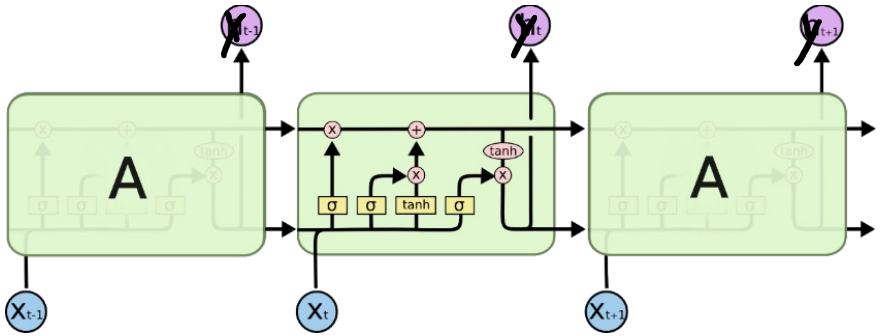
CPSC 440: Machine Learning

Attention and Transformers

Winter 2022

Last Time: LSTMs and Multi-Modal Learning

- We discussed long short term memory (LSTM) models:
  - RNNs with memory cells designed to remember information longer.



$$a_t = o_t \circ h_a(c_t)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t$$

$$g_t = h_g(W_g x_t + U_g a_{t-1})$$

activation/memory cell/new memory value

$$f_t = h(W_f x_t + U_f a_{t-1})$$

$$i_t = h(W_i x_t + U_i a_{t-1})$$

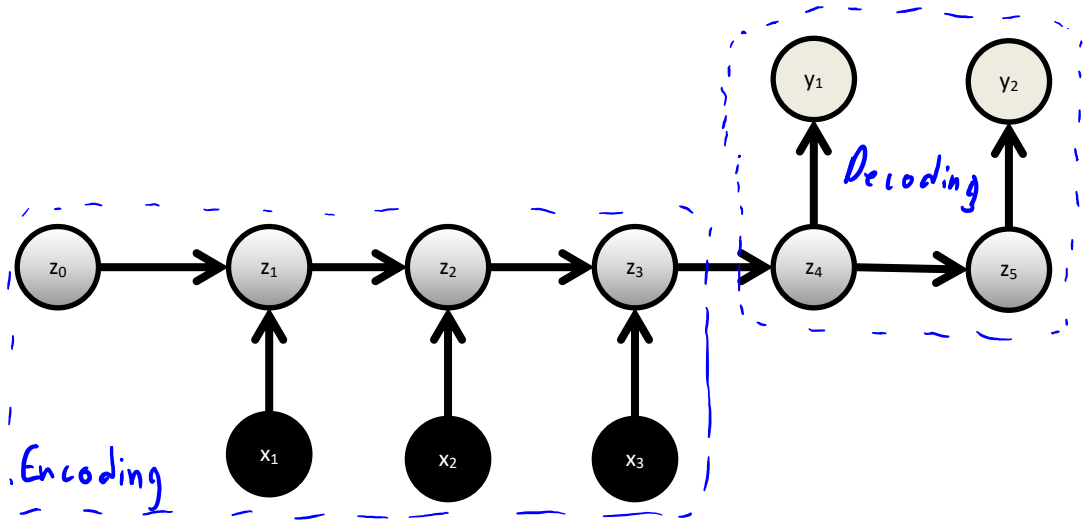
$$o_t = h(W_o x_t + U_o a_{t-1})$$

forget/input/output sigmoid "gates"

- We discussed using encoders and decoders of different data types:
  - Encoder takes an image and decoder outputs a sequence.
  - Image captioning, video annotation, lip reading, poetry about images.

Last Time: Sequence-to-Sequence RNNs

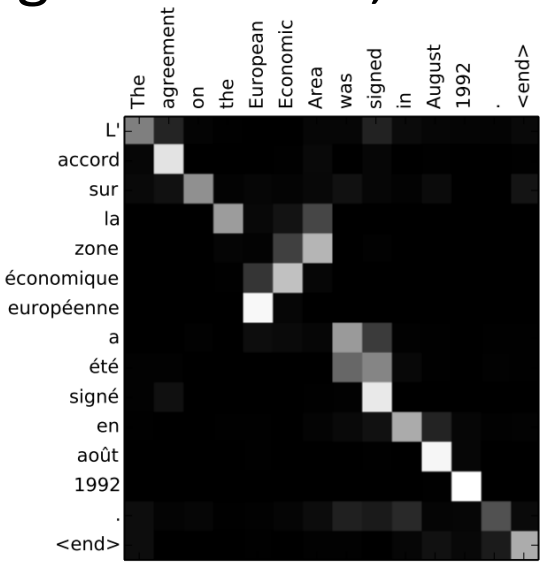
- Sequence-to-sequence:
  - Recurrent neural network for sequences of different lengths.



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

Attention

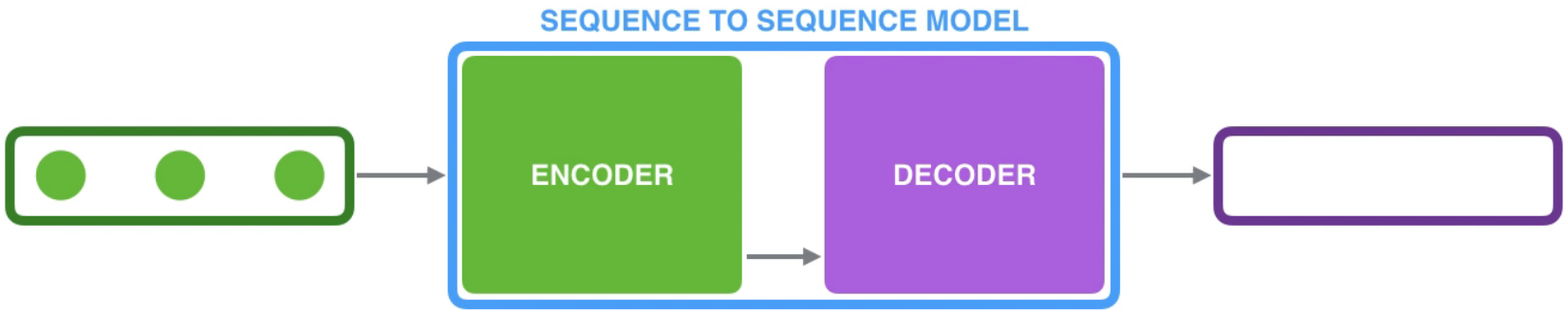
- Many recent systems use “attention” to focus on parts of input.
  - Including GPT, Google Translate, ….

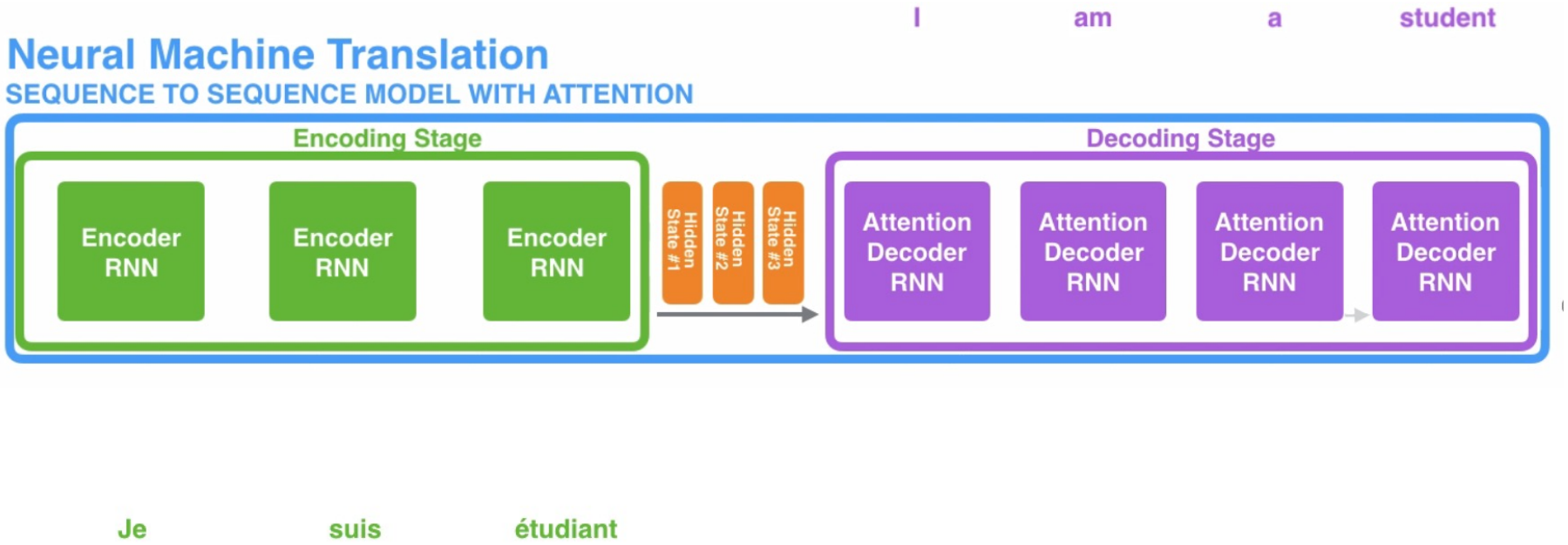


- Many variations on attention, but usually include the following:
  - Each decoding can use hidden state from each encoding step.
    - Used to re-weight during decoding to emphasize important parts.

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

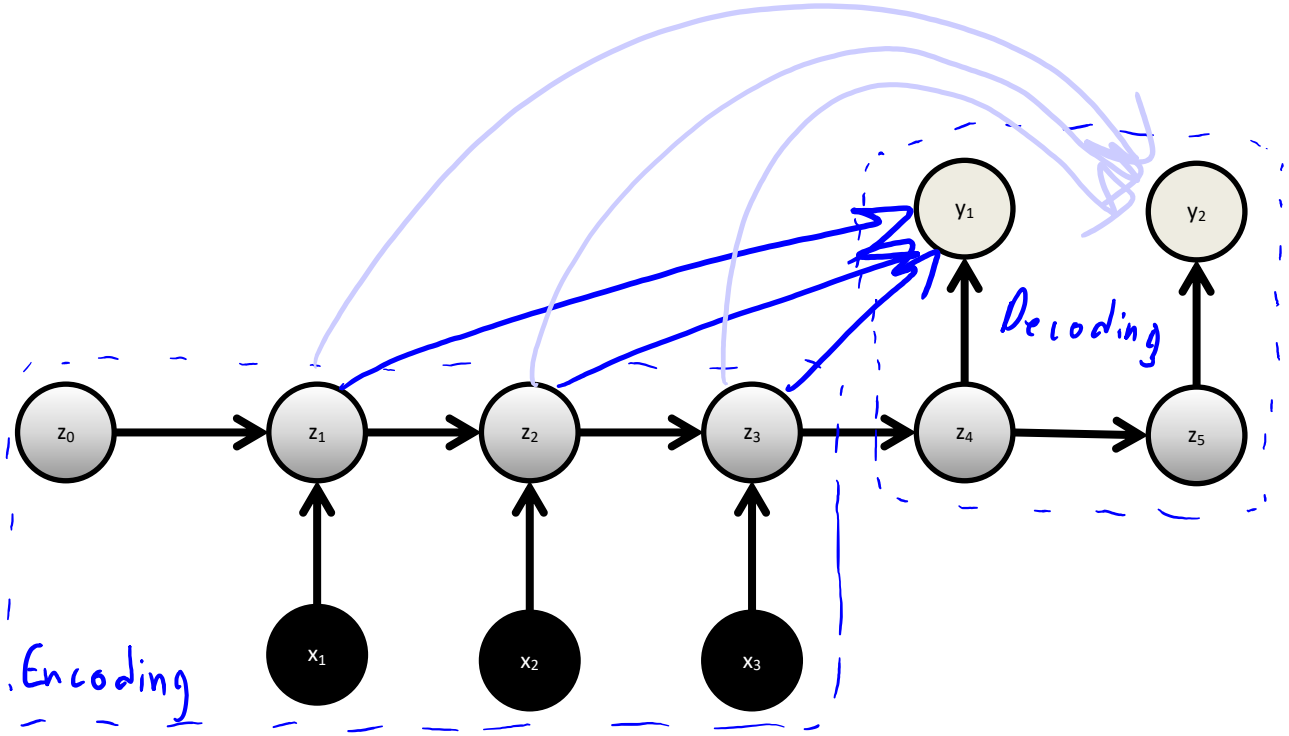
RNN vs. RNN with Attention Videos





Not-Very-Practical Attention

- A naïve "attention" method (no one uses this, but idea is similar):
  - At each decoding step, weight decoder state (as usual) and weight all encoder states.



  - Another variation on the "residual connection" or "denseNet" trick.
  - But this variant is not practical since number of decoding weights depends on input size.
    - Practical variations try to summarize encoder information through a "context vector".

Would require all inputs to have the same length.

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

Context Vectors

- A common way to generate the context vector:
  - Take current decoder state.
  - Compute inner product with each encoder state.
    - Gives a scalar for each encoding “time”.
  - Pass these scalars through the softmax function.
    - Gives a normalized weight for each time (what was previously shown in pairwise tables).
  - Multiply each encoder state by probability, add them up.
    - Gives fixed-length “context vector”.

- Alternate notation (like a hash function):
  - Input is “queries” and “keys”.
  - Output is “values”.

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

Context Vectors

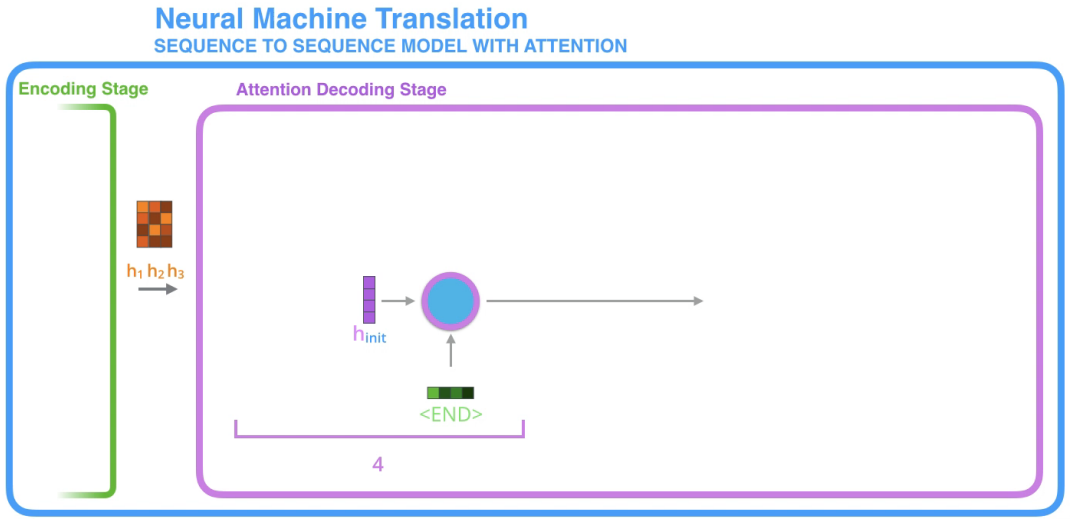
- A common way to generate the context vector:
  - Take current decoder state.
  - Compute inner product with each encoder state.
    - Gives a scalar for each encoding “time”.
  - Pass these scalars through the softmax function.
    - Gives a normalized weight for each time (can be shown in pairwise tables).
  - Multiply each encoder state by probability, add them up.
    - Gives fixed-length “context vector”.

- Alternate notation (like a hash function):
  - Input is “queries” and “keys”.
  - Output is “values”.

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

Using Context Vectors for Attention

- Context vector is usually appended to decoder's state when going to next layer.
  - Output could be generated directly from this, or passed through a neural net.
  - Common variation is "multi-headed attention": can get scores from different aspects.
    - One context vector for semantics, one for grammar, one for tense, and so on.
    - Each is appended to decoder state when going to next layer.
    - Context vectors are usually not included when updating the decoder state temporally.

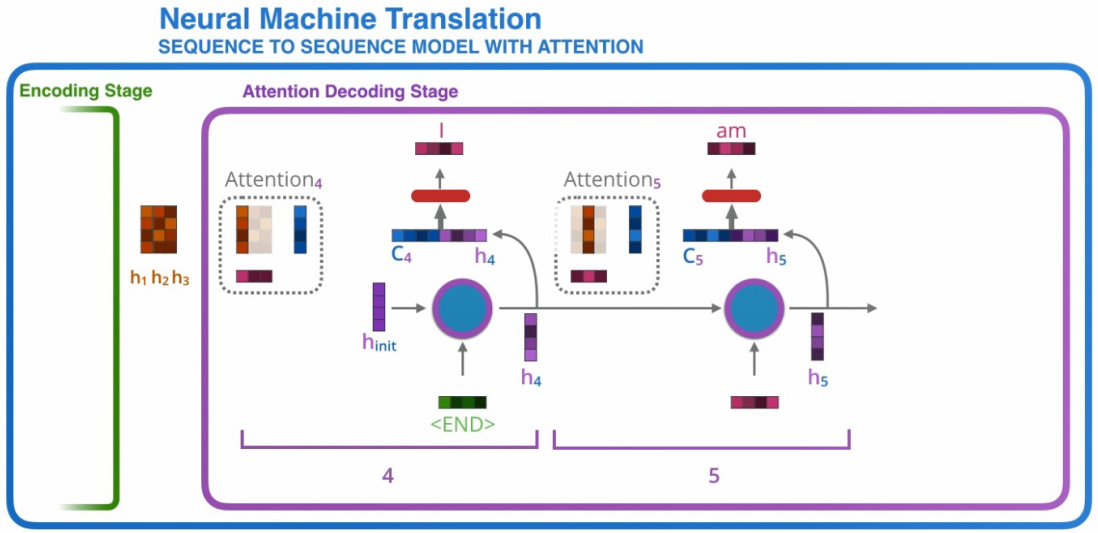


- Remember that we train the encoder and decoder at the same time.

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

Using Context Vectors for Attention

- Context vector is usually appended to decoder's state when going to next layer.
  - Output could be generated directly from this, or passed through a neural net.
  - Common variation is "multi-headed attention": can get scores from different aspects.
    - One context vector for semantics, one for grammar, one for tense, and so on.
    - Each is appended to decoder state when going to next layer.
    - Context vectors are usually not included when updating the decoder state temporally.



- Remember that we train the encoder and decoder at the same time.

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

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

bonus!

Neural Turing/Programmers

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

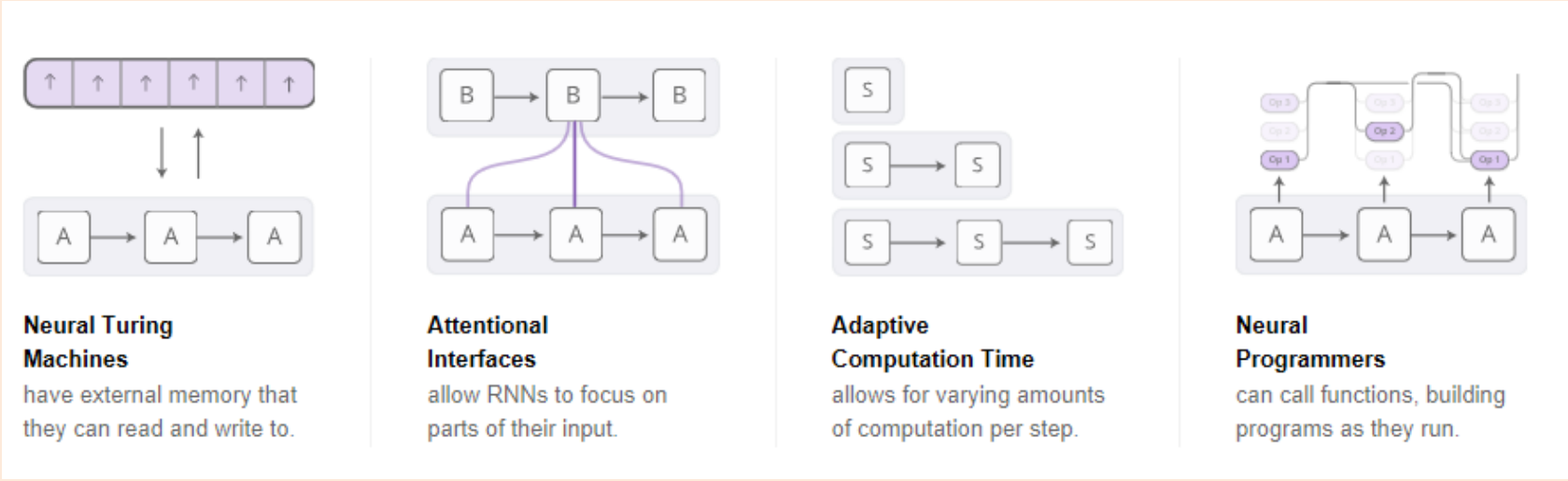
Here is an example of what the system can do. After having been trained, it was fed the following short story containing key events in JRR Tolkien's Lord of the Rings:

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".



  - We will focus next on a wildly-popular variant called "transformers".

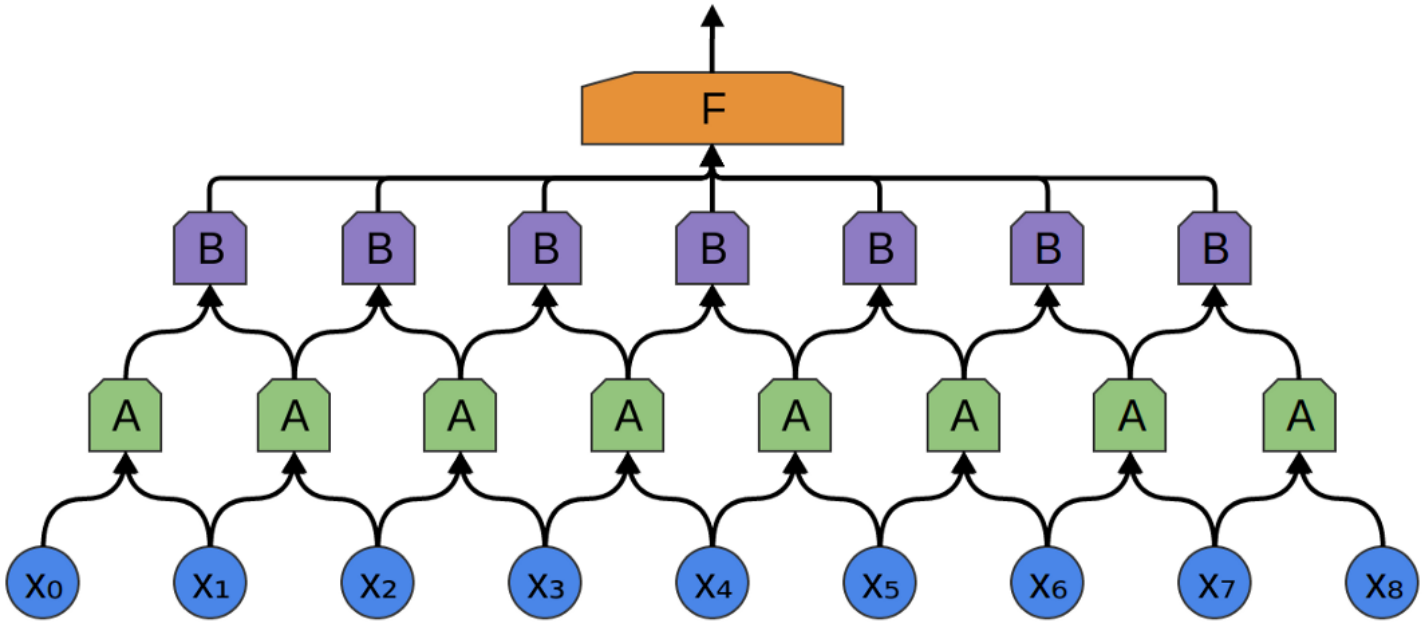
https://www.facebook.com/FBAIResearch/posts/362517620591864

Next Topic: Transformers

Convolutions for Sequences?

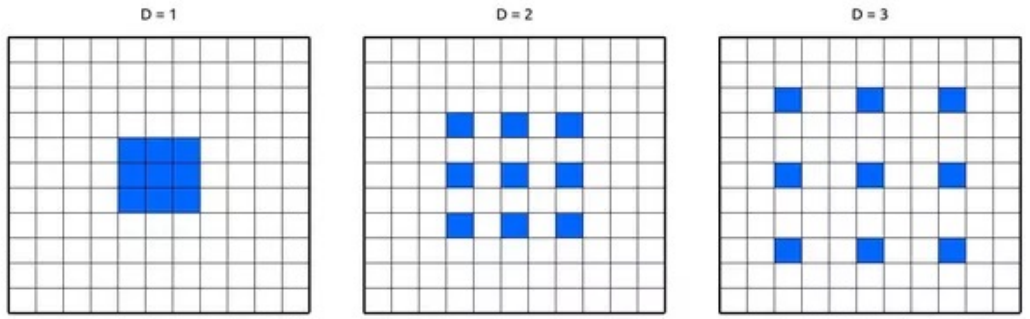
- Should we really be going through a sequence sequentially?
  - What if stuff in the middle is really important, and changes meaning?

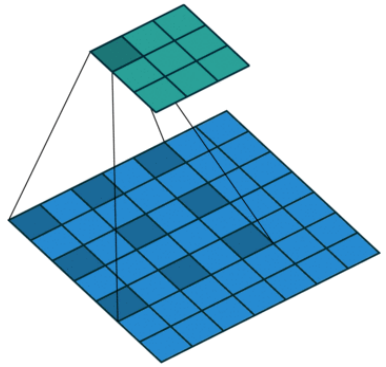
- Recent works have explored using convolutions for sequences.



Digression: Dilated Convolutions (“a trous”)

- 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”).



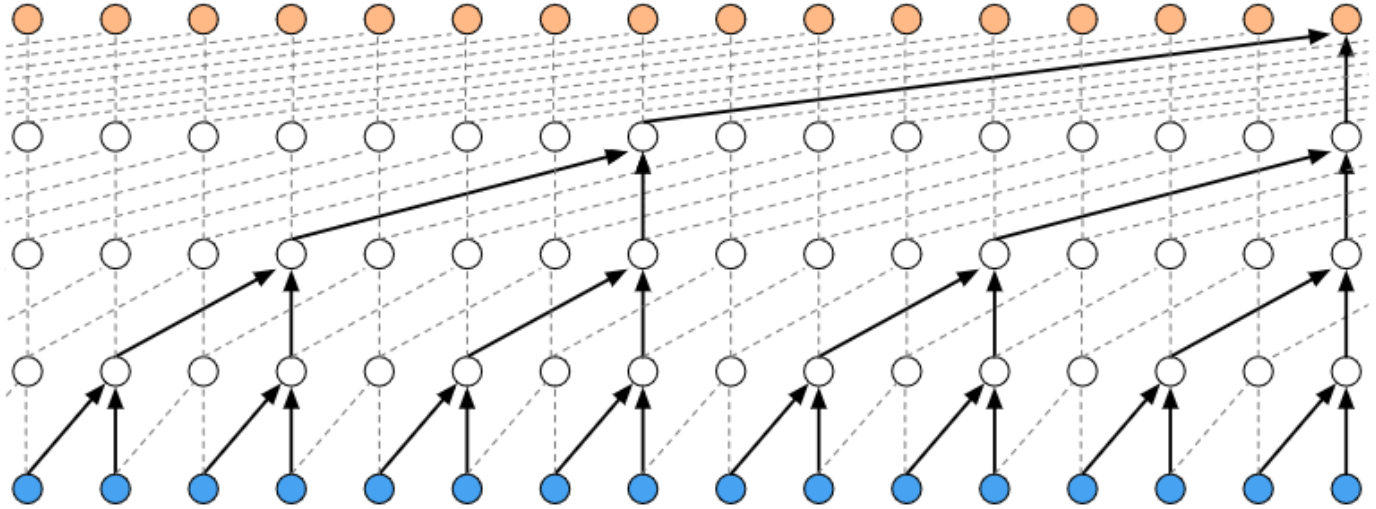


- Not the same as “stride” in a CNN:
  - Doing a 3x3 convolution at all locations, but using pixels that are not adjacent.

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

Dilated Convolutions (“a trous”)

- Modeling music and language and with dilated convolutions:



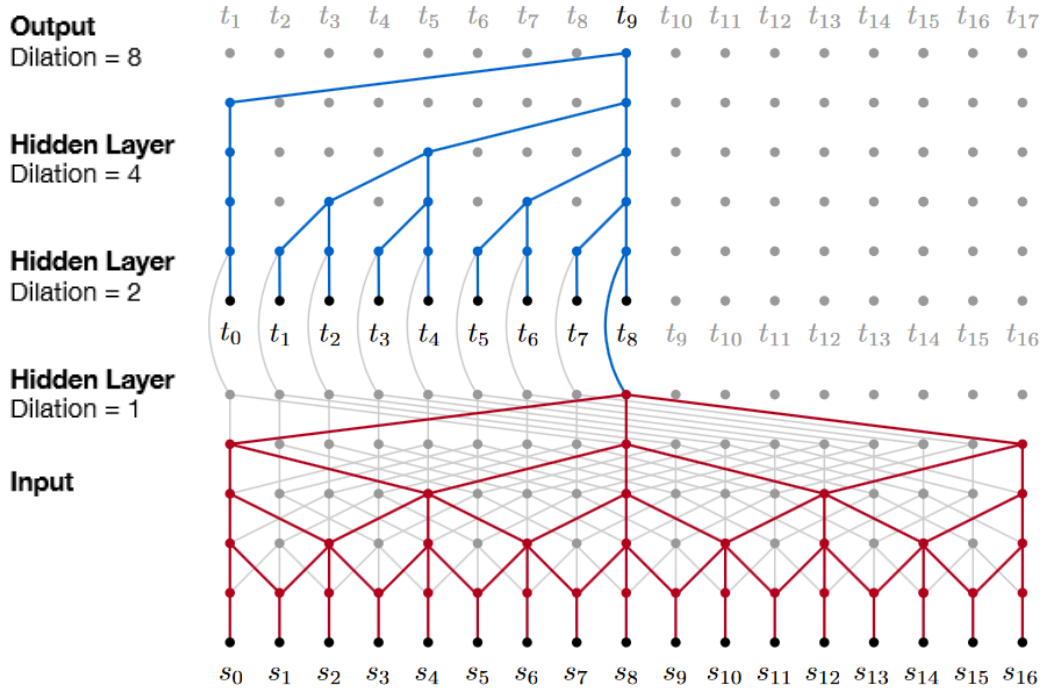


Figure 1. The architecture of the ByteNet. The target decoder (blue) is stacked on top of the source encoder (red). The decoder generates the variable-length target sequence using dynamic unfolding.

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

RNNs/CNNs/Attention for Music and Dance

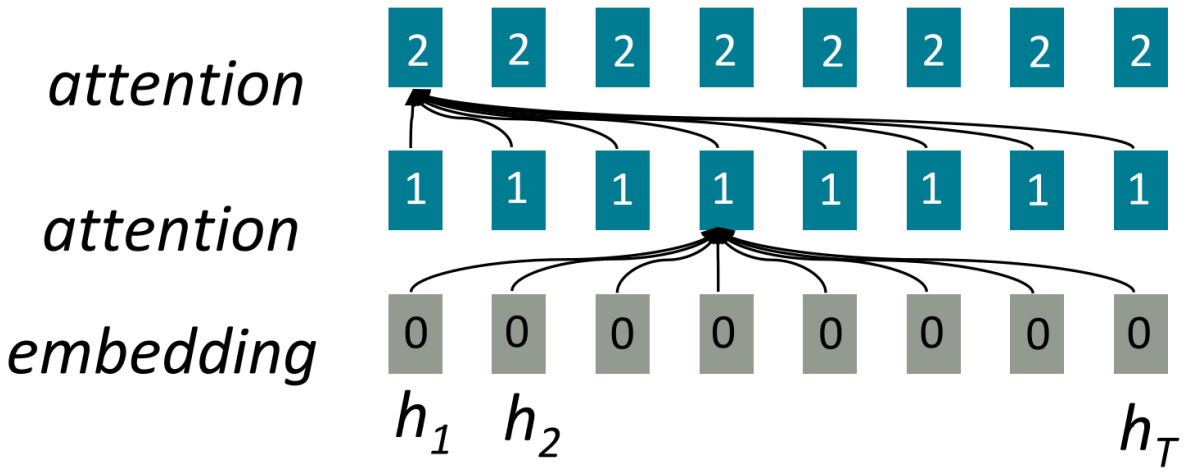
bonus!

- 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

Transformer Networks

- CNNs are less sequential, but take multiple steps to combine distant information.

- "Attention is all you need": keep the attention, ditch the RNN/CNN.
  - Constant time to transfer across positions.
  - Uses "self-attention" layers to model relationship between all words in input.
    - Queries/keys/values all come from input in these steps.

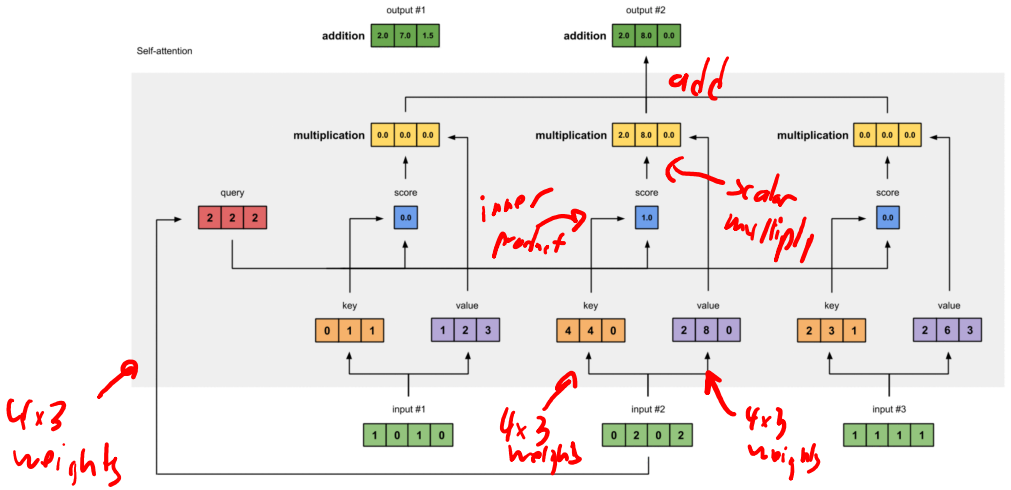


All words attend to all words in previous layer; most arrows here are omitted

- Sequence of representations of words, each depending on all other words.

Transformer Networks

- CNNs are less sequential, but take multiple steps to combine distant information.
- "Attention is all you need": keep the attention, ditch the RNN/CNN.
  - Constant time to transfer across positions.
  - Uses "self-attention" layers to model relationship between all words in input.
    - Take weighted combinations of each input to generate a "key", a "value", and a "query".
    - Compute inner product between "query" from word with "key" for each word to give scalar "score".
    - Compute softmax of "scores", multiplied by word's "value", add these across words to get context vector.

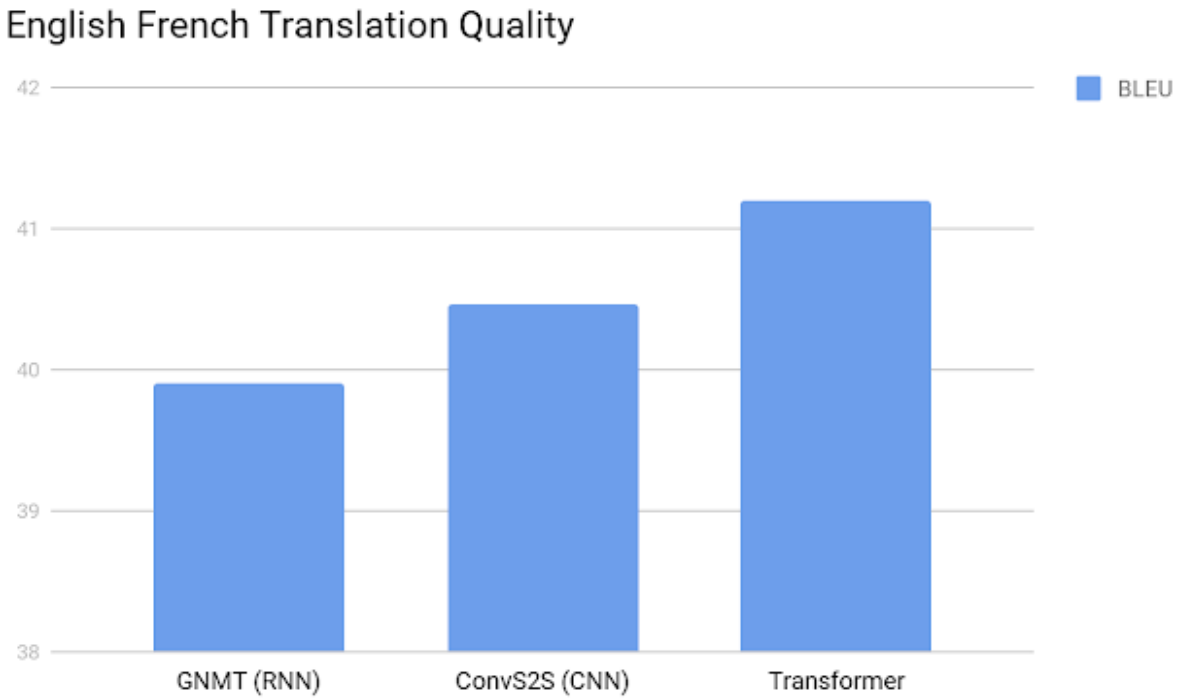


- Many variations exist.

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

Transformer Networks

- Multiple “self-attention” layers in transformers replacing RNN/CNN.
  - Has improved on state of the art results in many tasks.



(these models have around 100M parameters)

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Position encodings

- RNNs see sequences in order; CNNs have order built-in
- 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 learns them
  - Feature vector for word 1, word 2, … that you learn as a parameter

bonus!

Transformer Networks: Practical Issues

- "Self-attention" layers are basis for transformer networks.
  - Simple idea, but practical systems have a lot of moving pieces.

- Problem: information about the future can be visible in the past.
  - During training, prevent decoder from looking ahead.

- Further "standard" tricks to make it work better:
  - Multi-headed attention, skip/residual connections, and layer normalization.
  - Between layers, pass each embedding through a feedforward neural network.

bonus!

Transformer Architecture (from paper)

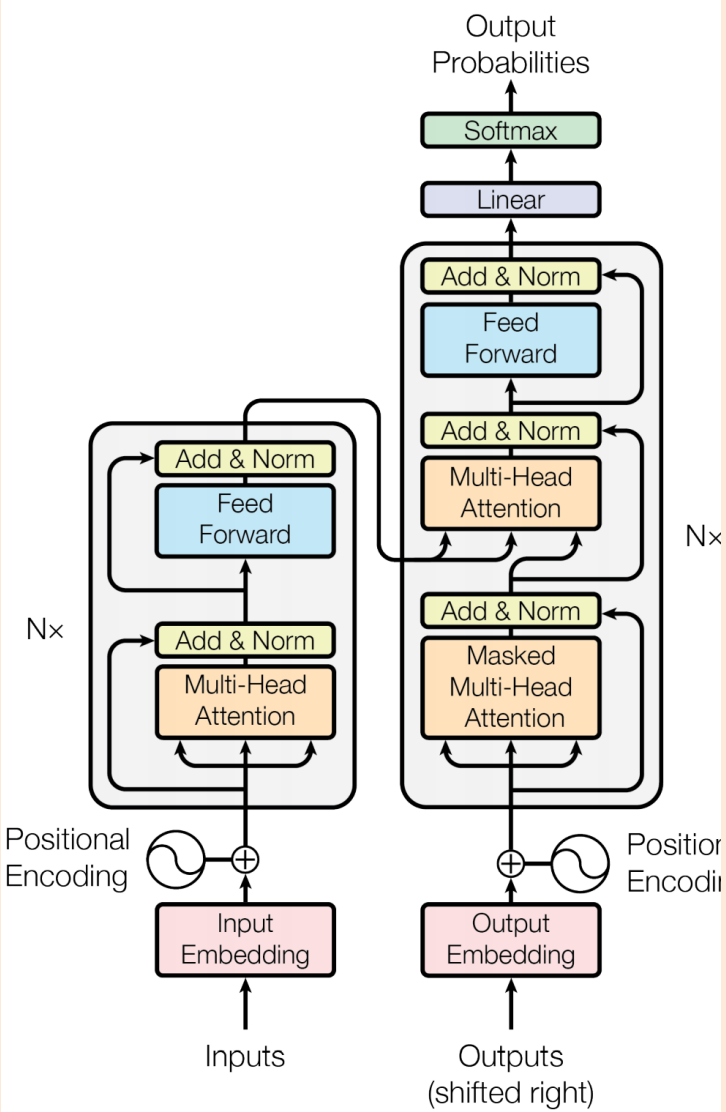


Figure 1: The Transformer - model architecture.

https://arxiv.org/pdf/1706.03762.pdf

bonus!

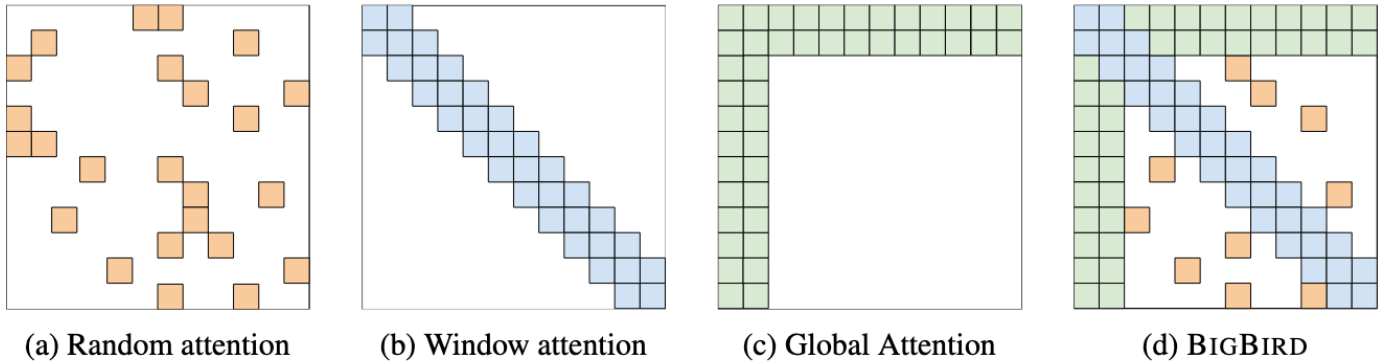
Input representation for text

- 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

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!

Bidirectional Encoder Representations from Transformers

- BERT: incredibly popular model in natural language processing.
  - Transformer model trained on masked sentences to predict masked words.
  - Then fine-tune the architecture on specific applications.

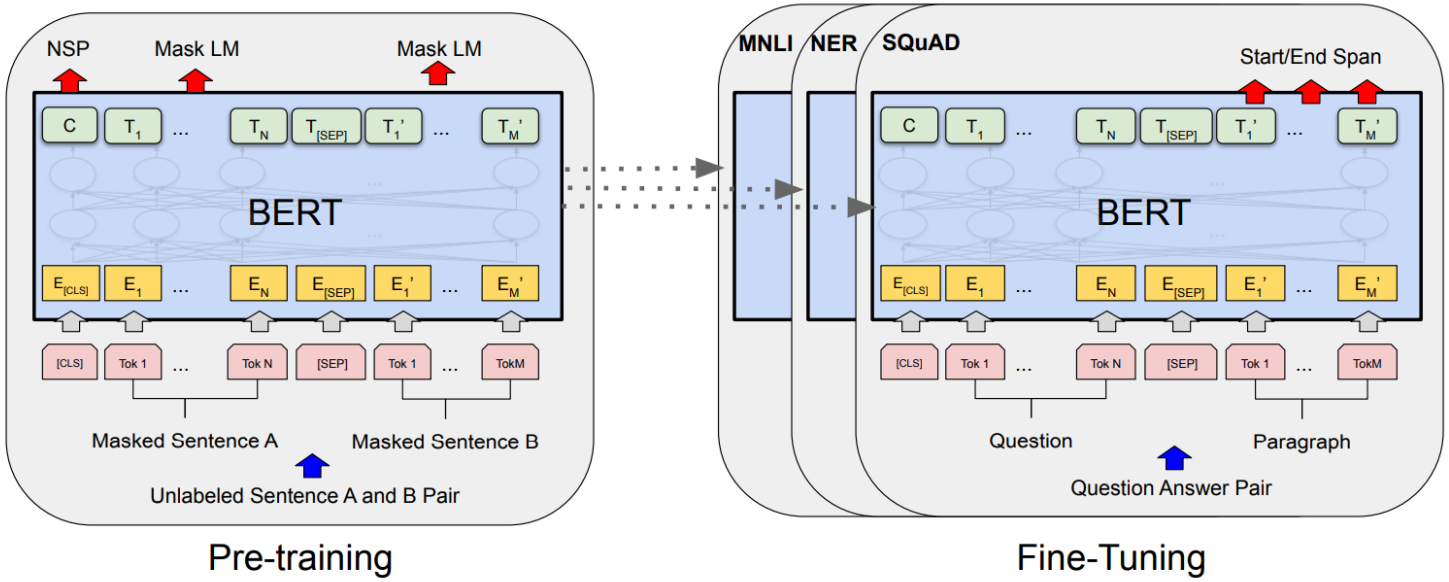
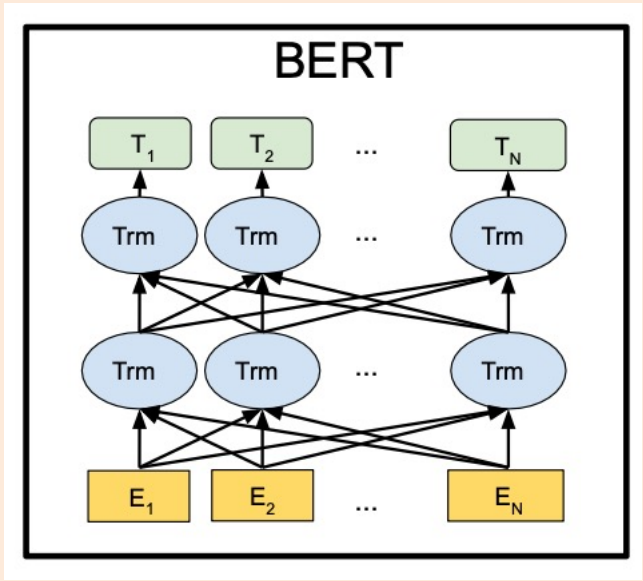


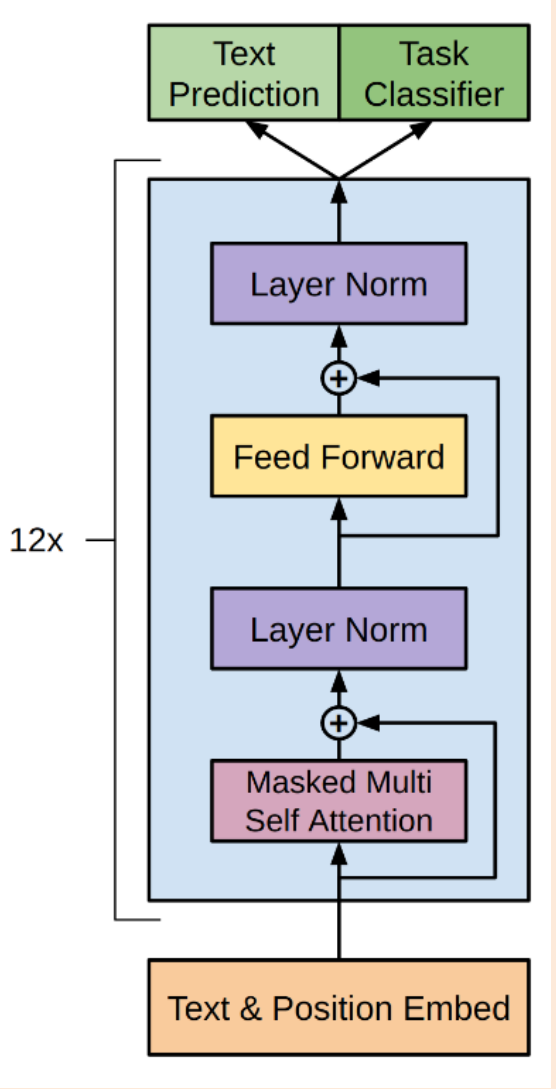
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are 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!

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:

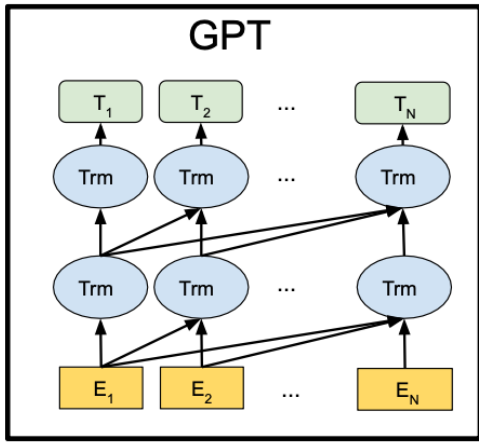
$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \texttt{softmax}(h_n W_e^T) \end{aligned} \quad (2)$$

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

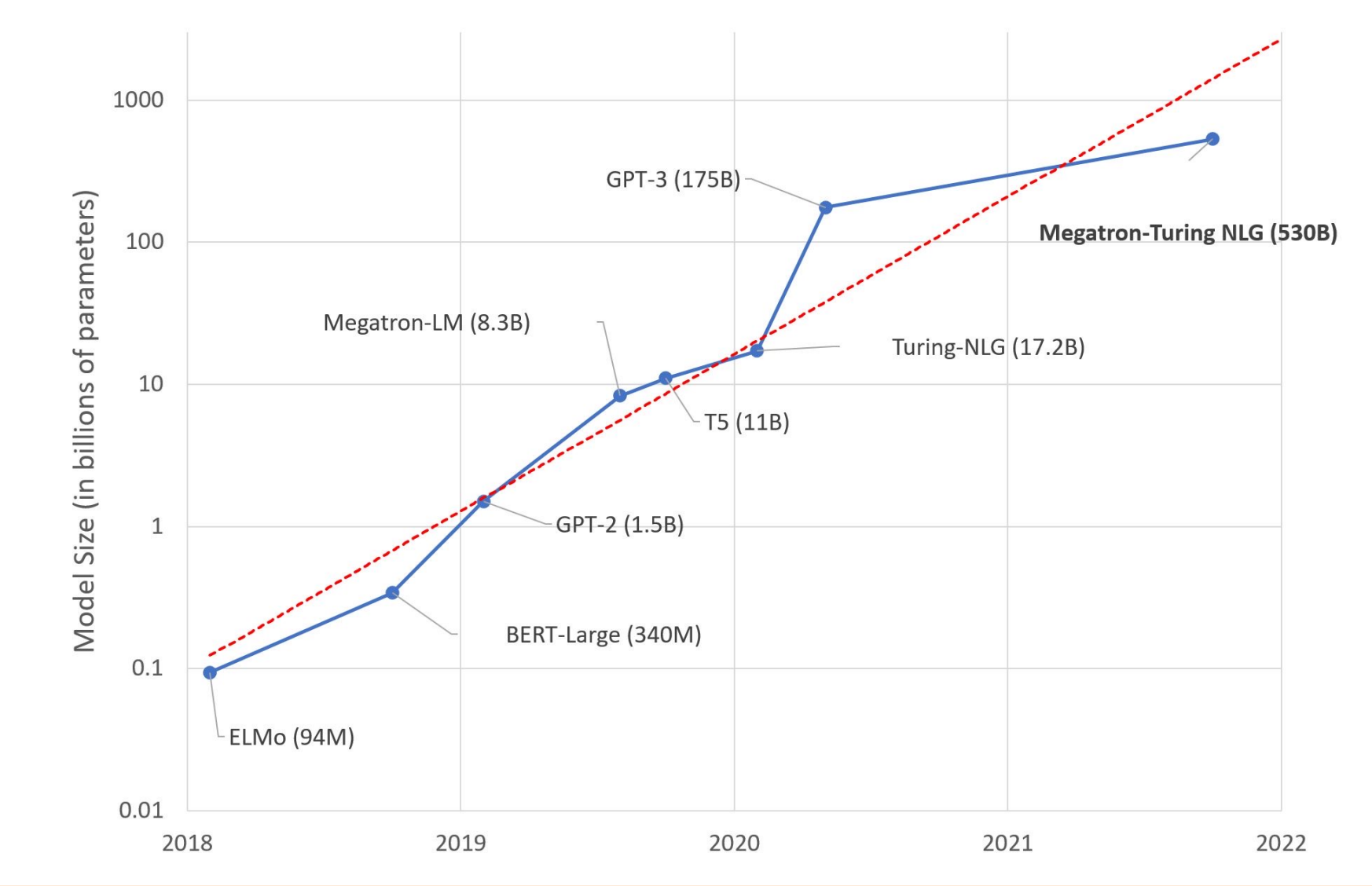
We use a Transformer (Vaswani et al., 2017) based architecture for our LMs. The model largely follows the details of the OpenAI GPT model (Radford et al., 2018) with a few modifications. Layer normalization (Ba et al., 2016) was moved to the input of each sub-block, similar to a pre-activation residual 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!

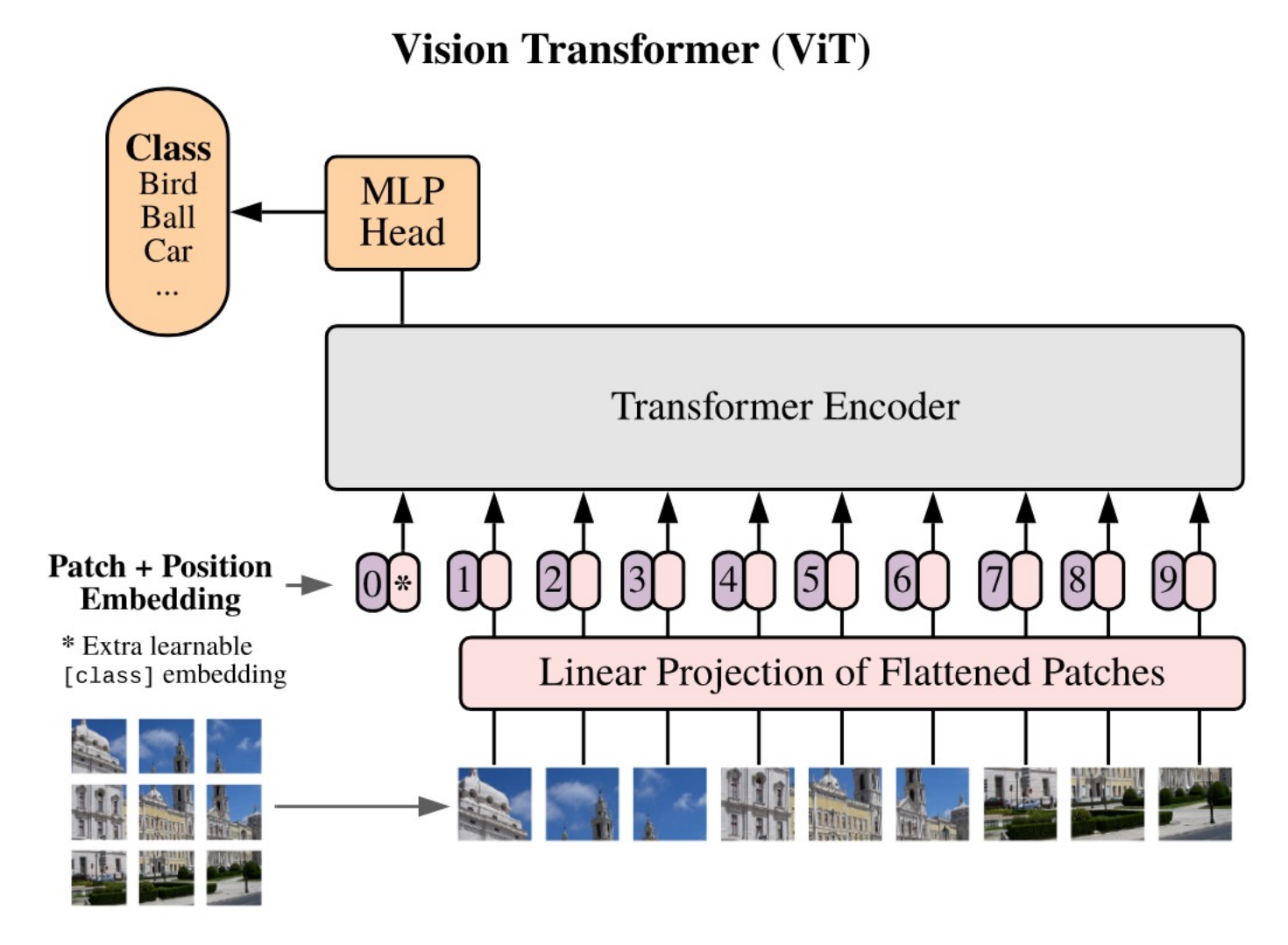
Bigger and bigger



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

bonus!

Vision Transformers

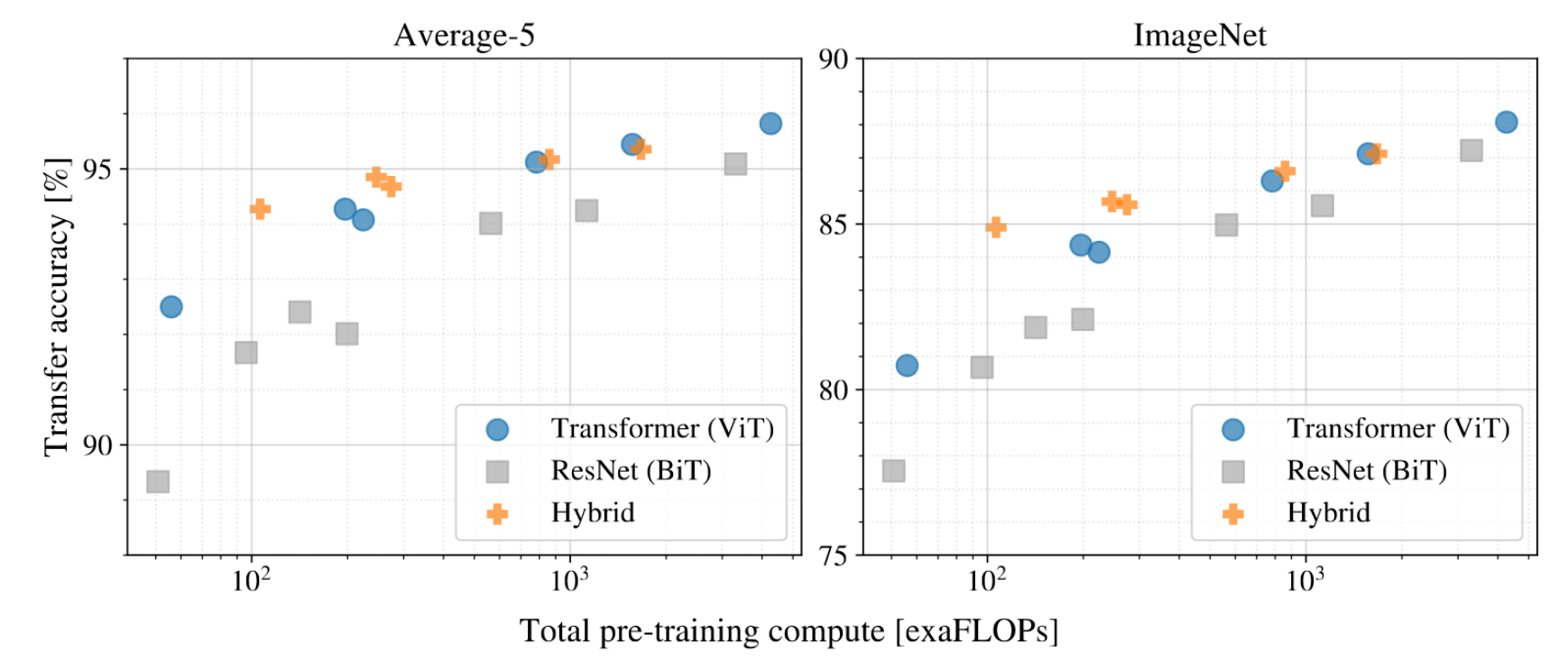


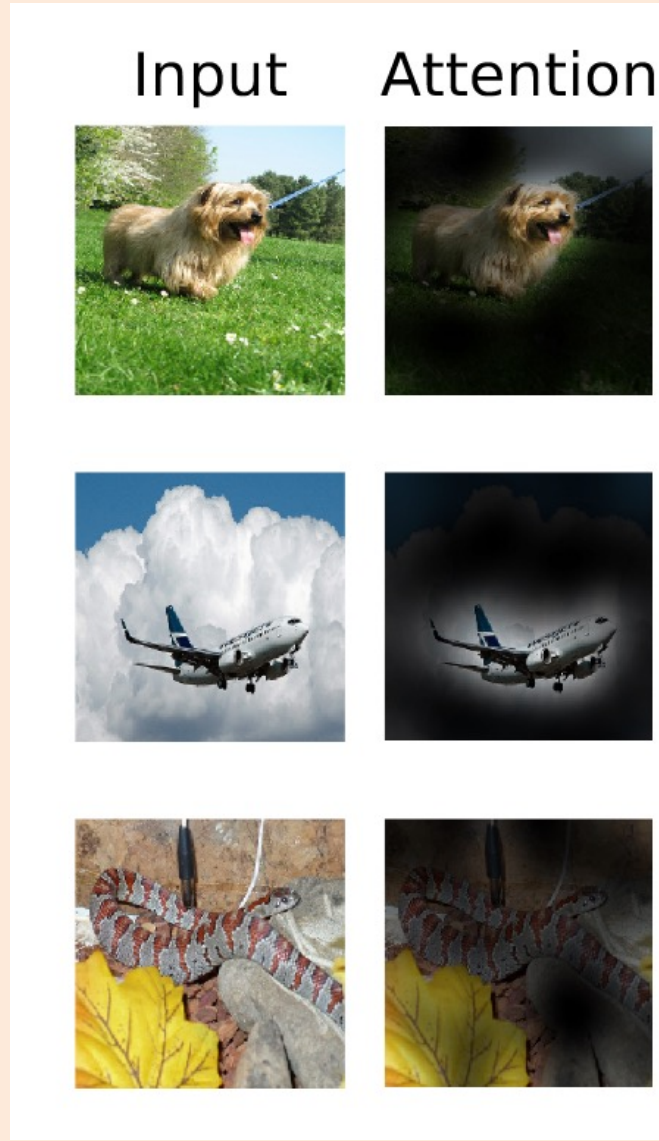
https://arxiv.org/abs/2010.11929

bonus!

Vision Transformers

- Usually outperform CNNs if you have enough data





https://arxiv.org/abs/2010.11929

bonus!

MLPs on patches might be enough

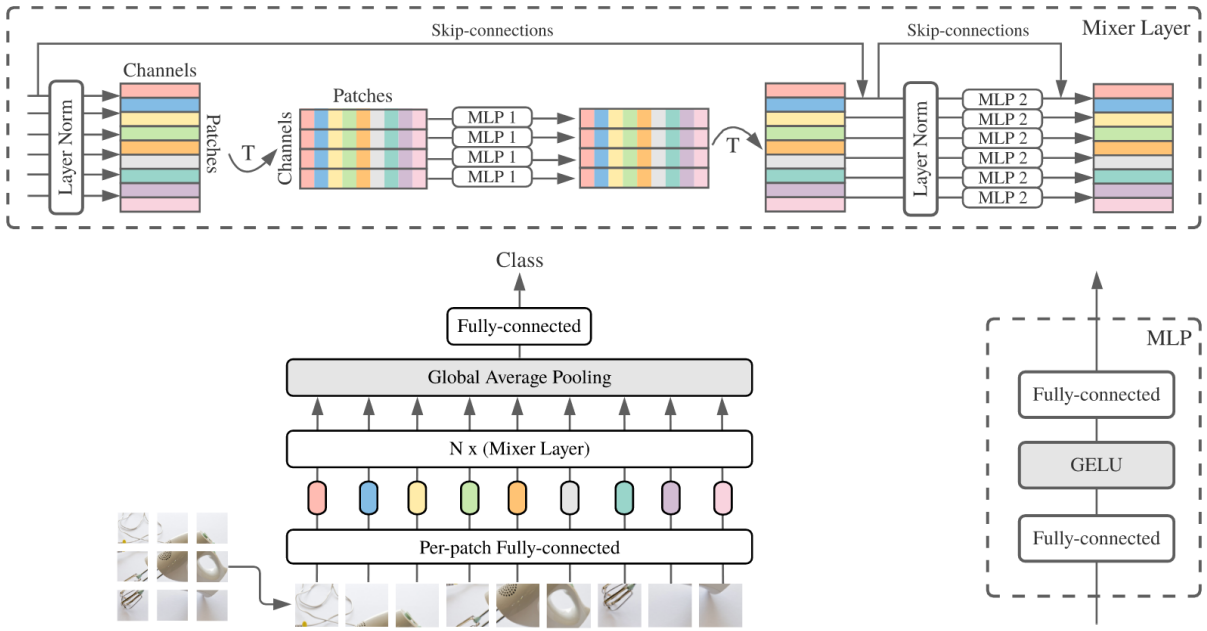
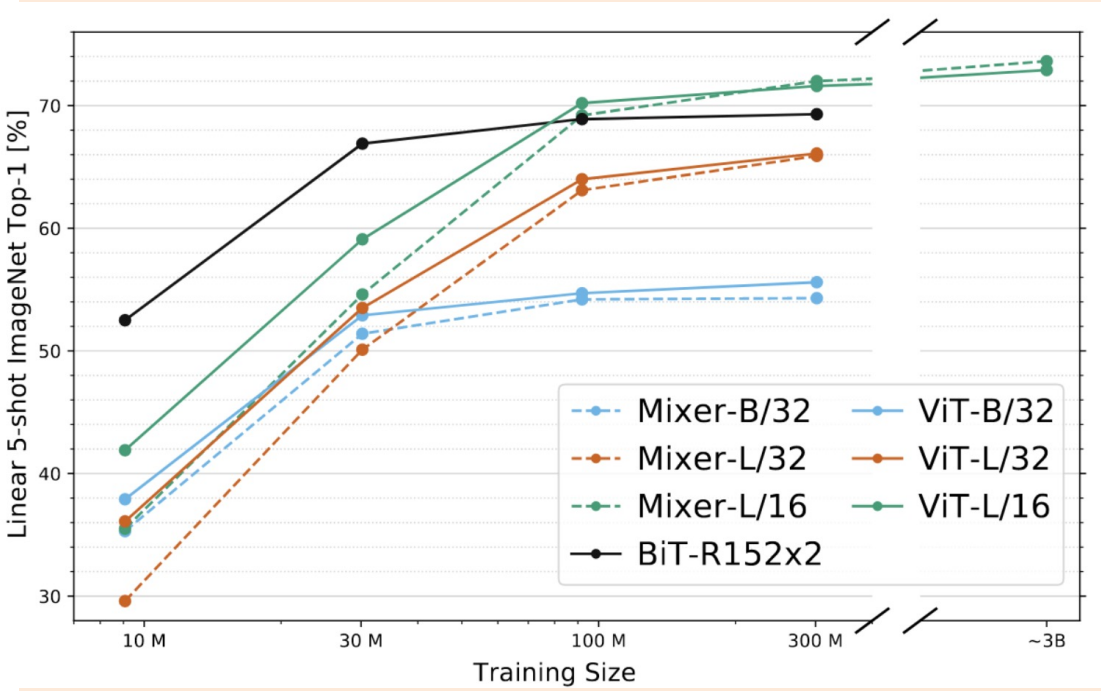


Figure 1: MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, and layer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

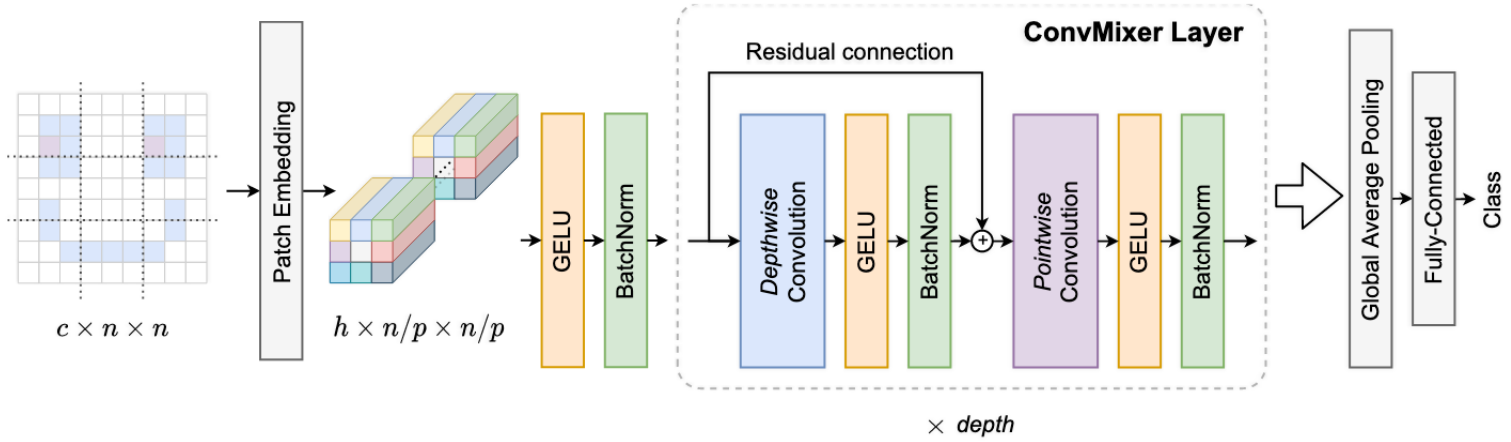


Figure 2: ConvMixer uses "tensor layout" patch embeddings to preserve locality, and then applies d copies of a simple fully-convolutional block consisting of *large-kernel* depthwise convolution followed by pointwise convolution, before finishing with global pooling and a simple linear classifier.

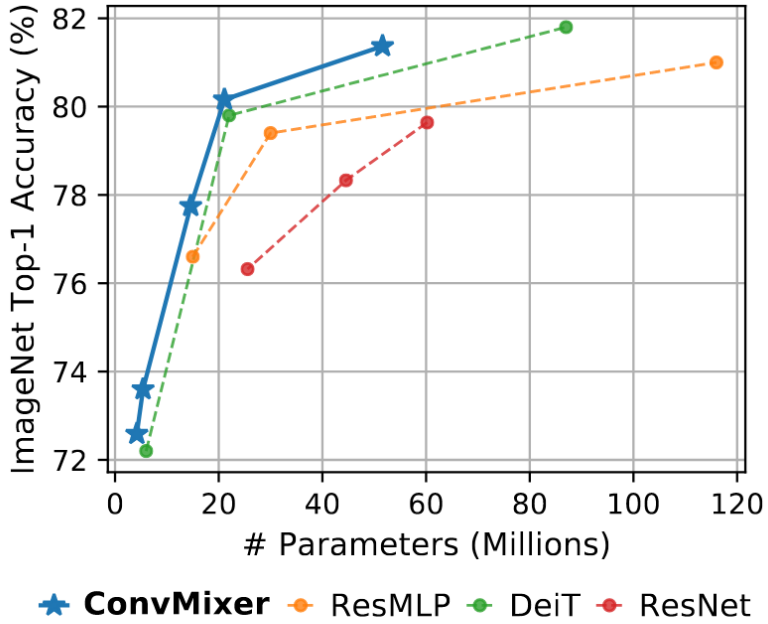


Figure 1: Accuracy *vs.* parameters, trained and evaluated on ImageNet-1k.

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
- Can also do these kinds of combinations in other domains

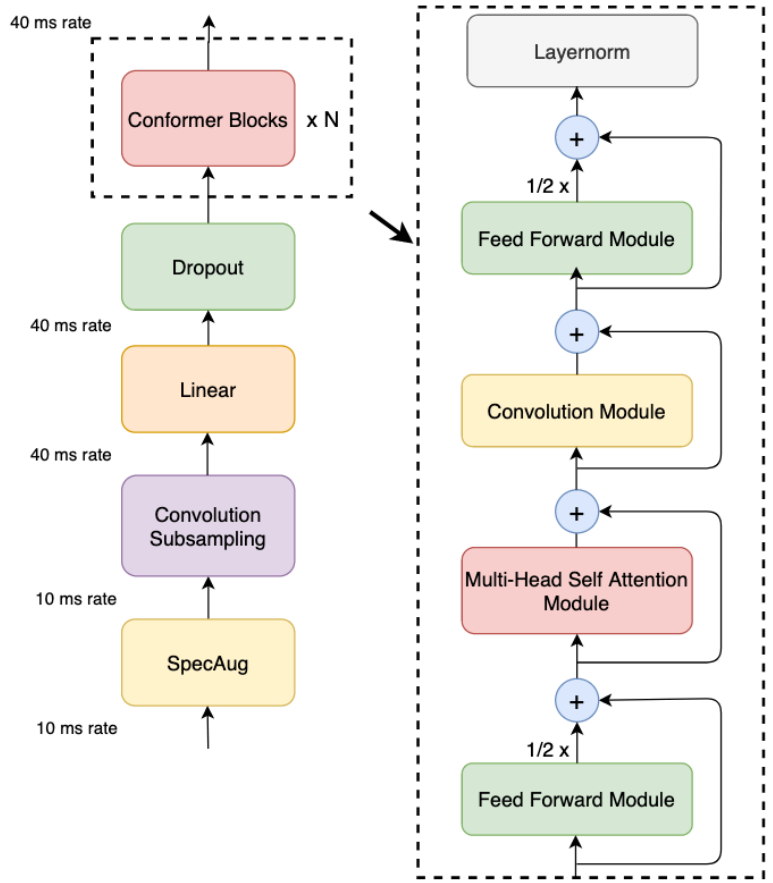


Figure 1: ***Conformer encoder model architecture.*** *Conformer comprises of two macaron-like feed-forward layers with half-step residual connections sandwiching the multi-headed self-attention and convolution modules. This is followed by a post layernorm.*

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: everyone’s favourite distribution.