5. Language Models and Recurrent Neural Networks

LING-581-Natural Language Processing 1

Instructor: Hakyung Sung

September 23, 2025

*Acknowledgment: These course slides are based on materials from CS224N @ Stanford University; Dr. Kilho Shin @ Kyocera

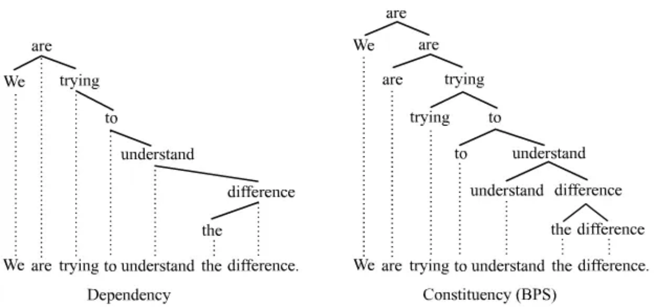
Table of contents

Review

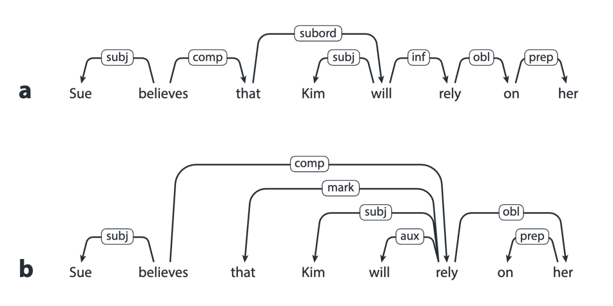
Review

- Syntactic structure: Consistency and dependency
- Dependency grammar and treebanks
- Dependency parsing
- Transition-based dependency parsing
- Neural dependency parsing

Review: Dependency grammar vs. Constituency parsing

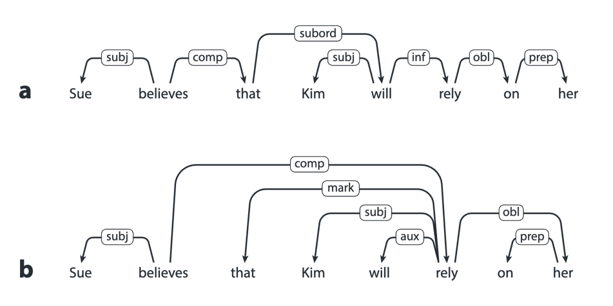


Review: Universal dependency grammar



Sourced from De Marneffe, M. C., & Nivre, J. (2019). Dependency grammar. Annual Review of Linguistics, 5(1), 197-218. Figure 4

Review: Universal dependency grammar



Sourced from De Marneffe, M. C., & Nivre, J. (2019). Dependency grammar. Annual Review of Linguistics, 5(1), 197-218. Figure 4

“UD gives priority to dependency relations between **content words**, while function words are attached to the content word.”

Review: Universal dependency grammar

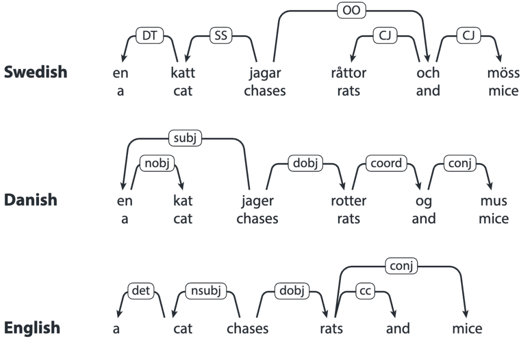


Figure 6

Dependency trees for parallel sentences in Swedish, Danish, and English.

Sourced from De Marneffe, M. C., & Nivre, J. (2019). Dependency grammar. Annual Review of Linguistics, 5(1), 197-218. Figure 6

Review: Universal dependency grammar

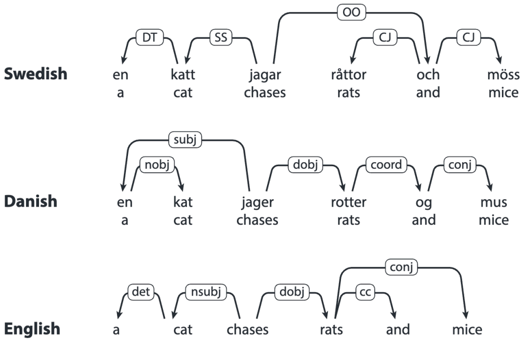


Figure 6

Dependency trees for parallel sentences in Swedish, Danish, and English.

Sourced from De Marneffe, M. C., & Nivre, J. (2019). Dependency grammar. Annual Review of Linguistics, 5(1), 197-218. Figure 6

"The goal is to support **multilingual research** in NLP and linguistics by enabling sound comparative evaluation across languages."

Review: Terminology

- treebank
- UAS vs. LAS

Greedy transition-based parsing: Example

Sentence: *I saw him*

Initial State: Stack = [ROOT], Buffer = [I, saw, him], Arcs = {}

Step	Stack	Buffer	Transition	New Arc
1	[ROOT]	[I, saw, him]	SHIFT	—
2	[ROOT, I]	[saw, him]	SHIFT	—
3	[ROOT, I, saw]	[him]	LEFT-ARC	saw $\rightarrow$ I (subj)
4	[ROOT, saw]	[him]	SHIFT	—
5	[ROOT, saw, him]	[ ]	RIGHT-ARC	saw $\rightarrow$ him (obj)
6	[ROOT, saw]	[ ]	RIGHT-ARC	ROOT $\rightarrow$ saw (root)

Choosing the next parsing action

How should we decide the next parsing action?

- Parsing hoice depends on the parsing algorithm:

Choosing the next parsing action

How should we decide the next parsing action?

- Parsing hoice depends on the parsing algorithm:
  - Greedy methods (Nivre, 2003)

Choosing the next parsing action

How should we decide the next parsing action?

- Parsing hoice depends on the parsing algorithm:
    - Greedy methods (Nivre, 2003)
    - Beam search (Nivre & Hall, 2005)

Choosing the next parsing action

How should we decide the next parsing action?

- Parsing hoice depends on the parsing algorithm:
    - Greedy methods (Nivre, 2003)
    - Beam search (Nivre & Hall, 2005)
    - Neural approaches (Chen & Manning, 2014)

Choosing the next parsing action

How should we decide the next parsing action?

- Parsing hoice depends on the parsing algorithm:
  - Greedy methods (Nivre, 2003)
  - Beam search (Nivre & Hall, 2005)
  - Neural approaches (Chen & Manning, 2014)
  - Graph-based Biaffine approaches (Dozat & Manning, 2017)

Choosing the next parsing action

How should we decide the next parsing action?

- Parsing hoice depends on the parsing algorithm:
    - Greedy methods (Nivre, 2003)
    - Beam search (Nivre & Hall, 2005)
    - Neural approaches (Chen & Manning, 2014)
    - Graph-based Biaffine approaches (Dozat & Manning, 2017)
    - Current SOTA: Pre-trained transformers + graph-based biaffine decoders?

Thursday Lab

We'll continue working on building/training a dependency parser; I've updated the dataset.

Lesson plan

Lesson plan

- Language modeling
- n-gram language models
- Window-based neural language models
- RNNs

Language modeling

Language modeling

- **Language modeling** is the task of predicting the next word in a sequence.

Language modeling

- **Language modeling** is the task of predicting the next word in a sequence.
- Example:

Language modeling

- **Language modeling** is the task of predicting the next word in a sequence.
- Example:

  the *students* *opened* *their* ____

  $\{books, \ laptops, \ exams, \ minds\}$

Language modeling

- **Language modeling** is the task of predicting the next word in a sequence.
- Example:

  the students opened their ____
  {*books*, *laptops*, *exams*, *minds*}

- Formally, given a sequence $x_1, x_2, \ldots, x_t$, we estimate

$$P(x_{t+1} \mid x_1, x_2, \ldots, x_t).$$

Language modeling

- **Language modeling** is the task of predicting the next word in a sequence.
- Example:

  the *students* *opened* *their* ____ $\{books, \ laptops, \ exams, \ minds\}$
- Formally, given a sequence $x_1, x_2, \ldots, x_t$, we estimate

$$P(x_{t+1} \mid x_1, x_2, \ldots, x_t).$$

- Here each x_i (and the predicted x_{t+1}) is drawn from a vocabulary

$$\mathcal{V} = \{ w_1, w_2, \ldots, w_{|\mathcal{V}|} \}.$$

  The symbol w_j denotes the j-th word in $\mathcal{V}$.

Language modeling

- A language model can also be viewed as a system that assigns a probability to an entire sequence of tokens.
- For a text $x_1, \dots, x_T$, the joint probability is

$$
\begin{aligned}
&P(x_1, \dots, x_T) \\
&= P(x_1)\, P(x_2 \mid x_1) \cdots P(x_T \mid x_1, \dots, x_{T-1}) \\
&= \prod_{i=1}^{T} P(x_i \mid x_1, \dots, x_{i-1})
\end{aligned}
$$

- This decomposition follows directly from the chain rule of probability.

Example: Sequence probability

- Consider the sentence: *"I like apples"*.

Example: Sequence probability

- Consider the sentence: *“I like apples”*.
- The joint probability is decomposed as:

Example: Sequence probability

- Consider the sentence: *“I like apples”*.
- The joint probability is decomposed as:
  - $P(\text{“I”})$ = probability that “I” starts the sentence

Example: Sequence probability

- Consider the sentence: *"I like apples"*.
- The joint probability is decomposed as:
  - $P($"I"$)$ = probability that "I" starts the sentence
  - $P($"like" | "I"$)$ = probability that "like" follows "I"

Example: Sequence probability

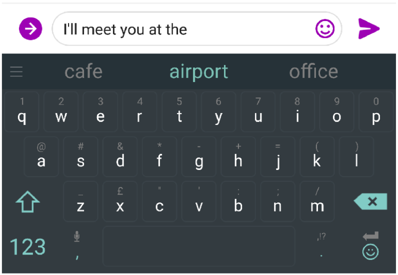
- Consider the sentence: *“I like apples”*.
- The joint probability is decomposed as:
    - $P($“I”$)$ = probability that “I” starts the sentence
    - $P($“like” | “I”$)$ = probability that “like” follows “I”
    - $P($“apples” | “I like”$)$ = probability that “apples” follows “I like”

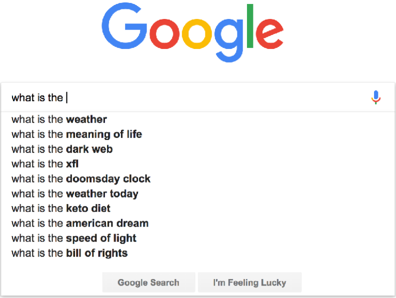
Example: Sequence probability

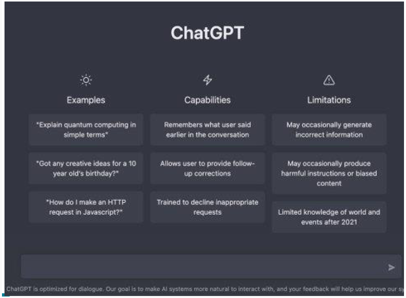
- Consider the sentence: *"I like apples"*.
- The joint probability is decomposed as:
    - $P(\text{"I"})$ = probability that "I" starts the sentence
    - $P(\text{"like"} \mid \text{"I"})$ = probability that "like" follows "I"
    - $P(\text{"apples"} \mid \text{"I like"})$ = probability that "apples" follows "I like"
- Multiplying these gives the overall probability of the sentence:

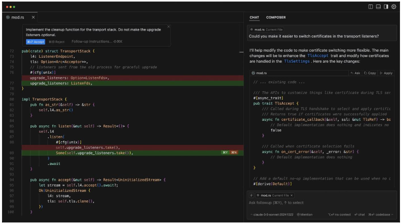
$$P(\text{"I like apples"}) = P(\text{"I"}) \cdot P(\text{"like"} \mid \text{"I"}) \cdot P(\text{"apples"} \mid \text{"I like"})$$

You use language models every day!






ChatGPT
Examples
Capabilities
Limitations
"Explain quantum computing in simple terms"
Remembers what user said earlier in the conversation
May occasionally generate incorrect information
"Got any creative ideas for a 10 year old's birthday?"
Allows user to provide follow-up corrections
May occasionally produce harmful instructions or biased content
"How do I make an HTTP request in Javascript?"
Trained to decline inappropriate requests
Limited knowledge of world and events after 2021
ChatGPT is optimized for dialogue. Our goal is to make AI systems more natural to interact with, and your feedback will help us improve our sy


mod.rs
Implement the cleanup function for the tranport stack. Do not make the upgrade listeners optional.
CHAT
COMPOSER
Could you make it easier to switch certificates in the transport listeners?

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
    - predictive typing

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
  - predictive typing
  - speech recognition

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
  - predictive typing
  - speech recognition
  - handwriting recognition

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
    - predictive typing
    - speech recognition
    - handwriting recognition
    - spelling/grammar correction

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
  - predictive typing
  - speech recognition
  - handwriting recognition
  - spelling/grammar correction
  - authorship identification

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
    - predictive typing
    - speech recognition
    - handwriting recognition
    - spelling/grammar correction
    - authorship identification
    - machine translation

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
  - predictive typing
  - speech recognition
  - handwriting recognition
  - spelling/grammar correction
  - authorship identification
  - machine translation
  - summarization

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
  - predictive typing
  - speech recognition
  - handwriting recognition
  - spelling/grammar correction
  - authorship identification
  - machine translation
  - summarization
  - dialogue

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
    - predictive typing
    - speech recognition
    - handwriting recognition
    - spelling/grammar correction
    - authorship identification
    - machine translation
    - summarization
    - dialogue
    - etc.

Why should we care about language modeling?

- Language modeling is a **benchmark task** that helps us measure our progress on predicting language use.
- Language modeling is a **sub-component of many NLP tasks**, especially those involving generating text or estimating the probability of text:
    - predictive typing
    - speech recognition
    - handwriting recognition
    - spelling/grammar correction
    - authorship identification
    - machine translation
    - summarization
    - dialogue
    - etc.
- Everything else in NLP has been rebuilt upon language modeling: ChatGPT is an LM!

How do we build a language model?

- **Question:** We want to estimate

$$P(x_1, \dots, x_T) = \prod_{i=1}^{T} P(x_i \mid x_1, \dots, x_{i-1})$$

How do we build a language model?

- **Question:** We want to estimate

$$P(x_1, \dots, x_T) = \prod_{i=1}^{T} P(x_i \mid x_1, \dots, x_{i-1})$$

- **First idea:** Approximate by looking only at a few previous words.

How do we build a language model?

- **Question:** We want to estimate

$$P(x_1, \dots, x_T) = \prod_{i=1}^{T} P(x_i \mid x_1, \dots, x_{i-1})$$

- **First idea:** Approximate by looking only at a few previous words.
- This leads us to n-gram language models.

n-gram language models

n-gram language models

- **Definition:** An n-gram is a sequence of *n* consecutive words.

n-gram language models

- **Definition:** An n-gram is a sequence of *n* consecutive words.
- Example: *"the students opened their ______"*

n-gram language models

- **Definition:** An n-gram is a sequence of *n* consecutive words.
- Example: *"the students opened their ______"*
    - Unigrams: the, students, opened, their

n-gram language models

- **Definition:** An n-gram is a sequence of *n* consecutive words.
- Example: *"the students opened their ______"*
  - Unigrams: the, students, opened, their
  - Bigrams: the students, students opened, opened their

n-gram language models

- **Definition:** An n-gram is a sequence of *n* consecutive words.
- Example: *“the students opened their ______”*
  - Unigrams: the, students, opened, their
  - Bigrams: the students, students opened, opened their
  - Trigrams:

n-gram language models

- **Definition:** An n-gram is a sequence of *n* consecutive words.
- Example: *"the students opened their ______"*
    - Unigrams: the, students, opened, their
    - Bigrams: the students, students opened, opened their
    - Trigrams:
    - 4-grams:

n-gram language models

- **Definition:** An n-gram is a sequence of *n* consecutive words.
- Example: *"the students opened their ______"*
    - Unigrams: the, students, opened, their
    - Bigrams: the students, students opened, opened their
    - Trigrams:
    - 4-grams:
- **Idea:** Collect statistics about how often n-grams occur and use them to predict the next word.

n-gram language models: 1. Markov assumption

$$P(x_{t+1} \mid x_t, \ldots, x_1) \;\approx\; P(x_{t+1} \mid x_t, \ldots, x_{t-n+2})$$

- t: position of the current token in the sequence
- n: size of the n-gram (the model looks back $n-1$ tokens)

Only the last $(n-1)$ words matter.

n-gram language models: 1. Markov assumption

$$P(x_{t+1} \mid x_t, \ldots, x_1) \;\approx\; P(x_{t+1} \mid x_t, \ldots, x_{t-n+2})$$

- t: position of the current token in the sequence
- n: size of the n-gram (the model looks back $n-1$ tokens)

Only the last $(n-1)$ words matter.

- We assume that distant history does not strongly influence the next word.

n-gram language models: 1. Markov assumption

$$P(x_{t+1} \mid x_t, \ldots, x_1) \approx P(x_{t+1} \mid x_t, \ldots, x_{t-n+2})$$

- t: position of the current token in the sequence
- n: size of the n-gram (the model looks back $n-1$ tokens)

Only the last $(n-1)$ words matter.

- We assume that distant history does not strongly influence the next word.
- This reduces the problem from "consider the whole history" to "just a short context window."

n-gram language models: 1. Markov assumption

$$P(x_{t+1} \mid x_t, \dots, x_1) \;\approx\; P(x_{t+1} \mid x_t, \dots, x_{t-n+2})$$

- t: position of the current token in the sequence
- n: size of the n-gram (the model looks back $n-1$ tokens)

Only the last $(n-1)$ words matter.

- We assume that distant history does not strongly influence the next word.
- This reduces the problem from "consider the whole history" to "just a short context window."
- **Analogy:** Predicting the next word is like continuing a conversation. (i.e., You don't need to remember everything said 5 minutes ago, just the last few words.)

n-gram language models: 2. conditional probability

- **Definition:** Conditional probability is

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

n-gram language models: 2. conditional probability

- **Definition:** Conditional probability is

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

- Apply this to n-grams:

$$P(x_{t+1} \mid x_t, \dots, x_{t-n+2}) = \frac{P(x_{t+1}, x_t, \dots, x_{t-n+2})}{P(x_t, \dots, x_{t-n+2})}.$$

n-gram language models: 2. conditional probability

- **Definition:** Conditional probability is

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

- Apply this to n-grams:

$$P(x_{t+1} \mid x_t, \ldots, x_{t-n+2}) = \frac{P(x_{t+1}, x_t, \ldots, x_{t-n+2})}{P(x_t, \ldots, x_{t-n+2})}.$$

- e.g., *the* ***cat is*** *cute*

n-gram language models: 3. Example (4-gram)

As the proctor started the clock, the students opened their _____

n-gram language models: 3. Example (4-gram)

As the proctor started the clock, the students opened their _____

~~As the proctor started the clock,the~~ students opened their _____

n-gram language models: 3. Example (4-gram)

As the proctor started the clock, the students opened their _____

~~*As the proctor started the clock,the*~~ *students opened their* _____

Conditioning only on the last three words:

$$\hat{P}(w \mid \text{students open their}) = \frac{\text{count(students open their}w)}{\text{count(students opened their)}}.$$

n-gram language models: 3. Example (4-gram)

As the proctor started the clock, the students opened their _____

~~*As the proctor started the clock,the*~~ *students opened their* _____

Conditioning only on the last three words:

$$\hat{P}(w \mid \text{students open their}) = \frac{\text{count}(\text{students open their}w)}{\text{count}(\text{students opened their})}.$$

Suppose in the corpus:

- ***students opened their*** occurs 1000 times,
- ***students opened their books*** occurs 400 times, so

n-gram language models: 3. Example (4-gram)

As the proctor started the clock, the students opened their _____

~~*As the proctor started the clock,the*~~ *students opened their* _____

Conditioning only on the last three words:

$$\hat{P}(w \mid \text{students open their}) = \frac{\text{count}(\text{students open their} w)}{\text{count}(\text{students opened their})}.$$

Suppose in the corpus:

- `students opened their` occurs 1000 times,
- `students opened their books` occurs 400 times, so

$$P(\text{books} \mid \text{students opened their}) = 0.4,$$

- `students opened their exams` occurs 100 times, so

n-gram language models: 3. Example (4-gram)

As the proctor started the clock, the students opened their _____

~~*As the proctor started the clock,the*~~ *students opened their* _____

Conditioning only on the last three words:

$$\hat{P}(w \mid \text{students open their}) = \frac{\text{count(students open their}w)}{\text{count(students opened their)}}.$$

Suppose in the corpus:

- `students opened their` occurs 1000 times,
- `students opened their books` occurs 400 times, so

$$P(\text{books} \mid \text{students opened their}) = 0.4,$$

- `students opened their exams` occurs 100 times, so

$$P(\text{exams} \mid \text{students opened their}) = 0.1.$$

Problems with n-gram language models

- **Data sparsity:** Zero probability if an n-gram never appears

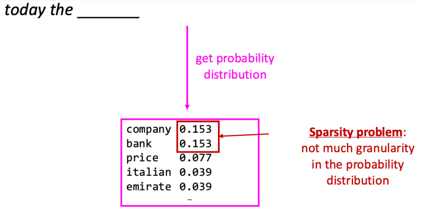
Problems with n-gram language models

- **Data sparsity:** Zero probability if an n-gram never appears
- **Storage cost:** Must store counts for all observed n-grams (e.g., early Google Translator used huge n-gram tables).

Problems with n-gram language models

- **Data sparsity:** Zero probability if an n-gram never appears
- **Storage cost:** Must store counts for all observed n-grams (e.g., early Google Translator used huge n-gram tables).
- **Practical note:** Easy to build (e.g., a trigram LM on a million-word corpus in seconds), but results could be sparse and limited.

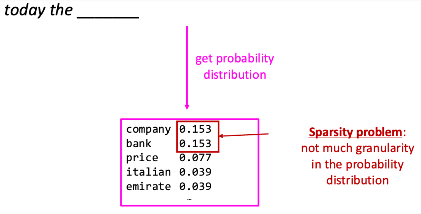
$$\hat{P}(w \mid \text{today the}) = \frac{\text{count(today the } w)}{\text{count(today the)}}$$



Problems with n-gram language models

- **Data sparsity:** Zero probability if an n-gram never appears
- **Storage cost:** Must store counts for all observed n-grams (e.g., early Google Translator used huge n-gram tables).
- **Practical note:** Easy to build (e.g., a trigram LM on a million-word corpus in seconds), but results could be sparse and limited.
  - Sparsity worsens as n increases (rarely $n > 5$ in practice).

$$\hat{P}(w \mid \text{today the}) = \frac{\text{count(today the } w\text{)}}{\text{count(today the)}}$$



Window-based neural language models

Idea (Bengio et al., 2000, 2003):

1. Use a small context window of previous words

~~*as the proctor started the clock*~~ *the students opened their* ______

discard

fixed window

Idea (Bengio et al., 2000, 2003):

1. Use a small context window of previous words
2. Map each word to an embedding

Fixed-window neural language models

Idea (Bengio et al., 2000, 2003):

1. Use a small context window of previous words
2. Map each word to an embedding
3. Combine them through a feed-forward network

Fixed-window neural language models

Idea (Bengio et al., 2000, 2003):

1. Use a small context window of previous words
2. Map each word to an embedding
3. Combine them through a feed-forward network
4. Output a probability distribution for the next word

Fixed-window neural language models

output distribution

$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$

hidden layer

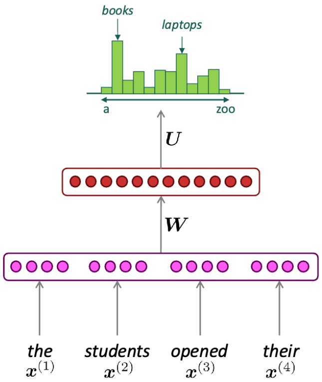
$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$

concatenated word embeddings

$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$

words / one-hot vectors

$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$



Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
    - No sparsity problem (embeddings generalize to unseen sequences).

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
  - No sparsity problem (embeddings generalize to unseen sequences).
  - No need to store all n-gram counts.

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
  - No sparsity problem (embeddings generalize to unseen sequences).
  - No need to store all n-gram counts.
- Limitations:

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
  - No sparsity problem (embeddings generalize to unseen sequences).
  - No need to store all n-gram counts.
- Limitations:
  - Fixed context window is too small; cannot capture long-range dependencies

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
  - No sparsity problem (embeddings generalize to unseen sequences).
  - No need to store all n-gram counts.
- Limitations:
  - Fixed context window is too small; cannot capture long-range dependencies
    - ***The students*** *who sat quietly in the large lecture hall* ***were*** *waiting for the professor.* (subject–verb agreement)

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
  - No sparsity problem (embeddings generalize to unseen sequences).
  - No need to store all n-gram counts.
- Limitations:
  - Fixed context window is too small; cannot capture long-range dependencies
    - ***The students*** *who sat quietly in the large lecture hall* ***were*** *waiting for the professor.* (subject–verb agreement)
    - *Jane met with Mary after work because* ***she*** *had an important announcement.* (pronoun resolution)

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
  - No sparsity problem (embeddings generalize to unseen sequences).
  - No need to store all n-gram counts.
- Limitations:
  - Fixed context window is too small; cannot capture long-range dependencies
    - ***The students*** *who sat quietly in the large lecture hall* ***were*** *waiting for the professor.* (subject–verb agreement)
    - *Jane met with Mary after work because* ***she*** *had an important announcement.* (pronoun resolution)
  - Enlarging the window makes the weight matrix W grow huge.

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
  - No sparsity problem (embeddings generalize to unseen sequences).
  - No need to store all n-gram counts.
- Limitations:
  - Fixed context window is too small; cannot capture long-range dependencies
    - ***The students*** *who sat quietly in the large lecture hall* ***were*** *waiting for the professor.* (subject–verb agreement)
    - *Jane met with Mary after work because* ***she*** *had an important announcement.* (pronoun resolution)
  - Enlarging the window makes the weight matrix W grow huge.
- Next step: We need architectures that can

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
    - No sparsity problem (embeddings generalize to unseen sequences).
    - No need to store all n-gram counts.
- Limitations:
    - Fixed context window is too small; cannot capture long-range dependencies
        - ***The students*** *who sat quietly in the large lecture hall* ***were*** *waiting for the professor.* (subject–verb agreement)
        - *Jane met with Mary after work because* ***she*** *had an important announcement.* (pronoun resolution)
    - Enlarging the window makes the weight matrix W grow huge.
- Next step: We need architectures that can
    1. handle arbitrary-length input,

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
    - No sparsity problem (embeddings generalize to unseen sequences).
    - No need to store all n-gram counts.
- Limitations:
    - Fixed context window is too small; cannot capture long-range dependencies
        - ***The students*** *who sat quietly in the large lecture hall* ***were*** *waiting for the professor.* (subject–verb agreement)
        - *Jane met with Mary after work because* ***she*** *had an important announcement.* (pronoun resolution)
    - Enlarging the window makes the weight matrix W grow huge.
- Next step: We need architectures that can
    1. handle arbitrary-length input,
    2. share parameters efficiently,

Fixed-window neural LMs: Pros & Cons

- Advantages over n-gram LMs:
  - No sparsity problem (embeddings generalize to unseen sequences).
  - No need to store all n-gram counts.
- Limitations:
  - Fixed context window is too small; cannot capture long-range dependencies
    - ***The students*** *who sat quietly in the large lecture hall* ***were*** *waiting for the professor.* (subject–verb agreement)
    - *Jane met with Mary after work because* ***she*** *had an important announcement.* (pronoun resolution)
  - Enlarging the window makes the weight matrix W grow huge.
- Next step: We need architectures that can
  1. handle arbitrary-length input,
  2. share parameters efficiently,
  3. capture sequential order and proximity.

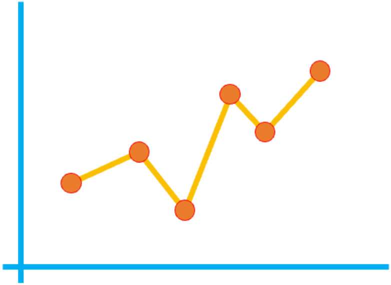
RNNs

Overview

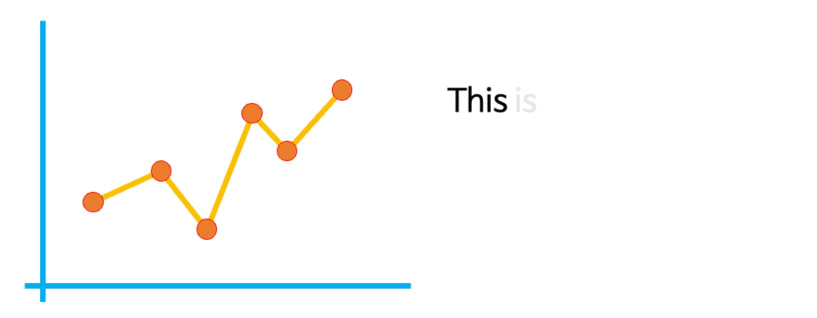
RNNs are widely used to process continuous data such as time series.

They work by **retaining past information** while processing new input.

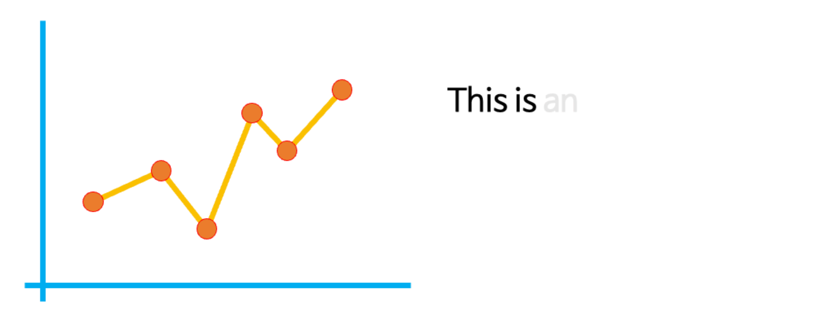
For example, changes in stock prices.



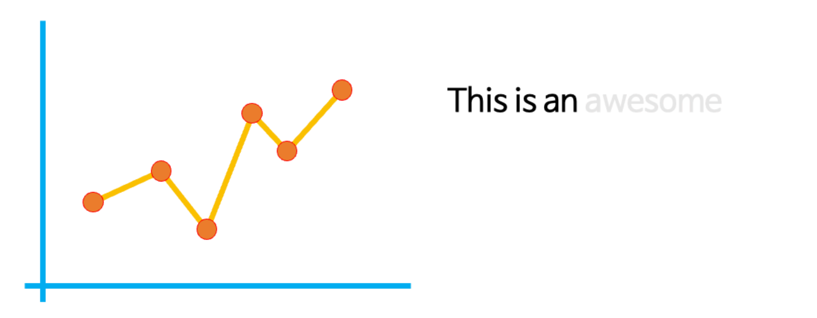
Or sequences like words in a sentence.



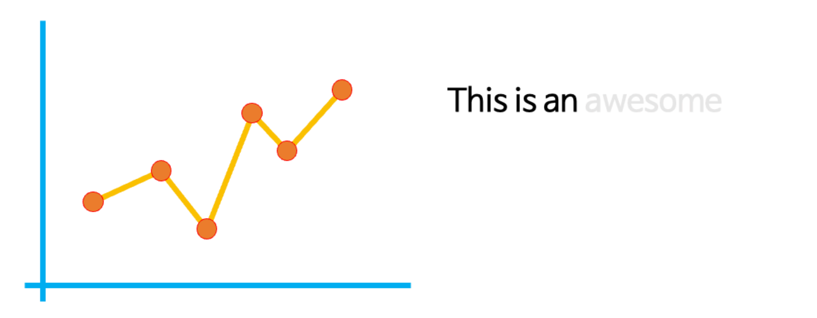
Or sequences like words in a sentence.



Or sequences like words in a sentence.

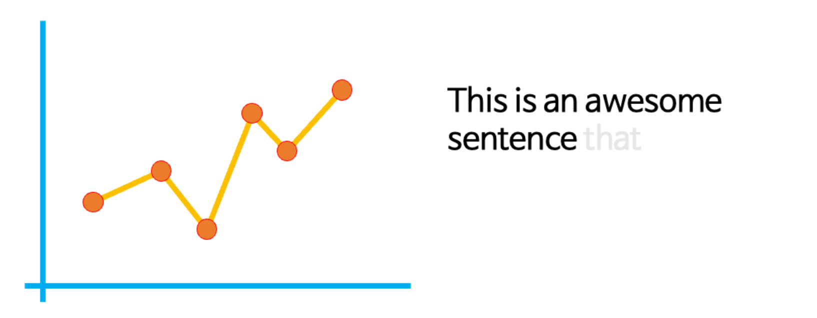


Or sequences like words in a sentence.

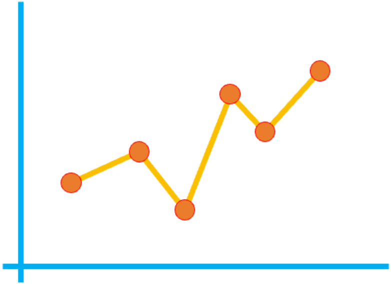


Or sequences like words in a sentence.

Or sequences like words in a sentence.

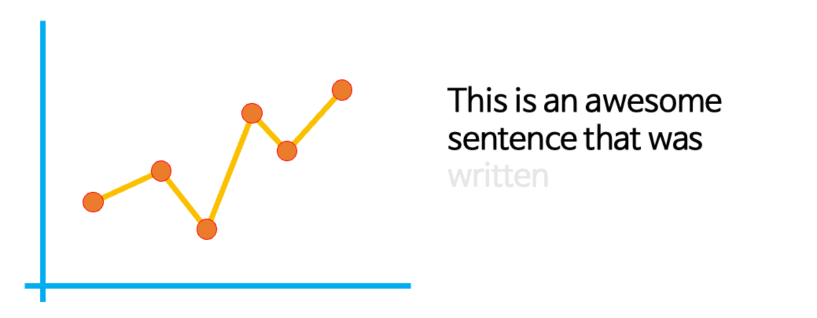


Or sequences like words in a sentence.

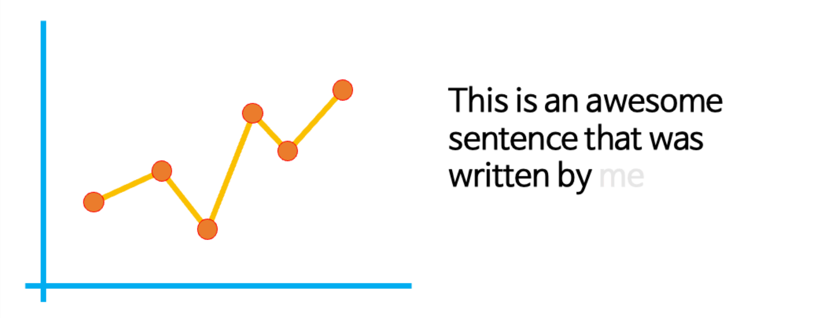


This is an awesome sentence that was

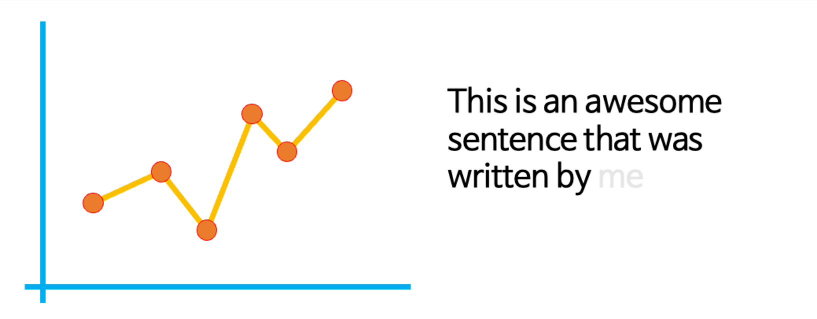
Or sequences like words in a sentence.



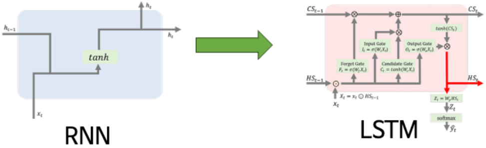
RNNs can effectively handle data where order matters.



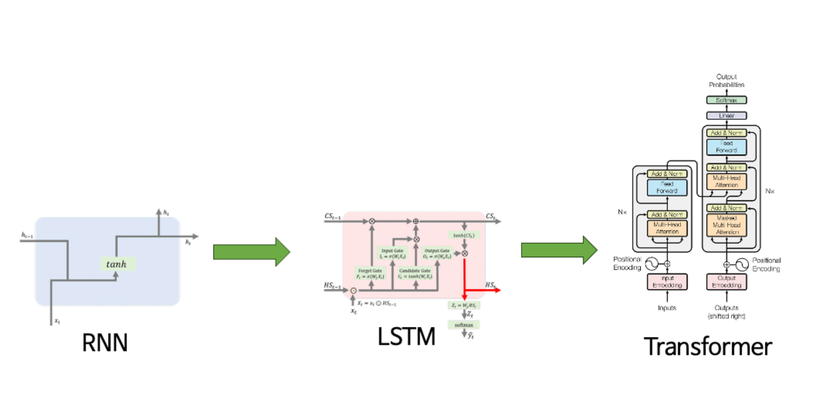
Think of RNNs as learning by extracting temporal features from time-series data.



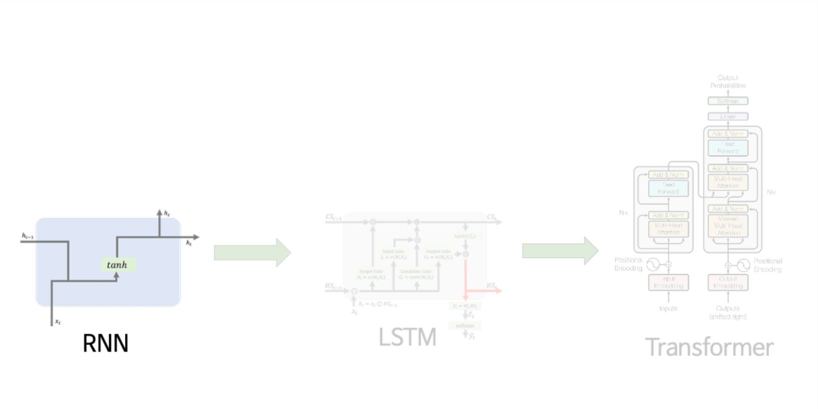
Moreover, RNNs evolved into LSTMs



Moreover, RNNs evolved into LSTMs and eventually into Transformers.

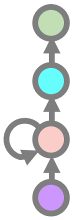


We will discuss (1) the structure, (2) the algorithms for learning sequential information, and (3) the uses of RNNs.

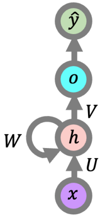


1. Structure

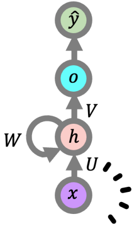
The structure of an RNN is simpler than you might think.



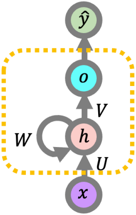
The structure of an RNN is simpler than you might think.



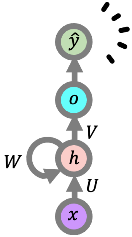
What an RNN does is to take an **input vector** x,



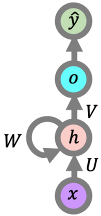
perform **internal computations**,



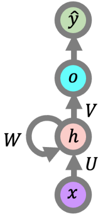
and produce an **output vector** $\hat{y}$.



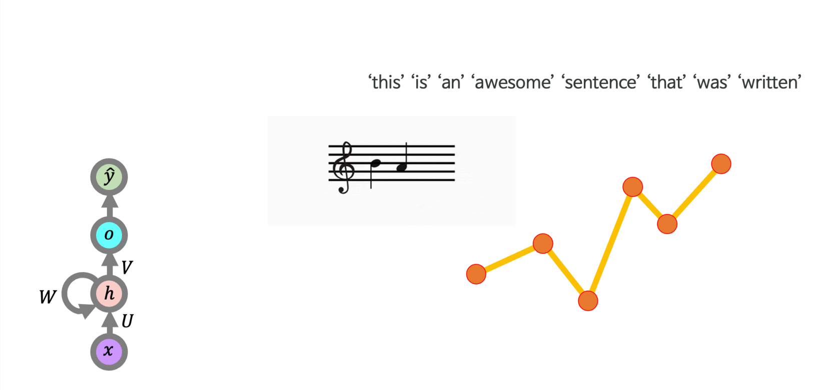
This is the **feedforward process** of the RNN.



The **types** of input vectors x and output vectors $\hat{y}$ can vary widely.

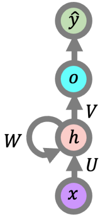


Time-series data (e.g., characters, stock price graphs, musical notes), as long as it can be represented sequentially, can be used as input.

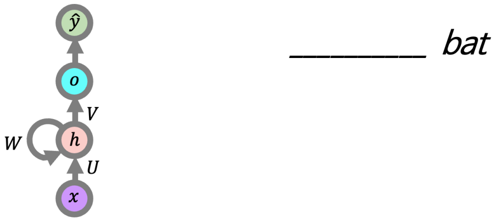


So, what is the benefit of processing sequential data?

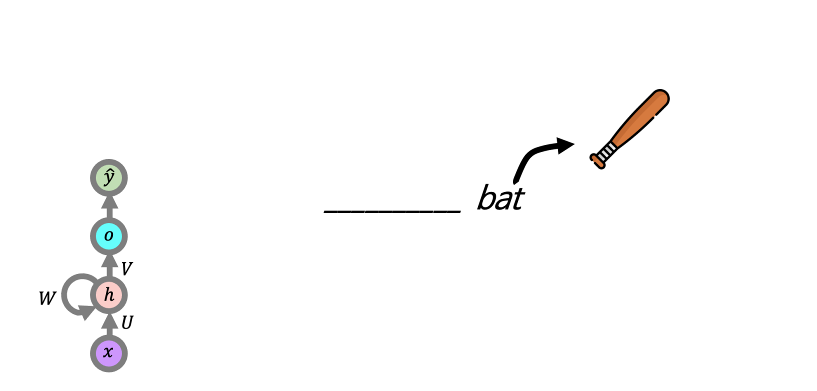
For example, let's assume this RNN is a model that translates *English* into *Spanish*.



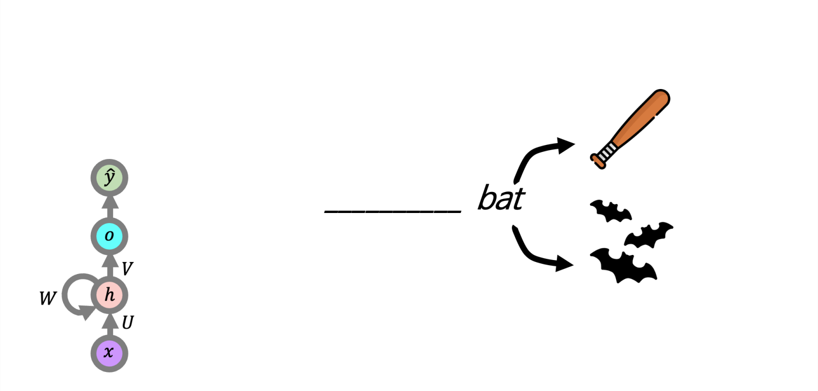
Suppose it encounters the word *bat* in English.



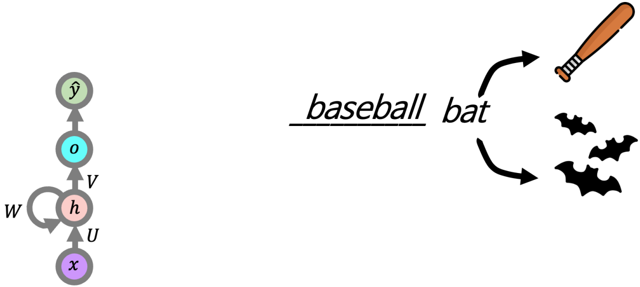
It has two possible meanings: a **baseball bat** or a flying bat.



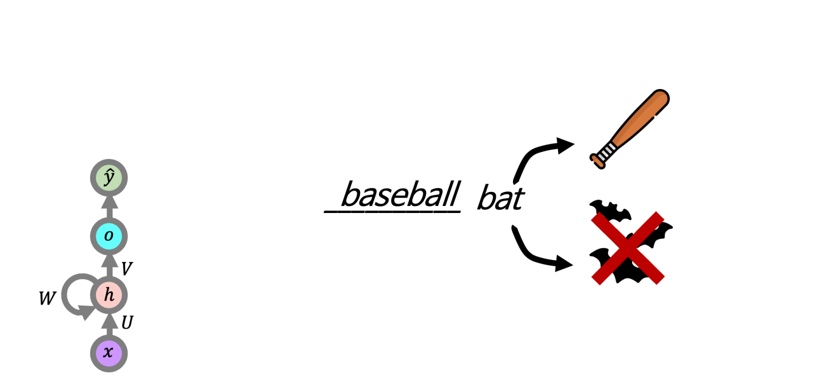
It has two possible meanings: a baseball bat or **a flying bat**.



However, if the previous word is *baseball*,



then with high probability, *bat* will be translated as *bate* in Spanish.

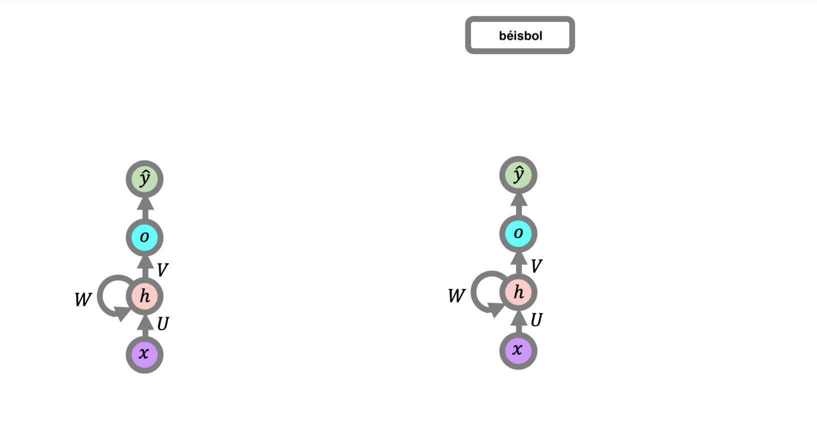


When translating *baseball*,

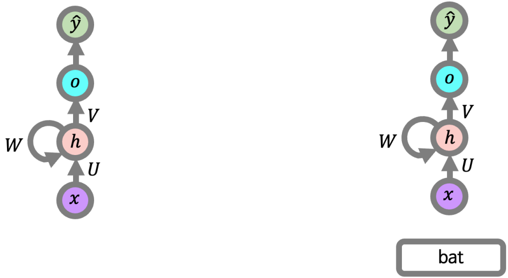
When translating *baseball*,

the internal state h is set with the **processed representation** of *baseball*.

Through this internal computation, *baseball* is translated into *béisbol*,



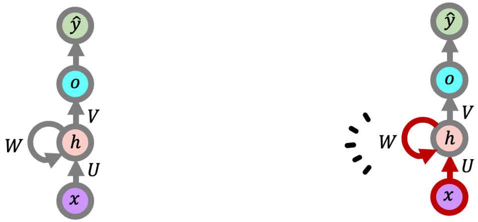
and when the model later encounters *bat*,



the hidden state created while translating *baseball*

influences how *bat* is translated – this is the core idea of an RNN.

The hidden state h gets updated to combine the context of both *baseball* and *bat*,

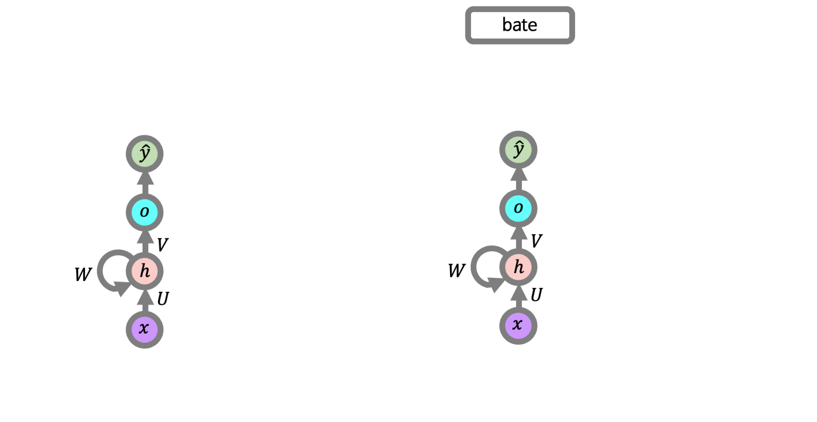


and thus, the probability of translating *bat* as *bate* (bat for baseball) becomes much higher than translating it as *murciélago* (the animal).

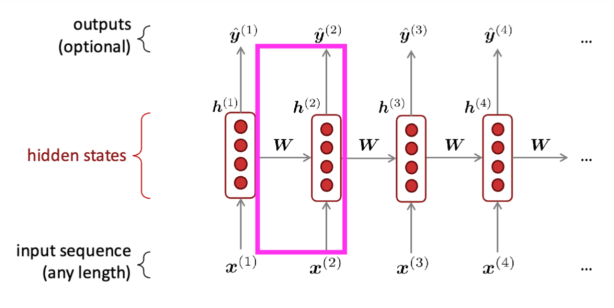
and thus, the probability of translating *bat* as *bate* (bat for baseball) becomes much higher than translating it as *murciélago* (the animal).

And thus, the probability of translating *bat* as *bate* (bat for baseball) becomes much higher than translating it as *murciélago* (the animal).

And thus, the probability of translating *bat* as *bate* (bat for baseball) becomes much higher than translating it as *murciélago* (the animal).



- Idea: Repeatedly apply the same weight matrix W at each time step
- Maintain a hidden state over time, feeding it back into the network to capture temporal dependencies



output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

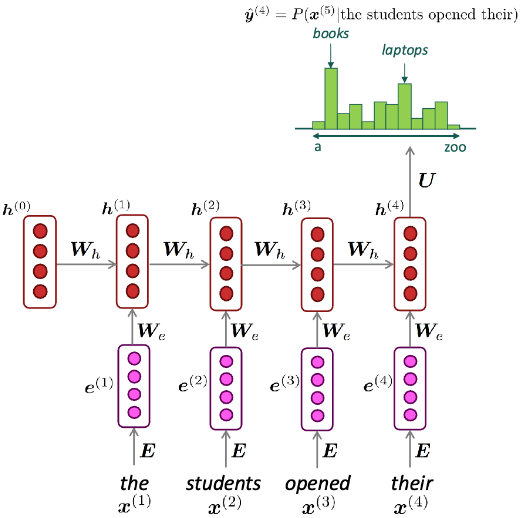
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



__Note__: this input sequence could be much longer now!

2-2. Training an RNN Language Model

1. Start with a large text corpus, represented as a sequence of words $w_1, \dots, w_{T-1}, w_T$.

2-2. Training an RNN Language Model

1. Start with a large text corpus, represented as a sequence of words $w_1, \ldots, w_{T-1}, w_T$.
2. Feed this sequence into the RNN-based language model.

2-2. Training an RNN Language Model

1. Start with a large text corpus, represented as a sequence of words $w_1, \ldots, w_{T-1}, w_T$.
2. Feed this sequence into the RNN-based language model.
3. At each time step t, the model outputs a probability distribution $\hat{\mathbf{y}}_t$ over the vocabulary.

2-2. Training an RNN Language Model

1. Start with a large text corpus, represented as a sequence of words $w_1, \dots, w_{T-1}, w_T$.
2. Feed this sequence into the RNN-based language model.
3. At each time step t, the model outputs a probability distribution $\hat{\mathbf{y}}_t$ over the vocabulary.
   - Internally, the RNN updates its hidden state $\mathbf{h}_t$, then applies a linear layer followed by softmax:

$$\hat{\mathbf{y}}_t = \mathrm{softmax}(W_o\,\mathbf{h}_t + b_o).$$

1. Start with a large text corpus, represented as a sequence of words $w_1, \dots, w_{T-1}, w_T$.
2. Feed this sequence into the RNN-based language model.
3. At each time step t, the model outputs a probability distribution $\hat{\mathbf{y}}_t$ over the vocabulary.
   - Internally, the RNN updates its hidden state $\mathbf{h}_t$, then applies a linear layer followed by softmax:

     $$\hat{\mathbf{y}}_t = \mathrm{softmax}(W_o\,\mathbf{h}_t + b_o).$$

   - Each component of $\hat{\mathbf{y}}_t$ corresponds to

     $$P(w_{t+1} = v_i \mid w_1, \dots, w_t),$$

     i.e., the probability that the next word is v_i.

2-2. Training an RNN Language Model

1. Start with a large text corpus, represented as a sequence of words $w_1, \dots, w_{T-1}, w_T$.
2. Feed this sequence into the RNN-based language model.
3. At each time step t, the model outputs a probability distribution $\hat{\mathbf{y}}_t$ over the vocabulary.
   - Internally, the RNN updates its hidden state $\mathbf{h}_t$, then applies a linear layer followed by softmax:
   $$\hat{\mathbf{y}}_t = \mathrm{softmax}(W_o\,\mathbf{h}_t + b_o).$$
   - Each component of $\hat{\mathbf{y}}_t$ corresponds to
   $$P(w_{t+1} = v_i \mid w_1, \dots, w_t),$$
   i.e., the probability that the next word is v_i.
   - Put simply, at every step t, the model predicts the likelihood of each possible next word given all preceding words

2-2. Training an RNN Language Model

1. Start with a large text corpus, represented as a sequence of words $w_1, \ldots, w_{T-1}, w_T$.
2. Feed this sequence into the RNN-based language model.
3. At each time step t, the model outputs a probability distribution $\hat{\mathbf{y}}_t$ over the vocabulary.
   - Internally, the RNN updates its hidden state $\mathbf{h}_t$, then applies a linear layer followed by softmax:

     $$\hat{\mathbf{y}}_t = \mathrm{softmax}(W_o\,\mathbf{h}_t + b_o).$$

   - Each component of $\hat{\mathbf{y}}_t$ corresponds to

     $$P(w_{t+1} = v_i \mid w_1, \ldots, w_t),$$

     i.e., the probability that the next word is v_i.
   - Put simply, at every step t, the model predicts the likelihood of each possible next word given all preceding words
   - *autoregressive, causal LM generation*

- **Loss** at step t:

$$\mathcal{J}^{(t)} = -\sum_{i=1}^{|V|} y_i^{(t)} \log \hat{y}_i^{(t)} = -\log \hat{y}_{w_{t+1}}^{(t)},$$

  where:

  - $y^{(t)}$: one-hot vector for the true next word w_{t+1}.
  - $\hat{y}^{(t)}$: predicted probability distribution over the vocabulary from the softmax layer.
  - This is the **cross-entropy loss** between the predicted distribution and the true label.

Example: Why one-hot vectors in the loss?

- Vocabulary: {apple, banana, orange} ($|V| = 3$)

Example: Why one-hot vectors in the loss?

- Vocabulary: {apple, banana, orange} ($|V| = 3$)
- Model prediction (*softmax output* at step t):

$$\hat{y}^{(t)} = [0.2,\ 0.7,\ 0.1]$$

Example: Why one-hot vectors in the loss?

- Vocabulary: {apple, banana, orange} ($|V| = 3$)
- Model prediction (*softmax output* at step t):

$$\hat{y}^{(t)} = [0.2,\ 0.7,\ 0.1]$$

- True next word: apple

$$y^{(t)} = [1,\ 0,\ 0] \quad \text{(one-hot vector)}$$

Example: Why one-hot vectors in the loss?

- Vocabulary: {apple, banana, orange} ($|V| = 3$)
- Model prediction (*softmax output* at step t):

$$\hat{y}^{(t)} = [0.2,\ 0.7,\ 0.1]$$

- True next word: **apple**

$$y^{(t)} = [1,\ 0,\ 0] \quad \text{(one-hot vector)}$$

- Cross-entropy loss:

$$\mathcal{J}^{(t)} = -\sum_{i=1}^{3} y_i^{(t)} \log \hat{y}_i^{(t)} = -\log \hat{y}_{\text{apple}}^{(t)} = -\log 0.2 \approx 1.609$$

Example: Why one-hot vectors in the loss?

- Vocabulary: {apple, banana, orange} ($|V| = 3$)
- Model prediction (*softmax output* at step t):

$$\hat{y}^{(t)} = [0.2,\ 0.7,\ 0.1]$$

- True next word: **apple**

$$y^{(t)} = [1,\ 0,\ 0] \quad \text{(one-hot vector)}$$

- Cross-entropy loss:

$$\mathcal{J}^{(t)} = -\sum_{i=1}^{3} y_i^{(t)} \log \hat{y}_i^{(t)} = -\log \hat{y}_{\text{apple}}^{(t)} = -\log 0.2 \approx 1.609$$

  - A one-hot vector marks the position of the true word in the vocabulary.

Example: Why one-hot vectors in the loss?

- Vocabulary: {apple, banana, orange} ($|V| = 3$)
- Model prediction (*softmax output* at step t):

$$\hat{y}^{(t)} = [0.2,\ 0.7,\ 0.1]$$

- True next word: **apple**

$$y^{(t)} = [1,\ 0,\ 0] \quad \text{(one-hot vector)}$$

- Cross-entropy loss:

$$\mathcal{J}^{(t)} = -\sum_{i=1}^{3} y_i^{(t)} \log \hat{y}_i^{(t)} = -\log \hat{y}_{\text{apple}}^{(t)} = -\log 0.2 \approx 1.609$$

  - A one-hot vector marks the position of the true word in the vocabulary.
  - In cross-entropy, this ensures the loss only depends on the probability of the true word.

Example: Why one-hot vectors in the loss?

- Vocabulary: {apple, banana, orange} ($|V| = 3$)
- Model prediction (*softmax output* at step t):

$$\hat{y}^{(t)} = [0.2,\ 0.7,\ 0.1]$$

- True next word: **apple**

$$y^{(t)} = [1,\ 0,\ 0] \quad \text{(one-hot vector)}$$

- Cross-entropy loss:

$$\mathcal{J}^{(t)} = -\sum_{i=1}^{3} y_i^{(t)} \log \hat{y}_i^{(t)} = -\log \hat{y}_{\text{apple}}^{(t)} = -\log 0.2 \approx 1.609$$

  - A one-hot vector marks the position of the true word in the vocabulary.
  - In cross-entropy, this ensures the loss only depends on the probability of the true word.
  - Compare: Higher probability for the correct word $\Rightarrow$ lower loss.

Example: Why one-hot vectors in the loss?

- Vocabulary: {apple, banana, orange} ($|V| = 3$)
- Model prediction (*softmax output* at step t):

$$\hat{y}^{(t)} = [0.2,\ 0.7,\ 0.1]$$

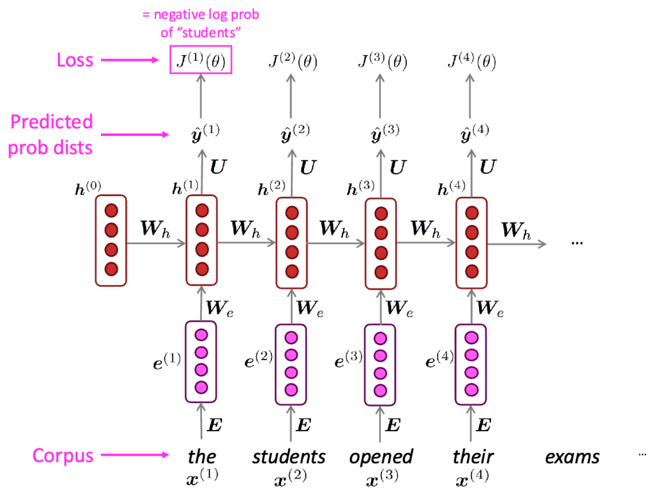
- True next word: **apple**

$$y^{(t)} = [1,\ 0,\ 0] \quad \text{(one-hot vector)}$$

- Cross-entropy loss:

$$\mathcal{J}^{(t)} = -\sum_{i=1}^{3} y_i^{(t)} \log \hat{y}_i^{(t)} = -\log \hat{y}_{\text{apple}}^{(t)} = -\log 0.2 \approx 1.609$$

  - A one-hot vector marks the position of the true word in the vocabulary.
  - In cross-entropy, this ensures the loss only depends on the probability of the true word.
  - Compare: Higher probability for the correct word $\Rightarrow$ lower loss.
  - Example: if $\hat{y}_{\text{apple}} = 0.9$, then loss $= -\log 0.9 \approx 0.105$.


= negative log prob
of "students"
Loss
$J^{(1)}(\theta)$
$J^{(2)}(\theta)$
$J^{(3)}(\theta)$
$J^{(4)}(\theta)$
Predicted
prob dists
$\hat{y}^{(1)}$
$\hat{y}^{(2)}$
$\hat{y}^{(3)}$
$\hat{y}^{(4)}$
U
$h^{(0)}$
$h^{(1)}$
$h^{(2)}$
$h^{(3)}$
$h^{(4)}$
W_h
...
W_e
$e^{(1)}$
$e^{(2)}$
$e^{(3)}$
$e^{(4)}$
E
Corpus
the
students
opened
their
exams
...
$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$x^{(4)}$

- Overall (average) loss over the sequence:

$$\mathcal{J}(\theta) = \frac{1}{T}\sum_{t=1}^{T}\mathcal{J}^{(t)}(\theta)$$

  - Sum the losses across all time steps.
  - Divide by the sequence length T to normalize for varying sequence lengths.
  - This gives the **average negative log-likelihood per word**, the main training objective of the RNN language model.

2-3. Training *in practice*

- However, computing the loss and gradients over the **entire corpus** $x_1, \dots, x_T$ is prohibitively **expensive**.

2-3. Training *in practice*

- However, computing the loss and gradients over the **entire corpus** $x_1, \dots, x_T$ is prohibitively **expensive**.
- In practice, we split the corpus into smaller units (e.g., individual sentences or documents).

2-3. Training *in practice*

- However, computing the loss and gradients over the **entire corpus** $x_1, \ldots, x_T$ is prohibitively **expensive**.
- In practice, we split the corpus into smaller units (e.g., individual sentences or documents).
- *Recall*: Stochastic Gradient Descent (SGD) computes the loss (using cross-entropy) and gradients on a small batch of data, then updates the model parameters.

- However, computing the loss and gradients over the **entire corpus** $x_1, \dots, x_T$ is prohibitively **expensive**.
- In practice, we split the corpus into smaller units (e.g., individual sentences or documents).
- *Recall*: Stochastic Gradient Descent (SGD) computes the loss (using cross-entropy) and gradients on a small batch of data, then updates the model parameters.
- Concretely, for each batch of sentences, we

2-3. Training *in practice*

- However, computing the loss and gradients over the **entire corpus** $x_1, \dots, x_T$ is prohibitively **expensive**.
- In practice, we split the corpus into smaller units (e.g., individual sentences or documents).
- *Recall*: Stochastic Gradient Descent (SGD) computes the loss (using cross-entropy) and gradients on a small batch of data, then updates the model parameters.
- Concretely, for each batch of sentences, we
    1. compute the batch loss $\mathcal{J}(\theta)$,

2-3. Training *in practice*

- However, computing the loss and gradients over the **entire corpus** $x_1, \ldots, x_T$ is prohibitively **expensive**.
- In practice, we split the corpus into smaller units (e.g., individual sentences or documents).
- *Recall*: Stochastic Gradient Descent (SGD) computes the loss (using cross-entropy) and gradients on <u>a small batch of data</u>, then updates the model parameters.
- Concretely, for each batch of sentences, we
    1. compute the batch loss $\mathcal{J}(\theta)$,
    2. compute the gradient $\nabla_\theta \mathcal{J}(\theta)$,

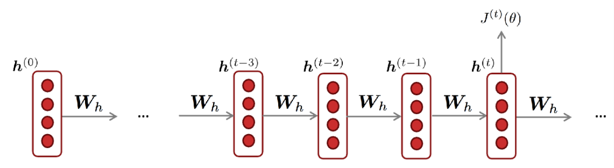
2-3. Training *in practice*

- However, computing the loss and gradients over the **entire corpus** $x_1, \ldots, x_T$ is prohibitively **expensive**.
- In practice, we split the corpus into smaller units (e.g., individual sentences or documents).
- *Recall*: Stochastic Gradient Descent (SGD) computes the loss (using cross-entropy) and gradients on a small batch of data, then updates the model parameters.
- Concretely, for each batch of sentences, we
    1. compute the batch loss $\mathcal{J}(\theta)$,
    2. compute the gradient $\nabla_\theta \mathcal{J}(\theta)$,
    3. update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{J}(\theta)$,

2-3. Training *in practice*

- However, computing the loss and gradients over the **entire corpus** $x_1, \ldots, x_T$ is prohibitively **expensive**.
- In practice, we split the corpus into smaller units (e.g., individual sentences or documents).
- *Recall*: Stochastic Gradient Descent (SGD) computes the loss (using cross-entropy) and gradients on a small batch of data, then updates the model parameters.
- Concretely, for each batch of sentences, we
  1. compute the batch loss $\mathcal{J}(\theta)$,
  2. compute the gradient $\nabla_\theta \mathcal{J}(\theta)$,
  3. update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{J}(\theta)$,
  4. and repeat.

2-4. Training: Backpropagation



- The loss $J^{(t)}(\theta)$ depends on the shared weight matrix W_h at every time step.

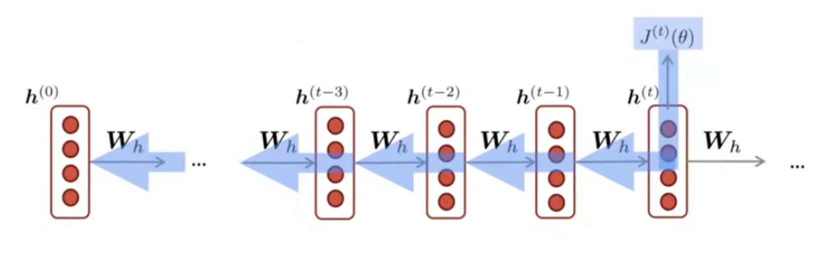
- Therefore,

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{k=1}^{t} \frac{\partial J^{(t)}}{\partial W_h^{(k)}}.$$

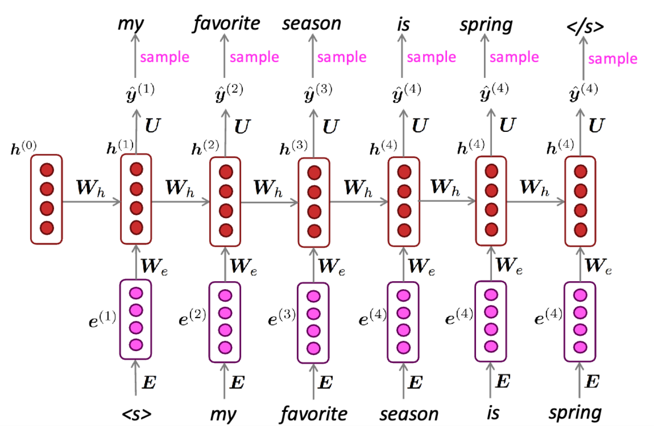
- In other words, the gradient w.r.t. the repeated parameter is the sum of its contributions over all time steps.

- **Why?** Because the RNN "unrolls" in time but reuses the same W_h at each step (parameter sharing).

2-4. Training: Backpropagation



Just like an n-gram language model, you can use an RNN model to generate text by **repeated sampling**. The sampled output becomes the next step's input.



Just like an n-gram language model, you can use an RNN model to generate text by **repeated sampling**. The sampled output becomes the next step's input.

Example

- Start token: `<S>`
- Step 1: Model predicts distribution, sample `my`
- Step 2: Input = `my`, sample `favorite`
- Step 3: Input = `favorite`, sample `season`
- Step 4: Input = `season`, sample `is`
- Step 5: Input = `is`, sample `spring`

$\Rightarrow$ "my favorite season is spring"

RNN-LM trained on *Harry Potter*:

"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

2-6. Evaluation: Perplexity

- The most common evaluation metric for language models is perplexity.

$$\mathrm{Perplexity}(w_{1:T}) = P(w_{1:T})^{-\frac{1}{T}} = \exp\Big(-\frac{1}{T}\sum_{t=1}^{T}\log P(w_t \mid w_{1:t-1})\Big)$$

https://github.com/asahi417/lmppl
https://github.com/Picovoice/llm-compression-benchmark

2-6. Evaluation: Perplexity

- The most common evaluation metric for language models is perplexity.

$$\text{Perplexity}(w_{1:T}) = P(w_{1:T})^{-\frac{1}{T}} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T}\log P(w_t \mid w_{1:t-1})\right)$$

- **Definition:** Exponential of the average **cross-entropy loss**.

https://github.com/asahi417/lmppl
https://github.com/Picovoice/llm-compression-benchmark

2-6. Evaluation: Perplexity

- The most common evaluation metric for language models is perplexity.

$$\text{Perplexity}(w_{1:T}) = P(w_{1:T})^{-\frac{1}{T}} = \exp\Big(-\frac{1}{T}\sum_{t=1}^{T}\log P(w_t \mid w_{1:t-1})\Big)$$

- **Definition:** Exponential of the average **cross-entropy loss**.
- **Intuition:**

https://github.com/asahi417/lmppl
https://github.com/Picovoice/llm-compression-benchmark

2-6. Evaluation: Perplexity

- The most common evaluation metric for language models is perplexity.

$$\text{Perplexity}(w_{1:T}) = P(w_{1:T})^{-\frac{1}{T}} = \exp\Big(-\frac{1}{T}\sum_{t=1}^{T}\log P(w_t \mid w_{1:t-1})\Big)$$

- **Definition:** Exponential of the average **cross-entropy loss**.
- **Intuition:**
  - Measures how “surprised” the model is when predicting the test data.

https://github.com/asahi417/lmppl
https://github.com/Picovoice/llm-compression-benchmark

2-6. Evaluation: Perplexity

- The most common evaluation metric for language models is perplexity.

$$\text{Perplexity}(w_{1:T}) = P(w_{1:T})^{-\frac{1}{T}} = \exp\Big(-\frac{1}{T}\sum_{t=1}^{T} \log P(w_t \mid w_{1:t-1})\Big)$$

- **Definition:** Exponential of the average **cross-entropy loss**.
- **Intuition:**
  - Measures how "surprised" the model is when predicting the test data.
  - Equivalent to the model's effective average branching factor (i.e., how many plausible next words it considers at each step).

https://github.com/asahi417/lmppl
https://github.com/Picovoice/llm-compression-benchmark

2-6. Evaluation: Perplexity

- The most common evaluation metric for language models is perplexity.

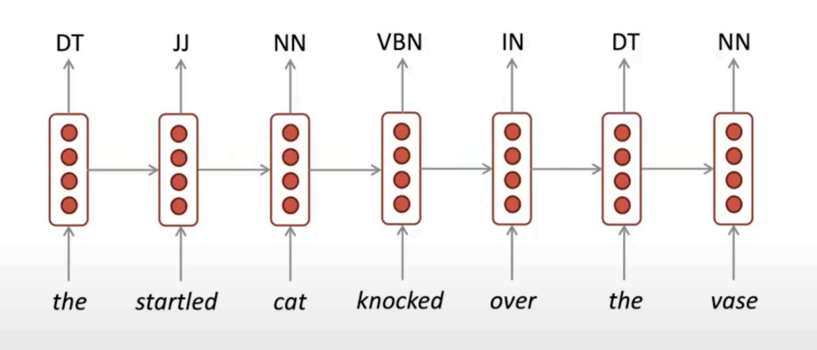
$$\text{Perplexity}(w_{1:T}) = P(w_{1:T})^{-\frac{1}{T}} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T}\log P(w_t \mid w_{1:t-1})\right)$$

- **Definition:** Exponential of the average **cross-entropy loss**.
- **Intuition:**
  - Measures how "surprised" the model is when predicting the test data.
  - Equivalent to the model's effective average branching factor (i.e., how many plausible next words it considers at each step).
- **Interpretation:** Lower perplexity $\Rightarrow$ model is less "perplexed" and makes more accurate predictions.

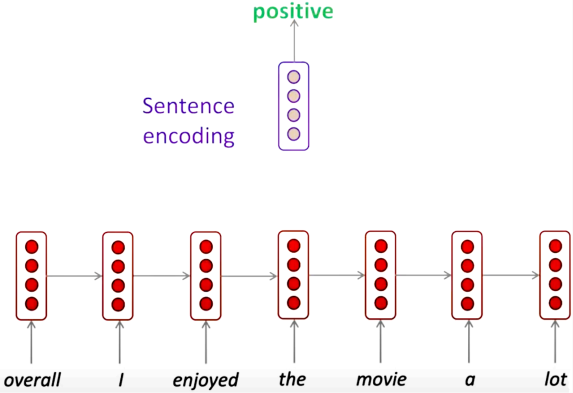
https://github.com/asahi417/lmppl
https://github.com/Picovoice/llm-compression-benchmark

2-7. Other uses: Sequence tagging

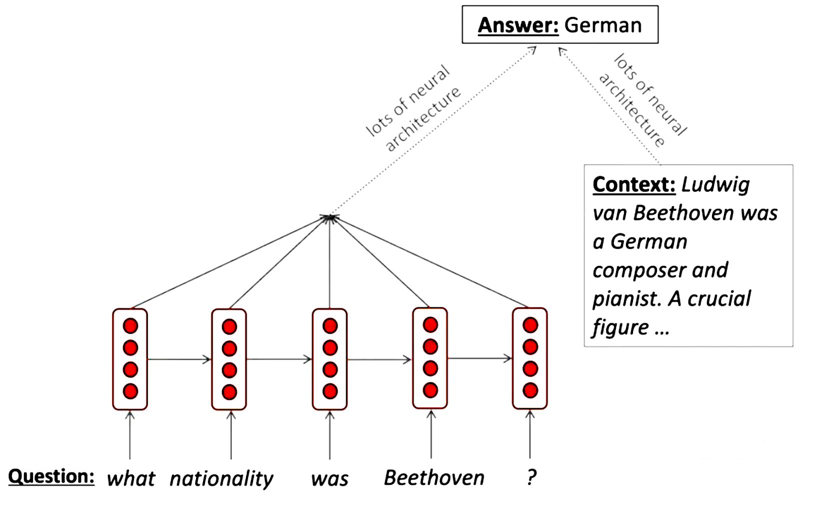
e.g., part-of-speech tagging, named-entity recognition



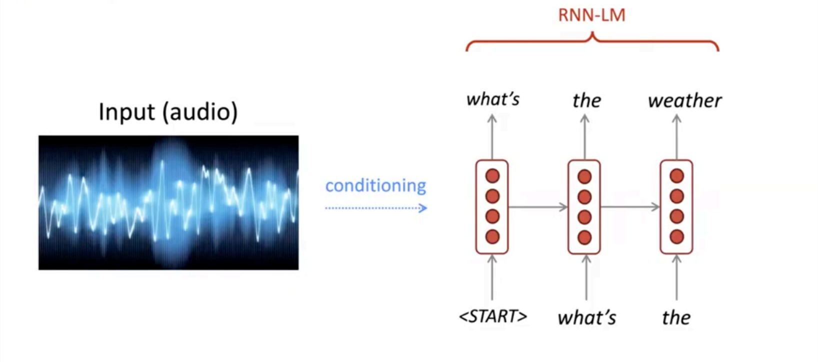
e.g., sentiment classification



e.g., question answering, machine translation



e.g., speech recognition, machine translation, summarization



Wrap-up

Wrap-up

- Language modeling: A system that predicts the next word

Wrap-up

- Language modeling: A system that predicts the next word
- n-gram language models

Wrap-up

- Language modeling: A system that predicts the next word
- n-gram language models
- Window-based neural language models

Wrap-up

- Language modeling: A system that predicts the next word
- n-gram language models
- Window-based neural language models
- RNNs

Wrap-up

- Language modeling: A system that predicts the next word
- n-gram language models
- Window-based neural language models
- RNNs
    - time series data

Wrap-up

- Language modeling: A system that predicts the next word
- n-gram language models
- Window-based neural language models
- RNNs
    - time series data
    - structure

Wrap-up

- Language modeling: A system that predicts the next word
- n-gram language models
- Window-based neural language models
- RNNs
    - time series data
    - structure
    - loss function

Wrap-up

- Language modeling: A system that predicts the next word
- n-gram language models
- Window-based neural language models
- RNNs
    - time series data
    - structure
    - loss function
    - evaluation

Wrap-up

- Language modeling: A system that predicts the next word
- n-gram language models
- Window-based neural language models
- RNNs
  - time series data
  - structure
  - loss function
  - evaluation
  - applications