Transformers and BERT

Disclaimer: Work in progress. Portions of these written materials are incomplete.

Outline

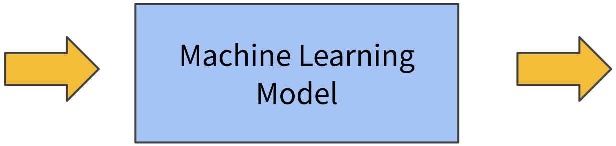
History of Language Representations	15 minutes
Self-Attention & Transformers	15 minutes
BERT	15 minutes

Natural Language Processing (NLP)

- NLP enables computers to process natural language
  - e.g. sentiment analysis, question answering, summarization, etc.
- Example: question answering

Q: The traveling salesman problem is an example of what type of problem?

P: A function problem is a computational problem ... Notable examples include the traveling salesman problem and the integer factorization problem.

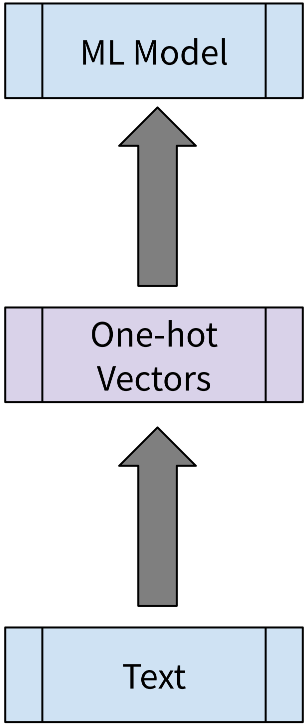


A: A function problem is a computational problem where a single output …. Notable examples include the traveling salesman problem and the integer factorization problem.

How should we encode text in ML models?

A reasonable start is a discrete representation via one-hot vectors

Token	Index	One-hot vector
aardvark	0	[1, 0, 0, ...]
...		
king	123	[0, ..., 0, 1, 0, ...]
queen	124	[0, ..., 0, 0, 1, ...]



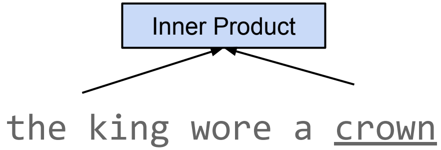
Distances between any two words...

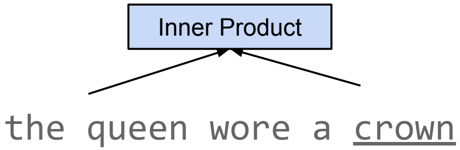
- are always the same!
- however, "queen" should be closer to "king" than "aardvark"

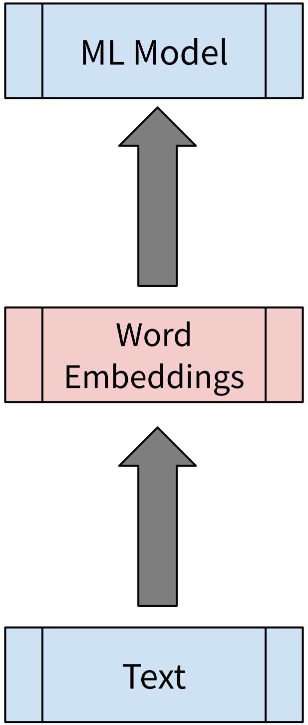
Continuous Representations of Words

Word embeddings = continuous representations **pre-trained** on an **unlabeled** corpus on **co-occurrence statistics**

Token	Index	Word Embedding
aardvark	0	[0.1, 1.9, -0.4, …]
...		
king	123	[-0.5, -0.9, 1.4, …]
queen	124	[-0.6, -0.8, -0.2, …]







Word2Vec [mikolov et-al, 13], PLSI [Hoffman, 99], GloVe [Pennington et-al, 14]

Contextual Representation of Words

- **Problem**: word embeddings are context-independent

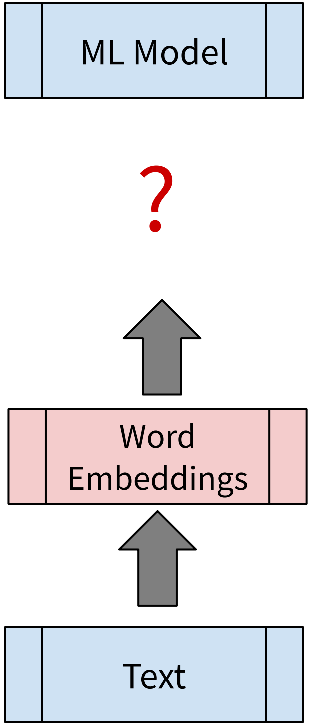
`open a bank account` `on the river bank`

`[0.3, 0.2, -0.8, …]`

- Ideally, representations should be contextual

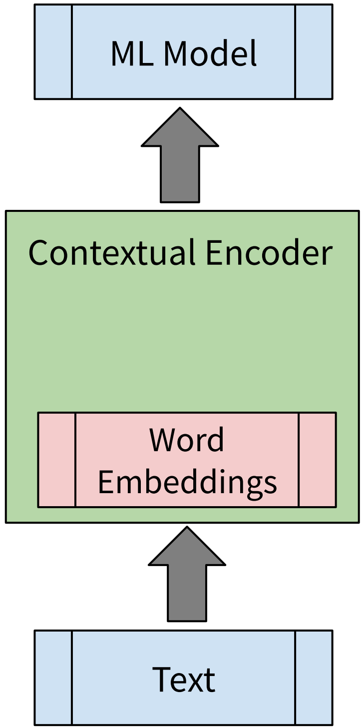
`open a bank account` → `[0.9, -0.2, 1.6, …]`

`on the river bank` → `[-1.9, -0.4, 0.1, …]`

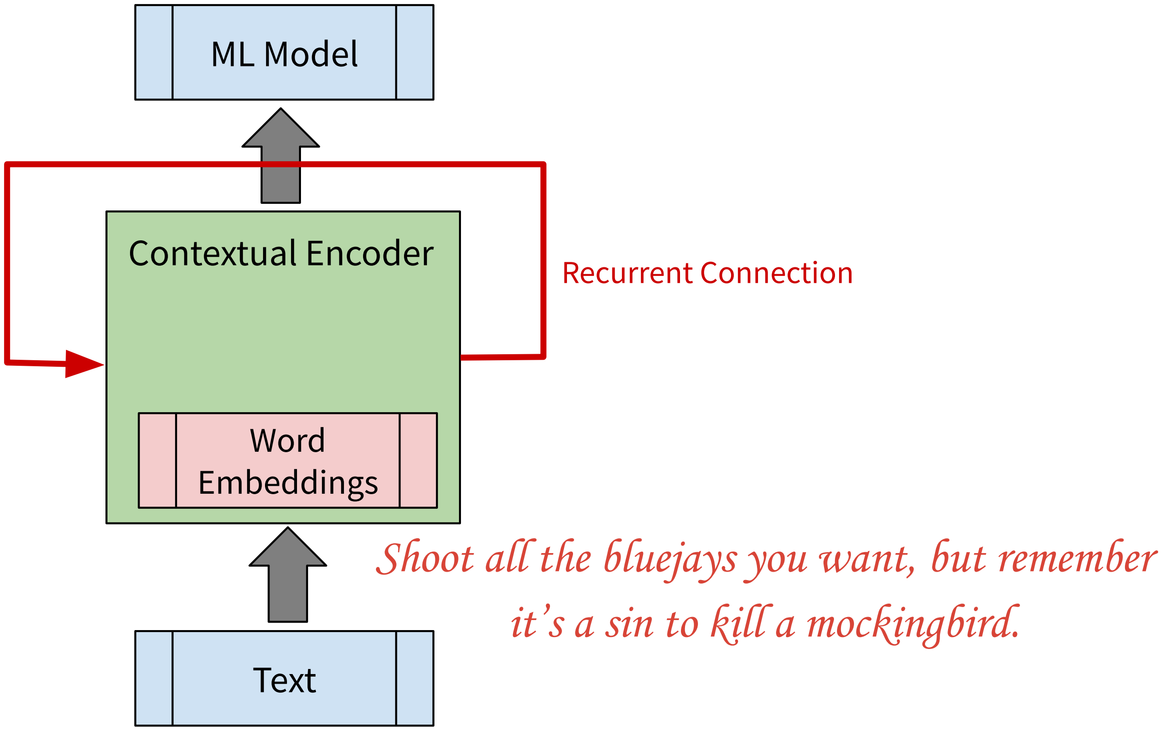


Contextual Encoders for Natural Language

- **Contextual encoders** go beyond a simple dictionary lookup of word embeddings
- They are **pre-trained** on an **unlabeled** corpus of general-domain text, usually with a **language model objective**

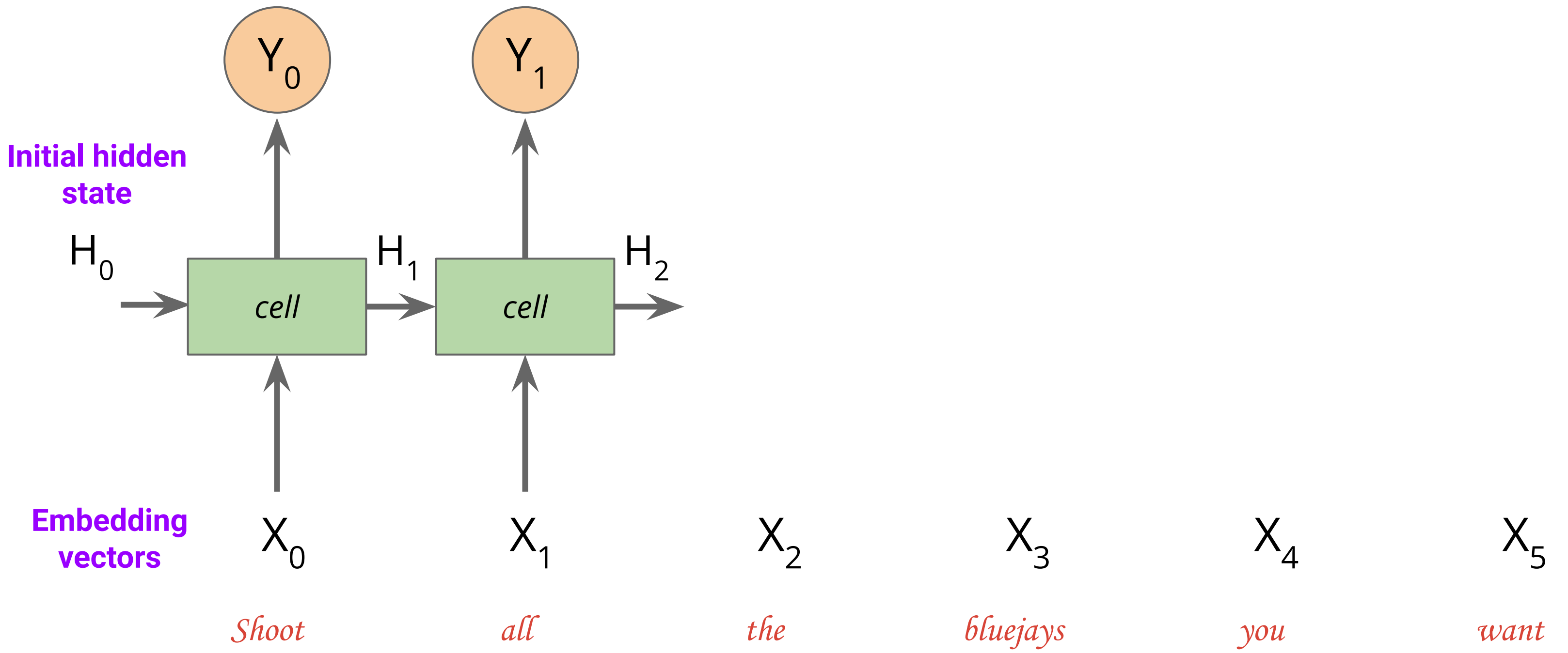


The Recurrent Neural Network (RNN) Encoder

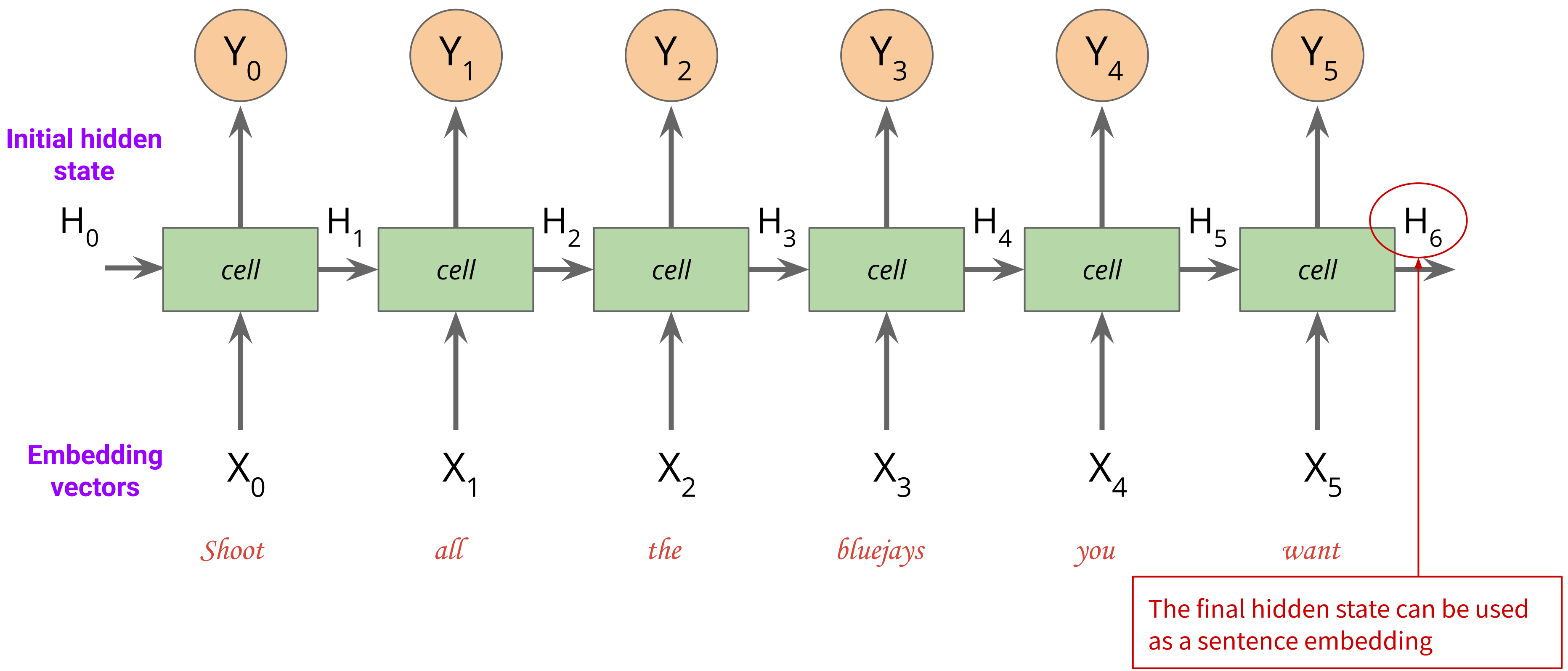


The RNN Encoder: Unfolding in Time

The RNN Encoder: Unfolding in Time

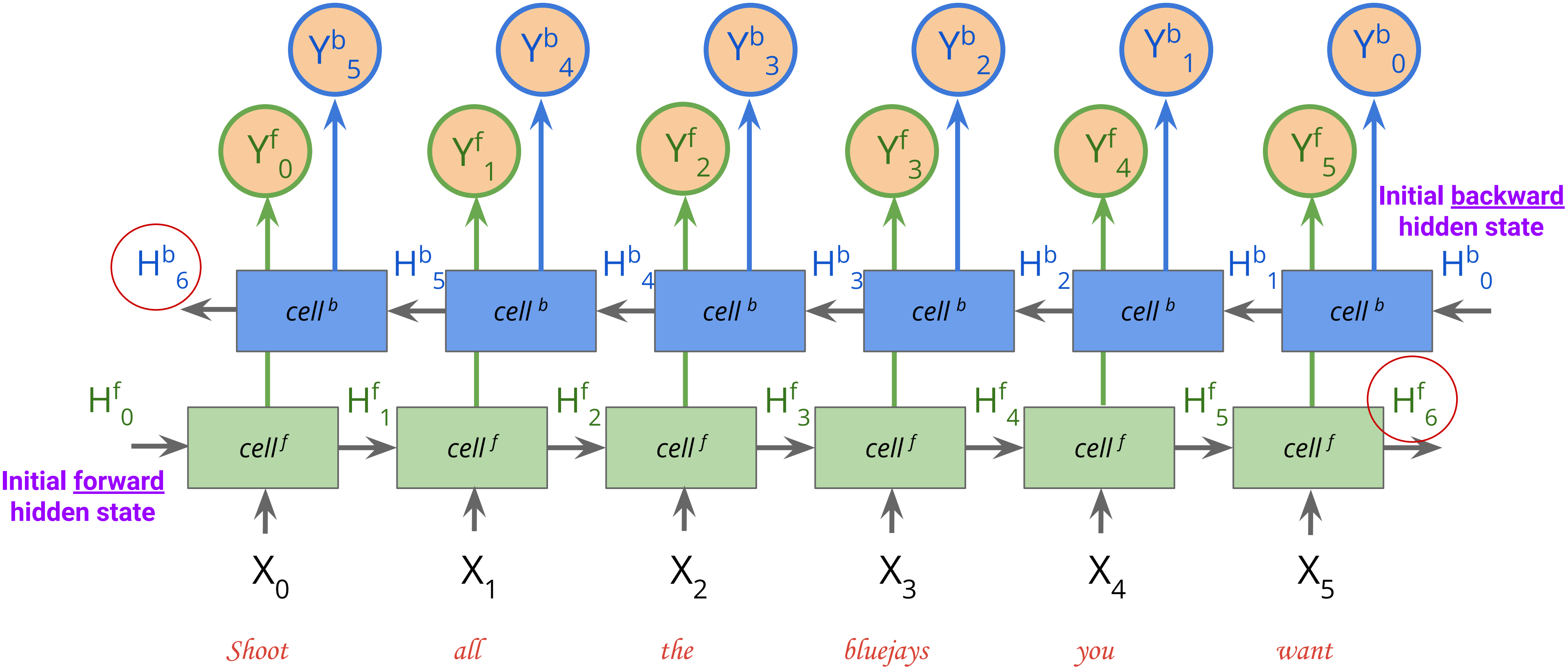


The RNN Encoder: Unfolding in Time



The Bidirectional RNN Encoder

The sentence embedding is the concatenation of the two final hidden states: $[H^f_6 ; H^b_6]$



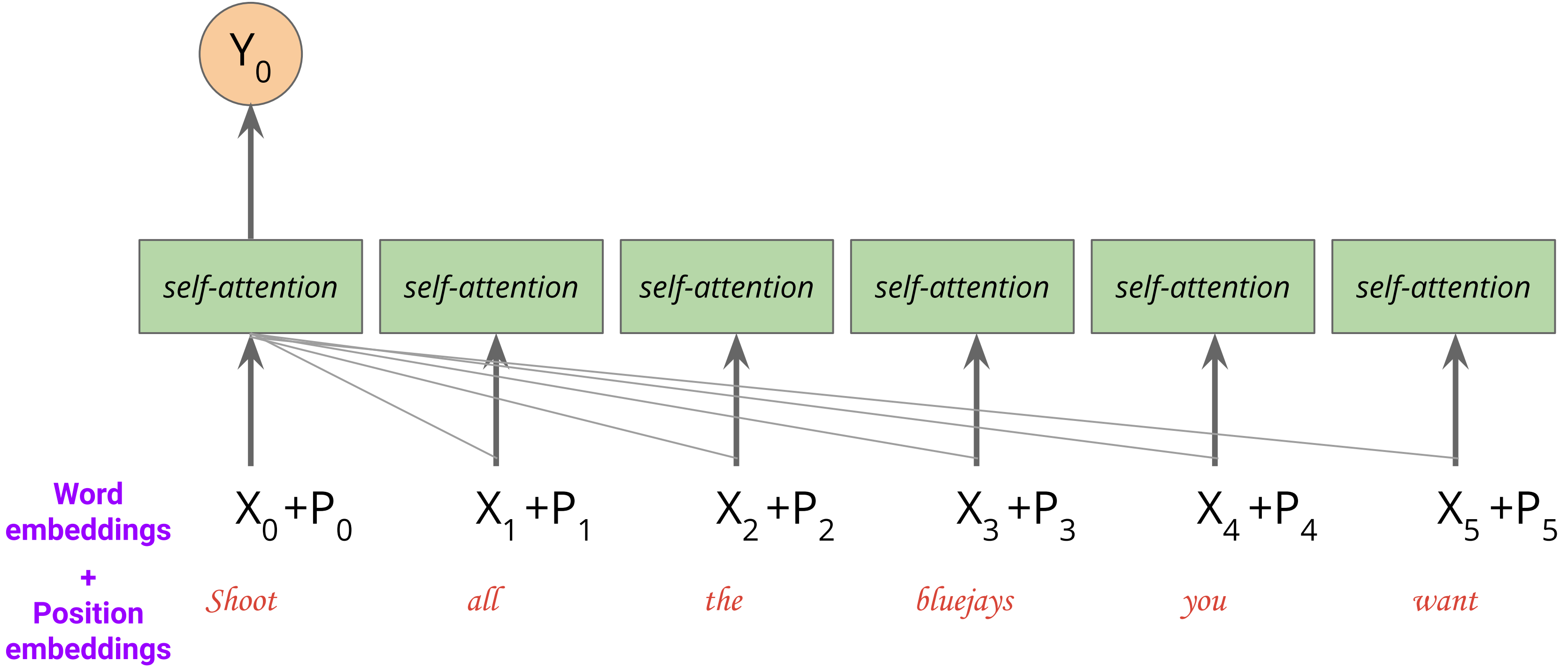
Disadvantages of RNN Encoders

1. **Slow**: O(N) in the number of tokens N
2. **Vanishing Gradient** => cannot process very long sequences
3. **Pseudo-Bidirectional**
4. etc.

Outline

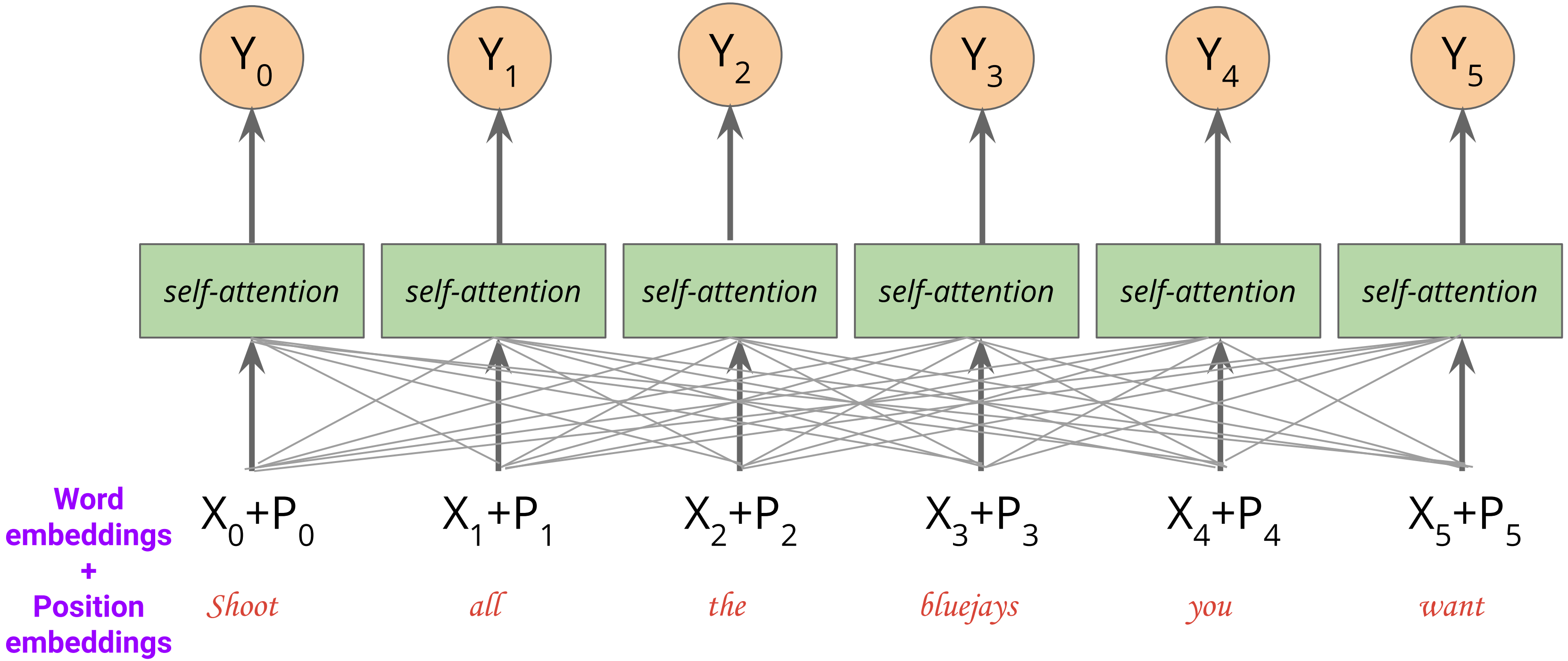
History of Language Representations	15 minutes
Self-Attention & Transformers	**15 minutes**
BERT	15 minutes
Colab Notebook	30 minutes

Self-Attention



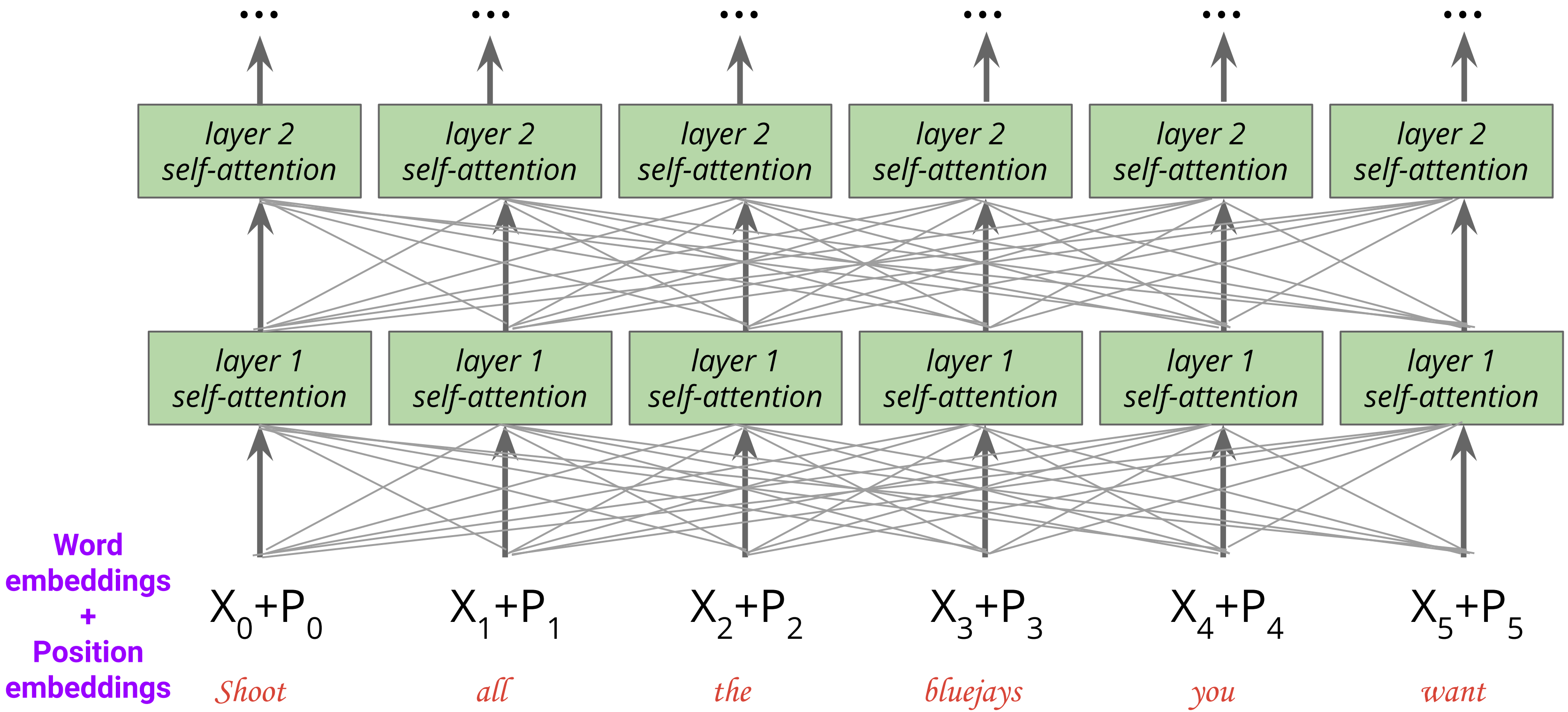
Self-Attention

This is quite expensive: $O(N^2)$ connections
But we can compute all Y's in parallel



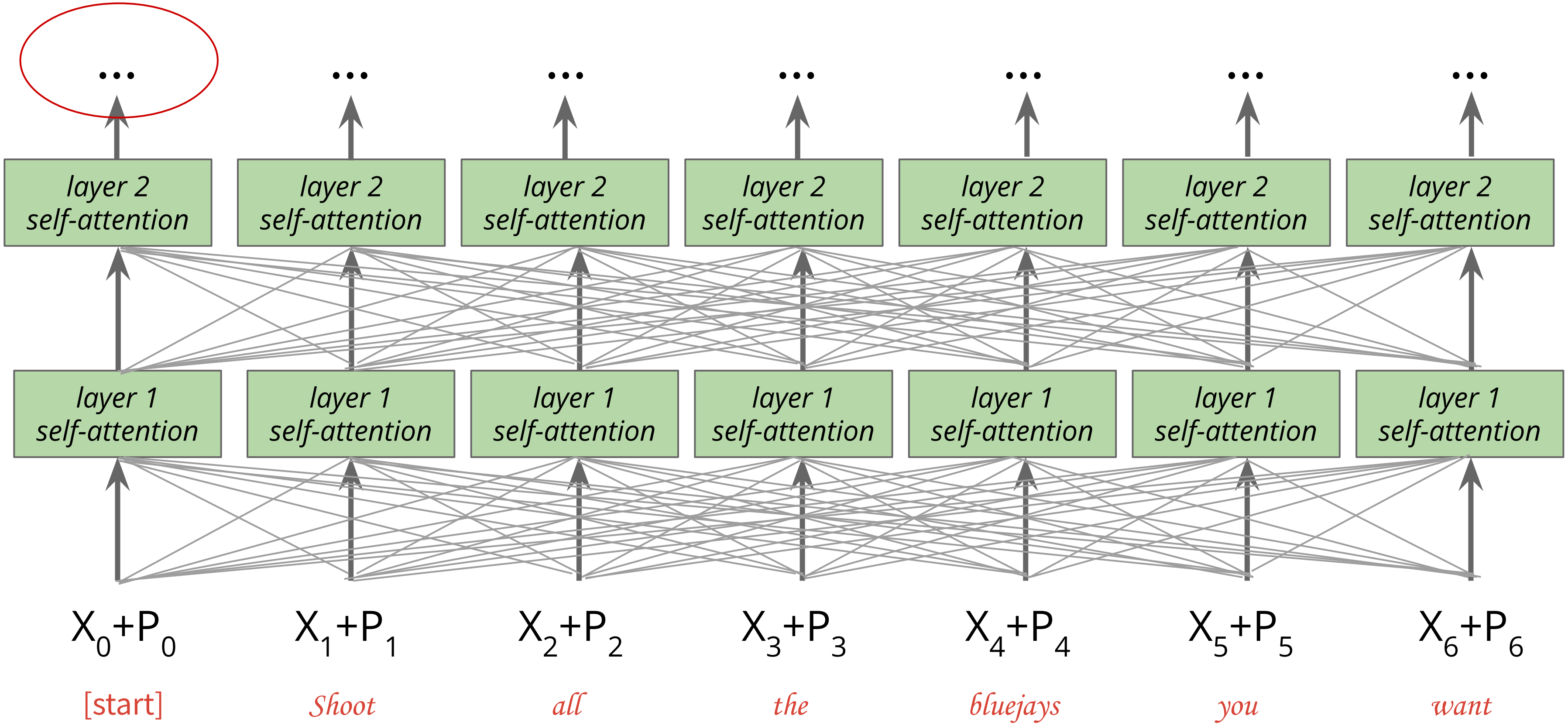
The Transformer Encoder

The transformer encoder is a stack of self-attention layers.



The Transformer Encoder

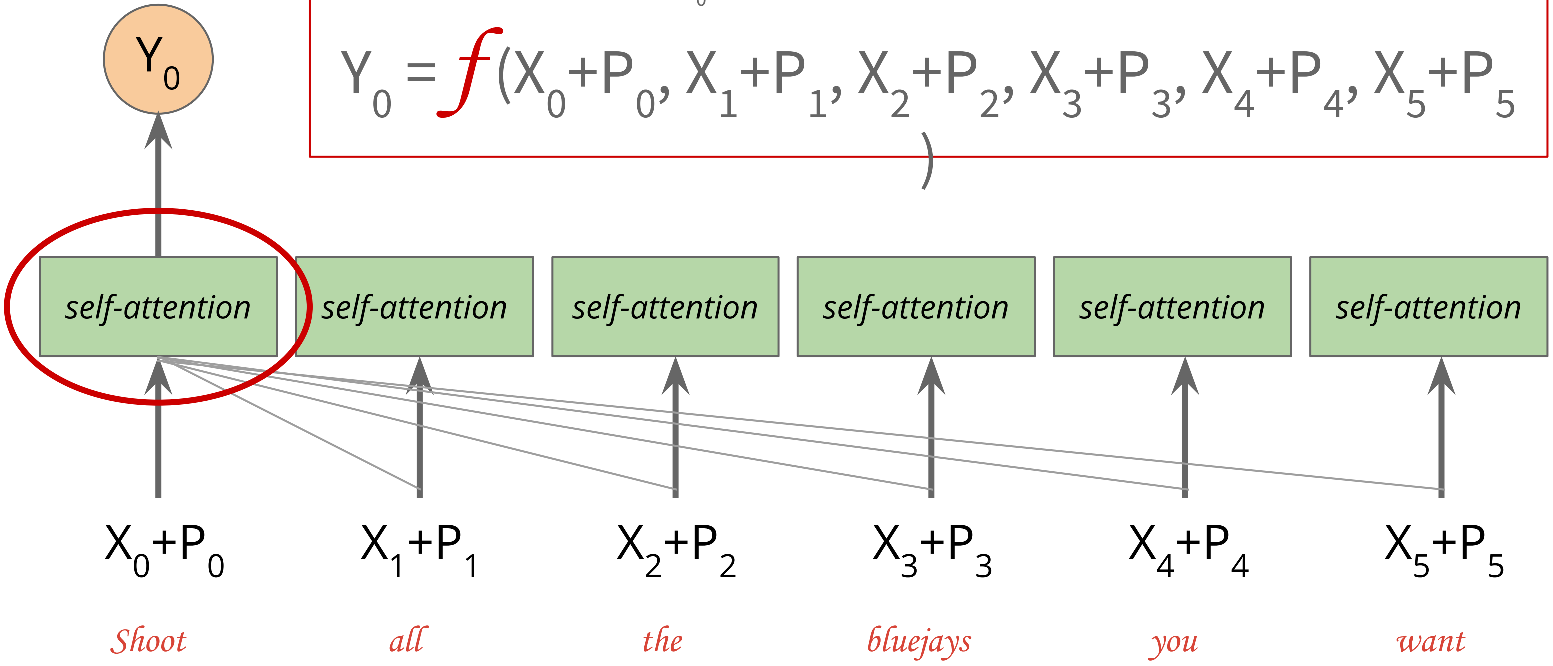
The sentence embedding is the embedding of the [start] token.



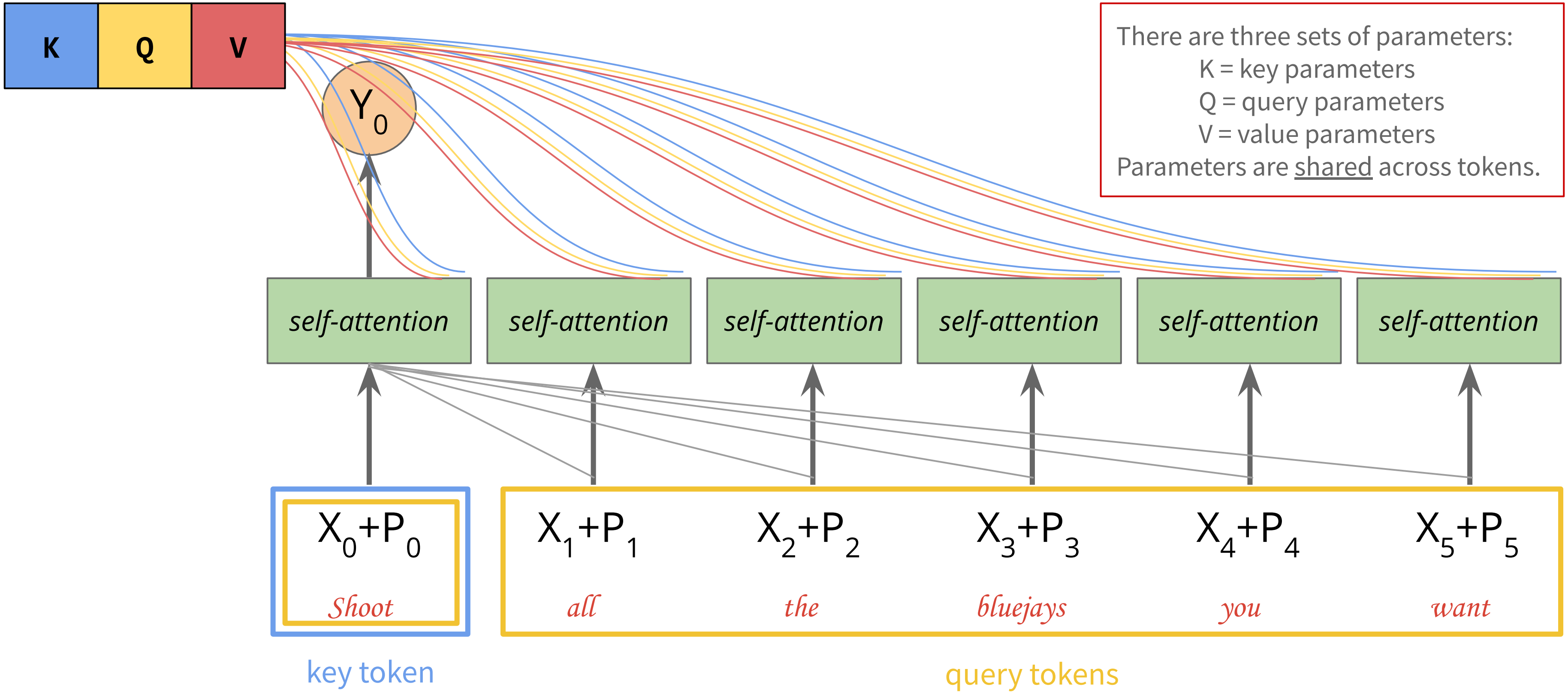
Advanced: Implementation of Self-Attention

Y_0 is a function of the entire sentence:

$$Y_0 = f(X_0+P_0, X_1+P_1, X_2+P_2, X_3+P_3, X_4+P_4, X_5+P_5)$$

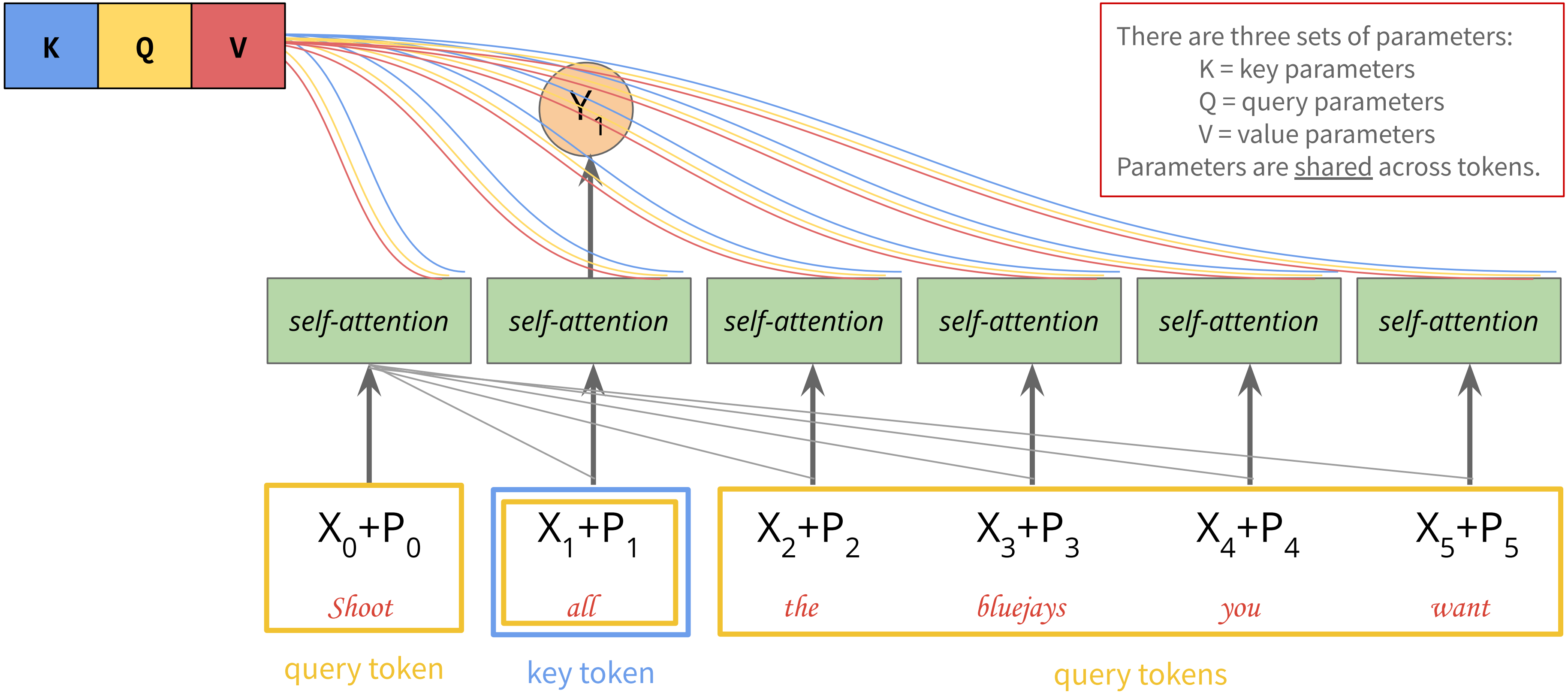


Advanced: Implementation of Self-Attention



There are three sets of parameters:
K = key parameters
Q = query parameters
V = value parameters
Parameters are <u>shared</u> across tokens.

Advanced: Implementation of Self-Attention



There are three sets of parameters:

K = key parameters

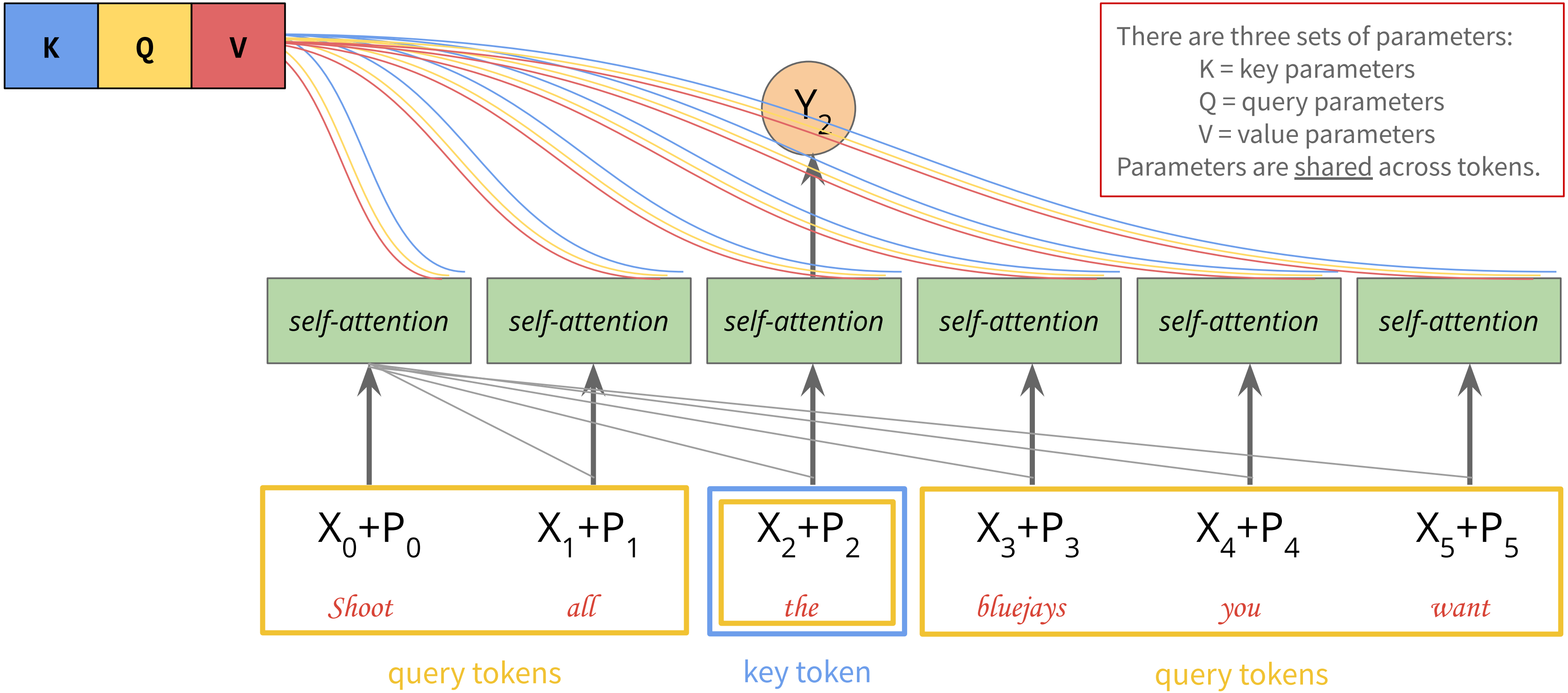
Q = query parameters

V = value parameters

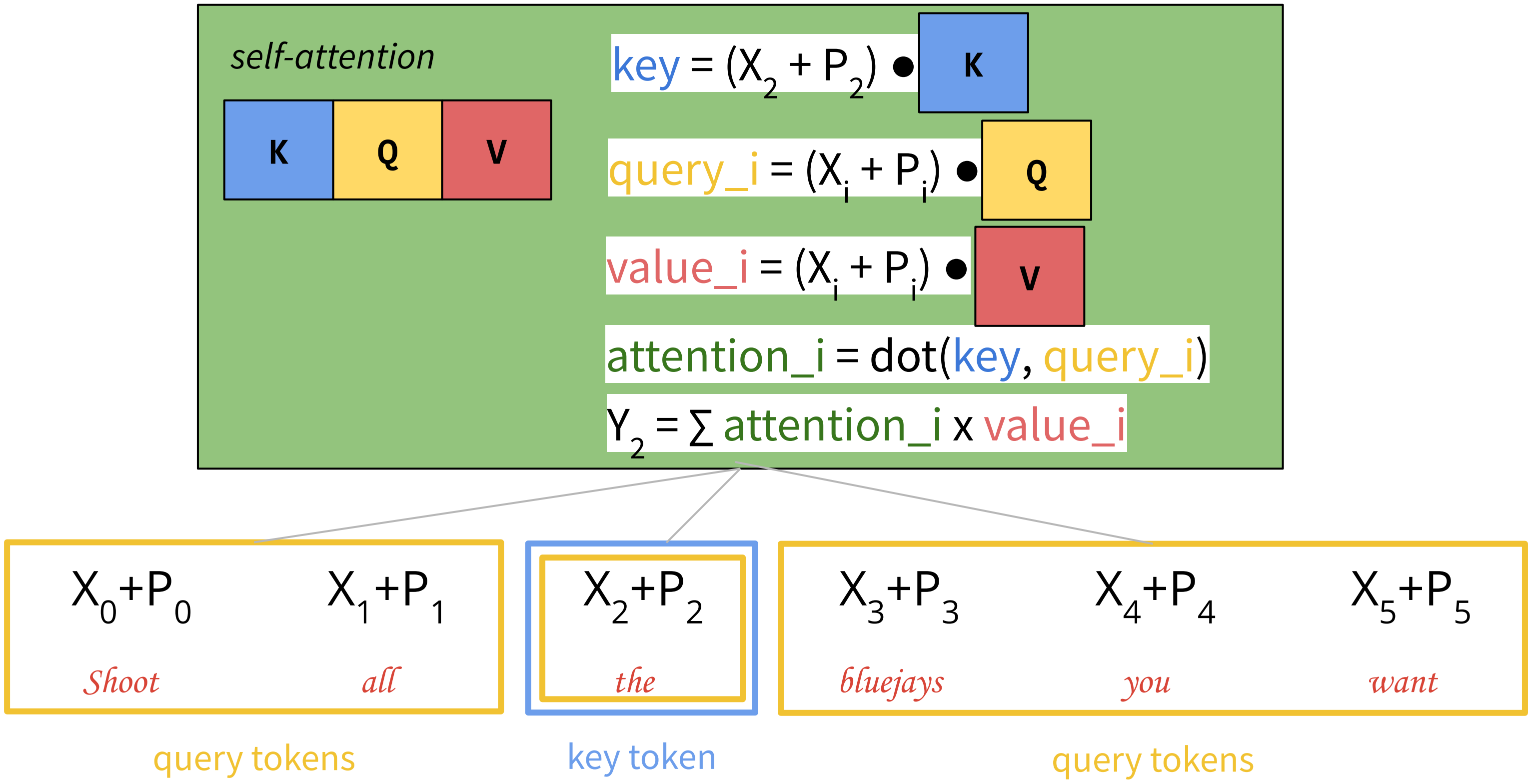
Parameters are <u>shared</u> across tokens.

Advanced: Implementation of Self-Attention

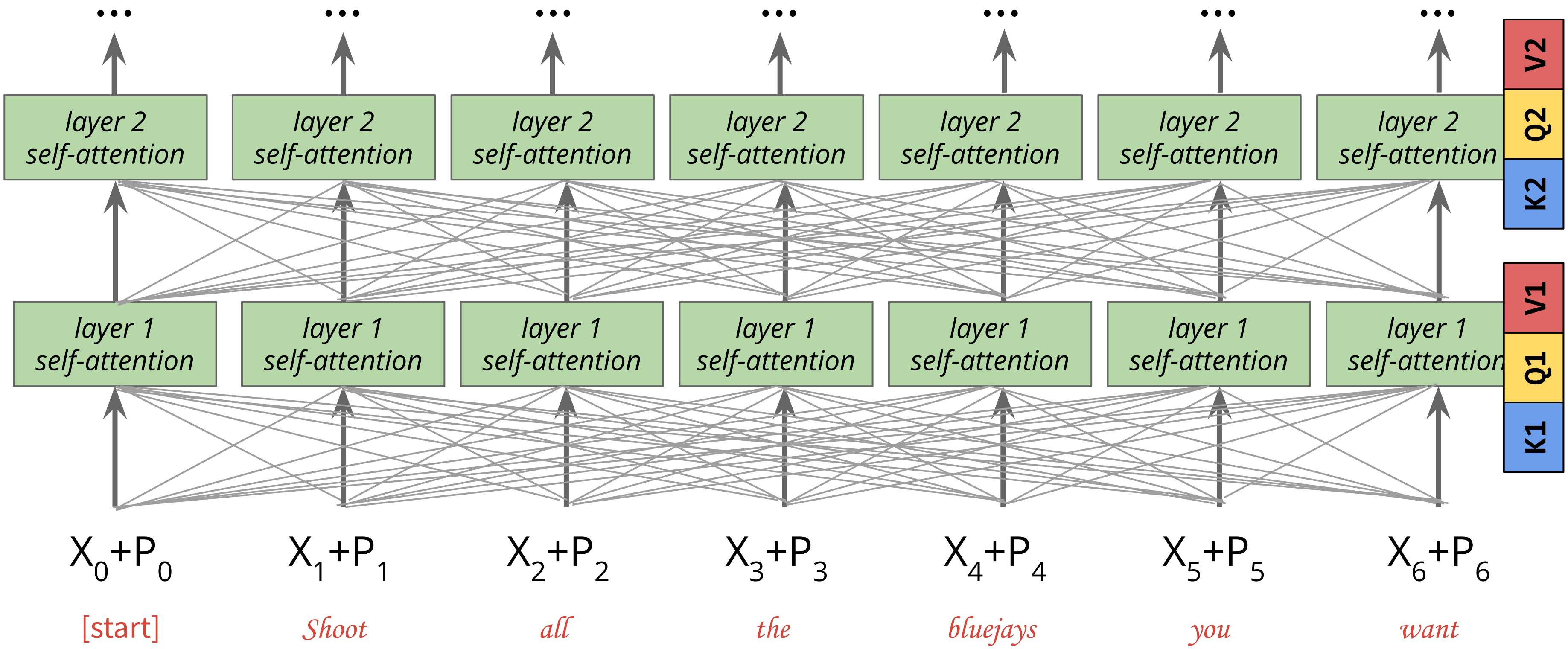
There are three sets of parameters:
K = key parameters
Q = query parameters
V = value parameters
Parameters are shared across tokens.



Computing Keys and Queries



The Transformer Encoder



The Transformer Encoder

Disclaimer: for simplicity, the presentation of the transformer encoder in these slides omits certain details:

- Self-attention is **multi-headed**
- Multi-headed self-attention is followed by a **feed-forward network**
- Residual/skip connections
- Layer normalization
- Dropout

Disadvantages of Transformer Encoders

1. **Computationally intense:** $O(N^2L)$
2. Input must have a **fixed number of tokens**
   - Because the number of position embeddings needs to be finite
   - All inputs are truncated or padded to e.g. 512 tokens

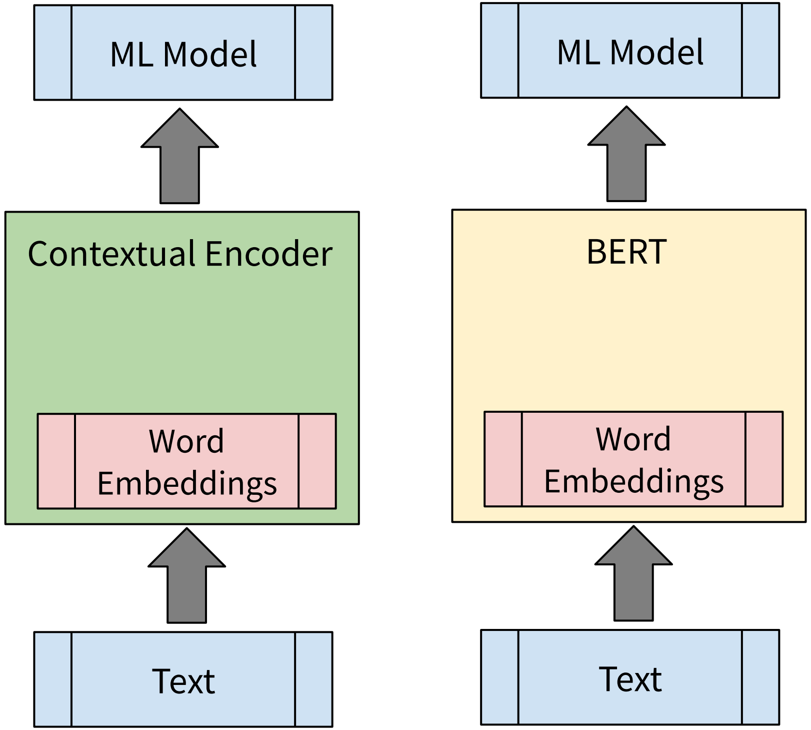
Outline

History of Language Representations — 15 minutes

Self-Attention & Transformers — 15 minutes

BERT — **15 minutes**

BERT: Bidirectional Encoder Representations from Transformers



BERT is a contextual encoder:

1. built with **Transformer** layers
2. operating on **WordPieces**
3. trained with two **training objectives**:
   a. Next Sentence Prediction
   b. Masked Language Model

Whole-Word Tokens vs WordPieces

Oh, supercalifragilisticexpialidocious
Is something quite atrocious

Whole-Word Tokens

Oh|,|supercalifragilisticexpialidocious
|Is|something|quite|atrocious

WordPieces

Oh|,|super|##cal|##if|##rag|##ilis|##tic|##e
x|##pia|##lid|##oc|##ious|is|something|qui
te|at|##ro|##cious

Advantages of WordPieces:

- fewer OOVs
- might enable reuse across languages in multilingual models

How do we feed *two* sentences to the model?

- Use two special tokens [CLS] (i.e. [start]) and [SEP] (i.e. separator)

[CLS]][Example]

(WordPiece tokenization)

↓

[CLS] Example]

But now we need to encode the *segments* (i.e. tell the model what token belongs to which sentence)!

Input Representation Details

Each token representation is the sum of three embeddings:

Masked Language Model Training Objective

Training Objective: Mask 15% of the input tokens and train the model to predict them.

[Example]

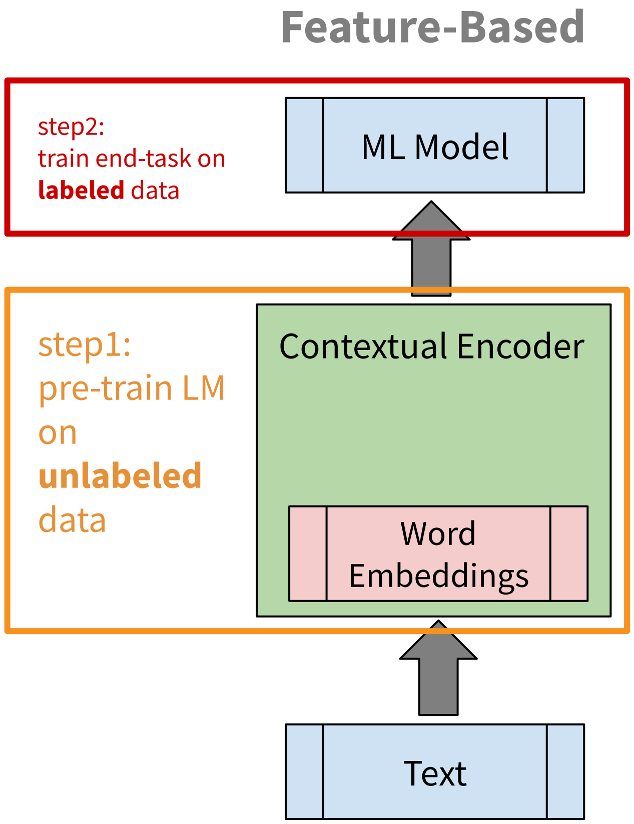
↓ (WordPiece tokenization)

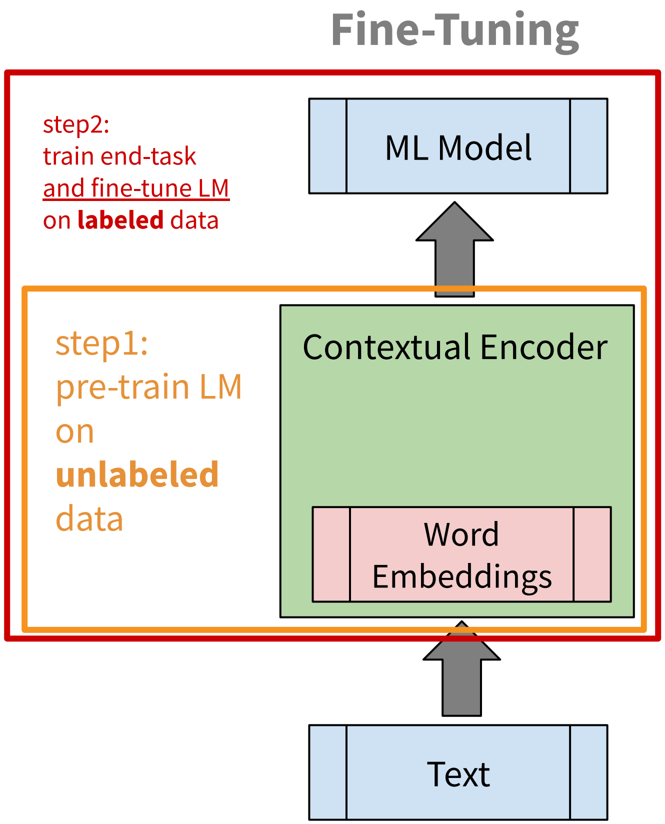
[Example]

↓ (WordPiece masking)

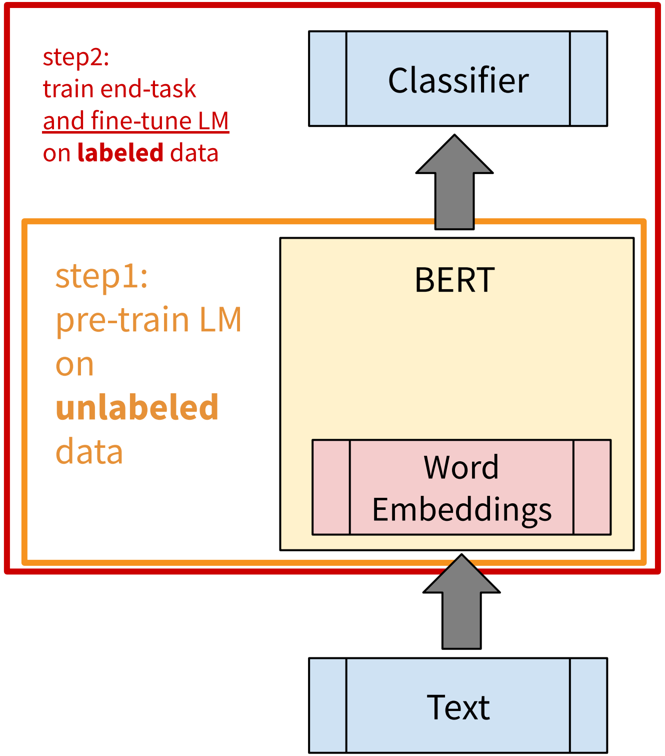
[Example]

Using Contextual Representations





Fine-Tuning BERT for Classification



1. Download **labeled** data & tokenize it.
2. Process it into BERT-compatible inputs:
   a. token_ids
   b. segment_ids
   c. input_mask
3. Download pre-trained BERT (graph + weights)
4. Define your ML Model on top
   a. input = embedding of [CLS] from BERT
5. Train on the labeled data from 2.

Where can I find BERT?

- The original paper open-sourced two BERT sizes, available on Github and TF-Hub:
  - BERT-Base (12 layers, embedding size 768)
  - BERT-Large (24 layers, embedding size 1024)
- BERT comes in multiple flavors:
  - uncased vs cased
  - English vs multilingual
  - (more recently: wordpiece-masking vs whole-word masking)

Smaller BERT Models

https://arxiv.org/pdf/1908.08962.pdf

In additional to the two original bert sizes (BERT-Base and BERT-Large), there are now 24 more sizes