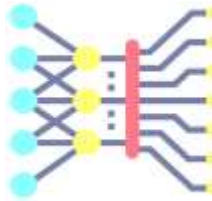


CS60010: Deep Learning

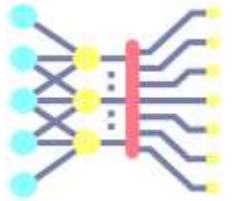
Spring 2023

Sudeshna Sarkar

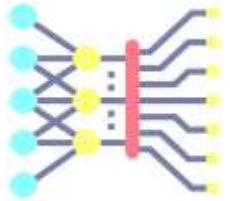
Transformer- Part 3

Sudeshna Sarkar

16 Mar 2023



Pre-training



CS60010: Deep Learning

Spring 2022

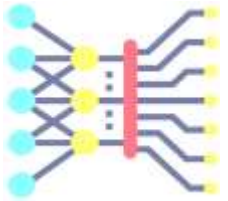
Sudeshna Sarkar

BERT etc

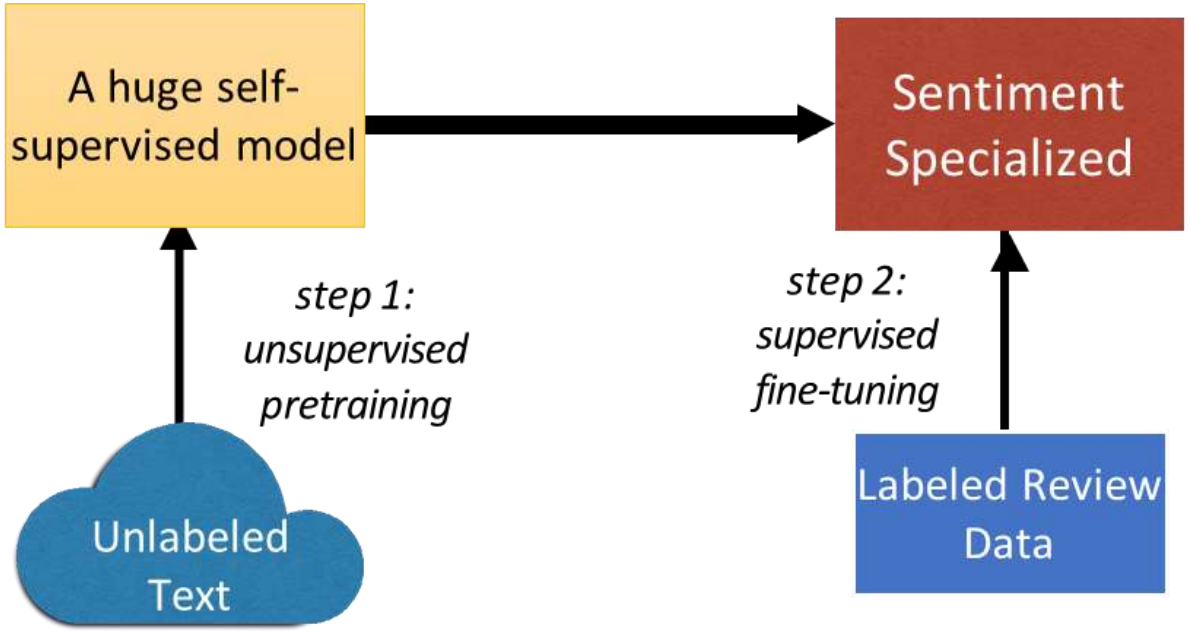
Sudeshna Sarkar

11 May 2022

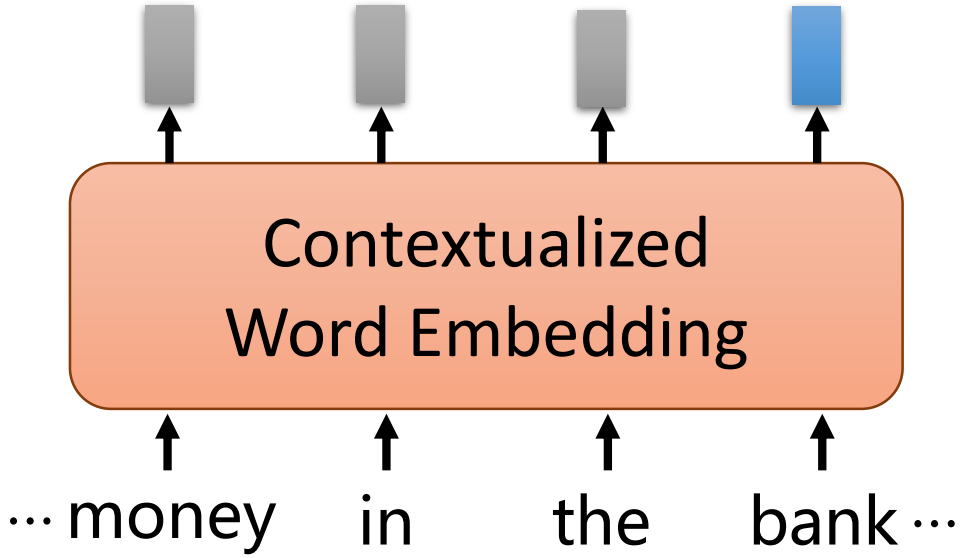
Transfer Learning

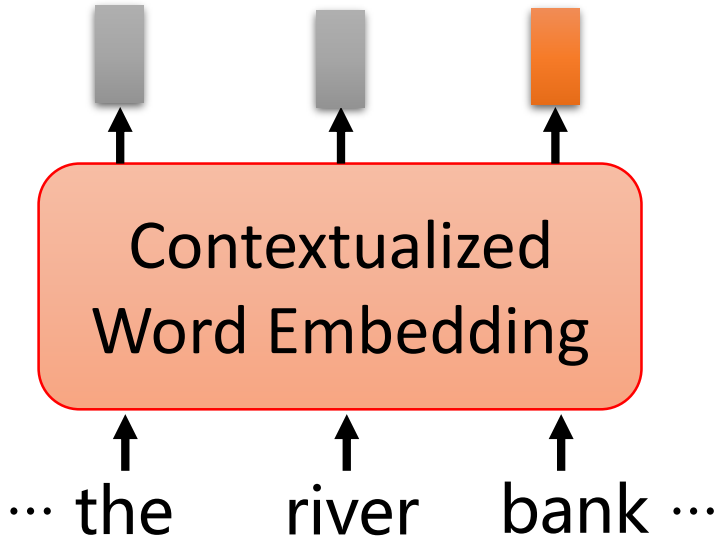


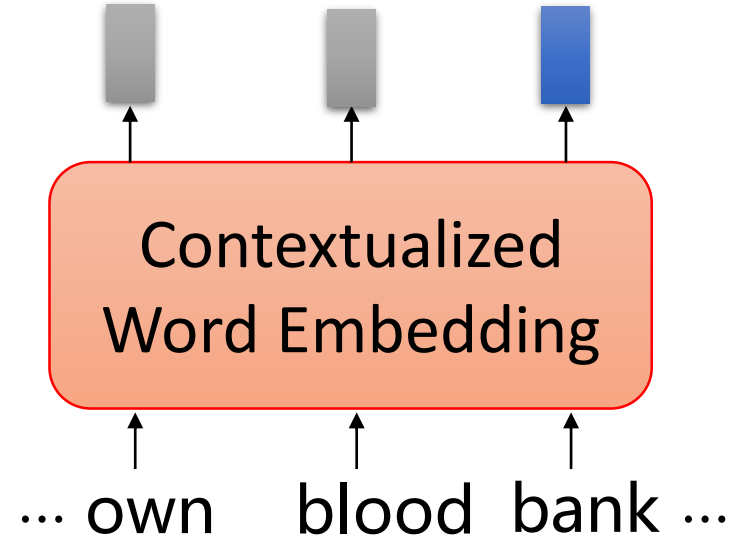
- Leverage unlabeled data to cut down on the number of labeled examples needed.
- Take a network trained on a task for which it is easy to generate labels, and adapt it to a different task for which it is harder.
- Train a really big language model on billions of words, transfer to every NLP task!



Contextualized Word Embedding

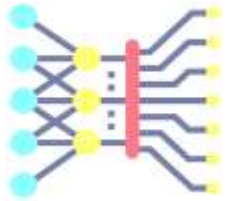






Train contextual representations on text corpus

Uni-directional language models

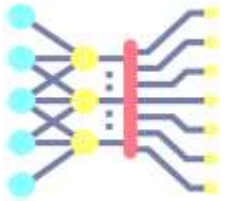


- Why are LMs unidirectional?
  - Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - Reason 2: Words can “see themselves” in a bidirectional encoder.

- Language models only use left context *or* right context, but language understanding is bidirectional.

Slide from Jacob Delvin

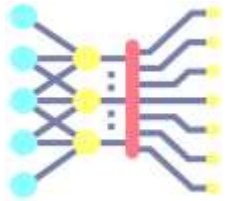
Word structure and subword models



- We assume a fixed vocab of tens of thousands of words, built from the training set.
- All novel words seen at test time are mapped to a single UNK.

  - taaaasty
  - laern
  - tarnsformerify

The byte-pair encoding algorithm



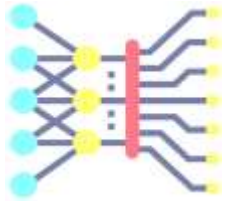
1. Start with a vocabulary containing only characters and an “end-of-word” symbol.
2. Using a corpus of text, find the most common adjacent characters “a,b”; add “ab” as a subword.
3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, words are split into as many subwords as they have characters.

- taa## aaa## sty
- la## ern##
- Transformer## ify

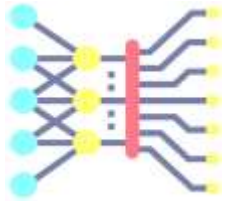
pretraining



1. pretrained word embeddings
2. Pretraining whole models

- In modern NLP:
  - All (or almost all) parameters in NLP networks are initialized via **pretraining**.
  - Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.
- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP
  - models.
  - **Probability distributions** over language that we can sample from

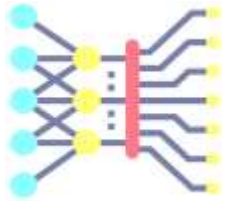
Pretraining through language modeling



- Recall the language modeling task:
  - Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.

- **Pretraining through language modeling:**
  - Train a neural network to perform language modeling on a large amount of text.
  - Save the network parameters.

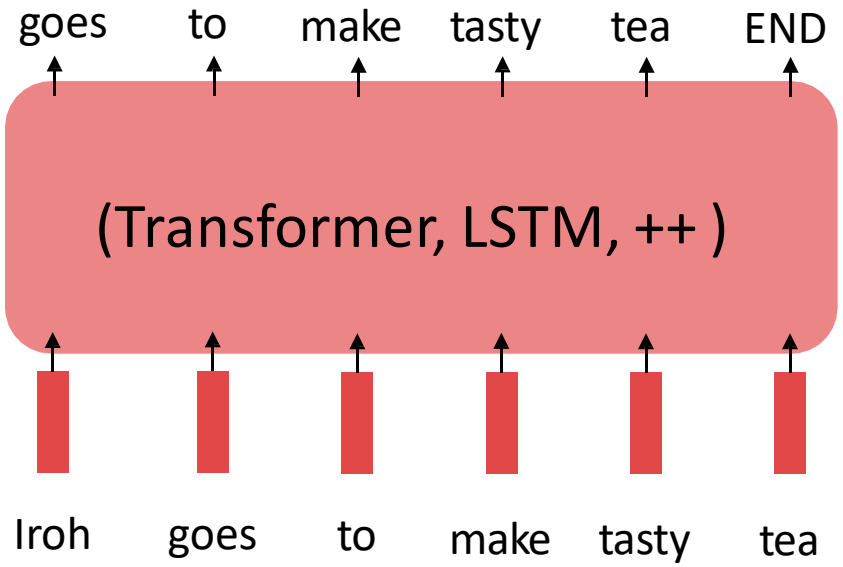
The Pretraining / Finetuning Paradigm



Pretraining can improve NLP applications by serving as parameter initialization.

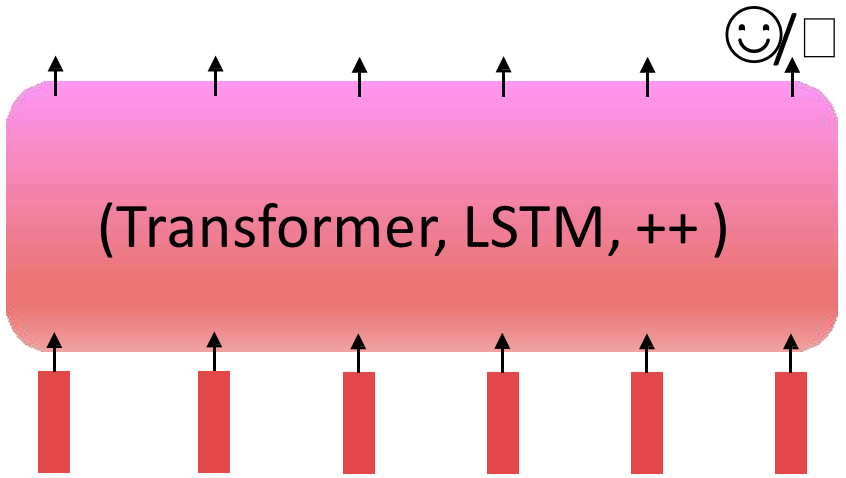
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



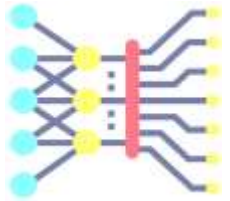
Step 2: Finetune (on your task)

Not many labels; adapt to the task!

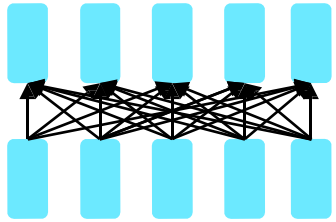


… the movie was …

Pretraining for three types of architectures

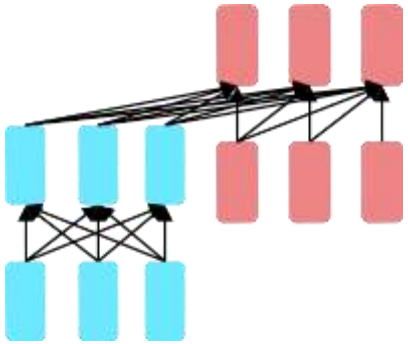


The neural architecture influences the type of pretraining, and natural use cases.



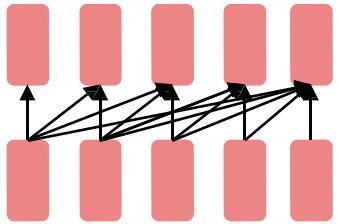
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

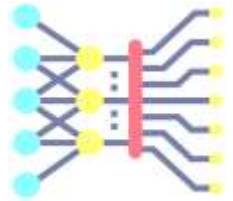
- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining encoders: what pretraining objective to use?

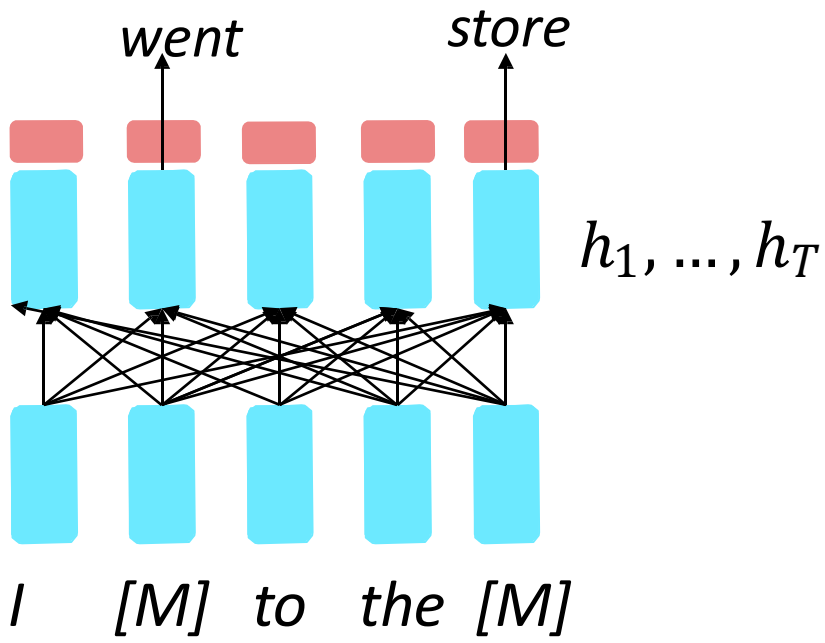


So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

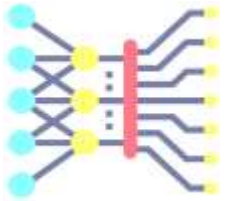
$$h_1, \dots, h_T = \text{Encoder}(w_1, w_2, \dots, w_T)$$

Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$
Called **Masked LM**.



[Devlin et al., 2018]

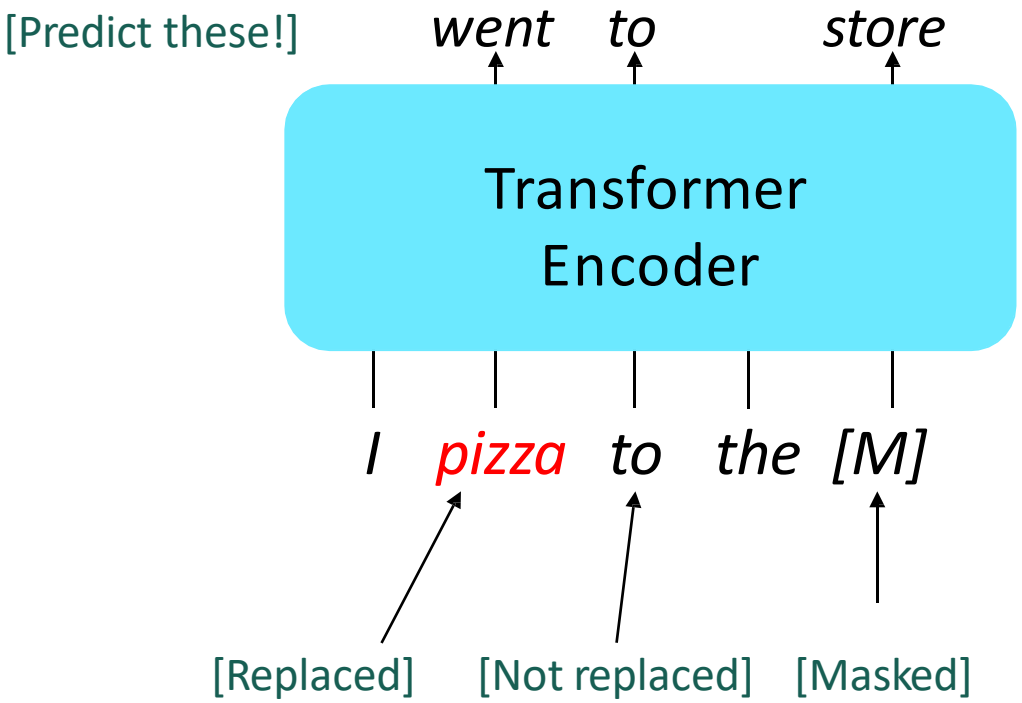
BERT: Bidirectional Encoder Representations from Transformers



Devlin et al., 2018 proposed the "Masked LM" objective and **released the weights of a pretrained Transformer**, a model they labeled BERT.

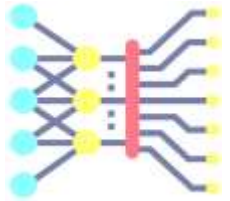
Some more details about Masked LM for BERT:

- 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!)

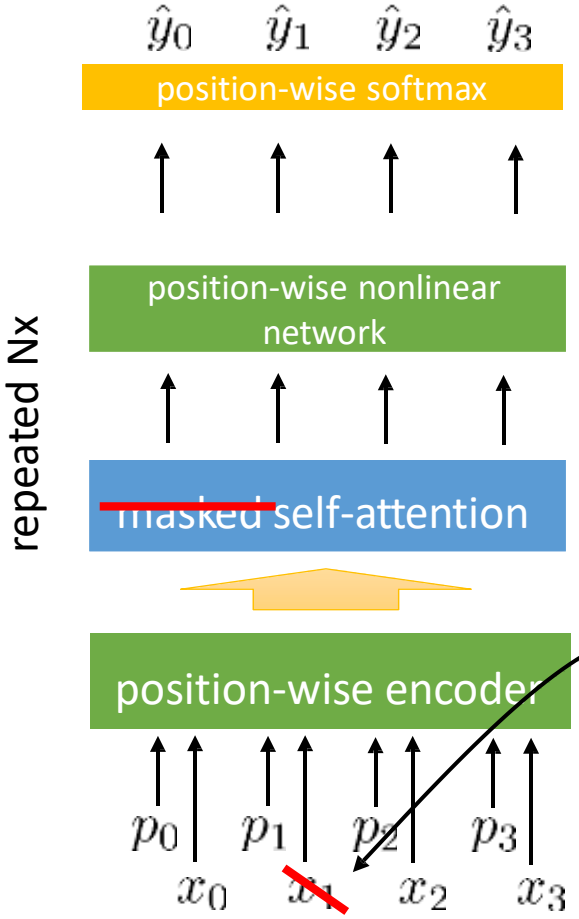


[Devlin et al., 2018]

Bidirectional transformer LMs



BERT is essentially the “encoder” part of a transformer with 15% of inputs replaced with [MASK]



Input: I [MASK] therefore I [MASK]

Output: I think therefore I am

Main idea: needing to predict missing words forces the model to “work hard” to learn a good representation

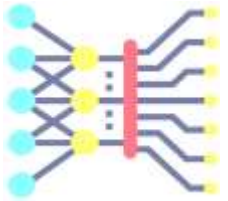
Without the need for masked self-attention!

This makes it **bidirectional**

randomly **mask out** some input tokens

mask = replace with [MASK]

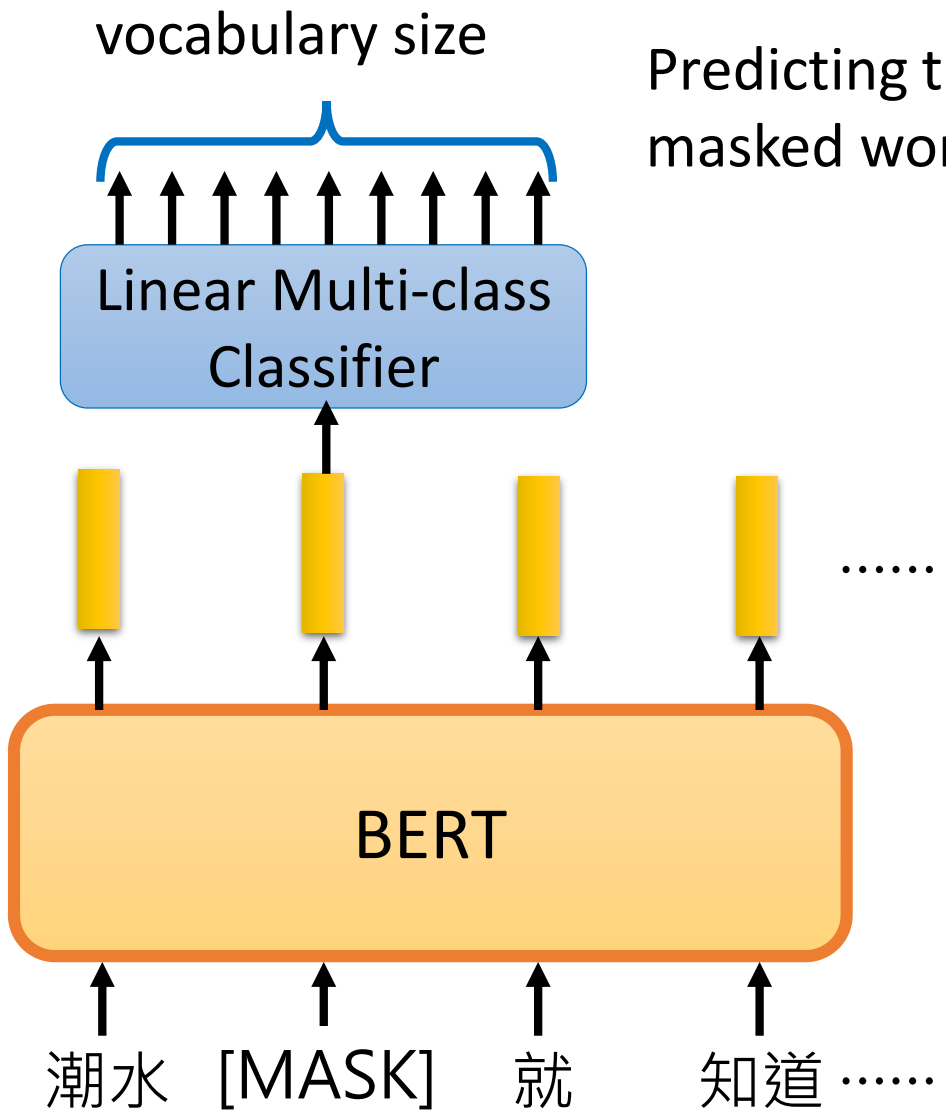
Training of BERT



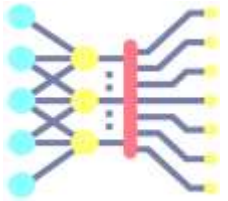
Approach 1:

Masked LM

Randomly replace 15% of the tokens with [MASK]



Training of BERT

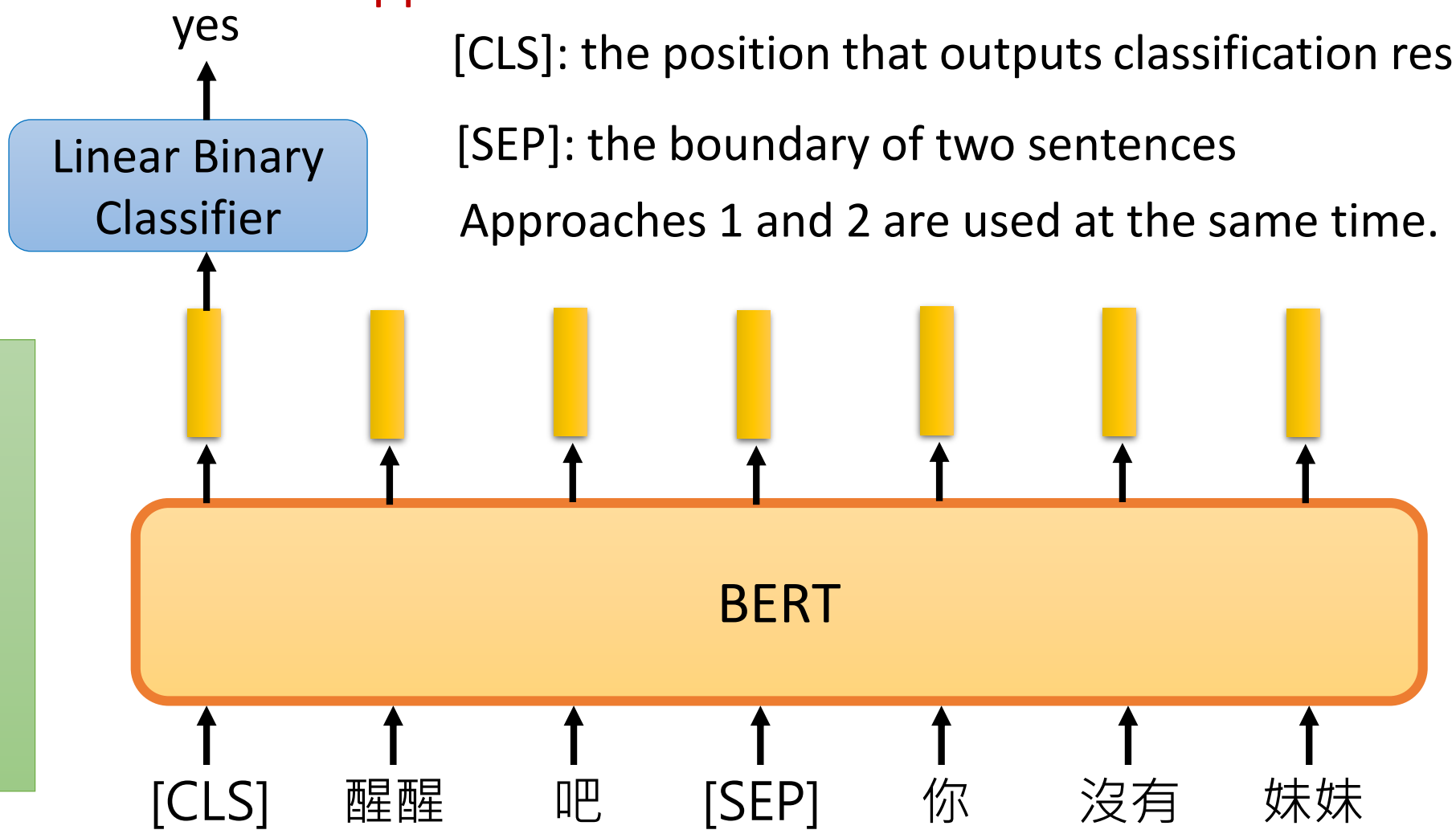


Approach 2: Next Sentence Prediction

[CLS]: the position that outputs classification results

[SEP]: the boundary of two sentences

Approaches 1 and 2 are used at the same time.

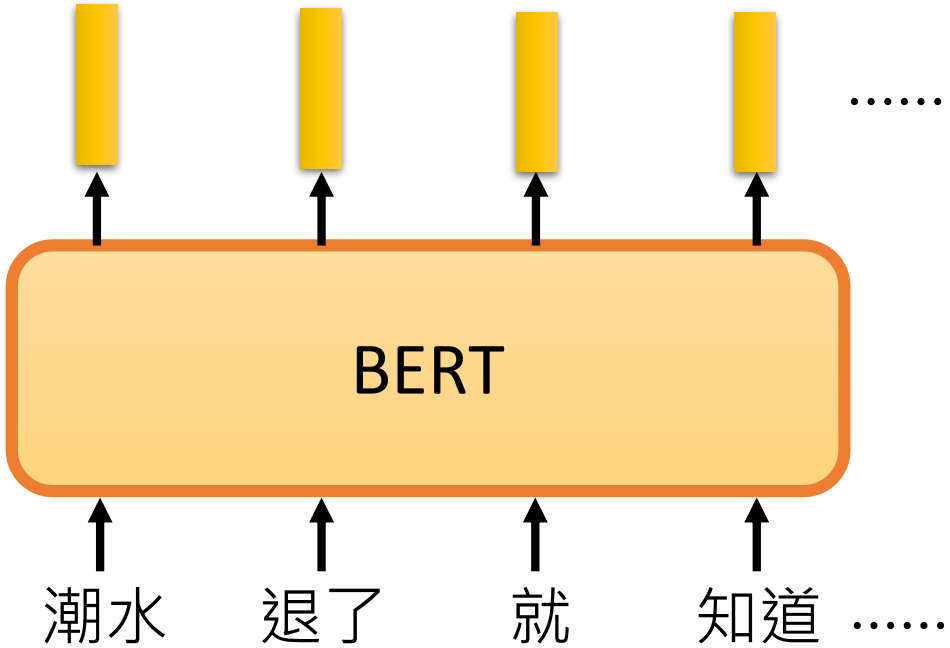


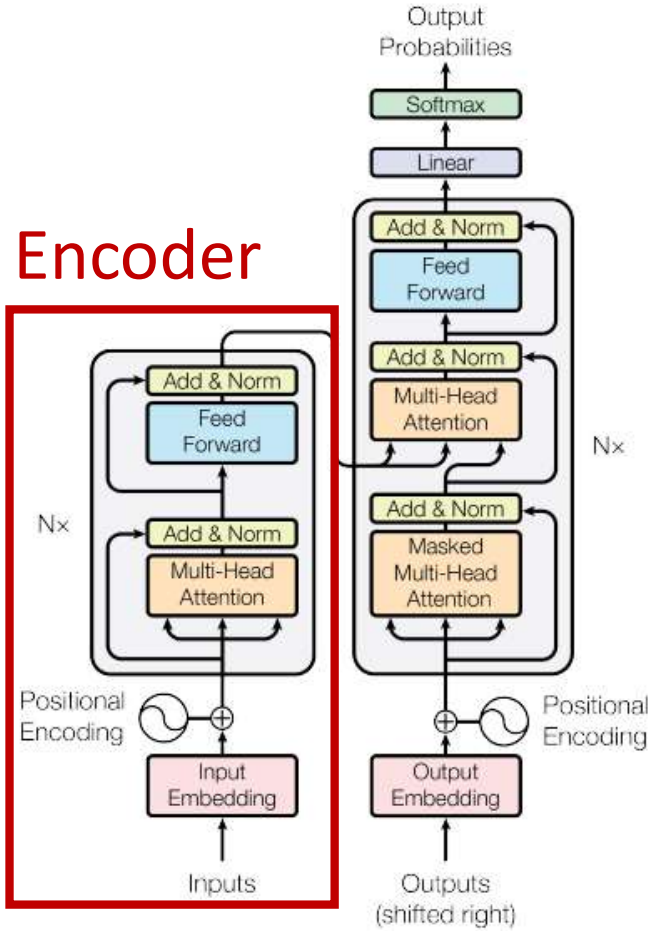
Randomly swap the order of the sentences 50% of the time

Bidirectional Encoder Representations from Transformers (BERT)

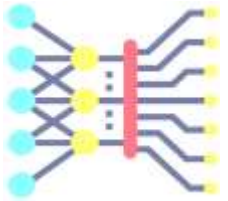
- BERT = Encoder of Transformer

Learned from a large amount of text without annotation

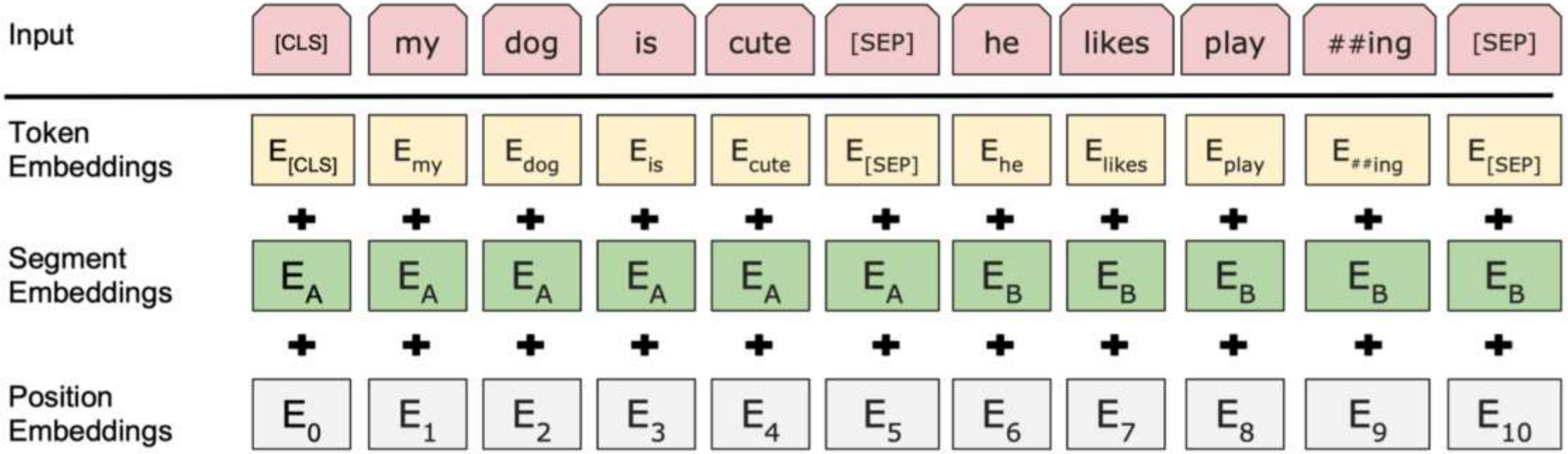




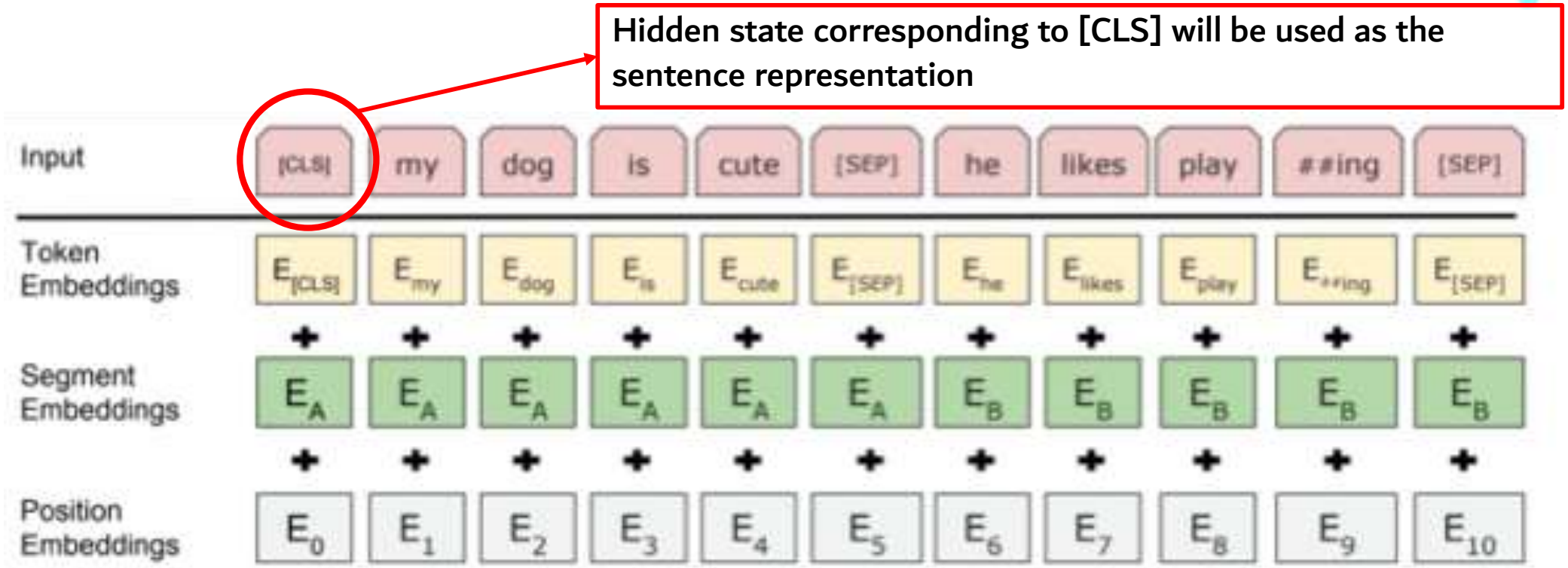
BERT



- Multi-layer self-attention (Transformer)
- Input: a sentence or a pair of sentences with a separator and subword representation



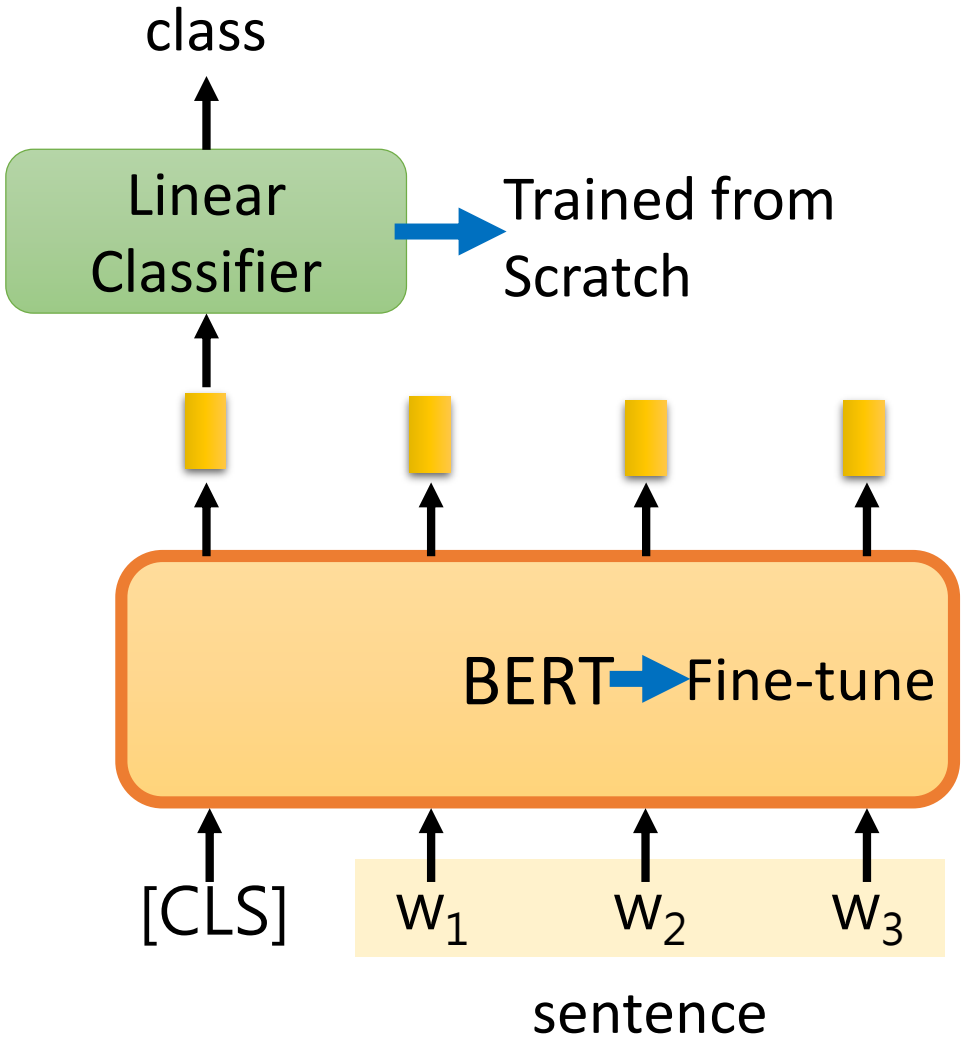
Input Representation



- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
- Single sequence is much more efficient.

Slide from Jacob Delvin

How to use BERT – Case 1

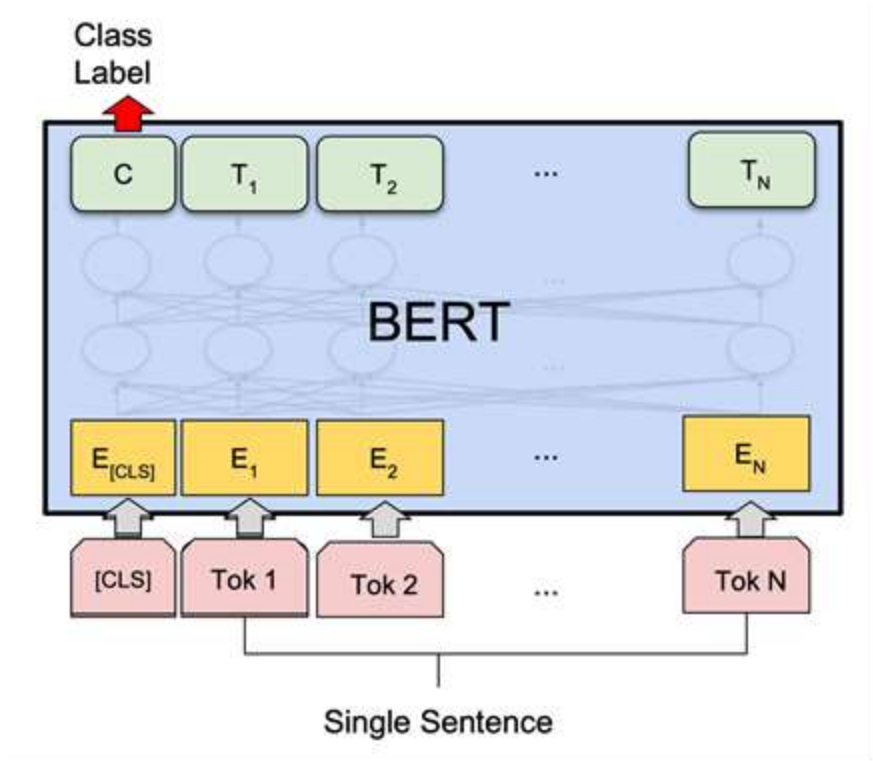


Input: single sentence, output: class

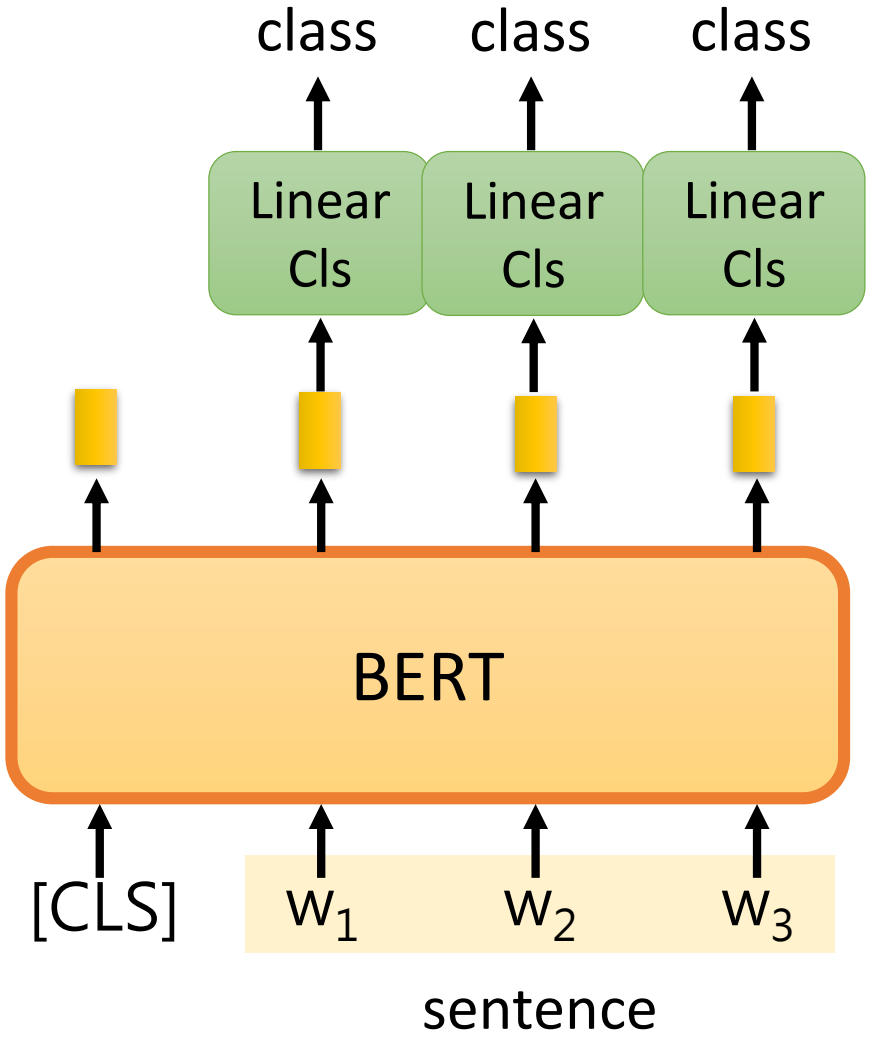
Example:
Sentiment analysis
Document Classification

Sentence Classification

CLS token is used to provide classification decision

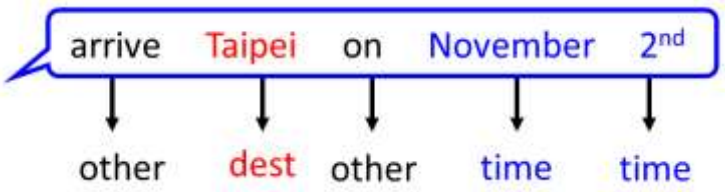


How to use BERT – Case 2

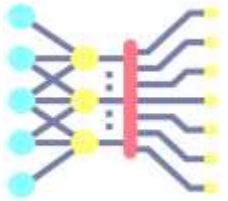


Input: single sentence, output: class of each word

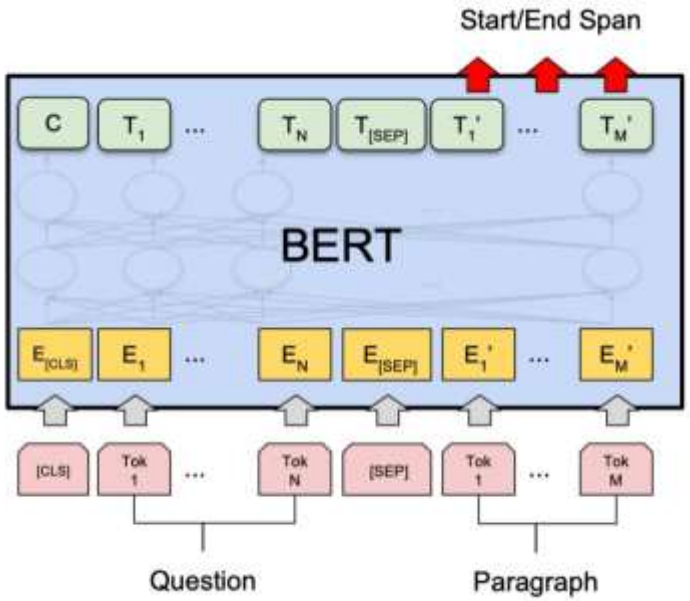
Example: Slot filling

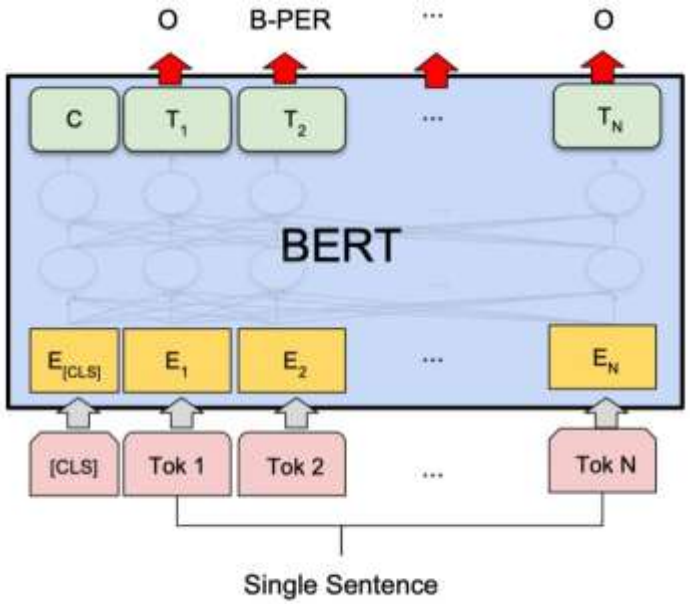


Tagging with BERT

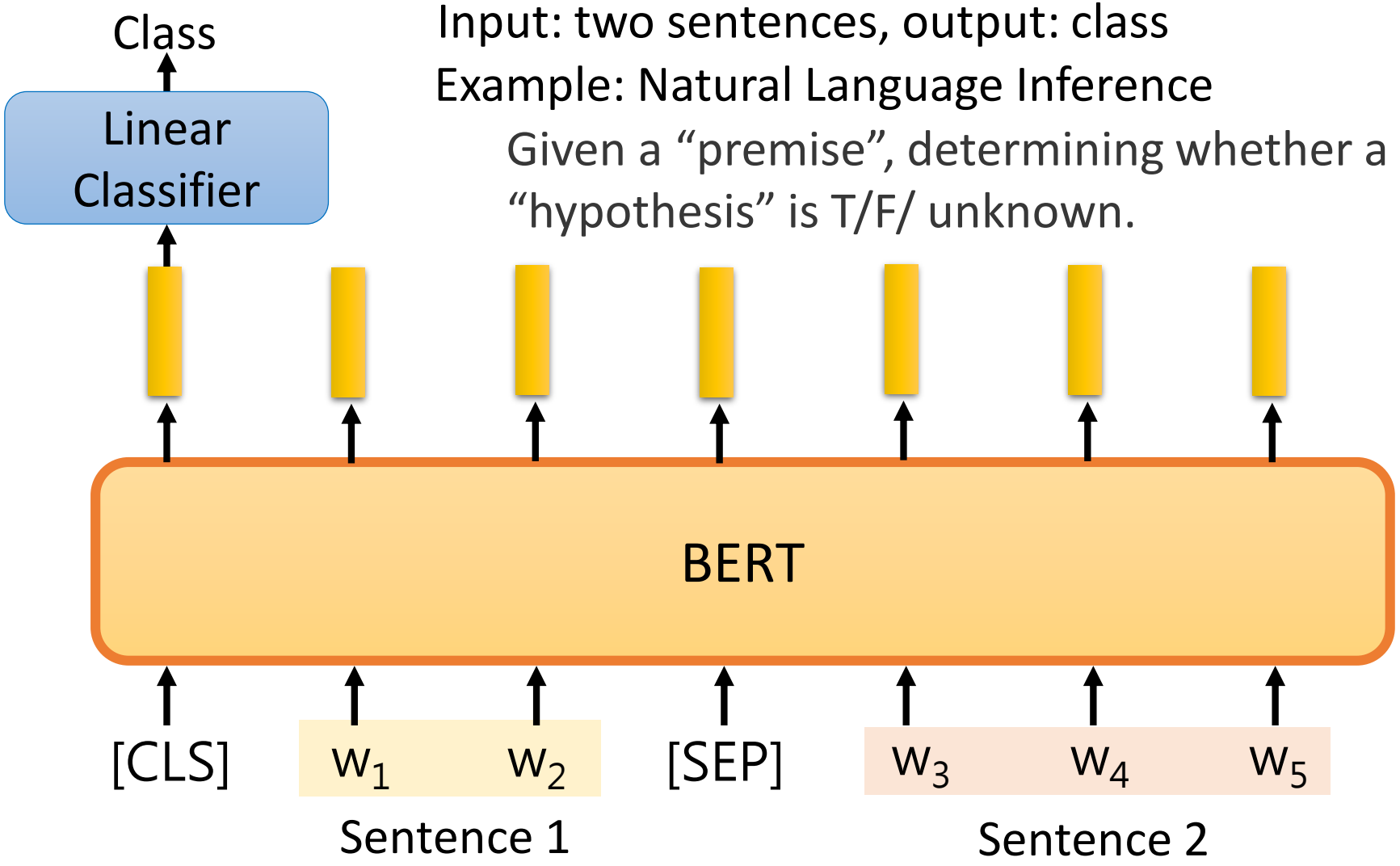


- Can do for a single sentence or a pair
- Tag each word piece
- Example tasks: span-based question answering, name-entity recognition, POS tagging





How to use BERT – Case 3



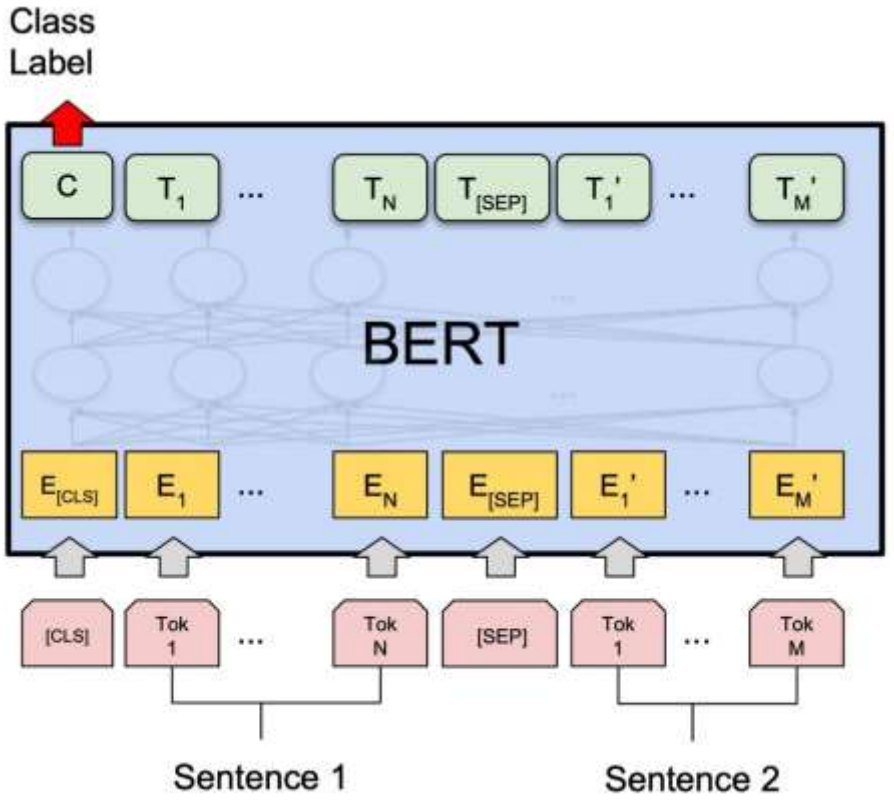
Input: two sentences, output: class

Example: Natural Language Inference

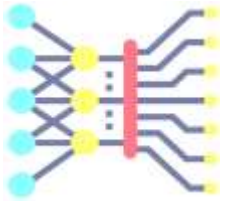
Given a “premise”, determining whether a “hypothesis” is T/F/ unknown.

Sentence-pair Classification with BERT

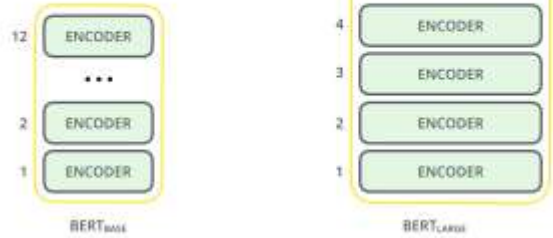
- Feed both sentences, and CLS token used for classification
- Example tasks:
  - Textual entailment
  - Question paraphrase detection
  - Question-answering pair classification
  - Semantic textual similarity
  - Multiple choice question answering



Model Architecture

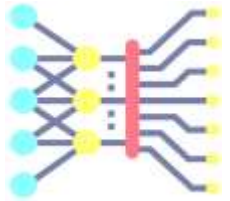


- BERT BASE
  - 12 layers, 768-dim per word-piece token
  - 12 heads.
  - Total parameters = 110M
- BERT LARGE
  - 24 layers, 1024-dim per word-piece token
  - 16 heads.
  - Total parameters = 340M



BERT is basically a trained Transformer Encoder stack.

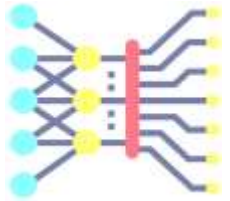
Model Details



- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

Slide from Jacob Delvin

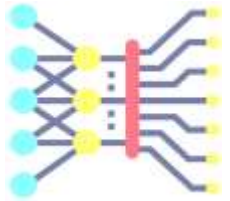
Evaluation of BERT



General Language Understanding Evaluation (**GLUE**) benchmark: Standard split of data to train, validation, test, where labels for the test set is only held in the server.

- Sentence pair tasks
  - **MNLI**, Multi-Genre Natural Language Inference
  - **QQP**, Quora Question Pairs
  - **QNLI,** Question Natural Language Inference
  - **STS-B** The Semantic Textual Similarity Benchmark
  - **MRPC** Microsoft Research Paraphrase Corpus
  - **RTE** Recognizing Textual Entailment
  - **WNLI** Winograd NLI is a small natural language inference dataset
- Single sentence classification
  - **SST-2** The Stanford Sentiment Treebank
  - **CoLA** The Corpus of Linguistic Acceptability

GLUE Results



System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$BERT_{BASE}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

MultiNLI

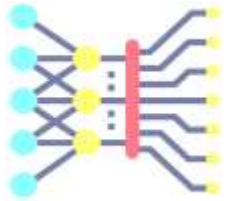
Premise: Hills and mountains are especially sanctified in Jainism.
Hypothesis: Jainism hates nature.
Label: Contradiction

CoLa

Sentence: The wagon rumbled down the road. Label: Acceptable
Sentence: The car honked down the road.
Label: Unacceptable

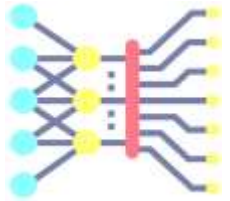
Slide from Jacob Delvin

Results



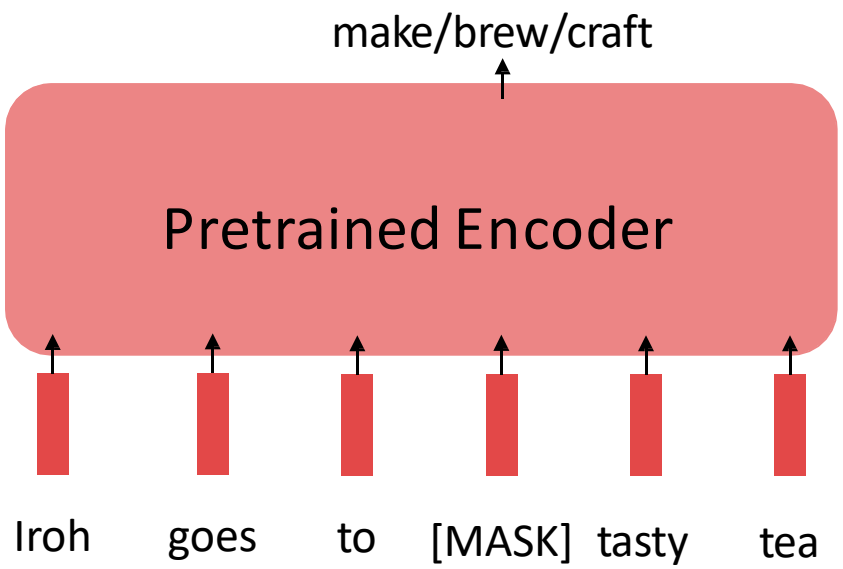
- Fine-tuned BERT outperformed previous state of the art on 11 NLP tasks
- Since then was applied to many more tasks with similar results
- The larger models perform better, but even the small BERT performs better than prior methods
- Variants quickly outperformed human performance on several tasks, including span-based question answering — but what does this mean is less clear
- Started an arms race (between industry labs) on bigger and bigger models

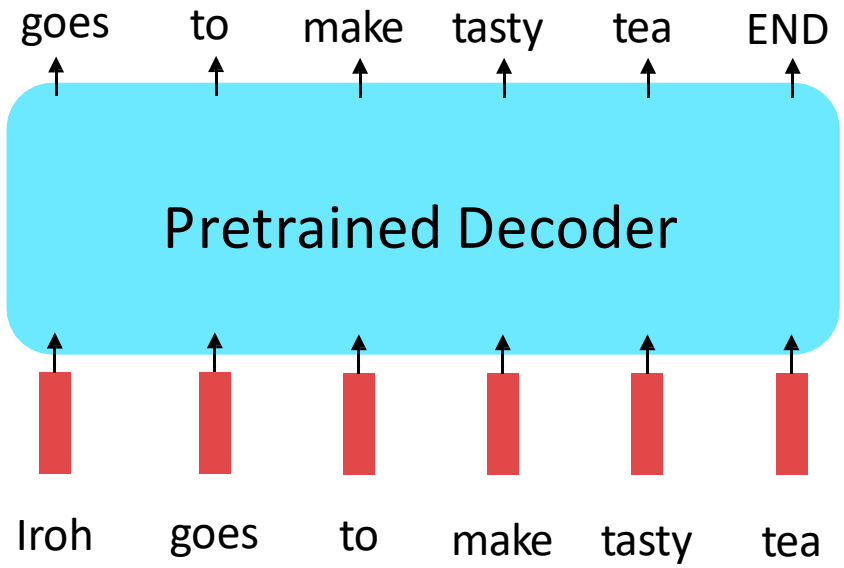
Limitations of pretrained encoders



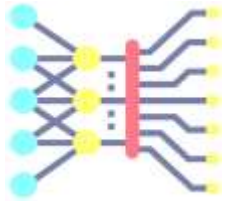
Those results looked great! Why not used pretrained encoders for everything?

If your task involves generating sequences, consider using a pretrained decoder; BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.





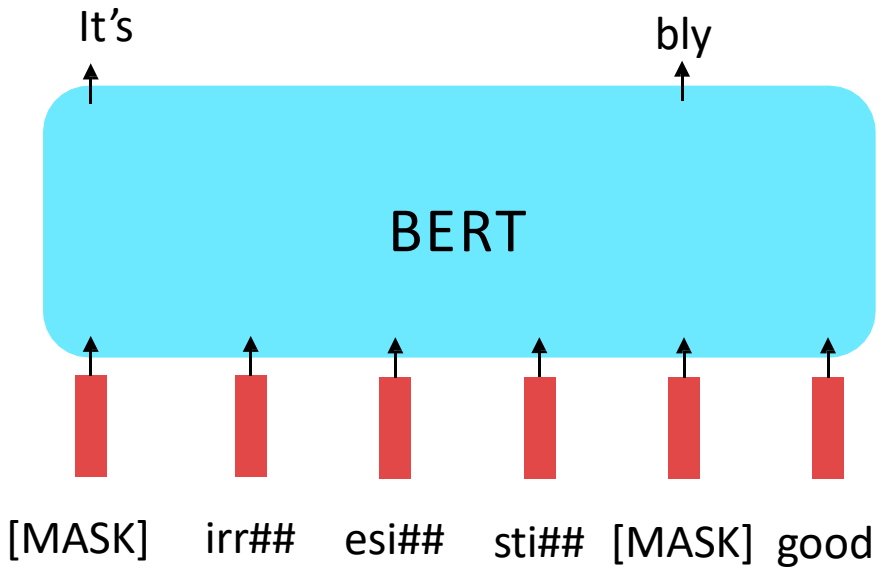
Extensions of BERT

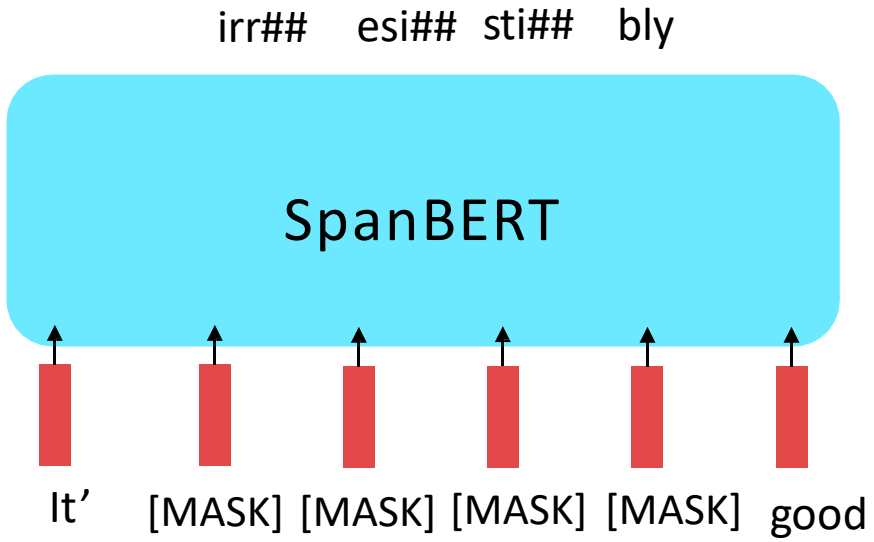


You'll see a lot of BERT variants like RoBERTa, SpanBERT, +++

Some generally accepted improvements to the BERT pretraining formula:

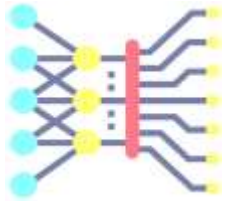
- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task





[Liu et al., 2019; Joshi et al., 2020]

Extensions of BERT

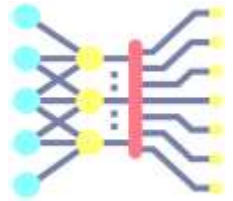


A takeaway from the RoBERTa paper: more compute, more data can improve pretraining even when not changing the underlying Transformer encoder.

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

[Liu et al., 2019; Joshi et al., 2020]

Full Finetuning vs. Parameter-Efficient Finetuning



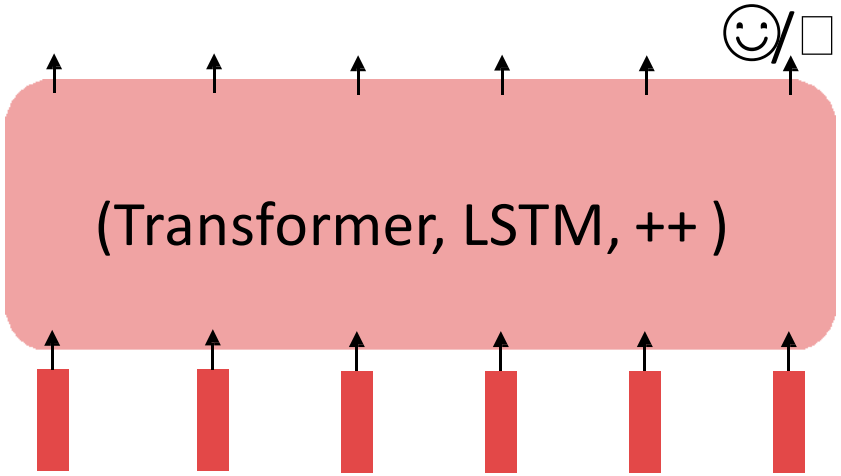
Finetuning every parameter in a pretrained model works well, but is memory-intensive.

But **lightweight** finetuning methods adapt pretrained models in a constrained way.

Leads to **less overfitting** and/or **more efficient finetuning and inference.**

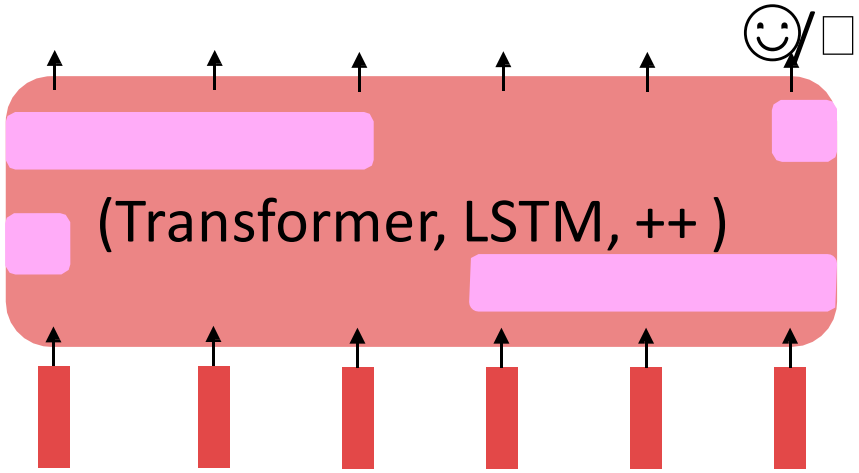
Full Finetuning

Adapt all parameters



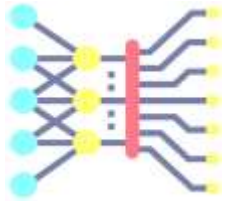
Lightweight Finetuning

Train a few existing or new parameters



[Liu et al., 2019; Joshi et al., 2020]

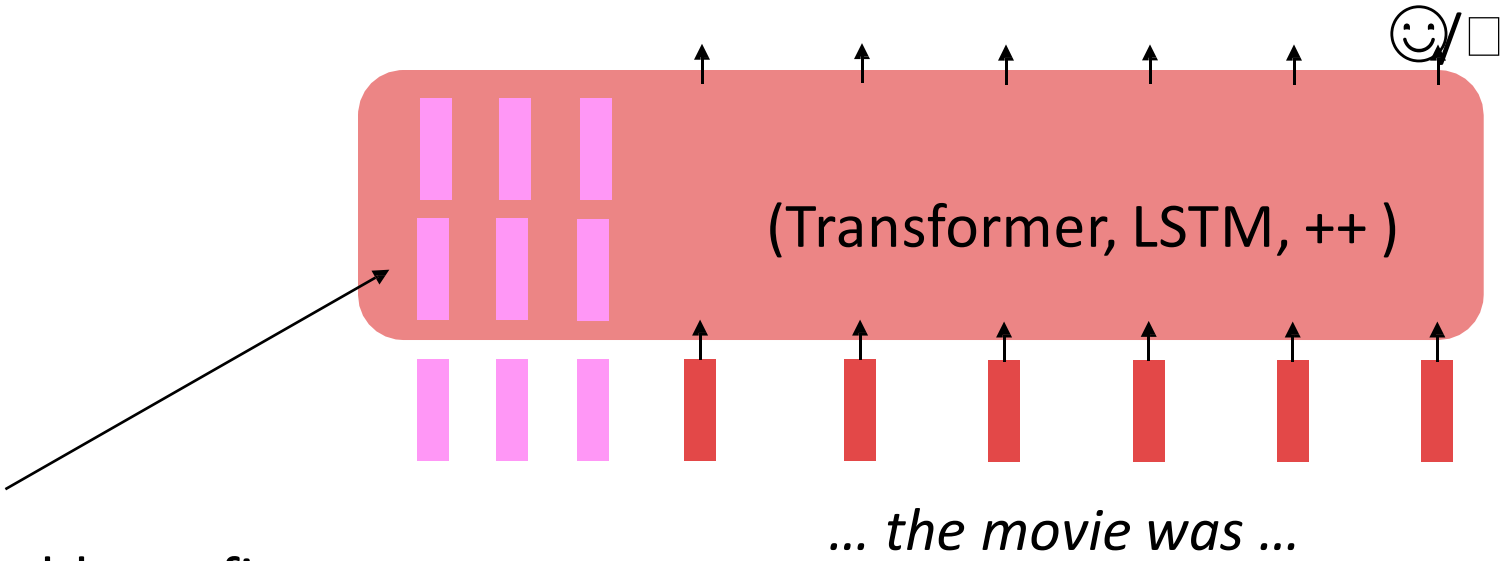
Parameter-Efficient Finetuning: Prefix-Tuning, Prompt tuning



Prefix-Tuning adds a **prefix** of parameters, and **freezes all pretrained parameters.**

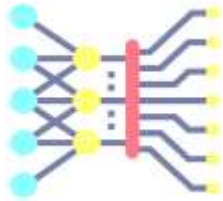
The prefix is processed by the model just like real words would be.

Advantage: each element of a batch at inference could run a different tuned model.



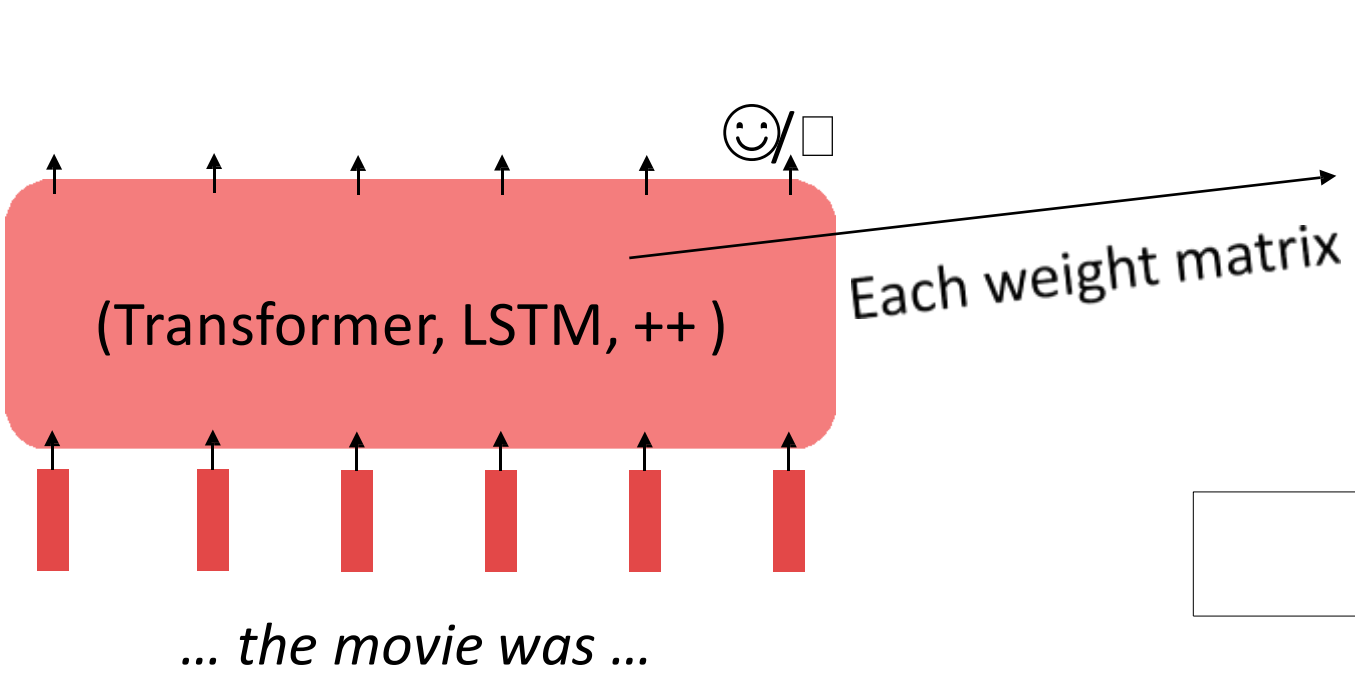
[Li and Liang, 2021; Lester et al., 2021]

Parameter-Efficient Finetuning: Low-Rank Adaptation

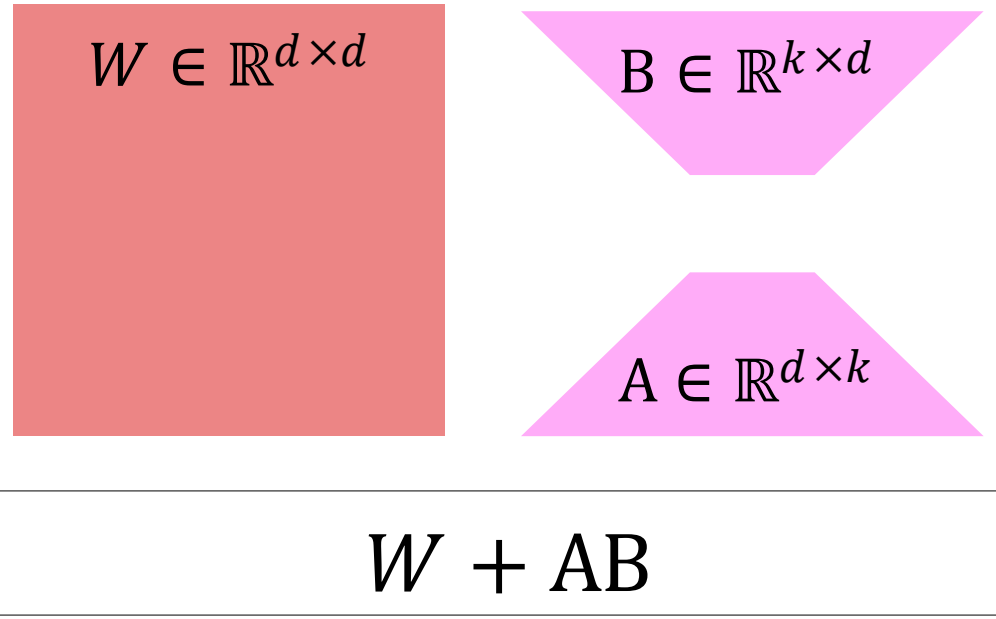


Low-Rank Adaptation Learns a low-rank "diff" between the pretrained and finetuned weight matrices.

Easier to learn than prefix-tuning.

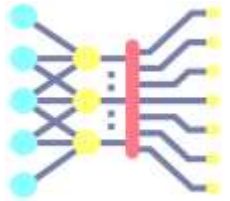


Each weight matrix



[Hu et al., 2021]

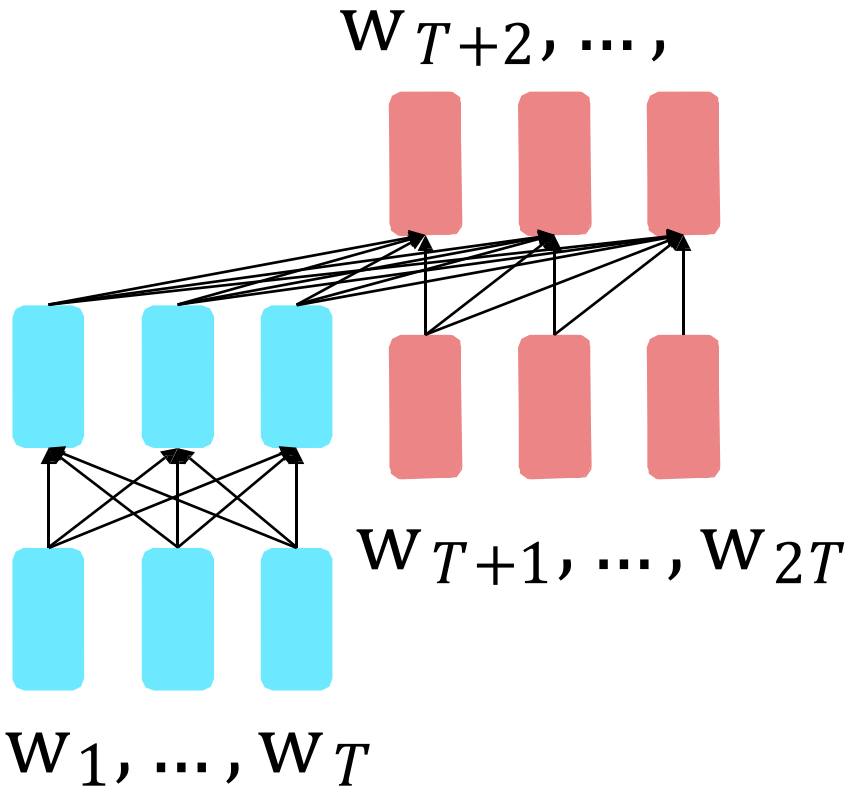
Pretraining encoder-decoders: what pretraining objective to use?



For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

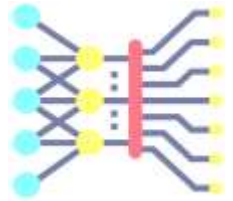
$$h_1, \ldots, h_T = \text{Encoder}\ (w_1, \ldots, w_T)$$
$$h_{T+1}, \ldots, h_2 = \text{Decoder}(w_1, \ldots, w_T, h_1, \ldots, h_T)$$

The **encoder** portion benefits from bidirectional context; the **decoder** portion is used to train the whole model through language modeling.



[Raffel et al., 2018]

Pretraining encoder-decoders: what pretraining objective to use?



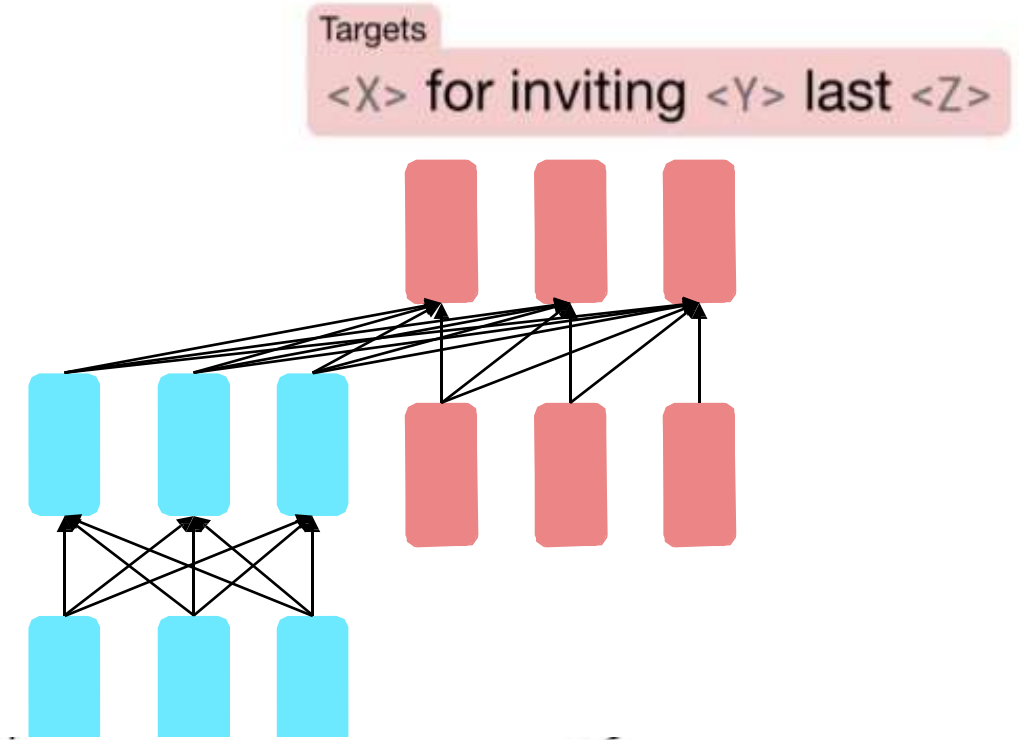
What Raffel et al., 2018 found to work best was **span corruption.** Their model: **T5**.

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

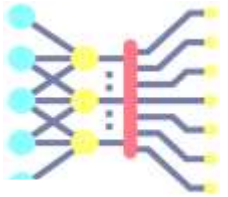
Original text

Thank you for inviting me to your party last week.

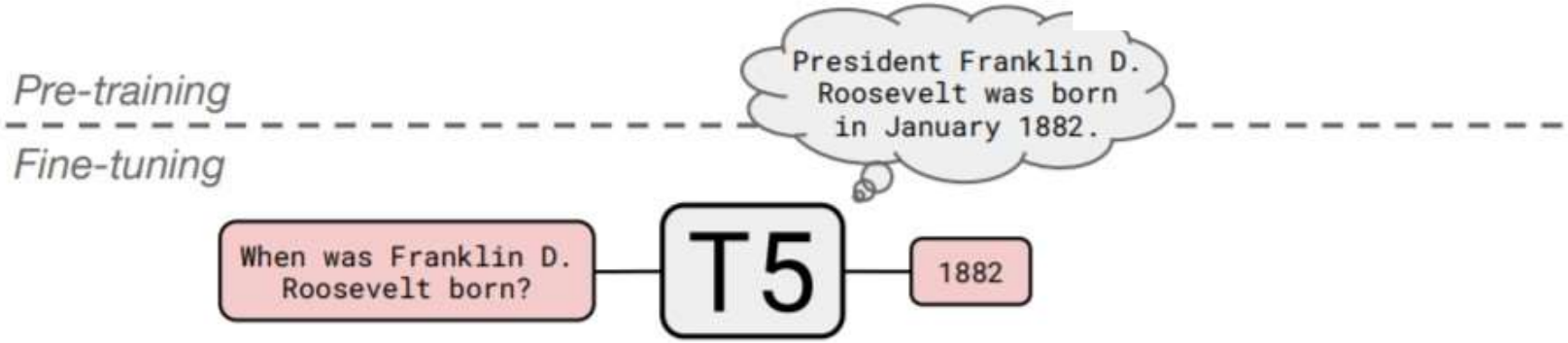
This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



Pretraining encoder-decoders: what pretraining objective to use?



A fascinating property of T5: it can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.



	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

NQ: Natural Questions

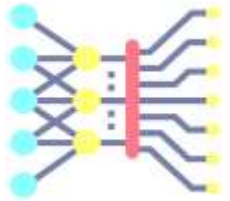
WQ: WebQuestions

TQA: Trivia QA

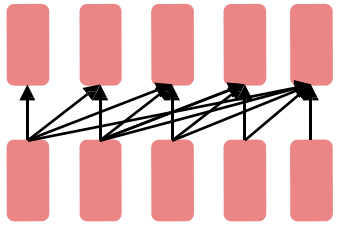
All “open-domain” versions

[Raffel et al., 2018]

Pretraining for three types of architectures



The neural architecture influences the type of pretraining, and natural use cases.



Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- All the biggest pretrained models are Decoders.

Pretraining decoders

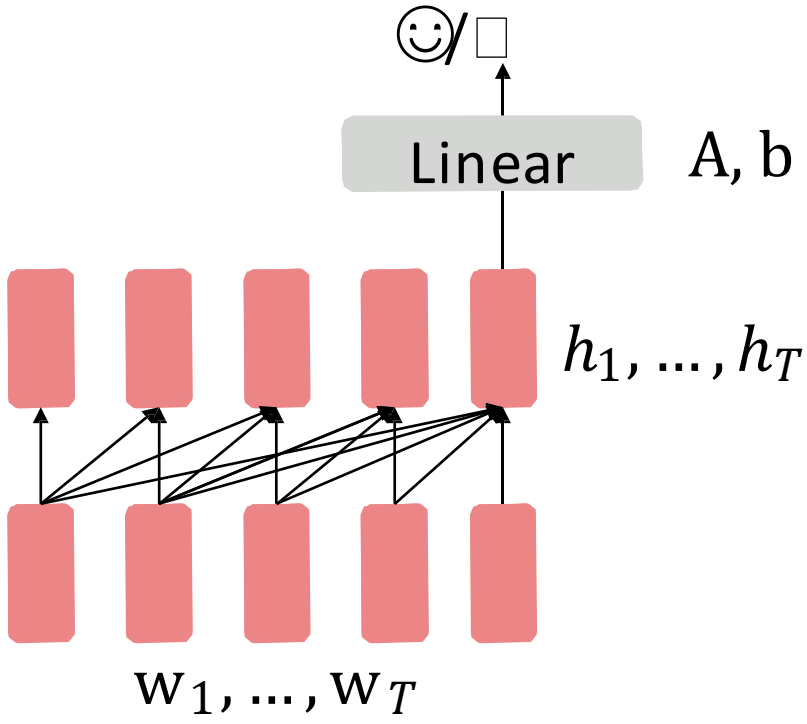
When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_{1:t-1})$.

We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \ldots, h_T = \text{Decoder}\ (\mathrm{w}_1, \ldots, \mathrm{w}_T)$$
$$y \sim A h_T + b$$

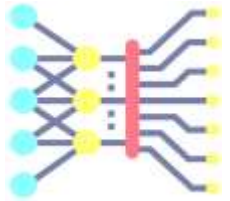
Where A and b are randomly initialized and specified by the downstream task.

Gradients backpropagate through the whole network.



the linear layer hasn't been pretrained and must be learned from scratch.]

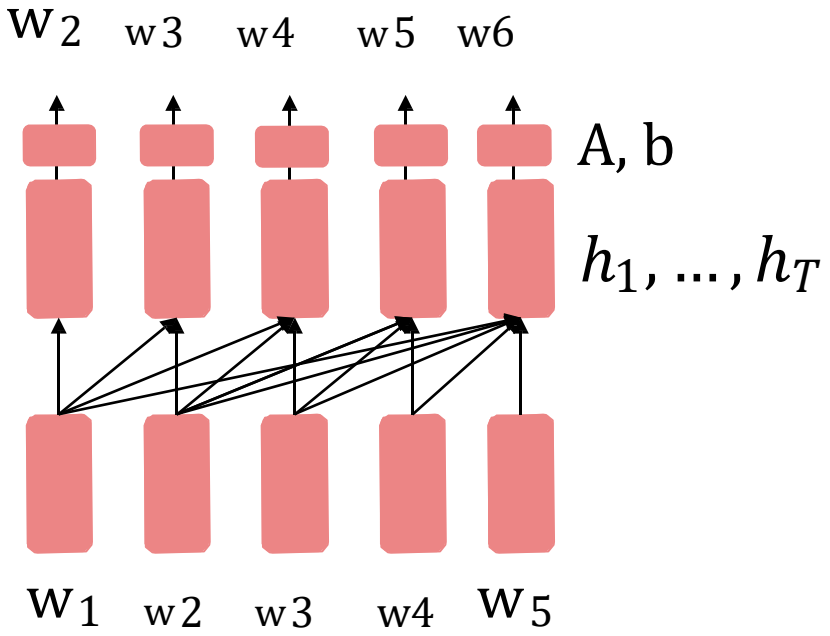
Pretraining decoders



It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$

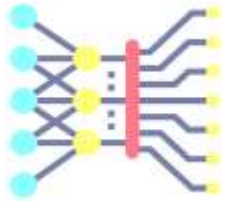
This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)



[Note how the linear layer has been pretrained.]

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

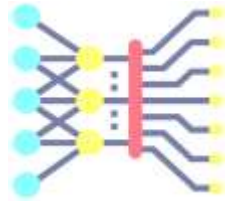


2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.

[Devlin et al., 2018]

Generative Pretrained Transformer (GPT) [Radford et al., 2018]



How do we format inputs to our decoder for **finetuning tasks?**

Natural Language Inference: Label pairs of sentences as *entailing/contradictory/neutral*

Premise: *The man is in the doorway*

Hypothesis: *The person is near the door*

} **entailment**

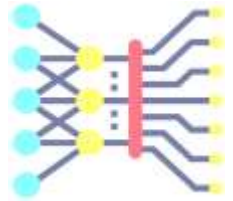
Radford et al., 2018 evaluate on natural language inference.

Here's roughly how the input was formatted, as a sequence of tokens for the decoder.

[START] *The man is in the doorway* [DELIM] *The person is near the door* [EXTRACT]

The linear classifier is applied to the representation of the [EXTRACT] token.

GPT-3, In-context learning, and very large models



So far, we've interacted with pretrained models in two ways:

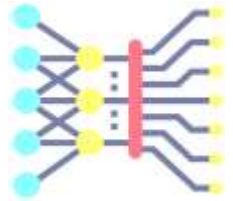
- Sample from the distributions they define (maybe providing a prompt)
- Fine-tune them on a task we care about, and take their predictions.

Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters.

GPT-3 has 175 billion parameters.

GPT-3, In-context learning, and very large models



Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.

Input (prefix within a single Transformer decoder context):

“ thanks -> merci

hello -> bonjour

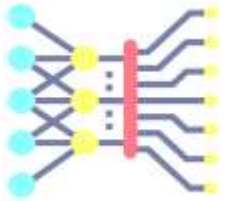
mint -> menthe

otter -> ”

Output (conditional generations):

loutre…”

The prefix as task specification and scratch pad: chain-of-thought



Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

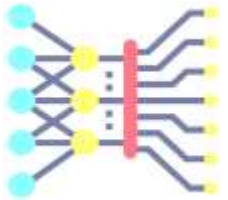
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

[Wei et al., 2023]

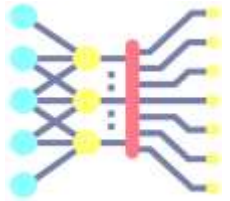
What kinds of things does pretraining teach?



There's increasing evidence that pretrained models learn a wide variety of things about the statistical properties of language. Taking our examples from the start of class:

- *Stanford University is located in _____, California.* [Trivia]
- *I put ____fork down on the table.* [syntax]
- *The woman walked across the street, checking for traffic over ____shoulder.* [coreference]
- *I went to the ocean to see the fish, turtles, seals, and _____.* [lexical semantics/topic]
- *Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___.* [sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____. [some reasoning – this is harder]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, _____[some basic arithmetic; they don't learn the Fibonnaci sequence]
- Models also learn – and can exacerbate racism, sexism, all manner of bad biases.
- More on all this in the interpretability lecture!

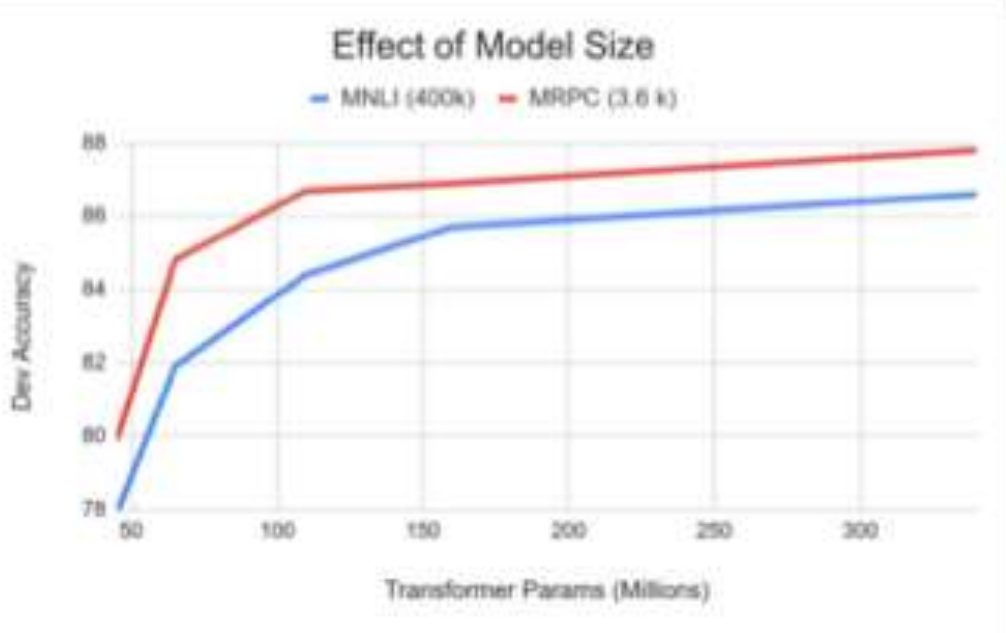
Hard to do with BERT



- BERT cannot generate text (at least not in an obvious way)

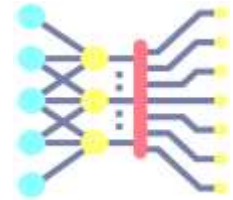
- Masked language models are intended to be used primarily for “analysis” tasks

Effects of Model Size

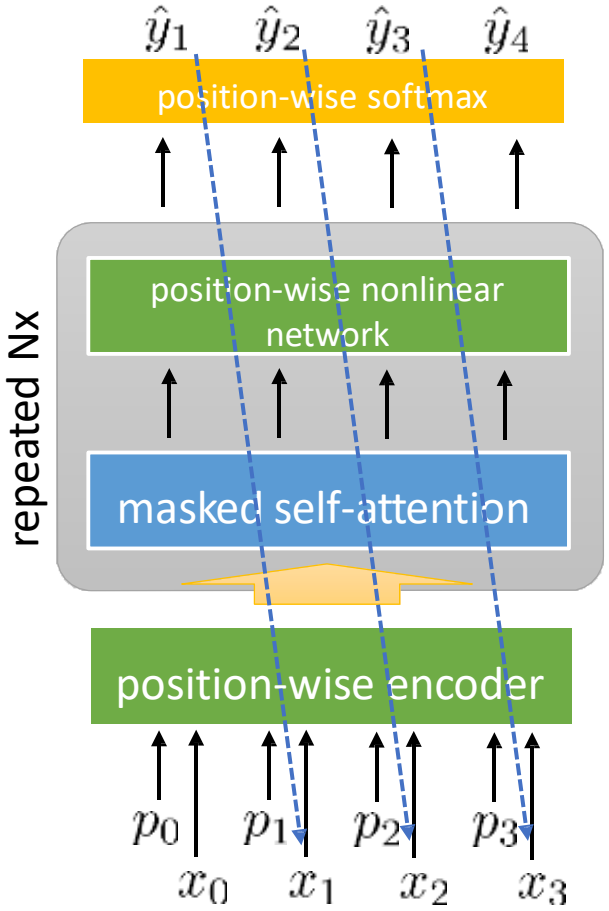


- Big models help *a lot*
- Going from 110M -> 340M params helps even on datasets with 3,600 labelled examples
- Improvements have *not* asymptoted

Slide from Jacob Delvin

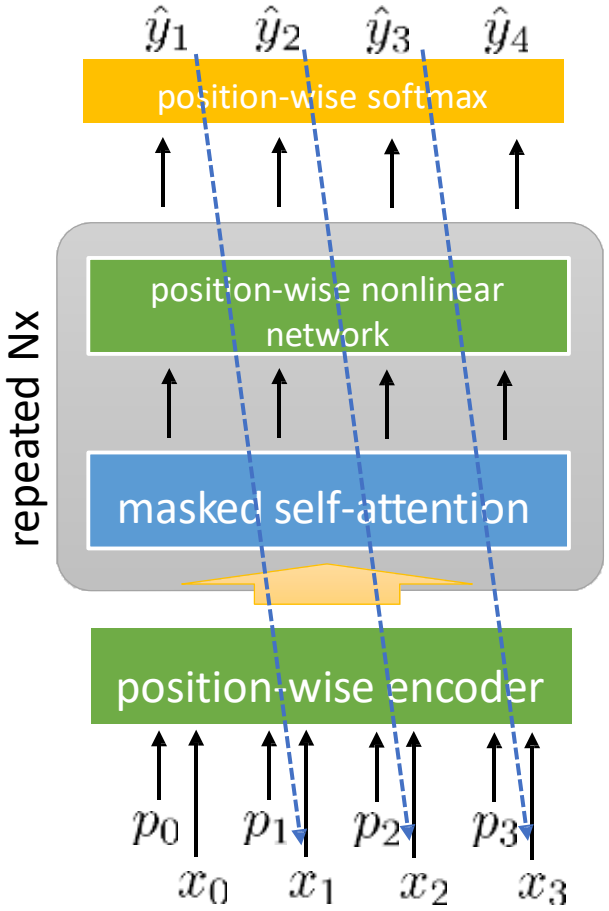


GPT et al.



- One-directional (forward) transformer models do have one **big** advantage over BERT.
- Generation is not really possible with BERT, but a forward (masked attention) model can do it!
- GPT (GPT-2, GPT-3, etc.) is a classic example of this

GPT et al.



OpenAI GPT-2 generated text

source

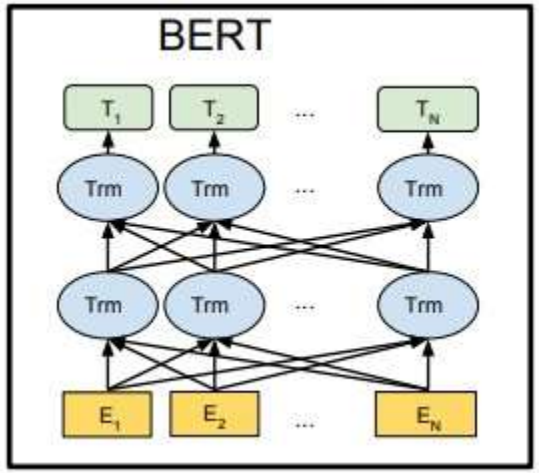
Input: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

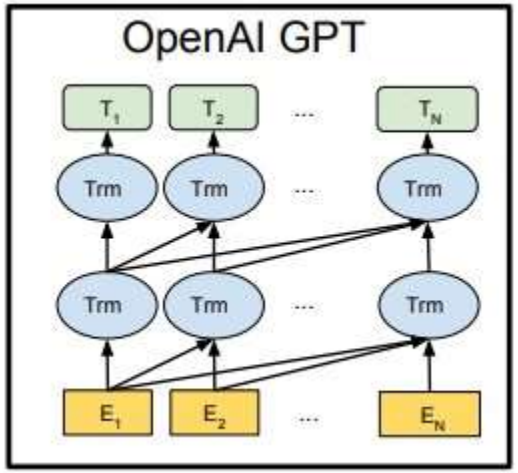
Output: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

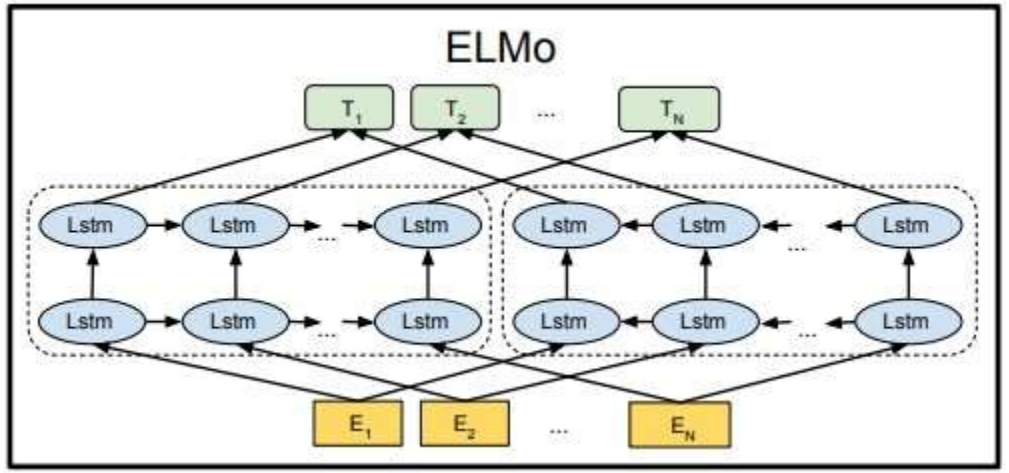
Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

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.

Pretrained language models summary







bidirectional transformer

+ great representations

- can't generate text

one-directional transformer

+ can generate text

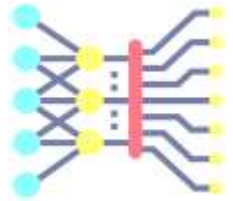
- OK representations

bidirectional LSTM

- OK representations

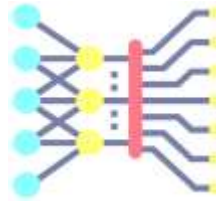
(largely supplanted by BERT)

Pretrained language models summary



- Language models can be trained on **very large and unlabeled** datasets of text.
- Internal **learned representations** depend on context: the meaning of a word is informed by the **whole sentence!**
- Can even get us representations of entire sentences (e.g., the first output token for BERT)
- Can be used to either **extract representations** to replace standard word embeddings…
- …or directly finetuned on downstream tasks (which means we modify all the weights in the whole language model, rather than just using pretrained model hidden states)

Improved variants of BERT



RoBERTa

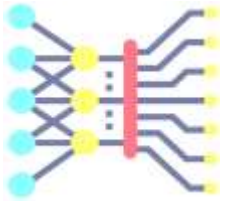
- Bigger batch size is better!
- Next sentence prediction doesn't help
- Pretrain on more data for as long as possible!

XLNet

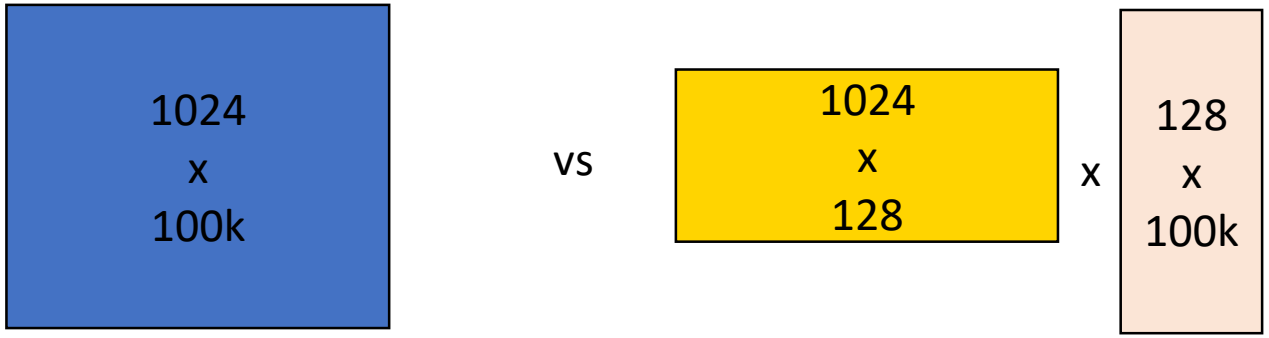
Key Ideas

- Autoregressive: use context to predict the next word
- Bidirectional context from permutation language modeling
- Self-attention mechanisms, uses Transformer-XL backbone

ALBERT

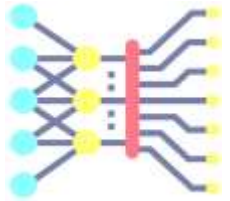


- *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations* (Lan et al, Google and TTI Chicago, 2019)
- Innovation #1: Factorized embedding parameterization
  - Use small embedding size (e.g., 128) and then project it to Transformer hidden size (e.g., 1024) with parameter matrix
- Innovation #2: Cross-layer parameter sharing
  - Share all parameters between Transformer layers



Slide from J. Delvin

GPT-2, BERT



- Transformers, GPT-2, and BERT
  1. A transformer uses Encoder stack to model input, and uses Decoder stack to model output (using input information from encoder side).
  2. But if we do not have input, we just want to model the “next word”, we can get rid of the Encoder side of a transformer and output “next word” one by one. This gives us GPT.
  3. If we are only interested in training a language model for the input for some other tasks, then we do not need the Decoder of the transformer, that gives us BERT.