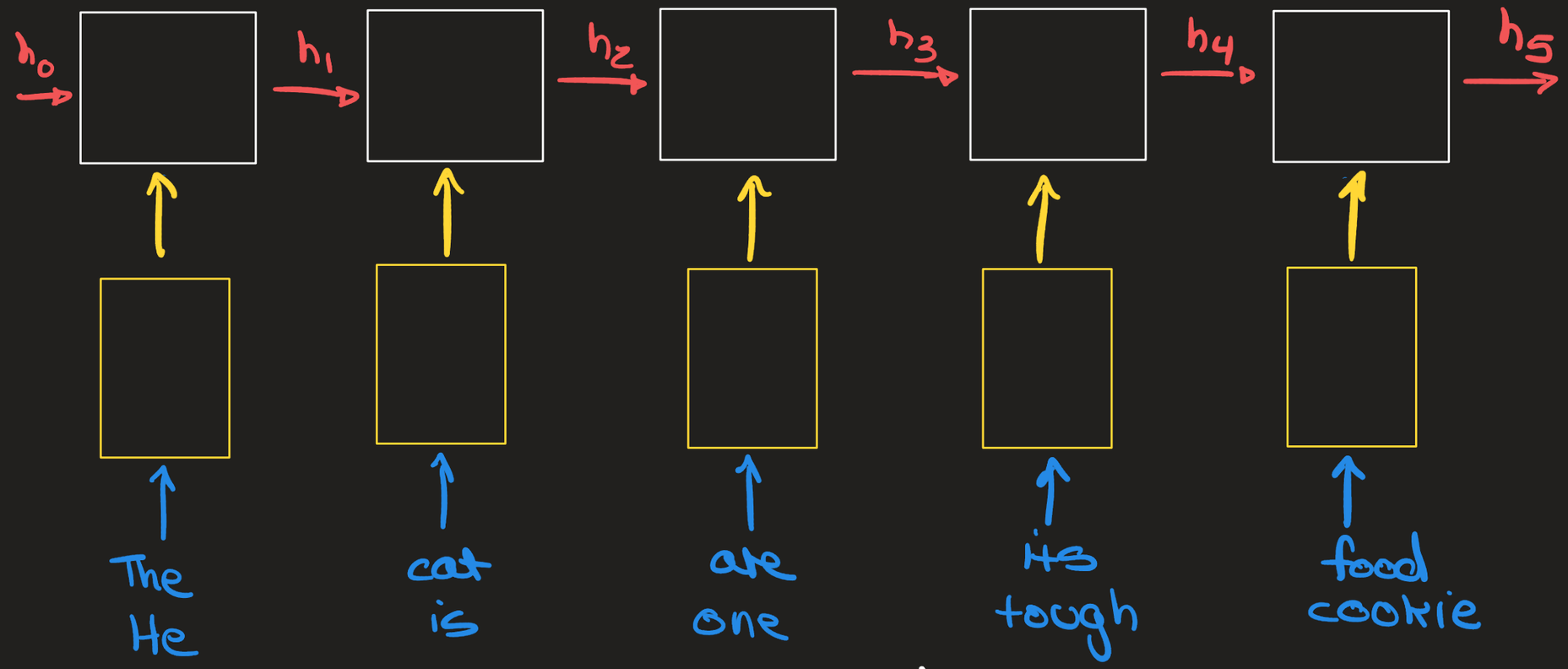
Machine Learning and Optimization Lecture 22

- Self-Attention as a replacement for many-to-many RNNs, for obtaining contextual representations
- Positional Encodings to counter the permutation invariance of Self-Attention
- The BERT model
- Unsupervised pretraining and supervised fine-tuning

Contextual Representations with RNNs

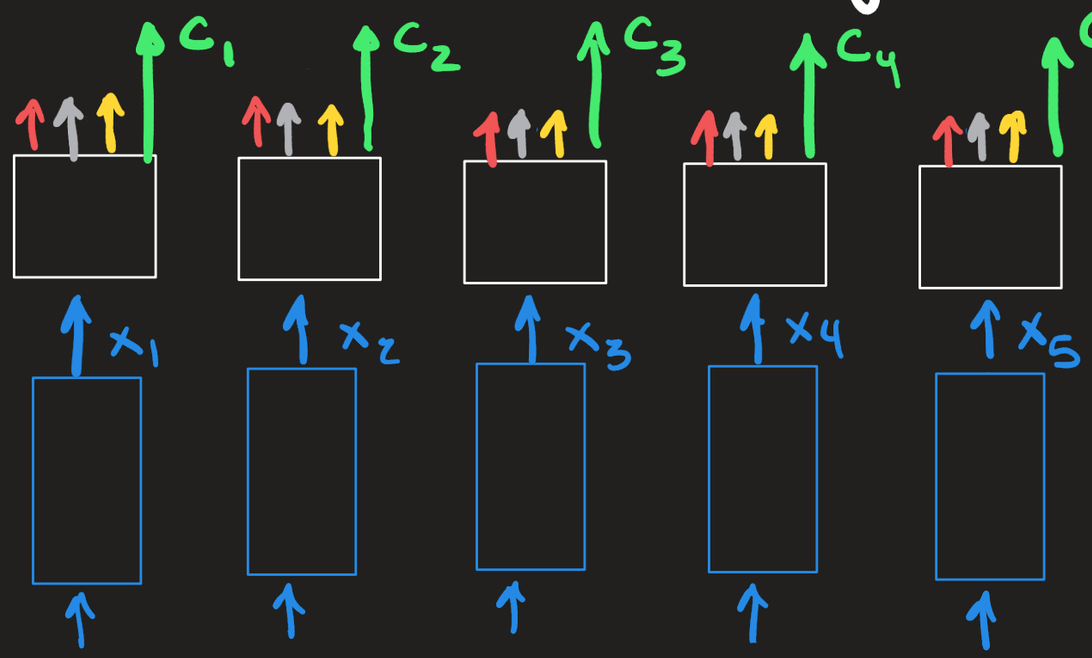
We saw that RNNs form contextual representations $h_1, \cdots, h_4$ of the input sequence $x_1, \cdots, x_4$ that are useful for downstream tasks.



When using RNNs we learn shared global embeddings for each token, which are turned into contextual representations

The drawbacks of using RNNs is that they are inherently serial (can be mitigated w/ bidirectional RNNs) and so they lose information about earlier portions of the sequence that are relevant for representing later tokens

Solution: use attention to get contextual representations



$$k_i = \omega_K x_i$$
$$q_i = \omega_Q x_i$$
$$v_i = \omega_V x_i$$

} key, query, value representation

For the ith token's contextual representation

$$\alpha_i = \text{softmax}\left(\begin{bmatrix} k_1^T \\ \vdots \\ k_5^T \end{bmatrix} q_i\right)$$

$$c_i = \sum_{j=1}^{5} (\alpha_i)_j \, v_j$$

(Note: you could add bias vectors to k_i, q_i, v_i but we typically don't)

Now the contextual representation of each token directly depends on each other token in the sequence

- $k_i = W_K x_i$: learn W_K to extract information from the global embeddings about how relevant x_i is to determining contextual representations of other tokens
- $q_i = W_Q x_i$: learn W_Q to extract information from the global embeddings about how relevant other tokens are in determining contextual representations of x_i
- $v_i = W_V x_i$: learn W_V to extract information from x_i that is useful in forming contextual representations of other tokens
- α_i : the weights determining the relative importance of all tokens in forming the contextual representation of x_i

Self-Attention

Given a sequence of vectors $X = [x_1 | \cdots | x_T] \in \mathbb{R}^{d_x \times T}$ and parameters

$W_K \in \mathbb{R}^{d_{attn} \times d_x}$, $W_Q \in \mathbb{R}^{d_{attn} \times d_x}$, $W_V \in \mathbb{R}^{d_{out} \times d_x}$,

compute

$$K = W_K X = [k_1 | \cdots | k_T] \in \mathbb{R}^{d_{attn} \times T}$$

$$Q = W_Q X = [q_1 | \cdots | q_T] \in \mathbb{R}^{d_{attn} \times T}$$

$$V = W_V X = [v_1 | \cdots | v_T] \in \mathbb{R}^{d_{out} \times T}$$

and note that

$$K^T Q = \begin{bmatrix} k_1^T \\ \hline \vdots \\ \hline k_T^T \end{bmatrix} [q_1 | \cdots | q_T] = \begin{bmatrix} k_1^T q_1 & \cdots & k_1^T q_T \\ \vdots & \ddots & \\ k_T^T q_1 & \cdots & k_T^T q_T \end{bmatrix}$$

This implies that

$$\text{softmax}(K^T Q) = [\alpha_1 | \cdots | \alpha_T] \in \mathbb{R}^{T \times T}$$

Therefore the contextual representations obtained through self-attention are

$$C = V\,\text{softmax}(K^T Q) = [v_1 | \cdots | v_T][\alpha_1 | \cdots | \alpha_T]$$

$$= \left[(\alpha_1)_1 v_1 + \cdots + (\alpha_1)_T v_T \;\middle|\; \cdots \;\middle|\; (\alpha_T)_1 v_1 + \cdots + (\alpha_T)_T v_T\right]$$

$$= [c_1 | \cdots | c_T]$$

In practice, for numerical stability we take

$$\text{Attention}(X \,|\, \omega_K, \omega_Q, \omega_V) = V\,\text{softmax}\left(\frac{K^T Q}{\sqrt{d_{attn}}}\right)$$

One drawback of self-attention is that it is permutation invariant: if $X_1 = \pi X_2$ where π is a permutation matrix, then you can check that

$$\text{Attention}(X_1) = \pi\,\text{Attention}(X_2)$$

This is not desirable in e.g. NLP because changing the order of words should change their contextual representations.

E.g. the sequences $\{\text{The}, \text{cat}, \text{are}, \text{its food}\}$ and $\{\text{The}, \text{food}, \text{are}, \text{its}, \text{cat}\}$ should have different contextual representations.

To ensure this, we add positional encodings to the inputs:

$$\tilde{X} = X + P = [x_1 + p_1 \mid \cdots \mid x_T + p_T]$$

The original Transformer paper takes

$$P = [p_1 | \cdots | p_T]$$

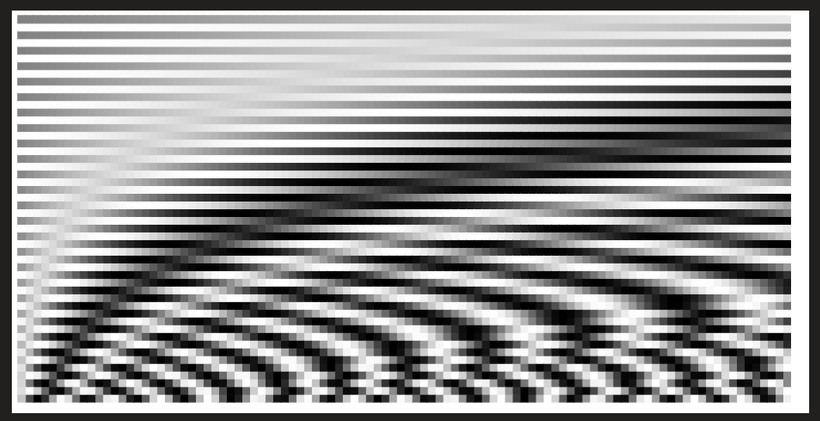
with

$$(p_t)_{2i-1} = \sin\left(t\omega^{2i/d_x}\right)$$

$$(p_t)_{2i} = \cos\left(t\omega^{2i/d_x}\right)$$

where $\omega = \frac{1}{T}$

Ex: $(T = 100)$

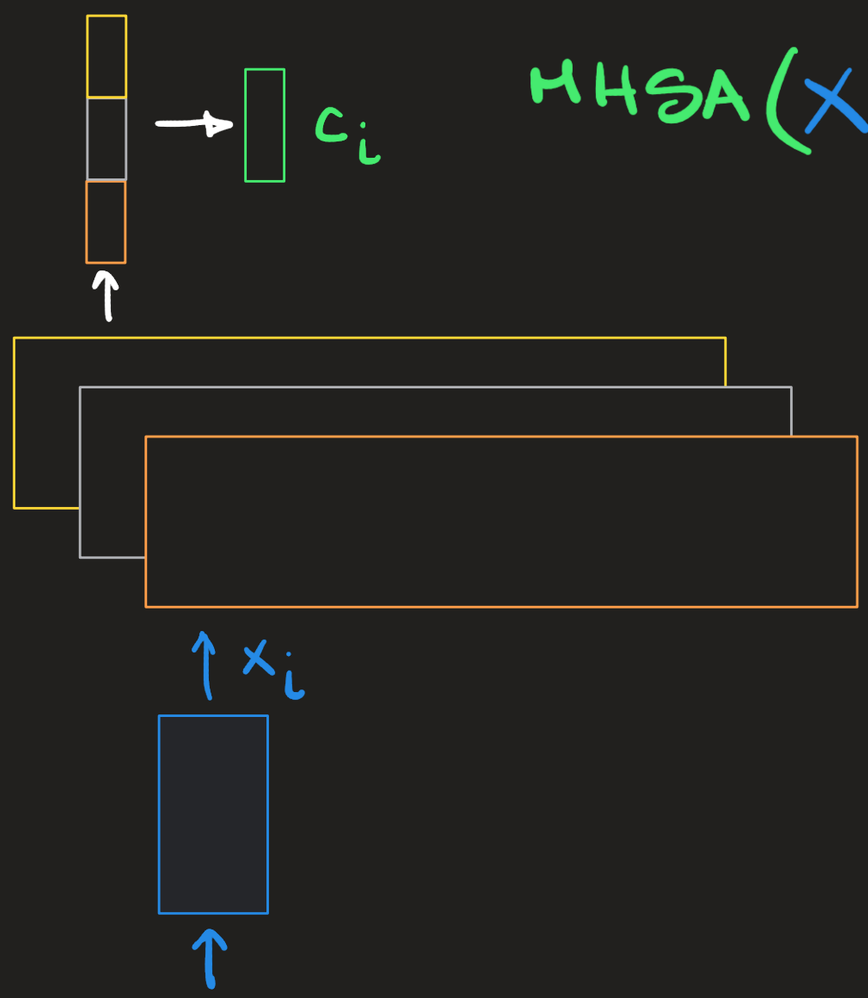


p_1 . . . . p_{100}

white corresponds to -1
black corresponds to 1

Multi-headed Self-Attention

Just as in CNNs, we prefer to learn multiple features (in this case, contextual representations) to extract multiple relevant aspects of the data:



$$MHSA(X) = \omega^0 \Big\|_{h=1}^{H} Attention(X \mid \omega_K^h, \omega_Q^h, \omega_V^h)$$

$\|$ means concatenation

$$= \omega^0 \begin{bmatrix} c_1^1 & \cdots & c_T^1 \\ \hline \\ c_1^H & \cdots & c_T^H \end{bmatrix}$$

Using Self-Attention in DNNs

DNNs transform their inputs into more useful/semantically relevant representations, by composing a series of layers.

MLP Layer

Input: a vector $[x_1, \dots, x_n]$

Output: a vector $[o_1, \dots, o_{n'}]$

Operation: mix the vector's entries linearly, $a = Wx + b$, then pass entrywise through nonlinearity $o = \sigma(a)$

CNN Layer

Input: channels $[X_1, \dots, X_c]$

Output: channels $[O_1, \dots, O_{c'}]$

Operation: linearly mix the channels

$$A_i = \sum_{j=1}^{c} k_j^i * X_j + b^i$$

then pass results pixelwise through nonlinearity

$$O_i = \sigma(A_i)$$

Transformer Layer (by analogy)

Input: a sequence of vectors $X = [x_1, \dots, x_T]$

Output: a sequence of vectors $C = [c_1, \dots, c_T]$

Operation: mix the vectors nonlinearly using MHSA, $A = \text{MHSA}(X)$, then pass result column-wise through nonlinearity $C = FF(A)$

where $FF(a) = W_2 \sigma(W_1 a + b_1) + b_2$

Encoder Transformer Block "Attention is All You Need", 2017

Just as we have deep RNNs, Encoder Transformer Blocks consist of stacked layers of MHSA

Encoder

$[c_1 \cdots c_T]$ contextual representations

↑

Block N

↑ ⋮ ↑

Block 2

↑

Block 1

↑

$[x_1, \cdots, x_T]$ global embeddings

where each block has its own set of parameters and computes:

given input $X = [x_1, \cdots, x_T] \in \mathbb{R}^{d_x \times T}$

i) $Y = \text{LayerNorm}(X + \text{MHSA}(X))$

ii) $Z = \text{LayerNorm}(Y + \text{FF}(Y))$

where $\text{FF}(Y) = [\text{FF}(y_1) | \cdots | \text{FF}(y_T)]$

for $\text{FF}(y) = w^2 \sigma(w^1 y + b^1) + b^2$

and $\sigma = \text{ReLU}$

$X \rightarrow \square \rightarrow Z$

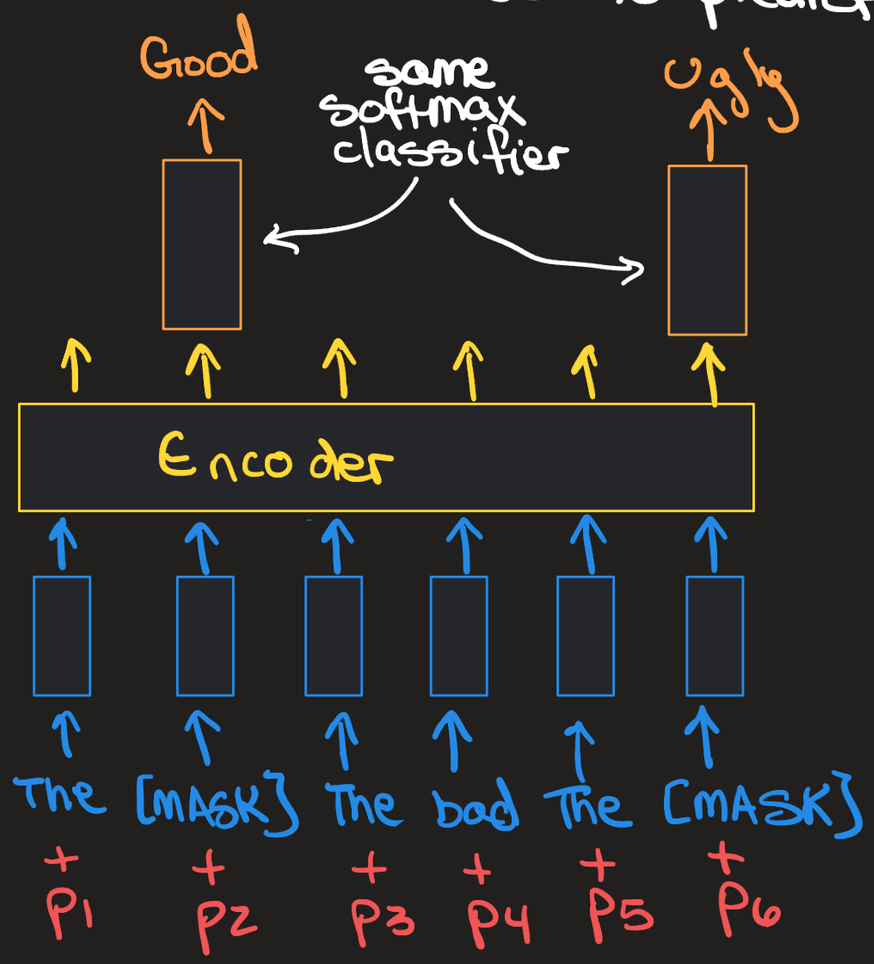
BERT

" BERT: Pretraining of Deep Bidirectional Transformers for Language Understanding ", 2018

- one of the first influential LLMs
- uses self-attention to learn contextual representations
- trains on large **unsupervised pretraining tasks**:
  - masked language modeling: predict missing words
  - next sentence prediction: predict whether sentence B really followed sentence A
- to use the model on supervised tasks, you **fine-tune** it by continuing to train for a small number of epochs at a lower learning rate on a supervised data set

Pretraining: Masked Language Modeling (MLM)

Given an input sequence {The, good, the, bad, the, ugly}, randomly select tokens to mask, with equal probability, and use the encoder to predict the masked tokens

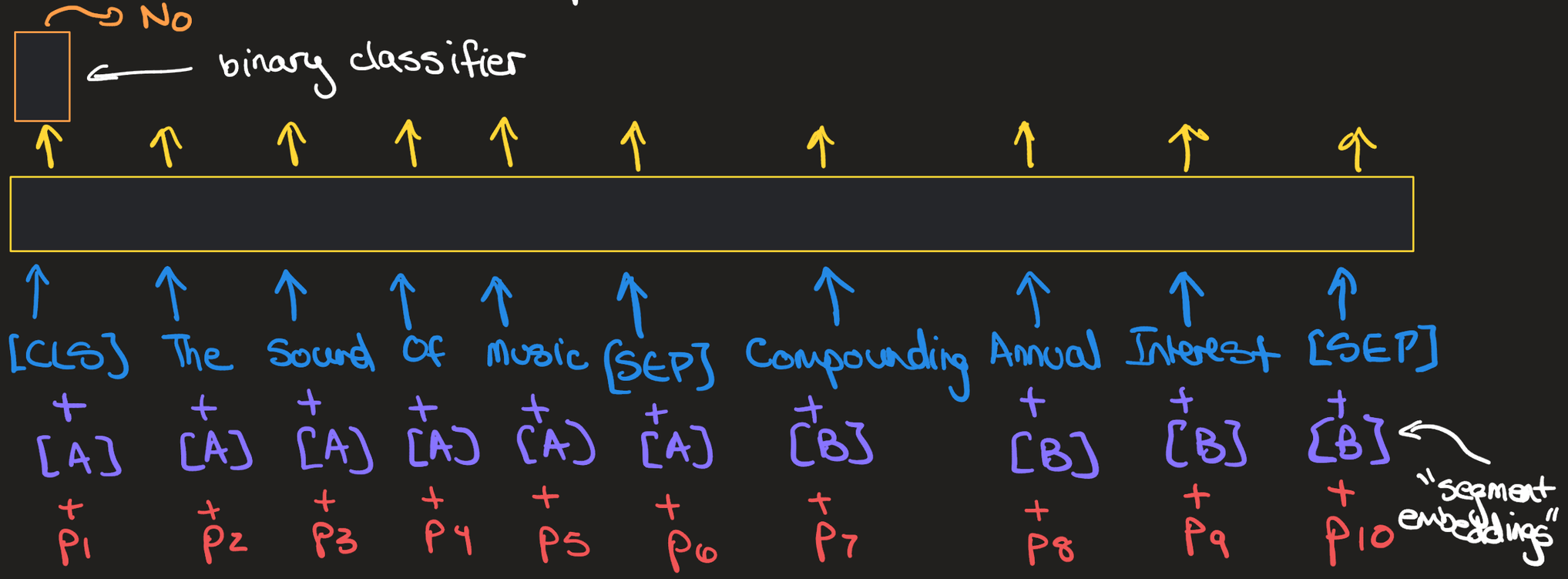


The classifier, encoder, and embedding parameters are learned to minimize error in predicting the masked tokens

The MLM objective teaches BERT the structure of language, the way words are used together.

Pretraining : Next Sentence Prediction (NSP)

Given two input sequences, {The, sound, of, music} and {Compounding, annual, interest}, BERT is trained to predict whether or not these sequences were consecutive.



This task also helps BERT learn how language is used

Finetuning:

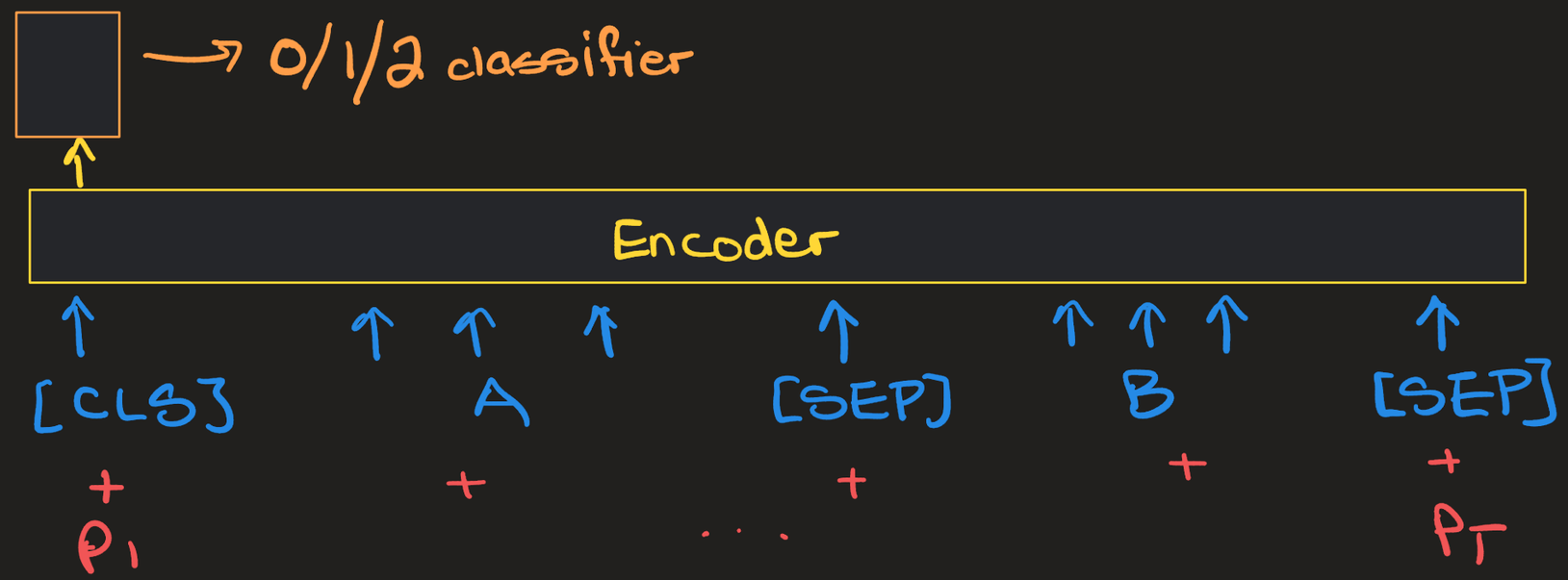
Pretraining on a large corpus helps BERT to learn about the structure of language. To use it on supervised tasks, one "fine-tunes" the model by:

- (potentially) modifying the architecture
- changing the loss appropriately
- continuing training the model, usually with a lower learning rate and fewer epochs

Fine-tuning balances between retaining "general knowledge" that is useful for the task, and specializing for the task. Pretraining + fine-tuning allows you to use large powerful architectures to perform well on tasks where you have comparatively smaller training sets, without overfitting.

Fine-tuning example : MNLI (Multi-genre Natural Language Inference)

- a standard benchmark task in the popular GLUE (General Language Understanding Evaluation) benchmark
- Given two sentences A, B, determine whether B is an entailment, contradiction, or neutral with respect to A
- Encode inputs as for the NSP task, but use a new classification head for the MNLI task



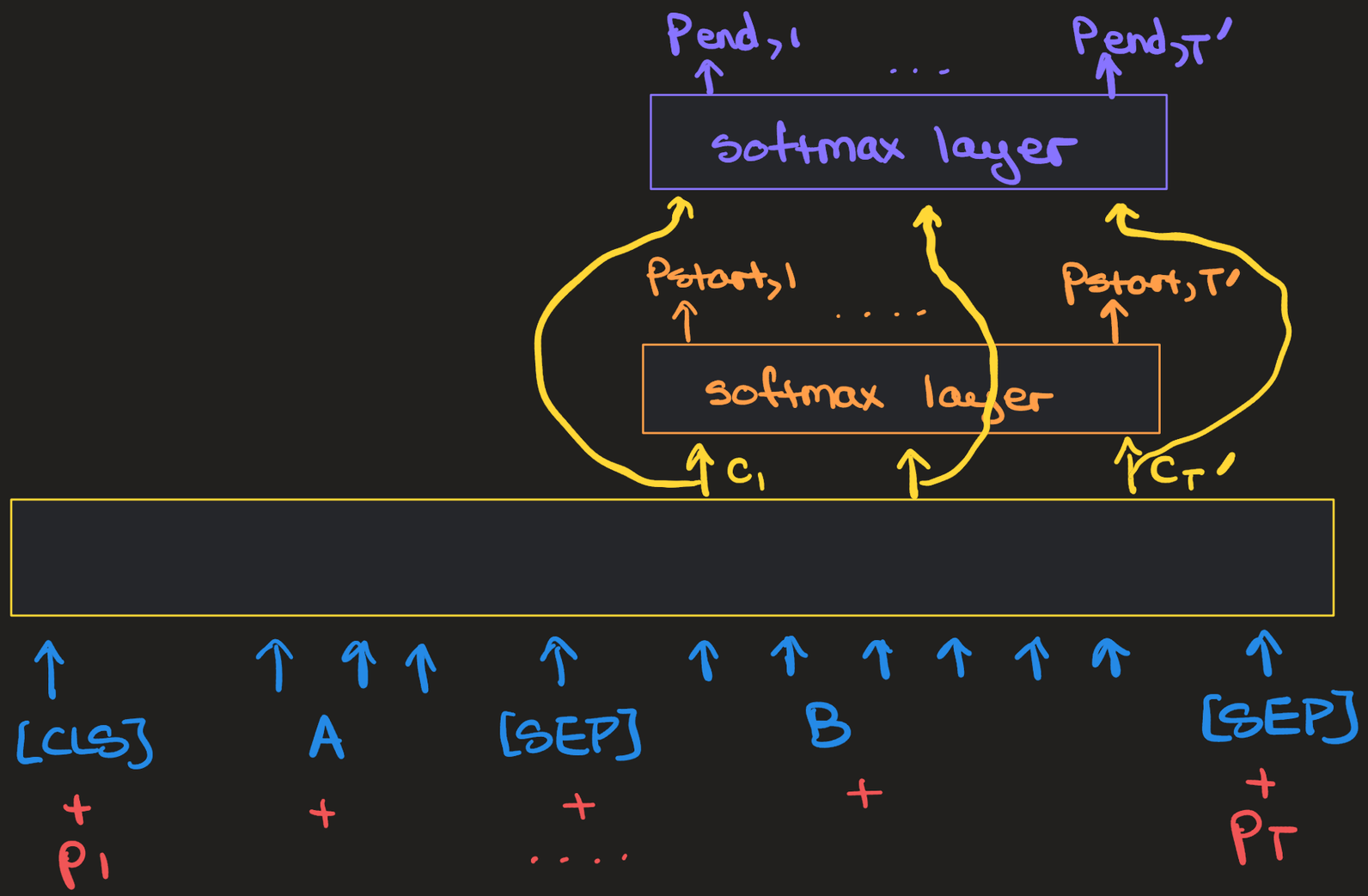
Fine-tuning example: SQUAD (Stanford question answering Dataset)

A B

- Given a question and a passage from Wikipedia with the answer, identify the answer text span in the passage.
- Introduce S and E embeddings that we will use to identify the start and end of a candidate answer
- Training: let $P_{start} = \text{softmax}\left(\begin{bmatrix} C_1^T \\ \vdots \\ C_T^T \end{bmatrix} S\right)$ and $P_{end} = \text{softmax}\left(\begin{bmatrix} C_1^T \\ \vdots \\ C_T^T \end{bmatrix} E\right)$ ← only apply over the tokens in B

be the probabilities that the answer starts (ends) at each token. Maximize the sum of the log probabilities of the true start and true end of the answer.

- Inference: score each span (where $j > i$) $[C_i, \cdots, C_j]$ with $\langle S, C_i \rangle + \langle E, C_j \rangle$ and return the span with the highest score


Pend,1
Pend,T'
softmax layer
Pstart,1
Pstart,T'
softmax layer
C1
CT'
[CLS]
A
[SEP]
B
[SEP]
+
P1
PT