Introduction to Large Language Models

Spring 2026

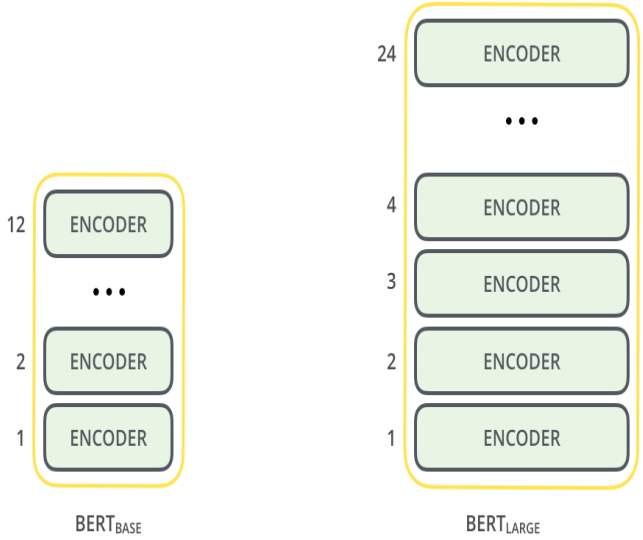
Transformer-Based Architectures/Models, and BERT

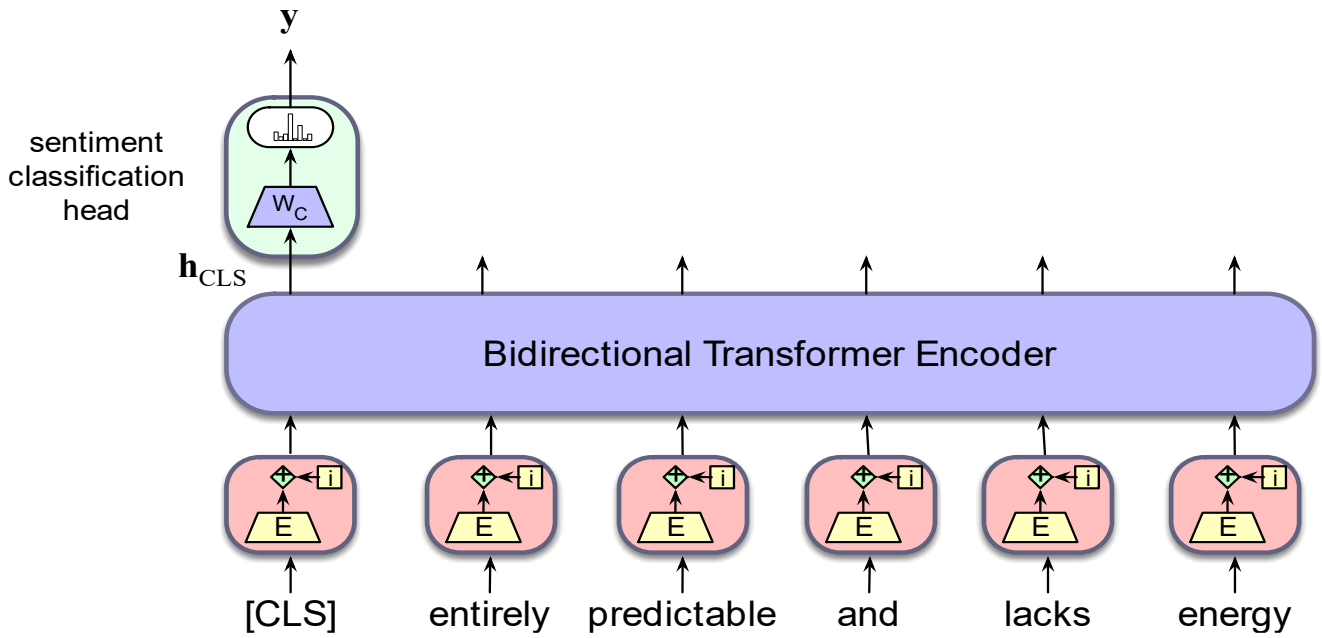
(Some slides adapted from Ralph Grishman at NYU, Yejin Choi at UWashington, N. Tomura at UDepaul, Jurafsky and Martin, CS224N, CS224d at Stanford and other resourses on the web)

Task head

- In Transformer architectures (BERT, GPT, LLaMA, etc.), the **Transformer blocks produce general-purpose hidden representations**.
- But the model still needs a **final layer** that turns those representations into something useful for a specific task.
- That final layer is called a:

  Task head
  (aka output head, prediction head, classification head, LM head)
- It is simply a small neural module sitting *on top of the Transformer encoder or decoder*.





Task head

Transformers output **vectors**, not actual task answers.

A head converts the Transformer's output representations into:

- class labels
- next-token probabilities
- start/end spans (QA)
- token labels (NER, POS tagging)
- sentence embeddings
- regression values

Different tasks → different heads.

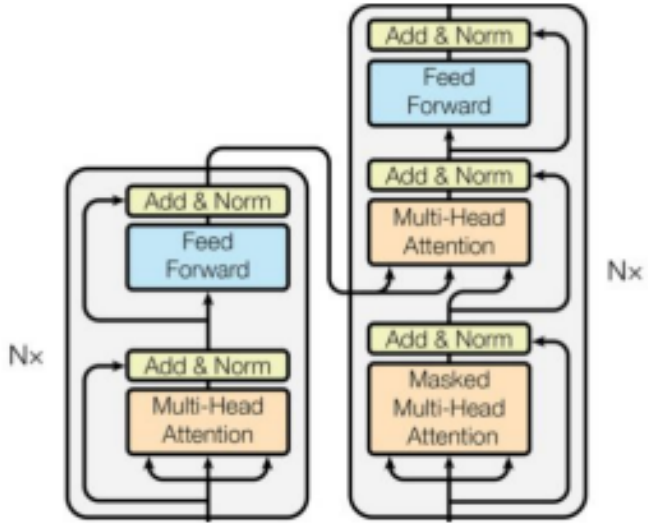
Common Types of Task Heads

- Language Modeling Head (LM Head)
- Next Sentence Prediction Head (NSP Head)
- Sequence Classification Head
- Token Classification Head (For NER, POS tagging, etc.)
- Question Answering Head (SQuAD-style)
- Masked Language Modeling Head (MLM Head)

Overview of Transformer-based models

3 categories of Transformer-based models:

Encoder-decoder	Text to text	T5, mT5, ByT5

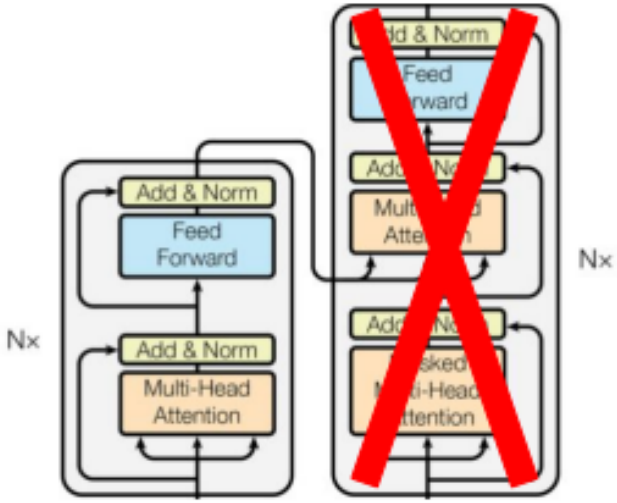


Source: https://cme295.stanford.edu/slides/fall25-cme295-lecture2.pdf

Overview of Transformer-based models

3 categories of Transformer-based models:

Encoder-decoder	Text to text	T5, mT5, ByT5
Encoder-only	Projection of embedding for class prediction (e.g. sentiment extraction)	BERT, DistilBERT, RoBERTa

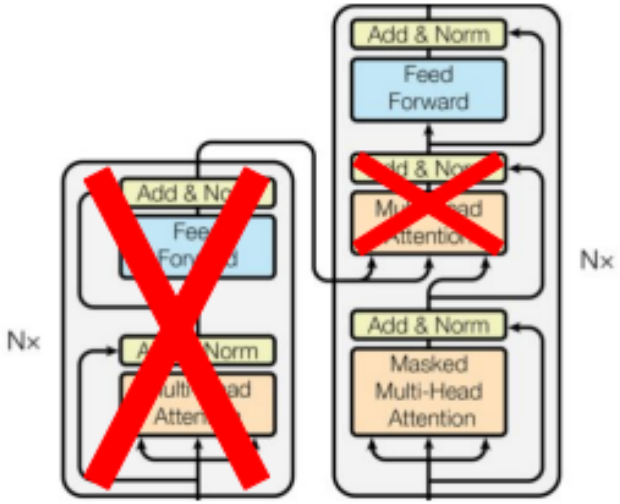


Source: https://cme295.stanford.edu/slides/fall25-cme295-lecture2.pdf

Overview of Transformer-based models

3 categories of Transformer-based models:

Encoder-decoder	Text to text	T5, mT5, ByT5
Encoder-only	Projection of embedding for class prediction (e.g. sentiment extraction)	BERT, DistilBERT, RoBERTa
Decoder-only	Text to text	GPT series

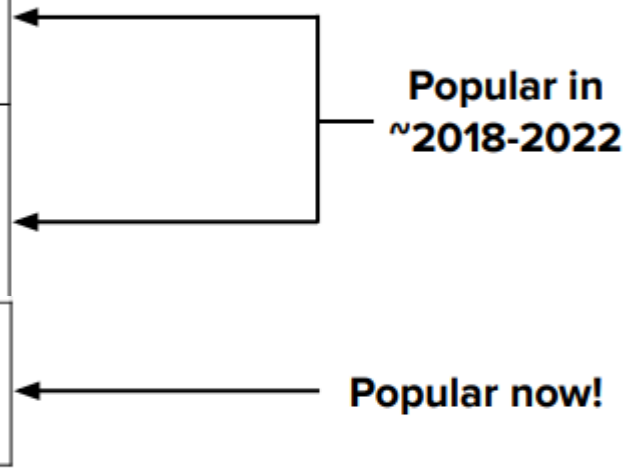


Source: https://cme295.stanford.edu/slides/fall25-cme295-lecture2.pdf

Overview of Transformer-based models

3 categories of Transformer-based models:

Encoder-decoder	Text to text	T5, mT5, ByT5
Encoder-only	Projection of embedding for class prediction (e.g. sentiment extraction)	BERT, DistilBERT, RoBERTa
Decoder-only	Text to text	GPT series



Source: https://cme295.stanford.edu/slides/fall25-cme295-lecture2.pdf

ELMO and BERT

Deep contextualized word representations

Matthew E. Peters[†], **Mark Neumann**[†], **Mohit Iyyer**[†], **Matt Gardner**[†],
{matthewp,markn,mohiti,mattg}@allenai.org

Christopher Clark[*], **Kenton Lee**[*], **Luke Zettlemoyer**[†*]
{csquared,kentonl,lsz}@cs.washington.edu

[†]Allen Institute for Artificial Intelligence
[*]Paul G. Allen School of Computer Science & Engineering, University of Washington

Abstract

We introduce a new type of *deep contextualized* word representation that models both (1) complex characteristics of word use (e.g., syn-

guage model (LM) objective on a large text corpus. For this reason, we call them ELMo (Embeddings from Language Models) representations. Unlike previous approaches for learning contextu-

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Rad-

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as addi-

ELMo paper
[submitted in Feb 2018]

Elmo

BERT paper
[submitted in Oct 2018]

Bert

ELMO

- **ELMo (Peters et al., 2018)** is a deep contextual word-embedding model that came *before* BERT. It produces word vectors that depend on the **entire sentence**, not just the word itself.
- **Key Characteristics**
  - **Contextual embeddings** → The embedding of a word changes depending on context
    - "bank" in *river bank* vs. *money bank*
  - **Bidirectional LSTM-based** → Built using stacked BiLSTMs, not Transformers
  - **Trained with a language modeling objective** → Predict next/previous words
  - **Deep representations** → Combines multiple internal LSTM layer outputs
- **Strengths**
  - Excellent at handling **polysemy** (words with multiple meanings)
  - Strong improvements in tasks like NER, sentiment, QA (when introduced)

BERT

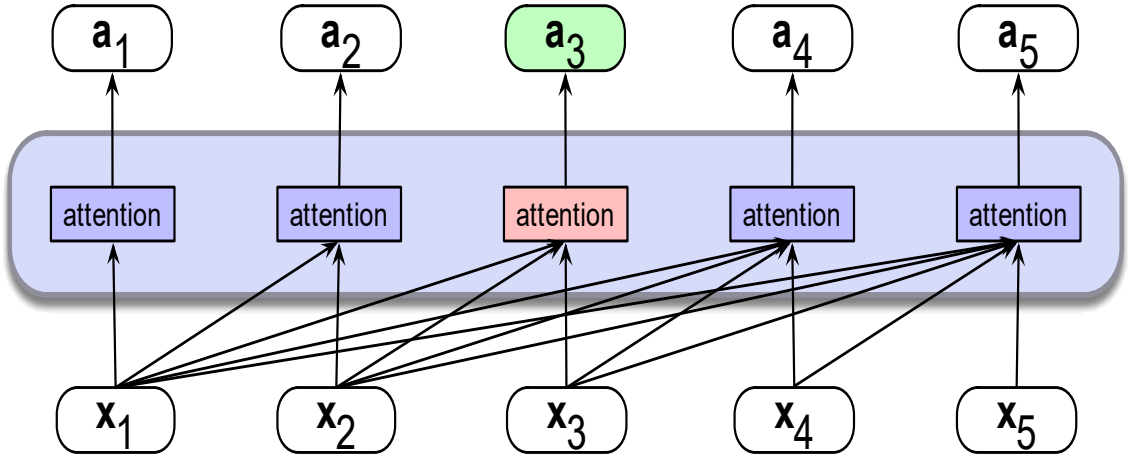
- BERT = **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
- BERT is a **pre-trained language model** introduced by Google AI in 2018. It's based on the **Transformer** architecture and is notable for being *bidirectional*, meaning it reads text **both left-to-right and right-to-left simultaneously**, which helps it understand context much more deeply.
- **Key Ideas Behind BERT**
  - **Bidirectional attention** → Better understanding of context within a sentence
  - **Masked Language Modeling (MLM)** → Randomly masks words and predicts them
  - **Next Sentence Prediction (NSP)** → Learns relationships between sentences
  - **Fine-tuning friendly** → You can adapt it easily for tasks like classification, Q&A, NER, etc.

Source: https://cme295.stanford.edu/slides/fall25-cme295-lecture2.pdf

Masked Language Modeling

- We've seen autoregressive (causal, left-to-right) LMs.
- But what about tasks for which we want to peak at future tokens?
  - Especially true for tasks where we map each input token to an output token
- **Bidirectional** encoders use **masked self-attention** to
  - map sequences of input embeddings (x_1,...,x_n)
  - to sequences of output embeddings of the same length (h_1,...,h_n),
  - where the output vectors have been contextualized using information from the entire input sequence, both left and right side.

Bidirectional Self-Attention

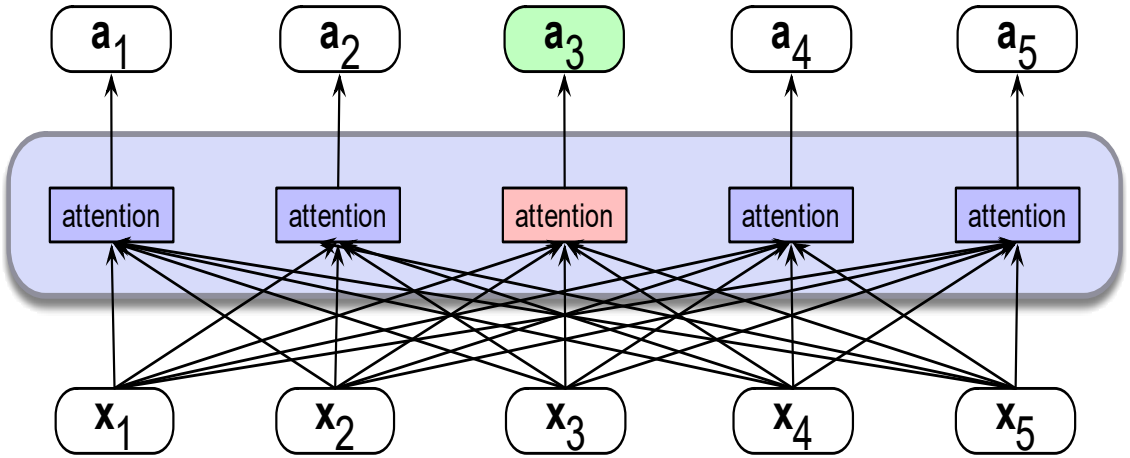


a) A causal self-attention layer

- A token can **only attend to previous tokens**, not future ones. Because causal models **generate text one token at a time**, predicting the next token.
  If they could see future tokens, generation would be impossible.

✔ **Properties**

- Enforces temporal ordering
- Triangular attention mask
- Best for: **text generation**



b) A bidirectional self-attention layer

- A token can attend to **any** other token — both *past* and *future* tokens.
- Because BERT's goal is **understanding text**, not generating it. Language understanding benefits from seeing the *whole sentence* at once.
- **Properties**
  - Fully bidirectional
  - No restrictions on attention
  - Best for: **classification, QA, NER, semantic tasks**
- **Not suitable for autoregressive generation.** Because, it "cheats" — it can see future tokens.

Easy! We just remove the mask

Casual self-attention

N

q1·k1	−∞	−∞	−∞
q2·k1	q2·k2	−∞	−∞
q3·k1	q3·k2	q3·k3	−∞
q4·k1	q4·k2	q4·k3	q4·k4

N

$$\mathbf{head} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

- Causal (autoregressive) attention — used in GPT — for next-token prediction.
- Apply a lower-triangular mask on the $QK^\intercal$ scores so tokens cannot attend to the future.

Bidirectional self-attention

N

q1·k1	q1·k2	q1·k3	q1·k4
q2·k1	q2·k2	q2·k3	q2·k4
q3·k1	q3·k2	q3·k3	q3·k4
q4·k1	q4·k2	q4·k3	q4·k4

N

$$\mathbf{head} = \text{softmax}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\mathbf{V}$$

- "Removing the causal mask" creates bidirectional attention.
- BERT's training objective requires **masked language modeling (MLM)**

Masked training intuition

- **Next-Token Training**
  - For **left-to-right LMs**, the model tries to predict the last word from prior words:

    The water of Walden Pond is so beautifully _____
  - And we train it to improve its predictions.
- **Masked Language Modeling (MLM)**
- **MLM** is the pretraining objective used by BERT and many encoder-based Transformer models.
  - For **bidirectional masked LMs**, the model tries to predict one or more words from all the rest of the words:

    The _____ of Walden Pond _____ so beautifully blue

  - Hide (mask) some tokens in the input and train the model to predict the missing words using full bidirectional context.
  - The model generates a probability distribution over the vocabulary for each missing token.
  - We use the cross-entropy loss from each of the model's predictions to drive the learning process.

MLM training in BERT

- Randomly select ~15% of tokens for prediction.

  Given a sentence:

  `The quick brown fox jumps over the lazy dog.`

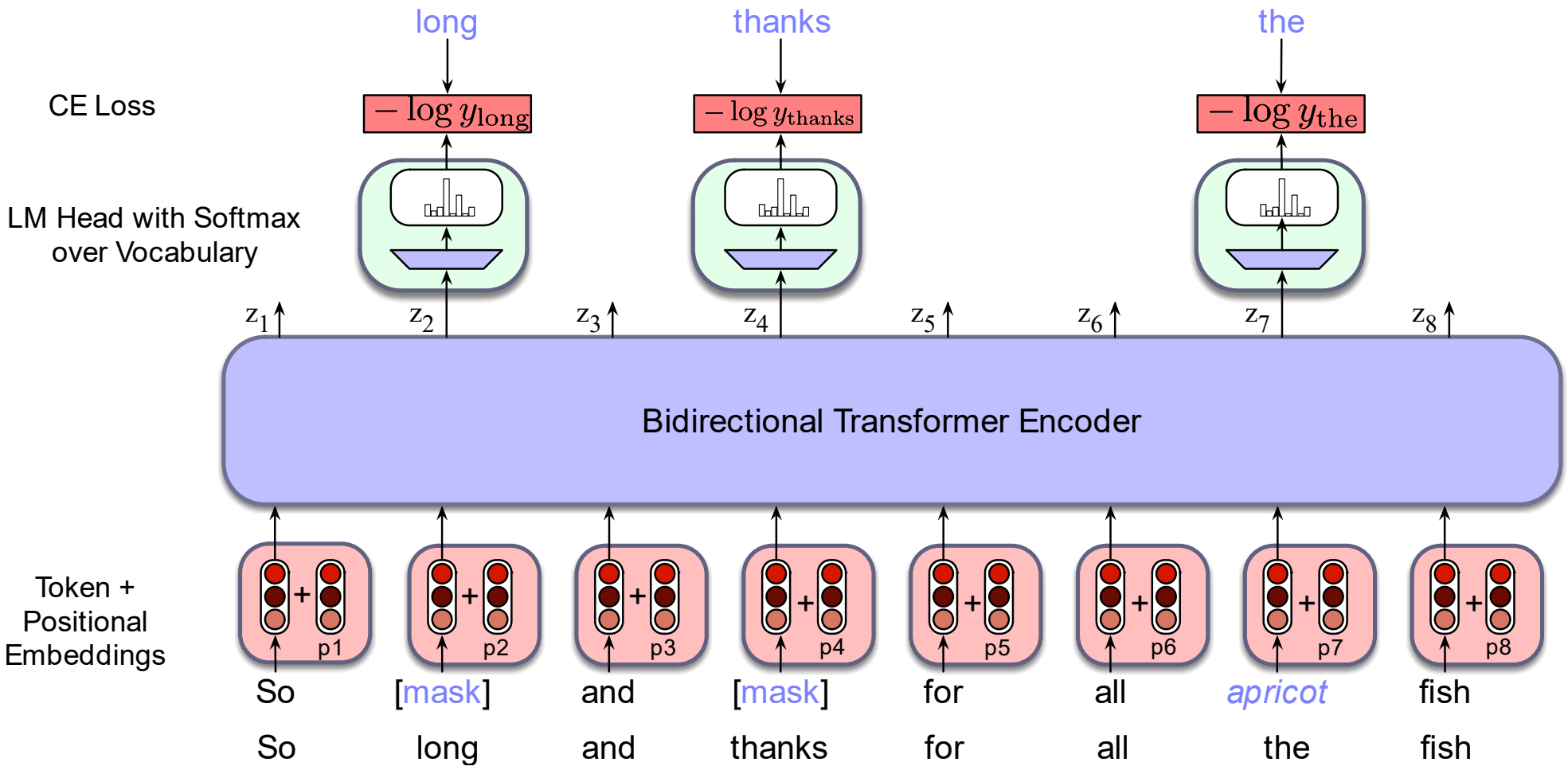
  Example masking:

  `The quick brown [MASK] jumps over the [MASK] dog.`

- For each selected token:
  - **80%** → replaced with `[MASK]`
  - **10%** → replaced with a random token
  - **10%** → kept unchanged

- During training, the model outputs predictions only for masked positions.

MLM head In detail



Why MLM Is Needed for Bidirectional Attention

- Without MLM, BERT would not know how to learn meaningful token representations because:
  - Bidirectional attention means the model *sees* the answer.
  - Masking hides the answer at selected positions.
  - This forces the model to rely on contextual clues.

Task	MLM (BERT)	Next-token prediction (GPT)
Context	Full bidirectional	Only left context
Masking	Yes (at input)	No
Generates text?	❌ Cannot	✔ Yes
Best for	Understanding	Generation
Predicts	Missing words	Next word

MLM loss

The LM head takes output of final transformer layer L, multiplies it by unembedding layer and turns into probabilities:

$$\mathbf{u}_i = \mathbf{h}_\mathbf{i}^\mathbf{L}\ \mathbf{E}^\mathbf{T}$$

$$\mathbf{y}_i = \text{softmax}(\mathbf{u_i})$$

E.g., for the x_i corresponding to "long", the loss is the probability of the correct word *long*, given output h^L_i):

$$L_{MLM}(x_i) = -\log P(x_i|\mathbf{h}_i^L)$$

We get the gradients by taking the average of this loss over the batch

$$L_{MLM} = -\frac{1}{|M|}\sum_{i\in M}\log P(x_i|\mathbf{h}_i^L)$$

Next Sentence Prediction

- **Next Sentence Prediction (NSP)** — the second pretraining task used in the original **BERT** paper.
- Given 2 sentences the model predicts if they are a real pair of adjacent sentences from the training corpus or a pair of unrelated sentences.
- BERT introduces two special tokens
  - [CLS] is prepended to the input sentence pair,
  - [SEP] is placed between the sentences, and also after second sentence
- And two more special tokens
- [1^{st} segment] and [2^{nd} segment]
- These are added to the input embedding and positional embedding
- h^L_{CLS} from the final layer [CLS] token is input to classifier head (weights W_{NSP}) that predicts two classes:.

$$y_i = \text{softmax}(h^L_{CLS} W_{NSP})$$

How NSP Works (Step-by-Step)

1. Build sentence pair A–B

Example positive pair (IsNext):

A: Alice went to the store.

B: She bought some milk.

Example negative pair (NotNext):

A: Alice went to the store.

B: The car engine failed to start.

2. Label the pair

- **50%** → *IsNext*
- **50%** → *NotNext*

3. Encode with special tokens

BERT adds a special token `[CLS]` **at the beginning of every input** :

[CLS] Sentence A [SEP] Sentence B [SEP]

BERT uses [CLS] as a container for the entire input's meaning.

4. The model pools the representation from [CLS].

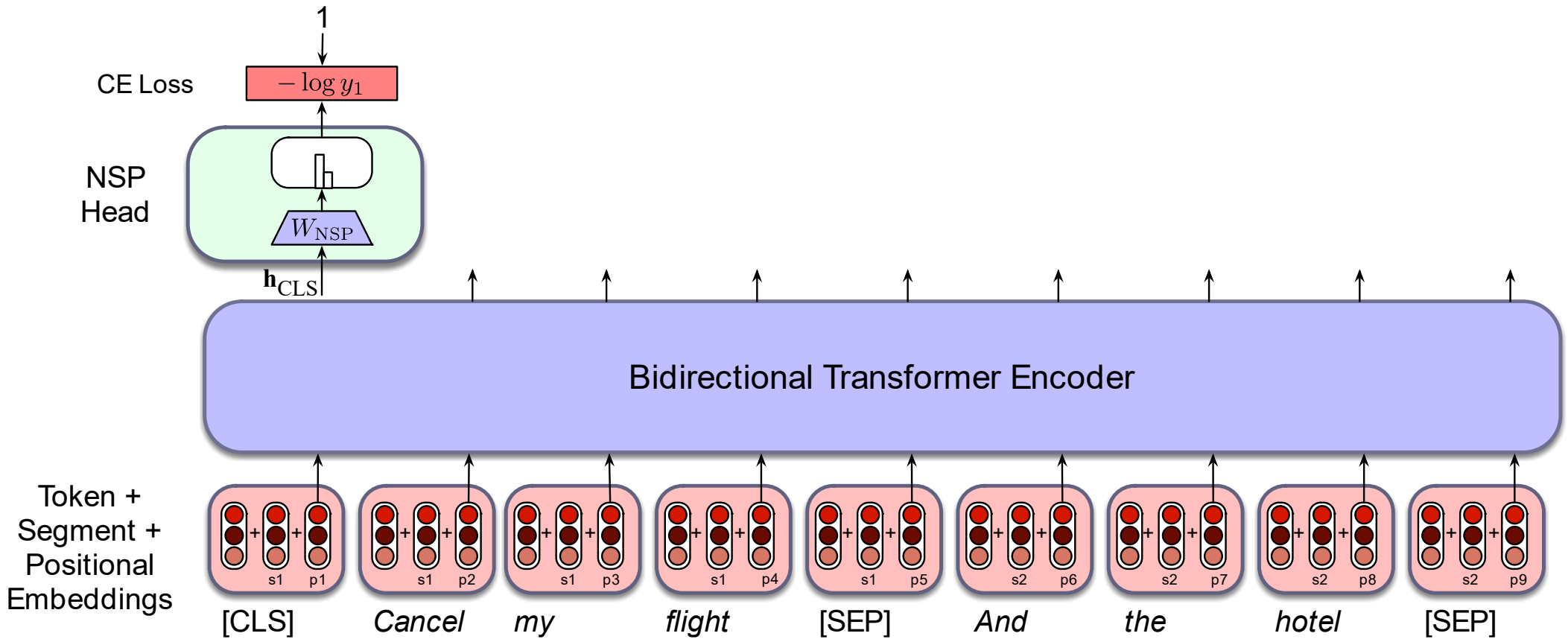
BERT takes the final hidden state of the [CLS] token.

[CLS] vector is a single, unified vector representing both sentences and their relationship

This single vector is used to predict:

IsNext OR NotNext

NSP Loss with classification head



More details

- Original model was trained with 40 passes over training data
- Some models (like RoBERTa) drop NSP loss
- Tokenizer for multilingual models is trained from stratified sample of languages (some data from each language)
- Multilingual models are better than monolingual models with small numbers of languages
- With large numbers of languages, monolingual models in that language can be better
- The "curse of multilinguality" :
  As you train a single model on more languages, the performance per language tends to decrease — unless the model becomes much larger.

More details

- 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.”

More details

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

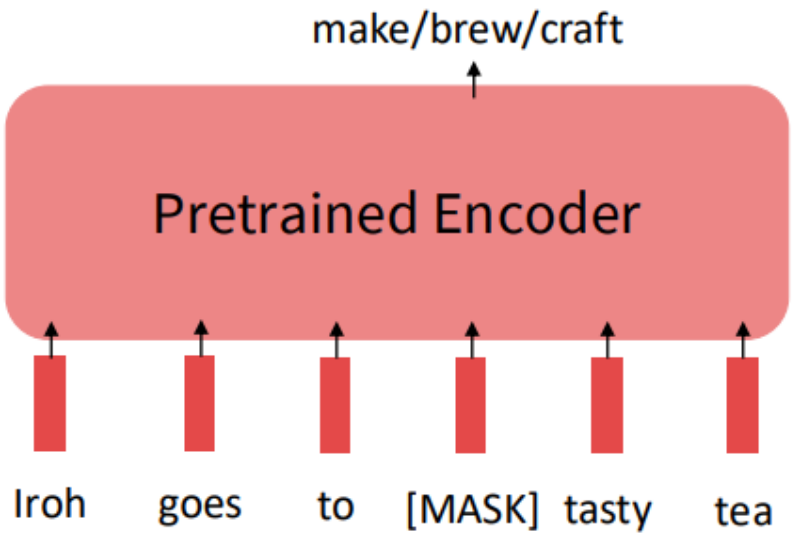
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	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**

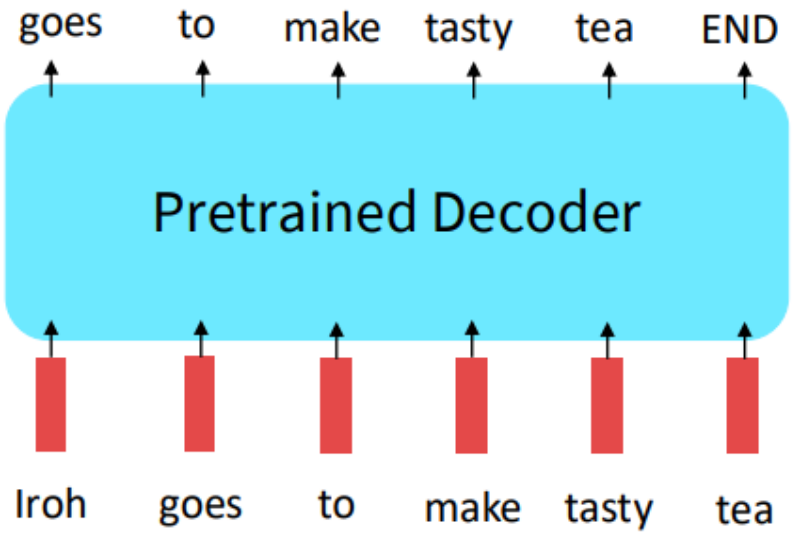
Devlin et al., 2018: https://arxiv.org/pdf/1810.04805

Limitations of pretrained encoders

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

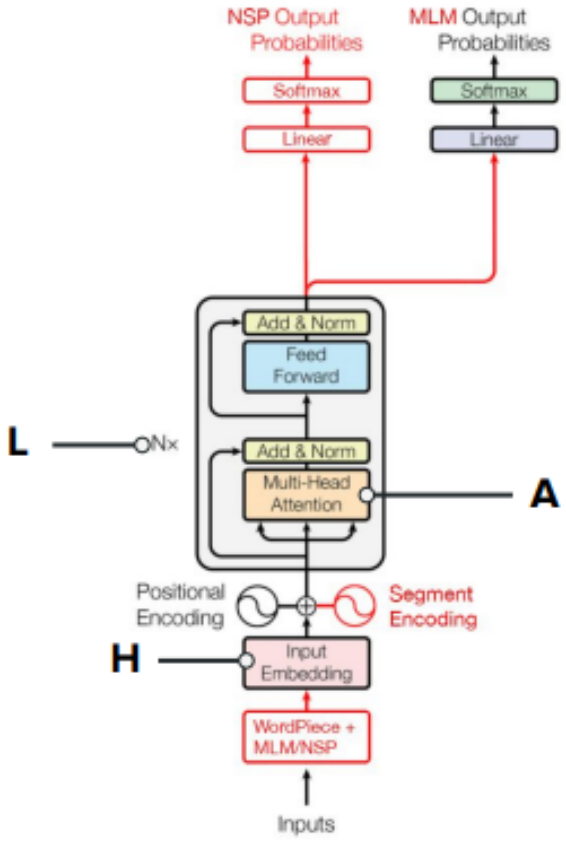
If your task involves generating sequences, consider using a pretrained decoder; BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.





Source: https://web.stanford.edu/class/cs224n/slides_w26/cs224n-2026-lecture07-pretraining.pdf

LLM Model Hyperparameters



Model.

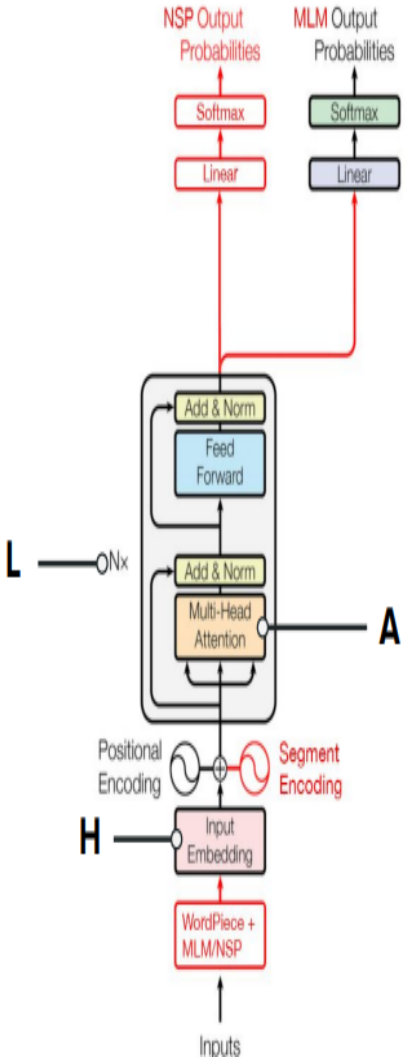
- **L** Layers.
- **H** Hidden layer size aka dimension of embeddings.
- **A** Attention heads operating in parallel.

Data.

- Language-specific / multilingual. Languages on which the model has been trained on.
- Cased / uncased. Whether inputs are converted to lowercase or not.

Source: https://cme295.stanford.edu/slides/fall25-cme295-lecture2.pdf

BERT Size-Scaled variants



	L	H	A	Parameters
BERT-Tiny	2	128	2	4M
BERT-Mini	4	256	4	11M
BERT-Small	4	512	8	30M
BERT-Medium	8	512	8	42M
BERT-Base	12	768	12	110M
BERT-Large	24	1024	16	340M

Performance Scales:

General rule from experiments:

- **BERT-Large > BERT-Base > Medium > Small > Mini > Tiny**

But:

- Small/Medium models often get **70–90%** of BERT-Base accuracy
- At **10–50x** faster inference

This is why scaled-down versions are useful.

The models represent a systematic scaling:

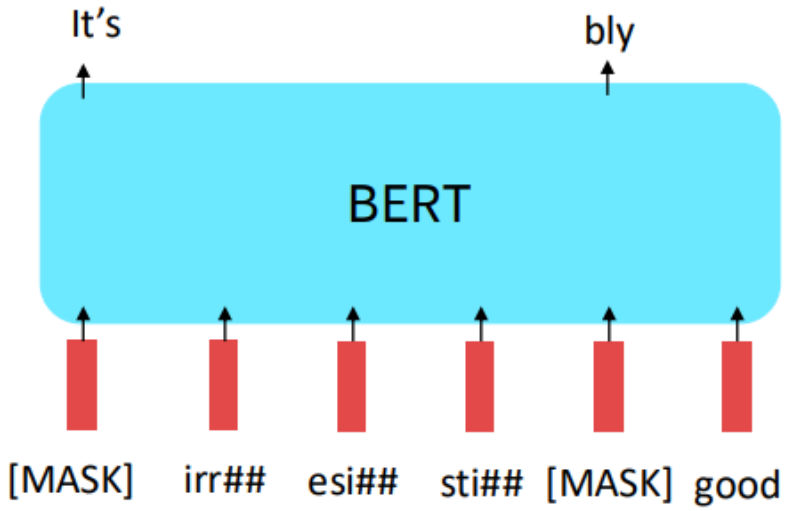
- More **layers** → deeper contextual understanding
- Larger **hidden size** → richer representations
- More **attention heads** → better modeling of token relationships
- More **parameters** → higher accuracy but higher compute

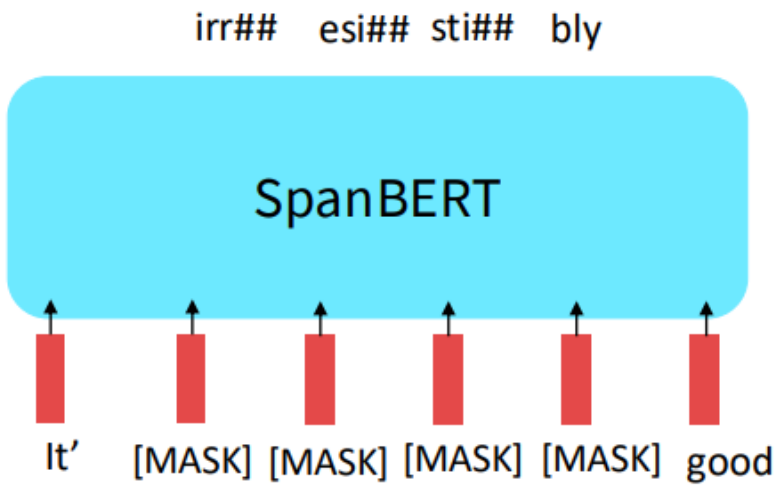
Architecture-Modified BERT Variants

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





Architecture-Modified BERT Variants

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

Architecture-Modified BERT Variants

Source: https://web.stanford.edu/class/cs224n/slides_w26/cs224n-2026-lecture07-pretraining.pdf

Liu et al., 2019; Joshi et al., 2020 : https://arxiv.org/abs/1907.11692

Architecture-Modified BERT Variants

These models change something in the architecture (layers, factorization, distillation, etc.) to make BERT faster, smaller, or better.

- **DistilBERT**

40% smaller
60% faster
Keeps **97%** of BERT-Base performance
Uses **knowledge distillation**

- **ALBERT (A Lite BERT)**

Drastically reduces parameters
Uses *two innovations*:
Factorized embeddings
Cross-layer parameter sharing
Very efficient while maintaining accuracy
ALBERT-xxlarge reached **SOTA** on GLUE when released

- **ELECTRA**

Introduced *replaced token detection* (RTD)
Much faster pretraining than BERT
Outperforms BERT with far less compute

- **SpanBERT**

Improves span-based tasks (QA, coreference)
Masks *spans*, not just tokens

- **RoBERTa**

“Robustly optimized BERT”
Removes NSP
Uses much more data
Better training schedule
Much stronger performance than original BERT

- **DeBERTa**

Adds **disentangled attention**
Uses **relative position encoding**
Very strong on many benchmarks

Training- Objective-Modified BERT Variants

- **BERT (Original)**

MLM (masked language modeling)

NSP (next-sentence prediction)

- **RoBERTa**

Removes NSP

Uses dynamic masking

Trains longer and on more data

→ Big improvement over BERT-Base

- **ALBERT**

Replaces NSP with **Sentence Order Prediction (SOP)**

→ Helps with coherence tasks

- **ELECTRA**

Replaces MLM with *Replaced Token Detection*

→ Learns from all tokens, not just masked ones

→ Far more sample-efficient

- **SpanBERT**

Span masking + span boundary objectives

→ Best for QA and coreference

- **TinyBERT / MobileBERT**

Use heavy **distillation**

Maintain strong performance with small size

Domain-Specific BERT Models

- Trained on specific domains:
  - **BioBERT** → biomedical text
  - **SciBERT** → scientific articles
  - **FinBERT** → finance news & filings
  - **ClinicalBERT** → clinical notes
  - **LegalBERT** → legal text
  - **Turkish BERT** / **CamemBERT** / **BERTje** → monolingual BERTs
  - **mBERT** → multilingual BERT (104 languages)
- These outperform vanilla BERT on domain-specific tasks.

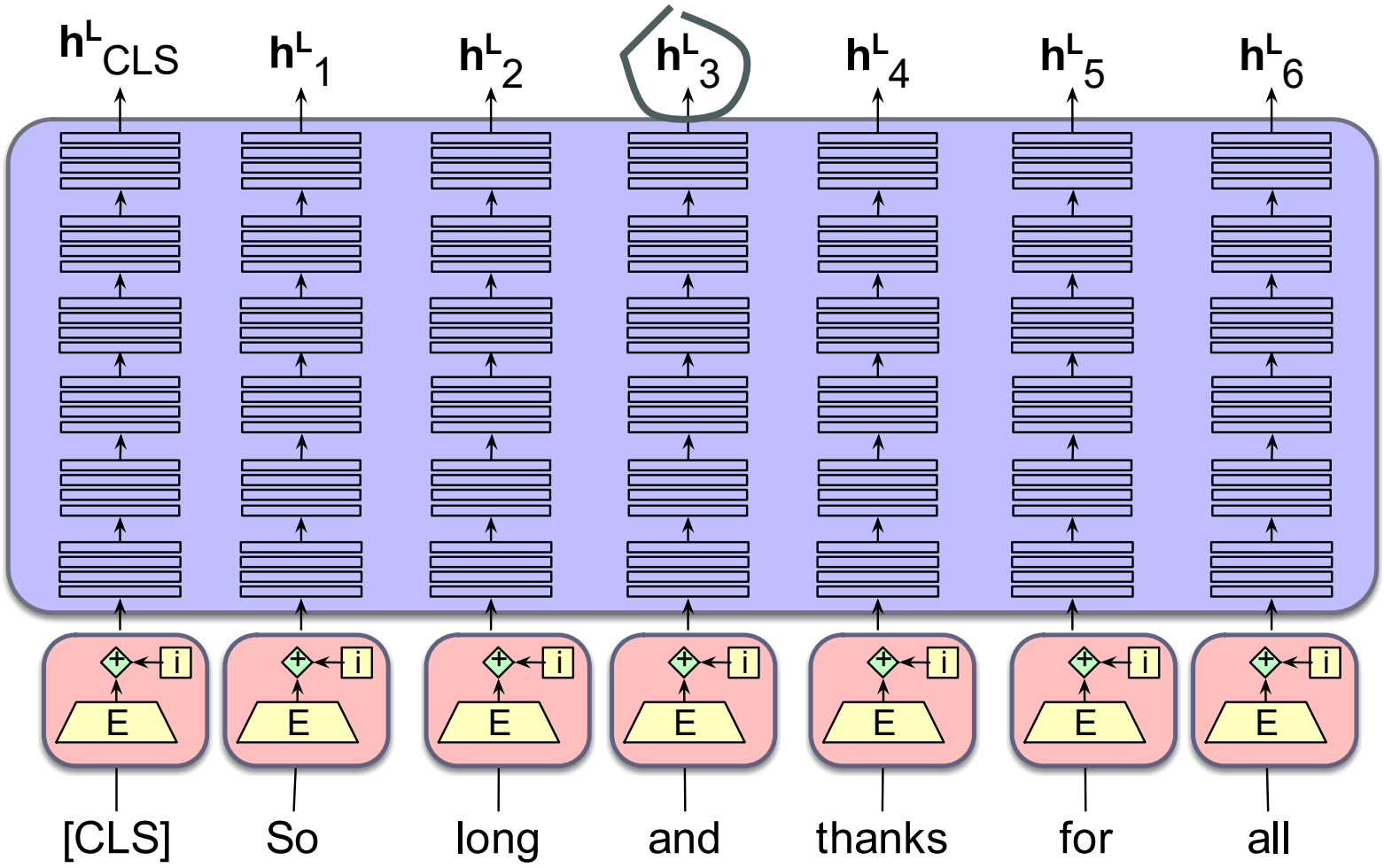
Contextual Embeddings

- **Contextual embeddings** are word representations whose meaning *depends on the context in which the word appears*.
- This means:

  The same word gets different embeddings depending on its surrounding words.

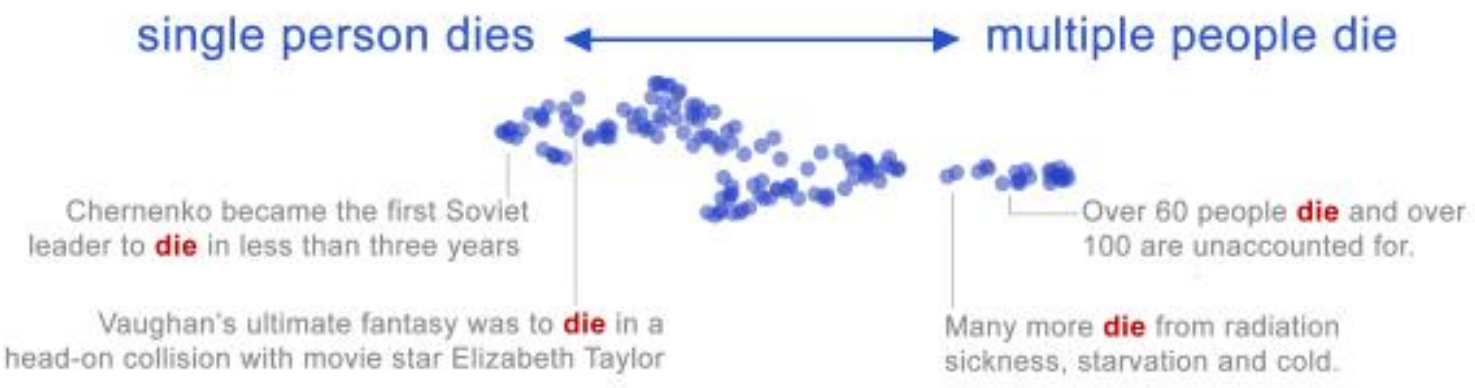
- This is the core feature introduced by models like **ELMo (2018)**, **BERT (2018)**, **GPT-style Transformers**, and later large language models.
- Static embeddings like **Word2Vec** or **GloVe** assign *one fixed vector* per word, which fails to capture multiple meanings. Contextual embeddings solve this.

Contextual Embeddings to represent words



Static vs Contextual Embeddings

- Static embeddings represent **word types** (dictionary entries)
- Contextual embeddings represent **word instances** (one for each time the word occurs in any context/sentence)



Word sense

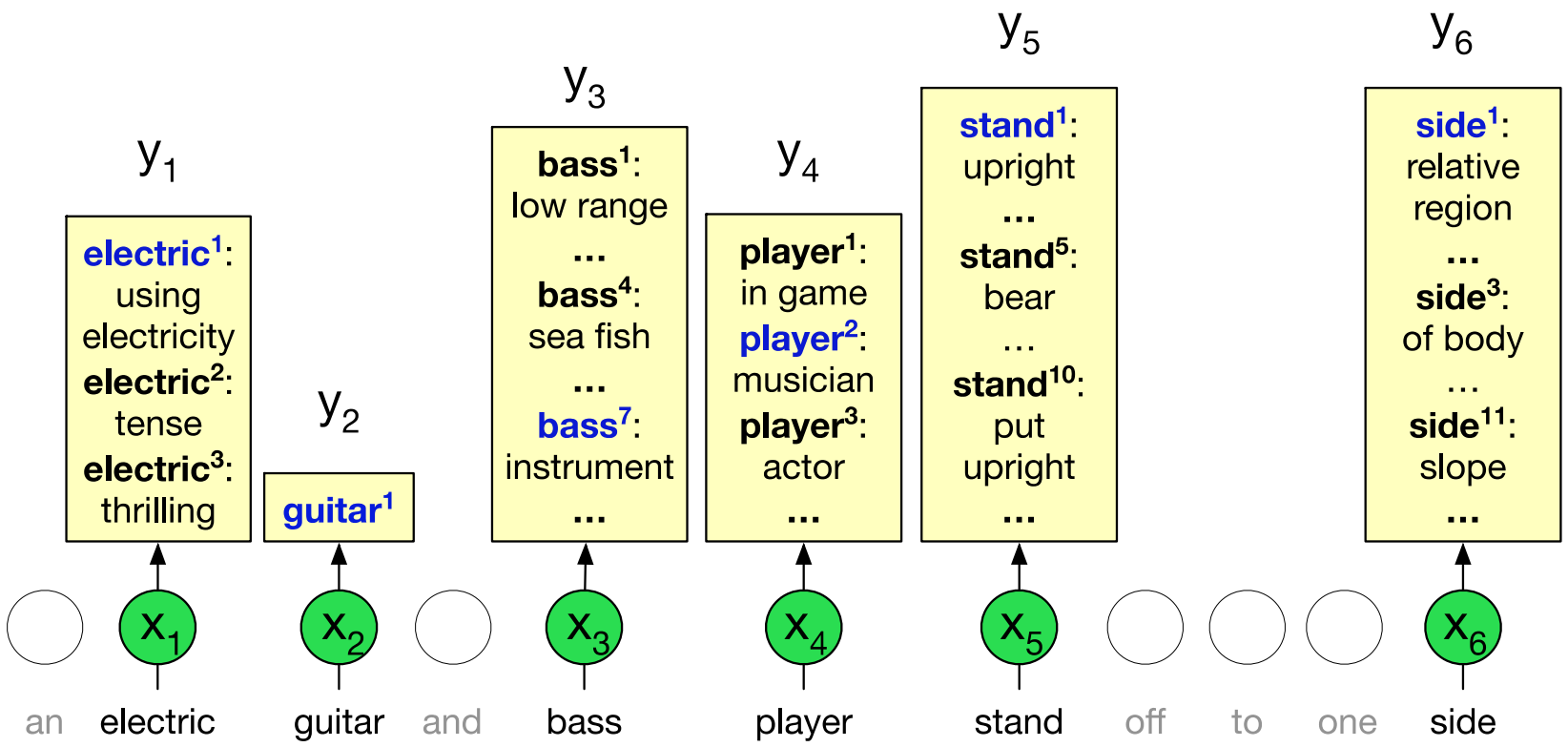
- Words are ambiguous
- A **word sense** is a discrete representation of one aspect of meaning

mouse^{1} : .... a *mouse* controlling a computer system in 1968.
mouse^{2} : .... a quiet animal like a *mouse*
bank^{1} : ...a *bank* can hold the investments in a custodial account ...
bank^{2} : ...as agriculture burgeons on the east *bank*, the river ...

- Contextual embeddings offer a continuous high-dimensional model of meaning that is more fine grained than discrete senses.

Word sense disambiguation (WSD)

- The task of selecting the correct sense for a word.



1-nearest neighbor algorithm for WSD

Melamud et al (2016), Peters et al (2018)

- At training time, take a sense-labeled corpus like SEMCOR
- Run corpus through BERT to get contextual embedding for each token
- E.g., pooling representations from last 4 BERT transformer layer
- Then for each sense s of word w for n tokens of that sense, pool embeddings:

$$\mathbf{v}_s = \frac{1}{n}\sum_i \mathbf{v}_i \qquad \forall \mathbf{v}_i \in \text{tokens}(s)$$

- At test time, given a token of a target word *t*, compute contextual embedding t and choose its nearest neighbor sense from training set

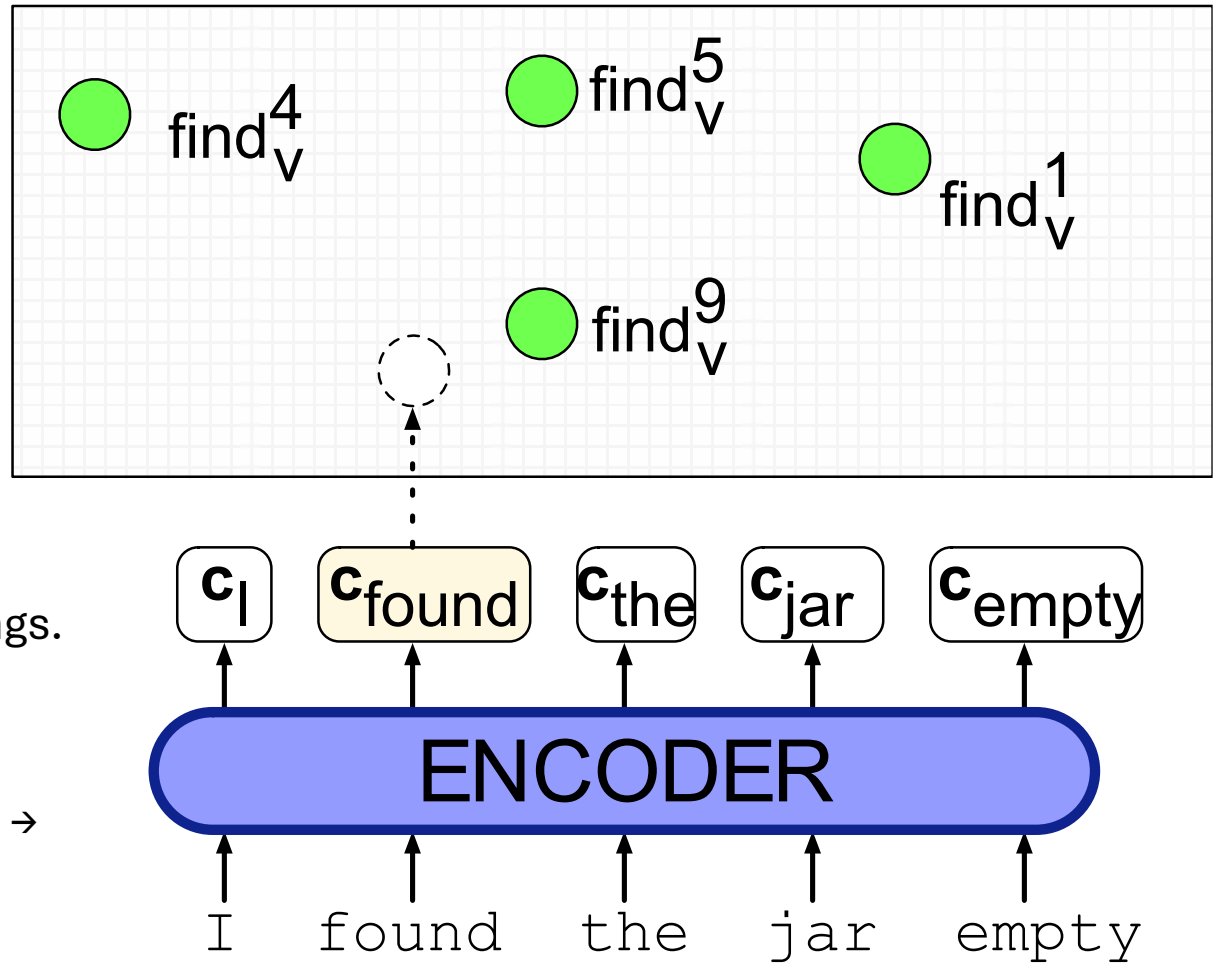
$$\text{sense}(t) = \underset{s \in \text{senses}(t)}{\text{argmax}}\ \text{cosine}(\mathbf{t}, \mathbf{v}_s)$$

1-nearest neighbor algorithm for WSD

Each green dot represents a training example of the verb "find" in different senses:

- **find$^{4}_{v}$**
- **find$^{5}_{v}$**
- **find$^{1}_{v}$**
- **find$^{9}_{v}$**

These numbers (1, 4, 5, 9) refer to *dictionary sense IDs*—for example, different meanings like: *"discover", "acquire", "establish", "come across"*, etc.



1. A sentence encoded into contextual word embeddings.
2. The embedding of *"found"* compared against known sense embeddings.
3. The algorithm selecting the **nearest sense example** → performing WSD using **1-nearest neighbor** classification.

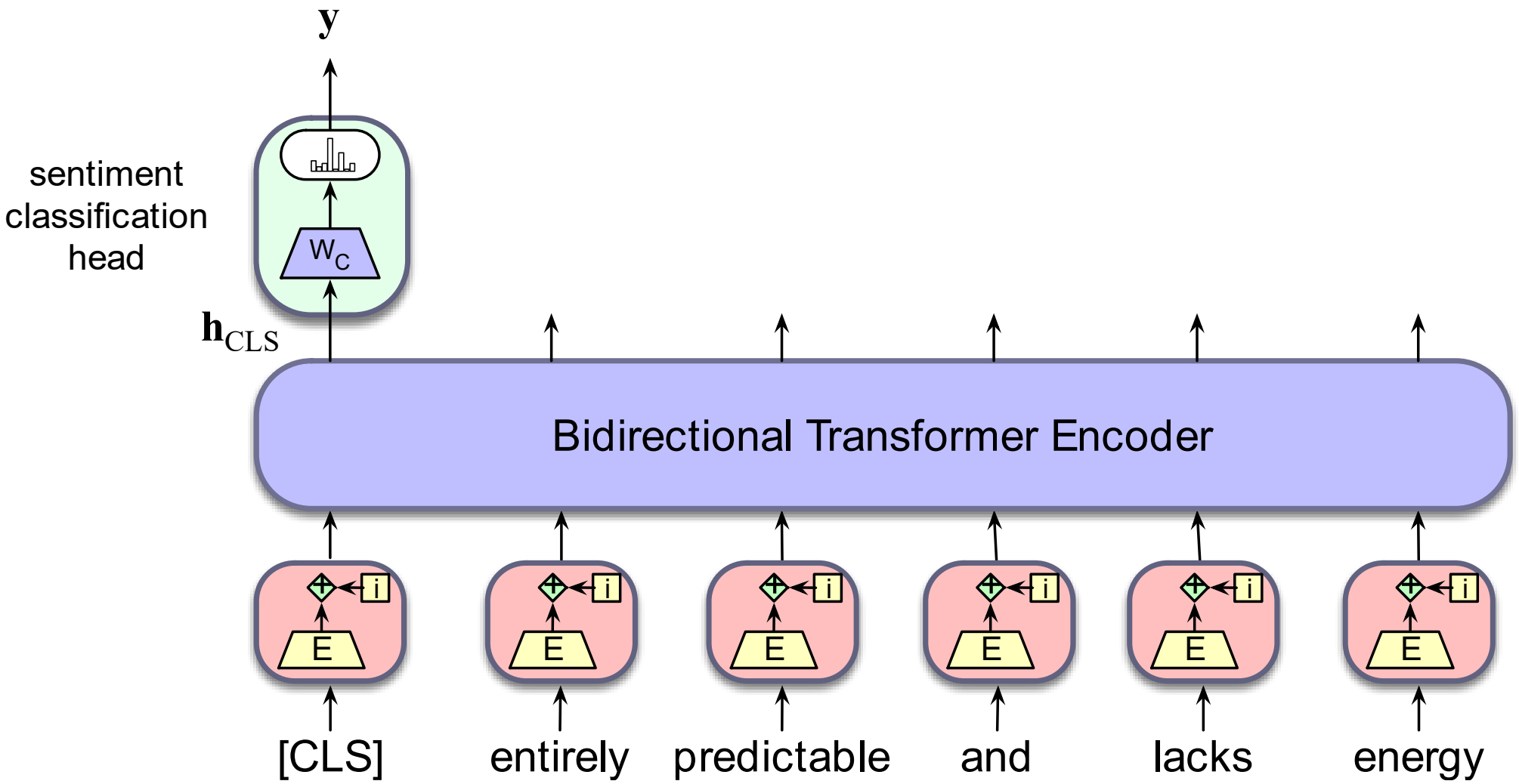
Similarity and contextual embeddings

- We generally use cosine as for static embeddings
- But some issues:
  - Contextual embeddings tend to be **anisotropic:** all point in roughly the same direction so have high inherent cosines (Ethayarajh 2019)
  - Cosine measure are dominated by a small number of "rogue" dimensions with very high values (Timkey and van Schijndel 2021)
  - Cosine tends to underestimate human judgments on similarity of word meaning for very frequent words (Zhou et al., 2022)

Fine-Tuning for Classification

- The power of pretrained language models lies in their ability to extract generalizations from large amounts of text—generalizations that are useful for myriad downstream applications.
- There are two ways to make practical use of the generalizations to solve downstream tasks:
  - The most common way is to use natural language to prompt the model, putting it in a state where it contextually generates what we want.
  - an alternative way to use pretrained language models for downstream applications: a version of the finetuning paradigm
- In the kind of finetuning used for masked language models, we add application specific circuitry (often called a special head) on top of pretrained models, taking their output as its input.
- The finetuning process consists of using labeled data about the application to train these additional application-specific parameters.
- Typically, this training will either freeze or make only minimal adjustments to the pretrained language model parameters.
- The following sections introduce finetuning methods for the most common kinds of applications:
  - sequence classification,
  - sentence-pair classification, and
  - sequence labeling.

Adding a sentiment classification head



Sequence-Pair classification

- Assign a label to pairs of sentences:
- paraphrase detection (are the two sentences paraphrases of each other?)
- logical entailment (does sentence A logically entail sentence B?)
- discourse coherence (how coherent is sentence B as a follow-on to sentence A?)

Example: Natural Language Inference

- Pairs of sentences are given one of 3 labels
  - `Neutral`
    - a: Jon walked back to the town to the smithy.
    - b: Jon traveled back to his hometown.
  - `Contradicts`
    - a: Tourist Information offices can be very helpful.
    - b: Tourist Information offices are never of any help.
  - `Entails`
    - a: I'm confused.
    - b: Not all of it is very clear to me.

- Algorithm: pass the premise/hypothesis pairs through a bidirectional encoder and use the output vector for the [CLS] token as the input to the classification head .

Fine-tuning for sequence labeling

- Assign a label from a small fixed set of labels to each token in the sequence.
  - Named entity recognition
  - Part of speech tagging

Named Entity Recognition

- A **named entity** is anything that can be referred to with a proper name: a person, a location, an organization
- **Named entity recognition (NER):** find spans of text that constitute proper names and tag the type of the entity

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	**Mt. Sanitas** is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states	**Palo Alto** is raising the fees for parking.

Named Entity Recognition

Citing high fuel prices, [ORG **United Airlines**] said [TIME **Friday**] it has increased fares by [MONEY **$6**] per round trip on flights to some cities also served by lower-cost carriers. [ORG **American Airlines**], a unit of [ORG **AMR Corp.**], immediately matched the move, spokesman [PER **Tim Wagner**] said. [ORG **United**], a unit of [ORG **UAL Corp.**], said the increase took effect [TIME **Thursday**] and applies to most routes where it competes against discount carriers, such as [LOC **Chicago**] to [LOC **Dallas**] and [LOC **Denver**] to [LOC **San Francisco**].

BIO Tagging

- A method that lets us turn a segmentation task (finding boundaries of entities) into a classification task

[PER **Jane Villanueva**] of [ORG **United**] , a unit of [ORG **United Airlines Holding**] , said the fare applies to the [LOC **Chicago**] route.

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

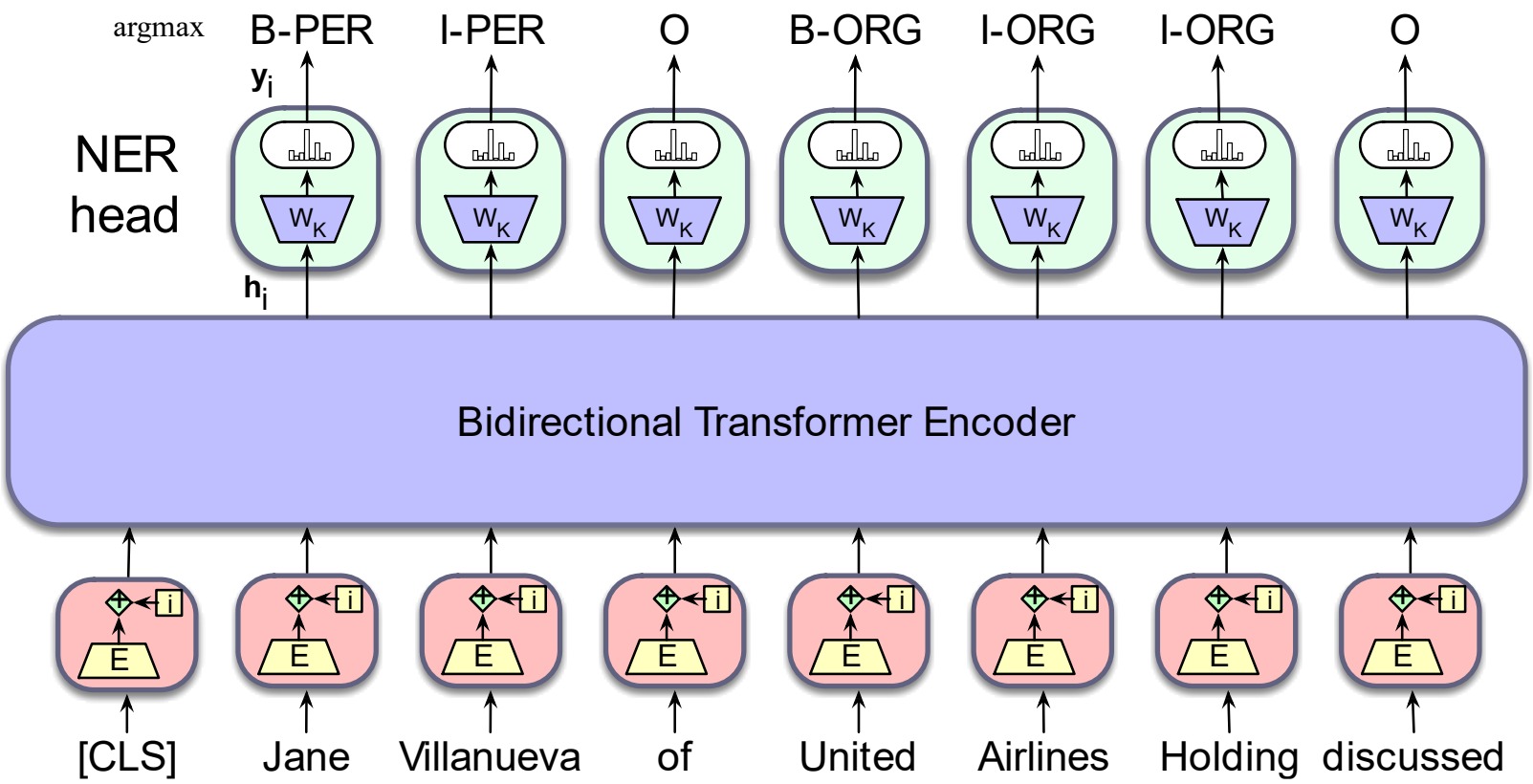
Ramshaw and Marcus (1995)

Tokens in text are often coded with the BIO scheme
– O : outside,
– B-XXX : first word in NE,
– I-XXX : all other words in NE

Sequence labeling

$$\mathbf{y_i} = \text{softmax}(\mathbf{h_i^L W_K})$$

$$\mathbf{t_i} = \text{argmax}_k(\mathbf{y}_i)$$



- We need to map between tokens (used by LLM) and words (used in definition of name entities.)
- We evaluate NER with F1 (precision/recall)

More details

- We need to map between tokens (used by LLM) and words (used in definition of name entities.)
- We evaluate NER with F1 (precision/recall)

Summary

- Bidirectional encoders can be used to generate contextualized representations of input embeddings using the entire input context.
- Pretrained language models based on bidirectional encoders can be learned using a masked language model objective where a model is trained to guess the missing information from an input.
- The vector output of each transformer block or component in a particular token column is a contextual embedding that represents some aspect of the meaning of a token in context.
- A word sense is a discrete representation of one aspect of the meaning of a word.
- Contextual embeddings offer a continuous high-dimensional model of meaning that is richer than fully discrete senses.

Summary

- The cosine between contextual embeddings can be used as one way to model the similarity between two words in context, although some transformations to the embeddings are required first.
- Pretrained language models can be finetuned for specific applications by adding lightweight classifier layers on top of the outputs of the pretrained model.
- These applications can include sequence classification tasks like
  - sentiment analysis,
  - sequence-pair classification tasks like natural language inference, or
  - sequence labeling tasks like named entity recognition.