Transfer Learning with Pretrained Language Models and Contextual Embeddings

Natalie Parde

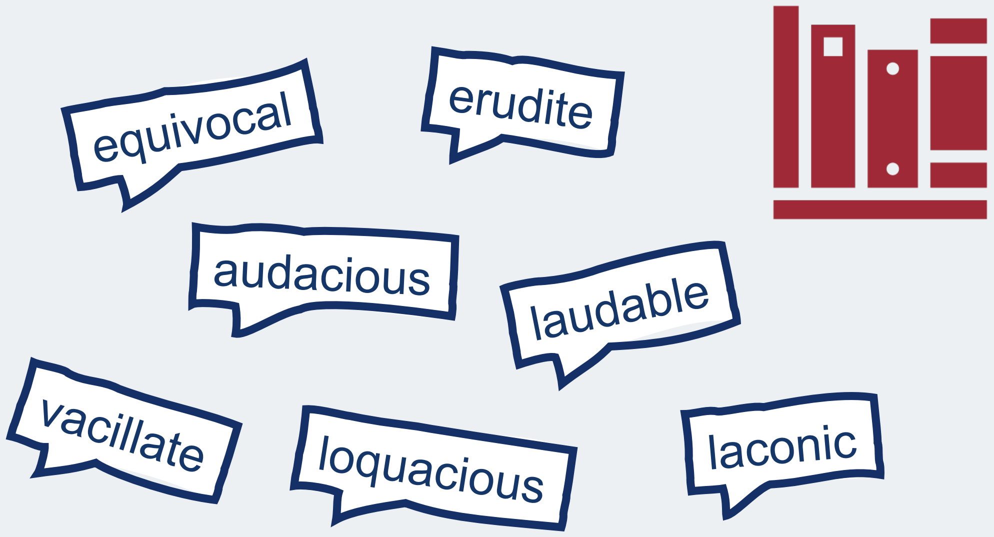
UIC CS 521

Language continually develops and evolves.

- Estimated vocabulary size of a young adult speaker of American English: ~30k-100k words
  - On average, 7-10 new words need to be learned per day through age 20!
  - Active day-to-day vocabulary: ~2k words

How do humans learn the bulk of their vocabulary?

- Early on: Vocabulary is learned via spoken interactions with peers and caregivers
  - Words learned this way form the majority of individuals' active, day-to-day vocabulary
- Later: Vocabulary is mostly learned as a by-product of reading



Can computers learn language in the same way?

- Learning language through experience (e.g., through spoken interactions with peers in a situated environment) is an example of **grounded language learning**
  - Meaning is tied to an experiential (either implied or explicit) **common ground** between speakers

Distributional Hypothesis

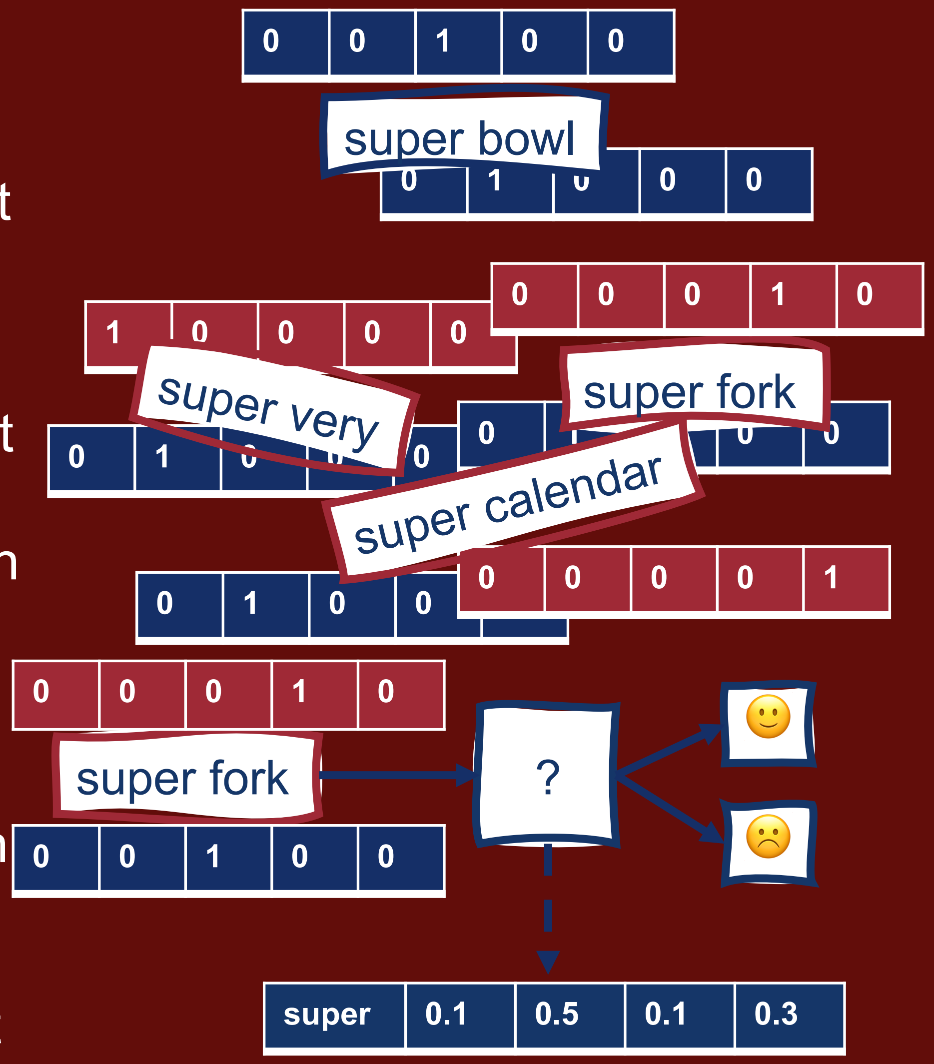
- Learning language based solely on its context is an example of the **distributional hypothesis**
  - Words are defined by the company that they keep!

The distributional hypothesis is the underlying intuition guiding modern word embedding approaches.

- Word embedding techniques “learn” meaning by measuring the frequency with which words occur close to one another in very large text corpora
- Recall:
  - **Word2Vec**
  - **GloVe**

High-Level Overview: How Word2Vec Works

- Represent all words in a vocabulary as a vector
- Treat the target word *w* and a neighboring context word *c* as positive samples
- Randomly sample other words in the lexicon to get negative samples
- Find the similarity for each (t,c) pair and use this to calculate P(+|(t,c))
- Train a classifier to maximize these probabilities to distinguish between positive and negative cases
- Use the weights from that classifier as the word embeddings

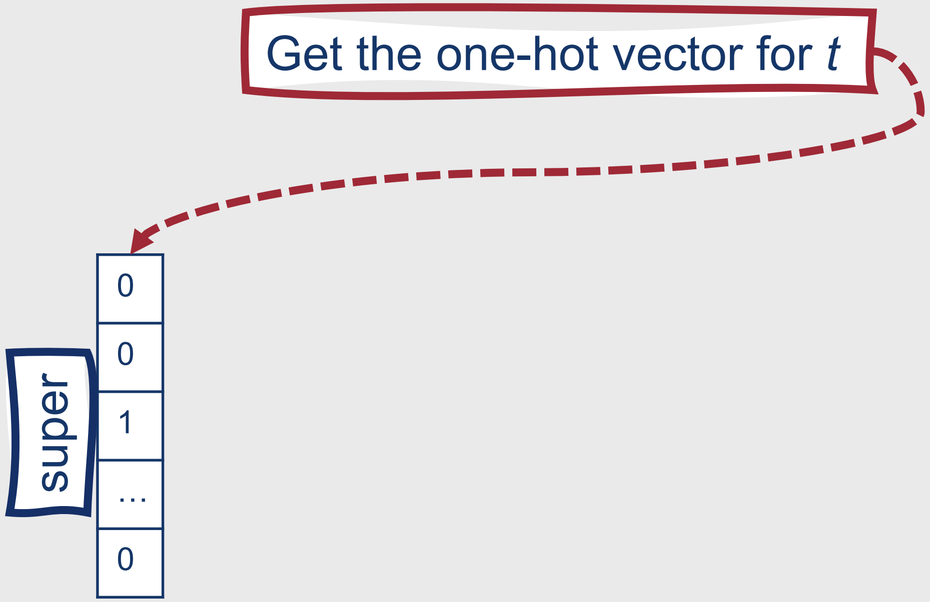


What does this look like?

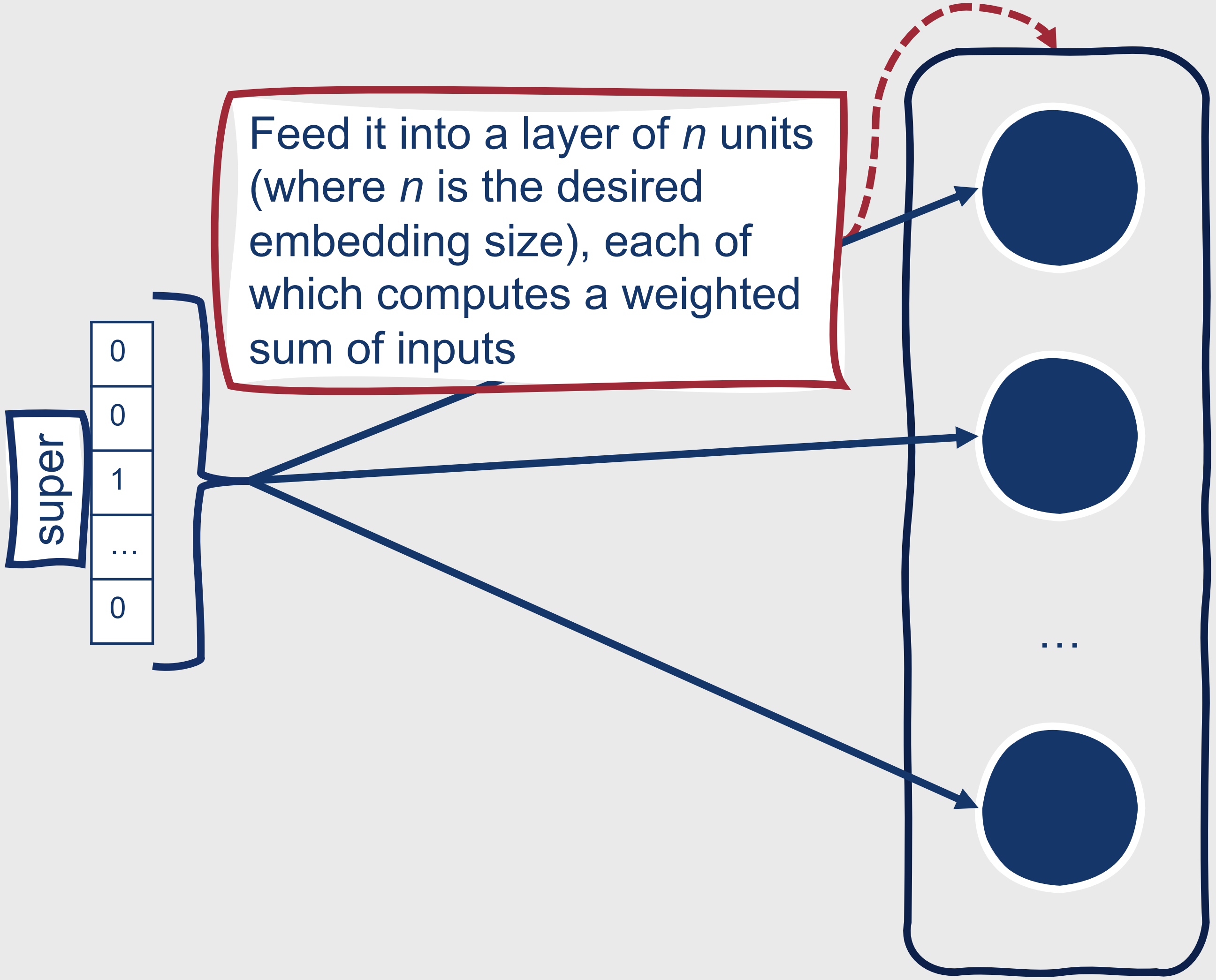
Start with an input *t*

super

What does this look like?

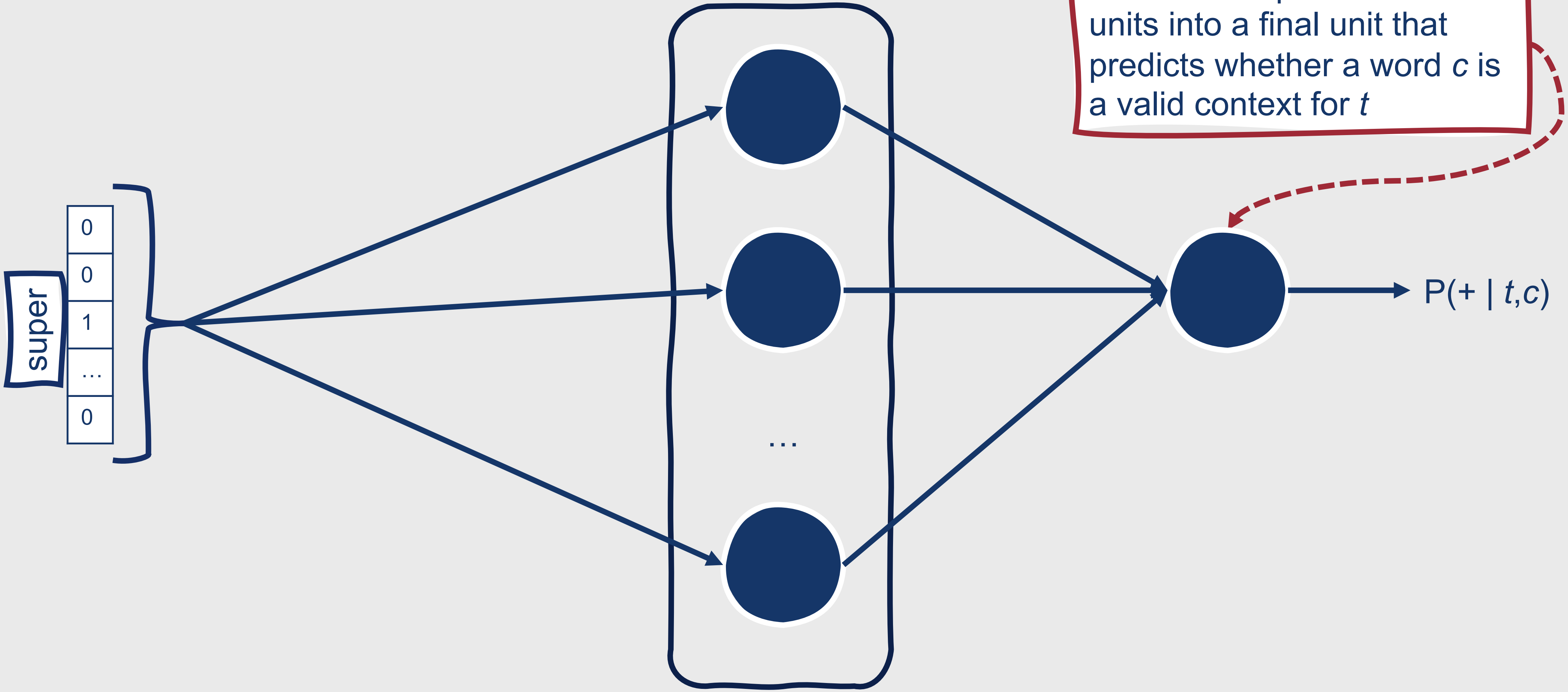


What does this look like?

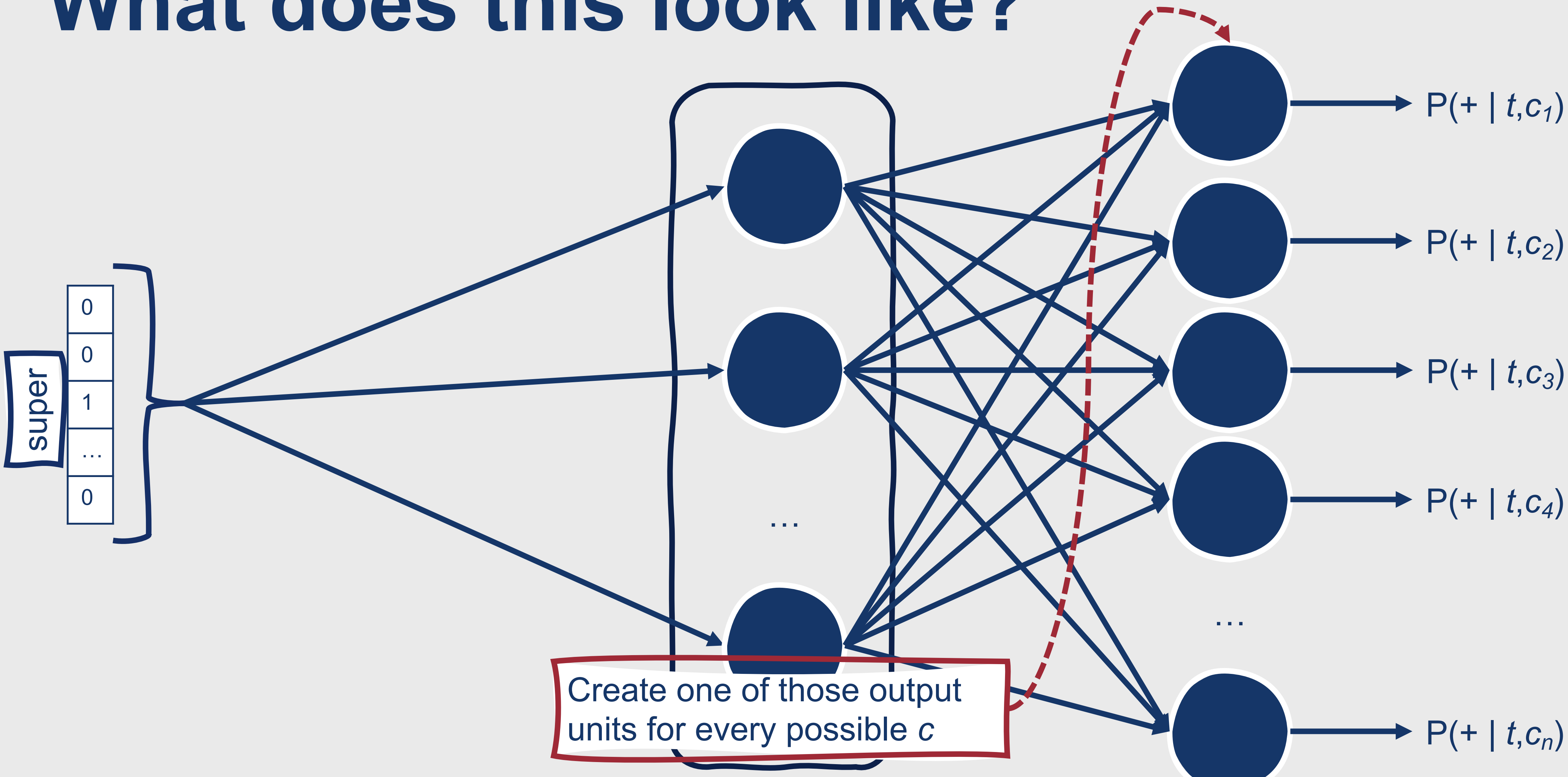


What does this look like?

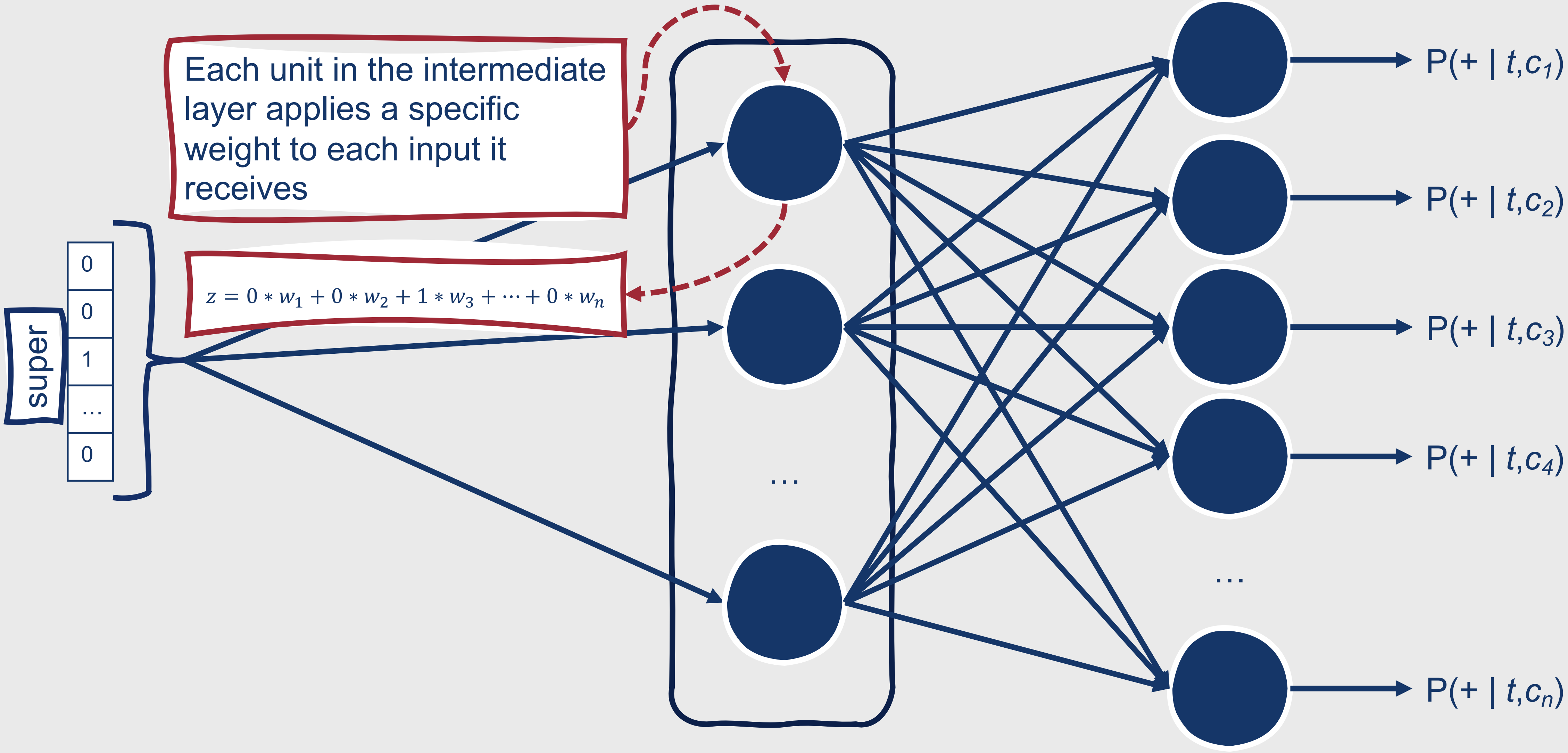
Feed the outputs from those units into a final unit that predicts whether a word *c* is a valid context for *t*



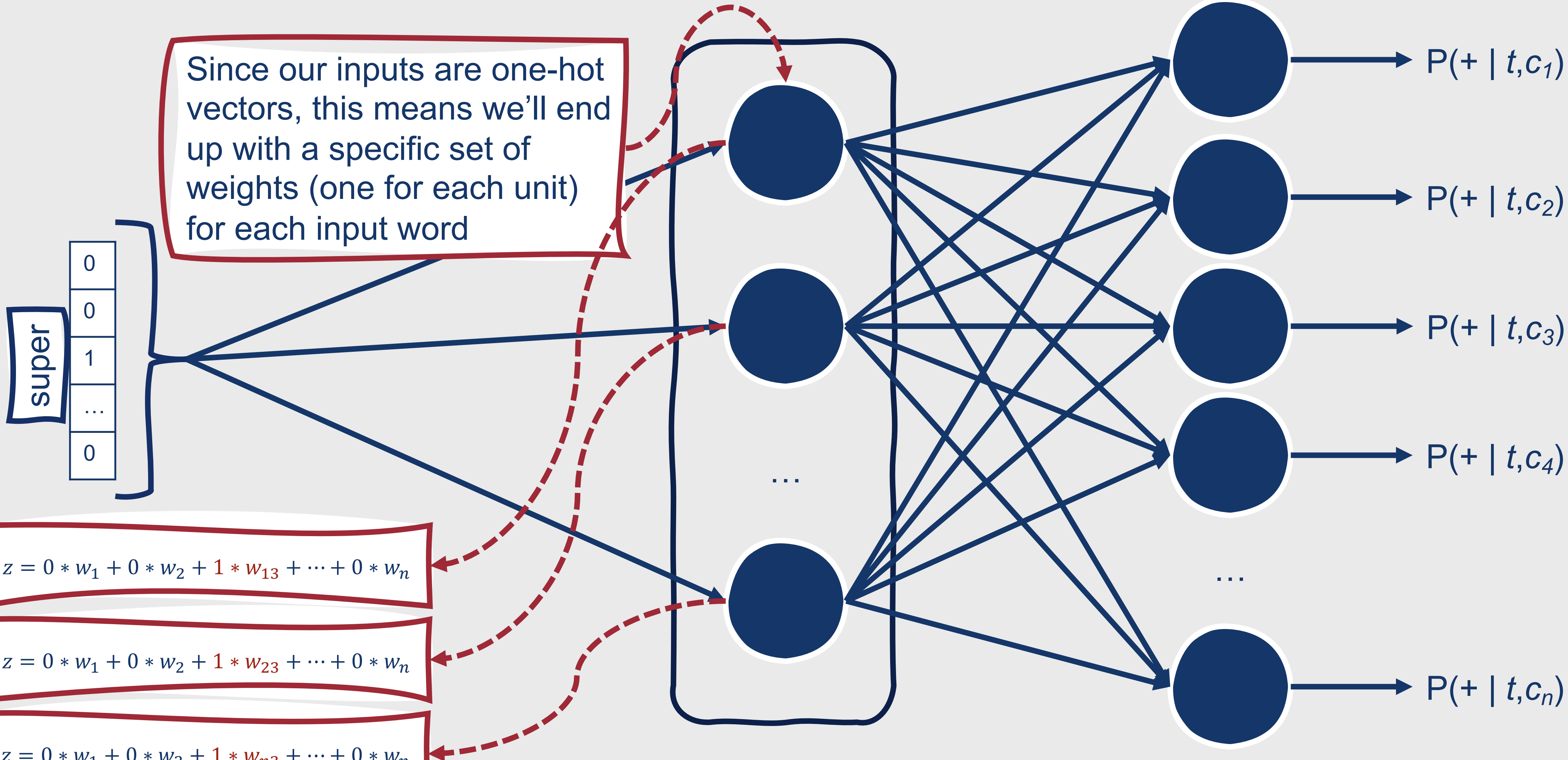
What does this look like?



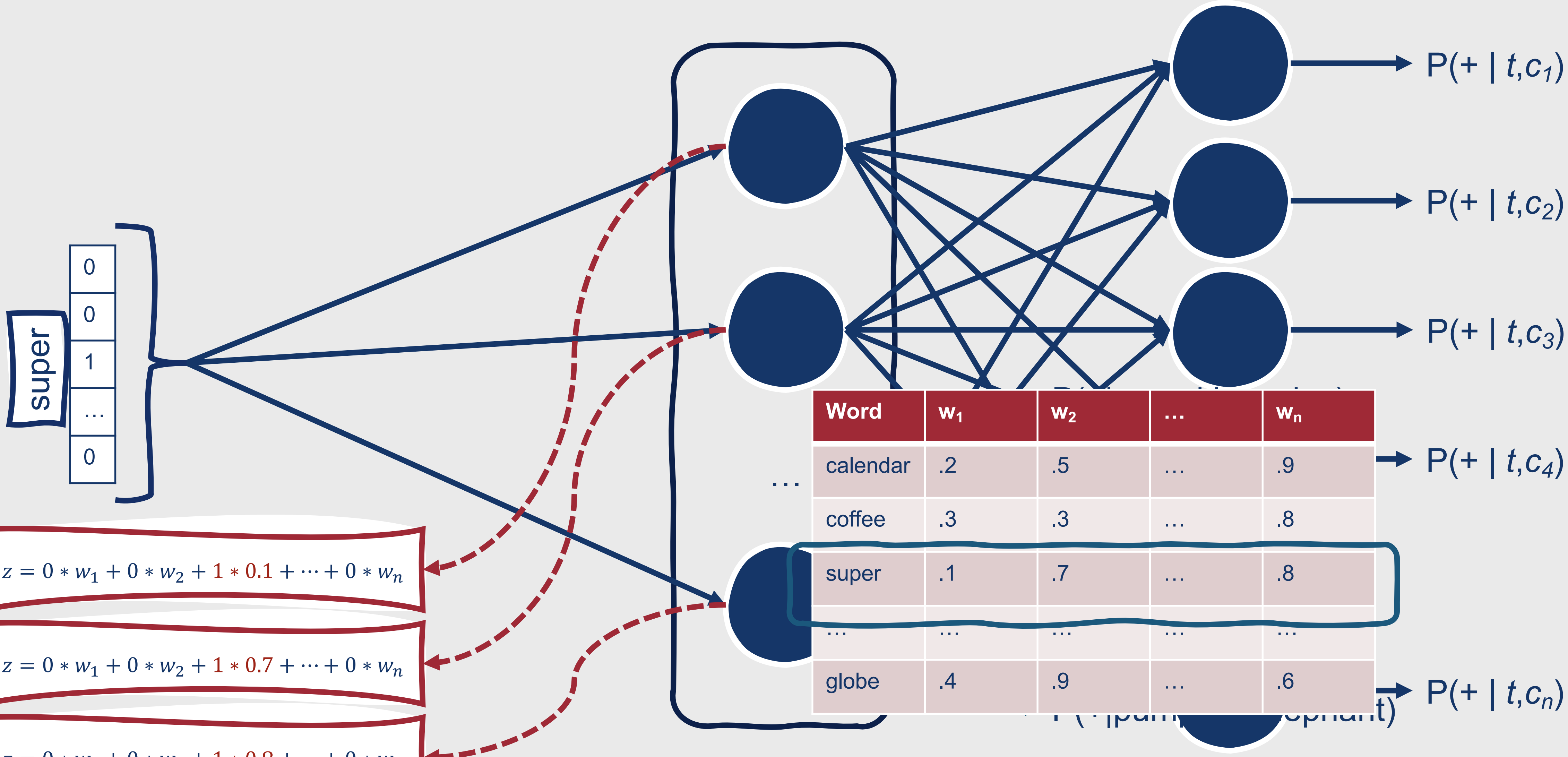
Behind the scenes….



Behind the scenes….



These are the weights we're interested in!



Word	w1	w2	…	wn
calendar	.2	.5	…	.9
coffee	.3	.3	…	.8
super	.1	.7	…	.8
…	…	…	…	…
globe	.4	.9	…	.6

$z = 0 * w_1 + 0 * w_2 + 1 * 0.1 + \cdots + 0 * w_n$

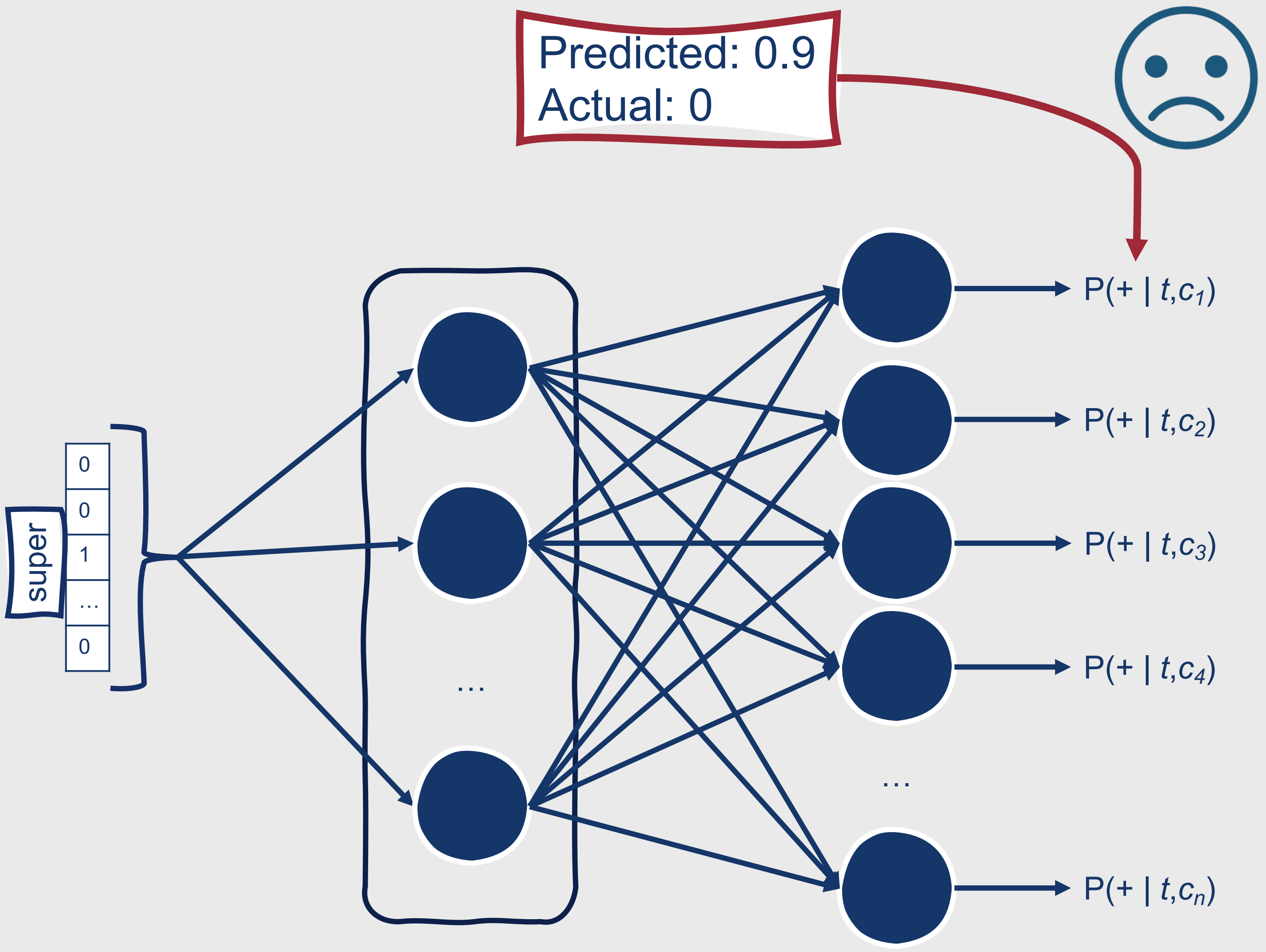
$z = 0 * w_1 + 0 * w_2 + 1 * 0.7 + \cdots + 0 * w_n$

$z = 0 * w_1 + 0 * w_2 + 1 * 0.8 + \cdots + 0 * w_n$

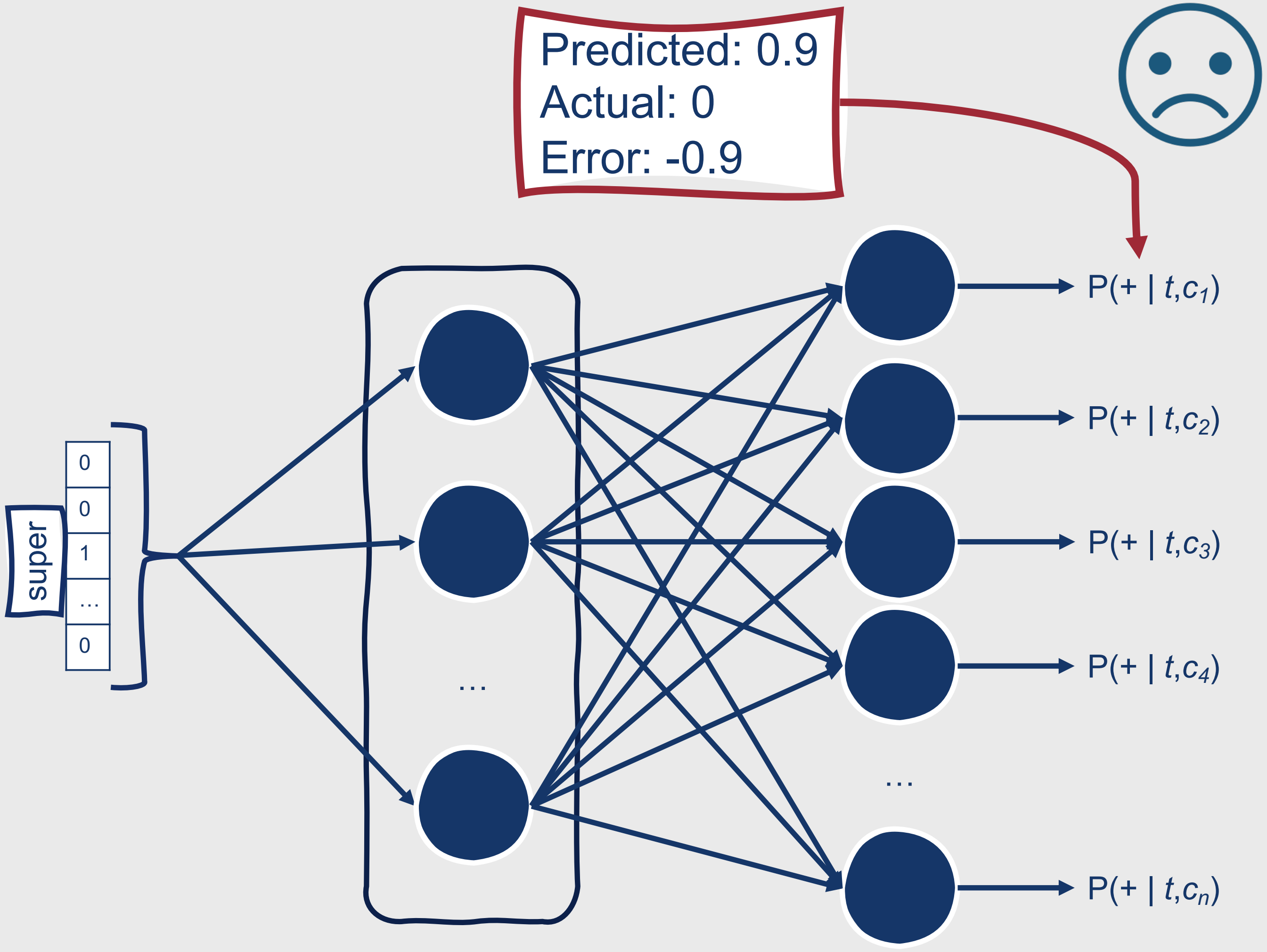
How do we optimize these weights over time?

- The weights are **initialized to some random value** for each word
- They are then iteratively updated to be more similar for words that occur in similar contexts in the training set, and less similar for words that do not
  - Specifically, we want to find weights that maximize $P(+|t,c)$ for words that occur in similar contexts and minimize $P(+|t,c)$ for words that do not, given the information we have at the time

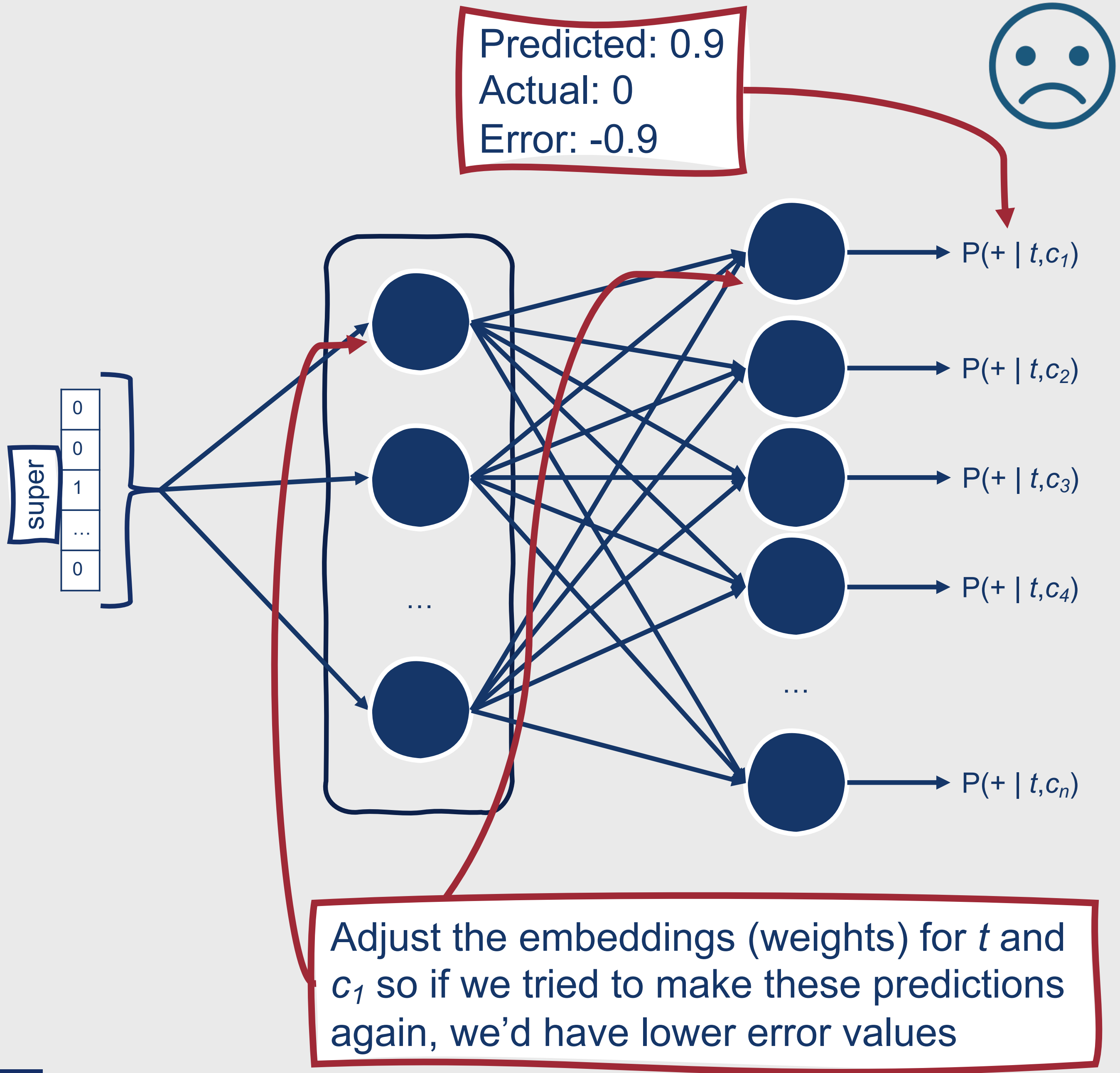
Since we initialize our weights randomly, the classifier's first prediction will almost certainly be wrong.



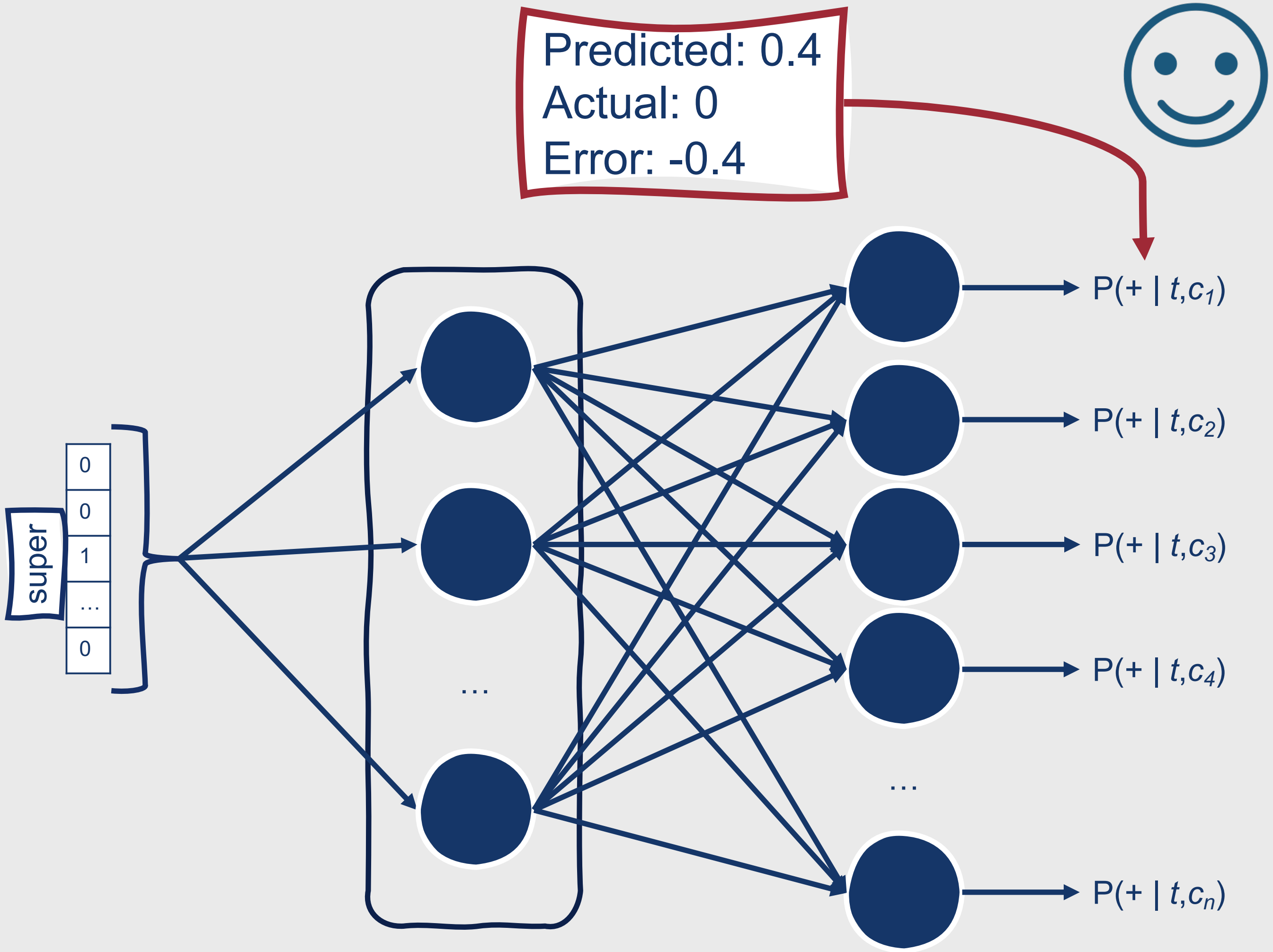
However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



What is our training data?

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

- We are able to assume that all occurrences of words in similar contexts in our training corpus are **positive samples**

What is our training data?

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

- However, we also need negative samples!
- In fact, Word2Vec uses more negative than positive samples (the exact ratio can vary)
- We need to create our own negative examples

What is our training data?

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

- How to create negative examples?
  - Target word + "noise" word that is sampled from the training set
  - Noise words are chosen according to their weighted unigram frequency $p_\alpha(w)$, where α is a weight:
    - $p_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha}$

What is our training data?

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

Negative Examples

t	c
super	calendar
super	exam
super	loud
super	bread
super	cellphone
super	enemy
super	penguin
super	drive

- How to create negative examples?
  - Often, $\alpha = 0.75$ to give rarer noise words slightly higher probability of being randomly sampled
- Assuming we want twice as many negative samples as positive samples, we can thus randomly select noise words according to weighted unigram frequency

Alternatives to Word2Vec

- Word2Vec is an example of a **predictive** word embedding model
  - Learns to predict whether words belong in a target word's context
- Other models are **count-based**
  - Remember co-occurrence matrices?
- GloVE combines aspects of both predictive and count-based models

Global Vectors for Word Representation (GloVe)

- Co-occurrence matrices quickly grow extremely large
- Intuitive solution to increase scalability?
  - Dimensionality reduction!
    - However, typical dimensionality reduction strategies may result in too much computational overhead
- GloVe learns to predict weights in a lower-dimensional space that correspond to the co-occurrence probabilities between words

GloVe

- Why is this useful?
  - Predictive models → black box
    - They work, but why?
  - GloVe models are easier to interpret
- GloVe models also encode the ratios of co-occurrence probabilities between different words …this makes these vectors useful for word analogy tasks

How does GloVe work?

	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

How does GloVe work?

	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$\underbrace{w_i^T}_{\text{Vector for } t_i} \underbrace{w_j}_{\text{Vector for } c_j} + \overbrace{b_i + b_j}^{\text{Scaler biases for } t_i \text{ and } c_j} = \log \underbrace{X_{ij}}_{\text{Co-occurrence count for } t_i c_j}$$

How does GloVe work?

	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Define a cost function

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij}) (w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Weighting function:

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{x_{max}}\right)^{\alpha}, & X_{ij} < XMAX \\ 1, & \text{otherwise} \end{cases}$$

How does GloVe work?

	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Define a cost function

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Minimize the cost function to learn ideal embedding values for w_i and w_j

How does GloVe work?

	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Define a cost function

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} \overbrace{f(X_{ij})}(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Minimize the cost function to learn ideal embedding values for w_i and w_j

0.4	0.7	1.2	4.3	0.9	6.7	1.3	0.5	0.7	5.3

Why does GloVe work?

- Ratios of co-occurrence probabilities have the potential to encode word similarities and differences
- These similarities and differences are useful components of meaning
  - GloVe embeddings perform particularly well on analogy tasks

Word2Vec and GloVe are both *static* word embeddings.

- A given word has the same embedding, regardless of its context
- Reasonable in many cases, but not always
  - What if a word has multiple senses?
  - What if a word starts appearing in new contexts?

Did you deposit that check at the **bank**?

0.4	0.2	0.5	0.7	0.1

A message in a bottle washed up on the **bank**.

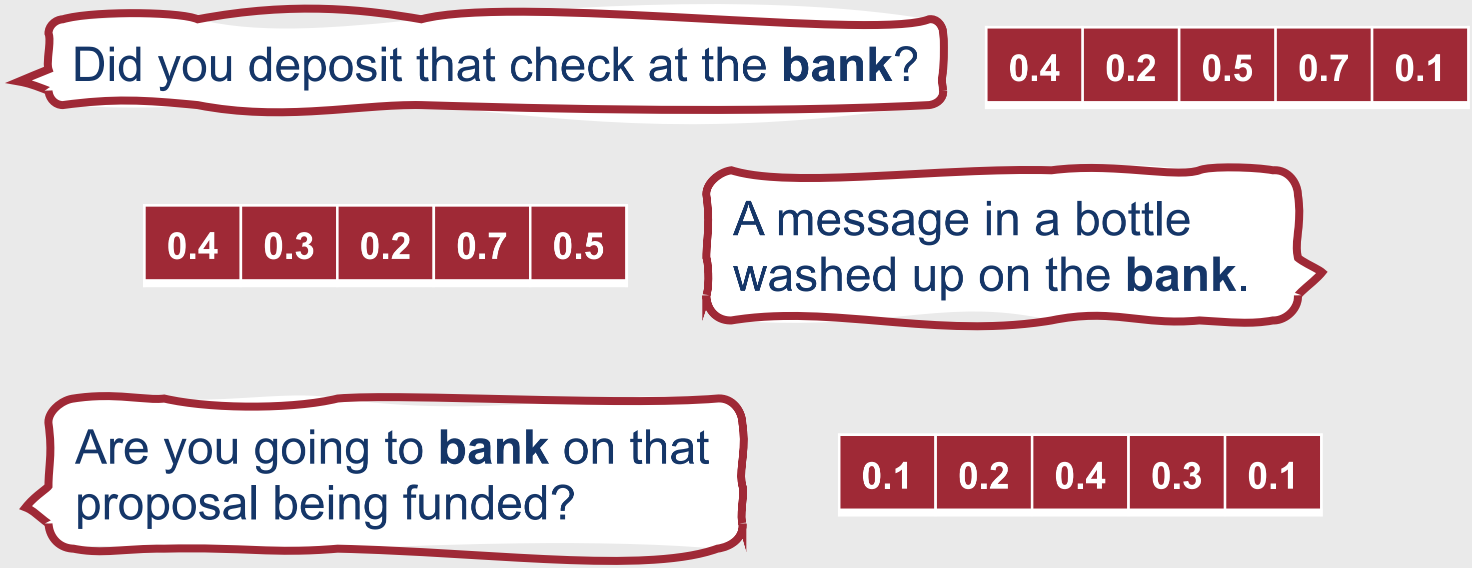
0.4	0.2	0.5	0.7	0.1

Are you going to **bank** on that proposal being funded?

0.4	0.2	0.5	0.7	0.1

Contextual Word Embeddings

- Word representations that differ depending on the context in which the word appears
- Vocabulary words do not map to specific, predefined vectors
- How are contextual word embeddings learned?
  - Often, pretrained language models
  - Popular method: BERT



Bidirectional Encoder Representations from Transformers (BERT)

- Popular method for learning contextual word representations
- Many variations
  - DistilBERT
  - RoBERTa
  - SpanBERT
  - ALBERT
- Makes use of a **bidirectional encoder model**

pdf bib abs **MPC-BERT: A Pre-Trained Language Model for Multi-Party Conversation Understanding**
Jia-Chen Gu | Chongyang Tao | Zhenhua Ling | Can Xu | Xiubo Geng | Daxin Jiang

pdf bib abs **BERTAC: Enhancing Transformer-based Language Models with Adversarially Pretrained Convolutional Neural Networks**
Jong-Hoon Oh | Ryu Iida | Julien Kloetzer | Kentaro Torisawa

pdf bib abs **Marginal Utility Diminishes: Exploring the Minimum Knowledge for BERT Knowledge Distillation**
Yuanxin Liu | Fandong Meng | Zheng Lin | Weiping Wang | Jie Zhou

pdf bib abs **ChineseBERT: Chinese Pretraining Enhanced by Glyph and Pinyin Information**
Zijun Sun | Xiaoya Li | Xiaofei Sun | Yuxian Meng | Xiang Ao | Qing He | Fei Wu | Jiwei Li

pdf bib abs **EarlyBERT: Efficient BERT Training via Early-bird Lottery Tickets**
Xiaohan Chen | Yu Cheng | Shuohang Wang | Zhe Gan | Zhangyang Wang | Jingjing Liu

pdf bib abs **Accelerating BERT Inference for Sequence Labeling via Early-Exit**
Xiaonan Li | Yunfan Shao | Tianxiang Sun | Hang Yan | Xipeng Qiu | Xuanjing Huang

pdf bib abs **Self-Guided Contrastive Learning for BERT Sentence Representations**
Taeuk Kim | Kang Min Yoo | Sang-goo Lee

pdf bib abs **LeeBERT: Learned Early Exit for BERT with cross-level optimization**
Wei Zhu

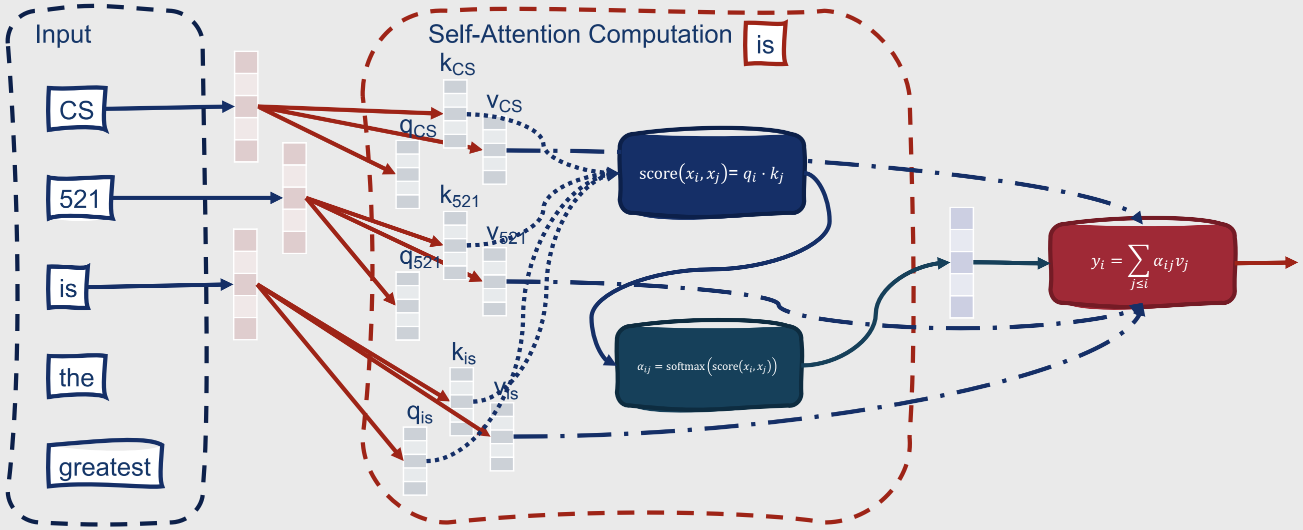
pdf bib abs **BoB: BERT Over BERT for Training Persona-based Dialogue Models from Limited Personalized Data**
Haoyu Song | Yan Wang | Kaiyan Zhang | Wei-Nan Zhang | Ting Liu

pdf bib abs **BERT is to NLP what AlexNet is to CV: Can Pre-Trained Language Models Identify Analogies?**
Asahi Ushio | Luis Espinosa Anke | Steven Schockaert | Jose Camacho-Collados

BERT is everywhere!

Bidirectional Transformer Encoders

- We've already seen how "causal" (left to right) Transformers work
  - Well suited for language modeling problems since they prevent consideration of future context
- However, these models are inherently constrained
  - What about tasks for which "future" context is readily available?



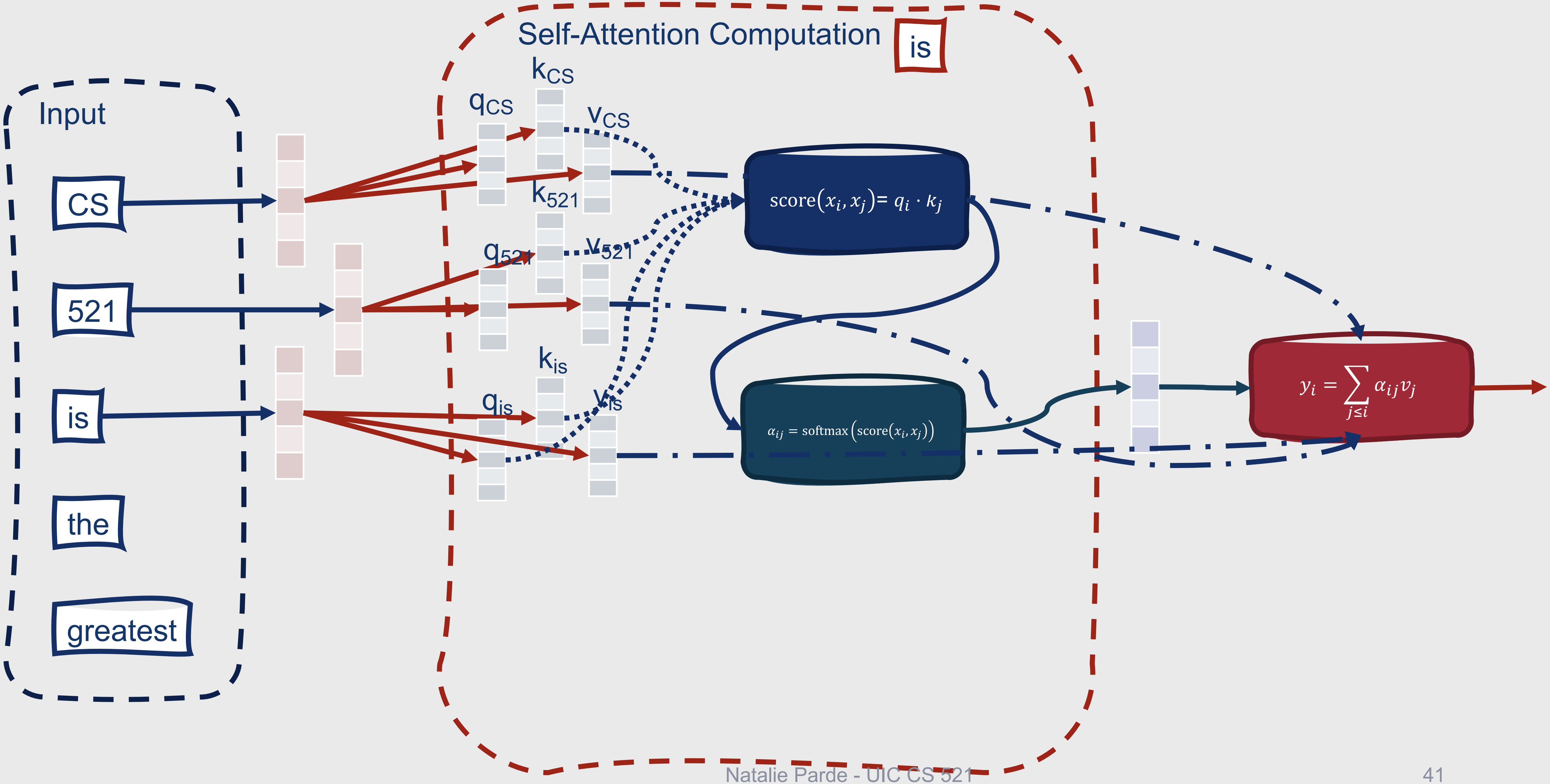
Many NLP tasks don’t need to restrict the model from viewing future context.

- Sequence classification
- (Sometimes) sequence labeling
- In general, most tasks that *aren’t* performed in real time

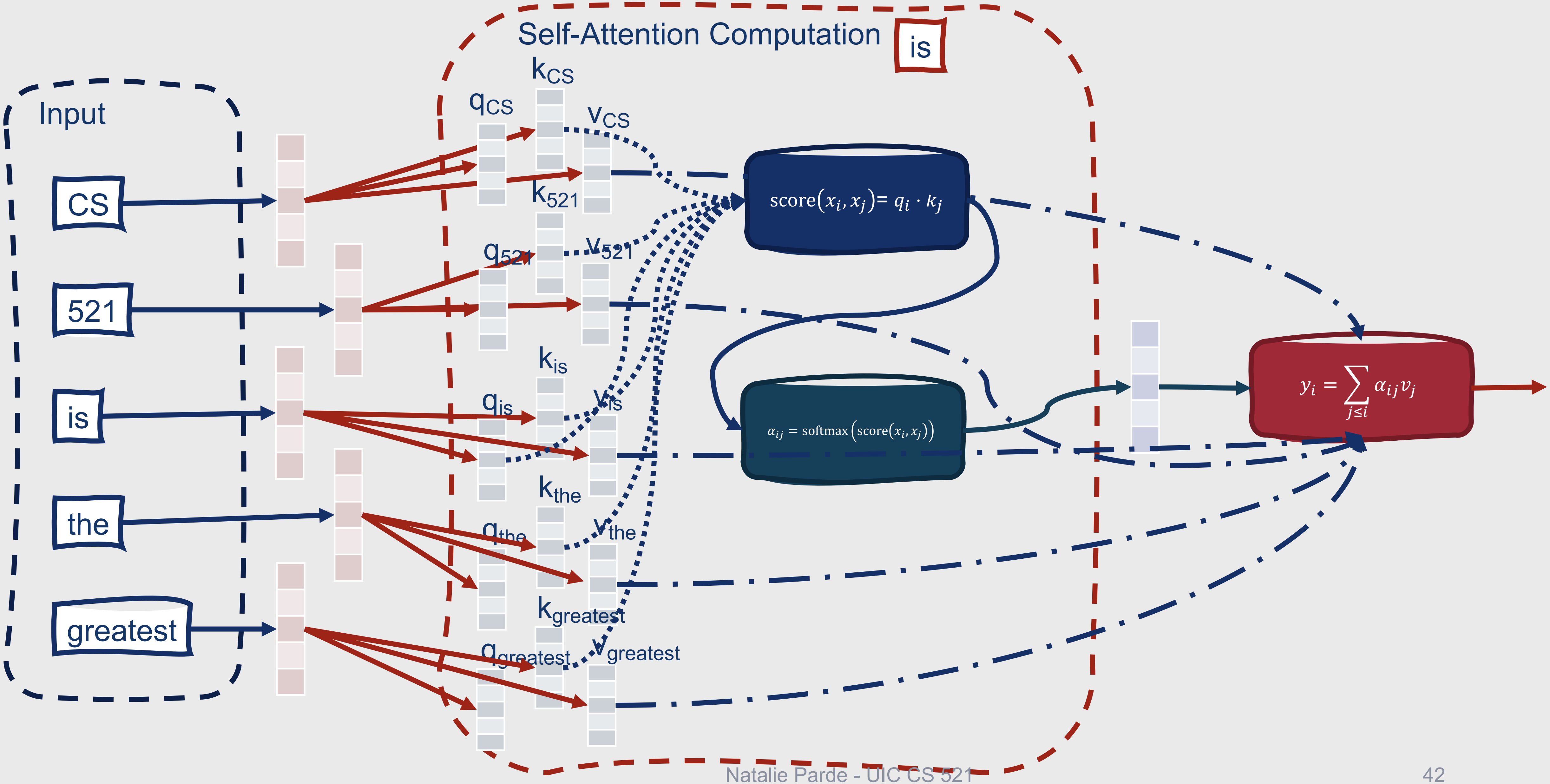
Transformers aren’t innately constrained to processing from left to right.

- With language modeling, self-attention computations are limited to current and prior context to avoid trivializing the problem
- Self-attention can be computed identically to the current equation when allowing future context to be considered
- This causes the encoder to produces sequences of output embeddings that are contextualized based on the entire input sequence

Bidirectional Self-Attention Layer



Bidirectional Self-Attention Layer



More formally….

- Step 1: Generate key, query, and value embeddings for each element of the input vector $\mathbf{x}$
  - $\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i$
  - $\mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i$
  - $\mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$

More formally….

- Step 2: Compute attention weights α by applying a softmax over the element-wise comparison scores between all possible query-key pairs in the full input sequence
  - $\text{score}_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$
  - $\alpha_{ij} = \frac{\exp(\text{score}_{ij})}{\sum_{k=1}^{n} \exp(\text{score}_{ik})}$

More formally….

- Step 3: Compute the output vector $\mathbf{y}_i$ as the attention-weighted sum of all of the input value vectors $\mathbf{v}$
  - $\mathbf{y}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{v}_j$

Additional Notes

- Each output vector $\mathbf{y}_i$ is computed independently
- This allows us to use matrix operations to parallelize the input processing

How can we do this?

- Let the embedding of each input token, $\mathbf{x}_i$, serve as one row of the input matrix $\mathbf{X} \in \mathbb{R}^{N \times d_h}$
- Multiply $\mathbf{X}$ by the key, query, and value *weight* matrices ($\mathbf{W}^{\mathbf{K}}, \mathbf{W}^{\mathbf{Q}}, \mathbf{W}^{\mathbf{V}} \in \mathbb{R}^{d \times d}$) to produce the key, query, and value matrices ($\mathbf{K}, \mathbf{Q}, \mathbf{V} \in \mathbb{R}^{N \times d}$)
  - $\mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}$
  - $\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}$
  - $\mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$
- This means that all key-query comparisons can be computed simultaneously by multiplying $\mathbf{Q}$ and $\mathbf{K}^{\mathbf{T}}$ in a single operation
- Scale the scores, take the softmax, and multiply the result by $\mathbf{V}$ to produce a matrix $\mathrm{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \in \mathbb{R}^{N \times d}$ where each row contains a contextualized output embedding corresponding to a given input token
  - $\mathrm{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$

Visually….

$\mathbf{QK^T}$ matrix for a standard, causal Transformer encoder

$q_1 \cdot k_1$	$q_1 \cdot k_2$	$q_1 \cdot k_3$	$q_1 \cdot k_4$	$q_1 \cdot k_5$
$q_2 \cdot k_1$	$q_2 \cdot k_2$	$q_2 \cdot k_3$	$q_2 \cdot k_4$	$q_2 \cdot k_5$
$q_3 \cdot k_1$	$q_3 \cdot k_2$	$q_3 \cdot k_3$	$q_3 \cdot k_4$	$q_3 \cdot k_5$
$q_4 \cdot k_1$	$q_4 \cdot k_2$	$q_4 \cdot k_3$	$q_4 \cdot k_4$	$q_4 \cdot k_5$
$q_5 \cdot k_1$	$q_5 \cdot k_2$	$q_5 \cdot k_3$	$q_5 \cdot k_4$	$q_5 \cdot k_5$

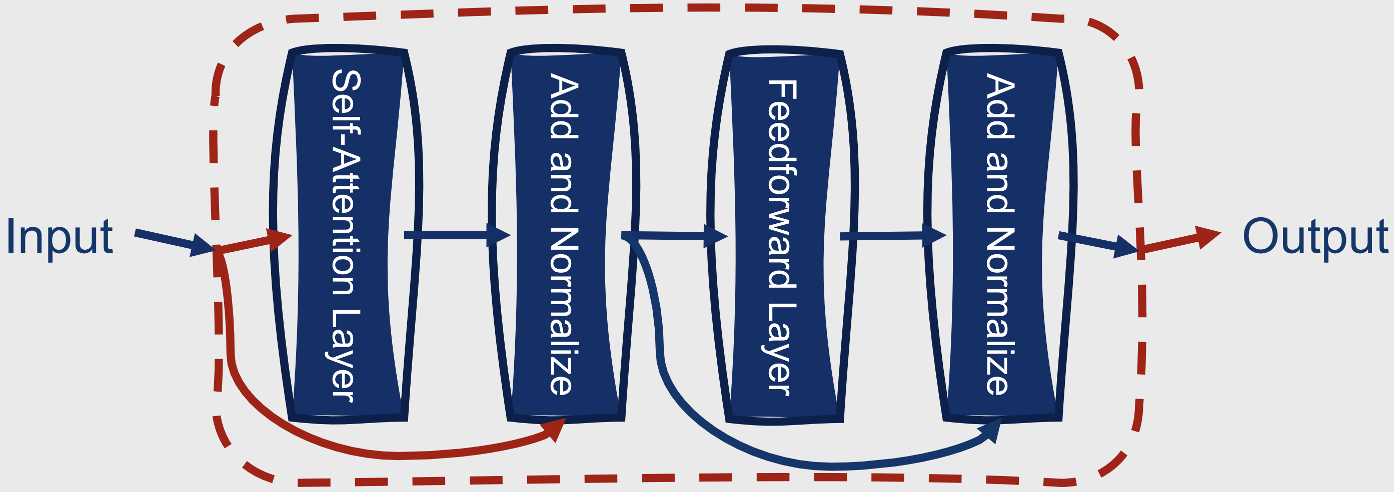
Visually….

$\mathbf{QK^T}$ matrix for a bidirectional Transformer encoder

$q_1 \cdot k_1$	$q_1 \cdot k_2$	$q_1 \cdot k_3$	$q_1 \cdot k_4$	$q_1 \cdot k_5$
$q_2 \cdot k_1$	$q_2 \cdot k_2$	$q_2 \cdot k_3$	$q_2 \cdot k_4$	$q_2 \cdot k_5$
$q_3 \cdot k_1$	$q_3 \cdot k_2$	$q_3 \cdot k_3$	$q_3 \cdot k_4$	$q_3 \cdot k_5$
$q_4 \cdot k_1$	$q_4 \cdot k_2$	$q_4 \cdot k_3$	$q_4 \cdot k_4$	$q_4 \cdot k_5$
$q_5 \cdot k_1$	$q_5 \cdot k_2$	$q_5 \cdot k_3$	$q_5 \cdot k_4$	$q_5 \cdot k_5$

Bidirectional Transformer Encoders

- All other elements remain the same as seen in causal Transformers!
  - Inputs are segmented using subword tokenization
  - Inputs are combined with positional embeddings
  - Transformer blocks include a self-attention layer and a feedforward layer, augmented with normalization layers and residual connections



How does BERT work specifically?

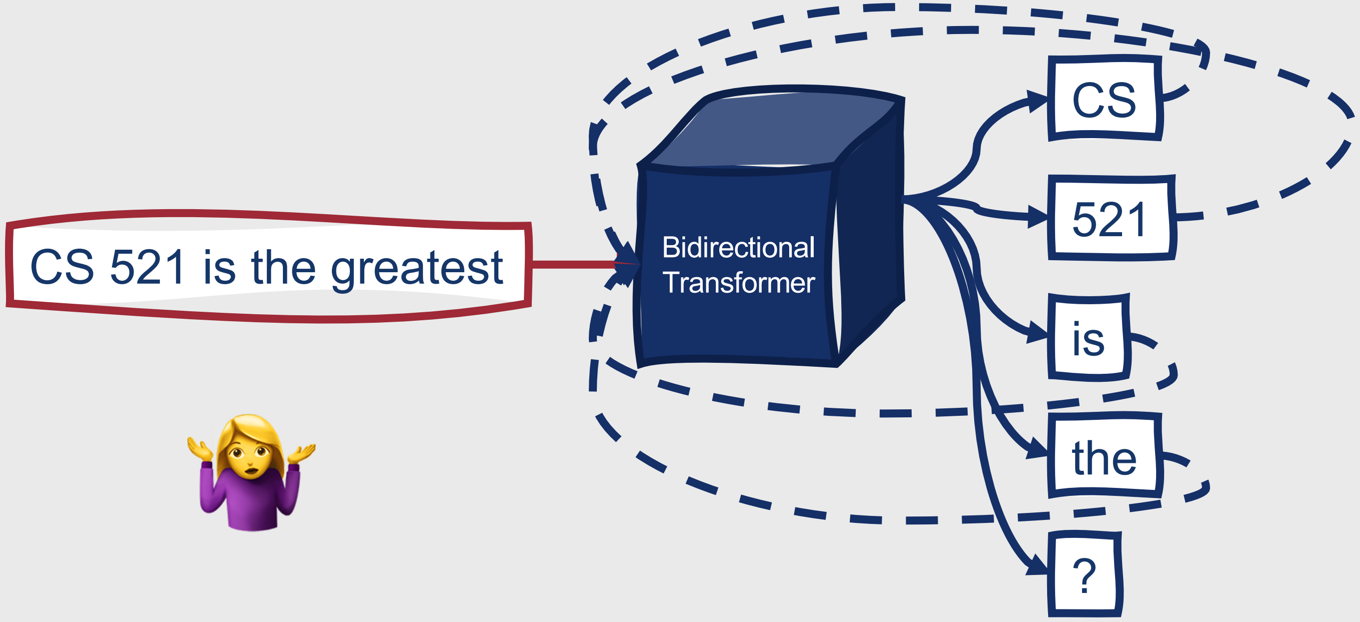
- BERT: The original bidirectional Transformer encoder model
- Subword vocabulary of 30k tokens generated using the Word-Piece algorithm
- 768-dimensional hidden layers
- 12 Transformer blocks
- 12 attention heads in each self-attention layer
- In total, this comprises 100M trainable parameters!

Additional BERT Details

- Since subword tokenization is used, for some NLP tasks (e.g., named entity tagging) it is necessary to map subwords back to words
- Costly to train (time and memory requirements grow quadratically with input length)
  - To increase efficiency, a fixed input length of 512 subword tokens is used---when working with longer texts, it's necessary to partition the text into different segments
- More details to come during our discussions of representation learning!

Training Bidirectional Encoders

- With causal Transformer encoders, we employed autoregressive training
  - Autoregressive training: Train the model to iteratively predict the next word in a text
- With bidirectional Transformer encoders, this task becomes trivial …the answer is now directly available from the context!



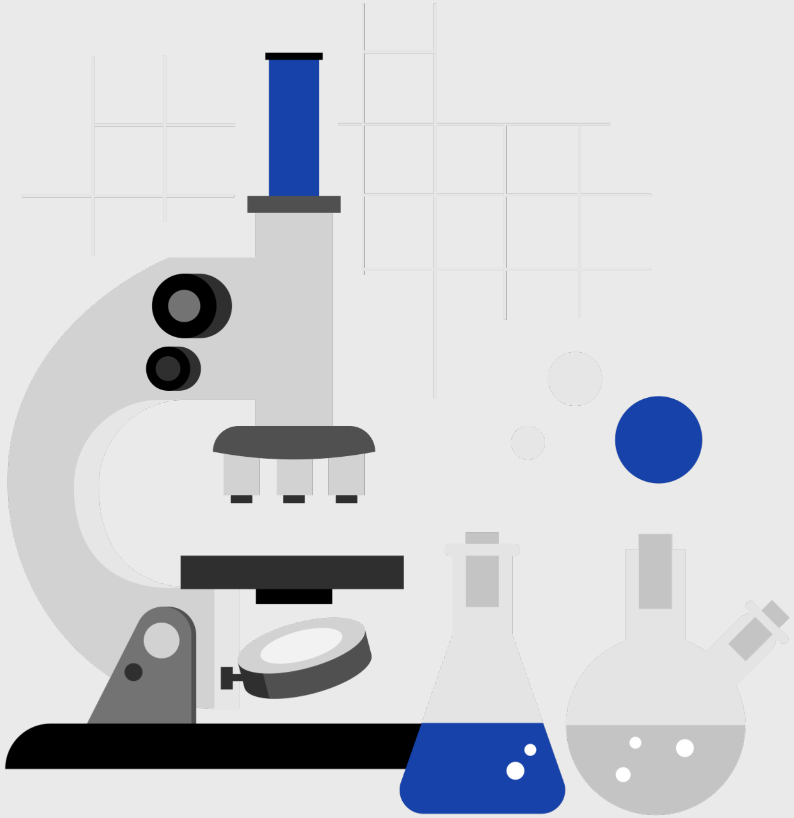
A new task is needed for training bidirectional encoders….

- **Cloze Task:** Instead of trying to predict the next word, learn how to predict the best word to fill in the blank
- How do we do this?
  - During training, **mask out** one or more elements from the input sequence
  - Generate a **probability distribution** over the vocabulary for each of the missing elements
  - Use the **cross-entropy loss** from these probabilities to drive the learning process

After such a late _____ working on my project, it was _____ to wake up this morning!

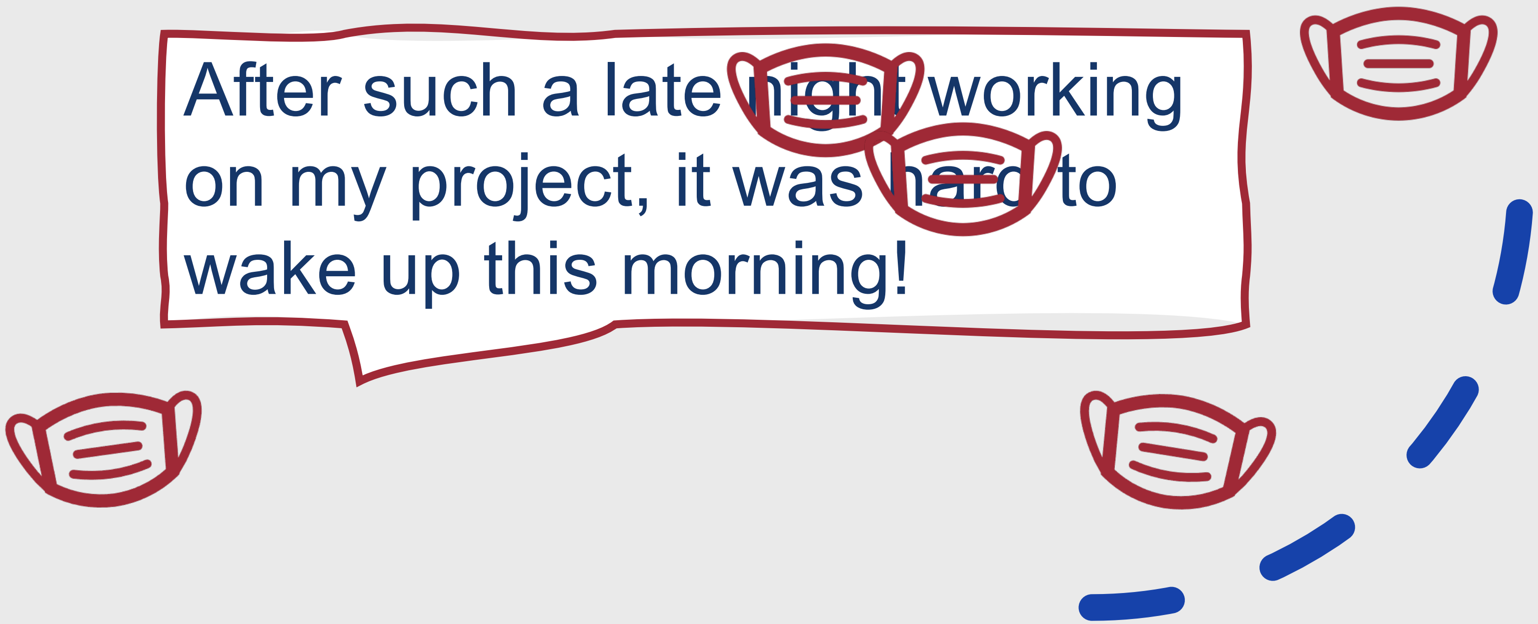
Cloze Task

- This task can be generalized to any method that:
  1. Corrupts the training input
  2. Asks the model to recover the original training input
- What are some ways to corrupt the training input?
  - Masks
  - Substitutions
  - Reorderings
  - Deletions
  - Extraneous insertions into the training text



Masking Words

- Original approach for corrupting input when training bidirectional Transformer encoders
- BERT uses a masking technique known as masked language modeling (MLM)



Masked Language Modeling

- Uses unannotated text from a large corpus
- Presents the models with a series of sentences from the corpus
- For each sentence, a random sample of tokens is selected to be used in one of the following ways:
  - The token is replaced with a [MASK] token
  - The token is replaced with another randomly sampled token
  - The token is left unchanged

Masked Language Modeling

After such a late night working on my project, it was hard to wake up this morning!

After such a [MASK] night working on my project, it was hard to wake up this driving!

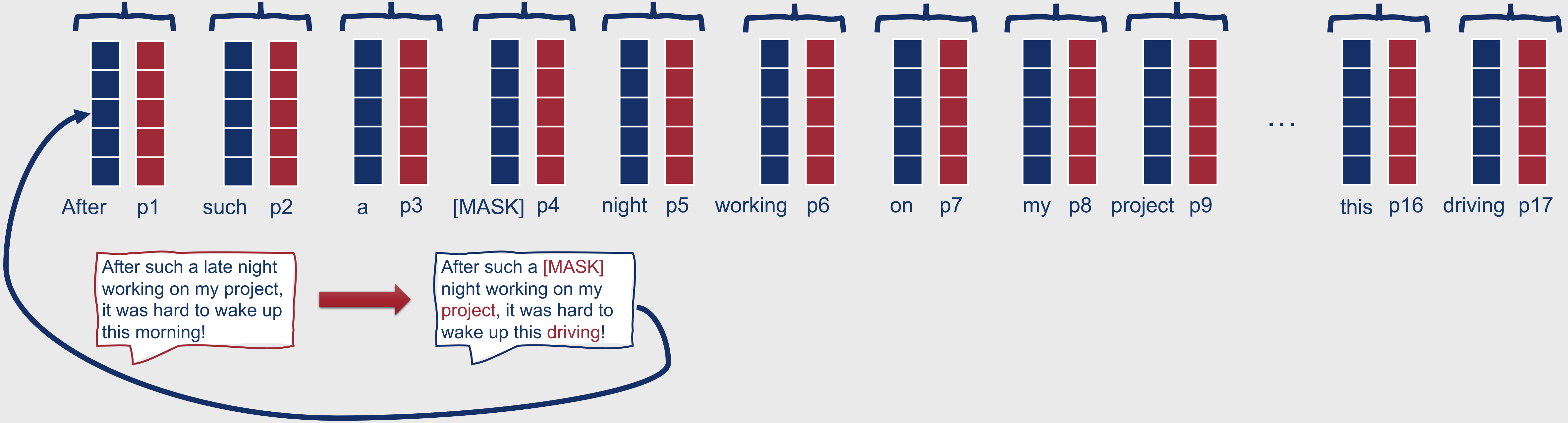
Masked Language Modeling

After such a late night working on my project, it was hard to wake up this morning!

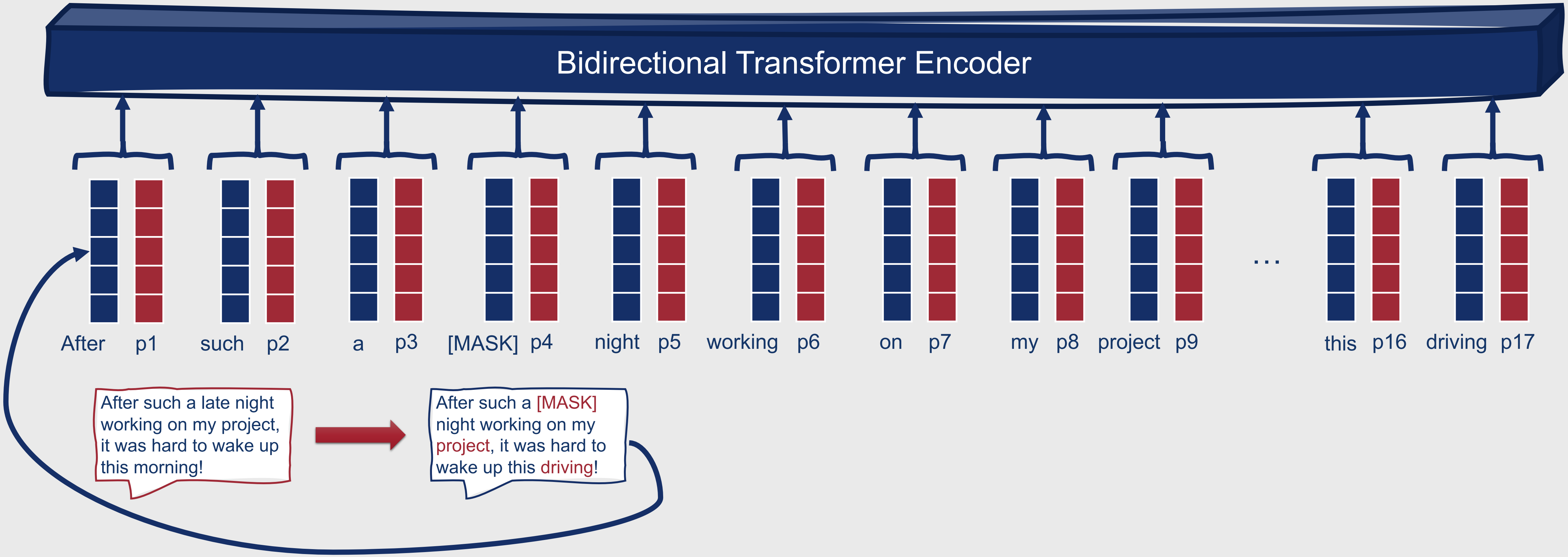
→

After such a [MASK] night working on my project, it was hard to wake up this driving!

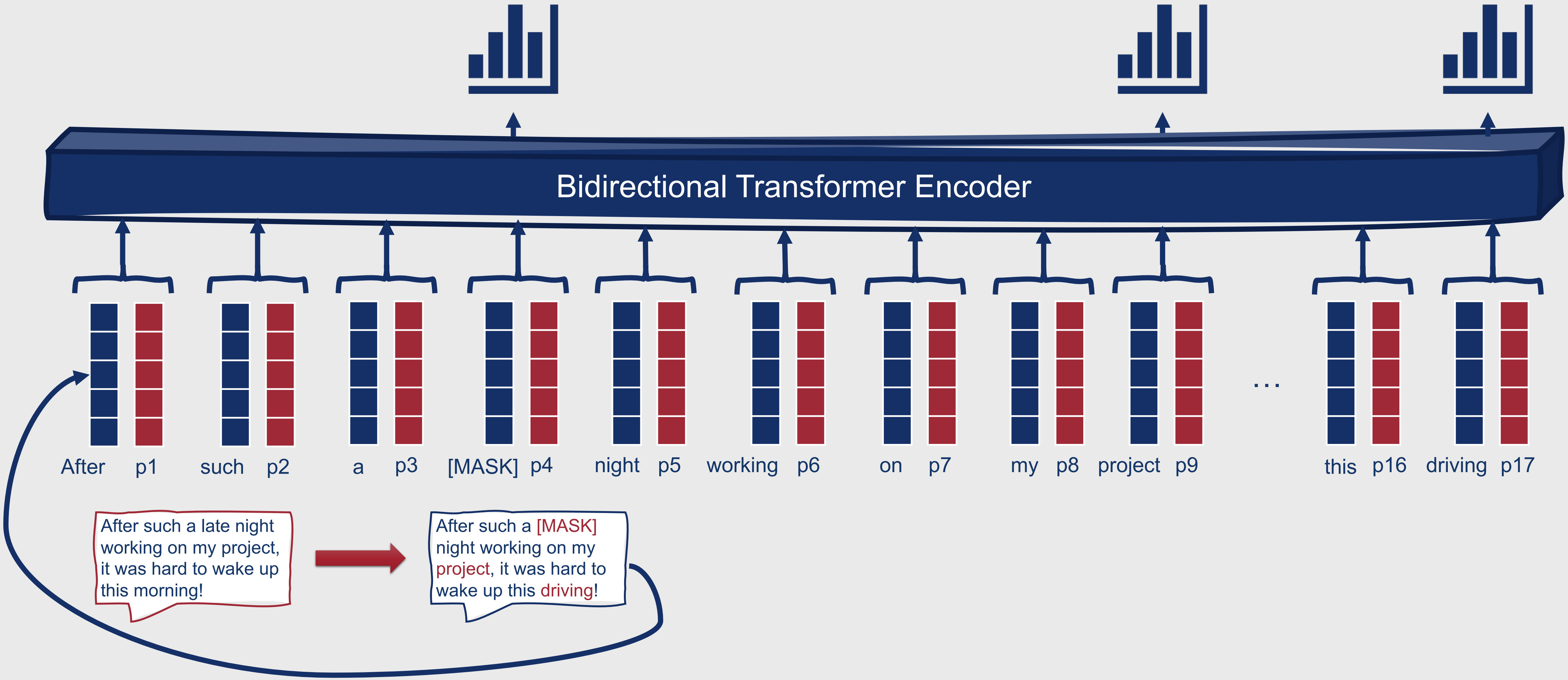
Masked Language Modeling



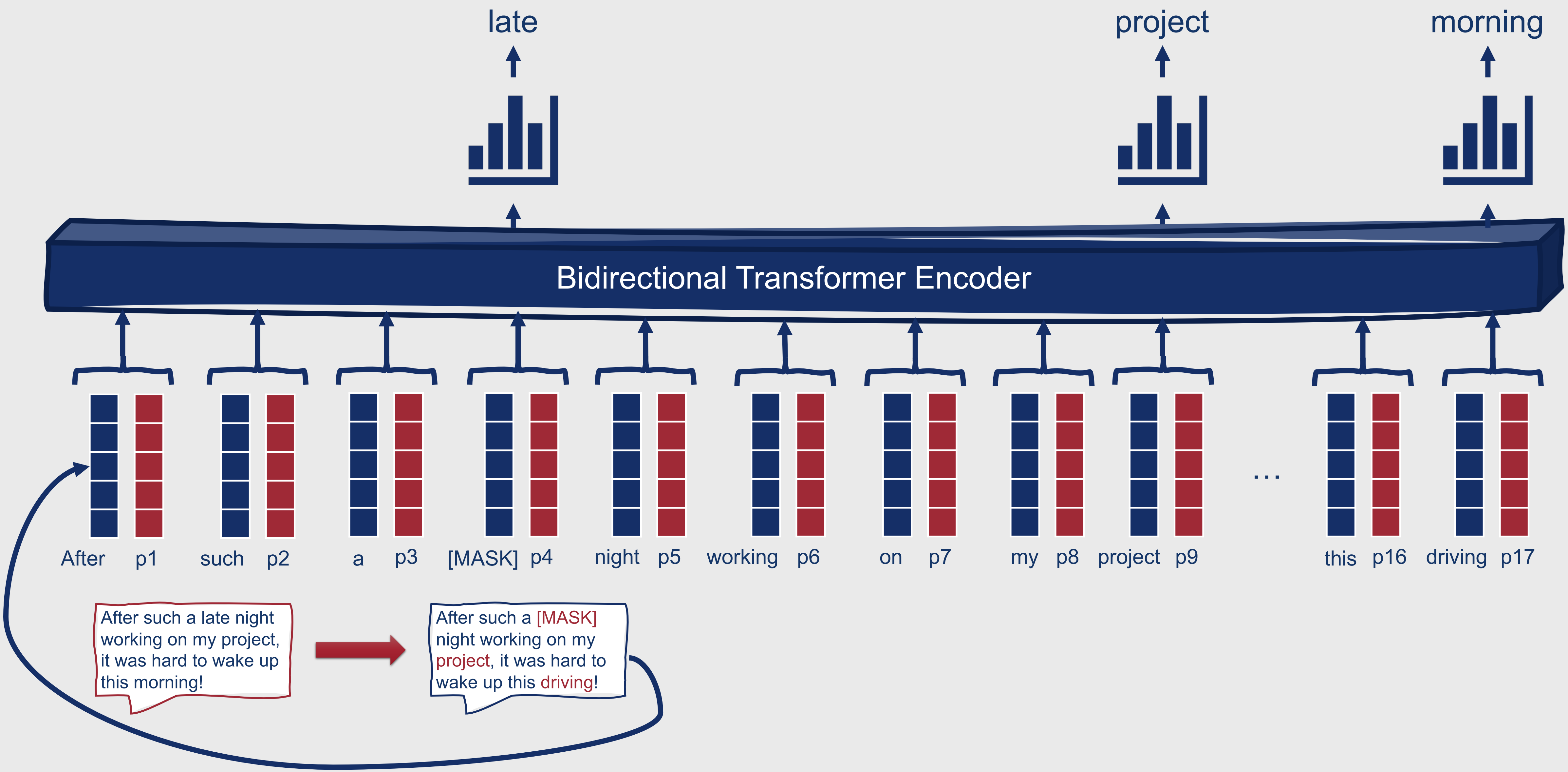
Masked Language Modeling



Masked Language Modeling



Masked Language Modeling



Masked Language Modeling

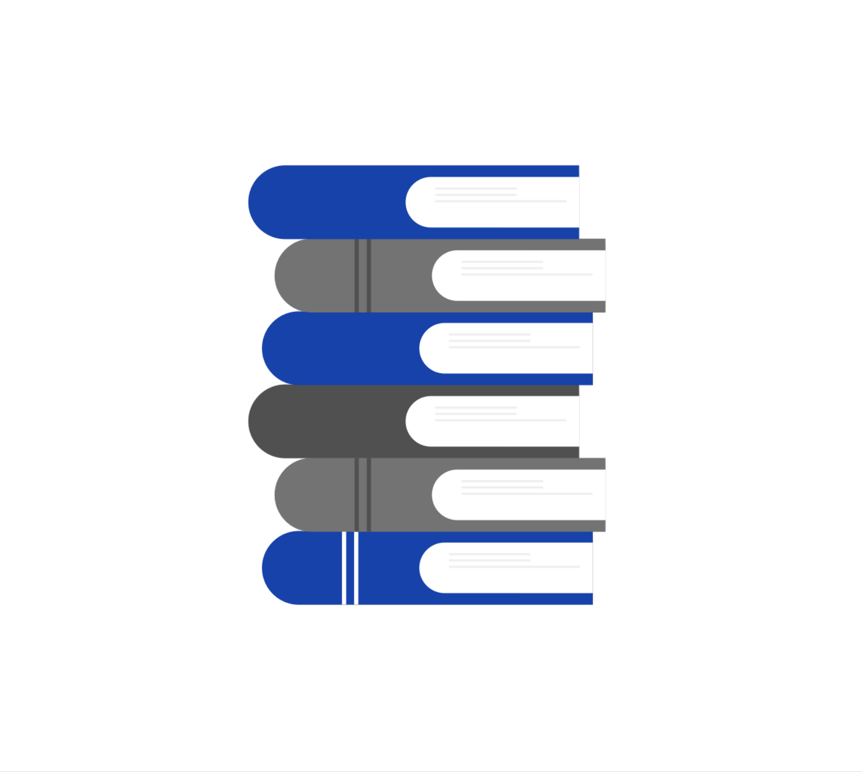
- Training objective:
  - Predict the original inputs for each of the sampled tokens using a bidirectional encoder
  - Make better predictions with each iteration based on cross-entropy loss
  - Gradients that form the basis for weight updates are based on average loss over the sampled learning tokens
- Although all tokens play a role in the self-attention layer, **only the sampled tokens are used for learning**

Masked Language Modeling in BERT

- Same process as shown, but uses subword tokens instead
- 15% of tokens in the training sequence are sampled
- Of these:
  - 80% are replaced with [MASK]
  - 10% are replaced with randomly selected tokens
  - 10% are left unchanged

What if the most useful language segment for our task isn't a single token?

- Lots of tasks have larger units of interest:
  - Question answering
  - Syntactic parsing
  - Coreference resolution
  - Semantic role labeling
- Solution: Apply a **span-oriented** masked learning objective



Masking Spans

- **Span:** A contiguous sequence of one or more words selected from a training sample, prior to subword tokenization
- How can we select spans for masking?
  1. Decide on a span length
     - In SpanBERT, this is sampled from a geometric distribution biased toward shorter spans, with an upper bound of 10
  2. Given this span length, sample a starting location

Masking Spans

- All sampling actions are performed at the span level
  - All tokens in the selected span are replaced with [MASK]
  - All tokens in the selected span are replaced with randomly sampled tokens
  - All tokens in the selected span are left as is
- After sampling actions are performed, the input is passed through the same Transformer architecture seen previously

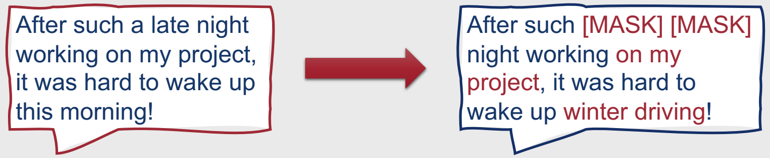
Masking Spans

After such a late night working on my project, it was hard to wake up this morning!

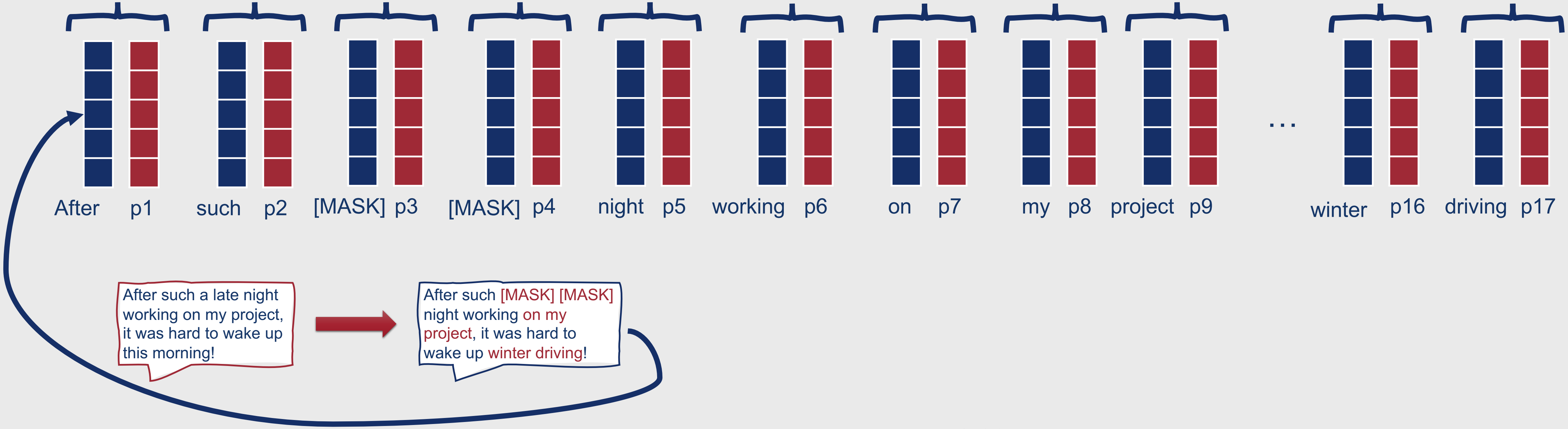
→

After such [MASK] [MASK] night working on my project, it was hard to wake up winter driving!

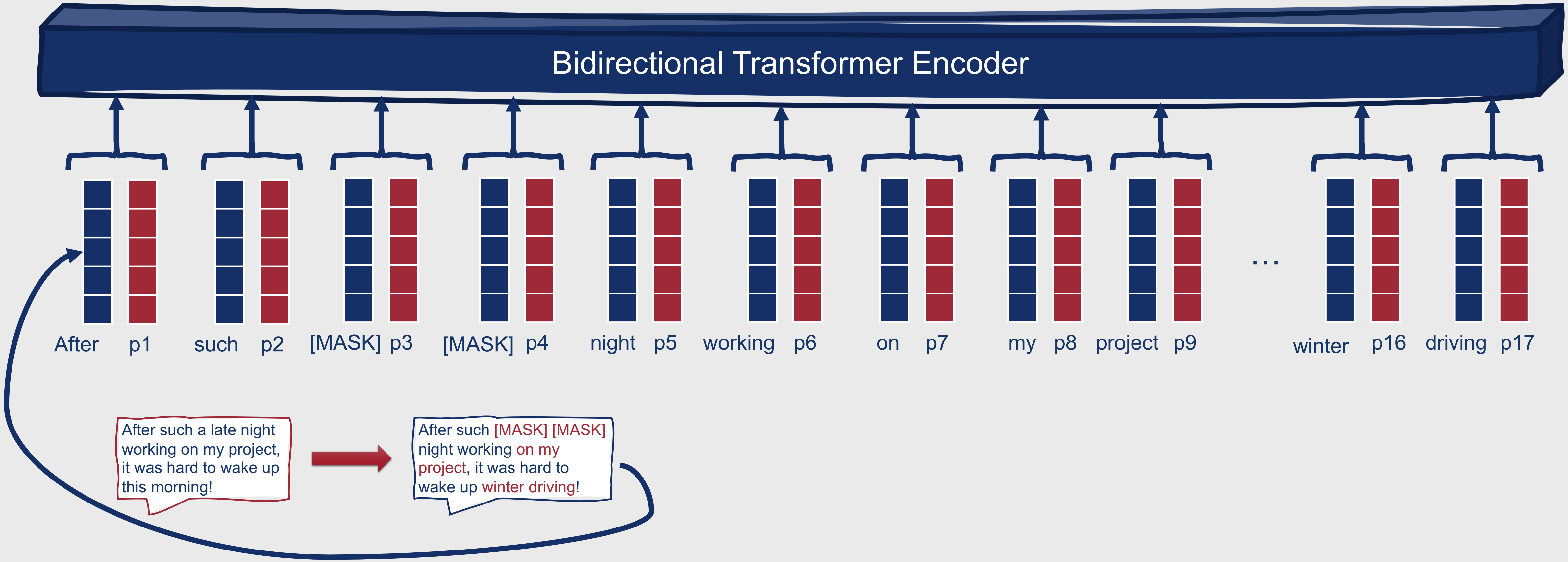
Span-Based Masked Language Modeling



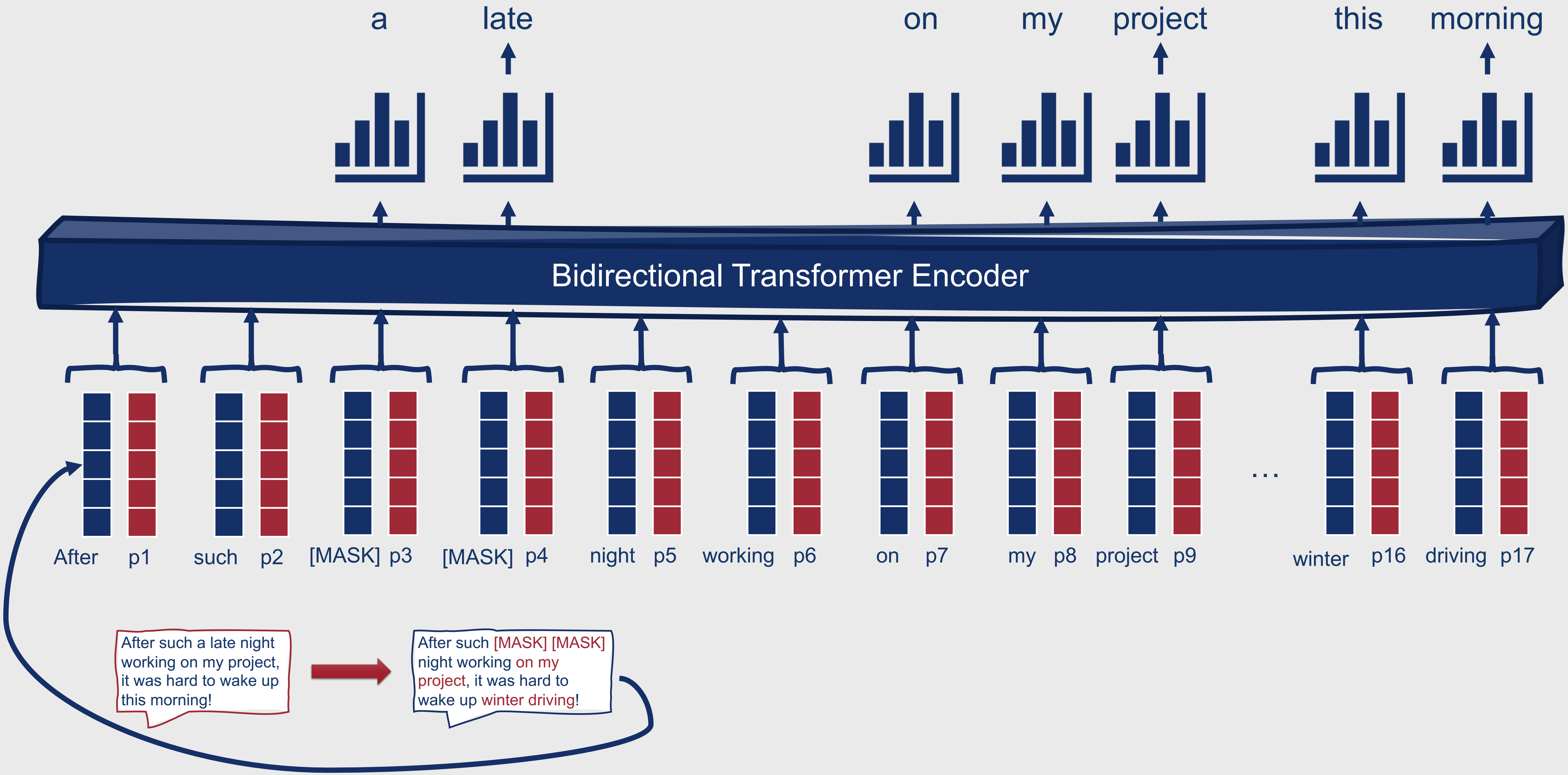
Span-Based Masked Language Modeling



Span-Based Masked Language Modeling



Span-Based Masked Language Modeling



Masked Language Modeling in SpanBERT

- Analogous to "standard" BERT:
  - In 80% of spans, tokens are replaced with [MASK]
  - In 10% of spans, tokens are replaced with randomly sampled tokens
  - In 10% of spans, tokens are left unchanged
- Total token substitution is limited to 15% of the input

How do downstream applications incorporate span representations?

- Create span-level representations from the representations of:
  - Tokens within the span
  - Span boundaries
- Boundary representations are usually derived from:
  - First and last words of the span
  - Words immediately before or after the span

Span Boundary Objective

- Augments the masked language modeling objective in SpanBERT
  - $L(\mathbf{x}) = L_{MLM}(\mathbf{x}) + L_{SBO}(\mathbf{x})$
- Leverages the model's ability to predict words inside a span based on those just outside of it
  - $L_{SBO}(\mathbf{x}) = -\log P(\mathbf{x}|\mathbf{x}_{s-1}, \mathbf{x}_{e+1}, \mathbf{p}_{i-s+1})$

Word before the span

Word after the span

Positional embedding indicating which word in the span is being predicted

Predicting Words within a Span

- The predicted word x_i at position i is produced by:
  - Concatenating the output embeddings for the words before and after the span, and the positional embedding for i
    - $[\mathbf{y}_s; \mathbf{y}_e; \mathbf{p}_{i-s+1}]$
  - Passing the result through a two-layer feedforward network
    - $\mathbf{s}_i = \text{FFNN}([\mathbf{y}_s; \mathbf{y}_e; \mathbf{p}_{i-s+1}])$
  - Finding the selected word using a softmax layer
    - $x_i = \text{softmax}(\mathbf{s}_i)$

Mask-based learning allows the model to produce effective word-level representations.

- **Word-level representations** are important for many NLP applications
- Another source of information that is important in many NLP tasks is the **relationship between pairs of sentences**
  - Detecting paraphrases
  - Determining entailment
  - Measuring discourse coherence

Can we also learn to capture this information using bidirectional Transformer encoders?

- Yes!
- BERT uses two learning objectives, with the second focusing on **next sentence prediction** (NSP)

Next Sentence Prediction

- Present the model with pairs of sentences
- Predict whether each pair consists of an *actual* pair of adjacent sentences, or a pair of unrelated sentences
  - In BERT, training pairs are evenly balanced across these two classes
- Base the loss on how well the model can distinguish actual pairs from unrelated pairs

After such a late night working on my project, it was hard to wake up this morning! I did though, because I had to give my project presentation.

After such a late night working on my project, it was hard to wake up this morning! A winter storm warning has been issued for your area.

How does NSP training work?

- Two new tokens are added to the input:
  - **[CLS]** is prepended to the input sentence pair
  - **[SEP]** is placed *between* the sentences and *after* the final token of the second sentence
- Embeddings representing each segment (first sentence and second sentence) are added to the word and positional embeddings

Additional Tokens

After such a late night working on my project, it was hard to wake up this morning! I did though, because I had to give my project presentation.

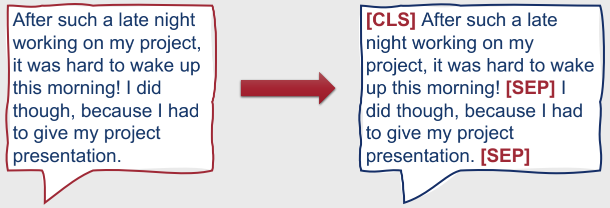
→

[CLS] After such a late night working on my project, it was hard to wake up this morning! **[SEP]** I did though, because I had to give my project presentation. **[SEP]**

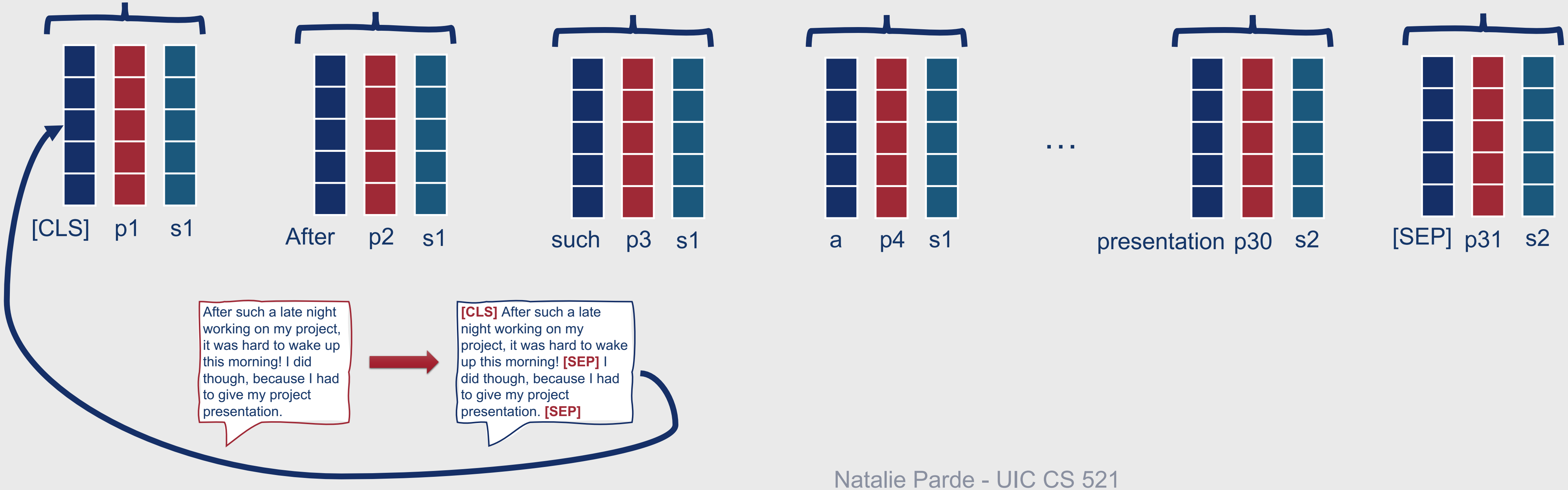
Once we’ve made these adjustments….

- The output vector associated with the [CLS] token represents the next sentence prediction
- Specifically, a learned set of classification weights $\mathbf{W_{NSP}} \in \mathbb{R}^{2 \times d_h}$ is used to predict one of two classes from the raw [CLS] vector $\mathbf{h}_i$
  - $y_i = \text{softmax}(\mathbf{W_{NSP}} \mathbf{h}_i)$
- A cross-entropy loss is used for the NSP loss
- In BERT, the final loss function is a linear combination of the NSP and MLM loss functions

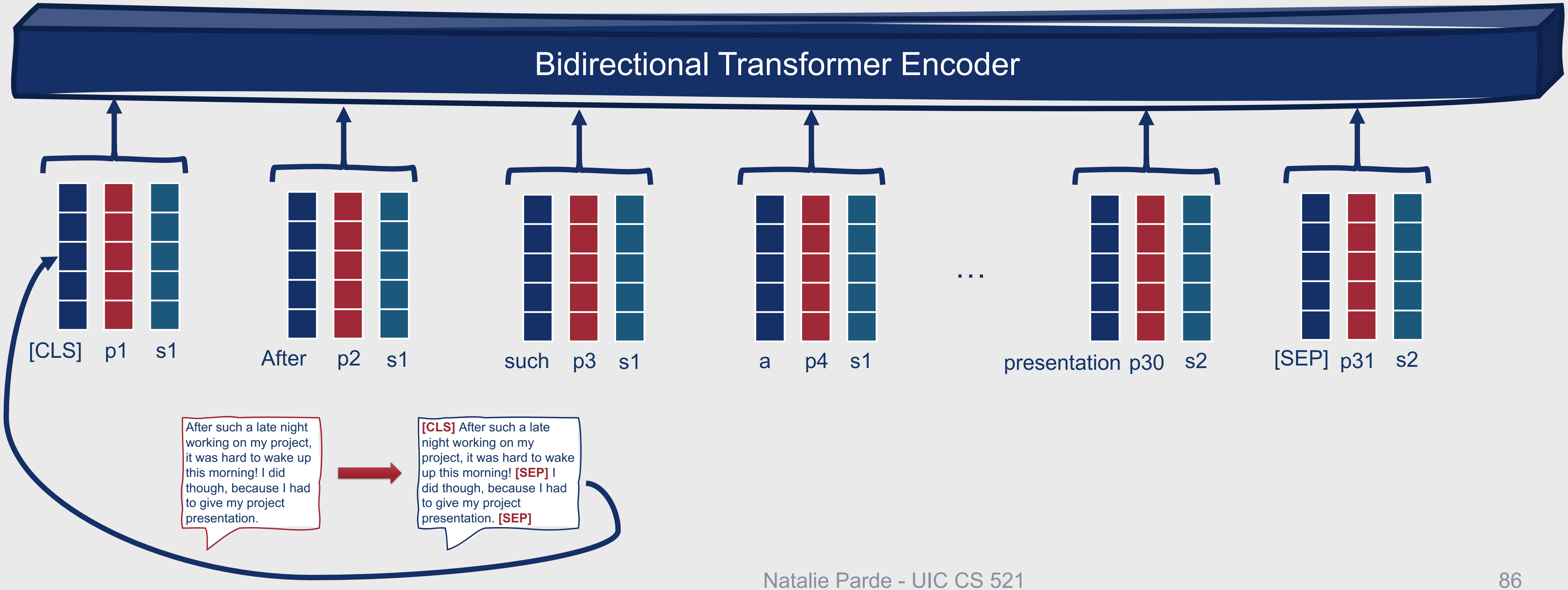
Next Sentence Prediction



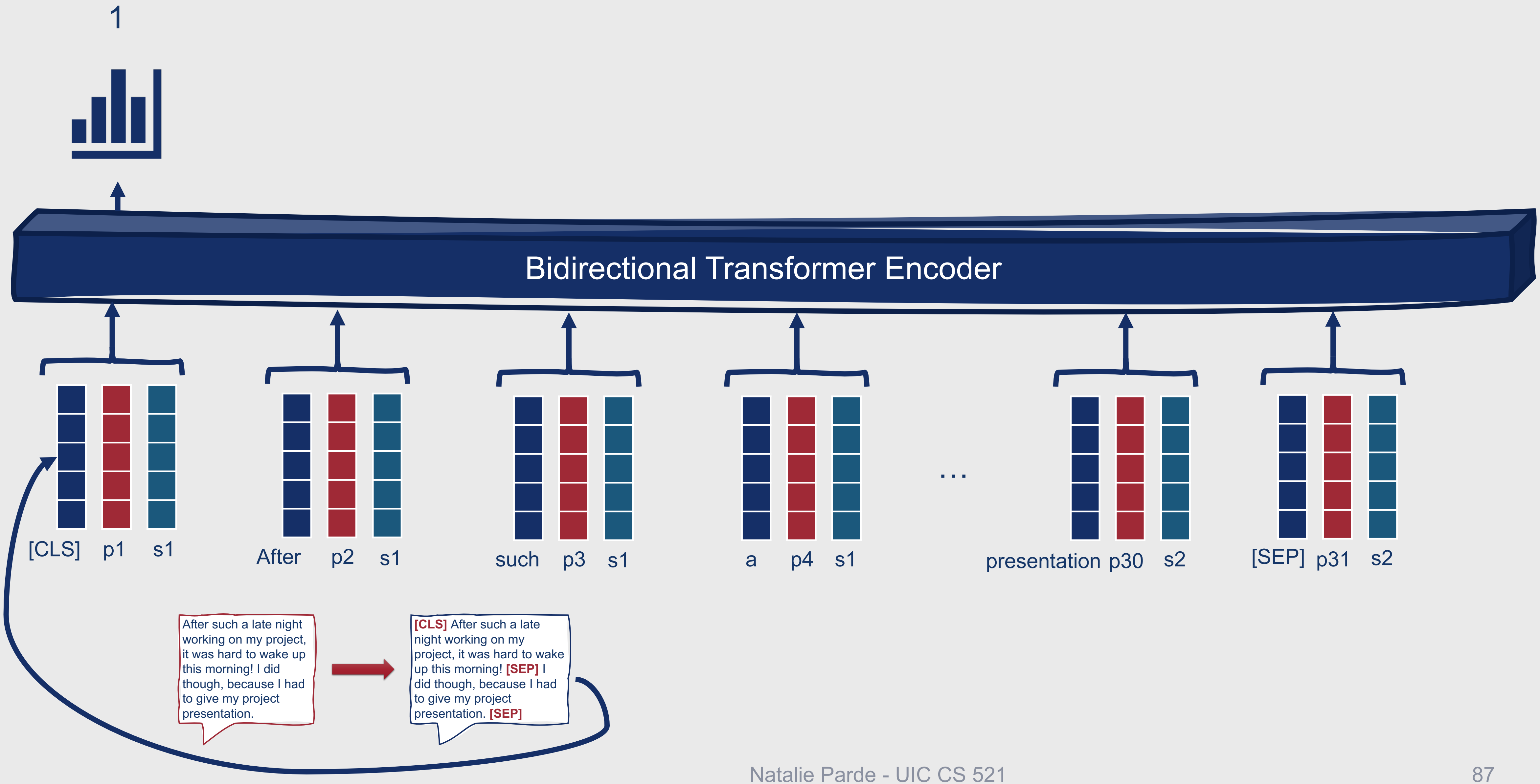
Next Sentence Prediction



Next Sentence Prediction

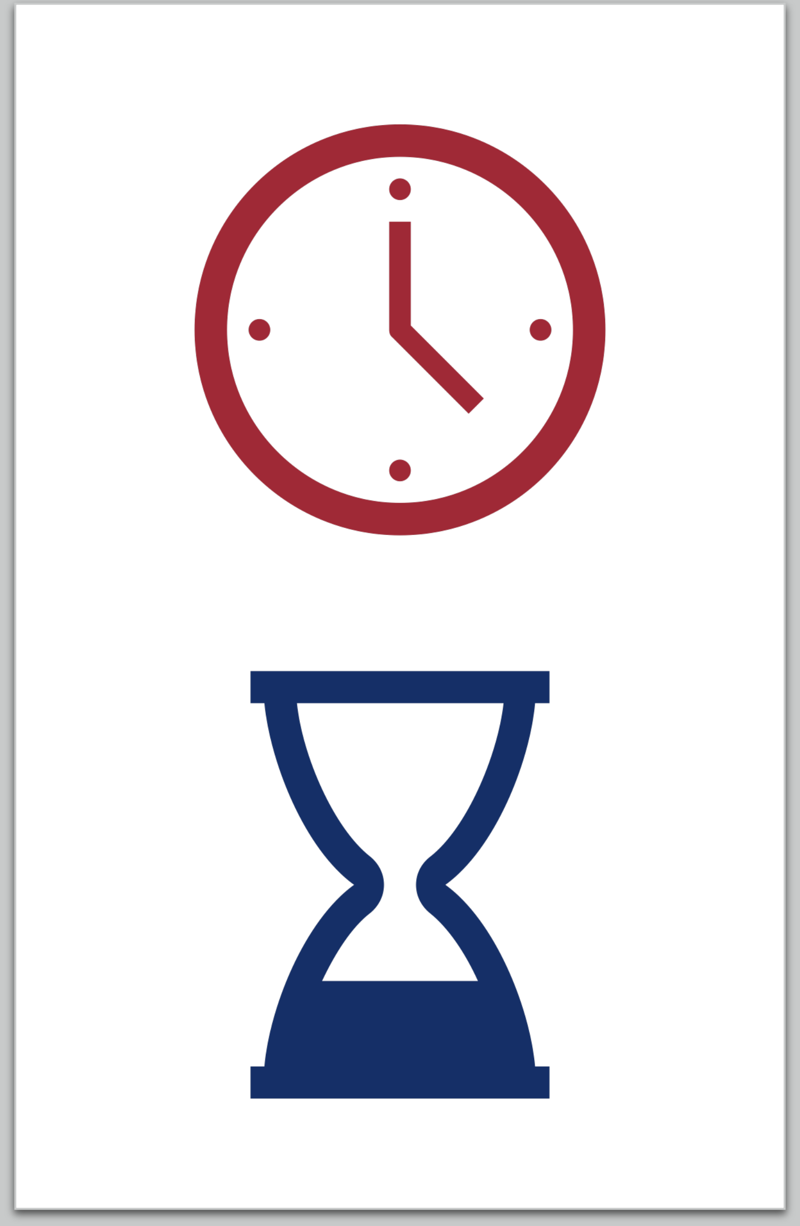


Next Sentence Prediction



Training Regimes

- Corpora:
  - Early Transformer-based language models (including BERT) used BooksCorpus (800M words) and English Wikipedia (2.5B words)
  - More recent state-of-the-art models learn from even larger corpora!
- When training BERT, pairs of sentences were sampled such that their maximum combined length does not exceed 512 tokens
- BERT converged after approximately 40 training iterations (epochs)



Training models like BERT is expensive and time-consuming….

- However, this pretraining process can result in models that can be used and reused for numerous tasks
  - Pretrained word embeddings can be leveraged just like other word embeddings
  - Learned parameters can be used to produce **contextual embeddings** for novel inputs

Contextual Embeddings

- Pass a novel input sentence into a pretrained language model
- Use the output for a given token as its contextual embedding
- Employ contextual embeddings in the same scenarios as static embeddings
  - Word representations for downstream classifiers
  - Corpus analysis

More concretely….

- Given a sequence of text with tokens $x_1, \ldots, x_n$, use the output vector $\mathbf{y}_i$ from the final layer of the pretrained model as the representation of token x_i *in the context of that sequence*
  - In practice, it's common to average across $\mathbf{y}_i$ from the last four layers of the pretrained model

Contextual Embeddings

- This means that contextual embeddings represent **tokens**, whereas static embeddings represented **types**
- Contextual embeddings are particularly useful for:
  - Tasks that require careful disambiguation of polysemous words
  - Tasks that require measuring semantic similarity of words in context
- Contextual embeddings are commonly used to represent input to classifiers during the **fine-tuning** process for downstream applications

Transfer Learning through Fine-Tuning

- Pretrained language models facilitate **generalization** across large text corpora
- This generalization makes it easier to incorporate these models effectively in downstream applications
- The process of learning an interface between a pretrained language model and a specific downstream task is called **fine-tuning**

Fine-Tuning

- Facilitates the creation of downstream applications on top of pretrained language models through the addition of a small set of application-specific parameters
- Labeled data from the downstream task domain is used to train these application-specific parameters
- In general, the pretrained language model is **frozen** or only minimally adjusted during this process

Many different applications have made use of fine-tuning!

- Sequence classification
- Sequence labeling
- Sentence-pair inference
- Span-based operations

Sequence Classification

- Models often represent an input sequence with a single representation
  - Final hidden layer of an RNN model
  - [CLS] vector in a bidirectional Transformer model (e.g., BERT)
  - This representation is sometimes referred to as a **sentence embedding**
- This representation serves as input to a **classifier head** for the downstream task

How do we fine-tune for sequence classification tasks?

- Learn a set of weights, $\mathbf{W_C} \in \mathbb{R}^{n \times d_h}$, to map the sequence representation (e.g., the output vector $\mathbf{y_{CLS}}$ for the [CLS] token) to a set of scores over n possible classes
  - d_h is the dimensionality of the language model's hidden layers

Fine-Tuning for Sequence Classification

- Requires supervised training data for the target task
- Learning process that optimizes $\mathbf{W_C}$ is driven by cross-entropy loss between the softmax output and the target task label

How do we classify test documents for sequence classification tasks?

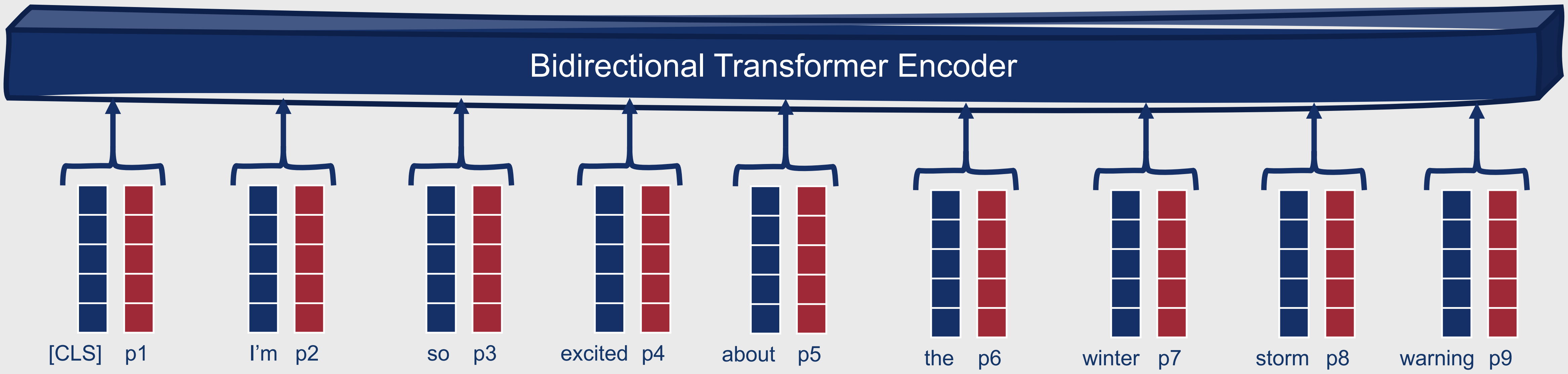
- Pass the input sample through the pretrained language model to generate an output representation $\mathbf{y_{CLS}}$
- Multiply the output representation by the learned weights $\mathbf{W_C}$
- Pass the resulting vector through a softmax:
  - $\mathbf{y} = \text{softmax}(\mathbf{W_C y_{CLS}})$

Example: Sequence Classification

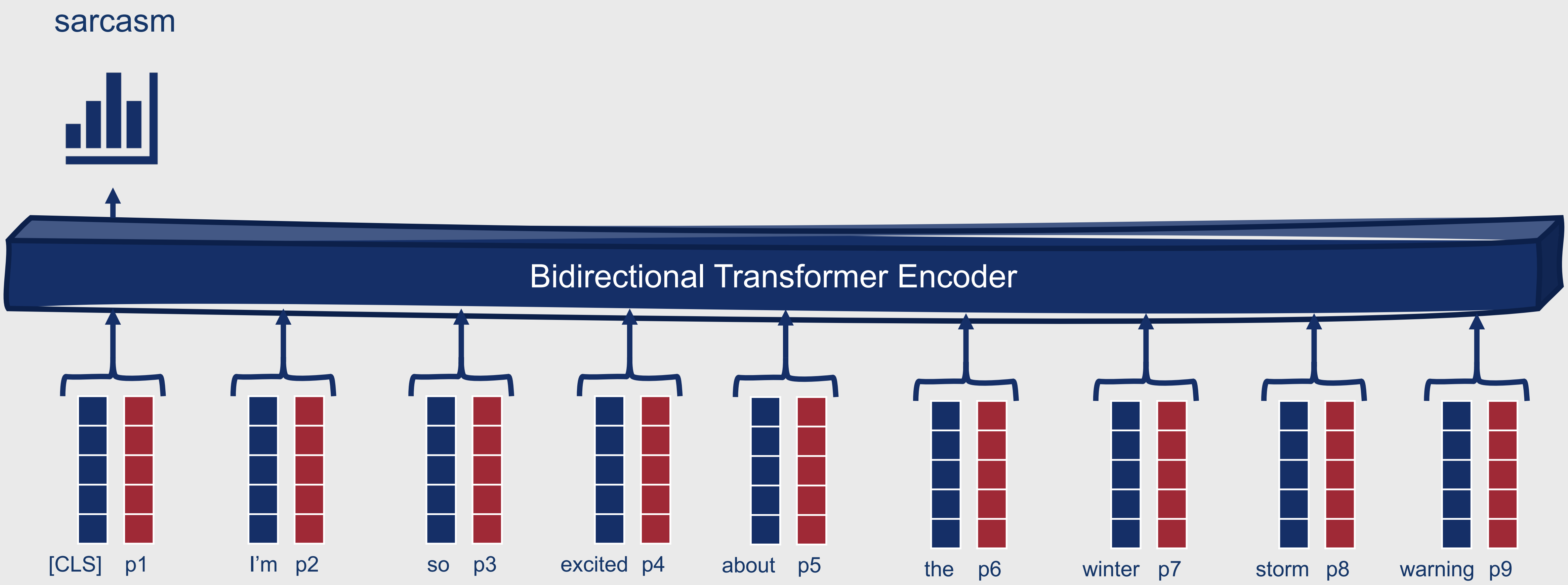
I’m so excited about the winter storm warning.

Example: Sequence Classification

Example: Sequence Classification



Example: Sequence Classification



What differs between this and earlier neural classifiers?

- If we want, we can use the computed loss to update not only the classifier weights, but also the weights for the pretrained language model itself
- Note that substantial changes are rarely necessary!
  - Reasonable classification performance is often achieved with only minimal changes to the language model parameters
  - These changes are generally limited to updates over the final few layers of the model

Pair-Wise Sequence Classification

- Subcategory of sequence classification that focuses on classifying **pairs** of input sentences
- Useful for:
  - Logical entailment
  - Paraphrase detection
  - Discourse analysis

How does fine-tuning work for pair-wise sequence classification?

- Similar to pretraining with the **NSP objective**
  - Pairs of labeled sentences are presented to the model, separated by [SEP] and prepended with [CLS]
- During classification, the output [CLS] vector is multiplied by classification weights and passed through a softmax to generate label predictions

Example: Pair-Wise Sequence Classification (Entailment Task)

- Popular NLP task, also referred to as **natural language inference**
- Classify sentence pairs such that:
  - Sentence A **entails** Sentence B
  - Sentence A **contradicts** Sentence B
  - The relationship between Sentence A and Sentence B is **neutral**

Example: Pair-Wise Sequence Classification (Entailment Task)

It’s a snow day! There
is snow outside.

Example: Pair-Wise Sequence Classification (Entailment Task)

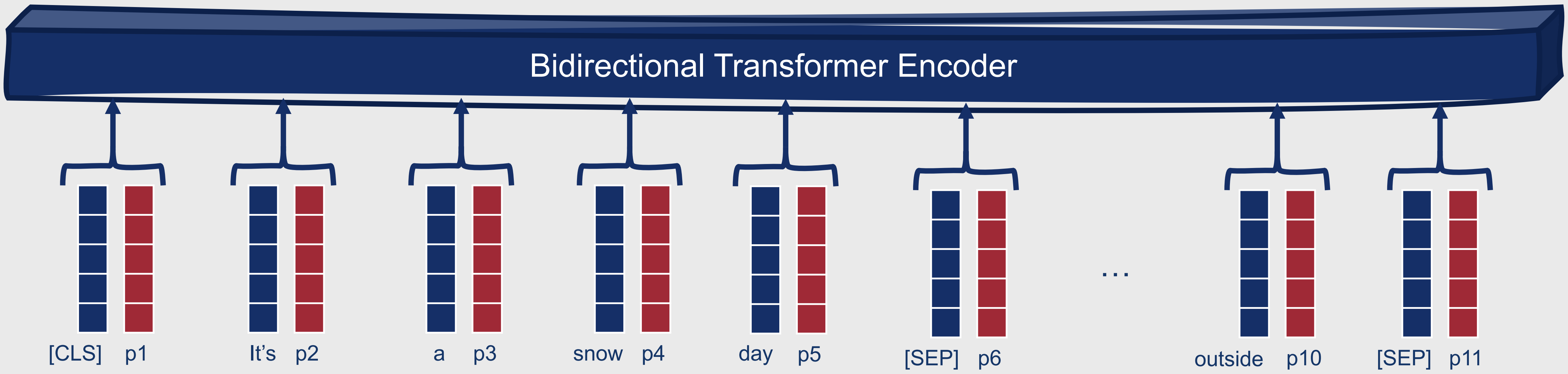
It's a snow day! There is snow outside.

[CLS] It's a snow day! [SEP] There is snow outside. [SEP]

Example: Pair-Wise Sequence Classification (Entailment Task)

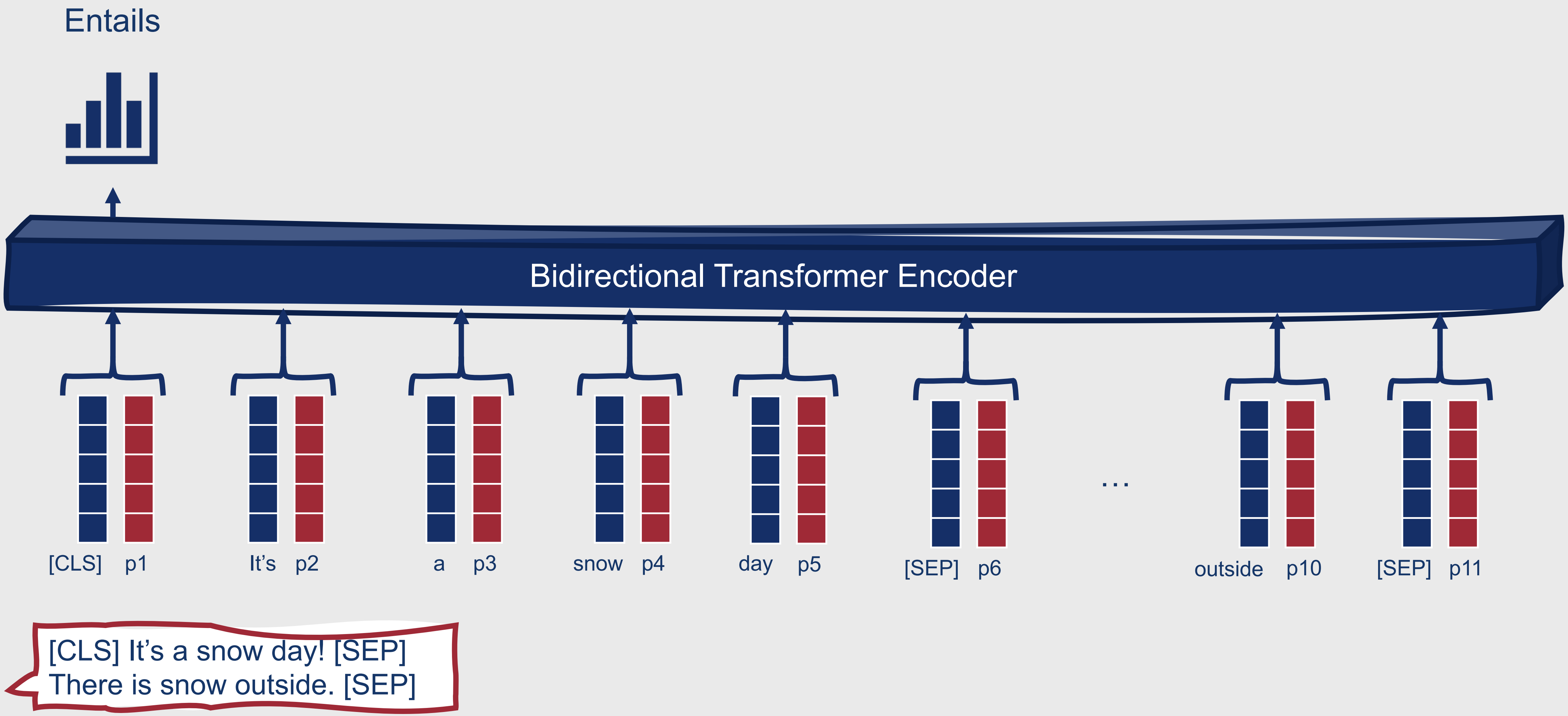
[CLS] It's a snow day! [SEP]
There is snow outside. [SEP]

Example: Pair-Wise Sequence Classification (Entailment Task)



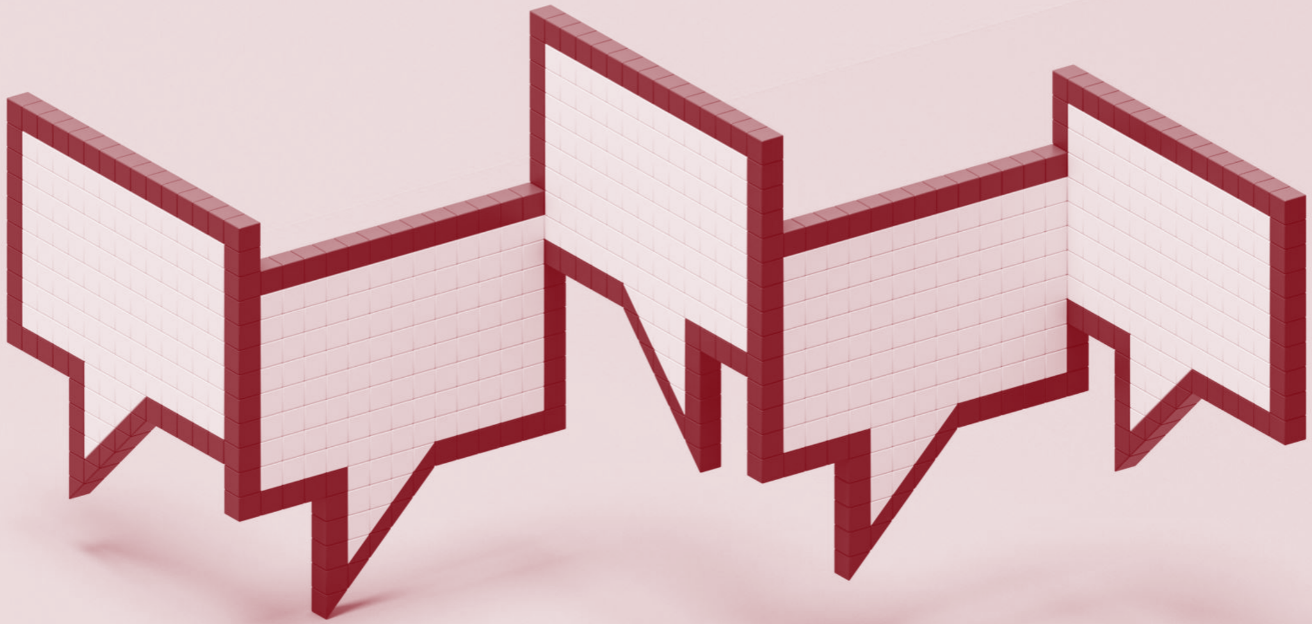
[CLS] It's a snow day! [SEP]
There is snow outside. [SEP]

Example: Pair-Wise Sequence Classification (Entailment Task)



Sequence Labeling

- Similar to approach used for sequence classification
- However, the output vector for **each input token** is passed to a classification head that produces a softmax distribution over the possible classes
- The output tag sequence can be determined by a variety of methods
  - Common: **Greedy approach** accepting the argmax class for each token
    - $\mathbf{y}_i = \text{softmax}(\mathbf{W}_K \mathbf{z}_i)$, where $k \in K$ is the set of tags for the task
    - $\mathbf{t}_i = \underset{k}{\text{argmax}}(\mathbf{y}_i)$
  - Alternative: Distribution over labels can be passed to a **CRF layer**, allowing consideration of global tag-level transitions



Common Sequence Labeling Tasks

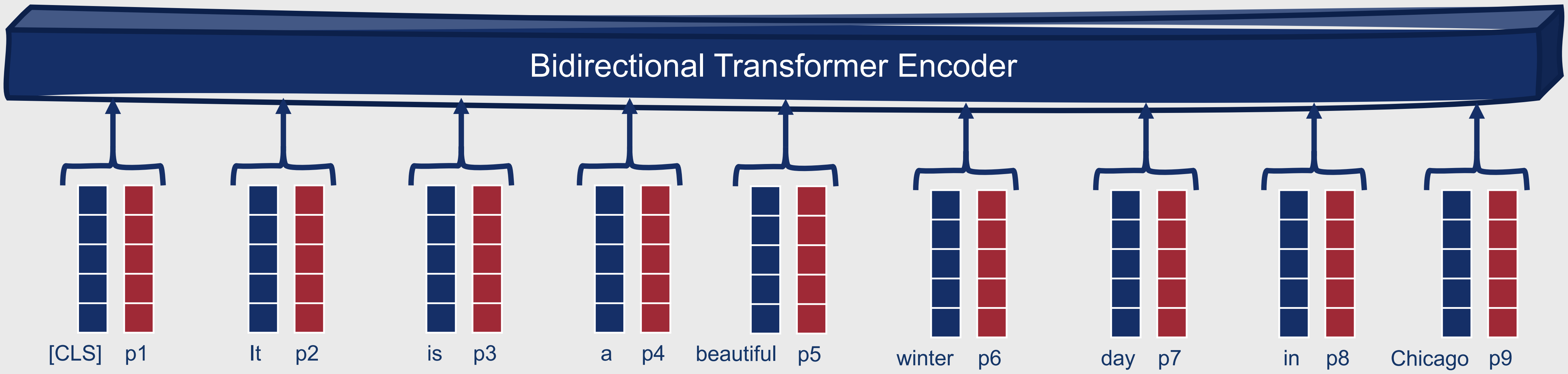
- Part-of-speech tagging
- Named entity recognition
- Shallow parsing

Example: Sequence Labeling

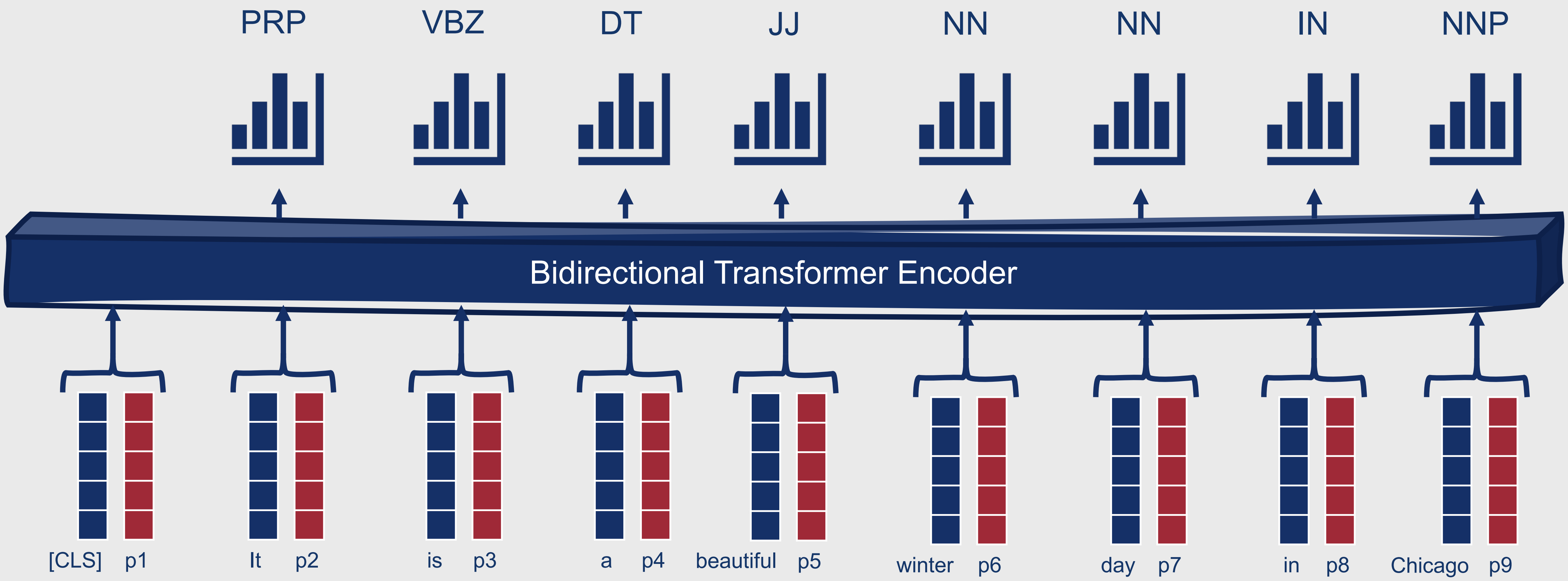
It is a beautiful winter day in Chicago.

Example: Sequence Labeling

Example: Sequence Labeling



Example: Sequence Labeling

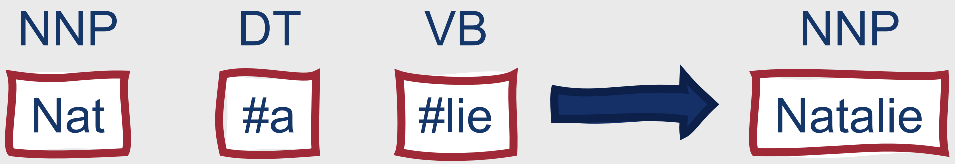


Complication with BERT (and related models)….

- Subword tokenization doesn't play well with tasks requiring word-level labels
- How to address this?
  - During training, assign the gold standard label for a word to all its constituent subwords
  - During testing, recover word-level labels from subwords as part of the decoding process

Recovering Word-Level Labels

- Simplest approach:
  - For a given word, use the predicted label for its first subword as the label for the entire word
- More complex approaches consider the distribution of label probabilities across all subwords for a given word



Span-Based Sequence Labeling

- Carries attributes of both sequence classification and token-level sequence labeling
  - Goal: **Generate and/or leverage representations of spans** of tokens
- Common Tasks:
  - Identify spans of interest
  - Classify spans
  - Determine relations among spans

Common Span-Based Sequence Labeling Applications

- Named entity recognition
- Question answering
- Syntactic parsing
- Semantic role labeling
- Coreference resolution

Span-Based Sequence Labeling

- Given an input sequence x comprising T tokens, $(x_1, x_2, \ldots, x_T)$ a span is a contiguous sequence of tokens from x_i to x_j such that $1 \leq i \leq j \leq T$
- This results in $\frac{T(T-1)}{2}$ total spans
- In practice, most span-based models impose an application-specific length limit L
- Legal spans are thus those were $j - i < L$
- Let the set of legal spans in x be represented as $S(x)$

How do we represent spans for span-based sequence labeling?

- Most span representations incorporate both:
  - **Span boundary representations**
  - **Summary representations** of span content
- These component representations are often concatenated with one another

Span Boundary Representations

- Simple approach: Just use the contextual embeddings of the start and end tokens of the span as the span boundary representations
  - However, internally this doesn't offer a way to distinguish *between* the start and end tokens
  - Words may carry different meaning at the beginning of a span than at the end!
- More complex approach: Use separate feedforward networks to learn representations for the beginning and end of the span
  - $\mathbf{s}_i = \mathrm{FFNN}_s(\mathbf{h}_i)$
  - $\mathbf{e}_j = \mathrm{FFNN}_e(\mathbf{h}_j)$

Summary Representations

- Simple approach: Just use the average of the output embeddings for words within the span as the summary representation
  - $\mathbf{g}_{ij} = \frac{1}{(j-i)+1} \sum_{k=i}^{j} \mathbf{h}_k$
- More complex approach: Place more representational emphasis on the head of the span
  - Can be done using syntactic parse information (if available) or a self-attention layer (if not)
  - $\mathbf{g}_{ij} = \text{SelfAttention}(\mathbf{h}_{i:j})$

How does fine-tuning work in span-based sequence labeling?

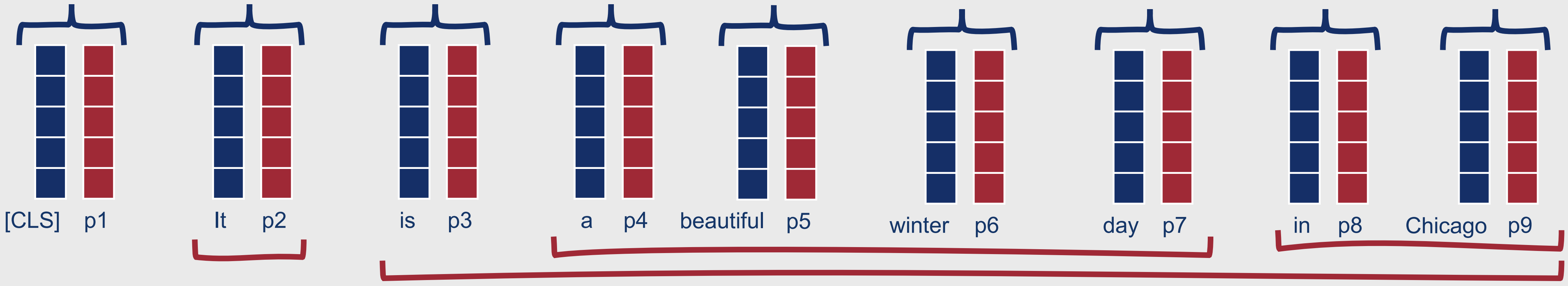
- Learn the weights/parameters for:
    - Task classification head
    - Boundary representations
    - Summary representation
- Final classification output:
    - $\mathbf{span}_{ij} = [\mathbf{s}_i; \mathbf{e}_j; \mathbf{g}_{ij}]$
    - $\mathbf{y}_{ij} = \mathrm{softmax}(\mathrm{FFNN}(\mathbf{span}_{ij}))$

Example: Span-Based Sequence Labeling

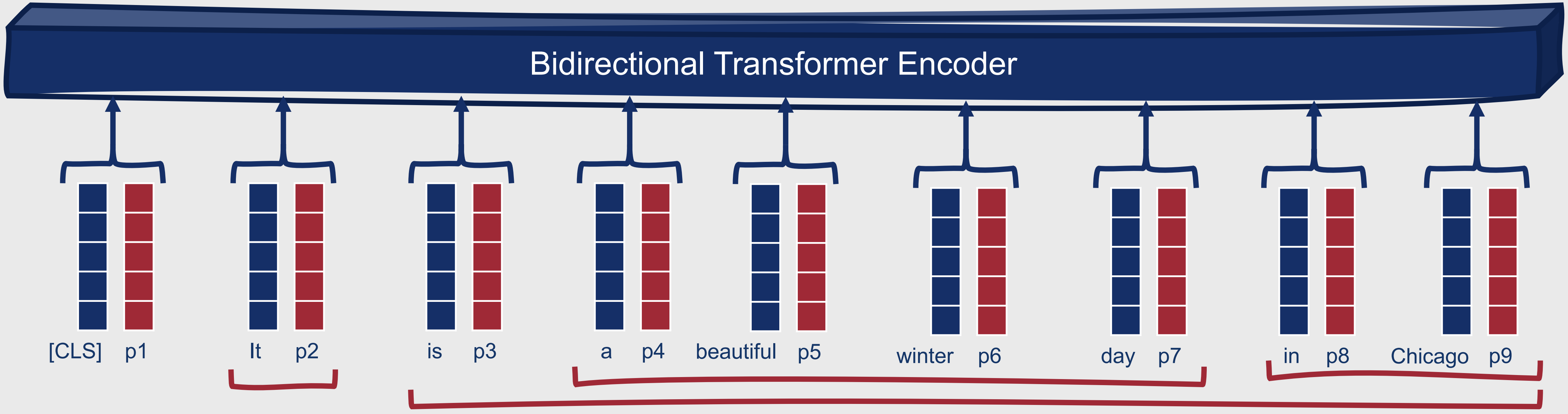
It is a beautiful winter day in Chicago.

Example: Span-Based Sequence Labeling

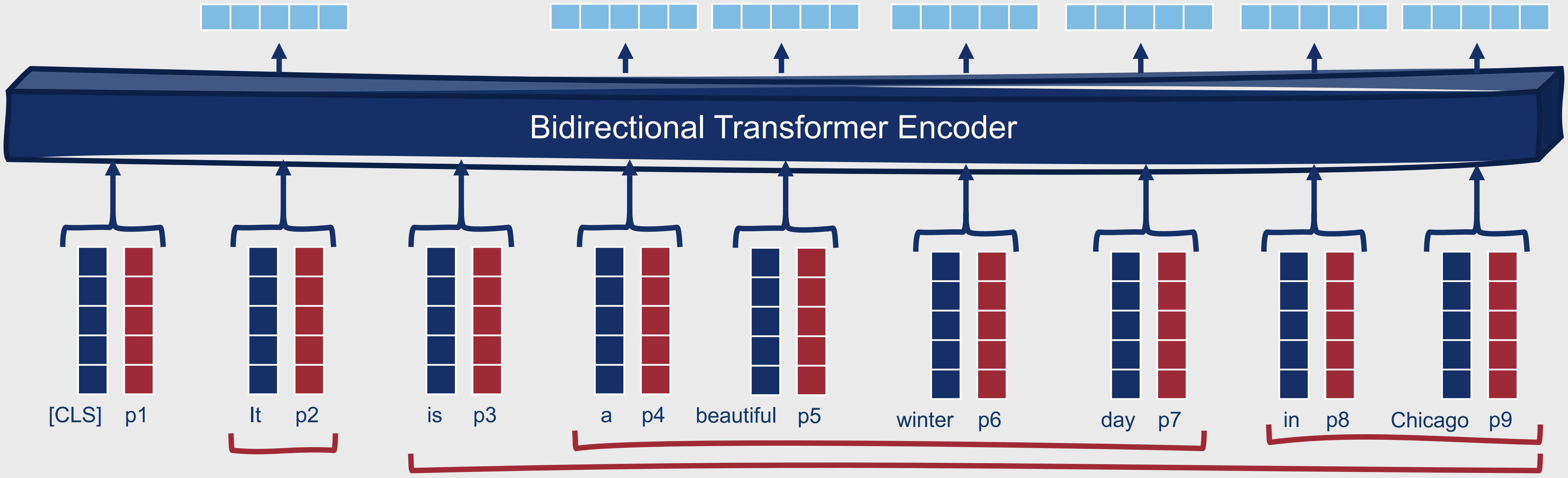
Example: Span-Based Sequence Labeling



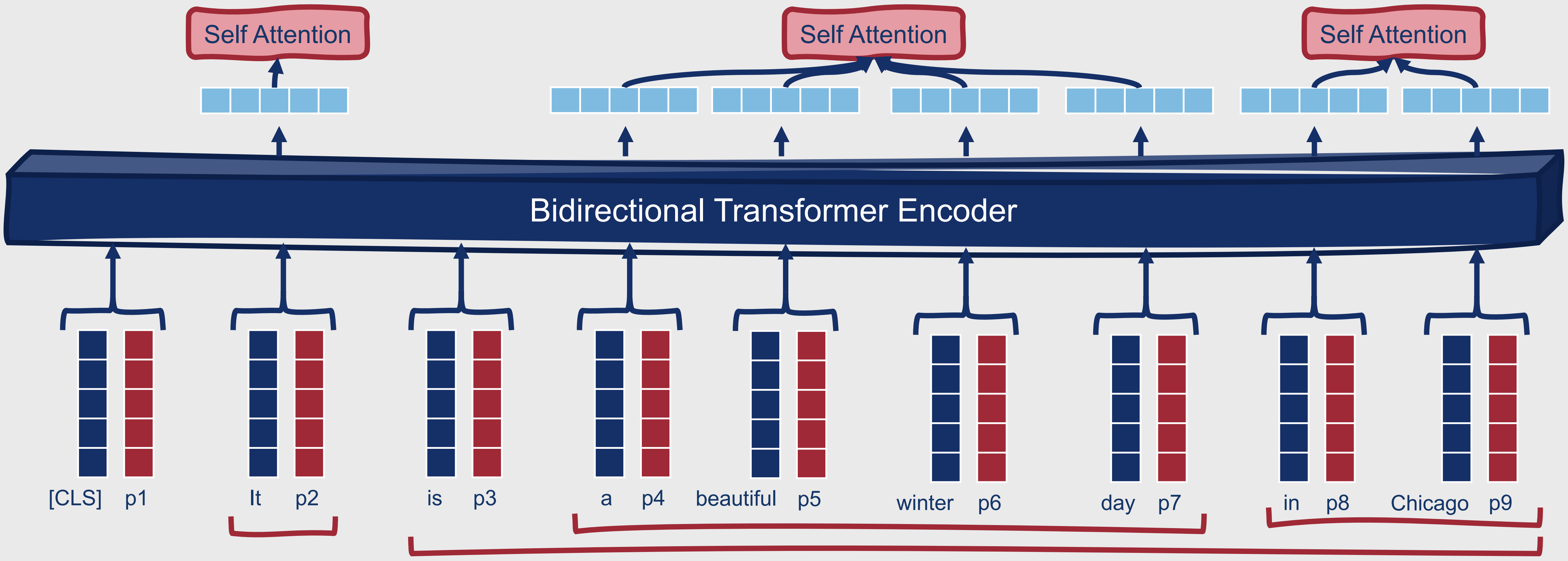
Example: Span-Based Sequence Labeling



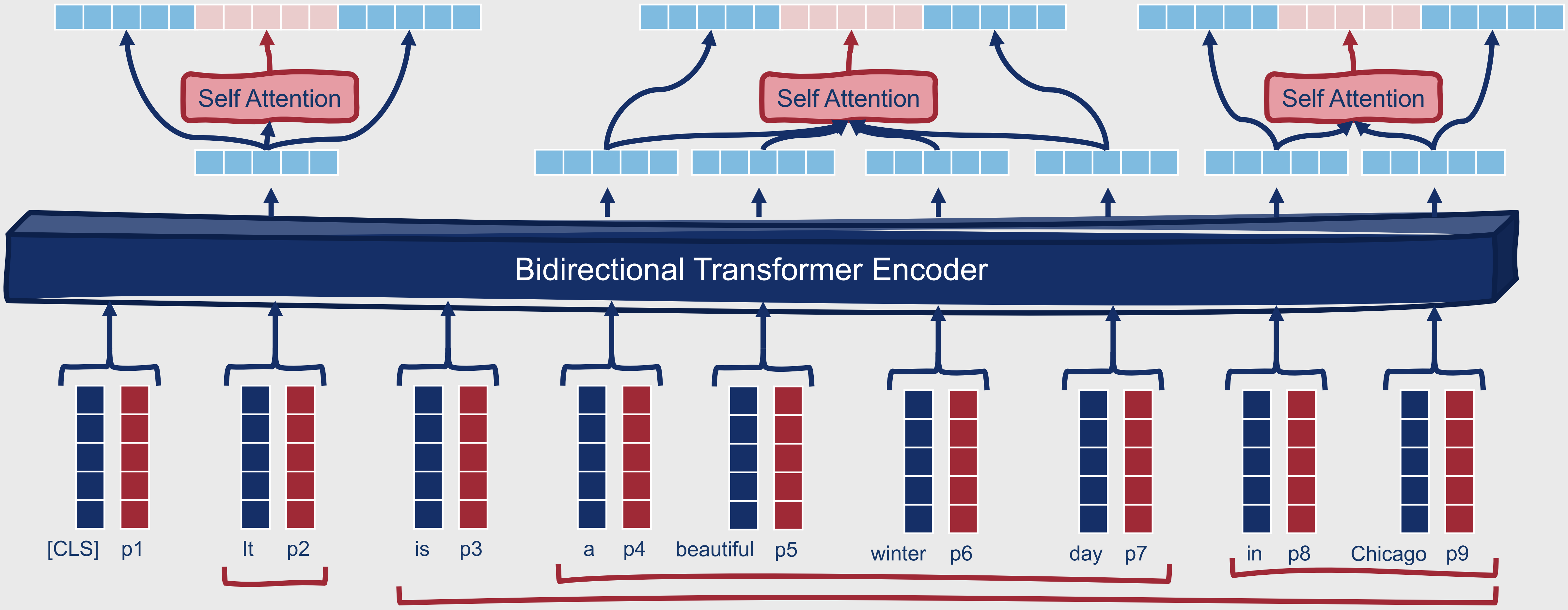
Example: Span-Based Sequence Labeling



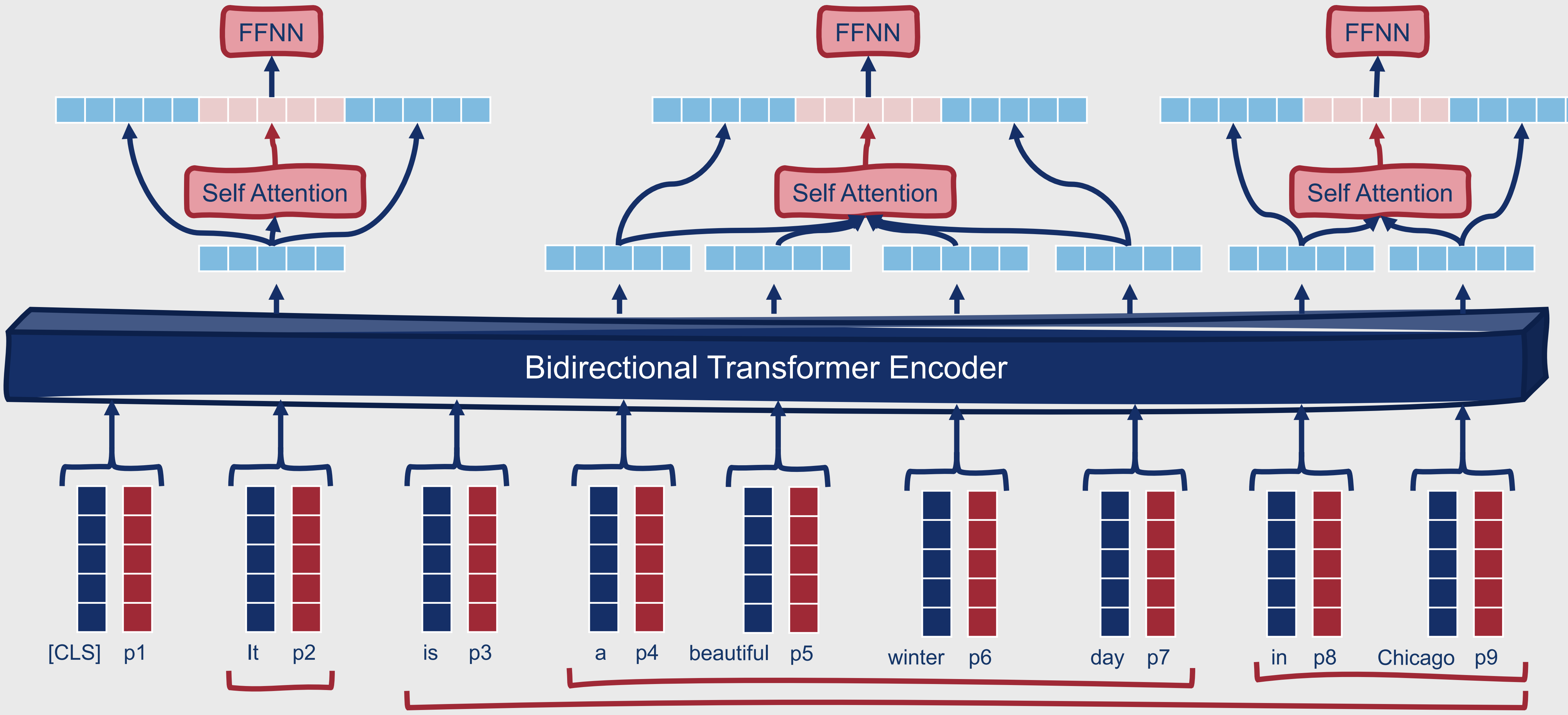
Example: Span-Based Sequence Labeling



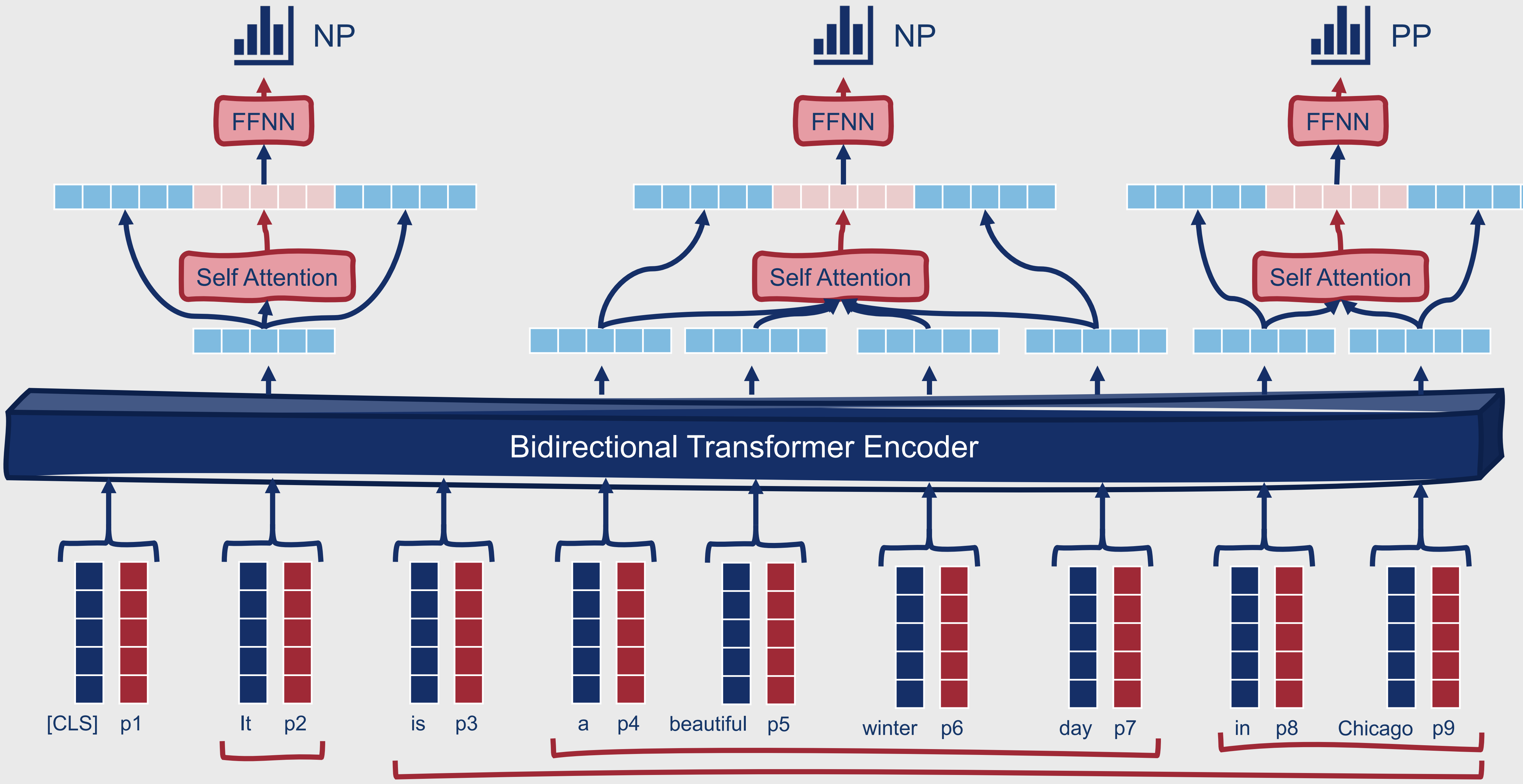
Example: Span-Based Sequence Labeling



Example: Span-Based Sequence Labeling



Example: Span-Based Sequence Labeling



Advantages of Span-Based Sequence Labeling

- Only require one label assignment per span
  - In comparison, BIO-based methods require labels for each constituent token
- Naturally accommodate hierarchical and/or overlapping labels
  - BIO-based methods assign a single label per token

We've learned a lot about transfer learning, pretrained models, and contextual embeddings …how can we implement them?

- 🤗
  - https://huggingface.co/docs/transformers/index
- TensorFlow
  - https://www.tensorflow.org/text/tutorials/classify_text_with_bert
- PyTorch
  - https://pytorch.org/hub/huggingface_pytorch-transformers/

Summary: Transfer Learning with Pretrained Language Models and Contextual Embeddings

- Word embeddings can be **static** or **contextual**
- **Contextual word embeddings** differ for each instance of the same vocabulary word depending on the surrounding context
- **Bidirectional Transformer encoders** are one way to generate contextual word embeddings
- Bidirectional Transformer encoders learn representations by optimizing for two tasks:
  - **Masked language modeling**
  - **Next sentence prediction**
- **Pretrained language models** can be **fine-tuned** for a variety of downstream tasks by adding classification heads to the end of the model and (optionally) updating the weight parameters in its last few layers