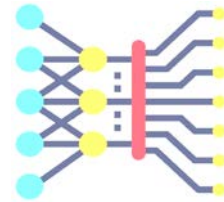


# CS60010: Deep Learning

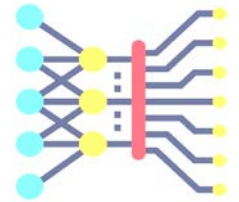
Spring 2023

Sudeshna Sarkar

## Transformer- Part 3

**Sudeshna Sarkar**

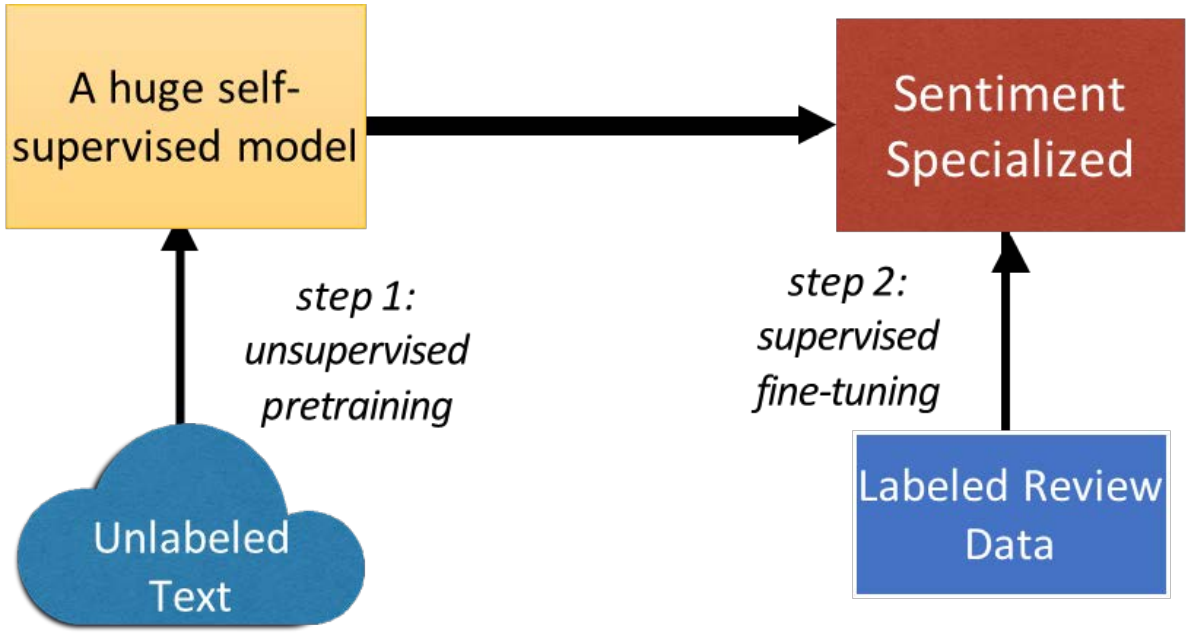
16 Mar 2023



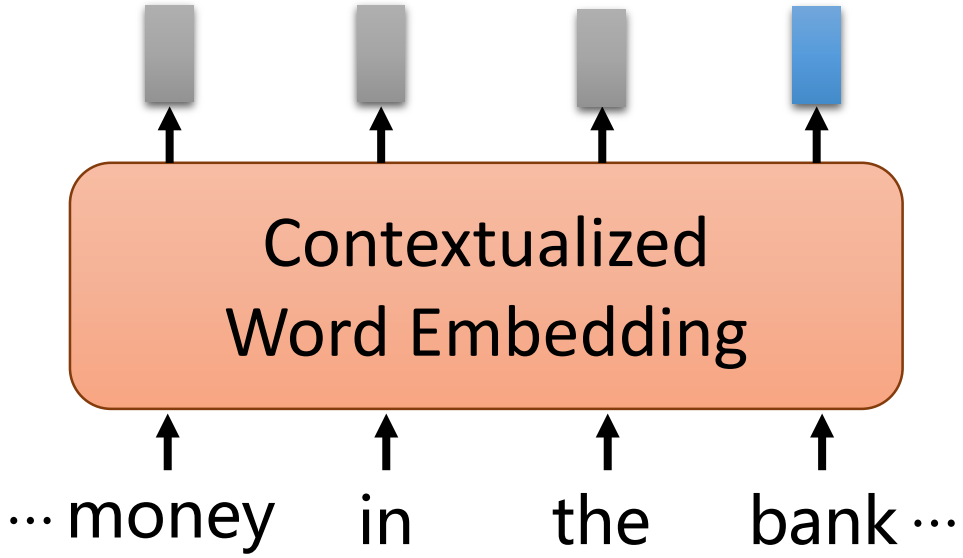
# Pre-training

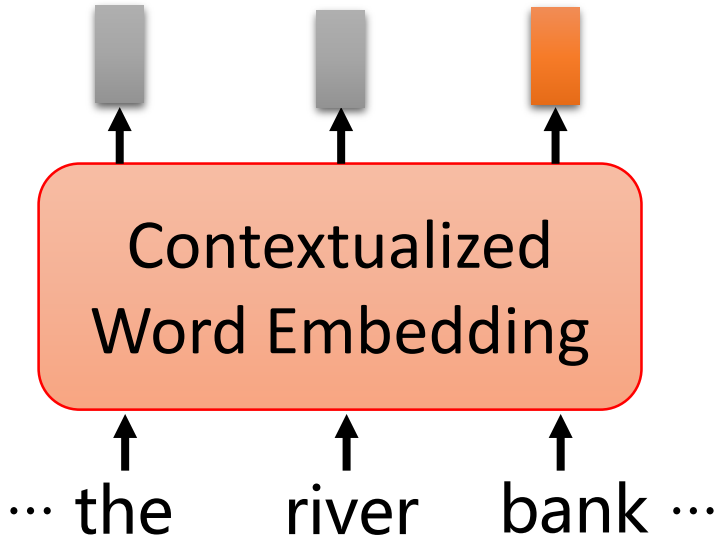
# Transfer Learning

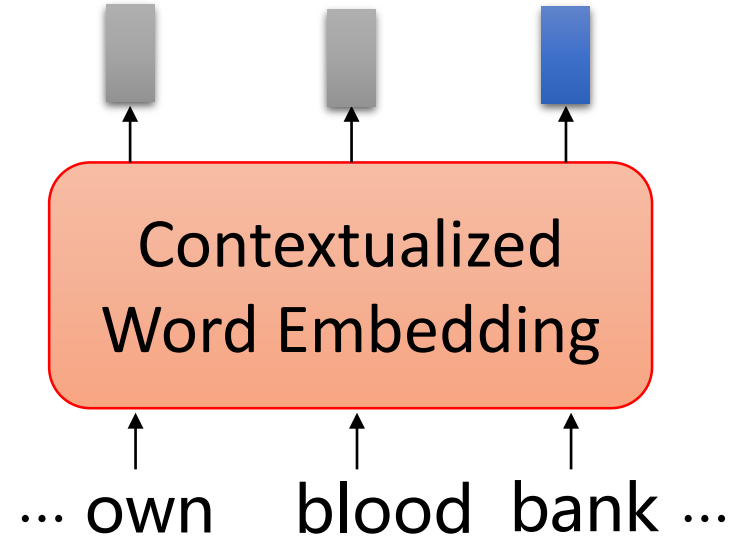
- Leverage unlabeled data to cut down on the number of labeled examples needed.
- Take a network trained on a task for which it is easy to generate labels, and adapt it to a different task for which it is harder.
- Train a really big language model on billions of words, transfer to every NLP task!



# Contextualized Word Embedding

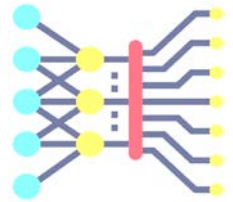



Train contextual representations on text corpus

# Uni-directional language models

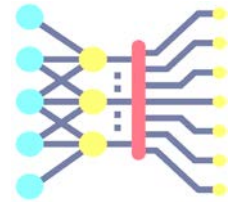


- Why are LMs unidirectional?
  - Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - Reason 2: Words can “see themselves” in a bidirectional encoder.

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

Slide from Jacob Delvin

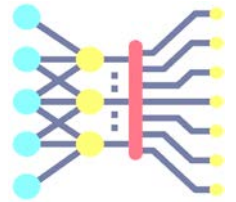
# Word structure and subword models



- 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



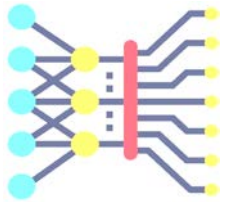
1. Start with a vocabulary containing only characters and an “end-of-word” symbol.
2. Using a corpus of text, find the most common adjacent characters “a,b”; add “ab” as a subword.
3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

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

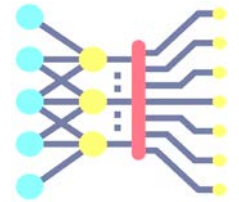
# Pretraining



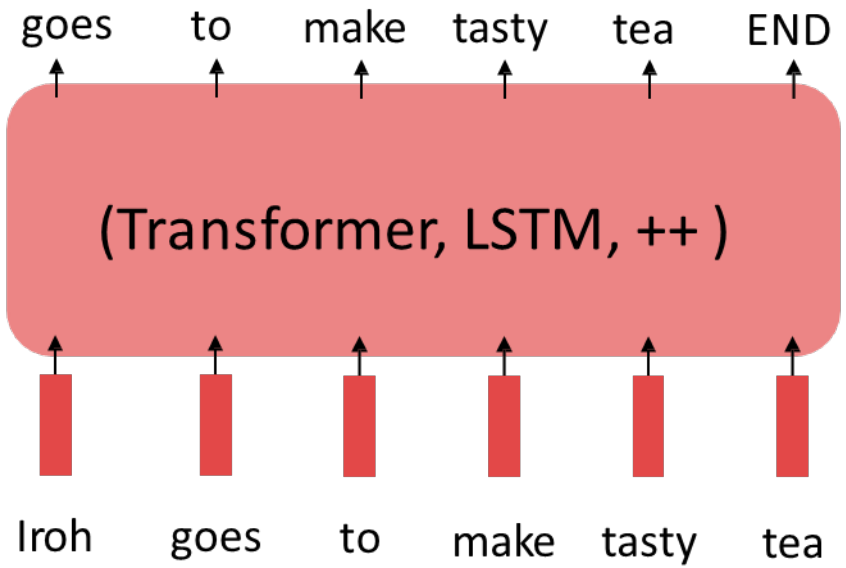
1. pretrained word embeddings
2. Pretraining whole models

- 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

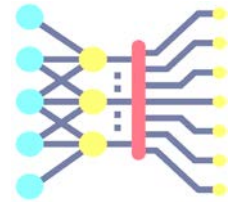


- Recall the language modeling task:
  - Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.




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

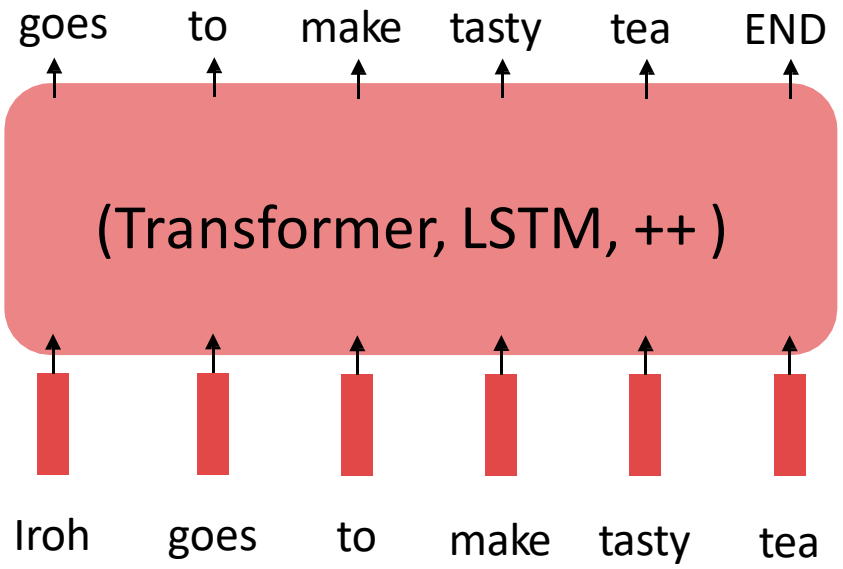
# The Pretraining / Finetuning Paradigm



Pretraining can improve NLP applications by serving as parameter initialization.

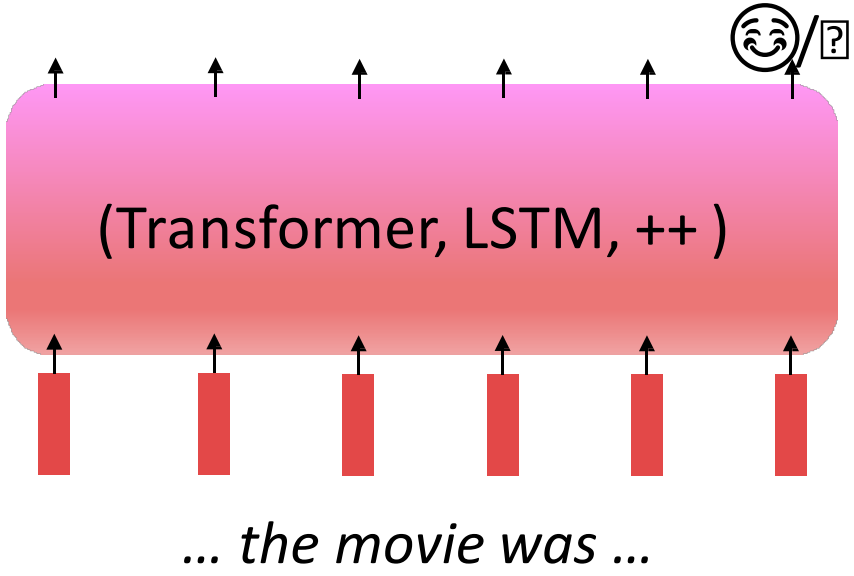
**Step 1: Pretrain (on language modeling)**

Lots of text; learn general things!

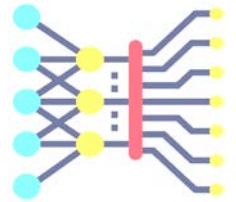



**Step 2: Finetune (on your task)**

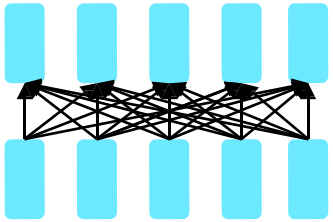
Not many labels; adapt to the task!



# Pretraining for three types of architectures

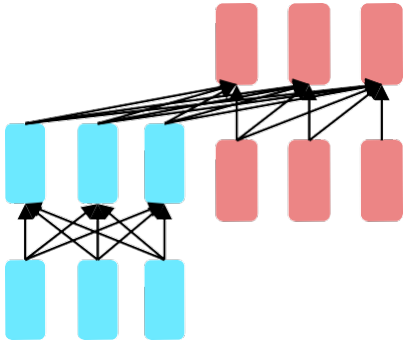


The neural architecture influences the type of pretraining, and natural use cases.



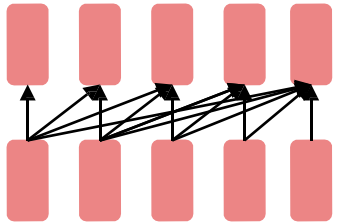
**Encoders**

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



**Encoder-Decoders**

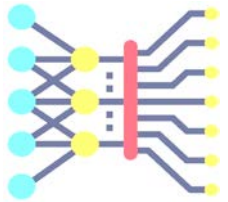
- Good parts of decoders and encoders?
- What's the best way to pretrain them?



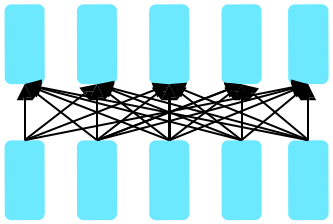
**Decoders**

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

# Pretraining for three types of architectures



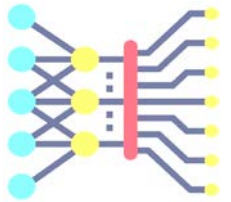
The neural architecture influences the type of pretraining, and natural use cases.



**Encoders**

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa

# Pretraining encoders: what pretraining objective to use?

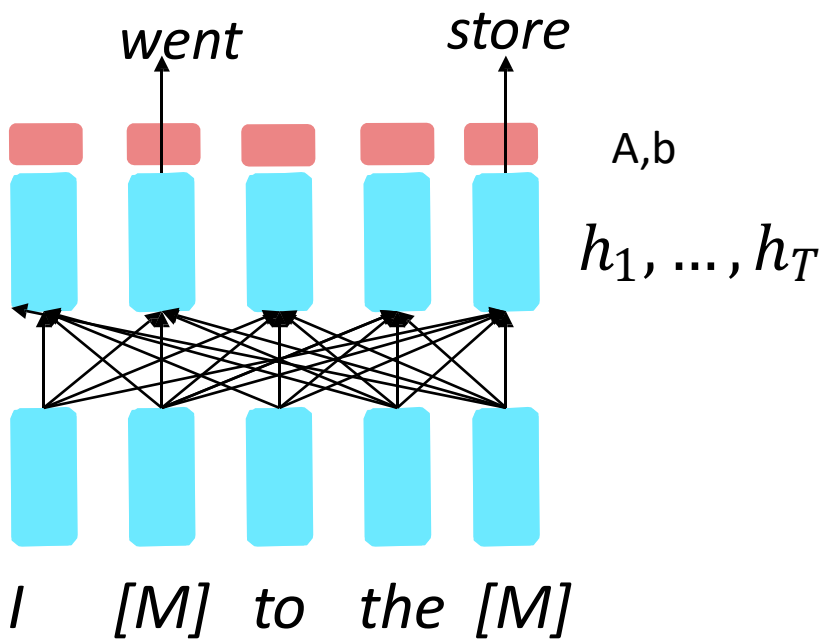


So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

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

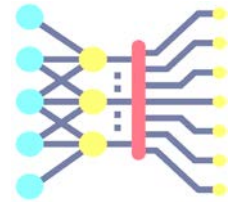
$$h_1, \ldots, h_T = \text{Encoder}(w_1, w_2, \ldots, w_T)$$

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]

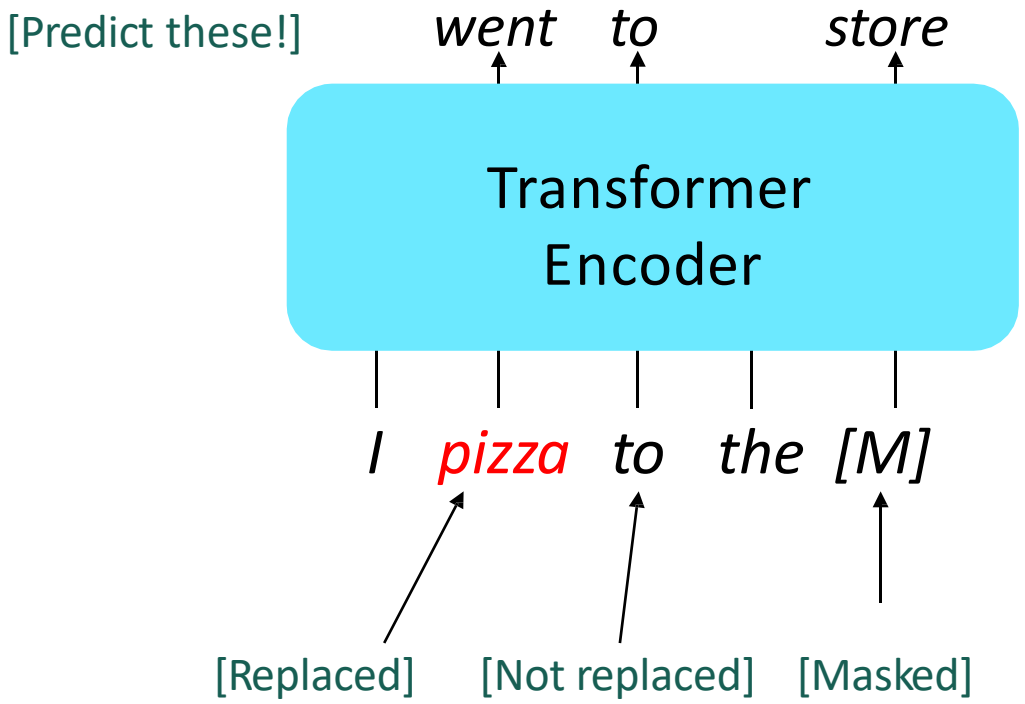
# BERT: Bidirectional Encoder Representations from Transformers



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

Details about Masked LM for BERT:

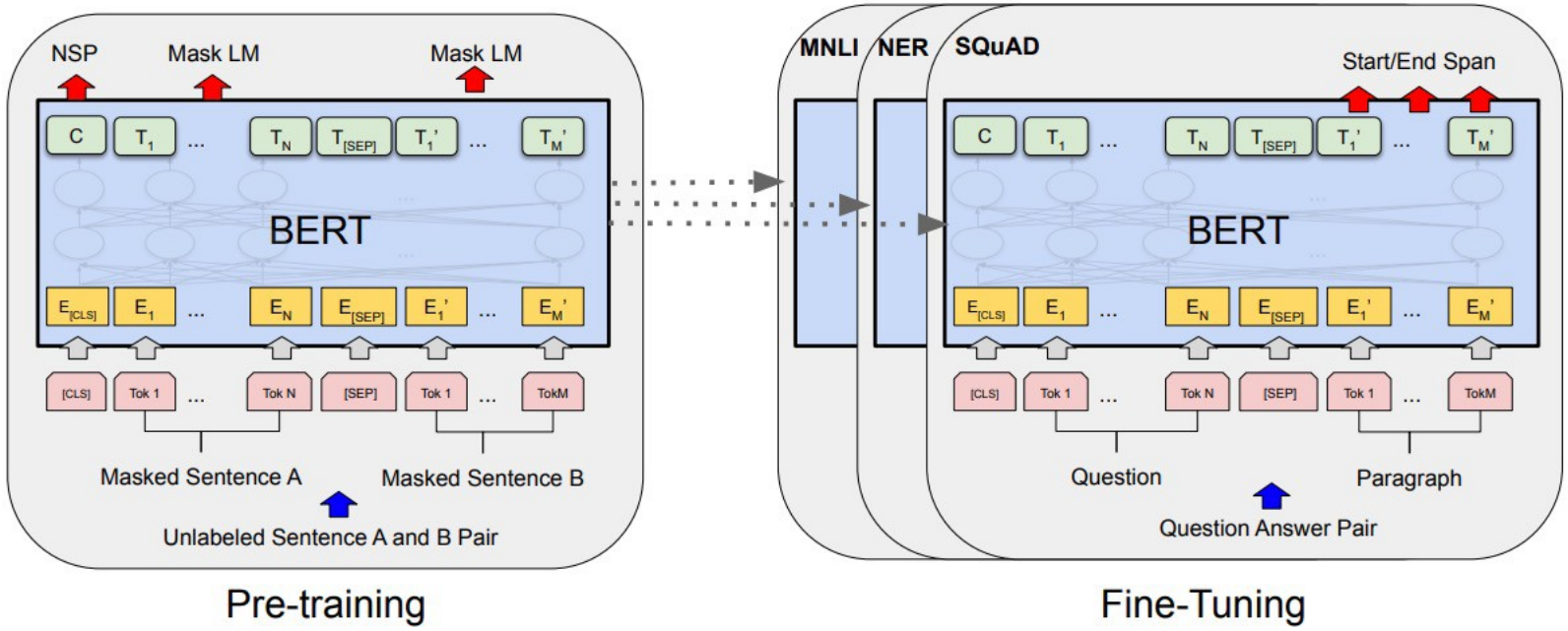
- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations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




[Devlin et al., 2018]

# BERT: Bidirectional Encoder Representations from Transformers

**Unified Architecture:** As shown below, there are minimal differences between the pre-training architecture and the fine-tuned version for each downstream task.



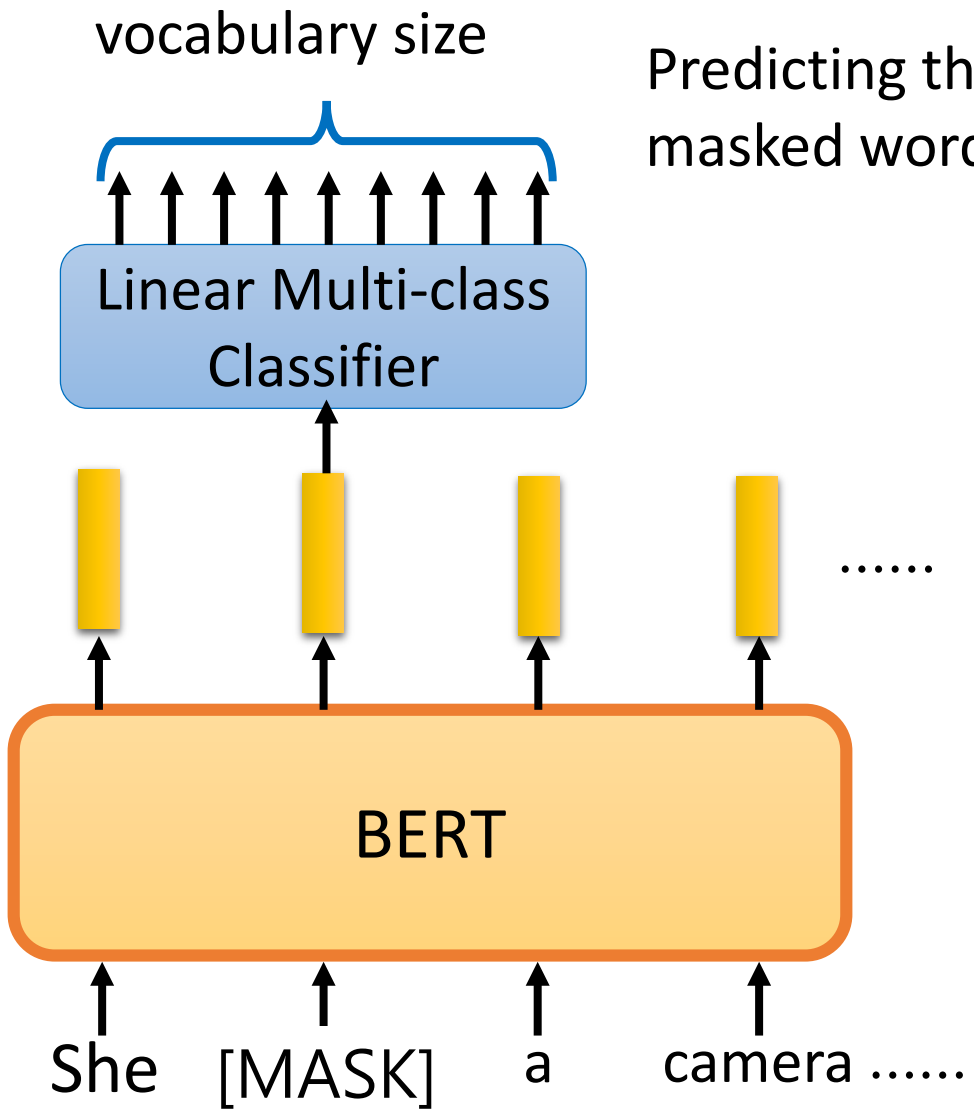

[Devlin et al., 2018]

# Training of BERT

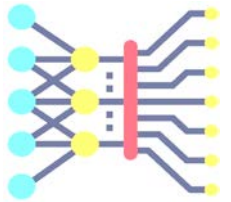
## Approach 1:

### Masked LM

Randomly replace 15% of the tokens with [MASK]



# Training of BERT

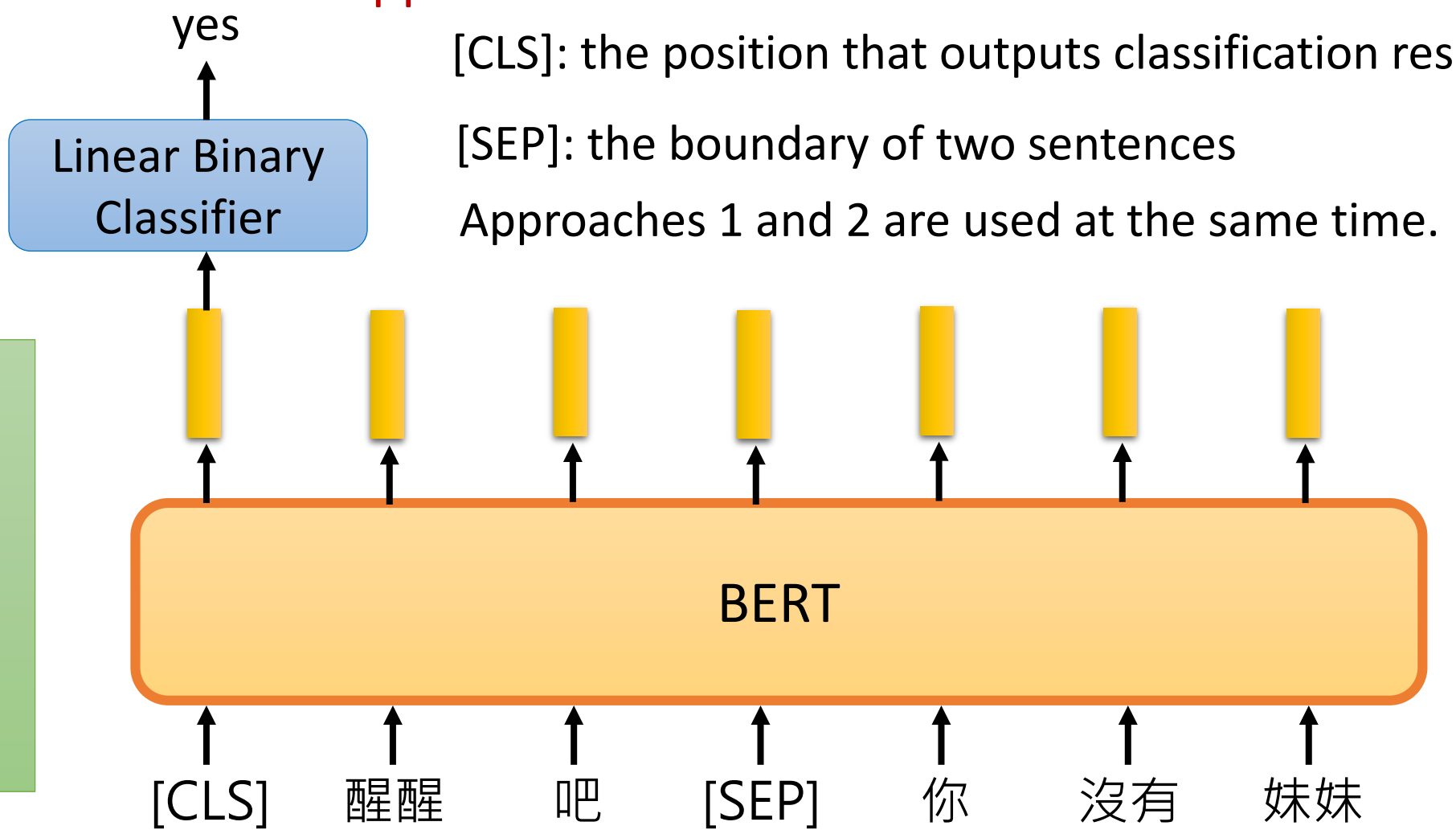


## Approach 2: Next Sentence Prediction

[CLS]: the position that outputs classification results

[SEP]: the boundary of two sentences

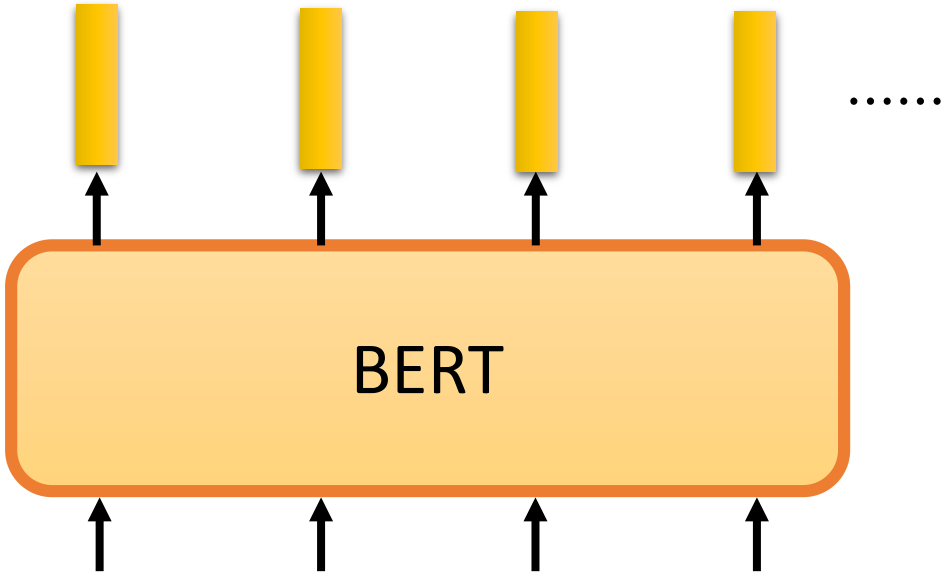
Approaches 1 and 2 are used at the same time.

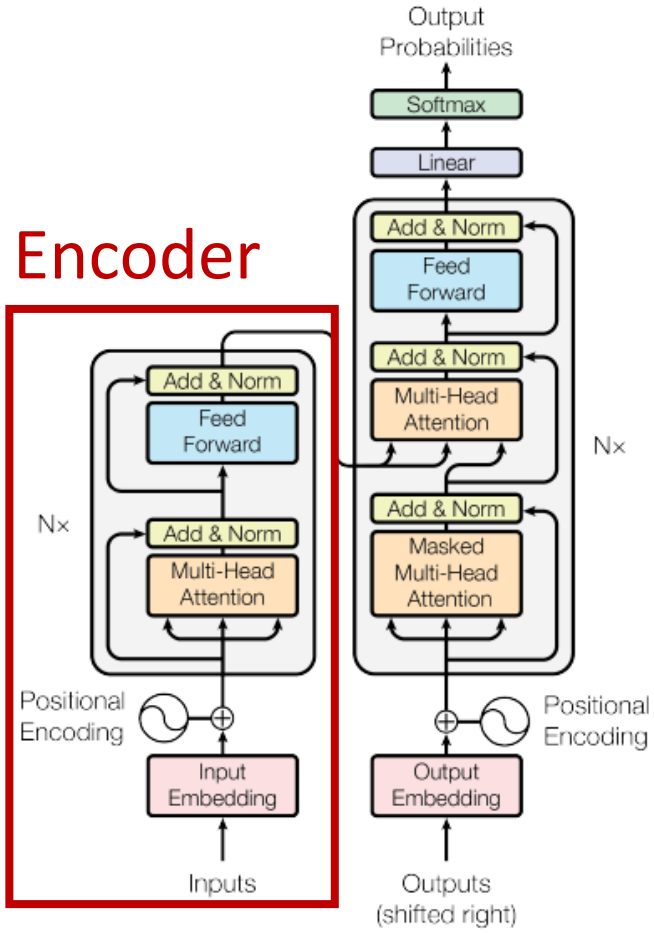



Randomly swap the order of the sentences 50% of the time

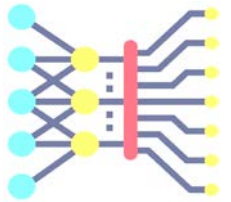
# Bidirectional Encoder Representations from Transformers (BERT)

Learned from a large amount of text without annotation

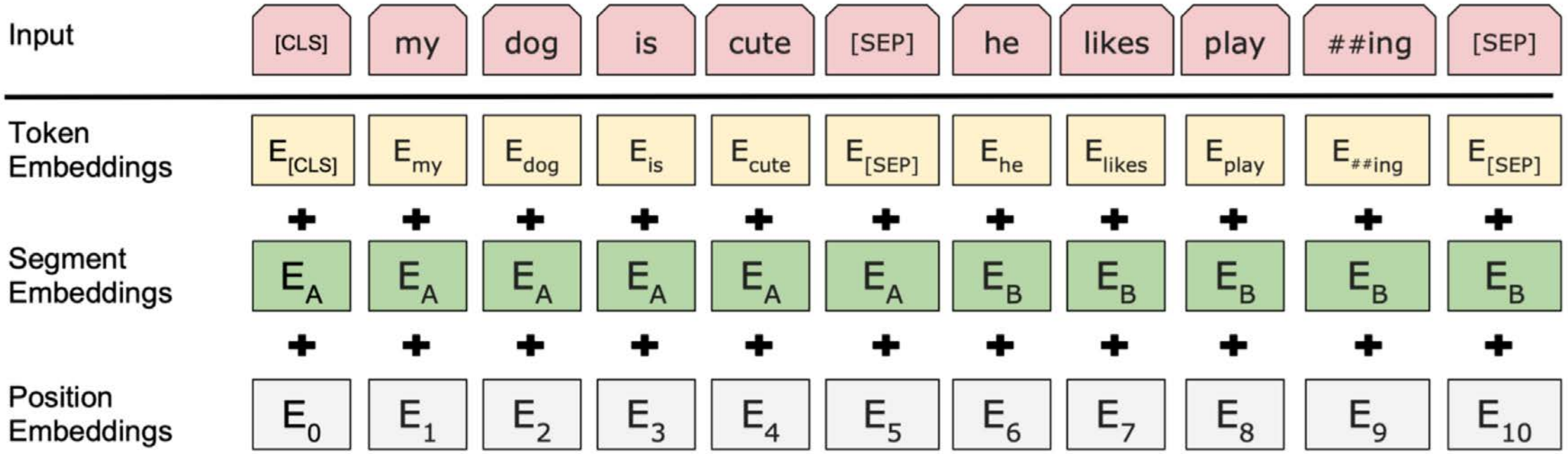


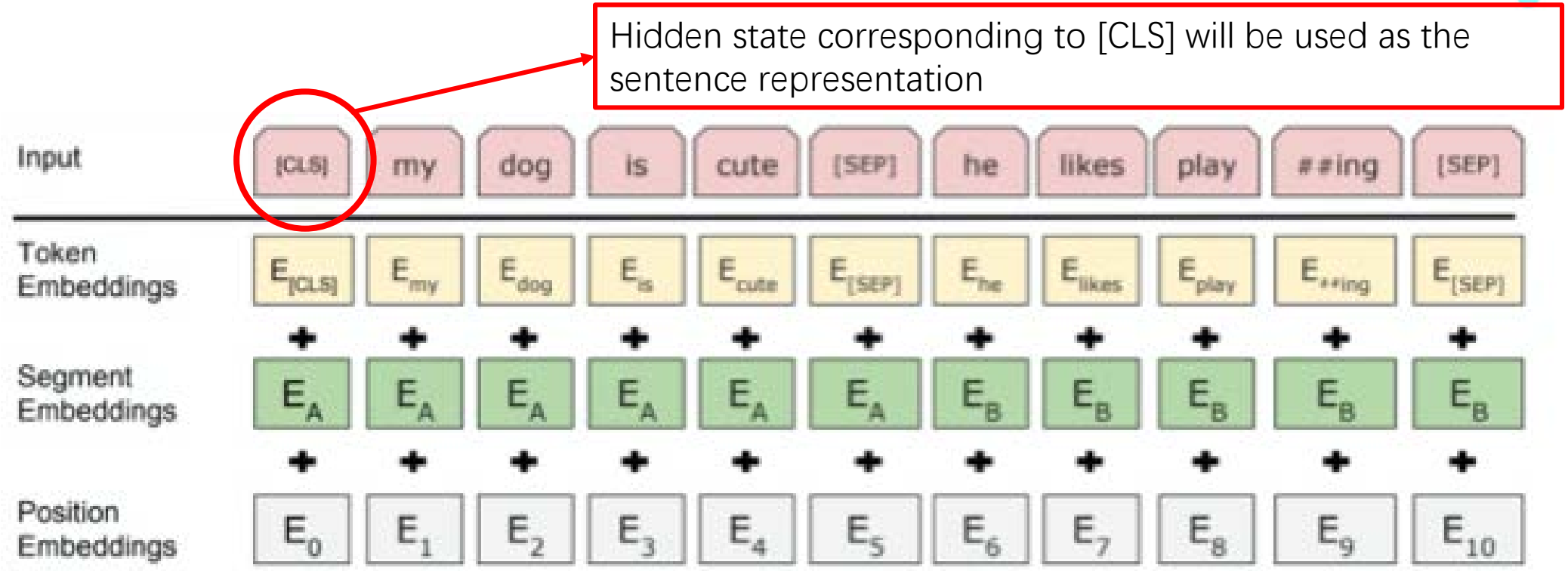
# BERT



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



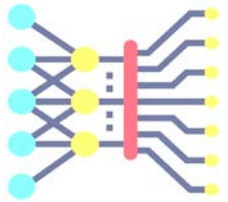
# Input Representation




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

Slide from Jacob Delvin

# BERT



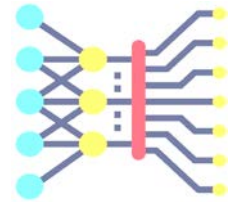
Details about BERT

- 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)
- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)
- Finetuning is practical and common on a single GPU
  - “Pretrain once, finetune many times.”



[Devlin et al., 2018]

# BERT



BERT was massively popular and hugely versatile; finetuning BERT led to new state-of-the-art results on a broad range of tasks.

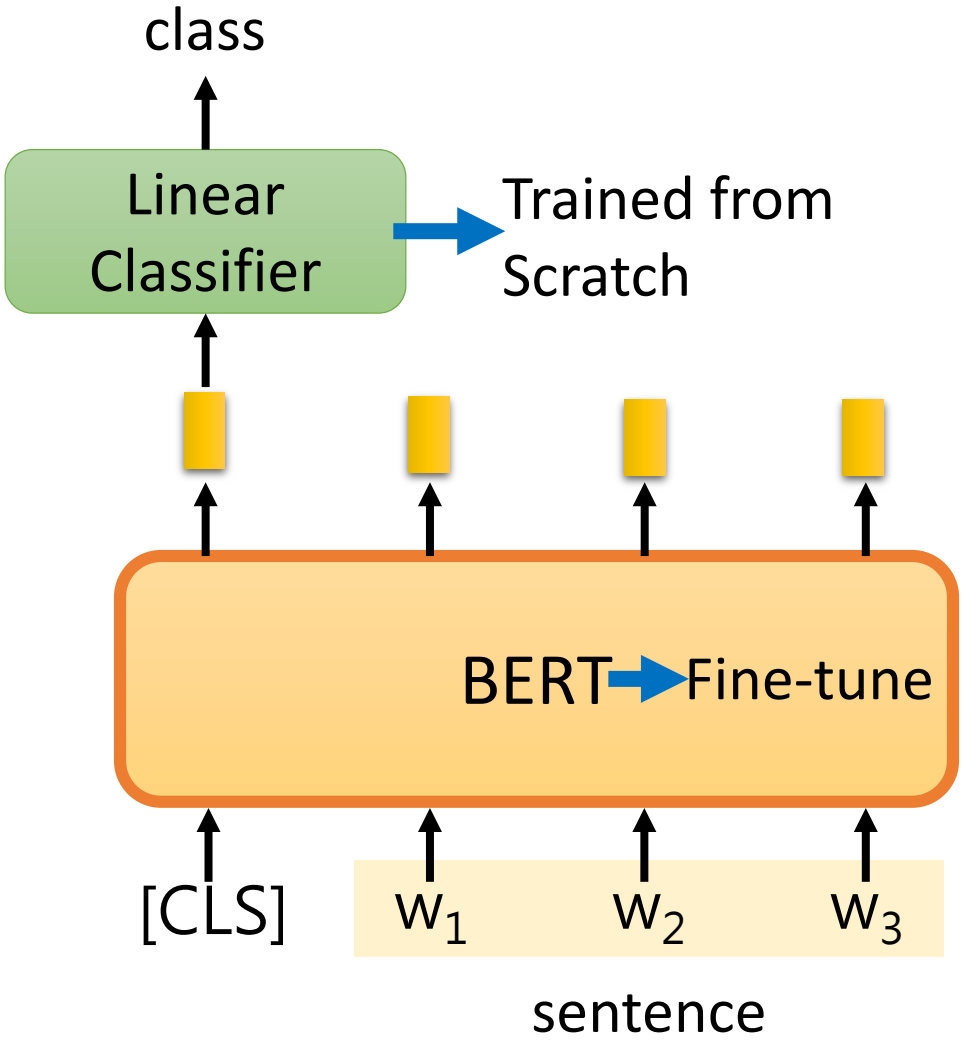
- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI**: natural language inference over question answering data
- **SST-2**: sentiment analysis
- **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B**: semantic textual similarity
- **MRPC**: microsoft paraphrase corpus
- **RTE**: a small natural language inference corpus

| System | MNLI-(m/mm) | QQP | QNLI | SST-2 | CoLA | STS-B | MRPC | RTE | **Average** |
|---|---|---|---|---|---|---|---|---|---|
| | 392k | 363k | 108k | 67k | 8.5k | 5.7k | 3.5k | 2.5k | - |
| 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 | 79.8 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 87.4 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.1 |
| $\text{BERT}_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.5 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| $\text{BERT}_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **92.7** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **82.1** |

Note that $\text{BERT}_{\text{BASE}}$ was chosen to have the same number of parameters as OpenAI GPT.

[Devlin et al., 2018]

# How to use BERT – Case 1

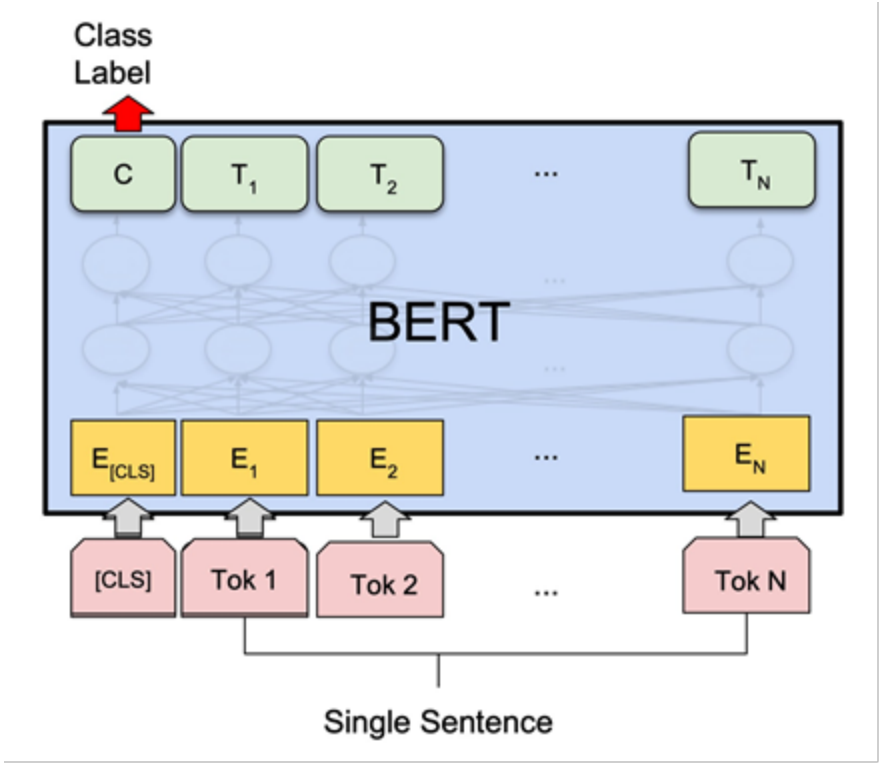



Input: single sentence, output: class

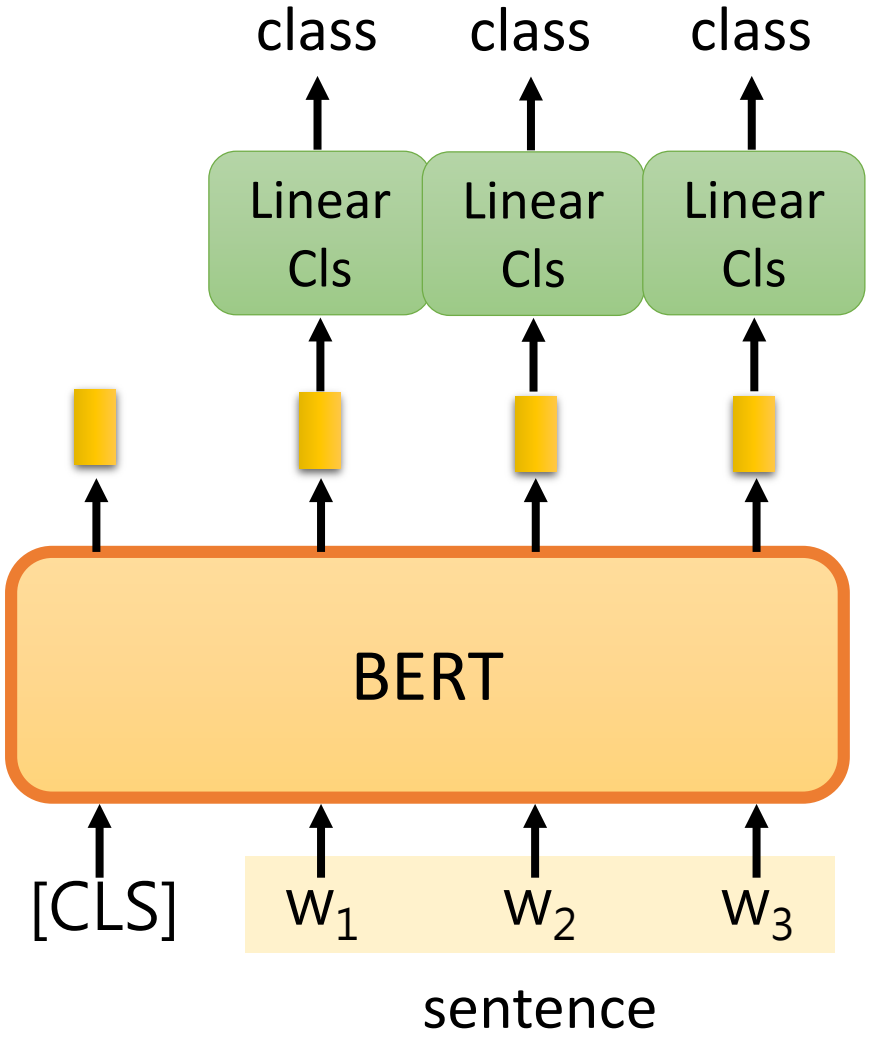
Example:
Sentiment analysis
Document Classification

# Sentence Classification

CLS token is used to provide classification decision

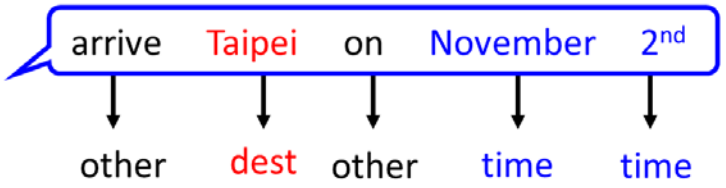


# How to use BERT – Case 2

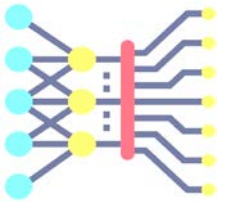



Input: single sentence,
output: class of each word

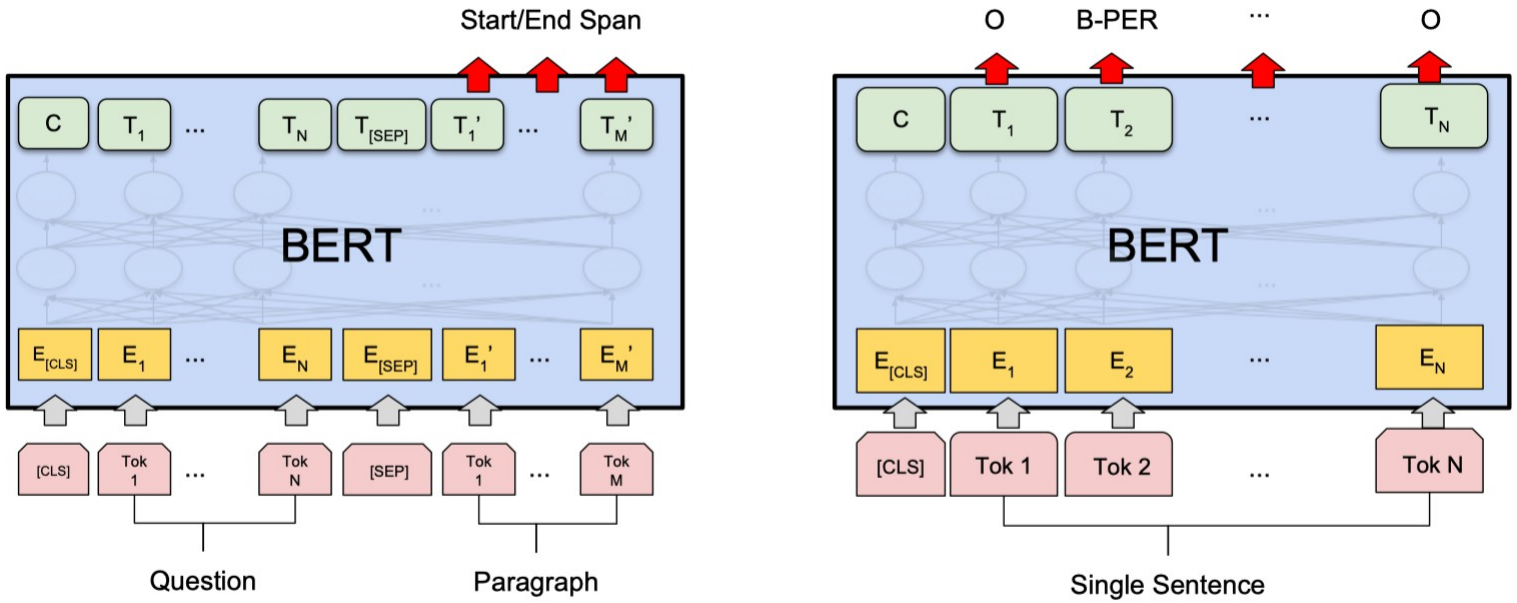
Example: Slot filling



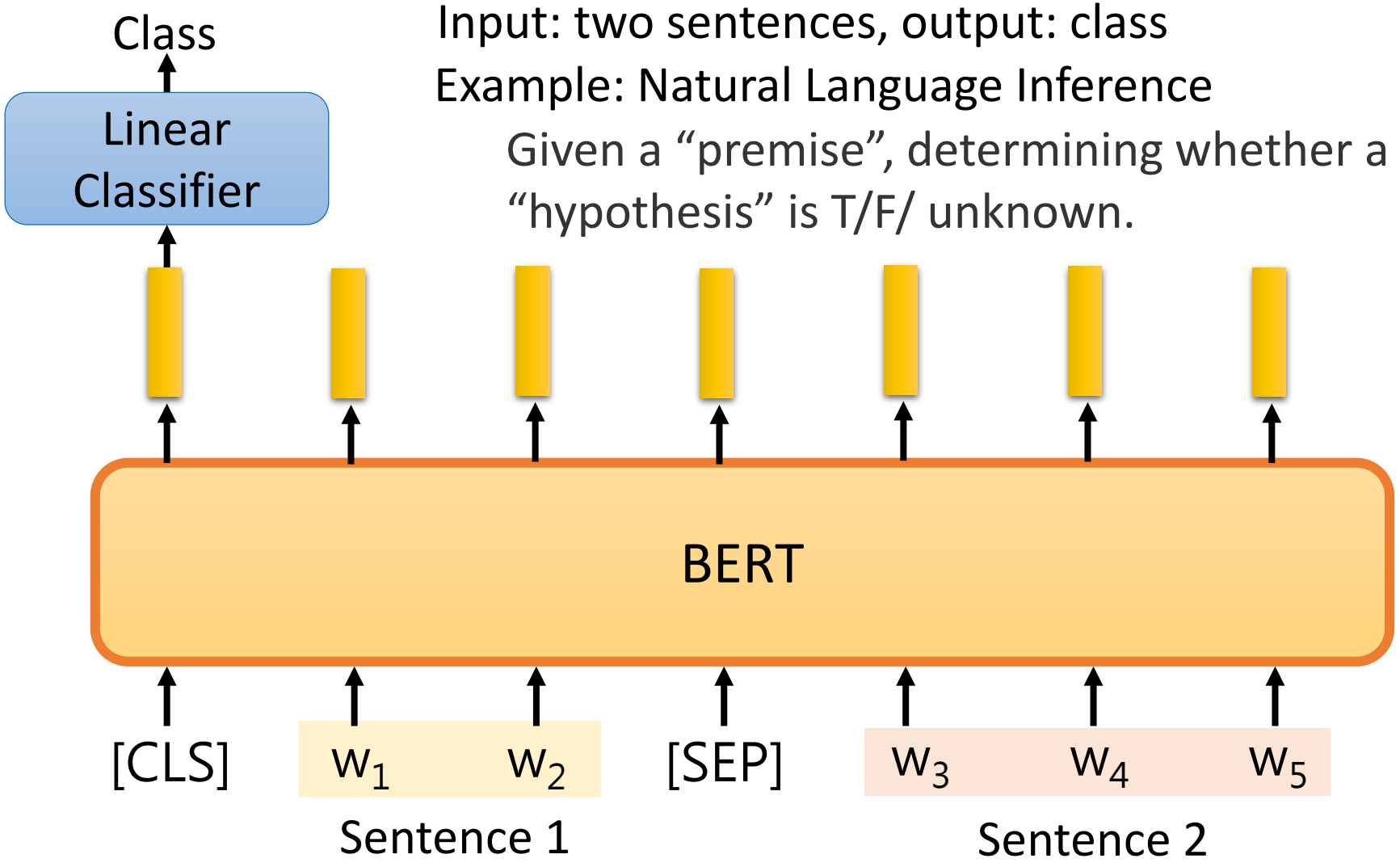
# Tagging with BERT



- 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

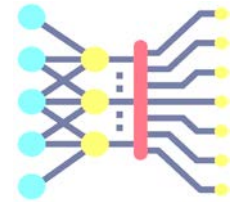



Input: two sentences, output: class

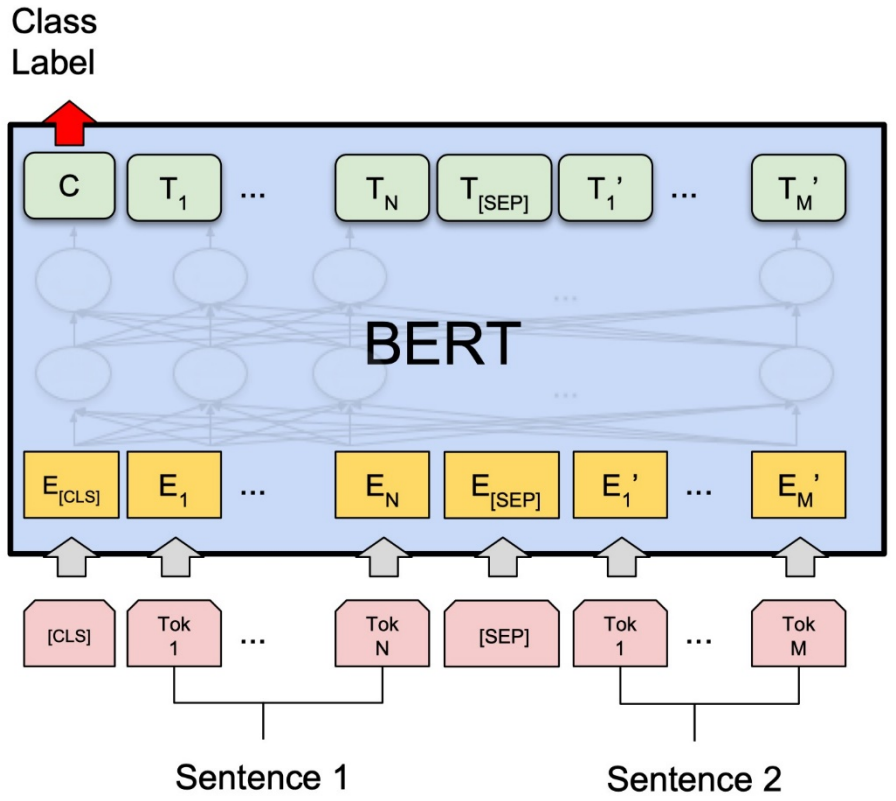
Example: Natural Language Inference

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

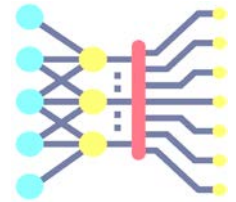
# Sentence-pair Classification with BERT



- 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

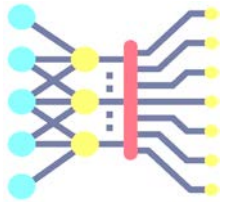


# Results



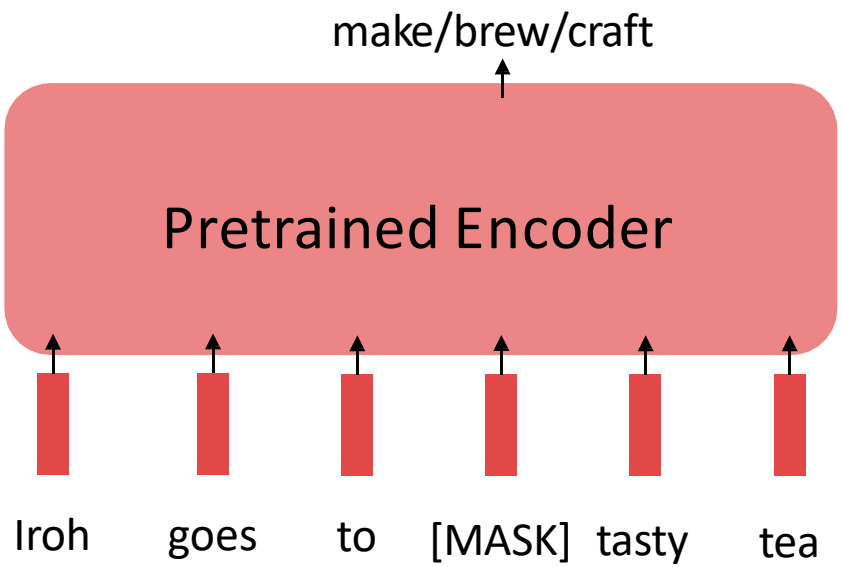
- Fine-tuned BERT outperformed previous state of the art on 11 NLP tasks
- Since then was applied to many more tasks with similar results
- The larger models perform better, but even the small BERT performs better than prior methods
- Variants quickly outperformed human performance on several tasks, 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

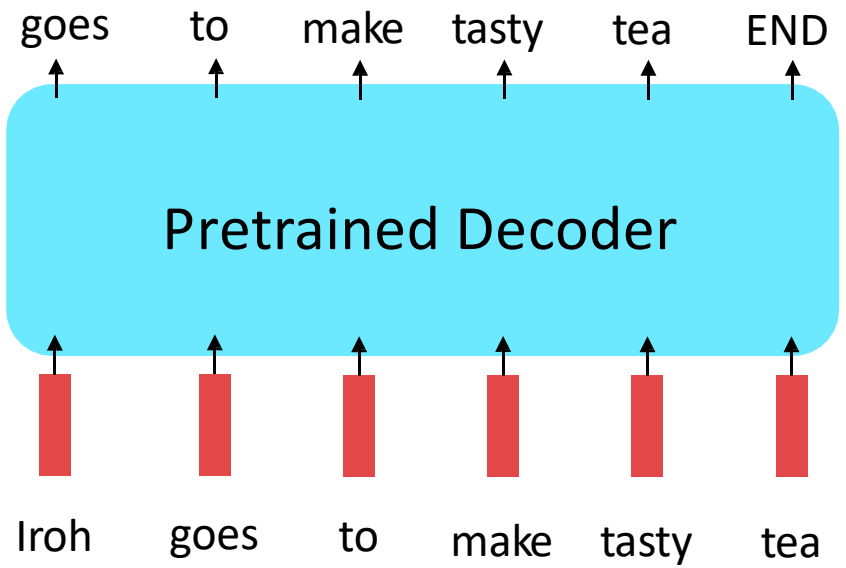
# Limitations of pretrained encoders



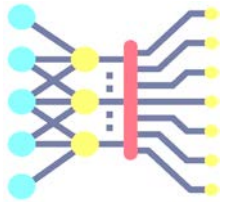
Those results looked great! Why not used pretrained encoders for everything?

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.



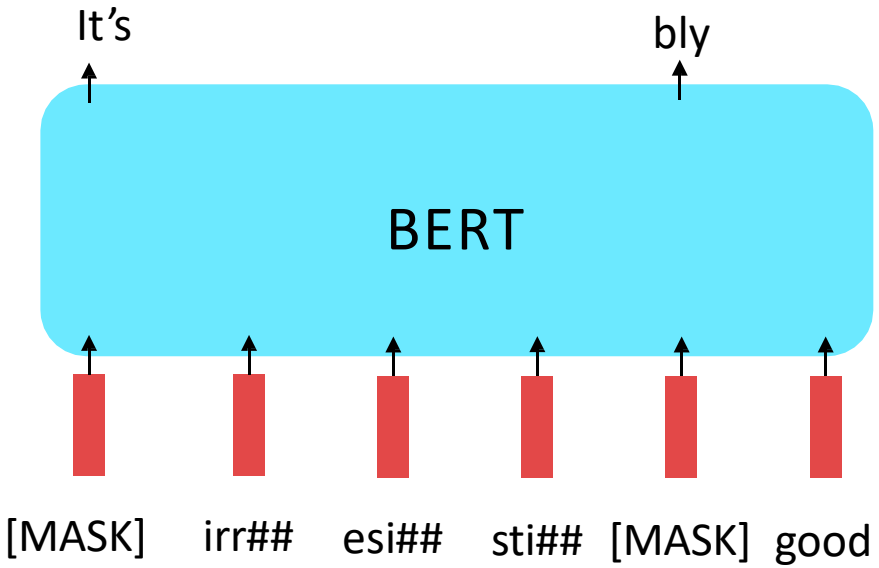
# Extensions of BERT

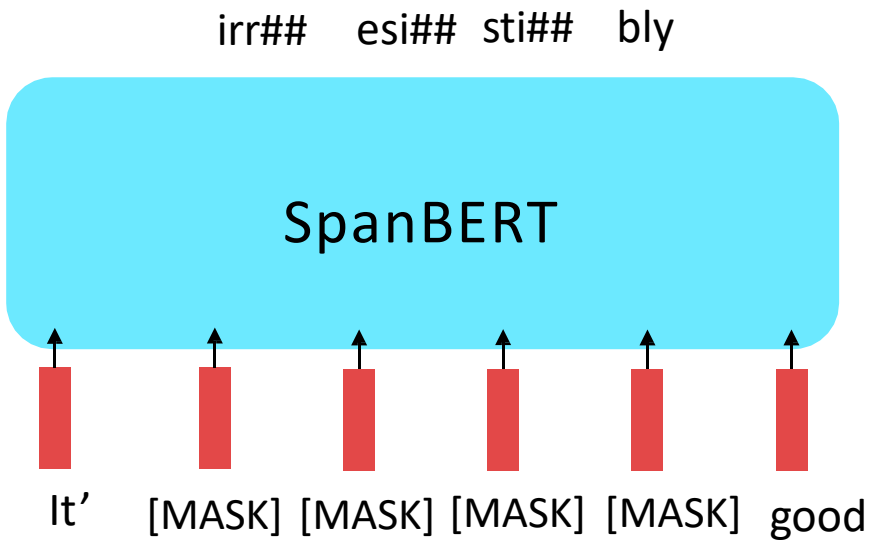


There are a lot of BERT variants like RoBERTa, SpanBERT, +++

Some generally accepted improvements to the BERT pretraining formula:

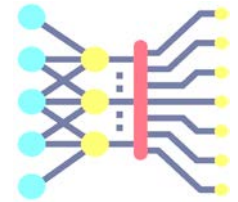
- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task




[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]