Masked Language Models: Bidirectional Transformer Encoders

- **Bidirectional Transformer Encoders**
  - **BERT - Bidirectional Encoder Representations from Transformers**
- **Fine-Tuning Language Models**

Bidirectional Transformer Encoders

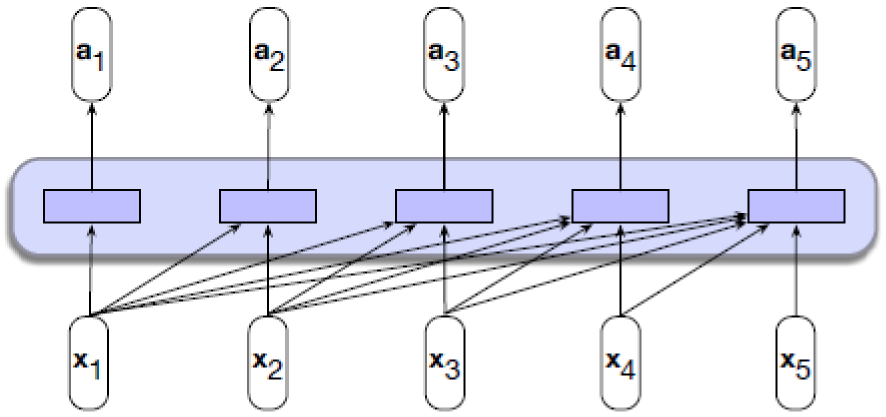
Bidirectional Transformer Encoders

- **Bidirectional Transformer Encoders** are ***pretrained language models***, trained via **masked language modeling**, a method that allows the model to see entire texts at a time, including both the right and left context.
  - The most widely-used masked language modeling architecture is **BERT** (**Bidirectional Encoder Representations from Transformers**)

- With **Contextual Embeddings**, each word w will be represented by a different vector each time it appears in a different context.
  - **contextual embeddings** are learned by masked language models like BERT.
  - **static embeddings**: word2vec (or GloVe) learns a single vector embedding for each unique word w in the vocabulary.

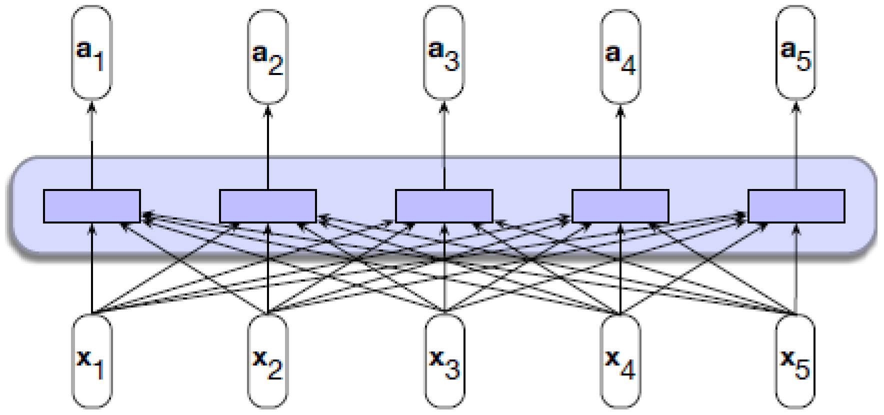
Bidirectional Transformer Encoders

- The focus of **bidirectional encoders** is on ***computing contextualized representations*** of the tokens in an input sequence that are useful in many NLP applications.



A self-attention layer in backward looking transformer model

- Each output is computed using only information seen earlier in the context.



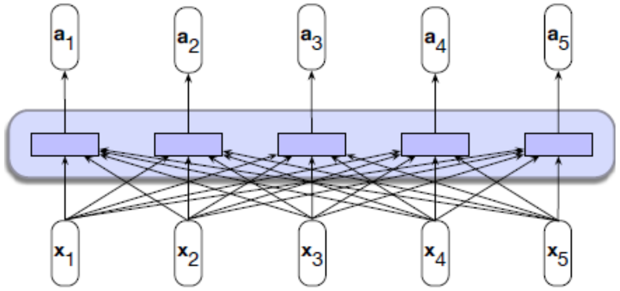
A self-attention layer in bidirectional encoder model

- Each output is computed using all inputs, both before and after the current.

Bidirectional Transformer Encoders

- Bidirectional Encoders (such as BERT) make use of Transformer, an attention mechanism that learns ***contextual relations between words*** (or sub-words) in a text.
  ➔ ***contextualized representations***

- Bidirectional encoders use self-attention to map input embeddings ($x_{1,\dots,}x_n$) to output embeddings ($a_{1,\dots,}a_n$) the same length, where the output vectors have been contextualized using information from the entire input sequence.



- **Encoder** that reads the text input and **Decoder** that produces a prediction for the task.

- Since BERT's goal is to generate a ***language model***, only the **encoder** mechanism is necessary.
  – Bidirectional Encoders produce an encoding for each input token but generally aren't used to produce running text by decoding/sampling.

Bidirectional Transformer Encoder Architecture

- Bidirectional Encoders use same transformer architecture except that
  - ***They contextualize each token using information from the entire input.***

- We can multiply input embeddings X by key, query, and value matrices to produce matrices **Q**, **K** and **V** containing all the query, key and value vectors:

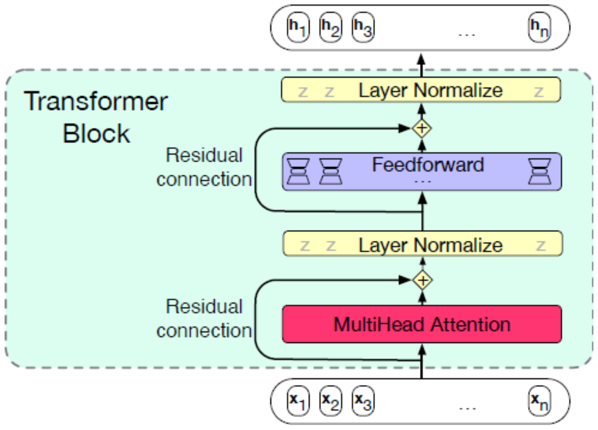
$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

- We can reduce the entire self-attention step for an entire sequence of N tokens to the following computation

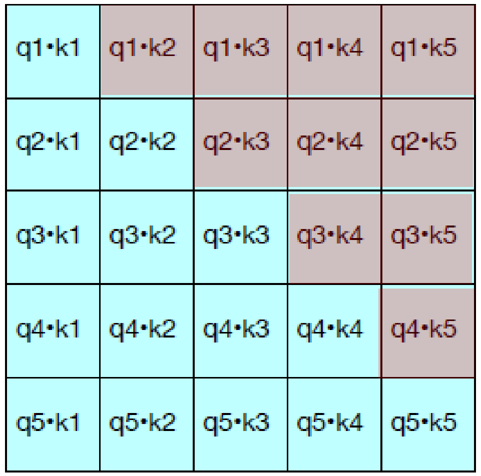
$$\text{SelfAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\intercal}}{\sqrt{d_k}}\right)\mathbf{V}$$

$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}} \qquad \mathbf{k}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{K}} \qquad \mathbf{v}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{V}}$$

$$\text{score}(\mathbf{x}_i,\mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j \qquad \alpha_{ij} = \frac{\exp(\text{score}_{ij})}{\sum_{k=1}^{n}\exp(\text{score}_{ik})} \qquad \mathbf{y}_i = \sum_{j=1}^{n}\alpha_{ij}\mathbf{v}_j$$



N×N QK^T matrix showing complete set of $q_i.k_i$ comparisons



N	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

BERT Architecture

- Original English-only bidirectional transformer encoder model, BERT consisted of:
  - An English-only subword vocabulary consisting of 30,000 tokens generated using the WordPiece algorithm (Schuster and Nakajima, 2012).
  - Hidden layers of size of 768,
  - 12 layers of transformer blocks, with 12 multihead attention layers each.
  - The resulting model has about 100M parameters.
- The larger multilingual XLM-RoBERTa model, trained on 100 languages, has
  - A multilingual subword vocabulary with 250,000 tokens generated using SentencePiece Unigram LM algorithm (Kudo and Richardson, 2018b).
  - 24 layers of transformer blocks, with 16 multihead attention layers each
  - Hidden layers of size 1024
  - The resulting model has about 550M parameters.
- The size of the input layer dictates the complexity of the model.
  - Both time and memory requirements in transformer grow quadratically with input length.
  - The input length should be long enough to provide sufficient context for the model to function and yet still be computationally tractable.
  - For BERT and XLM-RoBERTa, a fixed input size of 512 subword tokens was used.

Training Bidirectional Encoders

Training Bidirectional Encoders

- A ***causal (backward looking) transformer language model*** is trained by iteratively predicting the next word in a text.
  - This approach does not work with biderectional encoders because the next word is given in the context.

- **Biderectional transformer encoder language models** are trained by ***predicting the missing elements*** in a text.
  - Given an input sequence with one or more elements missing, the learning task is to predict the missing elements.
  - During training, the model must generate a probability distribution over the vocabulary for each of the missing items.
  - Example: the model is asked to predict a missing item in the sentence.

    Please turn _____ homework in.

Training Bidirectional Encoders

Masked Language Modeling (MLM)

- The ***original approach to training bidirectional encoders*** is called **Masked Language Modeling (MLM).**

- MLM uses unannotated text from a large corpus

- 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:
  a. It is replaced with the **unique vocabulary token [MASK]**
  b. It is replaced with **another token from the vocabulary**, randomly sampled based on token unigram probabilities
  c. It is **left unchanged**

- 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, and the remaining 10% are left unchanged

Training Bidirectional Encoders

Masked Language Modeling (MLM)

The **MLM training objective** is to predict the original inputs for each of the masked tokens using a bidirectional encoder

- ***The cross-entropy loss from these predictions drives the training process for all the parameters in the model***
- ***Note that all the input tokens play a role in the self-attention process, but only the sampled tokens are used for learning***

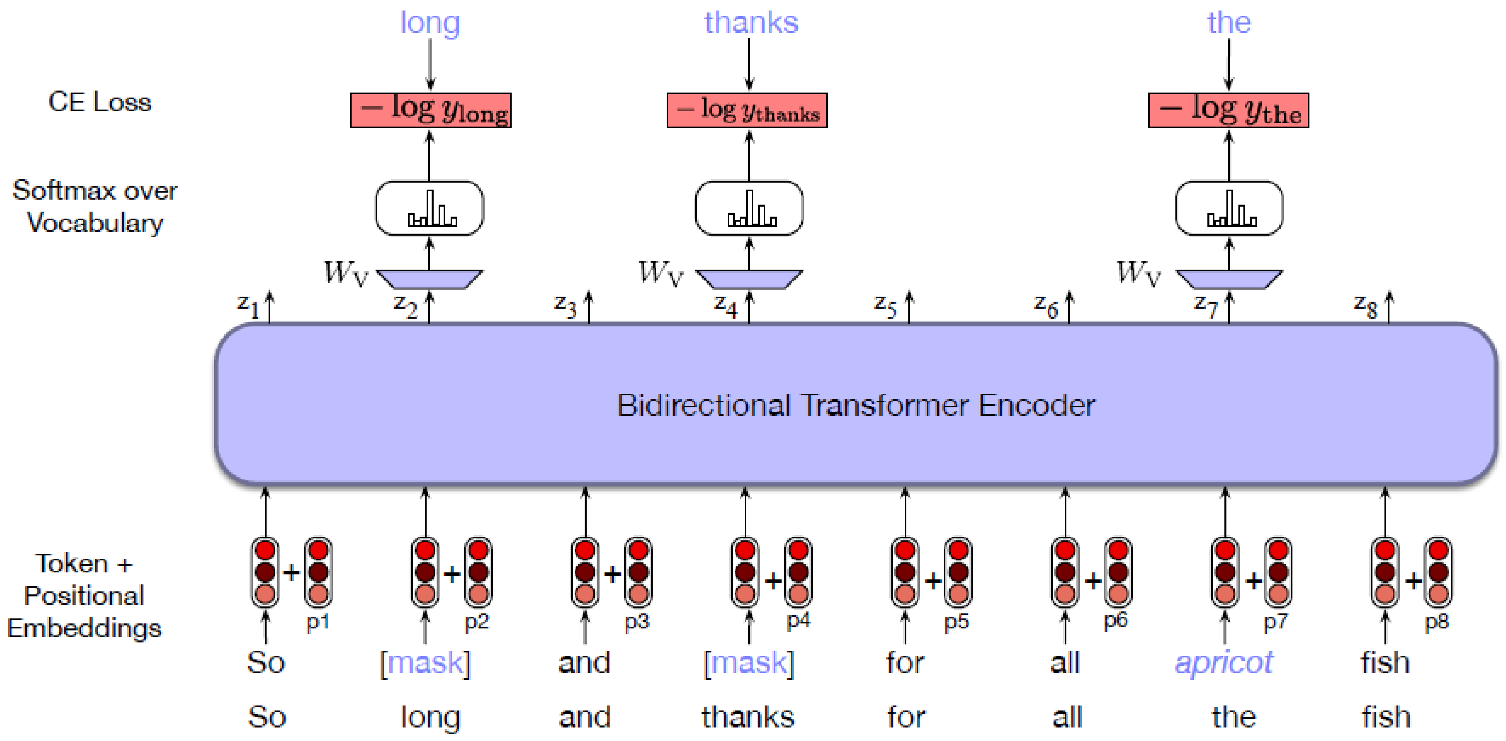
Input:

- Original input sequence is first tokenized using a subword model
- The sampled items which drive the learning process are chosen from among the set of tokenized inputs
- Word embeddings for all the tokens in the input are retrieved from the word embedding matrix and then combined with positional embeddings to form the input to the transformer

Training Bidirectional Encoders

Masked Language Modeling (MLM) Example

- Three of the input tokens (**long**, **thanks**, **the**) are selected, two of them (**long**, **thanks**) are **masked** and the third (**the**) is replaced with an unrelated word (**apricot**) .
- 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 BERT, words are subword tokens



long
thanks
the
CE Loss
$-\log y_{\text{long}}$
$-\log y_{\text{thanks}}$
$-\log y_{\text{the}}$
Softmax over Vocabulary
W_V
z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8
Bidirectional Transformer Encoder
Token + Positional Embeddings
p1 p2 p3 p4 p5 p6 p7 p8
So [mask] and [mask] for all apricot fish
So long and thanks for all the fish

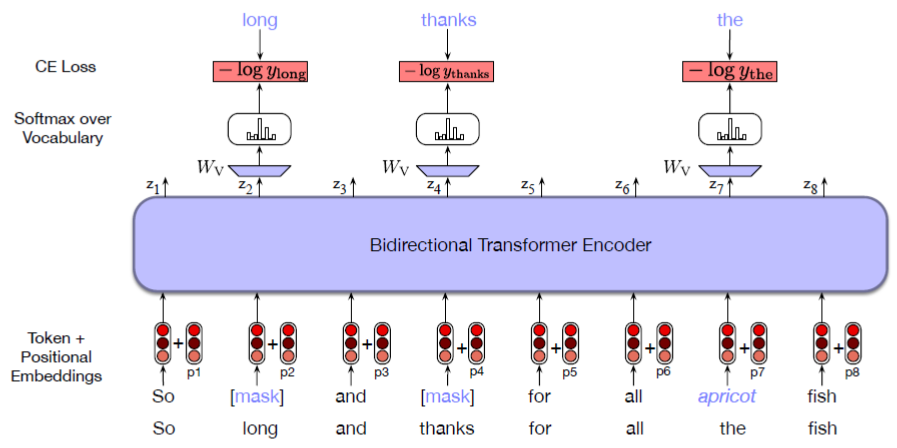
Training Bidirectional Encoders

Masked Language Modeling (MLM) Example

- In this example, **long**, **thanks** and **the** have been sampled from the training sequence, with the first two masked and the replaced with the randomly sampled token **apricot**.
- Resulting embeddings are passed through a stack of bidirectional transformer blocks.
- To produce a probability distribution over the vocabulary for each of the **masked (sampled)** tokens,
  - the output vector z_i from the final transformer layer for each masked token i is multiplied by a learned set of classification weights $W_V \in R^{|V| \times dh}$ and
  - then through a softmax to yield the required predictions over the vocabulary.

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

- With a predicted probability distribution for each masked item, we can use cross-entropy to compute the loss for each masked item
  - the negative log probability assigned to the actual masked word,



long
thanks
the
CE Loss
$-\log y_{\text{long}}$
$-\log y_{\text{thanks}}$
$-\log y_{\text{the}}$
Softmax over Vocabulary
W_V
W_V
W_V
z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8
Bidirectional Transformer Encoder
Token + Positional Embeddings
p1 p2 p3 p4 p5 p6 p7 p8
So [mask] and [mask] for all apricot fish
So long and thanks for all the fish

Training Bidirectional Encoders

Masked Language Modeling (MLM)

- For a given vector of input tokens **x**, let
  - **M** be the set of tokens that are **masked (sampled)**,
  - **z** be the sequence of output vectors.
- For a given ***masked input token*** $\mathbf{x_i}$, the **loss** is :

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

- The gradients that form the basis for the weight updates are based on the average loss over the sampled learning items from a single training sequence.

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

  - Only the tokens in M play a role in learning

Training Bidirectional Encoders

Next Sentence Prediction (NSP)

- The focus of **mask-based learning** is on predicting words from surrounding contexts with the goal of producing effective word-level representations.
- An important class of applications involves determining the relationship between pairs of sentences
  - paraphrase detection (detecting if two sentences have similar meanings)
  - entailment (detecting if the meanings of two sentences entail or contradict each other)
  - discourse coherence (deciding if two neighboring sentences form a coherent discourse)
- ***To capture the kind of knowledge required for applications such as these, BERT introduced a second learning objective*** called **Next Sentence Prediction (NSP)**
- **Training:** 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

Training Bidirectional Encoders

Next Sentence Prediction (NSP)

- 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***

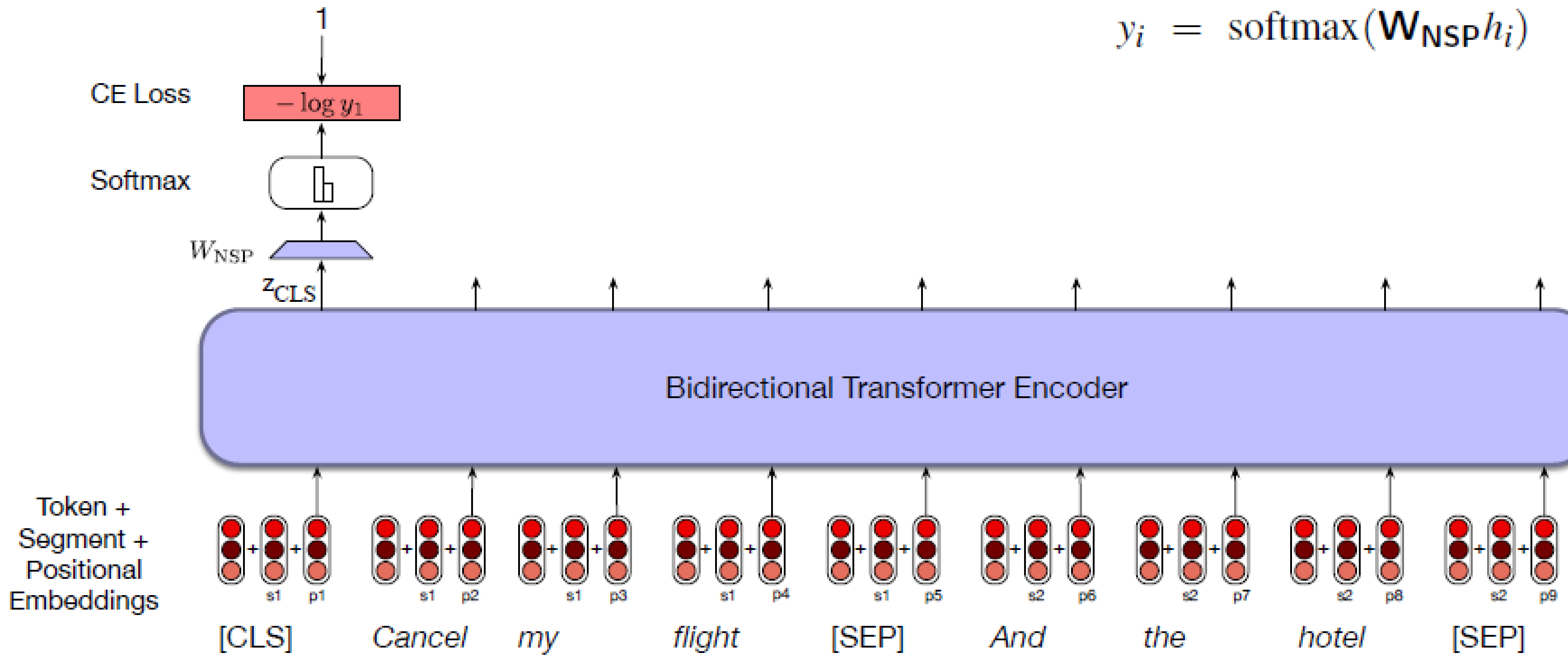
- BERT introduces two new tokens to the input representation
  - **After tokenizing the input with the subword model, the token [CLS] is prepended to the input sentence pair, and the token [SEP] is placed between the sentences and after the final token of the second sentence**
  - **During training, the output vector from the final layer associated with the [CLS] token represents the next sentence prediction**

Training Bidirectional Encoders

Next Sentence Prediction (NSP)

- The **output vector Z_{CLS}** from the final layer associated with the **[CLS]** token represents the next sentence prediction.
- A learned set of classification weights $W_{NSP} \in R^{2\times dh}$ is used to produce a two-class prediction from the raw **[CLS]** vector.
- Cross entropy is used to compute the NSP loss for each sentence pair presented to the model.

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



Training Bidirectional Encoders

BERT Training

- In BERT, the NSP loss was used in conjunction with the MLM training objective to form final loss.
- The original BERT models were trained on about 3.3 billion words.
  - Modern masked language models are now trained on much larger datasets of web text
  - The XLM-RoBERTa model was trained on about 300 billion tokens in 100 languages
- To train the original BERT models, pairs of text segments were selected from the training corpus according to the next sentence prediction 50/50 scheme.
  - Pairs were sampled so that their combined length was less than the 512 token input.
  - Tokens within these sentence pairs were then masked using the MLM approach with the combined loss from the MLM and NSP objectives used for a final loss.
- Some models, like the RoBERTa model, drop the next sentence prediction objective, and therefore change the training regime a bit.
  - Instead of sampling pairs of sentence, the input is simply a series of contiguous sentences.
  - If the document runs out before 512 tokens are reached, an extra separator token is added, and sentences from the next document are packed in, until we reach a total of 512 tokens.

Fine-Tuning Language Models

Fine-Tuning Language Models

- The power of **pretrained language models** lies in their ability to extract generalizations from huge amounts of text and these generalizations are useful for many NLP applications.
- **Fine-tuning** is the process of taking the network learned by ***pretrained models***, and further training the model, via an added neural net classifier that takes the top layer of the network as input, to perform some task like sequence classification or named entity tagging.
  - The **pretraining phase** learns a ***language model*** that instantiates rich representations of word meaning, that thus enables the model to more easily learn (**'be fine-tuned to'**) the requirements of a downstream language understanding task.
  - The **pretrain-finetune paradigm** is an instance of what is called **transfer learning** in machine learning: *the method of acquiring knowledge learning from one task or domain, and then applying it (transferring it) to solve a new task*
- The **fine-tuning process** consists of using labeled data about the application to train these additional application-specific parameters.
  - This training will either freeze or make only minimal adjustments to the pretrained language model parameters.

Fine-Tuning Language Models

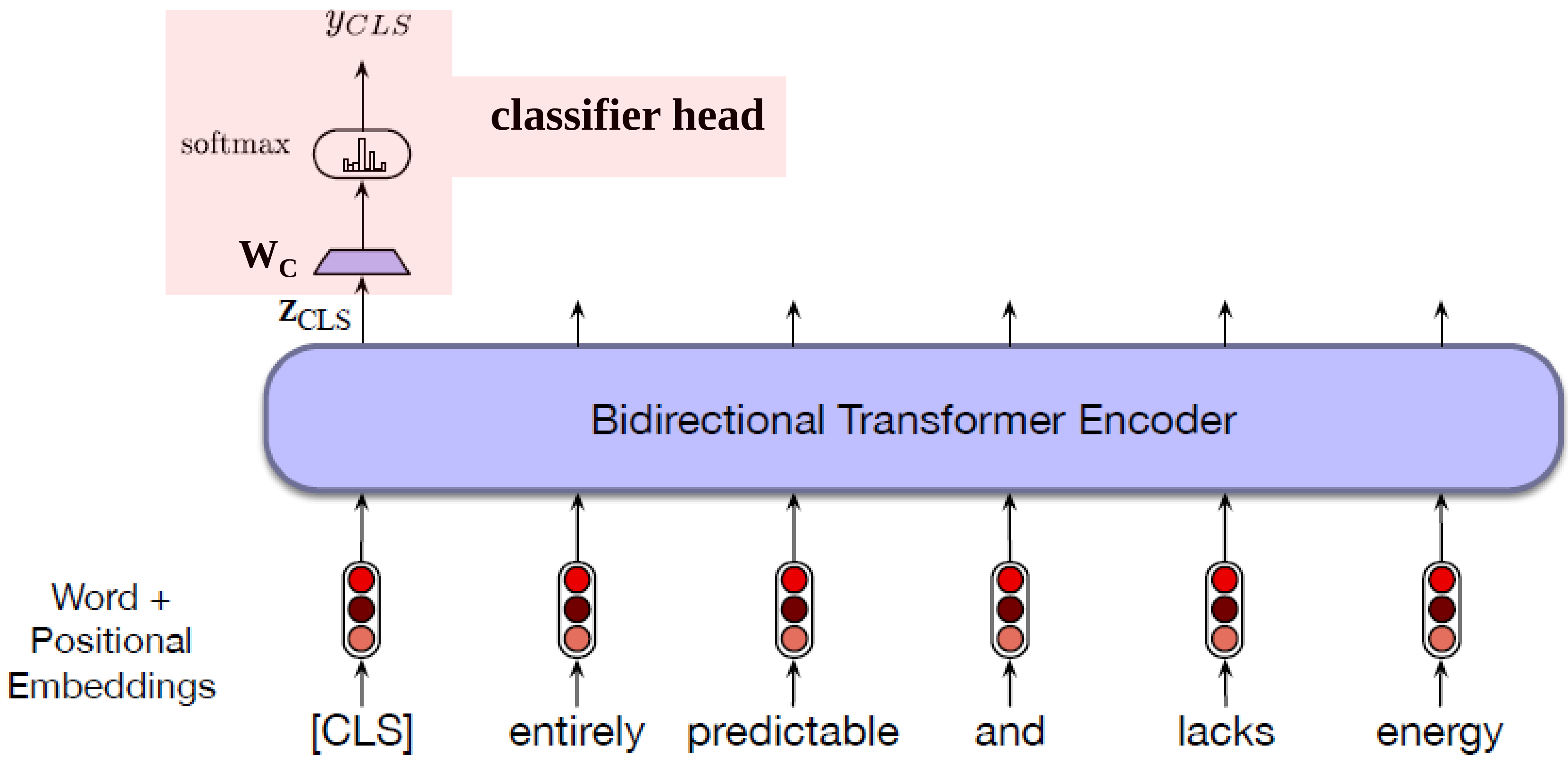
Sequence Classification

- **Sequence classification** applications often represent an input sequence with a single consolidated representation.
- In transformer models, an ***additional vector*** is added to the model to stand for the entire sentence sequence.
  - This vector is called the ***sentence embedding*** since it refers to the entire sequence.
  - In BERT, 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 a ***classifier head***, a logistic regression or neural network classifier that makes the relevant decision.

Fine-Tuning Language Models

Sequence Classification

- A simple approach to ***fine-tuning a classifier*** involves learning a set of weights $\mathbf{W_C}$ to map output vector $\mathbf{Z_{CLS}}$ for token **[CLS]** to a set of scores over the possible classes.

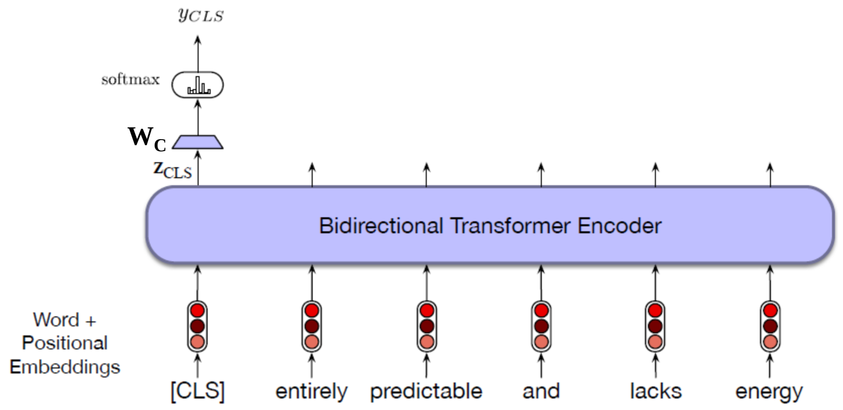


Fine-Tuning Language Models

Sequence Classification

- Assuming a three-way sentiment classification task (positive, negative, neutral) and dimensionality $\mathbf{d_h}$ for the size of the language model hidden layers gives $W_C \in R^{3\times dh}$.
- Classification of unseen documents proceeds by passing the input text through the pretrained language model to generate $\mathbf{Z_{CLS}}$, multiplying it by $\mathbf{W_C}$, and finally passing the resulting vector through a softmax.

$$\mathbf{y} = \text{softmax}(\mathbf{W_C z_{CLS}})$$

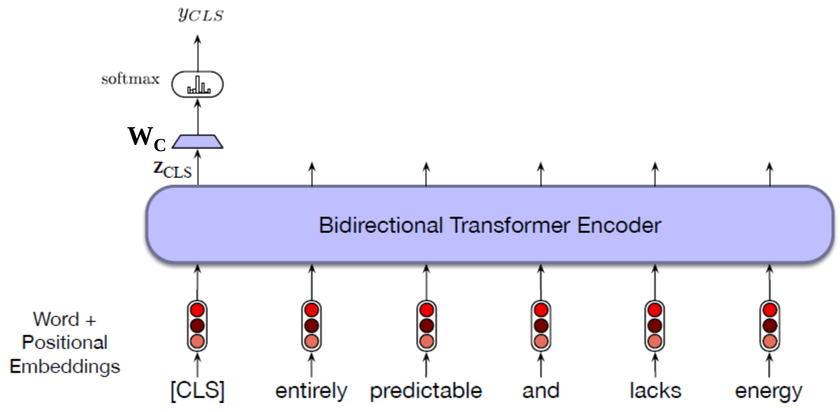


- **Finetuning** the values in $\mathbf{W_C}$ requires supervised training data consisting of input sequences labeled with the appropriate class.
  - Training proceeds in the usual way: cross-entropy loss between the softmax output and the correct answer is used to drive the learning that produce $\mathbf{W_C}$.

Fine-Tuning Language Models

Sequence Classification

- Normally, **finetuning** learns the values in $\mathbf{W_C}$.



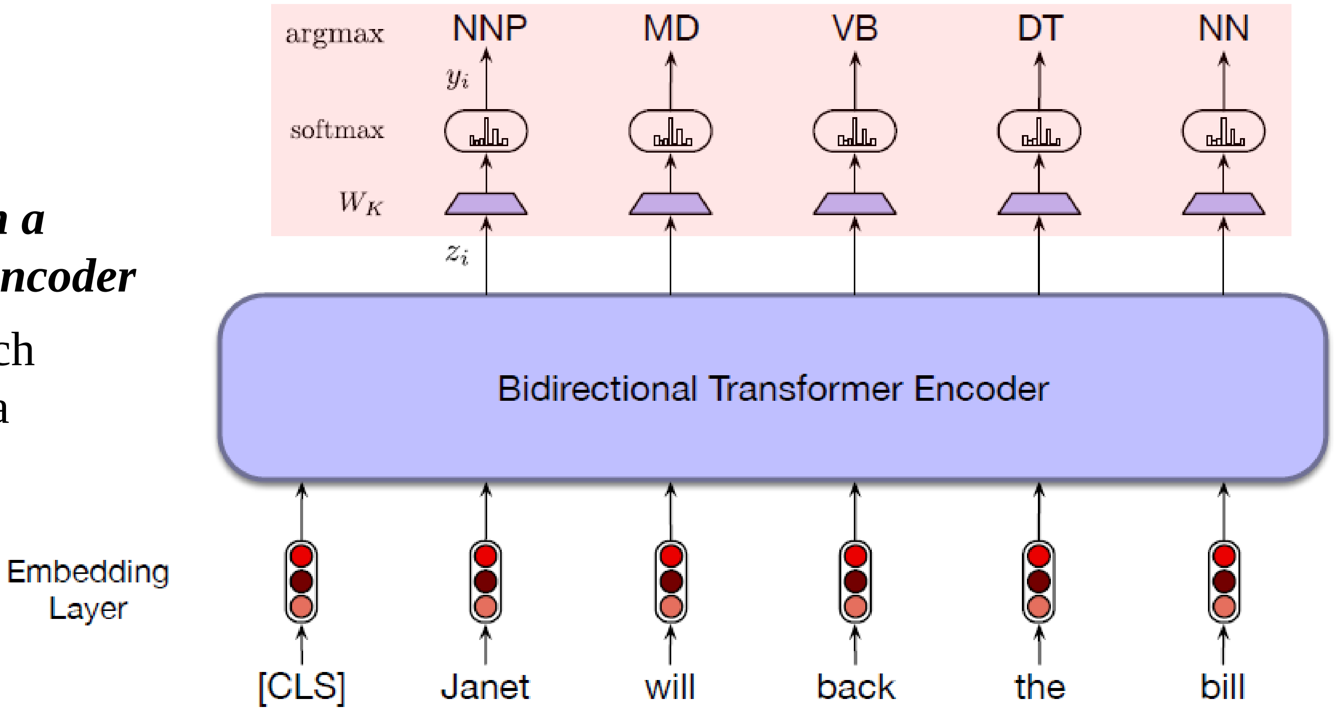
- The loss can be used to not only learn the weights of the classifier, but also to update the weights for the pretrained language model itself.
  - In practice, reasonable classification performance is typically achieved with only minimal changes to the language model parameters, often limited to updates over the final few layers of the transformer.

Fine-Tuning Language Models
Sequence Labelling

- **Sequence labelling** tasks, such as part-of-speech tagging (or BIO-based named entity recognition), follow the same basic classification approach.
- The final output vector corresponding to each input token is passed to a classifier that produces a softmax distribution over the possible set of tags.

Sequence labeling for part-of-speech tagging with a bidirectional transformer encoder

- The output vector for each input token is passed to a simple k-way classifier.

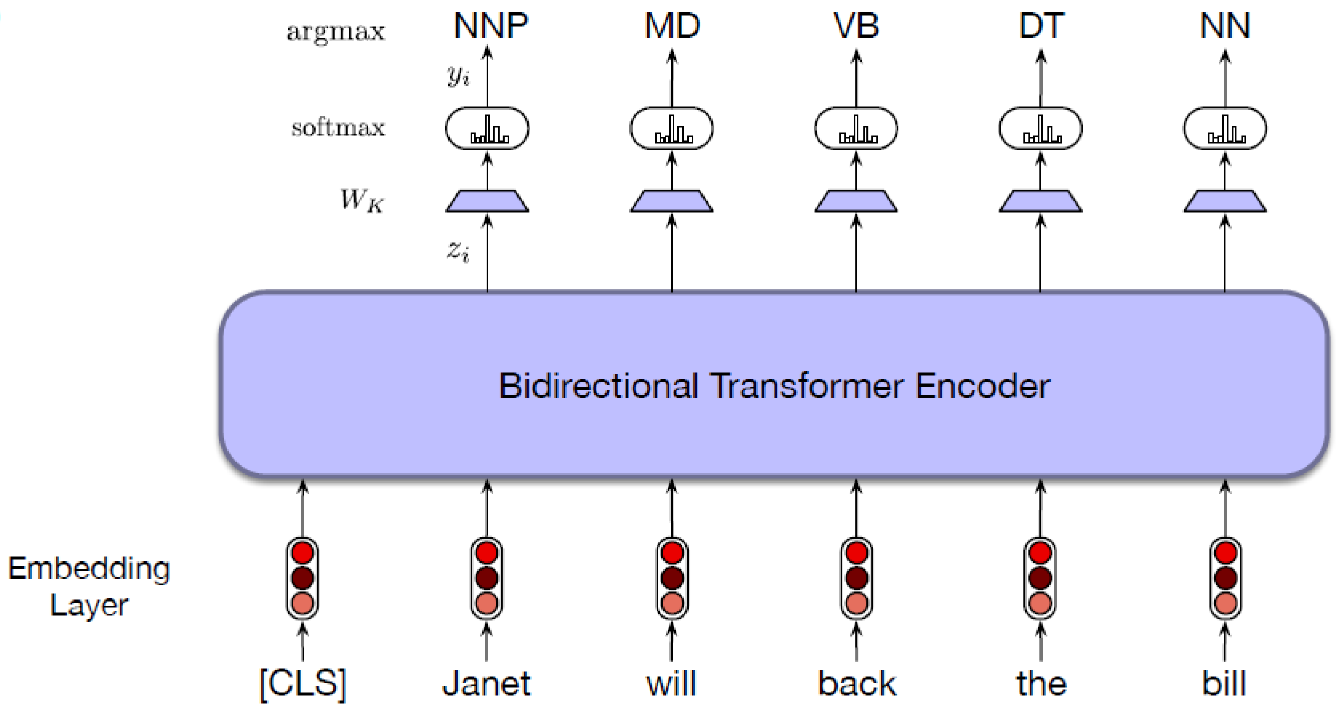


Fine-Tuning Language Models

Sequence Labelling

- Assuming a simple classifier consisting of a single feedforward layer followed by a softmax, the set of weights to be learned for this **additional layer** is $W_K \in R^{k \times dh}$, where k is the number of possible tags for the task.
- A greedy approach, where the argmax tag for each token is taken as a likely answer, can be used to generate the final output tag sequence.

$$\mathbf{y_i} = \text{softmax}(\mathbf{W_K z_i})$$
$$\mathbf{t_i} = \text{argmax}_k(\mathbf{y}_i)$$



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.

- Pretrained language models can be **fine-tuned** for specific applications by adding lightweight classifier layers on top of the outputs of the pretrained model.