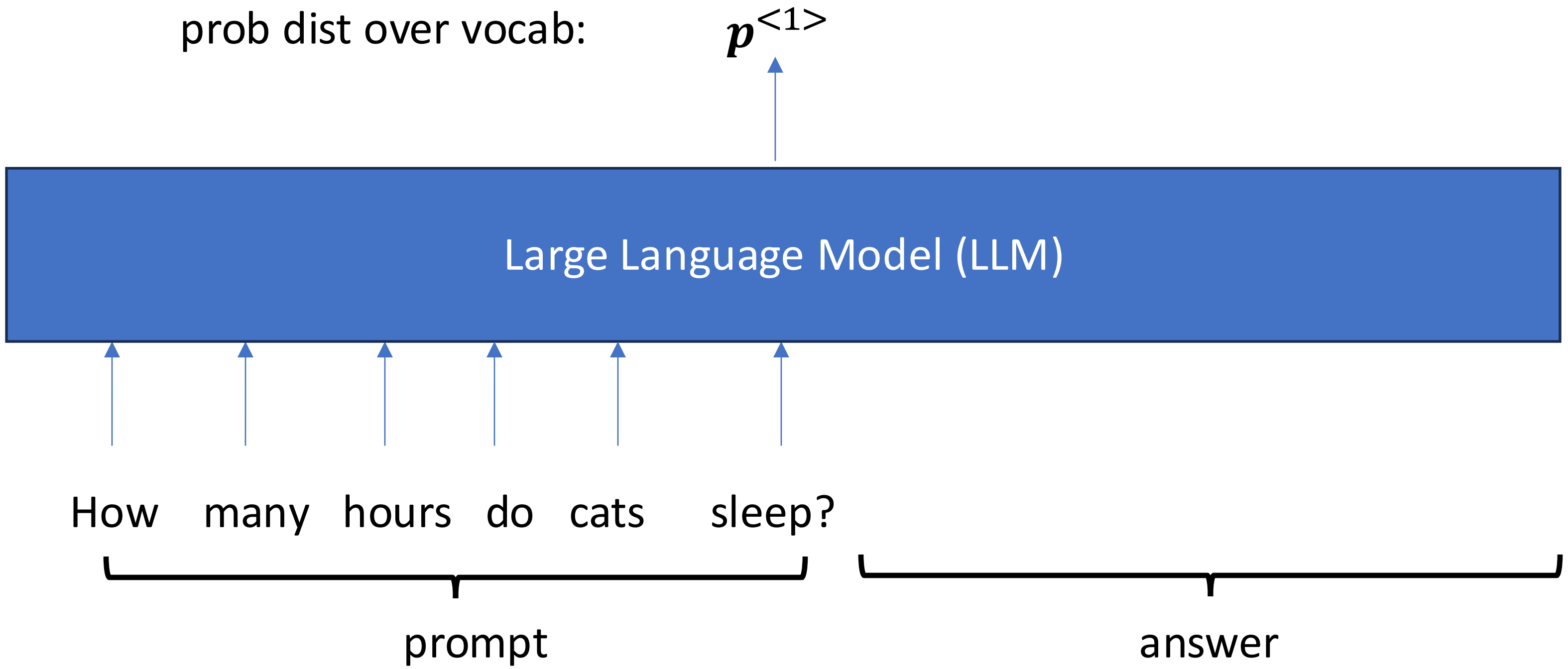
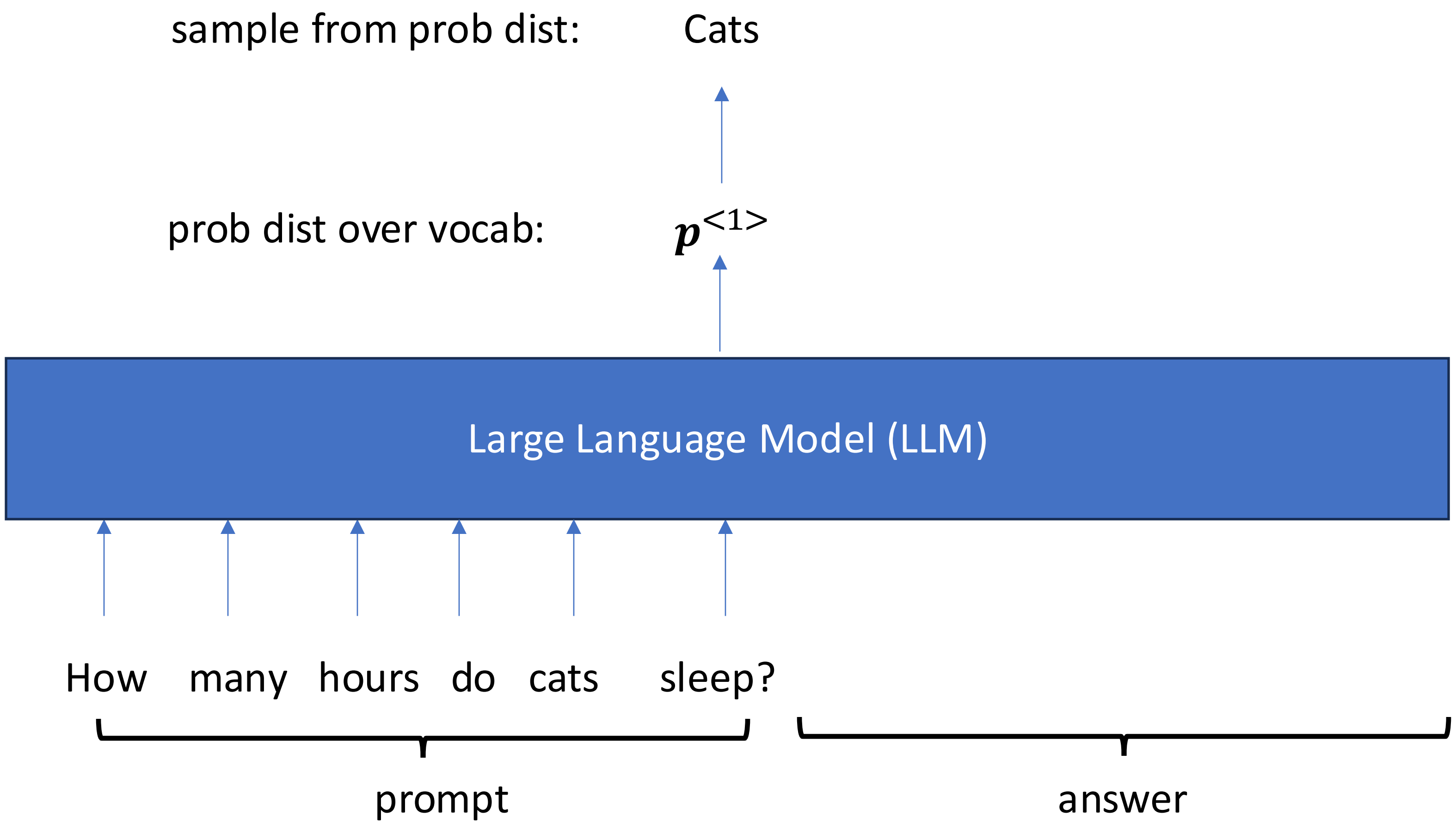
Language Model

- Let’s initially focus on generating a sentence.
- Large corpus of sentences used to train the language model.
- Once trained, during inference, the language model will generate *random* sentences reflecting the corpus.
- It will generate sentences contained and not contained in corpus.
- More common/likely sentences are generated with higher probability.
- Model can also provide the probability of any sentence.

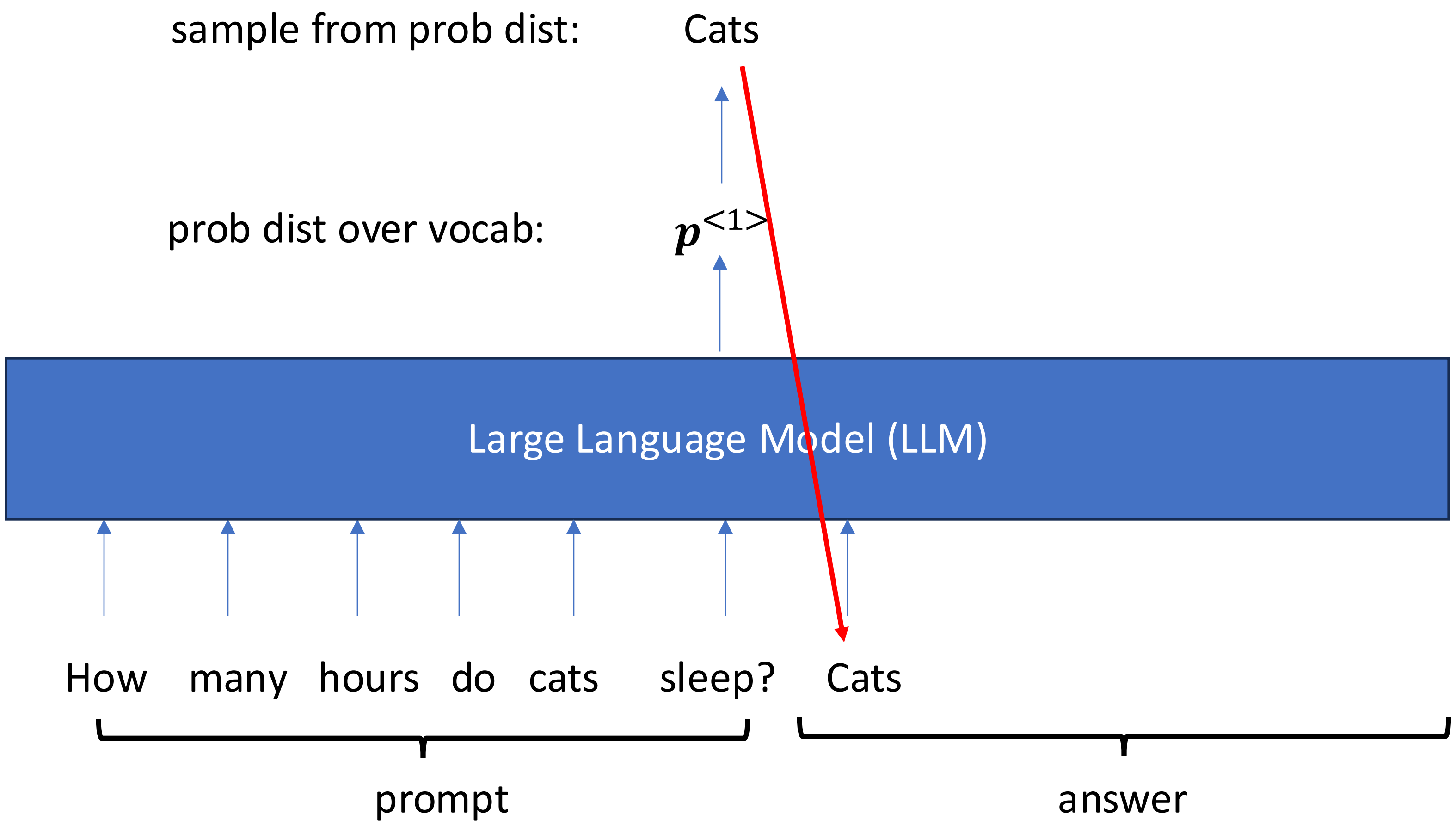
High-Level: Language Model at Inference



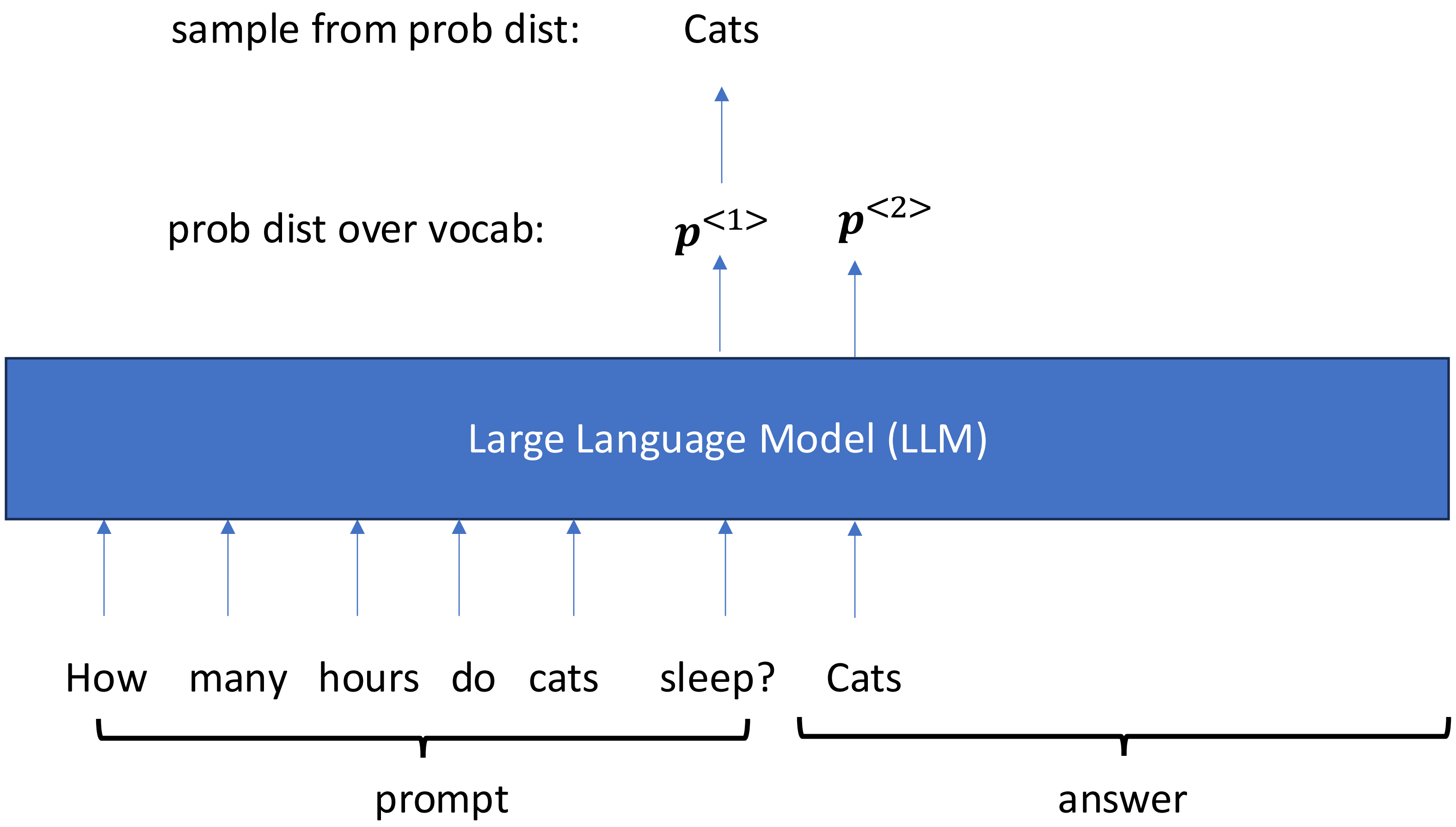
High-Level: Language Model at Inference



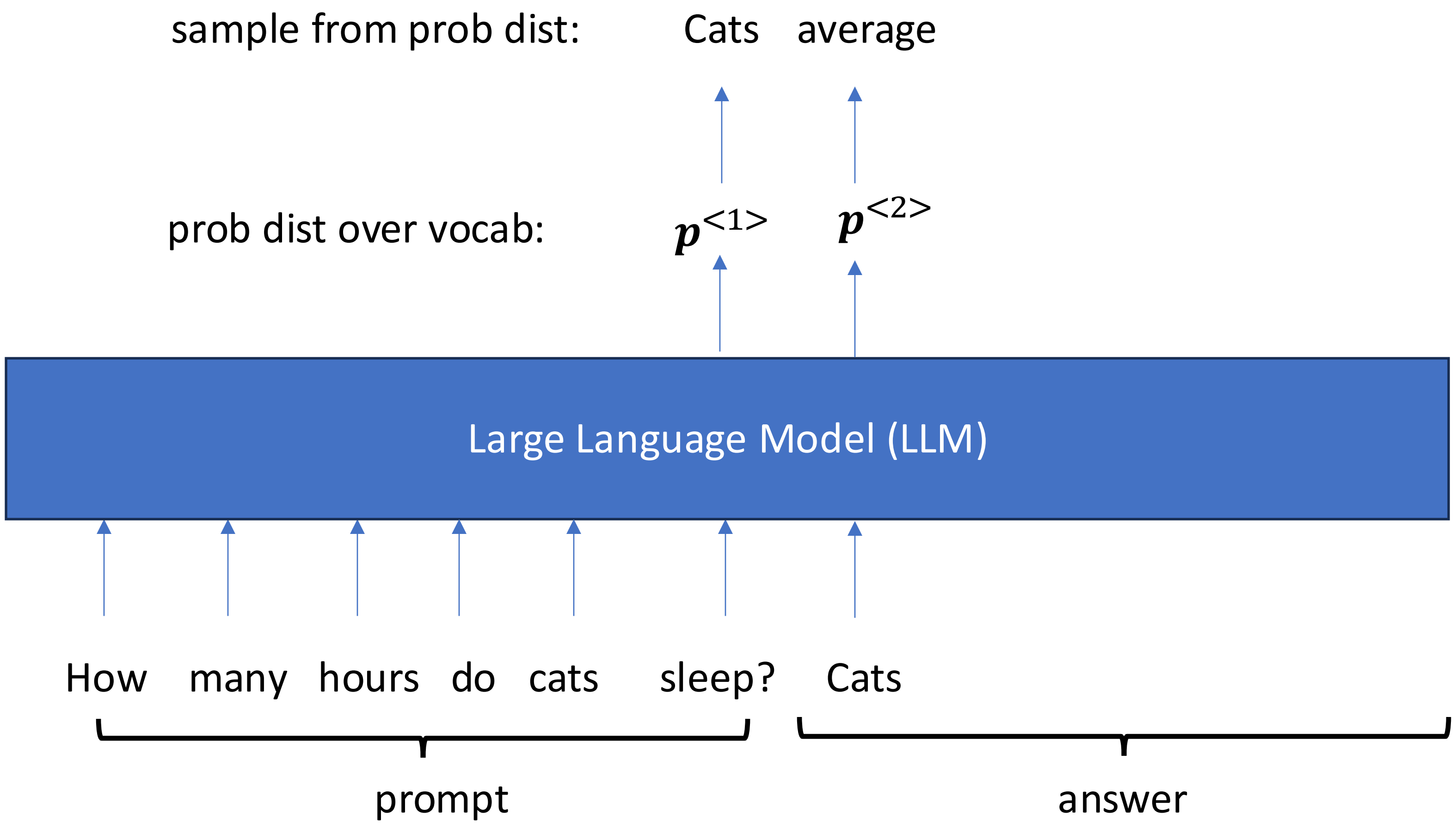
High-Level: Language Model at Inference



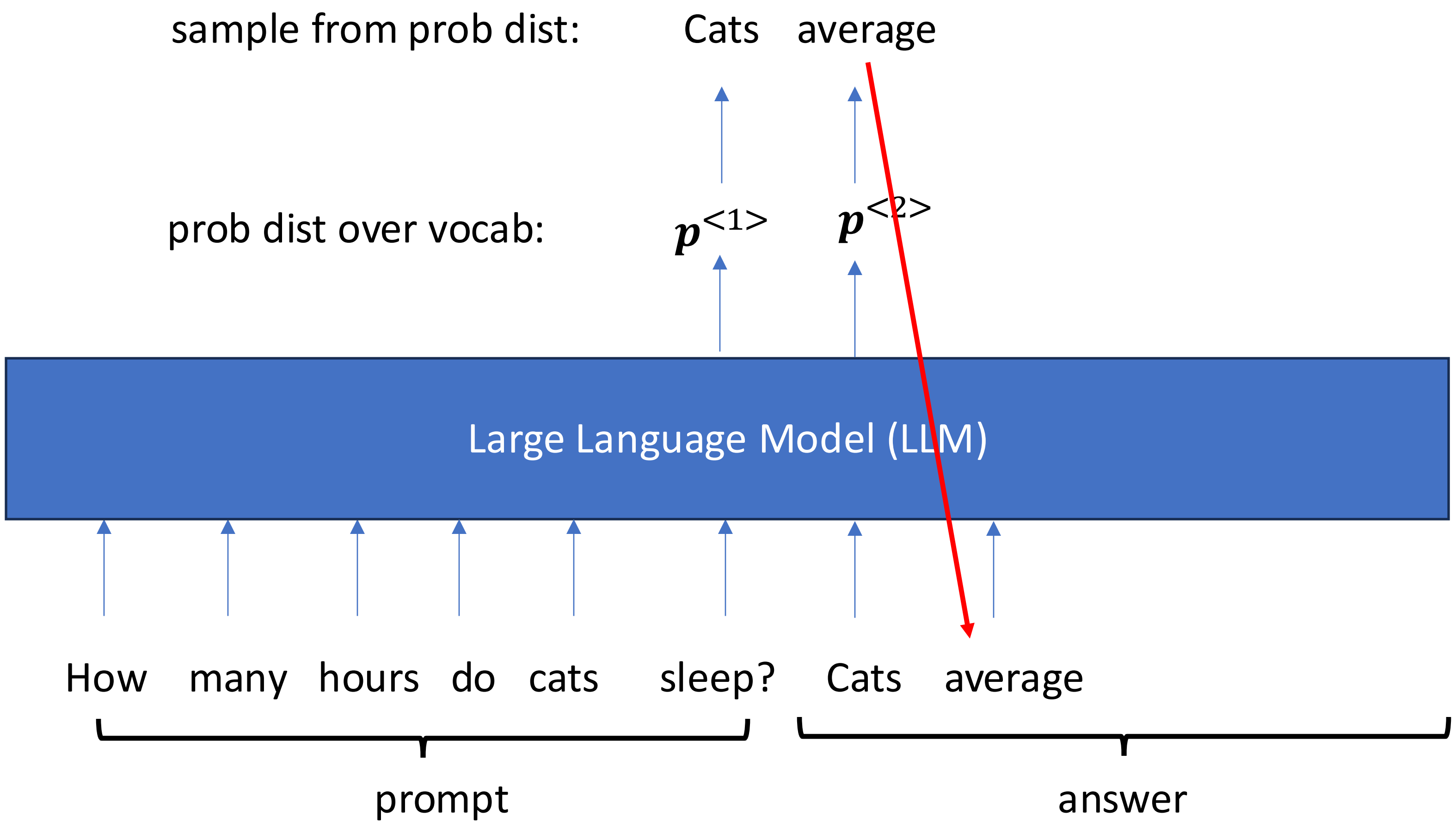
High-Level: Language Model at Inference



High-Level: Language Model at Inference



High-Level: Language Model at Inference



High-Level: LLM training

- Begin with huge corpus of data (Deepseek V3: 15 trillion tokens)
- Feed corpus into LLM

Next word prediction:

- The dog jumped over the cat.
- Ground truth label is “over”.
- LLM provides predicted probabilities over all possible words including P(over), P(under), P(in), P(love).
- Roughly, LLM parameters are optimized to maximize P(ground truth) over all tokens in corpus. [In this case, maximize P(over)].

Example use of LM from Speech Recognition

- Speech recognition outputs “The apple and pair/pear salad”
- Need to automatically choose between pair and pear.
- Recall trained language model can provide the probability of any sentence:
- P(The apple and pair salad) = P(The)P(apple)P(and)P(pair)P(salad)
- So can use trained language model to calculate
  - P(The apple and pair salad) = 3.3×10^{-13}
  - P(The apple and pear salad) = 5.7×10^{-10}

→ Choose pear

Let’s get more into the details now

Sequences and tokenization notation

Harry Potter is a fictional character in children books . <EOS>

$\mathbf{x} =$ $x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$ $x^{<6>}$ $x^{<7>}$ $x^{<8>}$ $x^{<9>}$ $x^{<10>}$ $x^{<11>}$

<EOS> is end of sentence token; $x^{<0>}$ is always start of sentence token

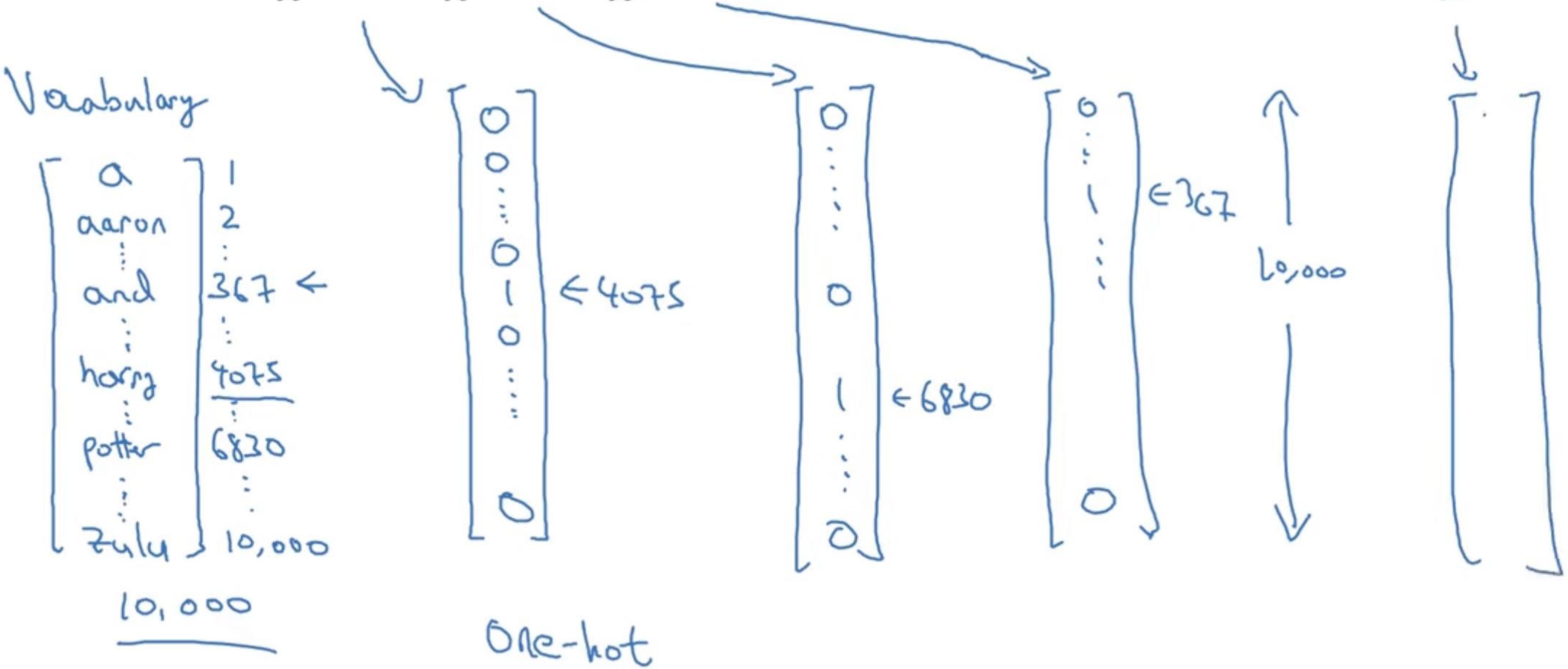
When dealing with multiple sentences, i=1,..,m, write: $\mathbf{x}^{(i)}$ for the i^{th} sentence; write $x^{(i)<3>}$ for 3^{rd} token in i^{th} sentence.

Denote |V| for the size of the tokenized vocabulary, including special tokens

One-hot vector representations of words

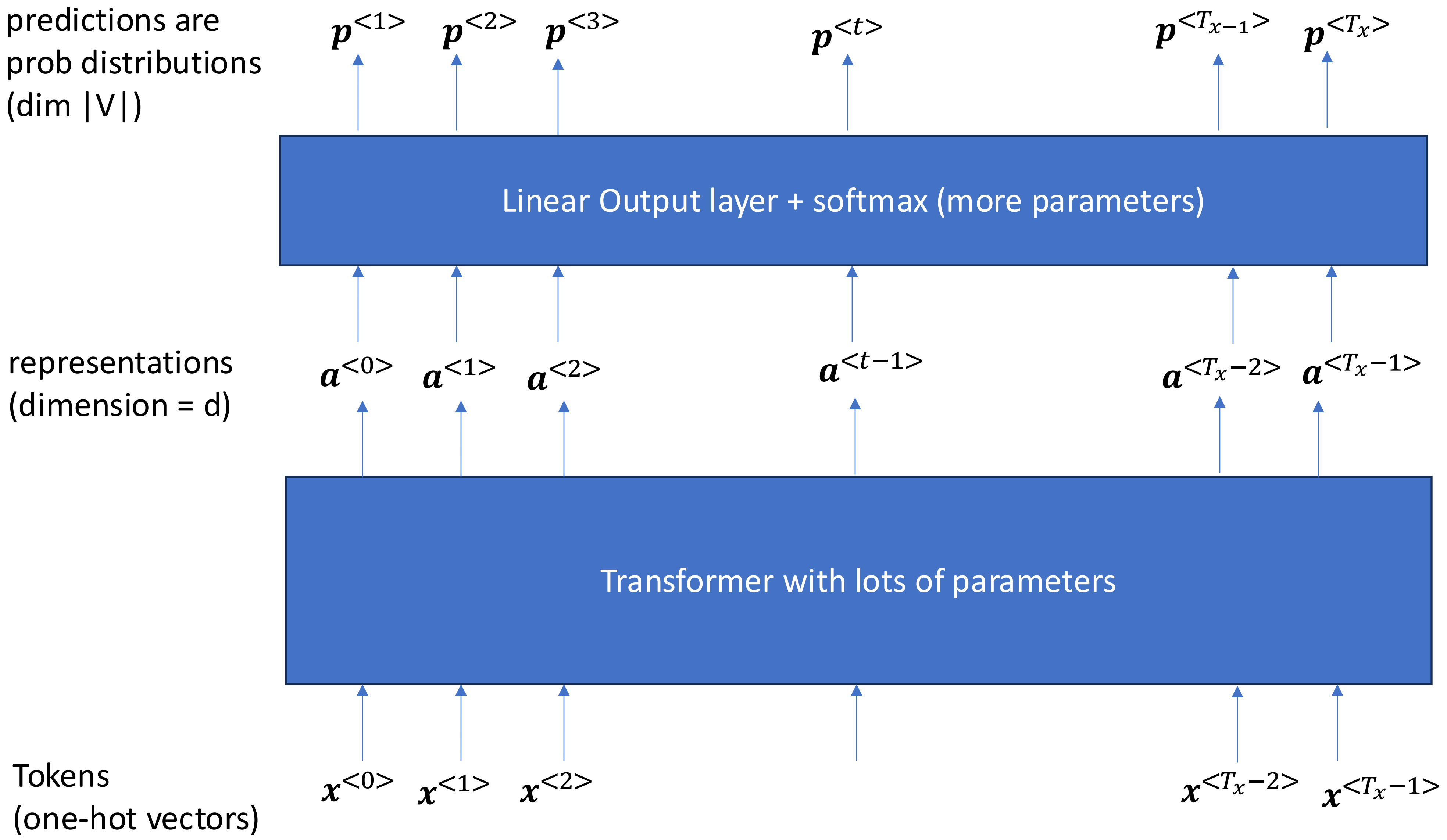
x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ … $x^{<7>}$ $x^{<9>}$



Andrew Ng

General set up for language models



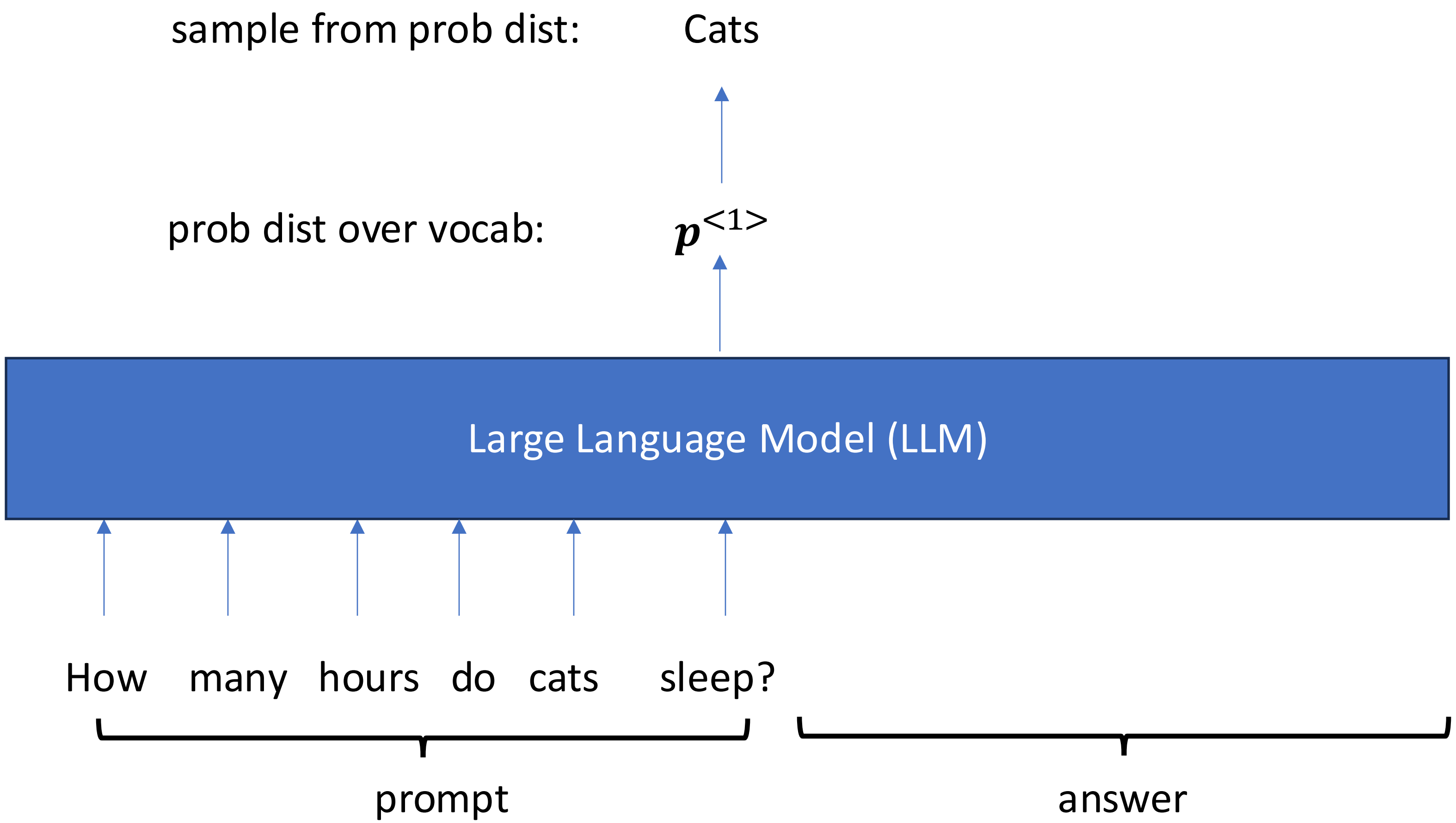
Linear output layer: from $\mathbf{a}^{<t-1>}$ to $\mathbf{p}^{<t>}$

- Transformer transforms $\mathbf{x}^{<0>}, \mathbf{x}^{<1>},\ldots,\mathbf{x}^{<T-1>}$ to $\mathbf{a}^{<0>}, \mathbf{a}^{<1>},\ldots,\mathbf{a}^{<T-1>}$. Will discuss transformer later.
- How do we go from representation vector (aka feature vector) $\mathbf{a}^{<t-1>}$ of dimension d to probability distribution $\mathbf{p}^{<t>}$ of dimension |V|?
- First: linear transformation: $\boldsymbol{z}^{<t>} = \mathbf{W}\boldsymbol{a}^{<t-1>} + \mathbf{b}$, where **W** is a |V|xd matrix and **b** is |V| dimensional. **W** and **b** are trainable (along with the parameters in the transformer).
- Second: apply softmax to $\boldsymbol{z}^{<t>}$:

$$p_i^{<t>} = \exp(z_i^{<t>}) / [\exp(z_1^{<t>}) + \ldots + \exp(z_{|V|}^{<t>})], \quad i = 1,\ldots,|V|$$

- $\mathbf{p}^{<t>} = (p_1^{<t>}, p_2^{<t>},\ldots, p_{|V|}^{<t>})$

Sampling: from $p^{<t>}$ to $x^{<t>}$



Sampling: from $\mathbf{p}^{<t>}$ to $\mathbf{x}^{<t>}$

- LLMs sample from distributions during inference.
- Suppose you have a distribution $\mathbf{p} = (p_1,\ldots,p_{|V|})$ over $|V|$ possible outcomes.
- How can you sample from it? Each sample should take on an integer value between 1 and $|V|$.
  - Generate uniformly distributed u from [0,1].
  - If $0 \leq u < p_1$, output 1; if $p_1 \leq u < p_1+p_2$, output 2, if $p_1+p_2 \leq u < p_1+p_2+p_3$, output 3, and so on.
  - Of course, Python functions do this for you. Don't need to implement it.

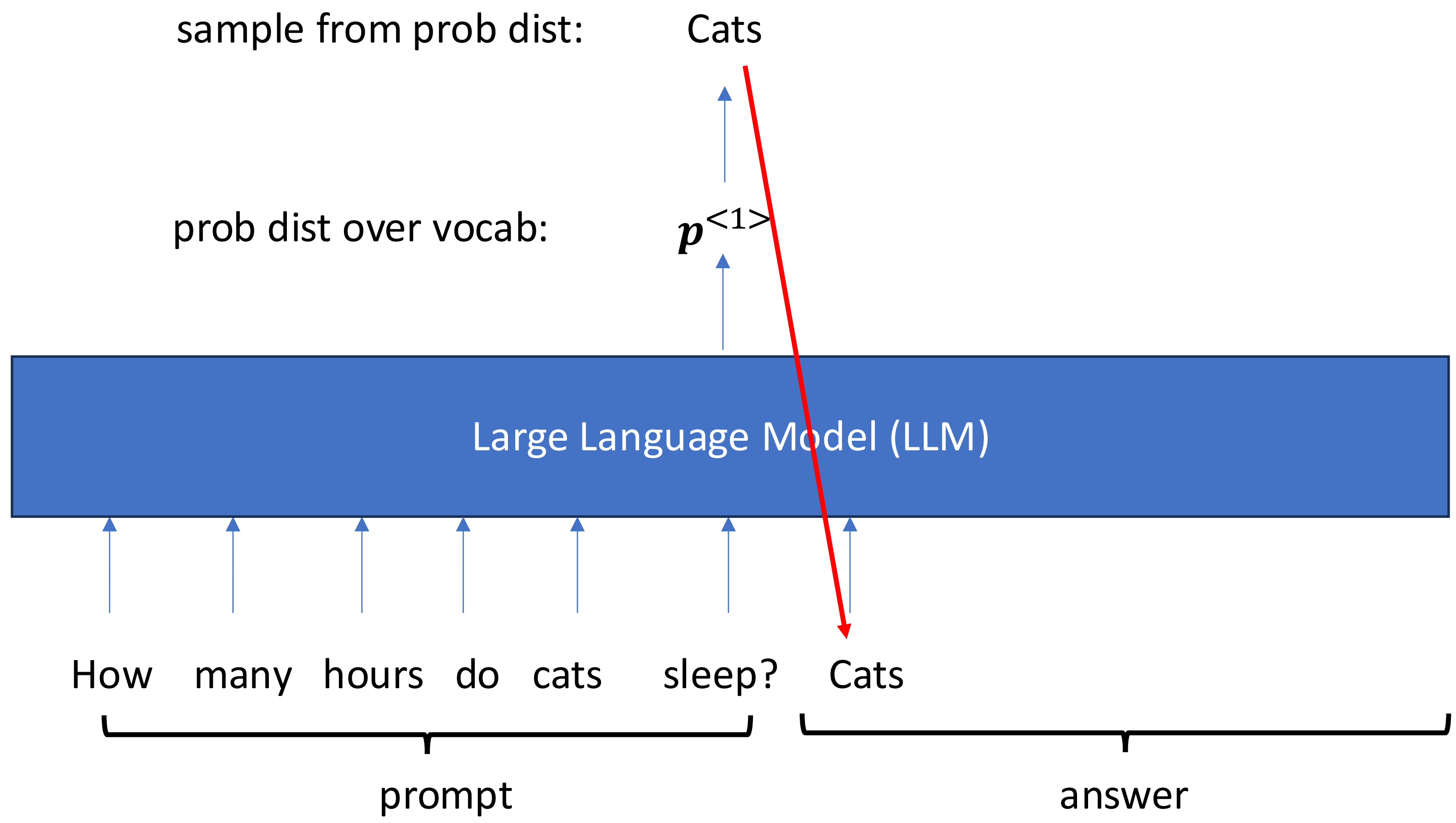
LM Inference: How to generate a sentence from a trained language model

- Suppose we have already generated tokens $\boldsymbol{x}^{<1>}, \boldsymbol{x}^{<2>}, \ldots, \boldsymbol{x}^{<t-1>}$
- Input $\boldsymbol{x}^{<0>}, \boldsymbol{x}^{<1>}, \ldots, \boldsymbol{x}^{<t-1>}$ into the transformer, obtain $\boldsymbol{a}^{<t-1>}$
- Obtain $\mathbf{p}^{<t>}$: from $\boldsymbol{a}^{<t-1>}$ as in previous slides (linear transformation then softmax).

- Sample a value k from dist $(p_1^{<t>}, p_2^{<t>}, \ldots, p_{|V|}^{<t>})$ as in previous slide
- Set $\boldsymbol{x}^{<t>}$ = k

- Then repeat the entire process until we sample <EOS>

High-Level: Language Model at Inference



Inference Example

- Initialize with $a^{<0>} = \mathbf{0}$ vector; thus $\boldsymbol{z}^{<1>} = \mathbf{W}\boldsymbol{a}^{<0>} + \mathbf{b} = \mathbf{b}$
- Apply softmax to $\boldsymbol{z}^{<1>} = \mathbf{b} = [b_1,\ldots,b_{|V|}]^T$ to get $(p_1^{<1>}, p_2^{<1>}, \ldots, p_{|V|}^{<1>})$
- Sample a token from $(p_1^{<1>}, p_2^{<1>}, \ldots, p_{|V|}^{<1>})$. Suppose sample $\boldsymbol{x}^{<1>}$ = "Cats"
- Feed $\boldsymbol{x}^{<1>}$ = "Cats" (one-hot vector) into transformer; obtain $\boldsymbol{a}^{<1>}$
- Calculate $\boldsymbol{z}^{<2>} = \mathbf{W}\boldsymbol{a}^{<1>} + \mathbf{b}$
- Obtain $(p_1^{<2>}, p_2^{<2>}, \ldots, p_{|V|}^{<2>})$ by applying softmax to $\boldsymbol{z}^{<2>}$
- Sample token from $(p_1^{<2>}, p_2^{<2>}, \ldots, p_{|V|}^{<2>})$. Suppose sample $\boldsymbol{x}^{<2>}$ = "average"
- Feed $\boldsymbol{x}^{<1>}$ = "Cats" , $\boldsymbol{x}^{<2>}$ = "average" into transformer; obtain $\boldsymbol{a}^{<2>}$
- Calculate $\boldsymbol{z}^{<3>} = \mathbf{W}\boldsymbol{a}^{<2>} + \mathbf{b}$
- Keep on going, maybe getting:

  "Cats average 15 hours of sleep a day. <EOS>"
- Note: sentence is random. Try again and maybe get "Cats sleep at night <EOS>"

Training an LM: Self Supervised Learning

- Consider a sentence $\mathbf{x} = (x^{<0>}, x^{<1>},\ldots, x^{<T>})$ from the corpus.
- This sentence consists of T prefixes:
  $[x^{<0>}]$, $[x^{<0>}, x^{<1>}]$, $[x^{<0>}, x^{<1>}, x^{<2>}]$, …, $[x^{<0>}, x^{<1>},\ldots, x^{<T-1>}]$
- We will consider each prefix as an example for supervised learning
- The label of a prefix is the next token.
- That is, label for the prefix $(x^{<1>}, x^{<2>},\ldots, x^{<t-1>})$ is $y^{<t>} = x^{<t>}$
- Each sentence **x** gives rise to T labeled examples
  $((x^{<0>}, x^{<1>},\ldots, x^{<t-1>}), y^{<t>})$, where $y^{<t>} = x^{<t>}$
- “Self supervised” because we do not need to explicitly label data
- Question: **x** = “I love music <EOS>”. What are the four labeled examples we obtain from this sentence?

Training a LLM: Classification problem

- Given a corpus of sentences, convert corpus into labeled examples using all the prefixes:

  $((x^{(i)<0>}, x^{(i)<1>},\ldots, x^{(i)<t-1>}) , y^{(i)<t>}),\ \ t= 0,..,T_x, i=1,\ldots,m$
- Supervised learning: For each prefix, we want to predict the label.
- Label can take values in {1,2,…,|V|}
- So we have a classification problem with |V| classes.
- Directly apply cross-entropy loss to this supervised learning problem.

- Note: we don’t actually have to explicitly convert corpus into labeled examples. More later.

Training on a sentence x = $(x^{<1>}, x^{<2>}, \ldots, x^{<Tx>})$

- Suppose sentence from corpus is "I love music <EOS>"
- We input the <u>entire</u> sentence into transformer
- As with inference, model will give us the probability distributions over the next word for all prefixes: $p_\theta(.)$, $p_\theta(.|\text{"I"})$, $p_\theta(.|\text{"I love"})$, $p_\theta(.|\text{"I love music"})$:
  - Obtain $\boldsymbol{a}^{<t-1>}$ from prefix $x^{<1>}, x^{<2>}, \ldots, x^{<t-1>}$
  - Obtain $\boldsymbol{z}^{<t>} = \mathbf{W}\boldsymbol{a}^{<t-1>} + \mathbf{b}$
  - Obtain $p_\theta^{<t>}$ from $\boldsymbol{z}^{<t>}$ for all t=0,…,T-1
- Cross-entropy loss for this sentence:

$$L(\theta) = -[\log p_\theta(\text{"I"}) + \log p_\theta(\text{"love"}|\text{"I"}) + \log p_\theta(\text{"music"}|\text{"I love"}) + \log p_\theta(\text{"<EOS>"}|\text{"I love music"})\ ]$$

- Quiz: How do we get $p_\theta(\text{"music"}|\text{"I love"})$? What is θ?
- We need to do this for all sentences in corpus
- Cross entropy loss: find model parameters that maximize the probability of the data (sentences in corpus).

Training algorithm: Gradient descent

- Pick a sentence **x** from corpus. (Or a mini-batch of sentences)
- Tokenize sentence: $x^{<0>}$, $x^{<1>}$,…, $x^{<T>}$
- Calculate $L(\theta) = -\log$ (probability of sentence) $= \sum_{t=1}^{T} L^{<t>}(p^{<t>}, y^{<t>}, \theta)$, where $L^{<t>}(p^{<t>}, y^{<t>}, \theta) = -\log P_\theta(y^{<t>} \mid x^{<1>}, x^{<2>}, \dots, x^{<t-1>})$ where $y^{<t>} = x^{<t>}$
- Use backpropagation to obtain $\nabla_\theta L(\theta)$
- Update θ using gradient descent: $\theta = \theta - \alpha \nabla_\theta L(\theta)$

Language Model Quiz

- Suppose 5% of the sentences in the corpus begin with “A”, and 0.5% begin with “A big”.
- After training model, we would expect P_θ(”A”) $\approx$?
- We would expect P_θ(”big|”A”) $\approx$?
- Suppose the sentence “I love to eat houses <EOS>” is not in the corpus. Will P_θ(”I love to eat houses <EOS>”) = 0 ?
- Suppose “I love music <EOS>” appears in the corpus 10 times, and “I love poetry <EOS>” 5 times. How should the model probabilities of the two sentences compare?

Language Models: Summary

- (Self) Supervised Learning
- During training we obtain the probability that model predicts correctly the next word in the sentence for each prefix in the sentence
- Training aims to maximize these probabilities over all the prefixes in the corpus.
- During inference, we generate (sample) one word at a time: each new word to be generated takes as input all the previously generated words. (So outputs become the inputs.)
- Generative AI: Generated sentences reflect the content in the corpus.