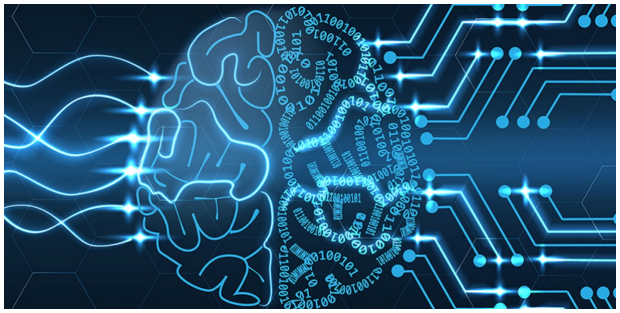
AI6130
Large Language Models

Lecture 3: Language Models



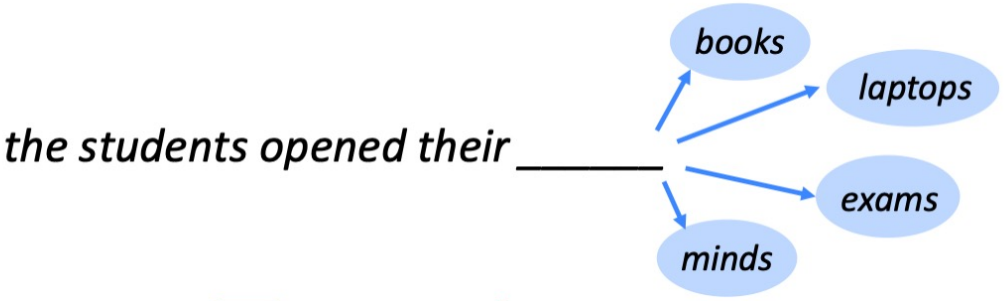
Instructor: Luu Anh Tuan

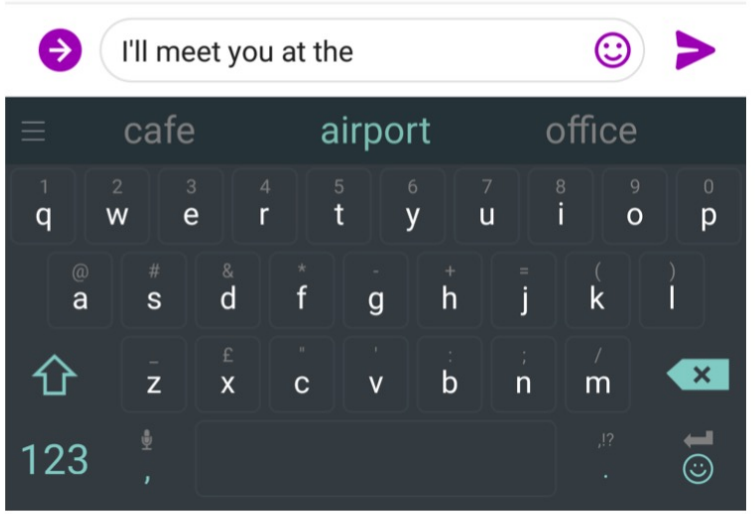
Email: anhtuan.luu@ntu.edu.sg

Office: #N4-02c-86

Language Model

A language model takes a list of words (history/ context), and attempts to predict the word that follows them





Language Model

(i) A language model takes a list of words (history/context), and attempts to predict the word that follows them

More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

Language Model

(ii) Language Models (LM) also assign a probability to a piece of text.

For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \, \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \, \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

Language Model - Ngram based Approach

A *n*-gram is a sequence of *n* consecutive words.

- unigrams: “the”, “students”, “opened”, ”their”
- bigrams: “the students”, “students opened”, “opened their”
- trigrams: “the students opened”, “students opened their”
- 4-grams: “the students opened their”

Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

Ngram based LM

Markov assumption: make a simplifying assumption that a word's probability depends only on the preceding *n-1* words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

How can we estimate this probability?

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

Ngram based LM

How can we estimate this probability?

$$= \frac{P(\boldsymbol{x}^{(t+1)}, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t-n+2)})}$$

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)}, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t-n+2)})}$$

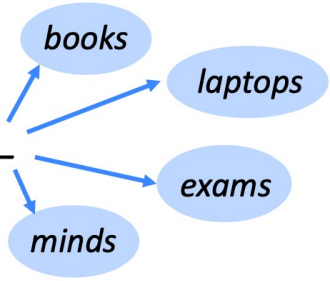
Count this in the corpus

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Problem with the Markov Assumption

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



For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Window-based Language Model

output distribution

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

hidden layer

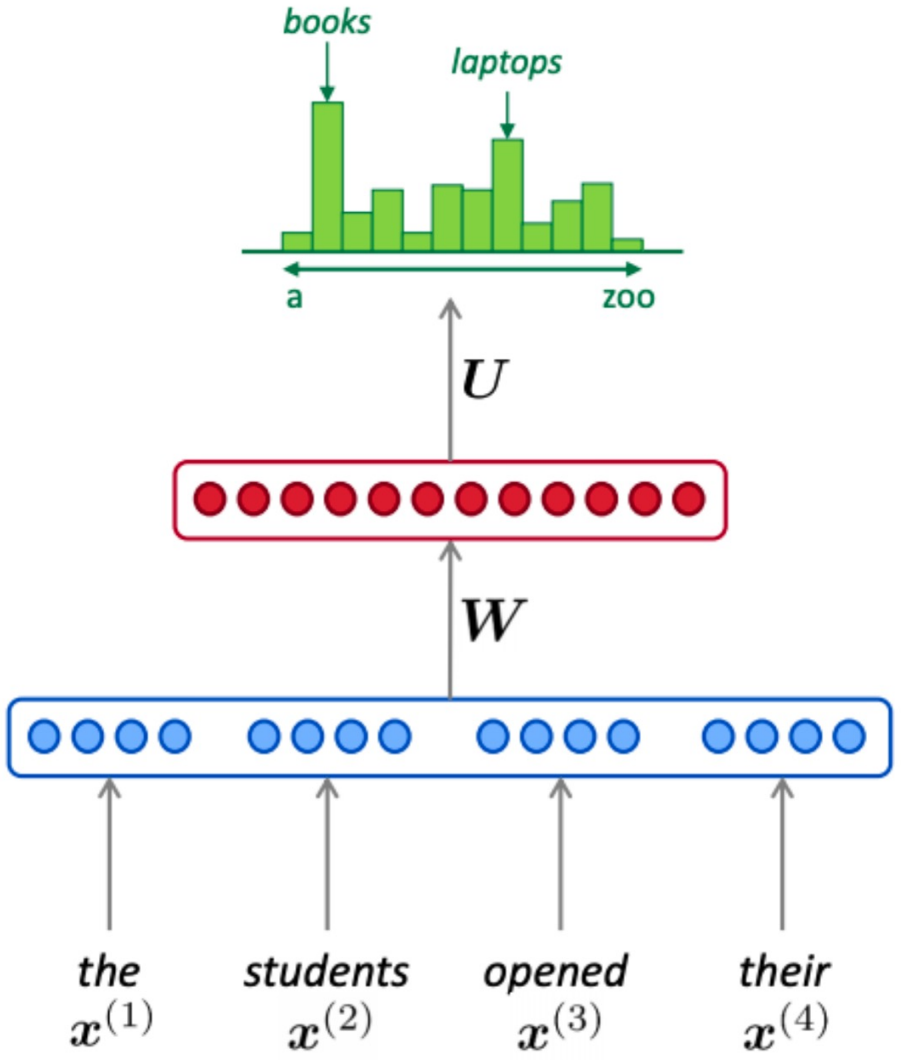
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

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

words / one-hot vectors

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



Neural Language Models: Conditioned vs. Unconditioned

Neural language models can either be designed to just predict the next word given the previous ones, or they can be designed to predict the next word given the previous ones *and* some additional conditioning sequence.

Unconditioned: $P(Y)$

At each step the LM predicts:

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

Examples:

- GPT-2 / GPT-3
- LLaMA

Conditioned: $P(Y \mid X)$

At each step the LM predicts: $P(y_t \mid y_{1:t-1}, x_{1:T})$

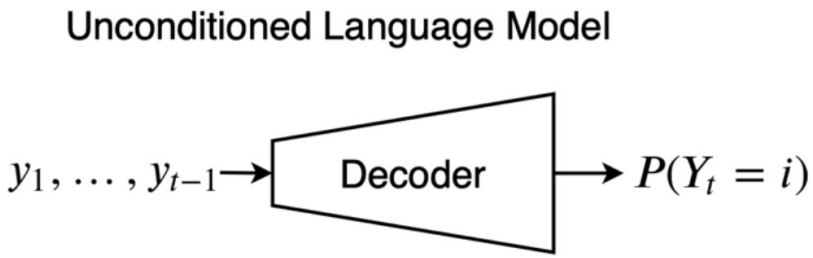
Examples

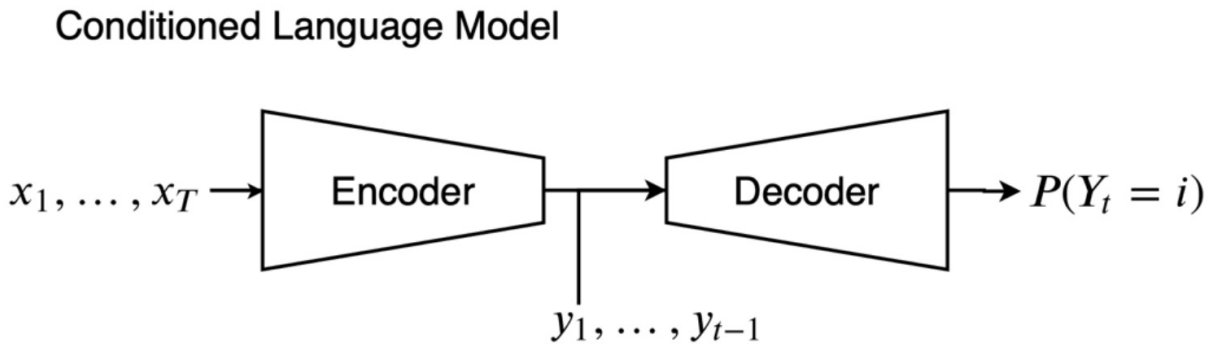
- T5
- Most machine translation models

Sometimes called sequence-to-sequence or seq2seq models.

Neural Language Models: Conditioned vs. Unconditioned

Unconditioned neural language models only have a decoder.
Conditioned ones have an encoder and a decoder.





Neural Language Models: Conditioned vs. Unconditioned

Theoretically, any task designed for a decoder-only architecture can be turned into one for an encoder-decoder architecture, and vice-versa.

TASK: Continue the sequence.

Decoder-only version:

P(Y="Once upon a time there lived a dreadful ogre.")

Encoder-decoder version:

P(Y="lived a dreadful ogre." | X="Once upon a time there")

Neural Language Models: Conditioned vs. Unconditioned

Theoretically, any task designed for a decoder-only architecture can be turned into one for an encoder-decoder architecture, and vice-versa.

TASK: Translate from English to French.

Decoder-only version:

P(Y="English: The hippo ate my homework. French: L'hippopotame a mangé mes devoirs.")

Encoder-decoder version:

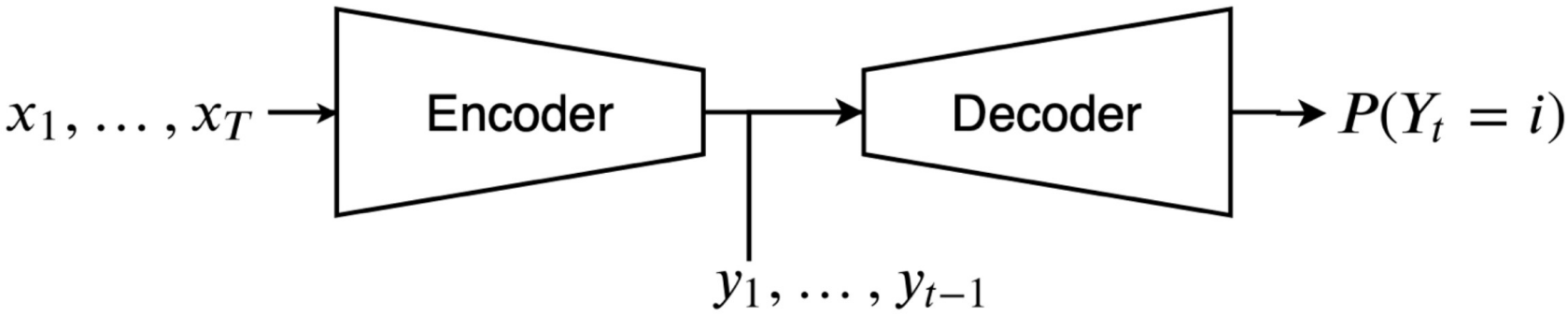
P(Y="L'hippopotame a mangé mes devoirs." | X="The hippo ate my homework.")

Conditioned vs. Unconditioned
Which one is better?

Fu, Zihao, Wai Lam, Qian Yu, Anthony Man-Cho So, Shengding Hu, Zhiyuan Liu, and Nigel Collier. "Decoder-only or encoder-decoder? interpreting language model as a regularized encoder-decoder."

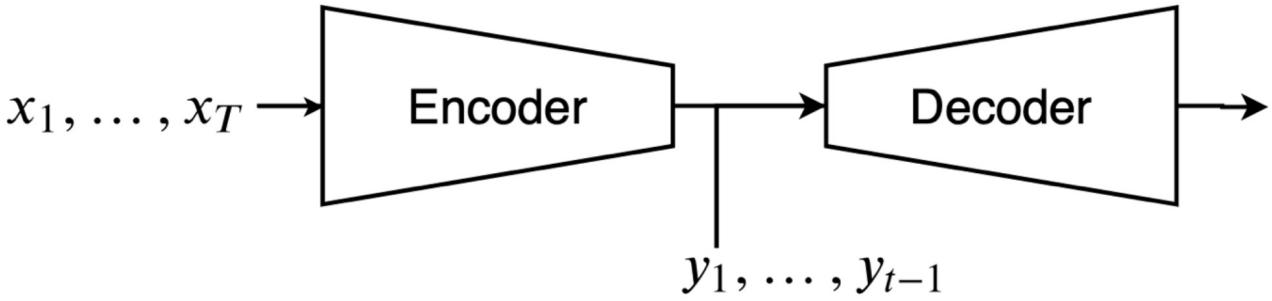
Summary of Terms You Should Know

Input sequence: $x_1, \ldots, x_T$
Target sequence: $y_1, \ldots, y_T$



Summary of Terms You Should Know

Input sequence: $x_1, \ldots, x_T$
Target sequence: $y_1, \ldots, y_T$



$P(Y_t = i)$

Or sometimes…

$P_\Theta(Y_t = i)$

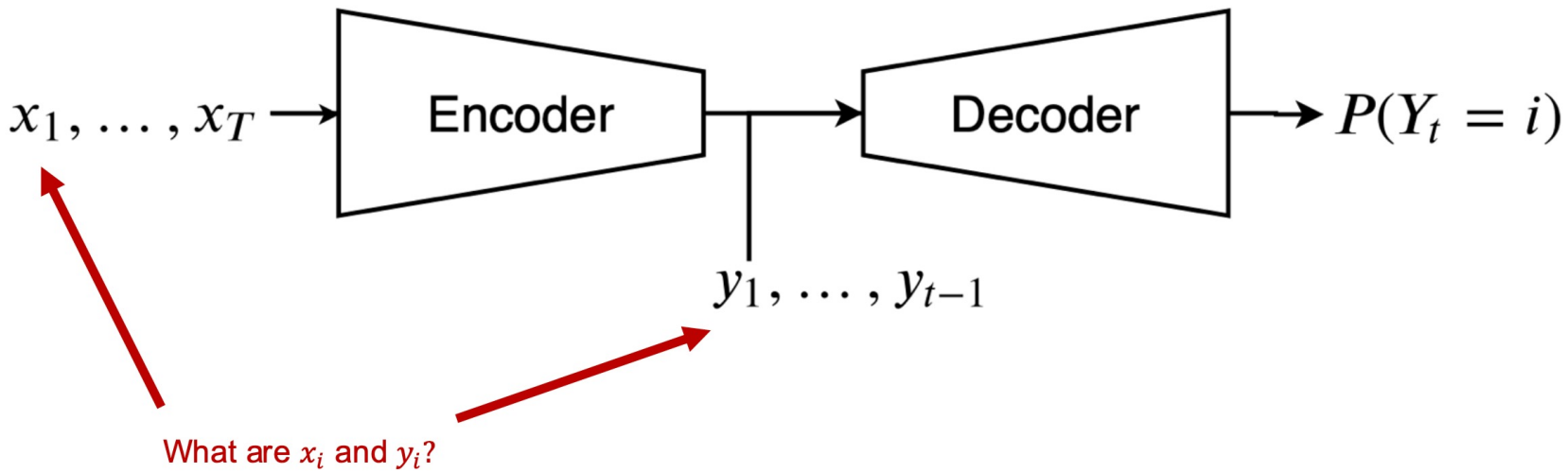
Or sometimes…

$P(Y_t = i | y_1, \ldots, y_{t-1}; x_1, \ldots, x_T; \Theta)$

Summary of Terms You Should Know

Input sequence: $x_1, \ldots, x_T$

Target sequence: $y_1, \ldots, y_T$



Tokenizing Text

A **tokenizer** takes text and turns it into a sequence of discrete **tokens.**

A **vocabulary** is the list of all available tokens.

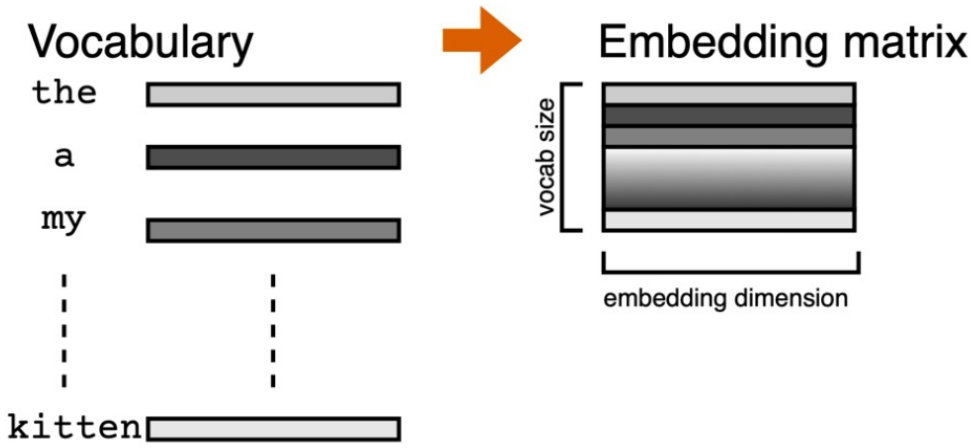
Let's tokenize: "A hippopotamus ate my homework."

Vocab Type	Example	Ex. length
character-level	['A', ' ', 'h', 'i', 'p', 'p', 'o', 'p', 'o', 't', 'a', 'm', 'u', 's', ' ', 'a', 't', 'e', ' ', 'm', 'y', ' ', 'h', 'o', 'm', 'e', 'w', 'o', 'r', 'k', '.']	31
subword-level	['A', 'hip', '##pop', '##ota', '##mus', 'ate', 'my', 'homework', '.']	9
word-level	['A', 'hippopotamus', 'ate', 'my', 'homework']	5

Turning Discrete Tokens into Embeddings

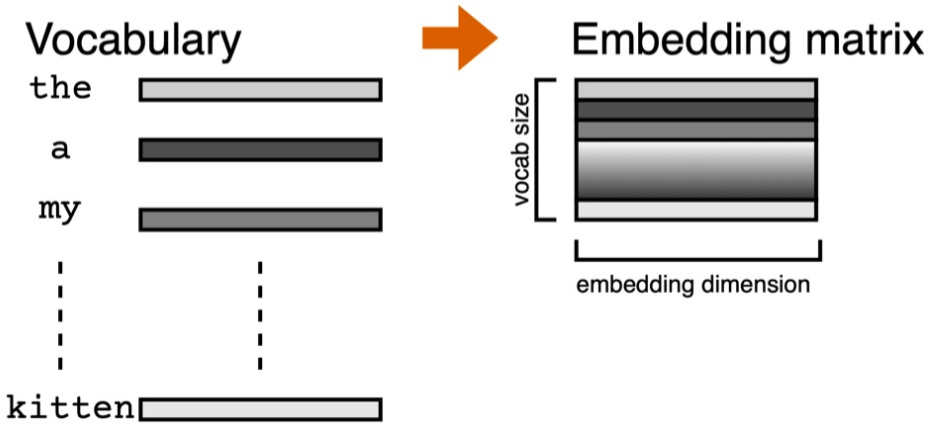
Neural networks cannot operate on discrete tokens.

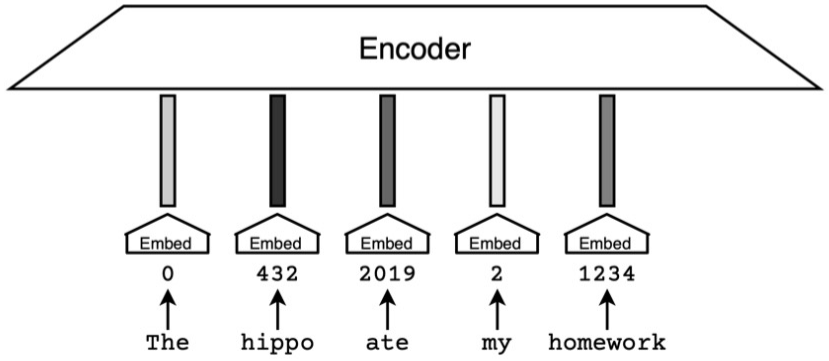
Instead, we build an **embedding matrix** which associates each token in the vocabulary with a vector embedding.



Encoder Inputs and Outputs

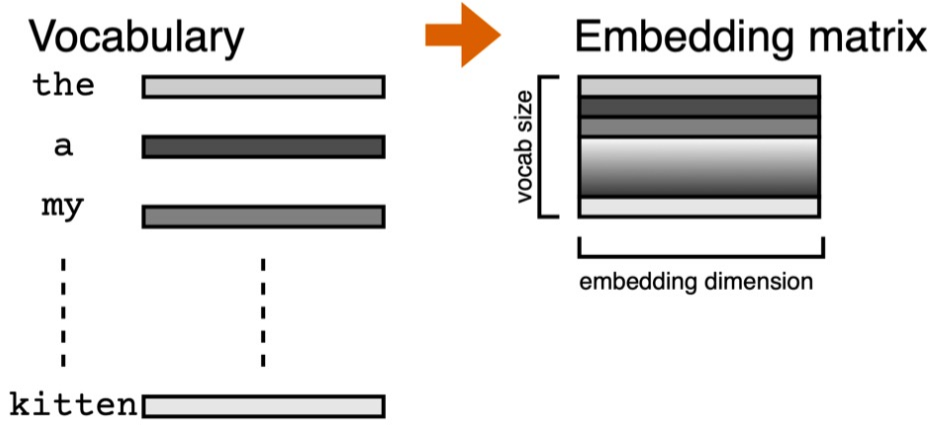
The encoder takes as input the vector representations of each token in the input sequence.

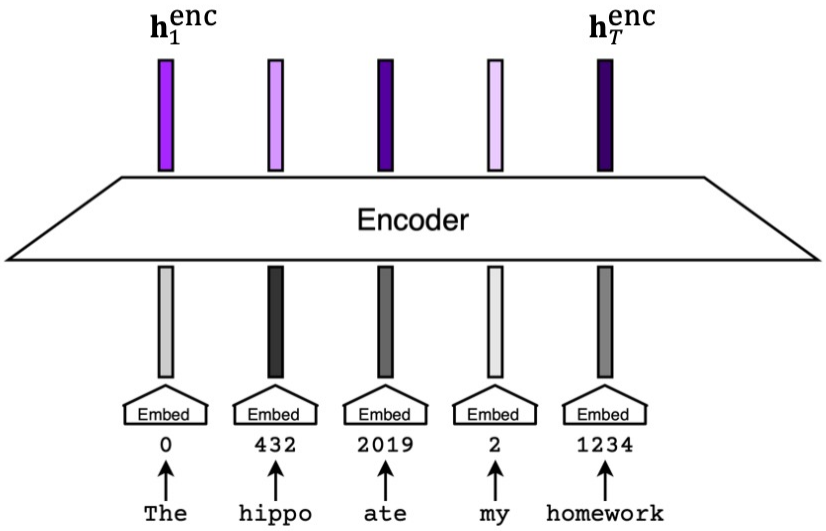




Encoder Inputs and Outputs

The encoder outputs a sequence of embeddings called **hidden states**.

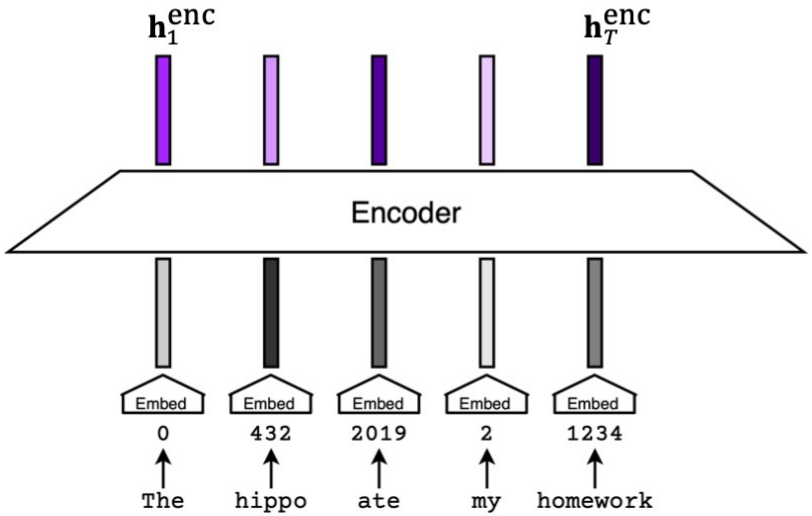


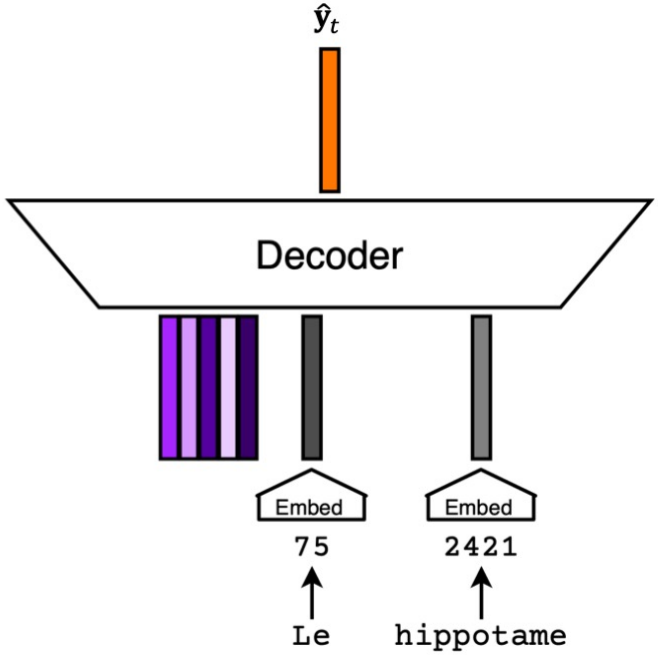


Decoder Inputs and Outputs

The decoder takes as input the hidden states from the encoder as well as the embeddings for the tokens seen so far in the target sequence.

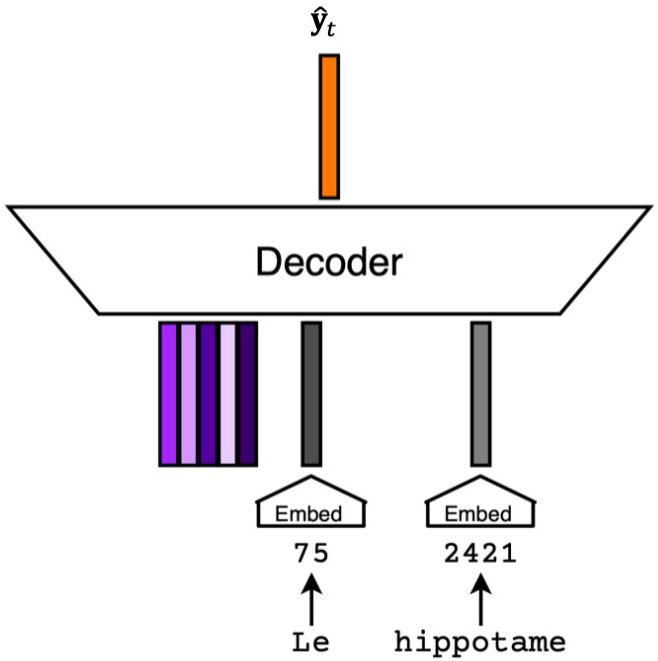
It outputs an embedding $\hat{\mathbf{y}}_t$.





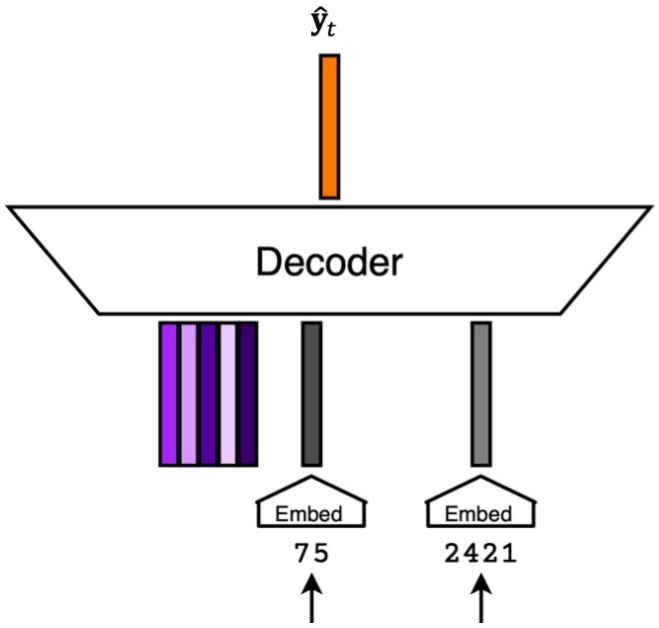
Decoder Inputs and Outputs

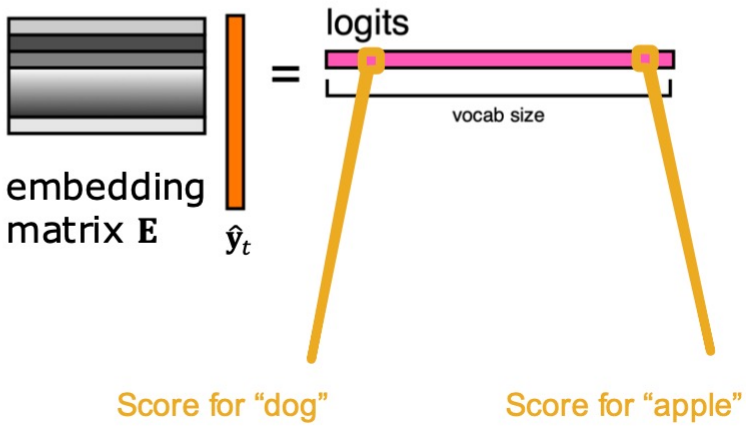
Ideally, $\hat{\mathbf{y}}_t$ would be as close as possible to the embedding of the true next token.



Decoder Inputs and Outputs

We multiply the predicted embedding $\hat{\mathbf{y}}_t$ by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as logits.





Decoder Inputs and Outputs

We multiply the predicted embedding $\hat{\mathbf{y}}_t$ by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as **logits**.

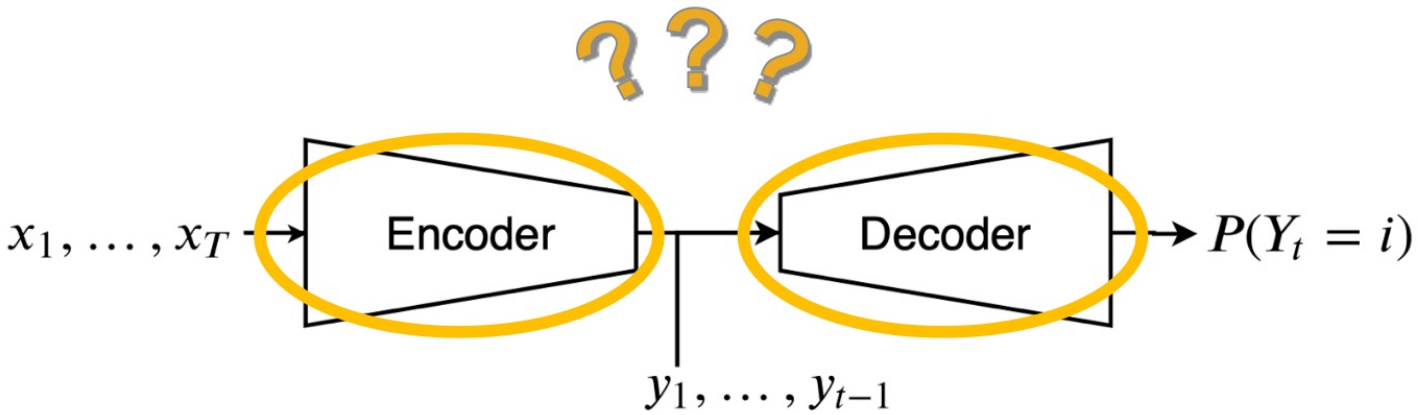
The **softmax function** is used to turn the logits into probabilities.

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

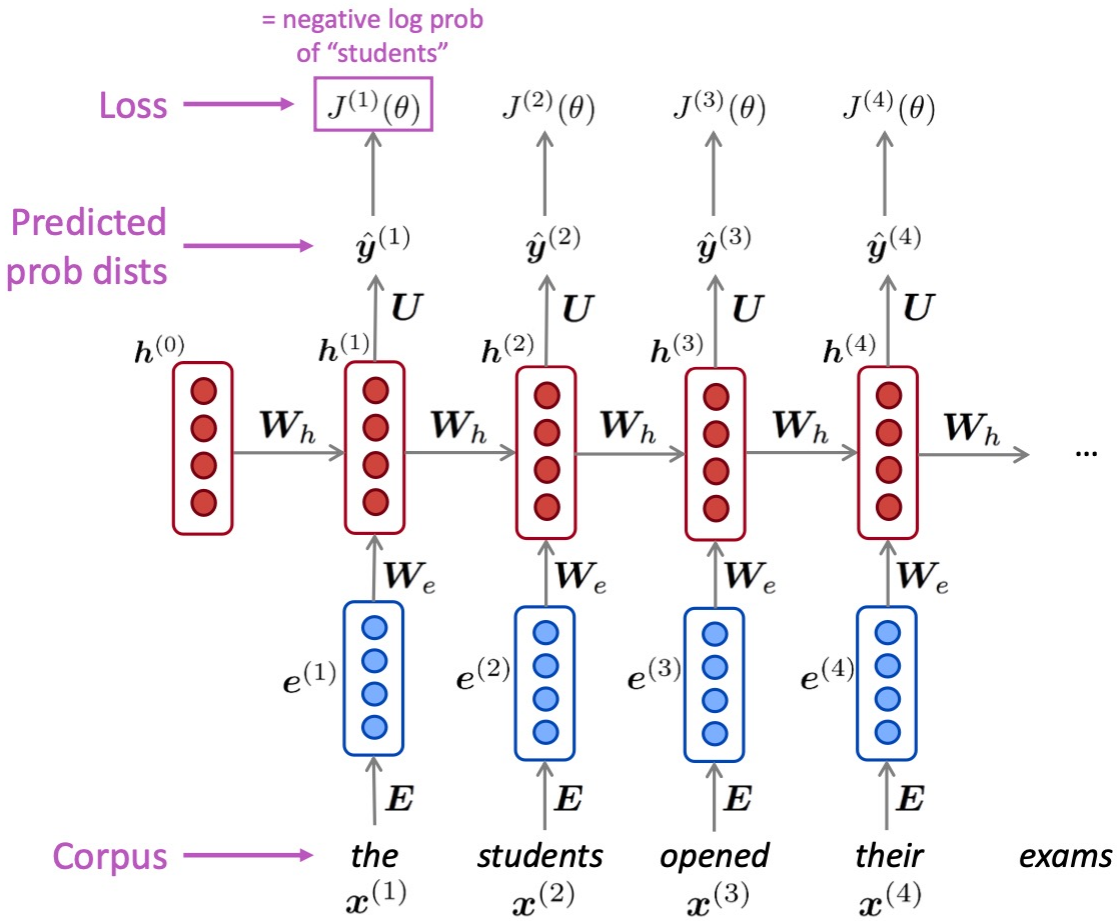
Example: Suppose we are trying to predict the 5th word in the sequence "the dog chased the". We want to know the probability the next word is "cat".

$$P(Y_5 = \text{“}cat\text{”} | \text{“}the\ dog\ chase\ the\text{”}) = \frac{\exp(\text{score in logits for “}cat\text{”})}{\text{normalization term}} = 0.321$$

What are these encoder/decoder things?

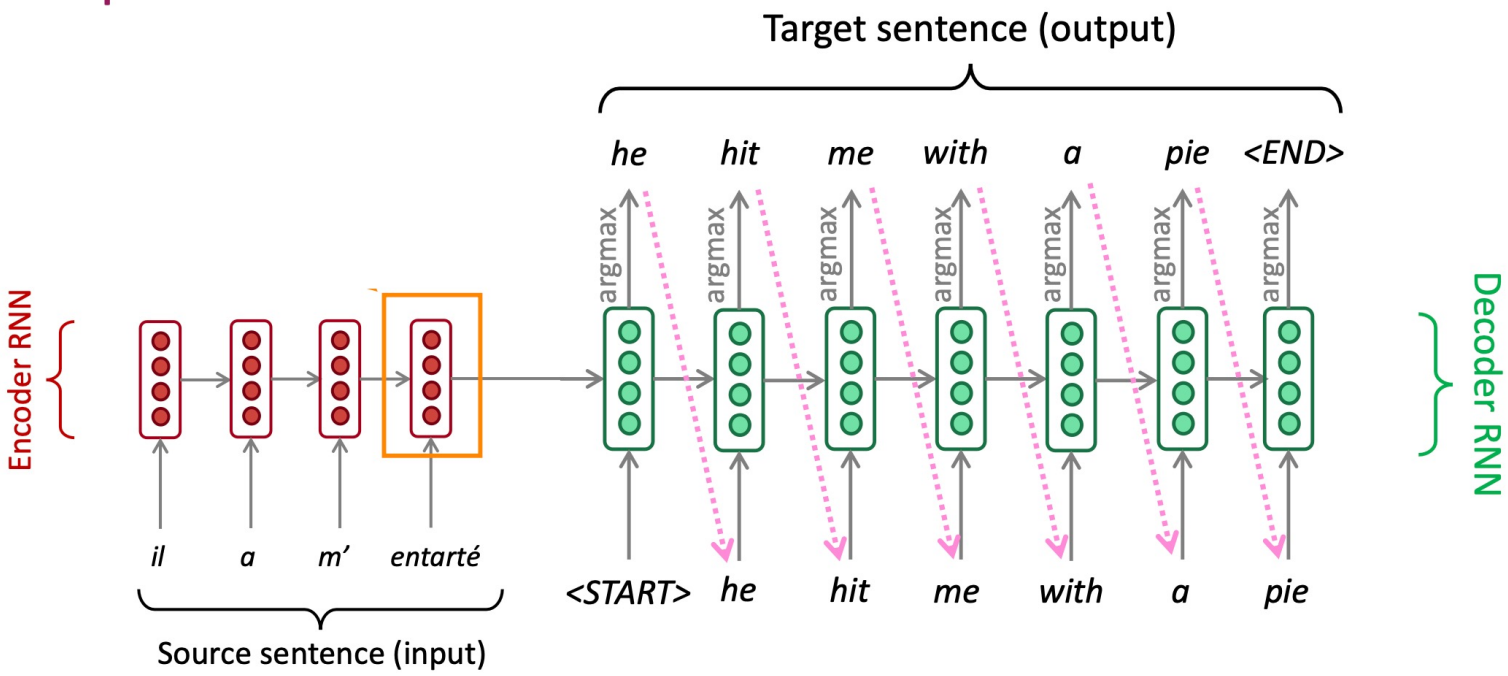


RNN as a LM


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

Seq2Seq models

Seq2Seq with Two RNNs

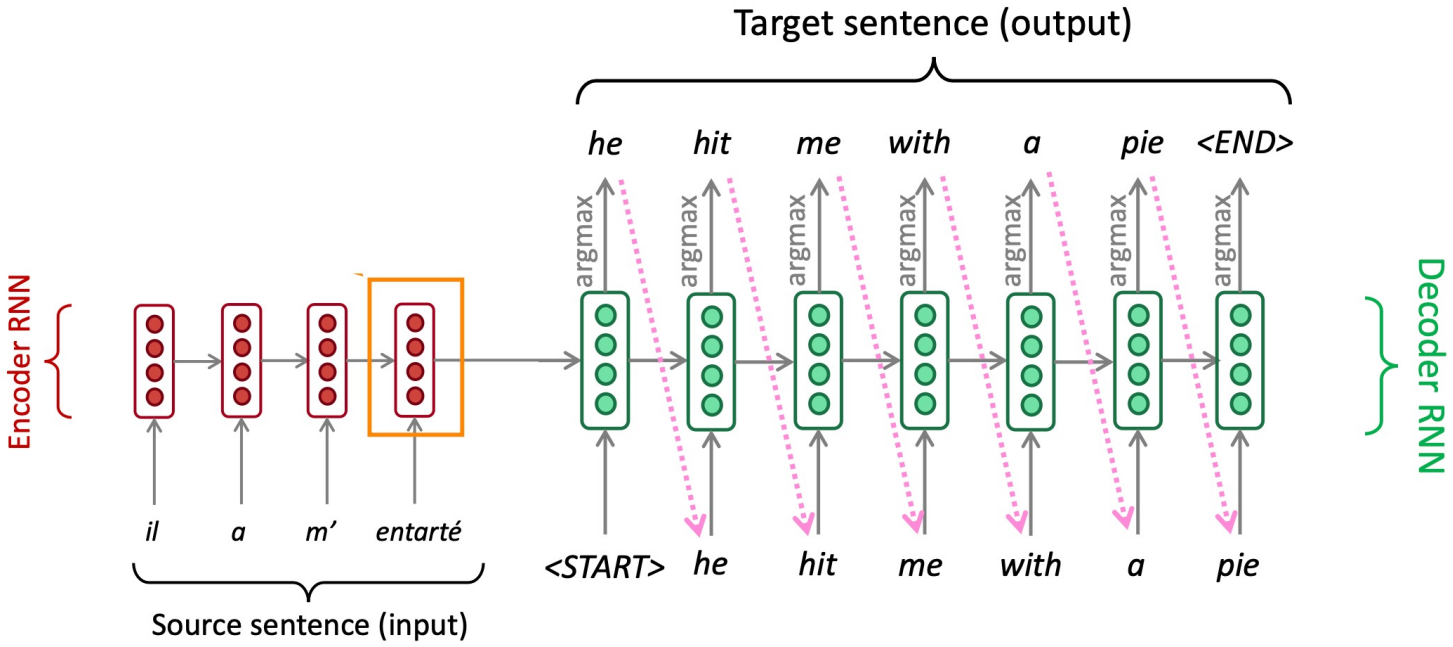


Encoder produces an encoding of the source sentence.

Provides Initial hidden state for Decoder

Decoder RNN is a **Conditional Language Model** that generates target sentence, *conditioned on encoding*.

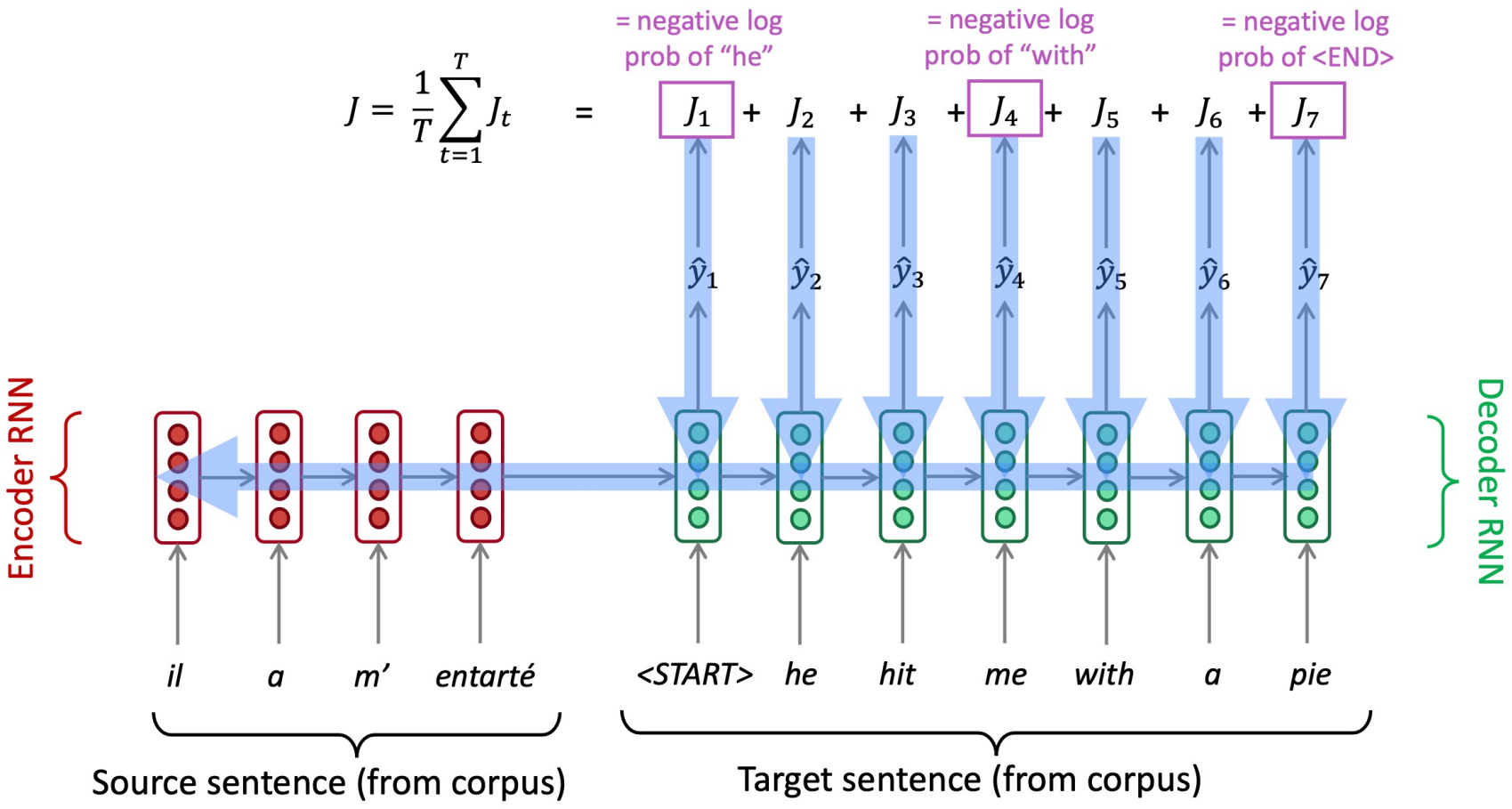
Seq2Seq models



- Directly models the conditional probability p(y|x) of translating a source sequence $x_1, ..., x_n$ to a target sequence $y_1, ..., y_m$.

$$p(y_j|y_{<j}, \mathbf{s}) = softmax(g(h_j))$$

Training a Seq2Seq Model



Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The probability the language model assigns to the true t^{th} word in the target sequence.

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The index of the true t^{th} word in the target sequence.

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \log \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Recall:

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Loss Function: Negative Log Likelihood

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \log \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

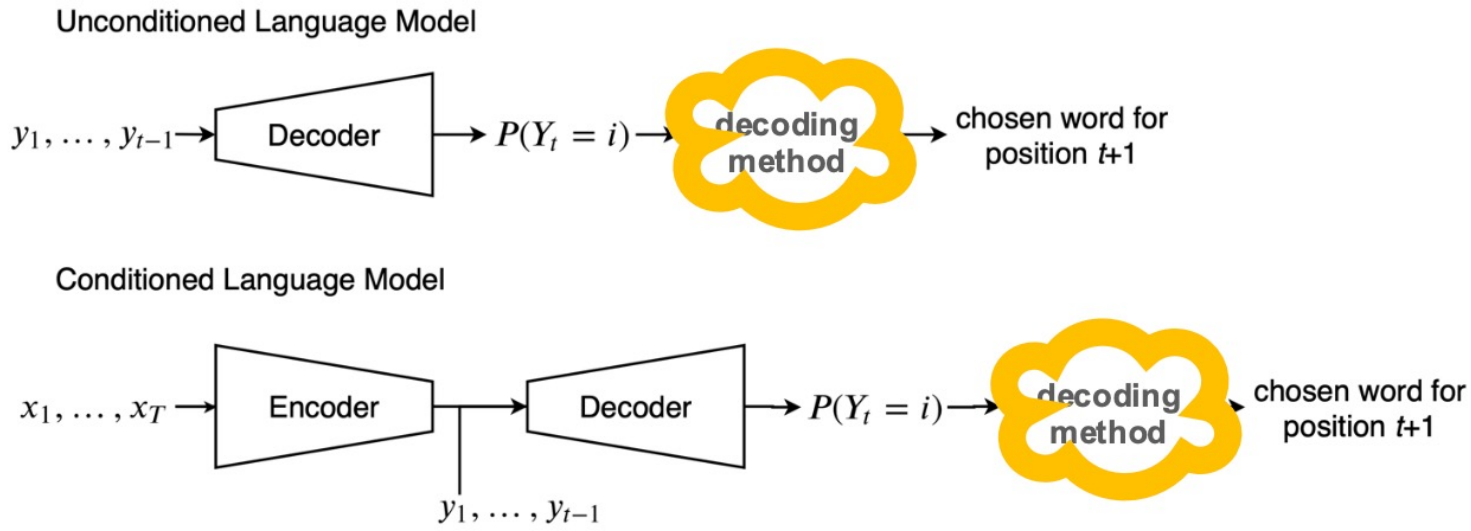
$$= -\sum_{t=1}^{T} \mathbf{E}\hat{\mathbf{y}}_t[i^*]$$

Recall:

$$P(Y_t = i | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Now how do we do generation?

To do generation, we need a **sampling algorithm** that selects a word given the predicted probability distribution $P(Y_t = i|y_{1:t-1})$.



How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\text{argmax}_i P(Y_t = i|\mathbf{y}_{1:t-1})$

TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V} = \{apple, banana, orange, plum\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

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

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

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\text{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V} = \{apple, banana, orange, plum\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

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

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

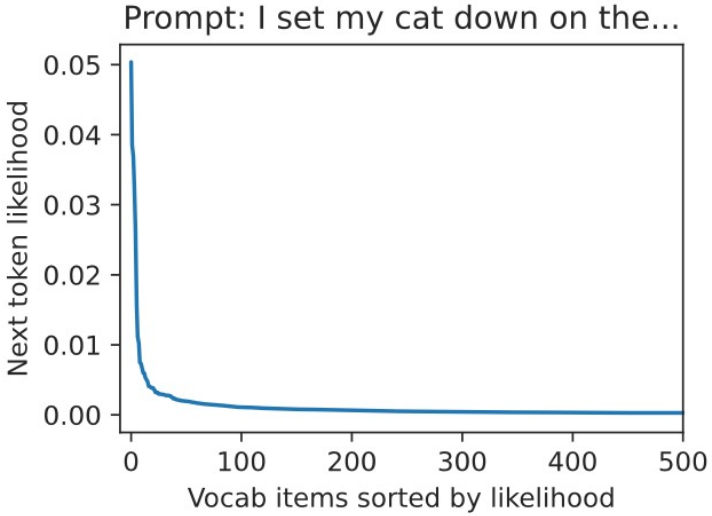
How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\text{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Problem with Random Sampling

Most tokens in the vocabulary get assigned very low probabilities but cumulatively, choosing any one of these low-probability tokens is pretty likely. In the example on the right, there is over a 29% chance of choosing a token $\mathbf{v}$ with $\mathbf{P(Y_t = v) \leq 0.01}$.



How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\text{argmax}_i P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

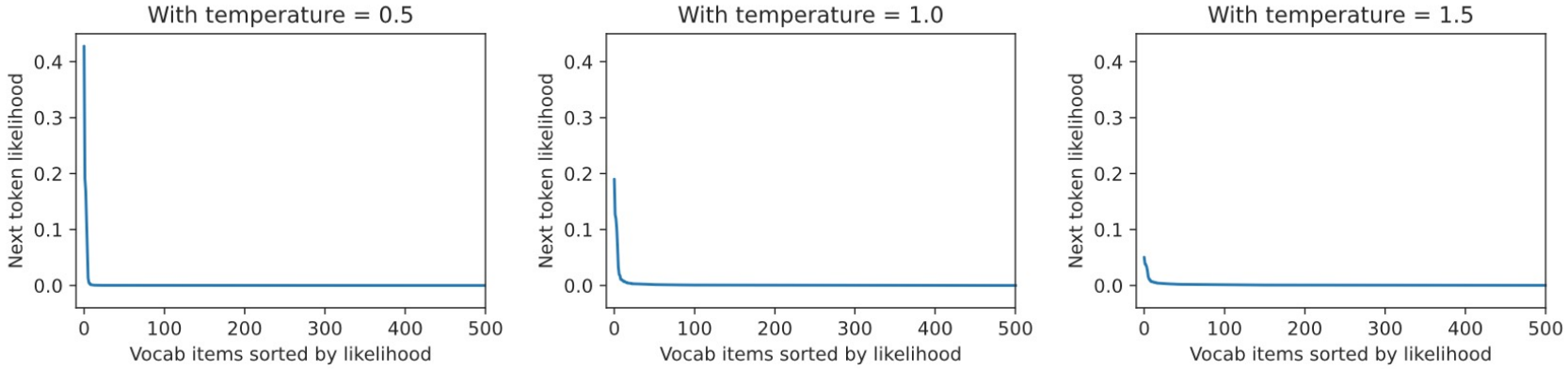
$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\text{argmax}_i P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

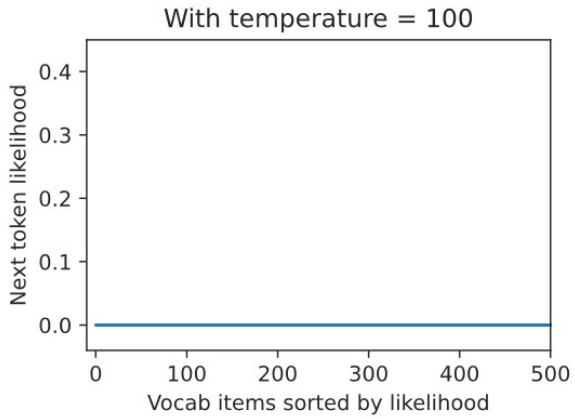


How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\underset{i}{\text{argmax}} P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.



TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V} = \{\text{apple}, \text{banana}, \text{orange}, \text{plum}\}$

We have primed our LM with "apple apple" and want to generate the next word in the sequence.

Our language model predicts:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

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%

How can we sample from $P(Y_t = i|\mathbf{y}_{1:t-1})$?

Option 1: Take $\text{argmax}_i P(Y_t = i|\mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

As T approaches 0, random sampling with temperature looks more and more like argmax.

TYPE YOUR ANSWER INTO CHAT

Suppose our vocab consists of 4 words:
$\mathcal{V}$ = {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:
$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$
$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$
$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$
$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $\text{T} = 0.00001$**?**

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

Beam search decoding

- Assumption: the best possible sequence to generate is the one with highest overall sequence likelihood (according to the model).

- It is computationally intractable to search all possible sequences for the most likely one, so instead we use beam search.

- Beam search is a search algorithm that approximates finding the overall most likely sequence to generate.

Beam search decoding

- Core idea: On each step of decoder, keep track of the *k* most probable partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10)

- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

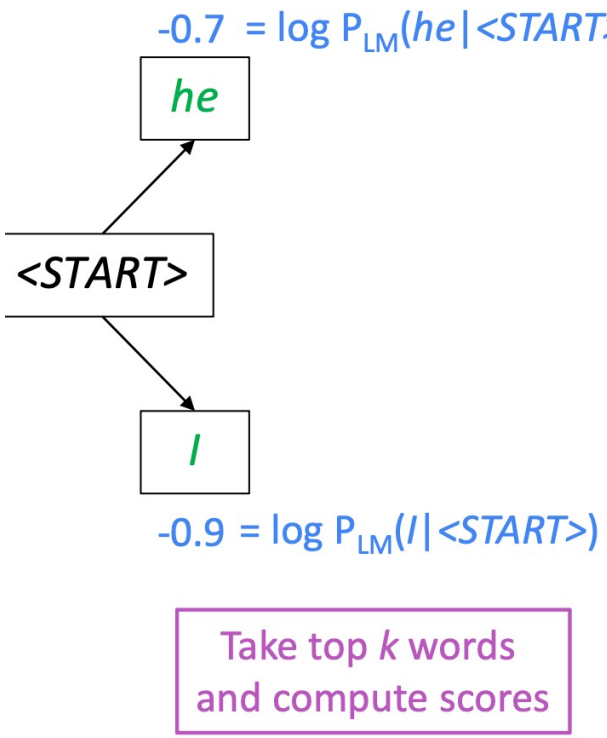
$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step

- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

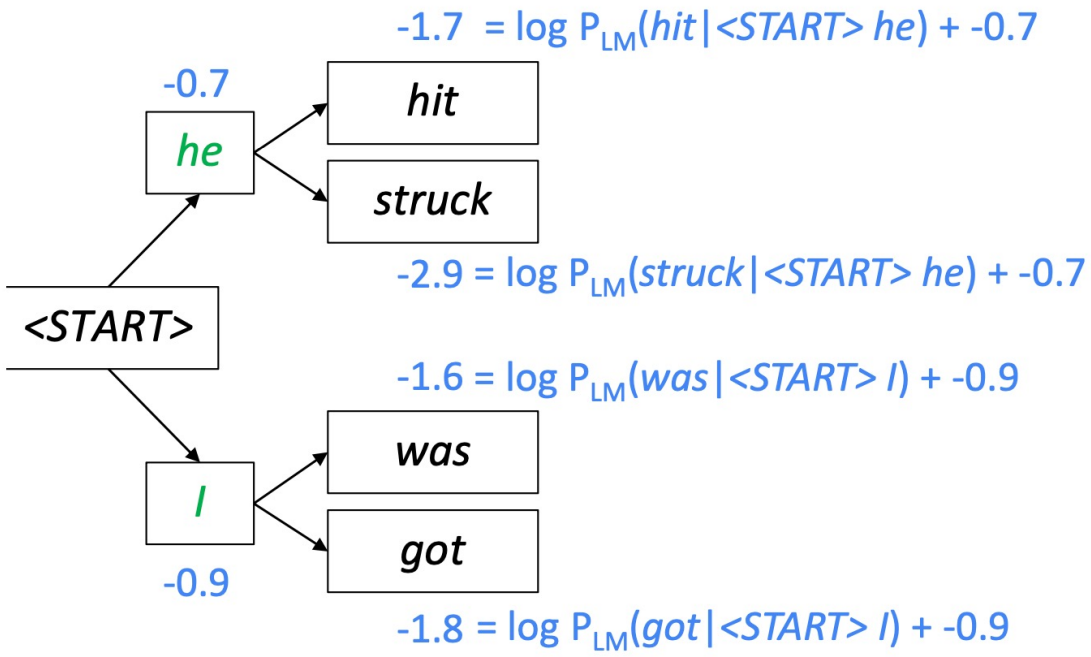
Beam search decoding

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Beam search decoding

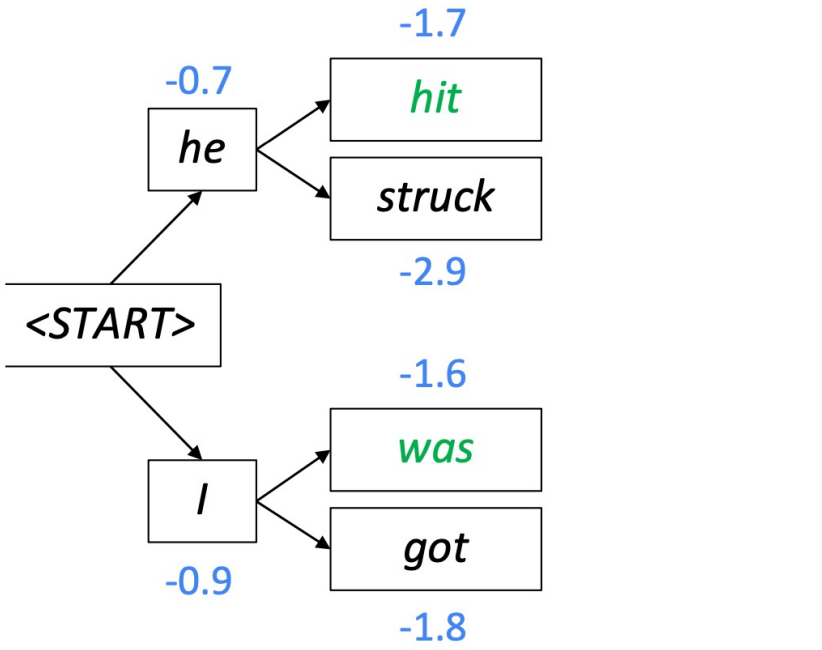
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding

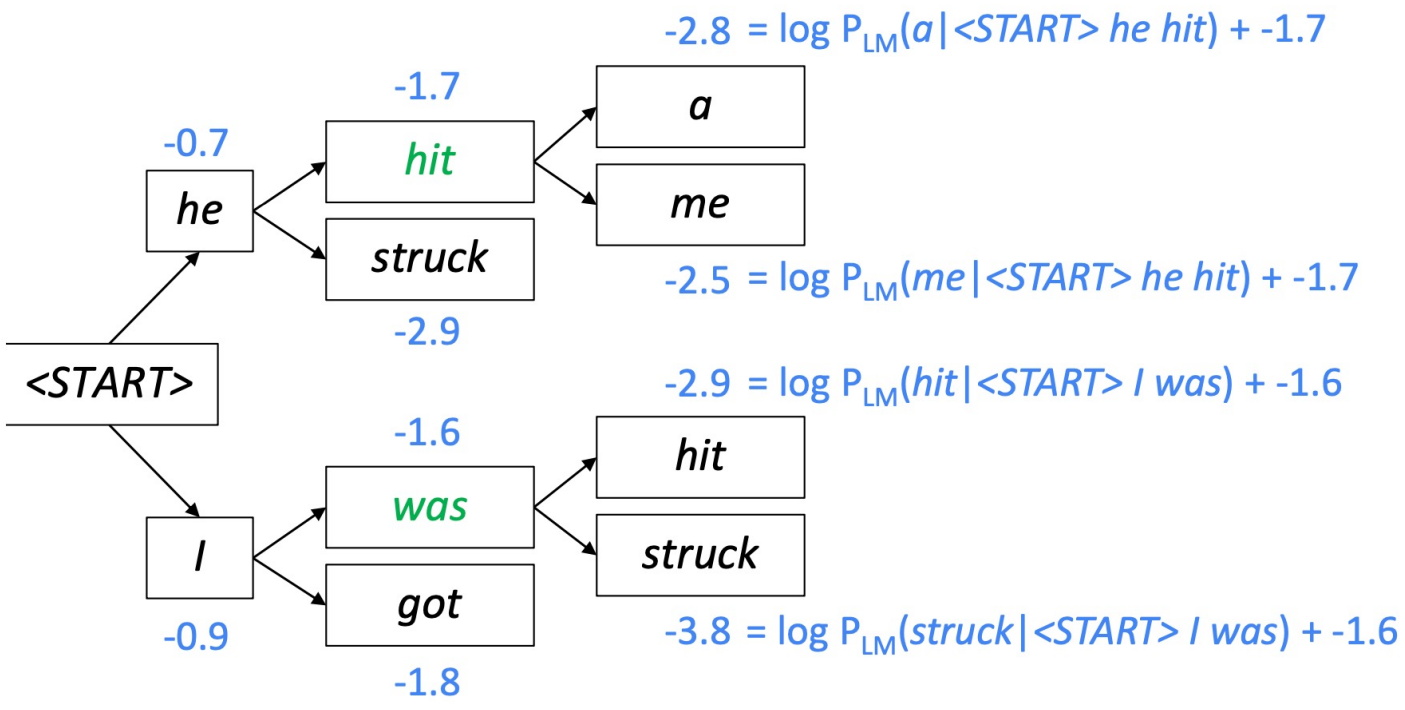
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding

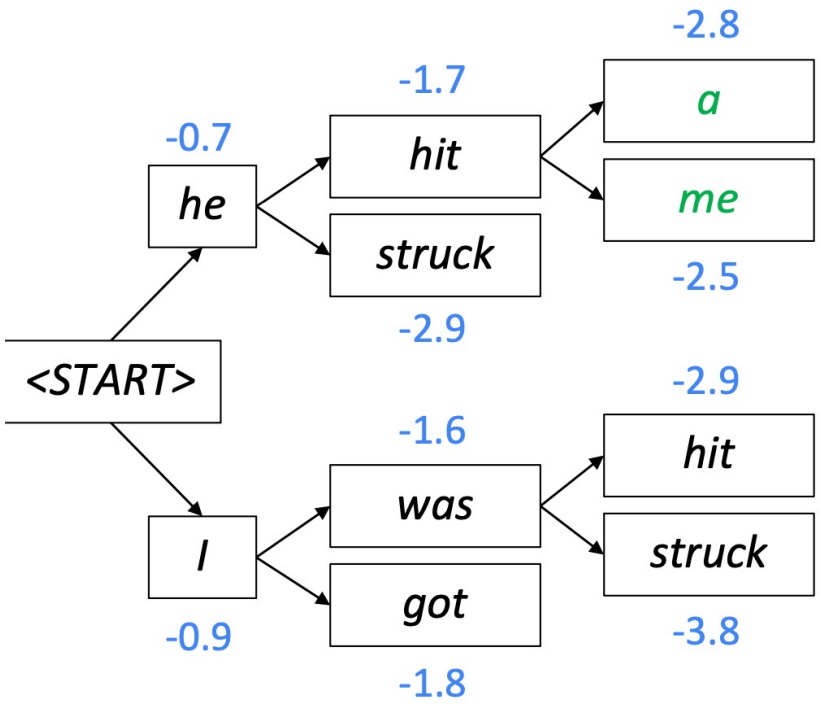
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding

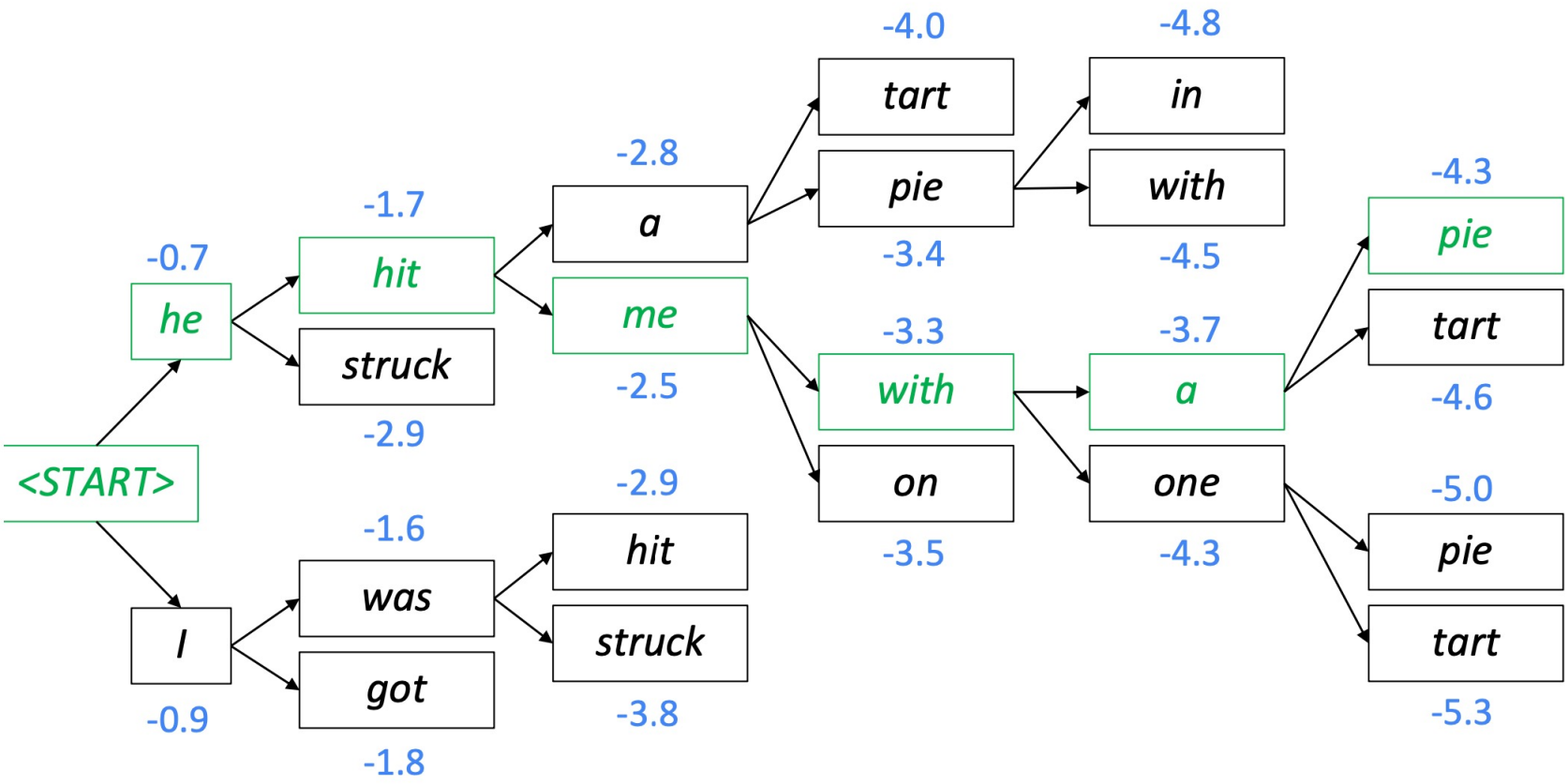
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses, just keep k with highest scores

Beam search decoding

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

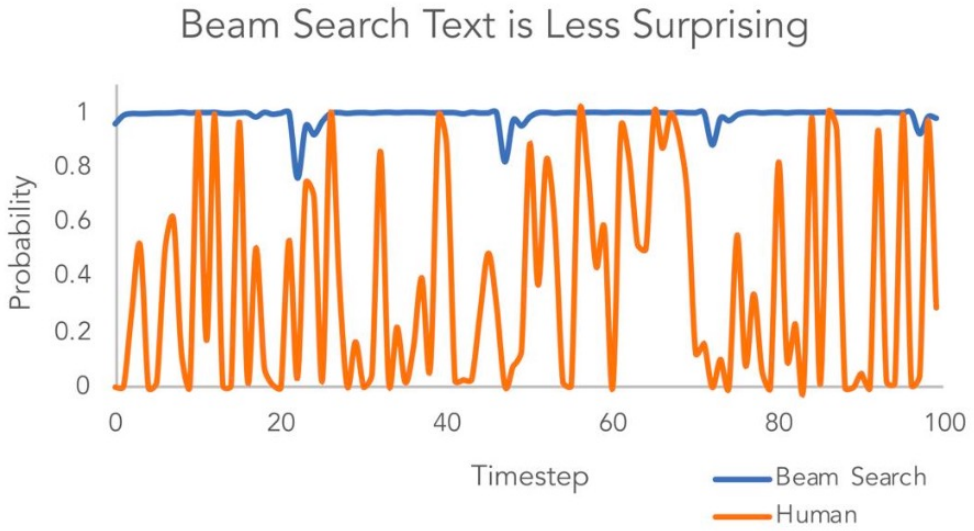
Beam search decoding

- In greedy decoding, usually we decode until the model produces a <END> token
  - For example: *<START> he hit me with a pie <END>*
- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
  - We reach timestep *T* (where *T* is some pre-defined cutoff), or
  - We have at least *n* completed hypotheses (where *n* is pre-defined cutoff)

Problems with Beam Search

It turns out for open-ended tasks like dialog or story generation, optimizing for the sequence with the highest possible $P(x_1, \ldots, x_T)$ isn't actually a great idea.

- Beam search generates text that is much for likely than human-written text



When to Use Beam Search

- Your task is very narrow, i.e., there is only ~1 “correct” sequence your model should generate.
  - Examples: question answering, machine translation
- You want to score possible generation with several signals of goodness, besides just model likelyhoods.


Q&A