Attention and Transformers

CPSC 440/550: Advanced Machine Learning

cs.ubc.ca/~dsuth/440/23w2

University of British Columbia, on unceded Musqueam land

2023-24 Winter Term 2 (Jan–Apr 2024)

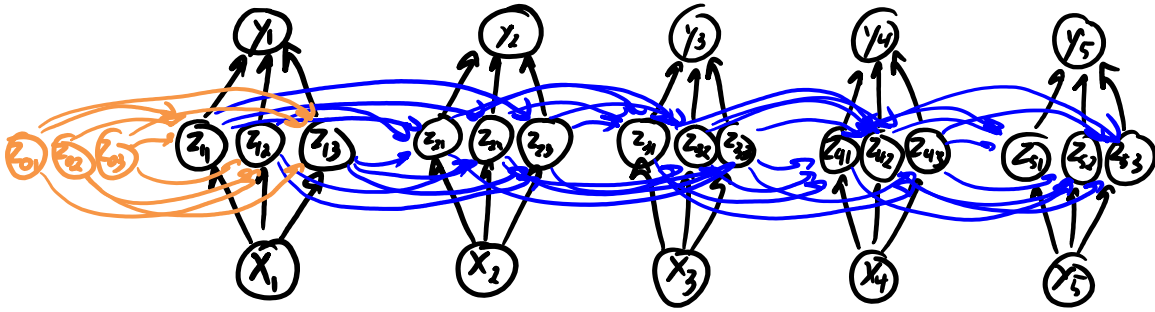
admin

Admin

- A2 is out
  - Due after break, but it's long
- A1 grades are out
  - If something seems wrong, send a regrade request
  - Please don't send regrade requests for small subjective things
  - Please do send regrade requests for something that's wrong
    - Might affect others too
- Quiz grading still ongoing
  - Hit some technical issues…

Last time: RNNs

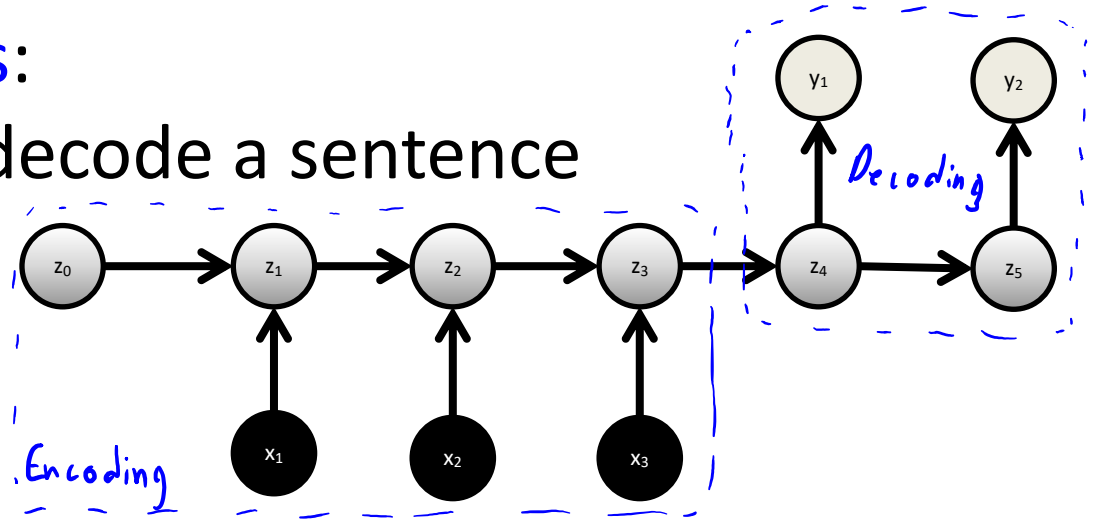
- Process sentences/whatever in sequence
  - Have a "hidden state" that updates as you read
- Closely-related challenges:
  - Remembering things for long enough
    - Exacerbated by state being fixed-size, no matter how much you have to remember
  - Vanishing/exploding gradients



- Approach that helps (doesn't solve): long-short memory (LSTM)
  - Adds "memory cells," and complicated machinery to store/load from memory
  - Similar motivation: state-space models
    - Uses complicated math we didn't cover (and Danica/Alan don't know!)

Last Time: Sequence-to-Sequence RNNs

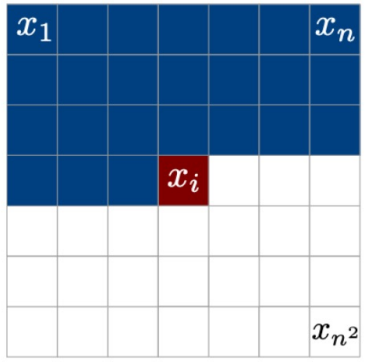
- Sequence-to-sequence:
  - Recurrent neural network for sequences of different lengths
- Similar idea for multimodal models:
  encode an image (e.g. with CNN), decode a sentence

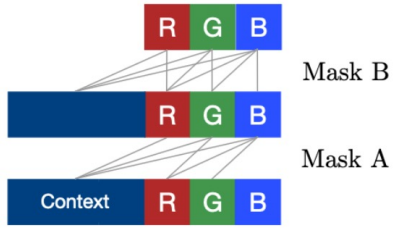


- Problem:
  - All "encoding" information must be summarized by last state (z_3 above)
  - Might "forget" earlier parts of sentence (or middle, for bi-directional)
  - Might want to "re-focus" on parts of input, depending on decoder state

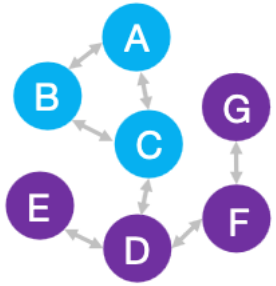
Problems with RNNs

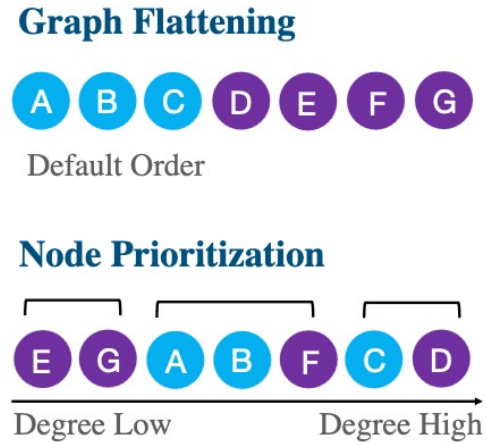
- Hard to "remember" relevant information for long enough
  - Fixed amount of "state" information has to store everything relevant
- Hard to optimize: vanishing/exploding gradients, huge memory usage
- Hard to parallelize: everything depends on everything before it
- Not always very natural for some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





PixelRNN (2016): density estimator for images, looks at each pixel one-by-one



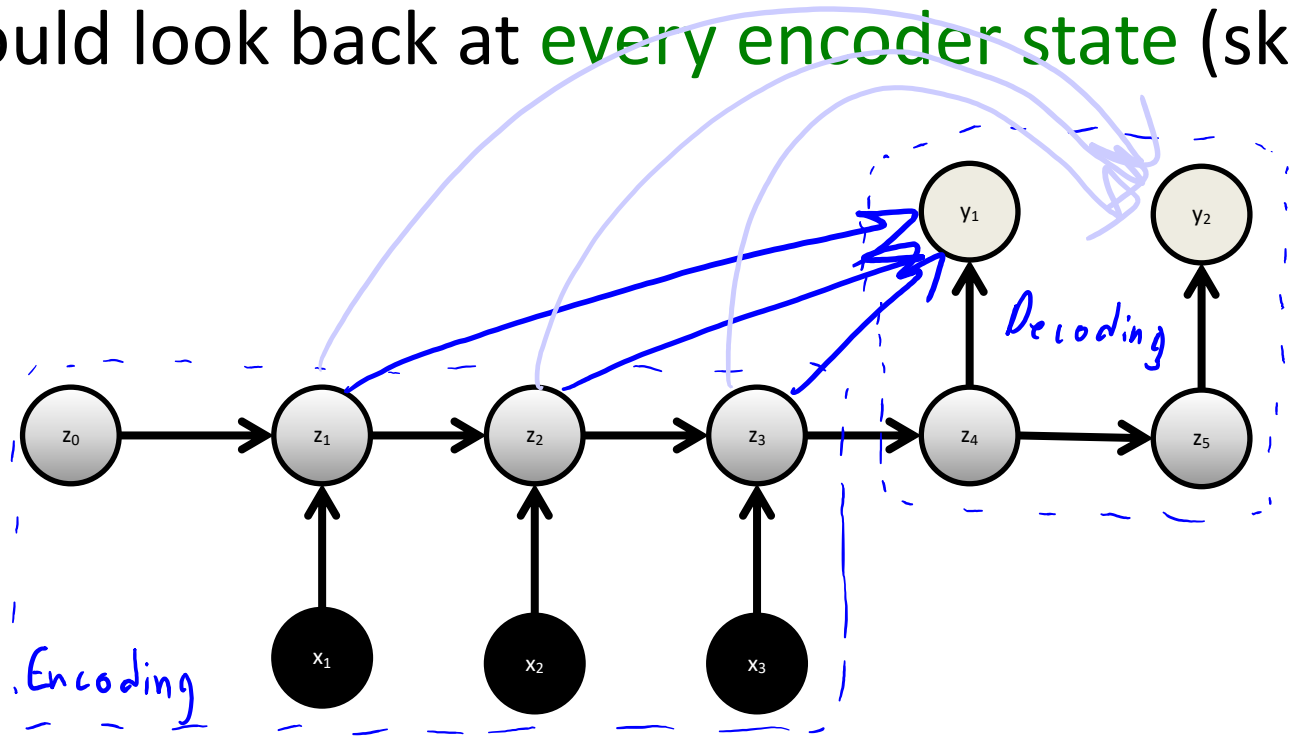


Graph-Mamba (2 weeks ago): complicated heuristics to walk over graph data in the "right order"

- Is looking at data sequentially always the right thing to do?

Looking back in history

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

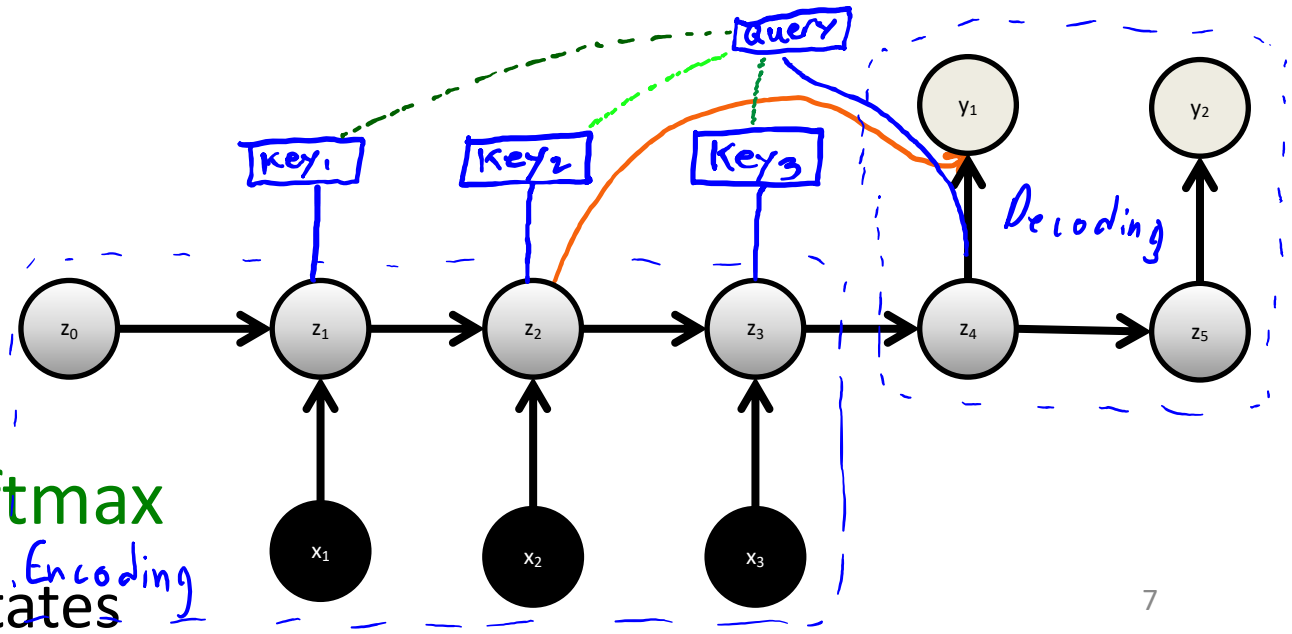


- But: number of weights depends on input length
  - We'd need every input to be the same length!

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention

- We can't "look at" everything in the history
- To start, only look at one old state at a time…but which one?
  - Fixed choice ("always look 10 steps ago") could work…but might not help
- Let the model choose what it wants to pay attention to!
- Treat history like a database:
  - Make keys for each encoder state
  - Decoder makes a query
  - Pick key with biggest "score"
  - Pass as context vector to decoder
- Max isn't differentiable…use softmax
  - Context: convex combination of states



Context vectors from attention

- Each decoder step can look at every encoder state
  - Each decoder step potentially looks at different inputs
- Decoder combines context vector and hidden state as inputs
- "Multi-head attention": several different attention mechanisms (with own queries+keys) at the same time
  - One "subject context", one "verb tense context," one "style context"…

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

How to score a query/key combination?

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Learn how to score

Most common: (scaled) dot product

dot product is big if vecs point in same direction
(and when either/both vectors are big)

(scaling only affects "temperature" of the softmax, not which is max)

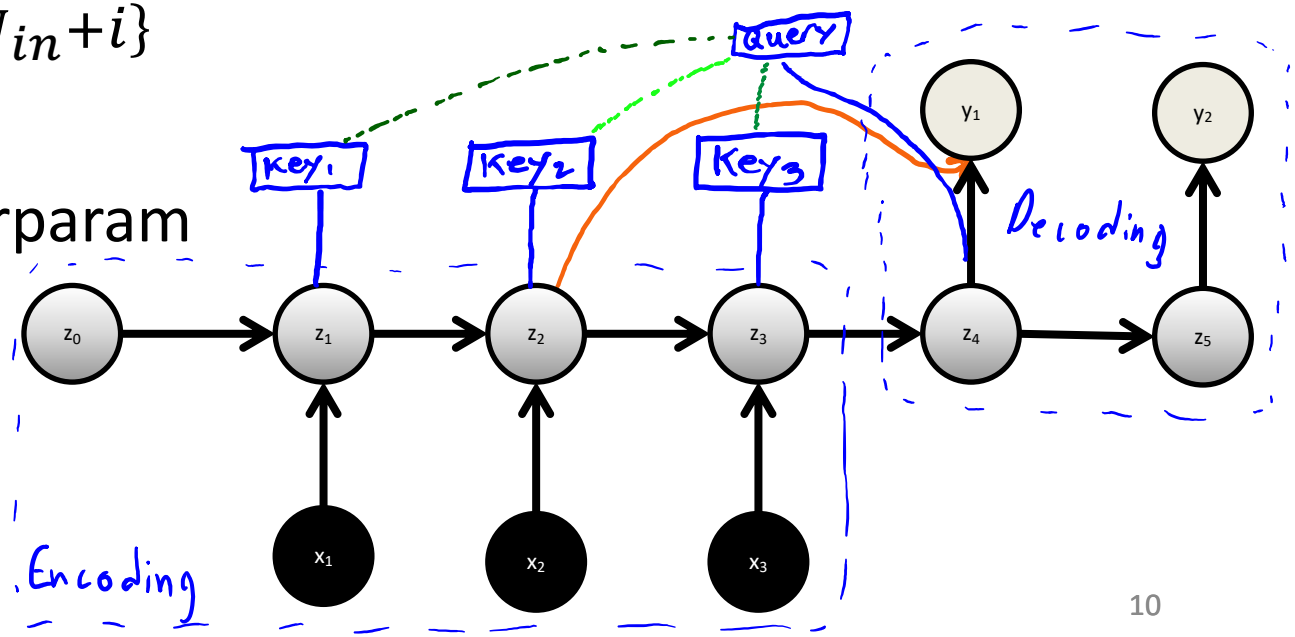
https://lilianweng.github.io/posts/2018-06-24-attention/

How to get keys / queries?

- Computing key/query of a hidden state:
- Conceptually, could use whatever computation you want
  - Some earlier work used a fully-connected layer or two
- These days, almost always just a linear transformation

$$key_i = K\ z_i \quad query_i = Q\ z_{\{N_{in}+i\}}$$

  - K and Q are matrices to learn
  - Dimension of key/query is a hyperparam
    - Needs to match to do inner product

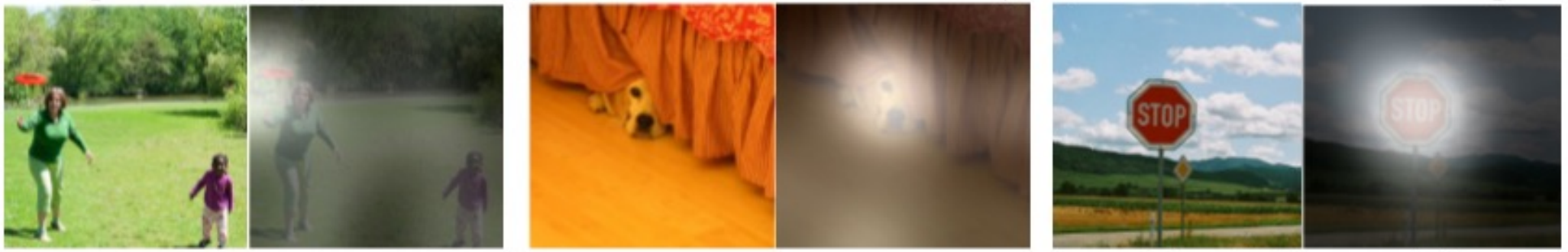


bonus!

Multi-Modal Attention

- Attention for image captioning:

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

https://arxiv.org/pdf/1502.03044.pdf

bonus!

Biological Motivation for Attention

- 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 interesting variations on memory/attention.
  - An out-of-date survey: https://distill.pub/2016/augmented-rnns

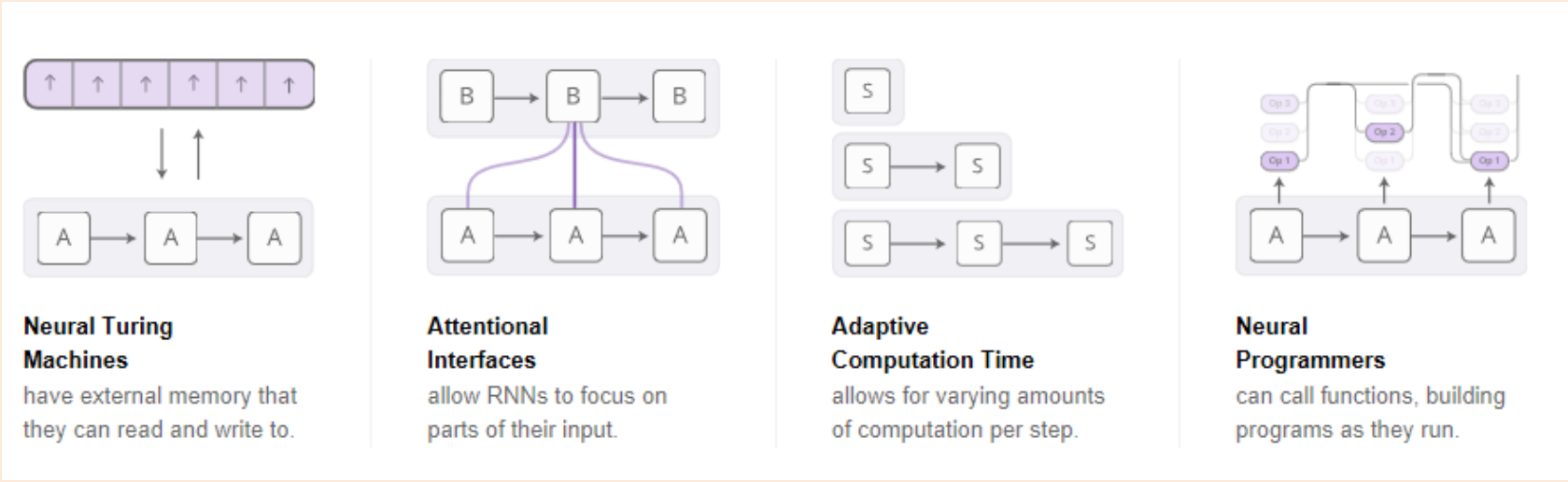
Here is an example of what the system can do. After having been trained, it was fed the following short story containing key events in JRR Tolkien's Lord of the Rings:

Bilbo travelled to the cave.
Gollum dropped the ring there.
Bilbo took the ring.
Bilbo went back to the Shire.
Bilbo left the ring there.
Frodo got the ring.
Frodo journeyed to Mount-Doom.
Frodo dropped the ring there.
Sauron died.
Frodo went back to the Shire.
Bilbo travelled to the Grey-havens.
The End.

After seeing this text, the system was asked a few questions, to which it provided the following answers:

Q: Where is the ring?
A: Mount-Doom
Q: Where is Bilbo now?
A: Grey-havens
Q: Where is Frodo now?
A: Shire

It's probably one of the few technical papers that cite "Lord of the Rings".



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.

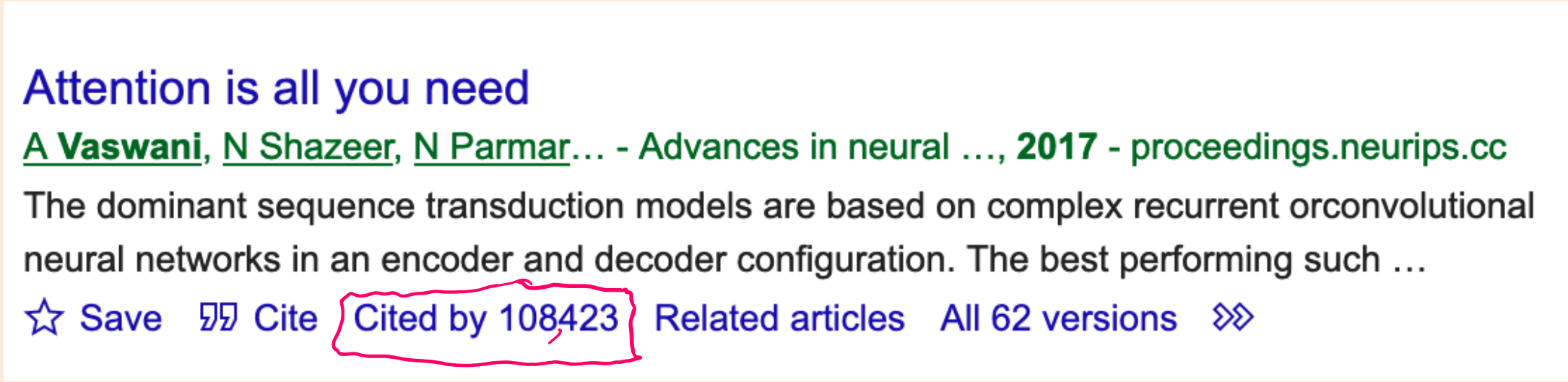
Neural Programmers
can call functions, building programs as they run.

https://www.facebook.com/FBAIResearch/posts/362517620591864

Next Topic: Transformers

bonus!

Transformers are taking over



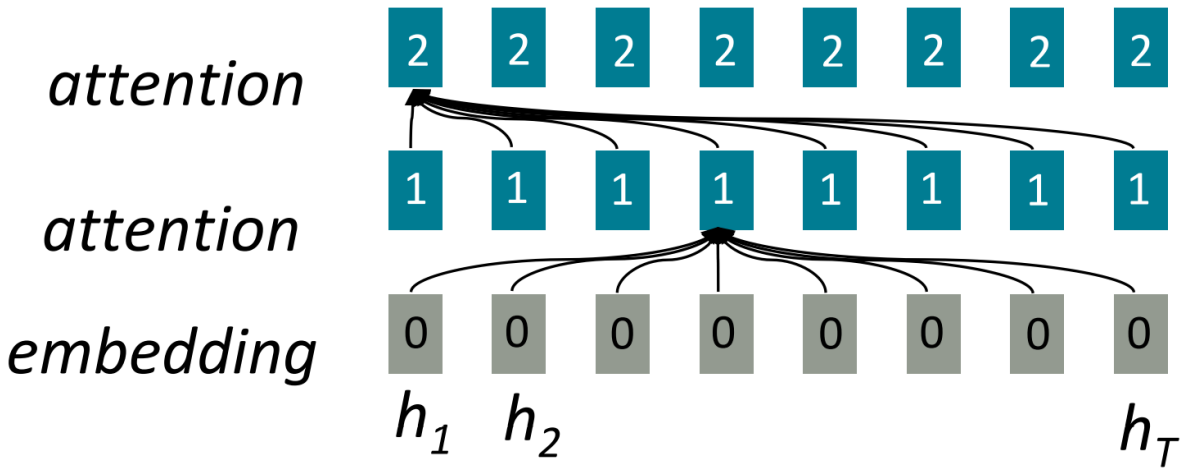
- As of 2014, the most-cited paper *ever* had ~300,000 citations…
  - An important experimental method in biology, from 1970

- The T in GPT – also ~every other LLM (Gemini, Claude, LLAMA, …)
- Also in AlphaFold2, current ~best vision models, graphs, …

https://www.facebook.com/FBAIResearch/posts/362517620591864

Transformer Networks

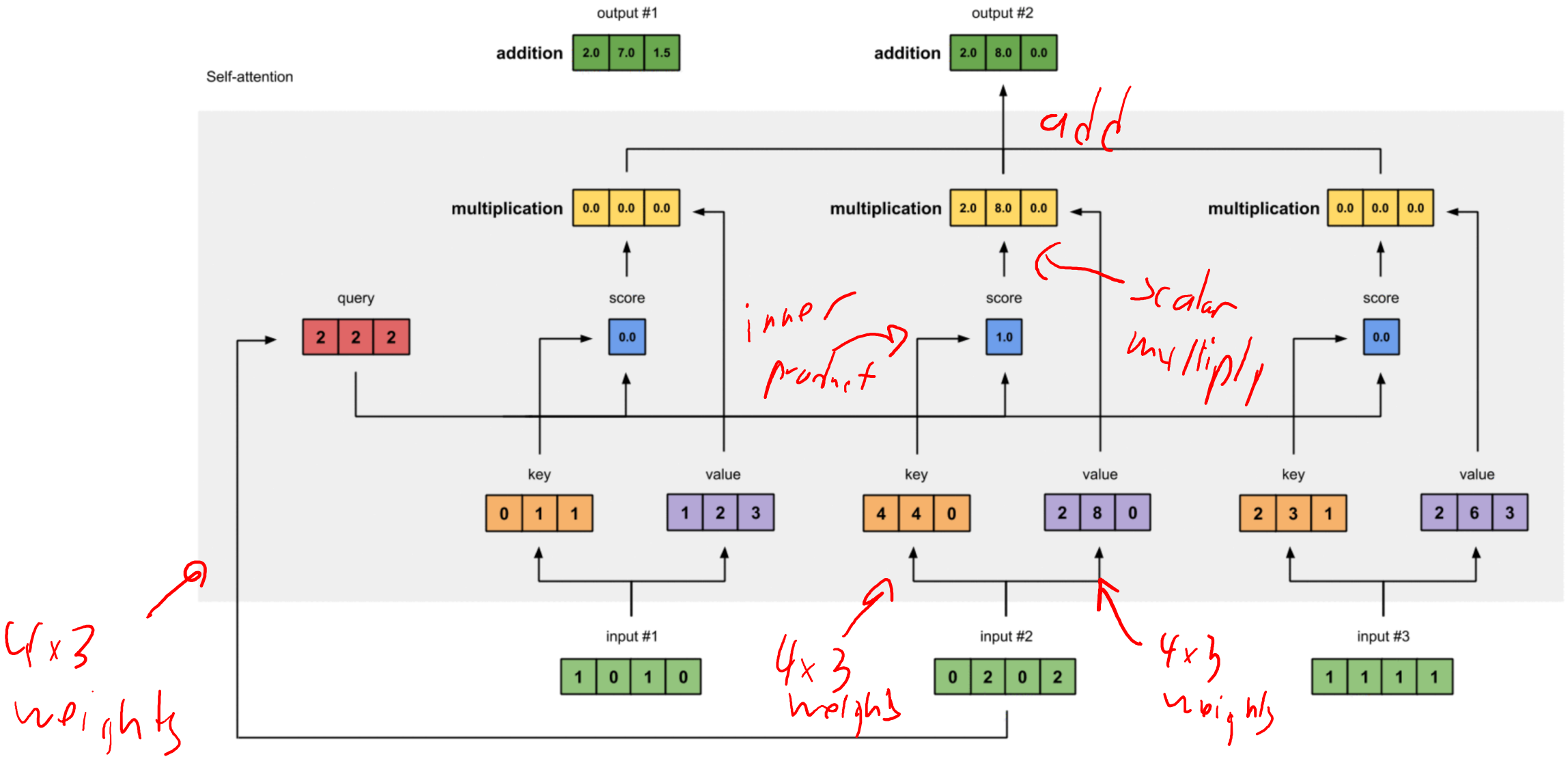
- "Attention is all you need": ditch the recurrent part
- Encoder gets input representations with "self-attention" layers
  - Each word representation attends to all words in the previous layer
  - In addition to query/keys, also values: instead of passing z_j forward, pass $V z_j$



All words attend to all words in previous layer; most arrows here are omitted

- Sequence of representations of words; each depends on *all* other words

Self-attention layer



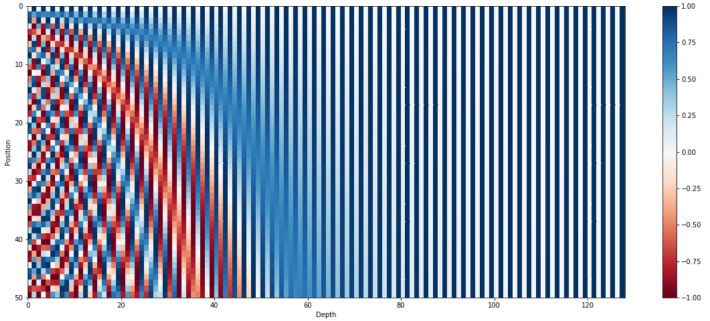
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Position encodings

- RNNs see sequences in order; CNNs have order built-in
- But attention mechanisms "look everywhere" (at everything, all at once)
  - Big advantage…except they don't get to see the order of the sentence!
  - Add position encodings to tell where a word is in the sequence

- Original transformers use trig features of the position

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



- Later work often learns them
  - Feature vector for word 1, word 2, … that you learn as a parameter
- Some variations on exactly what you do, but all ~similar

A couple other tricks

- Layer normalization almost always used in Transformers:
  - Computes the mean and standard deviation across a layer's activations for each input separately
  - Kind of like batch norm,
    internal covariate shift something something
  - Makes sense if you have big layers, avoids some issues of batch norm

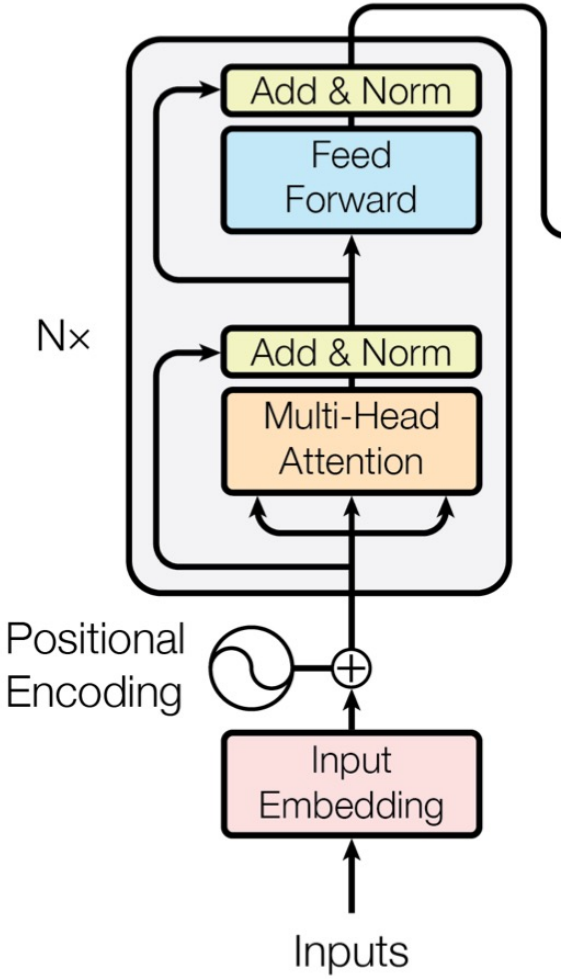
$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

$$\text{LayerNorm}(x) = \frac{\gamma}{\sigma^l}(x - \mu^l) + \beta$$

- Residual connections
  - Makes optimization easier if you don't "need to do anything" (identity map)

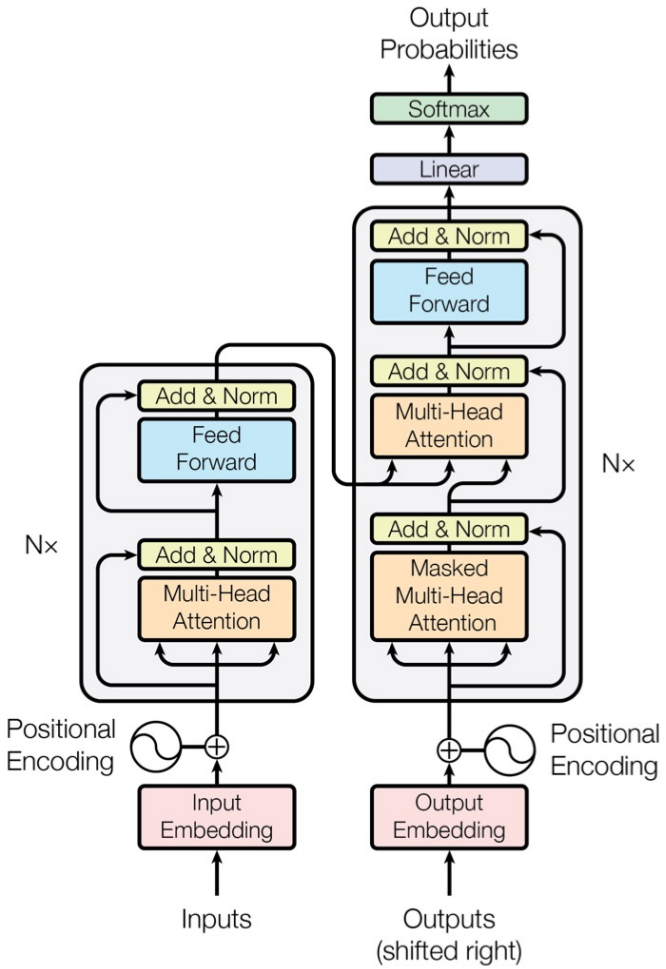
$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

Transformer encoder architecture

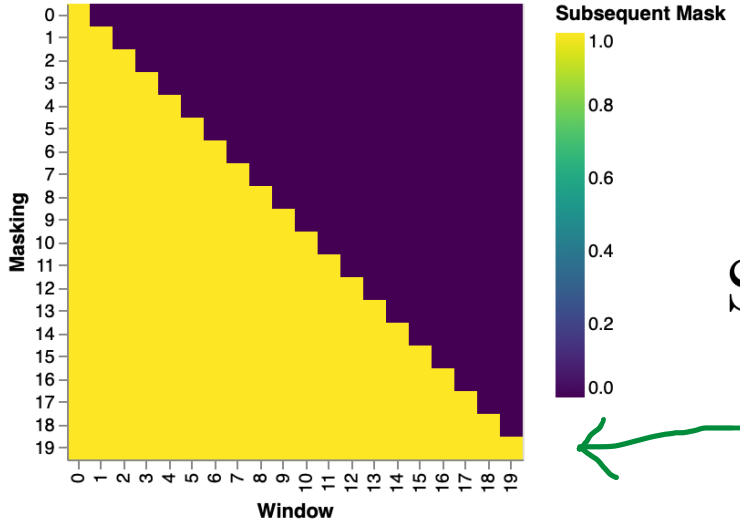


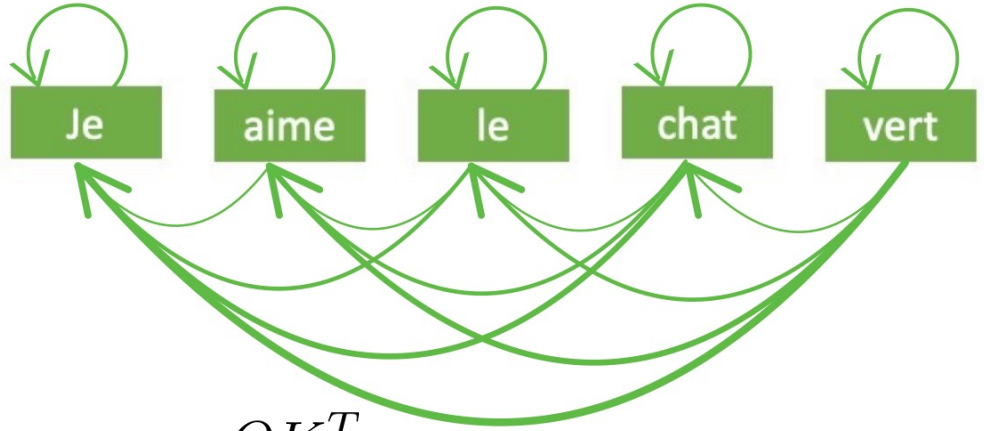
- Also have a simple two-layer ReLU network processing each individual embedding
- Repeat (attention + feed forward) a bunch of times
  - Vaswani et al. used N = 6, and 8 attention heads
- At the end, get an encoding vector for each input
  - Like an RNN! But here everything depends on everything

Transformer decoder



- Uses the same ingredients, with one twist
- "Masked attention" means that words can't attend to the words after them
- Encoder outputs depend on those words anyway!





$$\text{Softmax}(\frac{QK^T}{\sqrt{d_k}} \odot M)V$$

Transformer summary

- Encoder:
  - A bunch of self-attention layers intermixed with fully-connected
  - Maps a sequence of input representations to sequence of outputs
- Decoder:
  - A bunch of self-attention layers intermixed with fully-connected
  - Does big multiclass classifier at the end for each word
  - Uses both encoder embeddings and plain word embeddings of past words
    - Masking structure only on the “plain” part
- Now, you might ask…why do we need both?
  - They look pretty similar, except decoder only looks at past, encoder at all
  - Answer: we probably don’t! GPT, etc are decoder-only

Transformers vs RNNs/state space models

- RNNs/state space models:
  - Process things one at a time: order is very ”built in” and easy
  - Hard to “remember” things for a long time

- Transformers:
  - Avoids needing to “remember” things: just looks at history directly
  - Doesn’t have a built-in order; need to hack it with position features
    - Having the right positional features can be really important!

Positional Description Matters for Transformers Arithmetic

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

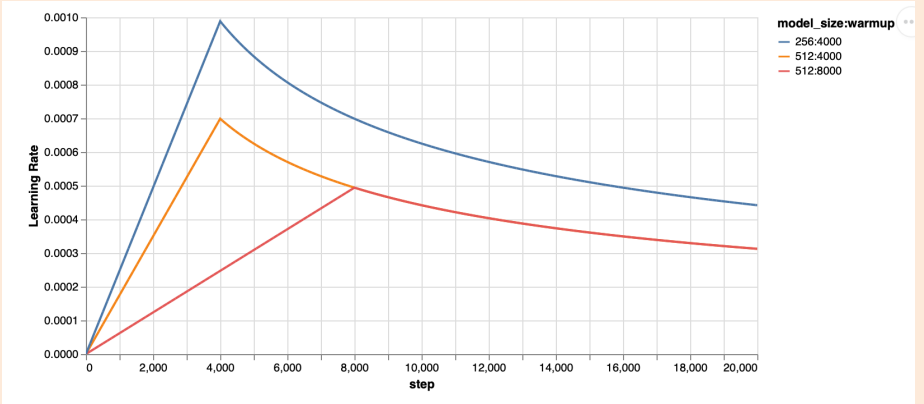
Transformers, central to the successes in modern Natural Language Processing, often falter on arithmetic tasks despite their vast capabilities --which paradoxically include remarkable coding abilities. We observe that a crucial challenge is their naive reliance on positional information to solve arithmetic 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

bonus!

Other tricks

- 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
  - With a weird learning rate schedule (??)
- Beam search to decode
  - Not just an iid sample, does a little search for ”likely samples”

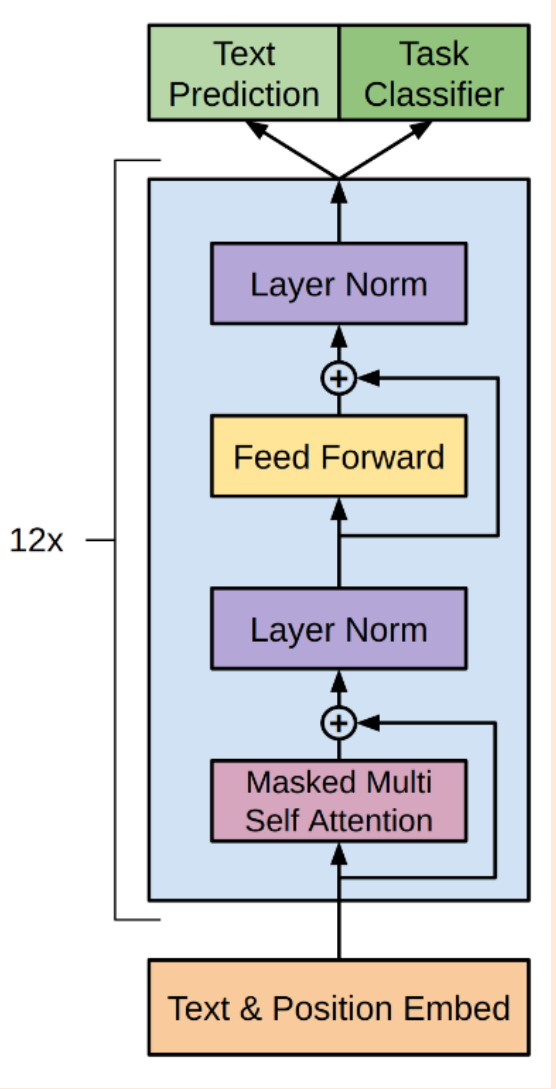
$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$



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

bonus!

GPT: Generative Pre-trained Transformer



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:

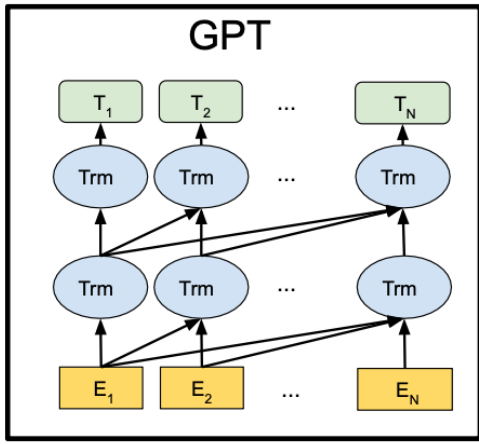
$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \texttt{softmax}(h_n W_e^T) \end{aligned} \quad (2)$$

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

bonus!

GPT-2 and GPT-3

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.

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://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf; https://arxiv.org/abs/2005.14165

bonus!

GPT-4

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.

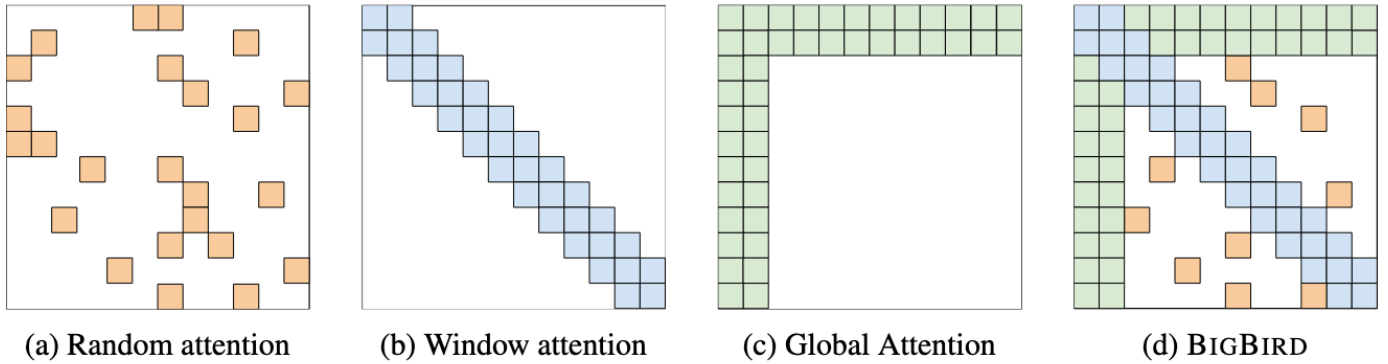
There were some leaks
It seems to be pretty similar to GPT-3,
but using a “product of experts” and other tricks

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

bonus!

Computational cost

- Each of *T* units attends to each *T* inputs: $O(T^2)$ cost per layer
- Various approaches to improving scalability
  - *Sparse attention*: just don't do all the connections, e.g. BigBird



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

  - Reformer approximates dot product with locality-sensitive hashing
  - Performer approximates better, based on fancy kernel methods

bonus!

Computational cost

- 14,400 GPUs: NVIDIA H100s, 80GB
  - Each one of these retails for about US$35,000 (if you can even get them)
  - 14,400 * $35,000 = $504,000,000
- Estimated that training GPT-3, ignoring hyperparameter search, costs ~US$5,000,000 in power costs (as much carbon as 500 NY-London flights)
  - Also involved a lot of flights – Danica has a friend-of-a-friend who flew cross-country ~weekly trying to get his feature into GPT-4

Pro

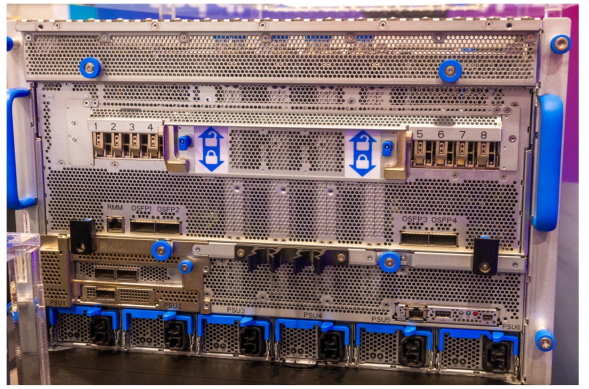
I think Microsoft Azure "Eagle" is probably the most important tech news of 2023 that you haven't heard of — here's why

News By Keumars Afifi-Sabet published November 23, 2023

Microsoft's Eagle supercomputer is the most powerful cloud-facing system in the world

Microsoft Azure Eagle is a Paradigm Shifting Cloud Supercomputer

By John Lee - November 15, 2023



Microsoft Azure HBv5 8x H100 System At SC23 Front

bonus!

Input representation for text

- Word-level: vocab gets *really big* to be multilingual, handle typos, …
- Character-level: more flexible!
  - Sequences really really long
  - 74,000+ Chinese characters, 3,000+ emoji
- Byte-level for UTF-8: can handle anything in 256 characters!
- Usual in-between these days using Byte-Pair Encoding:
    - Start with the 256 single bytes as tokens
    - Repeat: for the most commonly co-occurring pair (A, B), make a new token AB
    - Stop when you get to target size (usually a few tens of thousands)
    - Usually disallow merging "outside words": don't want "dog." "dog?" "dog!" tokens
  - Can assign probability to any Unicode string
  - Assign a Gaussian vector to each token, optimize as parameter from there

bonus!

Bidirectional Encoder Representations from Transformers

- BERT: very popular model in natural language processing (2018)
  - (Full) transformer model trained on masked sentences to predict masked words
    - Masked word prediction is a pretext task
  - Then fine-tune the architecture on specific applications

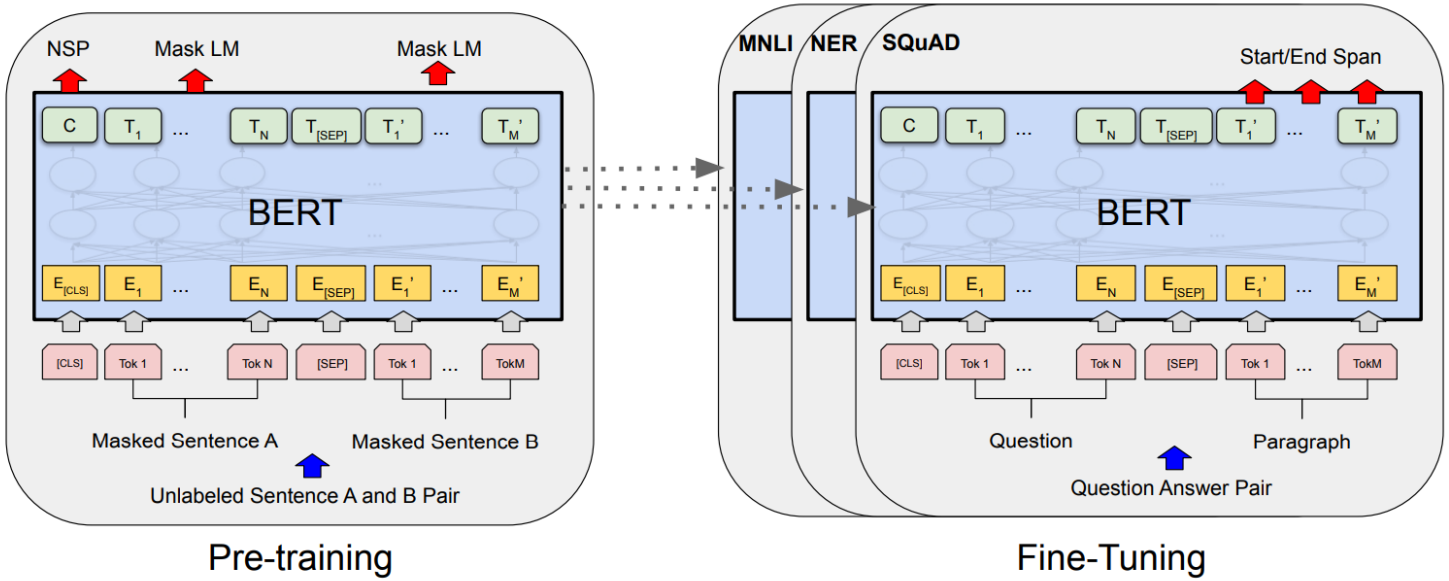
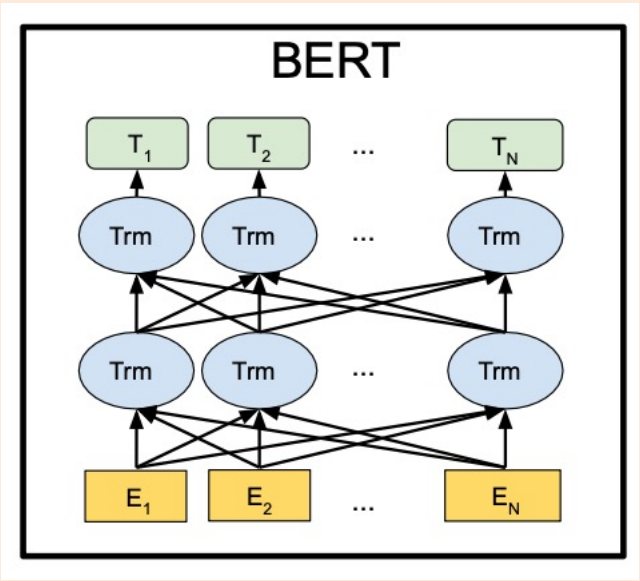


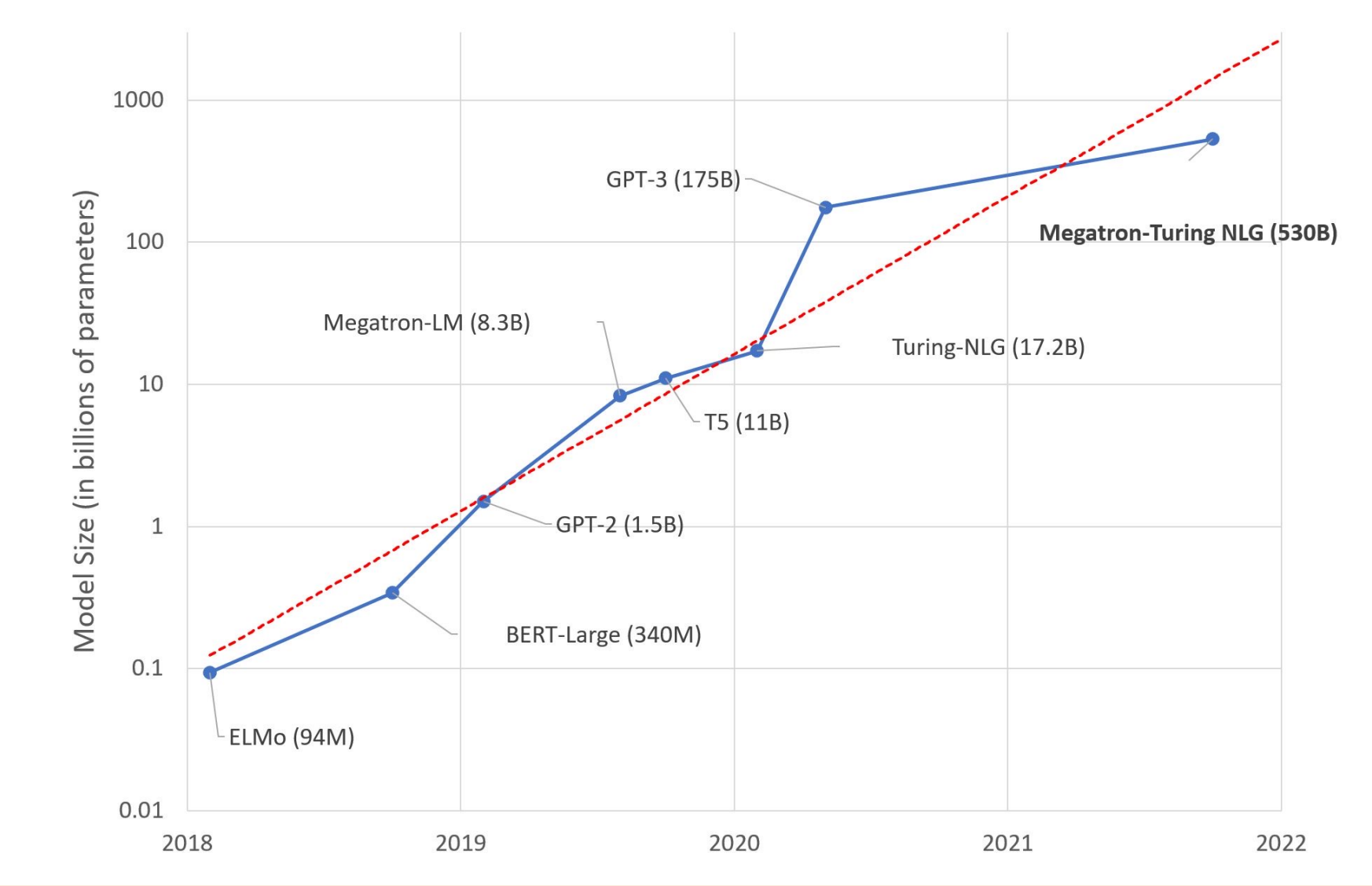
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).



https://arxiv.org/pdf/1810.04805.pdf

bonus!

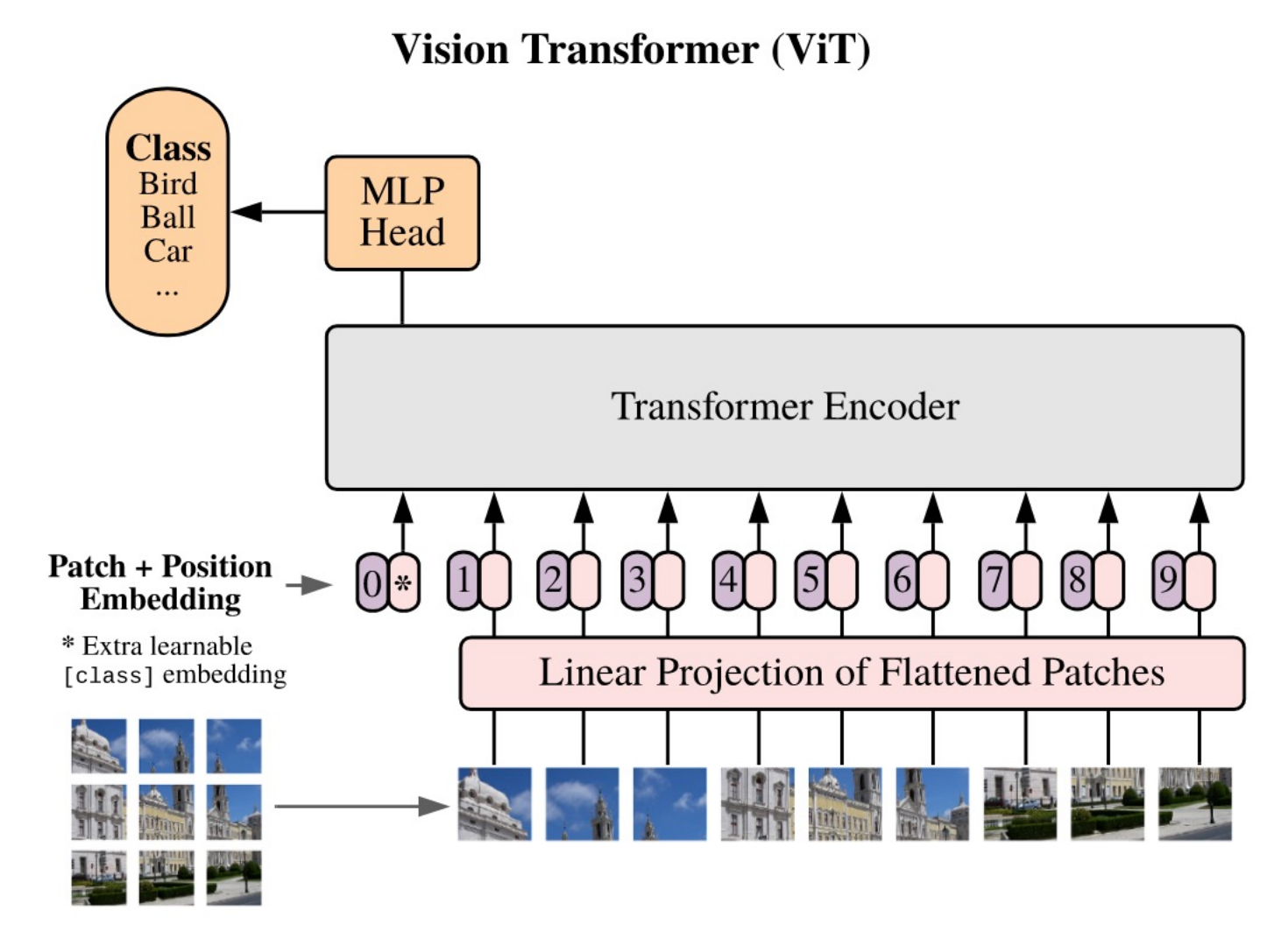
Bigger and bigger



https://huggingface.co/blog/large-language-models

bonus!

Vision Transformers

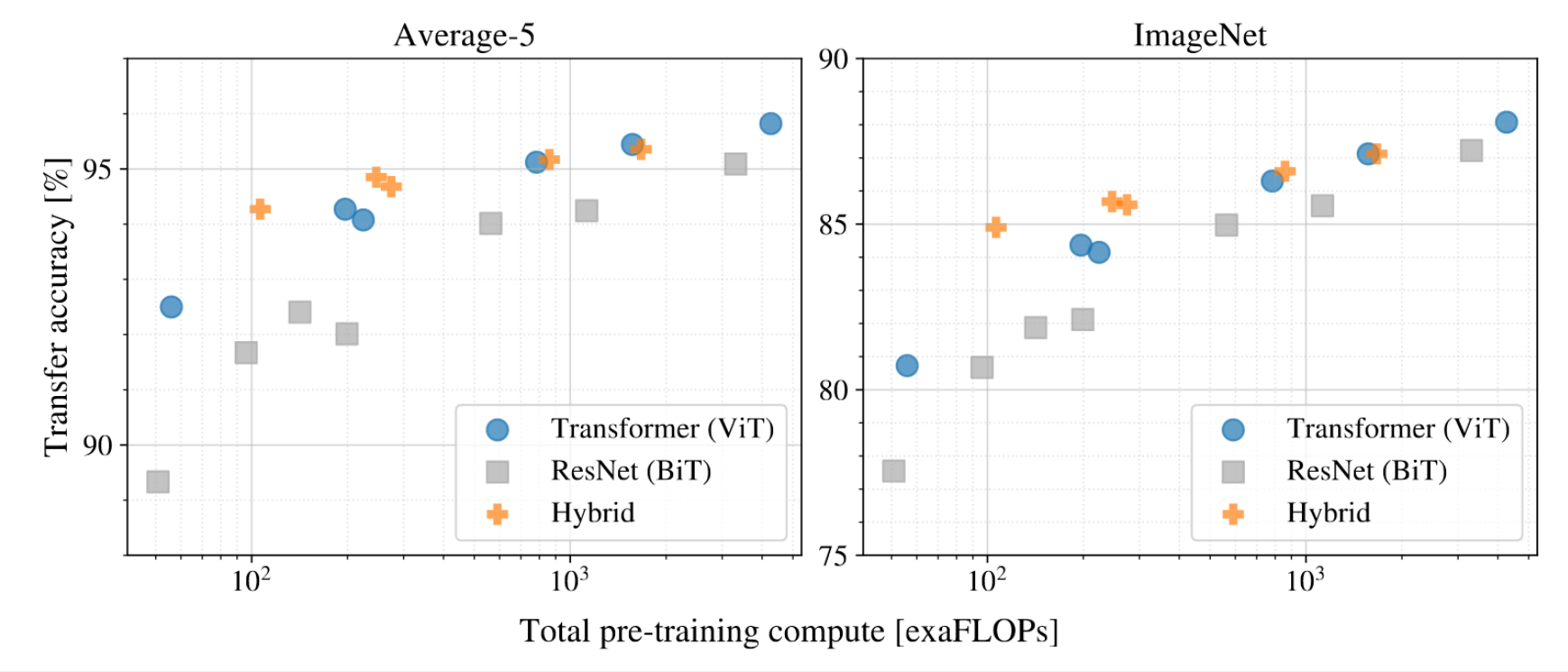


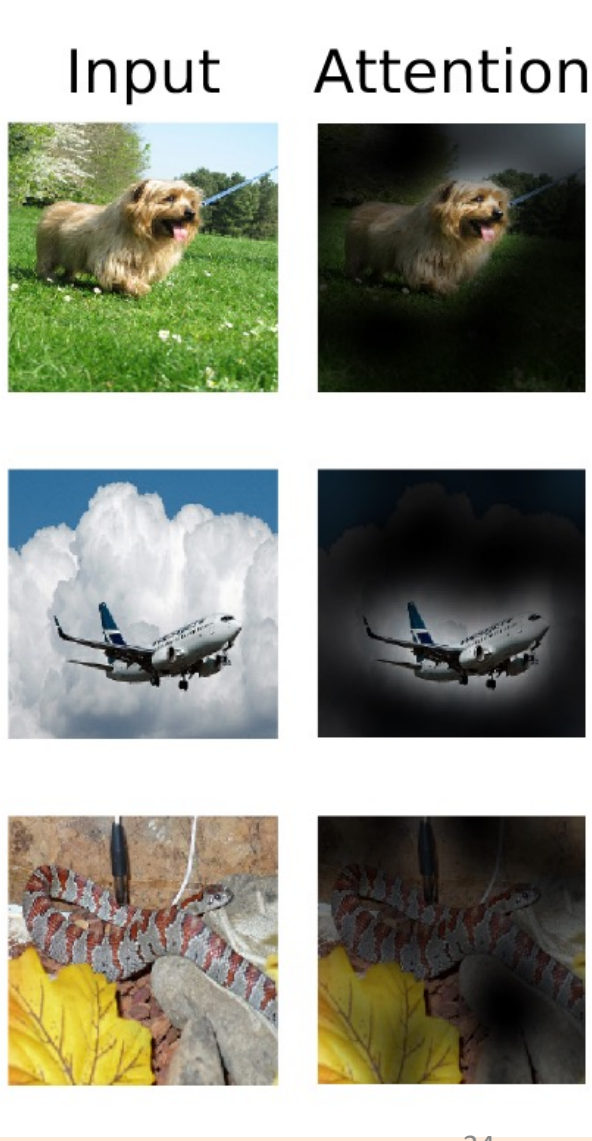
https://arxiv.org/abs/2010.11929

bonus!

Vision Transformers

- Usually outperform CNNs if you have enough data





https://arxiv.org/abs/2010.11929

bonus!

MLPs on patches might be enough

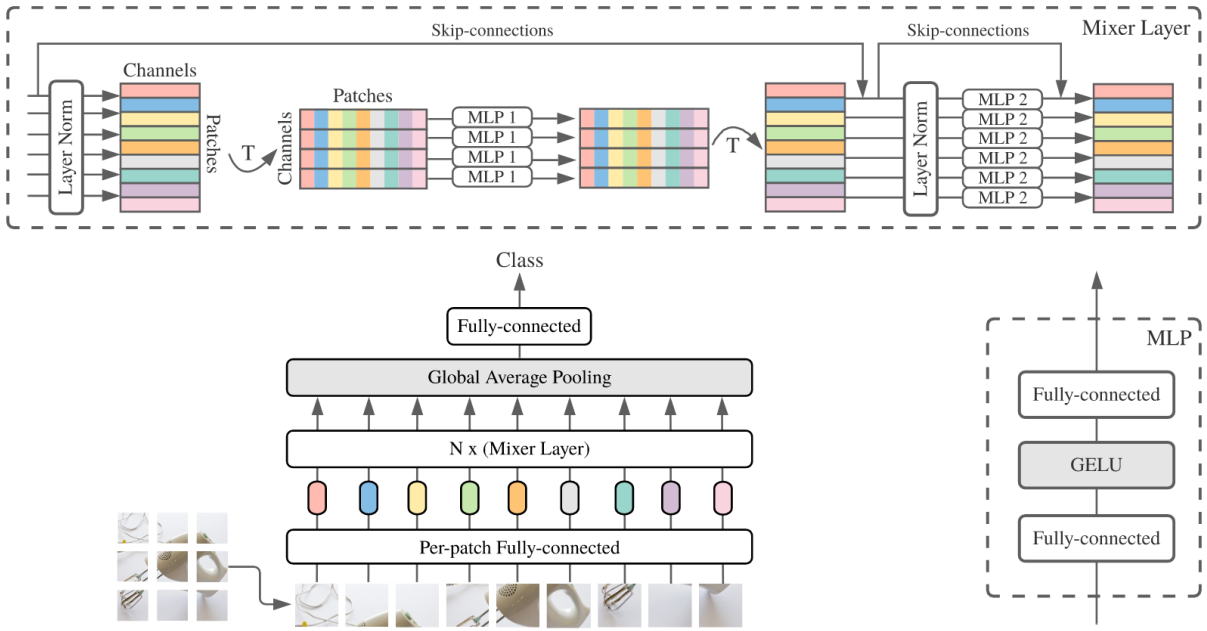
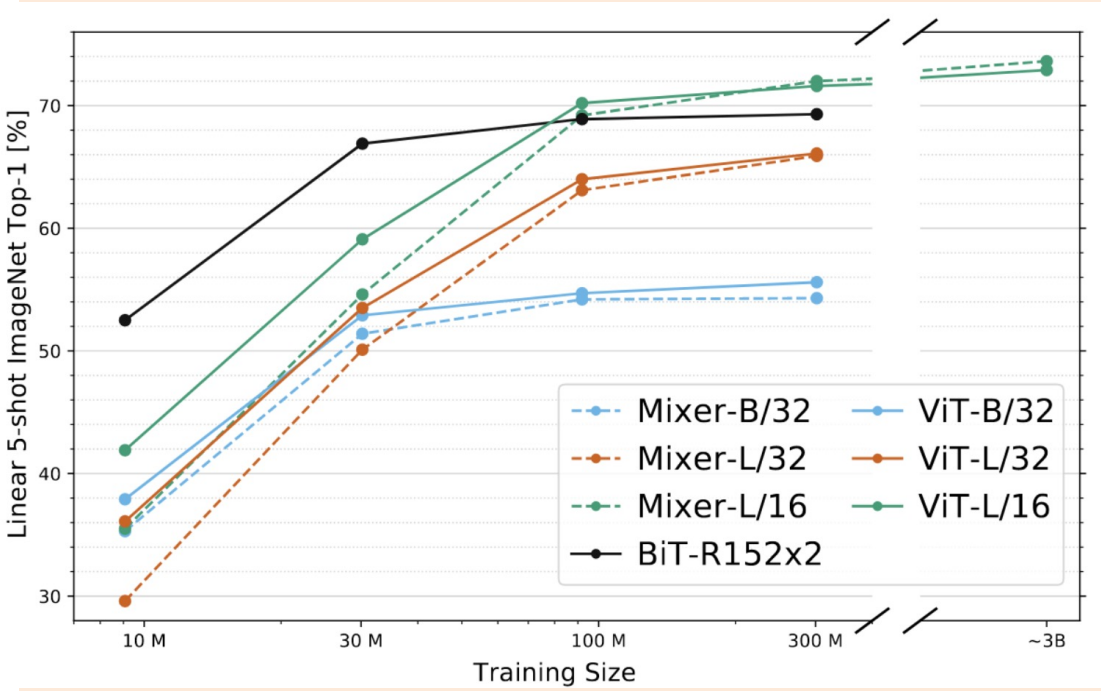


Figure 1: MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, and layer norm on the channels.



https://arxiv.org/abs/2105.01601

bonus!

Dropping Attention

~~Convolutions~~ ~~Attention~~ ~~MLPs~~

Patches Are All You Need? 🤷

Asher Trockman, J. Zico Kolter[1]
Carnegie Mellon University and [1]Bosch Center for AI

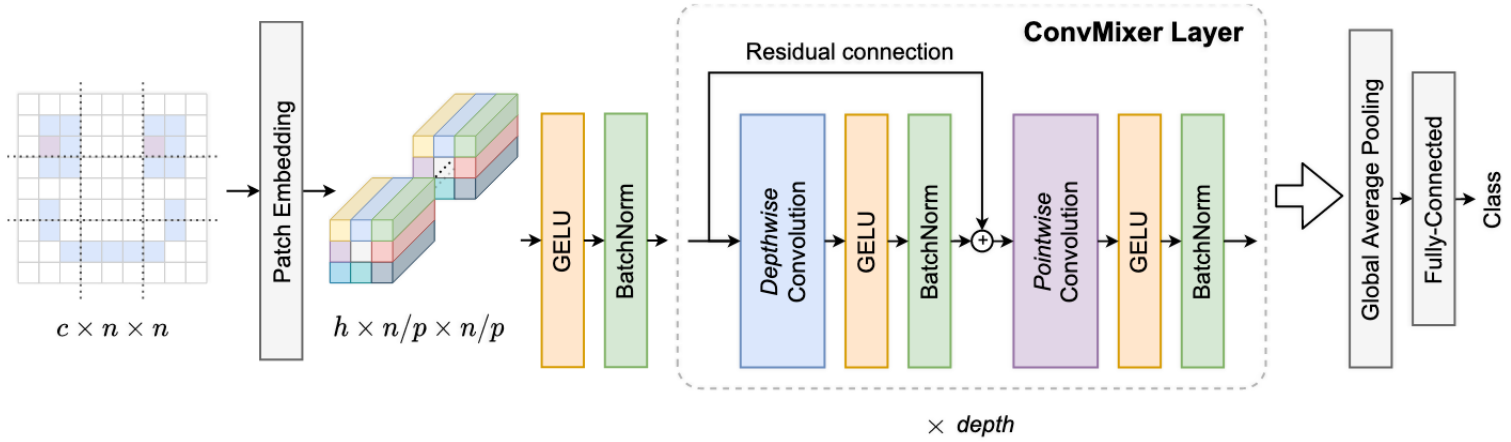


Figure 2: ConvMixer uses "tensor layout" patch embeddings to preserve locality, and then applies d copies of a simple fully-convolutional block consisting of *large-kernel* depthwise convolution followed by pointwise convolution, before finishing with global pooling and a simple linear classifier.

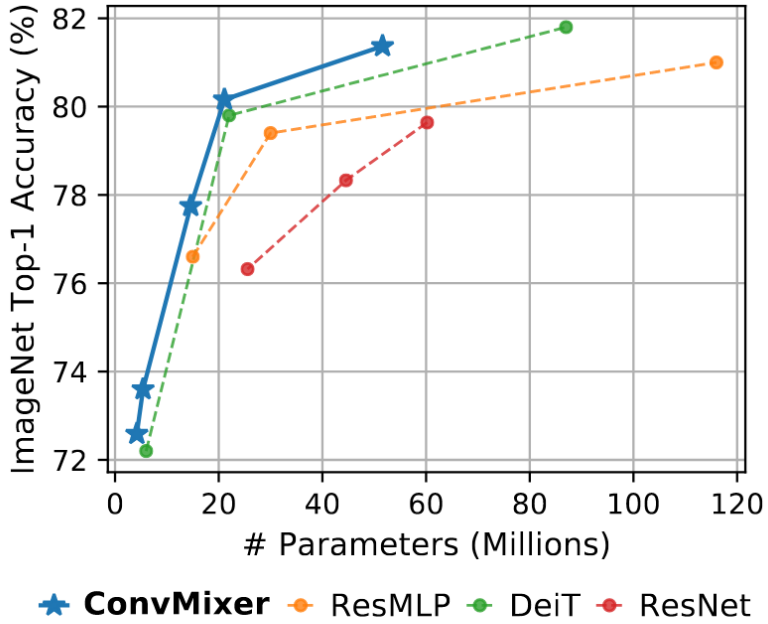


Figure 1: Accuracy *vs.* parameters, trained and evaluated on ImageNet-1k.

https://arxiv.org/abs/2201.09792

bonus!

Combining convolutions with attention

- Conformer: basis for recent top speech recognition systems
- Convolution might be better at "very local" features
  - See post-summary bonus slides about convolutions on sequences
- Can also do these kinds of combinations in other domains

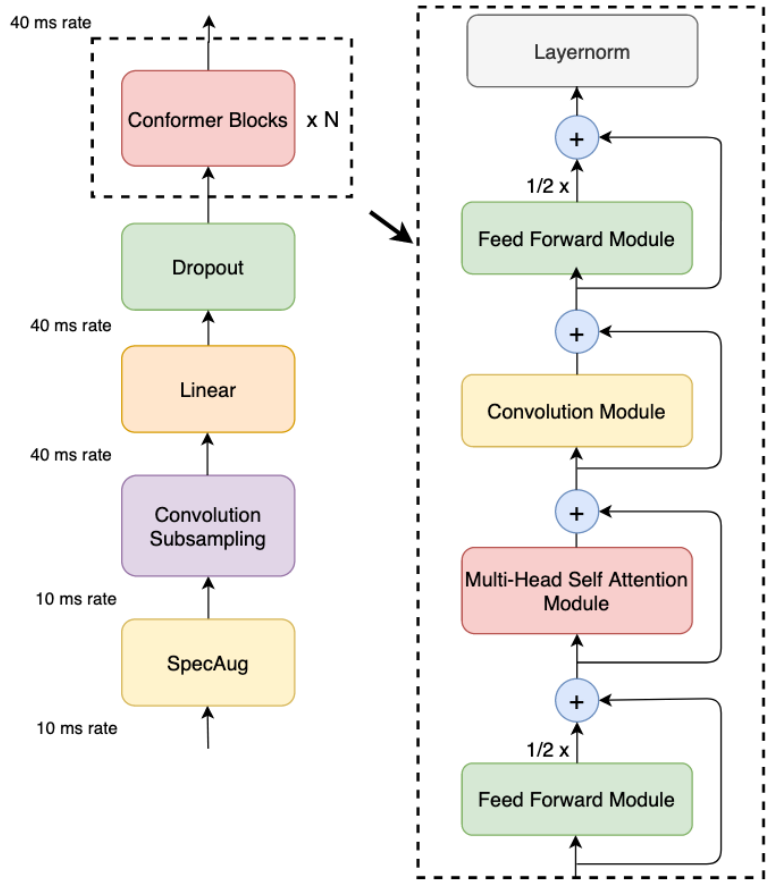


Figure 1: ***Conformer encoder model architecture.*** *Conformer comprises of two macaron-like feed-forward layers with half-step residual connections sandwiching the multi-headed self-attention and convolution modules. This is followed by a post layernorm.*

https://arxiv.org/abs/2005.08100

Summary

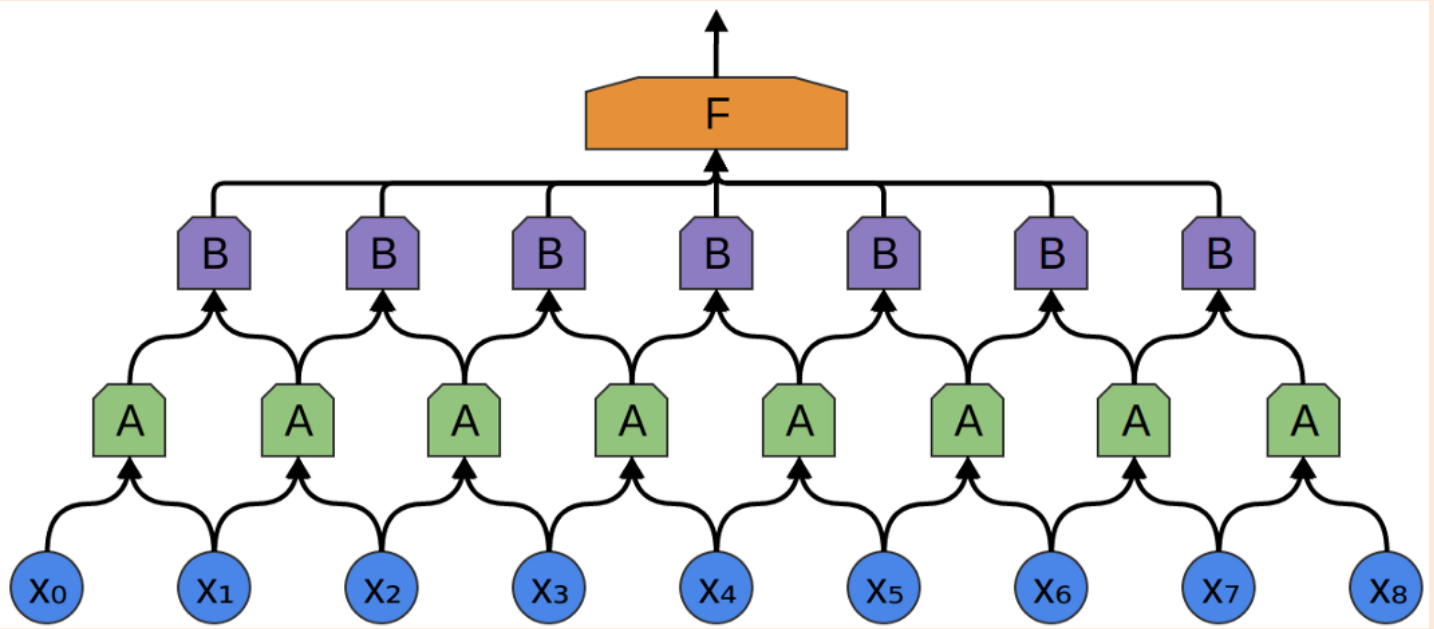
- Attention:
  - Allow decoder to look at previous states.
- Context vectors:
  - Combine previous states into a fixed-length vector.
- [Dilated] convolutions for sequences.
  - Alternative to sequential architectures like RNNs.
- Transformer networks:
  - Layers of “self-attention” to build context.
    - “Everything depends on everything”, and you learn how.
    - Lots of implementation details, but excellent performance on many tasks.
    - Basis for modern enormous/impressive language models and applications.

- Next time: what is our ~~children~~models learning?

bonus!

Convolutions for Sequences?

- Alternative approach, mostly predating Transformers
  - Convolutions for sequences

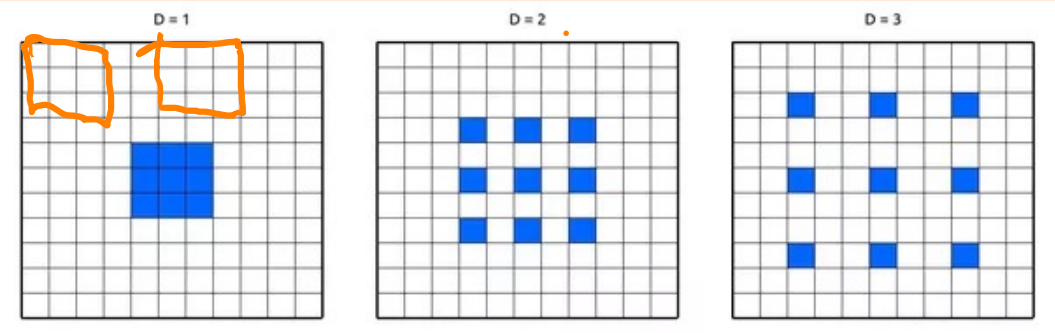


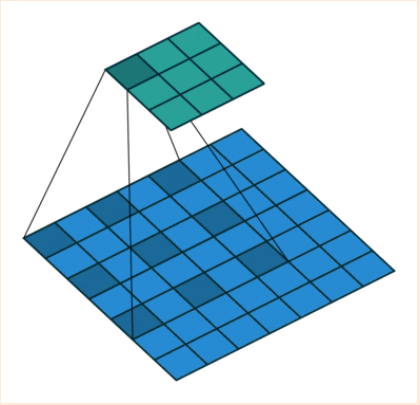
https://medium.com/@TalPerry/convolutional-methods-for-text-d5260fd5675f

Digression: Dilated Convolutions (“a trous”)

bonus!

- Best CNN systems have gradually reduced convolutions sizes
  - Many modern architectures use 3x3 convolutions, far fewer parameters
- Sequences of convolutions take into account larger neighbourhood
  - 3x3 convolution followed by another gives a 5x5 neighbourhood
  - But need many layers to cover a large area
- Alternative recent strategy is dilated convolutions (“a trous”)





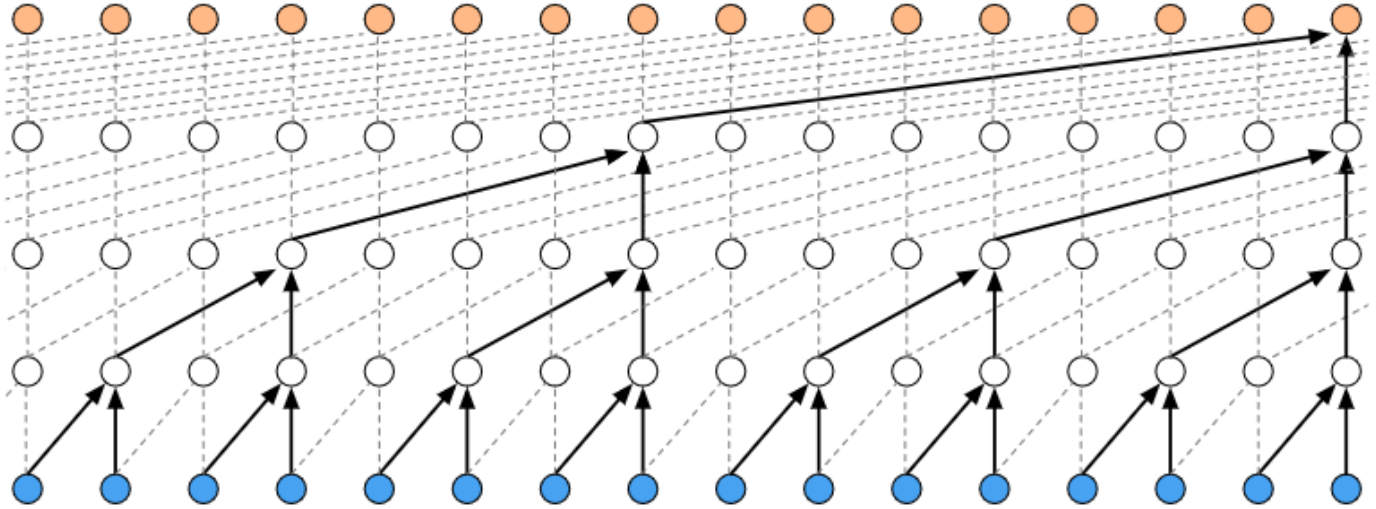
- Not the same as “stride” in a CNN:
  - Doing a 3x3 convolution at all locations, but using pixels that are not adjacent

https://medium.com/@TalPerry/convolutional-methods-for-text-d5260fd5675f
https://github.com/vdumoulin/conv_arithmetic

bonus!

Dilated Convolutions (“a trous”)

- Modeling music and language and with dilated convolutions:



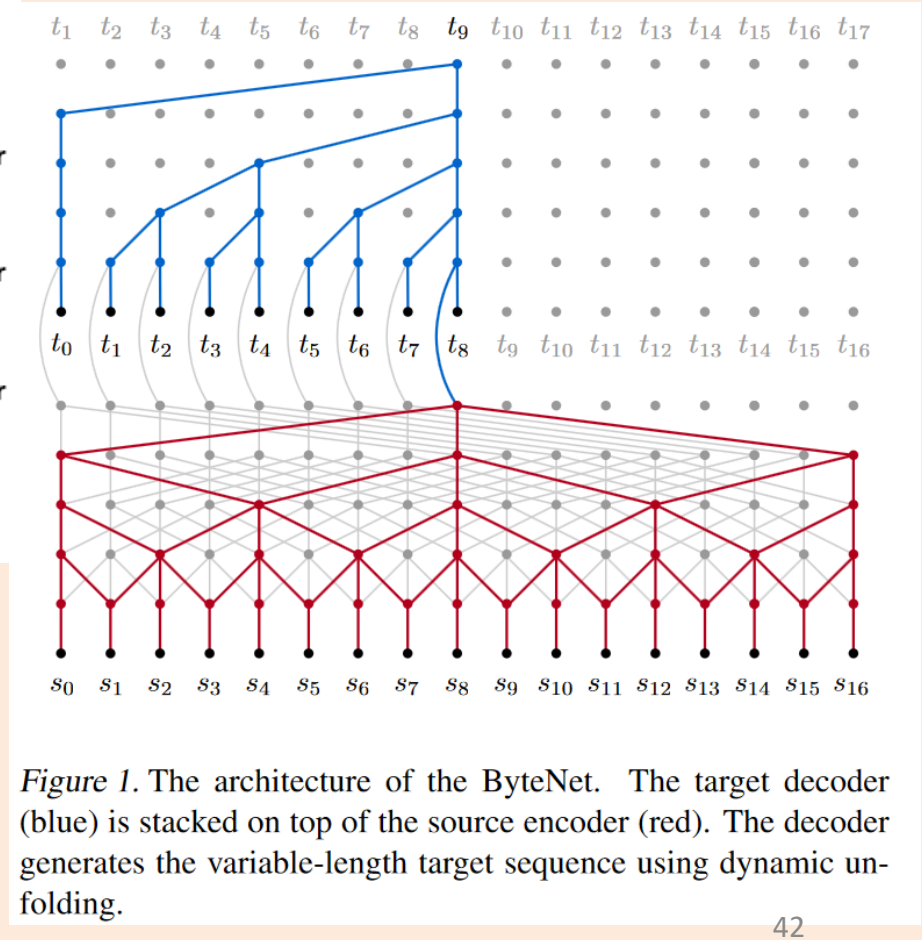


Figure 1. The architecture of the ByteNet. The target decoder (blue) is stacked on top of the source encoder (red). The decoder generates the variable-length target sequence using dynamic unfolding.

https://arxiv.org/pdf/1610.10099.pdf
https://arxiv.org/pdf/1609.03499.pdf

RNNs/CNNs/Attention for Music and Dance

bonus!

- Music generation:
  - https://www.youtube.com/watch?v=RaO4HpM07hE
- Text to speech and music waveform generation:
  - https://deepmind.com/blog/wavenet-generative-model-raw-audio
- Dance choreography:
  - http://theluluartgroup.com/work/generative-choreography-using-deep-learning
- Music composition:
  - https://www.facebook.com/yann.lecun/videos/10154941390687143