### Large Language Models

### Introduction to Large Language Models

### Language models

- Remember the simple n-gram language model
	- Assigns probabilities to sequences of words
	- Generate text by sampling possible next words
	- Is trained on counts computed from lots of text
- Large language models are similar and different:
	- Assigns probabilities to sequences of words
	- Generate text by sampling possible next words
	- **Are trained by learning to guess the next word**

- 
- 

### Large language models

- Even through pretrained only to predict words
- Learn a lot of useful language knowledge
- Since training on a **lot** of text

**Decoders Encoders Encoder-decoders** GPT, Claude, BERT family, Flan-T5, Whisper

Llama HuBERT Mixtral



### Three architectures for large language models  $\Gamma$ ectures for large language models









Many varieties!

- Popular: Masked Language Models (MLMs)
- BERT family
- Trained by predicting words from surrounding words on both sides
- Are usually **finetuned** (trained on supervised data) for classification tasks.

### Encoder-Decoders

- Trained to map from one sequence to another
- Very popular for:
	- machine translation (map from one language to another)
	- speech recognition (map from acoustics to words)



### Large Language Models

### Introduction to Large Language Models

### Large Language **Models**

### Large Language Models: What tasks can they do?





# Many tasks can be turned into tasks of predicting words!

# This lecture: decoder-only models

Also called:

- Causal LLMs
- Autoregressive LLMs
- Left-to-right LLMs

• Predict words left to right



### Conditional Generation: Generating text conditioned on previous text!



### The sentiment of the sentence "I like Jackie Chan" is: Prefix Text  $F_{\rm eff}=F_{\rm eff}=F_{\rm eff}=0.1$ models. As each to context and the context and the next to the next to the next to  $\mathbf{r}$

### 2. And see what word it thinks comes next: which is a see which words



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

Framing lots of tasks as conditional generation  $\Gamma$  cases in  $\Gamma$  late of tasks as as wolltipuol prediction. Fianning iots of tasks as conditional generation (for example a  $\sim$  $\mathcal{C}$  cast more complex tasks as word prediction. Consider  $\mathcal{C}$  as word prediction. Consider  $\mathcal{C}$  as we consider Fiaming iots of tasks as conditional generation (for example a  $\sim$ 

QA: "Who wrote The Origin of Species" a simple factual and must give a textual and must give a textual answer  $\mathcal{L}$ 

- 1. We give the language model this string: of question answering as word prediction by giving a language model a question and 1. We give the language model this string. by giving a language model a language model a suggesting that and a token like  $\mathcal{A}$ : suggesting that an answer L. We give 1. We give the next of  $\mathcal{L}$ 
	- Q: Who wrote the book ''The Origin of Species"? A: Who wrote the book ''The Origin of Species"?  $\overline{a}$  and  $\overline{b}$  is the book of  $\overline{b}$  of  $\overline{b}$  or  $\overline$ If we ask a language model to compute the probability distribution over possible  $\mathbf{F}$
- 2. And see what word it thinks comes next: If we ask a language model to compute the probability distribution over possible Z. And see what y 2. And see what w
	- $P(w|Q:$  Who wrote the book ''The Origin of Species"? A:) *w*<sup>|</sup>O: Who wrote the book ''The Origin of Species"? A:
- 3. And iterate: and look at which words *w* have high probabilities, we might expect to see that **S.** And iterate:
	- *P*(*w|*Q: Who wrote the book ''The Origin of Species"? A: Charles) *P*(*w|*Q: Who wrote the book ''The Origin of Species"? A: Charles)



# Summarization

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For \$89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

His website and social media accounts claim to have filled more than 133 orders for snow – more His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity. According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. [...] choking their yards and sidewalks as a nuisance, but warning saw an opportunity. According to Boston.com, it an started a few weeks ago, when waring and his wife were showjournalism the stuff to friends and family in warmer states, and family in warmer states, and an idea was born.

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box - enough  $\overline{\mathsf{F}}$  showhalls the save Rut not if you live in New England or surrounding states  $\sum_{i=1}^{n}$ for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

But not if you live in New England or surrounding states. "We will not ship snow to any states in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging snow!" But I'D I'D But walls, in Says.<br>Original and Rut not if you live in New England or surrounding states. "We will not ship snow to any states. Original

Summary

# LLMs for summarization (using tl;dr)

Generated Summary





### Large Language **Models**

### Large Language Models: What tasks can they do?



### Large Language Models

# Sampling for LLM Generation

# Decoding and Sampling

This task of choosing a word to generate based on the model's probabilities is called **decoding**.

• choose random words according to their probability assigned by the model.

The most common method for decoding in LLMs: **sampling**. Sampling from a model's distribution over words:

After each token we'll sample words to generate according to their probability *conditioned on our previous choices*,

• A transformer language model will give the probability

# Random sampling

 $R$ andom samnling a sequence of words  $R$ until we his to mean to we have to mean to mean to mean to mean to part of part is part of the mean of part of  $\overline{p}$ The algorithm above is called random sampling, and it turns out random sampling, and it turns out random sampling, and pling doesn't work well enough. The problem is that even though random sampling

pling from the distribution *p*(*x*):

 $i \leftarrow 1$  $w_i \sim p(w)$ while  $w_i$  != EOS  $i \leftarrow i+1$  $w_i \sim p(w_i | w_{$ 

probability model that tells us this probability model that tells us this probability. The probability is prob

is mostly going to generate sensible, high-probable words, there are many odd, low-

# Random sampling doesn't work very well

- Even though random sampling mostly generate sensible, high-probable words,
- There are many odd, low- probability words in the tail of the distribution
- Each one is low- probability but added up they constitute a large portion of the distribution So they get picked enough to generate weird

sentences



# Factors in word sampling: quality and diversity

### Emphasize **high-probability** words

- + **quality**: more accurate, coherent, and factual,
- **- diversity**: boring, repetitive.

Emphasize **middle-probability** words + **diversity**: more creative, diverse, - **quality**: less factual, incoherent

# Top-k sampling:

1. Choose # of words *k* 

2. For each word in the vocabulary *V* , use the language model to compute the likelihood of this word given the context *p*(*wt* |w<*<sup>t</sup>* )

3. Sort the words by likelihood, keep only the top *k* most probable words.

4. Renormalize the scores of the *k* words to be a legitimate probability distribution.

5. Randomly sample a word from within these remaining *k* mostprobable words according to its probability.

### Top-p sampling (= nucleus sampling) 2020), is to keep not the top *k* words, but the top *p* percent of the probability mass.  $T = Top-p$  sampling  $(=$  nucleus sampling)

Problem with top-*k: k* is fixed so may cover very different amounts of probability mass in different situations Idea: Instead, keep the top p percent of the probability mass Given a distribution  $P(w_t | w_{< t})$ , the top-*p* vocabulary  $V(p)$ is the smallest set of words such that But by measuring probability rather than the number of words, the hope is that the measure with top-x. A is lixed so may cover very different amounts of probability mass in different situations Given a distribution  $P(w_t | w_{< t})$ , the top-*p* vocabulary  $V(p)$  is the smallest set of words such that

words such that

10.2.3 Temperature sampling in the sampling samplin

Holtzman et al., 2020

$$
\sum_{w \in V(p)} P(w|\mathbf{w}_{
$$

# Temperature sampling

Reshape the distribution instead of truncating it Intuition from thermodynamics,

- a system at high temperature is flexible and can explore many possible states,
- a system at lower temperature is likely to explore a subset of lower energy (better) states.

### In **low-temperature sampling**,  $(\tau \leq 1)$  we smoothly

- increase the probability of the most probable words
- decrease the probability of the rare words.



of the most probability of the most probability of the probability of the probability of the probability of the rare words.

Divide the logit by a temperature parameter τ before<br>passing it through the softmax. passing it through the softmax. volumber and the following the following (repeated from  $\sigma$ ) is the following (repeated from (?)): **Divide the logit by a temperature parameter**  $\tau$  **before** 

Why does this work? Why does this work? When the distribution doesn't change much. Why doesn't change much. The



We implement this intuition by simply dividing the logit by a temperature param-

### Temperature sampling  $\Gamma$  instead first divide the logitarist divide the probability vector  $\Lambda$  $y = softmax(u/\tau)$  $0 \leq$  τ  $\leq$  1

# Why does this work?

- When τ is close to 1 the distribution doesn't change much.
	- The lower τ is, the larger the scores being passed to the softmax
- Softmax pushes high values toward 1 and low values toward 0. softmax pushes high values toward 1 and low values toward 0.
- Large inputs pushes high-probability words higher and low probability word lower, making the distribution more greedy. I arge inputs pushes high-probability words higher and low probability<br>word lower, making the distribution mare groady.
- As τ approaches 0, the probability of most likely word approaches 1

a distribution with increased probabilities of the most high-probability words and

### Large Language Models

# Sampling for LLM Generation

### Large Language Models

# Pretraining Large Language Models: Algorithm



# The big idea that underlies all the amazing performance of language models

First **pretrain** a transformer model on enormous amounts of text Then **apply** it to new tasks.



# Self-supervised training algorithm

We just train them to predict the next word! 1. Take a corpus of text

2. At each time step *t* 

- i. ask the model to predict the next word
- ii. train the model using gradient descent to minimize the error in this prediction

"**Self-supervised**" because it just uses the next word as the label!

# Intuition of language model training: loss

- Same loss function: **cross-entropy loss**
	- We want the model to assign a high probability to true word *w*
	- $\bullet$  = want loss to be high if the model assigns too low a probability to w
- CE Loss: The negative log probability that the model assigns to the true next word w
	- If the model assigns too low a probability to w
	- We move the model weights in the direction that assigns a higher probability to w

So in this sum, all terms get multiplied by zero except one: the logp the model assigns to the correct next word, so: next word. This is represented as a one-hot vector corresponding to the vocabulary  $\blacksquare$  So in this sum, all terms get multiplied by zero except of

model assigns to the next word in the next word in the training sequence. The training sequence in the training sequence.

### **Letter Cross-entropy loss for language modeling** Recall that the cross-entropy loss measures the difference between a predicted

model to minimize the error in predicting the true next word in the training sequence,

**CE loss**: difference between the **correct** probability distribution and the **predicted** distribution In the case of language modeling, the correct distribution y*<sup>t</sup>* comes from knowing the **next We CE loss:** difference between the <mark>correct</mark> probability distribution and the <mark>predicted</mark> distribution

at time *t* the CE loss in (10.5) can be simplified as the negative log probability the



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

The correct distribution  $y_t$  knows the next word, so is 1 for the actual next at time word and 0 for the others. In the correct distribution  $y_t$  knows the next word, so is  $1$  for the actual next<br>word and 0 for the others  $\blacksquare$ 

$$
L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]
$$

Thus at each word position *t* of the input, the model takes as input the correct se-

### Teacher forcing

- At each token position *t*, model sees correct tokens  $w_{1:t}$ ,
	- Computes  $loss$  (-log probability) for the next token  $w_{t+1}$
- At next token position t+1 we ignore what model predicted for  $W_{t+1}$ 
	- Instead we take the **correct** word  $w_{t+1}$ , add it to context, move on

# Training a transformer language model

…





…

…

…

…

### Large Language Models

# Pretraining Large Language Models: Algorithm

### Large Language Models

### Pretraining data for LLMs
### LLMs are mainly trained on the web

Common crawl, snapshots of the entire web produced by the non- profit Common Crawl with billions of pages

- Colossal Clean Crawled Corpus (C4; Raffel et al. 2020), 156 billion tokens of English, filtered
- What's in it? Mostly patent text documents, Wikipedia, and news sites

### The Pile: a pretraining corpus



### dialog

## Filtering for quality and safety

Quality is subjective

- Many LLMs attempt to match Wikipedia, books, particular websites
- Need to remove boilerplate, adult content
- Deduplication at many levels (URLs, documents, even lines) Safety also subjective
- Toxicity detection is important, although that has mixed results
- Can mistakenly flag data written in dialects like African American English

## What does a model learn from pretraining?

- There are canines everywhere! One dog in the front room, and two dogs
- It wasn't just big it was enormous
- The author of "A Room of One's Own" is Virginia Woolf
- The doctor told me that he
- The square root of 4 is 2



Text contains enormous amounts of knowledge Pretraining on lots of text with all that knowledge is what gives language models their ability to do so much

### But there are problems with scraping from the web

- Not clear if fair use doctrine in US allows for this use
- This remains an open legal question

**Copyright**: much of the text in these datasets is copyrighted

### **Data consent**

- Website owners can indicate they don't want their site crawled **Privacy**:
- Websites can contain private IP addresses and phone numbers

### Large Language Models

### Pretraining data for LLMs

### Large Language Models

### Finetuning

### Finetuning for daptation to new domains

What happens if we need our LLM to work well on a domain it didn't see in pretraining?

Perhaps some specific medical or legal domain?

Or maybe a multilingual LM needs to see more data on some language that was rare in pretraining?

### Finetuning



## "Finetuning" means 4 different things

We'll discuss 1 here, and 3 in later lectures In all four cases, finetuning means: **taking a pretrained model and further adapting some or all of its parameters to some new data**

### 1. Finetuning as "continued pretraining" on new data

- Further train all the parameters of model on new data
	- using the same method (word prediction) and loss function (cross-entropy loss) as for pretraining.
	- as if the new data were at the tail end of the pretraining data
- Hence sometimes called **continued pretraining**

### Large Language Models

### Finetuning

### Large Language Models

### Evaluating Large Language Models



### **Perplexity**

10.4 Evanguage Large Language Models in the Language Models in the Language Models in the Language Models in th

I lust as for n-gram grammars, we use perplexity to measure how assign the LM predicts unseen text and well the LM predicts unseen text

The perplexity of a model θ on an unseen test set is the **inverse Paramelle 1** and probability that θ assigns to the test set, normalized by the test **language model.** Also set length. Also set in an unseen page  $\mathbb{R}$  model q on an unseen perpendicular and unsertainty of a model q on an unsertainty of a model q on an unsertainty of a model q on an unsertainty of an u  $\blacksquare$ 

> For a test set of *n* tokens  $w_{1:n}$  the perplexity is :  $\blacksquare$  For a test set of *n* tokens  $w_i$  the nerolexity is  $\cdot$ set length. For a test set of *n* tokens *w*1:*n*, the perplexity is

> Perplexity As we first saw in Chapter 3, one way to evaluate language models is

$$
\begin{array}{rcl}\n\text{Perplexity}_{\theta}(w_{1:n}) & = & P_{\theta}(w_{1:n})^{-\frac{1}{n}} \\
& = & \sqrt[n]{\frac{1}{P_{\theta}(w_{1:n})}}\n\end{array}
$$

- Probability depends on size of test set
	- Probability gets smaller the longer the text
	- Better: a metric that is **per-word**, normalized by length
- **Perplexity** is the inverse probability of the test set, normalized by the number of words

(The inverse comes from the original definition of perplexity from crossentropy rate in information theory)

Probability range is  $[0,1]$ , perplexity range is  $[1,\infty]$ 

### Why perplexity instead of raw probability of the test set?

### **Perplexity**

- The higher the probability of the word sequence, the lower the perplexity.
- Thus the lower the perplexity of a model on the data, the better the model.
- **Minimizing perplexity is the same as maximizing probability**

Also: perplexity is sensitive to length/tokenization so best used when comparing LMs that use the same tokenizer.

## Many other factors that we evaluate, like:

### **Size**

Big models take lots of GPUs and time to train, memory to store

### **Energy usage**

Can measure kWh or kilograms of CO2 emitted

### **Fairness**

Benchmarks measure gendered and racial stereotypes, or decreased performance for language from or about some groups.

### Large Language Models

### Dealing with Scale

### Scaling Laws

LLM performance depends on

- Model size: the number of parameters not counting embeddings
- Dataset size: the amount of training data
- Compute: Amount of compute (in FLOPS or etc

Can improve a model by adding parameters (more layers, wider contexts), more data, or training for more iterations

The performance of a large language model (the loss) scales as a power-law with each of these three

Loss *L* as a function of # parameters *N*, dataset size *D*, compute budget *C* (if other two are held constant) two are held constant)

### **Loss** *Laws Laws* **Laws Laws Non-embedding parameters** *N D*, and the compute budget *C*, for models training with limited parameters, dataset,

**The Scaling laws can be used early in training to predict what the loss would be if we were** lows (ignoring biases, and with a dimensionality of the input and output dimensionality of the input of the in<br>In put and output dimensionality of the input to add more data or increase model size.

For example, Kaplan et al. (2020) found the following three relationships for

$$
L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}
$$

$$
L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}
$$

$$
L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}
$$

### Number of non-embedding parameters N lows (ignoring biases, and with *d* as the input and output dimensionality of the

model, *d*attn as the self-attention layer size, and *d*ff the size of the feedforward layer):

 $\n$  Thus GPT-3, with  $n = 96$  la The values of *Ncc*, and a *Ncc*, and a *Ncc*, and a *Ncc*, and the exact transformer of the exact transformer on the exact transformer of the exact transformer on the exact transformer on the exact transformer on the exa Thus GPT-3, with  $n = 96$  layers and dimensionality  $d = 12288$ , has  $12 \times 96 \times$ 

scaling laws focus on the relationship with loss.200 millionship w



$$
N \approx 2 d n_{\text{layer}} (2 d_{\text{attn}} + d_{\text{ff}})
$$
  
 
$$
\approx 12 n_{\text{layer}} d^2
$$
  
(assuming  $d_{\text{attn}} = d_{\text{ff}} / 4 = d$ )

Thus GPT-3, with *n* = 96 layers and dimensionality *d* = 12288, has 12 ⇥ 96 ⇥

architecture, tokenization, and vocabulary size, so rather than all the precise values, so rather than all the<br>The precise values, so rather than all the precise values, so rather than all the precise values, which is a r

### KV Cache KV Cache

In training, we can compute attention very efficiently in parallel:  $\frac{1}{2}$ 

But don't want to **recompute** the key and value vectors for all the prior tokens x<*<sup>i</sup>* But don't want to recompute the key and value vectors for all the prior tunciis  $\lambda_{zi}$ 

Instead, store key and value vectors in memory in the KV cache, and then we can just grab them from the cache instead, store key and value vectors in memory in the KV cache, and the prior steps we can just grab them from the cache and vectors in the computation of records when we compute the key we compute the key we compute the key we compute

But not at inference! We generate the next tokens **one at a time!** Unfortunately we can't do quite the same efficient computation in inference as

For a new token x, need to multiply by  $W^{Q}$ ,  $W^{K}$ , and  $W^{V}$  to get query, key, values For a new token x, heed to multiply by W<sup>Q</sup>, W', and W' to get query, key, one at a time. For a new token to a new top and the second to the se

We saw in Fig. ?? and in Eq. ?? (repeated below) how the attention vector can be

$$
\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}
$$

KV cache and value vectors we store them in memory in the KV cache, and then we can just

grab them from the cache when we need them. Fig. 10.7 modifies Fig. ?? to show



**a4**



A

**q1**

**q2**

**q3**

**q4**

N x dk

### A



 $=$ 



## Parameter-Efficient Finetuning

- Enormous numbers of parameters to train
- Each pass of batch gradient descent has to backpropagate through many many huge layers.
- Expensive in processing power, in memory, and in time. Instead, **parameter-efficient fine tuning** (PEFT)
- Efficiently select a subset of parameters to update when finetuning.
- E.g., freeze some of the parameters (don't change them),
- And only update some a few parameters.

Adapting to a new domain by continued pretraining (finetuning) is a problem with huge LLMs.

### LoRA (Low-Rank Adaptation)

- Trransformers have many dense matrix multiply layers
	- Like W<sup>Q</sup>, W<sup>K</sup>, W<sup>V</sup>, W<sup>O</sup> layers in attention
- Instead of updating these layers during finetuning,
	- Freeze these layers
	- Update a low-rank approximation with fewer parameters.

### LoRA

- Consider a **matrix W** (shape [*N* <sup>×</sup> *<sup>d</sup>*]) that needs to be updated during finetuning via gradient descent.
	- Normally updates are ∆W (shape [*N* × *d*])
- In LoRA, we freeze W and update instead a low-rank decomposition of W:
	- A of shape [*N*×*r*],
	- B of shape [ $r \times d$ ], r is very small (like 1 or 2)
	- That is, during finetuning we update A and B instead of W.
	- Replace W +  $\Delta$ W with W + BA.

Forward pass: instead of

$$
\mathbf{h} = \mathbf{x} \mathbf{W}
$$

We do

$$
h = xW + xAB
$$

### LoRA



r



### Large Language Models

### Dealing with Scale

### Large Language<sup>1</sup> Models

### Harms of Large Language Models

Hallucination

## **Chatbots May 'Hallucinate'** More Often Than Many Realize

## What Can You Do When A.I. Lies **About You?**

People have little protection or recourse when the technology creates and spreads falsehoods about them.

### Air Canada loses court case after its chatbot hallucinated fake policies to a customer

The airline argued that the chatbot itself was liable. The court disagreed.







### **Authors Sue OpenAI Claiming Mass Copyright** Infringement of Hundreds of Thousands of Novels

## The Times Sues OpenAI and Microsoft Over A.I. Use of Copyrighted Work

Millions of articles from The New York Times were used to train chatbots that now compete with it, the lawsuit said.







# How Strangers Got My Email Address From ChatGPT's Model





### The New AI-Powered Bing Is Threatening Users.

### **Cleaning Up ChatGPT Takes Heavy Toll on Human Workers**

Contractors in Kenya say they were traumatized by effort to screen out descriptions of violence and sexual abuse during run-up to OpenAI's hit chatbot

### Misinformation

## **Chatbots are generating false and** misleading information about U.S. elections

### Large Language<sup>1</sup> Models

### Harms of Large Language Models