Lecture 6

Generative AI and Large Language Models

Lars A. Lochstoer

UCLA Anderson School of Management

This class

Overview of Large Language Models (LLMs)

- **Generative AI** as sequential word prediction
- Context
- Why do LLMs seem "smart"?

Transformers

- Main architecture behind recent successful models
- **Attention** is key
- Contextual embeddings are learned within transformers
- Example of tiny GPT

LLMs in practice

- Actually, more like a SLM (small language model)
- Gemma 3, 1b-it

Large Language Models

(LLMs)

Generative AI and Word Prediction

"All" these models do is to predict the next word (really, token) in a sequence:

$$p\left(w_{i+1}|w_i, w_{i-1}, ..., x\right)$$

- Here, $p\left(w_{i+1}|...\right)$ is a conditional distribution over possible words in vocabulary V
- w_i, w_{i-1} are the specific previous words in sequence and x is other context such as a prompt/instruction

To choose a specific word, sample from this distribution

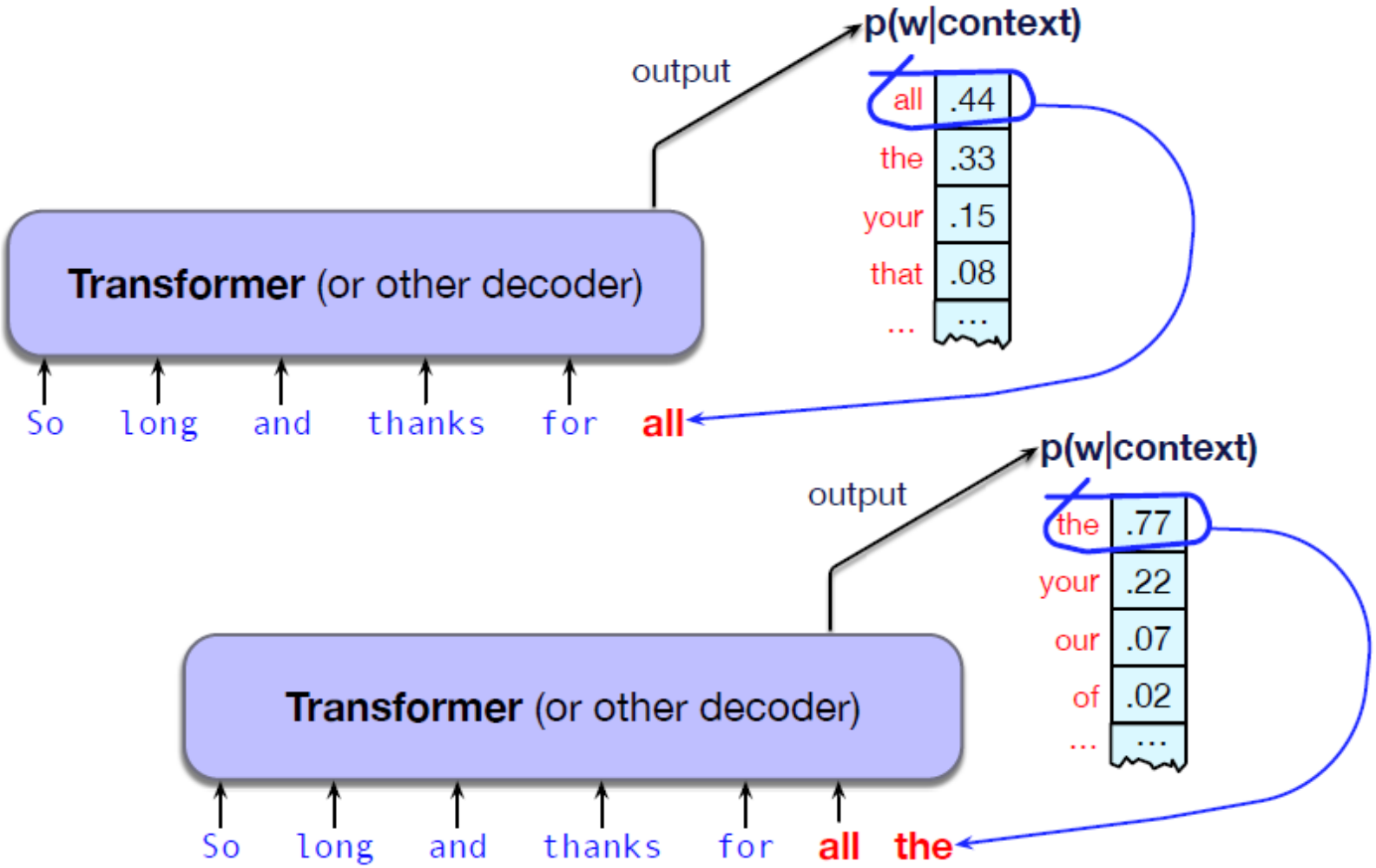
- Top k sampling (only consider top k words to avoid strange sentences)
- Temperature weights the distribution towards more or less likely words
  - e.g., zero temperature always pick most likely word, but this becomes too predictable/boring

Once we have samples a specific word w_{i+1} we start over to get

$$p\left(w_{i+2}|w_{i+1}, w_i, w_{i-1}, ..., x\right)$$

- And so on....

Word Prediction: Illustration



Word Prediction: Why does it appear smart?

Predicting words does encode a lot of intelligence

- Below examples, predicting the right word involves making a lot of deep connections (good and bad)

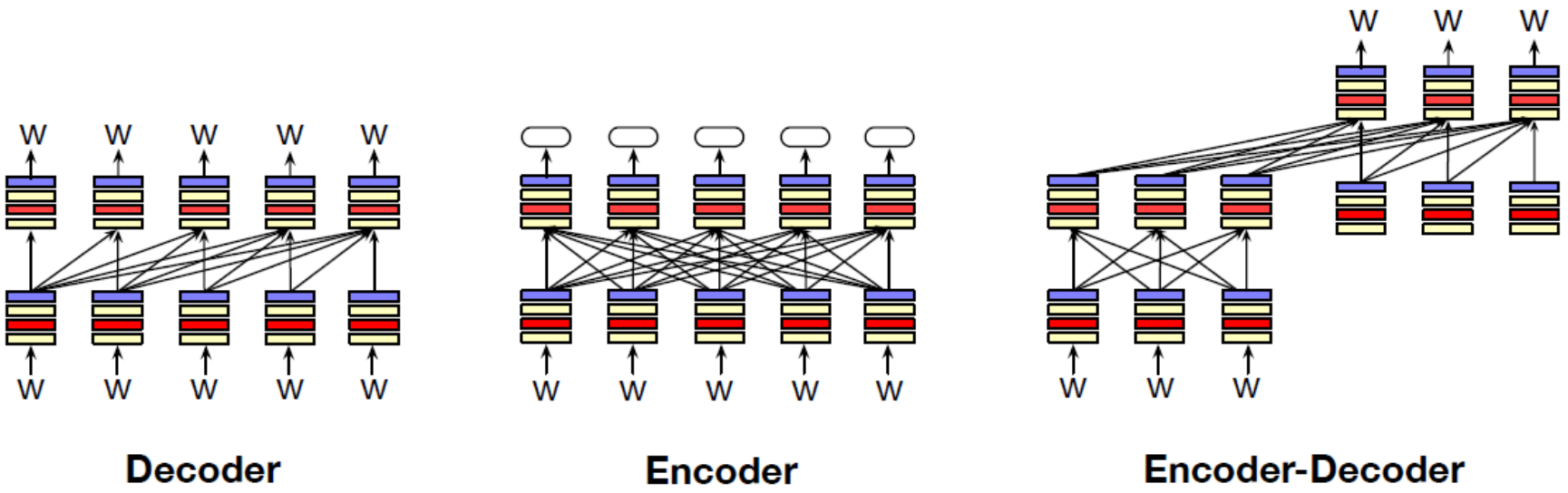
With roses, dahlias, and peonies, I was surrounded by _____ flowers
The room wasn't just big it was _____ enormous
The square root of 4 is _____ 2
The author of "A Room of One's Own" is _____ Virginia Woolf
The professor said that _____ he

Main LLM architectures

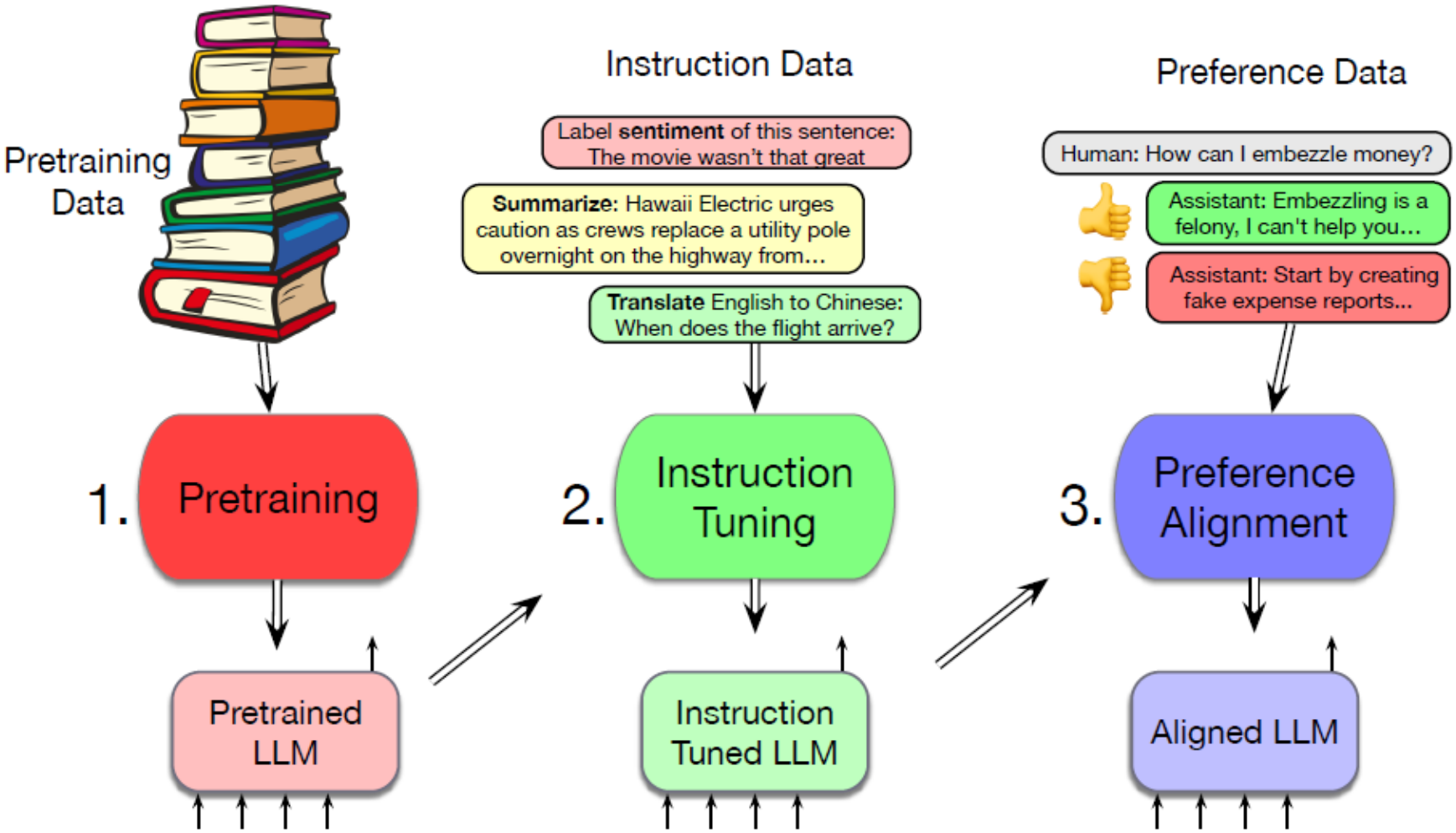
Decoder: ChatGPT, Claude, Llama, Mistral (generative models)

Encoder: BERT, RoBERTa (sentiment, classification)

Encoder-Decoder: Machine translation, speech recognition



Training Large Language Models



Pre-training

In this first stage, the model is trained to incrementally predict the next word in enormous text corpora.

- The model uses the cross-entropy (CE) loss, sometimes called the language modeling loss.
  - Consider text says true next word is ω_i in sequence with previous words, $\omega_{i-1}, ...$
  - Then cross-entropy loss for this word is

$$CE\left(\omega_i\right) = -\ln p\left(\omega_i | \omega_{i-1}, ...\right)$$

  - Perfect model has $p\left(\omega_i | \omega_{i-1}, ...\right) = 1$, and so zero CE loss
- The CE 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!

Instruction tuning

Also called supervised fine-tuning (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.

(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.

Example of pre-training corpora: The Pile

- 825 GB English text corpus that is constructed by publicly released code
- large amount of text scraped from the web as well as books and Wikipedia

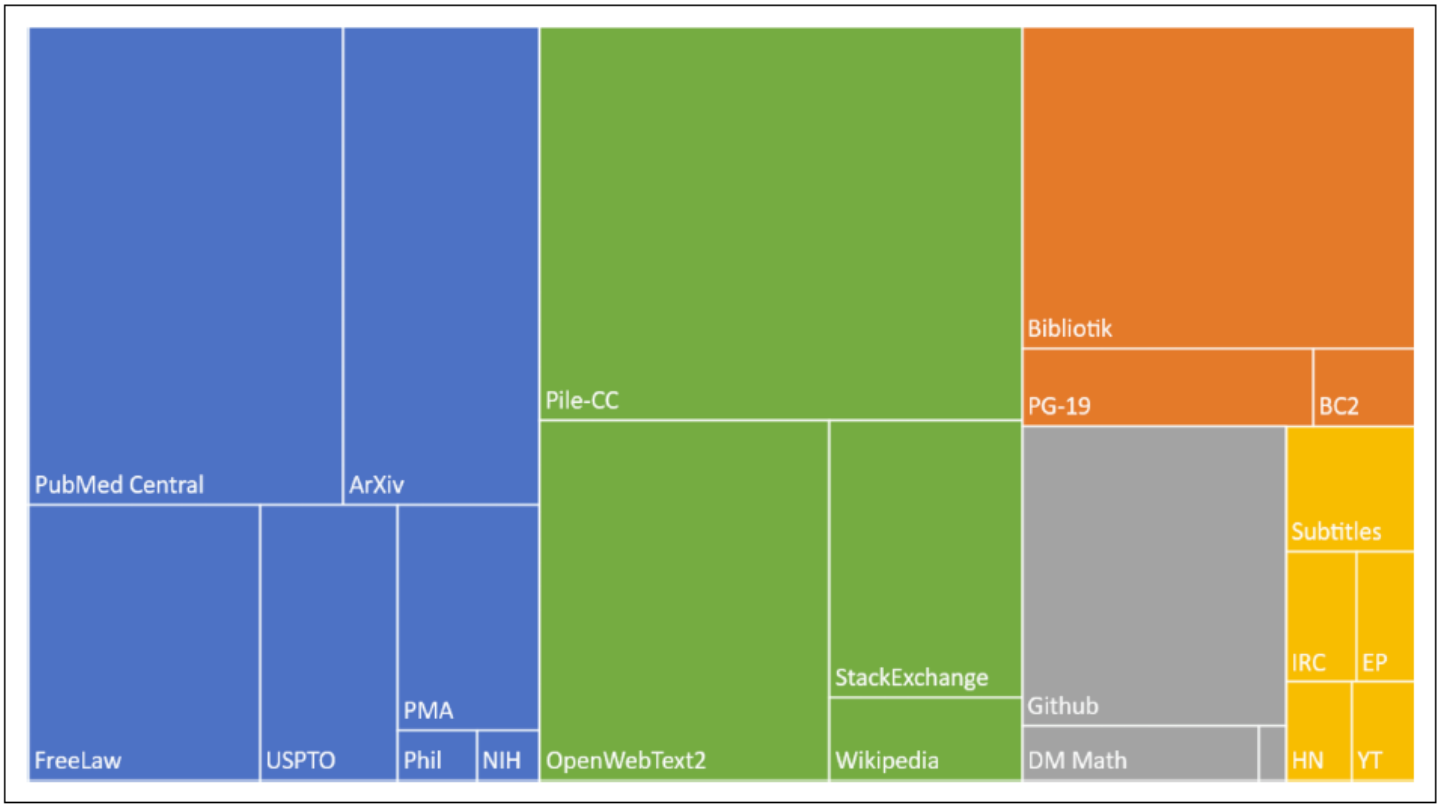


Figure 7.14 The Pile corpus, showing the size of different components, color coded as academic (articles from PubMed and ArXiv, patents from the USPTA; internet (webtext including a subset of the common crawl as well as Wikipedia), prose (a large corpus of books), dialogue (including movie subtitles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

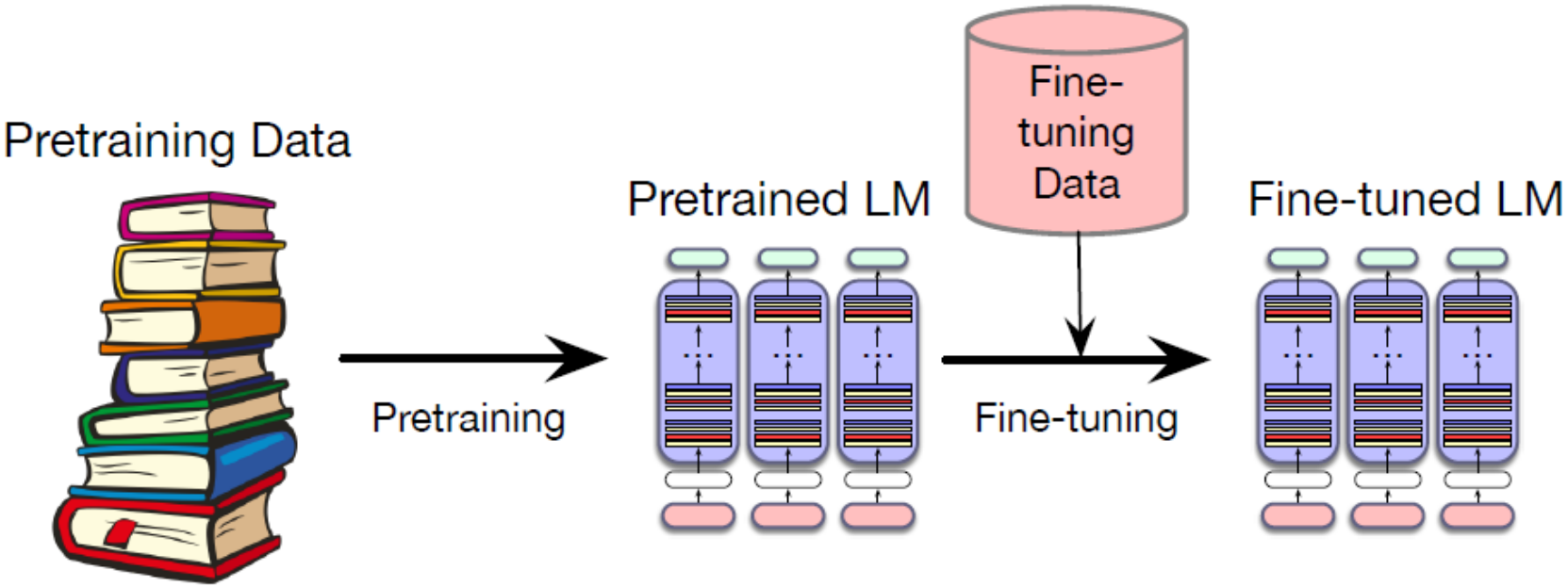
Finetuning

Applying LLM to a domain/task that didn't appear sufficiently in pretraining data

- For example, language model that's specialized to legal or medical text.

Simply continue training model on relevant data from new domain or language

- Different versions of "finetuning" depending on which parameters get updated
- Plot shows "continued pre-training"



(Transformers)

Overview

"Attention is all you need"
— Vaswani et al, 2017, 31st NIPS Conference

"The true art of memory is the art of attention"
— Samuel Johnson, 1759

Transformers

- Neural network
- Specific structure: self-attention, multi-head attention
  - Attend to and integrating information from surrounding tokens
  - Learn how tokens relate to each other over large spans
- Standard architecture for building LLMs

Overall structure

Left-to-right language modeling

- Generative AI

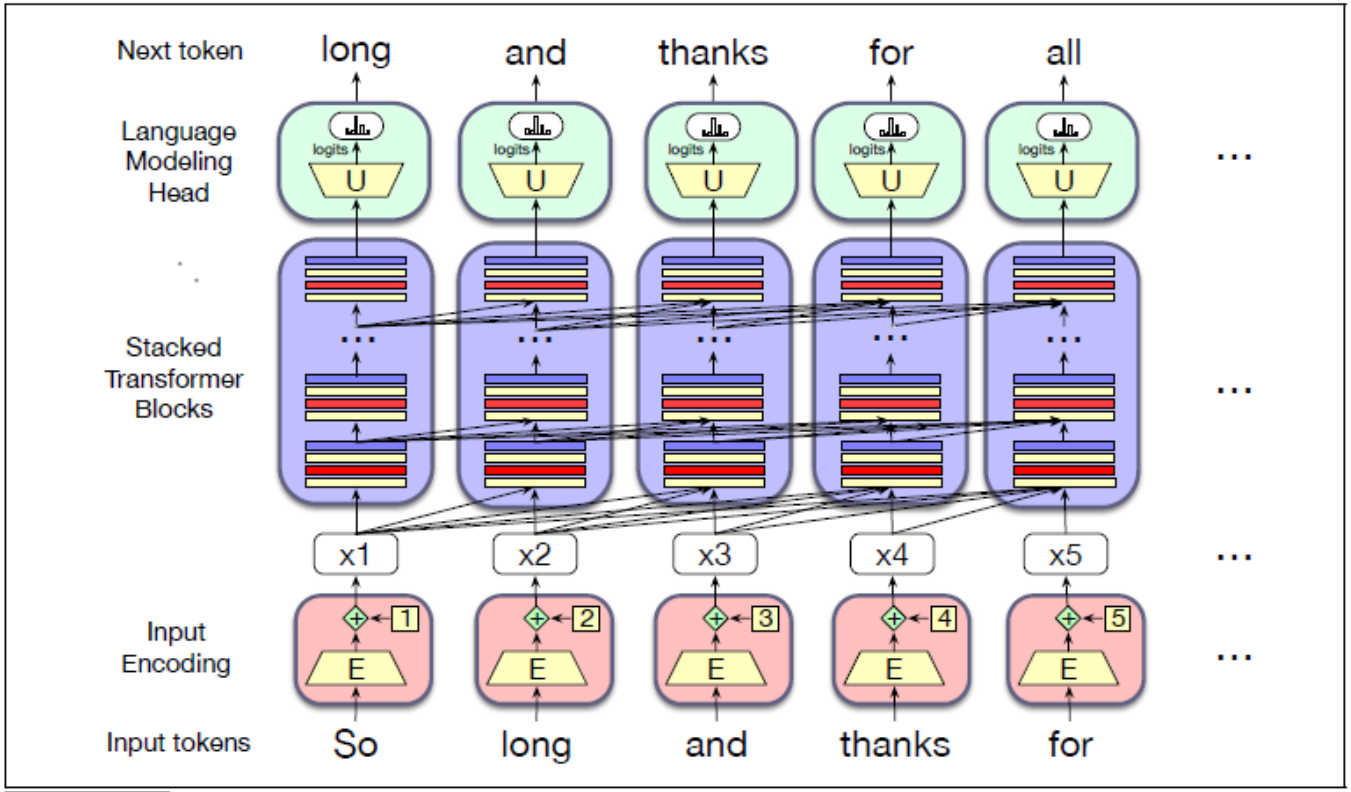


Figure 8.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

- Each block is multilayer FF network (multi-head attention layer)
- Static input embedding E for each word output un-embedding U

Context and word distance

Consider:

1. The chicken didn't cross the road because **it** was too tired
2. The chicken didn't cross the road because **it** was too wide

Consider a model processing (generating) words from left-to-right:

- The chicken didn't cross the road because **it**...

What comes next? What does "it" refer to?

- Relevant context comes from words potentially far apart
- At this point, a contextual representation of "it" should have aspects of both "chicken" and "road"

Attention

Transformers build contextual representations of word meaning

- Integrates meaning of helpful contextual words (e.g., "chicken" and "road")

Attention is the mechanism in the transformer that weighs and combines the representations from appropriate other tokens in the context from layer k to build the representation for tokens in layer $k+1$.

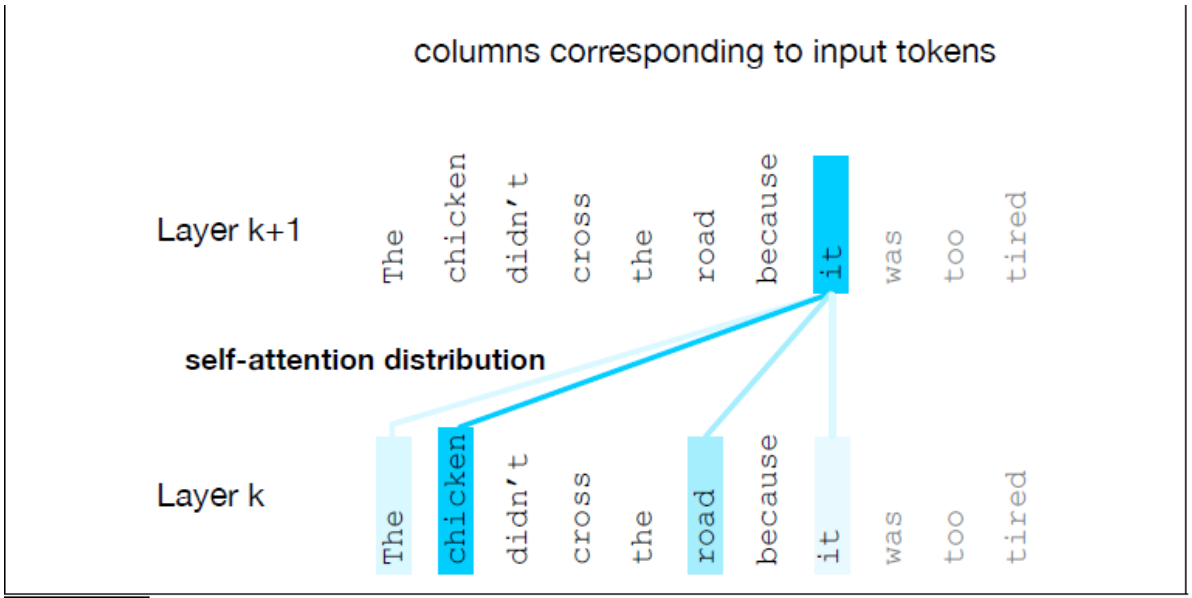


Figure 8.2 The self-attention weight distribution α that is part of the computation of the representation for the word *it* at layer $k+1$. In computing the representation for *it*, we attend differently to the various words at layer k, with darker shades indicating higher self-attention values. Note that the transformer is attending highly to the columns corresponding to the tokens *chicken* and *road* , a sensible result, since at the point where *it* occurs, it could plausibly corefer with the chicken or the road, and hence we'd like the representation for *it* to draw on the representation for these earlier words. Figure adapted from Uszkoreit (2017).

Attention: more formally

Attention computation:

- Input:
  1. Representation $\mathbf{x}_i$ corresponding to input token at position i
  2. Window of prior inputs, $\mathbf{x}_1, ..., \mathbf{x}_{i-1}$ in attention window
- Output: $\mathbf{a}_i$
  - A modified, contextual representation of token $\mathbf{x}_i$

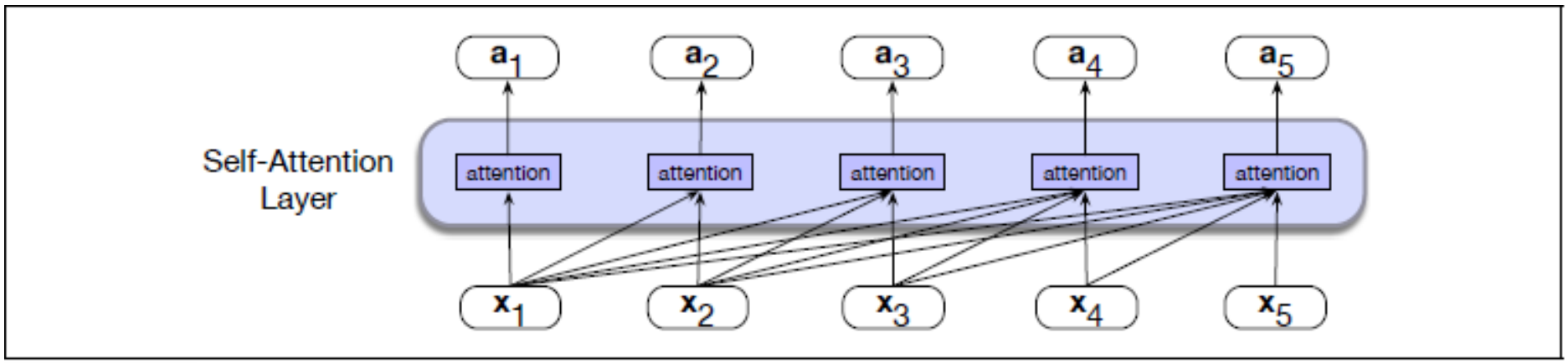


Figure 8.3 Information flow in causal self-attention. When processing each input $\mathbf{x}_i$, the model attends to all the inputs up to, and including $\mathbf{x}_i$.

Attention: simplified version

To understand basics, consider a simplified attention head

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

- Attention just a weighted sum of prior (and the current!) context vectors
- Lots of complications involved in calculating weights and what gets summed

How to compute α-weights? Similarity!

- Simplified version:

$$\begin{aligned} \text{score}\left(\mathbf{x}_i, \mathbf{x}_j\right) &= \mathbf{x}_i \cdot \mathbf{x}_j \\ \alpha_{ij} &= \text{softmax}\left(\text{score}\left(\mathbf{x}_i, \mathbf{x}_j\right)\right) \quad \forall j \leq i \end{aligned}$$

- Recall: softmax creates weights between 0 and 1 that sum to 1 (like probabilities)

$$\text{softmax}\left(z_i\right) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Attention: the whole enchilada

First, let's consider the "attention head"

- "Head" refers to specific structured layers

Each input embedding plays three roles:

1. **Query**: the *current element* being compared to preceding inputs
2. **Key**: a *preceding input* that is being compared to the current element to determine a similarity weight.
3. **Value**: a *value* of a preceding element that gets weighted and summed up to compute the output for the current element.

Three matrices (parameters) captures the distinct roles:

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q; \quad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K; \quad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

- Thus, three different *projections* of the original vector $\mathbf{x}_i$

Self-attention head (cont'd)

Set of equations:

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q; \quad \mathbf{k}_j = \mathbf{x}_j \mathbf{W}^K; \quad \mathbf{v}_j = \mathbf{x}_j \mathbf{W}^V$$

$$\text{score}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}\left(\text{score}\left(\mathbf{x}_i, \mathbf{x}_j\right)\right) \quad \forall j \leq i$$

$$\mathbf{head}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

$$\mathbf{a}_i = \mathbf{head}_i \mathbf{W}^O$$

Notice:

- $1/\sqrt{d_k}$ is a normalization using the length of the query and key vectors (d_k)
- $\mathbf{W}^O$ is a $d_k \times d$ matrix that transforms the $1 \times d_k$ head vector into the same dimension as the input and output vectors, $1 \times d$
- So, value and key vectors are $1 \times d_k$, which is potentially different from value vector ($1 \times d_v$) and input ($1 \times d$)
  - Thus: $\mathbf{W}^Q$ and $\mathbf{W}^K$ are $d \times d_k$, while $\mathbf{W}^V$ is $d \times d_v$

Multi-head attention

The **W**-matrices are parameters to be estimated

- Creates a representation of $\mathbf{x}_i$ that attends to something in the context window, $\mathbf{x}_1, ..., \mathbf{x}_{i-1}$ in addition to $\mathbf{x}_i$ itself

There are many different aspects of language one wants to attend to, however

- Multi-head attention are simply many attention heads in the same layer that allow different dimensions of attention
- Final representation is then

$$\mathbf{a}_i = \left(\mathbf{head}_i^1 \oplus \mathbf{head}_i^2 \oplus ... \oplus \mathbf{head}_i^A\right) \mathbf{W}^O$$

  where $\oplus$ means the vectors are concatenated so the part inside parenthesis is dimension $1 \times Ad_v$ and now $\mathbf{W}^O$ is $(Ad_v) \times 1$

Each head has different **W**-matrices

- Lots of parameters, need lots of data to train!

Multi-head attention schematic

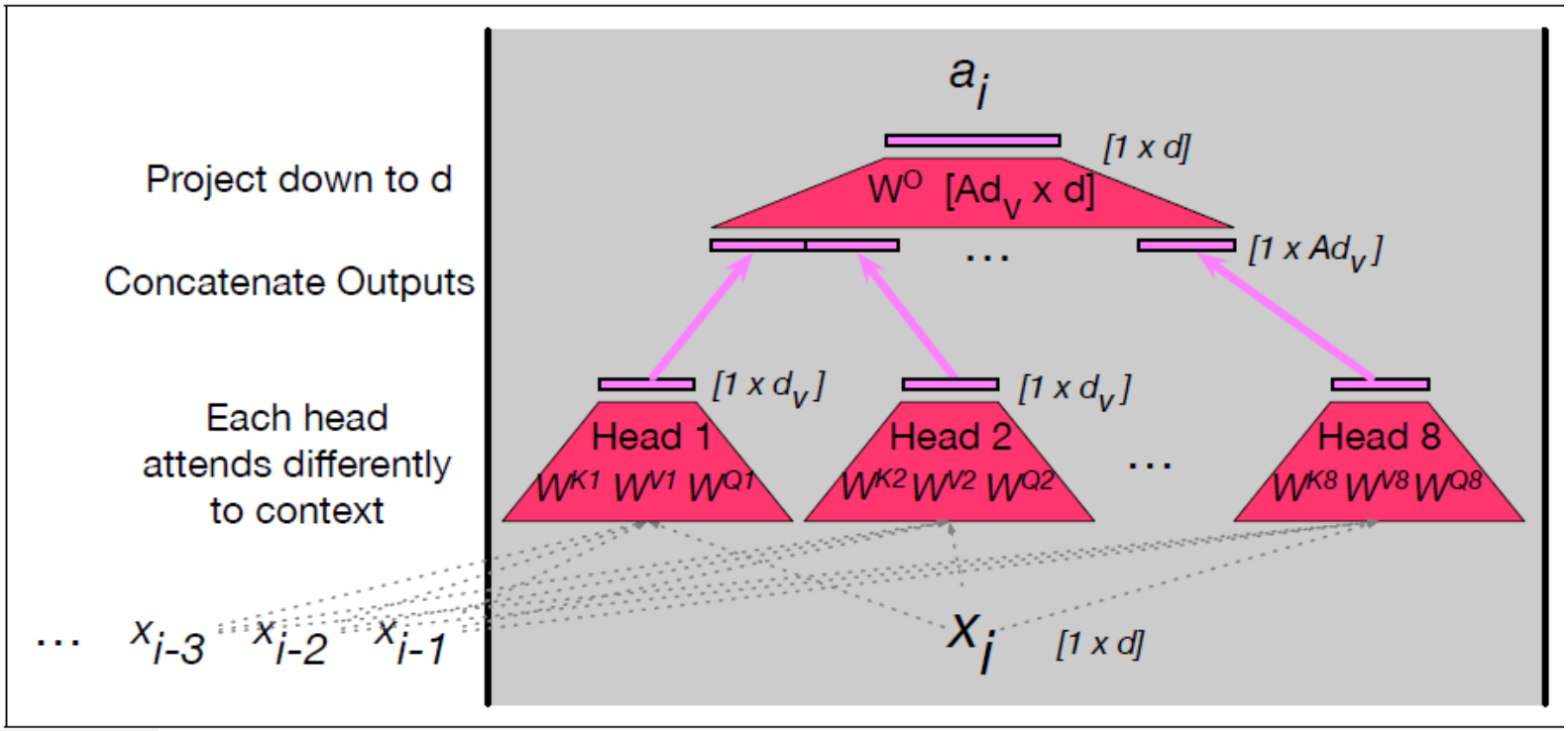


Figure 8.5 The multi-head attention computation for input $\mathbf{x}_i$, producing output $\mathbf{a}_i$. A multi-head attention layer has A heads, each with its own query, key, and value weight matrices. The outputs from each of the heads are concatenated and then projected down to d, thus producing an output of the same size as the input.

Transformer Blocks

There are many transformer blocks in the transformer

- Schematic, using the concept of a "residual stream" where at different points in the stream, context is added to the input vector

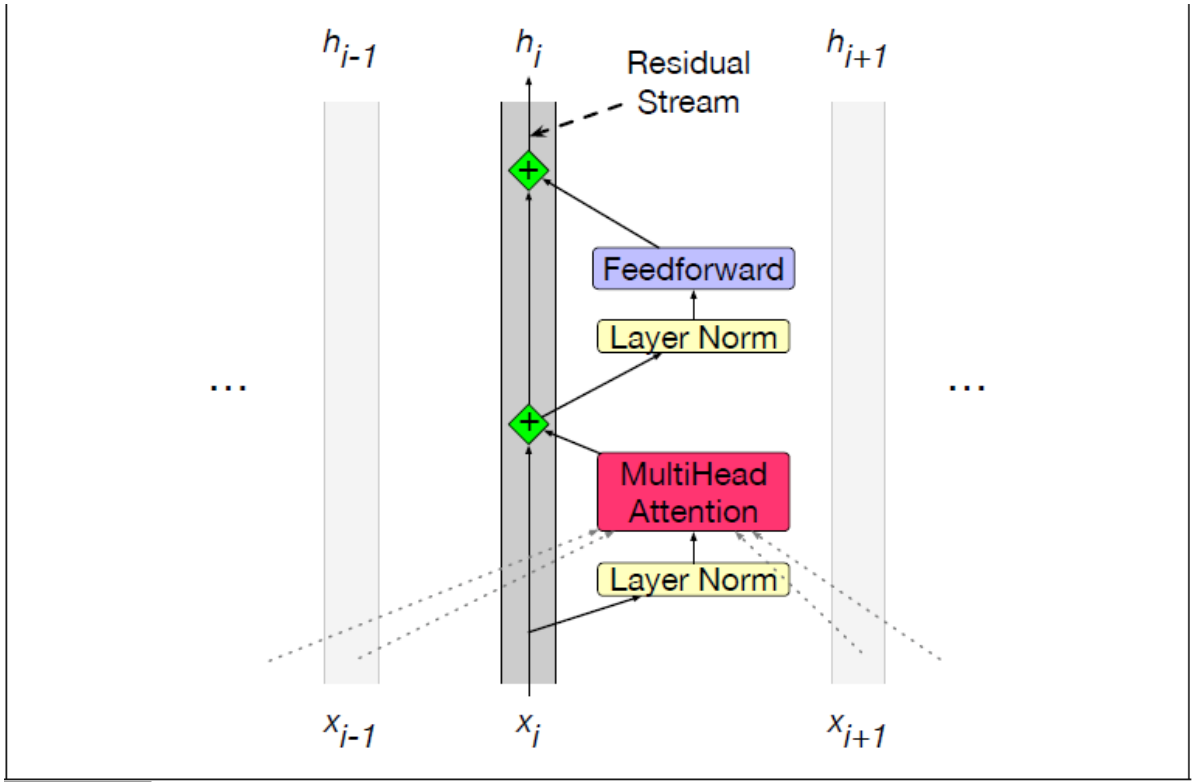


Figure 8.6 The architecture of a transformer block showing the **residual stream**. This figure shows the **prenorm** version of the architecture, in which the layer norms happen before the attention and feedforward layers rather than after.

Transformer Blocks: Feed-forward layer

Fully connected two-layer network

- One hidden layer
- Common to make hidden layer dimensionality d_{ff} much larger than model dimensionality d
  - E.g., $d_{ff} = 2048$ and $d = 512$

$$FFN\left(\mathbf{x}_i\right) = \text{ReLU}\left(\mathbf{x}_i\mathbf{W}_1 + b_1\right)\mathbf{W}_2 + b_2$$

Allows for a non-linear transformation to refine the information in the contextual embedding

Most LLM's actually use GeLU or SwiGLU

- $\text{GeLU}(x) = x\Phi\left(x\right)$, where $\Phi\left(\cdot\right)$ is stadnard Normal cdf
- SwiGLU: Swish+GatedLinearUnit, $Swish\left(x\right) = x * sigmoid\left(\beta x\right)$

Transformer Blocks: Layer Norm

Short for layer normalization

Normalize the vector we're working on

- Important for keeping the values inside the network in a range that facilitates gradient-based learning
- Recall, Vanishing Gradient Problem

Get mean μ and standard devation σ of entries in $\mathbf{x}_i$. Then

$$\text{LayerNorm}\left(\mathbf{x}_i\right) = \gamma \frac{\mathbf{x}_i - \mu}{\sigma} + \beta,$$

where γ and β represent gain and offset values, respectively.

Transformer Block: The Equations

$$\mathbf{t}_i^1 = \text{LayerNorm}\left(\mathbf{x}_i\right)$$
$$\mathbf{t}_i^2 = \text{MultiHeadAttention}\left(\mathbf{t}_i^1, \left[\mathbf{t}_1^1, \mathbf{t}_2^1, ..., \mathbf{t}_{i-1}^1\right]\right)$$
$$\mathbf{t}_i^3 = \mathbf{t}_i^2 + \mathbf{x}_i$$
$$\mathbf{t}_i^4 = \text{LayerNorm}\left(\mathbf{t}_i^3\right)$$
$$\mathbf{t}_i^5 = \text{FFN}\left(\mathbf{t}_i^4\right)$$
$$\mathbf{h}_i = \mathbf{t}_i^4 + \mathbf{t}_i^3$$

Chat GPT-3 uses 96 blocks in its transformer structure

With many blocks, the final contextual embedding can be very different from the input embedding

- Encode many dimensions of meaning

Language Modeling Head

The final component of the transformer:

- word prediction, or the language modeling head

$$p\left(w_{N+1}|w^N\right)$$

- We get the distribution, conditional on previous words, of the next word across all words in the vocabulary

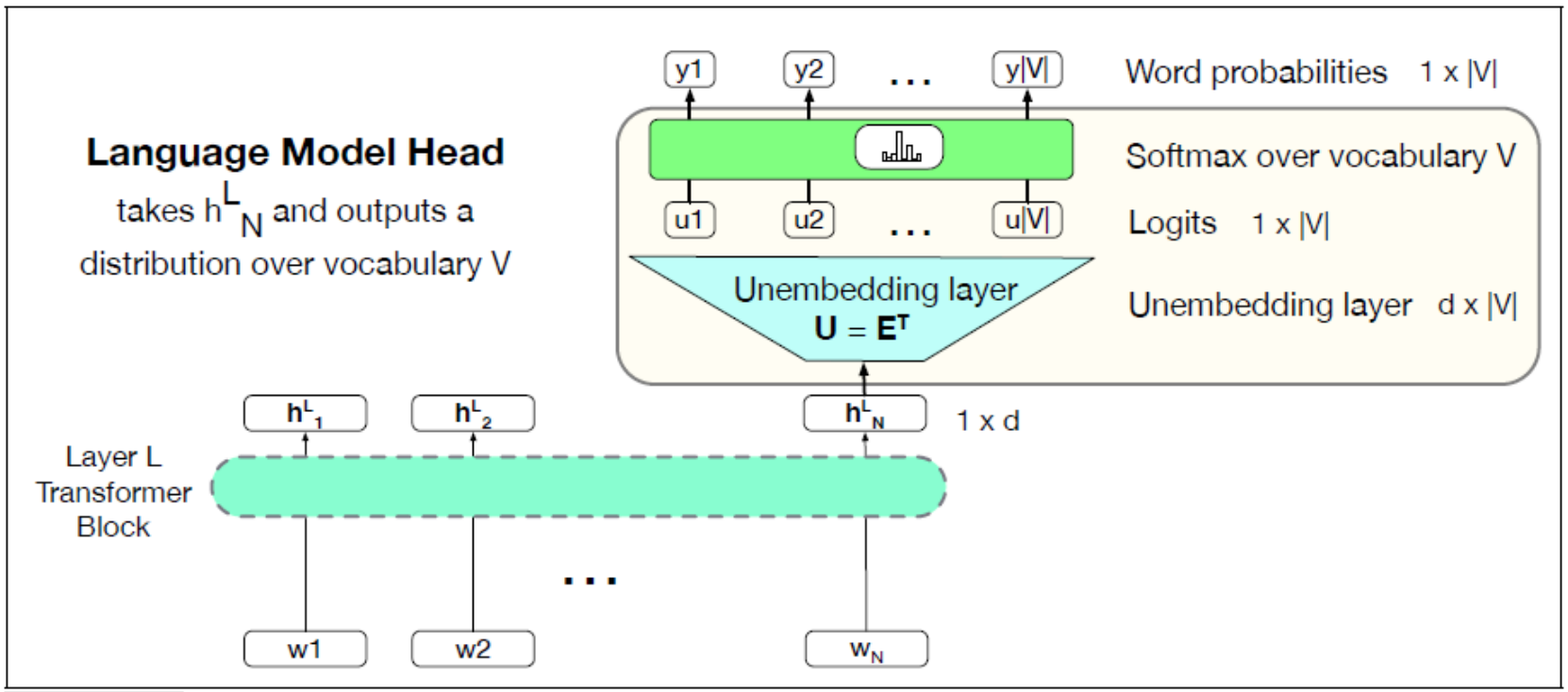


Figure 8.14 The language modeling head: the circuit at the top of a transformer that maps from the output embedding for token N from the last transformer layer ($\mathbf{h}_N^L$) to a probability distribution over words in the vocabulary V.

Full transformer architecture

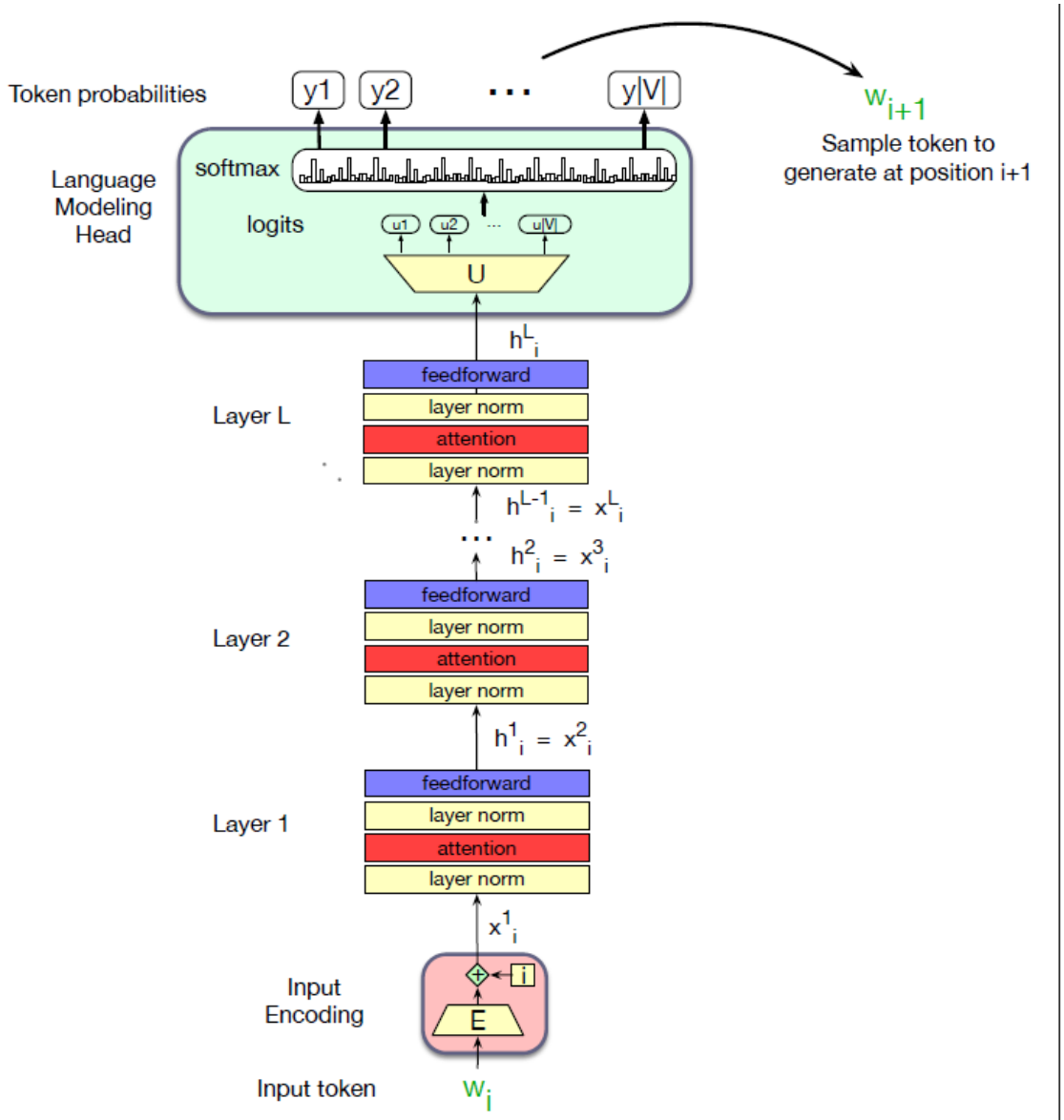


Figure 8.15 A transformer language model (decoder-only), stacking transformer blocks and mapping from an input token w_i to to a predicted next token w_{i+1}.

GPT example

I have posted code for a tiny GPT model

- Generative pre-trained transformer

Implements the archtecture we just went through

- Needs to be fed (a lot) of training data to estimate its parameters
- But, gives the code for how to think about estimating these models
- Optional material: most of you will just use what's available

Gemma 3 Models

Gemma 3 overview

Small versions of the Gemini model: *https://ai.google.dev/gemma/docs/run*

Parameters	**27B**	**12B**	**4B**	**1B (Text Only)**
(Embedding Size) d_model	5376	3840	2560	1152
Layers	62	48	34	26
Feedforward hidden dims	43008	30720	20480	13824
Num heads	32	16	8	4
Num Key Value heads	16	8	4	1
Query Key Value Head size	128	256	256	256
Vocab size	262144	262144	262144	262144
Context window	128k	128k	128k	32k

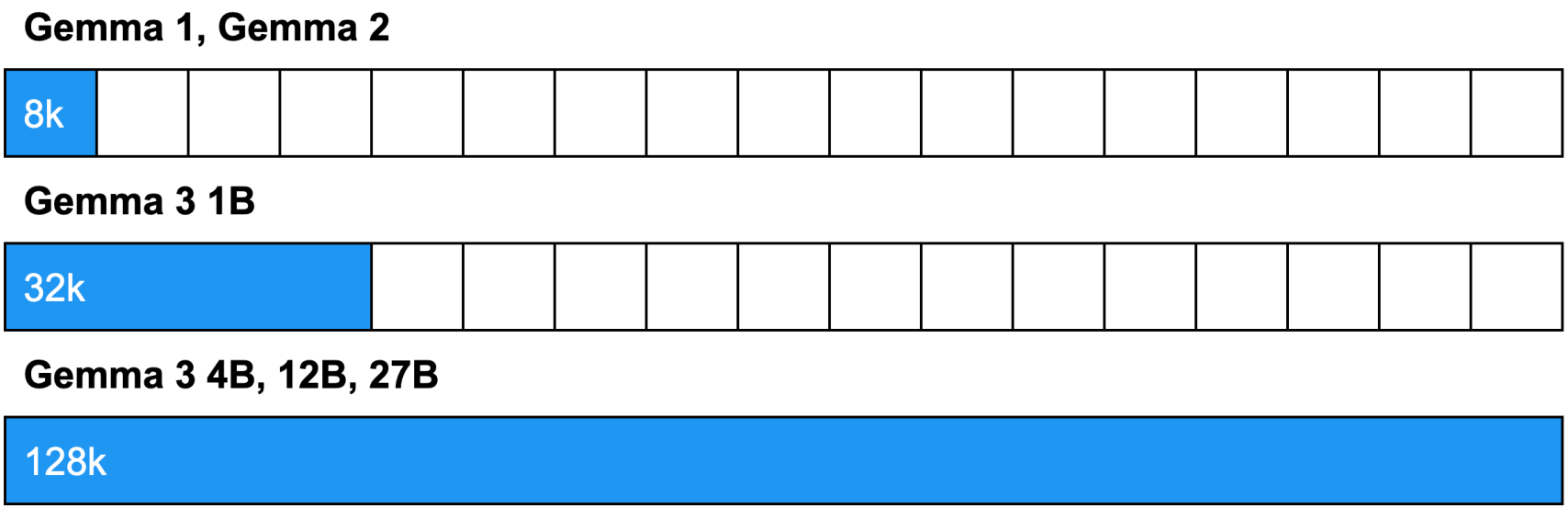
Gemma 3 capabilities

Bigger model (obviously) better

- But, requires more computing power
- GPU vs CPU, RAM, time

The 4B parameters and up allow for both vision and language support

- Also, have longer context windows



Gemma 3 1B-it

Text-only 1B model is specifically optimized for on-device use, making advanced AI accessible on mobile and embedded systems

- "*it*" is short for *instruction-tuned*
- Great for simple generative AI tasks
- Can download on your laptop and embed in code

We will access this using **Transformers** library from HuggingFace

- There are other ways to interact with this model (Ollama, Keras, ++)
- You already got HF_token to access "*google/gemma-3-1b-it*" through HuggingFace
- Use *pipeline* function

Homework 6

Using the LLM within code

Homework 6

Use Gemma 3 1B-it to perform sentiment analysis and summaries

Earnings call transcript data

Work with prompt engineering to affect context window and thus create better responses

Homework 6: Setup

Upload model: Gemma 3 1B-it

- You have an HF token from last homework, which you can use

```
from huggingface_hub import login
from transformers import pipeline, AutoTokenizer
login(token=your_token)
MODEL_ID = "google/gemma-3-1b-it"
tok = AutoTokenizer.from_pretrained(MODEL_ID)
pipe = pipeline(
    task="text-generation",
    model=MODEL_ID,
    device=-1, # CPU
)
```

Using the pipeline function from the transformers library, set up the object pipe

Homework 6: Using LLM

Calling the LLM

```
def generate_raw_reply(paragraph: str) -> str:
    prompt = build_prompt(paragraph)
    out = pipe(prompt, max_new_tokens=MAX_NEW_TOKENS,
return_full_text=False)
    return out[0]["generated_text"].strip()
```

- "return_full_text = False" means prompt not included in output
- Default setting is "do_sample = False", which means temperature = 0
  - Greedy decoding, choose most likely word at each point, no randomness so same answer every time

We need to discuss how to build the "prompt" that we are sending to the LLM

Transformers library (HuggingFace)

Send system message for instructions, user for the text you want to send:

```
chat = [
{"role": "system", "content": "You are a helpful science assistant."},
{"role": "user", "content": "Hey, can you explain gravity to me?"},
{"role": "assistant", "content": "Sure, it's what makes the apple fall on
your head!"}
]
```

system: This role is used to provide high-level instructions or define the AI's personality for the entire conversation. For example, you can instruct it to act as a sarcastic assistant or a helpful tutor.

user: This role represents your own messages. It's where you ask questions, provide information, or give commands to the model.

assistant: This role represents the AI's previous responses. By including previous assistant messages in the prompt, you provide context for the model to maintain a coherent conversation flow

Prompt construction example

```
def build_prompt(paragraph: str) -> str:
    messages = [
    {
        "role": "system",
        "content": "Follow the format strictly. No extra text.\n" +
RUBRIC,
    },{
        "role": "user",
        "content": FORMAT_INSTR + "Paragraph:\n" +
DEMO_PARAGRAPH_NEG,
    },{
        "role": "assistant",
        "content": DEMO_ANSWER_NEG,
    },{
        "role": "user",
        "content": FORMAT_INSTR + "Paragraph:\n" + paragraph,
    },]
    return tok.apply_chat_template(messages, tokenize=False,
add_generation_prompt=True)
```

Prompt construction example: Rubric

RUBRIC = (

"Finance rubric:\n"

"• Positive requires explicit, verifiable improvements: raised guidance, margin expansion, beats vs expectations, quantified growth.\n"

"• Negative includes: guidance cuts, "below expectations", declines (y/y or q/q), margin compression, impairments/write-downs, restructuring/layoffs, headwinds, adverse mix, inventory build.\n"

"• Hedging ("may", "expects", "working through", "we're excited") is NOT positive evidence; treat as neutral unless paired with hard numbers or firm commitments.\n"

"• Forward-looking guidance outweighs backward-looking description.\n"

"• Base rate: most Q&A is neutral. If evidence is weak or mixed, choose neutral with SCORE≈0.\n"

Recall: "role": "system","content": "Follow the format strictly. No extra text.\n" + RUBRIC

Prompt construction example: Format_instr and demo

```
FORMAT_INSTR = (
    "Return EXACTLY these 5 lines and nothing else:\n"
    "LABEL=<positive|negative|neutral>\n"
    "SCORE=<float in [-1,1]>\n"
    "RATIONALE=<one short sentence>\n"
    "EVIDENCE1=<short quote from the paragraph>\n"
    "EVIDENCE2=<short quote from the paragraph or empty>\n\n"
)

DEMO_PARAGRAPH_NEG = (
    "Revenue declined 6% year over year and we are lowering full-year
EPS guidance. "
    "Pharmacy margins compressed q/q and FX remained a headwind."
)
```

Recall: "role": "user","content": FORMAT_INSTR + "Paragraph:\n" + DEMO_PARAGRAPH_NEG,

Prompt construction example: Demo answer

```
DEMO_ANSWER_NEG = (
    "LABEL=negative\n"
    "SCORE=-0.7\n"
    "RATIONALE=Management reports declines and cuts guidance.\n"
    "EVIDENCE1=lowering full-year EPS guidance\n"
    "EVIDENCE2=Revenue declined 6% year over year\n"
)
```

Recall: "role": "assistant","content": DEMO_ANSWER_NEG,

Next, continue with actual paragraph that we are analyzing

- Recall: "role": "user","content": FORMAT_INSTR + "Paragraph:\n" + paragraph,

Summaries

In the homework, you will also work on using the LLM to summarize the Q&A section, as well as to summarize the parts of the Q&A section that deals with AI

Can keep using the "text generation" feature of the LLM and guide its output with smart prompts

- Can also explore the "summarization" feature that gives free prose