RNNs (using LSTMs and GRUs) vs. Transformers

- **Long-term dependencies**
  - RNNs have problems in dealing with long-term dependencies between words that are spread far apart in a long sentence;
  - Transformers do not have the above problem, as long as the long-term dependencies are in the range of the maximum allowed input length;
- **Parallel computation**
  - RNNs process the input sequence sequentially one token at a time: before starting the computation for time step t the computation for time step $t-1$ should be completed; training and inference are slowed down;
  - Transformers can process in parallel all the tokens in the input sequence (as well as all sequences in a batch) exploiting matrix multiplication (very efficient in GPUs);
- **Context fragmentation**
  - Attention can only deal with fixed-length sequences, so long sequences should be split into a certain number of segments (chunks) before being fed into a Transformer;
  - RNNs do not have the above problem;
- **Out-of-distribution generalization**
  - Transformers are not able to implement recurrent rules (if they exist in data), so in principle they do not generalize well to sequences longer than the training ones;
  - RNNs in principle can learn recurrent rules (if they exist in data);

Transformers: additional issues

- **Attention in Transformers scales quadratically with the length of the input sequence**

  D: hidden dimension of the transformer (size embeddings times # heads);

  T: (maximum) length of the input sequence;

Module	Complexity
self-attention	$O(T^2 \cdot D)$
position-wise FFN	$O(T \cdot D^2)$

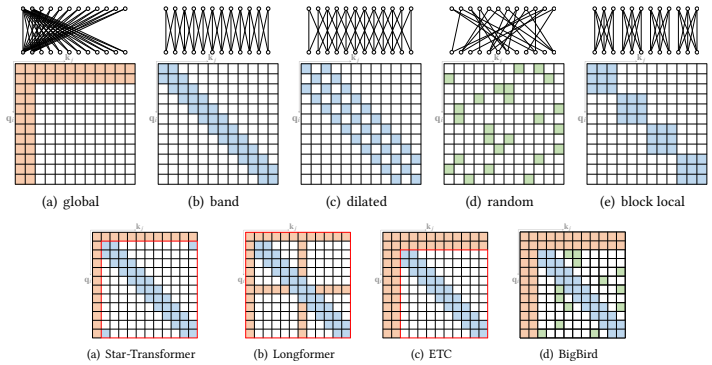
  - short sequences:
    - the hidden dimension D dominates the complexity and FFN is the bottleneck;
  - long sequences:
    - the sequence length T dominates the complexity and self-attention becomes the bottleneck;
    - the computation of self-attention requires to store a $T \times T$ attention distribution matrix is stored, making the computation of Transformer infeasible for long-sequence scenarios.

- **Overfitting: hard to train on small-scale data**
  - Self-attention does no assume any structural bias (not even order) over inputs.

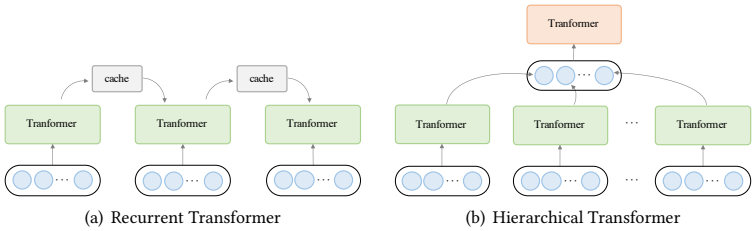
X-formers

X-formers improve the vanilla Transformer from different perspectives

- **Model Efficiency**
  - sparse attention variants;

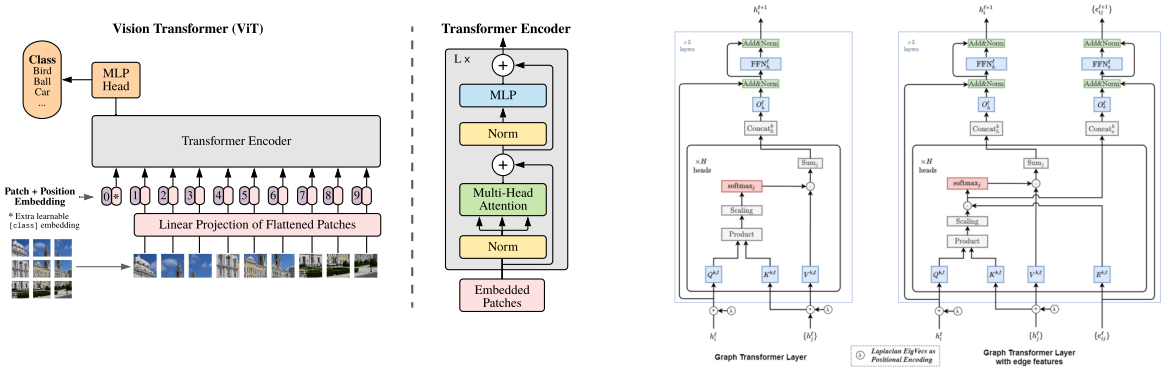


  - Divide-and-conquer methods (e.g., recurrent and hierarchical mechanism);



X-formers

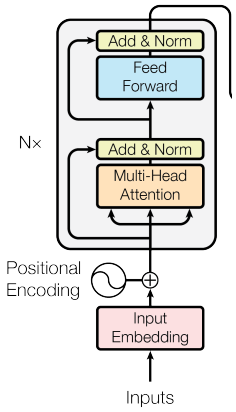
- **Model Generalization**
  - structural bias or regularization;
  - pre-training on large-scale unlabeled data;
- **Model Adaptation**
  - Transformers adapted to specific downstream tasks and applications;



Model Pre-training

Word2vec embeddings:

- Obtained by training only on text data (no specific task supervision)
- A dress for all seasons… Think about the sentence ``I *record* the *record*'': same embedding for two different meanings!
- Word embeddings are obtained by collapsing all contexts…
- .. but the learning task where they are used require to discriminate among different contexts!



Model Pre-training:

- Instead of just learning meaningful (distributed) representations for words/tokens, pre-train also the parameters of the whole model by just using text data (no specific task supervision)
- How can this be done ?
  hide parts of the input from the model, and then train the model to reconstruct those parts!

Model Pre-training

Same principle, three different approaches:

1. Learn a Language Model via a decoder
   Language Model → learn $P(w_n | w_1 w_2 w_3 ... w_{n-1})$

2. Learn embeddings that are context dependent via an encoder
   Learn a Masked Language Model

3. Combine the two approaches above
   Learn sequence-to-sequence with Span Corruption

Model Pre-training can be used to build strong:

- representations of language
- parameter initializations for very effective NLP models
- probability distributions over language from which to sample

Language Model via a decoder

sentence

Task: learn *f*(<bos> w_1 w_2 … w_{N-1} w_N <eos>)

Idea:

1. Train a Decoder to predict the next word/token in the sentence

   Decoder(<bos>) = w_1

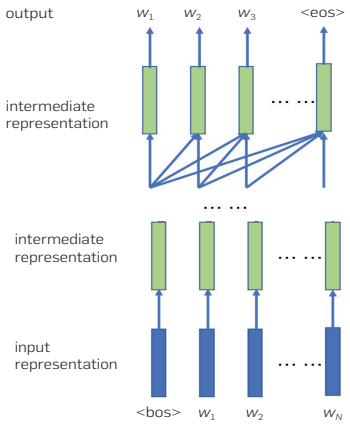
   Decoder(<bos> w_1) = w_2

   Decoder(<bos> w_1 w_2) = w_3

   Decoder(<bos> w_1 w_2 w_3) = w_4

   …

   Decoder(<bos> w_1 w_2 … w_N) = <eos>



2. Fine-tune the obtained model to perform the learning task

Generative Pretrained Transformer (GPT)

1. Given a corpus of unsupervised tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$ train a multi-layer Transformer Decoder to maximize

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

with

number of layers

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

where $U = (u_{-k}, \ldots, u_{-1})$ is the contex vector of tokens

2. Given a corpus of supervised tokens $\mathcal{C}$ with instances $x^1, \ldots, x^m \rightarrow y$ add on top of the Decoder a linear classifier

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

and fine-tune the whole model (Decoder + linear classifier):

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y | x^1, \ldots, x^m)$$

Generative Pretrained Transformer (GPT)

Transformer decoder with 12 layers; 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers; Byte-pair encoding with 40,000 merges; Trained on BooksCorpus: over 7000 unique books

Linear classifier applied to representation [EXTRACT] token

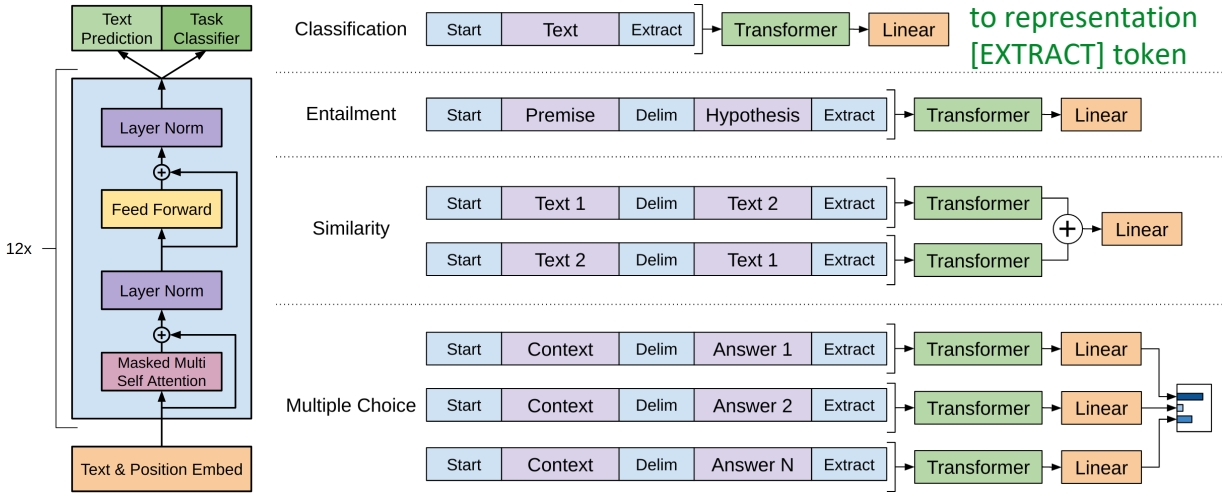


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

GPT-2

Larger version of GPT (1.5 billion parameters) trained on more data, was shown to produce relatively convincing samples of natural language

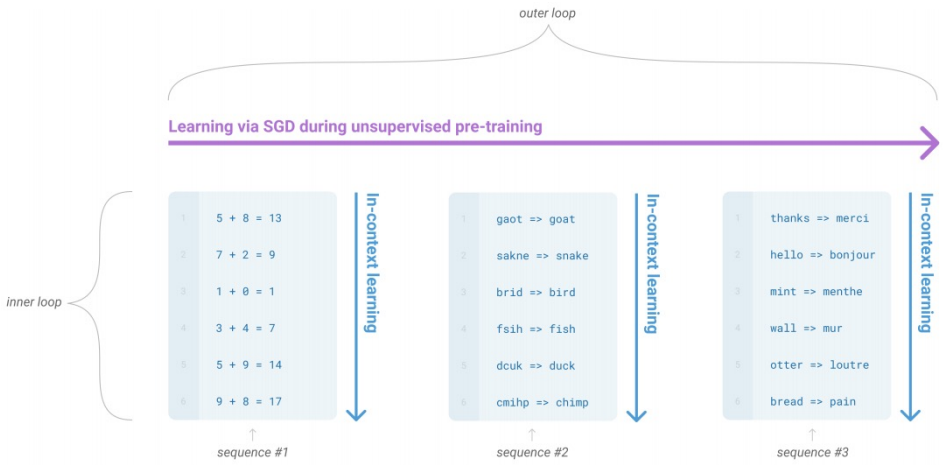
Conditional generation on an out-of-distribution context by GPT-2

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

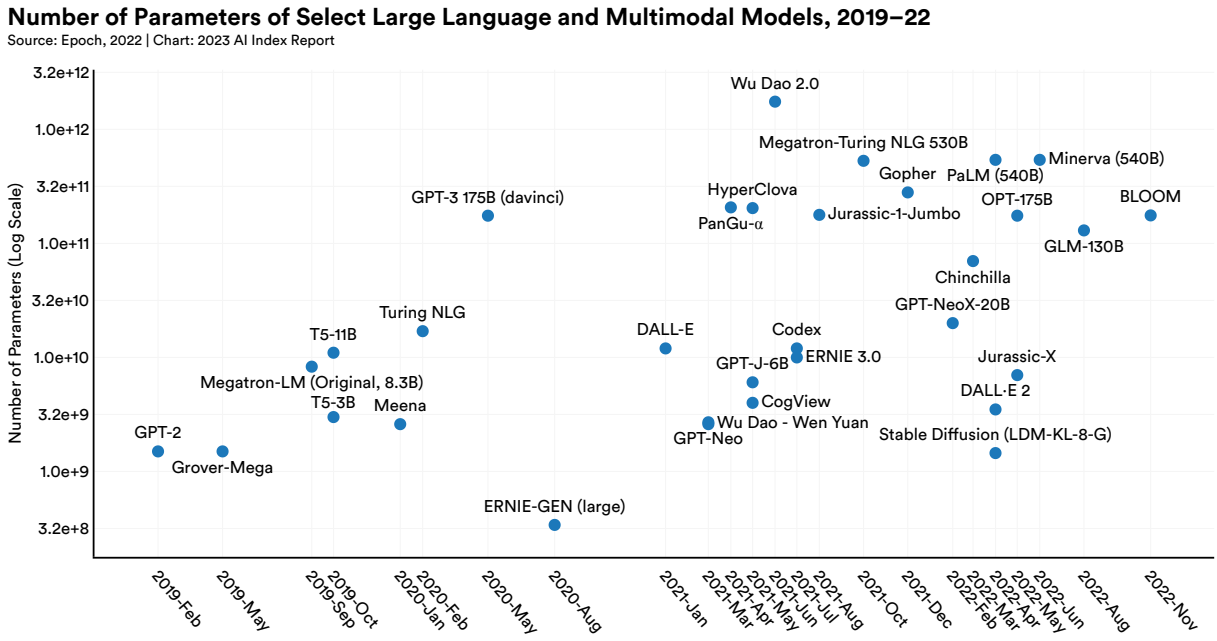
GPT-3

Huge model: 175 billion parameters

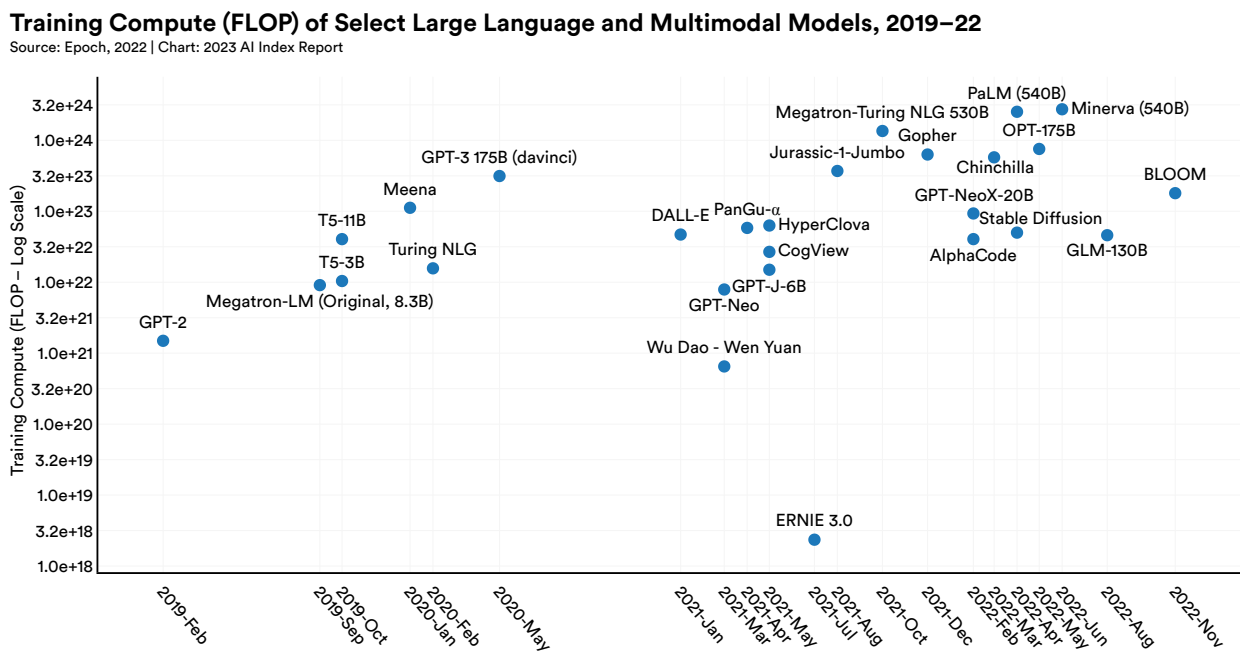
- Learning **without gradient steps**: just provide few examples in the context
- Possible because conditional distribution "continues" the pattern present in the examples in the context



Recent Large Models



Large Models: Compute



Masked Language Model via encoder

Remark: Decoders only take left context, while encoders are bidirectional (both left and right context)

But how to train an encoder to perform Language Modeling ?

Solution: Masked Language Model: replace some fraction of words in the input with a special [MASK] token; predict these words

s sentence, s_{mask} masked sentence → learn $P(s|s_{mask})$

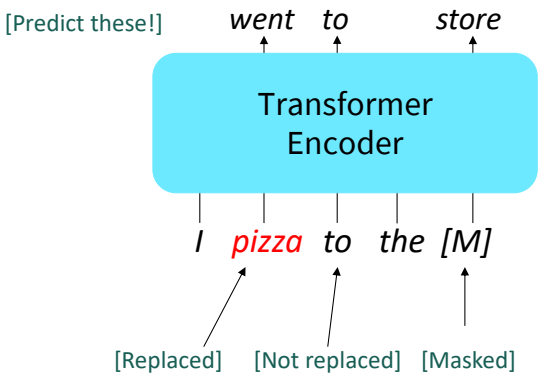
Example:

original sentence: *I went to the store*

masked sentence: *I [M] to the [M]*

BERT: Bidirectional Encoder Representations from Transformers

- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



Two models were released:

- BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
- BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.

Trained on:

- BooksCorpus (800 million words)
- English Wikipedia (2,500 million words)

BERT: Bidirectional Encoder Representations from Transformers

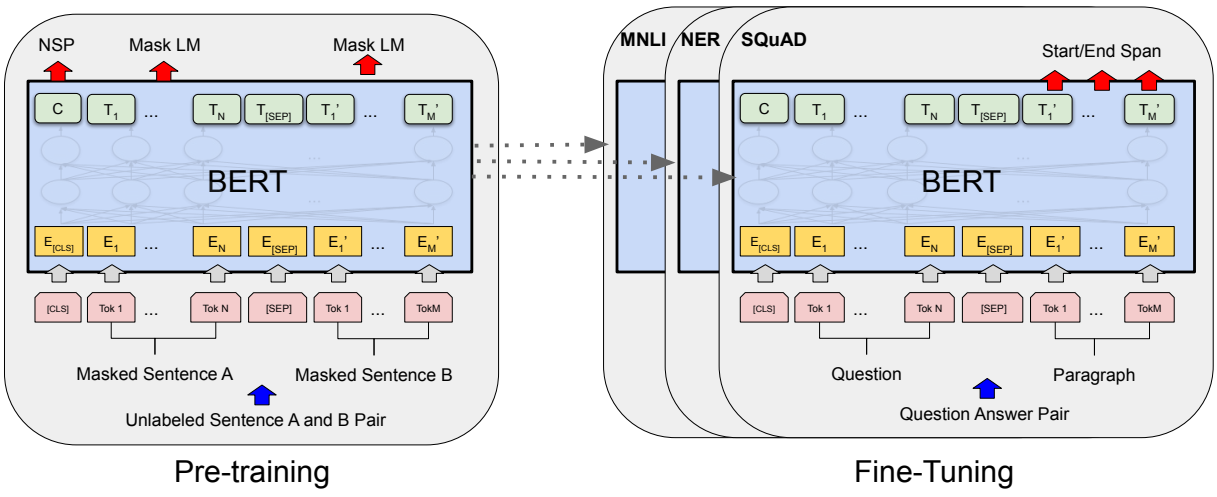


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are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

Language Model via encoder-decoder

Remark: Encoders don't naturally lead to effective autoregressive (1-word-at-a-time) generation methods as decoders do

Idea: use both encoders and decoders

How: Language Model with Span Corruption (T5)

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

Original text

Thank you for inviting me to your party last week.

This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.