AI-BigData Convergence (ABC) Forum
2023
Tutorial

Demystifying Large Language Models and GPT

Won Kim

Objective of This Tutorial

- Provide a broad introduction to beginners as a good starting point for in-depth learning on
- what's inside the LLM blackbox, in particular,
  - training, inference, and performance of LLMs.
  - how training text data is transformed to numbers for input to the Transformer.
  - how the input data flows through the Transformer architecture for training and inference.
- and a snapshot and prognosis of LLMs.

Roadmap: Tutorial

- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

Roadmap: Introduction

- A Snapshot of the Field
- LLM and Foundational Model

LLMs Galore (1/2)

- https://arxiv.org/abs/2303.18223ttps://analyticsindiamag.com/top-10-alternatives-to-gpt-3/
- https://en.wikipedia.org/wiki/Foundation_models
- https://businesschief.asia/technology/how-china-is-racing-to-launch-genai-models-to-rival-chatgpt
- OpenAI/MS (GPT-1, GPT-2, GPT-3, GPT-3.5, GPT-4)
- Google (BERT, T5, LaMDA, GLaM, PaLM, PaLM2)
- DeepMind (Gopher, ChinChilla)
- MS (Turing-NLG)
- MS and NVidia (Megatron-Turing NLG)
- Meta (XLM, RoBERTa, LLaMA 2)
- Amazon (Titan, AlexaTM)
- Anthropic (Claude), Cohere (Cohere Command)
- IBM (Watson X)

LLMs Galore (2/2)

- China (238 LLMs)
  - Baidu (Ernie 3.0, 3.5), Alibaba Cloud (Tongyi Qianwen), Tencent (Hunyuan), Huawei Cloud (Pangu 3.0), JD.com (ChatRhino)
- Korea
  - Naver (HyperCLOVA X), LG AI (ExaOne), KT (MiDeum), SKT, Kakao, Conan (ConanLLM), TUNiB (Polyglot), etc.
- Israel
  - AI21 Labs (Jurassic 1, Jurassic 2)
- Arabic
  - NOOR

Open Source LLMs (1/2)

- Hugging Face (open source ML community)
  - develops tools and resources, including Transformer library and APIs, to build, deploy, and train machine learning models
- BLOOM
  - 176 billion parameters, trained on 1.6TB of text data
  - can generate text in 46 natural languages and 13 programming languages
- (Meta) OPT, LLaMA 2 (for research only), Open LLaMA
- Falcon (Abu Dhabi)
  - Apache 2.0 license (can use commercially)
  - trained on 11 languages
- (Databricks) Dolly 2.0

Open Source LLMs (2/2)

- (ELeutherAI) GPT-Neo (open source GPT-2), GPT-J (open source GPT-3)
- (100+ derived from LLaMA)
  - Guanaco, Alpaca, Vicuna, Cerebras, etc.
- (Mosaic ML) MPT
- (Stability) StableLM
- (Google) T5
- (Together.ai, etc.) RedPajama
- (China, Singapore) NExT-GPT
- (Mistral AI, France) Mistral 7B

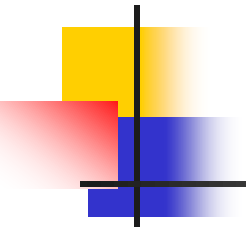
of Parameters of Some LLMs

- https://kyleake.medium.com/data-behind-the-large-language-models-llm-gpt-and-beyond-8b34f508b5de

Model	Organization	Date	Size (# Params)
ELMO	AI2	Feb 2018	94 (million)
GPT	OpenAI	June 2018	110
BERT	Google	Oct 2018	340
XLM	Facebook	Jan 2019	655
GPT-2	OpenAI	Mar 2019	1,500
RoBERTa	Facebook	July 2019	355
Megatron-LM	NVidia	Sep 2019	8300
T5	Google	Oct 2019	11,000
Turing-NLG	Microsoft	Feb 2020	17,000
GPT-3	OpenAI	May 2020	**175,000**
Megatron-Turing NLG	MS/NVidia	Oct 2021	**530,000**
Gopher	DeepMind	Dec 2021	**280,000**

GPT-4 : Some Incredible Numbers

- https://eightify.app/summary/artificial-intelligence-and-technology/gpt-4-leak-unveiling-all-details#:~:text=GPT%2D4's%20batch%20size%20increased,and%20accessible%20for%20small%20businesses.
- https://beebom.com/best-large-language-models-llms/
- parameters : 1.76 trillion
  - mixture of 8 models, each with 220 billion parameters
- batch size :16 million
- context window size : 32k tokens
- Transformer layers : 128 layers
- Training cost : US$63 million, 25,000 NVidia A100 GPUs



Roadmap: Introduction

- A Snapshot of the Field
- LLM and Foundational Model

Language Model (1/2)

- https://dugas.ch/artificial_curiosity/GPT_architecture.html#:~:text=Of%20course%2C%20the%20embedding%20dimensions,a%2012288%20dimension%20embedding%20vector
- A language model determines the next word to come after a given sequence of words.
- Inputs and outputs of a language model
  - The input is a sequence of N words (tokens).
  - The output is a guess for the word most likely to be put at the end of the input sequence.
  - (e.g.) sequence: Not all heroes wear -> (next word) capes

Language Model (2/2)

- But how do we append more than one word?
  - Simple.
  - After we get the next word, we add it to the sequence, and get the following word.
  - Not all heroes wear capes -> but
  - Not all heroes wear capes but -> all
  - Not all heroes wear capes but all -> villains
  - Not all heroes wear capes but all villians -> do
- repeat as much as desired, and we end up with long generated texts.

Large Language Model (LLM) (1/2)

- A language model can be of varying complexity, from simple n-gram models to more sophisticated neural network models.
- A large language model (LLM) refers to the use of deep learning techniques and a large number of parameters, which can range from millions to hundreds of billions.
- An LLM is typically too massive to run on a single computer and is, therefore, provided as a service over an API or web interface.

Large Language Model (LLM) (2/2)

- An LLM can “learn” complex textual data and generate new text that is coherent and grammatically accurate.
- Thus an LLM can perform many types of natural language processing(NLP) tasks,
  - such as language translation, text summarization, sentiment analysis, chatbot conversations, and more.

Image by NLPlanet

Note: GPT vs. LMM, GPT vs. ChatGPT

- GPT is an LLM, i.e., a language model, not an app.

- ChatGPT is a web app (you can access it in your browser) designed specifically for chatbot applications—and optimized for dialogue.
- ChatGPT relies on GPT to produce text, like explaining code or writing poems.

Chatbots Using LLM

- https://www.zdnet.com/article/best-ai-chatbot/
- OpenAI ChatGPT (uses GPT-3.5/4)
- MS Bing Chat (uses GPT-4)
- Perplexity AI (uses GPT-3/4)
- Jasper (for businesses) (uses GPT-3) (* for pay)
- YouChat (uses GPT-3)
- Chatsonic (by Whitesonic) (uses GPT-4) (* for pay)
- Google Bard (uses PaLM 2)
- …

Multimodal LLMs

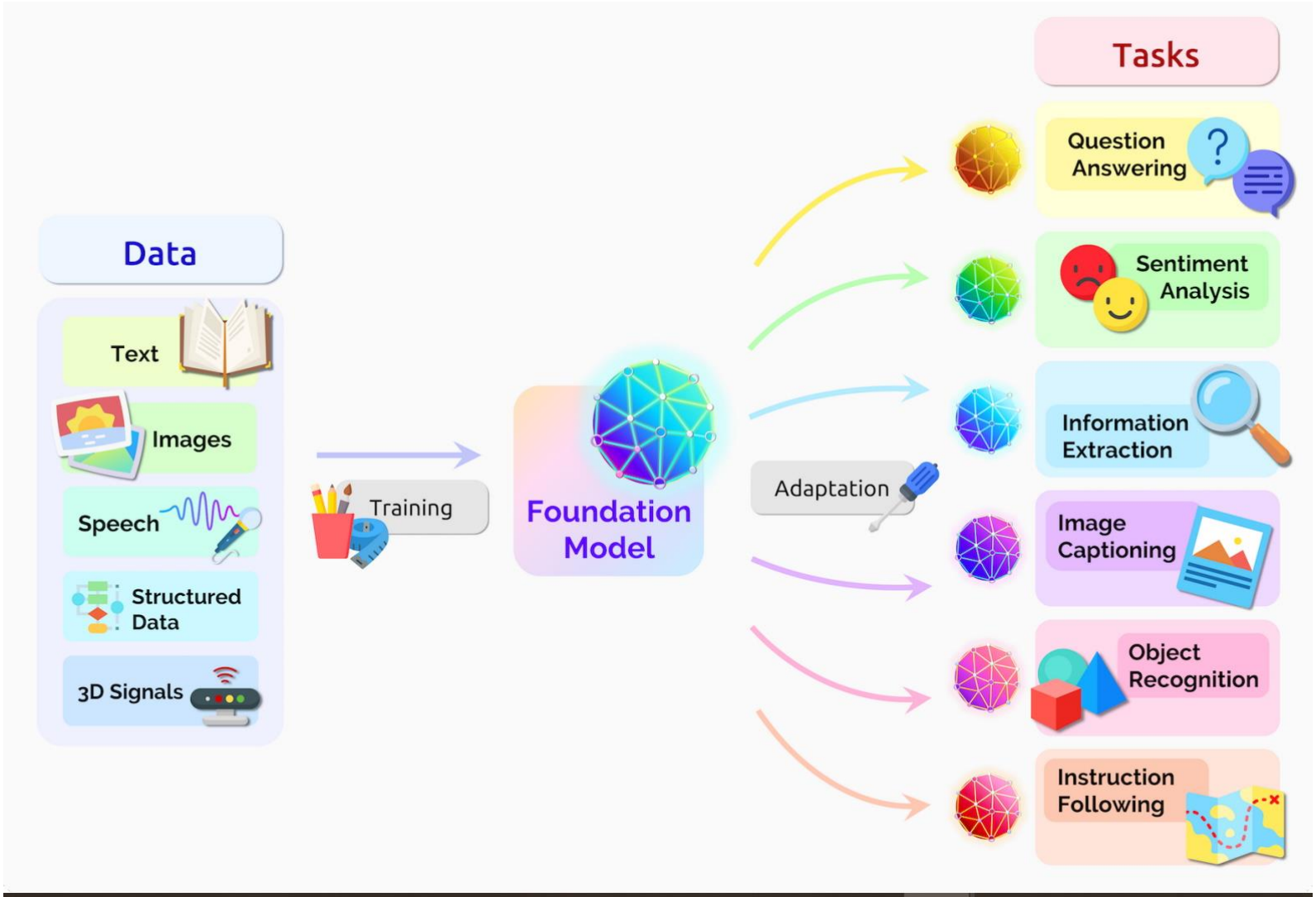
- https://en.wikipedia.org/wiki/Generative_artificial_intelligence
- Input and output
  - (input: text + image, text + audio, text + music,..)
    (output: image, music, video,…)
  - (input: image, output: text)
- (OpenAI) DALL-E, GPT-4 (Stability, open source) Stable Diffusion, Midjourney, FireFly, (Google) Imagen
- (OpenAI) Codex
- (Google) MusicLM, MusicLM Pytorch (open source), (Meta) AudioCraft (open source), (UK) AudioLDM
- (Google) PaLM-E, (MS) Florence
- (ModelScope, China) ZeroScope

Foundational Models

- https://en.wikipedia.org/wiki/Foundation_models
- A foundation model (also called base model) is a large machine learning (ML) model
- trained on a vast quantity of data
  - (often by self-supervised learning or semi-supervised learning) and
- generates data, such that it (the model) can be adapted to a wide range of downstream tasks.

Visualization

- https://blogs.nvidia.com/blog/2023/03/13/what-are-foundation-models/



Types of Foundation Models

- (* a simple taxonomy – there is no consensus taxonomy.)
- Large Language Models (LLMs)
  - text input – text output
  - GPT, BERT, T5, …
- Multimodal Models
  - text input – image, music, etc. output
  - Stable Diffusion, DALL-E, Imagen, Visual ChatGPT, MusicLM
- Computer Vision Models
  - generates labels from image, video data
  - ResNet, EfficientNet, YOLO
- Generative Models
  - generates data similar to input data
  - GANs, VAEs (variational autoencoders)

Technology Trends

- Note: Moving at a break-neck pace, and there will be an inevitable shakeout and settling in a few years.
- Formation of an LLM Ecosystem
  - LLM, enabling tools, end-user applications and services
- Upsizing of LLM
  - by BigTech companies
- Downsizing of LLM (sLLMs)
  - 150 (as of September 2023)
- On-Premise LLMs/sLLMs
- Spread of Open Source LLMs
  - many derived from Meta LLaMA
- Emergence of Multimodal LLMs

Roadmap: Tutorial

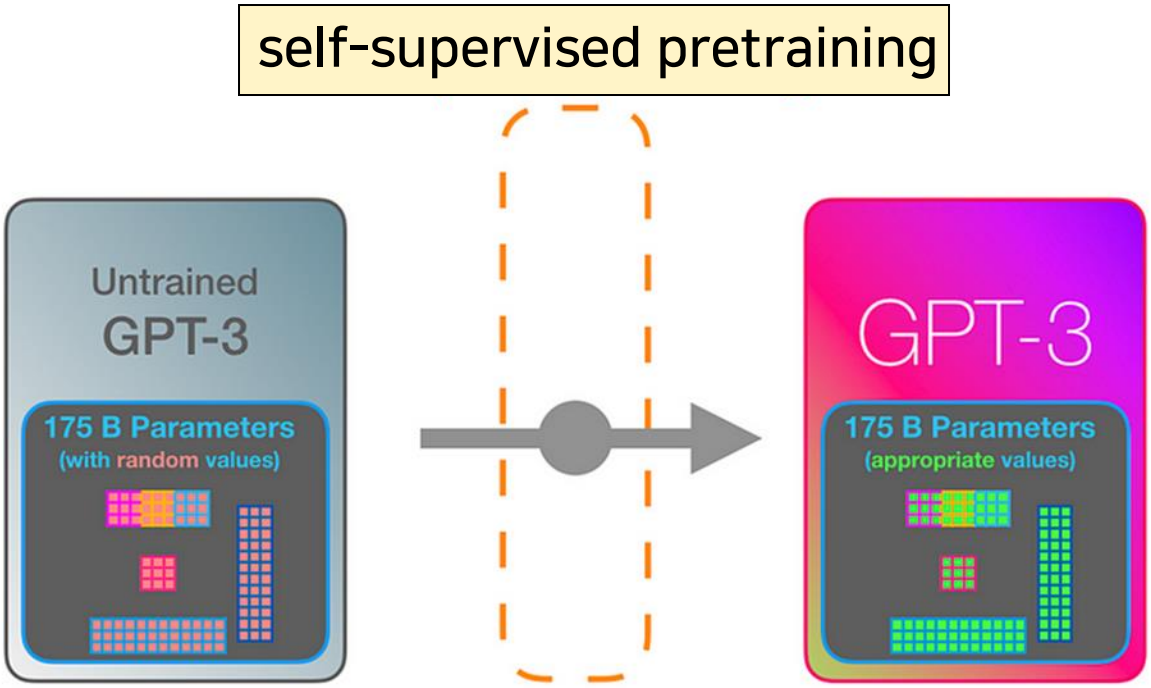
- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

Roadmap: Training and Inference

- Pretraining
- Fine-Tuning
- Inferencing
- Hallucination

Pretraining

- https://jalammar.github.io/how-gpt3-works-visualizations-animations/
- Turns initial random values of the model's untrained weights (parameters) to appropriate trained values.



Two Stages of Pretraining: A First Look (1/2)

- https://ai.stackexchange.com/questions/40179/how-does-the-decoder-only-transformer-architecture-work
- https://medium.com/data-driven-fiction/decoder-only-transformer-model-521ce97e47e2
- First stage: Basic training
- The basic training proceeds roughly as follows:
  - gather lots of text.
  - strip the last word from that text.
  - feed it as input into the Transformer.
  - check if the prediction matches the word that was stripped, and
  - backpropagate the error.
- After basic training is completed, we have an LLM which can predict the next word based on a context.
- The basic training process involves self-supervised learning.

Preparing the Training Data

- For each sequence of 'N' words (tokens), prepare an input sequence of 'N-1' words, and expected output word.
- Example text : "I am learning data science".
  - Strip the last word from the text consecutively.
  - (N=2) Input : ['I'] Expected_output : ['am']
  - (N=3) Input : ['I', 'am'] Expected_output : ['learning']
  - (N=4) Input : ['I', 'am', 'learning']
    Expected_output : ['data']
  - (N=5) Input : ['I', 'am', 'learning', 'data']
    Expected_output : ['science']
- Every text/sentence/book/webpage can be separated into sequences of words (tokens).

Training: A First Look (2/2)

- Second stage: Fine-tuning
- Note: Fine-tuning is not always necessary.
- Fine-tuning an LLM involves adjusting and adapting a pretrained model to perform specific tasks or to cater to a particular domain more effectively.
- The process usually entails training the model further on a smaller, targeted dataset that is relevant to the desired task or subject matter.

Format of the Training and Validation Datasets for Fine-Tuning

- The dataset is a set of question ('prompt') and answer ('completion') pairs.
- The required Q&A format (and example in red)

```
{
 "prompt": "What is the primary function of the
           heart? ->",
 "completion": """ The primary function of the heart is
          to pump blood throughout the body.₩n ₩n"""
}
```

- The Q&A pairs can be generated using ChatGPT.

Training Datasets and Context Windows for the GPT Series

- https://www.makeuseof.com/gpt-models-explained-and-compared/

Model	Launch date	Training data	# of Parameters	Dataset size	Max Sequence Length
GPT-1	June 2018	Common Crawl, BookCorpus	117 million	4.5 GB	1024
GPT-2	Feb. 2019	Common Crawl, BookCorpus, WebText	1.5 billion	40 GB	2048
GPT-3	June 2020	Common Crawl, BookCorpus, Wikipedia, Book, Articles, etc.	175 billion	580 GB	4096
GPT-4	March 2023		1760 billion	1 PB	8192

Training Cost

- Training of the GPT-2 (1.5 billion parameters) in 2019 cost $50,000.
- Training of the PaLM (540 billion parameters) in 2022 cost $8 million.
- Training ChatGPT took 10,000 NVidia GPUs.
- Training GPT-4 took 25,000 NVidia A100 GPUs.
- Lambdalabs estimated a hypothetical cost of around $4.6 million US dollars and 355 years to train GPT-3 (175 billion parameters) on a single GPU in 2020, with lower actual training time by using more GPUs in parallel.
- The explosion of the training cost (and unwanted inclusion of private data to the training data) is leading to the development of small specialized LLMs(sLLM).

Memory Needs for Training

- LLMs are typically trained with full- or half-precision floating point numbers (float32 and float16). One float16 has 16 bits, or 2 bytes.
- So one billion parameters require 2 gigabytes.
- The largest models typically have 100 billion parameters, requiring 200 gigabytes to load,
  - outside the range of most consumer electronics.

Meeting Major Challenges for LLM Pretraining

- Approaches to speed up the training
  - distributed training
  - finding optimal architecture hyperparameters
  - mixed precision training
  - optimized (GPU) memory use
- Approach to improve performance and speed up the training
  - data preprocessing

Distributed Training

- https://huggingface.co/blog/pytorch-fsdp
- https://huggingface.co/docs/transformers/perf_train_gpu_one
- Data parallelism, using ZERO optimizer
  - shard model parameters, gradients and optimizer states across data parallel workers/GPUs
  - CPU offload -- offload shards to the host CPU
- Tensor parallelism
  - shard parameters of individual layers across accelerators/GPUs
- Pipeline parallelism
  - put different layers of the model across different accelerators/GPUs and use pipelining
- Combination of the above

Finding Optimal Architecture Hyperparameters

- https://huggingface.co/docs/transformers/perf_train_gpu_one
- Experiment and find an optimal choice
  - batch size
  - (gradient descent) optimizer
    - Adam, AdamW, AdaFactor, 8-bit Adam, ⋯
  - architecture
    - number of hidden layers, number of attention heads, etc.

Optimized Memory Use: ZeRO (1/2)

- https://www.microsoft.com/en-us/research/blog/zero-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/
- https://arxiv.org/pdf/1910.02054.pdf
- https://huggingface.co/blog/zero-deepspeed-fairscale
- ZeRO (Zero Redundancy Optimizer) paper from Microsoft Research
  - sharding and distributed storage of optimizer states, gradients, and model parameters across GPUs, with zero overlap.
  - use of reduced precision or mixed precision for the model parameters (weights) from float32 to float16 or bfloat(brainfloat)16.
  - Offloading some processing and memory needs to the host CPU.

Optimized Memory Use: ZeRO (2/2)

- Two open-source implementations of ZeRO
  - (Microsoft) DeepSpeed library
  - (Meta) FairScale or PyTorch FSDP(Fully Sharded Data Parallel) library
- Integrated with the Transformer and included in the Hugging Face Trainer

Data Preprocessing

- https://wandb.ai/wandb_gen/llm-data-processing/reports/Processing-Data-for-Large-Language-Models--VmlldzozMDg4MTM2
- remove junk data
- remove machine-generated text
- remove duplicated data
- normalize data
- augment data
- seperate training data and testing data
- filter toxic and biased data
- ...

Roadmap: Training and Inference

- Pretraining
- Fine-Tuning
- Inferencing
- Hallucination

Fine-Tuning

- Fine-tuning can be done on the entire neural network, or on a subset of its layers.
- Fine-tuning can be combined with a reinforcement learning from human feedback.
  - ChatGPT and (DeepMind) Sparrow are examples.
- A model may also be augmented with adapters that consist of far fewer parameters than the pretrained model.
  - Only the weights of the adapters are fine-tuned, leaving the rest of the model's weights frozen.

Parameter-Efficient Fine-Tuning (PEFT)

- https://www.leewayhertz.com/parameter-efficient-fine-tuning/
- Full fine-tuning trains the entire pretrained model, including all its layers and parameters.
  - This can be computationally very expensive.
- Parameter-efficient fine-tuning focuses on training only a subset of the pretrained model's parameters.
  - This approach involves identifying the most important parameters for the new task and only updating those parameters during training.

PEFT Techniques

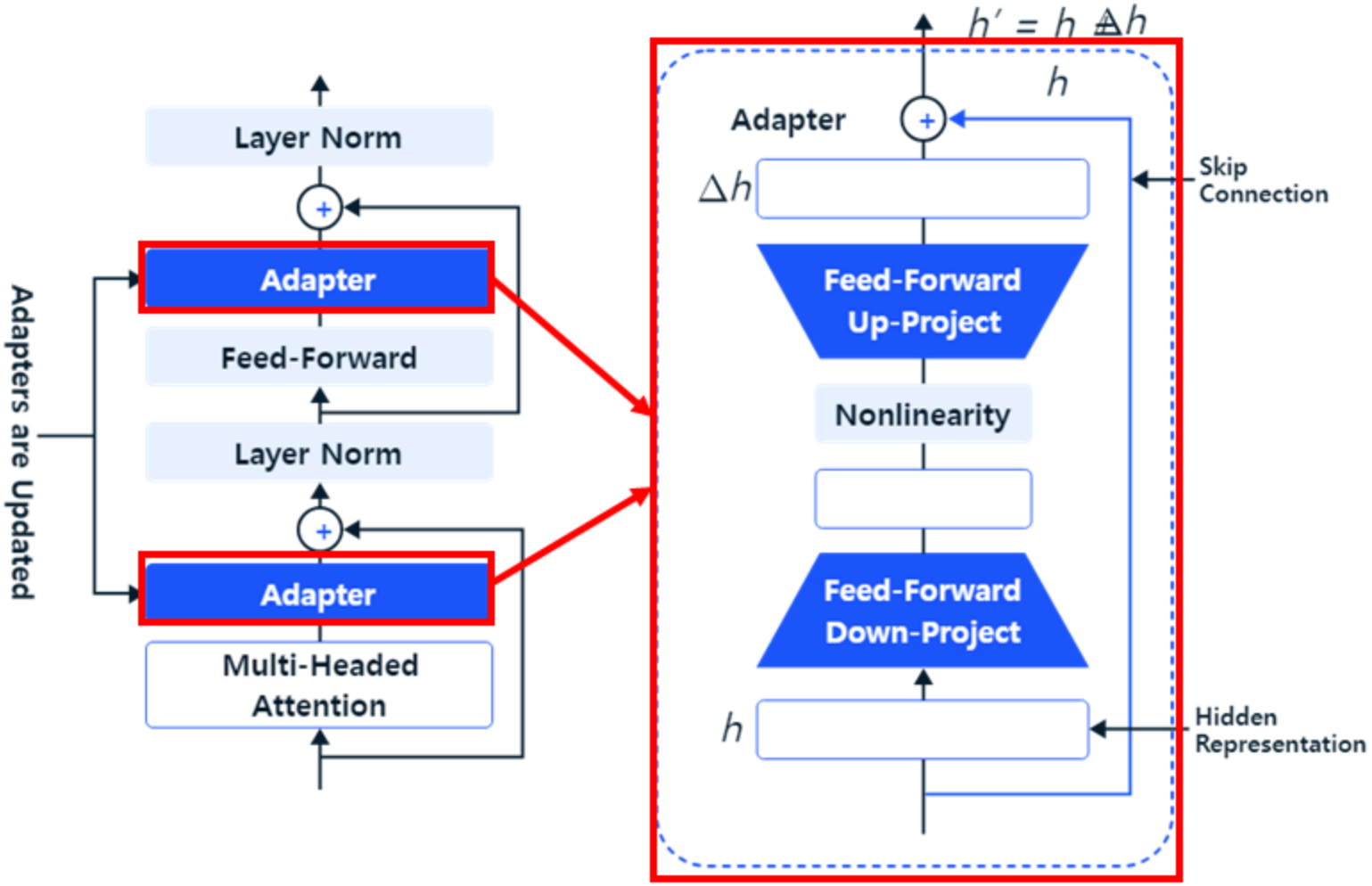
- Adapter
- LoRA (Low Rank Adaptation)
- Prefix Tuning
- Prompt Tuning
- P Tuning
- IA3 (Infused Adapter by Inhibiting and Amplifying Inner Activations)

Adapter

- https://www.leewayhertz.com/parameter-efficient-fine-tuning/
- An adapter is a special submodule that can be added to a pretrained LLM to modify its hidden representation during fine-tuning.
- It is inserted after the attention and feed-forward layers in the transformer architecture.
- Only the parameters in the adapters are updated during fine-tuning
  - while the rest of the model parameters is frozen.

Position and Architecture of the Adapter in the Transformer

- Note: h : hidden representation of the model
- Δh: hidden representation of the adapter



LoRA (Low Rank Adaptation)

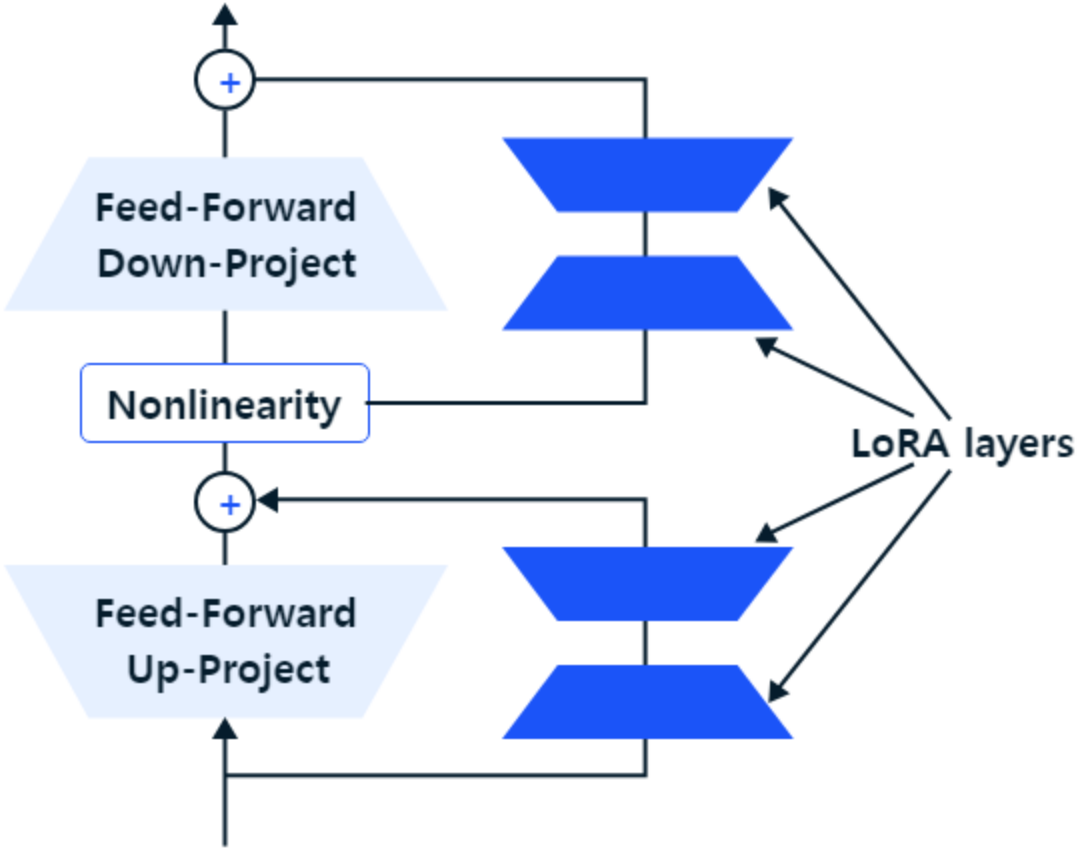
- https://www.leewayhertz.com/parameter-efficient-fine-tuning/
- https://www.anyscale.com/blog/fine-tuning-llms-lora-or-full-parameter-an-in-depth-analysis-with-llama-2
- LoRA is implemented in the Hugging Face Parameter-Efficient Fine-Tuning(PEFT) library.
- Similar to the adapters, LoRA is a small trainable submodule that can be inserted into the Transformer architecture.
- It involves freezing the pre-trained model weights and injecting trainable "low rank decomposition matrices" into each layer of the transformer architecture.
- LoRA can minimize the number of trainable parameters by up to 10,000 times and the GPU memory needs by 3 times while still performing on par or better than fine-tuning model quality on various tasks.

LoRA in the Transformer Architecture

- LoRA is inserted in parallel to the modules in the pre-trained transformer model,
  - (specifically) in parallel to each of the two feed-forward layers.
- A feed-forward layer has two projection layers and a non-linear layer in between them,
  - where the input vector is projected into an output vector with a different dimensionality using an affine transformation.

Visualization

- Note: LoRA layers are inserted in parallel to the Feed-Forward Layer.



Power of PEFT

- Guanaco (up to LLaMA-65B) using QLoRA achieved 99% ChatGPT performance on the Vicuna benchmark
  - trained on one GPU with 48GB of VRAM in 24 hours
- LLaMA-13B (13 billion parameters) outperformed GPT-3 (with 175 billion parameters)

Roadmap: Training and Inference

- Pretraining
- Fine-Tuning
- Inferencing
- Hallucination

Inference (1/2)

- Doing inference with a Transformer is just like training.
- You insert a prompt and out comes the next word/classification/other.
- The prompt is extended one word at a time.
  - You insert the prompt, and out comes the first word of the answer.
  - The first word of the answer is now added to the prompt, creating a new, slightly different prompt.
  - This prompt is again forwarded through the model, giving the prediction of a new word. …

Inference (2/2)

- As the output is the probability for each token to be the next one, you can do one of the following during inference.
- (1) Take the token that has the highest probability.
  - The model becomes deterministic.
- (2) Sample from the probability distribution; that is, you take a token that does not have the highest probability.
  - This induces some randomness into the algorithm.

Methods for Selecting the Next Token

- https://peterchng.com/blog/2023/05/02/token-selection-strategies-top-k-top-p-and-temperature/#:~:text=They%20are%20just%20ways%20to,certain%20probability%20mass%20(p).
- Many LLMs support sampling methods that control the randomness of the selection of the next token.
- Three sampling methods
  - top-k sampling
  - top-p sampling
  - temperature
- Note: The temperature parameter should be used in conjunction with top-p or top-k.

Top-k Sampling

- Top-k sampling proceeds as follows:
  - 1. Order the tokens in descending order of probability.
  - 2. Select the first k tokens to create a new distribution.
  - 3. Sample from those tokens.
- Example: k=3 ?

token	probability
t_0	0.4
t_1	0.2
t_2	0.2
t_3	0.15
t_4	0.15

Top-p Sampling (1/2)

- This method (also called nucleus sampling), instead of selecting top-k tokens, selects enough tokens to "cover" a certain amount of probability defined by the parameter p.
- Top-p sampling proceeds as follows:
  - Order the tokens in descending order of probability.
  - Select the smallest number of top tokens such that their cumulative probability is at least p.
  - Sample from those tokens.

Top-p Sampling (2/2)

- Suppose p=0.5. Using the current example, top-(p=0.5) sampling proceeds as follows:
  - The top token, t_0 is selected. It has a probability of 0.4 and the cumulative probability is 0.4 (less than p=0.5)
  - So we select the next token.
  - The next token, t_1 has a probability of 0.2, and now the cumulative probability is 0.6.
  - The cumulative probability is at least p=0.5, so we stop.
- This results in only the top 2 tokens being selected:

token	probability
t_0	0.4
t_1	0.2
t_2	0.2
t_3	0.15
t_4	0.15

token	probability
t_0	0.4
t_1	0.2

Temperature (1/2)

- Top-k and top-p samplings operate on the softmax output probabilities.
- However, the introduction of temperature results in a change to the softmax function itself, as follows:

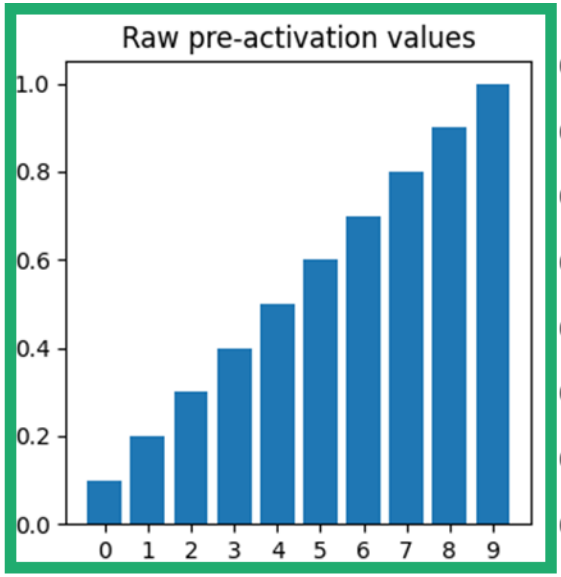
$$\text{softmax}(z_i \div T) = \exp(z_i \div T) / \Sigma_j \exp(z_j \div T)$$

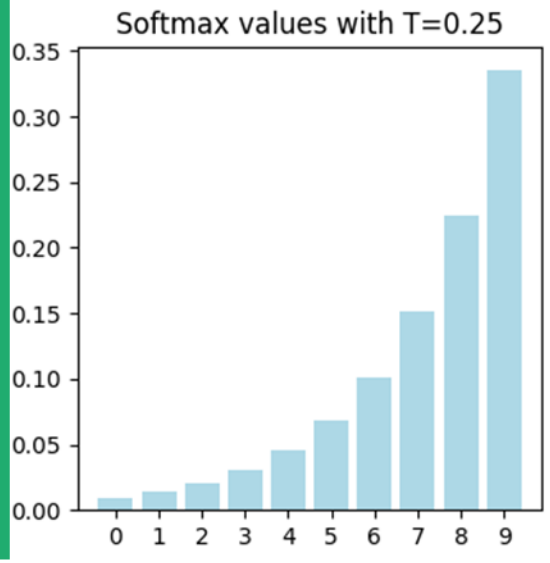
where i=1,…K, and T (>0) is the temperature

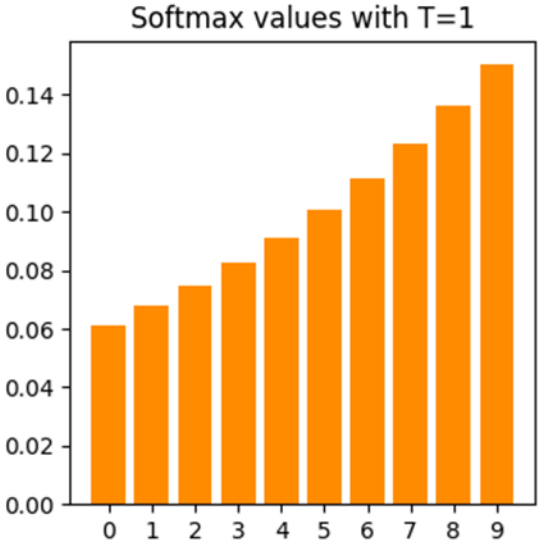
- The impact of 1/T on the softmax function
  - If 0 < T < 1, z_i values get pushed away from 0 and the differences between input values get amplified.
  - If T > 1, z_i values get pushed toward 0 and the differences between input values get reduced.

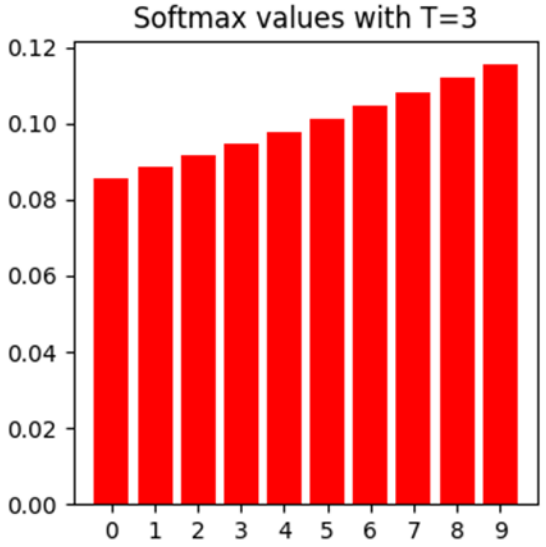
Visualization

- Note how different values of T (0.25, 1, 3) alter the relative differences between elements of the raw pre-softmax vector.









Temperature (2/2)

- The temperature changes the shape of the softmax output probability distribution.
- If the temperature increases, differences in output probabilities are reduced, and more tokens end up with probabilities close to the highest probability.
  - The generated text will be more diverse, but there is a higher possibility of grammar mistakes and generation of nonsense (i.e., hallucination).
- If the temperature decreases, the highest probability become more pronounced.
  - The model will probably output the most correct text, but rather boring, with small variation.

Examples

- https://algowriting.medium.com/gpt-3-temperature-setting-101-41200ff0d0be
- (e.g.,) temperature=0
  - prompt: “how to make“
  - completion: “a good impression on a first date”
- (e.g.,) temperature=1
  - prompt: “how to make“
  - completion: “turban wrapped coconut hairdo” or
    “mayo without almost no oil at all” or
    “a new server in minecraft 1” or ···

Beam Search

- https://towardsdatascience.com/foundations-of-nlp-explained-visually-beam-search-how-it-works-1586b9849a24
- Selects the 'k' best sequences of tokens
  - considers the probabilities of the combination of all of the preceding tokens along with the token in the current position.
  - Computationally more expensive, but produces more accurate result
- "k" is a hyperparameter called "beam width".

Prompt Engineering (1/2)

- https://en.wikipedia.org/wiki/Prompt_engineering
- LLMs can solve various target-specific tasks without being fine-tuned.
- They only need to be "prompted", often using a few examples of similar problems and their respective solutions.
- Such “few-shot prompting” has sometimes given even better results than fine-tuning in various tasks.
  - translation, question answering, cloze tasks, unscrambling words, using a novel word in a sentence
- The creation and optimization of such prompts is called prompt engineering.

Note: Zero-Shot, One-Shot, Few-Shot Prompt

- Zero-shot prompt

  task description: Translate English to French

  prompt: cheese →

- One-shot prompt

  task description: Translate English to French

  example: sea otter → loutre de mer

  prompt: cheese →

- Few-shot prompt

  task description: Translate English to French

  examples: sea otter → loutre de mer
  peppermint → menthe poivre
  plush giraffe → giraffe peluche

  prompt: cheese →

Prompt Engineering (2/2)

- Prompt engineering involves
  - phrasing a query
  - specifying a style
  - providing relevant context, or
  - assigning a role to LLM, such as "Act as a native French speaker".
- The user can use either a single prompt or multiple successive prompts.

Prompt Engineering: Multiple Successive Prompts (1/2)

- chain-of-thought prompt
  - prompts the model to solve a problem as a series of intermediate steps
- generated-knowledge prompt
  - prompts the model to generate relevant facts for completing the prompt, then proceed to complete the prompt
- self-refine prompt
  - prompts the model to solve the problem, then prompt it to critique its solution, then prompt it to solve the problem again

Prompt Engineering: Multiple Successive Prompts (2/2)

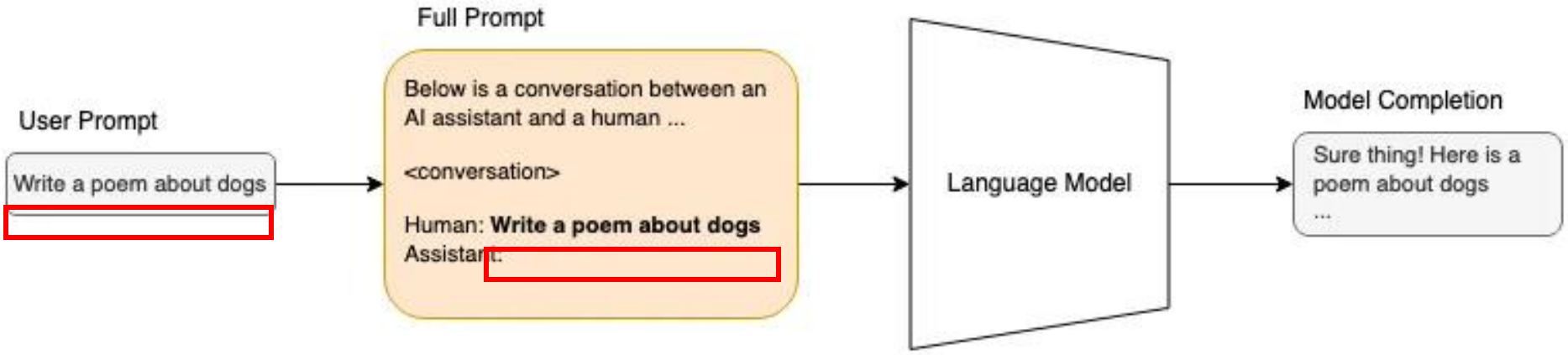
- tree-of-thought prompt
  - prompts the model to generate "possible next steps", and then run the model on each of the possible next steps
- least-to-most prompt
  - prompts the model to list the sub-problems to a problem, then solve them in sequence, with the help of answers to previous sub-problems.

Prompt Marketplaces

- https://writesonic.com/blog/ai-prompt-marketplaces/
- Chatsonic prompt library (free, for ChatGPT)
- AIPRM : (for ChatGPT)
- PromptBase, ChatX
  - (for DALL-E, Stable Diffusion, Midjourney, ChatGPT)
- FlowGPT : (for ChatGPT)
- PromptHero : (for ChatGPT, art prompts)
- …

In-Context Learning (1/2)

- https://medium.com/geekculture/how-to-build-chatgpt-a01f9bd6d8ab
- The prompt is actually inserted into a larger prompt that contains the entire conversation.
- This allows the language model to understand the context of the conversation and respond appropriately.



In-Context Learning (2/2)

- Note that the model figures out what words come next based on probabilities it learned during pretraining.
- Each new prompt-completion pair is added to the current conversation. This is how the model "memorizes" the context of the new prompt.
  - There is a limit on the sequence length (or context window size).
  - It is 4096 tokens for GPT-3.

Retrieval Augmented Generation (RAG) (1/3)

- https://research.ibm.com/blog/retrieval-augmented-generation-RAG
- It is a popular technique that supplements LLM's internal representation of information with facts fetched from external sources.
- Benefits
  - The model has access to the most current, reliable facts.
  - Users have access to the model's sources, ensuring that its claims can be checked for accuracy and ultimately trusted.
  - It reduces the chances that an LLM will leak sensitive data, or 'hallucinate' incorrect or misleading information.

Retrieval Augmented Generation (RAG) (2/3)

- Retrieval phase
  - It searches for and retrieves information relevant to the user's prompt.
    - In an open-domain, consumer setting, they can come from indexed documents on the internet.
    - In a closed-domain, enterprise setting, a narrower set of sources are typically used for added security and reliability.
  - The external knowledge is appended to the user's prompt and passed to the language model.

Retrieval Augmented Generation (RAG) (3/3)

- Generative phase
  - LLM draws from the augmented prompt and its internal representation of its training data to synthesize an answer tailored to the user.
  - The answer can then be passed to the application (e.g., chatbot) with links to its sources.

Roadmap: Training and Inference

- Pretraining
- Fine-Tuning
- Inferencing
- Hallucination

Hallucination

https://blog.finxter.com/hallucinations-in-ai-with-chatgpt-examples/

Hallucination

- https://machinelearningmastery.com/a-gentle-introduction-to-hallucinations-in-large-language-models/
- https://www.telusinternational.com/insights/ai-data/article/generative-ai-hallucinations
- Hallucination refers to a phenomenon where the model generates text that is incorrect, nonsensical, or not real.
- LLMs generate text as an extrapolation from the prompt.
  - The result of extrapolation is not necessarily supported by any training data, but is the most correlated with the prompt.
- Hallucinations make the result of an LLM worse than the combined results of multiple Internet searches.
- Hallucinations are unavoidable in a language model.

Examples (1/2)

- In response to a prompt "tell me about Professor Jonathan Turley", ChatGPT answered that "Professor Jonathan Turley has been accused of sexual harassment in a 2018 Washington Post article after groping law students on a trip to Alaska."
  - Professor Turley: "It was a surprise to me since I have never gone to Alaska with students; The Washington Post never published such an article, and I have never been accused of sexual harassment or assault by anyone."

Examples (2/2)

- prompt: "(with a spec) write a program to build a decision tree classifier in COBOL" (SQL, HTML)
- completion: *(shown the beginning part only)*

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DecisionTree.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Example.
      05 District       PIC X(10).
      05 House-Type     PIC X(12).
      05 Income         PIC X(5).
      05 Previous-Cust  PIC X(3).
      05 Outcome        PIC X(15).
01 Node.
     05 Attribute      PIC X(12).
      05 ChildNodes OCCURS 0 TO 10 TIMES DEPENDING ON Num-ChildNodes.
        10 ChildNode.
           15 AttributeValue PIC X(10).
           15 Pointer       USAGE IS POINTER.
   05 Outcome        PIC X(15).
```

GPT-4 and Factuality

- GPT-4 scored close to 80% in factuality (free of hallucination) evaluations across several categories,
  - while GPT-3.5 scored under 50%.

Causes of Hallucination

- Hallucination is the Achilles heel of LLM.
- On a high level, the following problems related to the dataset and model training can lead to hallucination.
  - insufficient data, insufficient context in the training data
  - noise data
  - overfitting of the model
  - problems with data encoding and decoding
  - randomness in sampling the next-word

Mitigating Hallucinations: Two General Approaches

- Train LLM on a diverse and representative dataset that covers a wide range of real-world examples.
  - RAG(Retrieval Augmented Generation)
- Include human feedback in the training process.
  - Incorporate an understanding of the world that only a human can have into the trained model.

Utilizing Hallucinations as a Valuable Feature of LLM

- We may want LLM to hallucinate if we want it to be “creative”.
  - (e.g.) If we want LLM to give a plot of a fantasy story, we may not want it to copy from any existing one but to generate a new character, scene, and storyline.
- We may want LLM to brainstorm for us.
  - (e.g.) If we seek ideas, hallucinations can help explore different possibilities.

Roadmap: Tutorial

- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

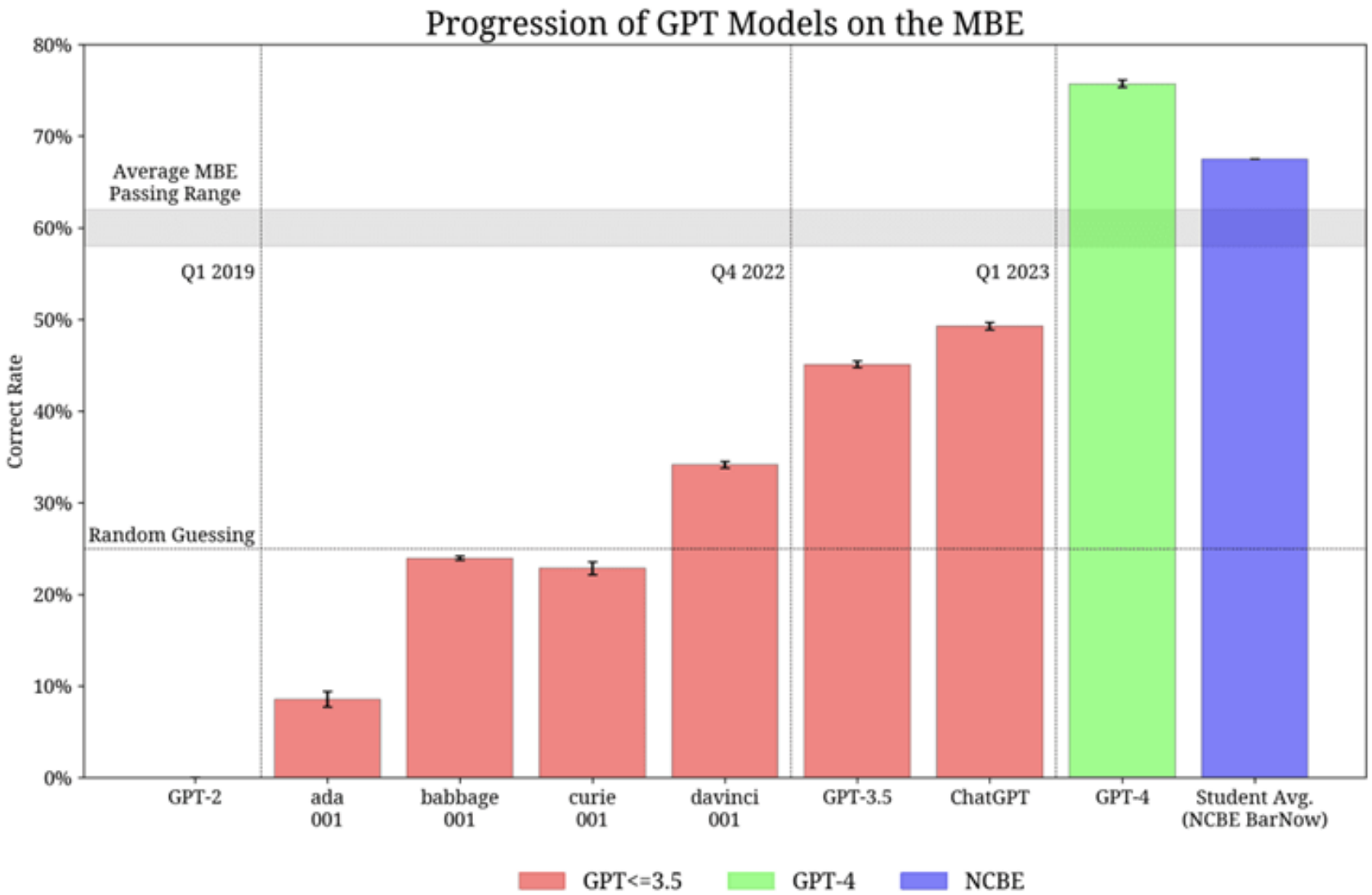
Performance of GPT on Standard NLP Modeling Tasks

Scores of GPT series in standard NLP Modeling tasks: GLUE, LAMBADA and SQuAD.

All numbers are in percentages. (source – BARD)

Model	GLUE	LAMBADA	SQuAD F1	SQuAD Exact Match
GPT-1	68.4	48.4	82.0	74.6
GPT-2	84.6	60.1	89.5	83.0
GPT-3	93.2	69.6	92.4	88.8
GPT-3.5	93.5	79.3	92.4	88.8
GPT-4	94.2	82.4	93.6	90.4

GPT Performance on MBE (multistate bar exam)



LLM Performance Benchmarks

- https://www.whytryai.com/p/llm-benchmarks
- https://msandbu.org/benchmarking-llms-and-what-is-the-best-llm/
- https://deepgram.com/learn/llm-benchmarks-guide-to-evaluating-language-models
- https://analyticsindiamag.com/the-problems-with-llm-benchmarks/
- There are many benchmarks and tests, and they are evolving.
- They can be helpful to users and developers of LLMs as a guide for adoption and further development.
- However, they have limitations.
  - They do not necessarily represent real-world usage.
  - They are often too narrow in scope.
  - The training dataset may contain errors.

Categories of LLM Benchmarks

- https://www.whytryai.com/p/llm-benchmarks
- (* Note: There is no consensus taxonomy.)
- Natural language processing (NLP)
- General knowledge and common sense
- Problem solving and advanced reasoning
- Coding

NLP Benchmarks (1/3)

- GLUE (General Language Understanding Evaluation)
- SuperGLUE
- Natural Questions
- WinoGrande
- TriviaQA
- MultiMLI (Multi-Genre Natural Language Inference)
- QuAC (Question Answering in Context)
- HellaSwag
  - (Harder Endings, Longer contexts, and Low-shot Activities for Situations With Adversarial Generations)

General Knowledge and Common Sense Benchmarks

- ARC (AI2 Reasoning Challenge)
- MMLU (Massive Multitask Language Understanding)
- OpenBookQA
- PIQA (Physical Interaction: Question Answering)
- SciQ
- TruthfulQA (* factuality test)

Problem Solving and Advanced Reasoning Benchmarks

- AGIEval
- BigBench (Beyond the Imitation Game)
- BOOIQ
- GSM8K

Coding Benchmarks

- CodeXGLUE (General Language Understanding Evaluation benchmark for CODE)
- HumanEval
- MBPP (Mostly Basic Python Programming)

Sample Q&A (1/5)

- WinoGrande

	Twin sentences	options
a	**Ann asked Mary what time the library closes, because she had forgotten**	Ann/Mary
b	**Ann asked Mary what time the library closes, but she had forgotten**	Ann/Mary

Sample Q&A (2/5)

- HellaSwag

A woman is outside with a bucket and a dog. The dog is running around trying to avoid a bath. She...

A. rinses the bucket off with soap and blow dry the dog's head.
B. uses a hose to keep it from getting soapy.
C. gets the dog wet, then it runs away again.
D. gets into a bath tub with the dog.

Sample Q&A (3/5)

- ARC (AI2 Reasoning Challenge)

Which property of a mineral can be determined just by looking at it? (A) luster [correct] (B) mass (C) weight (D) hardness

Sample Q&A (4/5)

- Big-Bench (Beyond the Imitation Game)

Input:

```
In the following chess position, find a checkmate-in-one move.

1.  e4 b6 2.  d4 Bb7 3.  Nc3 e6 4.  Nf3 Bb4 5.  Bd3 d5 6.  e5 Ne7 7.  a3 Bxc3+ 8.
bxc3 c5 9.  h4 Nf5 10.  Bxf5 exf5 11.  Ng5 g6 12.  e6 fxe6 13.  Nxe6 Qe7 14.  Qe2
Bc8 15.  Nc7+ Kf7 16.  Qxe7+ Kxe7 17.  Nxa8 Kd6 18.  Bf4+ Kc6 19.  dxc5 Nd7 20.  cxb6
axb6 21.  Rb1 Re8+ 22.  Kd2 Ne5 23.  Rxb6+ Kc5 24.  Be3+ Kc4 25.
```

Sample Q&A (5/5)

- HumanEval
- (* the last line is the code.)

```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

Benchmark Platforms

- https://msandbu.org/benchmarking-llms-and-what-is-the-best-llm/
- HuggingFace Leaderboards for open LLMs includes
  - ARC, HellaSwag, MMLU, TruthfulQA
- GPT4ALL includes
  - HellaSwag, BOOIQ, PIQA, WinoGrade, OpenBookQA, AGIEval
- Alpaca Evaluation Leaderboards
- (Stanford) Holistic Evaluation of LM (HELM)

Benchmark Scores and Number of Model Parameters

- https://msandbu.org/benchmarking-llms-and-what-is-the-best-llm/
- There is a positive correlation between the number of LLM model parameters and the benchmark scores.
  - GPT models (especially GPT-4) and LLaMA show that the higher the number of parameters the higher the score in the different benchmarks.

Roadmap: Tutorial

- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

GPT-3 Tokenization Example

- https://jalammar.github.io/illustrated-transformer/
- Note: An LLM token is NOT necessarily a full word; it may be a full word, a sub-word, a character, or a symbol.
- Example Input sentence
  - "The animal didn't cross the street because it was too tired"
- Tokenized input sentence
  - The _ animal_ didn_ '_ t_ cross_ the_ street_ because_ it_ was_ too_ tire_ d_

Tokenization

- https://huggingface.co/docs/transformers/tokenizer_summary
- LLMs are mathematical functions whose input and output are lists of numbers. Consequently, words must be converted to numbers.
- Tokenizer splits a text into words or subwords, which then are converted to integer indexes into a vocabulary table.
- The integer indexes are in the range {0,1,2,3,⋯,V-1}, where V is the vocabulary size, which can be about 50,000.
- Transformers use 3 different tokenizers: Byte-Pair Encoding (BPE), WordPiece, and SentencePiece.

Roadmap: Tutorial

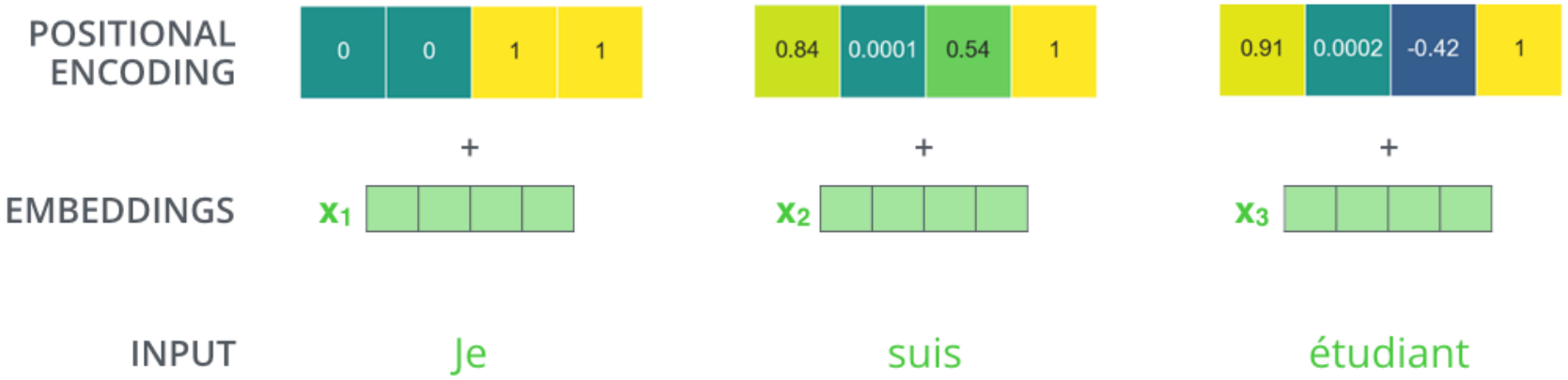
- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

Embedding and Position Encoding

- Each input word (token) enters the Transformer after undergoing two types of encoding.
  - embedding
  - position encoding
- These encodings are trainable operations.
  - This means that the encodings are not pre-decided but are learned by the model.
- The embedding encodes the meaning of the word.
- The position encoding represents the position of the word in the input sequence.
- These two encodings are vectors of numbers and they are added.

Visualization

- https://datascience.stackexchange.com/questions/55901/in-a-transformer-model-why-does-one-sum-positional-encoding-to-the-embedding-ra
- For each input word, its corresponding embedding vector and position encoding are element-wise added and the combined vector is processed by the Transformer.



Word Embedding Example

- An example word embedding for the word "king" is shown below. A word embedding is a vector of N numbers
- N could be 50 (below), 256, 512, or even12288.

[0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 , -0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961 , -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 , -0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042]

- Each number has some meaning. But ordinary mortals cannot make sense of these numbers.

Two Types of Position Encoding (1/2)

- https://towardsdatascience.com/master-positional-encoding-part-i-63c05d90a0c3
- https://towardsdatascience.com/master-positional-encoding-part-ii-1cfc4d3e7375
- https://www.linkedin.com/posts/bai-li-73a844aa_rotary-positional-embeddings-rope-combining-activity-7094803097801142272-gLjw
- absolute and relative position encoding
- Absolute position encoding captures the absolute position of a word in the input sequence.
  - (e.g.) The first word has position 1, the 50th word has position 50.
  - sinusoidal position encoding – used in GPT series

Two Types of Position Encoding (2/2)

- Relative position encoding captures the relative position two words have to each other.
  - (e.g.) In “A dog chases a cat.”, the relative position between words “dog” and “cat” would be 3.
- Some relative position encoding methods:
  - rotary position embedding (RoPE) -- used in Google PaLM, Meta LLaMA, EleutherAI (GPT-Neo, GPT-NeoX, GPT-J)
  - T5 Bias (Google T5: Text-to-Text-Transfer-Transformers)
  - ALiBi (Attention with Linear Biases)

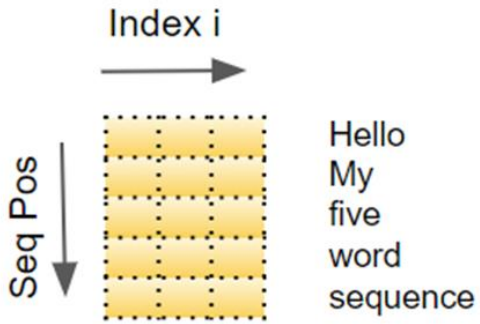
Sinusoidal Position Encoding (1/2)

- https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34
- The position constants are computed using the formula below, where

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

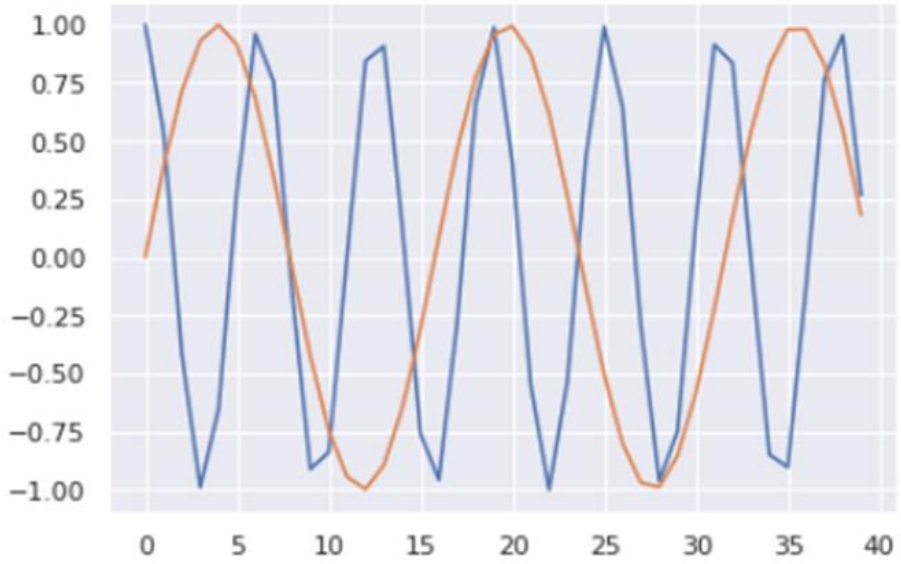
- *pos* is the position of the word in the input sequence,
- d_{model} is the length of the encoding (embedding) vector,
- *i* is the index value into the embedding vector.



Position Encoding
(Seq Len x Encoding size)

Sinusoidal Position Encoding (2/2)

- This encoding interleaves a sine curve and a cosine curve, with sine values for all even indexes and cosine values for all odd indexes.
- Suppose the input sequence has 40 words. The figure below shows the encodings for the 0^{th} index and 1^{st} index of each of the 40 words.



- The blue curve shows the encoding of the 0^{th} index, and the orange curve shows the encoding of the 1^{st} index.
- There are similar curves for all other index values.

Note

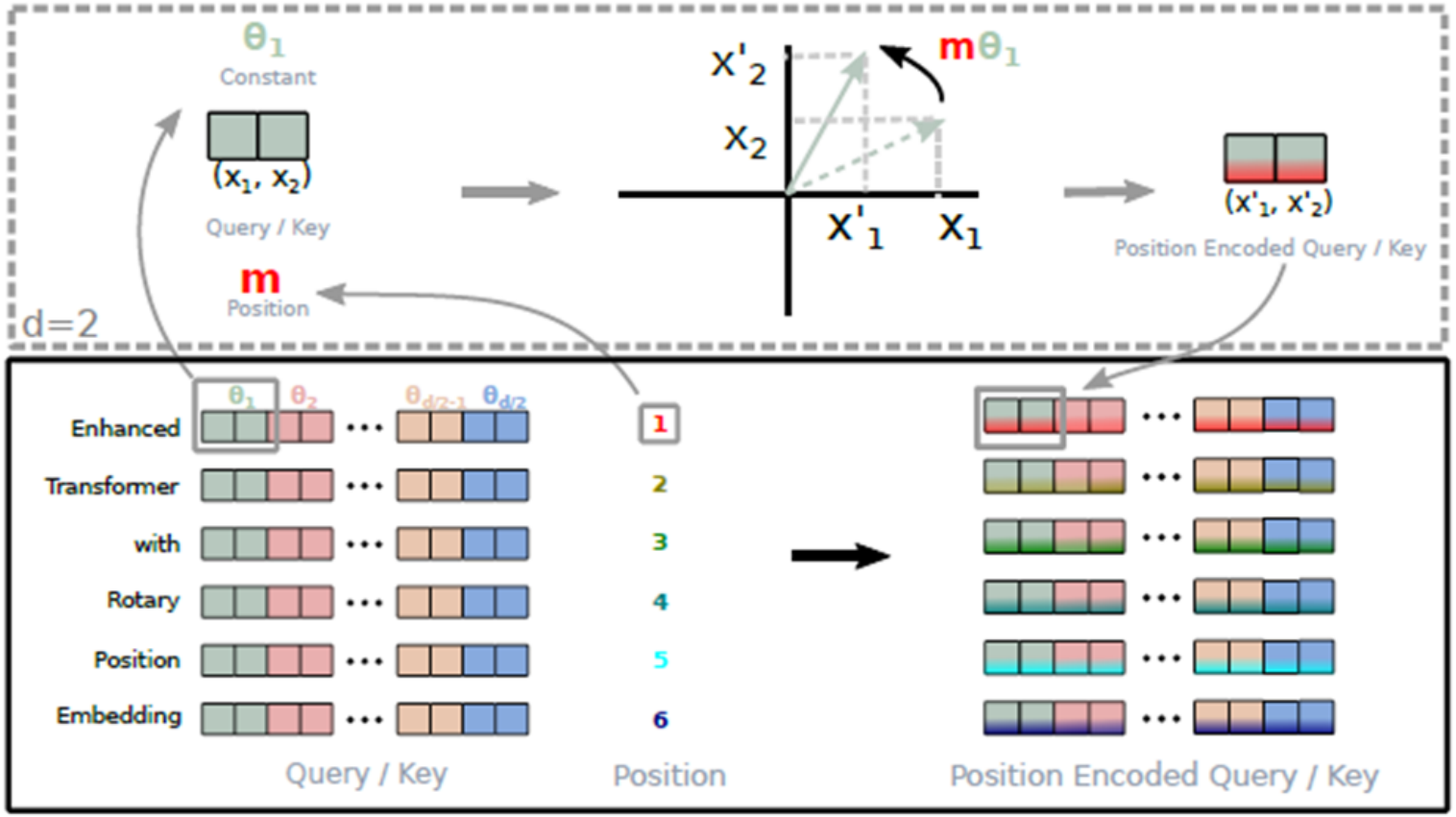
- For each word (token), position encoding vector is generated corresponding to its position in the input sequence.
  - The position encoding vector for each word is different.
- The length of the position encoding vector is the same as the length of the input embedding of the word.
- Each element of the position encoding vector has a different value, and monotonically increases as the index value increases.
- The position encoding vector and the input embedding vector are element-wise added.

Rotary Position Embedding

- https://medium.com/@andrew_johnson_4/understanding-rotary-position-embedding-a-key-concept-in-transformer-models-5275c6bda6d0
- https://www.linkedin.com/posts/bai-li-73a844aa_rotary-positional-embeddings-rope-combining-activity-7094803097801142272-gLjw
- Rotary Position Embedding (RoPE) uses absolute sinusoidal positional encoding.
- However, instead of adding the positional information to the token embedding, RoPE rotates the token embedding in a high-dimensional space, based on relative position of the tokens.
- The rotations preserve the token embedding, while reflecting both absolute and relative positional information.

Illustration

- https://arxiv.org/pdf/2104.09864.pdf
- Below, x1 and x2 are the first 2 elements of a token embedding; $\Theta_i = 10000^{-2(i-1)/d}$, $i \in [1, 2, \cdots, d/2]$



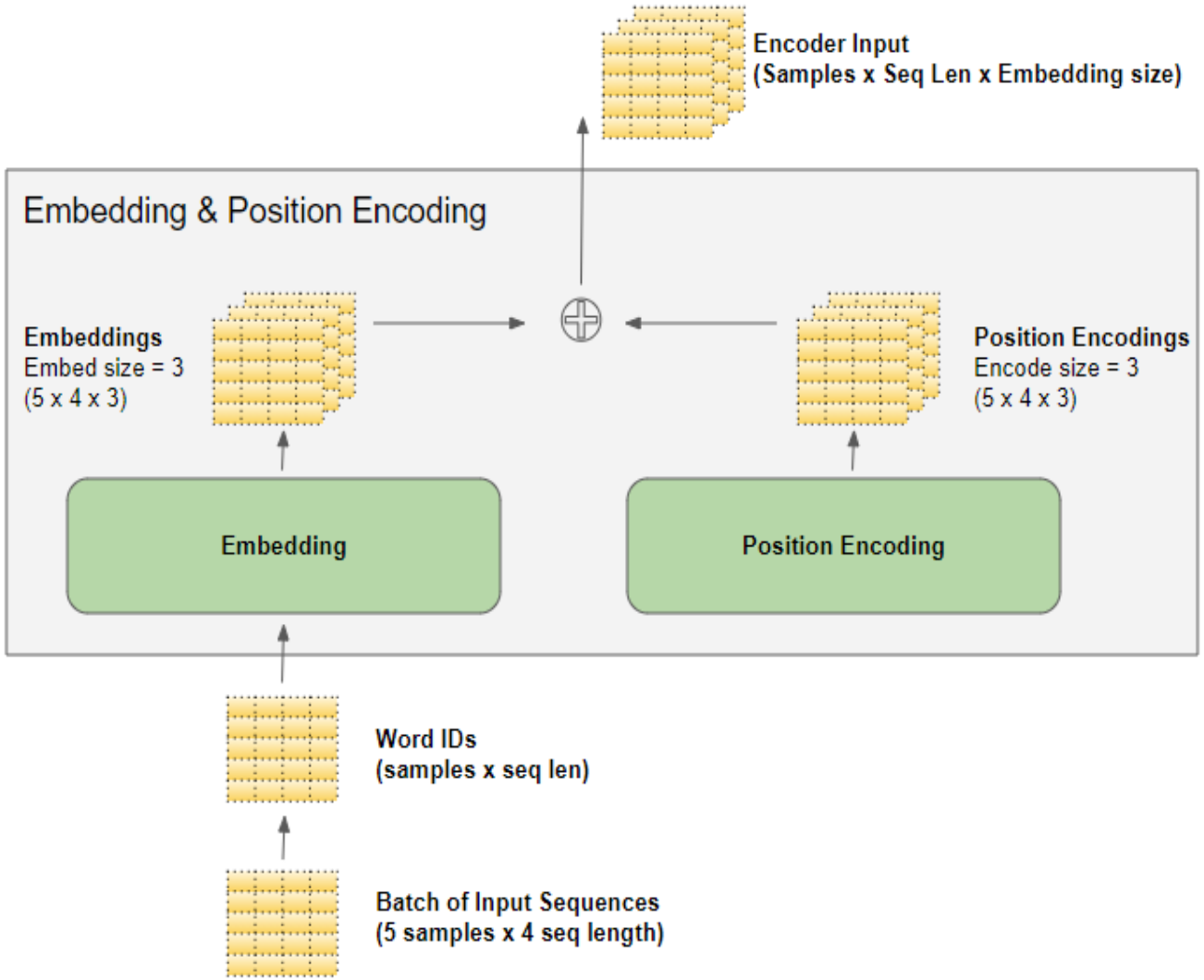
Matrix Dimensions (1/2)

- Deep learning models process a batch of training samples at a time.
- The Embedding and Position Encoding layers operate on matrices representing a batch of sample sequence.
- The Embedding layer takes a (#samples x sequence length) shaped matrix of word IDs.
- It encodes each word ID into a word vector whose length is the embedding size, resulting in a (#samples x sequence length x embedding size) shaped output matrix.

Matrix Dimensions (2/2)

- The (#samples x sequence length x embedding size) shape produced by the Embedding and Position Encoding layers is preserved through the Transformer.
- The 3D matrix is shown in the next page.

Visualization



Matrix Dimensions in GPT-3

- https://dugas.ch/artificial_curiosity/GPT_architecture.html#:~:text=Of%20course%2C%20the%20embedding%20dimensions,a%2012288%20dimension%20embedding%20vector
- sequence length: 2048 tokens
- embedding size (for each token): 12288 numbers
- position encoding size (for each position): 12288 numbers
- vocabulary size: 50257 tokens.
- batch size (#samples): 3.2 million
  - 64 and 512 for GPT-1 and GPT-2, respectively.

Transformer Output

- The Transformer generates a vector *y*, which is a probability distribution over the Transformer's vocabulary.
- The vector y is passed through softmax function to obtain *softmax(y)* for output.
- *softmax(y)* has V entries, where V is the model's vocabulary size.

Roadmap: Tutorial

- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

Attention

Attention Mechanism

- https://towardsdatascience.com/demystifying-efficient-self-attention-b3de61b9b0fb#:~:text=The%20difference%20between%20regular%20attention,focuses%20on%20a%20single%20sequence.
- The key to the Transformer's ground-breaking performance is its use of the Attention mechanism.
- It is basically a mechanism that dynamically assigns a greater importance to a few key tokens in the input sequence by altering the token embeddings.
- The goal of Attention is to allow the model to focus on (pay attention to) important parts of the input sequence of tokens.

What Does Attention Do? (1/2)

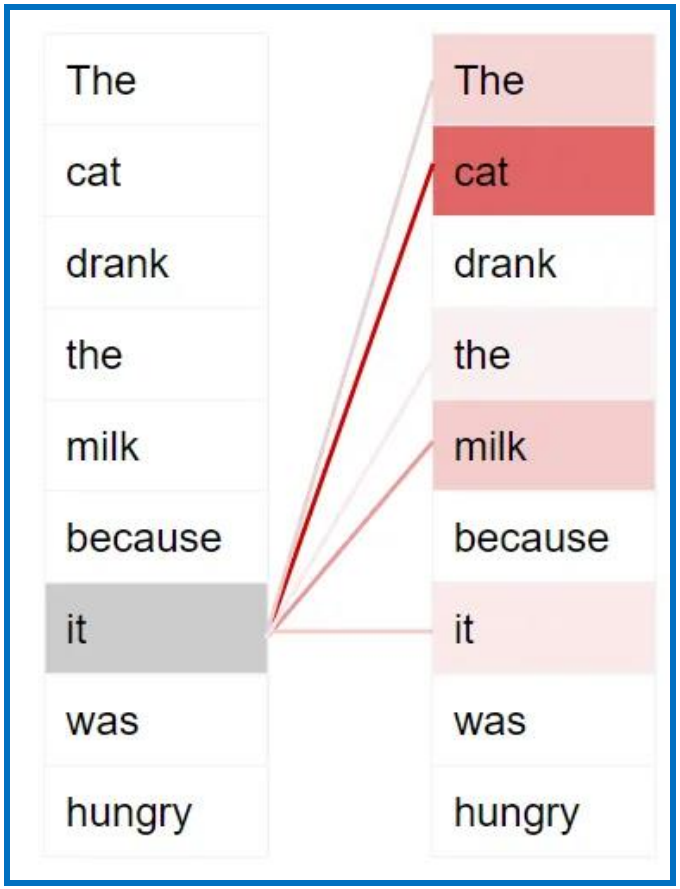
- https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452
- Attention relates every word in the input sequence to every other word in the input sequence.
- Example: Consider two sentences:

  The *cat* drank the milk because it was hungry.

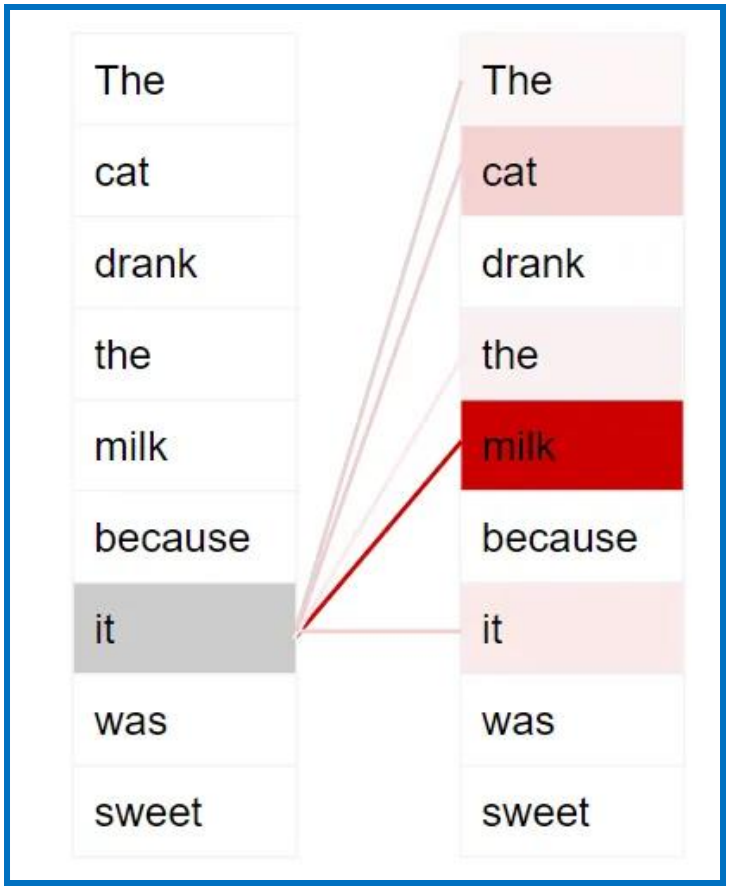
  The cat drank the *milk* because it was sweet.
- In the first sentence, (for example) the word 'it' refers to 'cat', while in the second 'it' refers to 'milk.
- When the model processes the word 'it', Attention gives the model more information about its meaning, so that it can associate 'it' with the correct word.

Visualization (of the previous page)

- In the figures below, darker shade indicates closer relationship (after training).



first sentence



second sentence

Roadmap: Tutorial

- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

Transformer

Transformer Did Not Fall from the Sky One Starry Night!

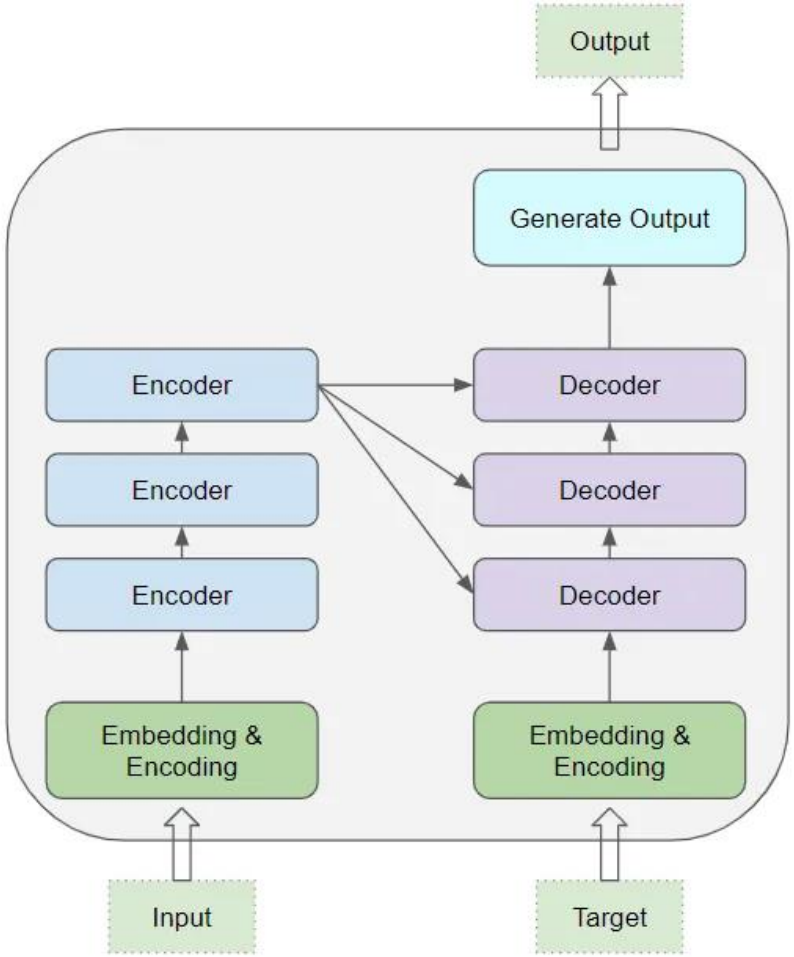
- https://en.wikipedia.org/wiki/Large_language_model#:~:text=A%20large%20language%20model%20(LLM,millions%20to%20billions%20of%20weights.
- Key technologies that have led to the Transformer
- Markov chain (1907), n-gram model
- long short term memory (LSTM) network (1997)
  - The idea was not feasible until GPUs became available in the 2010s
- AlexNet (2012) for image recognition -- by University of Toronto
- Word2vec (2013) – by Google
  - Created word embeddings
- seq2seq (2014) -- by Google
  - Used two LSTMs to perform machine translation
- Attention mechanism (2014) – by Google
- Google Translate (2016)
  - Changed from statistical machine translation to neural machine translation

Roadmap: Architecture -- Overall

- A Quick Look
- Decoder-only Architecture

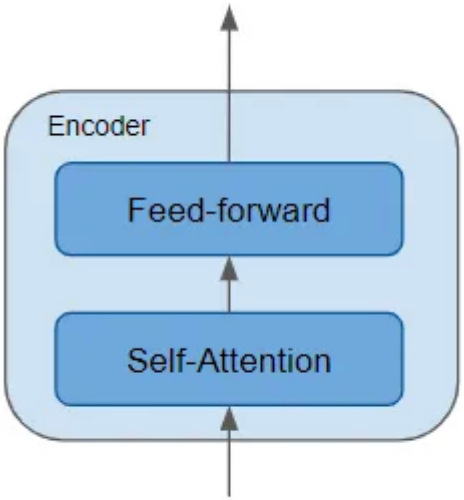
Original Architecture

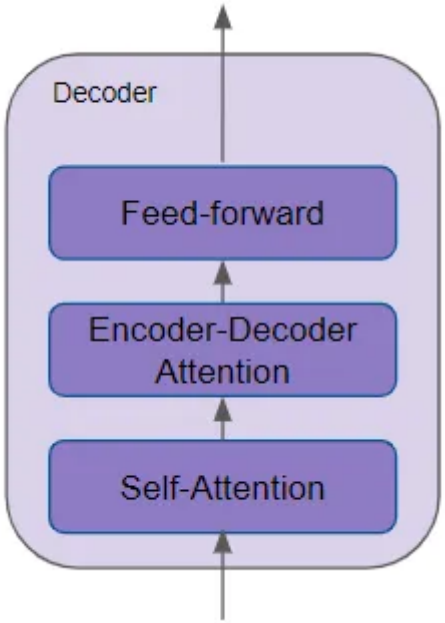
- The Transformer contains a stack of Encoders and a stack of Decoders.
- It contains Input Embedding and Position Encoding layers for both the Encoder and Decoder stacks.
- It contains an Output layer, which generates the final output.



Encoder and Decoder (1/2)

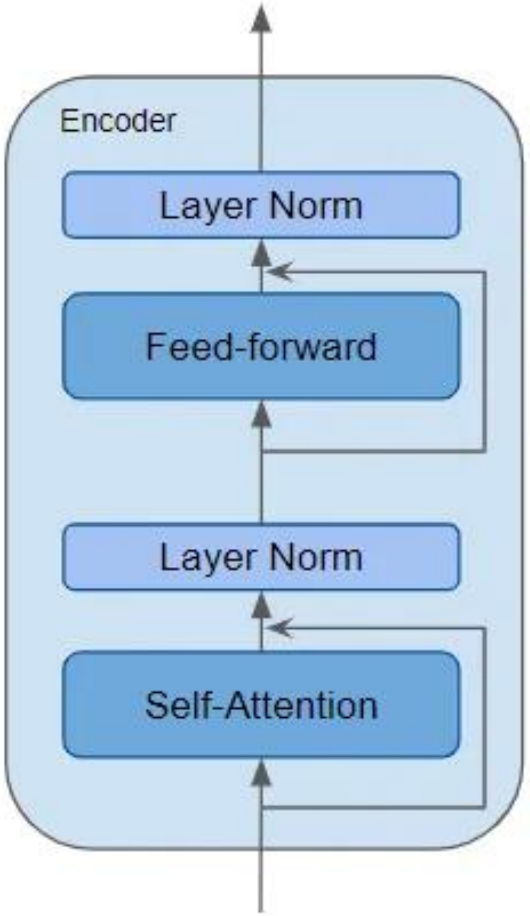
- All the Encoders are identical to one another.
- Similarly, all the Decoders are identical.
- The Encoder contains the Self-Attention layer and a Feed-Forward layer
- The Decoder also contains the Self-Attention layer and the Feed-Forward layer, plus the Encoder-Decoder Attention layer.





Encoder and Decoder (2/2)

- The Encoder and Decoder each also has the following:
- Feed-Forward layer: learns higher-level abstractions of the input text.
- Two Layer Normalization layers: enable smoother gradients by normalizing the distributions of the previous layers.
- Residual skip-connections around the Self-Attention layer and the Feed-Forward layer: ensure proper backpropagation of the gradients.



Attention Layer

- This is the key to the Transformer.
- The Attention mechanism allows LLM to attend to the input text's most relevant parts and generate more accurate predictions.

Roadmap: Architecture -- Overall

- A Quick Look
- Decoder-only Architecture

Three Types of Transformer Architecture

- https://medium.com/@yulemoon/an-in-depth-look-at-the-transformer-based-models-22e5f5d17b6b
- Encoder-only
  - BERT, RoBERTa, distilBERT, distilRoBERTa
- Encoder-Decoder
  - T5
- Decoder-only
  - GPT series, XLNet, LaMDa

Decoder-Only Transformer (1/2)

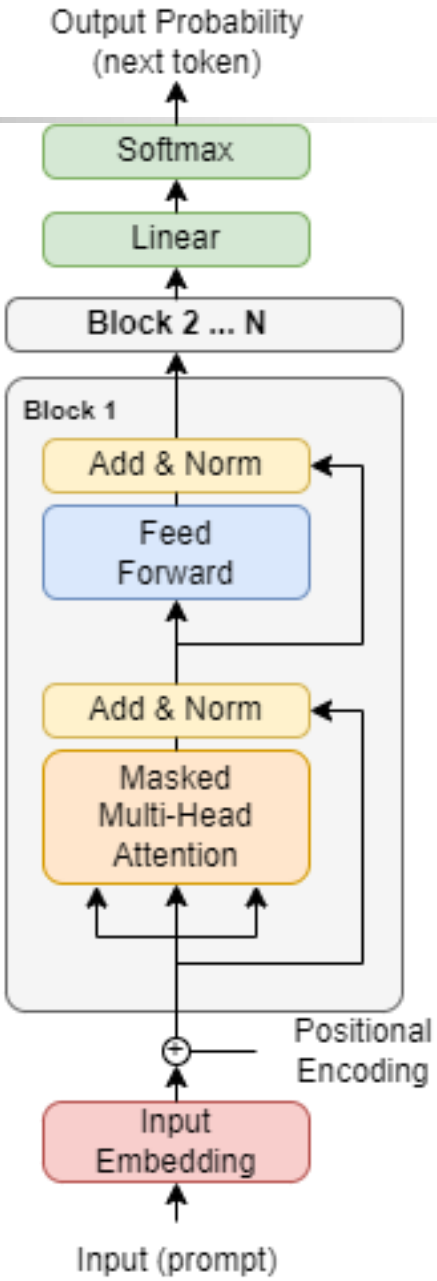
- The original encoder-decoder architecture fits well with its primary application – machine translation.
  - However, it requires a significant amount of task-specific training to fine-tune the model.
- The decoder-only architecture is substantially more efficient to train, and is the dominant form at very large scales.

Decoder-Only Transformer (2/2)

- Input and output of decoder-only Transformers
  - The input is a prompt (often referred to as context) fed into the Transformer.
  - The output depends on the goal of the model. For GPT models, the output is a probability distribution of the next word that comes after the prompt.

Architecture

- Consisting of N decoder blocks (N=12 to 96 for GPT-3)



GPT-3 Attention Dimensions (1/2)

- https://dugas.ch/artificial_curiosity/GPT_architecture.html#:~:text=Of%20course%2C%20the%20embedding%20dimensions,a%2012288%20dimension%20embedding%20vector
- https://lambdalabs.com/blog/demystifying-gpt-3#:~:text=The%20smallest%20GPT%2D3%20model%20(125M)%20has%2012%20attention,with%2096x%20128%2Ddimension%20heads.
- GPT-3 comes in eight sizes, ranging from 125M to 175B parameters.
- The smallest GPT-3 model (125 million parameters) has 12 Attention layers.
- The largest GPT-3 model (175 billion parameters) uses 96 Attention layers.

GPT-3 Attention Dimensions (2/2)

- The input to each Attention head is a (#samples x sequence length x embedding size) shaped matrix.
  - sequence length = 2048
  - embedding size = 12288
- Note:
  - The result of each Attention head is a single 2048 x 128 matrix.
  - The results of the 96 Attention heads are concatenated together, yielding a 2048 x 12288 matrix.
    - 96 x 128 = 12288 = embedding size
- This result is then multiplied with a linear projection (which does not change the matrix shape).

Roadmap: Tutorial

- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

Roadmap: Architecture -- Attention

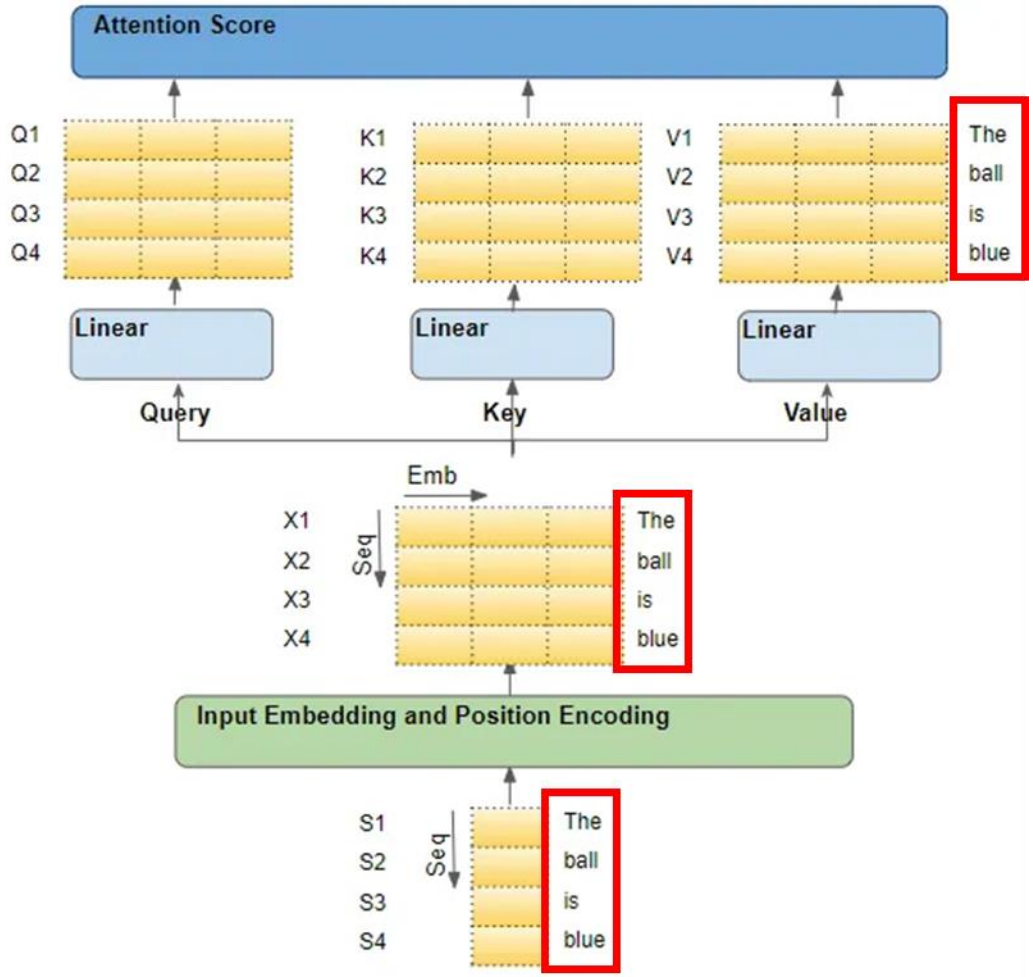
- Overview
- Intuition
- Attention Score Calculation
- Matrix Flow

Architecture of the Attention Module

- The Attention module consists of three linear layers.
- The three layers produce three separate matrices — known as the Query, Key, and Value matrices.
- The three matrices are used to compute the attention scores.
- Each 'row' of the matrices corresponds to one word (token) in the input sequence.
  - The Query word can be interpreted as the word for which we are calculating attention scores.
  - The Key and Value word is the word to which we are paying attention; i.e., how relevant that word is to the Query word.

Visualization of the Three Linear Layers

- Input sequence: "The ball is blue"
- * Note: The 3 matrices have the same input sequence.



Query, Key, Value Matrices

- The Attention module has 3 matrix pairs
  - Query matrix Q and query weight matrix (W_Q)
  - Key matrix K and key weight matrix (W_K)
  - Value matrix V and value weight matrix (W_V)
- At start of training, all weights are random. After many backpropagations, the weights become better.

Roadmap: Architecture -- Attention

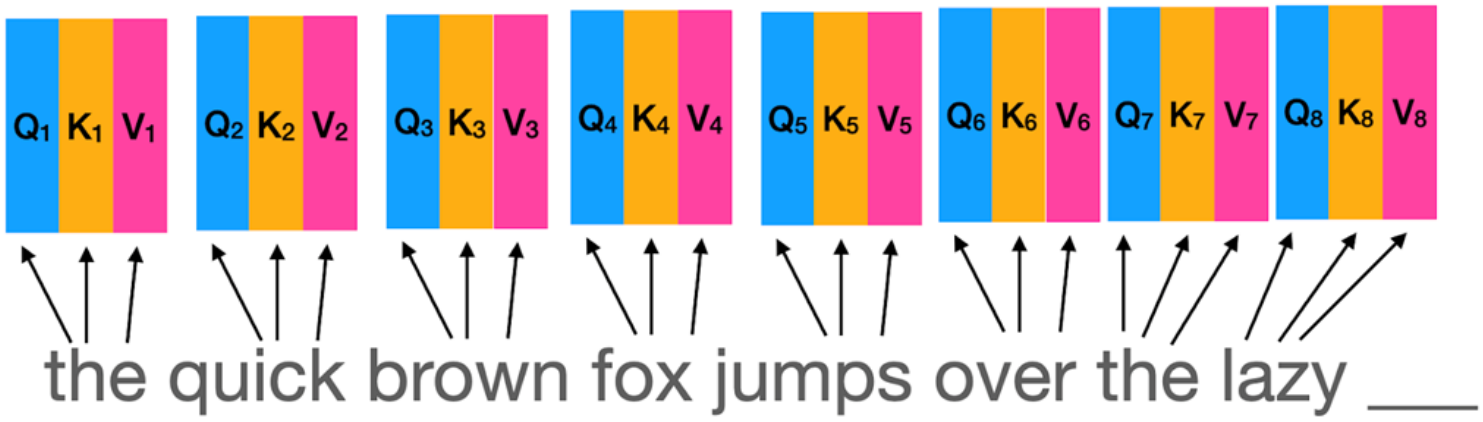
- Overview
- Intuition
- Attention Score Calculation
- Matrix Flow

Query, Key, Value: Think of Querying a Key-Value Database

- https://medium.com/mlearning-ai/demystifying-the-attention-logic-of-transformers-unraveling-the-intuition-and-implementation-fa6e3cf05a79
- Retrieving the value from a key-value pairs database
  - The user issues a query (search key, which is one of the stored keys).
  - The database searches all the stored keys, calculating similarities between the query and keys of the database.
  - The database returns the values of the key most similar to the query.

Analogy to Broadcasting a Question and Answer (1/2)

- https://learnopencv.com/attention-mechanism-in-transformer-neural-networks/
- Suppose the input sequence is "the quick brown fox jumps over the lazy"
- Each word is represented as a query Q, key K, and value V vector.



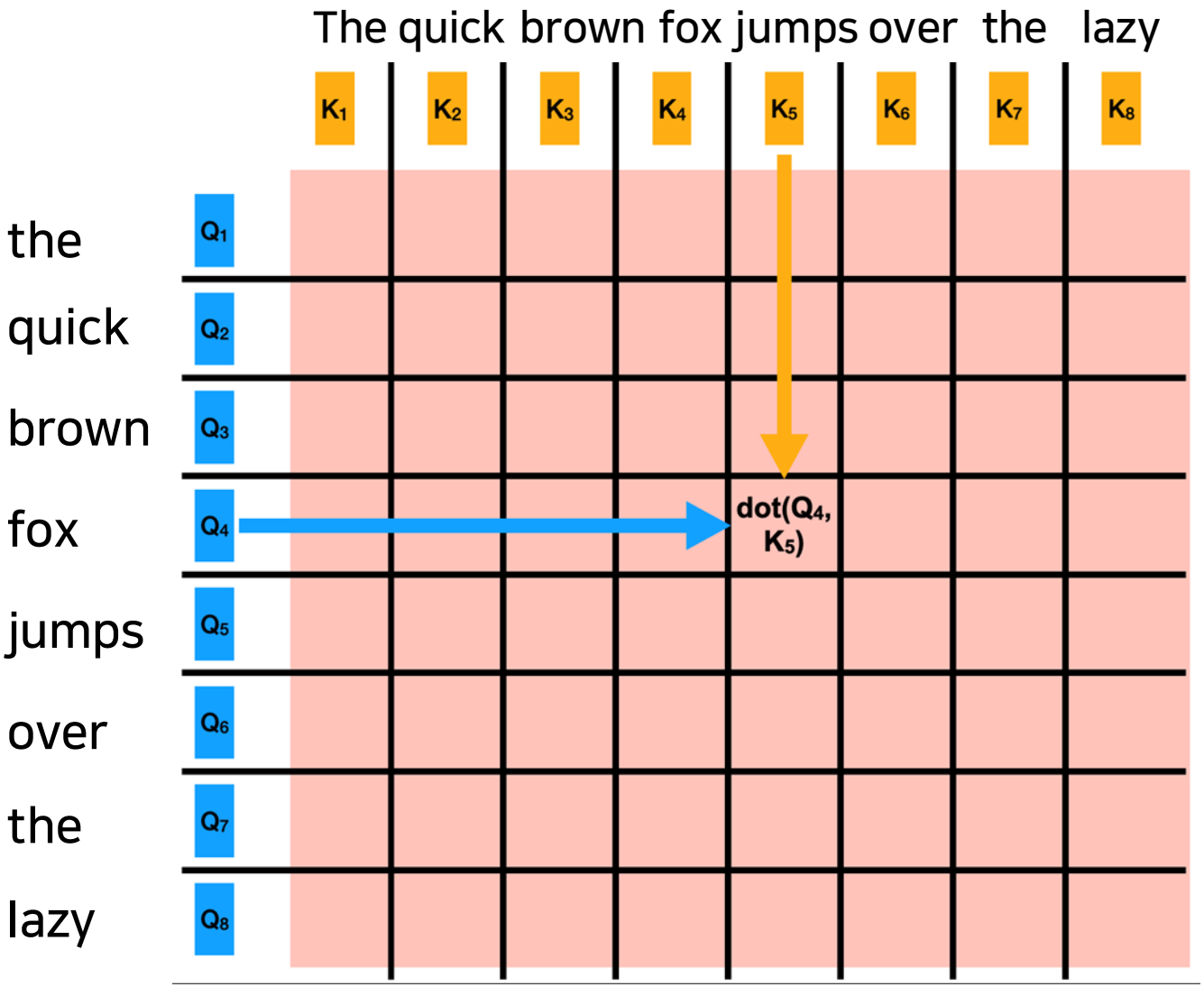
Analogy to Broadcasting a Question and Answer (2/2)

- To model relevance of words in a sequence, we let each word ask questions and receive answers.
- A word (e.g., "fox") broadcasts the same question to ALL words in the sequence using the 'Query' vector.
- Similarly, a word broadcasts the same answer to all Query words using the 'Key' vector.
- A separate 'Value' vector is used to allow the model to combine the outputs of Query and Key vectors non-linearly.

Importance Matrix

- If we have a sequence of n words, we will have *n* Query, *n* Key and *n* Value vectors.
- How does a word broadcast questions and answers?
- We create an nxn Importance Matrix of n Query vectors and n Key vectors.
- We take the dot product of all the *n* Query vectors with all the *n* Key vectors.
- Each element of the matrix is the dot product of the i^{th} Query vector with the j^{th} Key vector. (shown in the next page).

Visualization of the Importance Matrix

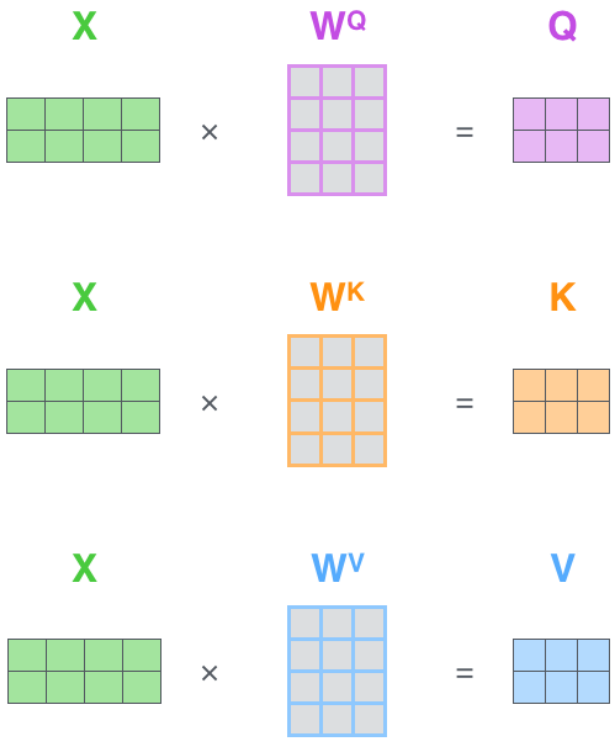


Roadmap: Architecture -- Attention

- Overview
- Intuition
- Attention Score Calculation
- Matrix Flow

Calculating Attention Scores (1/2)

- Pack the input tokens into a matrix X
  - (note: The figure shows 2 tokens, with embedding size 4.)
- Calculate the Query, Key, and Value matrices by multiplying X by the weight matrices W_Q, W_K, W_V.



Calculating Attention Scores (2/2)

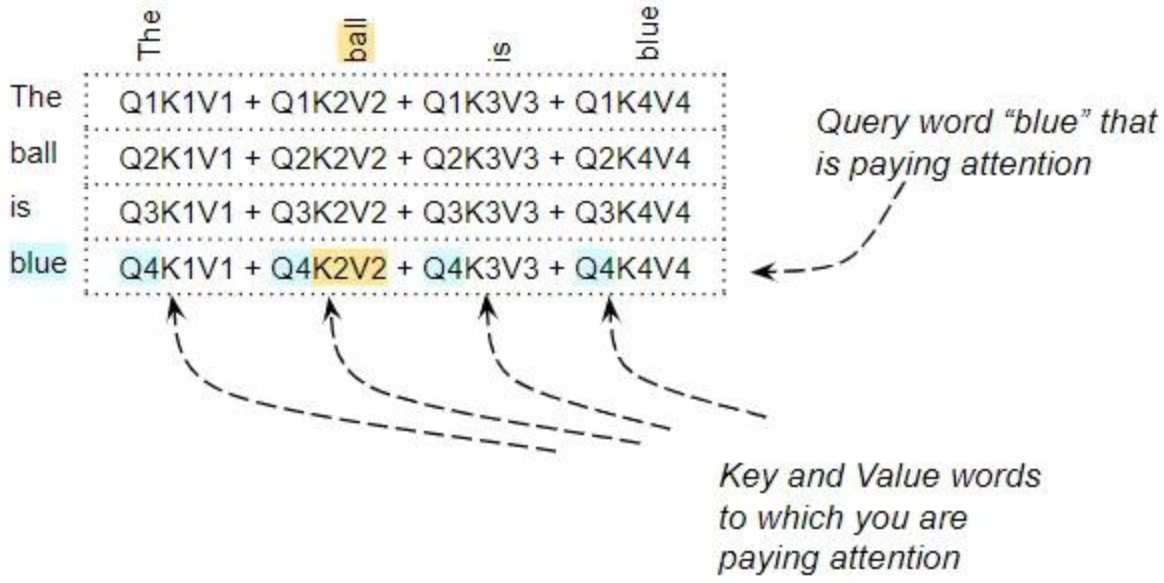
- The attention score is calculated by implementing the following formula.

$$Z = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where Q is the Query matrix, K is the Key matrix,
V is the Value matrix; and
d_k is the dimension of k
(k is embedding size ÷ number of heads)

Meaning of the Calculation

- For example, for the sentence "The ball is blue",
  - the row for the word blue will contain the attention scores for blue with every other word,
  - while blue is the Query word, and the other words are the "Key·Value" words.



Multi-Head Attention

- Multi-Head Attention has multiple Attention heads working in parallel.
- The Linear layer weights are partitioned uniformly across the Attention heads.
  - For expository purpose, we will assume the embedding size= 512, and # of heads=8. Then each head will take a (512 ÷ 8 =) 64-dimension embedding.
- This improves the performance of the Attention layer
  - because each head can learn different patterns.
- Each of the matrices in each head is randomly initialized.

Roadmap: Architecture -- Attention

- Overview
- Intuition
- Attention Score Calculation
- Matrix Flow

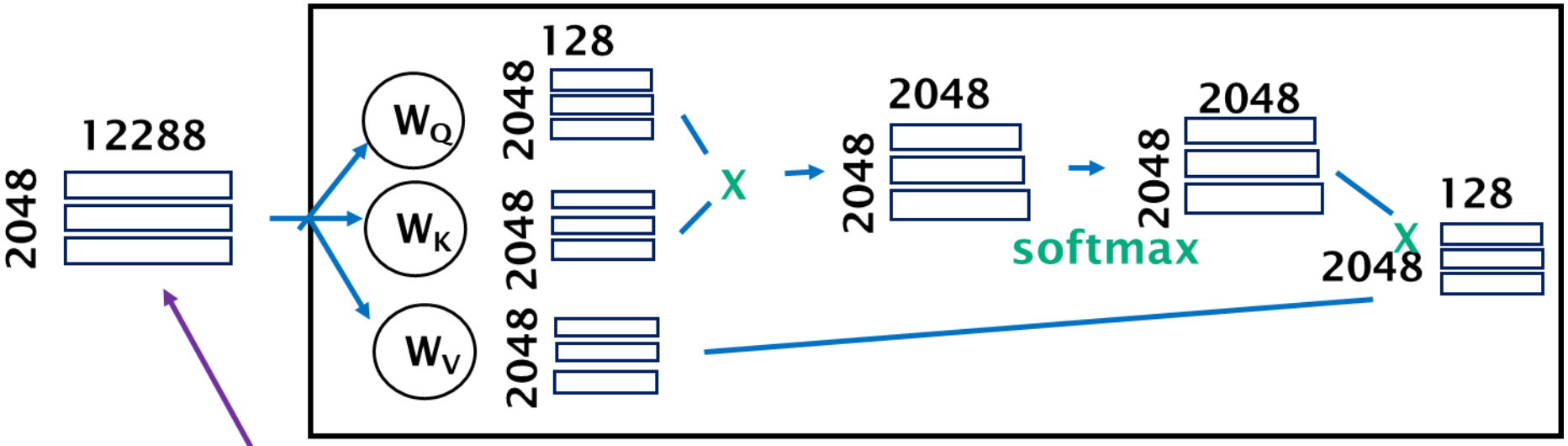
GPT Series Architecture Comparison

- https://360digitmg.com/blog/types-of-gpt-in-artificial-intelligence

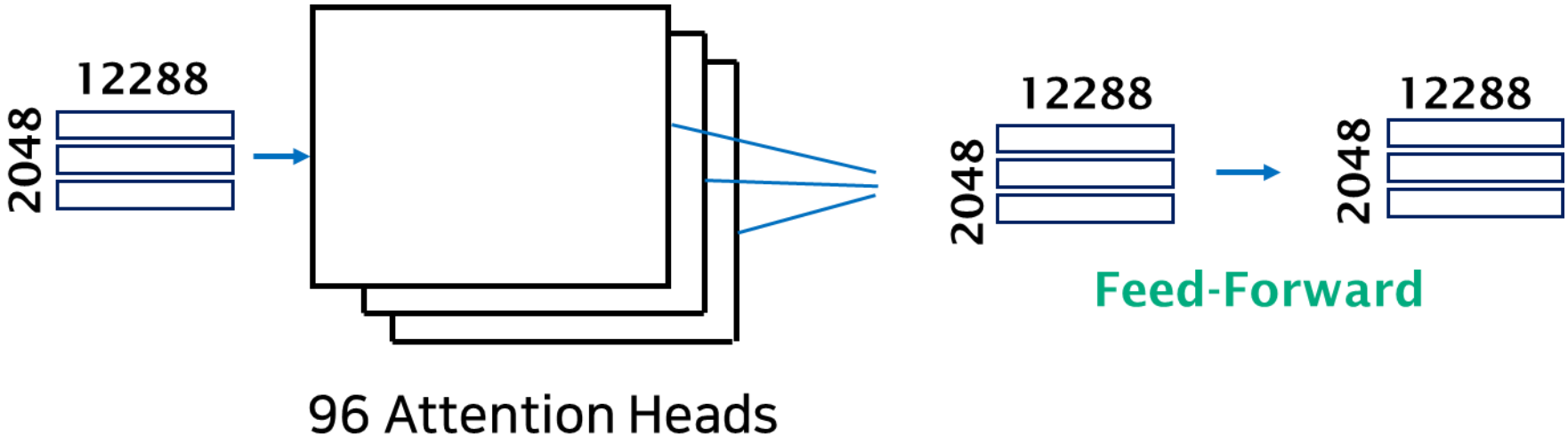
	GPT-1	GPT-2	GPT-3
Parameters	117 Million	1.5 Billion	175 Billion
Decoder Layers	12	48	96
Context Window Size	512	1024	2048
Hidden Layer	768	1600	12288
Batch Size	64	512	3.2M

Matrix Flow Through the GPT-3 Decoder (1/3)

- https://dugas.ch/artificial_curiosity/GPT_architecture.html#:~:text=Of%20course%2C%20the%20embedding%20dimensions,a%2012288%20dimension%20embedding%20vector
- *Note: For simplicity, the batch size dimension is omitted from the figure. (For GPT-3, the batch size is 3.2 million.)

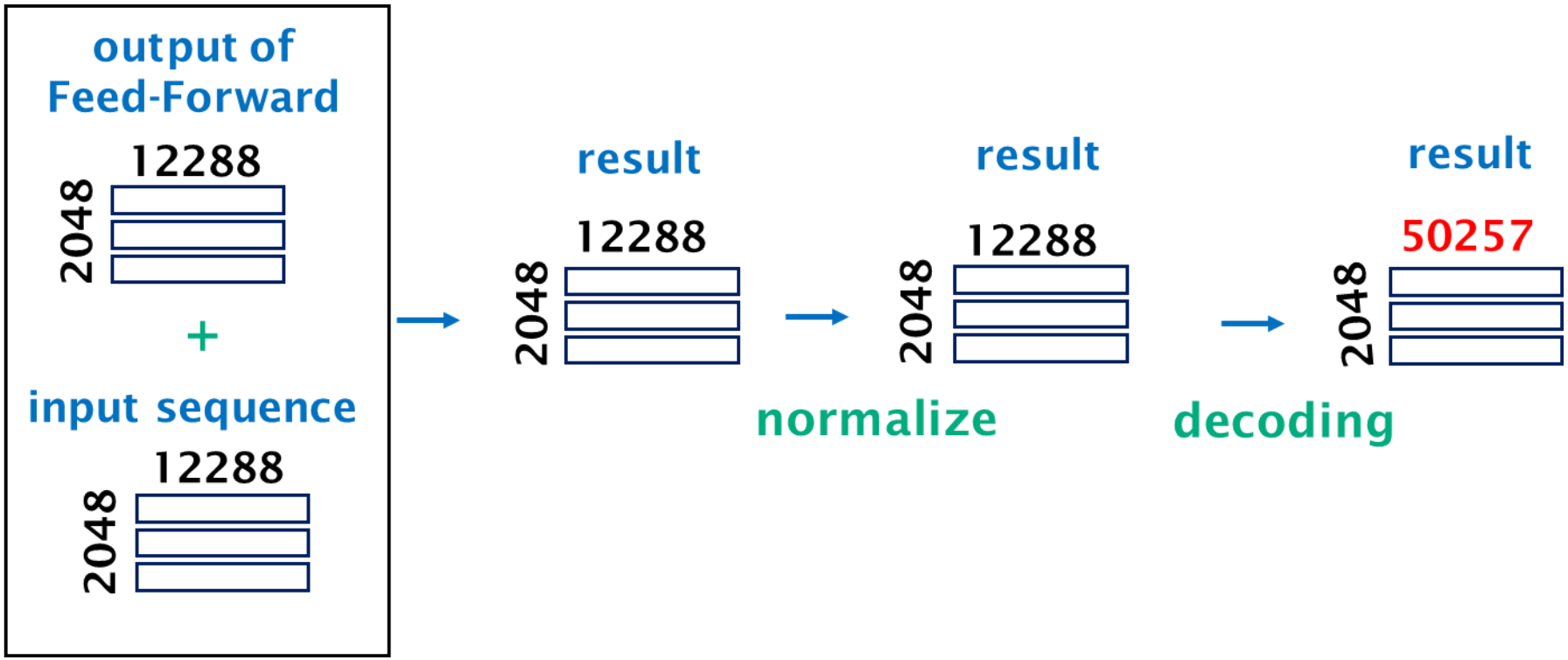


Matrix Flow Through the GPT-3 Decoder (2/3)



Matrix Flow Through the GPT-3 Decoder (3/3)

- * Note: For simplicity, the normalization step is shown just once.



Roadmap: Tutorial

- Introduction
- Training and Inference
- Performance
- Tokenization
- Input Embedding and Position Encoding
- Attention Concept
- Architecture: Overall
- Architecture: Attention
- Prognosis and Challenges

Prognosis: Positive

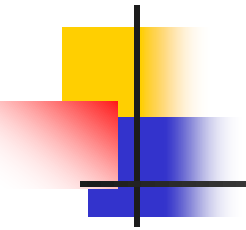
- LLMs show possibilities for adoption across many sectors of the industry, education, economy and society.
- LLMs hold much of the world's knowledge, and additional knowledge can be added to it.
- LLMs can interact with people not only in natural languages, but also programming languages.
- However, everything LLMs generate must be fact-checked.

Prognosis: Negative

- LLMs are platforms (like OS, DBMS), and there will be the inevitable shakeout in a few years.
- Many tech companies have entered into the fray, but it appears that not many are profitable.
- The challenge is to establish areas within businesses that benefit significantly from LLMs.

R&D Opportunities

- Every topic examined in this Tutorial and more.
- Training, finetuning
  - improve output quality, throughput
  - reduce required computing resources
  - improve benchmarks
- Prompt engineering
- Hallucination
- Improve Embedding, Position Encoding, Tokenization, Attention
- Making sLLMs competitive to LLMs
- Build tools and applications for wide adoption
- Combat the Dark Side of LLM



Challenges

- There are serious concerns about how LLMs are created, used, and abused.
- Three types of issues
  - Issues inherent in LLMs
  - Issues with the use/misuse of LLMs
  - Issues for the LLM developers

Issues Inherent in LLMs

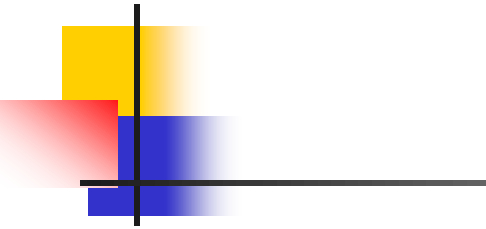
- Hallucination (and spread of wrong information)
- Violation of privacy
- Violation of intellectual property rights
- Toxic and biased results due to training data
- Temporal shifting (trained data becoming obsolete)
- Specialization & diversity issues
  - Models trained in specialized domains may not generalize to new domains.
  - Models trained in a diverse range of domains may not perform as well in specific areas.

Issues with the Use/Misuse of LLMs

- Users blindly trusting the outputs of LLMs
- Criminal uses
  - phishing, hacking, malware dissemination
  - jail-breaking GPT, WormGPT, FraudGPT
- Plagiarism
- Consequences of integrating with APIs and other software tools
- Job loss (?)
  - StackOverflow layoffs, Hollywood writers' strike,…
  - may be offset by job creation
  - may be just a historical inevitability

Issues for the LMM Developers

- Indecipherable results (non-explainable AI)
  - There is a lack of understanding of how these models work, why it exhibits certain behaviors, and how it reaches specific conclusions.
  - This lack of understanding makes it difficult to troubleshoot issues or make improvements to the models.
- Heavy use of resources
  - Developers need large, high-quality datasets and powerful computing resources to train and fine-tune the models effectively.
  - Human evaluation necessary for fine-tuning, which can be time-consuming and expensive.



Closing

NVidia GPUs

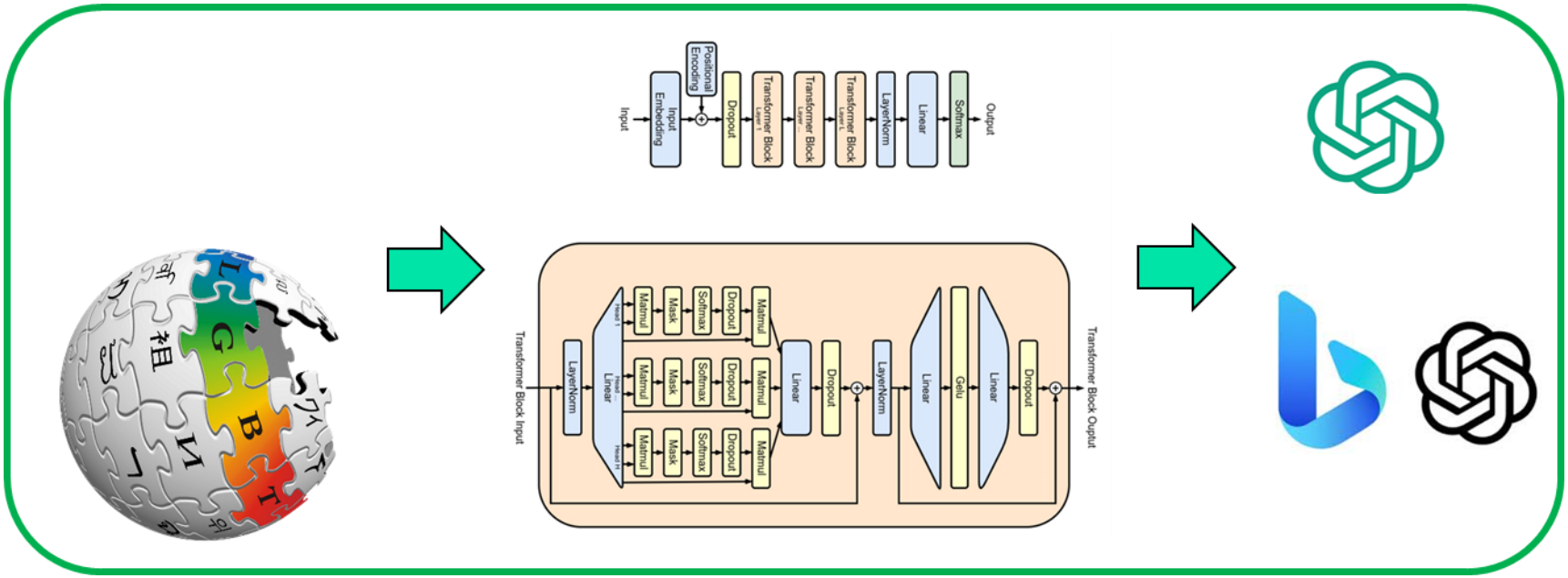
	Titan V (V100)	GeForce RTX 4090
Tensor Cores	640	512
Tera FLOPS	110	83

GPT-3/GPT-4 (and Comparable LLMs) =

Data and Electricity-Eating Monster Tamed by a Massive Herd of Monsters named NVidia GPUs

The Monster Kraken
(in the Movie Clash of the Titans)

Thank You


Input
Input Embedding
Positional Encoding
Dropout
Transformer Block
Transformer Block
Transformer Block
LayerNorm
Linear
Softmax
Output
Transformer Block Input
LayerNorm
Linear
Matmul
Mask
Softmax
Dropout
Matmul
Linear
Dropout
LayerNorm
Linear
Gelu
Linear
Dropout
Transformer Block Output