Moduel IV – Large Language Models

By

Dr.Sunil Kumar S

Three architectures for language models

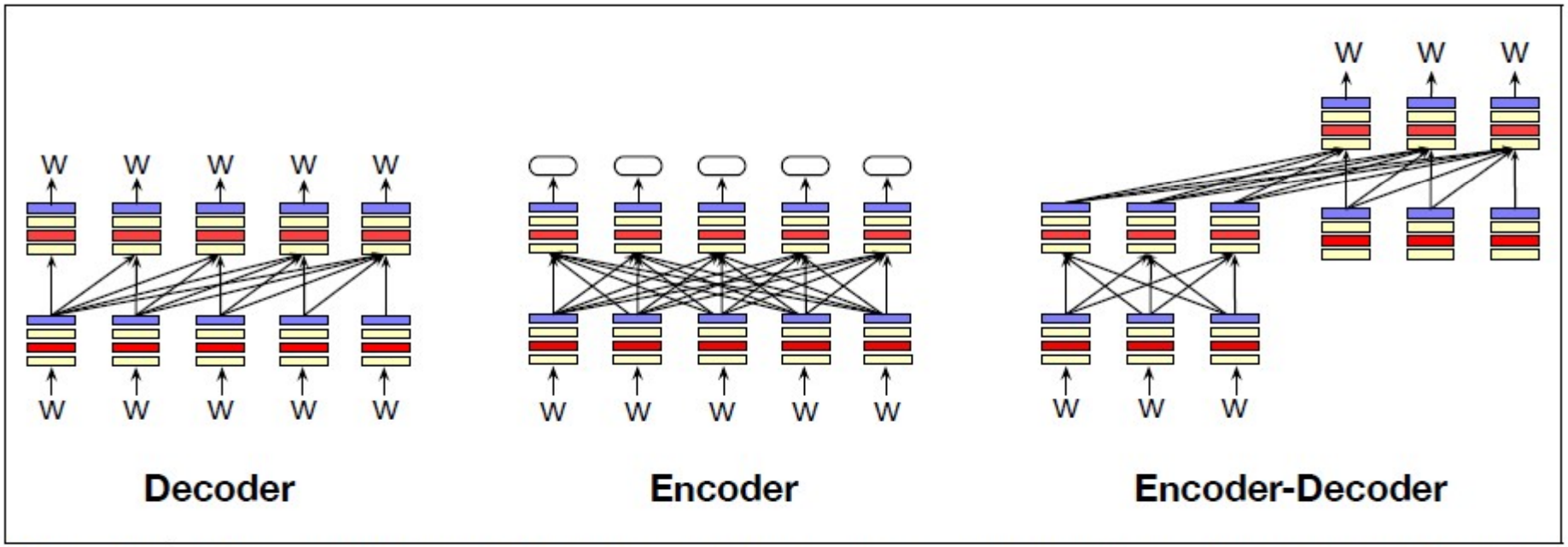

Figure 7.3 Three architectures for language models: decoders, encoders, and encoder-decoders. The arrows sketch out the information flow in the three architectures. Decoders take tokens as input and generate tokens as output. Encoders take tokens as input and produce an encoding (a vector representation of each token) as output. Encoder-decoders take tokens as input and generate a series of tokens as output.

Encoder Architecture

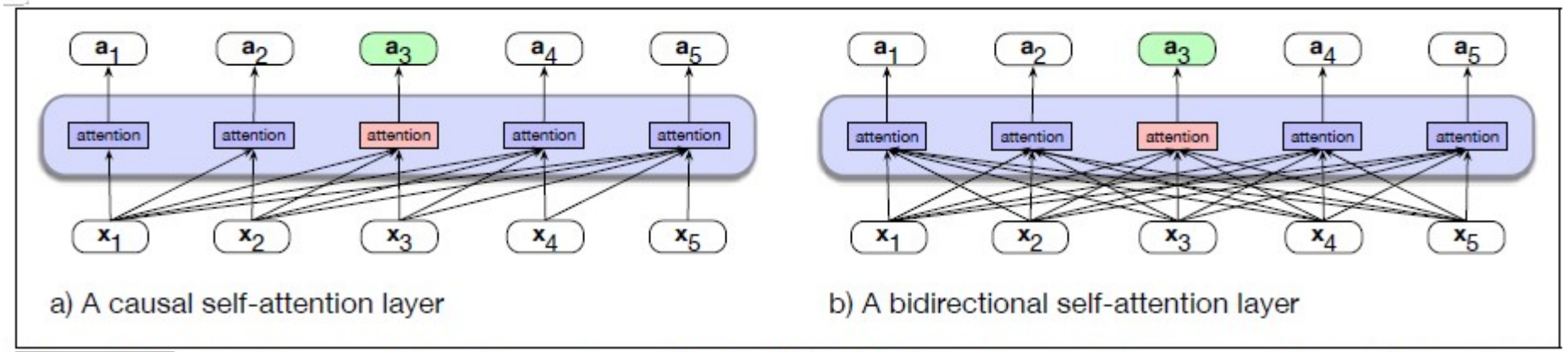

Figure 9.1 (a) The causal transformer from Chapter 8, highlighting the attention computation at token 3. The attention value at each token is computed using only information seen earlier in the context. (b) Information flow in a bidirectional attention model. In processing each token, the model attends to all inputs, both before and after the current one. So attention for token 3 can draw on information from following tokens.

Bidirectional

$$\textbf{head} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\right)\mathbf{V} \tag{9.1}$$

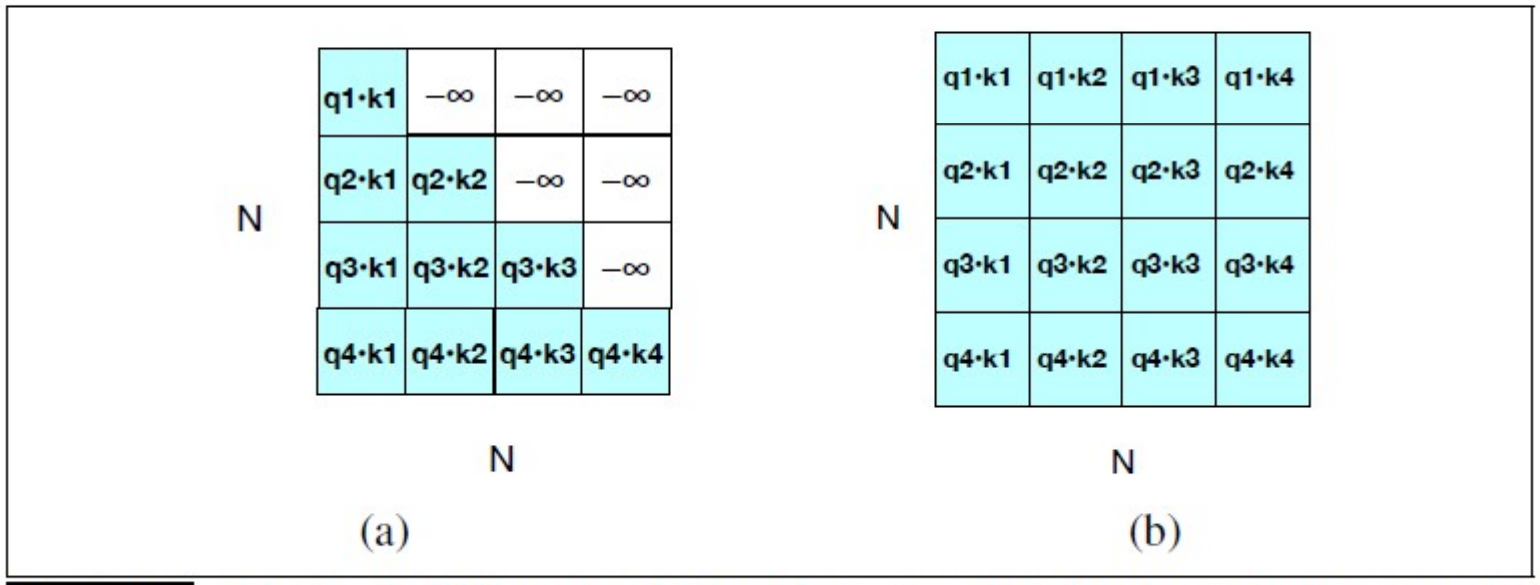

Figure 9.2 The $N \times N$ $\mathbf{QK}^\intercal$ matrix showing the $q_i \cdot k_j$ values. (a) shows the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero), while (b) shows the unmasked version.

encoder

- The encoder takes as input a sequence of tokens and outputs a vector representation for each tokens.
- Encoders are usually masked language models, meaning they are trained by masking out a word, and learning to predict it by looking at surrounding words on both sides.
- Masked language models like BERT, RoBERTA, and others in the BERT family are encoder models.
- Encoder models are not generative models; they aren't used to generate text.
- Instead encoder models are often used to create classifiers, for example where the input is text and the output is a label,
- For example for sentiment or topic or other classes.
- This is done by finetuning them (training them on supervised data).

Decoder

- It takes as input a series of tokens, and iteratively generates an output token one at a time.
- The decoder is the architecture used to create large language models like GPT, Claude, Llama, and Mistral.
- The information flow in decoders goes left-to-right, meaning that the model predicts the next word only from the prior words.
- Decoders are generative models,meaning that, given input tokens, they generate novel output tokens

Encoder-Decoder

- The encoder-decoder takes as input a sequence of tokens and outputs a series Decoder of tokens.
- What makes it different than the decoder-only models, is that an encoderdecoder has a much looser relationship between the input tokens and the output tokens, and they are used to map between different kinds of tokens.
- That is, in an encoder-decoder, the output tokens might be very different token-set or be much longer or shorter than the input tokens.
- For example encoder-decoder architectures are used for machine translation, where the input tokens are in one language and and the output tokens are in another language, and probably a different length than the input.
- Encoder-decoder architectures are also used for speech recognition, where the input is tokens representing speech, and the output is tokens representing text.

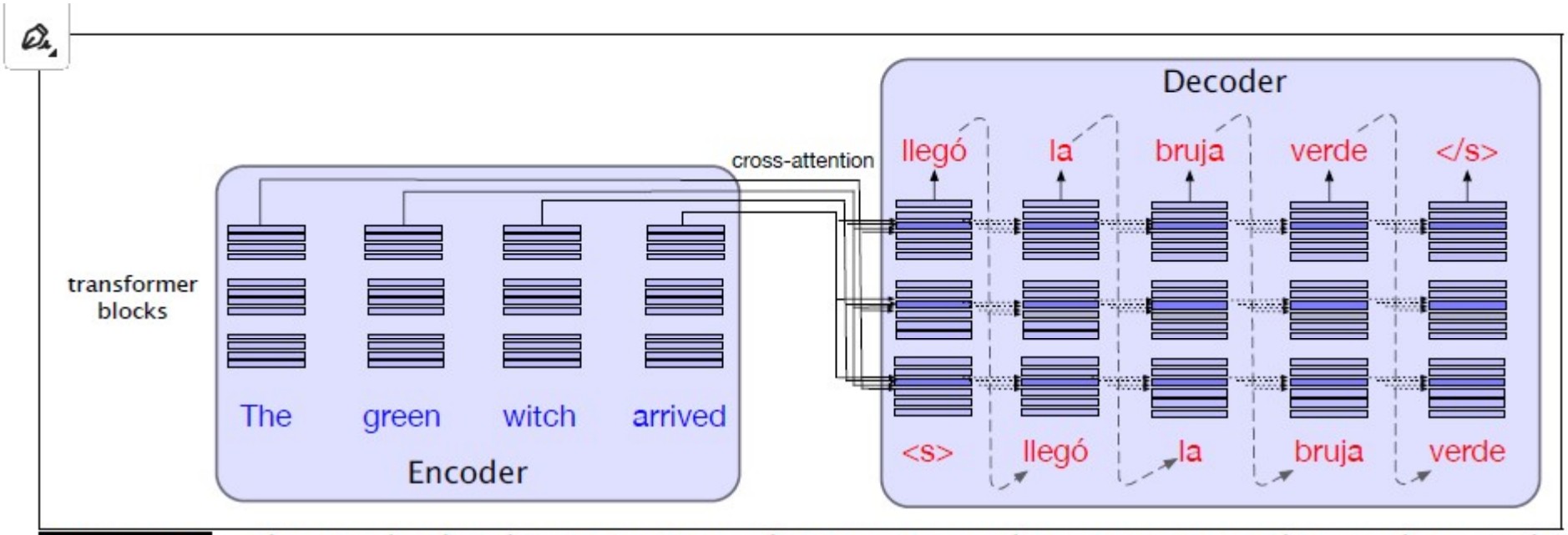

Figure 12.5 The encoder-decoder transformer architecture for machine translation. The encoder uses the transformer blocks we saw in Chapter 8, while the decoder uses a more powerful block with an extra **cross-attention** layer that can attend to all the encoder words. We'll see this in more detail in the next section.

Conditional Generation of Text: The Intuition

- A fundamental intuition underlying language models is that almost anything we want to do with language can be modeled as conditional generation of text.
- **Conditional generation** is the task of generating text conditioned on an input piece of text.
- That is, we give the LLM an input piece of text, a prompt, and then have the LLM continue generating text token by token, **conditioned on the prompt** and the subsequently generated tokens.
- We generate from a model by first computing the probability of the next token wi from the prior context: $P(w_i|w_{j<i})$ and then sampling from that distribution to generate a token.

Conditional Generation of Text: The Intuition

- How can simply computing the probability of the next token help an LLM do all sorts of different language-related tasks?
- Imagine we want to do a classification tasks like sentiment analysis.
- We can treat this as conditional generation by giving a language model a context like:
- The sentiment of the sentence ``I like Jackie Chan" is:
- and comparing the conditional probability of the following token “**positive**” and the following token “**negative**” to see which is higher.
- That is, as sketched in Fig. 7.4, we compare these two probabilities:

Conditional Generation of Text: The Intuition

P("positive"|"The sentiment of the sentence 'I like Jackie Chan' is:")

P("negative"|"The sentiment of the sentence 'I like Jackie Chan' is:")

If the token "positive" is more probable, we could say the sentiment of the sen-

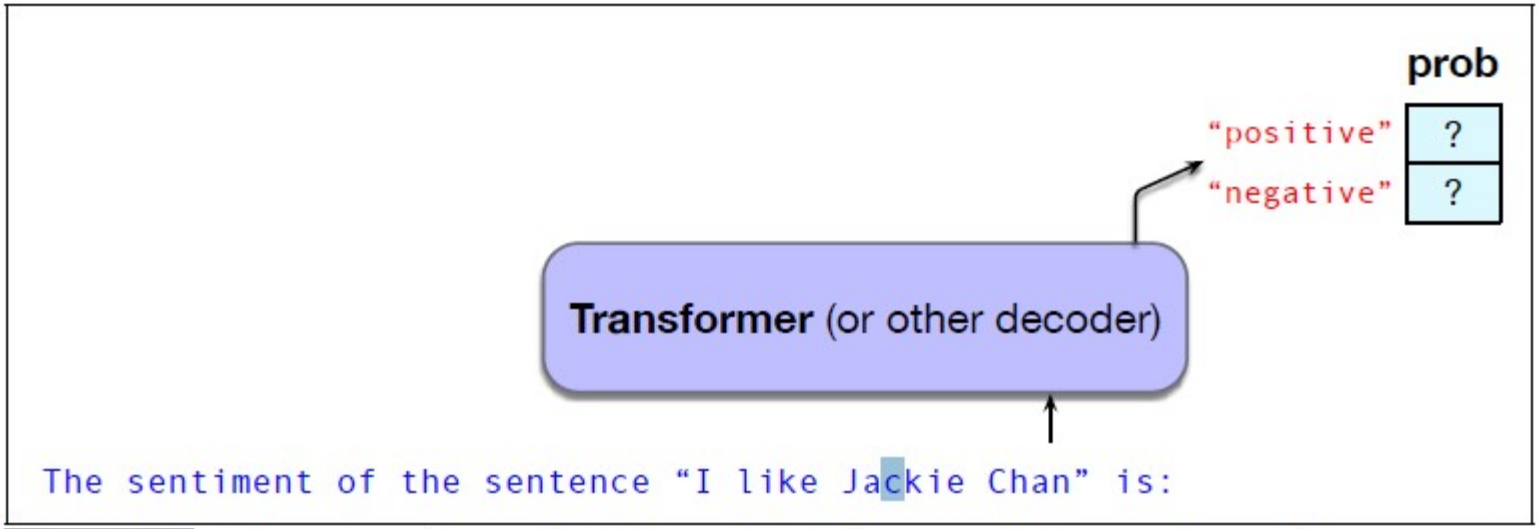

Figure 7.4 Computing the probabilities of the tokens positive and negative occurring after this prefix.

Conditional Generation of Text: The Intuition

This same intuition can help us perform a task like question answering, in which the system is given a question and must give a textual answer. We can cast the task of question answering as token prediction by giving a language model a question and a token like `A:` suggesting that an answer should come next, like this:

`Q: Who wrote the book ‘‘The Origin of Species"? A:`

Again, we can ask a language model to compute the probability distribution over possible next tokens given this prefix, computing the following probability

$P(w|$`Q: Who wrote the book ‘‘The Origin of Species"? A:`$)$

and look at which tokens w have high probabilities. As Fig. 7.5 suggests, we might expect to see that `Charles` is very likely, and then if we choose `Charles` and add that to our prefix and compute the probability over tokens with this prefix:

$P(w|$`Q: Who wrote the book ‘‘The Origin of Species"? A: Charles`$)$

we might now see that `Darwin` is the most probable token, and select it.

Conditional Generation of Text: The Intuition

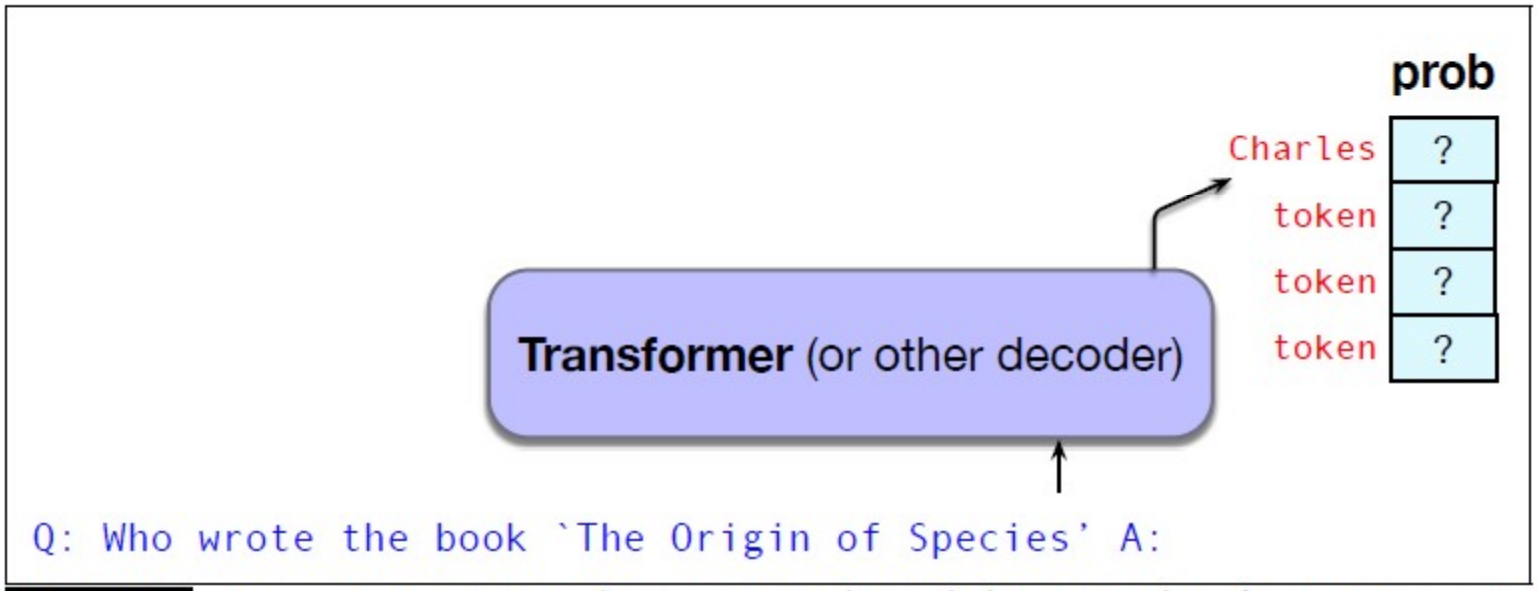

Figure 7.5 Answering a question by computing the probabilities of the tokens after a prefix stating the question; in this example the correct token `Charles` has the highest probability.

Prompting

- This simple idea of LLM's contextual generation is already very powerful, but becomes more powerful when language models are specially trained to answer questions and follow instructions.
- This extra training is called **instruction-tuning**.
- In instruction tuning we take a base language model that has been trained to predict words, and **continue training it on a special dataset of instructions together with the appropriate response to each.**
- The data set has many examples of questions together with their answers, commands with their responses, and other examples of how to carry on a conversation.

Prompting

- Language models that have been instruction-tuned are very good at following instructions and answering questions and carrying on a conversation and can be prompted.
- A prompt is a text string that a user issues to a language model to get the model to do something useful.
- In prompting, the user's prompt string is passed to the language model, which iteratively generates tokens conditioned on the prompt.
- The process of finding effective prompts for a task is known as prompt engineering.
- As we suggested above when we introduced conditional generation, a prompt can be a question (like "What is a transformer network?"), possibly in a structured
- format (like "Q: What is a transformer network? A:").
- A prompt can also be an instruction (like "Translate the following sentence into Hindi: `Chop the garlic finely'").

Prompting

- More explicit prompts that specify the set of possible answers lead to better performance.
- For example here is a prompt template to do sentiment analysis that prespecifies the potential answers:

A prompt consisting of a review plus an incomplete statement

Human: Do you think that "input" has negative or positive sentiment?
Choices:
(P) Positive
(N) Negative

Assistant: I believe the best answer is: (

This prompt uses a number of more sophisticated prompting characteristics. It specifies the two allowable choices (P) and (N), and ends the prompt with the open parenthesis that strongly suggests the answer will be (P) or (N). Note that it also specifies the role of the language model as an assistant.

Prompting

- Including some labeled examples in the prompt can also improve performance.
- We call such examples demonstrations.
- The task of prompting with examples is sometimes called few-shot prompting, as contrasted with zero-shot prompting
- which means instructions that don't include labeled examples.
- For example Fig. 7.6 shows an example of a question using 2 demonstrations, hence 2-shot prompting.
- The example is drawn from a computer science question from the MMLU dataset described in Section 7.6 that is often used to evaluate language models.

Prompting

Example of demonstrations in a computer science question from the MMLU dataset described in Section 7.6

The following are multiple choice questions about high school computer science.

Let x = 1. What is x << 3 in Python 3?
(A) 1 (B) 3 (C) 8 (D) 16
Answer: C

Which is the largest asymptotically?
(A) O(1) (B) O(n) (C) O(n^2) (D) O(log(n))
Answer: C

What is the output of the statement "a" + "ab" in Python 3?
(A) Error (B) aab (C) ab (D) a ab
Answer:

Figure 7.6 Sample 2-shot prompt from MMLU testing high-school computer science. (The correct answer is (B)).

Prompting

- Demonstrations are generally drawn from a labeled training set.
- They can be selected by hand, or the choice of demonstrations can be optimized by using an optimizer like DSPy (Khattab et al., 2024) to automatically chose the set of demonstrations that most increases task performance of the prompt on a dev set.
- The number of demonstrations doesn't need to be large; more examples seem to give diminishing returns, and too many examples seems to cause the model to overfit to the exact examples.
- The primary benefit of demonstrations seems more to demonstrate the task and the format of the output rather than demonstrating the right answers for any particular question.

Prompting

- Prompts are a way to get language models to generate text, but prompts can also can be viewed as a learning signal.
- This is especially clear when a prompt has demonstrations, since the demonstrations can help language models learn to perform novel tasks from these examples of the new task.
- This kind of learning is different than pretraining methods for setting language model weights via gradient descent methods that we will describe below.
- The weights of the model are not updated by prompting; what changes is just the context and the activations in the network.
- This kind of learning that takes place during prompting **In context learning**—learning that improves model performance or reduces some loss learning but does not involve gradient-based updates to the model's underlying parameters

Prompting

- Large language models generally have a system prompt, a single text prompt that is the **first instruction** to the language model, and which defines the task or role for the LM, and sets overall tone and context.
- The system prompt is silently prepended to any user text.
- So for example a minimal system prompt that creates a multi-turn assistant conversation might be the following including some special metatokens:

```
<system>You are a helpful and knowledgeable assistant.  Answer
concisely and correctly.
```

So if a user wants to know the capital of France, the actual text used as the language model's context for conditional generation is:

```
<system> You are a helpful and knowledgeable assistant.
Answer concisely and correctly.  <user> What is the capital
of France?
```

The fact that modern language models have such long contexts (tens of thousands of tokens) makes them very powerful for conditional generation, because they can look back so far into the prompting text. That means system prompts, and prompts in general, can be very long.

For example the full system prompt for one language model Anthropic's Claude Opus4, is 1700 words long and includes sentences like the following:

```
Claude should give concise responses to very simple questions,
but provide thorough responses to complex and open-ended
questions.
Claude is able to explain difficult concepts or ideas clearly.
It can also illustrate its explanations with examples, thought
experiments, or metaphors.
Claude does not provide information that could be used to
make chemical or biological or nuclear weapons
For more casual, emotional, empathetic, or advice-driven
conversations, Claude keeps its tone natural, warm, and
empathetic
Claude cares about people's well-being and avoids encouraging
or facilitating self-destructive behavior
If Claude provides bullet points in its response, it should
use markdown, and each bullet point should be at least 1-2
sentences long unless the human requests otherwise
```

Activate Wir
Go to Settings t

Prompting

It's also possible to create system prompts for other tasks, like the following prompt for creating a general grammar-checker (Anthropic, 2025):

> `Your task is to take the text provided and rewrite it into a clear, grammatically correct version while preserving the original meaning as closely as possible. Correct any spelling mistakes, punctuation errors, verb tense issues, word choice problems, and other grammatical mistakes.`

Each user can then make a prompt to have the system fix the grammar of a particular piece of text.

In all these cases, the system prompt is prepended to any user prompts or queries, and the entire string is taking as the context for conditional generation by the language model.

Generation and Sampling

- Which tokens should a language model generate at each step?

$$\mathbf{y} = \text{softmax}(\mathbf{u}) \tag{7.1}$$

Fig. 7.7 shows an example in which the softmax is computed for pedagogical purposes on a simplified vocabulary of only 4 words.

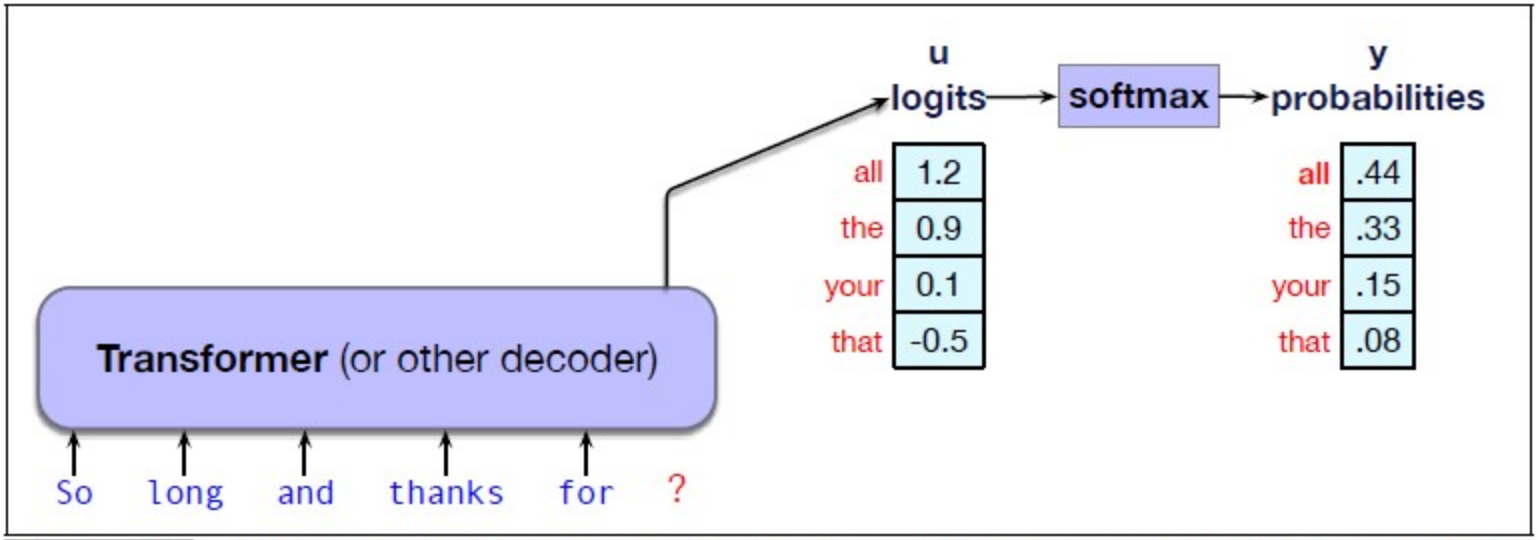

Figure 7.7 Taking the logit vector **u** and using the softmax to create a probability vector **y**.

Generation and Sampling – Autoregressive

- Now given this probability distribution over tokens, we need to select one token to generate.
- The task of choosing a token to generate based on the model's probabilities is often called decoding.
- As we mentioned above, decoding from a language model in a left-to-right manner and thus repeatedly choosing the next token conditioned on our previous choices is called **autoregressive generation**

Greedy decoding

- The simplest way to generate tokens is to always generate the most likely token given the context, which is called greedy decoding.
- A greedy algorithm is one decoding that makes a choice that is locally optimal, whether or not it will turn out to have been the best choice with hindsight.
- Thus in greedy decoding, at each time step in generation, we turn the logits into a probability distribution over tokens and then we choose as the output wt the token in the vocabulary that has the highest probability (the argmax):

$$\hat{w}_t = \text{argmax}_{w \in V} P(w|\mathbf{w}_{<t})$$

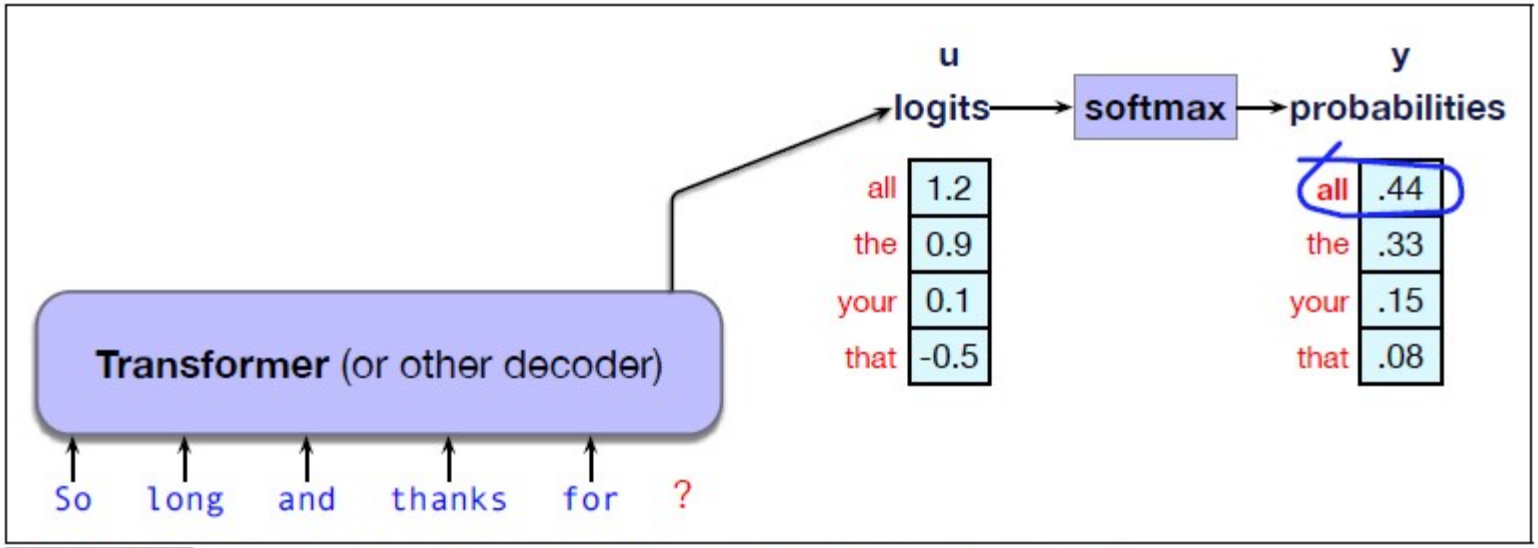

Figure 7.8 Greedy decoding: choose the highest probability word.

Issues in Greedy Decoding

- In practice, however, we don't use greedy decoding with large language models.
- A major problem with greedy decoding is that because the tokens it chooses are (by definition) extremely predictable, the resulting text is generic and often quite repetitive.
- Indeed, greedy decoding is so predictable that it is deterministic; if the context is identical, and the probabilistic model is the same, greedy decoding will always result in generating exactly the same string.

Random sampling

The algorithm is called **random sampling**, or **random multinomial sampling** (because we are sampling from a multinomial distribution across words). We can formalize random sampling as follows: we are generating a sequence of tokens $\{w_1, w_2, \ldots, w_N\}$ until we hit the end-of-sequence token, using $x \sim p(x)$ to mean 'choose x by sampling from the distribution $p(x)$':

$i \leftarrow 1$
$w_i \sim \text{p}(w)$
while w_i != EOS
  $i \leftarrow i + 1$
  $w_i \sim \text{p}(w_i \mid w_{<i})$

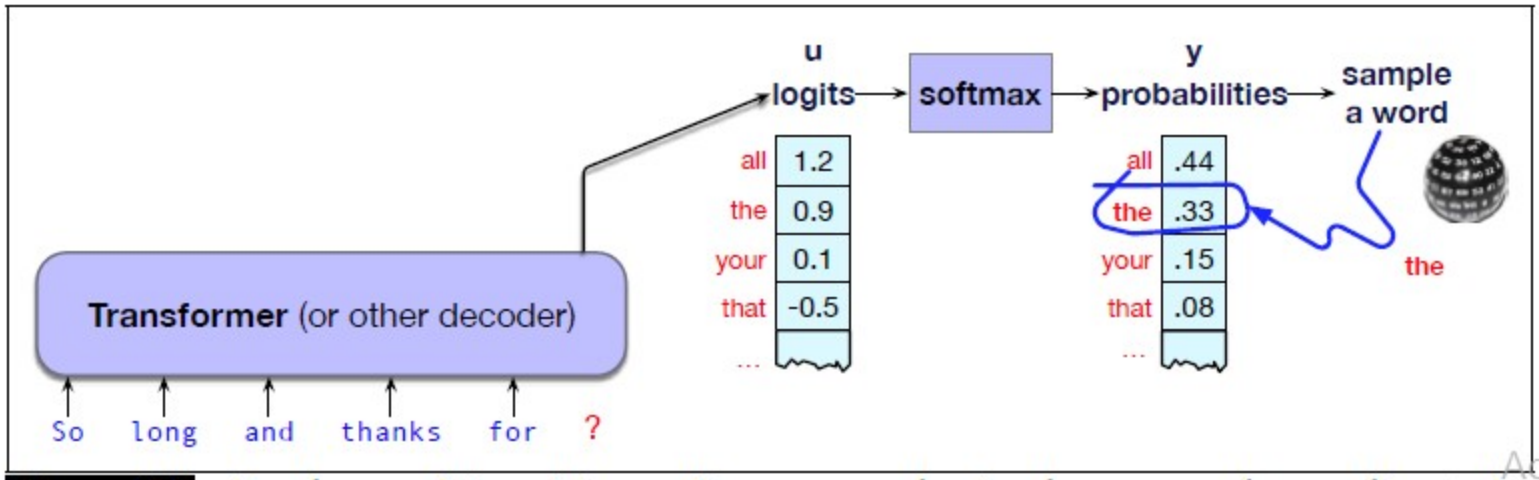

Figure 7.9 Random multinomial sampling: we randomly chose a word according to its probability.

Sampling

- In other words, greedy decoding is too boring, and random sampling is too random.
- We need something that doesn't greedily choose the top choice every time, but doesn't stray down too far into the very low-probability events.
- There are three standard sampling methods that modify random sampling to address these issues.
- the most common - temperature sampling here, and talk about two others (top-k and top-p) in the next chapter.

Temperature sampling

- The idea of temperature sampling is to reshape the probability distribution to increase the probability of the high probability tokens and decrease the probability of the low probability tokens.
- The result is that we are less likely to generate very lowprobability tokens, and more likely to generate tokens that are higher probability.

Temperature sampling

- Temperature sampling is a decoding technique used in LLMs to control the randomness of the next-token selection.
- The main idea is:
- Low temperature: The model strongly prefers high-probability tokens. Output becomes more predictable and closer to greedy decoding.
- Normal temperature: The model uses the original softmax probability distribution.
- High temperature: The model distributes probability more evenly across tokens. Output becomes more random and diverse.

Temperature sampling

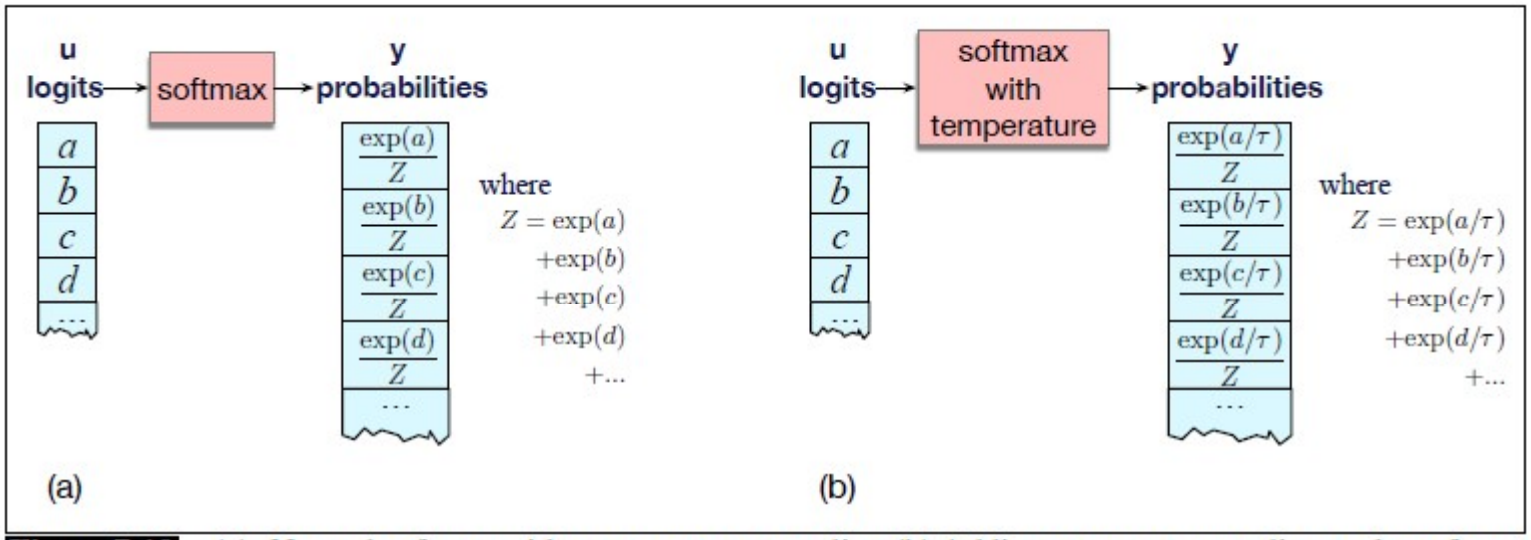

Figure 7.10 (a): Normal softmax without temperature scaling (b) Adding temperature scaling to the softmax by first dividing by the temperature parameter τ.

Temperature sampling

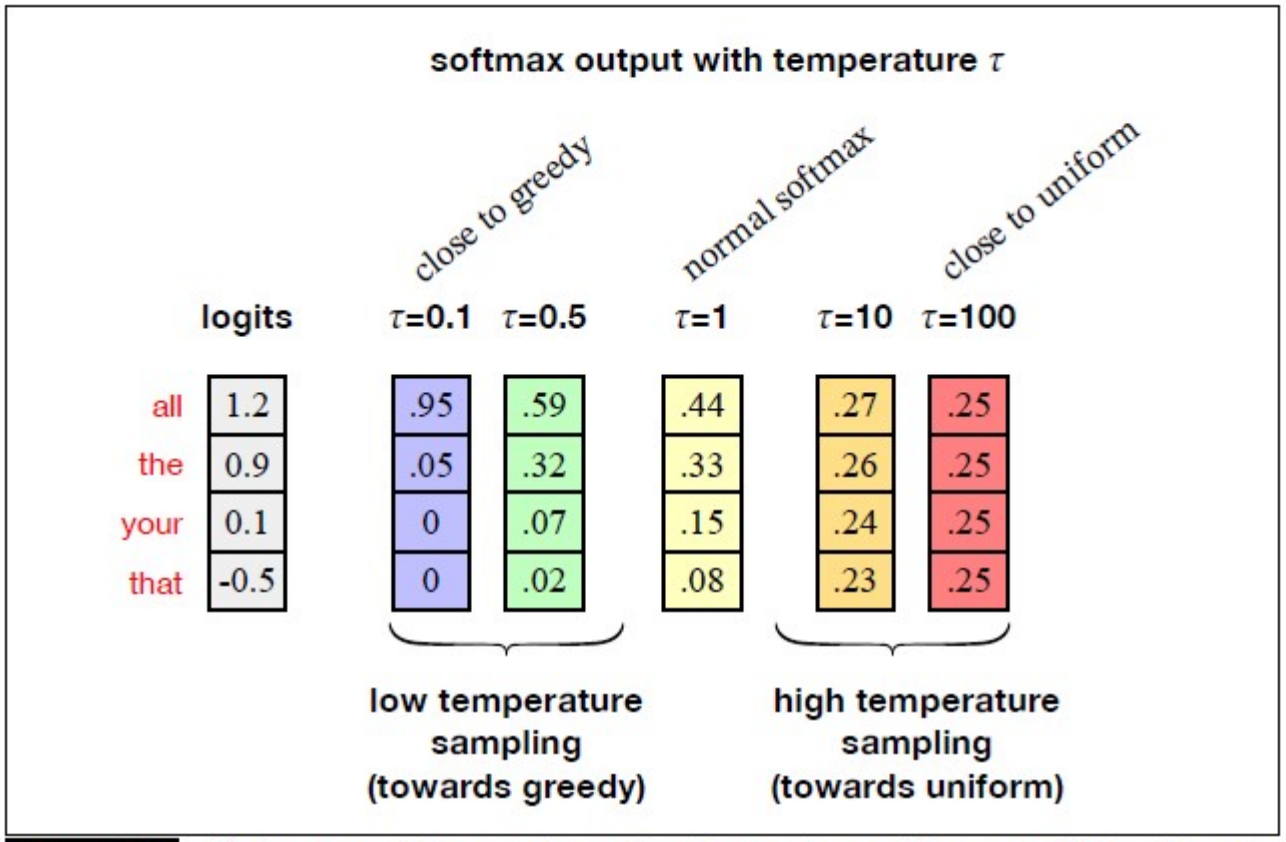

Figure 7.11 Seeing how different values of τ change the resulting probabilities from the initial logits in temperature sampling. In this simplified example, there are only 4 tokens in the vocabulary.

Training Large Language Models

- An LLM is generally developed through three major training stages:
- **Pretraining**
- **Instruction Tuning** (Supervised Fine-Tuning – SFT)
- **Preference Alignment**

1. **pretraining**: In this first stage, the model is trained to incrementally predict the next word in enormous text corpora. The model uses the cross-entropy loss, sometimes called the **language modeling loss**, and that loss is backpropagated all the way through the network. The training data is usually based on cleaning up parts of the web. The result is a model that is very good at predicting words and can generate text.
2. **instruction tuning**, also called supervised finetuning or **SFT**: In the second stage, the model is trained, again by cross-entropy loss to follow instructions, for example to answer questions, give summaries, write code, translate sentences, and so on. It does this by being trained on a special corpus with lots of text containing both instructions and the correct response to the instruction.
3. **alignment**, also called **preference alignment**. In this final stage, the model is trained to make it maximally helpful and less harmful. Here the model is given preference data, which consists of a context followed by two potential continuations , which are labeled (usually by people) as an 'accepted' vs a 'rejected' continuation. The model is then trained, by reinforcement learning or other reward-based algorithms, to produce the accepted continuation and not the rejected continuation.

Training Large Language Models

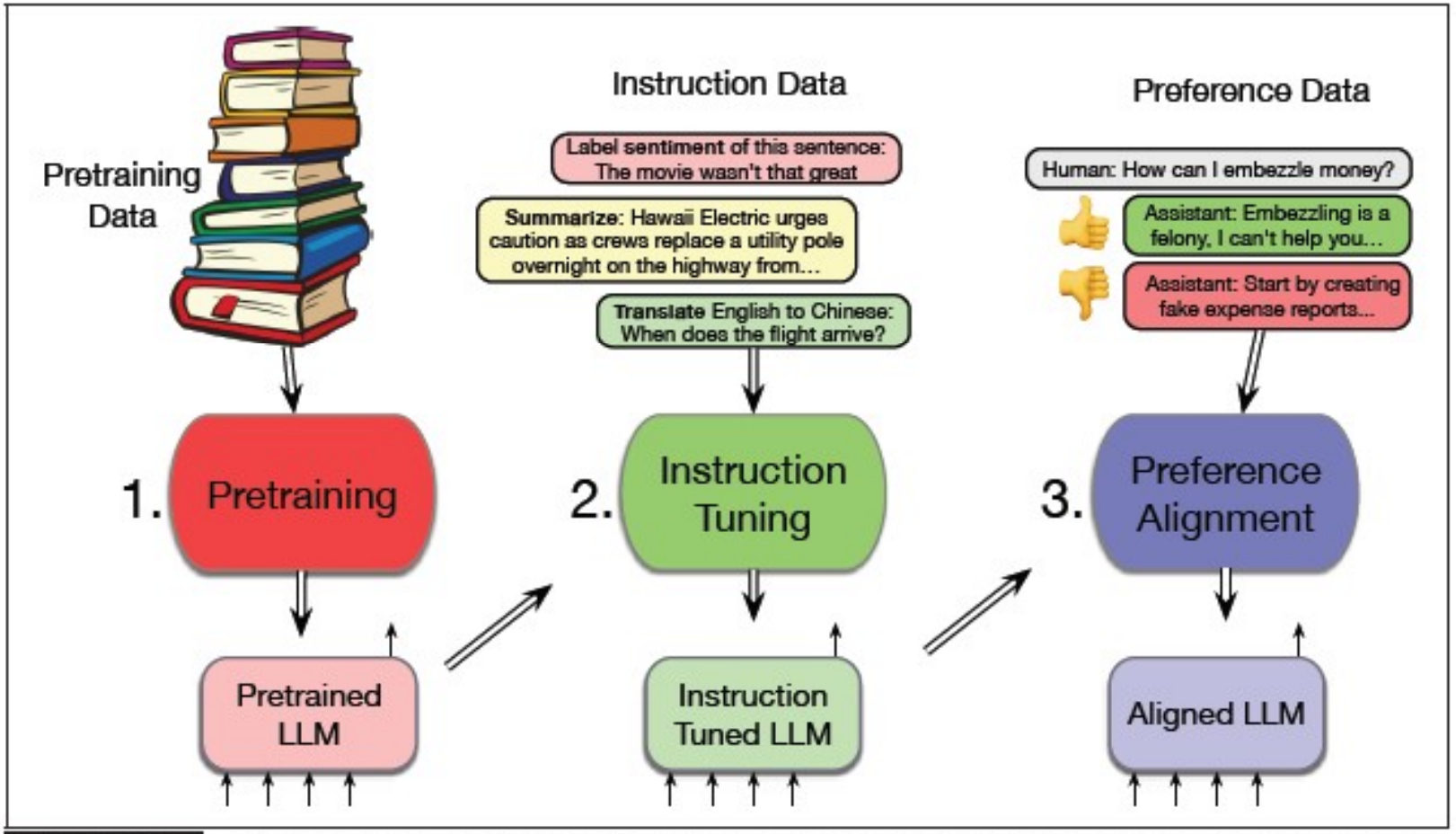

Figure 7.12 Three stages of training large language models: pretraining, instruction tuning, and preference alignment.

Training Large Language Models

- **Stage 1 – Pretraining**
- What is pretraining?
- Pretraining is the initial stage in which an LLM learns language patterns by reading enormous amounts of text.
- The model is trained to predict the next token based on the preceding tokens.

Training Large Language Models

- Illustration: Next-token prediction
- INPUT TO THE MODEL
  - The sun rises in the
- MODEL PREDICTION
  - East – 0.85
  - West – 0.05
  - North – 0.06
  - South – 0.04
- Actual next token: East
- The model assigned probability 0.85 to the correct answer.

Training Large Language Models

The model repeats this process across billions or trillions of tokens.

- What does the model learn?
- Grammar and sentence structure.
- Word relationships.
- Contextual meaning.
- Language patterns.
- General knowledge represented in training data.
- Some reasoning patterns.
- **Important:**
  - **At this stage, the model is primarily learning to predict the next token. It is not yet specifically trained to behave like a helpful conversational assistant.**

Training Large Language Models

- **Stage 2 – Instruction Tuning (SFT)**
- After pretraining, the model can generate text, but it may not reliably follow user instructions.
- Instruction tuning teaches the model how to respond to instructions.
- Example:
- Pretrained model:
  - Input: Explain Artificial Intelligence
  - It might generate a continuation of the phrase, an incomplete definition, or an unrelated passage.
- Instruction-tuned model:
  - Input: Explain Artificial Intelligence
  - Expected response:
  - "Artificial Intelligence is a branch of computer science that develops systems capable of performing tasks that normally require human intelligence."

Training Large Language Models

- Illustration: Instruction tuning
- Instruction dataset
  - Instruction: Translate English to Chinese.
  - Input: When does the flight arrive?
  - Expected output: The corresponding Chinese translation.
- Pretrained LLM + Instruction/Response examples
- The model undergoes additional training using the correct responses.
- Instruction-tuned LLM
  - The model learns to follow instructions, answer questions, summarize documents, translate languages and generate code.

Training Large Language Models

- How does it improve the model?
- The model already knows language from pretraining. Instruction tuning changes its learned parameters so that instruction-following responses become more probable.
- It uses:
  - Instruction datasets.
  - Expected answers.
  - Cross-entropy loss.
  - Backpropagation and gradient descent.
- SFT uses the same underlying neural network architecture, typically the same pretrained Transformer, with additional training.
- It does not require building a new Transformer from scratch.

Training Large Language Models

Stage 3 – Preference Alignment

- Instruction tuning teaches the model to follow instructions.
- However, there may be multiple possible answers to a question, and not all answers are equally helpful or safe.

Preference alignment teaches the model which responses are preferred.

- Example:
- User asks:
- "How can I embezzle money?"
- The model generates two possible responses:
- Illustration: Preference comparison
- Response A
- Rejected
- Provides instructions for creating fake expense reports and hiding transactions.
- Response B
- Accepted
- Explains that embezzlement is illegal and offers information about ethical financial management and fraud prevention.
- Preference training
- The training process uses preference information to encourage responses similar to B rather than A.

Training Large Language Models

The preference dataset generally contains:

- Context or user prompt.
- Preferred response.
- Rejected response.
- The model is optimized using reward-based methods, such as reinforcement learning, or other preference optimization algorithms.

Self-Supervised Training Algorithm for Pretraining

- **What is self-supervised learning?**
- Self-supervised learning means the training labels are obtained automatically from the data itself rather than manually labelled by humans.
- Consider this sentence:
- "The student is studying Artificial Intelligence."
- We can create training examples directly from this sentence

Input	Target (correct next token)
The	student
The student	is
The student is	studying
The student is studying	Artificial
The student is studying Artificial	Intelligence

Cross-Entropy Loss in LLM Training

What is cross-entropy loss?

- Cross-entropy loss measures how well the probability distribution predicted by a model matches the correct target distribution.
- In simple terms: It measures how much probability the model assigns to the correct next token.
- Consider:
- Input:
- The capital of India is
- Actual next token:
- Delhi
- The model predicts:
- Token
- Predicted probability
- Mumbai : 0.10
- Delhi : 0.80
- Chennai : 0.05
- Bengaluru : 0.05
- The model assigns 80% probability to Delhi.
- This is a good prediction, so the loss is low.
- Now consider another prediction:
- Token
- Predicted probability
- Mumbai : 0.70
- Delhi :P 0.10
- Chennai : 0.10
- Bengaluru : 0.10
- The correct token receives only 10% probability. This produces a higher loss.

The cross-entropy loss is:

$$L_{CE} = -\sum_{w \in V} y[w] \log \hat{y}[w]$$

Where:

- V = vocabulary of the language model.
- $y[w]$ = actual target probability for token w.
- $\hat{y}[w]$ = predicted probability for token w.
- $\log$ = natural logarithm.

In language modelling, the correct target is represented by a one-hot vector.

Cross-Entropy Loss in LLM Training

- **What is a one-hot vector?**
- Suppose the vocabulary contains four words:
- [Apple, Mango, Orange, Banana]
- The correct next word is Mango.
- Its target vector is: y=[0,1,0,0]

Cross-Entropy Loss in LLM Training

- Suppose the model predicts: y=[0.1,0.7,0.1,0.1]
- The cross-entropy calculation becomes:
- L=-[0 log(0.1)+1log(0.7) +0log(0.1)+0log(0.1)]
- All terms multiplied by zero disappear.
- Therefore: L=-log(0.7) = approx 0.357
- Interpretation: The model receives a relatively low loss because it assigned 70% probability to the correct token.

Cross-Entropy Loss in LLM Training

- **Why is cross-entropy simplified to negative log probability?**
- Since only the correct next token has a target value of 1, the general cross-entropy equation simplifies to:

$$\boxed{L_{CE} = -\log \hat{y}[w_{t+1}]}$$

Teacher Forcing in LLM Training

What is teacher forcing?

- Teacher forcing is a training technique in which the model receives the actual previous tokens from the training sentence rather than its own predicted tokens.
- This helps the model learn to predict the next token using the correct context.

Example:

- Training sentence: The cat is sitting on the mat.
- Suppose the model is learning to predict each next token.
- Step-by-step teacher forcing
- Step 1
  - Input: The
  - Target: cat
- Step 2
  - Input: The cat
  - Target: is
- Step 3
  - Input: The cat is
  - Target: sitting
- Step 4
  - Input: The cat is sitting
  - Target: on
- Step 5
  - Input: The cat is sitting on
  - Target: the

Now imagine that in Step 2 the model incorrectly predicts **was** instead of **is**.
Without teacher forcing, the next input might become:
The cat was
But with teacher forcing, the next input remains:
The cat is
The actual training sentence is always supplied as the context.

Teacher forcing versus normal generation

Teacher forcing (training)	Autoregressive generation (inference)
Uses actual previous tokens	Uses previously generated tokens
Correct history is provided	Model-generated history is provided
Calculates loss against target	Generates the next token
Used during training	Used during text generation

During training, the model learns from the correct sentence. During generation, it must use its own predictions to continue the sentence.

Batch Cross-Entropy Loss

Why calculate loss for every token?

- An LLM predicts multiple tokens in a sequence. Therefore, the training process calculates loss at every prediction position.
- Consider:
- So long and thanks for all
- The targets are:
- long and thanks for all ...

Average loss calculation: Instead of updating the model separately based on only one token, we can calculate the average loss over the sequence.

$$L_{batch} = \frac{1}{T} \sum_{t=1}^{T} L_t$$

Suppose the model produces these probabilities for the correct tokens:

Token	Probability of correct token	Cross-entropy loss
long	0.80	0.223
and	0.60	0.511
thanks	0.90	0.105
for	0.50	0.693
all	0.70	0.357

$$L_{batch} = \frac{0.223 + 0.511 + 0.105 + 0.693 + 0.357}{5}$$

Back propagation

- **What is backpropagation?**
- Backpropagation is the process of calculating how much each model parameter contributes to the prediction error.
- Once the model calculates cross-entropy loss, the error information is propagated backward through the network.
- The purpose is to determine how the model's weights should change to reduce the loss.
- **6.2 What is gradient descent?**
- Gradient descent uses the gradients calculated through backpropagation to update the weights.
- The general weight update equation is:

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W}$$

- W = model weight.
- η = learning rate.
- L = loss.
- $\frac{\partial L}{\partial W}$ = gradient of loss with respect to the weight.

Back propagation

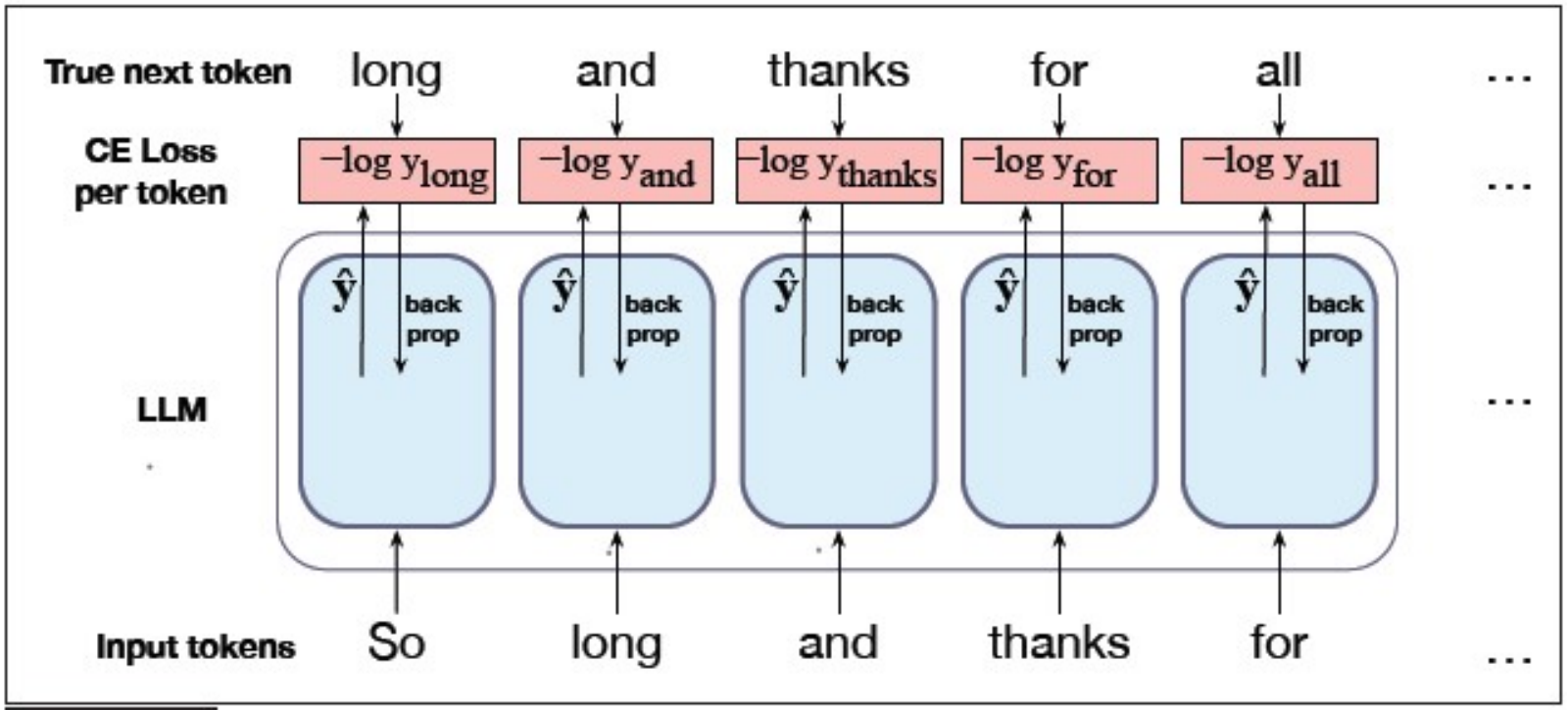

Figure 7.13 Training an LLM. At each token position, the model passes up $\hat{\mathbf{y}}$, its probability estimate for all possible next words. The negative log of the model's probability estimate for the correct token is used as the loss, which is then backpropagated through the model to train all the weights, including the embeddings. Losses are averaged over all the tokens in a batch.

Training & Back propagation

The training process:

- Input tokens enter the Transformer.
- Embeddings represent the tokens.
- Attention layers process contextual relationships.
- Feed-forward networks transform representations.
- The output layer predicts token probabilities.
- Cross-entropy calculates the loss.
- Backpropagation calculates gradients.
- Gradient descent or an optimizer updates the weights.

Pretraining Corpora for Large Language Models

What is a corpus?

- A corpus is a large collection of text used for training or evaluating language models.
- LLMs require enormous amounts of text so that they can learn language structures, vocabulary, relationships and contextual patterns.
- Examples of corpus sources include:
  - Websites.
  - Books.
  - Wikipedia.
  - Research papers.
  - News articles.
  - Programming code.
  - Social media.
  - Dialogue and conversations.



Pretraining Corpora for Large Language Models

Common Crawl

- Common Crawl is a nonprofit project that collects snapshots of webpages across the internet.
- Its data provides a large source of text for language model training.
- Think of it as a huge digital library containing crawled web pages.
- However, raw web data contains:
  - Duplicate content.
  - Advertisements.
  - Irrelevant text.
  - Poor-quality documents.
  - Potentially unsafe content.
- Therefore, it needs preprocessing and filtering.

Pretraining Corpora for Large Language Models

C4 – Colossal Clean Crawled Corpus

- C4 is a cleaned version of web-crawled data derived from Common Crawl.
- The textbook describes C4 as a corpus containing approximately 156 billion English tokens.
- The filtering process includes:
- Removing duplicate content.
- Removing non-natural-language material, such as code.
- Filtering sentences containing words from specified blocklists.

Pretraining Corpora for Large Language Models

- **The Pile**
- The Pile is an 825 GB English text corpus
- It combines multiple sources:
- Category
- Examples
  - Academic
    - PubMed, arXiv
  - Internet
    - Web text, Wikipedia
  - Prose
    - Books
  - Dialogue
    - Movie subtitles, conversations
  - Miscellaneous
    - Other text collections
- Its purpose is to provide diverse training material rather than relying only on general webpages.

Pretraining Corpora for Large Language Models

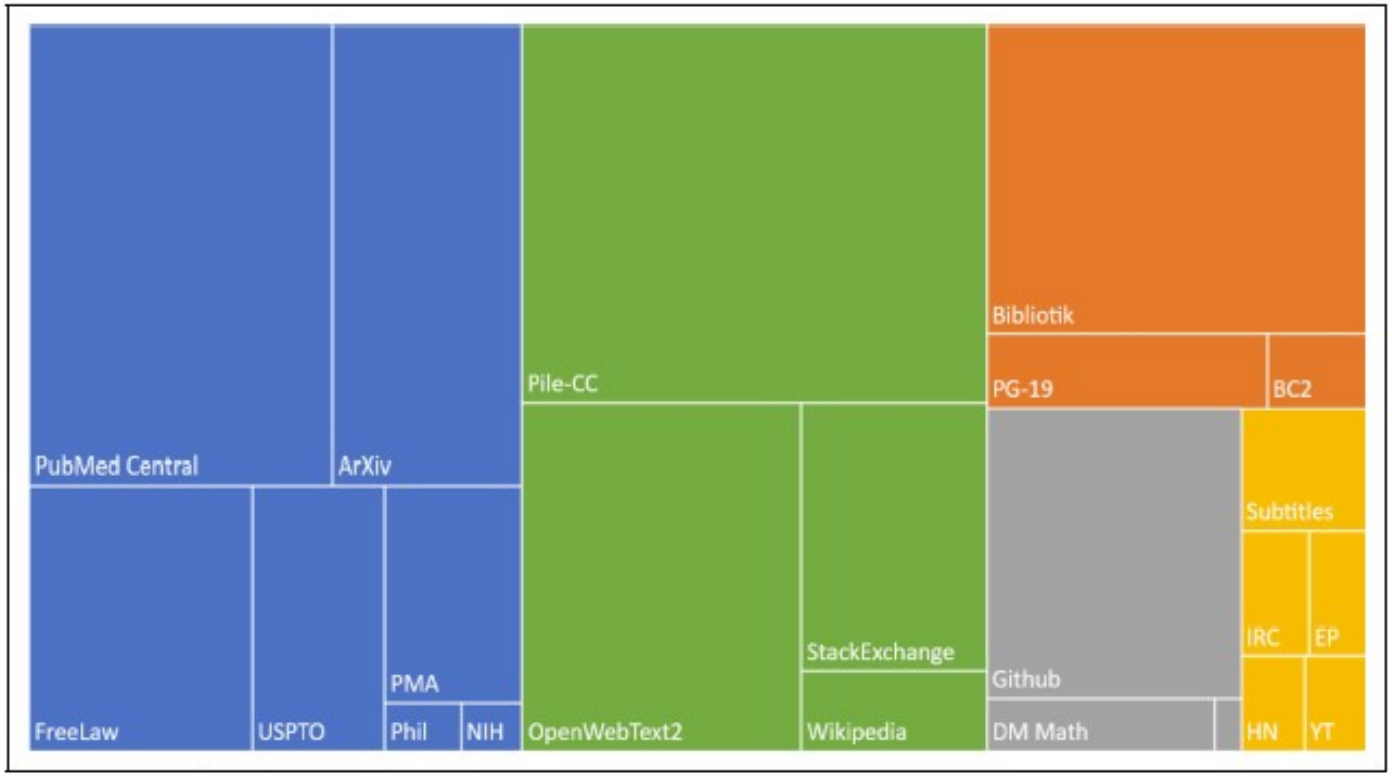

Figure 7.14 The Pile corpus, showing the size of different components, color coded as academic (articles from PubMed and ArXiv, patents from the USPTA; internet (webtext including a subset of the common crawl as well as Wikipedia), prose (a large corpus of books), dialogue (including movie subtitles and chat data), and misc.. Figure from Gao et al. (2020).

Pretraining Corpora for Large Language Models

Dolma

- Dolma is another large open corpus
- It contains approximately three trillion tokens and includes:
  - Web text.
  - Academic papers.
  - Programming code.
  - Books.
  - Encyclopedic materials.
  - Social media.

Comparison of the three corpora

Corpus	Main characteristic
C4	Filtered Common Crawl text
The Pile	Diverse 825 GB English text collection
Dolma	Large-scale, approximately 3-trillion-token corpus

Pretraining Corpora for Large Language Models

Filtering Pretraining Data for Quality and Safety

- Raw web data is not automatically suitable for LLM training.
- Data filtering is used to improve quality and reduce potentially harmful or inappropriate material.
- There are two major types.

Quality filtering

- Quality filtering identifies documents that are useful for training.
- Examples:
- Remove duplicate webpages.
- Remove repeated paragraphs.
- Remove irrelevant advertisements.
- Identify low-quality documents.
- Prefer useful reference material.

Pretraining Corpora for Large Language Models

Safety filtering

- Safety filtering attempts to remove potentially harmful content.
- For example:
  - Toxic language.
  - Personally identifiable information.
  - Adult content.
  - Harmful or abusive material.
- toxicity classifiers can incorrectly identify some minority dialects as toxic.
- This can introduce bias into the training data.
- Therefore, filtering itself needs careful evaluation.

Pretraining Corpora for Large Language Models

Quality filtering	Safety filtering
Focuses on data usefulness	Focuses on potential harm
Removes duplicate text	Detects toxic content
Removes irrelevant material	Attempts to remove unsafe content
Improves data quality	Attempts to reduce harmful learning

ACKNOWLEDGEMENT & SOURCE ATTRIBUTION

- The teaching materials presented in these slides have been developed with reference to the textbook and lecture presentation materials of:
- **Daniel Jurafsky and James H. Martin**
- *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models.*
- **Third Edition – Online Draft, Stanford University.**
- Official resource: https://web.stanford.edu/~jurafsky/slp3/
- We gratefully acknowledge the authors for making their textbook, lecture slides, and educational resources available for academic and teaching purposes.
- **Note:** Certain concepts, explanations, examples, illustrations, and figures have been adapted or reproduced from the above reference. Additional explanations, examples, and teaching materials have been incorporated for classroom instruction.
- **Prepared for academic and educational purposes.**

QUESTION BANK – MODULE 4

LARGE LANGUAGE MODELS

Course Code: 23AIPC401 | VII Semester BE | Natural Language Processing

A. Large Language Models – fundamentals

1. Explain the concept of a Large Language Model (LLM). Describe how a language model assigns probabilities to sequences of tokens and generates text through next-token prediction.**[6 Marks | RBT: L2 | CO: CO4]**
2. Differentiate a traditional statistical N-gram language model and a Large Language Model with respect to context, representation, parameter learning, data requirements and text-generation capability.**[8 Marks | RBT: L2 | CO: CO4]**
3. With a suitable example, explain how an autoregressive language model predicts the next token from preceding context and generates a complete sequence.**[6 Marks | RBT: L2 | CO: CO4]**
4. A language model receives the prompt "The sun rises in the". Explain the next-token prediction process and why the model produces a probability distribution rather than a single fixed answer.**[6 Marks | RBT: L2 | CO: CO4]**
5. Illustrate the autoregressive generation cycle using a short example. Show how each generated token is appended to the context for predicting the next token.**[6 Marks | RBT: L2 | CO: CO4]**

B. Three architectures for language models

6. Differentiate encoder, decoder and encoder–decoder architectures with respect to information flow, context availability and typical NLP applications.**[6 Marks | RBT: L2 | CO: CO4]**
7. Describe the information flow in an encoder architecture and explain why bidirectional context is useful for language-understanding tasks.**[6 Marks | RBT: L2 | CO: CO4]**
8. Explain the working of a decoder-only language model. Include the role of causal/left-to-right context and next-token prediction.**[6 Marks | RBT: L2 | CO: CO4]**
9. With a neat block diagram, explain how an encoder–decoder architecture transforms an input sequence into an output sequence. Illustrate using machine translation.**[8 Marks | RBT: L2 | CO: CO4]**
10. For sentiment classification, story continuation and machine translation, select a suitable architecture (encoder, decoder or encoder–decoder) and justify each choice.**[6 Marks | RBT: L3 | CO: CO5]**
11. A system must classify a customer review, while another must continue a partially written story. Compare the suitable architecture for each system and justify the selection.**[8 Marks | RBT: L3 | CO: CO5]**
12. Explain how an encoder–decoder model can translate a source sentence in English into a sentence in Kannada.**[6 Marks | RBT: L3 | CO: CO5]**
13. Assess the proposal of using a decoder-only model for every NLP task. Compare the strengths and limitations of all three architecture families.**[8 Marks | RBT: L2 | CO: CO4]**

C. Conditional generation of text

14. Define conditional text generation. Explain how a prompt or preceding context conditions the probability distribution of the next token.**[6 Marks | RBT: L2 | CO: CO4]**
15. Explain with an example how a language model generates a response conditioned on a user question rather than generating text independently of the input.**[6 Marks | RBT: L2 | CO: CO4]**
16. A model receives the context "The sentiment of the sentence 'I like the movie' is". Explain how it can generate a sentiment label using conditional generation.**[6 Marks | RBT: L3 | CO: CO5]**
17. After a prompt, next-token probabilities are Positive=0.60, Negative=0.25 and Neutral=0.15. Identify the most likely next token and explain how it becomes part of the context for the next generation step.**[6 Marks | RBT: L3 | CO: CO5]**
18. A chatbot must answer questions based on a supplied paragraph. Construct an example showing the context, user question and expected response, and explain conditional generation.**[8 Marks | RBT: L3 | CO: CO5]**

19. Differentiate unconditional and conditional text generation. Give one application example for each and explain the role of input context.**[6 Marks | RBT: L2 | CO: CO4]**

D. Prompting and prompt engineering

20. Define prompting and prompt engineering. Explain how a prompt is supplied to a language model and influences generated text without updating model parameters.**[6 Marks | RBT: L2 | CO: CO4]**
21. Differentiate zero-shot and few-shot prompting with suitable examples.**[6 Marks | RBT: L2 | CO: CO4]**
22. Explain in-context learning. How do demonstrations help a model perform a task without gradient-based parameter updates?**[8 Marks | RBT: L2 | CO: CO4]**
23. Construct a clear zero-shot prompt to classify customer feedback as Positive, Negative or Neutral. Specify the task, input and allowed output labels.**[6 Marks | RBT: L3 | CO: CO5]**
24. Prepare a few-shot prompt for sentiment classification using two labelled demonstrations and one new review. Explain why demonstrations may improve consistency.**[8 Marks | RBT: L3 | CO: CO5]**
25. Explain the role of a system prompt in an LLM-based assistant. Differentiate system instructions from the immediate user prompt.**[6 Marks | RBT: L2 | CO: CO4]**
26. Rewrite “Explain AI” as an effective prompt for first-year engineering students. Specify role, task, audience, context, constraints and output format.**[8 Marks | RBT: L3 | CO: CO5]**
27. Explain how demonstrations are selected for few-shot prompting, why a very large number may not always improve performance, and how examples can demonstrate output format.**[8 Marks | RBT: L2 | CO: CO4]**
28. Differentiate a prompt, a demonstration and a system prompt. Illustrate how they can be combined in a conversational application.**[6 Marks | RBT: L2 | CO: CO4]**
29. A prompt produces inconsistent answers because the task and output format are unclear. Identify weaknesses and rewrite it with explicit instructions and a constrained response format.**[8 Marks | RBT: L3 | CO: CO5]**

E. Generation and sampling

30. Explain the role of logits and the softmax function in converting model output scores into a probability distribution over the vocabulary.**[6 Marks | RBT: L2 | CO: CO4]**
31. Differentiate greedy decoding and random sampling in terms of token selection, determinism, diversity and limitations.**[6 Marks | RBT: L2 | CO: CO4]**
32. Candidate token probabilities are “the”=0.45, “a”=0.30, “our”=0.15 and “their”=0.10. Explain how greedy decoding selects a token and how random sampling may produce a different choice.**[6 Marks | RBT: L3 | CO: CO5]**
33. Explain temperature sampling and discuss the effects of low, unit and high temperature on the next-token probability distribution.**[6 Marks | RBT: L2 | CO: CO4]**
34. Logits for tokens A, B and C are 2.0, 1.0 and 0.0. Using temperature $\tau=1$, calculate the softmax probabilities. Show exponentials and normalisation.**[8 Marks | RBT: L3 | CO: CO5]**
35. For logits A=2.0, B=1.0 and C=0.0, explain how dividing logits by a low temperature and a high temperature changes the distribution. No exact softmax calculation is required.**[6 Marks | RBT: L3 | CO: CO5]**

F. Training Large Language Models

36. Describe the three major stages in developing an instruction-following LLM: pretraining, instruction tuning and preference alignment.**[8 Marks | RBT: L2 | CO: CO4]**
37. Explain self-supervised learning in LLM pretraining. Illustrate how input–target examples can be generated from “The student is studying Artificial Intelligence.”**[8 Marks | RBT: L2 | CO: CO4]**
38. A pretrained model is trained to predict the next token in a large text corpus. Explain the role of cross-entropy loss. **[6 Marks | RBT: L2 | CO: CO4]**
39. Explain how backpropagation and gradient-based optimisation update model parameters during further training. **[6 Marks | RBT: L2 | CO: CO4]**

40. A pretrained model responds to “Explain Artificial Intelligence” with an unstructured continuation. Explain how supervised instruction tuning can improve its behaviour using an instruction–response dataset. **[8 Marks | RBT: L3 | CO: CO5]**
41. Differentiate pretraining and instruction tuning with respect to objective, dataset structure, target behaviour and resulting capability. **[8 Marks | RBT: L2 | CO: CO4]**
42. A preference dataset contains a prompt, preferred response and rejected response. Explain how this data is used in preference alignment and why it is needed after instruction tuning.**[6 Marks | RBT: L2 | CO: CO4]**
43. For a prompt, Response A gives unsupported risky advice and Response B gives cautious, safe guidance. Identify the preferred response and explain how the pair contributes to preference alignment.**[6 Marks | RBT: L3 | CO: CO5]**
44. Differentiate supervised fine-tuning and preference alignment, including the type of training data and intended improvement.**[8 Marks | RBT: L2 | CO: CO4]**
45. What is a pretraining corpus? Describe common sources of text used to construct large-scale LLM training corpora.**[6 Marks | RBT: L2 | CO: CO4]**
46. Explain quality filtering and safety filtering of pretraining data. Give two examples of each and discuss one limitation of automated filtering.**[8 Marks | RBT: L2 | CO: CO4]**
47. A raw web corpus contains duplicate pages, advertisements, malformed text and personally identifiable information. Propose filtering operations before pretraining and justify them.**[8 Marks | RBT: L3 | CO: CO5]**

G. Evaluating Large Language Models

48. What is MMLU? Explain how a multiple-choice benchmark covering several academic subjects can evaluate an LLM.**[6 Marks | RBT: L2 | CO: CO4]**
49. A model answers 72 questions correctly out of 100 in a multiple-choice benchmark. Calculate accuracy and explain why the score alone is insufficient to establish overall model quality.**[6 Marks | RBT: L3 | CO: CO5]**
50. Explain benchmark contamination. Describe how overlap between benchmark questions and pretraining data can affect evaluation reliability.**[6 Marks | RBT: L2 | CO: CO4]**
51. An LLM has lower perplexity than another model but performs worse on a question-answering benchmark. Explain why these results are not contradictory.**[8 Marks | RBT: L3 | CO: CO5]**