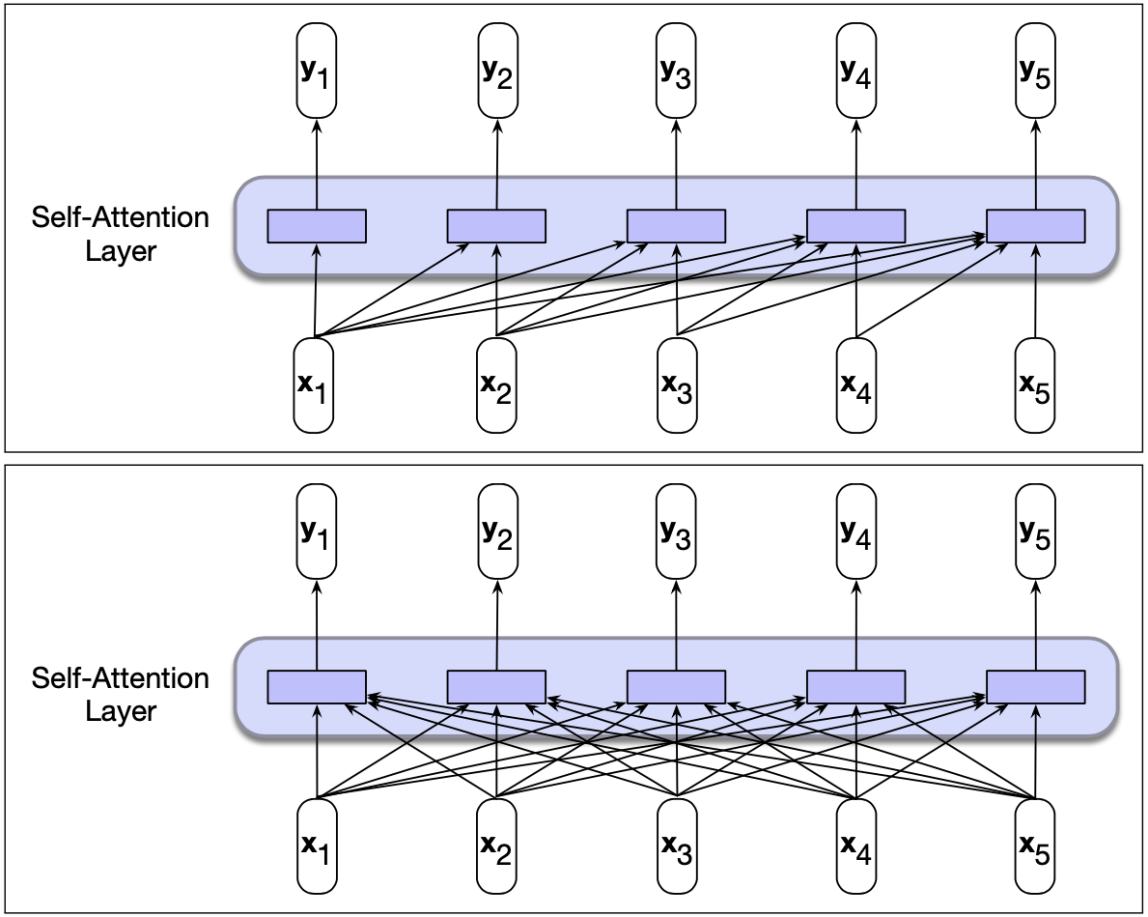
Leading Ideas

- Two paradigms for pre-training large language models:
  - Causal, backward looking transformer model (see chapter 10)
  - Bidirectional transformer encoder, trained via masked language modelling (this chapter)
- Mono-lingual transformer models: Bidirectional Encoder Representations from Transformers (BERT)
- Multi-lingual transformer models: XLM-RoBERTa (XLM-R)
- Transfer learning: Fine-tuning a pre-trained base model for particular tasks
- Contextual embeddings as representations in context: representing words by a different vector each time it appears in a different context.

Left-to-right and Bidirectional Transformer Encoders



Bidirectional Transformer Encoders

$$\mathbf{q_i} = \mathbf{W^Q x_i}; \quad \mathbf{k_i} = \mathbf{W^K x_i}; \quad \mathbf{v_i} = \mathbf{W^V x_i} \tag{11.1}$$

$$\mathbf{y_i} = \sum_{j=i}^{n} \alpha_{ij} \mathbf{v_j} \tag{11.2}$$

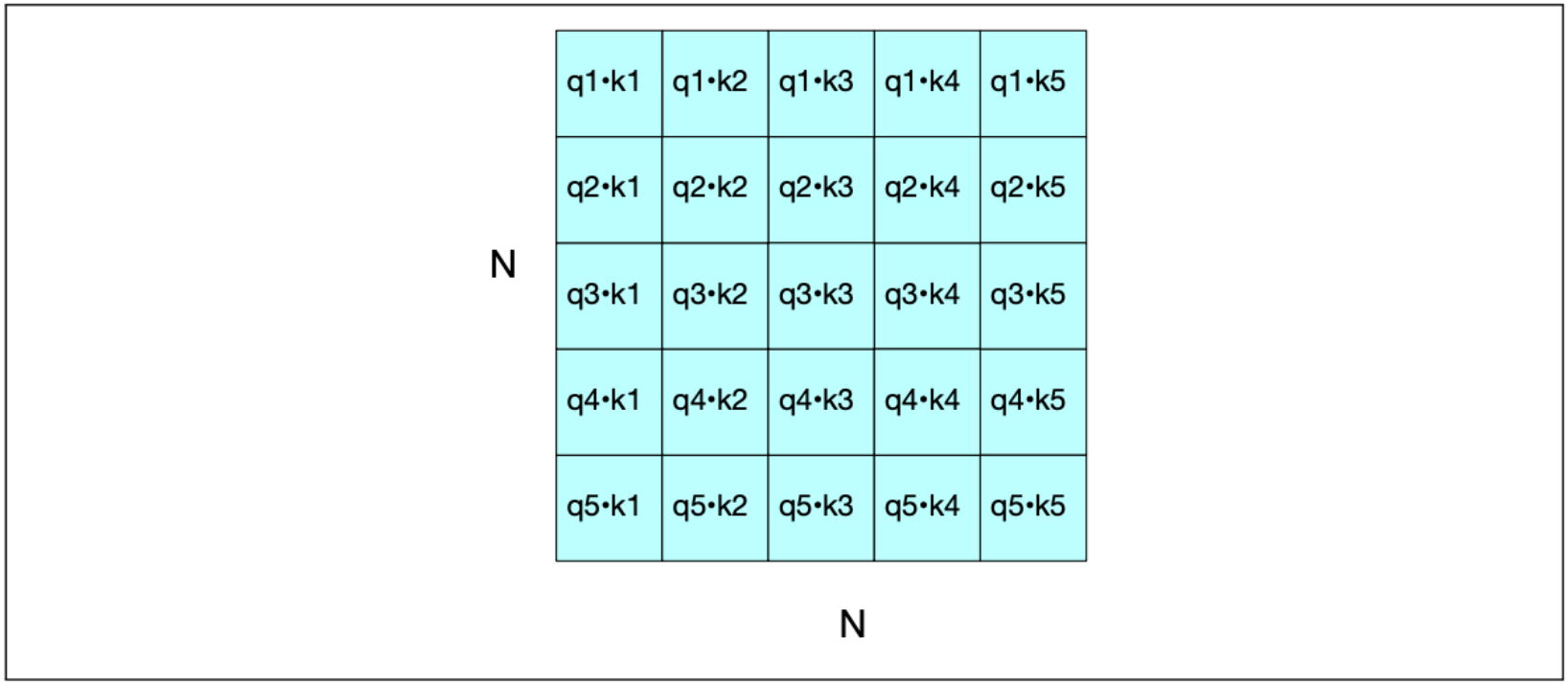
$$\alpha_{ij} = \frac{exp(score_{ij})}{\sum_{k=1}^{n} exp(score_{ik})} \tag{11.3}$$

$$score_{ij} = \mathbf{q_i} \cdot \mathbf{k_j} \tag{11.4}$$

Bidirectional Transformer Encoders

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \quad \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \quad \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}} \tag{11.5}$$

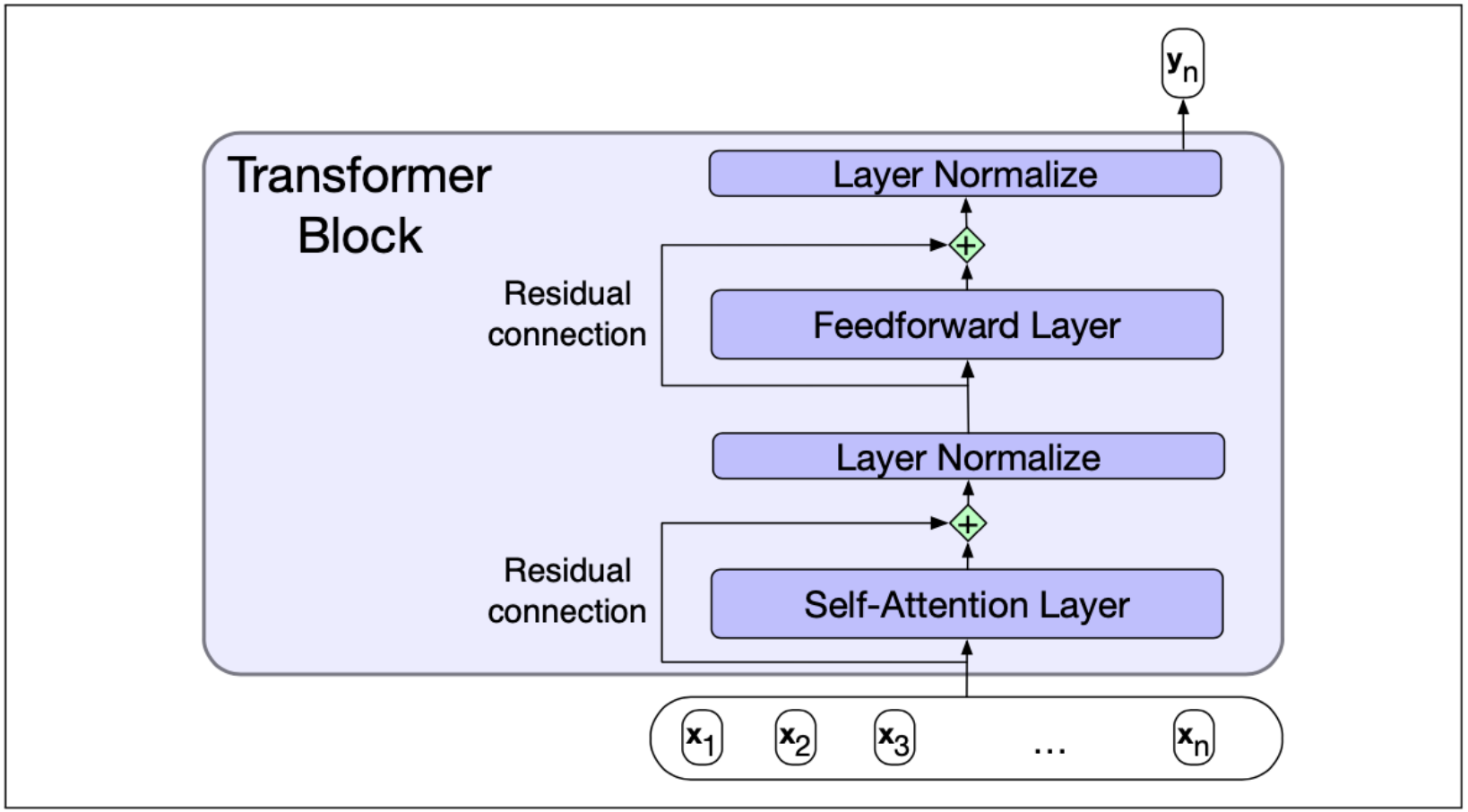
Bidirectional Transformer Encoders



Self-Attention

$$\textit{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{11.6}$$

Bidirectional Transformer Encoders



Bidirectional Encoder Representations from Transformers Model, BERT (Devlin et al., 2019)

To make this more concrete, the original bidirectional transformer encoder model, BERT (Devlin et al., 2019), consisted of the following:

- A subword English-only vocabulary consisting of 30,000 tokens generated using the Word-Piece algorithm (Schuster and Nakajima, 2012),
- Hidden layers of size of 768,
- 12 layers of transformer blocks, with 12 multihead attention layers each.
- The result is a model with over 100M parameters.

Training Bidirectional Encoders

- Cloze Task: Instead of predicting which words are likely to come next in this example, predict the missing elements, given the rest of the sentence:
  - Please turn your homework ____.
  - Please turn ____ homework in.

Masked Language Modeling (MLM)

- MLM uses unannotated text from a large corpus.
- The model is presented with a series of sentences from the training corpus where a random sample of tokens from each training sequence is selected for use in the learning task.
- Once chosen, a token is used in one of three ways:
  - It is replaced with the unique vocabulary token [MASK].
  - It is replaced with another token from the vocabulary, randomly sampled based on token unigram probabilities.
  - It is left unchanged.

MLM in Training BERT

- In BERT, 15 % of the input tokens in a training sequence are sampled for learning.
  - Of these, 80% are replaced with [MASK]
  - 10% are replaced with randomly selected tokens,
  - the remaining 10% are left unchanged.
- To produce a probability distribution over the vocabulary for each of the masked tokens, the output vector $\mathbf{z}_i$ from the final transformer layer for each masked token i is multiplied by a learned set of classification weights $\mathbf{W}_V \in \mathbb{R}^{|V| x d_h}$ and then through a softmax to yield the required predictions over the vocabulary: $\mathbf{y}_i = \text{softmax}(\mathbf{W}_V \mathbf{z}_i)$

More formally ...

For a given vector $\mathbf{x}$ of input tokens in a training sentence or a batch of training examples, let the set of tokens that are masked be M, the version of that sentence or batch with some tokens replaced by masks be $\mathbf{x}^{mask}$ and the sequence of output vectors be $\mathbf{z}$.

For a given token x_i, the loss is the probability of the correct input word, given $\mathbf{x}^{mask}$ (as identified in the output vector by $\mathbf{z_i}$) is:

$$L_{MLM}(x_i) = -\log P(x_i \mid \mathbf{z}_i) \tag{1}$$

and the average loss for $\mathbf{x}^{mask}$ is:

$$L_{MLM} = -\frac{1}{|M|} \sum_{i \in M} \log P(x_i \mid \mathbf{z}_i) \tag{2}$$

Calculating CE Loss for Masking Words

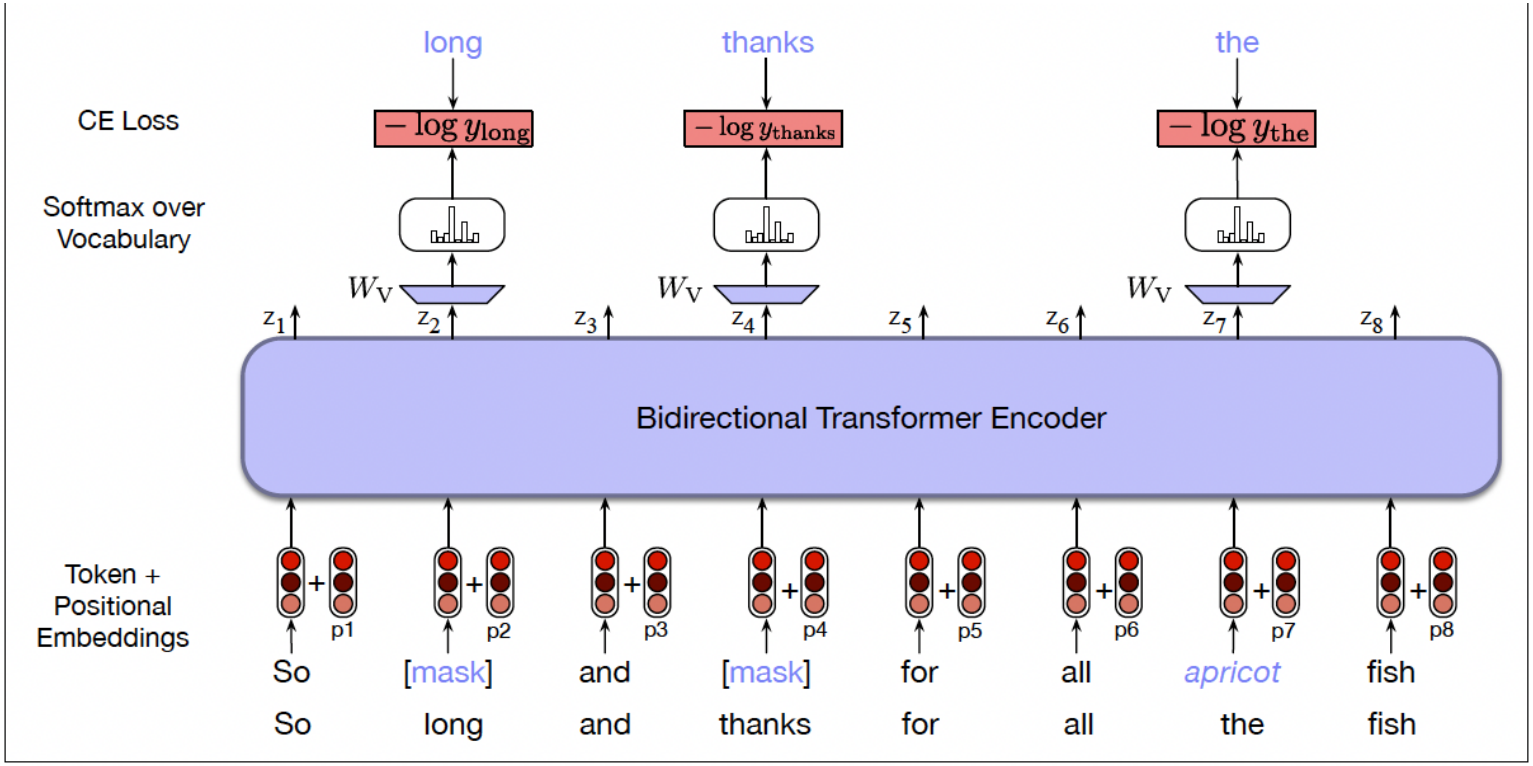


Figure 11.4 Masked language model training. In this example, three of the input tokens are selected, two of which are masked and the third is replaced with an unrelated word. The probabilities assigned by the model to these three items are used as the training loss. The other 5 words don't play a role in training loss. (In this and subsequent figures we display the input as words rather than subword tokens; the reader should keep in mind that BERT and similar models actually use subword tokens instead.)

$$y_i = \text{softmax}\,(W_V z_i)$$

Next Sentence Prediction (NSP)

- Binary prediction task: the model is presented with pairs of sentences and is asked to predict whether each pair consists of an actual pair of adjacent sentences from the training corpus or a pair of unrelated sentences.
- In BERT, 50% of the training pairs consisted of positive pairs, and in the other 50% the second sentence of a pair was randomly selected from elsewhere in the corpus.
- The NSP loss is based on how well the model can distinguish true pairs from random pairs.

Next Sentence Prediction Markup

- To facilitate NSP training, BERT introduces two new tokens to the input representation. After tokenizing the input with the subword model, the token [CLS] is prepended to the input sentence pair, and the token [SEP] is placed between the sentences and after the final token of the second sentence.
- During training, the output vector from the final layer associated with the [CLS] token represents the next sentence prediction.
- As with the MLM objective, a learned set of classification weights $\mathbf{W}_{NSP} \in \mathbb{R}^{2xd_h}$ is used to produce a two-class prediction from the raw [CLS] vector.

$$y_i = softmax(\mathbf{W}_{NSP} h_i)$$

Calculation CE Loss for Next Sentence Prediction

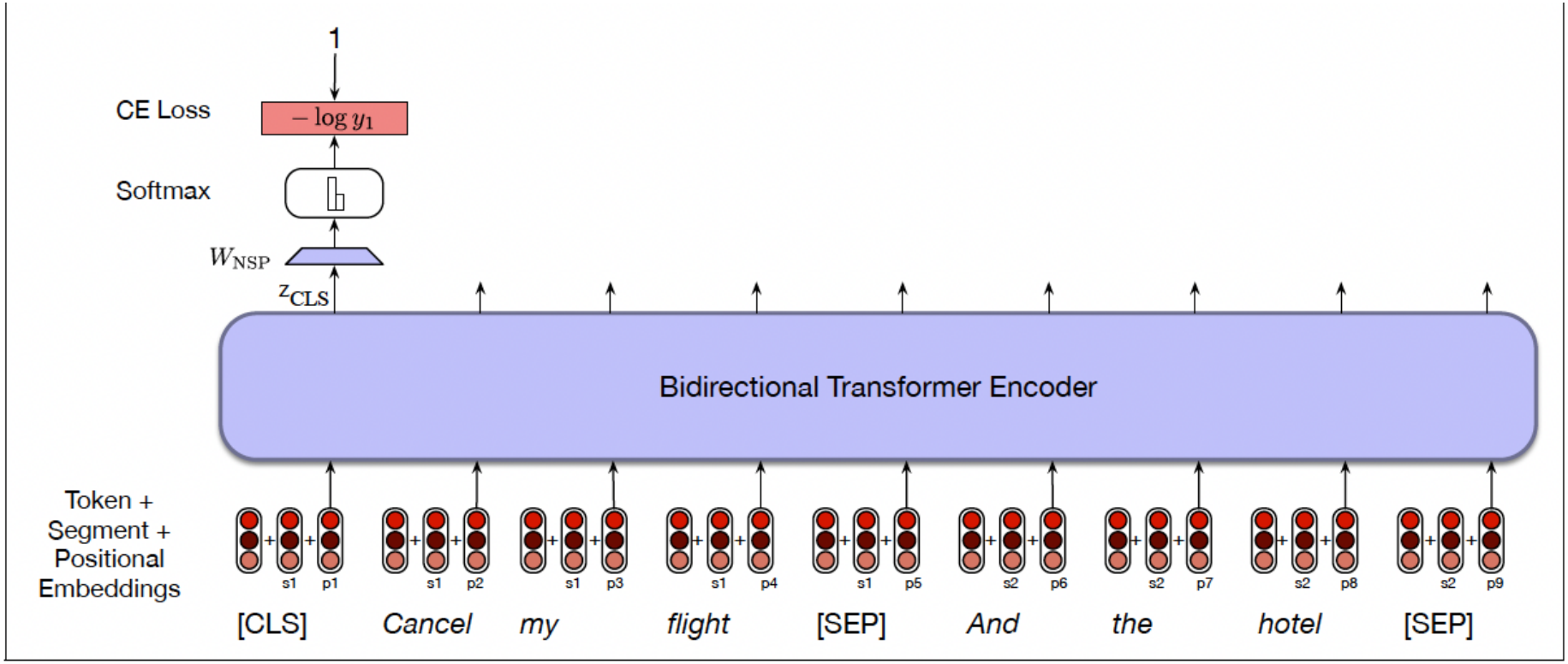
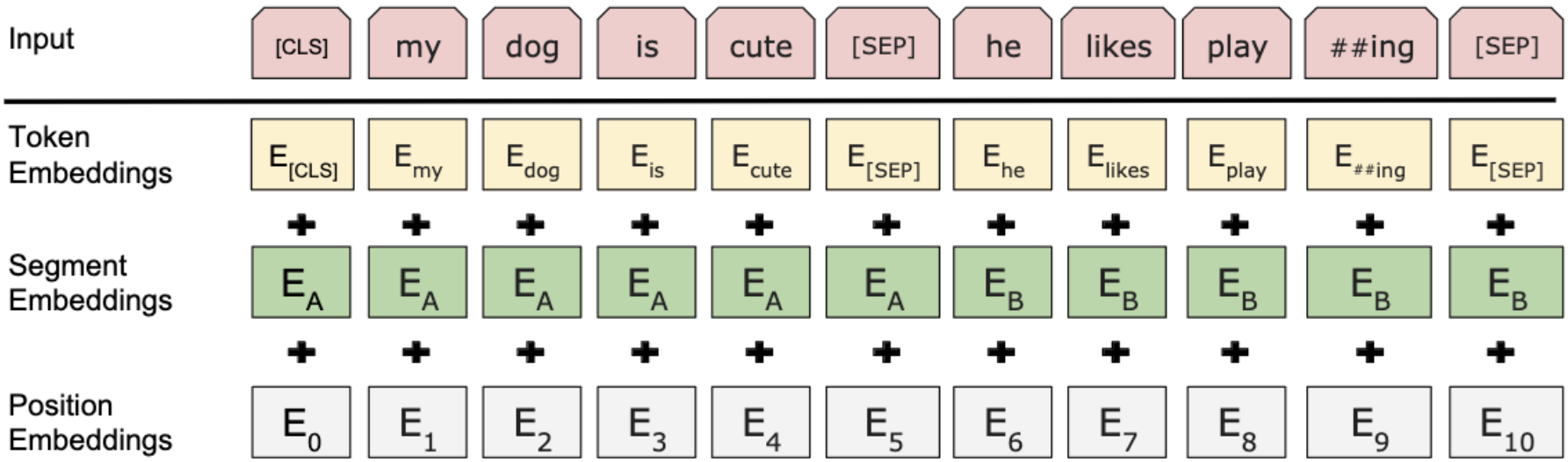


Figure 11.5 An example of the NSP loss calculation.

BERT Input Layer



Data Sampling in XLM-RoBERTa Model

Sampling is performed according to a multinomial distribution with probabilities $\{q_i\}_{i=1,...,N}$ and with n being the number of languages in the training data.

$$q_i = \frac{p_i^\alpha}{\sum_{j=1}^{N} p_i^\alpha} \text{ with } p_i = \frac{n_i}{\sum_{k=1}^{N} n_k} \tag{11.7}$$

Remark: setting of α to a value between 0 and 1 will give higher weight to lower probability samples. See also the discussion in chapter 6 on α exponents for PPMI calculation for improved performance of embeddings on a wide range of tasks.

XLM-RoBERTa model

The larger multilingual XLM-RoBERTa model, trained on 100 languages, has

- A multilingual subword vocabulary with 250,000 tokens generated using the SentencePiece Unigram LM algorithm (Kudo and Richardson, 2018).
- 24 layers of transformer blocks, with 16 multihead attention layers each
- Hidden layers of size 1024
- The resulting model has about 550M parameters.

Training Data for XLM-RoBERTa model

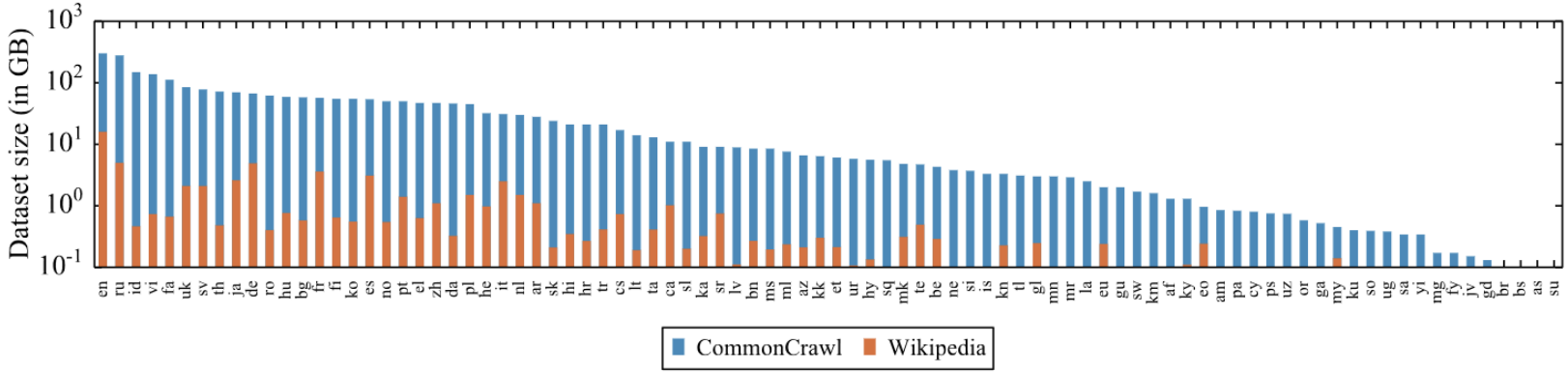


Figure 1: Amount of data in GiB (log-scale) for the 88 languages that appear in both the Wiki-100 corpus used for mBERT and XLM-100, and the CC-100 used for XLM-R. CC-100 increases the amount of data by several orders of magnitude, in particular for low-resource languages.

Slide taken from Alexis Conneau et al. (2020): Unsupervised Cross-lingual Representation Learning at Scale. `https://arxiv.org/pdf/1911.02116`

Advantages of Pretrained Multilingual Models

- Avoids the need to build many (e.g. in the case of 100 languages: 100!) separate monolingual models.
- Can improve performance on low-resourced languages by leveraging linguistic information from a similar language in the training data that happens to have more resources.

Disadvantages of Pretrained Multilingual Models

- When the number of languages grows very large, multilingual models exhibit what has been called *the curse of multilinguality* (Conneau et al., 2020): the performance on each language degrades compared to a model training on fewer languages.
- They 'have an accent': grammatical structures in higher resource languages (often English) bleed into lower-resource languages. For example: In pro-drop languages like Spanish, pronouns are not dropped in machine translation output.
- The vast amount of English language in training makes the model's representations for low-resource languages slightly more English-like.

Static vs. Contextual Embeddings

- Static embeddings (e.g. word2vec, GloVe) produce one embedding vector for each word type.
- Contextual embeddings represent the meaning of word instances: instances of a particular word type in a particular context.
- BERT and RoBERTa models produce contextual embeddings.

Generating Contextual Embeddings

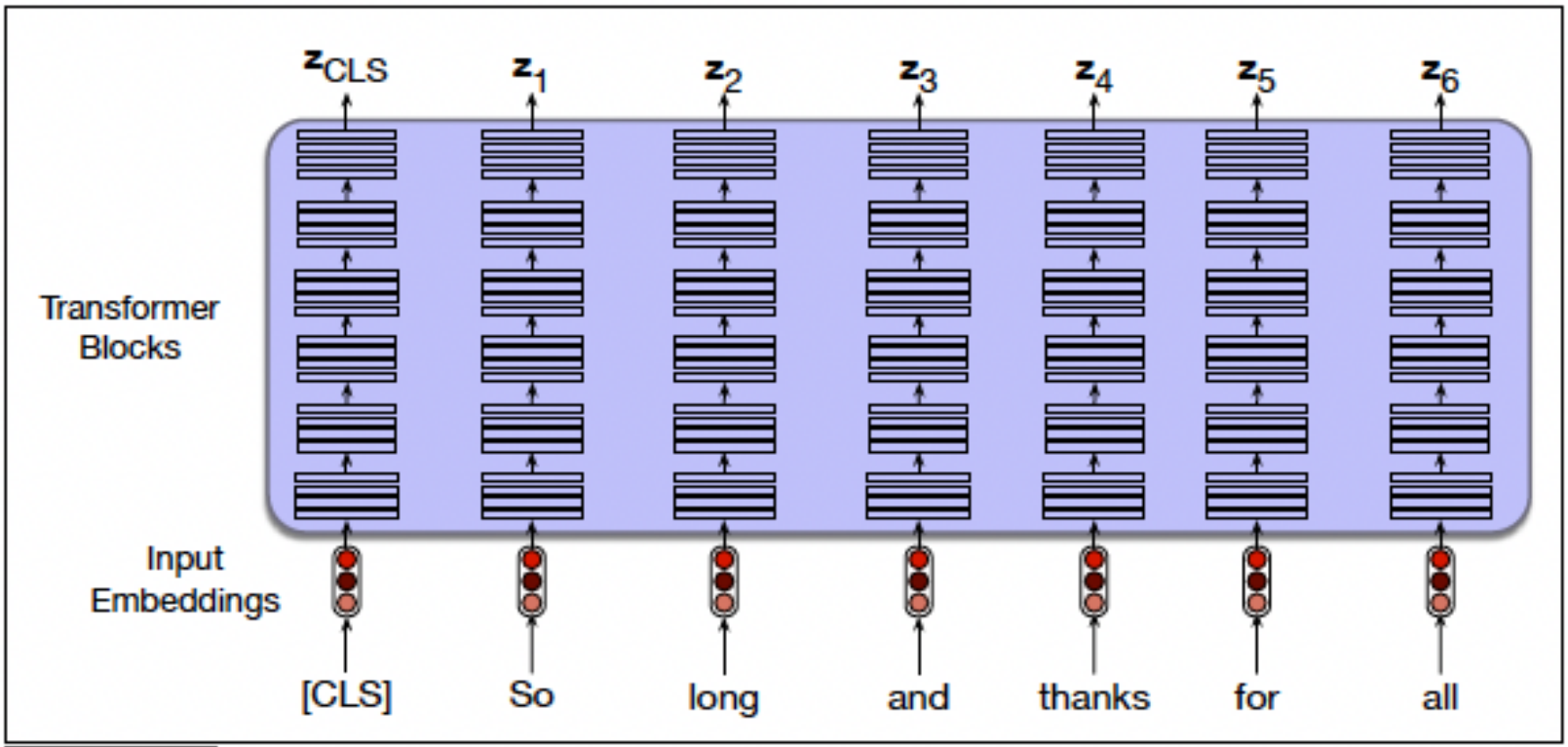


Figure 11.6 The output of a BERT-style model is a contextual embedding vector $\mathbf{z}_i$ for each input token x_i.

Word Senses

Some examples from English:

$mouse_1$:	.... 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 ...

Visualizing BERT Contextual Embeddings

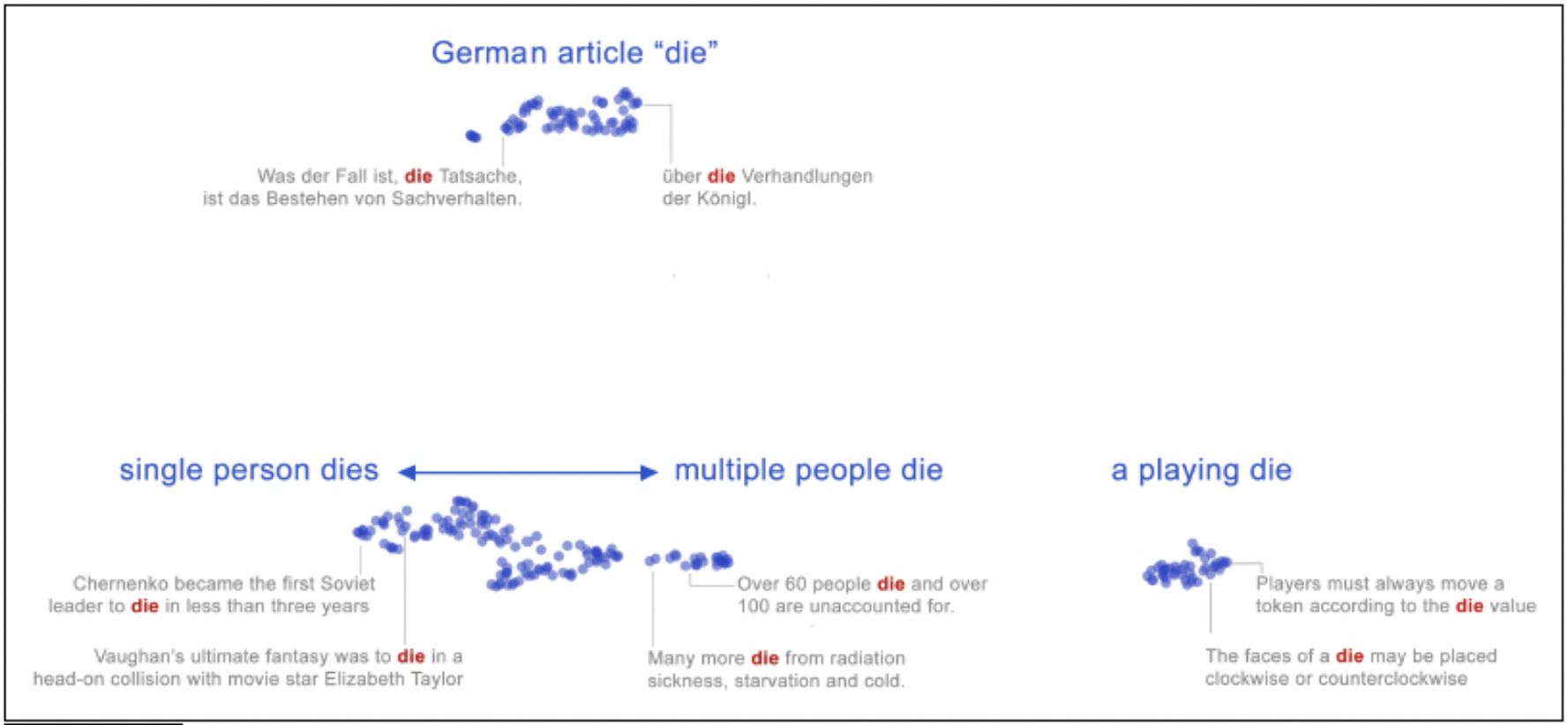


Figure 11.7 Each blue dot shows a BERT contextual embedding for the word *die* from different sentences in English and German, projected into two dimensions with the UMAP algorithm. The German and English meanings and the different English senses fall into different clusters. Some sample points are shown with the contextual sentence they came from. Figure from Coenen et al. (2019).

All-Words Word Sense Disambiguation (WSD)

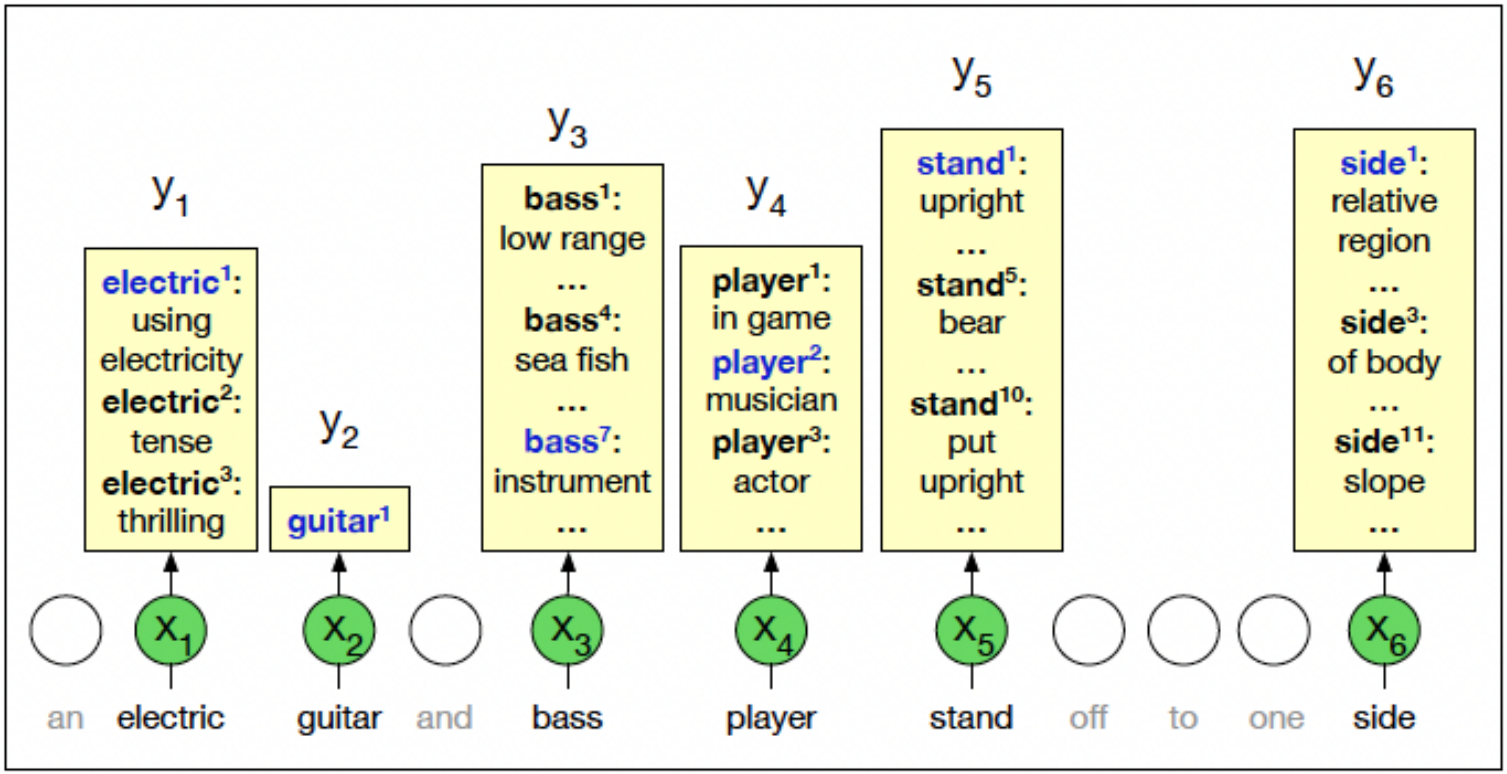


Figure 11.8 The all-words WSD task, mapping from input words (x) to WordNet senses (y). Figure inspired by Chaplot and Salakhutdinov (2018).

Nearest Neighbor Algorithm for WSD

- At training time, each sentence in some sense-labeled dataset (like the SemCore or SenseEval datasets in various languages) is passed through any contextual embedding (e.g., BERT).
- This results in a contextual embedding for each labeled token.
- For BERT it is common to pool multiple layers by summing the vector representations of a token *i* from the last four BERT layers).

Nearest Neighbor Computation

For each sense s of any word in the corpus, for each of the n tokens of that sense, average their n contextual representations v_i to produce a contextual sense embedding v_s for each sense s:

$$v_s = \frac{1}{n} \sum_i v_i \quad \forall v_i \in \text{tokens(s)} \tag{11.8}$$

At test time, given a token of a target word t in context, compute its contextual embedding t and choose its nearest neighbor sense from the training set, i.e., the sense whose sense embedding has the highest cosine with t:

$$\text{sense(t)} = \text{argmax}_{s \in \text{senses(t)}} \text{cosine}(t, v_s) \tag{11.9}$$

Illustrating the Nearest Neighbor Algorithm for WSD

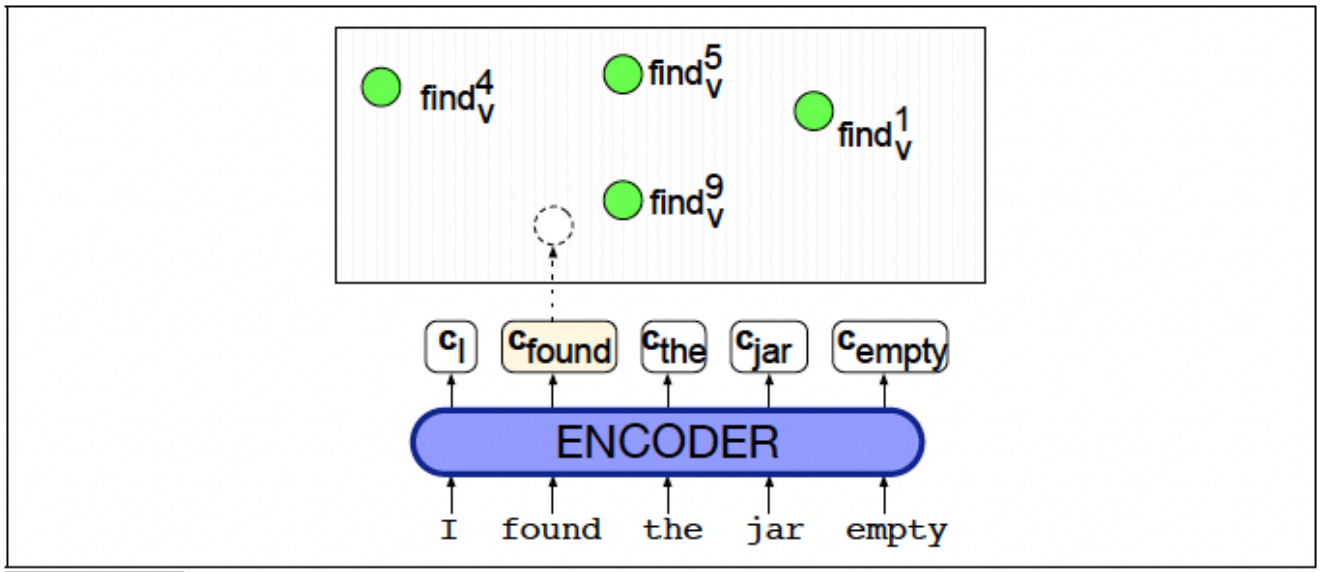


Figure 11.9 The nearest-neighbor algorithm for WSD. In green are the contextual embeddings precomputed for each sense of each word; here we just show a few of the senses for *find*. A contextual embedding is computed for the target word *found*, and then the nearest neighbor sense (in this case $\textbf{find}^9_v$) is chosen. Figure inspired by Loureiro and Jorge (2019).

Contextual Embeddings and Word Similarity

- Given a pretrained language model and a novel input sentence, one can think of the contextual sequence of model outputs as constituting contextual embeddings for each token in the input.
- The similarity of two words can measured by considering how close they are geometrically, by using the cosine as a similarity function
- If we look at the embeddings from the final layer of BERT or other models, embeddings for instances of any two randomly chosen words will have extremely high cosines that can be quite close to 1, meaning all word vectors tend to point in the same direction.

Isotropy and Anisotropy of a Model

- **Isotropy**: a measure of how uniformely the variance of a dataset is spread across its vector dimensions.
- In an isotropic model, the collection of vectors should point in all directions, and the expected cosine between a pair of random embeddings would be zero.
- An ideally isotropic embedding space has two properties :
  - All dimensions have the same variance.
  - All dimensions are uncorrelated with each other.

Anisotropy of transformer-based pre-trained LLMs

- The property of vectors in a system all tending to point in the same direction is known as **anisotropy**.
- Contextual embedding spaces for transformer-based pre-trained language models like BERT are highly anisotropic.
- Embeddings from the final layer of BERT or other models, embeddings for instances of any two randomly chosen words will have extremely high cosines that can be quite close to 1, meaning all word vectors tend to point in the same direction.

Causes of Anisotropy of transformer-based pre-trained LLMs

Timkey and van Schindel (2022):

- One cause of anisotropy is that cosine measures are dominated by a small number of dimensions of the contextual embedding, so-called **rogue dimensions**, whose values have very large magnitudes and very high variance
- the embeddings more isotropic by standardizing (z-scoring) the vectors, i.e., subtracting the mean and dividing by the variance.

Standardizing (z-scoring) the vectors

Given a set C of all the embeddings in some corpus, each with dimensionality d (i.e., $x \in \mathbb{R}^d$), the mean vector $\mu \in \mathbb{R}^d$ is:

$$\mu = \frac{1}{|C|} \sum_{\mathbf{x} \in |C|} \mathbf{x} \tag{11.10}$$

The standard deviation in each dimension $\sigma \in \mathbb{R}^d$ is:

$$\sigma = \sqrt{\frac{1}{|C|} \sum_{\mathbf{x} \in |C|} (\mathbf{x} - \mu)^2} \tag{11.11}$$

Then each word vector**x** is replaced by a standardized version **z**:

$$\mathbf{z} = \frac{(\mathbf{x} - \mu)}{\sigma} \tag{11.12}$$

Fine-Tuning Language Models

- The power of pretrained language models derives from extracting generalizations from very large amount of (typically unlabed) text.
- Pretrained language models can be fine-tuned for specific downstream applications.
- The fine-tuning process consists of using labeled, application-specific data to train additional application-specific parameters.

BERT PreTraining and Fine-Tuning

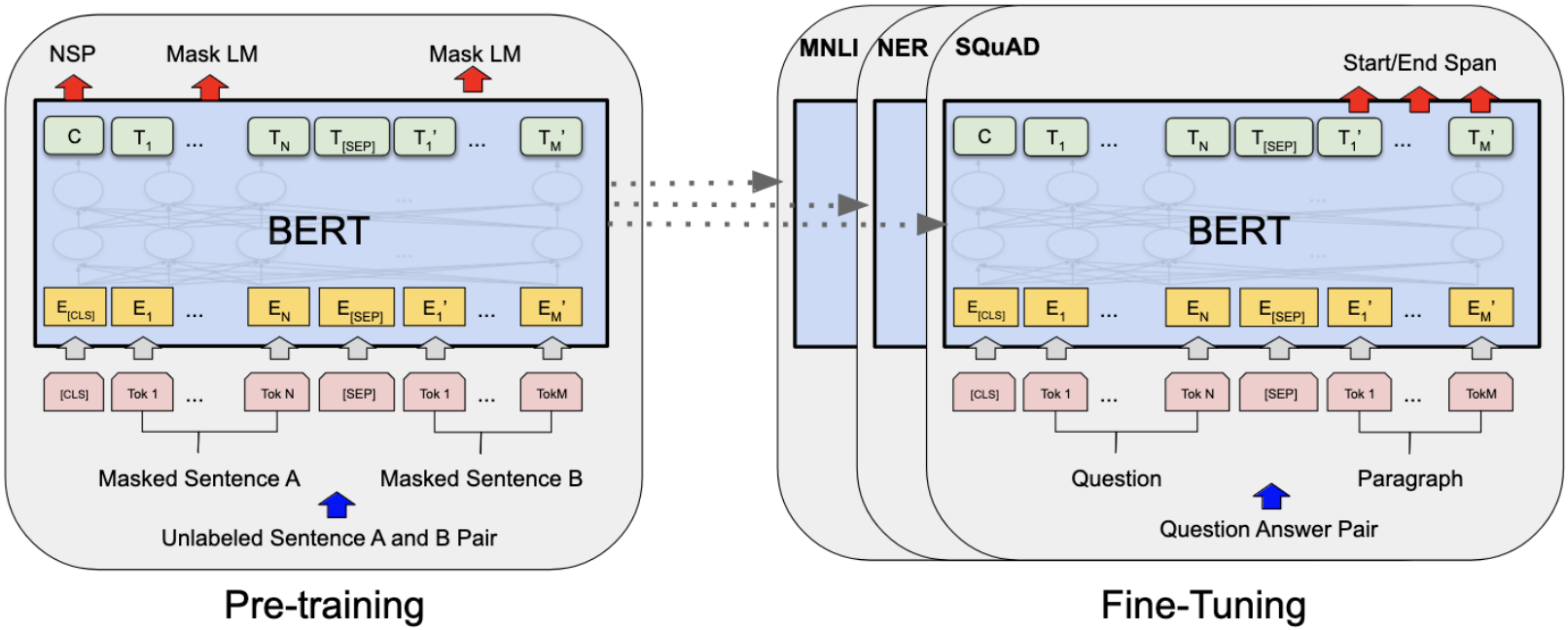


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

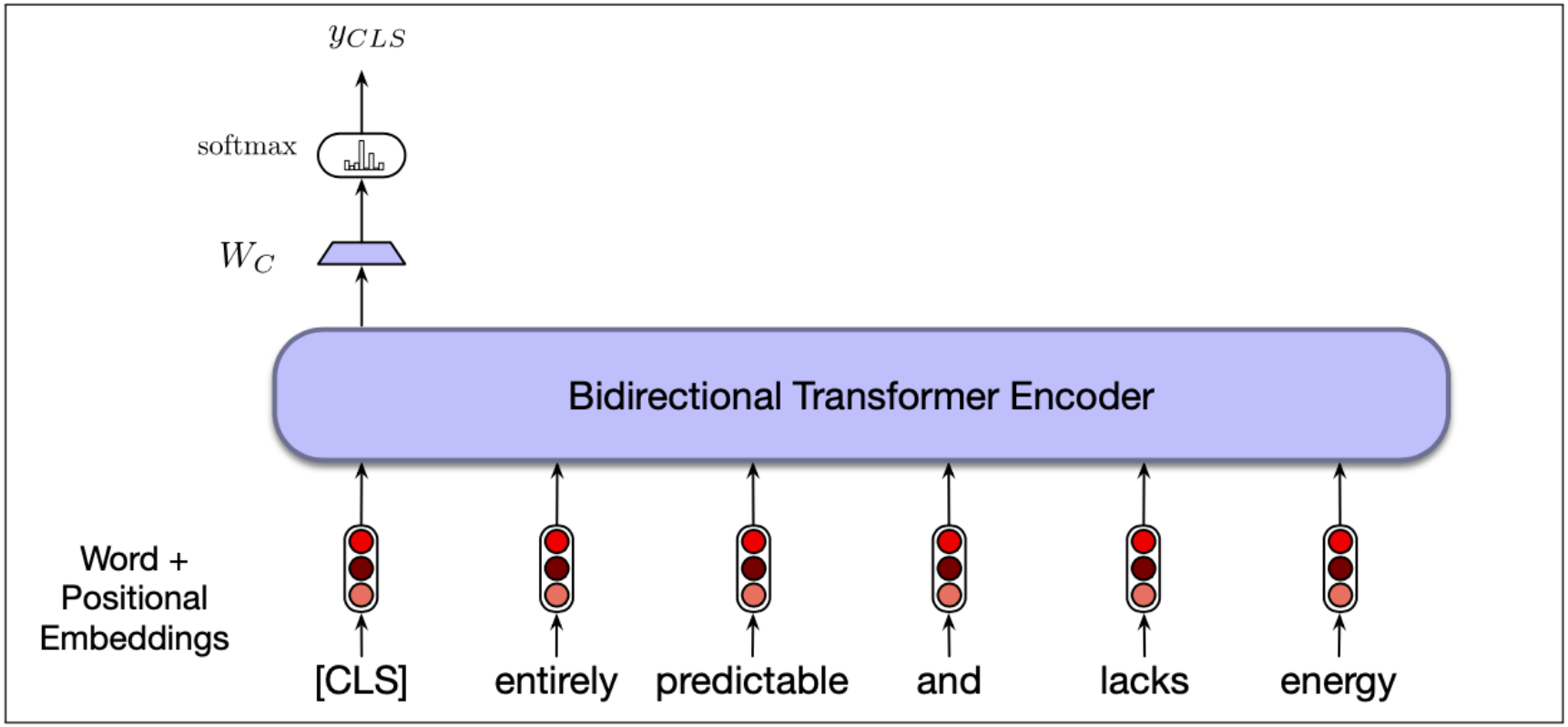
Sentence Embeddings and Sequence Classification

- In an RNN, the hidden-layer representation of the final input token of a sequence is used to stand for the entire sequence.
- In a Transformer model, the FFN output of the final transformer block is for the final input token of an input sequence is used to stand for the entire sequence and is connected to a classification head, for example for language modelling or for text categorization.

Sentence Embeddings and Sequence Classification

- In BERT models, the [CLS] token plays the role of this embedding. This unique token is added to the vocabulary and is prepended to the start of all input sequences, both during pretraining and encoding.
- The output vector in the final layer of the model for the [CLS] input represents the entire input sequence and serves as the input to classifier head a classifier head, a logistic regression or neural network classifier that makes the relevant decision.

Sequence Classification



Pair-Wise Sequence Classification

- During fine-tuning, pairs of labeled sentences from the supervised training data are presented to the model, and run through all the layers of the model to produce the z outputs for each input token.
- As with sequence classification, the output vector associated with the prepended [CLS] token represents the model's view of the input pair.
- And as with NSP training, the two inputs are separated by the [SEP] token.
- To perform classification, the [CLS] vector is multiplied by a set of learning classification weights and passed through a softmax to generate label predictions, which are then used to update the weights.

Pair-Wise Sequence Classification

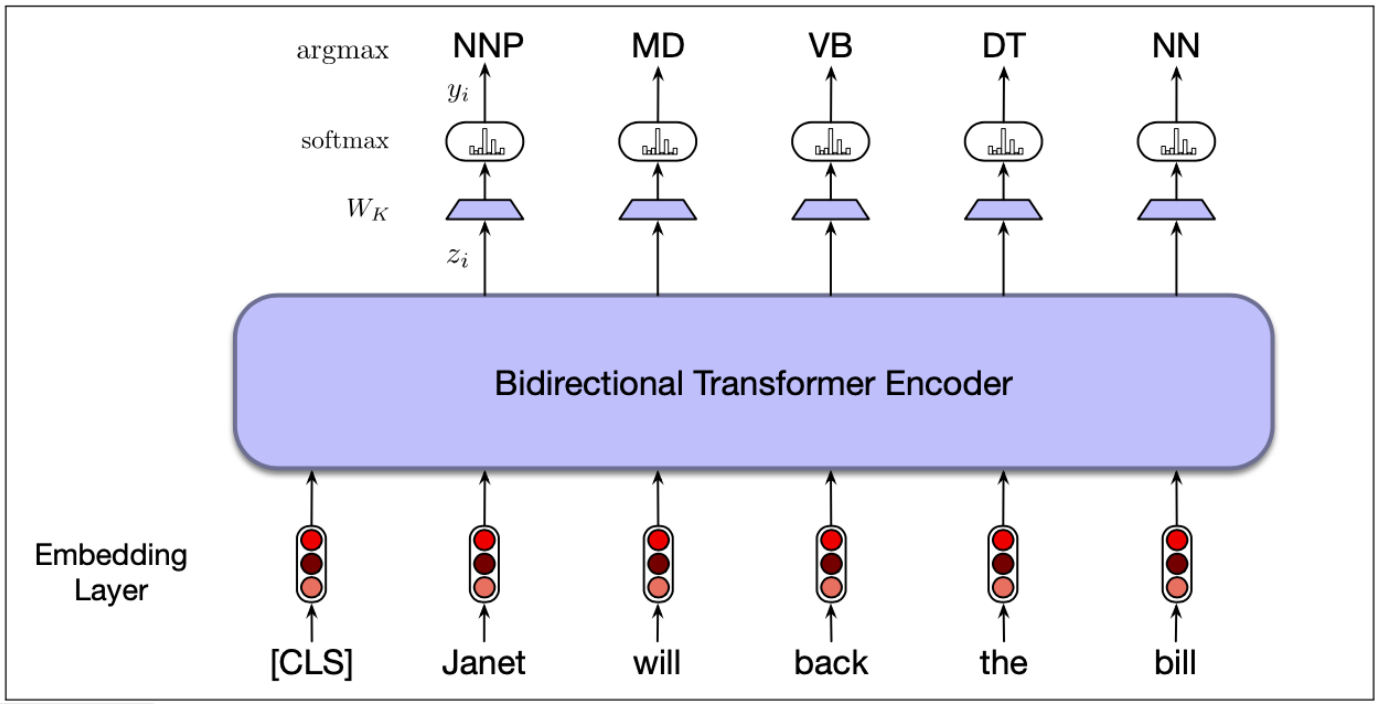
Examples of each class from the NLI corpus

- 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.

Sequence Labelling

$$\mathbf{y_i} = \text{softmax}(\mathbf{W_K z_i}) \tag{11.14}$$

$$\mathbf{t_i} = \text{argmax}_k(\mathbf{y}_i) \tag{11.15}$$



Sequence Labelling

Example. The following sentence containing two named entities

[LOC **Mt. Sanitas**] is in [LOC **Sunshine Canyon**] .
would have the following set of per-word BIO tags.

(11.16) *Mt. Sanitas is in Sunshine Canyon.*
B-LOC I-LOC O O B-LOC I-LOC O

Unfortunately, the WordPiece tokenization for this sentence yields the following sequence of tokens which doesn't align directly with BIO tags in the ground truth annotation:

```
'Mt' , '.', 'San', '##itas', 'is', 'in', 'Sunshine',
'Canyon' '.'
```

Masking Spans: SpanBERT (Joshi et al. 2020)

- NLP Applications with units of interest larger than individual words: question answering, syntactic parsing, coreference, or semantic role labelling.
- For these applications, spans, i.e. contiguous sequences of one or more words, are the units of interest.
- In span-based masking, a set of randomly selected spans from a training sequence are chosen.
- Within a randomly selected span, all the tokens are substituted according to the same recipe as in BERT, with the total token substitution is limited to 15%.

Span Representations

- Downstream span-based applications rely on span representations derived from the tokens within the span, as well as the start and end points, or the boundaries of a span.
- Representations for these boundaries are typically derived from the first and last tokens of a span, the tokens immediately preceding and following the span, or some combination of them.
- The SpanBERT learning objective augments the MLM objective with a boundary oriented component called the **Span Boundary Objective** (SBO).
- The SBO relies on a model's ability to predict the tokens within a masked span from the tokens immediately preceding and following the span.

Span Representations

- Let the sequence of output from the transformer encoder for the n input tokens $s_1, \ldots, s_n$ be $z_1, \ldots, z_n$
- A token x_i in a masked span of tokens $x_s \ldots x_e$, i.e., starting with token x_s and ending with token x_e , is represented by concatenating 3 embeddings:
  - The first two are the embeddings of two external boundary tokens x_{s-1} and x_{e+1}, i.e. , the token preceding x_s , the token following x_e.
  - The third embedding that is concatenated is the relative position embedding of the target token p_{i-s+1}.
- The position embeddings $p_1 p_2 \ldots$ represent relative positions of the tokens with respect to the left boundary token x_{s-1}.

Masking Spans: Span Boundary Objective (SBO)

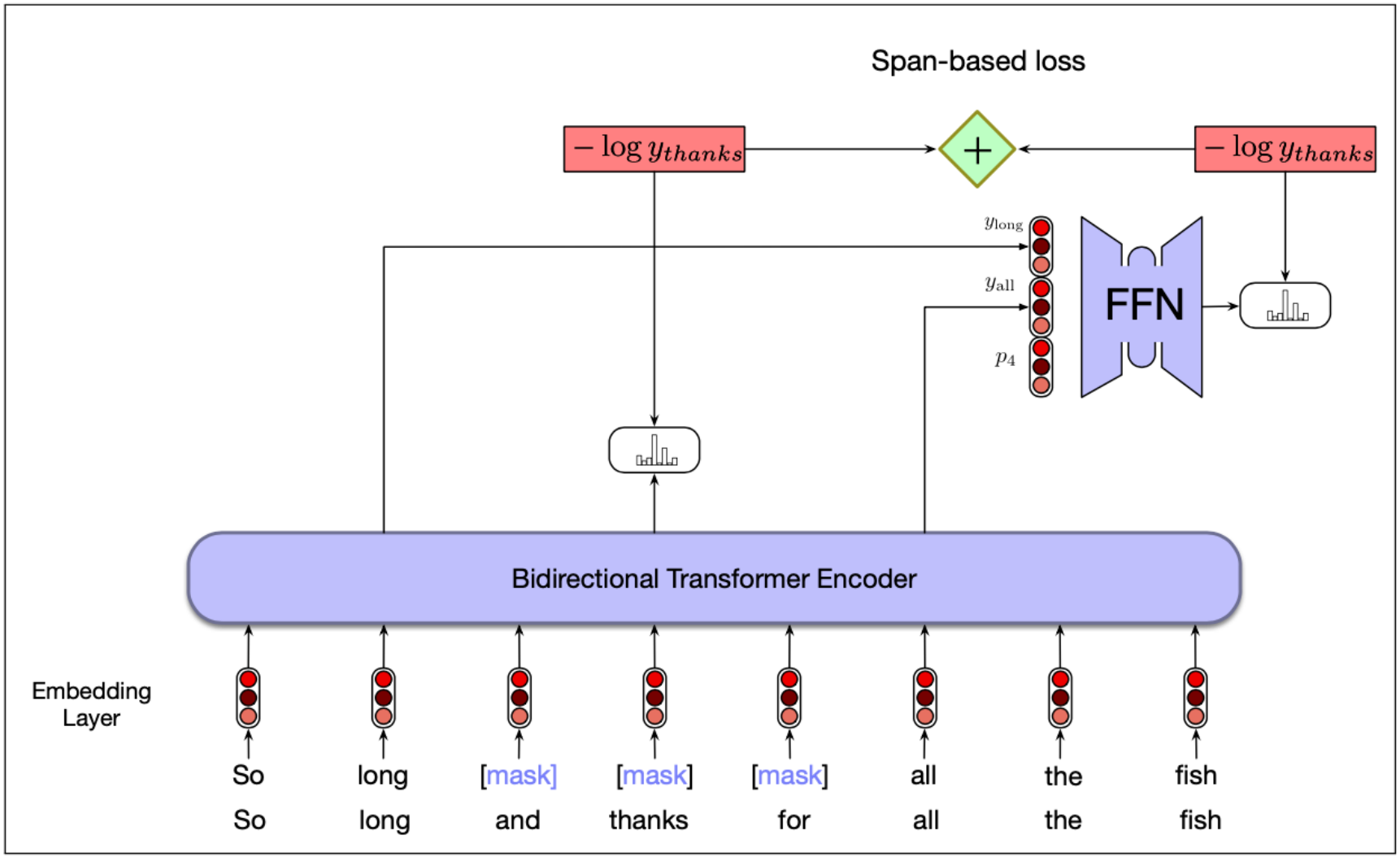
$$L(x) = L_{MLM}(x) + L_{SBO}(x) \tag{11.17}$$

$$L_{SBO}(x_i) = -\log P(x_i | x_{s-1}, x_{e+1}, p_{i-s+1}) \tag{11.18}$$

$$\mathbf{s}_i = \text{FNN}([\mathbf{z}_{s-1}; \mathbf{z}_{e+1}; \mathbf{p}_{i-s+1}]) \tag{11.19}$$

$$y_i = \text{softmax}(W_V s_i)) \tag{11.10}$$

Masking Spans



Fine-tuning for Span-Based Applications

Simplest Approach:

$$g_{ij} = \frac{1}{(j-i)+1} \sum_{k=i}^{j} z_k \tag{11.21}$$

$$\text{spanRep}_{ij} = [\mathbf{z}_i; \mathbf{z}_j; \mathbf{g}_{i,j}] \tag{11.22}$$

More elaborate approach with FFNs for start and end points:

$$\mathbf{s}_i = \text{FFN}_{start}(\mathbf{z}_i) \tag{11.23}$$

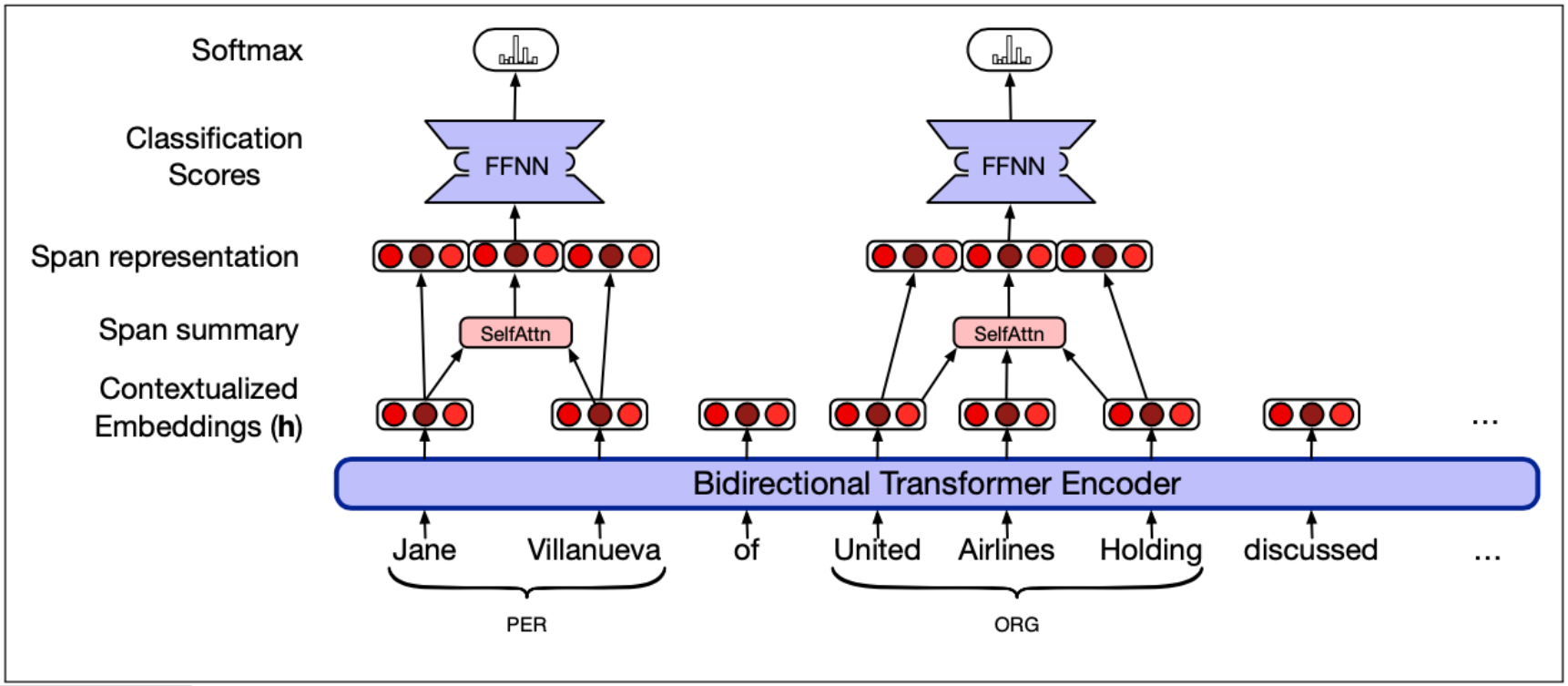
$$\mathbf{e}_j = \text{FFN}_{end}(\mathbf{z}_j) \tag{11.24}$$

$$\text{spanRep}_{ij} = [\mathbf{s}_i; \mathbf{e}_j; \mathbf{g}_{i,j}] \tag{11.25}$$

$$\mathbf{g_{i,j}} = \text{SelfATTN}(\mathbf{z}_{i:j}) \tag{11.26}$$

Fine-tuning for Span-Based Applications

$$y_{ij} = \text{softmax}(spanRep_{ij}) \quad (11.27)$$



Comparative 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	79.8	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**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

Comparative Architectures

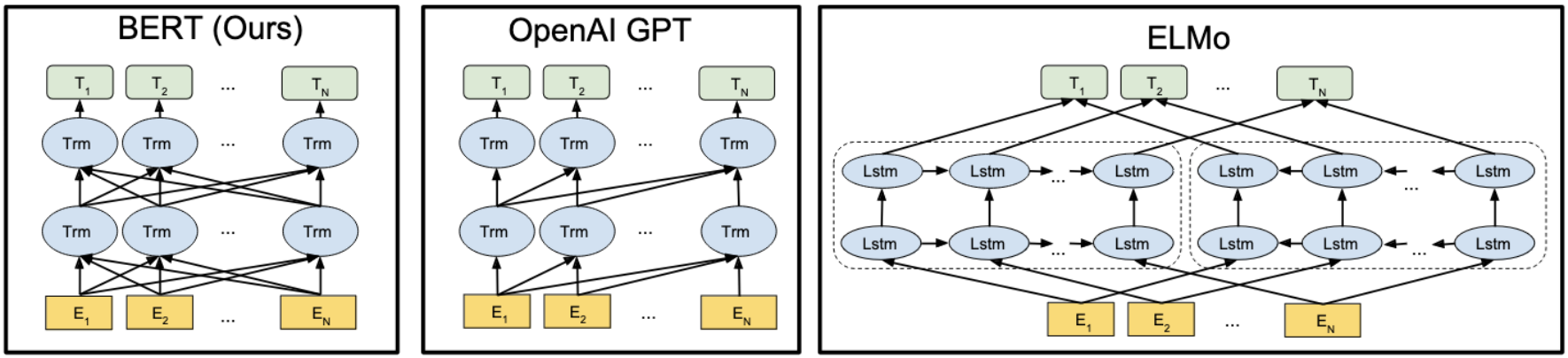


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

BERTology

- The original BERT paper and its implementation have spawned off a lot of follow-up research in the last three years.
- Here are some well-known examples:
  - SpanBERT
  - RoBERTa
  - AlBERT
  - XLNet
  - ...

Contributions of BERT

1. Models bi-directionally.
2. Makes it possible to re-use the model architecture for many different tasks.
3. Achieves state-of-the-art results for word-based and sentence-based tasks.