CS310 Natural Language Processing
自然语言处理
Lecture 08 - Pretraining

Instructor: Yang Xu

主讲人：徐炀

xuyang@sustech.edu.cn

Overview

- Architecture
- Tokenization
- Pretraining
- Fine-Tuning
- Contextual Embedding and Word Sense

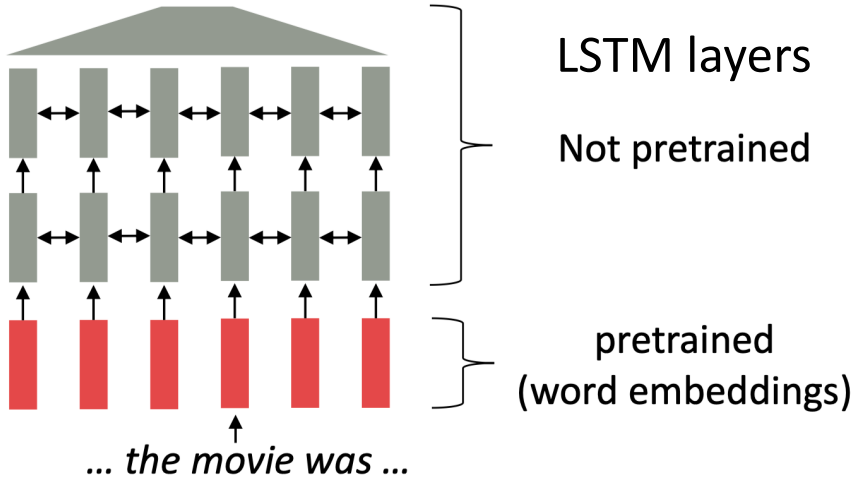
Overview

- **Architecture**
  - **Autoregressive decoder: GPT**
  - **Autoencoding: BERT**
  - **Encoder-decoder: T5**
  - **General language modeling (GLM)**
- Tokenization
- Pretraining
- Fine-Tuning
- Contextual Embedding and Word Sense

Background

- "pre-" means "before"
- Pretraining: train the model before applying it to a specific task
- Around 2017, state-of-the-art NLP = pretrained word embedding + LSTMs

Sentiment: $\hat{y}$

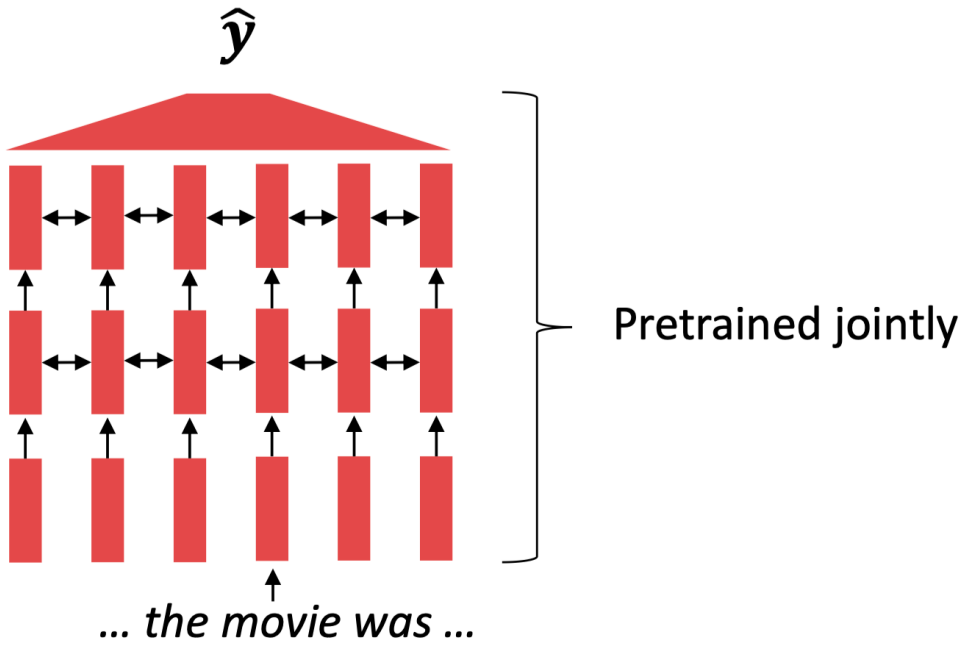


Issues:

- The training data must be sufficient to cover broad aspects of language
- Most params are randomly initialized
- Word representations are not contextualized:
  Ex. "movie" has fixed meaning no matter which sentence it appears in

figure from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Pretraining the whole models



Ex. “movie” has dynamic meaning decided by its context

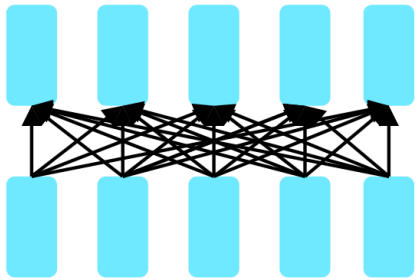
In modern NLP, especially since 2018, **all (or almost all) parameters** are initialized via pretraining

Pretraining:

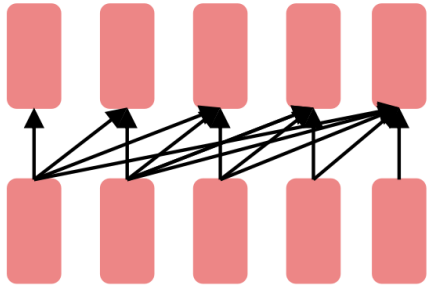
- Better contextualized representations of language
- “Warm up” model params with better initialization
- Make use of large text data

Architecture

- “Pretraining” can apply to all three types of neural architecture

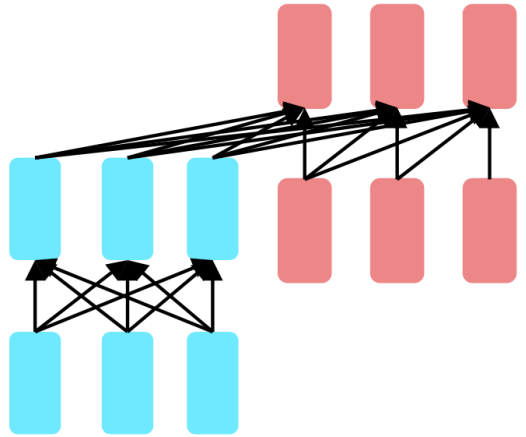


Encoders



Decoders

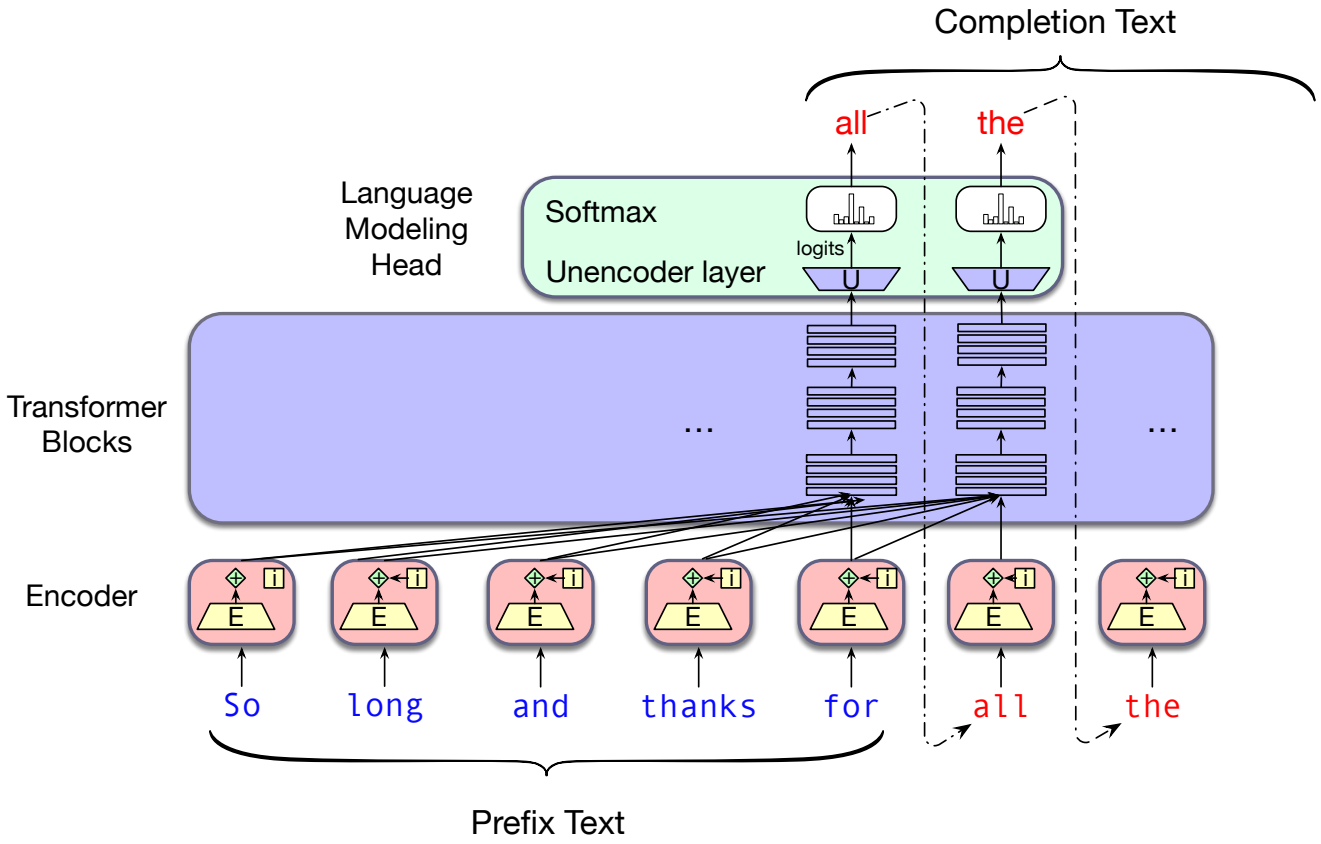
Primary cases



Encoder-decoders

Autoregressive

- Autoregressive = causal = left-to-right = decoder-only

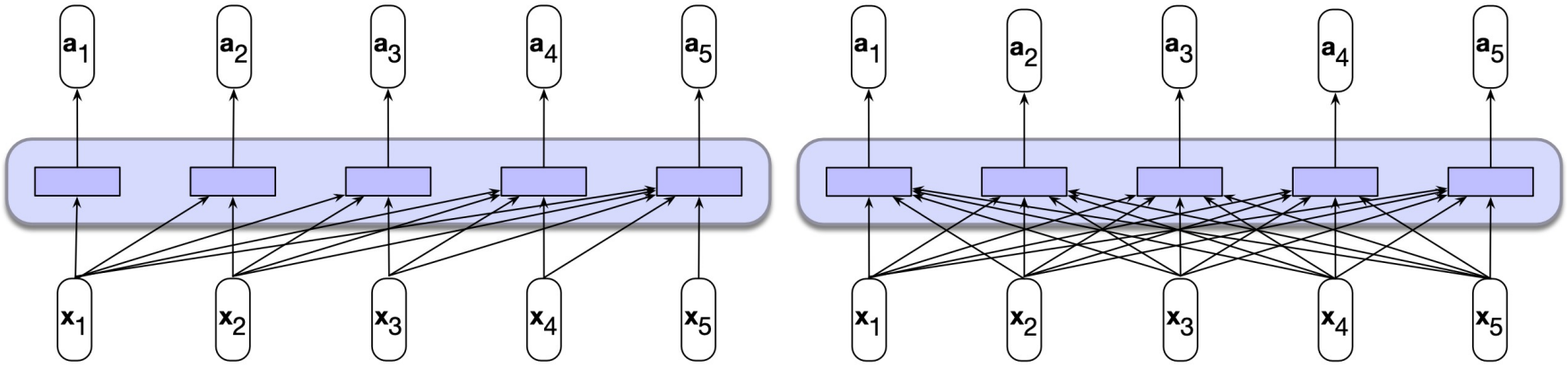


- The model can only see the preceding context
- Ignoring information from the right context

Encoder Model

- Bidirectional Encoder Representations from Transformers (**BERT**) (Devlin et al., 2018)
- Its variants: **RoBERTa** (Liu et al., 2019), **ALBERT** (Lan et al., 2019) etc.

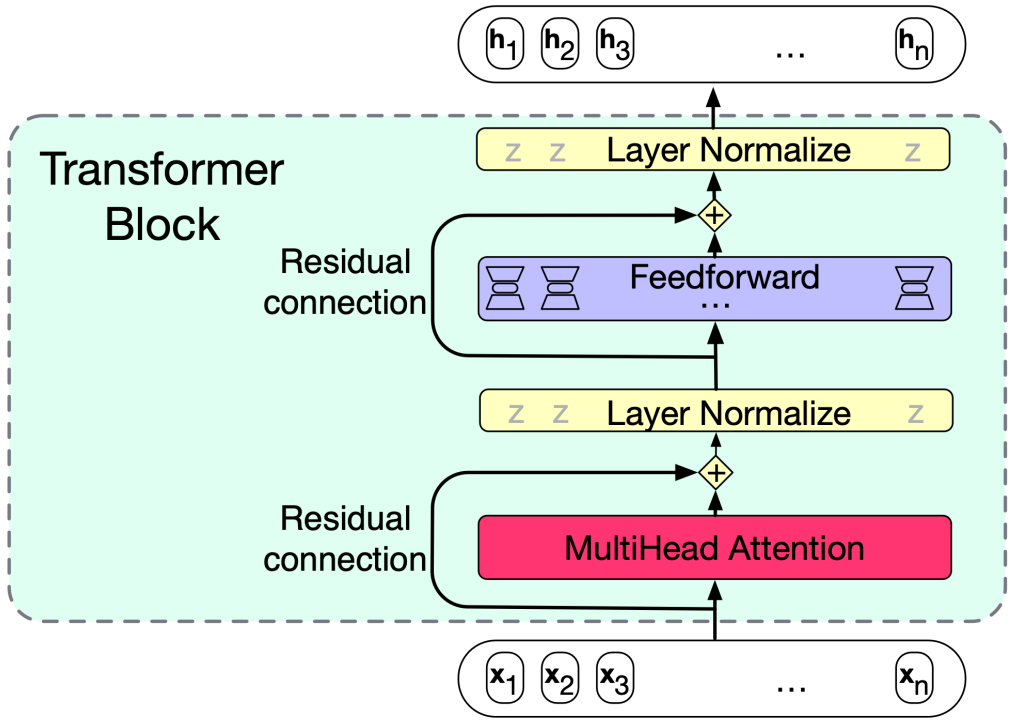
- Bidirectional encoders overcome this limitation by allowing self-attention to range over the entire input



a) A causal self-attention layer

b) A bidirectional self-attention layer

BERT Architecture



Uses the same self-attention as causal transformer

$$\boldsymbol{Q}_i = \boldsymbol{X}\boldsymbol{W}_i^Q$$
$$\boldsymbol{K}_i = \boldsymbol{X}\boldsymbol{W}_i^K$$
$$\boldsymbol{V}_i = \boldsymbol{X}\boldsymbol{W}_i^V$$

$$\mathbf{head}_i = \text{SelfAttention}(\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i)$$

$$\boldsymbol{A} = \text{MultiheadAttention}(\boldsymbol{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\boldsymbol{W}^O$$

$$\text{SelfAttention}(Q, K, V) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}$$

No future mask is used!

BERT: No Future Mask Used

q1•k1	q1•k2	q1•k3	q1•k4	q1•k5
q2•k1	q2•k2	q2•k3	q2•k4	q2•k5
q3•k1	q3•k2	q3•k3	q3•k4	q3•k5
q4•k1	q4•k2	q4•k3	q4•k4	q4•k5
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

N

The $QK^\top$ matrix contains **all pairs** of comparison between input queries and keys

allowing the model to contextualize each token using *information from the entire input*

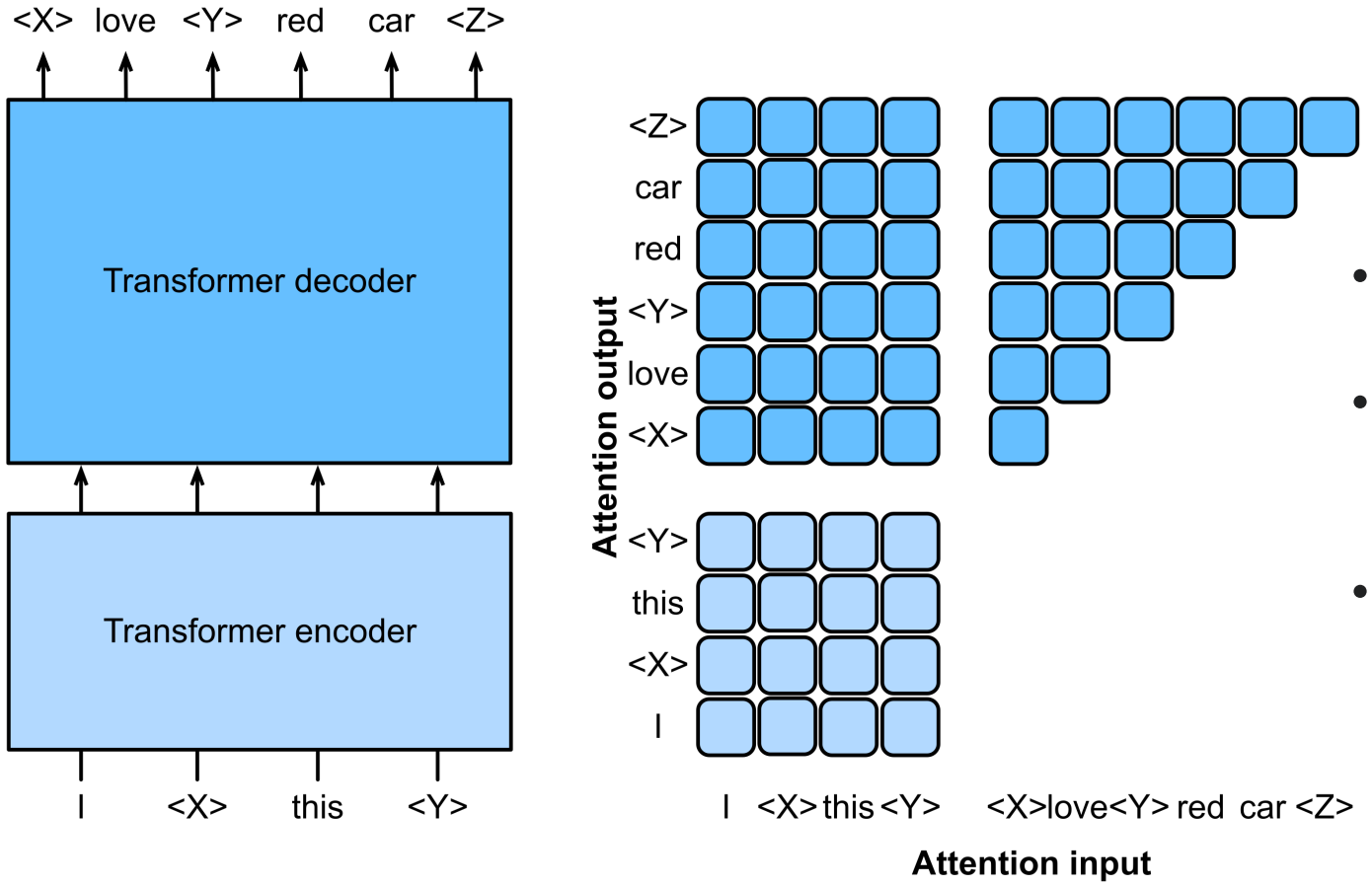
BERT: Hyper Parameters

- **BERT** (Devlin et al., 2018)
- Vocabulary size: 30,000 tokens (from WordPiece algorithm)
- Hidden layer size $d = 768$
- 12 layers of transformer blocks, with 12 multihead attention layers each
- Total params: about **100 M**

- **XLM-RoBERTa** (Liu et al., 2019)
- Vocabulary size: 250,000 tokens (from SentencePiece Unigram LM algorithm)
- Hidden layer size $d = 1024$; 24 layers of transformer blocks, with 16 multihead attention layers each
- Total params: about **550 M**

Encoder-Decoder Model

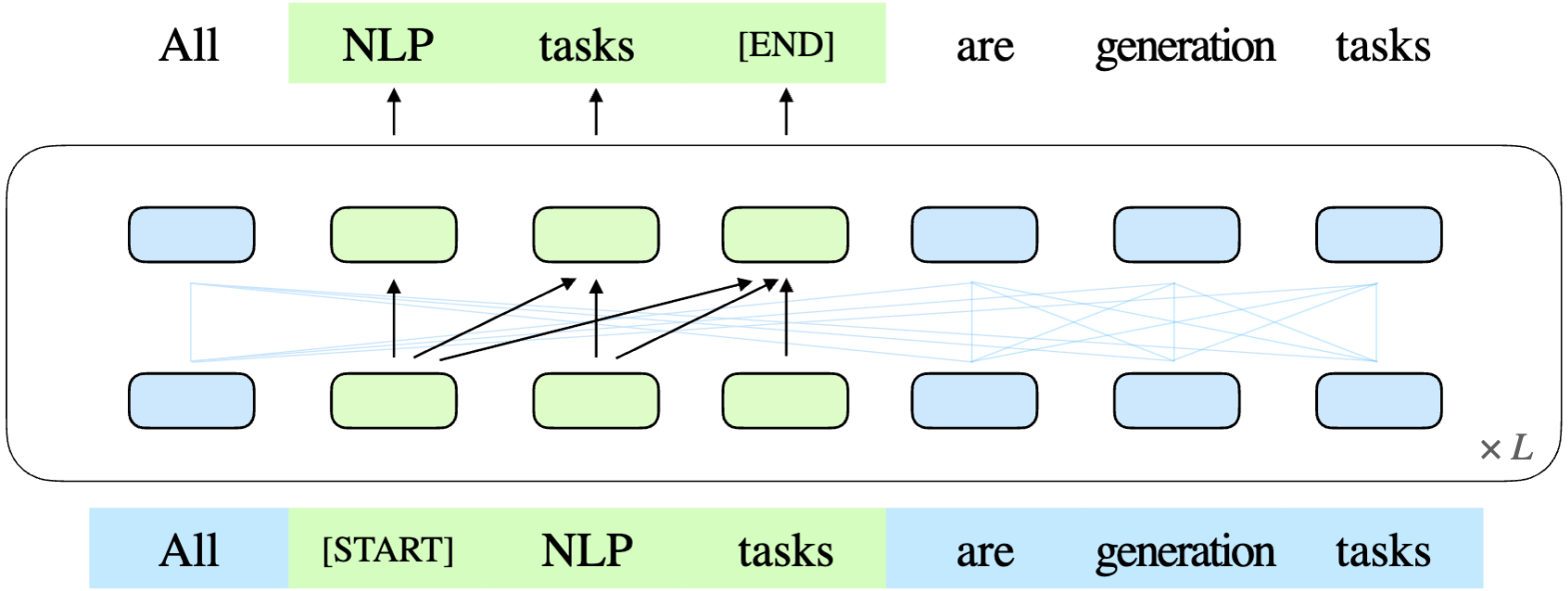
- T5 (Google, 2019)



- Encoder self-attention (lower square), all input tokens attend to each other;
- Encoder–decoder cross-attention (upper rectangle), each target token attends to all input tokens
- Decoder self-attention (upper triangle), each target token attends to present and past target tokens only

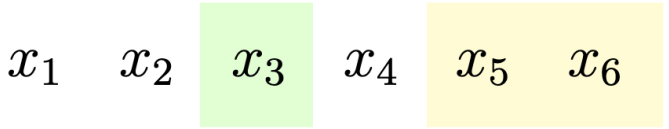
General language modeling (GLM) Du et al., 2022

- GLM: Randomly blank out spans of tokens and train the model to sequentially reconstruct the spans.



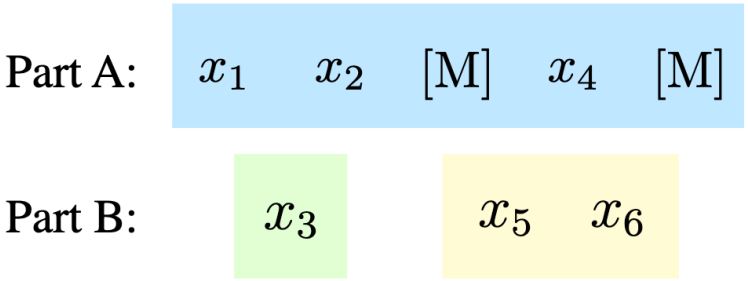
GLM

Du et al., 2022



(a) Sample spans from the input text

The original text is $[x_1, x_2, x_3, x_4, x_5, x_6]$.
Two spans $[x_3]$ and $[x_5, x_6]$ are sampled.

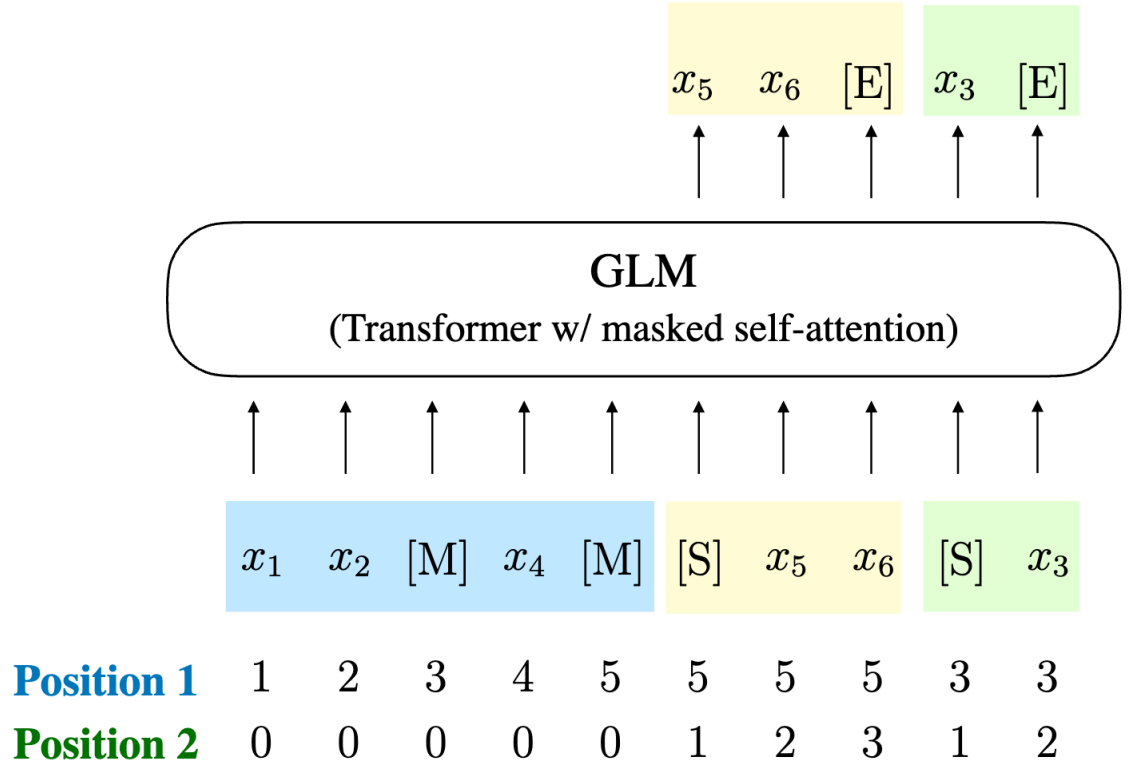


(b) Divide the input into Part A / Part B

Replace the sampled spans with [M] in Part A, and shuffle the spans in Part B

GLM

Du et al., 2022

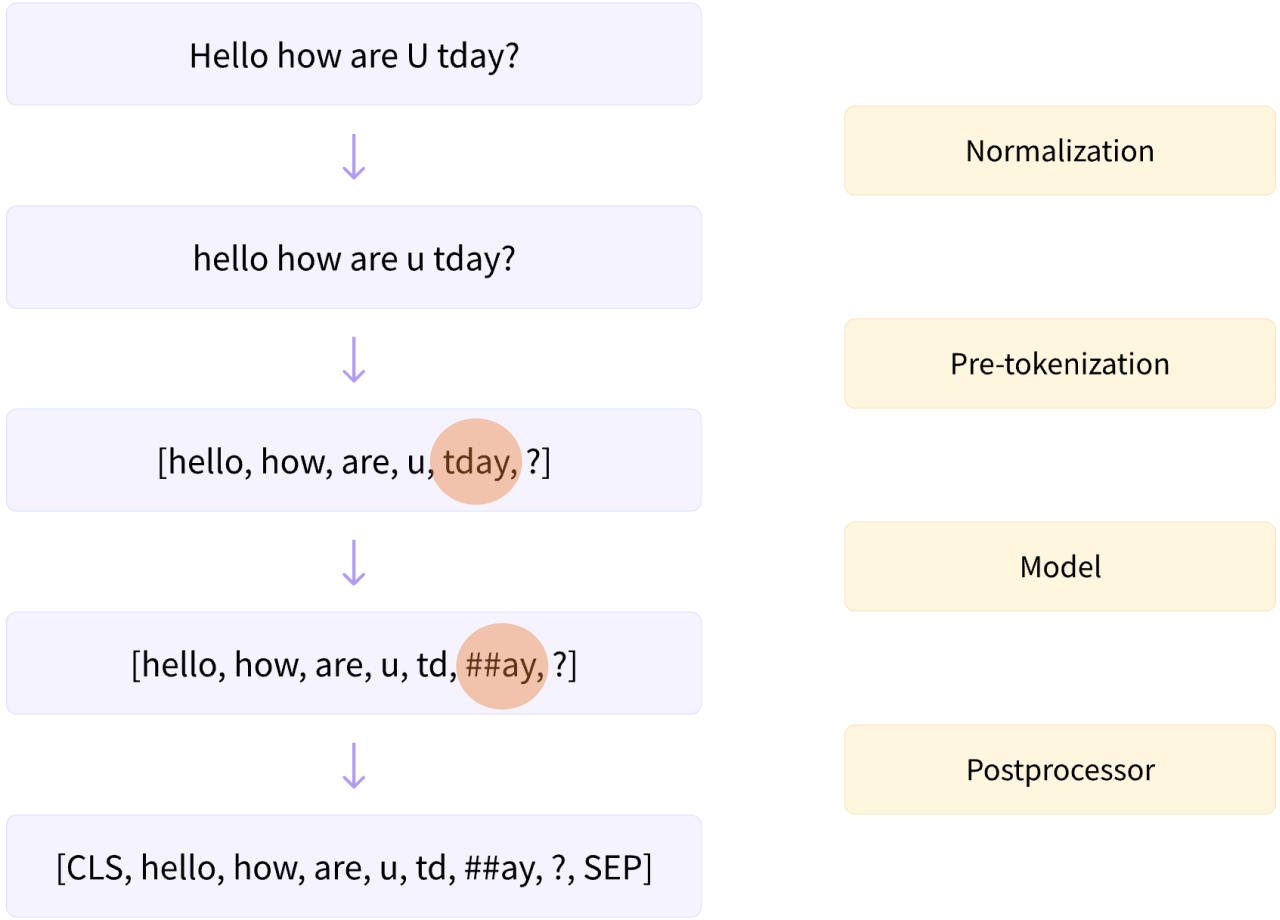


(c) Generate the Part B spans autoregressively

Overview

- Architecture
- **Tokenization**
- Pretraining
- Fine-Tuning
- Contextual Embedding and Word Sense

General tokenization workflow



Here “model” means a **tokenizer model**

Example: `tday => td, ##ay`

Figure source: https://huggingface.co/learn/llm-course/en/chapter6/4?fw=pt

Why subword is necessary

- Weakness of word-level tokenization:
- All *novel* words are mapped to UNK at testing time

	word		vocab mapping
Common words	hat	→	pizza (index)
	learn	→	tasty (index)
Variations	taaaaasty	→	UNK (index)
misspellings	laern	→	UNK (index)
novel items	Transformerify	→	UNK (index)

Subword-Level Tokenization

- Subword tokenization learns a vocabulary of subword tokens
- Split each word into a sequence of known subwords, for both training and testing

The hashtag “#” usage is from the WordPiece tokenization algorithm used in BERT

Byte-Pair Encoding (BPE) Tokenization

- A “bottom-up” algorithm: building the vocabulary by incrementally merging the most frequent pairs

Example corpus:

```
"hug", "pug", "pun", "bun", "hugs"
```

Initial vocabulary:

```
["b", "g", "h", "n", "p", "s", "u"]
```

with frequency:

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

Example credits to: https://huggingface.co/learn/llm-course/en/chapter6/5?fw=pt

BPE Tokenization cont.

- Splitting each word into characters

```
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

Look at **pairs**, and **merge** the most frequent one: `("u", "g") -> "ug"`

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

BPE Tokenization cont.

- Given the new corpus, merge the next most frequent pair `("u", "n") -> "un"`

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)
```

- Next `("h", "ug") -> "hug"`

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]
Corpus: ("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)
```

- …
- repeat until a desired vocabulary size is reached

Deal with Unicode characters

- GPT-2, RoBERTa, and recent Qwen, DeepSeek series use **Byte-level BPE**
- Process words not with Unicode characters, but with bytes
- Every character (Chinese, Japanese, Korean, emoji, …) will be included, but not end up being unknown tokens

Overview

- Architecture
- Tokenization
- **Pretraining Task**
  - **Token level: Masked Language Models**
  - **Sequence level: Next Sentence Prediction**
  - **Training Method**
- Fine-Tuning
- Contextual Embedding and Word Sense

Pretraining of Bidirectional Encoders

- A new training scheme other than next-word-prediction:
- **Fill-in-Blank task** -- **close task**
- Predict a missing item given the rest of the sentence

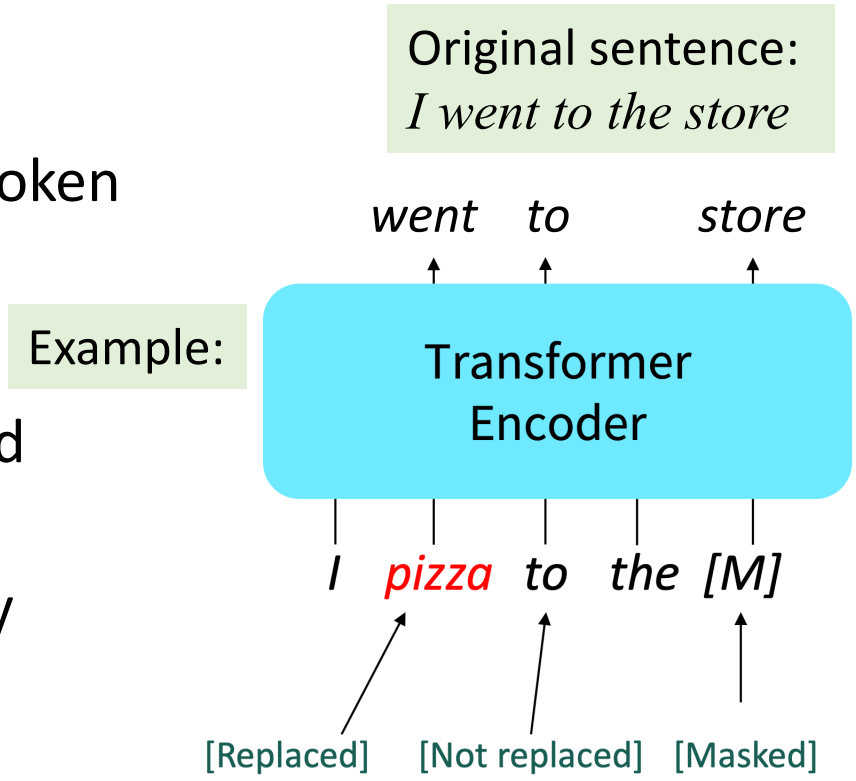
```
Please turn _____ homework in
```

- instead of predicting the next word

```
Please turn your homework _______
```

Masked Language Modeling

- The original approach used in BERT: Masked Language Modeling (MLM)
- **Randomly sample 15%** of the input tokens from each training sentence
- Among these sampled tokens:
- ⇒ 80% are replaced with a special token **[MASK]**
- ⇒ 10% are replaced with another randomly selected token
- ⇒ 10% are left unchanged
- Why? Prevent the model from getting complacent and not building strong representations of non-masked words
- All tokens play a role in self-attention process, but only the sampled tokens are used for learning



MLM Overview

$$\mathcal{L}_{\text{MLM}} = -\frac{1}{|M|}\sum_{i \in M} \log P(x_i|\boldsymbol{z}_i)$$

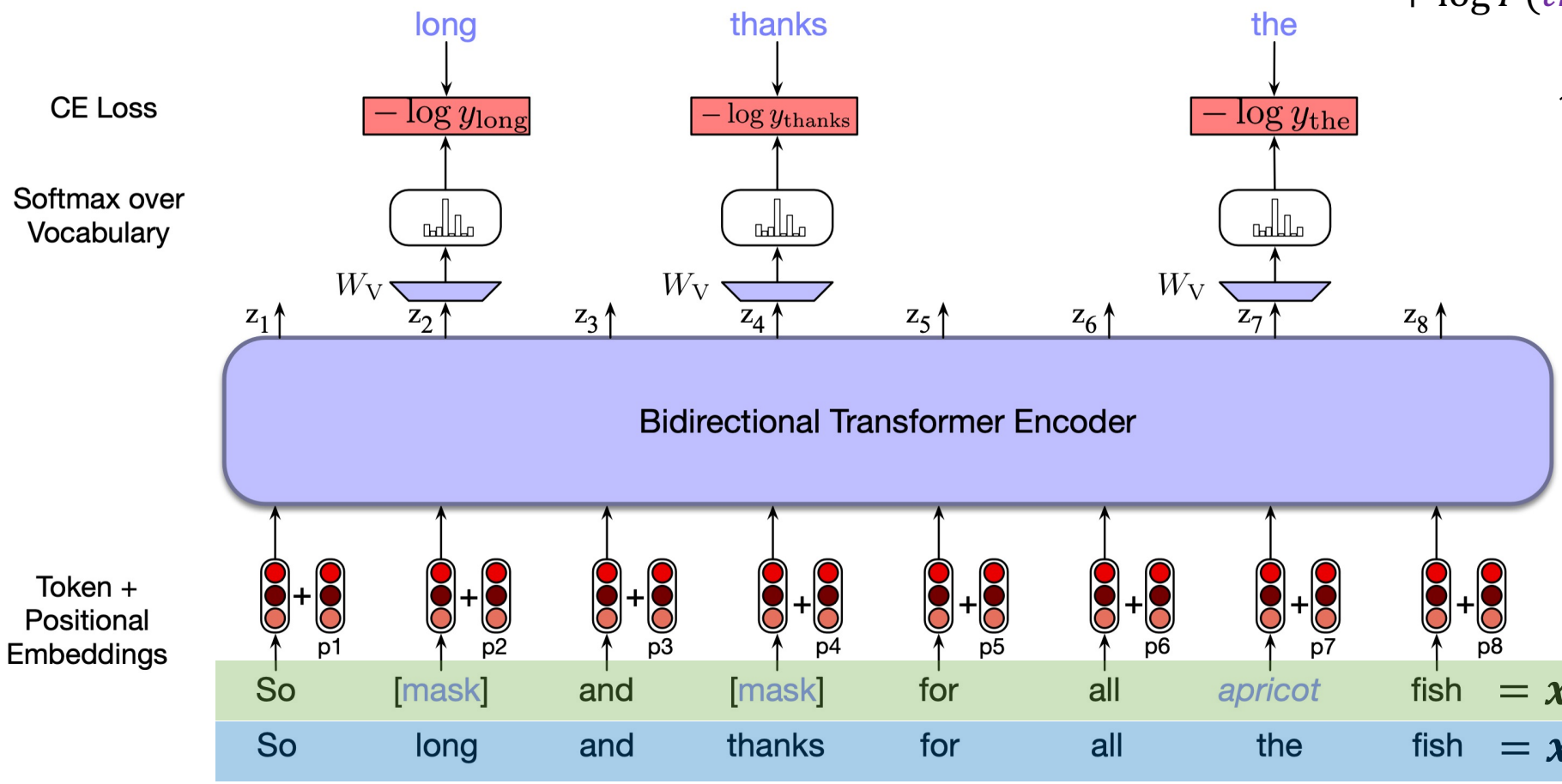
$$= -\frac{1}{3}(\log P(long|\boldsymbol{z}_2) + \log P(thanks|\boldsymbol{z}_4) + \log P(the|\boldsymbol{z}_7))$$

$$\mathcal{L}_{\text{MLM}}(x_i) = -\log P(x_i|\boldsymbol{z}_i)$$

$$\boldsymbol{y}_i = \text{softmax}(\boldsymbol{W}_V \boldsymbol{z}_i)$$

$$\boldsymbol{z}_i = \text{TransformerEnc}(\boldsymbol{x}^{\text{mask}})$$

$$\boldsymbol{z}_i \in \mathbb{R}^d$$



The masked tokens $M = \{long, thanks, the\}$ have been sampled:

long ⇒ [MASK]

thanks ⇒ [MASK]

the ⇒ *apricot*

MLM Notes

- Only 15% of the input data are used for training
- Because on the sampled tokens in M play a role in the loss $\mathcal{L}_{MLM} = -\frac{1}{|M|}\sum_{i \in M} \log P(x_i|\boldsymbol{z}_i)$
- Other tokens play no role
- Thus, training BERT with MLM is inefficient
- Some members of BERT family use all examples for training, e.g., ELECTRA (Clark et al., 2020)

Next Sentence Prediction

- Some NLP applications involves determining the relationship between pairs of sentences, such as:
- **Paraphrase (改述)** detection: if two sentences have similar meanings
- **Entailment (蕴含)**: detect if s_1 entails s_2, or contradict
- **Discourse coherence** (话语连贯性): detect if two neighboring sentences form a coherent discourse

Paraphrase

- s1: "这个苹果很甜。" (This apple is very sweet.)
- s2: "这个苹果味道很好。" (This apple tastes really good.)

Entailment

- Premise: "The sky is blue."
- Hypothesis: "The sky is a shade of azure."

In this example, the hypothesis logically follows from the premise, indicating entailment.

Discourse coherence

"我想要一杯咖啡。狗在睡觉。" (I want a cup of coffee. The dog is sleeping.)

This is an negative example

Next Sentence Prediction

- Presented with pairs of sentences, the model's task is to predict whether each pair is an **ACTUAL** pair of adjacent sentences from the training corpus, or a pair of randomly sampled unrelated sentences
- BERT uses 50% of sentences in true pairs, and the other 50% random pairs
- Two new tokens to facilitate training:
- **[CLS]**: prepended to input pair s_1, s_2 ("CLS" for "classification")
- **[SEP]**: placed between s_1 and s_2, and after s_2
- Finally, add extra embeddings to distinguish s_1 and s_2

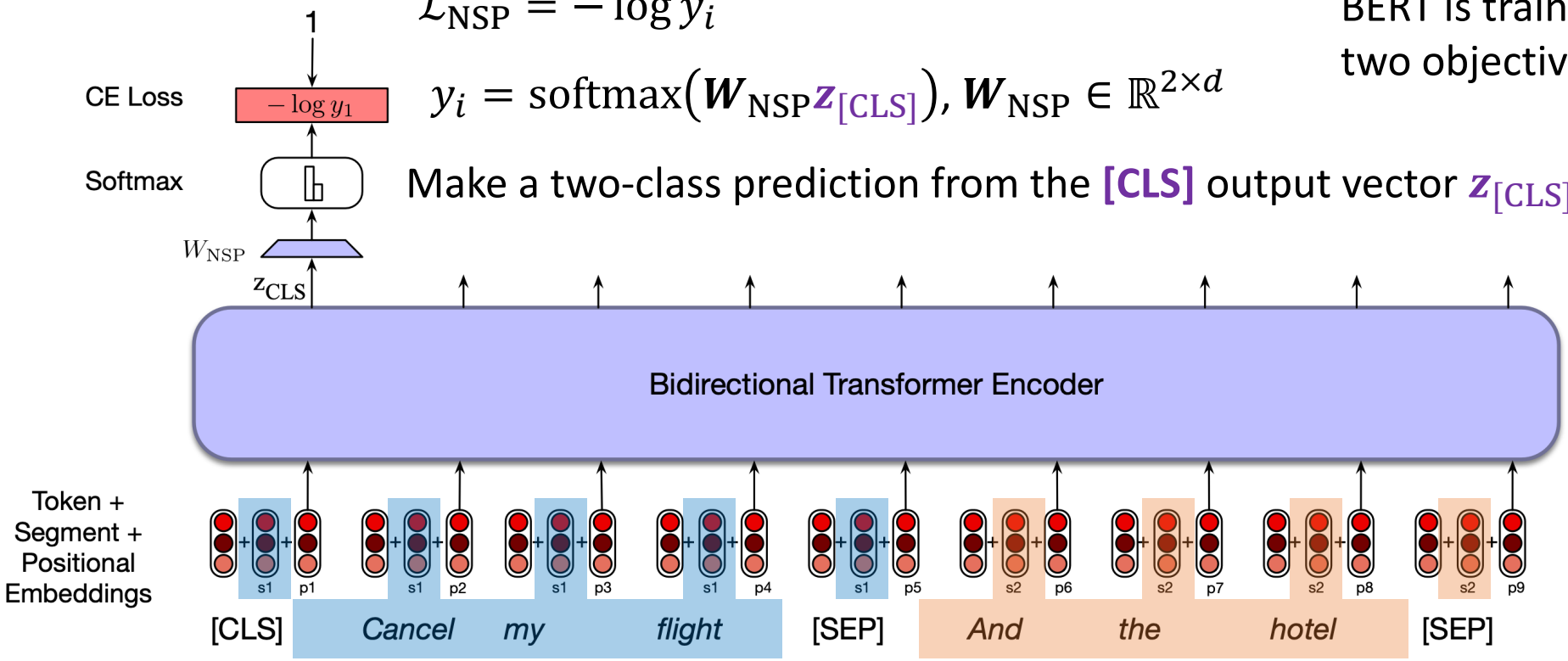
Next Sentence Prediction

$$\mathcal{L}_{\text{NSP}} = -\log y_i$$

$$y_i = \text{softmax}\left(\boldsymbol{W}_{\text{NSP}}\boldsymbol{z}_{[\text{CLS}]}\right), \boldsymbol{W}_{\text{NSP}} \in \mathbb{R}^{2\times d}$$

BERT is trained by combining the two objectives: $\mathcal{L}_{\text{MLM}} + \mathcal{L}_{\text{NSP}}$

Make a two-class prediction from the **[CLS]** output vector $\boldsymbol{z}_{[\text{CLS}]}$



Sentence 1 with extra embedding s_1

Sentence 2 with extra embedding s_2

Training Details of BERT

- Two models released (Devlin et al., 2018)
  - BERT-base: 12 layers (transformer blocks), 768-dim, 12 attention heads, 110 million params
  - BERT-large: 24 layers, 1024-dim, 16 attention heads, 340 million params
- Training data
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)
- Trained on 64 TPUs for 4 days
  - Pretraining is impractical on a single GPU
- Yet, fine-tuning is practical and common on a single GPU
  - “Pretrain once, fine-tune many times”

Overview

- Architecture
- Tokenization
- Pretraining
- **Fine-Tuning**
- Contextual Embedding and Word Sense

Fine-Tuning

- **Fine-tuning**: add a small set of application-specific parameters on top of pretrained models
- Use ***labeled data*** to train these application-specific parameters
- Either ***freeze*** or make only ***minimal*** adjustments to the pretrained parameters
- Common applications:
  - Sequence classification
  - Pair-Wise sequence classification
  - Sequence labeling
  - Span-based operations (more advanced)

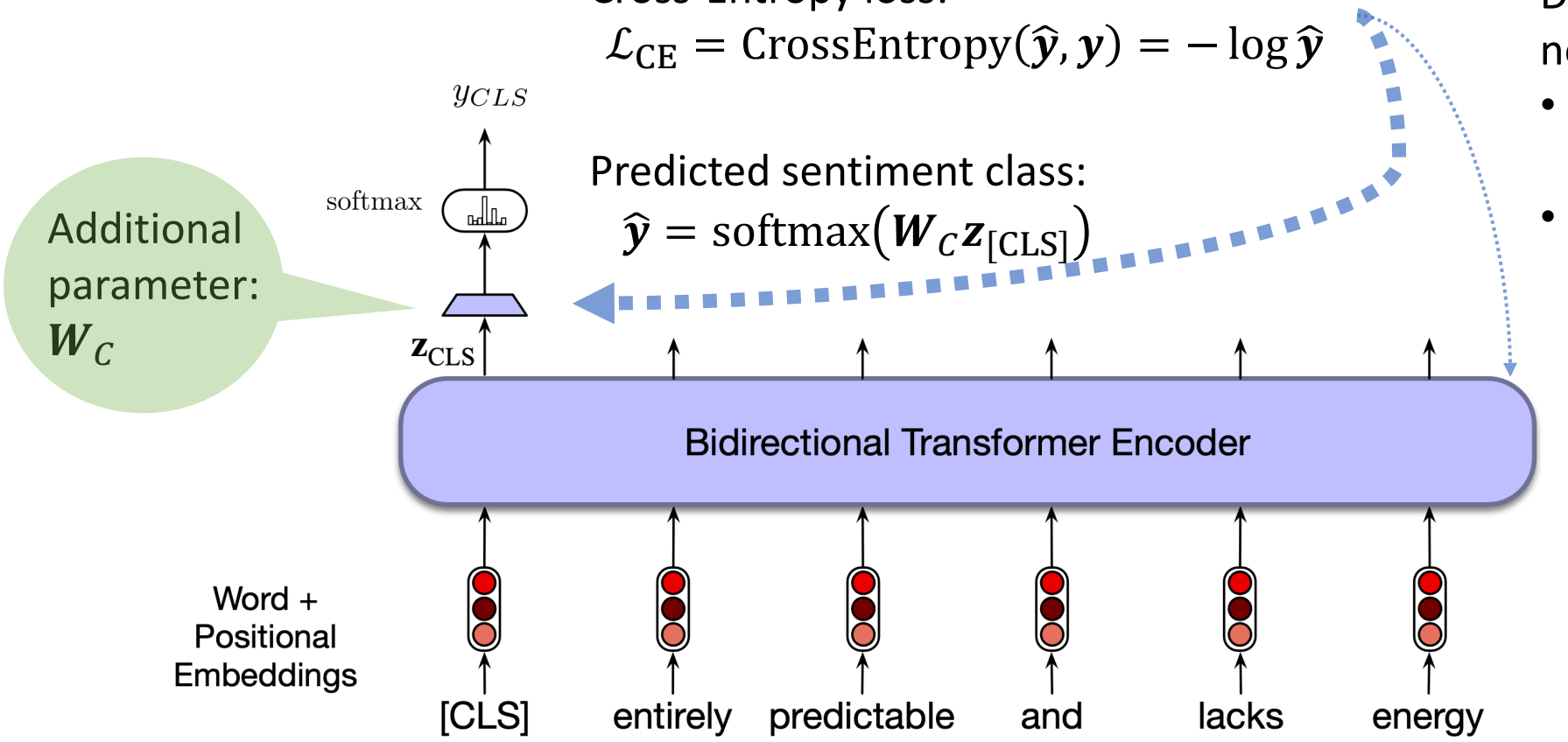
Sequence Classification

- An input sequence is represented with a single consolidated representation
- Recall RNN: use the final hidden state to stand for the entire sequence
- In transformer-based model: use an additional **vector** to represent the entire sequence ⇒ sometimes called **sentence embedding**

- In BERT, use the output vector of **[CLS] ,** $z_{[\mathrm{CLS}]}$**,** plays this role
  - [CLS] is added to vocabulary and prepended to all input sequences during pretraining
- $z_{[\mathrm{CLS}]}$ serves as input to a **classifier head** -- a network classifier to make relevant predictions

Sequence Classification

Example: Sentiment classification

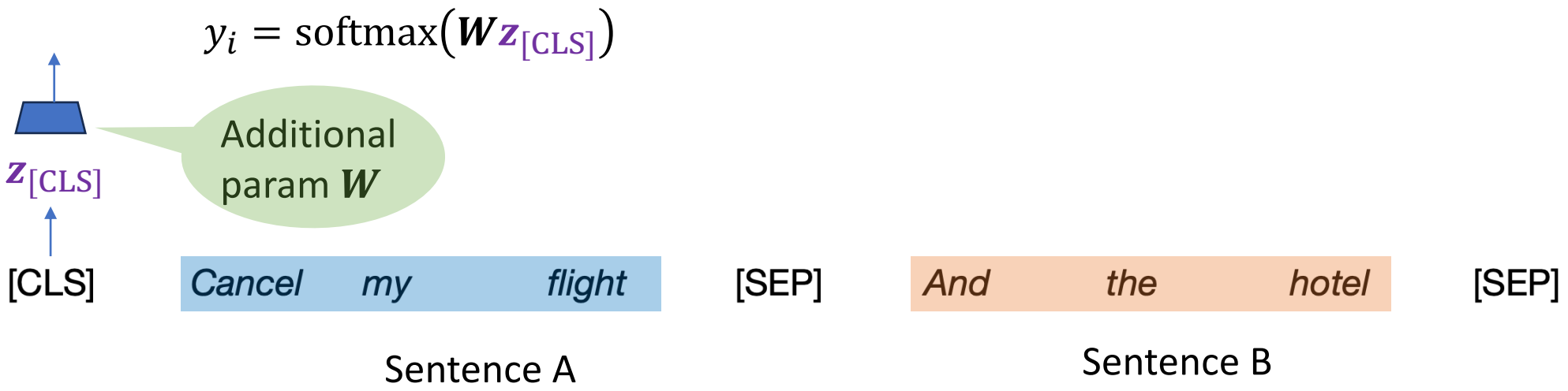


Difference from training a small neural classifier:

- Loss is mainly used to update param $\boldsymbol{W}_C$
- Minimal changes are made to the encoder params: limited to updates over the **final few transformer layers**

Pair-Wise Sequence Classification

- **Paraphrase detection**: are sentence A and B 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?
- Fine-tuning these tasks proceeds just as with the next sentence prediction (NSP) pretraining task



Example: entailment classification

- Dataset: Multi-Genre Natural Language Inference (MultiNLI)
- Pairs of sentences mapped to one of 3 labels: {entail, contradicts, neutral}

Entails
a: I'm confused.
b: Not all of it is very clear to me.

Neutral
a: Jon walked back to the town to the smithy.
b: Jon traveled back to his hometown.

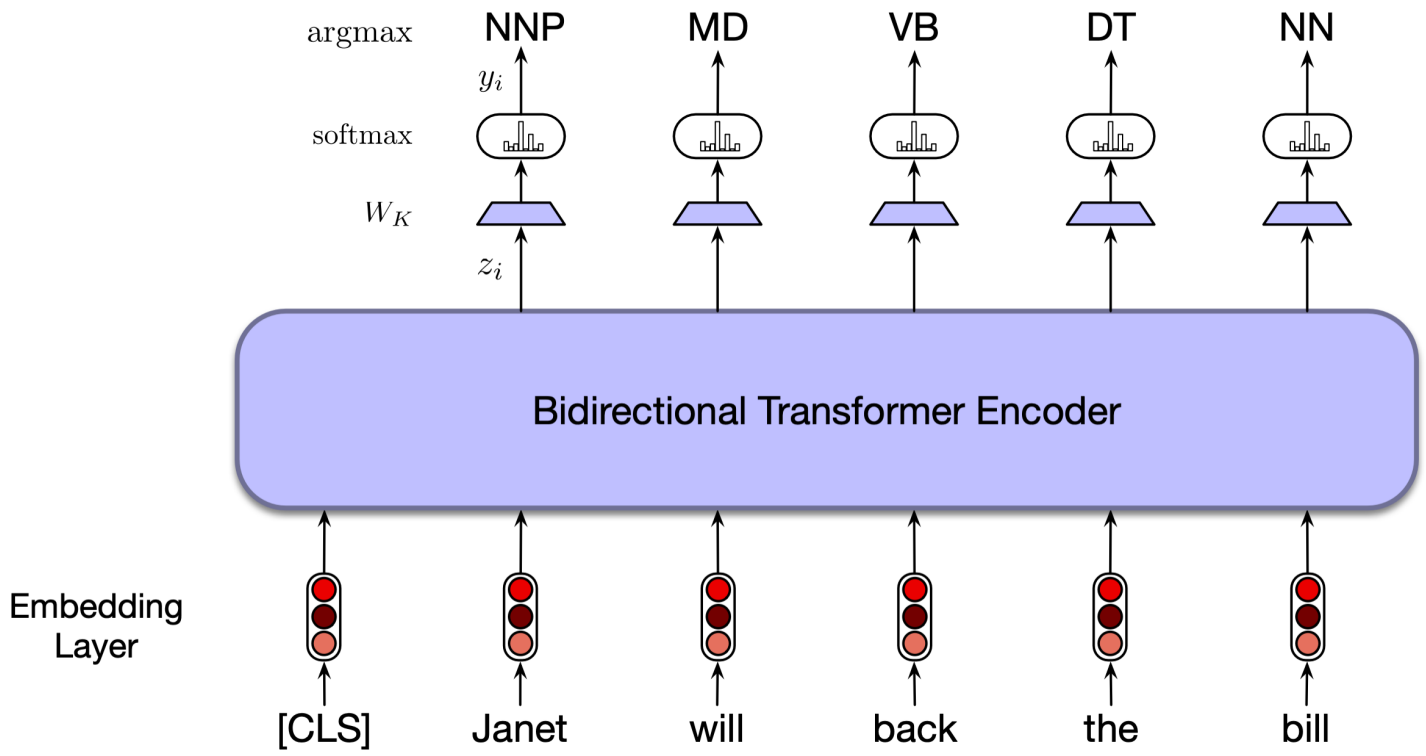
Contradicts
a: Tourist Information offices can be very helpful.
b: Tourist Information offices are never of any help.

$$y_i = \text{softmax}\left(\boldsymbol{W}\boldsymbol{z}_{[\text{CLS}]}\right) \in \mathbb{R}^3$$

(Williams et al., 2018)

Sequence Labeling

- The final output vector z_i corresponding to **each input token** x_i is passed to a classifier to produce a probability distribution over possible labels
- Example: POS tagging, BIO-based NER



Additional param: $W_K \in \mathbb{R}^{k \times d}$ for k possible tags

with greedy approach:

$\hat{y}_i = \text{softmax}(\boldsymbol{W}_K \boldsymbol{z}_i)$

$y_i = \arg\max_k \hat{y}_i$

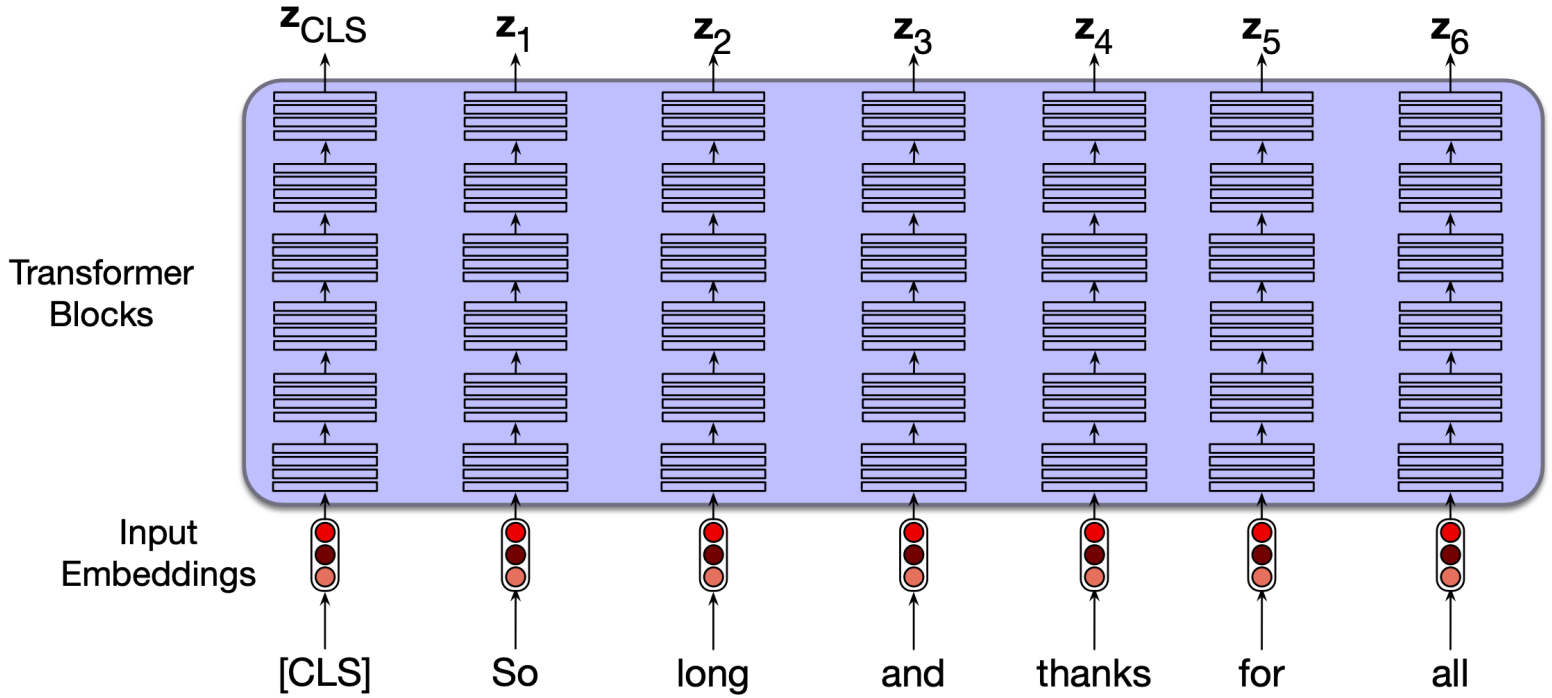
More advanced: pass $\hat{y}_i$ to a conditional random field (CRF) layer and use viterbi decoding

Overview

- Architecture
- Tokenization
- Pretraining
- Fine-Tuning
- **Contextual Embedding and Word Sense**

Contextual Embeddings

- The output of a BERT-style model is a contextual embedding vector z_i for each input token x_i
  - z_i represents some aspects of the meanings of x_i
  - Sometimes, instead of just using the final layer, we can average the z_i from the last four layers



Contextual Embeddings

- Static embedding (e.g., word2vec) ⇒ meaning of a word ***types***
  - a type is a static entry in the vocabulary
- Contextual embedding ⇒ meaning of word ***instances***
  - Instances of a particular word type in a particular context
- Thus, contextual embeddings are useful in linguistic tasks that require models of word meaning

  - "I would like some orange juice" ⇒ fruit
  - "Paint this part orange" ⇒ color

Word Sense

- Words are ambiguous: same word can be used to mean different things
- **Polysemous** (多义词), Geek "many senses"
- A **word sense** is a discrete representation of *one meaning* of a word

> **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 ...

- The senses can be visualized geometrically by contextual embeddings

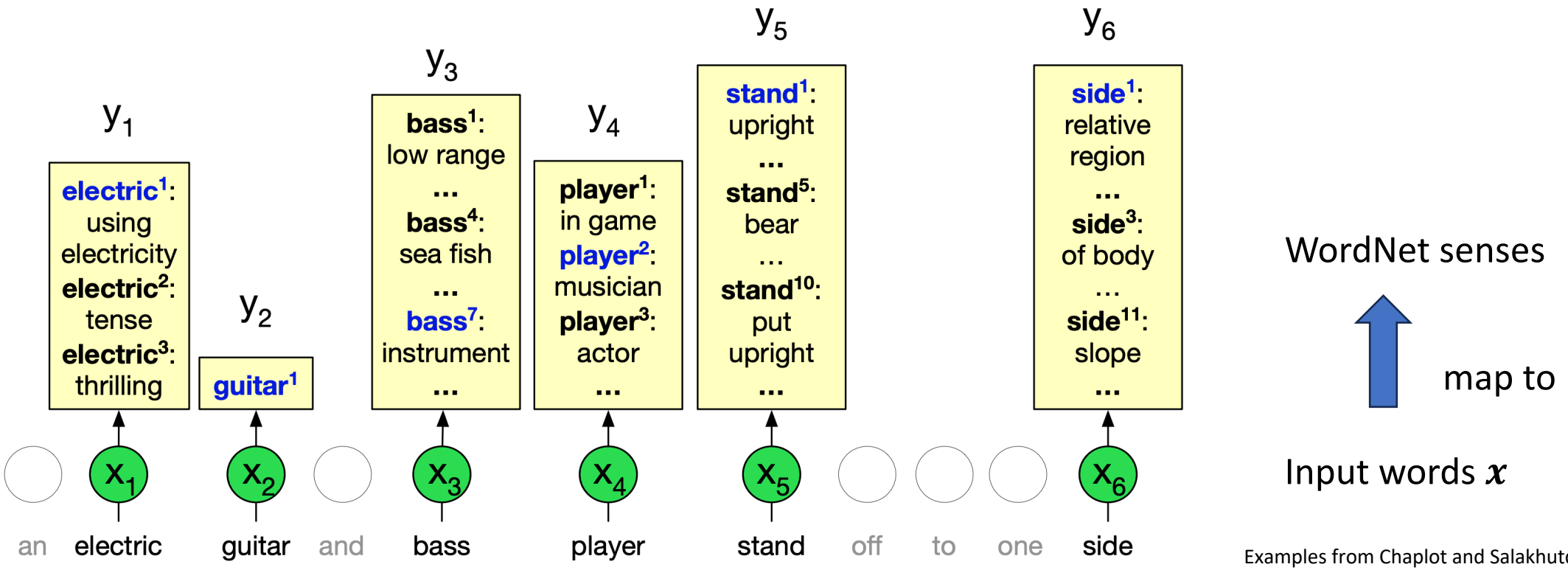
Visualize Word Senses

Figure from Coenen et al. (2019)

- Dictionary and thesauruses like WordNet give **discrete** lists of senses
- Embeddings (whether contextual or static) provides a **continuous high-dimensional** view of meaning

Word Sense Disambiguation

- **Word Sense Disambiguation**: the task of selecting the correct sense for a word
- Takes as input a word in context and a fixed inventory of potential word senses (like the ones in WordNet) and outputs the correct sense in context



Examples from Chaplot and Salakhutdinov, 2018

WSD Algorithm: 1-nearest-neighbor

- At training time: pass a sense-labeled dataset through any contextual embedding (e.g., BERT) ⇒ vector v_i for each token i
- For each sense s of any word, and for each of the n tokens of that sense, produce a **sense embedding** $\boldsymbol{v}_s$ by averaging the n contextual embeddings:

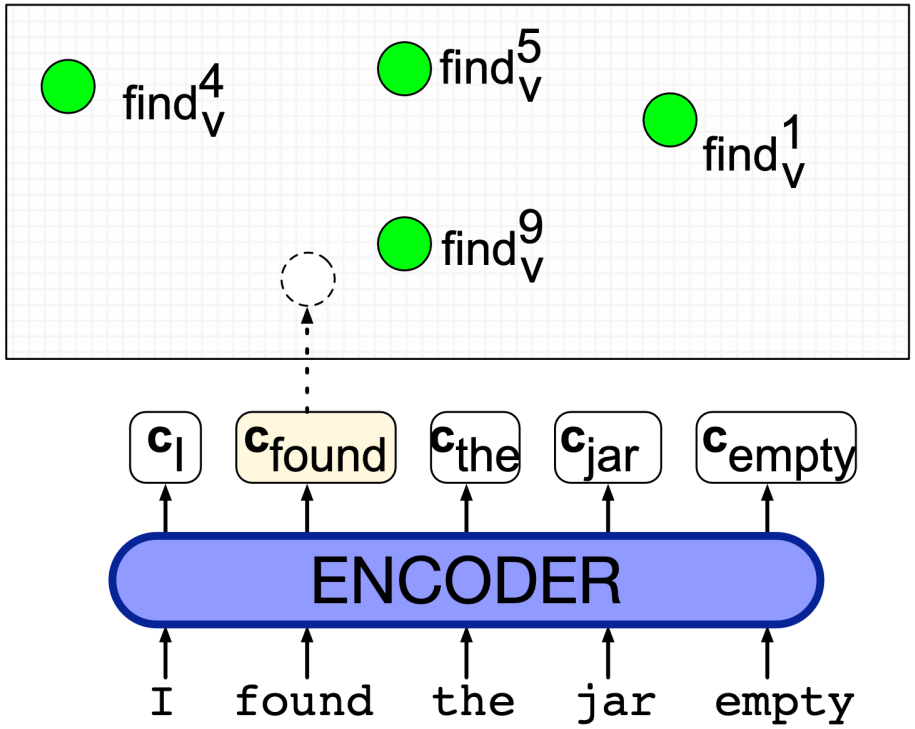
$$\boldsymbol{v}_s = \frac{1}{n}\sum_i v_i \qquad \forall \boldsymbol{v}_i \in \text{tokens}(s)$$

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

$$\text{sense}(t) = \arg\max_{s \in \text{senses}(t)} \cos(\boldsymbol{t}, \boldsymbol{v}_s)$$

WSD 1-nearest-neighbor Example

- Contextual embedding for *found* c_{found} is computed, and the nearest neighbor sense $\mathbf{find}_v^9$ is chosen



Examples from Loureiro and Jorge, 2019

Word Similarity is Tricky

- **Fact**: Contextual embeddings for all words are *extremely similar*
- **Fact**: BERT embeddings of any two randomly chosen words will have extremely high cosines $\approx 1 \Rightarrow$ All word vectors tend to point in the same direction
- A property known as **anisotropy** (各向异性)
- Anisotropy $\triangleq$ the expected cosine similarity of any pair of words in a corpus (Ethayarajh, 2019)
- If all vectors are uniformly distributed, then the expected cosine should be 0, which we call **isotropy** (各向同性)
- Cause of anisotropy: cosine measures are dominated by a small number of rogue dimensions that have very large magnitudes and high variance (Timkey and van Schijndel, 2021)

Solution to Anisotropy

- Standardizing (z-scoring) the vectors, i.e., subtracting the mean and dividing by variance

$$\mu = \frac{1}{|C|}\sum_{\boldsymbol{x}\in C}\boldsymbol{x} \qquad \sigma = \sqrt{\frac{1}{|C|}\sum_{\boldsymbol{x}\in C}(\boldsymbol{x}-\mu)^2} \qquad z = \frac{x-\mu}{\sigma}$$

- Remaining problem: cosine tends to underestimate similarity of word meanings for very frequent words (according to human judgements) (Zhou et al., 2022)

Reference

- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
- Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.
- Ethayarajh, K. (2019). How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. *arXiv preprint arXiv:1909.00512*.
- Timkey, W., & Van Schijndel, M. (2021). All bark and no bite: Rogue dimensions in transformer language models obscure representational quality. *arXiv preprint arXiv:2109.04404*.
- Zhou, K., Ethayarajh, K., Card, D., & Jurafsky, D. (2022). Problems with cosine as a measure of embedding similarity for high frequency words. *arXiv preprint arXiv:2205.05092*.