Transfer Learning with Pretrained Language Models and Contextual Embeddings

Reference:
- D. Jurafsky and J. Martin, “Speech and Language Processing”

Introduction

- Recall that word2vec or GloVe learned a single vector embedding for each unique word.
- Contextual embeddings such as BERT or GPT represent each word by a different vector each time it appears in a different context.
- Pretraining - the process of learning some sort of representation of meaning for words or sentences by processing very large amounts of text.
- We'll call these pretrained models *pretrained language models*, since they can take the form of the transformer language models.
- Fine-tuning - the process of taking the representations from these pretrained models, and further training the model, often via an added neural net classifier, to perform some downstream task like QA.

Introduction

- The pretraining phase learns a language model that instantiates a rich representations of word meaning
  - 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
- Transfer learning - the method of acquiring knowledge from one task or domain, and then applying it (transferring it) to solve a new task.

Introduction

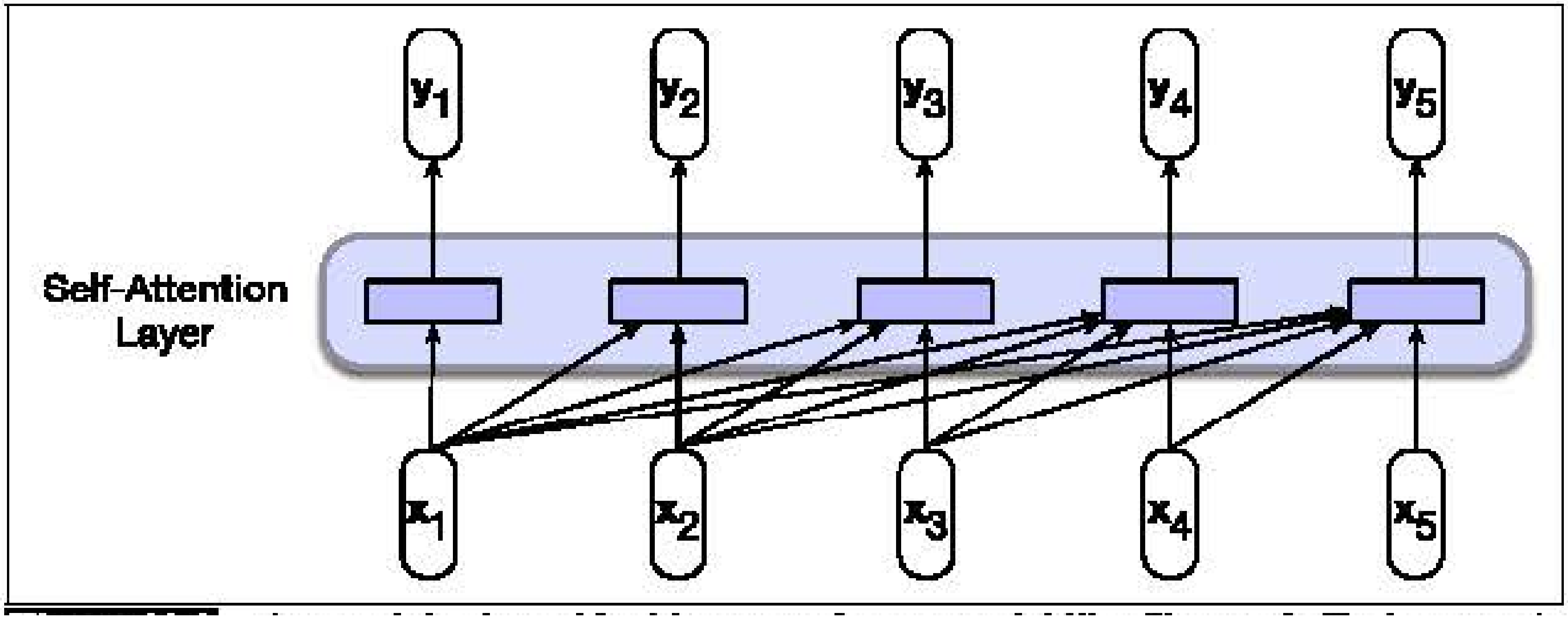
- There are two common paradigms for pretrained language models.
  - One is the causal or left-to-right transformer model we introduced before.
  - A second paradigm, called the bidirectional transformer encoder
- The method of masked language modeling, introduced with the BERT model that allows the model to see entire texts at a time, including both the right and left context.
- The contextual embeddings from these pretrained language models can be used to transfer the knowledge embodied in these models to novel applications via fine-tuning.
  - Fine-tuned to tasks from parsing to question answering

Bidirectional Transformer Encoders

- We introduce the bidirectional transformer encoder that underlies models like BERT and its descendants like RoBERTa or SpanBERT.
- We explored causal (left-to-right) transformers that can serve as the basis for powerful language models.
    - However, when applied to sequence classification and labeling problems causal models have obvious shortcomings since they are based on an incremental, left-to-right processing of their inputs.
    - If we want to assign the correct named-entity tag to each word in a sentence, or other sophisticated linguistic labels, we'll want to be able to take into account information from the right context as we process each element.

Bidirectional Transformer Encoders

Fig. 1



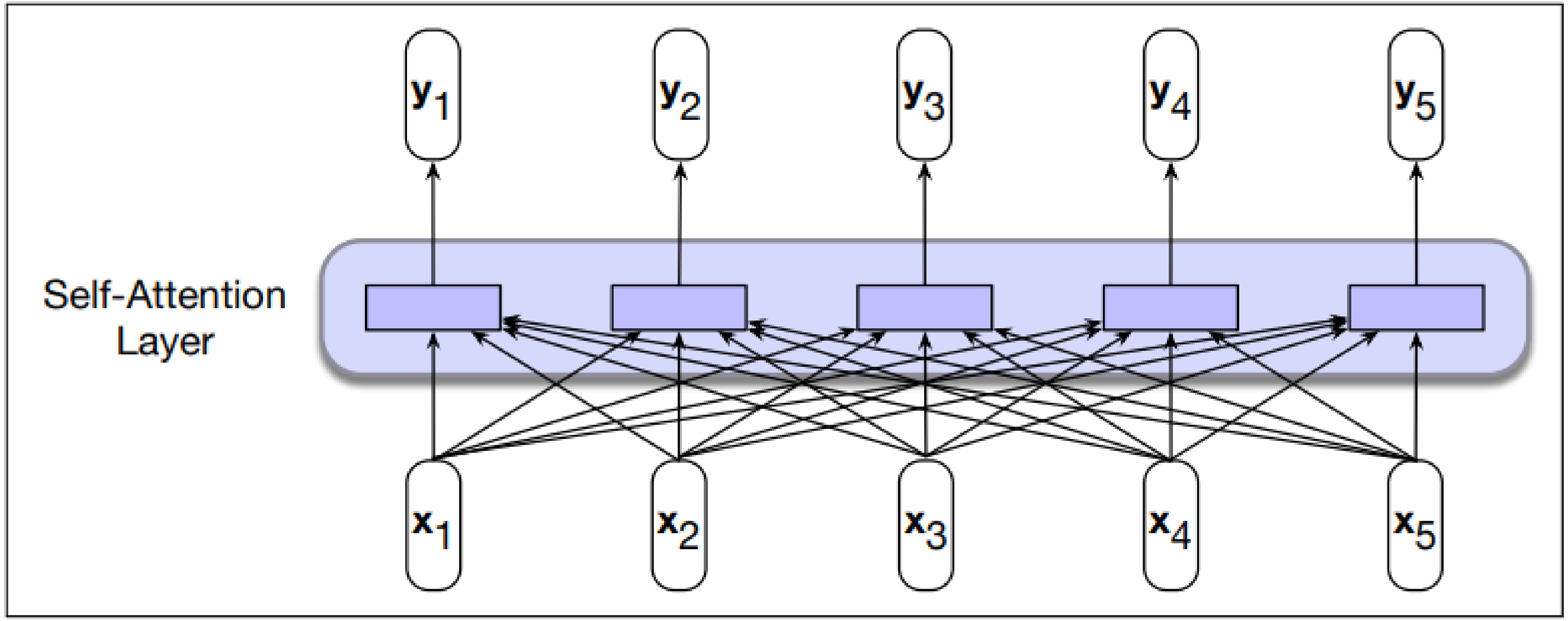
A causal, backward looking, transformer model. Each output is computed independently of the others using only information seen earlier in the context.

- As can be seen, the hidden state computation at each point in time is based solely on the current and earlier elements of the input,
  - ignoring potentially useful information located to the right of each tagging decision.

Bidirectional Transformer Encoders

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

Fig. 2



Information flow in a bidirectional self-attention model. In processing each element of the sequence, the model attends to all inputs, both before and after the current one.

Bidirectional Transformer Encoders

- The focus of bidirectional encoders is on computing contextualized representations of the tokens in an input sequence that are generally useful across a range of downstream applications.
- Therefore, bidirectional encoders use self-attention to map sequences of input embeddings $(x_1, \cdots, x_n)$ to sequences of output embeddings the same length $(y_1, \cdots, y_n)$,
  - where the output vectors have been contextualized using information from the entire input sequence.

Bidirectional Transformer Encoders

- This contextualization is accomplished through the use of the same self-attention mechanism used in causal models.
- As with these models, the first step is to generate a set of key, query and value embeddings for each element of the input vector x through the use of learned weight matrices W^Q, W^K, and W^V.
- These weights project each input vector x_i into its specific role as a key, query, or value.

$$q_i = W^Q x_i;\ k_i = W^K x_i;\ v_i = W^V x_i$$

Bidirectional Transformer Encoders

- The output vector y_i corresponding to each input element x_i is a weighted sum of all the input value vectors v, as follows:

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

- The α weights are computed via a softmax over the comparison scores between every element of an input sequence considered as a query and every other element as a key, where the comparison scores are computed using dot products.

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

$$score_{ij} = q_i \cdot k_j$$

Bidirectional Transformer Encoders

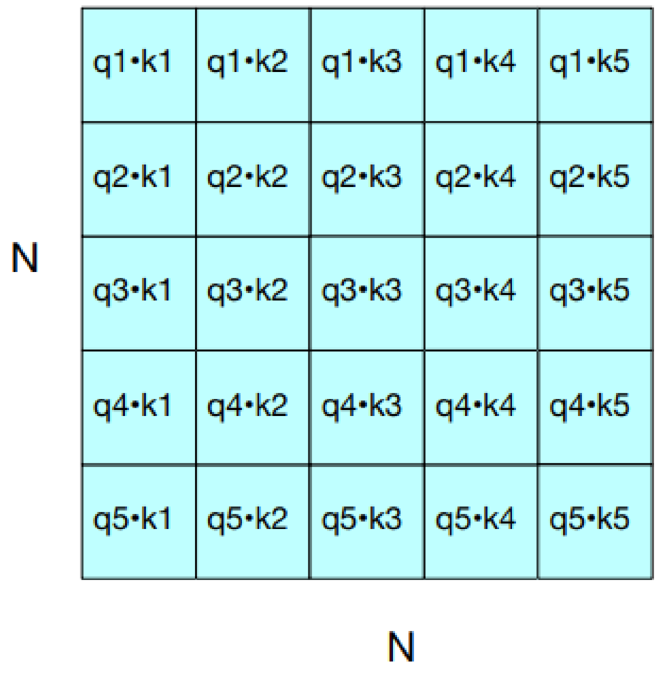
- Since each output vector, y_i, is computed independently, the processing of an entire sequence can be parallelized via matrix operations.
- The first step is to pack the input embeddings x_i into a matrix $X \in R^{N \times d_h}$
  - That is, each row of X is the embedding of one token of the input.
- We then multiply X by the key, query, and value weight matrices (all of dimensionality $d \times d$) to produce matrices $Q \in \mathbb{R}^{N \times d}$, $K \in \mathbb{R}^{N \times d}$, and $V \in \mathbb{R}^{N \times d}$,
  - containing all the key, query, and value vectors in a single step.

$$Q = XW^Q; K = XW^K; V = XW^V$$

Bidirectional Transformer Encoders

- Given these matrices we can compute all the requisite query-key comparisons simultaneously by multiplying Q and K^T in a single operation.
- The result for an input with length 5 is illustrated below.

Fig. 3



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

The $N \times N$ QK^T matrix showing the complete set of $q_i \cdot k_j$ comparisons.

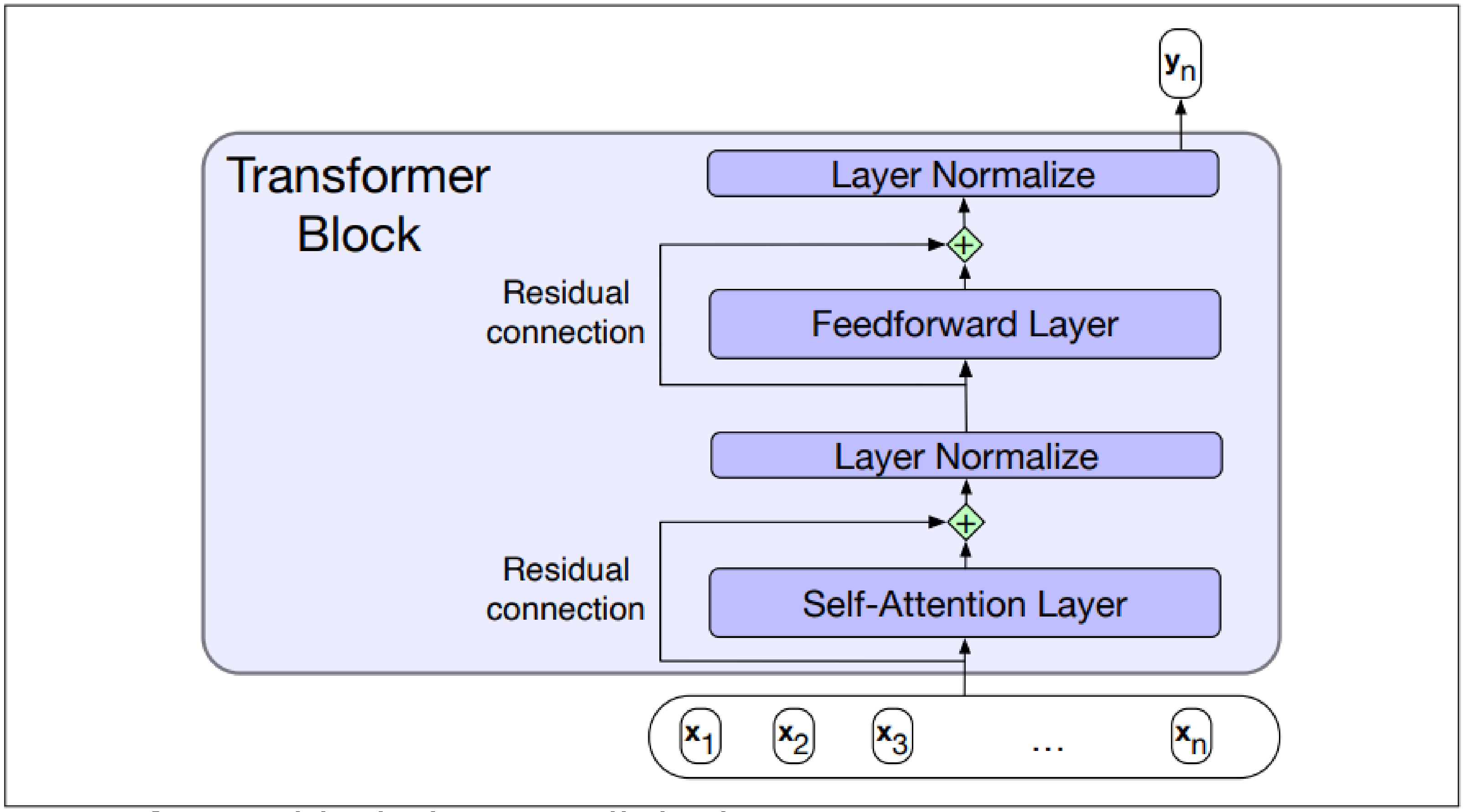
Bidirectional Transformer Encoders

- As shown in the previous figure, the full set of self-attention scores represented by QK^T constitute an all-pairs comparison between the keys and queries for each element of the input.
- In the case of causal language models we saw before, we masked the upper triangular portion of this matrix to eliminate information about
- With bidirectional encoders we simply skip the mask, allowing the model to contextualize each token using *information from the entire input*.
- Beyond this simple change, all of the other elements of the transformer architecture remain the same for bidirectional encoder models.

Bidirectional Transformer Encoders

- Inputs to the model are segmented using subword tokenization and are combined with positional embeddings before being passed through a series of standard transformer blocks consisting of self-attention and feedforward layers augmented with residual connections and layer normalization.

Fig. 4



A transformer block showing all the layers.

Bidirectional Transformer Encoders

- To make this more concrete, the original bidirectional transformer encoder model, BERT, consisted of the following:
  - A subword vocabulary consisting of 30,000 tokens generated using the WordPiece algorithm,
  - 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.

Bidirectional Transformer Encoders

- The use of WordPiece (one of the large family of subword tokenization algorithms that includes the BPE algorithm) means that BERT and its descendants are based on subword tokens rather than words.
- Every input sentence first has to be tokenized, and then all further processing takes place on subword tokens rather than words.
- As shown later, this will require that for some NLP tasks that require notions of words (like named entity tagging, or parsing), we will occasionally need to map subwords back to words.

Byte-Pair Encoding for Tokenization

- To deal with this unknown word problem, modern tokenizers often automatically induce sets of tokens that include tokens smaller than words, called subwords.
- Subwords can be arbitrary substrings, or they can be meaning-bearing units like the morphemes -est or -er.
- Most tokens are words, but some tokens are frequently occurring morphemes or other subwords like -er.
- Every unseen word like lower can thus be represented by some sequence of known subword units, such as low and er, or even as a sequence of individual letters if necessary.

Byte-Pair Encoding for Tokenization

- Most tokenization schemes have two parts: a token learner, and a token segmenter.
- The token learner takes a raw training corpus (sometimes roughly pre-separated into words, for example by whitespace) and induces a vocabulary, a set of tokens.
- The token segmenter takes a raw test sentence and segments it into the tokens in the vocabulary.
- Three algorithms are widely used: byte-pair encoding (BPE), unigram language modeling, and WordPiece.

Byte-Pair Encoding for Tokenization

- The BPE token learner begins with a vocabulary that is just the set of all individual characters.
- It then examines the training corpus, chooses the two symbols that are most frequently adjacent (say 'A', 'B'), adds a new merged symbol 'AB' to the vocabulary, and replaces every adjacent 'A' 'B' in the corpus with the new 'AB'.
- It continues to count and merge, creating new longer and longer character strings, until k merges have been done creating k novel tokens
    - k is thus a parameter of the algorithm.
- The resulting vocabulary consists of the original set of characters plus k new symbols.

Byte-Pair Encoding for Tokenization

- The algorithm is usually run inside words (not merging across word boundaries), so the input corpus is first white-space-separated to give a set of strings, each corresponding to the characters of a word, plus a special end-of-word symbol _, and its counts.
- Let’s see its operation on the following tiny input corpus of 18 word tokens with counts for each word (the word low appears 5 times, the word newer 6 times, and so on), which would have a starting vocabulary of 11 letters:

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _	
6 n e w e r _	
3 w i d e r _	
2 n e w _	

Byte-Pair Encoding for Tokenization

- The BPE algorithm first counts all pairs of adjacent symbols: the most frequent is the pair e r because it occurs in newer (frequency of 6) and wider (frequency of 3) for a total of 9 occurrences.
- We then merge these symbols, treating er as one symbol, and count again:

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er,
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

Byte-Pair Encoding for Tokenization

- Now the most frequent pair is er _, which we merge; our system has learned that there should be a token for word-final er, represented as er_ :

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

- Next n e (total count of 8) get merged to ne:

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2	l o w e s t _	
6	ne w er _	
3	w i d er_	
2	ne w _	

Byte-Pair Encoding for Tokenization

- If we continue, the next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Byte-Pair Encoding for Tokenization

- Once we’ve learned our vocabulary, the token parser is used to tokenize a test sentence.
- The token parser just runs on the test data the merges we have learned from the training data, greedily, in the order we learned them.
- So first we segment each test sentence word into characters. Then we apply the first rule: replace every instance of e r in the test corpus with er.
- Then the second rule: replace every instance of er _ in the test corpus with er_ , and so on.
- By the end, if the test corpus contained the word n e w e r _, it would be tokenized as a full word.
- But a new (unknown) word like l o w e r _ would be merged into the two tokens low er_ .

Byte-Pair Encoding for Tokenization

- Of course in real algorithms BPE is run with many thousands of merges on a very large input corpus.
- The result is that most words will be represented as full symbols, and only the very rare words (and unknown words) will have to be represented by their parts.

Wordpiece Algorithm for Tokenization

1. Initialize the wordpiece lexicon with characters (for example a subset of Unicode characters, collapsing all the remaining characters to a special unknown character token).

2. Repeat until there are V wordpieces:

    (a) Train an n-gram language model on the training corpus, using the current set of wordpieces.

    (b) Consider the set of possible new wordpieces made by concatenating two wordpieces from the current lexicon. Choose the one new wordpiece that most increases the language model probability of the training corpus.

Bidirectional Transformer Encoders

- A fundamental issue with transformers is that the size of the input layer dictates the complexity of model.
- Both the time and memory requirements in a transformer grow quadratically with the length of the input.
- Consequentially, it is necessary to set a fixed input length that is long enough to provide sufficient context for the model to function and yet still be computationally tractable.
    - For BERT, a fixed input size of 512 subword tokens was used.

Training Bidirectional Encoders

- Previously, we trained causal transformer language models by making them iteratively predict the next word in a text.
- The guess-the-next-word language modeling task is not suitable.
- Fortunately, the traditional learning objective suggests an approach that can be used to train bidirectional encoders.
- Cloze task - instead of trying to predict the next word, the model learns to perform a fill-in-the-blank task.

Training Bidirectional Encoders

- Instead of predicting which words are likely to come next in this example:

  Please turn your homework _____ .

  we're asked to predict a missing item given the rest of the sentence.

  Please turn _____ homework in.

- That is, given an input sequence with one or more elements missing, the learning task is to predict the missing elements.
  - During training the model is deprived of one or more elements of an input sequence and must generate a probability distribution over the vocabulary for each of the missing items.
  - We then use the cross-entropy loss from each of the model's predictions to drive the learning process.

Training Bidirectional Encoders

- This approach can be generalized to any of a variety of methods that corrupt the training input and then asks the model to recover the original input.
- Examples of the kinds of manipulations that have been used include masks, substitutions, reorderings, deletions, and extraneous insertions into the training text

Training Bidirectional Encoders – Masking Words

- Masked Language Modeling (MLM) - the original approach to training bidirectional encoders.
- MLM uses unannotated text from a large corpus, as with the language model training methods we’ve already seen.
- 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.

Training Bidirectional Encoders – Masking Words

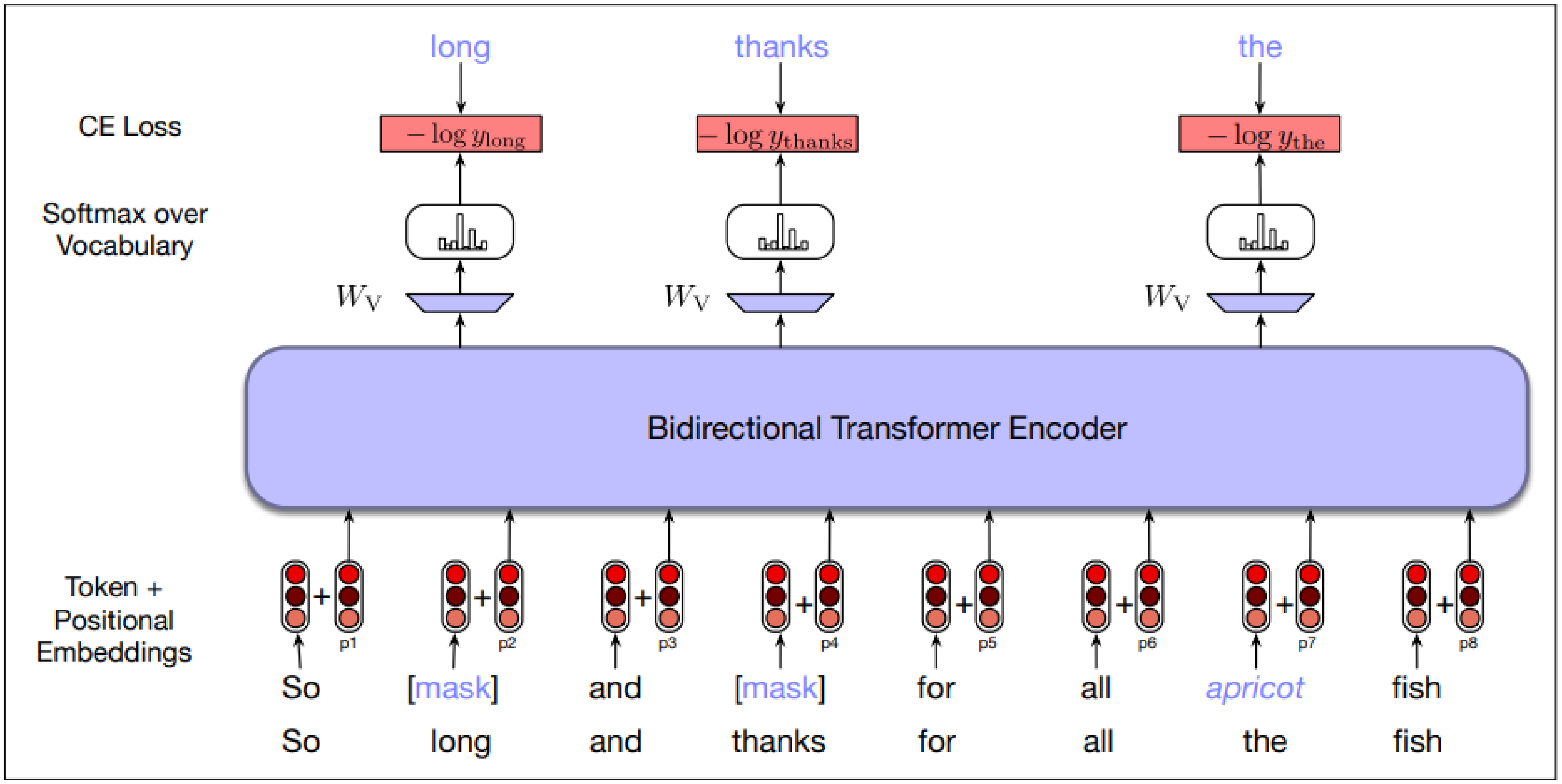
- In BERT, 15% of the input tokens in a training sequence are sampled for learning. Of these,
  - 80% are replaced with [MASK],
  - 10% are replaced with randomly selected tokens, and
  - the remaining 10% are left unchanged.
- The MLM training objective is to predict the original inputs for each of the masked tokens using a bidirectional encoder of the kind described in the last section.
- The cross-entropy loss from these predictions drives the training process for all the parameters in the model.
- All of the input tokens play a role in the self-attention process, but only the sampled tokens are used for learning.

Training Bidirectional Encoders – Masking Words

- Specifically, the 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 of 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 – Masking Words

Fig. 5



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. (Although the input is displayed as words rather than subword tokens in examples, keep in mind that BERT and similar models actually use subword tokens instead.)

Training Bidirectional Encoders – Masking Words

- In the example shown, *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*.
- The resulting embeddings are passed through a stack of bidirectional transformer blocks.
- To produce a probability distribution over the vocabulary for each of the masked tokens, the output vector from the final transformer layer for each of the masked tokens is multiplied by a learned set of classification weights $W_V \in \mathbb{R}^{|V| \times d_h}$ and then through a softmax to yield the required predictions over the vocabulary.

$$y_i = softmax(W_V h_i)$$

Training Bidirectional Encoders – Masking Words

- 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, as shown in the example.
- 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 (or batch of sequences).

Training Bidirectional Encoders – Masking Spans

- For many NLP applications, the natural unit of interest may be larger than a single word (or token).
- Question answering, syntactic parsing, coreference and semantic role labeling applications all involve the identification and classification of constituents, or phrases.
- This suggests that a span-oriented masked learning objective might provide improved performance on such tasks.

Training Bidirectional Encoders – Masking Spans

- A span is a contiguous sequence of one or more words selected from a training text, prior to subword tokenization.
- In span-based masking, a set of randomly selected spans from a training sequence are chosen.
- In the SpanBERT work that originated this technique, a span length is first chosen by sampling from a geometric distribution that is biased towards shorter spans an with upper bound of 10.
- Given this span length, a starting location consistent with the desired span length and the length of the input is sampled uniformly.

Training Bidirectional Encoders – Masking Spans

- Once a span is chosen for masking, all the words within the span are substituted according to the same regime used in BERT:
  - 80% of the time the span elements are substituted with the [MASK] token,
  - 10% of the time they are replaced by randomly sampled words from the vocabulary, and
  - 10% of the time they are left as is.
- This substitution process is done at the span level
  - all the tokens in a given span are substituted using the same method.
- As with BERT, the total token substitution is limited to 15% of the training sequence input.
- Having selected and masked the training span, the input is passed through the standard transformer architecture to generate contextualized representations of the input tokens.

Training Bidirectional Encoders – Masking Spans

- 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 words of a span, the words immediately preceding and following the span, or some combination of them.
- Span Boundary Objective (SBO) - a boundary oriented component with which the SpanBERT learning objective augments the MLM objective.
    - The SBO relies on a model’s ability to predict the words within a masked span from the words immediately preceding and following it.

Training Bidirectional Encoders – Masking Spans

- This prediction is made using the output vectors associated with the words that immediately precede and follow the span being masked, along with positional embedding that signals which word in the span is being predicted:

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

$$L_{SBO}(x) \; = -logP(x|x_s, x_e, p_x)$$

  where s denotes the position of the word before the span and e denotes the word after the end.

- The prediction for a given position *i* within the span is produced by concatenating the output embeddings for words s and e span boundary vectors with a positional embedding for position i and passing the result through a 2-layer feedforward network.

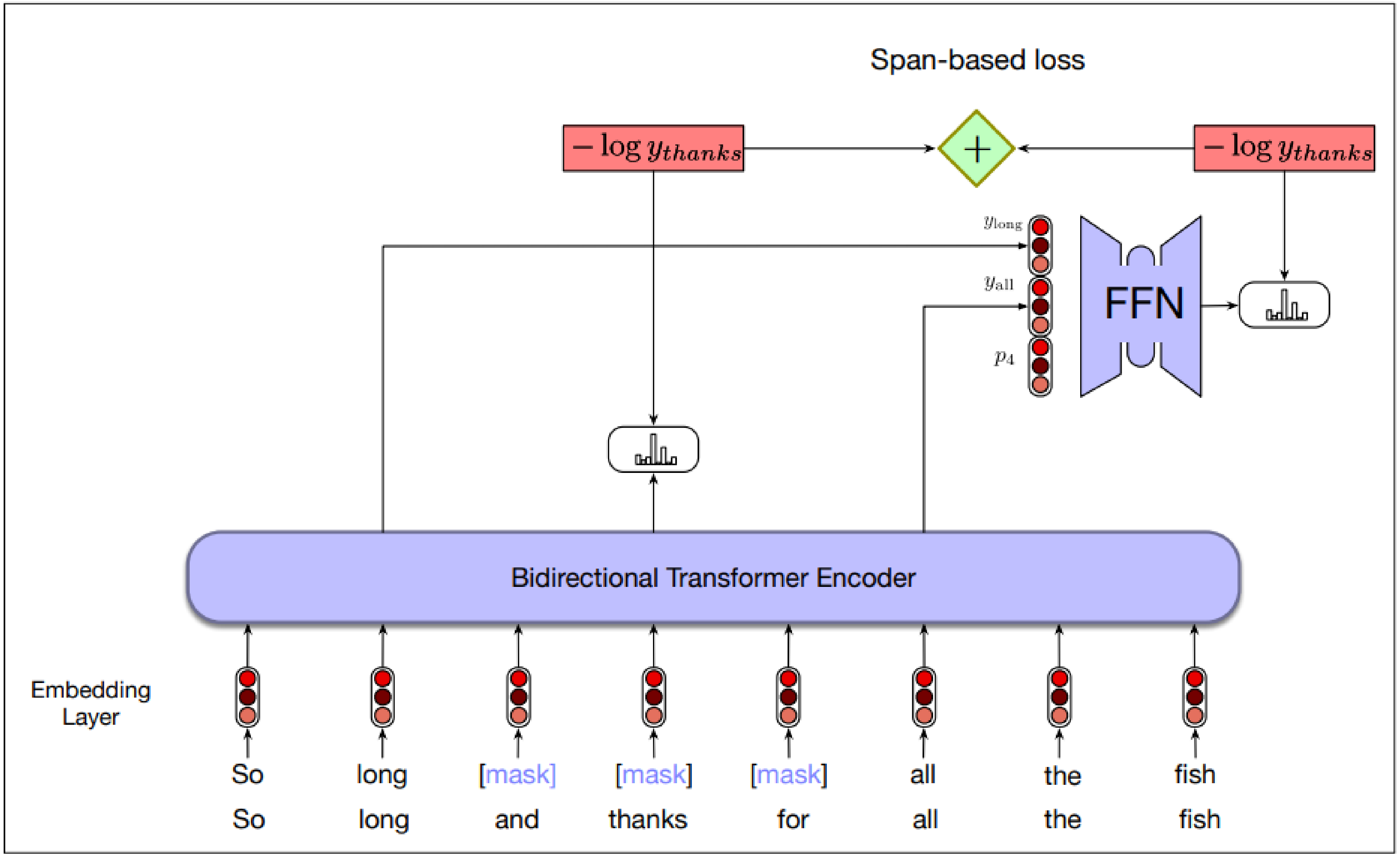
$$s = FFNN([y_{s-1}; y_{e+1}; p_{i-s+1}])$$

$$z = softmax(Es)$$

- The final loss is sum of the BERT MLM loss and the SBO loss.

Training Bidirectional Encoders – Masking Spans

Fig. 6


Span-based loss
$-\log y_{thanks}$
$-\log y_{thanks}$
y_{long}
y_{all}
p_4
FFN
Bidirectional Transformer Encoder
Embedding Layer
So long [mask] [mask] [mask] all the fish
So long and thanks for all the fish

Span-based language model training. In this example, a span of length 3 is selected for training and all of the words in the span are masked. The figure illustrates the loss computed for word *thanks*; the loss for the entire span is based on the loss for all three of the words in the span.

Training Bidirectional Encoders – Masking Spans

- The above figure illustrates span-based language model training with an earlier example.
- The span selected is *and thanks for* which spans from position 3 to 5.
- The total loss associated with the masked token *thanks* is the sum of the cross-entropy loss generated from the prediction of *thanks* from the output y_4, plus the cross-entropy loss from the prediction of *thanks* from the output vectors for y_2, y_6 and the embedding for position 4 in the span.

Training Bidirectional Encoders – Next Sentence Prediction

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

Training Bidirectional Encoders – Next Sentence Prediction

- In this 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.

Training Bidirectional Encoders – Next Sentence Prediction

- To facilitate NSP training, BERT introduces two new tokens to the input representation (tokens that will prove useful for fine-tuning as well).
- 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.
- Finally, embeddings representing the first and second segments of the input are added to the word and positional embeddings to allow the model to more easily distinguish the input sentences.

Training Bidirectional Encoders – Next Sentence Prediction

- 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 $W_{NSP} \in \mathbb{R}^{2 \times d_h}$ is used to produce a two-class prediction from the raw [CLS] vector.

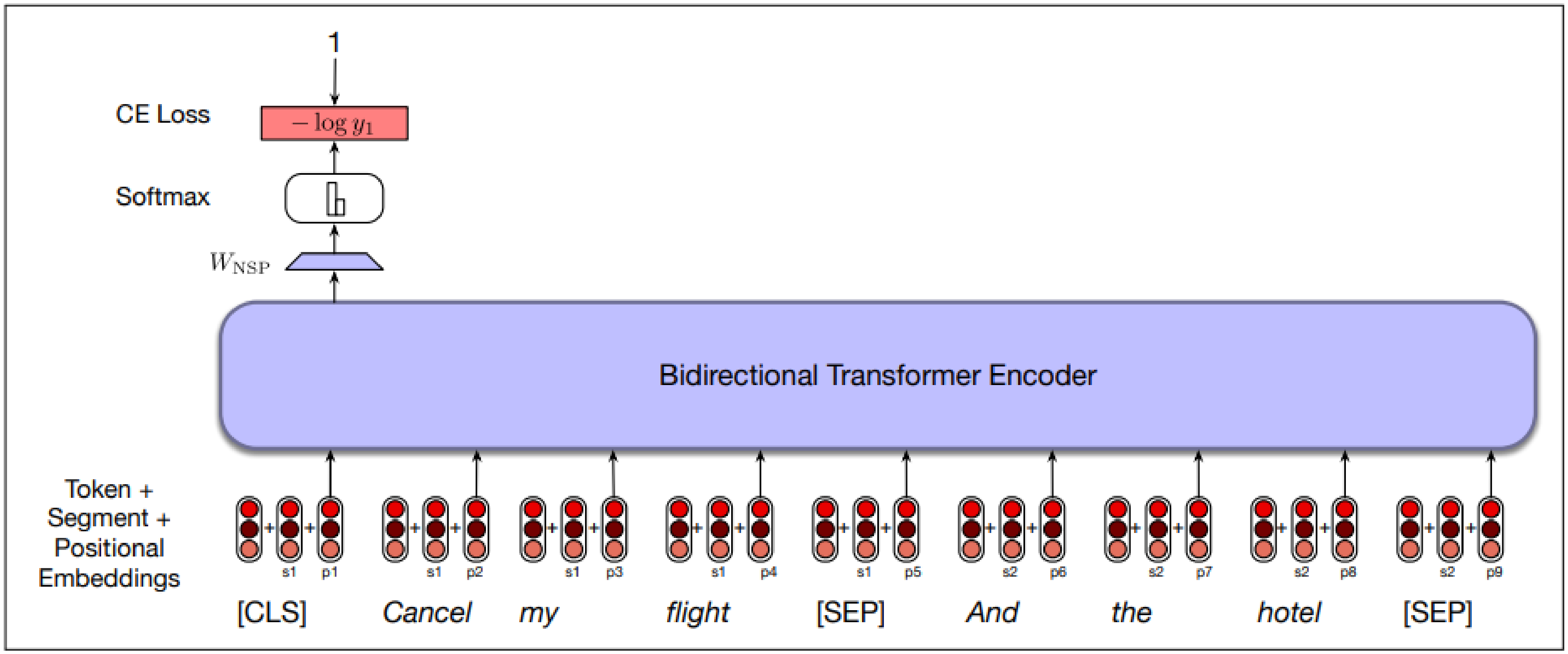
$$y_i = softmax(W_{NSP} h_i)$$

- Cross entropy is used to compute the NSP loss for each sentence pair presented to the model.

Training Bidirectional Encoders – Next Sentence Prediction

- The overall NSP training setup is illustrated below.
- In BERT, the NSP loss was used in conjunction with the MLM training objective to form final loss.

Fig. 7


1
CE Loss
$-\log y_1$
Softmax
W_{NSP}
Bidirectional Transformer Encoder
Token + Segment + Positional Embeddings
s1 p1 s1 p2 s1 p3 s1 p4 s1 p5 s2 p6 s2 p7 s2 p8 s2 p9
[CLS] Cancel my flight [SEP] And the hotel [SEP]

An example of the NSP loss calculation.

Training Bidirectional Encoders – Training Regimes

- The corpus used in training BERT and other early transformer-based language models consisted of
  - an 800M word corpus of book texts called BooksCorpus
  - a 2.5B word corpus derived from the English Wikipedia

  for a combined size of 3.3 Billion words.
- State-of-the-art models employ corpora that are orders of magnitude larger than these early efforts.
  - The BooksCorpus is no longer used.

Training Bidirectional Encoders – Training Regimes

- To train the original BERT models
  - Pairs of sentences were selected from the training corpus according to the NSP 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.
  - Approximately 40 passes (epochs) over the training data was required for the model to converge.
- The result of this pretraining process consists of both
  - Learned word embeddings
  - All the parameters of the bidirectional encoder that are used to produce contextual embeddings for novel input

Training Bidirectional Encoders – Contextual Embeddings

- Contextual embeddings - Given a pretrained language model and a novel input sentence, we can think of the output of the model as constituting contextual embeddings for each token in the contextual input.
- These contextual embeddings can be used as a contextual representation of the meaning of the input token for any task requiring the meaning of word.

Training Bidirectional Encoders – Contextual Embeddings

- Contextual embeddings are thus vectors representing some aspect of the meaning of a token in context.
  - For example, given a sequence of input tokens $x_1, \dots, x_n$, we can use the output vector y_i from the final layer of the model as a representation of the meaning of token x_i in the context of sentence $x_1, \dots, x_n$.
  - Or instead of just using the vector y_i from the final layer of the model, it's common to compute a representation for x_i by averaging the output tokens y_i from each of the last four layers of the model.

Transfer Learning through Fine-Tuning

- Just as we used static embeddings like word2vec to represent the meaning of words, we can use contextual embeddings as representations of word meanings in context for any task that might require a model of word meaning.
- Static embeddings represent the meaning of word *types* (vocabulary entries)
- Contextual embeddings represent the meaning of word *tokens*: instances of a particular word type in a particular context.
- Contextual embeddings can thus be used for tasks like measuring the semantic similarity of two words in context, and are useful in linguistic tasks that require models of word meaning.
- We'll see the most common use of these representations: as embeddings of word or even entire sentences that are the inputs to classifiers in the fine-tuning process for downstream NLP applications.

Transfer Learning through Fine-Tuning

- The power of pretrained language models lies in their ability to extract generalizations from large amounts of text
  - generalizations useful for myriad downstream applications.
- Fine-tuning - a process through which we create interfaces from these models to downstream applications in order to make practical use of these generalizations.
- Fine-tuning facilitates the creation of applications on top of pretrained models through the addition of a small set of application-specific parameters.
- The fine-tuning process consists of using labeled data from the application to train these application-specific parameters.
- Typically, this training will either freeze or make only minimal adjustments to the pretrained language model parameters.
- We'll introduce fine-tuning methods for the most common applications including sequence classification, sequence labeling, sentence-pair inference, and span-based operations.

Transfer Learning through Fine-Tuning – Sequence Classification

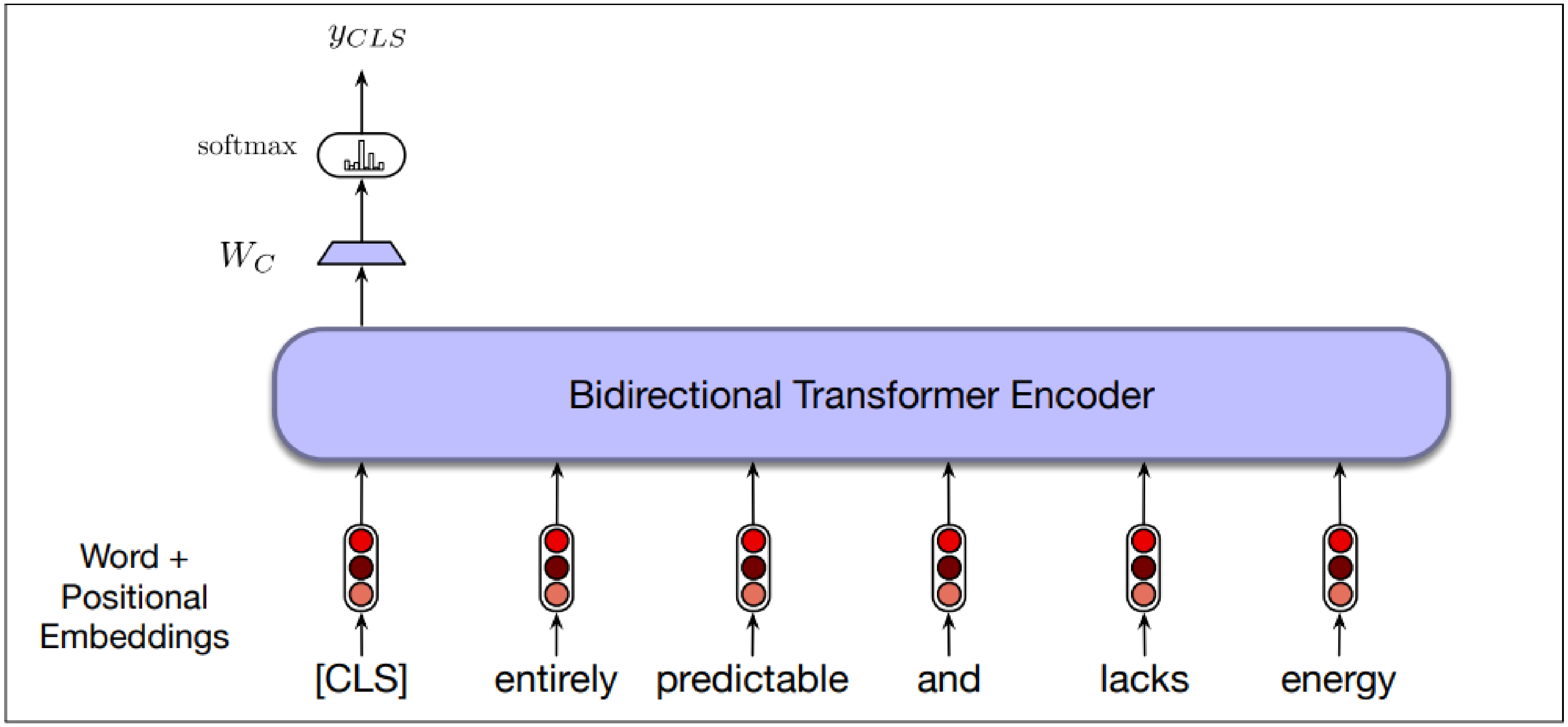
- Sequence classification applications often represent an input sequence with a single consolidated representation.
- With RNNs, we used the hidden layer associated with the final input element to stand for the entire sequence.
- With transformers, a similar approach is used.
- Sentence embedding - an additional vector added to the model to stand for the entire sequence.
    - although the term ‘sentence embedding’ is also used in other ways.

Transfer Learning through Fine-Tuning – Sequence Classification

- 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.
- Classifier head - a logistic regression or neural network classifier that makes the relevant decision.

Transfer Learning through Fine-Tuning – Sequence Classification

Fig. 8


y_{CLS}
softmax
W_C
Bidirectional Transformer Encoder
Word + Positional Embeddings
[CLS]
entirely
predictable
and
lacks
energy

Sequence classification with a bidirectional transformer encoder. The output vector for the [CLS] token serves as input to a simple classifier.

Transfer Learning through Fine-Tuning – Sequence Classification

- For sentiment classification, a simple approach to fine-tuning a classifier for this application involves learning a set of weights, W_C, to map the output vector for the [CLS] token, y_{CLS} to a set of scores over the possible sentiment classes.
- For example, a three-way sentiment classification task (positive, negative, neutral) and dimensionality d_h for the size of the language model hidden layers gives $W_C \in \mathbb{R}^{3\times d_h}$.
- Classification of unseen documents proceeds by passing the input text through the pretrained language model to generate y_{CLS}, multiplying it by W_C, and finally passing the resulting vector through a softmax.

$$y = softmax(W_C y_{CLS})$$

Transfer Learning through Fine-Tuning – Sequence Classification

- Finetuning the values in 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 produces W_C.
- A key difference from what we've seen earlier with neural classifiers is that this 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.

Transfer Learning through Fine-Tuning – Pair-Wise Sequence Classification

- As mentioned earlier, an important type of problem involves the classification of pairs of input sequences.
  - Practical applications include logical entailment, paraphrase detection and discourse analysis.
- Fine-tuning an application for one of these tasks proceeds just as with pretraining using the NSP objective.
- During fine-tuning, pairs of labeled sentences from the supervised training data are presented to the model.
  - As with sequence classification, the output vector associated with the prepended [CLS] token represents the model's view of the input pair.
  - As with NSP training, the two inputs are separated by a [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.

Transfer Learning through Fine-Tuning – Pair-Wise Sequence Classification

- As an example, let’s consider an entailment classification task with the Multi-Genre Natural Language Inference (MultiNLI) dataset.
- Natural language inference (NLI, also called recognizing textual entailment) - a task in which a model is presented with a pair of sentences and must classify the relationship between their meanings.
- In the MultiNLI corpus, pairs of sentences are given one of 3 labels: *entails*, *contradicts* and *neutral*.
- These labels describe a relationship between the meaning of the first sentence (the premise) and the meaning of the second sentence (the hypothesis).

Transfer Learning through Fine-Tuning – Pair-Wise Sequence Classification

- Here are representative examples of each class from the 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.

Transfer Learning through Fine-Tuning – Pair-Wise Sequence Classification

- A relationship of *contradicts* means that the premise contradicts the hypothesis; *entails* means that the premise entails the hypothesis; *neutral* means that neither is necessarily true.
  - The meaning of these labels is looser than strict logical entailment or contradiction indicating that a typical human reading the sentences would most likely interpret the meanings in this way.
- To fine-tune a classifier for the MultiNLI task, we pass the premise/hypothesis pairs through a bidirectional encoder as described before and use the output vector for the [CLS] token as the input to the classification head.
- As with ordinary sequence classification, this head provides the input to a three-way classifier that can be trained on the MultiNLI training corpus.

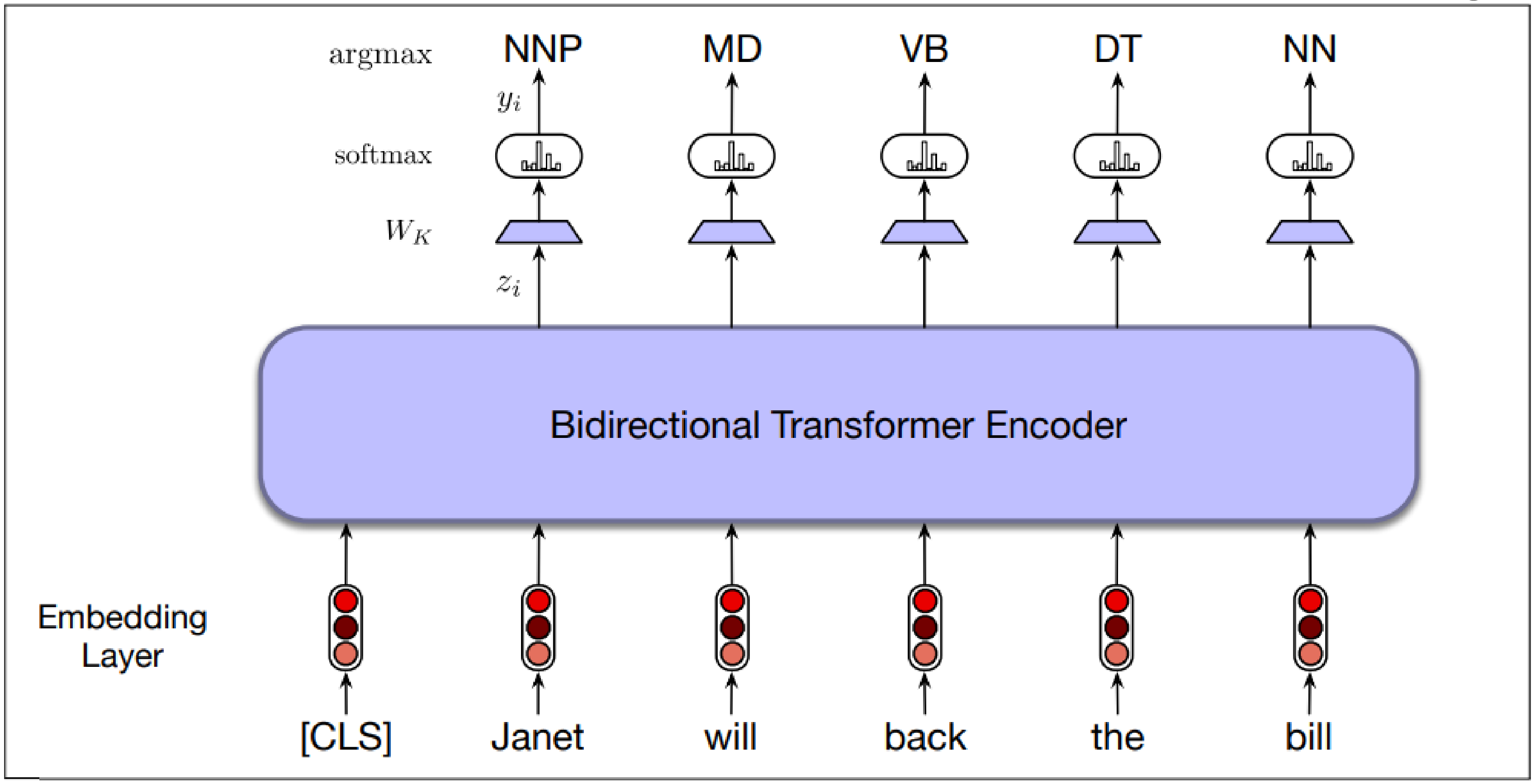
Transfer Learning through Fine-Tuning – Sequence Labelling

- 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.
- With 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 $W_K \in \mathbb{R}^{k \times d_h}$, where k is the number of possible tags for the task.
- As with RNNs, 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.

$$y_i = softmax(W_K z_i)$$
$$t_i = argmax_k(y_i)$$

Transfer Learning through Fine-Tuning – Sequence Labelling

Fig. 9



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.

- Alternatively, the distribution over labels provided by the softmax for each input token can be passed to a conditional random field (CRF) layer which can take global tag-level transitions into account.

Transfer Learning through Fine-Tuning – Sequence Labelling

- A complication with this approach arises from the use of subword tokenization such as WordPiece or Byte Pair Encoding.
- Supervised training data for tasks like named entity recognition (NER) is typically in the form of BIO tags associated with text segmented at the word level.
- For example the following sentence containing two named entities:

  $[_{\text{LOC}}$ **Mt. Sanitas**] is in $[_{\text{LOC}}$ **Sunshine Canyon**] .

  would have the following set of per-word BIO tags.

  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' '.'

Transfer Learning through Fine-Tuning – Sequence Labelling

- To deal with this misalignment, we need a way to assign BIO tags to subword tokens during training and a corresponding way to recover word-level tags from subwords during decoding.
- For training, we can just assign the gold-standard tag associated with each word to all of the subword tokens derived from it.
- For decoding, the simplest approach is to use the argmax BIO tag associated with the first subword token of a word.
    - Thus, in our example, the BIO tag assigned to "Mt" would be assigned to "Mt." and the tag assigned to "San" would be assigned to "Sanitas", effectively ignoring the information in the tags assigned to "." and "##itas".
- More complex approaches combine the distribution of tag probabilities across the subwords in an attempt to find an optimal word-level tag.

Transfer Learning through Fine-Tuning – Span-Based Applications

- Span-oriented applications operate in a middle ground between sequence level and token level tasks.
  - That is, in span-oriented applications the focus is on generating and operating with representations of contiguous sequences of tokens.
- Typical operations include identifying spans of interest, classifying spans according to some labeling scheme, and determining relations among discovered spans.
- Applications include named entity recognition, question answering, syntactic parsing, semantic role labeling and coreference resolution.

Transfer Learning through Fine-Tuning – Span-Based Applications

- Formally, given an input sequence x consisting of T tokens, $(x_1, x_2, \dots, x_T)$, a span is a contiguous sequence of tokens with start i and end j such that $1 \leq i \leq j \leq T$.
  - This formulation results in a total set of spans of size equal to $\frac{T(T-1)}{2}$.
- For practical purposes, span-based models often impose an application-specific length limit L, so the legal spans are limited to those where $j - i < L$.

Transfer Learning through Fine-Tuning – Span-Based Applications

- The first step in fine-tuning a pretrained language model for a span-based application using the contextualized input embeddings from the model to generate representations for all the spans in the input.
- Most schemes for representing spans make use of two primary components: representations of the span boundaries and summary representations of the contents of each span.
- To compute a unified span representation, we concatenate the boundary representations with the summary representation.

Transfer Learning through Fine-Tuning – Span-Based Applications

- In the simplest possible approach, we can use the contextual embeddings of the start and end tokens of a span as the boundaries, and the average of the output embeddings within the span as the summary representation.

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

$$spanRep_{ij} = [h_i; h_j; g_{i,j}]$$

- A weakness of this approach is that it doesn’t distinguish the use of a word’s embedding as the beginning of a span from its use as the end of one.

Transfer Learning through Fine-Tuning – Span-Based Applications

- Therefore, more elaborate schemes for representing the span boundaries involve learned representations for start and end points through the use of two distinct feedforward networks:

$$s_i = FFNN_{start}(h_i)$$

$$e_j = FFNN_{end}(h_j)$$

$$spanRep_{ij} = [s_i; e_j; g_{i,j}]$$

Transfer Learning through Fine-Tuning – Span-Based Applications

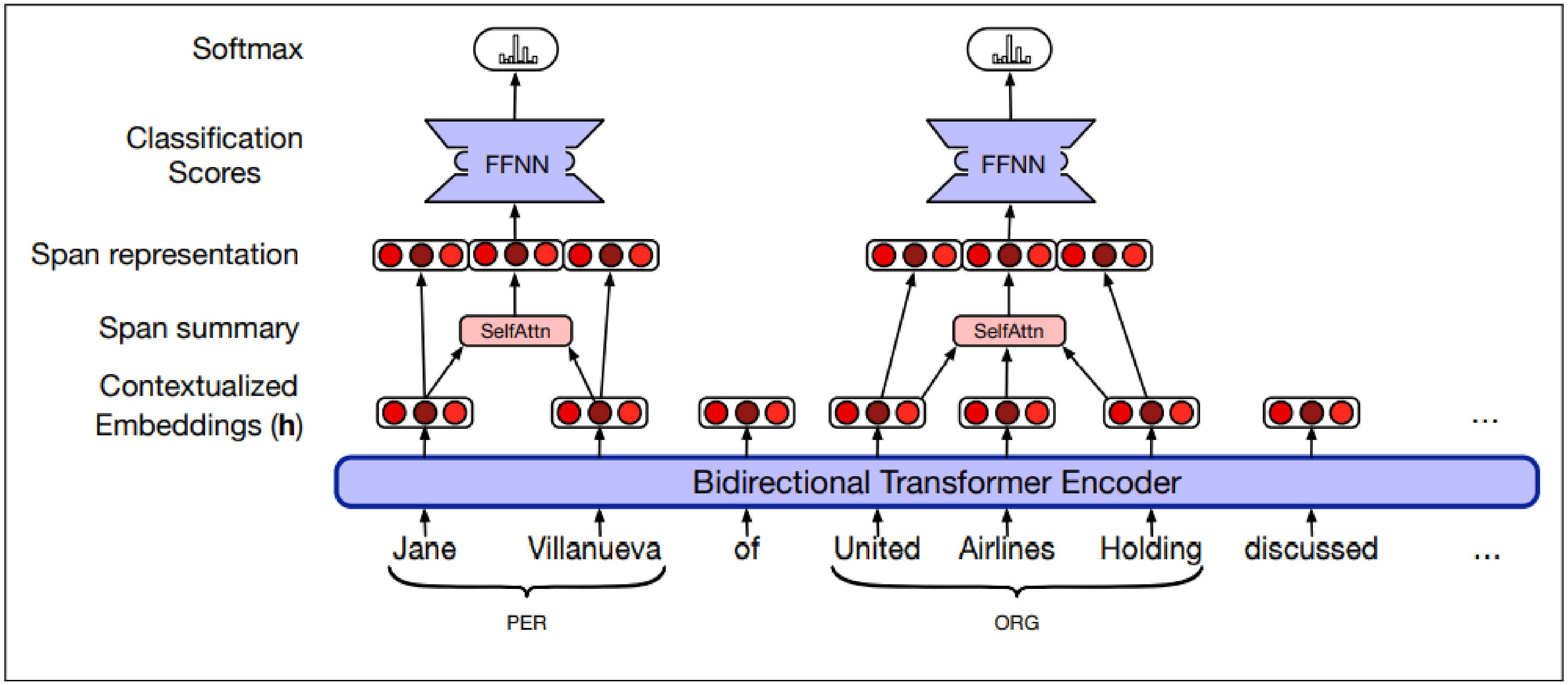
- Similarly, a simple average of the vectors in a span is unlikely to be an optimal representation of a span since it treats all of a span’s embeddings as equally important.
- For many applications, a more useful representation would be centered around the head of the phrase corresponding to the span.
- One method for getting at such information in the absence of a syntactic parse is to use a standard self-attention layer to generate a span representation.

$$g_{ij} = SelfATTN(h_{i:j})$$

- Given span representations g for each span in the total set $S(x)$, classifiers can be fine-tuned to generate application-specific scores for various span-oriented tasks: binary span identification (is this a legitimate span of interest or not?), span classification (what kind of span is this?), and span relation classification (how are these two spans related?).

Transfer Learning through Fine-Tuning – Span-Based Applications

Fig. 10



A span-oriented approach to named entity classification. The figure only illustrates the computation for 2 spans corresponding to ground truth named entities. In reality, the network scores all of the $\frac{T(T-1)}{2}$ spans in the text. That is, all the unigrams, bigrams, trigrams, etc. up to the length limit.

Transfer Learning through Fine-Tuning – Span-Based Applications

- Using NER as an example, given a scheme for representing spans and set of named entity types, a span-based approach to NER is a straightforward classification problem where each span in an input is assigned a class label.
- More formally, given an input sequence x, we want to assign a label y, from the set of valid NER labels, to each of the spans in the total set $S(x)$.
- Since most of the spans in a given input will not be named entities we'll add the label NULL to the set of types in Y.

$$y_{ij} = softmax(FFNN(g_{ij}))$$

- With this approach, fine-tuning entails using supervised training data to learn the parameters of the final classifier, as well as the weights used to generate the boundary representations, and the weights in the self-attention layer that generates the span content representation.

Transfer Learning through Fine-Tuning – Span-Based Applications

- During training, the model's predictions for all spans are compared to their gold-standard labels and cross-entropy loss is used to drive the training.
- During decoding, each span is scored using a softmax over the final classifier output to generate a distribution over the possible labels, with the argmax score for each span taken as the correct answer.
    - A variation on this scheme designed to improve precision adds a calibrated threshold to the labeling of a span as anything other than NULL.

Transfer Learning through Fine-Tuning – Span-Based Applications

- There are two significant advantages to a span-based approach to NER over a BIO-based per-word labeling approach.
- The first advantage is that BIO-based approaches are prone to a labeling mis-match problem.
  - That is, every label in a longer named entity must be correct for an output to be judged correct.
- The following labeling would be judged entirely wrong due to the incorrect label on the first item.

Jane	*Villanueva*	*of*	*United*	*Airlines*	*Holding*	*discussed*	*...*
B-PER	I-PER	O	I-ORG	I-ORG	I-ORG	O	

- Span-based approaches only have to make one classification for each span, as shown in the previous illustration.

Transfer Learning through Fine-Tuning – Span-Based Applications

- The second advantage to span-based approaches is that they naturally accommodate embedded named entities.
- For example, in the example both *United Airlines* and *United Airlines Holding* are legitimate named entities.
- The BIO approach has no way of encoding this embedded structure.
- The span-based approach can naturally label both since the spans are labeled separately.