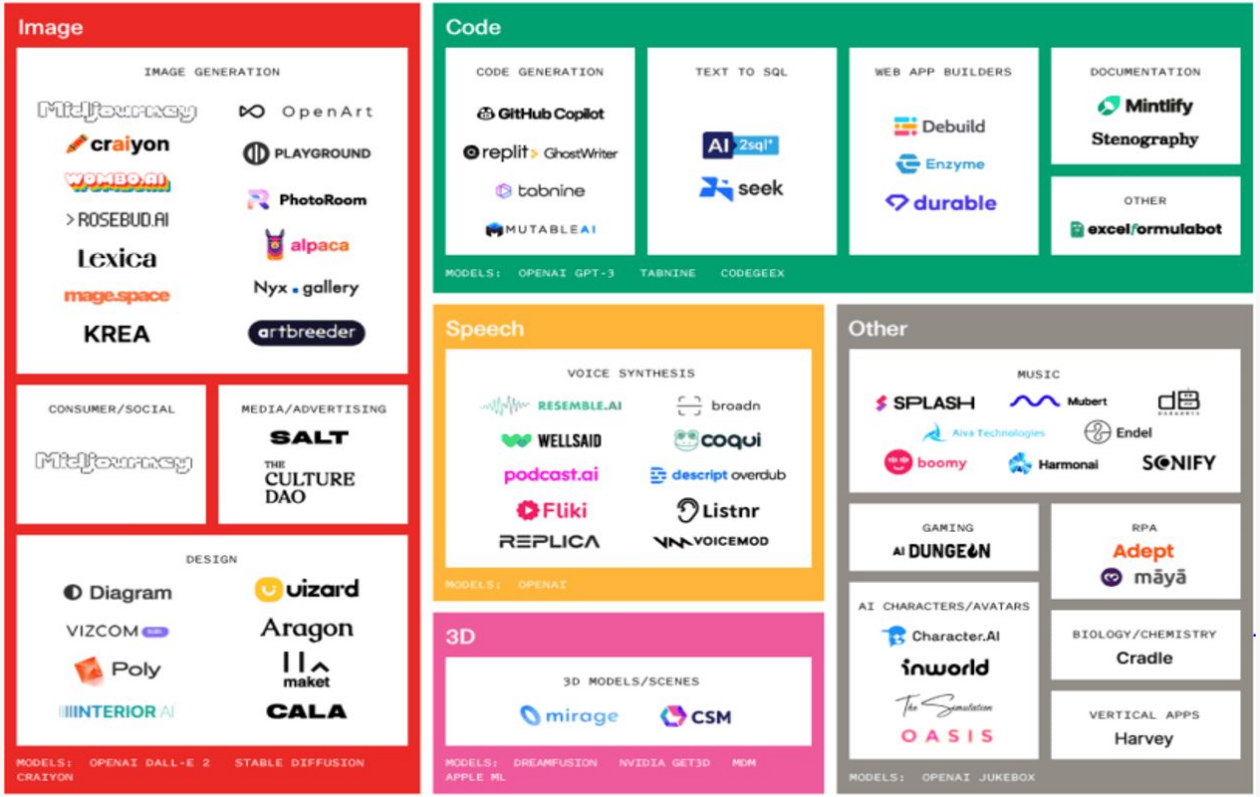
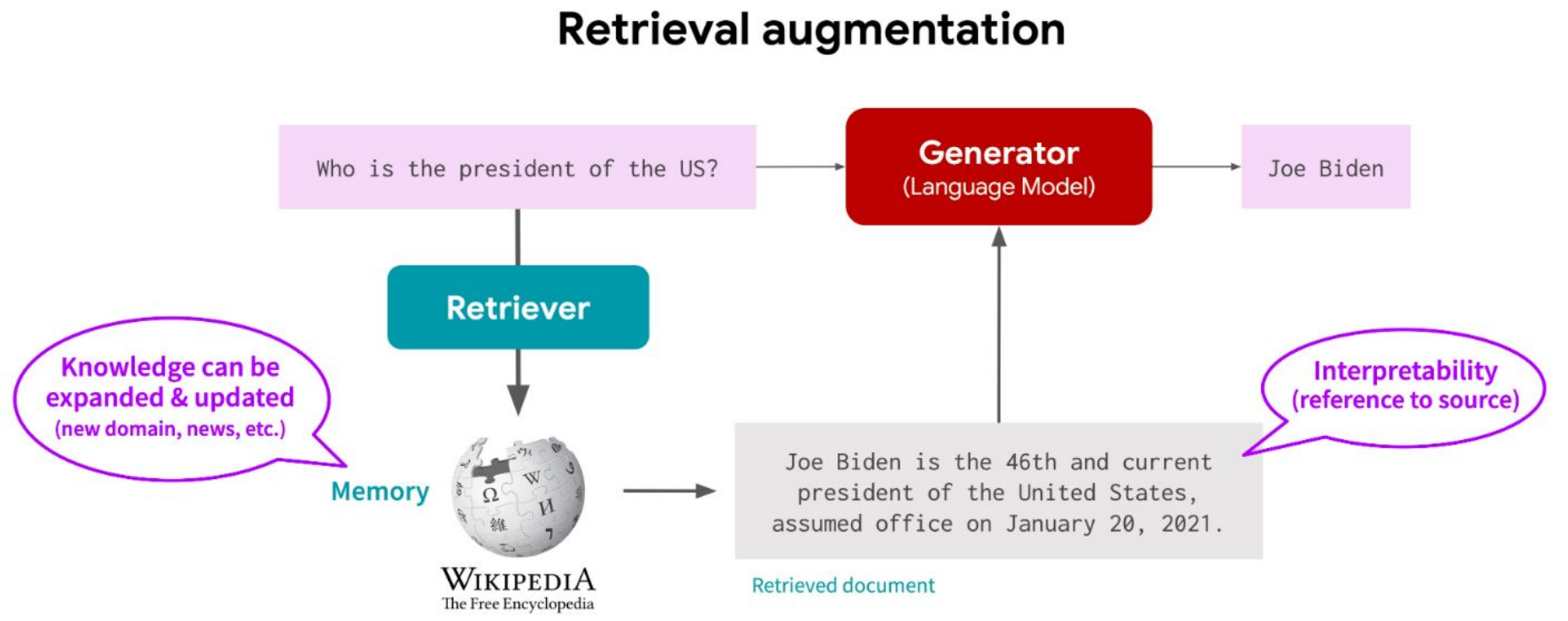
Module 7

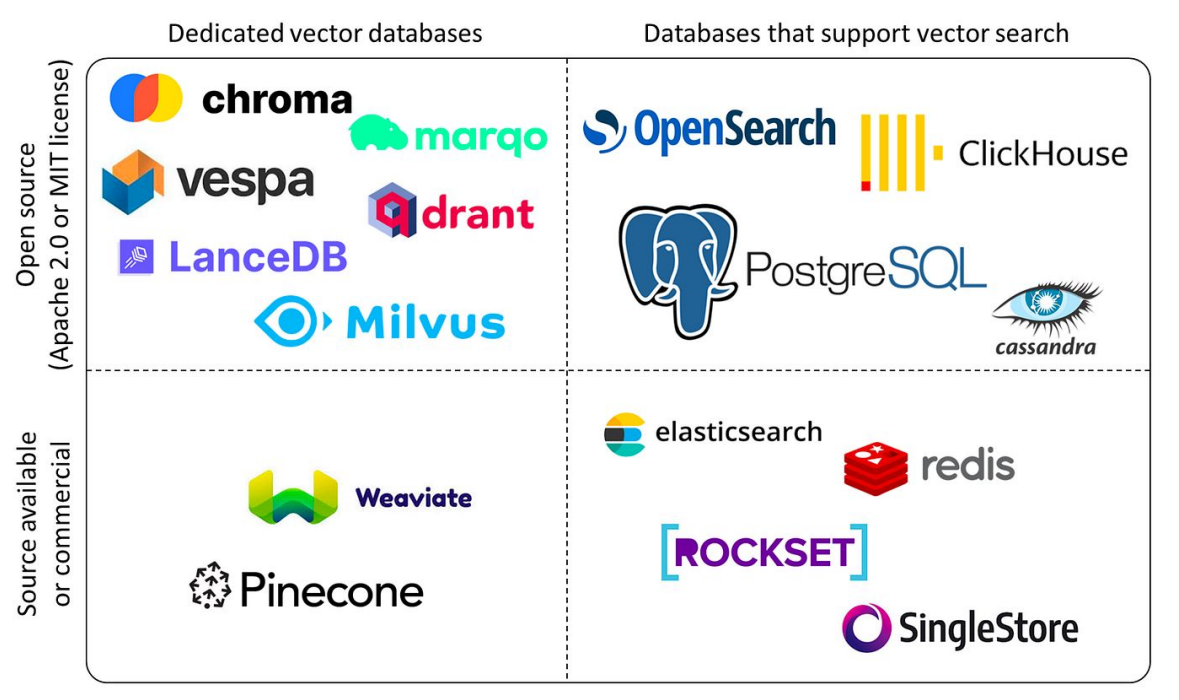
Real world Applications of Gen AI



RAG Architecture for Modern AI apps

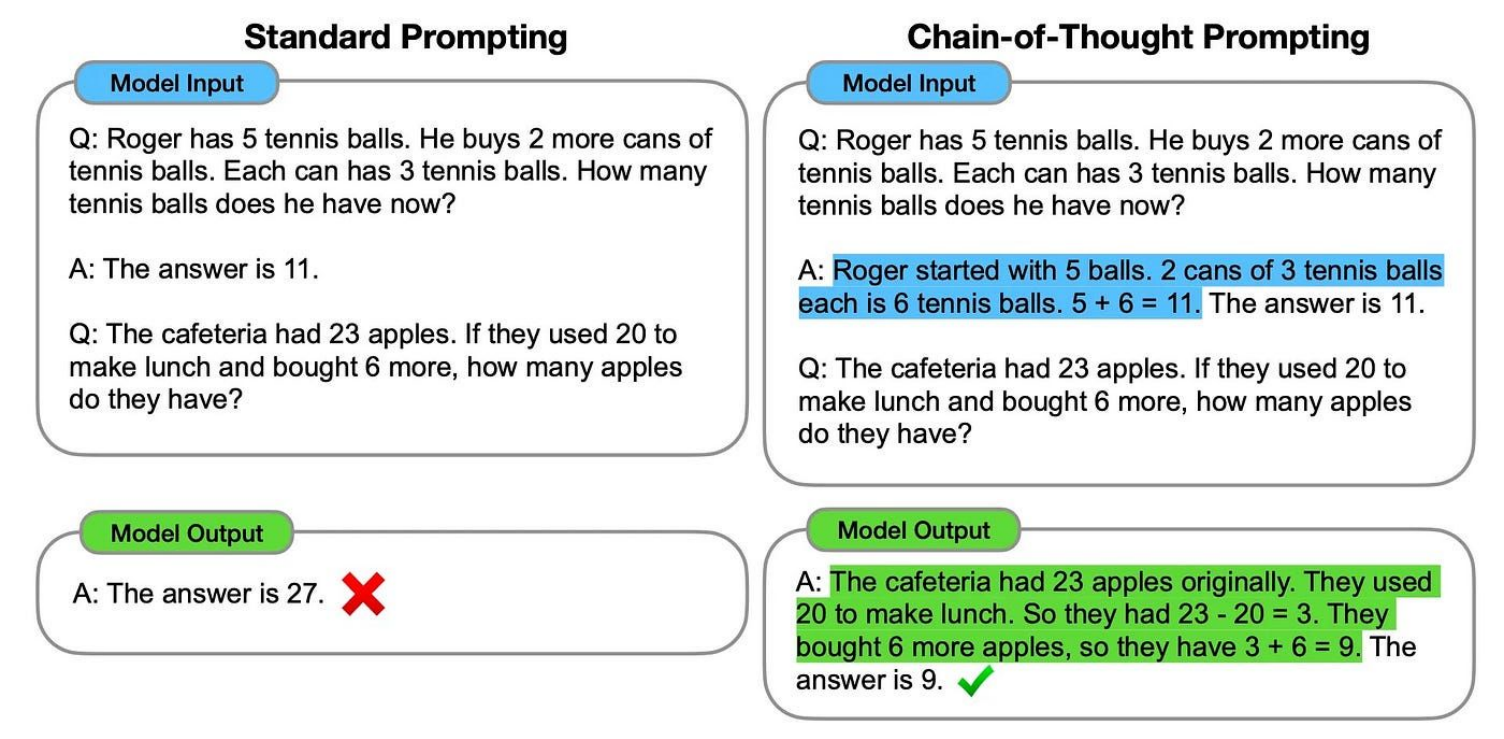


Vector Databases



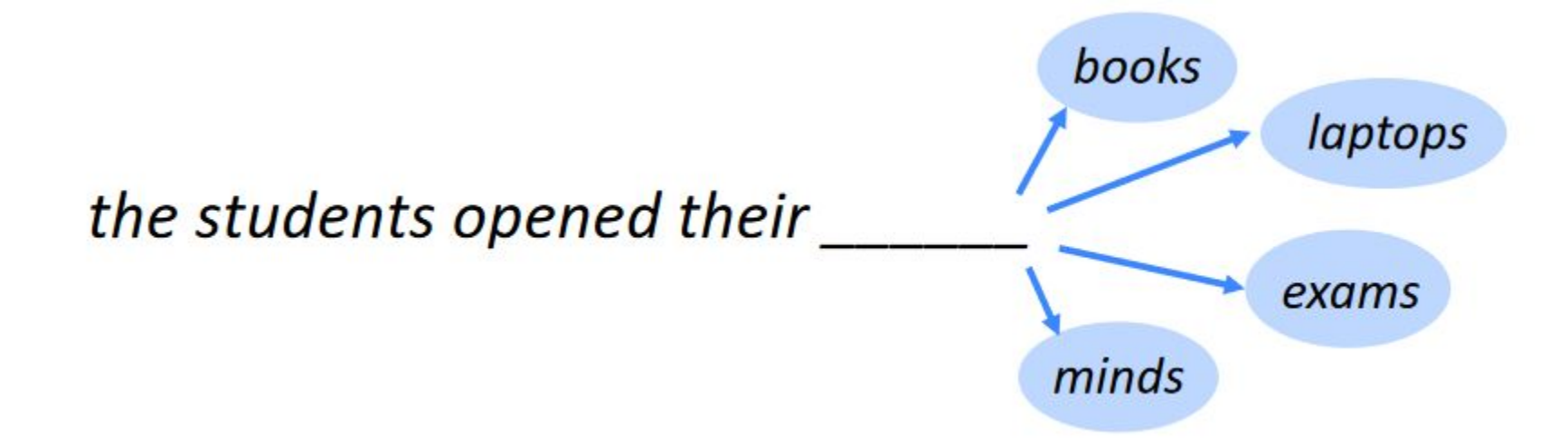
From vectors to numpy arrays to vector stores.

Chain of Thought



Think step by step!

Language models - Theory



Predict the next word - Fill in the blank

Language models - Theory

Journal of Machine Learning Research 3 (2003) 1137–1155 Submitted 4/02; Published 2/03

A Neural Probabilistic Language Model

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Réjean Ducharme DUCHARME@IRO.UMONTREAL.CA
Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Christian Jauvin JAUVINC@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal, Montréal, Québec, Canada

Editors: Jaz Kandola, Thomas Hofmann, Tomaso Poggio and John Shawe-Taylor

Abstract

A goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the **curse of dimensionality**: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. Traditional but very successful approaches based on n-grams obtain generalization

Introducing Perplexity (score)

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

perplexed yet??

Perplexity

"I love to eat ice cream"

Now, the language model will try to predict the next word, which should be "because" to complete the sentence correctly:

"I love to eat ice cream **because**"

- P(**because**) = 0.6
- P(swim) = 0.2
- P(rain) = 0.1
- P(sleep) = 0.05
- P(apple) = 0.05

Perplexity

To calculate perplexity, we take the inverse probability of the entire test sentence and normalize it by the number of words in the sentence.

Perplexity = (1 / (P(**because**) * P(swim) * P(rain) * P(sleep) * P(apple))) ^ (1/5)

In this example, the perplexity value would be relatively low since the model assigned a high probability to the correct next word "because" and low probabilities to the other options. Lower perplexity indicates that the language model is more confident and less surprised when predicting the next word.

Perplexity

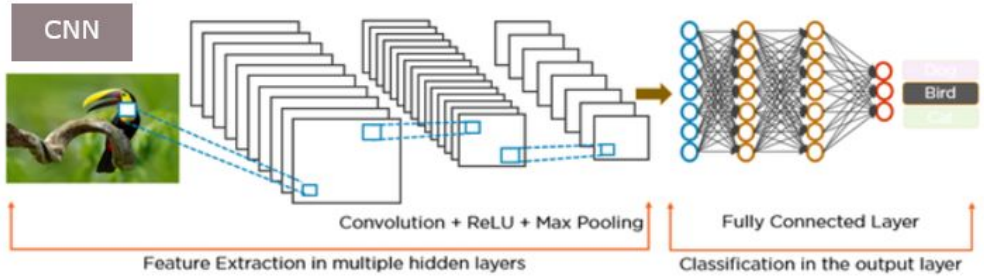
"On the other hand, if the language model was not well-trained or didn't perform well, it might assign similar probabilities to all the possible next words, like this:

- P(**because**) = 0.2
- P(swim) = 0.2
- P(rain) = 0.2
- P(sleep) = 0.2
- P(apple) = 0.2

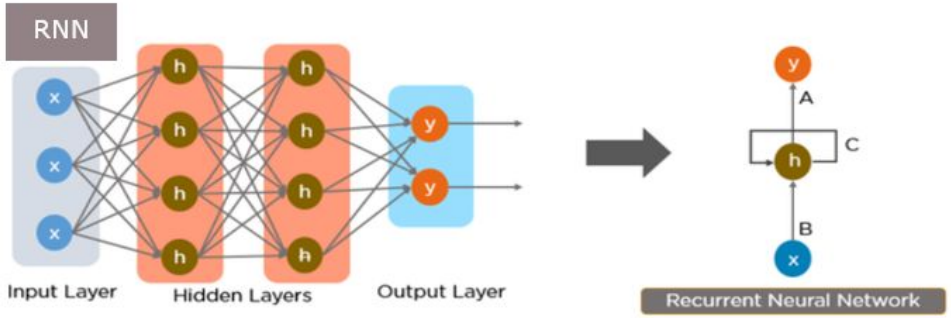
In this case, the perplexity would be higher because the model is uncertain about the next word and doesn't give any word a significantly higher probability.

Language Modeling

Convolutional Neural Network

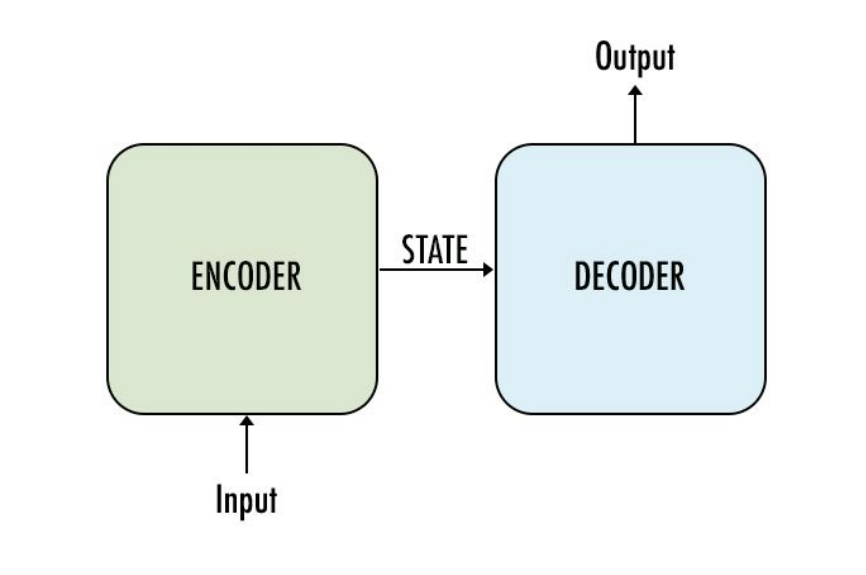


Recurrent Neural Network

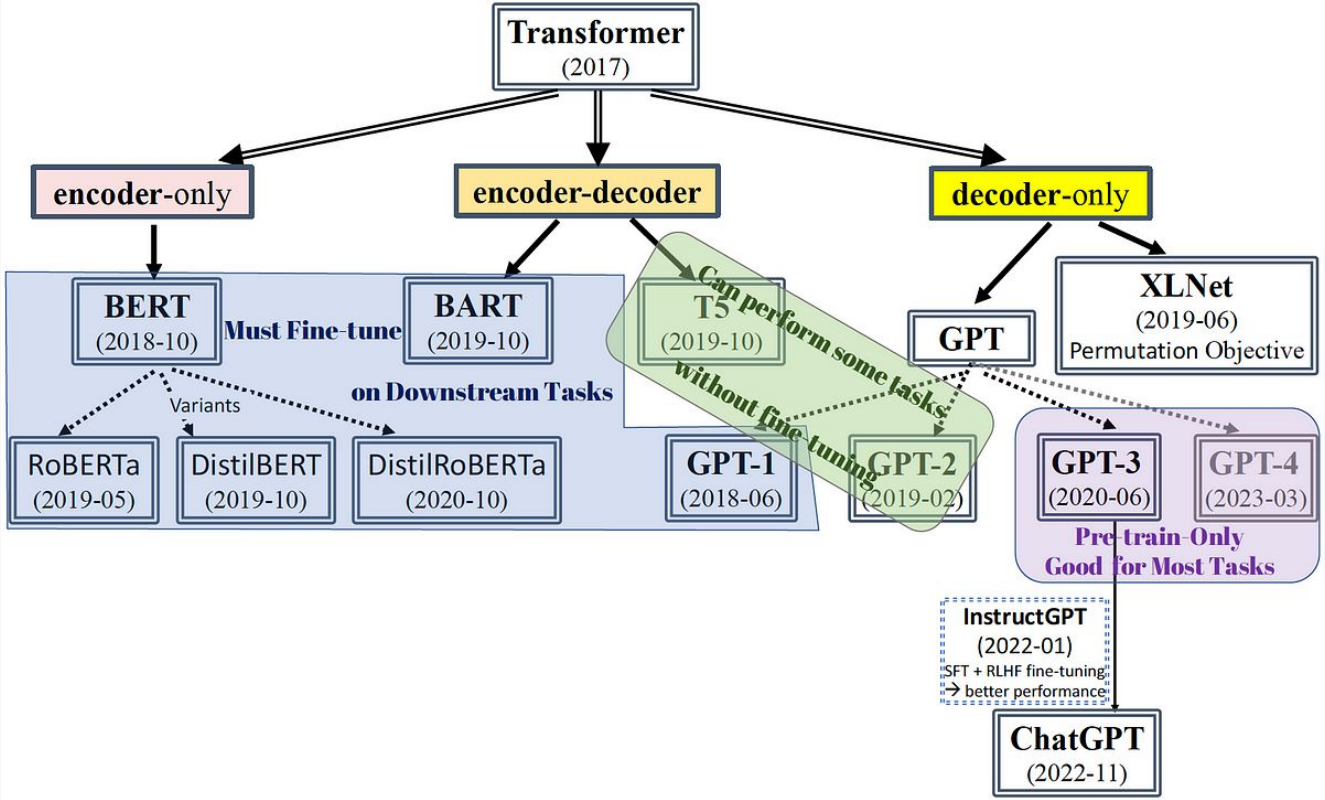


Transformer Model

- Autoregressive Models
- Parallel processing unlike RNNs
- Embedding
- Predicting the next word

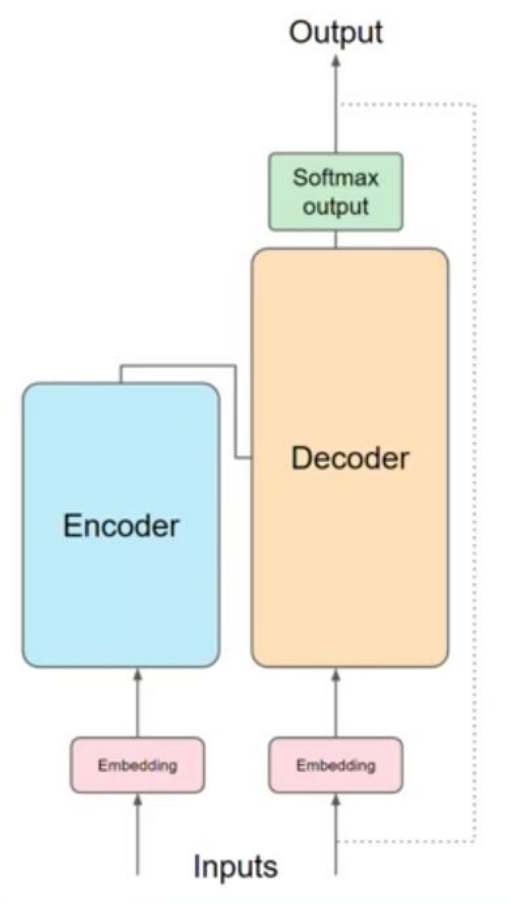


Language Models 101



Transformer Model

- Attention
- Multi-head attention
- Latent space vector
- Vocab size
- Softmax distributions
- Impact of top-p and temperature



Reference Tensorflow Colab NB

Introduction to the Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

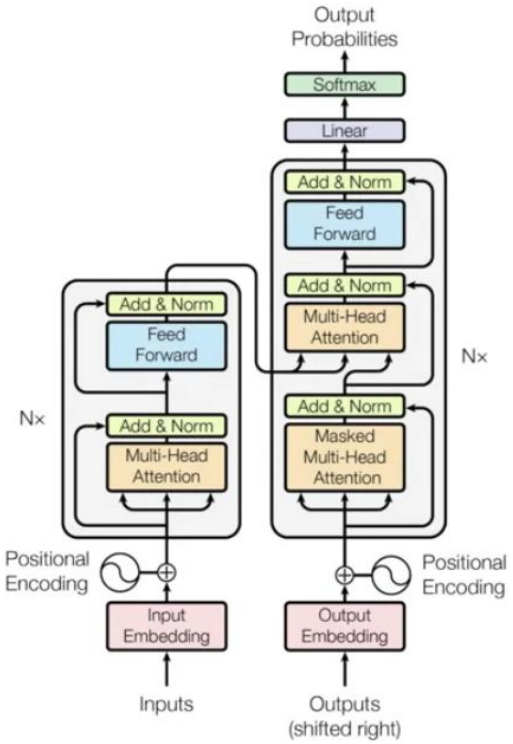
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

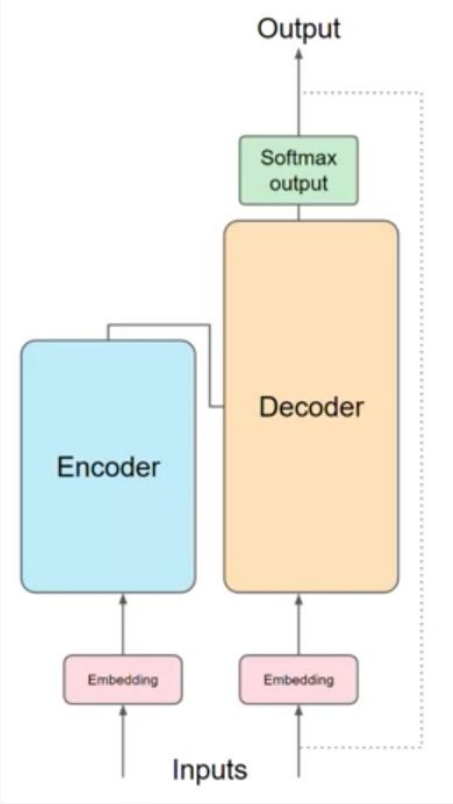
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

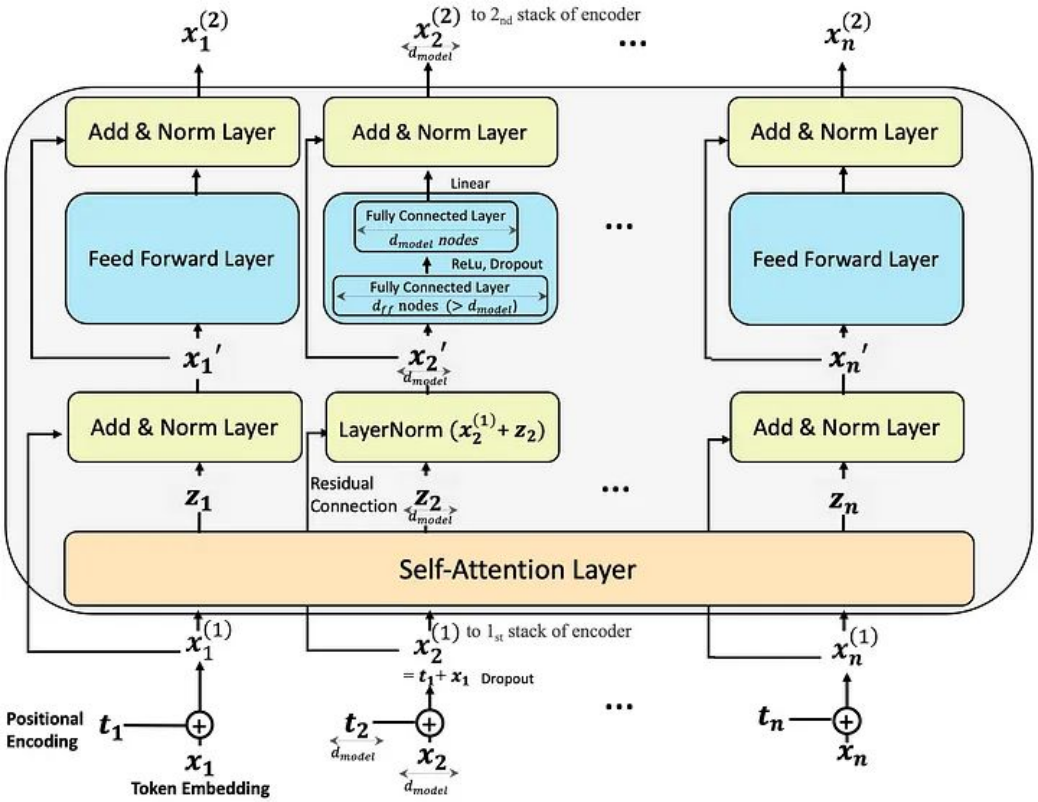
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to

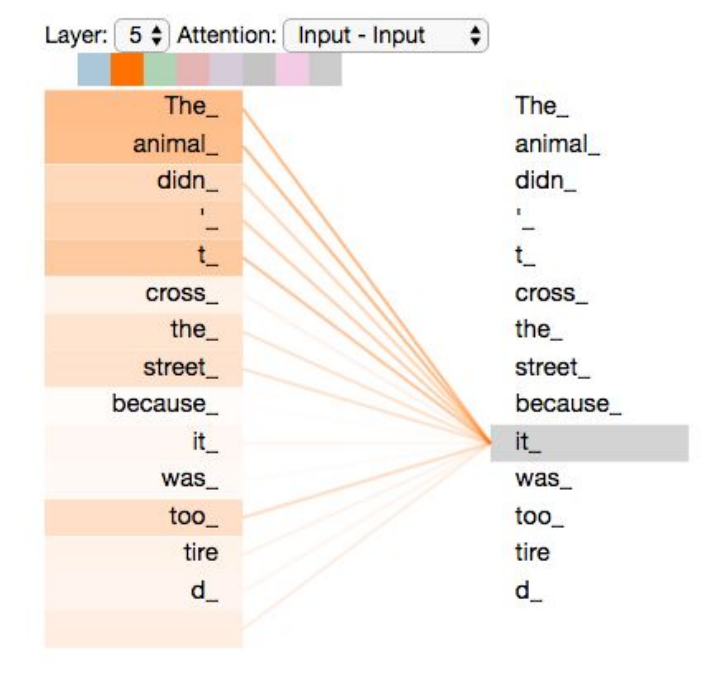


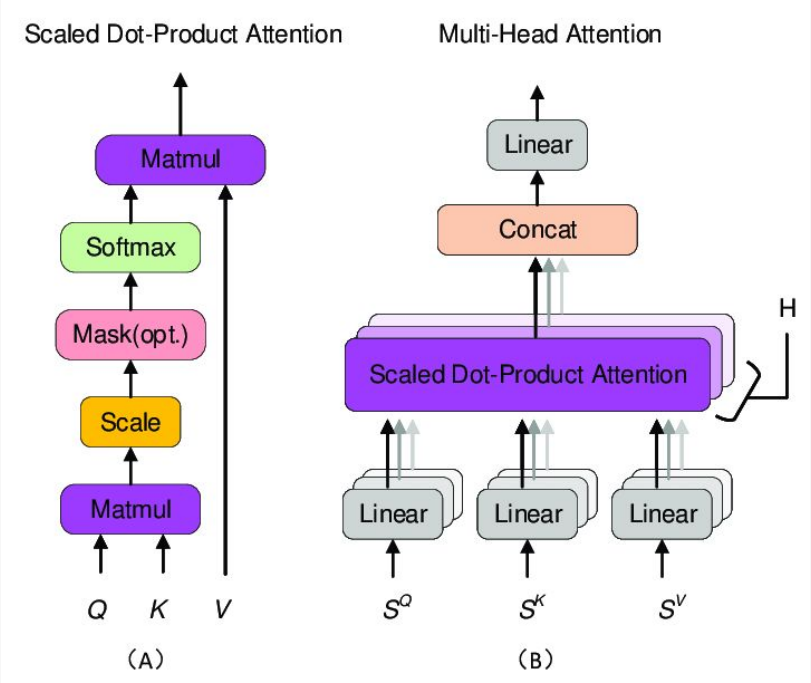
Transformer Model





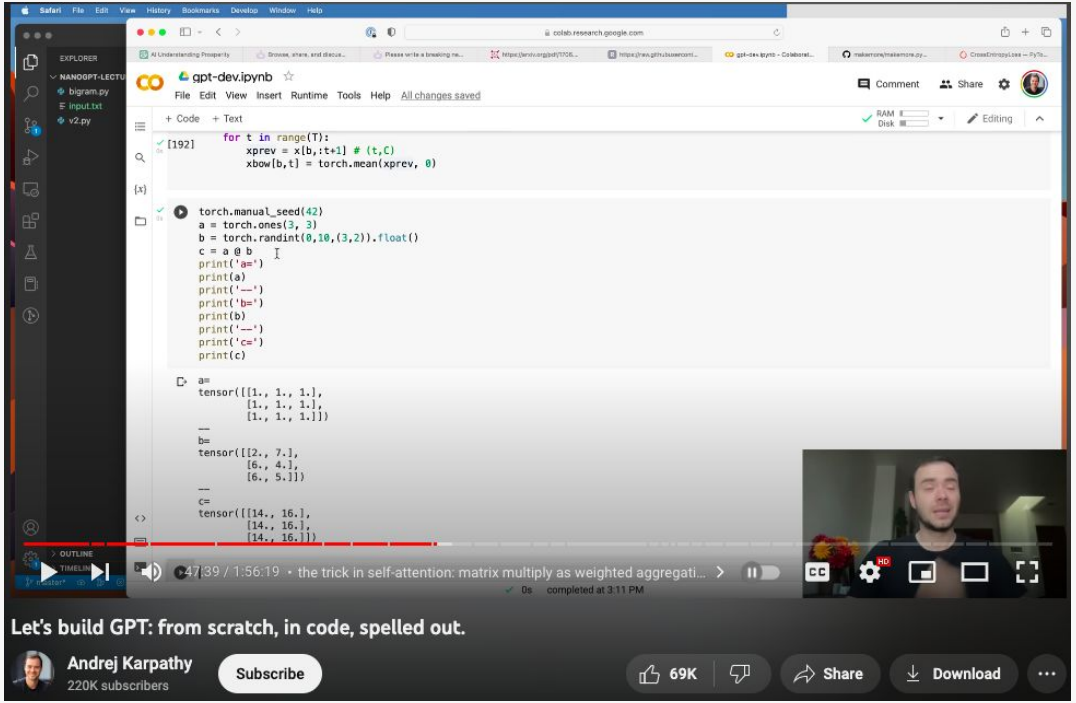
Attention Maps





Source: Jay Alammar's fantastic blogpost

GPT from scratch - reference



Andrej Karpathy Live build video

Stages of LLM Training

1. PRE TRAINING
2. SUPERVISED FINE TUNING (SFT)
3. REINFORCEMENT LEARNING WITH HUMAN FEEDBACK (RLHF)

GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
		init from	init from	init from SFT use RM
Model	**Base model**	**SFT model**	**RM model**	**RL model**
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM can deploy this model	1-100 GPUs days of training ex: Vicuna-13B can deploy this model	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude can deploy this model

Source: Andrej Karpathy's talk at Microsoft Build 2023

Pre-Training Stage - Training data

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

Tokenization

Tokenization

Transform all text into one very long list of integers.

Typical numbers:

~10-100K possible tokens

1 token ~= 0.75 of word

Typical algorithm:

Byte Pair Encoding

Raw text

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

Tokens

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

Integers

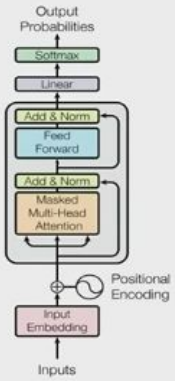
[464, 402, 11571, 1641, 286, 4981, 1429, 2420, 1262, 16326, 11, 543, 389, 2219, 16311, 286, 3435, 1043, 287, 2420, 13, 383, 4981, 1833, 262, 13905, 6958, 1022, 777, 16326, 11, 290, 27336, 379, 9194, 262, 1306, 11241, 287, 257, 8379, 286, 16326, 13, 198, 198, 1639, 460, 779, 262, 2891, 2174, 284, 1833, 703, 257, 3704, 286, 2420, 561, 307, 11241, 1143, 416, 262, 7824, 11, 290, 262, 2472, 954, 286, 16326, 287, 326, 3704, 286, 2420, 13]

Pretraining

Each cell only "sees" cells in its row, and only cells before it (on the left of it), to predict the next cell (on the right of it)

Green = a random highlighted token
Yellow = its context
Red = its target

50,257 numbers
(probabilities for the next token)
correct index (label): 513



Transformer
(neural network)

One training batch, array of shape (B,T)

T = 10

B = 4

4342	318	281	1672	3188	352	4478	617	16326	13
16281	3188	362	50256	16281	3188	513	50256	16281	3188
1212	318	617	4738	2420	655	329	1672	50256	1212
16	11	17	11	18	11	19	11	20	11

Training process

Training data (Shakespeare)

```
First Citizen:
We cannot, sir, we are undone already.

MENENIUS:
I tell you, friends, most charitable care
Have the patricians of you. For your wants,
Your suffering in this dearth, you may as well
Strike at the heaven with your staves as lift them
Against the Roman state, whose course will on
The way it takes, cracking ten thousand curbs
Of more strong link asunder than can ever
Appear in your impediment. For the dearth,
The gods, not the patricians, make it, and
Your knees to them, not arms, must help. Alack,
You are transported by calamity
Thither where more attends you, and you slander
The helms o' the state, who care for you like fathers,
When you curse them as enemies.
```

Samples at initialization

```
z'v}yy_RMV(7ea
AOCEi2tfEi lermh`
`88]gLNSSx|6Mj"i1wdcf,WezVII<4x?OBhS7D-}.8wCkGFgB(kC-
h'Ywa.QhjPo,3C.dA!3;_]!AKa.e0MI Iz(DqAfE8.}nm32<Z2ma1,6DAp
xOrA"jA[V;yhD]<g?BjKXbuptt|W:RT8,ti"(h8J"b"](ZPv3uExA.2r<&;wl?
`mnGs]MG8saNr3"u7tAftthhQBt`GEu66DxN'[[`LU!fUXhy!LI2DjK a
b("8GL``Z66Dhv0,ooqv.
5nmUeh _'j}jjjW33ECIY(5I
0vwdE;_Ze`veBbUv<y'TTBk(m]67q`1N`pd|EobQQ]RtKDXii0Y,LwOZ8d'y1)u
7d|N''CIE2y4hS"MI0od3vtDVV<P``J1ONNn]Y4S<`Q}I2e9d2r8_
ccw[h'9TKFz]8IIDBIh'0y91i?<SKKL'sBv}v
```

[GPT from scratch, NYT, 2023]

Samples after 250 iterations of training

ONom hende beer'TIAFRO.
Rome thecoramerert BENRABENBUR. Nore se. he llod hears hy pid gof
wiere the the paron deread boan: ins wtherk hof at f o otherira coust Soot,
Hyou seealler sheron mer w f shathe thatchie anden wer by he thew bat

Samples after 500 iterations of training

For but te aser if the coouldlavilcoon Creater?
RANTEBR. In fease. Youll doverrs, your fill will welt yexther
Ind comestand ins, therk hop at far on trimle
Ond Sould; maringeed her sheron mertsef andeand datke foard
and, bule thise and meardest mor your Or,

Samples after 5,000 iterations of training

Hor. I have been me, thereof my life, and he concludes him.
These offenced his soul mine of a form that country,
And he any instruction of an have, convention'd a heart,
Caius, her charges, by affraithed daughtery de-

Samples after 30,000 iterations of training

Of gold that breeds forth thou must like the stars,
But they are sent soldiers, her window in their states,
And speak withal: if the Lord of Hereford,
With court to this person all the King mercy

Base models can be prompted into completing tasks

Make your model look like a document!

Context (passage and previous question/answer pairs)

Tom goes everywhere with Catherine Green, a 54-year-old secretary. He moves around her office at work and goes shopping with her. "Most people don't seem to mind Tom," says Catherine, who thinks he is wonderful. "He's my fourth child," she says. She may think of him and treat him that way as her son. He moves around buying his food, paying his health bills and his taxes, but in fact Tom is a dog.

Catherine and Tom live in Sweden, a country where everyone is expected to lead an orderly life according to rules laid down by the government, which also provides a high level of care for its people. This level of care costs money.

People in Sweden pay taxes on everything, so aren't surprised to find that owning a dog means more taxes. Some people are paying as much as 500 Swedish kronor in taxes a year for the right to keep their dog, which is spent by the government on dog hospitals and sometimes medical treatment for a dog that falls ill. However, most such treatment is expensive, so owners often decide to offer health and even life _ for their dog.

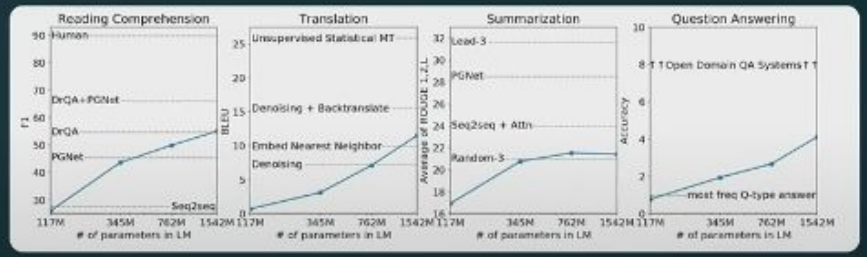
In Sweden dog owners must pay for any damage their dog does. A Swedish Kennel Club official explains what this means: if your dog runs out on the road and gets hit by a passing car, you, as the owner, have to pay for any damage done to the car, even if your dog has been killed in the accident.

Q: How old is Catherine?
A: 54

Q: where does she live?
A:

GPT-2 is "tricked" into performing a task by completing the document

GPT-2 kicked off the era of prompting over finetuning



Language Models are Unsupervised Multitask Learners, Radford et al. 2019 (GPT-2)

GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
		init from	init from	init from SFT use RM
Model	**Base model**	**SFT model**	**RM model**	**RL model**
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM can deploy this model	1-100 GPUs days of training ex: Vicuna-13B can deploy this model	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude can deploy this model

Source: Andrej Karpathy's talk at Microsoft Build 2023

Instruction Tuning

Prompt *Explain the moon landing to a 6 year old in a few sentences.*

Completion GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and
sent them back to the earth so we could all see them.
```

Source: OpenAI blogpost

SFT Dataset

Prompt:

"Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopsonies in the labour market and cite relevant research."

Response:

""Monopsony" refers to a market structure where there is only one buyer for a particular good or service. In economics, this term is particularly relevant in the labor market, where a monopsony employer has significant power over the wages and working conditions of their employees. The presence of a monopsony can result in lower wages and reduced employment opportunities for workers, as the employer has little incentive to increase wages or provide better working conditions. Recent research has identified potential monopsonies in industries such as retail and fast food, where a few large companies control a significant portion of the market (Bivens & Mishel, 2013). In these industries, workers often face low wages, limited benefits, and reduced bargaining power, leading to a situation where they are dependent on the employer for their livelihood. This dependence can result in further suppression of wages and a decline in working conditions. Overall, the concept of monopsony is essential to understanding the dynamics of labor markets and the impact of market power on workers. Further research is needed to understand the extent and impact of monopsonies on the economy and to develop policies to address this issue. References: Bivens, J., & Mishel, L. (2013). The Pay of Corporate Executives and Financial Professionals as Evidence of Rents in Top 1 Percent Incomes. Journal of Economic Perspectives, 27(3), 57-78."

[example from OpenAssistant Conversations Dataset (OASST1)]

Labeling instructions

Excerpt of labeling instructions on the API prompt distribution

You are given a text-based description of a task, submitted by a user. This task description may be in the form of an explicit instruction (e.g. "Write a story about a wise frog."). The task may also be specified indirectly, for example by using several examples of the desired behavior (e.g. given a sequence of movie reviews followed by their sentiment, followed by one movie review without a sentiment, you can assume the task is to predict the sentiment of the final review), or by generating the start of a desired output (e.g. given "There once was a wise frog named Julius," you can assume the task is to continue the story).

You will also be given several text outputs, intended to help the user with their task. Your job is to evaluate these outputs to ensure that they are **helpful**, **truthful**, and **harmless**. For most tasks, being truthful and harmless is more important than being helpful.

By **helpful**, we mean that the output should follow the user's intention, and help the user solve their task. Some examples of helpful behavior:

- Writing in clear language.
- Answering the question they meant to ask, even if they mis-asked it.
- Being sensitive to internationality (e.g. "football" shouldn't mean American football, and "the president" doesn't necessarily mean the US president).
- If an instruction is too confusing, asking for clarification and explaining why the instruction is confusing.
- Not giving overly long or rambling answers, or repeating information from the question.
- Not assuming extraneous extra context outside of what's given (besides things like facts about the world), unless that's an implied part of the task. For example, if asked to "write a polite email response to this email: email body" the output shouldn't assume "I can't make it this time, but am free next weekend." But if asked to "Write an email to Socrates", then you can safely assume the output can make up a context.

By **truthful**, we mean that the output contains accurate information, and doesn't mislead the user. Some examples of truthful behavior:

- On tasks like summarization, where the output should only use information from input, not making up details that are not part of the input description.
- Not producing clearly false information about the world (e.g. making up facts or promoting conspiracies). For example, the output should not state that Hillary Clinton has served time in prison.
- Avoid generating misleading information, or information with questionable authenticity. For example, if the input asks "Why did Hillary Clinton go to jail?", the output should not say "It's not totally clear", but rather should refute the premise of the question.

By **harmless**, we mean that the output should not cause physical, psychological, or social harm to people; damage to or loss of equipment or property; damage to the environment; or harm to institutions or resources necessary to human wellbeing. Some examples of harmless behavior:

- Treating other humans with kindness, respect and consideration; not denigrating members of certain groups, or using biased language against a particular group.
- Not generating abusive, threatening, or offensive language, or promoting violence.
- Not writing sexual or violent content if it's not asked for.
- Not giving bad real-world advice, or promoting illegal activity.

Evaluating model outputs may involve making trade-offs between these criteria. These trade-offs will depend on the task. Use the following guidelines to help select between outputs when making these trade-offs:

For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful / harmless; and (c) the task does not seem to be in a "high stakes domain" (e.g. loan applications, therapy, medical or legal advice, etc.); then rate the more helpful output higher. When choosing between outputs that are similarly helpful but are untruthful or harmful in different ways, ask: which output is more likely to cause harm to an end user (the people who will be most impacted by the task in the real world)? This output should be ranked lower. If this isn't clear from the task, then mark these outputs as tied.

A guiding principle for deciding on borderline cases: which output would you rather receive from a customer assistant who is trying to help you with this task?

Ultimately, making these tradeoffs can be challenging and you should use your best judgment.

[InstructGPT]

GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
		init from	init from	init from SFT use RM
Model	**Base model**	**SFT model**	**RM model**	**RL model**
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM can deploy this model	1-100 GPUs days of training ex: Vicuna-13B can deploy this model	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude can deploy this model

Source: Andrej Karpathy's talk at Microsoft Build 2023

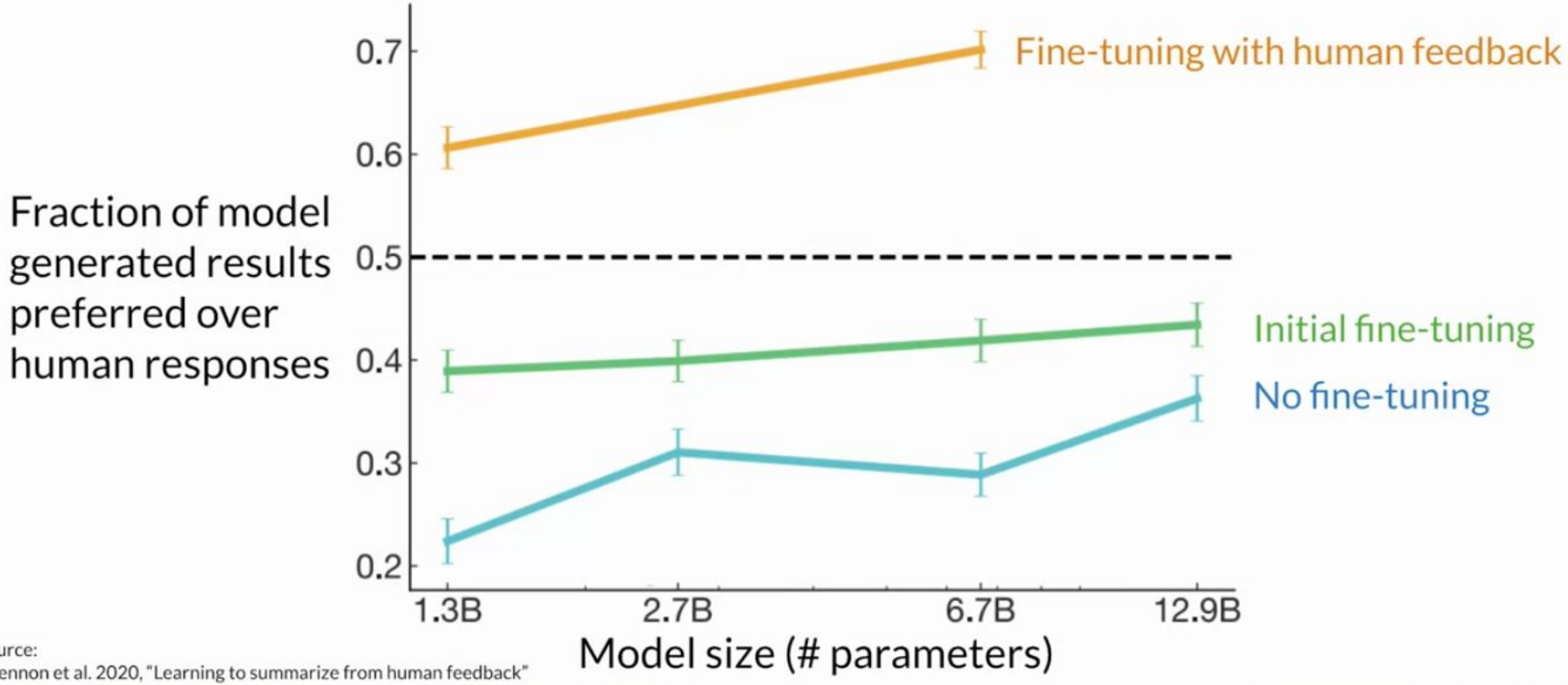
Model Alignment

Are the models:

- Helpful?
- Honest?
- Harmless?

RLHF to the rescue!

Fine-tuning with human feedback



Source:
Stiennon et al. 2020, "Learning to summarize from human feedback"

RM Dataset

Write a Python function that checks if a given string is a palindrome.

Certainly! Here's a Python function that checks if a given string is a palindrome:

```python
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```

Here's how you can use the function:

```python
word1 = "radar"
print(is_palindrome(word1))  # Output: True

word2 = "Hello World"
print(is_palindrome(word2))  # Output: False
```

The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing (`[::-1]`). If the word is equal to its reverse, it is a palindrome, and the function returns `True`. Otherwise, it returns `False`.

Write a Python function that checks if a given string is a palindrome.

Certainly! Here's a Python function that checks if a given string is a palindrome:

```python
def is_palindrome(string):
    # Convert the string to lowercase and remove spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns `True`, indicating that the string is a palindrome. Otherwise, it returns `False`.

Here's an example of how you can use the function:

```python
word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Output:

```csharp
radar is a palindrome.
```

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.

Write a Python function that checks if a given string is a palindrome.

Certainly! Here's a Python function that checks if a given string is a palindrome:

```python
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join(char.lower() for char in s if char.isalnum())

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```

In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing (`[::-1]` creates a reversed copy of the string). If they are equal, the string is a palindrome, and the function returns `True`; otherwise, it returns `False`.

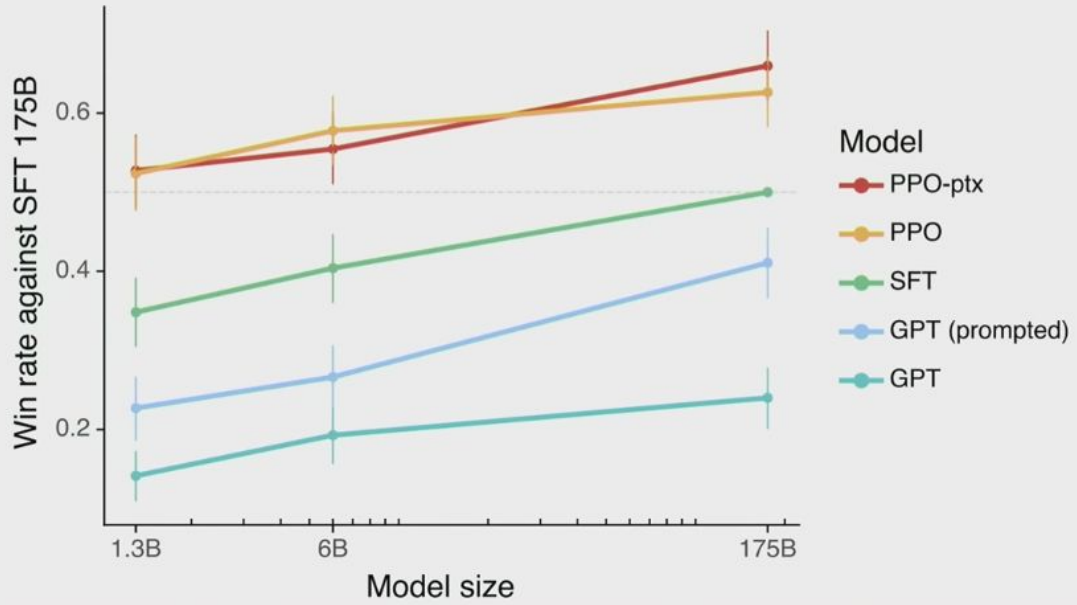
Here's an example usage of the function:

```python
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1))  # True

string2 = "race car"
print(is_palindrome(string2))  # True

string3 = "hello world"
print(is_palindrome(string3))  # False
```

In the above example, `is_palindrome` is called with three different strings. The function correctly identifies the palindromes ("A man, a plan, a canal: Panama" and "race car") and returns `True`. For the non-palindrome "hello world," it returns `False`.

Why RLHF?

It works better.



[Training language models to follow instructions with human feedback, OpenAI, 2022]

GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
		init from	init from	init from SFT use RM
Model	**Base model**	**SFT model**	**RM model**	**RL model**
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM can deploy this model	1-100 GPUs days of training ex: Vicuna-13B can deploy this model	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude can deploy this model

Source: Andrej Karpathy's talk at Microsoft Build 2023

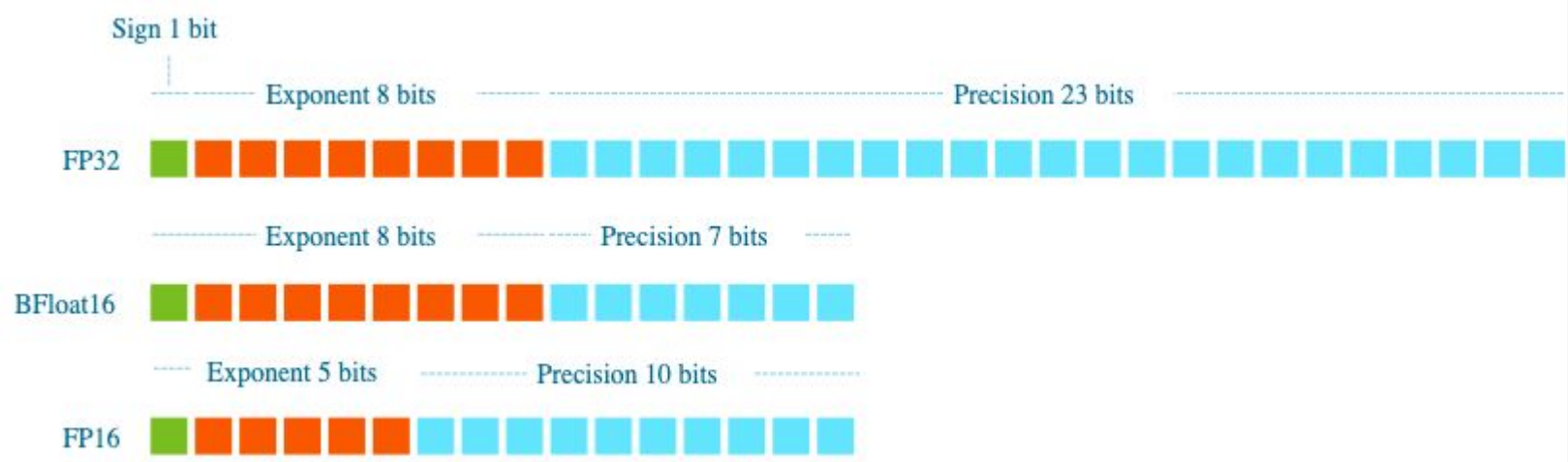
Advanced Langchain

- Memory
- Tools usage
- Routing
- Chains
- Agents

LLM Theory

- Transformer Model
- LLM Performance metrics - ROUGE and BLEU
- Scaling Laws
- Model Quantization - 32 bit full precision, FP16, BFLOAT16, INT8 and INT4
- Fine Tuning - Full vs Adapter
- Parameter Efficient fine tuning

Loading LLMs into the GPU



LLM Pre-training scaling laws

arXiv > cs > arXiv:2001.08361

Search...

Help | Advanc

Computer Science > Machine Learning

[Submitted on 23 Jan 2020]

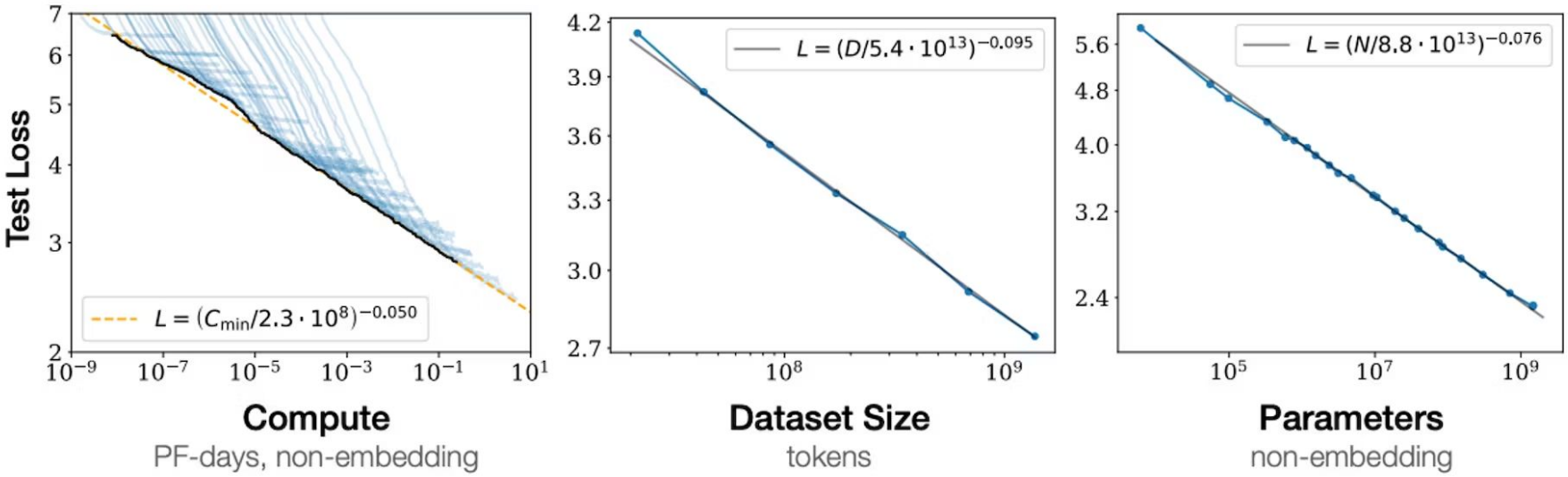
Scaling Laws for Neural Language Models

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, Dario Amodei

We study empirical scaling laws for language model performance on the cross-entropy loss. The loss scales as a power-law with model size, dataset size, and the amount of compute used for training, with some trends spanning more than seven orders of magnitude. Other architectural details such as network width or depth have minimal effects within a wide range. Simple equations govern the dependence of overfitting on model/dataset size and the dependence of training speed on model size. These relationships allow us to determine the optimal allocation of a fixed compute budget. Larger models are significantly more sample-efficient, such that optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.

Scaling_Laws_Paper_Link

LLM Pre-training scaling laws



Scaling Laws Paper Link

How to pick parameters

arXiv > cs > arXiv:2203.15556

Search...

Help | Advanced

Computer Science > Computation and Language

[Submitted on 29 Mar 2022]

Training Compute-Optimal Large Language Models

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, Laurent Sifre

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly undertrained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, Chinchilla, that uses the same compute budget as Gopher but with 70B parameters and 4× more more data. Chinchilla uniformly and significantly outperforms Gopher (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that Chinchilla uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, Chinchilla reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over Gopher.

Subjects: **Computation and Language (cs.CL)**; Machine Learning (cs.LG)
Cite as: arXiv:2203.15556 **[cs.CL]**
(or arXiv:2203.15556v1 **[cs.CL]** for this version)
https://doi.org/10.48550/arXiv.2203.15556

Submission history

From: Jordan Hoffmann [view email]
[v1] Tue, 29 Mar 2022 13:38:03 UTC (10,143 KB)

Bibliographic Tools | Code, Data, Media | Demos | Related Papers

Compute optimal training datasize is ~**20x** number of parameters

Chinchilla Paper link

LLM Performance Metrics

Rouge - Summarization.

Rouge recall = no. of common ngrams / no. of ngrams in the ground truth
Rouge precision = no. of common ngrams / no. of ngrams in the prediction

BLEU - Translation.

BLEU precision = Avg (multiple ngram precision scores)

Enterprise + ML-OPS

- PEFT & LoRa
- FINETUNING
- DEPLOYMENTS

Model Finetuning

- SFT -> high quality / low quantity (compared to pre-training phase)
- Types of finetuning:
  - Full Finetuning
  - Partial finetuning

- Partial finetuning saves GPU memory! *Parameter Efficient finetuning.*

PEFT

Andrej Karpathy
@karpathy

There's a chance that LoRA finetunes work so well that it dramatically alters the finetuning vs. retrieval + few-shot prompting power dynamic in favor of the former for many applications.

PEFT (Parameter Efficient Finetuning, LoRA included) are emerging techniques that make it very cheap to finetune LLMs because most of the parameters can be kept frozen and in very low precision during training. The cost of pretraining and finetuning decouple.
huggingface.co/blog/peft

+LoRA (the code is very short/readable)
github.com/microsoft/LoRA

🤗 PEFT

State-of-the-art Parameter-Efficient Fine-Tuning (PEFT) methods

Parameter-Efficient Fine-Tuning (PEFT) methods enable efficient adaptation of pre-trained language models (PLMs) to various downstream applications without fine-tuning all the model's parameters. Fine-tuning large-scale PLMs is often prohibitively costly. In this regard, PEFT methods only fine-tune a small number of (extra) model parameters, thereby greatly decreasing the computational and storage costs. Recent State-of-the-Art PEFT techniques achieve performance comparable to that of full fine-tuning.

Seamlessly integrated with 🤗 Accelerate for large scale models leveraging DeepSpeed and Big Model Inference.

Supported methods:

1. LoRA: LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS
2. Prefix Tuning: Prefix-Tuning: Optimizing Continuous Prompts for Generation, P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks
3. P-Tuning: GPT Understands, Too
4. Prompt Tuning: The Power of Scale for Parameter-Efficient Prompt Tuning
5. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning
6. $(IA)^3$: Infused Adapter by Inhibiting and Amplifying Inner Activations

Getting started

```
from transformers import AutoModelForSeq2SeqLM
from peft import get_peft_config, get_peft_model, LoraConfig, TaskType
model_name_or_path = "bigscience/mt0-large"
tokenizer_name_or_path = "bigscience/mt0-large"
```

Packages

No packages published

Used by 924

+ 916

Contributors 79

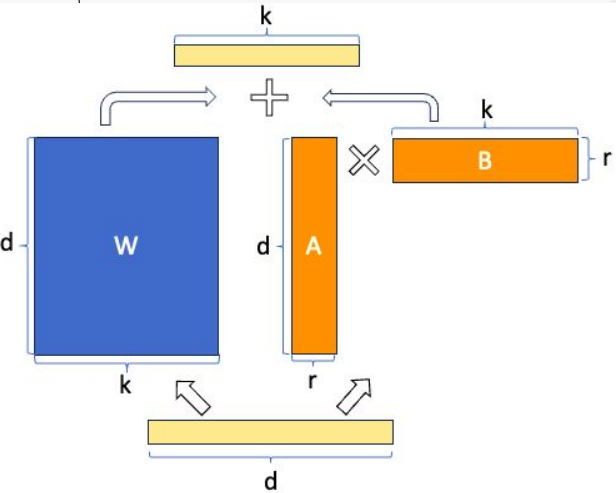
+ 68 contributors

Languages

Python 98.8% Other 1.2%

PEFT Huggingface Blogpost Link

LoRA



arXiv > cs > arXiv:2106.09685

Search...

Help | Ad

Computer Science > Computation and Language

[Submitted on 17 Jun 2021 (v1), last revised 16 Oct 2021 (this version, v2)]

LoRA: Low-Rank Adaptation of Large Language Models

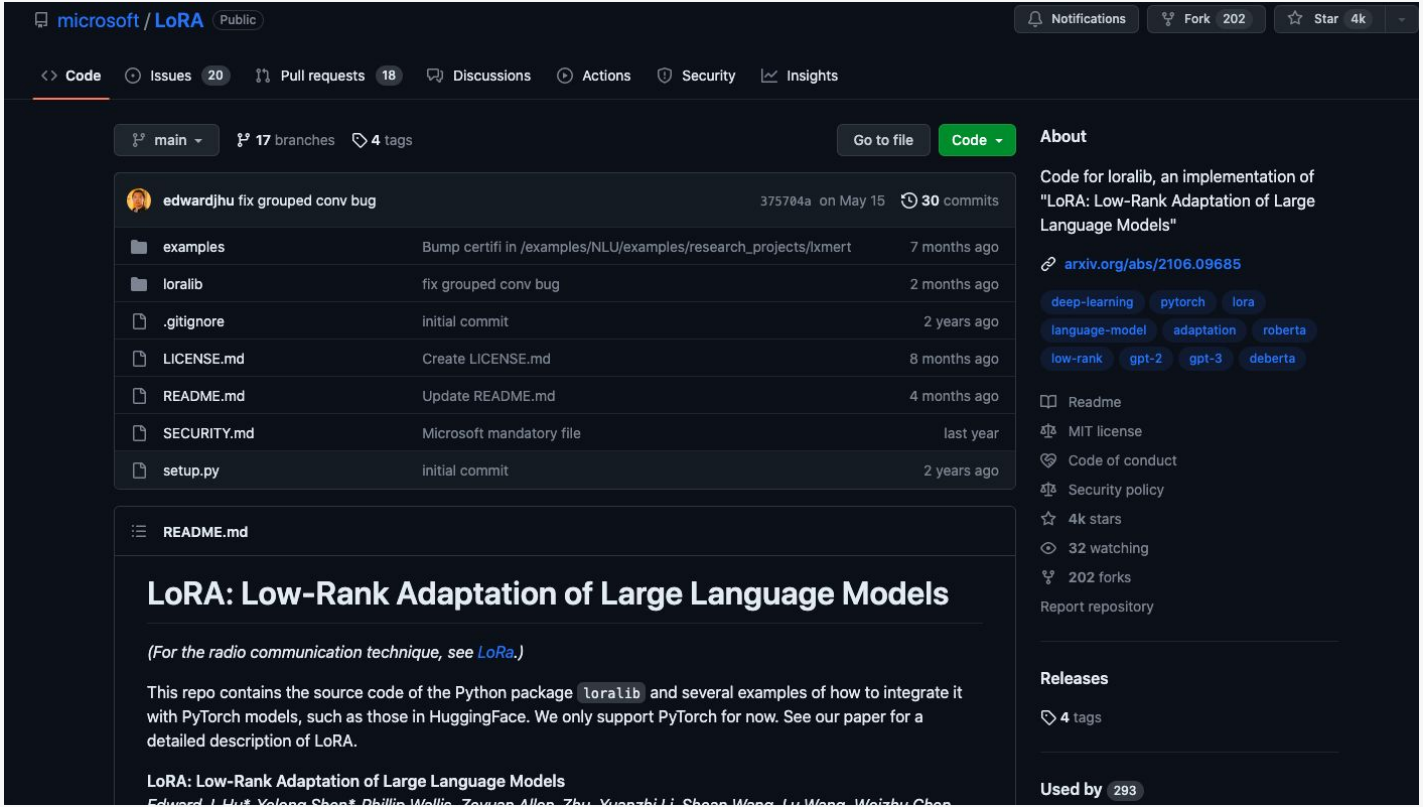
Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example -- deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose Low-Rank Adaptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, no additional inference latency. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at this https URL.

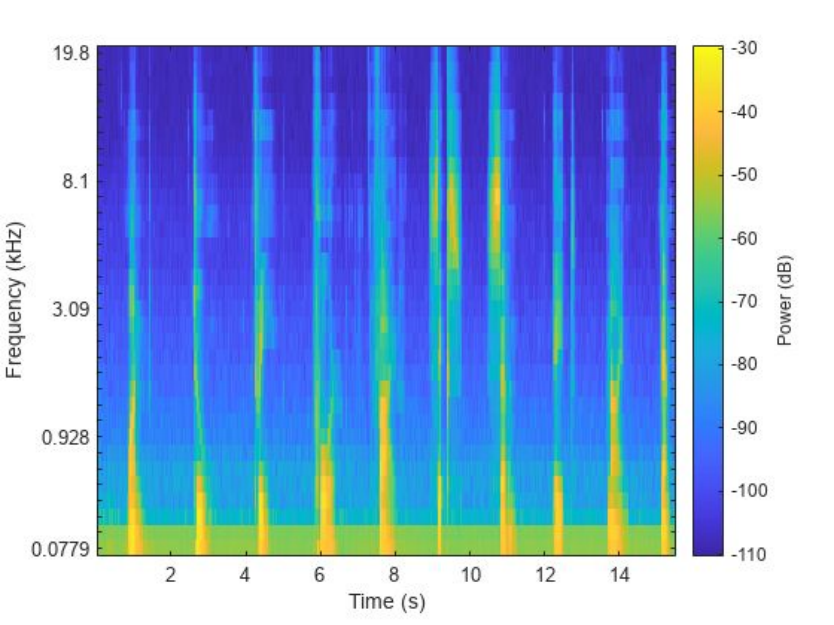
Comments: Draft V2 includes better baselines, experiments on GLUE, and more on adapter latency
Subjects: **Computation and Language (cs.CL)**; Artificial Intelligence (cs.AI); Machine Learning (cs.LG)
Cite as: arXiv:2106.09685 **[cs.CL]**
(or arXiv:2106.09685v2 **[cs.CL]** for this version)

LoRA Paper Link

LoRA


microsoft / LoRA Public
Notifications
Fork 202
Star 4k
Code
Issues 20
Pull requests 18
Discussions
Actions
Security
Insights
main
17 branches
4 tags
Go to file
Code
edwardjhu fix grouped conv bug
375704a on May 15
30 commits
examples
Bump certifi in /examples/NLU/examples/research_projects/lxmert
7 months ago
loralib
fix grouped conv bug
2 months ago
.gitignore
initial commit
2 years ago
LICENSE.md
Create LICENSE.md
8 months ago
README.md
Update README.md
4 months ago
SECURITY.md
Microsoft mandatory file
last year
setup.py
initial commit
2 years ago
README.md
LoRA: Low-Rank Adaptation of Large Language Models
(For the radio communication technique, see LoRa.)
This repo contains the source code of the Python package loralib and several examples of how to integrate it with PyTorch models, such as those in HuggingFace. We only support PyTorch for now. See our paper for a detailed description of LoRA.
LoRA: Low-Rank Adaptation of Large Language Models
About
Code for loralib, an implementation of "LoRA: Low-Rank Adaptation of Large Language Models"
arxiv.org/abs/2106.09685
deep-learning
pytorch
lora
language-model
adaptation
roberta
low-rank
gpt-2
gpt-3
deberta
Readme
MIT license
Code of conduct
Security policy
4k stars
32 watching
202 forks
Report repository
Releases
4 tags
Used by 293

Audio



- MEL Spectrograms
- Transformers for Audio & Vision
- OpenAI Whisper Model

Mel Spectrograms

```
import librosa
import librosa.display
import matplotlib.pyplot as plt

# Load the audio file
audio_file = 'path_to_your_audio_file.wav'
y, sr = librosa.load(audio_file)

# Compute the Mel spectrogram
mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)

# Convert power spectrogram to dB scale
mel_spectrogram_db = librosa.power_to_db(mel_spectrogram, ref=np.max)

# Display the Mel spectrogram
plt.figure(figsize=(10, 4))
librosa.display.specshow(mel_spectrogram_db, y_axis='mel', x_axis='time')
plt.colorbar(format='%+2.0f dB')
plt.title('Mel Spectrogram')
plt.tight_layout()
plt.show()
```

OpenAI Whisper

arXiv > eess > arXiv:2212.04356

Search...

Help | Adva

Electrical Engineering and Systems Science > Audio and Speech Processing

[Submitted on 6 Dec 2022]

Robust Speech Recognition via Large-Scale Weak Supervision

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, Ilya Sutskever

We study the capabilities of speech processing systems trained simply to predict large amounts of transcripts of audio on the internet. When scaled to 680,000 hours of multilingual and multitask supervision, the resulting models generalize well to standard benchmarks and are often competitive with prior fully supervised results but in a zero-shot transfer setting without the need for any fine-tuning. When compared to humans, the models approach their accuracy and robustness. We are releasing models and inference code to serve as a foundation for further work on robust speech processing.

Subjects: **Audio and Speech Processing (eess.AS)**; Computation and Language (cs.CL); Machine Learning (cs.LG); Sound (cs.SD)
Cite as: arXiv:2212.04356 **[eess.AS]**
(or arXiv:2212.04356v1 **[eess.AS]** for this version)
https://doi.org/10.48550/arXiv.2212.04356

Submission history

From: Jong Wook Kim [view email]
[v1] Tue, 6 Dec 2022 18:46:04 UTC (1,312 KB)

Paper Link

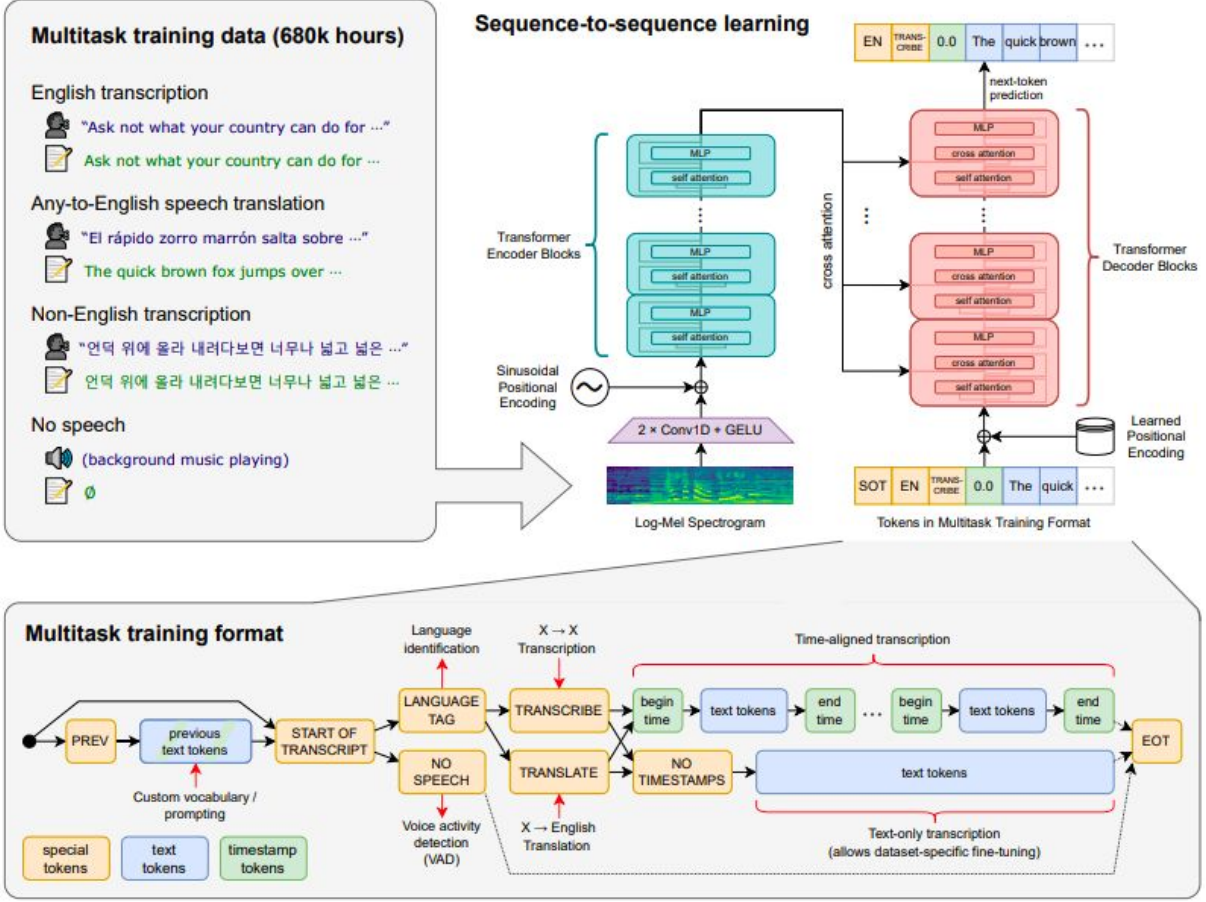


Figure 1. **Overview of our approach.** A sequence-to-sequence Transformer model is trained on many different speech processing tasks, including multilingual speech recognition, speech translation, spoken language identification, and voice activity detection. All of these tasks are jointly represented as a sequence of tokens to be predicted by the decoder, allowing for a single model to replace many different stages of a traditional speech processing pipeline. The multitask training format uses a set of special tokens that serve as task specifiers or classification targets, as further explained in Section 2.3.

LLMs - The cutting edge

- ALiBi - positional encoding
- Flash Attention
- Sparse Attention
- Multi-Query Attention vs Multi-head Attention
- Conditional Computation
- Mixed-precision training
- Model Quantization

Quantization - Less memory footprint

arXiv > cs > arXiv:2209.05433

Search...

Help | Advanced

Computer Science > Machine Learning

[Submitted on 12 Sep 2022 (v1), last revised 29 Sep 2022 (this version, v2)]

FP8 Formats for Deep Learning

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, Hao Wu

FP8 is a natural progression for accelerating deep learning training inference beyond the 16-bit formats common in modern processors. In this paper we propose an 8-bit floating point (FP8) binary interchange format consisting of two encodings - E4M3 (4-bit exponent and 3-bit mantissa) and E5M2 (5-bit exponent and 2-bit mantissa). While E5M2 follows IEEE 754 conventions for representatio of special values, E4M3's dynamic range is extended by not representing infinities and having only one mantissa bit-pattern for NaNs. We demonstrate the efficacy of the FP8 format on a variety of image and language tasks, effectively matching the result quality achieved by 16-bit training sessions. Our study covers the main modern neural network architectures - CNNs, RNNs, and Transformer-based models, leaving all the hyperparameters unchanged from the 16-bit baseline training sessions. Our training experiments include large, up to 175B parameter, language models. We also examine FP8 post-training-quantization of language models trained using 16-bit formats that resisted fixed point int8 quantization.

Subjects: **Machine Learning (cs.LG)**
Cite as: arXiv:2209.05433 **[cs.LG]**
(or arXiv:2209.05433v2 **[cs.LG]** for this version)
https://doi.org/10.48550/arXiv.2209.05433

Paper Link

Flash Attention - Better than O(N^2)

arXiv > cs > arXiv:2205.14135

Search...

Help | Advan

Computer Science > Machine Learning

[Submitted on 27 May 2022 (v1), last revised 23 Jun 2022 (this version, v2)]

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, Christopher Ré

Transformers are slow and memory-hungry on long sequences, since the time and memory complexity of self-attention are quadratic in sequence length. Approximate attention methods have attempted to address this problem by trading off model quality to reduce the compute complexity, but often do not achieve wall-clock speedup. We argue that a missing principle is making attention algorithms IO-aware -- accounting for reads and writes between levels of GPU memory. We propose FlashAttention, an IO-aware exact attention algorithm that uses tiling to reduce the number of memory reads/writes between GPU high bandwidth memory (HBM) and GPU on-chip SRAM. We analyze the IO complexity of FlashAttention, showing that it requires fewer HBM accesses than standard attention, and is optimal for a range of SRAM sizes. We also extend FlashAttention to block-sparse attention, yielding an approximate attention algorithm that is faster than any existing approximate attention method. FlashAttention trains Transformers faster than existing baselines: 15% end-to-end wall-clock speedup on BERT-large (seq. length 512) compared to the MLPerf 1.1 training speed record, 3× speedup on GPT-2 (seq. length 1K), and 2.4× speedup on long-range arena (seq. length 1K-4K). FlashAttention and block-sparse FlashAttention enable longer context in Transformers, yielding higher quality models (0.7 better perplexity on GPT-2 and 6.4 points of lift on long-document classification) and entirely new capabilities: the first Transformers to achieve better-than-chance performance on the Path-X challenge (seq. length 16K, 61.4% accuracy) and Path-256 (seq. length 64K, 63.1% accuracy).

Paper Link

Alibi - Variable context windows

arXiv > cs > arXiv:2108.12409

Search...

Help | Advanced

Computer Science > Computation and Language

[Submitted on 27 Aug 2021 (v1), last revised 22 Apr 2022 (this version, v2)]

Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation

Ofir Press, Noah A. Smith, Mike Lewis

Since the introduction of the transformer model by Vaswani et al. (2017), a fundamental question has yet to be answered: how does a model achieve extrapolation at inference time for sequences that are longer than it saw during training? We first show that extrapolation can be enabled by simply changing the position representation method, though we find that current methods do not allow for efficient extrapolation. We therefore introduce a simpler and more efficient position method, Attention with Linear Biases (ALiBi). ALiBi does not add positional embeddings to word embeddings; instead, it biases query-key attention scores with a penalty that is proportional to their distance. We show that this method trains a 1.3 billion parameter model on input sequences of length 1024 that extrapolates to input sequences of length 2048, achieving the same perplexity as a sinusoidal position embedding model trained on inputs of length 2048 but training 11% faster and using 11% less memory. ALiBi's inductive bias towards recency also leads it to outperform multiple strong position methods on the WikiText-103 benchmark.

Subjects: **Computation and Language (cs.CL)**
Cite as: arXiv:2108.12409 **[cs.CL]**
(or arXiv:2108.12409v2 **[cs.CL]** for this version)
https://doi.org/10.48550/arXiv.2108.12409

Paper Link

LoRA - Efficient LLM Finetuning

arXiv > cs > arXiv:2106.09685

Search...

Help | Advanced

Computer Science > Computation and Language

[Submitted on 17 Jun 2021 (v1), last revised 16 Oct 2021 (this version, v2)]

LoRA: Low-Rank Adaptation of Large Language Models

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example -- deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose Low-Rank Adaptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, no additional inference latency. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at this https URL.

Comments: Draft V2 includes better baselines, experiments on GLUE, and more on adapter latency

Subjects: **Computation and Language (cs.CL)**; Artificial Intelligence (cs.AI); Machine Learning (cs.LG)

Cite as: arXiv:2106.09685 [cs.CL]

Paper Link