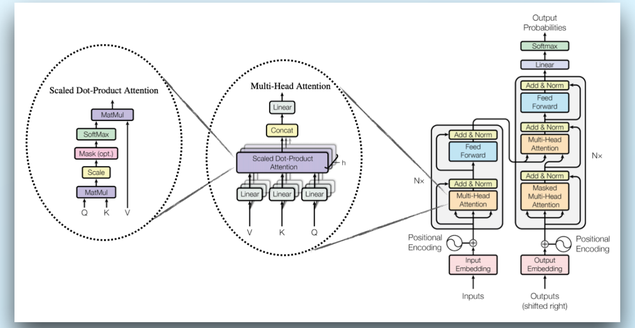
Time-series, Transformers, and Natural Language Processing

Inar Timiryasov (NBI)
inar.timiryasov@nbi.ku.dk

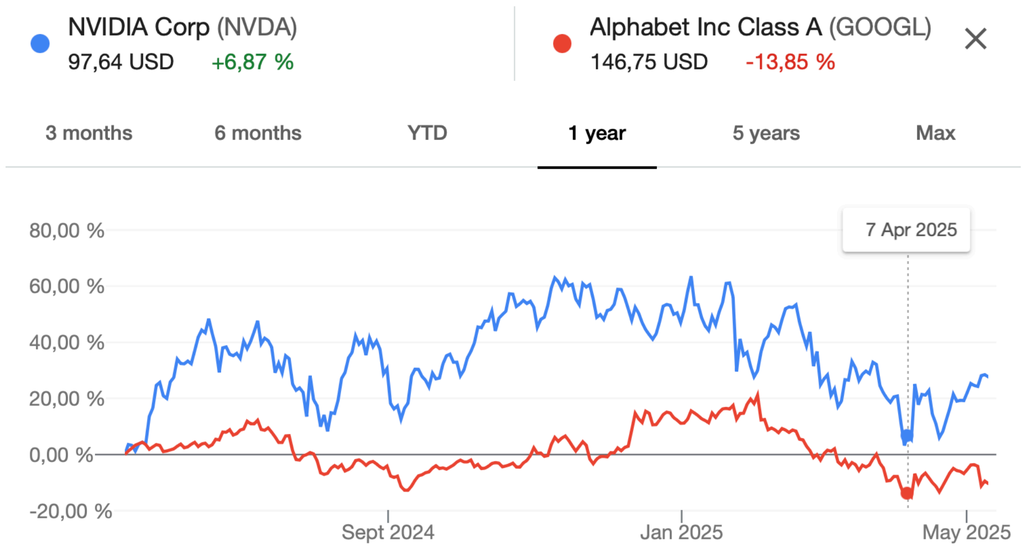
May 18, 2026

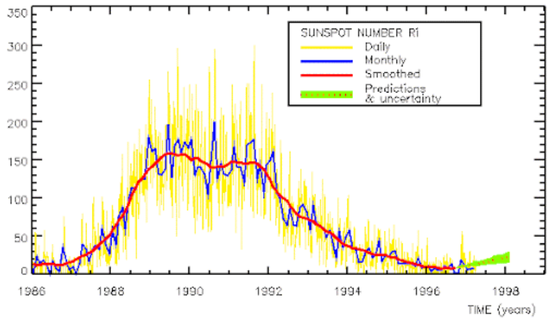


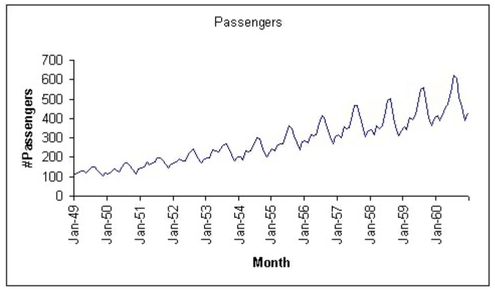
Plan

- Handling Sequential Data: Time Series & Recurrent Architectures
- The Attention Mechanism: A Leap Forward
- Transformers: Architecture & Impact
- Transformers in Natural Language Processing (NLP)

Time series data and limitations of the usual network architectures



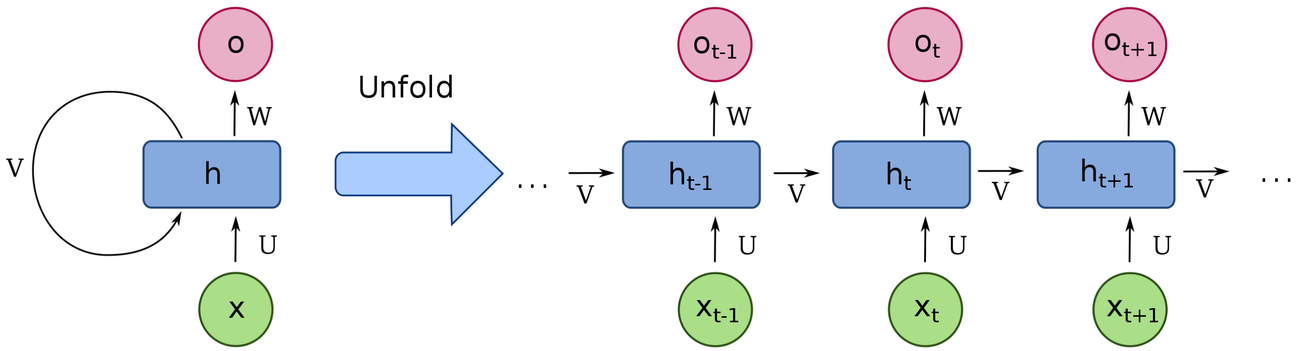




- Time series data are ubiquitous:
  Stock prices; weather patterns; sensor readings; flight passenger numbers (today's exercise); speech; text
- Why is it special? Order matters, dependencies between points.
- Usual NN approaches are not well suited for these types of data:
  - Lack of temporal dynamics
    (treating input features independently)
  - Fixed input size
  - CNNs capture only local dependencies by design

Recurrent Neural Network (RNN)

- A network with memory (hidden state).
- Analogy: Reading a sentence, understanding depends on previous words. Compressed information about previous words is stored in the hidden state.
- We apply the same model to each time step (unfold)!
- We can only process one element at a time — this is a limitation

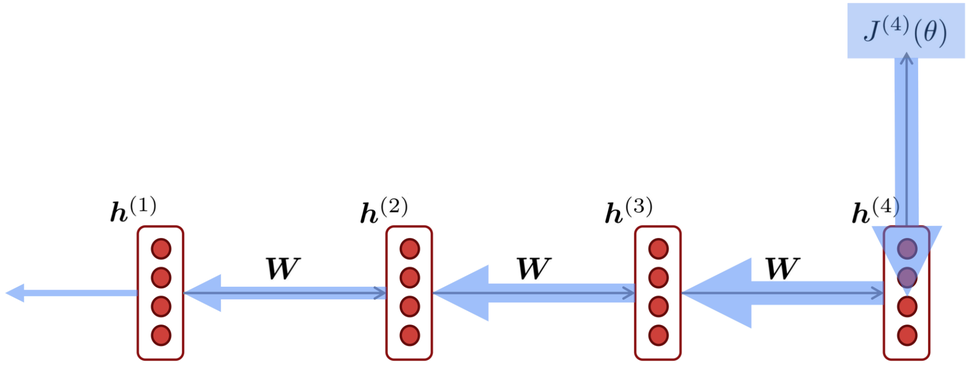


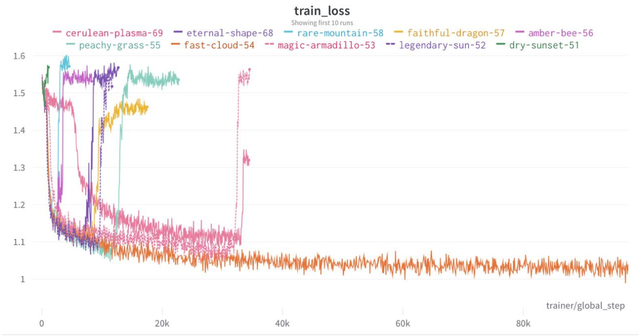
h_t - hidden state

$$a_t = V h_{t-1} + U x_t + b$$
$$h_t = \tanh(a_t)$$
$$o_t = W h_t + c$$

Challenges with Simple RNNs & Evolution

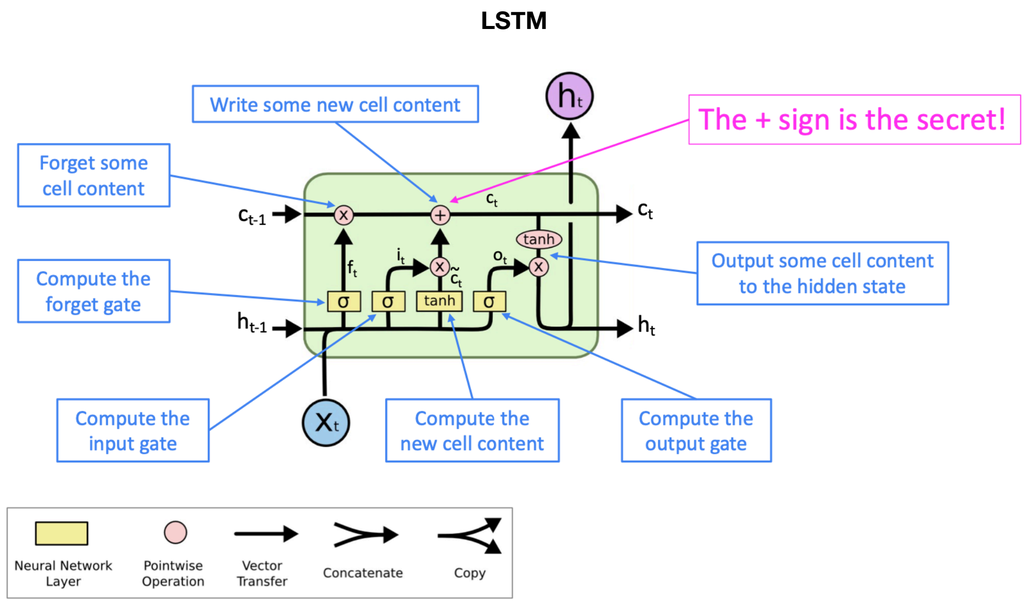
- Training RNNs: backpropagation through time.
- Each time step of the unrolled recurrent neural network may be seen as an additional layer.
- Vanishing/Exploding Gradients: Difficulty learning long-range dependencies.
- A fix is a “skip connection” (similar to U-Net and ResNet, but in time, not in depth)
- LSTM, GRU

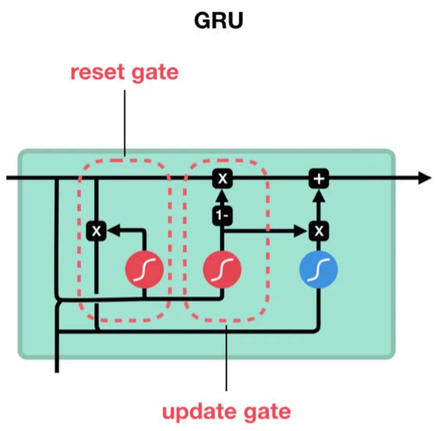




Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU)

- Skip connection — long term memory — a “highway" in time
- Gates decide what to write to the long term memory
- They are better at capturing longer-term patterns.





Exercise: Flight Passenger Predictions

Sources: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture06-fancy-rnn.pdf

Recap on RNNs

- The same network is used at every time step (hence recurrent).

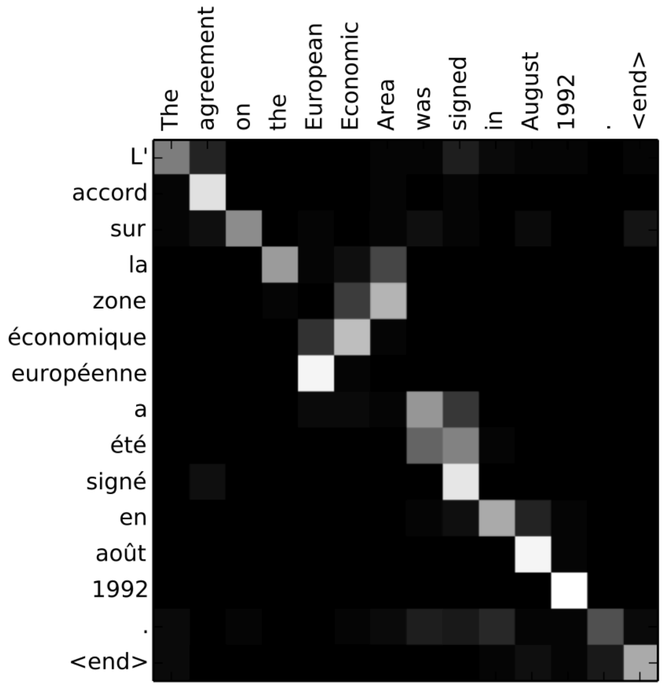
- Vanilla RNNs are hard to train.
  LSTM/GRU partly solve this problem.

- All information about the past sequence must be compressed into the hidden state.
  Pros: compression is intelligence [see "An Observation on Generalization" by Ilya Sutskever].
  Intuition: there are exponentially many almost orthogonal vectors in D>>1 dimensions.
  Cons: in practice, only a limited amount of information can be stored in the hidden state.
  RNNs tend to forget the past context after a certain length.

- Hard to train in parallel, since we need to process steps sequentially.
  (but there are ways around, keywords: RWKV, State Space Models, e.g. Mamba.)

Attention!

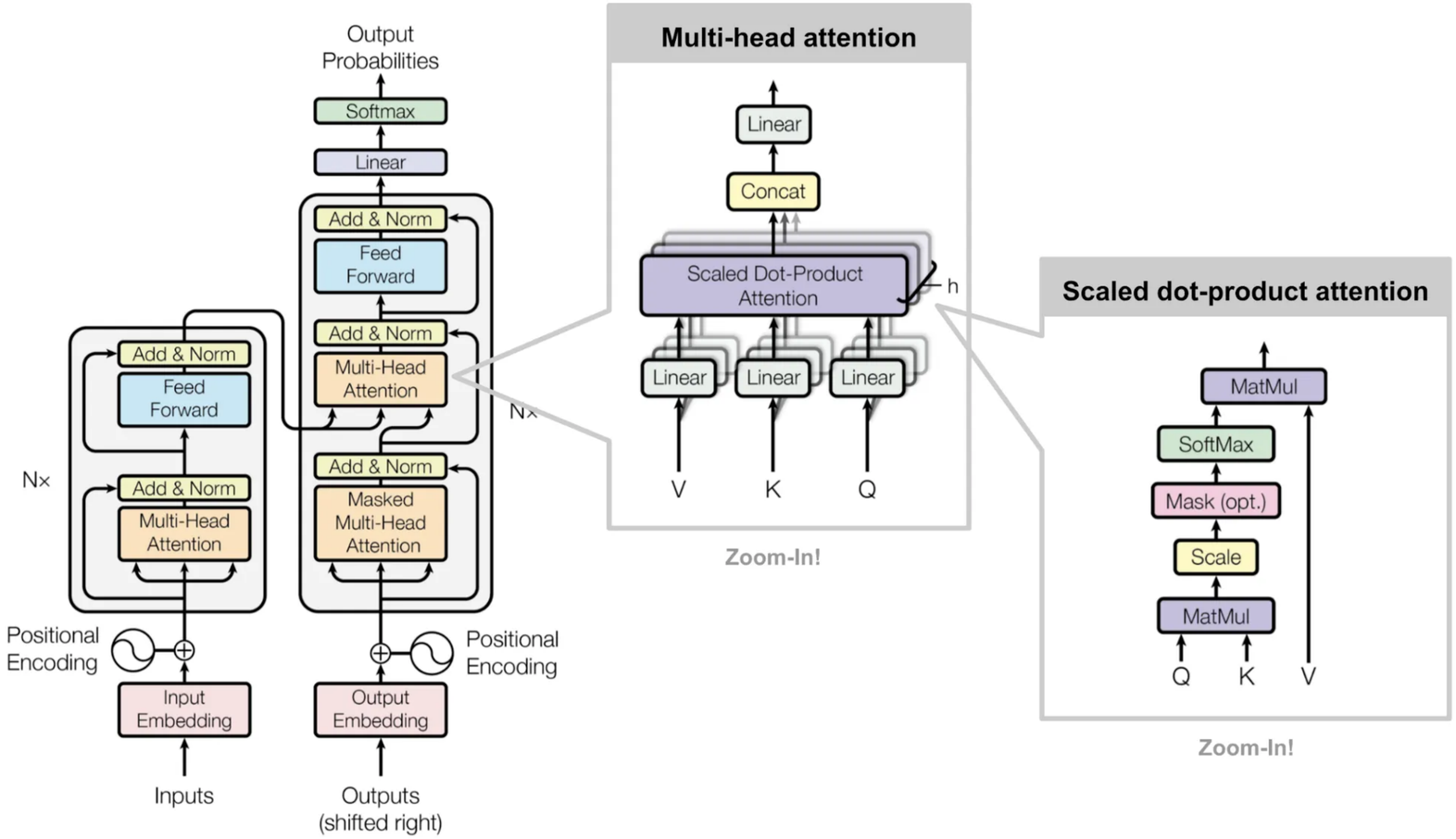
- Core Idea: Allow the model to "look back" at different parts of the input sequence when processing or generating an output, and weigh their importance.
- Analogy: Human translators don't just read a whole sentence and then translate; they focus on relevant parts.
- Benefit: Mitigates information bottleneck, better context.
- Self-Attention: Attention within a Single Sequence



Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio
Neural Machine Translation by Jointly Learning to Align and Translate
arXiv:1409.0473

RNN with attention

How Self-Attention works?



How Self-Attention works?

- Input x_s at time step s

- We project each input x_s using three different transformations:
  $v_s = W^V x_s$
  $k_s = W^K x_s$
  $q_s = W^Q x_s$

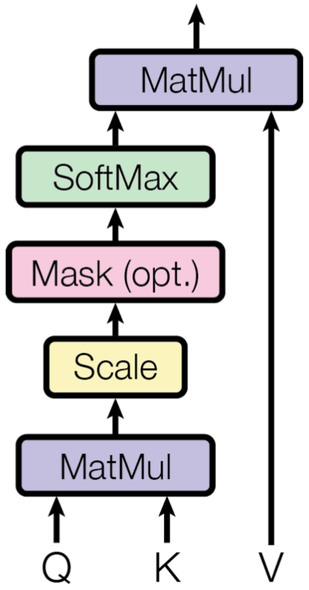
- Output at step t is the sum of v_s over all steps s with weights A_{ts} :

$$y_t = \sum_s A_{ts} v_s$$

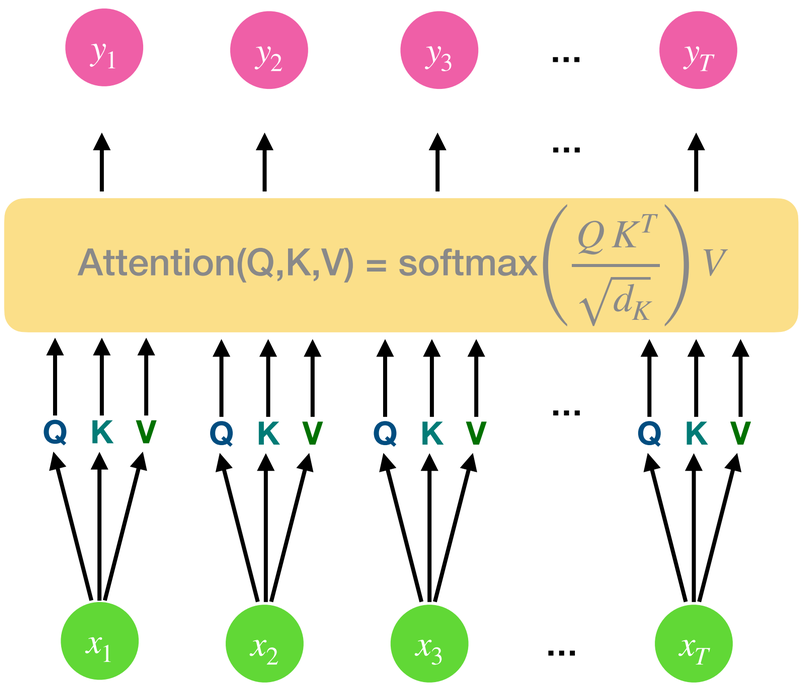
- where attention weights $A_{ts} = \text{softmax}\left(\frac{q_t \cdot k_s}{\sqrt{d}}\right)$

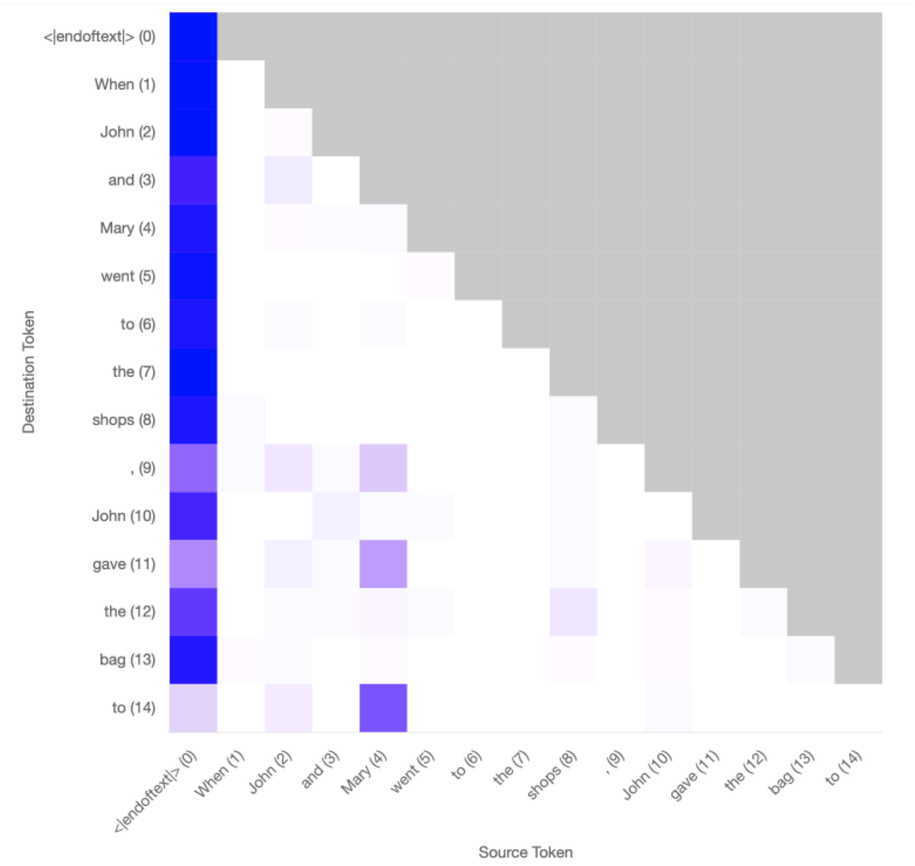
- $y_t = \sum_s \text{softmax}\left(\frac{(W^Q x_t) \cdot (W^K x_s)^T}{\sqrt{d}}\right)(W^V x_s)$

Scaled Dot-Product Attention



How Self-Attention works?





$$y_t = \sum_s \text{softmax}\left(\frac{(W^Q x_t)\cdot(W^K x_s)^T}{\sqrt{d_K}}\right)(W^V x_s)$$

attention becomes non-trivial with multiple layers
see [Transformer Circuits Thread]

Transformer

- Initially proposed by Google for machine translation tasks.

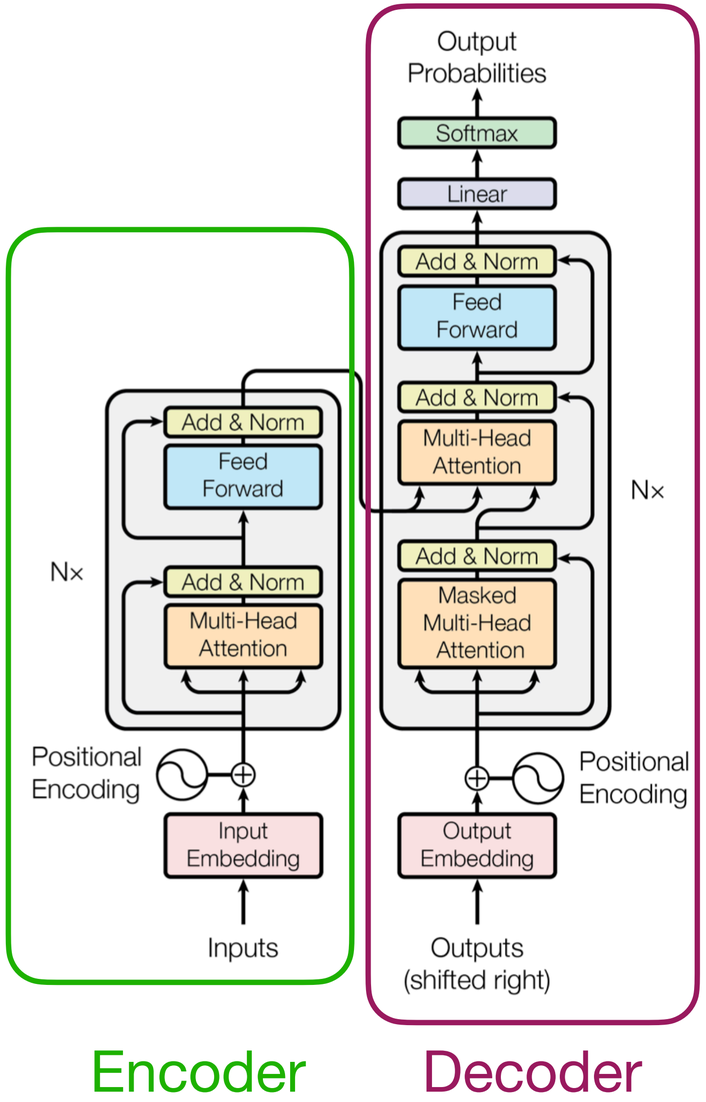
- They utilize the attention mechanism introduced in [1409.0473], but without an RNN component.

- The original architecture consisted of an encoder and a decoder:

  - **Encoder**: Processes input sequence (e.g., sentence to be translated).
    Each position attends to all other positions in the input sequence.

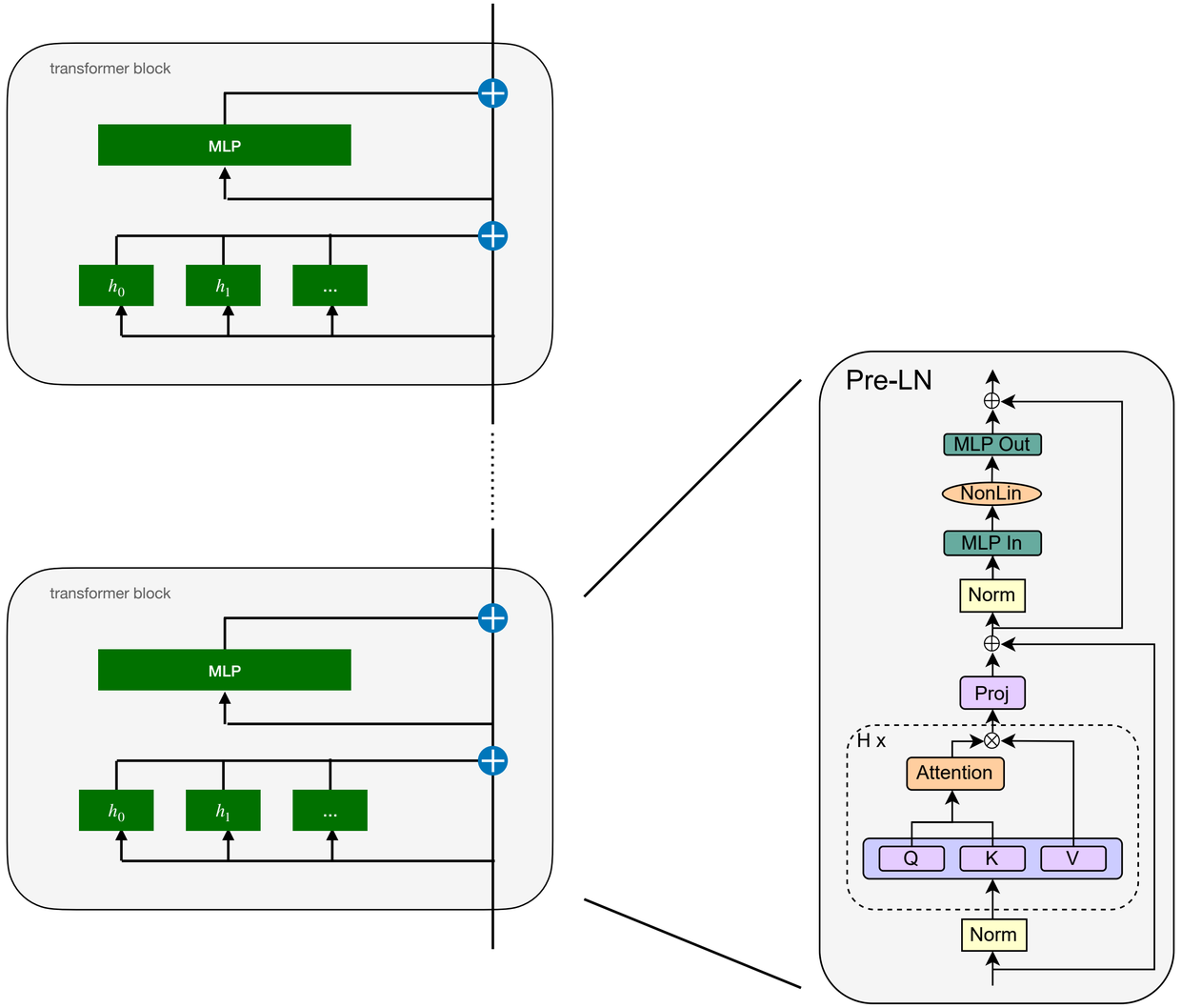
  - **Decoder**: Generates output sequence (e.g., the translation), attending to encoder output.
    Each position attends to previous positions (including itself) in the output sequence (causal attention).

- Generative models, such as GPT, are decoder-only.
  In physics, we often need encoder models.
  (Technically, the only difference: decoders feature a causal attention mask.)



Transformer

- Now that diagram is not so scary!
- Stack multiple attention heads.
- Add MLP.
- Add norms (before) Attention and MLP.
- Repeat n_layer times.
- Positional embeddings to break permutation invariance. (nowadays: RoPE)



Why Transformers are a Breakthrough

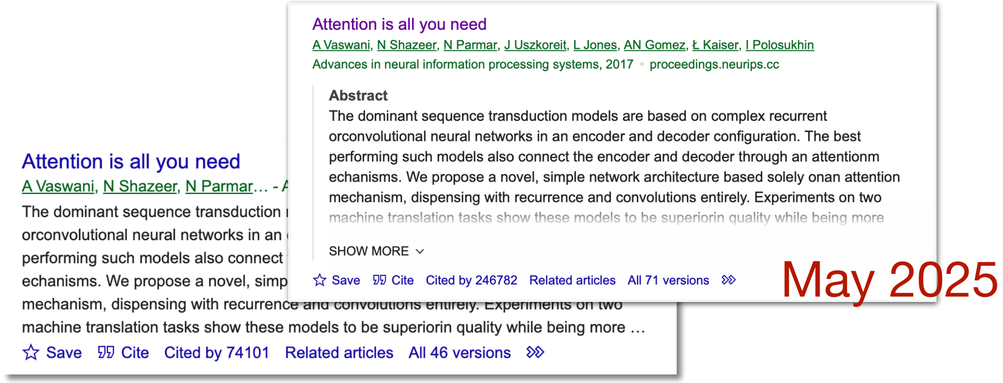
- Superior performance on many sequence tasks, especially NLP. Also on vision.

- Handles long-range dependencies effectively.
  Frontier models have context length **1M**

- Parallelization: Faster training on modern hardware.
  Computations for all tokens within a layer can be performed in parallel, unlike RNN.

- The price to pay is $\mathcal{O}(N^2)$ computational and memory complexity with respect to sequence length N.
  Mitigations:
  - Efficient implementations: FlashAttention
  - KV Caching for generation.

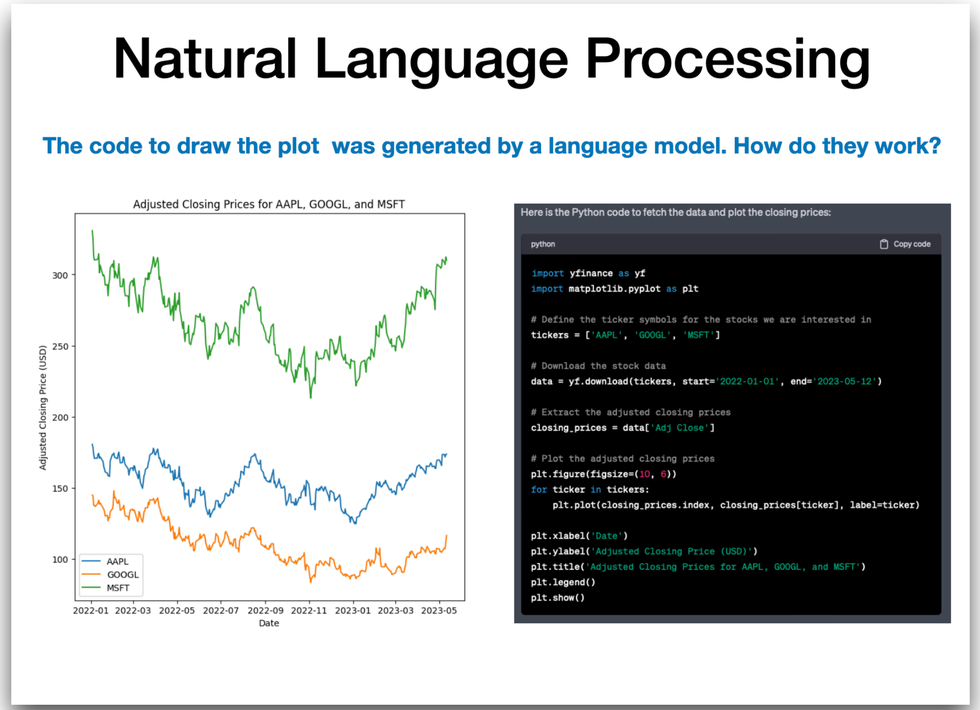
- Scalability: Foundation for very large models (LLMs).



May 2025

May 2023

Natural Language Processing

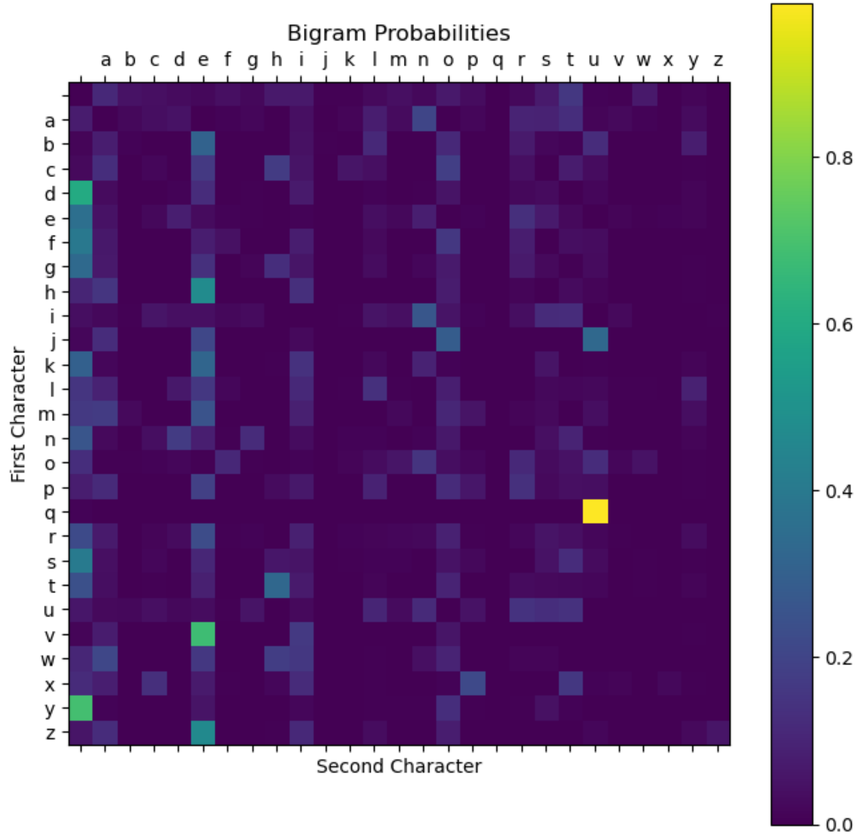


My slide from 2023
Hard to impress in 2026!

- Goal: Enabling computers to understand, interpret, and generate human language.
  - Text Classification (e.g., sentiment analysis)
  - Machine Translation
  - **Text Generation** <- today's focus
    Also known as *language modeling, autoregressive generation*

Language modeling: why does it even work?

- Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ estimate $P(x^{(t+1)} \mid x^{(1)}, x^{(2)}, \ldots, x^{(t)})$
- N-gram models: estimate $P(x^{(N)} \mid x^{(1)}, x^{(2)}, \ldots, x^{(N-1)})$ from a corpus of texts by simple counting
- for a vocabulary of 50 000 words:
- $50000^2 = 2500000000$ bigrams
- 1.25×10^{14} trigrams
- ...
- 6.1×10^{65} 14-grams



Representing words

- Vocabulary: enumerate all words
  But there are too many words in many languages

- Tokenization: Breaking text into smaller units (words, sub-words)
  Note! Tokenization causes its own issues
  [very detailed lecture]

Tokens 220 Characters 747

```
Week 4 (Convolutional Neural Networks (CNNs), Recurrent Neural Networks (
RNNs), and Auto-Encoders (AE)):
May 15: 13:15-17:00: Convolutional Neural Networks (CNNs) and image
 analysis (Daniel Murnane).
    Exercise: Recognize images (MNIST dataset, sparse chips for
 radiation, and/or insoluables from Greenland ice cores) with a CNN.
May 17: 9:15-12:00: Recurrent Neural Networks (RNN), Long Short Term
 Memory (LSTM) and Natural Language Processing (NLP) (Inar Timiryasov).
    Exercise: Use an LSTM to predict flight traffic and do Natural
 Language Processing on IMDB movie reviews.
May 17: 13:15-17:00: (Variational) Auto-Encoder and anomaly detection (TP
).
    Exercise: Compress images using Auto-Encoder, and cluster latent
 space with UMAP.
```

TEXT TOKEN IDS

A helpful rule of thumb is that one token generally corresponds to ~4 characters of text for common English text. This translates to roughly ¾ of a word (so 100 tokens ~= 75 words).

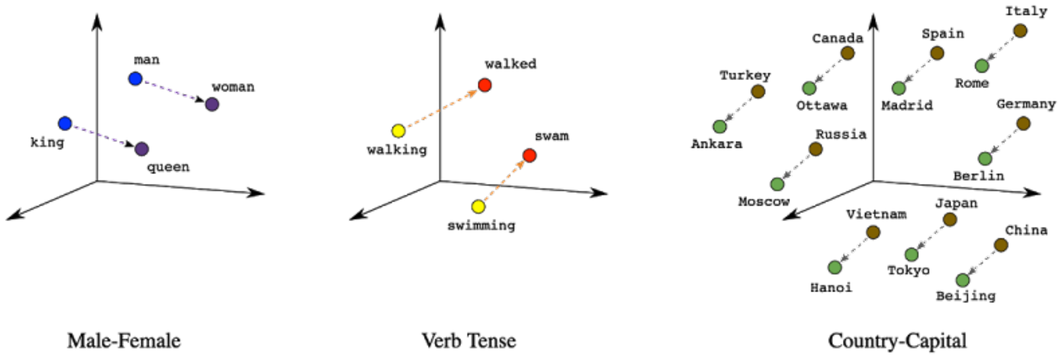
Tokens 220 Characters 747

```
[20916, 604, 357, 3103, 85, 2122, 282, 47986, 27862, 357, 18474, 82, 828,
3311, 6657, 47986, 27862, 357, 49, 6144, 82, 828, 290, 11160, 12, 4834,
19815, 364, 357, 14242, 8, 2599, 220, 198, 6747, 1315, 25, 1511, 25,
1314, 12, 1558, 25, 405, 25, 34872, 2122, 282, 47986, 27862, 357, 18474,
82, 8, 290, 2939, 3781, 357, 19962, 337, 700, 1531, 737, 198, 220, 220,
220, 220, 32900, 25, 31517, 1096, 4263, 357, 39764, 8808, 27039, 11,
29877, 12014, 329, 11881, 11, 290, 14, 273, 35831, 84, 2977, 422, 30155,
4771, 21758, 8, 351, 257, 8100, 13, 198, 6747, 1596, 25, 860, 25, 1314,
12, 1065, 25, 405, 25, 3311, 6657, 47986, 27862, 357, 49, 6144, 828,
5882, 10073, 35118, 14059, 357, 43, 2257, 44, 8, 290, 12068, 15417,
28403, 357, 45, 19930, 8, 357, 818, 283, 5045, 9045, 292, 709, 737, 198,
220, 220, 220, 220, 32900, 25, 5765, 281, 406, 2257, 44, 284, 4331, 5474,
4979, 290, 466, 12068, 15417, 28403, 319, 8959, 11012, 3807, 8088, 13,
198, 6747, 1596, 25, 1511, 25, 1314, 12, 1558, 25, 405, 25, 357, 23907,
864, 8, 11160, 12, 27195, 12342, 290, 32172, 13326, 357, 7250, 737, 198,
220, 220, 220, 220, 32900, 25, 3082, 601, 4263, 1262, 11160, 12, 27195,
12342, 11, 290, 13946, 41270, 2272, 351, 471, 33767, 13]
```

TEXT TOKEN IDS

https://platform.openai.com/tokenizer

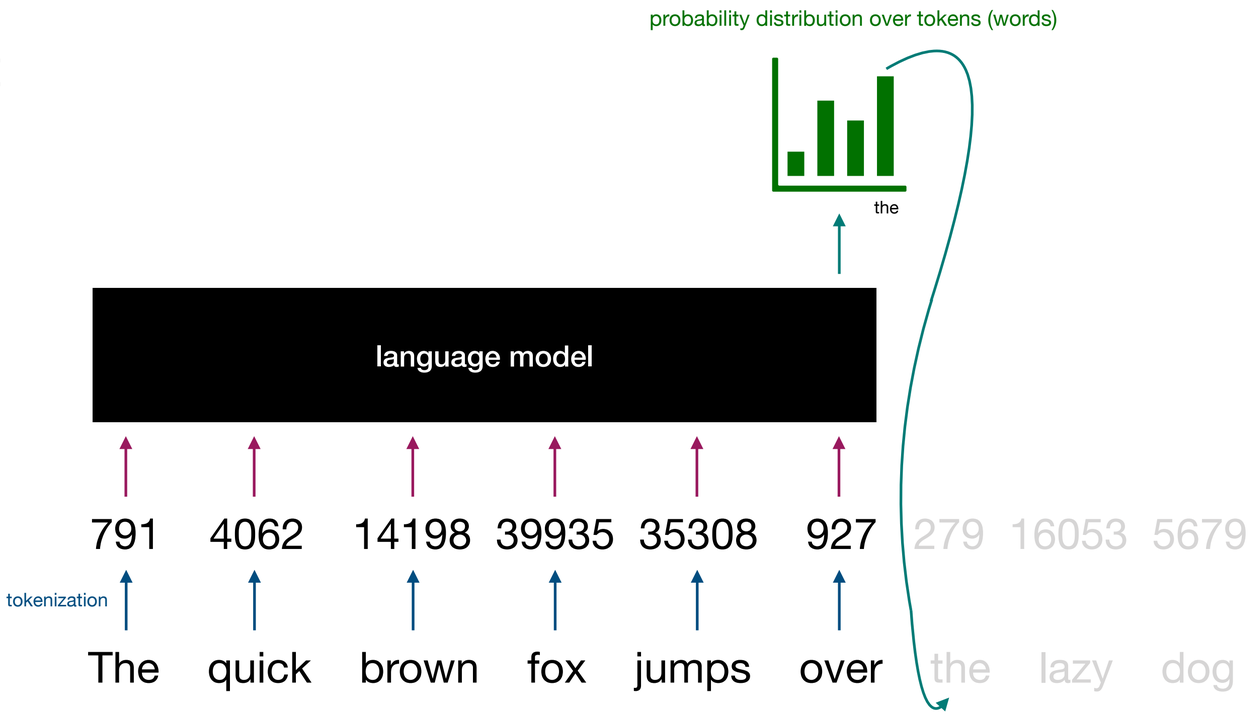
- Embeddings — every token is a vector in a multidimensional space
  (Word2Vec)



Operations over vectors: `king - man + woman ~= queen`

Language modeling: generic picture

- Process the first t words
- Predict the probability of the next word $P(x^{(t+1)} \mid x^{(1)}, x^{(2)}, \ldots, x^{(t)})$
- Sample the next word
- Process the first $t+1$ words
- …

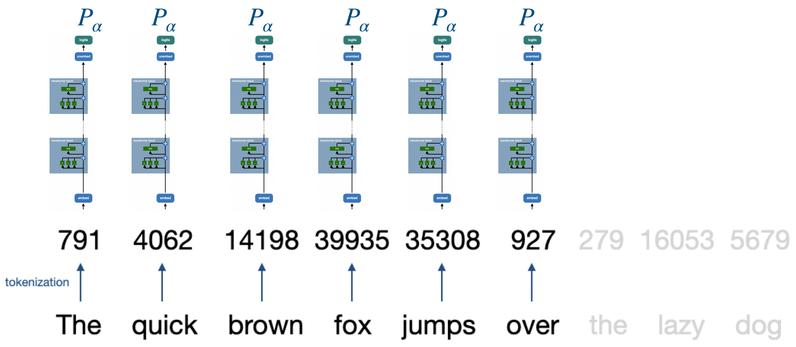


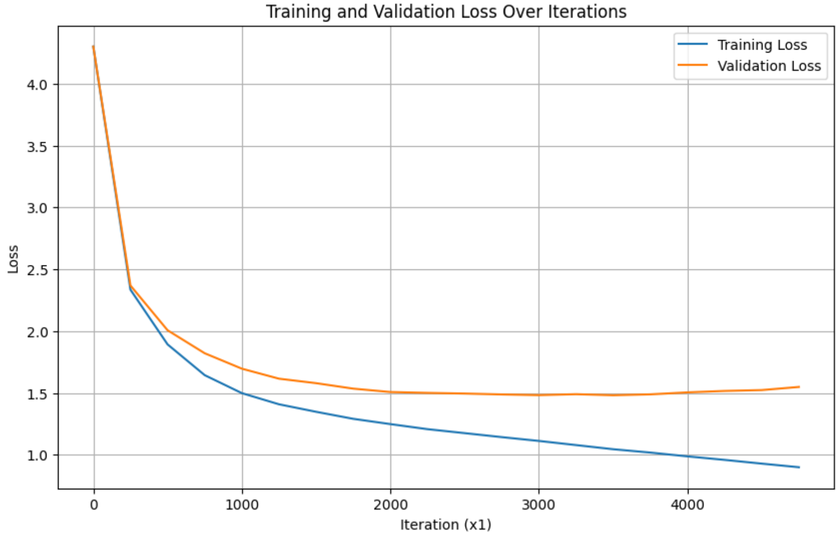
How to train an LLM

- **Self-supervised pretraining**
- Output probabilities over all tokens (words)

$$P_\alpha = softmax(\ell_\alpha) = \frac{\exp(\ell_\alpha)}{\sum_\beta \exp(\ell_\beta)}, \text{ where } \ell_\alpha - \text{logits (raw outputs)}$$

- Cross-Entropy Loss $= -\sum_s \sum_\alpha y_\alpha \log(P_\alpha)$, where y_α are true next tokens (one-hot encoded).
- Typically trained with very large batch sizes (millions of tokens). Highly parallelizable.
- Adam optimizer (or similar). SGD doesn't work for transformers!





Bonus: nanoGPT exercise!

How large are Large Language Models?

- Open weights example: **DeepSeek R1**, 671B parameters
- 671B parameters × 2 bytes per parameter
- For maximum context (32,092 tokens): KV cache requires around 280GB RAM
- Total VRAM Needed: ~1,680GB
- H100 GPU Count Calculation: 1,680GB ÷ 80GB per H100 = 21 GPUs (minimum)
  Including overhead for system operations, tensor parallelism, and other processes: 22-24 H100 GPUs
- Total Estimated Cost: 22 H100s at approximately $30,970 each = ~$681,340

- Try it yourself! https://huggingface.co/deepseek-ai/DeepSeek-R1
- You can try smaller models locally! https://huggingface.co/Qwen/Qwen3-0.6B (use vLLM if you have a GPU)

Further training steps

- Self-supervised training is the first step.

- Instruction Fine-Tuning (IFT / SFT with Instructions):

  - A crucial step for making models more "helpful and harmless" and better at following user commands.
  - Fine-tune on a dataset of (instruction, response) pairs.
  - Teaches the model to respond to prompts in a desired format and style.
  - Example data: "Translate this sentence to French: {sentence}" -> "{French translation}"

- Alignment with Human Preferences:

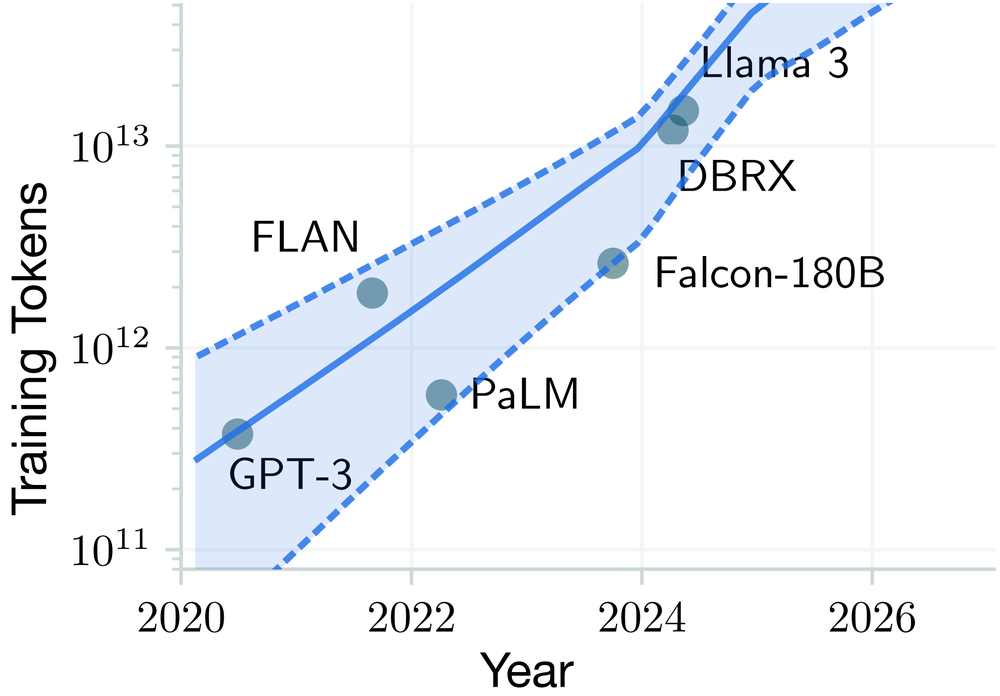
  - Reinforcement Learning from Human Feedback (RLHF)
  - Direct Preference Optimization (DPO)

- Reasoning (RL in verifiable domains, e.g GRPO)

see https://rlhfbook.com/

Success of self-supervise training

- Labeled data is limited
- Unlabeled data is abundant (text, image, video)
- Led to GenAI revolution
- Some recent models were trained on 30 trillion tokens!



source:
2211.04325 "Will we run out of data? Limits of LLM scaling based on human-generated data"

BERT - 3.3B tokens

1810.04805 "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

Self-supervise training: Scaling Laws

Performance predictably improves with scale

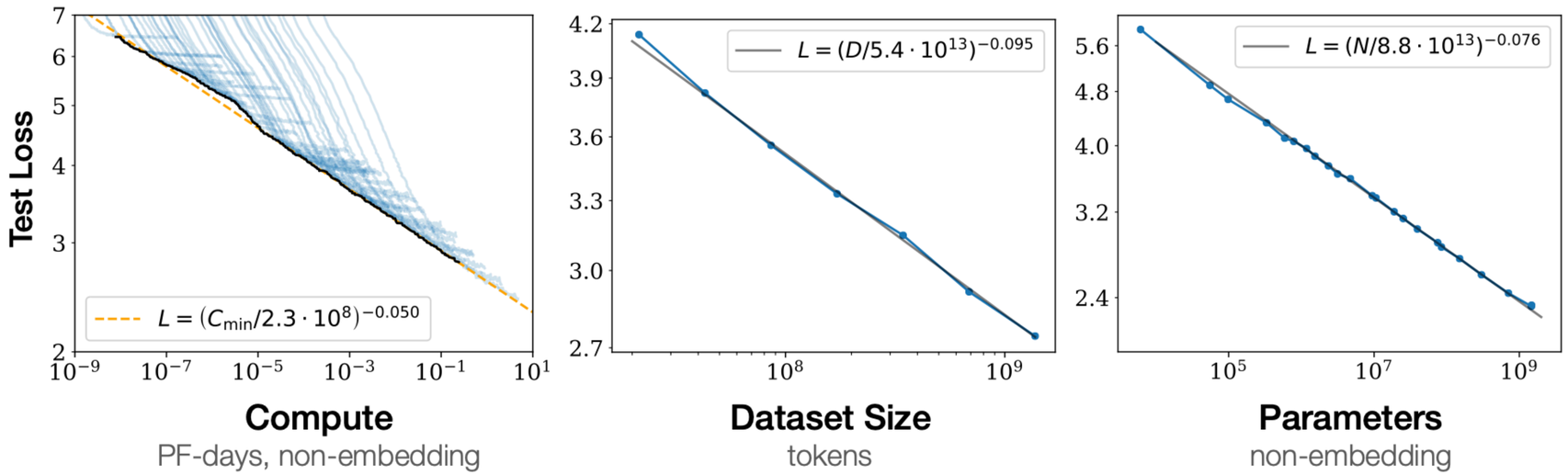


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

https://arxiv.org/pdf/2001.08361
Scaling Laws for Neural Language Models

Success of self-supervise training

Power

Probably the single biggest constraint on the supply-side will be power. Already, at nearer-term scales (1GW/2026 and especially 10GW/2028), power has become the binding constraint: there simply isn't much spare capacity, and power contracts are usually long-term locked-in. And building, say, a new gigawatt-class nuclear power plant takes a decade. (I'll wonder when we'll start seeing things like tech companies buying aluminum smelting companies for their gigawatt-class power contracts.[57])

https://situational-awareness.ai/
Leopold Aschenbrenner, June 2024

MICROSOFT / TECH / SCIENCE

Microsoft wants Three Mile Island to fuel its AI power needs

Photo by Andrew Caballero-Reynolds / AFP via Getty Images

/ Microsoft has signed a 20-year deal to exclusively access 835 megawatts of energy from a nuclear plant.

By Tom Warren, a senior editor and author of *Notepad*, who has been covering all things Microsoft, PC, and tech for over 20 years.

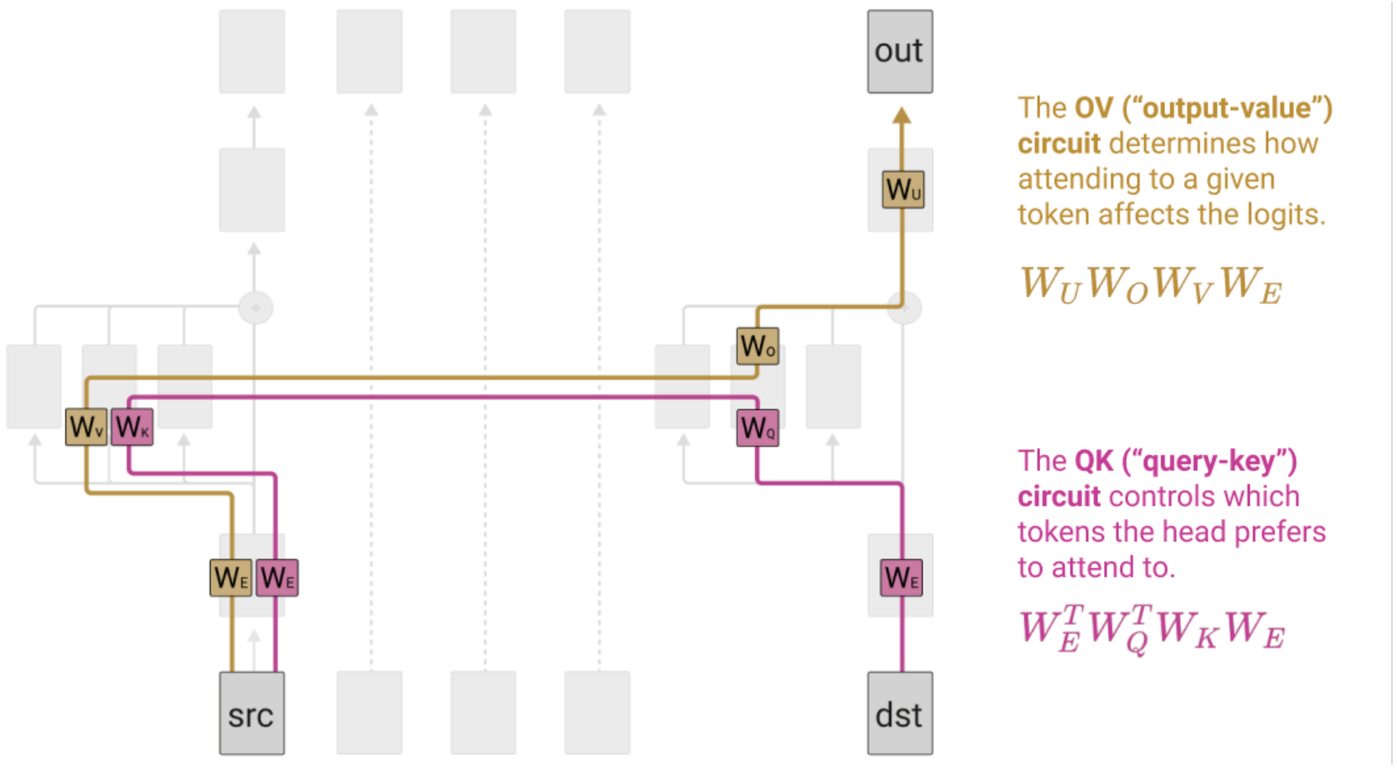
Sep 20, 2024 at 2:23 PM GMT+2

69 Comments (69 New)

https://www.theverge.com/2024/9/20/24249770/

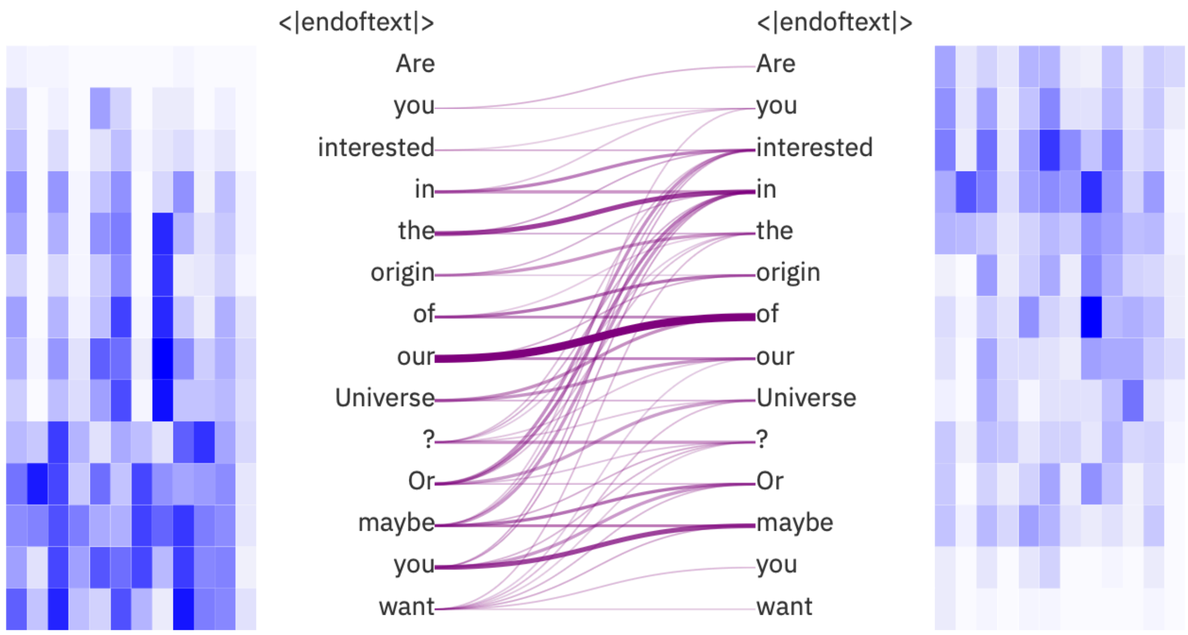
Thank you for your *Attention*

How it works

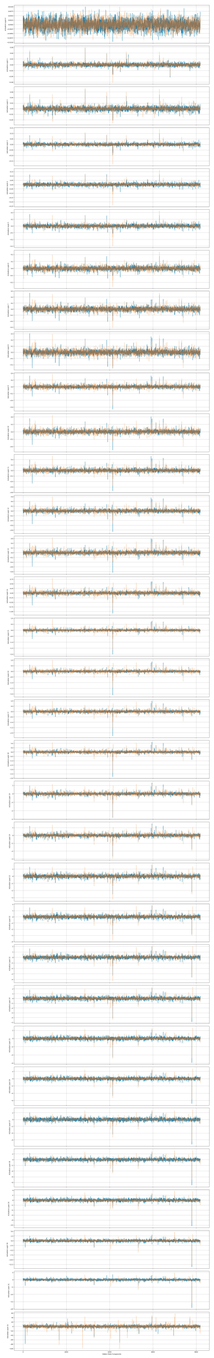


Source https://transformer-circuits.pub/2021/framework/index.html

How it works: attention patterns and activations

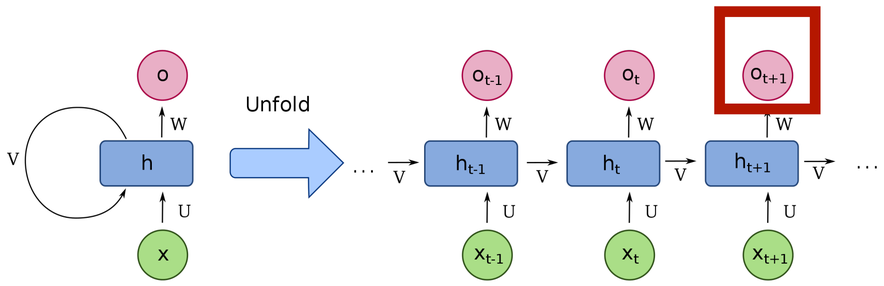


Demo: https://huggingface.co/spaces/simon-clmtd/exbert

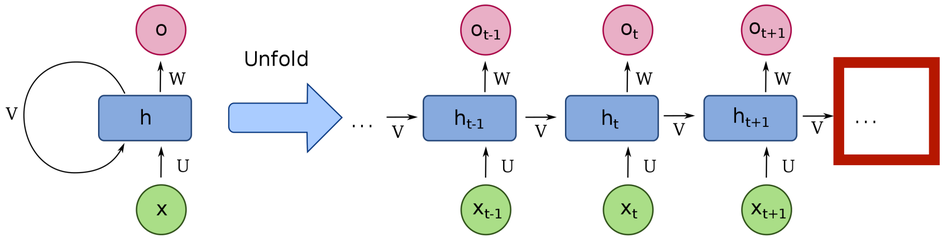


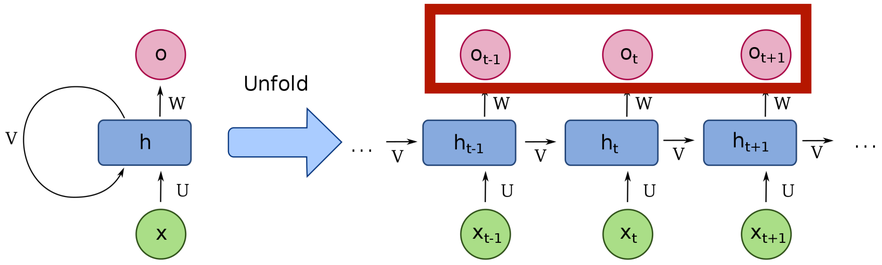
Time Series

- Predicting the next value

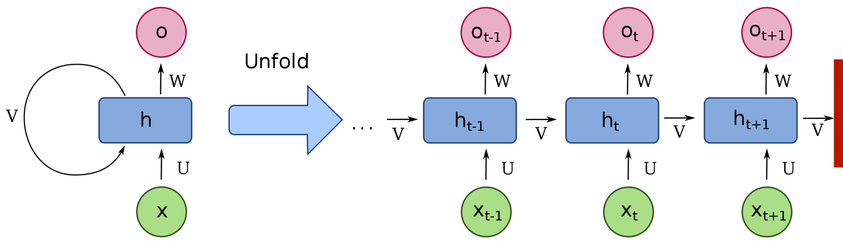


- Global properties of time series





Time Series

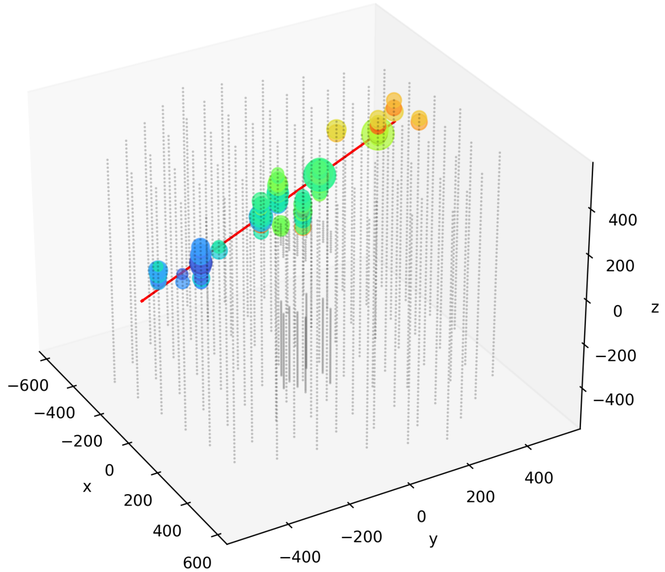


Neutrino direction in IceCube

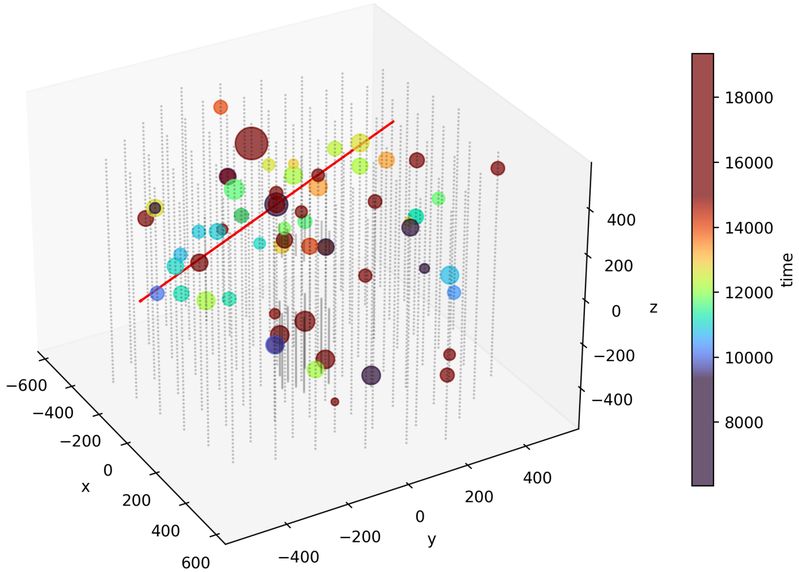
- `event_id` (`int`): the event ID. Saved as the index column in parquet.
- `time` (`int`): the time of the pulse in nanoseconds in the current event time window. The absolute time of a pulse has no relevance, and only the relative time with respect to other pulses within an event is of relevance.
- `sensor_id` (`int`): the ID of which of the 5160 IceCube photomultiplier sensors recorded this pulse.
- `charge` (`float32`): An estimate of the amount of light in the pulse, in units of photoelectrons (p.e.). A physical photon does not exactly result in a measurement of 1 p.e. but rather can take values spread around 1 p.e. As an example, a pulse with charge 2.7 p.e. could quite likely be the result of two or three photons hitting the photomultiplier tube around the same time. This data has `float16` precision but is stored as `float32` due to limitations of the version of pyarrow the data was prepared with.
- `auxiliary` (`bool`): If `True`, the pulse was not fully digitized, is of lower quality, and was more likely to originate from noise. If `False`, then this pulse was contributed to the trigger decision and the pulse was fully digitized.

Example event from the dataset:
(azimuth = 4.86 rad, zenith = 1.96 rad)

auxiliary == False



auxiliary == True



https://www.kaggle.com/competitions/icecube-neutrinos-in-deep-ice/overview