Sequence Models: Attention, Transformers

CPSC 440/550: Advanced Machine Learning

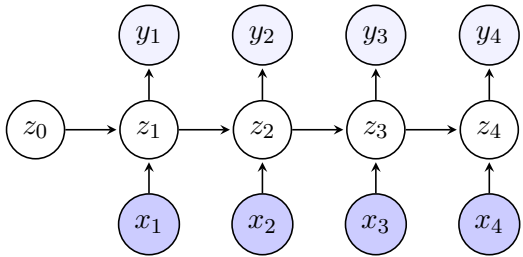
cs.ubc.ca/~dsuth/440/24w2

University of British Columbia, on unceded Musqueam land

2024-25 Winter Term 2 (Jan–Apr 2025)

Last time

- Recurrent neural networks (RNNs): process sentences/whatever in sequence

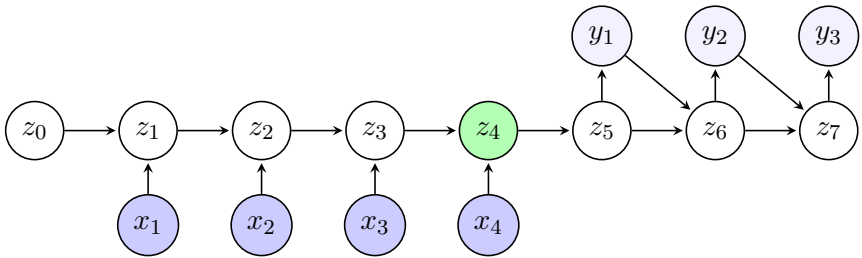


  - Have a hidden state that updates as they "read"
  - Hard to "remember things": dynamics are complicated and state is fixed size
  - Closely related problem: vanishing/exploding gradients

- Approach that helps: long short-term memory (LSTM)
  - Adds "memory cells" and complicated machinery to use them
- Approach that helps: state-space models
  - Fancy math to help remember, but still fixed-size state, limits dynamics

Last time: Sequence-to-Sequence RNNs

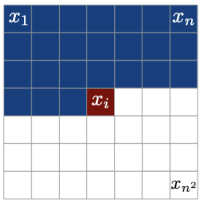
- seq2seq variant gives a way to handle variable-length outputs
- Similar idea for multimodal models (e.g. encode image, decode caption)



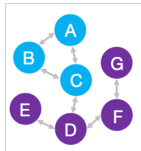
- Problem:
  - Everything about the input needs to end up in one state
  - Easy to "forget" stuff you processed earlier
  - Different parts of output care about different parts of input

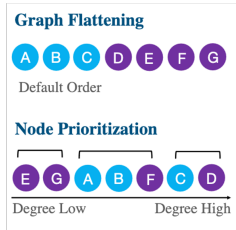
Problems with RNNs

- Hard to "remember" relevant information for long enough
  - Fixed amount of state
- Hard to optimize: vanishing/exploding gradients, big memory usage
- Hard to parallelize: everything depends on everything before
- Not always natural to give an order to some (most?) data types



PixelRNN (2016): looks at pixels one-by-one, order matters a lot

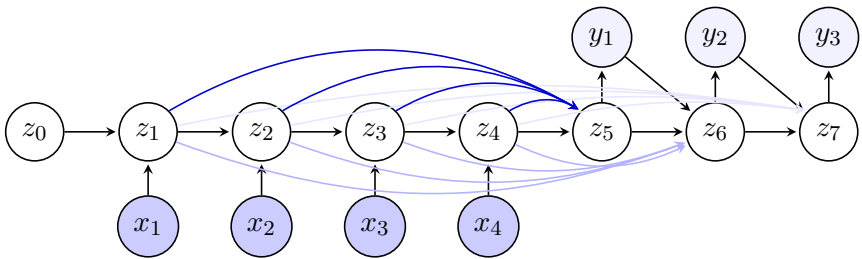




Graph-Mamba (2024): complicated heuristics to process graph data in "right order"

Looking back in history

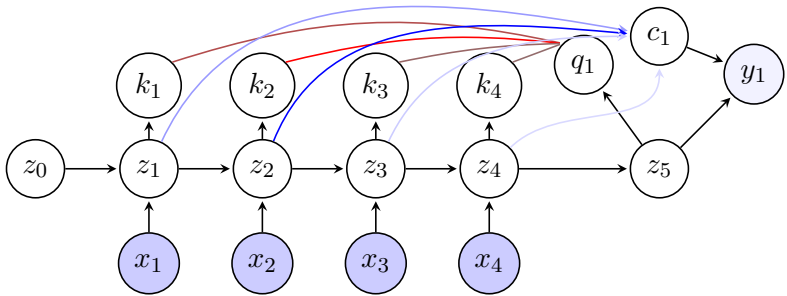
- What if we didn't have to "remember" everything?
- Decoder could have skip connections back to every encoder state



- But: number of connections depends on input length
- And (unlike U-nets), "how" they connect may depend on particular input

Attention

- We can't "look at" everything. Maybe one old state at a time... but which one?
  - Fixed choice like "10 steps ago" might not be the right choice
- Let the model choose, by treating history like a database:
  - Each encoder state gets a key
  - Decoder makes a query, matches against the keys
  - Pass best-matching state's value as context to the decoder



- To be differentiable, we actually use $c = \sum_t \text{score}(k_t, q) z_t$ (nonnegative scores that sum to 1)

How to get and match keys / queries?

- Conceptually, could use whatever computation you want
- In practice: almost always use softmax of scaled dot product

$$\text{score}(k_t, q) = \frac{\exp(k_t \cdot q \,/\, \sqrt{d_{kq}})}{\sum_s \exp(k_s \cdot q \,/\, \sqrt{d_{kq}})}$$

  - Scaling by dimension acts as "temperature," changes sharpness but not order

- These days, keys/queries are almost always a linear transformation of input

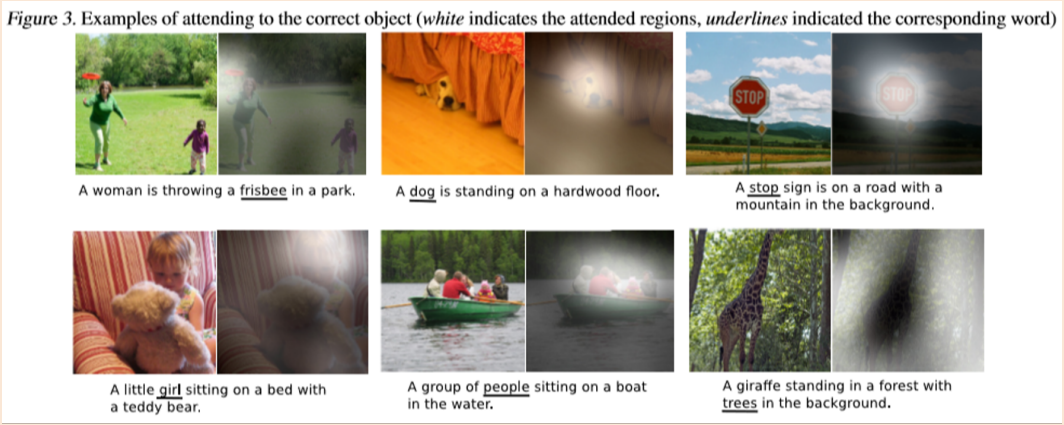
$$k_t = W_K z_t \qquad q_{N+t} = W_Q z_{N+t}$$

  - W_K and W_Q are weight matrices to learn
  - Dimension of the keys/queries is a hyperparameter

Multi-modal attention

bonus!

- Attention for image captioning:



https://arxiv.org/abs/1502.03044

Biological motivation for attention

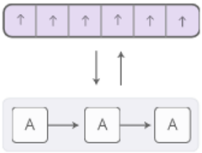
bonus!

- Gaze tracking (https://www.youtube.com/watch?v=QUbiHKucljw)
- Selective attention test (https://www.youtube.com/watch?v=vJG698U2Mvo)
- Change blindness (https://www.youtube.com/watch?v=EARtANyz98Q)
- Door study (https://www.youtube.com/watch?v=FWSxSQsspiQ)

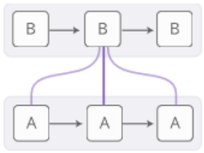
Neural Turing Machine, Neural Programmers

bonus!

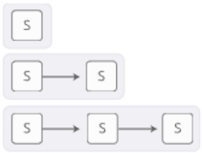
- Many variants of RNNs using different versions of attention
- Survey from the peak of when people did this:
  `https://distill.pub/2016/augmented-rnns`



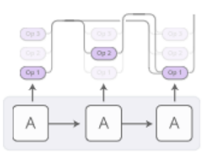
Neural Turing Machines
have external memory that they can read and write to.



Attentional Interfaces
allow RNNs to focus on parts of their input.



Adaptive Computation Time
allows for varying amounts of computation per step.



Neural Programmers
can call functions, building programs as they run.

Outline

More than meets the eye

bonus!

Attention is all you need

A Vaswani, N Shazeer, N Parmar... - Advances in neural ..., 2017 - proceedings.neurips.cc

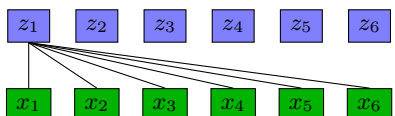
... to attend to **all** positions in the decoder up to and including that position. **We need** to prevent ... **We** implement this inside of scaled dot-product **attention** by masking out (setting to $-\infty$) ...

☆ Save 99 Cite Cited by 172734 Related articles All 73 versions »

- Already one of the most-cited papers of all time
- The T in GPT; also the basis of basically every other large language model
- Also in most of the current best models for vision, protein folding, graphs, ...

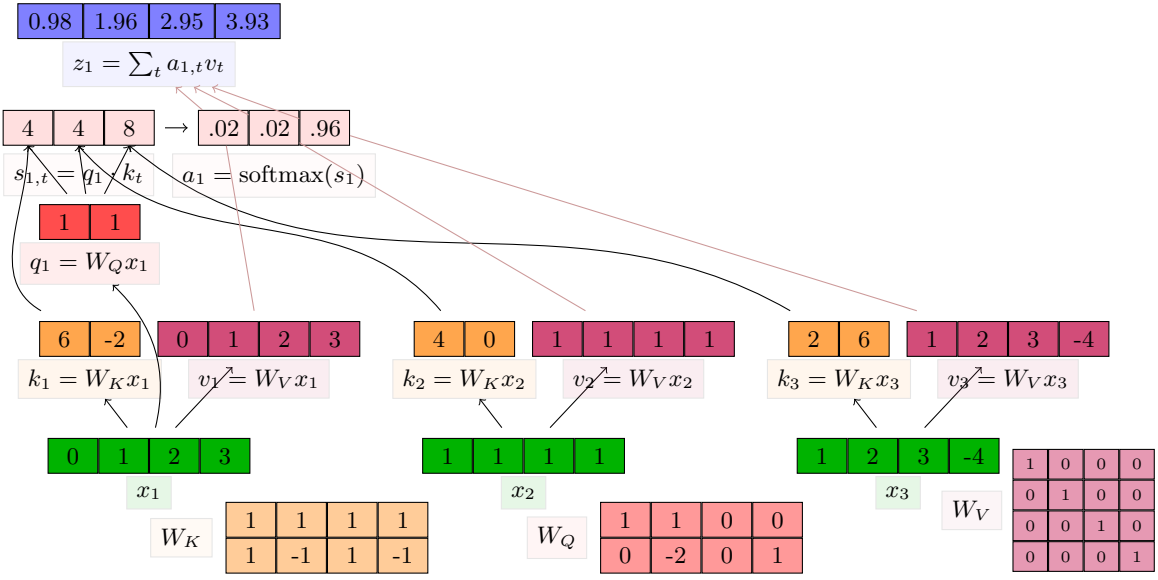
Transformer architecture

- "Attention is all you need": ditch the recurrent part of RNNs
- Self-attention layers: have a sequence of representations
- Each representation depends on all other words in the input

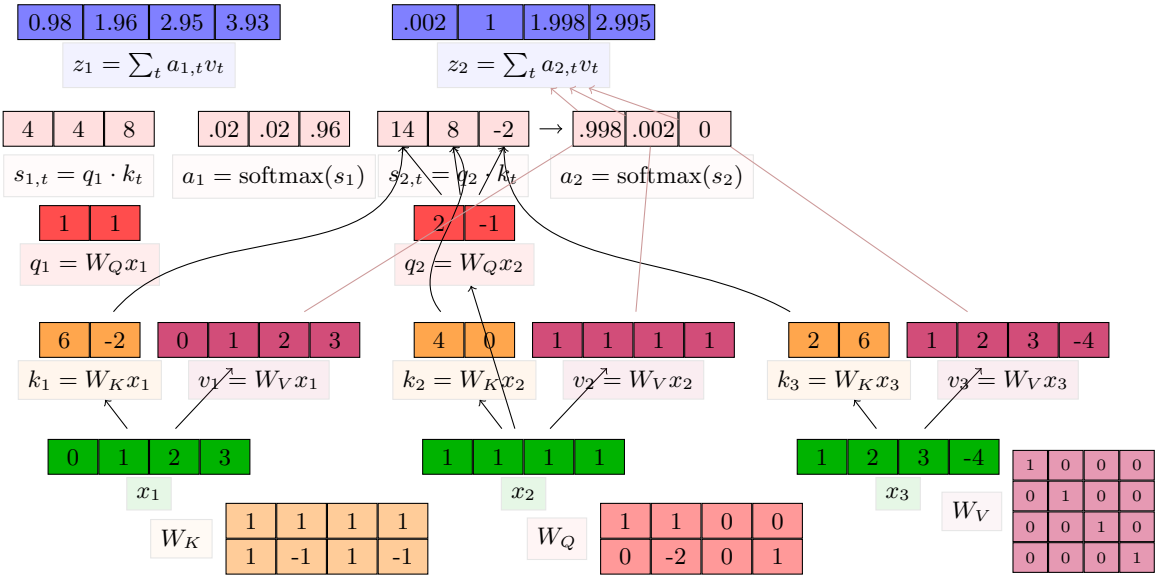


- Instead of passing forward the old state, pass a computed value
- Mapping sequences (x_t) to (z_t) is the core structure of e.g. part-of-speech tagging
  - We'll also massage setups like seq2seq into this shape, too!

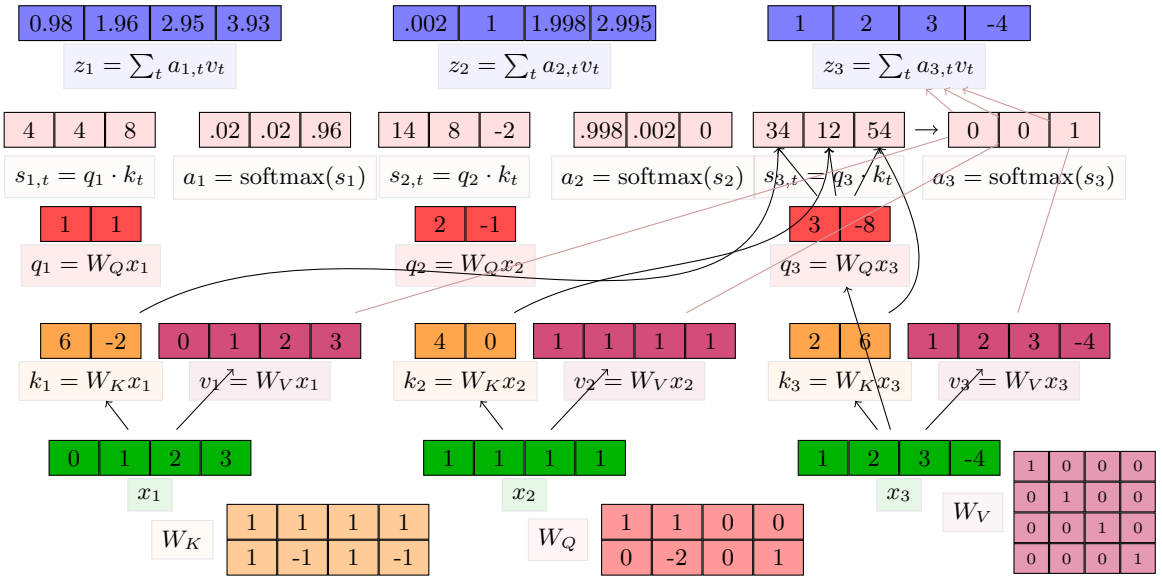
Self-attention layer



Self-attention layer



Self-attention layer



Position encodings

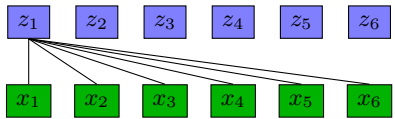
- RNNs see sequences in order; CNNs have order built-in
- But attention mechanisms "look everywhere"
  - Big advantage... except they don't get to see the order of the sequence at all!
- To tell where a token is in the sequence, we use position encodings
- Concatenate features for t onto the input features x_t (or weirder schemes)
- Original paper uses trigonometric features of the position t

$$\left[\sin(t/10\,000^{2/d}) \quad \cos(t/10\,000^{2/d}) \quad \sin(t/10\,000^{4/d}) \quad \cos(t/10\,000^{4/d}) \quad \cdots\right]$$

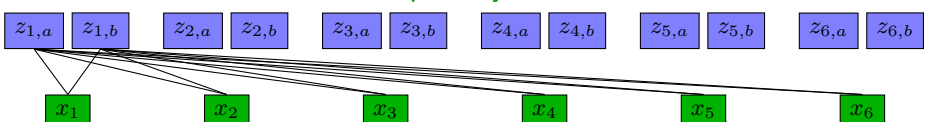
- Later work often learns them: fixed vector for position 1, position 2, ...
- Many variations on the exact scheme, but they mostly perform roughly similarly

Multi-head attention

- A self-attention layer maps a sequence of inputs to a sequence of outputs



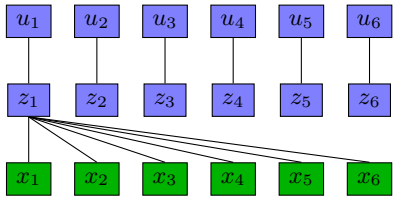
- What if z_4 should depend on both x_1 and x_6 in "different ways"?
- Could carefully make sure that the query-key dot products line up similarly...
  - Would get out $0.44v_1 + 0.53v_6 +$ rest
- Or we could look at both of them separately with multi-head attention



- Run more than one self-attention layer (with separate params), concatenate results

Introducing nonlinearity

- Each output of a self-attention layer is $z_t = \sum_{s=1}^{T} a_{t,s}(W_V x_s)$
  - Actually, Transformers use a residual connection here: $z_t = x_t + \sum_{s=1}^{T} a_{t,s} W_V x_s$
  - The $a_{t,s}$ are nonlinear, but z_t is a convex combination of $\{x_t, W_V x_1, \ldots, W_V x_T\}$
- We also want to be able to introduce more "fundamental" nonlinearity!
- Simple scheme: add per-token MLP layers



- Each $u_t = \mathrm{MLP}(z_t)$; also use a residual connection, so

$$\mathrm{MLP}(z) = z + (W_2 \,\mathrm{ReLU}(W_1 z + b_1) + b_2)$$

  - Same parameters across positions; a convolutional layer with filter width 1

Layer normalization

- Also use layer normalization after each layer (MLP and self-attention)
- Computes the mean, std for each layer's activations separately for each input

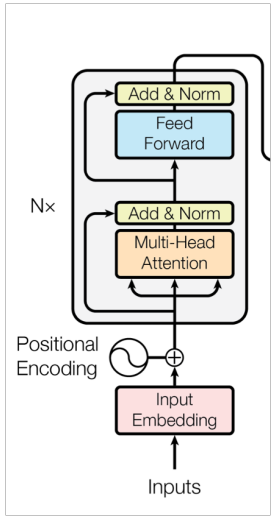
$$\mu_\ell^{(i)} = \frac{1}{d_\ell}\sum_{j=1}^{d_\ell} z_{\ell,j}^{(i)} \qquad \sigma_\ell^{(i)} = \sqrt{\frac{1}{d_\ell}\sum_{i=1}^{d_\ell}\left(z_{\ell,j}^{(i)} - \mu_\ell^{(i)}\right)^2 + \varepsilon}$$

$$\text{LayerNorm}\left(z_\ell^{(i)}\right)_j = \gamma_{\ell,j}\frac{z_{\ell,j}^{(i)} - \mu_\ell^{(i)}}{\sigma_\ell^{(i)}} + \beta_{\ell,j}$$

- γ_ℓ, β_ℓ are learned scale / shifts; initialized to 1, 0
- Makes sense with wide layers (terrible idea if d_ℓ is 1!)
- Kind of like batch norm but avoids some of its issues
- something something internal covariate shift

Transformer encoder

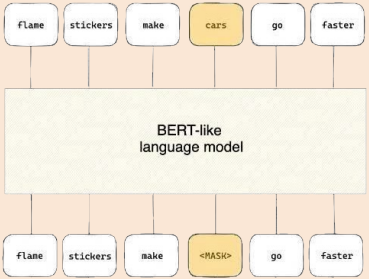
- Each (multi-head self-attention, MLP) chunk is a Transformer block/layer
- Repeat these a bunch of times
- Final result maps a sequence of features to a sequence of features
  - Kind of like an RNN (or a convnet), but the connectivity pattern is different
- Can use this architecture directly to solve per-token prediction problems
  - Just put a per-token prediction at the end, like we did in RNNs



BERT

bonus!

- Bidirectional Encoder Representation from Transformers
- Uses an encoder-only architecture to map tokens to token-level features
- Pre-trained mostly with masked language modeling:



https://www.linkedin.com/posts/kartikeybartwal_masked-language-modeling-akin-to-word-puzzles-activity-7185299798697140224-u6JJ

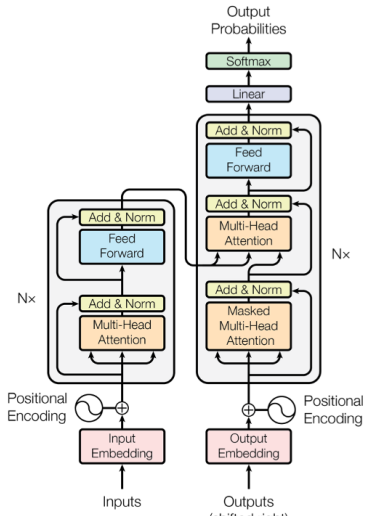
- Can then fine-tune for a particular task or use embeddings elsewhere
- Standard approach these days for content moderation, entity retrieval, ...
  - `https://huggingface.co/blog/modernbert` is recent variant

seq2seq

- How to handle seq2seq-type problems?
- Usually frame it as next-token prediction:
  - `I like the green cat. [BOS]` $\Longrightarrow$ `J'aime`
  - `I like the green cat. [BOS] J'aime` $\Longrightarrow$ `le`
  - `I like the green cat. [BOS] J'aime le` $\Longrightarrow$ `chat`
  - `I like the green cat. [BOS] J'aime le chat` $\Longrightarrow$ `vert`
  - `I like the green cat. [BOS] J'aime le chat vert` $\Longrightarrow$ `.`
  - `I like the green cat. [BOS] J'aime le chat vert.` $\Longrightarrow$ `[EOS]`
- How to predict? Just do a linear layer + softmax on last token's embedding
- Doing this, we'd need to re-run on the full sequence at each step

Transformer decoder

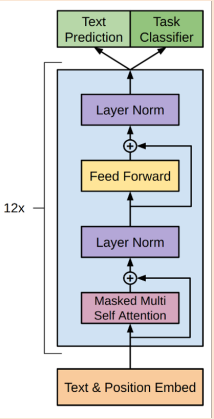
- Transformers have an encoder-decoder split
- Encoder is as before: processes `I like the green cat.`, is same for each output token
- Decoder gets `J'aime le` and wants to predict `chat`
- In each decoder block:
  - Self-attention on `J'aime le`
  - Cross-attention to those and encoder outputs
  - Feed-forward layer as before
- Can train decoder steps in parallel with masked attention
  - Output 3 can look at outputs 1 and 2, but not 3, 4, ...
  - Just "remove arrows" in the query-key dot products



GPT: Generative Pre-trained Transformer

bonus!

- GPT (1): decoder-only but otherwise basically exactly as before



3.1 Unsupervised pre-training

Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$, we use a standard language modeling objective to maximize the following likelihood:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \quad (1)$$

where k is the size of the context window, and the conditional probability P is modeled using a neural network with parameters Θ. These parameters are trained using stochastic gradient descent [51].

In our experiments, we use a multi-layer *Transformer decoder* [34] for the language model, which is a variant of the transformer [62]. This model applies a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens:

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \texttt{softmax}(h_n W_e^T) \end{aligned} \quad (2)$$

where $U = (u_{-k}, \ldots, u_{-1})$ is the context vector of tokens, n is the number of layers, W_e is the token embedding matrix, and W_p is the position embedding matrix.

3.2 Supervised fine-tuning

After training the model with the objective in Eq. 1, we adapt the parameters to the supervised target task. We assume a labeled dataset $\mathcal{C}$, where each instance consists of a sequence of input tokens, $x^1, \ldots, x^m$, along with a label y. The inputs are passed through our pre-trained model to obtain the final transformer block's activation h_l^m, which is then fed into an added linear output layer with parameters W_y to predict y:

$$P(y | x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y). \quad (3)$$

https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-2

bonus!

2.3. Model

We use a Transformer (Vaswani et al., 2017) based architecture for our LMs. The model largely follows the details of the OpenAI GPT model (Radford et al., 2018) with a few modifications. Layer normalization (Ba et al., 2016) was moved to the input of each sub-block, similar to a pre-activation residual network (He et al., 2016) and an additional layer normalization was added after the final self-attention block. A modified initialization which accounts for the accumulation on the residual path with model depth is used. We scale the weights of residual layers at initialization by a factor of $1/\sqrt{N}$ where N is the number of residual layers. The vocabulary is expanded to 50,257. We also increase the context size from 512 to 1024 tokens and a larger batchsize of 512 is used.

https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

GPT-3

bonus!

We use the same model and architecture as GPT-2 [RWC+19], including the modified initialization, pre-normalization, and reversible tokenization described therein, with the exception that we use alternating dense and locally banded sparse attention patterns in the layers of the transformer, similar to the Sparse Transformer [CGRS19]. To study the dependence

https://arxiv.org/abs/2005.14165

GPT-4 and on

bonus!

> This report focuses on the capabilities, limitations, and safety properties of GPT-4. GPT-4 is a Transformer-style model [39] pre-trained to predict the next token in a document, using both publicly available data (such as internet data) and data licensed from third-party providers. The model was then fine-tuned using Reinforcement Learning from Human Feedback (RLHF) [40]. Given both the competitive landscape and the safety implications of large-scale models like GPT-4, this report contains no further details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar.

https://arxiv.org/pdf/2303.08774.pdf

- Based on leaks/etc, seems to be pretty similar to GPT-3 but bigger
- Also incorporating tricks like mixture-of-experts, etc
- Open-weights models (Llama, DeepSeek, ...) are also all pretty similar

Transformers vs RNNs/state space models

- RNNs/state space models:
  - Process things one at a time; order is "built in" and easy
  - Hard to "remember things" from long ago
- Transformers:
  - Easy to "remember things": just go back and look at it
  - Doesn't have a built-in order; need to hack it with position features

Positional Description Matters for Transformers Arithmetic

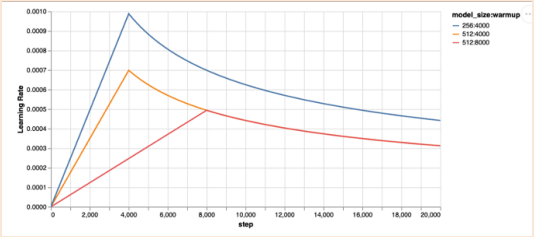
Ruoqi Shen, Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, Yuanzhi Li, Yi Zhang

Transformers, central to the successes in modern Natural Language Processing, often falter on arithmetic tasks despite their vast capabilities --which paradoxically include remarkable coding abilities. We observe that a crucial challenge is their naive reliance on positional information to solve arithmetic problems with a small number of digits, leading to poor performance on larger numbers. Herein, we delve deeper into the role of positional encoding, and propose several ways to fix the issue, either by

Other tricks

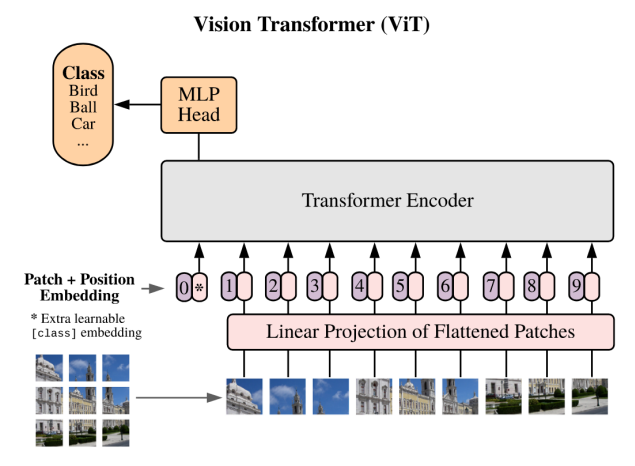
bonus!

- Weight decay / L2 regularization
- Dropout
- Label smoothing
  - Make “true labels” 0.9 probability instead of 1
  - Penalizes wrong predictions a little less
  - Can help discourage overconfidence
- Optimized with Adam (usually AdamW)
  - With a weird learning rate schedule...



Vision transformers

bonus!

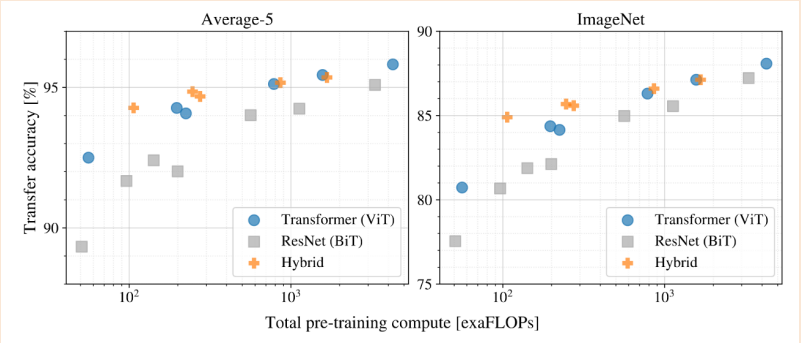


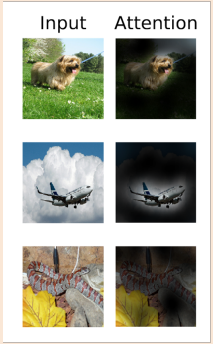
https://arxiv.org/abs/2010.11929

Vision transformers

bonus!

- Usually outperform CNNs, if you have enough data





https://arxiv.org/abs/2010.11929

Summary

- Attention allows an RNN decoder to look at previous states
- Self-attention is a direct sequence-level layer
  - Everything depends on everything; layer params determine how
- Combining self-attention layers with per-token MLPs gets you a Transformer
  - Excellent performance on many tasks
- Next time: more about what happens with these huge models