Large Language Models

An introduction

Jeremie Clos

Cyber-physical Health and Assistive Robotics Technologies Research Group (https://www.chartresearch.org)
Mixed Reality Lab
Brain and Physiological Data Group (https://brain-data-uon.gitlab.io)

- **COMP3074** Human-AI Interaction
  - With Prof. Joel Fischer
  - Human-Computer Interaction, Natural Language Processing
- **COMP4030** Data Science and Machine Learning
  - With Prof. Praminda Caleb-Solly
  - Data Science, Machine Learning

Graphical outline

0. Introduction
- Natural Language Processing
- Language Models, Large Language Models

1. Representing text
- Fixed-length/variable-length representations
- One-hot, word embeddings

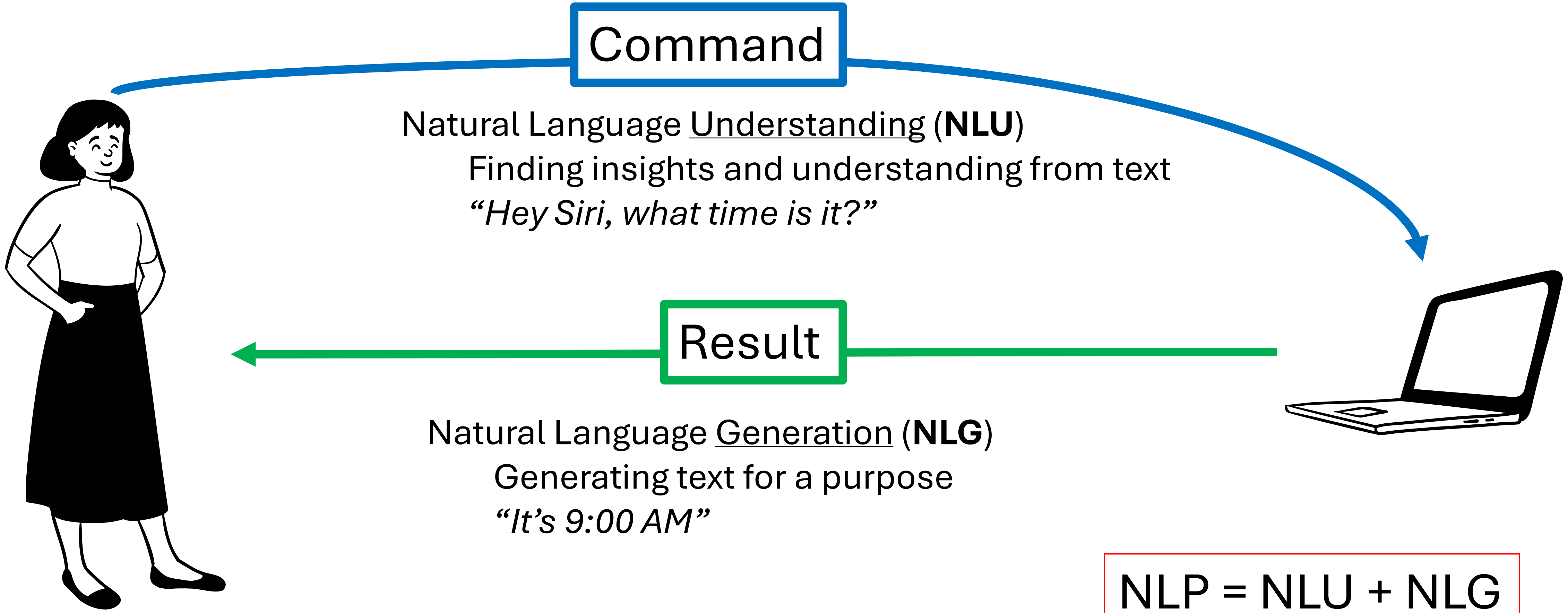
2. Modelling text
- Simple perceptrons
- Elman networks, Jordan networks
- LSTM
- Transformers

3. Attention
- Self attention
- Masked attention
- Multi-head attention

4. Large Language Models
- Generative Pre-Training

Natural Language Processing

- Language is the basis of interaction



Natural Language Understanding (**NLU**)

Finding insights and understanding from text

“Hey Siri, what time is it?”

Natural Language Generation (**NLG**)

Generating text for a purpose

“It’s 9:00 AM”

NLP = NLU + NLG

Language Modelling and Natural Language Processing

- You are building a voice-powered personal assistant. The assistant hears something ambiguous. Which sentence is the most likely?

Sentence 1: I want a **caramel iced** latte.

Sentence 2: I want a **caramelised** latte.

How do we measure that?

Probabilistic language models

- Core problem of language modelling:

  Given a sequence of tokens, predict the next token
- P(I want a **caramel iced** latte) vs P(I want a **caramelised** latte)
- Let's recall some Probability
  - $P(B|A) = \frac{P(A,B)}{P(A)}$ or stated differently $P(A,B) = P(A) \times P(B|A)$
- What does it mean for our sentence?
  - $P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1, w_2) \times \cdots \times P(w_n|w_1, w_2, \ldots, w_{n-1})$
    $$= \prod_i P(w_i|w_1 w_2 \ldots w_{i-1})$$
  - $P(I) \times P(want|I) \times P(a|I, want) \times P(caramel|I, like, a) \times \cdots$

Probabilistic language models

- The simple n-gram language model
  - Assigns probabilities to sequences of words
  - Generate text by **sampling** possible next words
  - Trained on counts computed from lots of text
- **Large Language Models (LLMs)** are similar:
  - Assigns probabilities to sequences of words
  - Generate text by sampling possible next words
  - Are trained by learning to guess the next word

Neural language models

- Self-supervised learners
  - Take a text, remove a word
  - Use your model to guess what the word was
  - If the model is wrong, use stochastic gradient descent to make the model guess better next time
- Advantages:
  - All we need is a **lot of** text (GPT3: 500 billion tokens)

How Do LLMs Work?

- At their most basic LLMs are statistical prediction systems
- LLMs output the next likely word (“token”) in a sentence (“sequence”)
    - Token: unit of text
    - Sequence: section (“window”) of text e.g. sentence, paragraph, book
- The likelihood of the next work appearing is determined by the context in which the words are seen in the corpus

LLMs “Understand”[1] “Meaning”[2]

- Learning from a large corpus allows LLMs to *understand* the meaning of words.

For example

  - the training data may consist of many sentences beginning with “my favourite colour is…”
  - the next word will be a colour, allowing LLMs to (implicitly) cluster the words “red, blue, green…” into a set that represents the concept of “colour”

- LLMs do not really understand anything in an epistemic sense, only in a statistical sense.

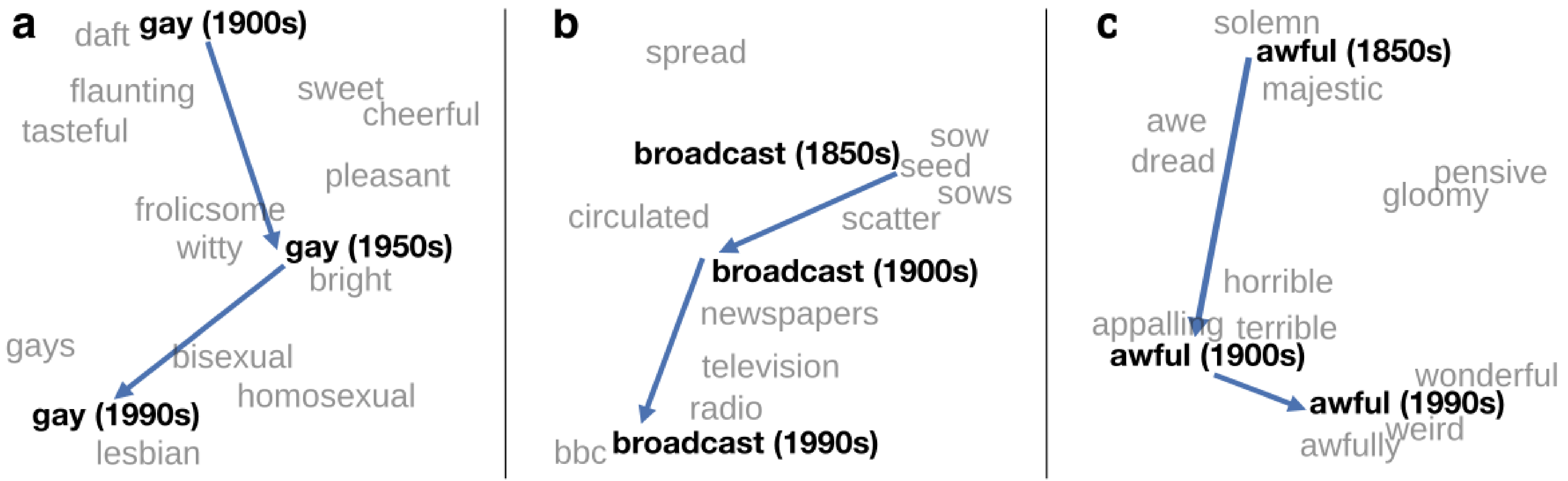
[1] for some definition of understand
[2] for some definition of meaning

Data used to Train LLMs

- LLMs are trained in an unsupervised manner on open source, licensed, and “““borrowed””” data, e.g.
  - The Pile (825GB, including web, papers, books, computer code, …)
  - Common Crawl (~20B URLs)
  - GPT3: 175B parameters
  - GPT4: undisclosed, but probably a lot more
- Responses are refined using Q/A pairs (e.g., “InstructGPT”)
- Reinforcement learning with human feedback (RLHF) is used to reward LLMs to give appropriate responses (“guardrails”)

Meanings can change based on context

- This is why "machine unlearning" is becoming a hot topic
  - Some word meanings change
  - Some things should be forgotten



LLMs can (seem to) be creative[1]

Context-based learning + randomness allow the LLMs to generate surprising outputs

We can use LLMs to:

- Identify weakly similar concepts from different disciplines and help understand different disciplines
- Generate narratives
- Reformulate text at different reading levels
- Role playing
- And more!

- Research projects I am involved in:
  - LLMs and legal text
  - LLMs and therapy
  - LLMs and robots

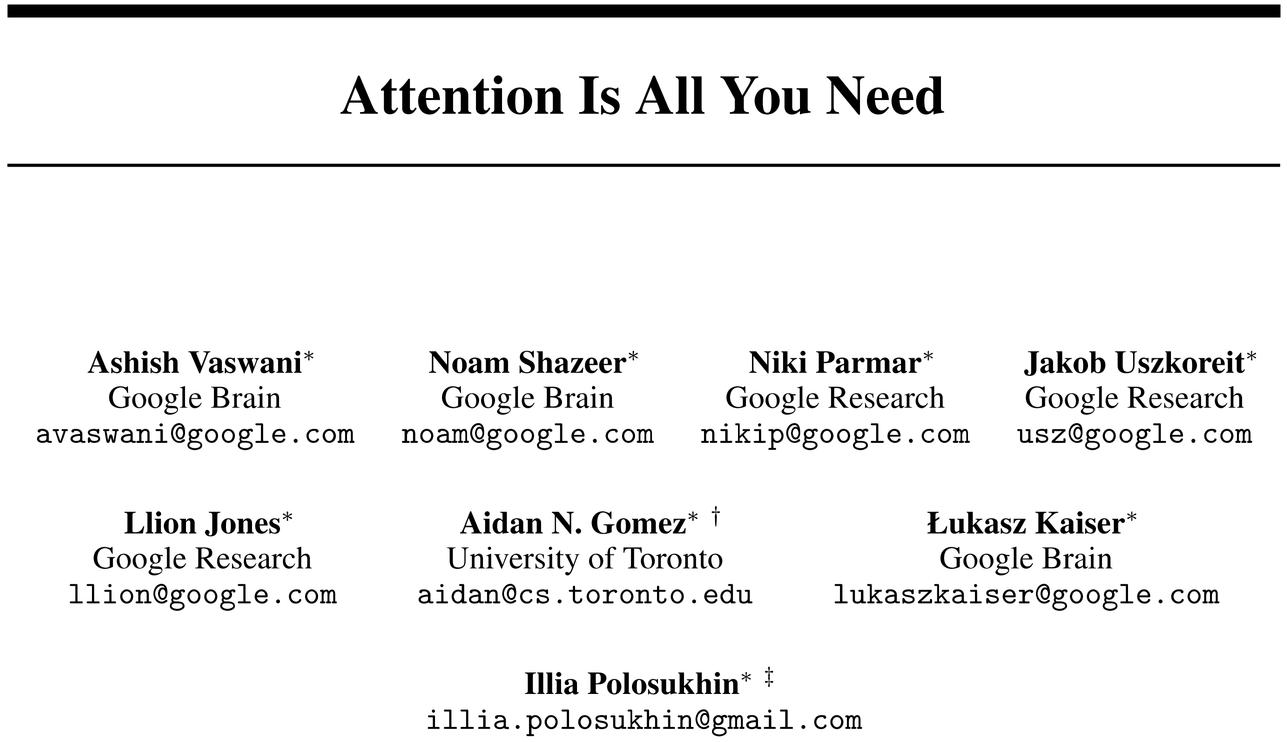
[1] for some definition of creative

Challenges in LLMs

- LLMs may seem to “lie” and “hallucinate” i.e. give factually-incorrect responses to questions
- This is some function of data, learning, search and probability
- Remember that the output of a LLM is determined by both what the system has been trained on and what information you give it
- **Prompt engineering** means tailoring your questions and input so you can get the most out of an LLM
- Prompts can take many forms, from instructing the LLM to take on a role (e.g. a helpful teacher, a pirate) or guiding the way it should process its output (e.g. “chain of thought” or a particular method).

What is inside of a LLM?

- Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention



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

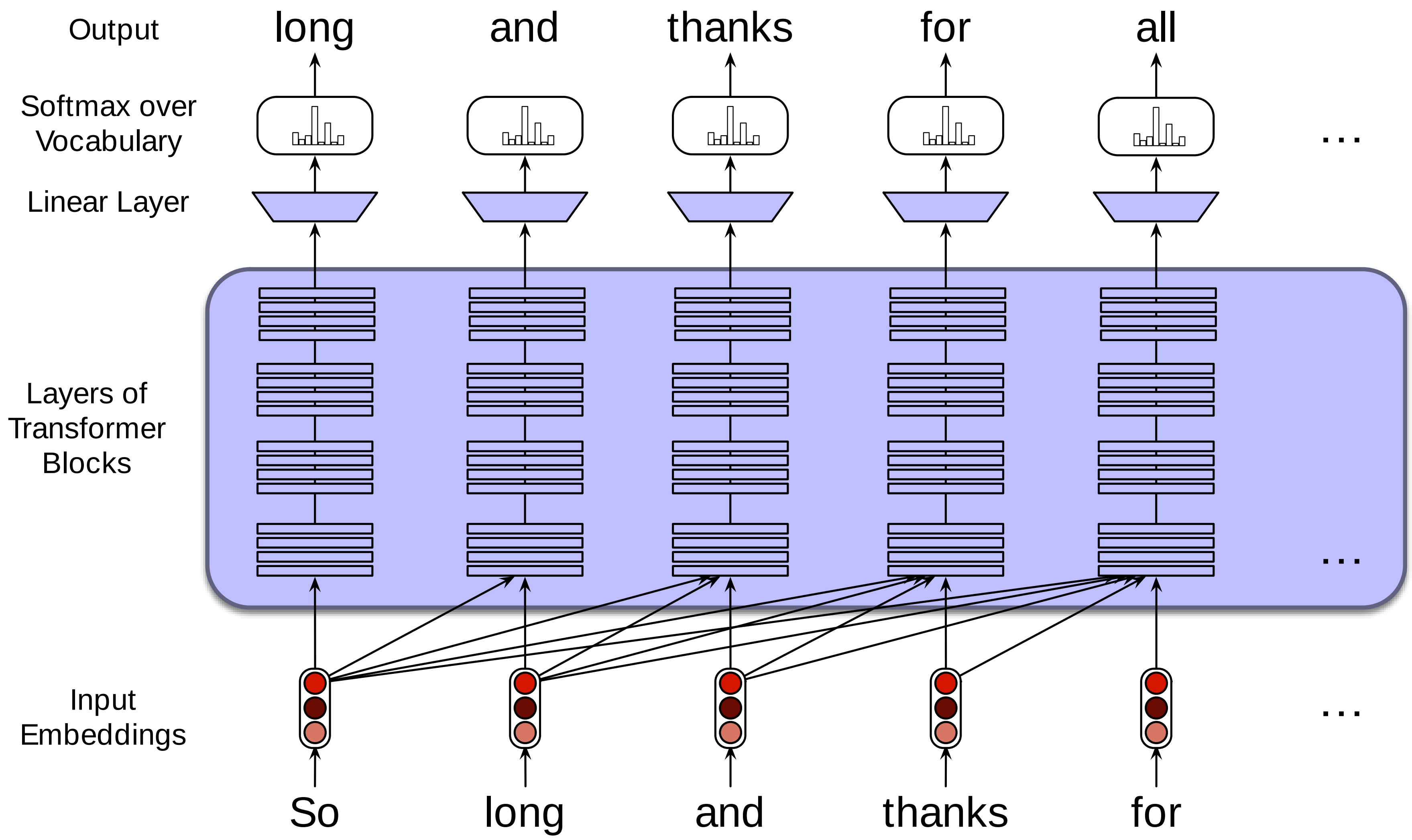
Llion Jones[*]
Google Research
llion@google.com

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

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

A transformer language model



Word embeddings

- Example:
  - d$_1$: “The red cat”
  - d$_2$: “The red dog”
  - d$_3$: “Cat, red cat and dog”
  - Vocabulary: the, red, cat, dog, and
  - Embedding dimension: 3

$$d_1 = \begin{bmatrix} 0.04551 & 1.22451 & 2.91134 \\ -0.91345 & 1.94851 & 0.41140 \\ 1.41251 & -3.14014 & 1.45951 \end{bmatrix}$$

$$d_2 = \begin{bmatrix} 0.04551 & 1.22451 & 9.91134 \\ -0.91345 & 1.94851 & 1.41140 \\ 1.41251 & -3.14014 & 3.45951 \end{bmatrix}$$

$$d_3 = \begin{bmatrix} 2.91134 & 1.22451 & 2.91134 & 5.47311 & 9.91134 \\ 0.41140 & 1.94851 & 0.41140 & -1.46841 & 1.41140 \\ 1.45951 & -3.14014 & 1.45951 & 3.89421 & 3.45951 \end{bmatrix}$$

- Most word embeddings encode some notion of similarity
  - Cosine similarity of word vectors is meaningful
- That similarity is extracted from text, so don’t think for one second it is objective
  - Racist, sexist text = racist, sexist embeddings
- As many dimensions as you need (e.g., GPT uses 12,288 dimensions)

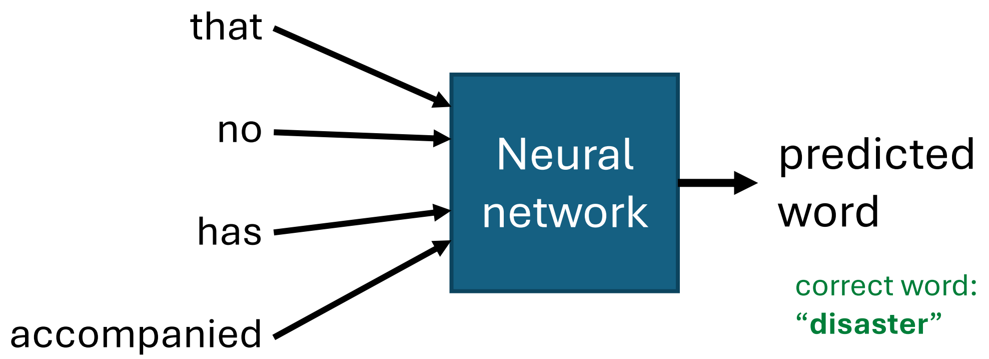
Word embeddings

- A word embedding is a dense vector representation of words
- Word embeddings preserve some properties of words
  - Let $v(\mathrm{x})$ be the embedding vector of word x
  - **Similarity**: $\mathrm{Similarity}\big(v(\mathrm{dog}), v(\mathrm{cat})\big) > \mathrm{Similarity}(v(\mathrm{dog}), v(\mathrm{car}))$
  - **Analogy**: $v(\mathrm{king}) - v(\mathrm{man}) + v(\mathrm{woman}) \approx v(\mathrm{queen})$
- The properties depend on how the embeddings were computed
- Until 2005, mostly latent semantic analysis (old school stuff)
- 2005-2012 neural probabilistic language models (Bengio et al.)
- 2013 Word2Vec (Mikolov et al.), 2014 GloVe (Manning et al.)

Word embeddings

- Word embeddings are old, but Word2Vec revived them
- Continuous Bag-of-Words model (CBOW)
  - Initialise a fully-connected, standard neural network
  - Task of the network:
    - Travel through a very large document word by word
    - Train the network to predict words based on their context
    - Example, window of 5 words

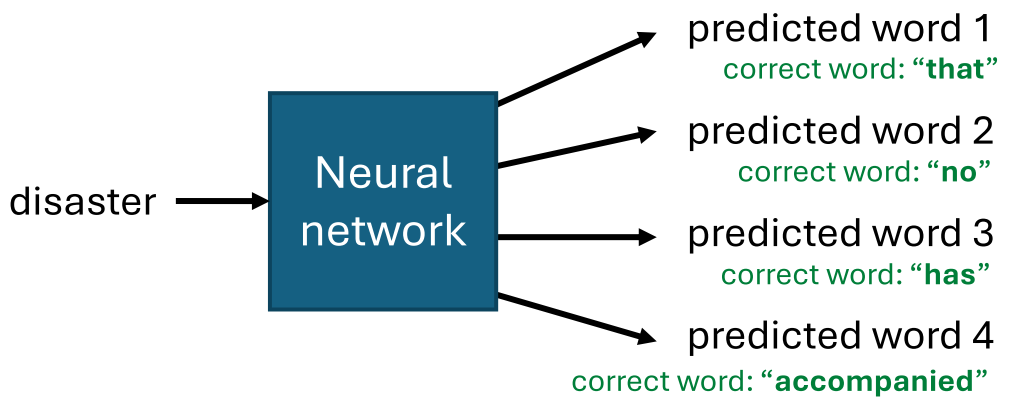
You will rejoice to hear that no disaster has accompanied the commencement of an enterprise which you have regarded with such evil forebodings. I arrived here yesterday, and my first task is to assure my dear sister of my welfare and increasing confidence in the success of my undertaking.



Word embeddings

- Skip-gram does the opposite – try to predict the context from the pivot word

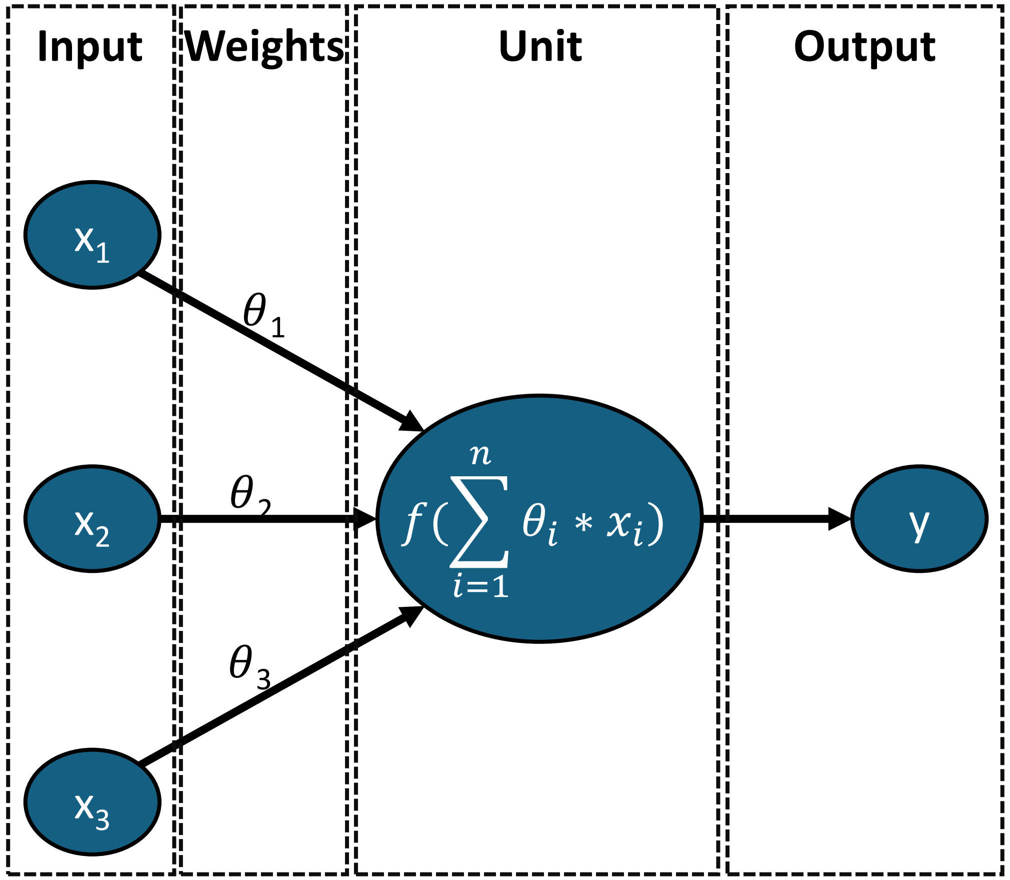
You will rejoice to hear that no disaster has accompanied the commencement of an enterprise which you have regarded with such evil forebodings. I arrived here yesterday, and my first task is to assure my dear sister of my welfare and increasing confidence in the success of my undertaking.



- Word embeddings are extracted from the weights of the network
- Because they are trained on context, words which can be replaced by one another end up with similar embedding vectors

Neural networks

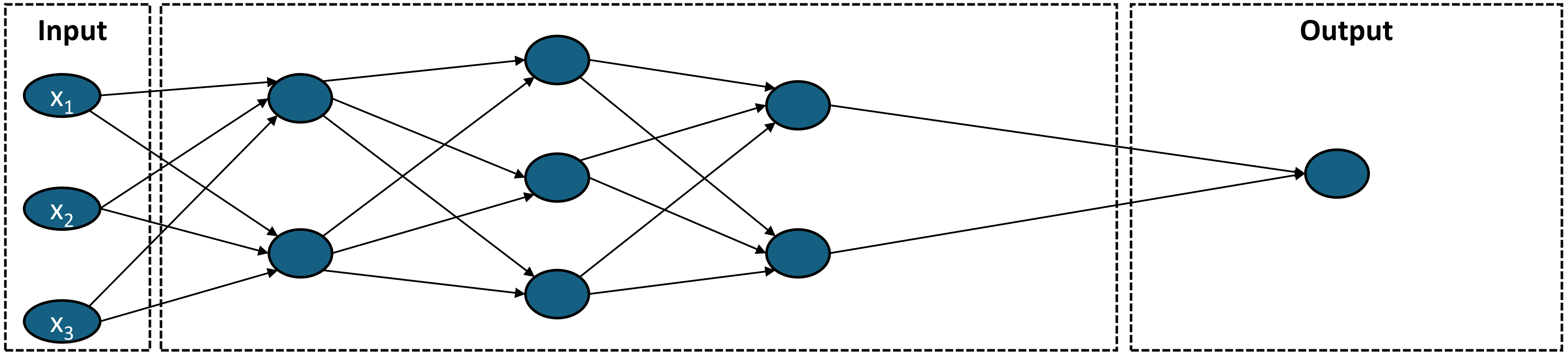
- Neurons are small **logistic regression** models chained together.



- Function f introduces **non-linearity**
- Typical functions:
  - Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
  - Hyperbolic tangent: $f(x) = \tanh(x)$
  - ReLU: $f(x) = \max(0, x)$
  - GELU: $f(x) \cong 0.5x(1 + \tanh[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)])$

Neural networks

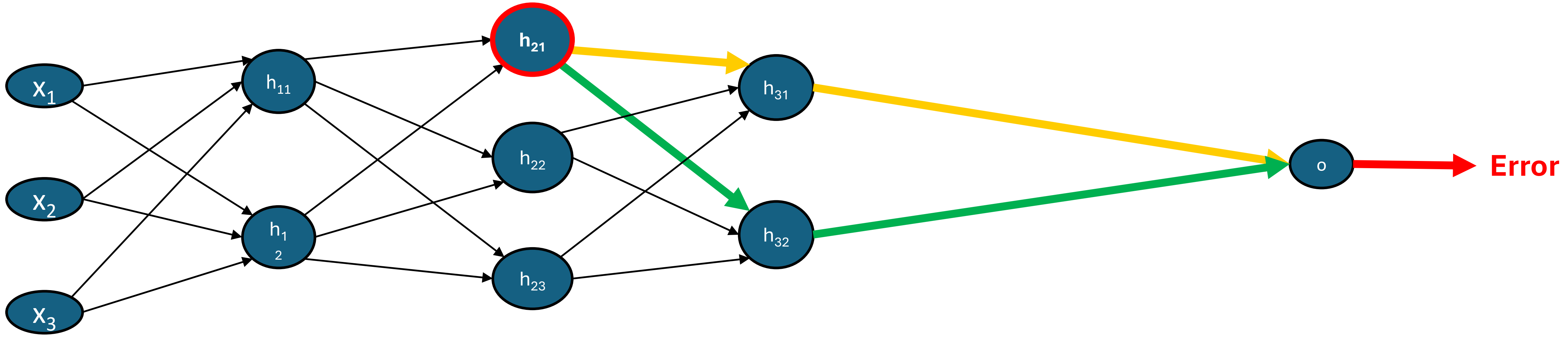
- Forward pass:
  - Output of intermediate neurons fed to the next layer
- Backward pass:
  - The partial derivative targets each neuron with its specific contribution to the prediction
  - Neurons can contribute to the prediction **via multiple paths** (fully connected network)



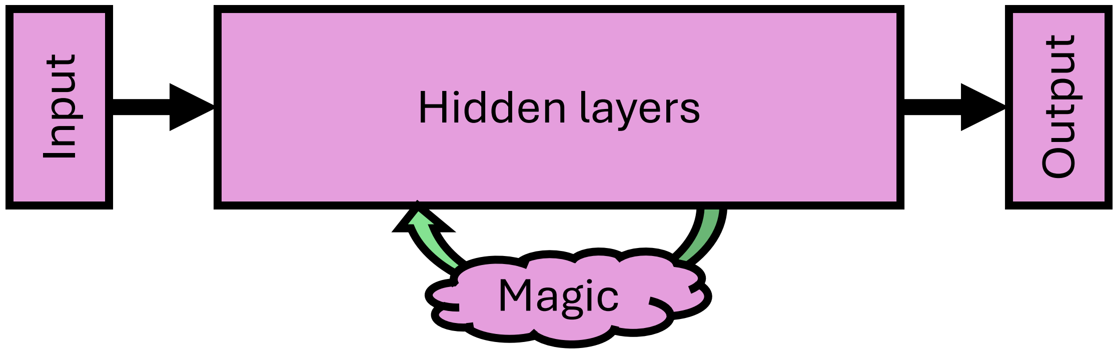
More on the backward pass

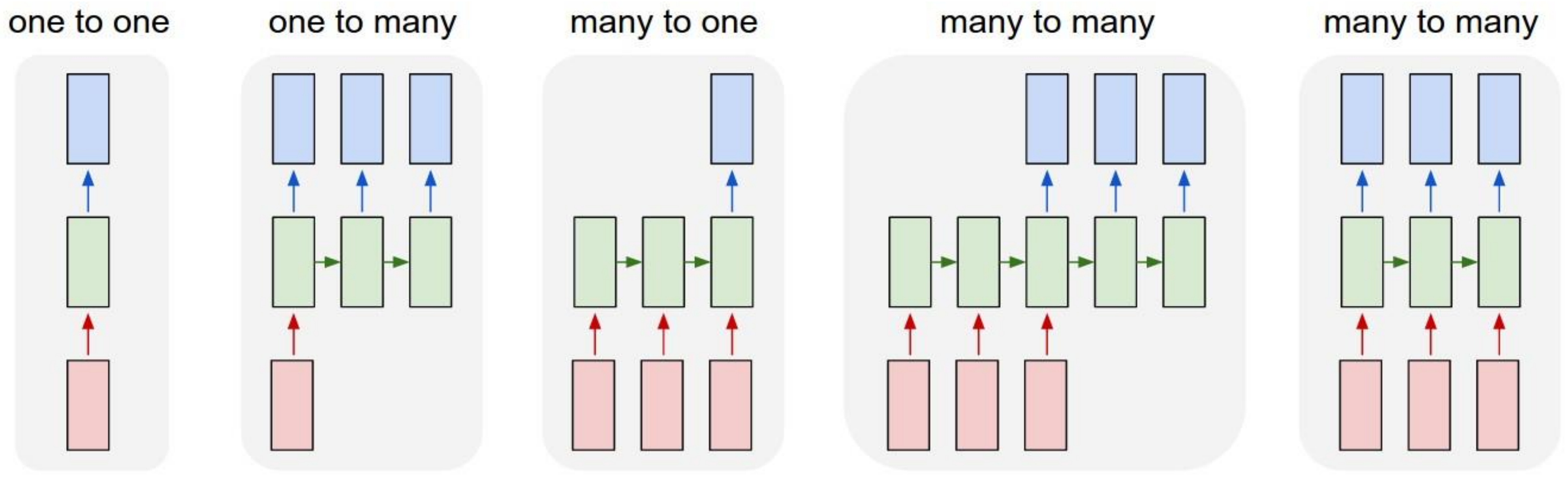
- Use the chain rule to propagate the gradient to the hidden layers

$$h_{21} \leftarrow h_{21} - \alpha \cdot \left(\frac{\partial J}{\partial o} \cdot \left(\frac{\partial o}{\partial h_{31}} \cdot \frac{\partial h_{31}}{\partial h_{21}} + \frac{\partial o}{\partial h_{32}} \cdot \frac{\partial h_{32}}{\partial h_{21}} \right) \right)$$



Recurrent neural networks

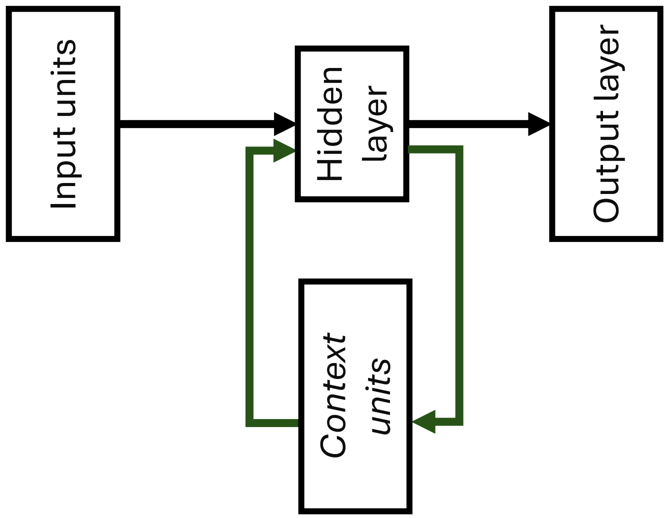




Examples: Classification | Image captioning | Sentiment analysis | Language translation | Video frame labelling

Recurrent neural networks

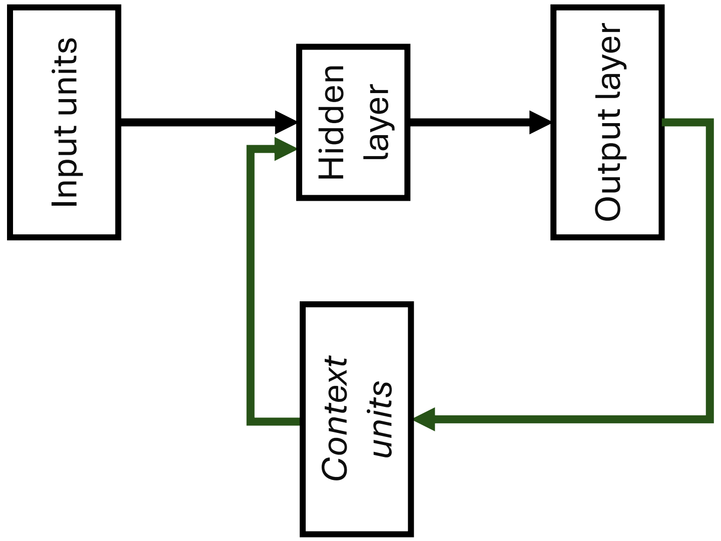
- Elman network



Base equations:

- $h_t = \sigma(W_h x_t + U_h h_{t-1} + b_h)$
- $y_t = \sigma_y(W_y h_t + b_y)$

- Jordan network

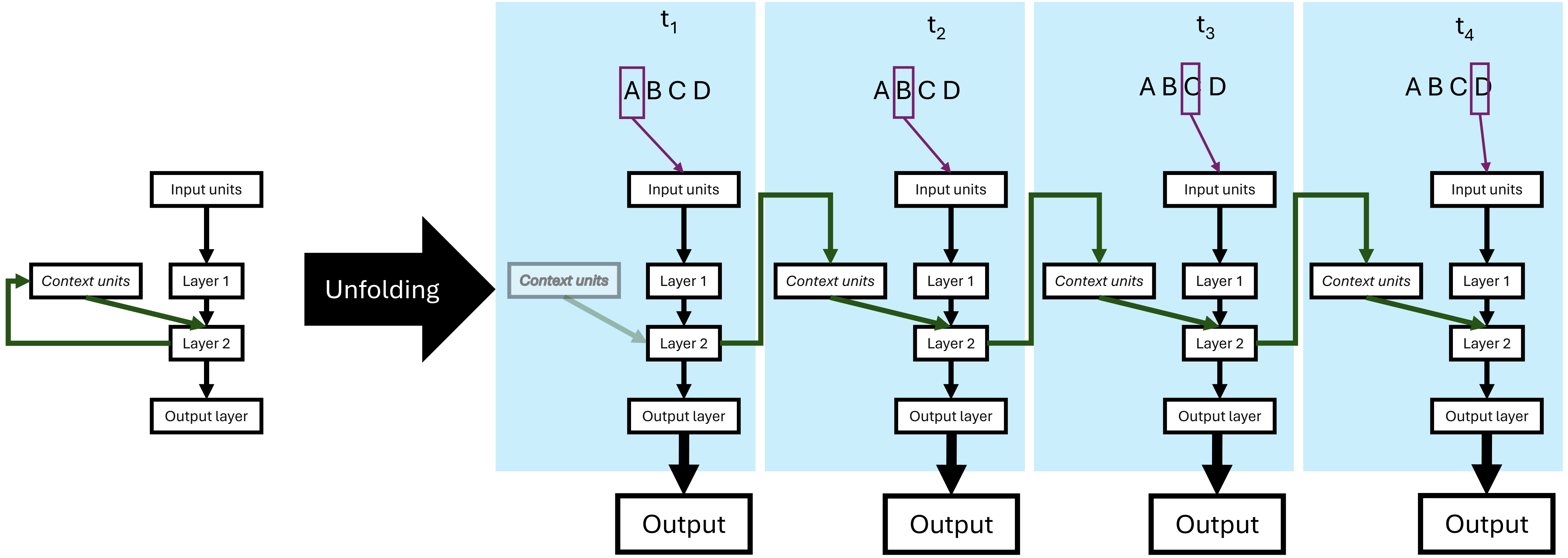


Base equations:

- $h_t = \sigma(W_h x_t + U_h y_{t-1} + b_h)$
- $y_t = \sigma_y(W_y h_t + b_y)$

Training recurrent neural networks

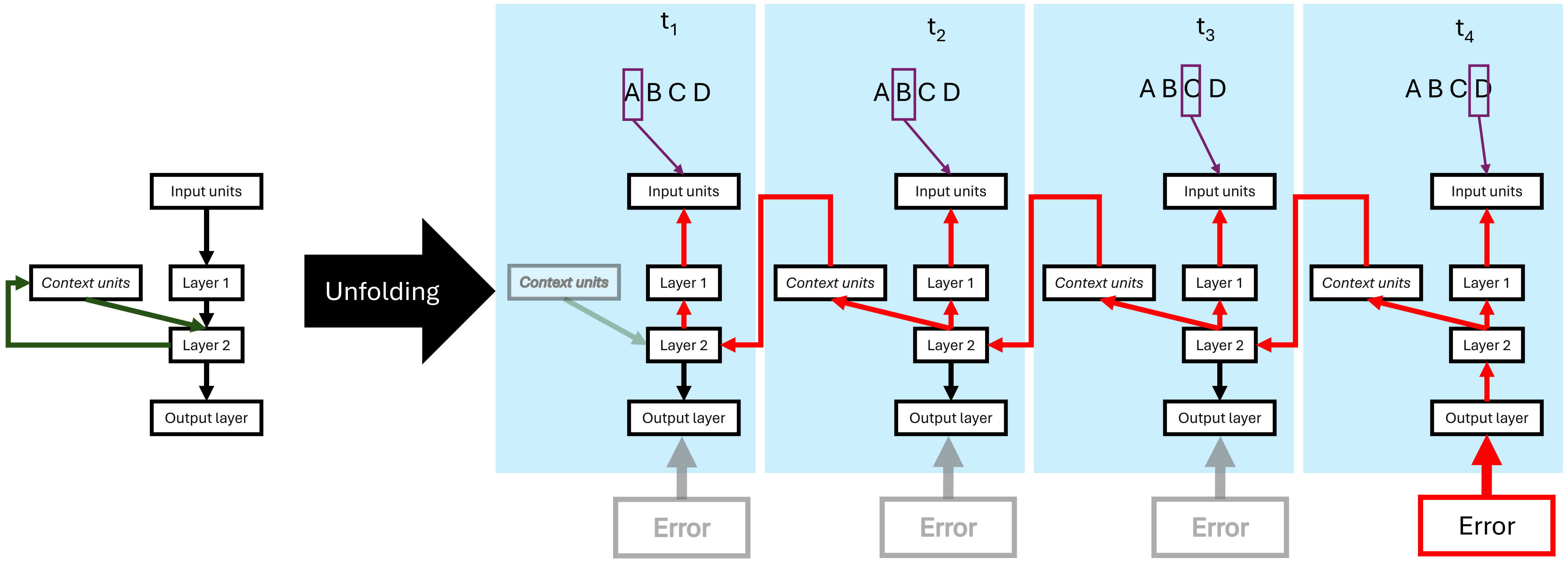
A, B, C, D… ➔ tokens (binary values, embeddings, etc.)



Training recurrent neural networks

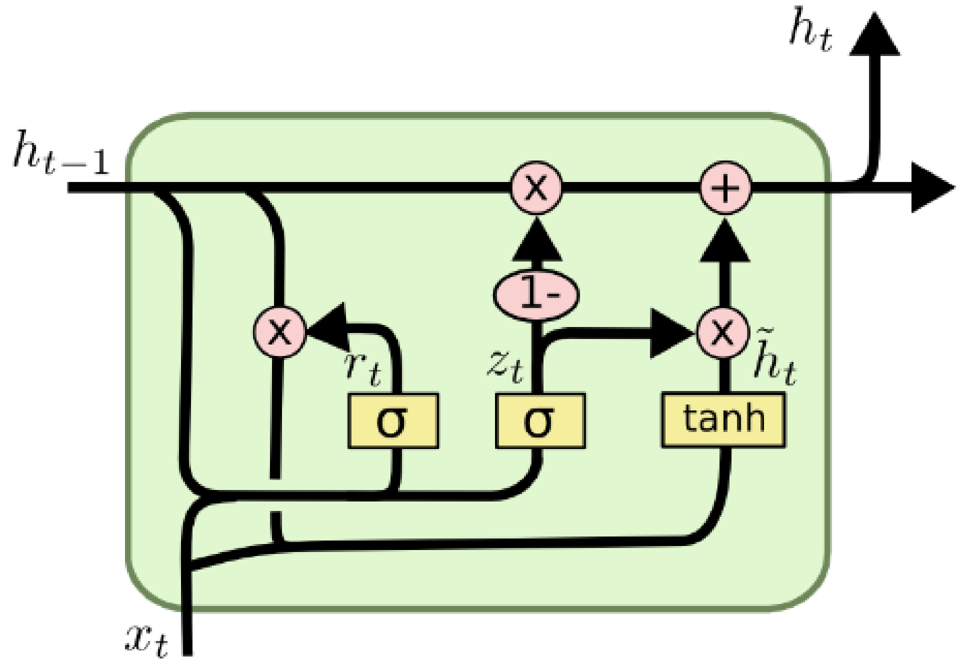
A, B, C, D… ➔ tokens (bin

Backpropagation Through Time (BPTT)



Long Short-Term Memory networks

- Recurrence + **Gating** = LSTM



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

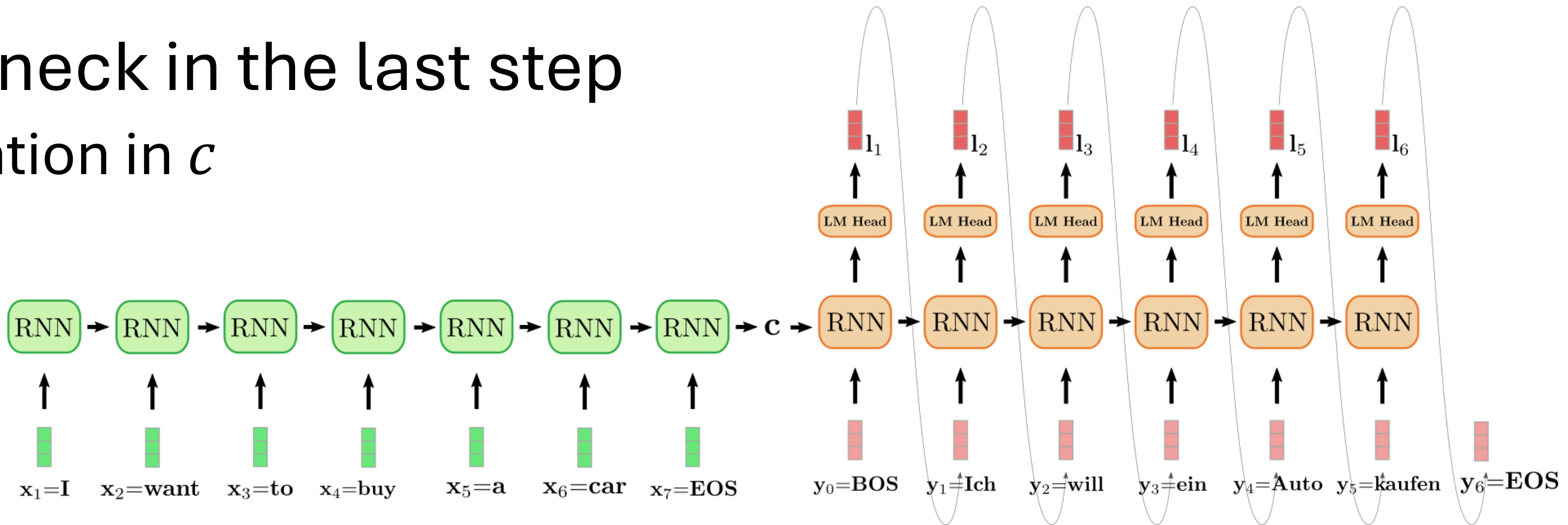
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

(image shamelessly stolen from Christopher Olah)

Encoder-decoder architecture

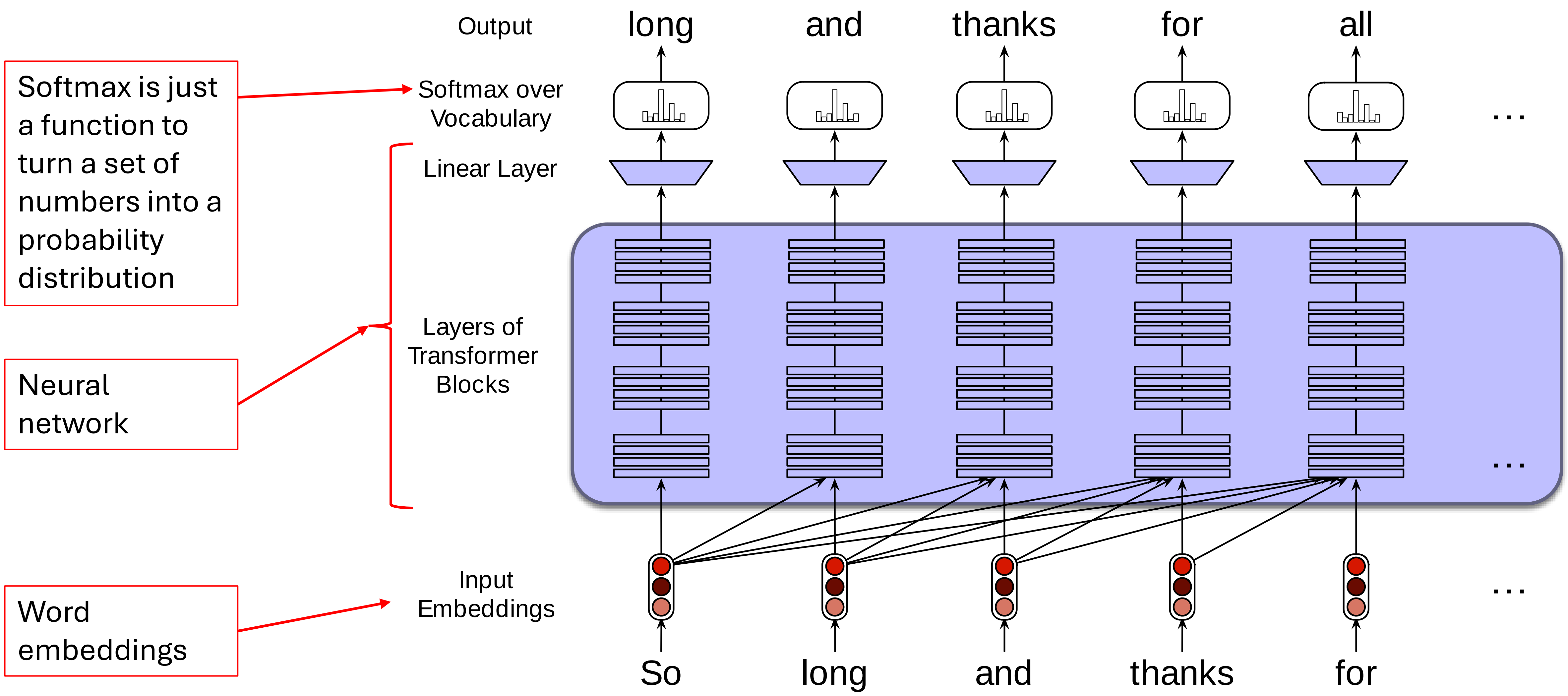
- Encoder
  - Takes an input sequence
  - Processes it through multiple encoder blocks
  - Builds and outputs a context vector c
- Decoder
  - Progressively generates output based on c and previous output
- Information bottleneck in the last step
  - Too much information in c



RNN
c
LM Head
l_1 l_2 l_3 l_4 l_5 l_6
x_1=I x_2=want x_3=to x_4=buy x_5=a x_6=car x_7=EOS
y_0=BOS y_1=Ich y_2=will y_3=ein y_4=Auto y_5=kaufen y_6=EOS

Illustration from
https://huggingface.co/blog/encoder-decoder

A picture of a transformer language model



Training phases of LLMs

- Training tends to be done in phases, e.g.
  - Unsupervised pre-training
    - The model is fed text and trained to predict the next token in a sequence. This teaches the model fundamental **language structure**, **grammar**, and common **word associations**.
    - Pre-training often requires a lot of computational resources and can take weeks or months, even with powerful hardware.
  - Supervised fine-tuning
    - The pre-trained model is refined for specific tasks like text summarisation, question answering, or translation. This involves training on smaller datasets of labeled data tailored to the intended task.
    - Fine-tuning helps the model learn the nuances and specific requirements of the task at hand.

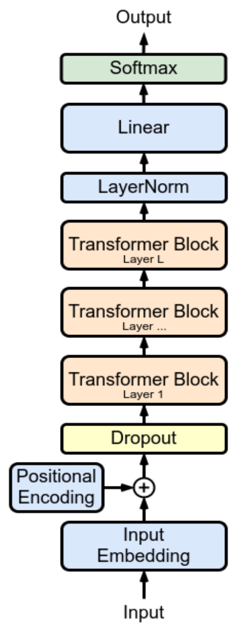
Training phases of LLMs

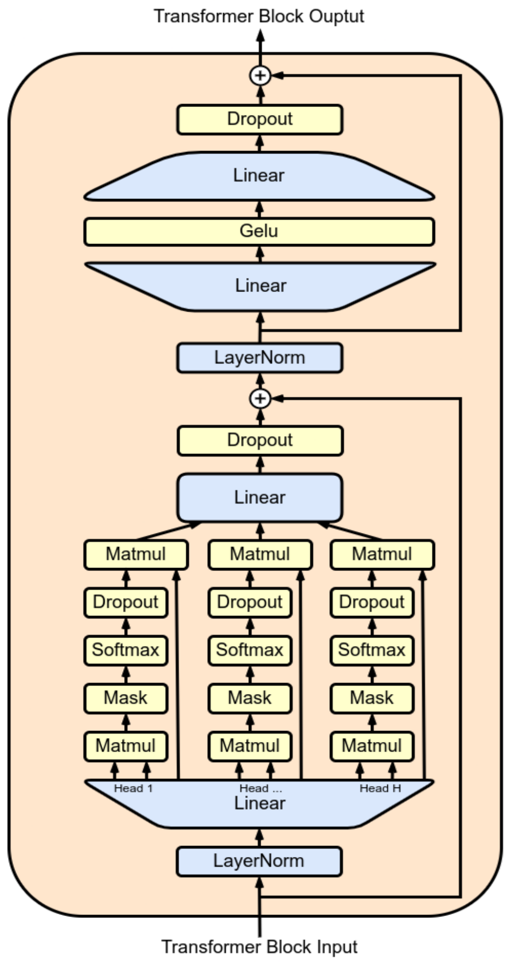
- Reinforcement Learning with Human Feedback
  1. Reward Model Training:
     1. Human Feedback: Humans rate and compare several different responses generated by the language model. This data conveys which kinds of outputs are seen as better.
     2. Reward Model: A separate model is trained to predict human preferences based on the feedback data. This model learns to assign a reward score to a given piece of text.
  2. Policy Optimisation using RL:
     1. Fine-tuning the LLM: The original language model (like GPT) is further fine-tuned, but instead of directly using a task-specific reward function, it uses the reward scores from the trained reward model.
     2. Reinforcement learning loop: The updated model generates new text, and the reward model again provides feedback, guiding the LLM to produce outputs more in line with what humans would consider desirable.

GPT-3

- Trained for a total of 300 billion tokens
- Based on raw Common Crawl dataset of up to 1T words

Model Name	n_{params}	n_{layers}
GPT-3 Small	125M	12
GPT-3 Medium	350M	24
GPT-3 Large	760M	24
GPT-3 XL	1.3GB	24
GPT-3 2.7B	2.7GB	32
GPT-3 6.7B	6.7GB	32
GPT-3 13B	13.0B	40
GPT-3 175B or “GPT-3”	175.0B	96

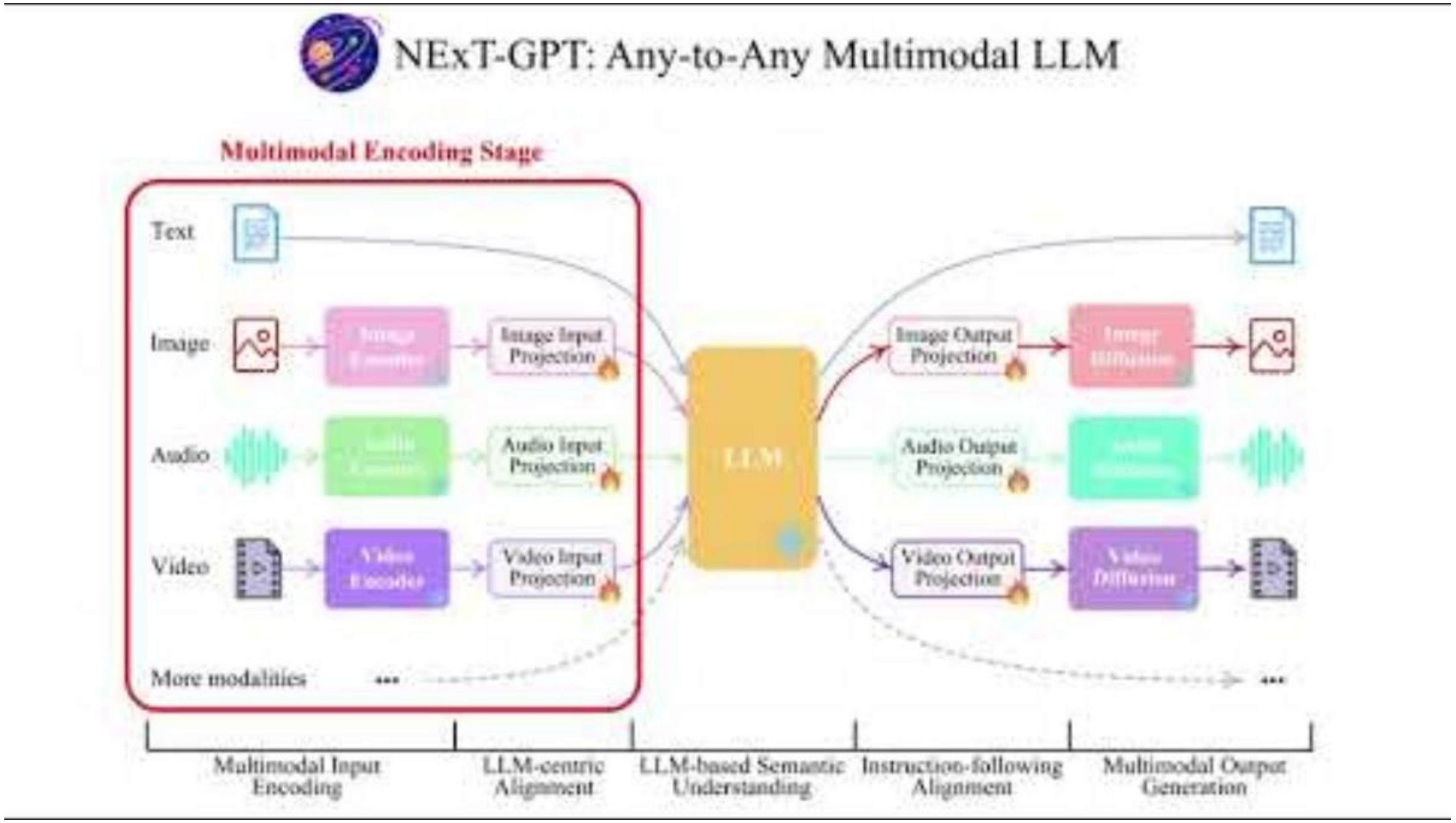




New(-ish) trends in LLMs

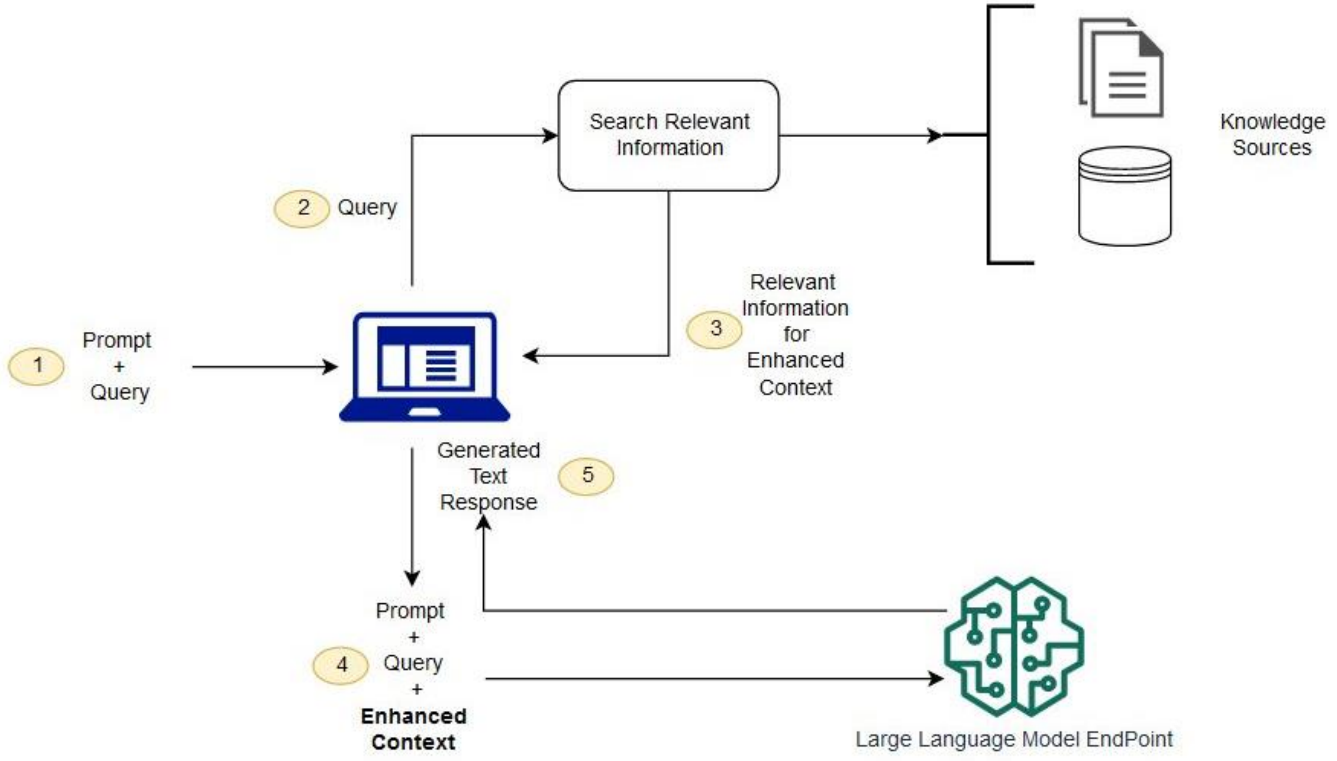
- Multimodal LLMs
  - Text, image, sound
- Retrieval-Augmented Generation
  - What if your LLM had a memory?
- Quantised LLMs (they fit in your phone!)
  - Using low rank adapters to approximate full-precision tensors with smaller representations (e.g., 32 bits → 4 bits)
- Specialised LLMs
  - Storytelling (the LLM puts you to sleep)
  - Embodiment (the LLM is in your robot)
  - Code generation (the LLM steals your job)
- Open source LLMs!

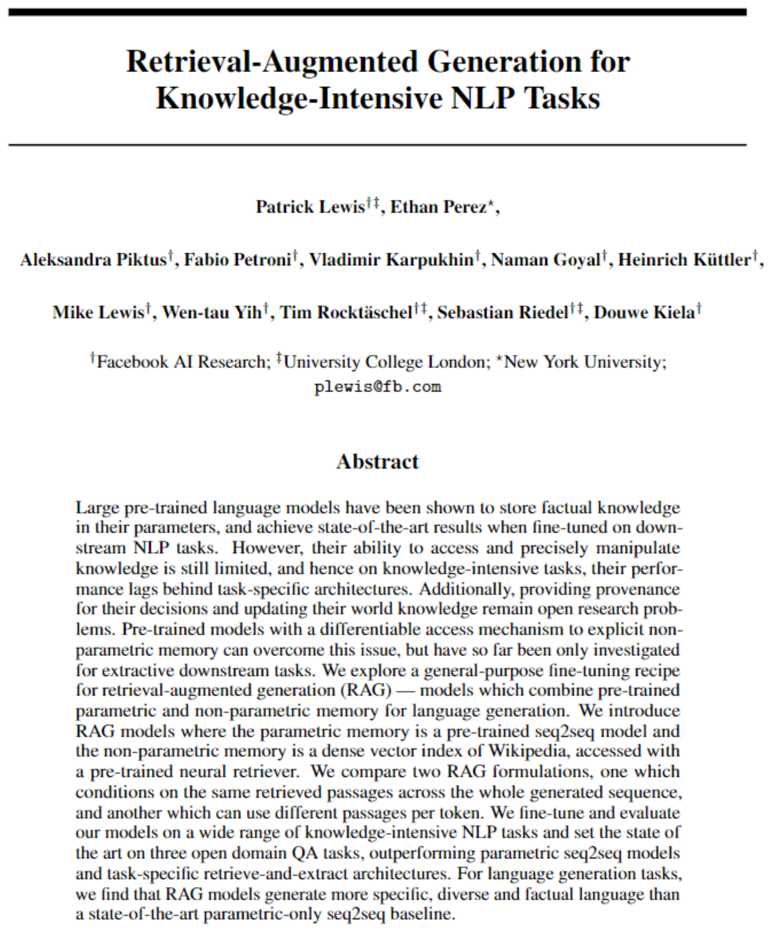
Multimodal LLMs



Retrieval-Augmented Generation

- What if your LLM had a memory?
  - Retrieve relevant documents from database
  - Feed as additional context in LLM


Search Relevant Information
Knowledge Sources
2 Query
Relevant Information for Enhanced Context
3
1
Prompt + Query
Generated Text Response
5
Prompt + Query + Enhanced Context
4
Large Language Model EndPoint



Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Patrick Lewis[†‡], Ethan Perez[*],

Aleksandra Piktus[†], Fabio Petroni[†], Vladimir Karpukhin[†], Naman Goyal[†], Heinrich Küttler[†],

Mike Lewis[†], Wen-tau Yih[†], Tim Rocktäschel[†‡], Sebastian Riedel[†‡], Douwe Kiela[†]

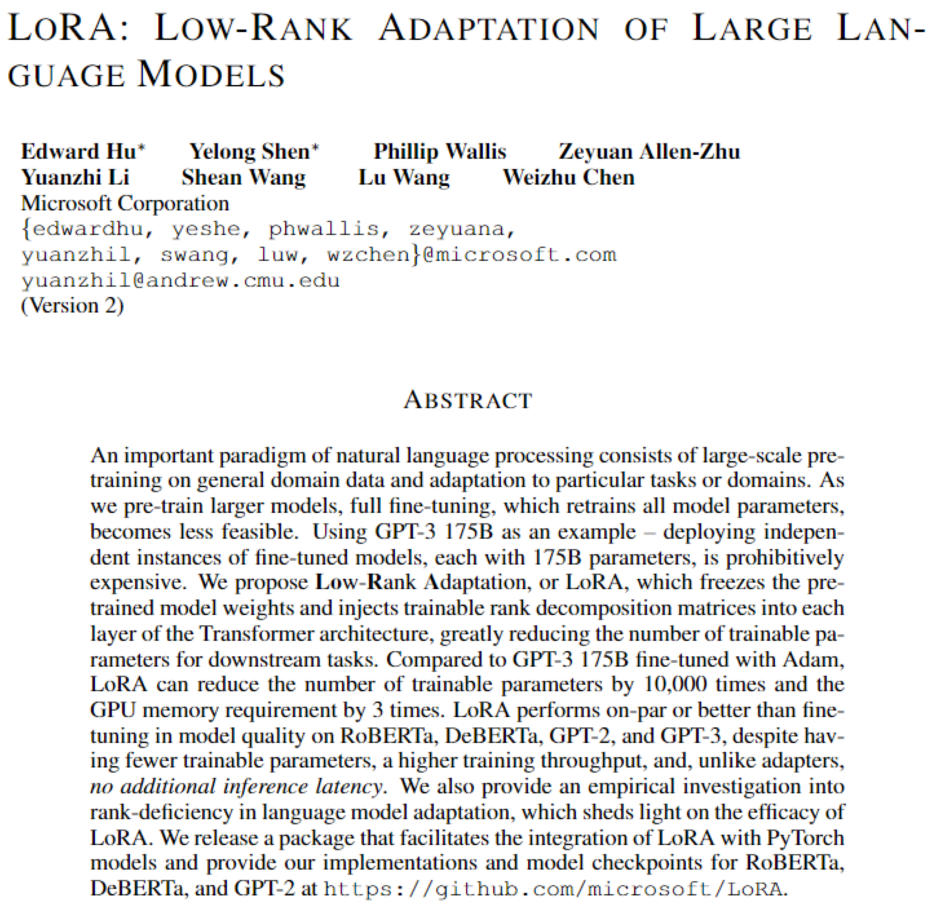
[†]Facebook AI Research; [‡]University College London; [*]New York University;
plewis@fb.com

Abstract

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory can overcome this issue, but have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric memory for language generation. We introduce RAG models where the parametric memory is a pre-trained seq2seq model and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We compare two RAG formulations, one which conditions on the same retrieved passages across the whole generated sequence, and another which can use different passages per token. We fine-tune and evaluate our models on a wide range of knowledge-intensive NLP tasks and set the state of the art on three open domain QA tasks, outperforming parametric seq2seq models and task-specific retrieve-and-extract architectures. For language generation tasks, we find that RAG models generate more specific, diverse and factual language than a state-of-the-art parametric-only seq2seq baseline.

Illustration from https://aws.amazon.com/what-is/retrieval-augmented-generation/

Quantised LLMs

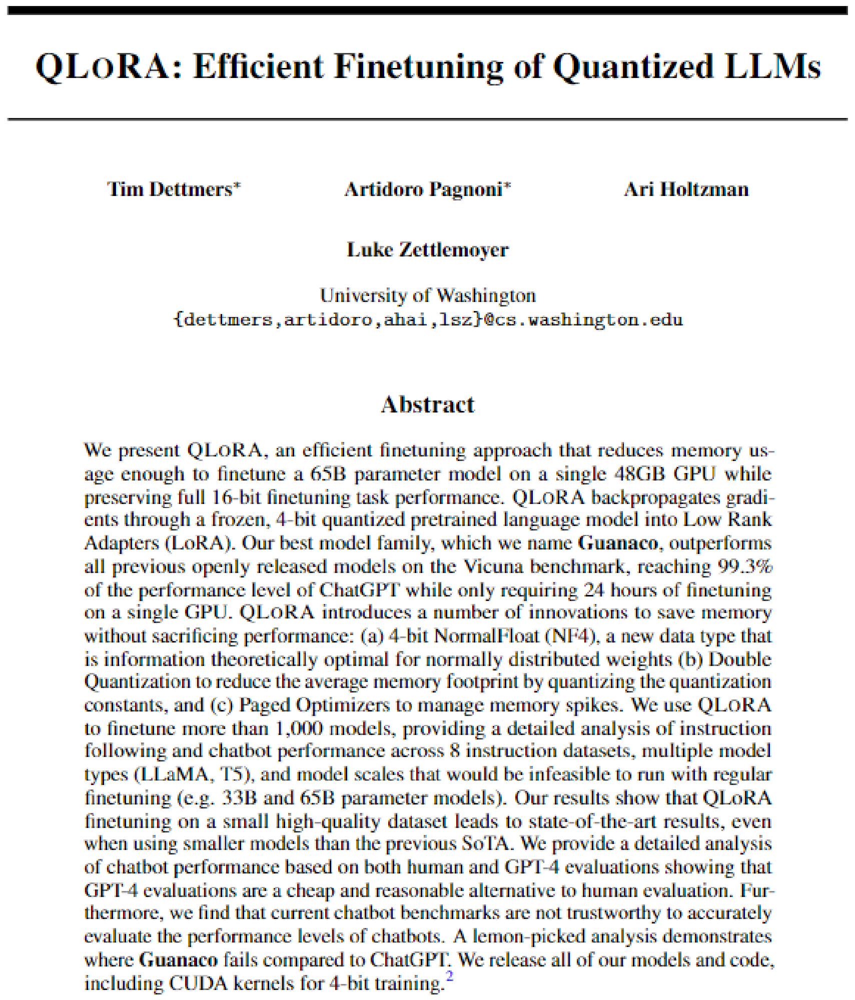


LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
`{edwardhu, yeshe, phwallis, zeyuana, yuanzhil, swang, luw, wzchen}@microsoft.com`
`yuanzhil@andrew.cmu.edu`
(Version 2)

Abstract

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 **Lo**w-**R**ank **A**daptation, 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 `https://github.com/microsoft/LoRA`.



QLoRA: Efficient Finetuning of Quantized LLMs

Tim Dettmers* **Artidoro Pagnoni*** **Ari Holtzman**
Luke Zettlemoyer
University of Washington
`{dettmers,artidoro,ahai,lsz}@cs.washington.edu`

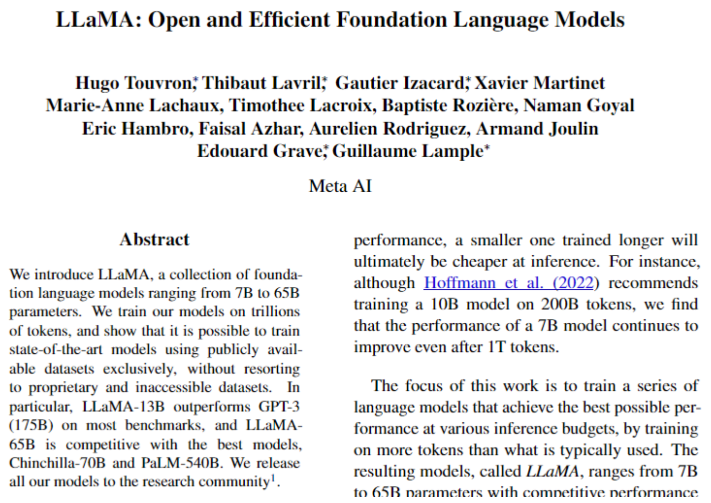
Abstract

We present QLoRA, an efficient finetuning approach that reduces memory usage enough to finetune a 65B parameter model on a single 48GB GPU while preserving full 16-bit finetuning task performance. QLoRA backpropagates gradients through a frozen, 4-bit quantized pretrained language model into Low Rank Adapters (LoRA). Our best model family, which we name **Guanaco**, outperforms all previous openly released models on the Vicuna benchmark, reaching 99.3% of the performance level of ChatGPT while only requiring 24 hours of finetuning on a single GPU. QLoRA introduces a number of innovations to save memory without sacrificing performance: (a) 4-bit NormalFloat (NF4), a new data type that is information theoretically optimal for normally distributed weights (b) Double Quantization to reduce the average memory footprint by quantizing the quantization constants, and (c) Paged Optimizers to manage memory spikes. We use QLoRA to finetune more than 1,000 models, providing a detailed analysis of instruction following and chatbot performance across 8 instruction datasets, multiple model types (LLaMA, T5), and model scales that would be infeasible to run with regular finetuning (e.g. 33B and 65B parameter models). Our results show that QLoRA finetuning on a small high-quality dataset leads to state-of-the-art results, even when using smaller models than the previous SoTA. We provide a detailed analysis of chatbot performance based on both human and GPT-4 evaluations showing that GPT-4 evaluations are a cheap and reasonable alternative to human evaluation. Furthermore, we find that current chatbot benchmarks are not trustworthy to accurately evaluate the performance levels of chatbots. A lemon-picked analysis demonstrates where **Guanaco** fails compared to ChatGPT. We release all of our models and code, including CUDA kernels for 4-bit training.[2]

Specialised LLMs

- E.g., embodiment!

Open source LLMs



LLaMA: Open and Efficient Foundation Language Models

Hugo Touvron*, Thibaut Lavril*, Gautier Izacard*, Xavier Martinet
Marie-Anne Lachaux, Timothee Lacroix, Baptiste Rozière, Naman Goyal
Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin
Edouard Grave*, Guillaume Lample*

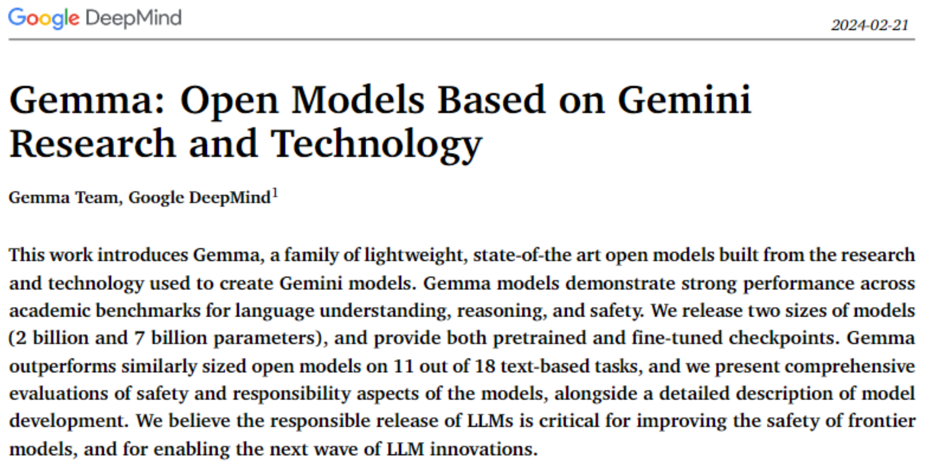
Meta AI

Abstract

We introduce LLaMA, a collection of foundation language models ranging from 7B to 65B parameters. We train our models on trillions of tokens, and show that it is possible to train state-of-the-art models using publicly available datasets exclusively, without resorting to proprietary and inaccessible datasets. In particular, LLaMA-13B outperforms GPT-3 (175B) on most benchmarks, and LLaMA-65B is competitive with the best models, Chinchilla-70B and PaLM-540B. We release all our models to the research community[1].

performance, a smaller one trained longer will ultimately be cheaper at inference. For instance, although Hoffmann et al. (2022) recommends training a 10B model on 200B tokens, we find that the performance of a 7B model continues to improve even after 1T tokens.

The focus of this work is to train a series of language models that achieve the best possible performance at various inference budgets, by training on more tokens than what is typically used. The resulting models, called *LLaMA*, ranges from 7B to 65B parameters with competitive performance

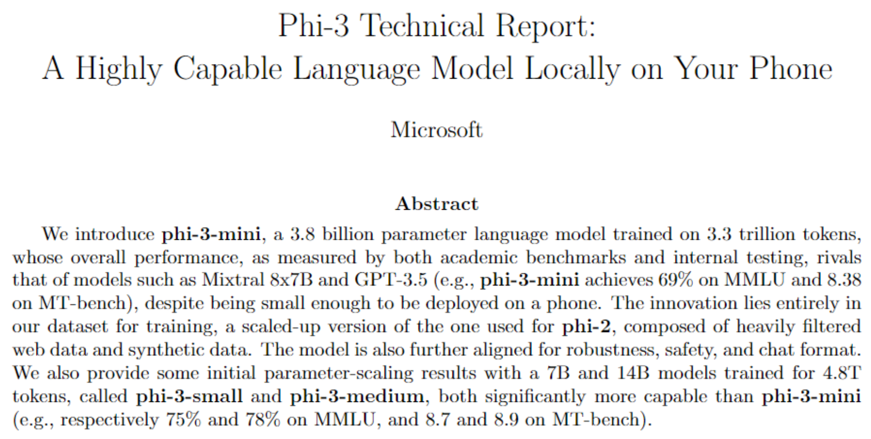


Google DeepMind 2024-02-21

Gemma: Open Models Based on Gemini Research and Technology

Gemma Team, Google DeepMind[1]

This work introduces Gemma, a family of lightweight, state-of-the art open models built from the research and technology used to create Gemini models. Gemma models demonstrate strong performance across academic benchmarks for language understanding, reasoning, and safety. We release two sizes of models (2 billion and 7 billion parameters), and provide both pretrained and fine-tuned checkpoints. Gemma outperforms similarly sized open models on 11 out of 18 text-based tasks, and we present comprehensive evaluations of safety and responsibility aspects of the models, alongside a detailed description of model development. We believe the responsible release of LLMs is critical for improving the safety of frontier models, and for enabling the next wave of LLM innovations.

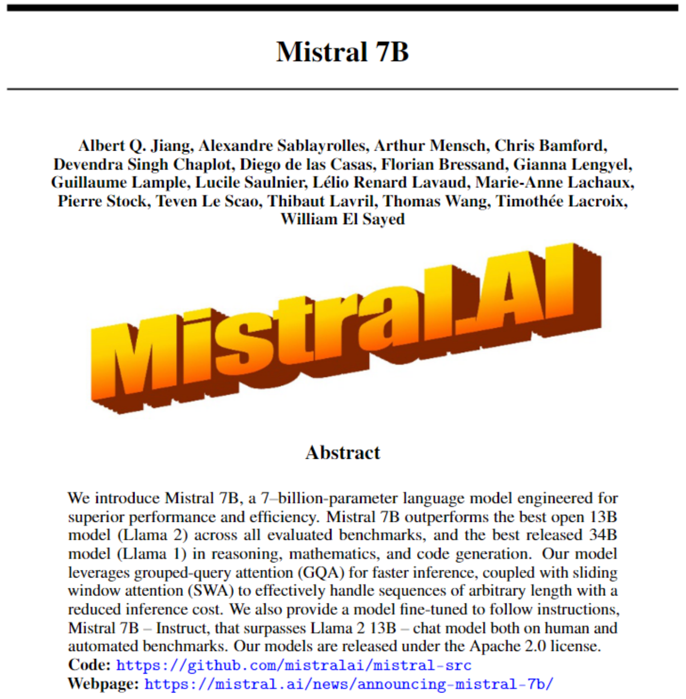


Phi-3 Technical Report:
A Highly Capable Language Model Locally on Your Phone

Microsoft

Abstract

We introduce **phi-3-mini**, a 3.8 billion parameter language model trained on 3.3 trillion tokens, whose overall performance, as measured by both academic benchmarks and internal testing, rivals that of models such as Mixtral 8x7B and GPT-3.5 (e.g., **phi-3-mini** achieves 69% on MMLU and 8.38 on MT-bench), despite being small enough to be deployed on a phone. The innovation lies entirely in our dataset for training, a scaled-up version of the one used for **phi-2**, composed of heavily filtered web data and synthetic data. The model is also further aligned for robustness, safety, and chat format. We also provide some initial parameter-scaling results with a 7B and 14B models trained for 4.8T tokens, called **phi-3-small** and **phi-3-medium**, both significantly more capable than **phi-3-mini** (e.g., respectively 75% and 78% on MMLU, and 8.7 and 8.9 on MT-bench).



Mistral 7B

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed

Abstract

We introduce Mistral 7B, a 7–billion-parameter language model engineered for superior performance and efficiency. Mistral 7B outperforms the best open 13B model (Llama 2) across all evaluated benchmarks, and the best released 34B model (Llama 1) in reasoning, mathematics, and code generation. Our model leverages grouped-query attention (GQA) for faster inference, coupled with sliding window attention (SWA) to effectively handle sequences of arbitrary length with a reduced inference cost. We also provide a model fine-tuned to follow instructions, Mistral 7B – Instruct, that surpasses Llama 2 13B – chat model both on human and automated benchmarks. Our models are released under the Apache 2.0 license.
Code: https://github.com/mistralai/mistral-src
Webpage: https://mistral.ai/news/announcing-mistral-7b/