{cat}) = 5/50 = 0.10.$$

What if count$(u, v, w) = 0$ or count$(u, v) = 0$?

Outline

Perplexity

A good language model assigns high probability to natural sentences, and low probability to unnatural ones.

When the model sees the true next word, it is either not surprised (high probability) or surprised (low probability).

Perplexity summarizes this surprise over a test set.

Intuition: Perplexity $\approx$ the model's effective number of choices per word.

- Perplexity $= 1$: the model is almost never surprised.
- Perplexity $= 50$: it feels like choosing among ~ 50 words each step.

Perplexity: Definition

Test set: sentences $\{x^{(i)}\}_{i=1}^{N}$ with total number of predicted tokens T.

Average negative log-likelihood (per token):

$$\mathrm{NLL} \;=\; -\frac{1}{T}\sum_{i=1}^{N}\log p\Big(x^{(i)}\Big)\,.$$

Perplexity:

$$\mathrm{PPL} \;=\; 2^{\mathrm{NLL}}.$$

Important: if the model assigns probability 0 to any test sentence, then $\log p(x) = -\infty$ and PPL becomes infinite.

A Toy Example

Training corpus:

START I like to eat STOP
START I want to drink STOP

Test sentence:

START I like to eat STOP

Key estimate probabilities:

$$p_{\text{bi}}(\texttt{like} \mid \texttt{I}) = \tfrac{1}{2}, \qquad p_{\text{bi}}(\texttt{eat} \mid \texttt{to}) = \tfrac{1}{2}$$
$$p_{\text{tri}}(\texttt{like} \mid \texttt{START}, \texttt{I}) = \tfrac{1}{2}, \qquad p_{\text{tri}}(\texttt{eat} \mid \texttt{like}, \texttt{to}) = 1$$

Sentence probability:

$$p_{\text{bi}}(x) = \tfrac{1}{2} \cdot \tfrac{1}{2} = \tfrac{1}{4}, \qquad p_{\text{tri}}(x) = \tfrac{1}{2} \cdot 1 = \tfrac{1}{2}.$$

Perplexity: $\mathrm{PPL}(x) = p(x)^{-1/T}$ with $T = 5$.

$$\mathrm{PPL}_{\text{bi}} = \left(\tfrac{1}{4}\right)^{-1/5} = 4^{1/5} \approx 1.32, \qquad \mathrm{PPL}_{\text{tri}} = \left(\tfrac{1}{2}\right)^{-1/5} = 2^{1/5} \approx 1.15.$$

Improvement: Interpolation

Problem: Higher-order n-grams use more context, and with more contexts rare or unseen $\implies$ zero probability & infinite perplexity.

Idea: Combine trigram, bigram and unigram so we use context when we have it and fall back when we don't.

$$\theta_{\text{mix}}(w \mid u, v) = \lambda_3 \theta_3^*(w \mid u, v) + \lambda_2 \theta_2^*(w \mid v) + \lambda_1 \theta_1^*(w)$$
$$\lambda_i \geq 0, \quad \lambda_1 + \lambda_2 + \lambda_3 = 1.$$

Notice:

- If the context (u, v) is frequent, we want λ_3 larger.
- If (u, v) is rare, rely more on θ_2^* or θ_1^*.
- In practice: use heuristic $(0.6, 0.35, 0.05)$ as a quick baseline, or pick λ on a validation set to minimize perplexity.

Improvement: Add-k Smoothing

Problem: Higher-order n-grams use more context, and with more contexts rare or unseen $\implies$ zero probability & infinite perplexity.

Idea: Add a small pseudo-count to every possible next word:

$$\theta_k(w \mid u, v) = \frac{\text{count}(u, v, w) + k}{\text{count}(u, v) + k \cdot (|\mathcal{V}| + 1)}, \quad w \in \mathcal{V} \cup \{\texttt{STOP}\},\ k > 0.$$

Notice:

- If $\text{count}(u, v, w) = 0$, then $\theta_k(w \mid u, v) > 0$.
- Intuition: "pretend we saw each possible next token k extra times".
- Trade-off: bigger k $\implies$ safer but more uniform (less context-sensitive).

Next Week

Thank you for your attendance!

Questions?

Next week (Lab session):

- Environment setup: create a `conda` env; install required packages.
- Data pipeline: load a corpus; preprocess; split train/validation/test.
- Modeling: implement n-gram; next-word top-k prediction.
- Evaluation: compute perplexity and Hit@k on validation/test.
- Improvement: add-k smoothing and interpolation; compare results.