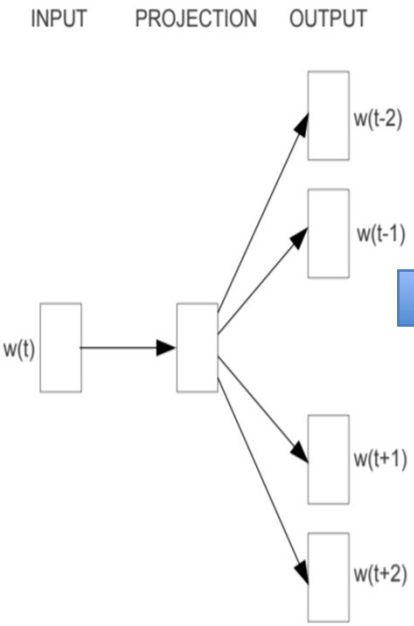
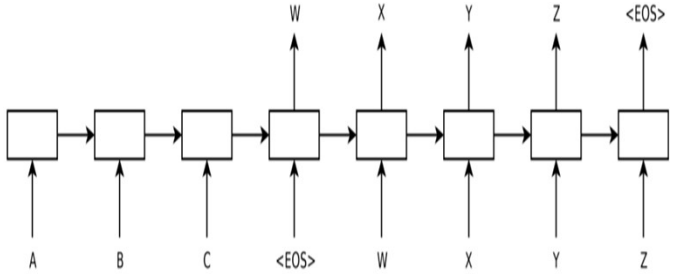
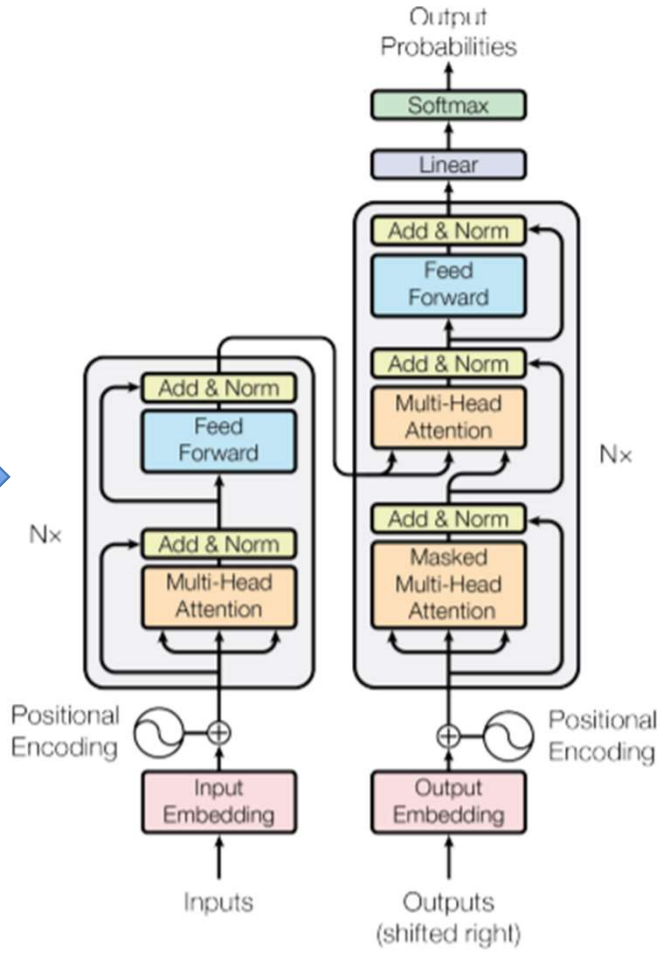
# Natural Language Processing

## From Skip-Gram to Seq2Seq to Transformers (and Beyond)




**Skip-Gram (2013)**

**Sequence-to-Sequence (2014)**

**Transformer (2017)**

# Outline

- Introduction
  - A Brief Overview of Natural Language Processing
- Representational Learning with Word Embeddings
  - Word Embedding Concepts
  - Context-free Embeddings
  - Conditional Embeddings
  - Other Seminal Works
- Language Modeling and Machine Translation
  - Language Modeling
  - Machine Translation
- **Recurrent Neural Networks for Sequence Learning**
  - Sequence to Sequence (Seq2Seq) Learning with Neural Networks
  - Limitations of Seq2Seq Models

# Outline

- Attention
  - **Introduction**
  - **“Neural Machine Translation by Jointly Learning to Align and Translate”**
  - Types of Scoring Functions
  - Types of Attention Mechanisms
- Transformers
  - **“Attention is All You Need”**
  - “Character-Level Language Modeling with Deeper Self-Attention”
  - **“BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”**
  - ***“Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context”***
  - ***“XLNET: Generalized Autoregressive Pretraining for Language Understanding”***
- What’s Next in NLP?
  - (Textual) Q&A Systems
  - Multi-task Learning
  - Unsupervised Neural Machine Translation
  - Natural Language Understanding
- References

# Introduction

## A Brief Overview of Natural Language Processing

# Introduction

## A Brief Overview of Natural Language Processing

- Natural Language Processing (NLP) is a rapidly growing field and is growing quickly in industry in two big areas

# Introduction

## A Brief Overview of Natural Language Processing

- Natural Language Processing (NLP) is a rapidly growing field and is growing quickly in industry in two big areas
- Dialogue
  - Chatbots
  - Customer Service

# Introduction

## A Brief Overview of Natural Language Processing

- Natural Language Processing (NLP) is a rapidly growing field and is growing quickly in industry in two big areas
- Dialogue
  - Chatbots
  - Customer Service


- Healthcare
  - Understanding health records
  - Understanding biomedical literature

# Introduction

## A Brief Overview of Natural Language Processing

- Natural Language Processing (NLP) is a rapidly growing field and is growing quickly in industry in two big areas
- Dialogue
  - Chatbots
  - Customer Service


- Healthcare
  - Understanding health records
  - Understanding biomedical literature


- Rapid progress over the last 5 years due to deep learning and larger models
- NLP is reaching the point of having a huge social impact, making issues like bias and security increasingly important

# Introduction

## A Brief Overview of Natural Language Processing

- What is exactly is NLP?
- Human language is a system designed to convey **meaning**
  - Not produced by a physical manifestation of any kind → very different from vision and other machine learning tasks
- **Words** (signifier) **are symbols** for an extra-linguistic entity (signified)
  - "Rocket" (word) refers to the *concept* of a rocket (signified)
- Symbols of language can be encoded in several modalities (voice, gesture, writing, etc.)
- **Goal of NLP** → Design algorithms to allow computers to "understand" natural/human language in order to perform a task
  - Easy tasks: Spell checking, keyword search, finding synonyms
  - Medium tasks: Parsing information from websites, documents, etc.
  - Hard tasks: Machine Translation, Semantic Analysis, Coreference, Q&A

# Introduction

## A Brief Overview of Natural Language Processing

- Major milestones
  - 1950s → Advent of NLP with the task of machine translation (MT)
    - Early efforts aided in code-breaking during World War II (Russian-to-English)
  - 1960s → ELIZA (chatbot) and SHRDLU (language program with user interaction) developed
  - 1960s – 1980s → NLP primarily driven by complex sets of hand-written rules and parameters
  - Late 1980s → Introduction of statistical NLP and ML-driven algorithms for language processing
  - 1990s → Idea of using similarities between words to generalize from training to test sequences
    - Latent Semantic Indexing (Deerwester et al., 1990)
    - First use of neural networks for language modeling (Miikkulainen and Dyer, 1991)
    - *N*-gram language models (Brown et al., 1992)
- **Each milestone is about a decade or more**!

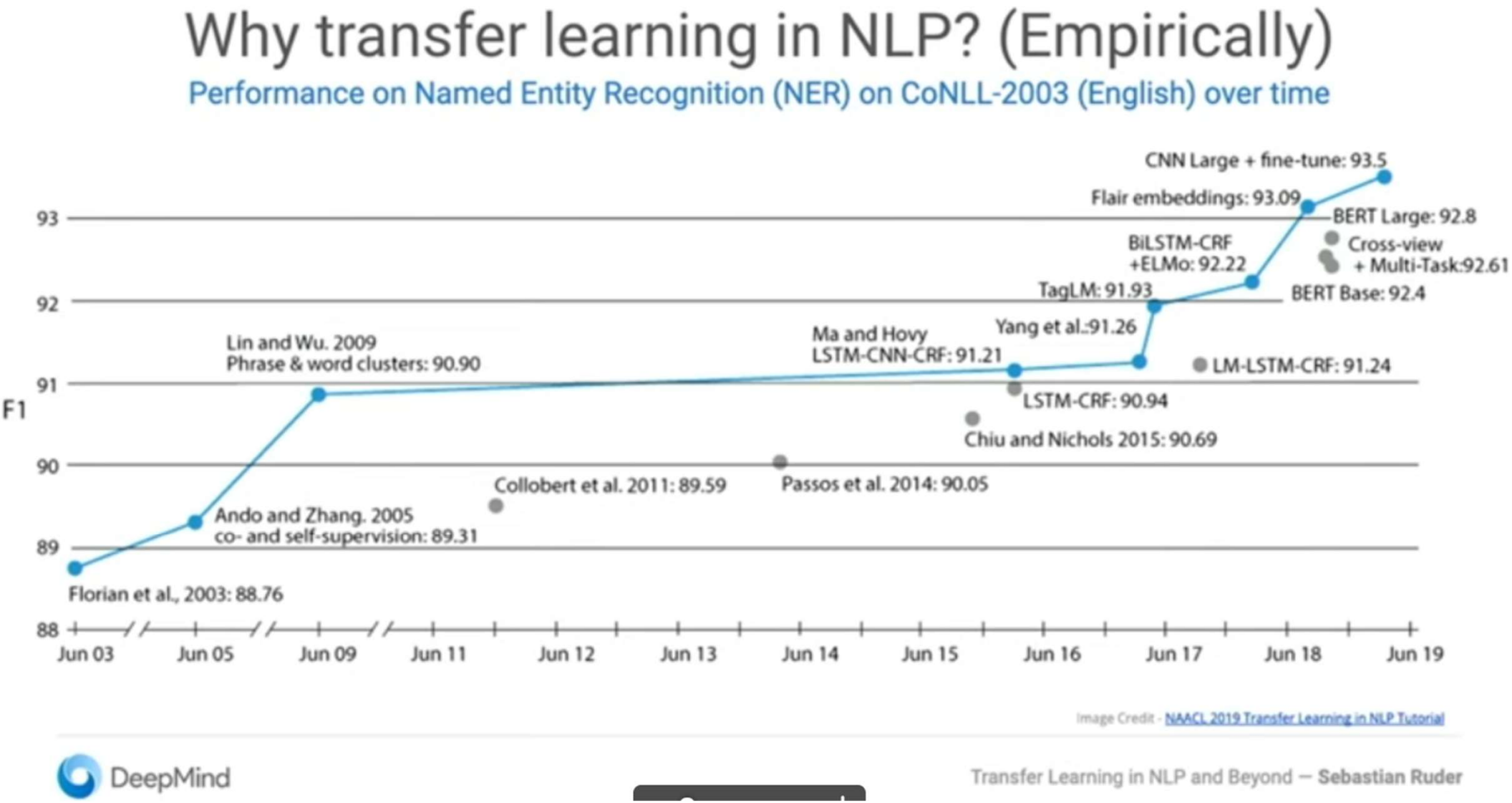
# Introduction

## A Brief Overview of Natural Language Processing

- Major milestones
  - 2001 → Conditional Random Fields for sequence labeling
  - 2003 → **Statistical** neural language modeling and Latent Dirichlet Allocation
  - 2008 → Multi-task learning
  - 2013 → **Word embeddings** for representation learning
  - 2014 → Seq2Seq and Convolutional Neural Network models for MT
  - 2015 → **Use of Attention in NLP** and Memory-based networks
  - 2017 → **Transformer networks**
  - 2018 → Pre-trained Language Models and **Transfer Learning**
- ***All*** milestones encompass about one decade!

# Introduction

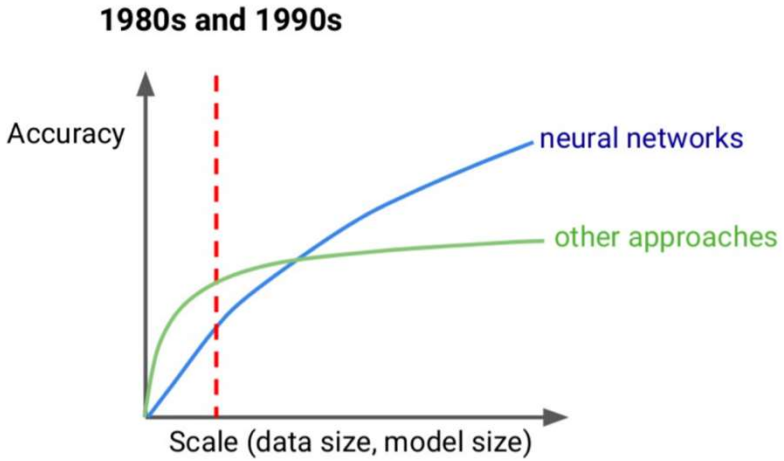
## A Brief Overview of Natural Language Processing



# Introduction

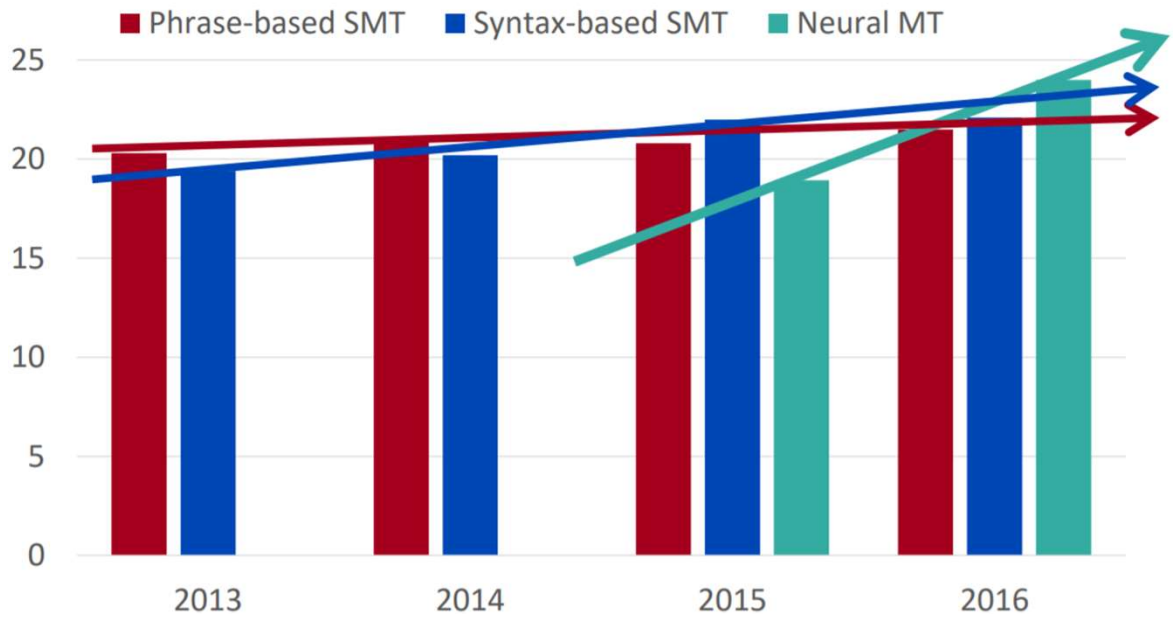
## A Brief Overview of Natural Language Processing

**Why has deep learning been so successful recently?**




**MT progress over time**

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]



# Introduction

## A Brief Overview of Natural Language Processing

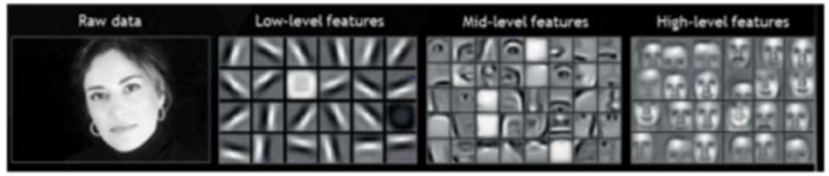
- Machine Learning brought huge successes

# Introduction

## A Brief Overview of Natural Language Processing

- Machine Learning brought huge successes

- Image Recognition
  - Widely used by Google, Facebook, etc.

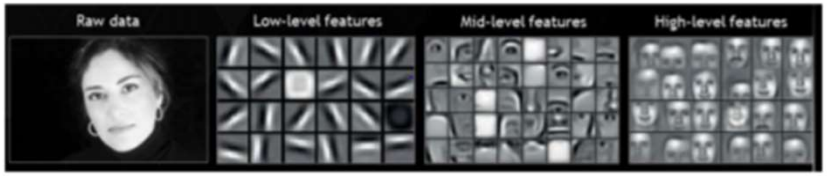


# Introduction

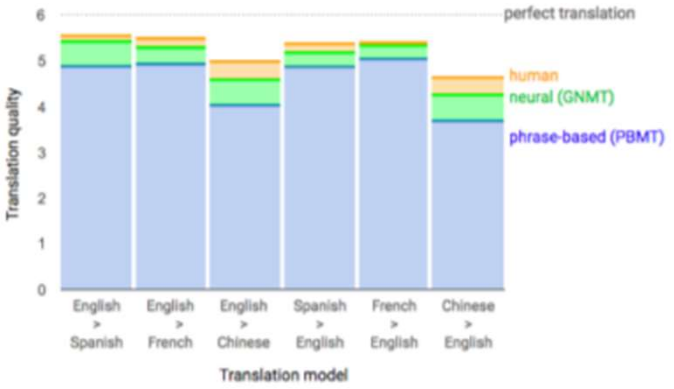
## A Brief Overview of Natural Language Processing

- Machine Learning brought huge successes

- Image Recognition
  - Widely used by Google, Facebook, etc.




- Machine Translation
  - Google Translate, etc.

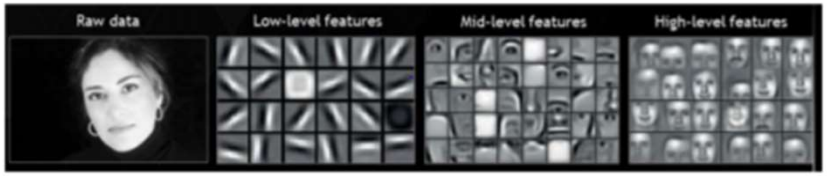


# Introduction

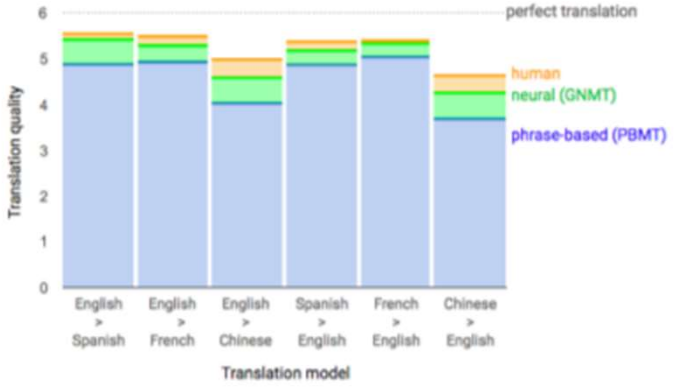
## A Brief Overview of Natural Language Processing

- Machine Learning brought huge successes

- Image Recognition
  - Widely used by Google, Facebook, etc.




- Machine Translation
  - Google Translate, etc.




- Game Playing
  - Atari Games, AlphaGo

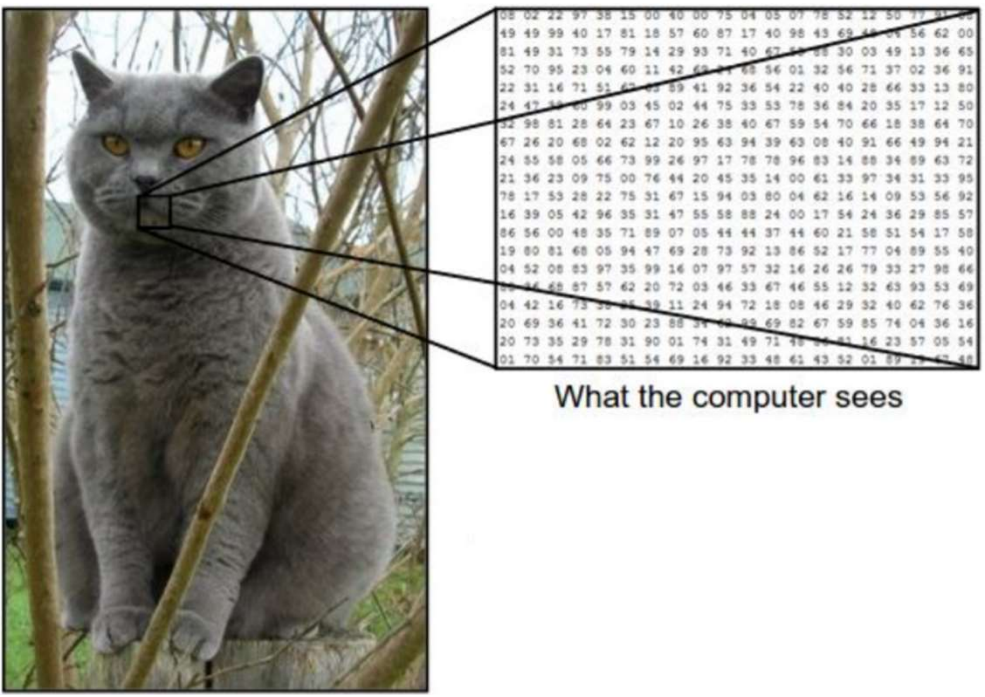
# Representational Learning with Word Embeddings

## Word Embedding Concepts

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- How do we represent words to a computer?
- Vision → data already comes to us in a format that is suitable for machine learning models

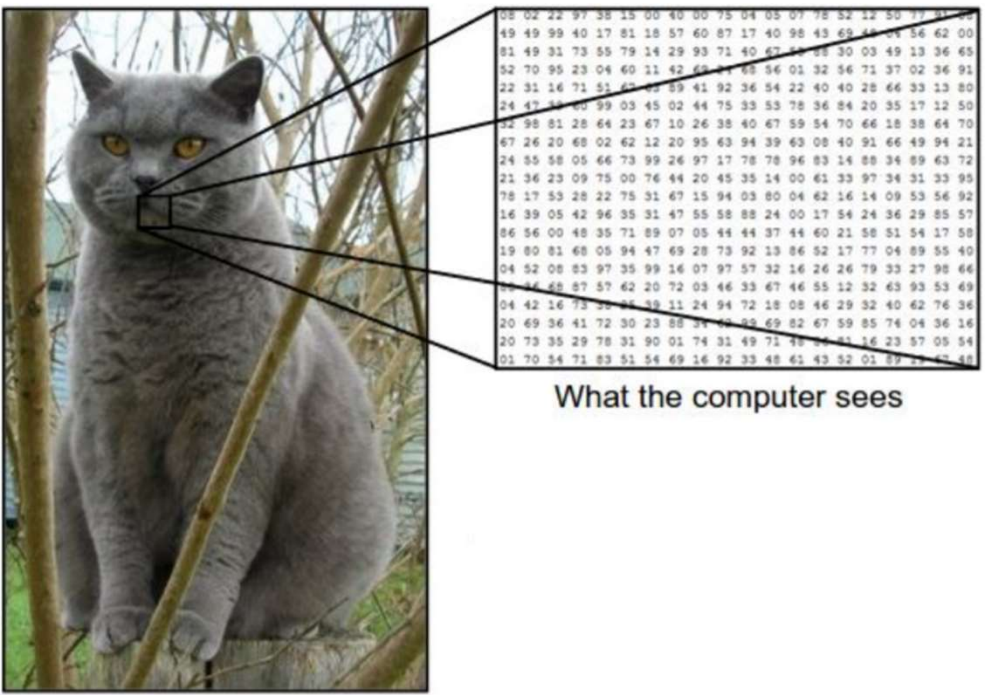



Camera gives visual data as matrix of RGB values

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- How do we represent words to a computer?
- Vision → data already comes to us in a format that is suitable for machine learning models
- Language → data is a sequence of words (“the cat jumped over the puddle”)
  - Need some form of **numeric** representation



What the computer sees

Camera gives visual data as matrix of RGB values

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- How do we represent words to a computer?
- Recall → want to convey the **meaning** of a word
- One common solution → use WordNet

*e.g. synonym sets containing "good":*

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

*e.g. hypernyms of "panda":*

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- WordNet
  - Great resource but missing nuance (context) of words
  - Missing meaning of new words
  - Subjective and requires human labor to maintain
  - Can’t compute accurate word similarities (no context!)

*e.g. synonym sets containing “good”:*

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

*e.g. hypernyms of “panda”:*

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- Naïve approach → Represent each word as a discrete symbol
  - Use a one-hot vector for each word
  - Length of vector = # of words in vocabulary (e.g., ~200K for English)

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- Naïve approach → Represent each word as a discrete symbol
  - Use a one-hot vector for each word
  - Length of vector = # of words in vocabulary (e.g., ~200K for English)

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vectors are orthogonal → no natural notion of **similarity** for one-hot vectors (motel ≈ hotel)

- How would we compute the similarity between “motel” and “hotel”?
- Could try to use WordNet’s list of synonyms to get similarity
  - WordNet is known to fail due to incompleteness, etc.

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- Better approach → **Learn to encode similarity** in the vectors themselves
  - Word vectors are sometimes called **word embeddings** or **word representations** → they are **dense** vectors and form a **distributed** representation
  - Length of vector → hyperparameter but much smaller than 200K

*banking* =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

*expect* =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- But how do we learn these vectors?

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- But how do we learn these vectors?

"You shall know a word by the company it keeps"

--- J. R. Firth (1957)

**Arguably, the most successful idea in NLP**

# Representational Learning with Word Embeddings

## Word Embedding Concepts

- But how do we learn these vectors?

"You shall know a word by the company it keeps"

--- J. R. Firth (1957)

**Arguably, the most successful idea in NLP**

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
- Use the many contexts of a word, *w*, to build up a representation of *w*

| | | |
|---|---|---|
| *…government debt problems turning into* | ***banking*** | *crises as happened in 2009…* |
| *…saying that Europe needs unified* | ***banking*** | *regulation to replace the hodgepodge…* |
| *…India has just given its* | ***banking*** | *system a shot in the arm…* |

These context words will represent ***banking***

# Representational Learning with Word Embeddings

## Context-Free Embeddings

# Representational Learning with Word Embeddings

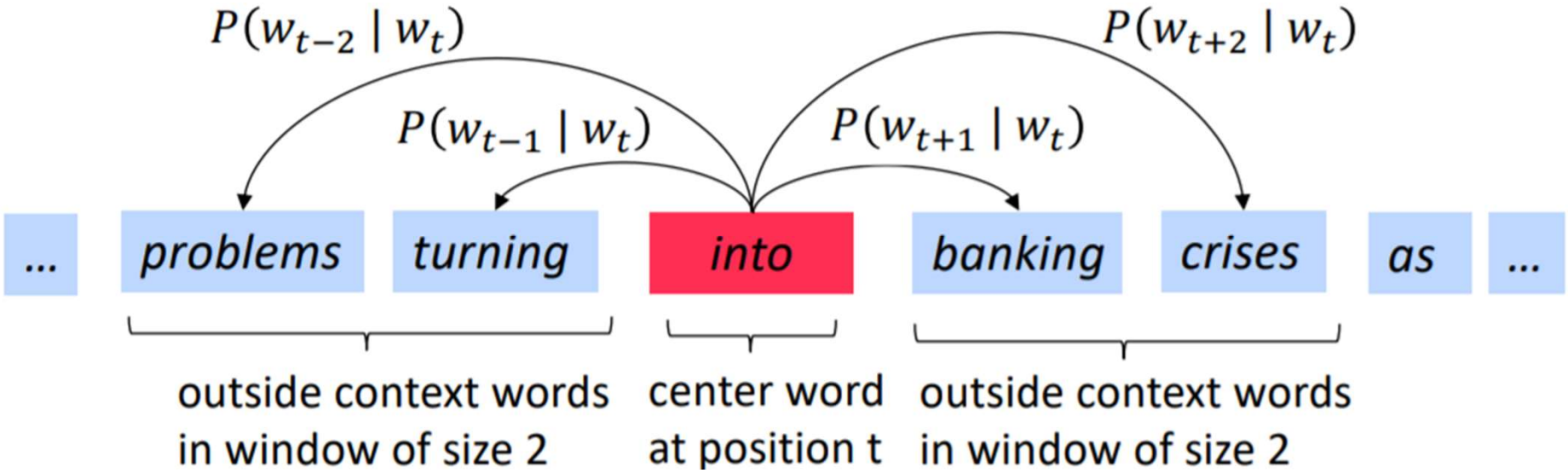
## Context-Free Embeddings

- But how do we learn these vectors?
- Word2Vec approach
  1) We have a large corpus of (unlabeled) text → unsupervised training
  2) Every word represented by a dense vector
  3) Go through each position *t* in the text, which has a center word *c* and context words *o*
  4) Use the similarity (i.e., dot product) of the word vectors for *c* and *o* to calculate the **probability** of *c* given *o* (CBOW) or vice versa (Skip-gram)

# Representational Learning with Word Embeddings

## Context-Free Embeddings

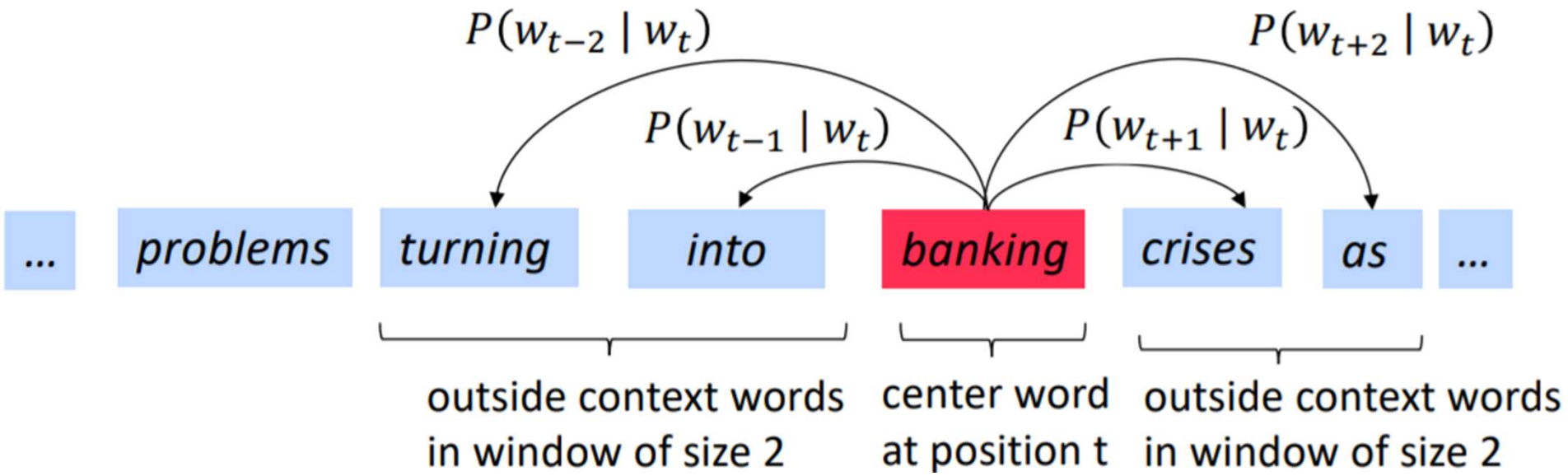
- But how do we learn these vectors?
- Word2Vec approach
  1) We have a large corpus of (unlabeled) text → unsupervised training
  2) Every word represented by a dense vector
  3) Go through each position *t* in the text, which has a center word *c* and context words *o*
  4) Use the similarity (i.e., dot product) of the word vectors for *c* and *o* to calculate the **probability** of *c* given *o* (CBOW) or vice versa (Skip-gram)



# Representational Learning with Word Embeddings

## Context-Free Embeddings

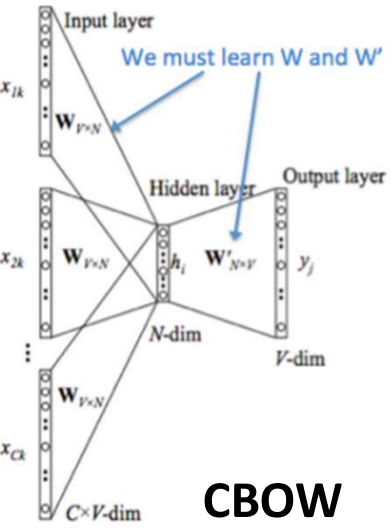
- But how do we learn these vectors?
- Word2Vec approach
  1) We have a large corpus of (unlabeled) text → unsupervised training
  2) Every word represented by a dense vector
  3) Go through each position *t* in the text, which has a center word *c* and context words *o*
  4) Use the similarity (i.e., dot product) of the word vectors for *c* and *o* to calculate the **probability** of *c* given *o* (CBOW) or vice versa (Skip-gram)



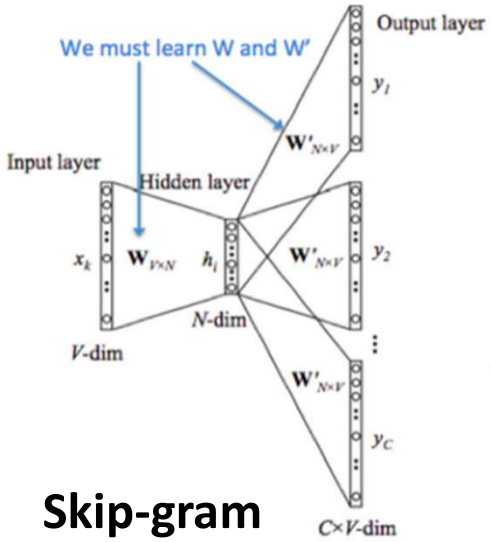
# Representational Learning with Word Embeddings

## Context-Free Embeddings

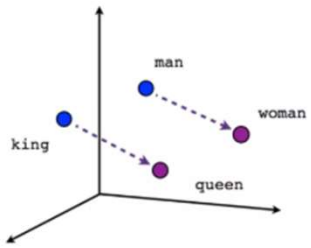
- But how do we learn these vectors?
- Word2Vec approach
  1) We have a large corpus of (unlabeled) text → unsupervised training
  2) Every word represented by a dense vector
  3) Go through each position *t* in the text, which has a center word *c* and context words *o*
  4) Use the similarity (i.e., dot product) of the word vectors for *c* and *o* to calculate the **probability** of *c* given *o* (CBOW) or vice versa (Skip-gram)



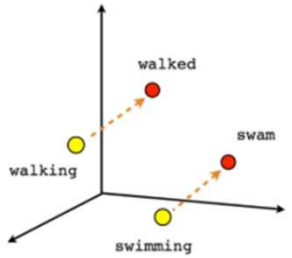

**CBOW**



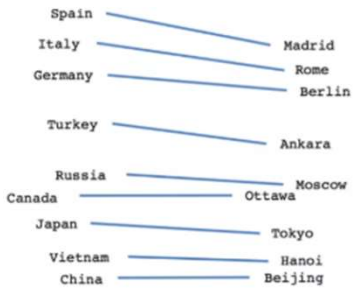

**Skip-gram**




Male-Female




Verb tense




Country-Capital

# Representational Learning with Word Embeddings

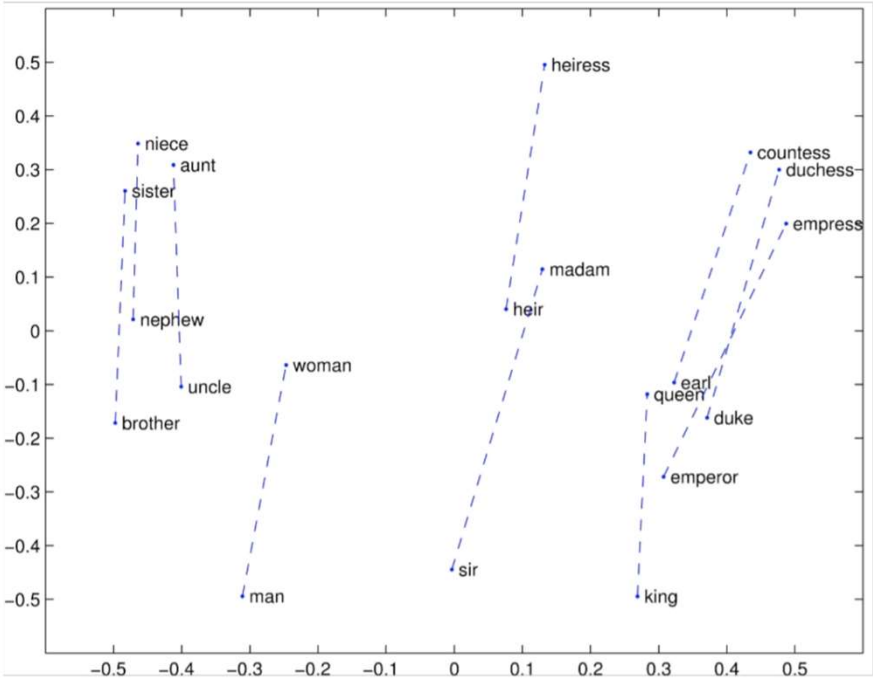
## Context-Free Embeddings

- Why is the Word2Vec model not enough?
  - Lacks **global statistics**
- GloVe: Global Vectors for Word Representation
  - **Co-occurrence ratios between two words in a context are strongly connected to meaning**
  - Build co-occurrence matrix between pairs of words in corpus
  - Train on word-word co-occurrence ratios → makes efficient use of global statistics

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

| | $x$ = solid | $x$ = gas | $x$ = water | $x$ = fashion |
|---|---|---|---|---|
| $P(x\|\text{ice})$ | $1.9 \times 10^{-4}$ | $6.6 \times 10^{-5}$ | $3.0 \times 10^{-3}$ | $1.7 \times 10^{-5}$ |
| $P(x\|\text{steam})$ | $2.2 \times 10^{-5}$ | $7.8 \times 10^{-4}$ | $2.2 \times 10^{-3}$ | $1.8 \times 10^{-5}$ |
| $\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$ | 8.9 | $8.5 \times 10^{-2}$ | 1.36 | 0.96 |

**Glove Visualizations**



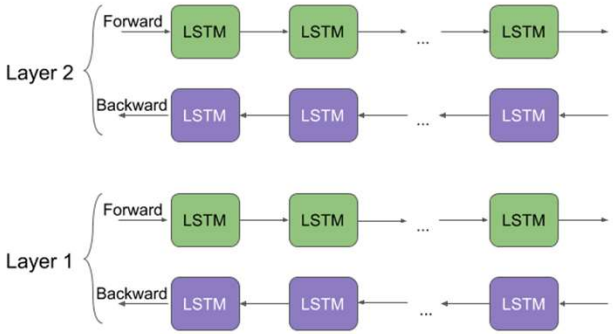
# Representational Learning with Word Embeddings

## Conditional Embeddings

# Representational Learning with Word Embeddings

## Conditional Embeddings

- Why is the GloVe model not enough?
  - GloVe captures global statistics but not **bi-directional context**
  - No distinction between different OOV words
  - **GloVe embeddings are static**
- ELMo: Embeddings from Language Models
  - Learn word embeddings based on the context it's used in → **contextual word *token* embeddings**
  - Character-level embedding vectors → **creates embeddings on the fly by passing text through the model**

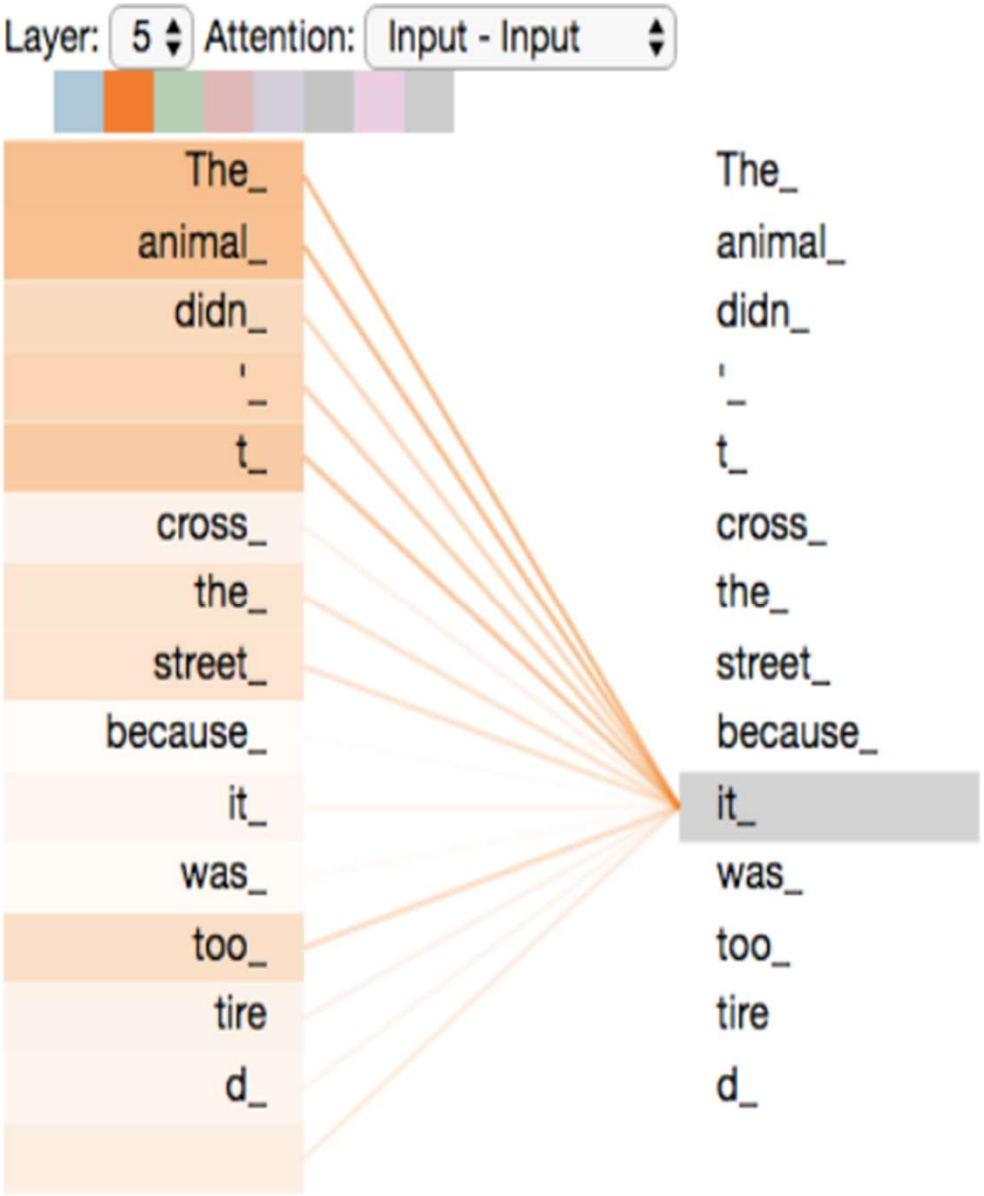



- Bi-directional LSTM for LM
  - Embedding is weighted sum of concatenated hidden layer vectors from forward and backward layers
  - Layer 1 → captures syntax
  - Layer 2 → captures semantics

# Representational Learning with Word Embeddings

## Conditional Embeddings

- Transformer models also learn conditional embeddings



# Representational Learning with Word Embeddings

## Other Seminal Works

# Representational Learning with Word Embeddings

## Other Seminal Works

- **Distributed Representations of Words and Phrases and their Compositionality** → Introduced three techniques to help Skip-Gram model train (Hierarchical Softmax using binary Huffman tree encoding, Negative Sampling, sub-sampling of frequent words)
- **Distributed Representations of Sentences and Documents** → Learn fixed representations of variable-length pieces of text (sentences, paragraphs, documents, etc.)
- **Skip-Thought Vectors** → Encode a sentence to predict sentences around it
- **Enriching Word Vectors with Subword Information** → Learn representations for character n-grams, and represent words as the sum of n-gram vectors
- **Neural Machine Translation of Rare Words with Subword Units** → Byte-pair encoding of words

# Language Modeling and Machine Translation

## Language Modeling

# Language Modeling and Machine Translation

## Language Modeling

- A **language model (LM)** is a model that **generates a probability distribution** over sequences of words
  - Technically, these models can only rely on *previous* words in a sequence

"The cat jumped over the puddle"

$P(w_1, w_2, \cdots, w_n)$ — P(The, cat, jumped, over, the puddle)

$P(w_1, w_2, \cdots, w_n) = \prod_{i=1}^{n} P(w_i)$ — P(The)*P(cat)*P(jumped)*P(over)*P(the)*P(puddle)
**Uni-gram model**

$P(w_1, w_2, \cdots, w_n) = \prod_{i=2}^{n} P(w_i|w_{i-1})$ — P(The|-)*P(cat|The)*P(jumped|cat)*P(over|jumped)*P(the|over)*P(puddle|the)
**Bi-gram model**

- Given a starting word, **how do we predict the rest of the words in the sequence**?
- This question encapsulates the majority of tasks in NLP
  - Q&A → Given a question, how do we generate a sensible answer?
  - Sentiment analysis → Given a statement, how do we classify its sentiment?
  - Paraphrasing → Given a pair of sentences, are they semantically equivalent?
  - Inference → Given a pair of sentences, how does the second sentence relate to the first?

# Language Modeling and Machine Translation

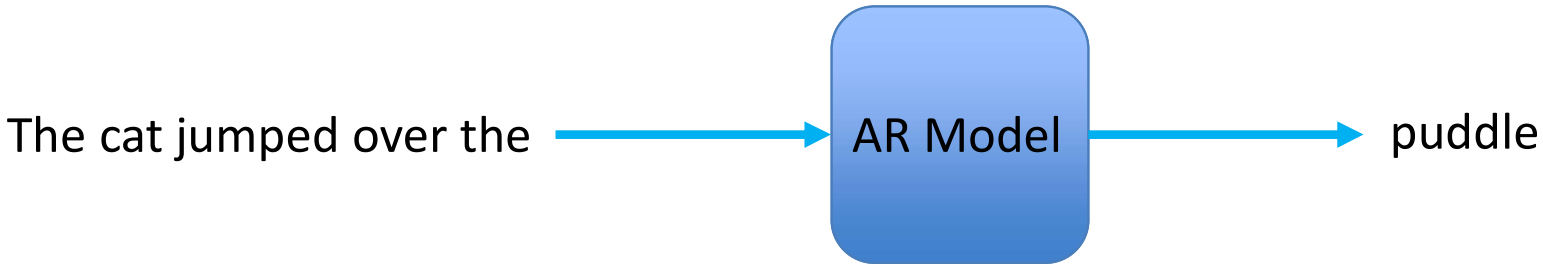
## Language Modeling

- LMs typically fall under one of two categories:

# Language Modeling and Machine Translation

## Language Modeling

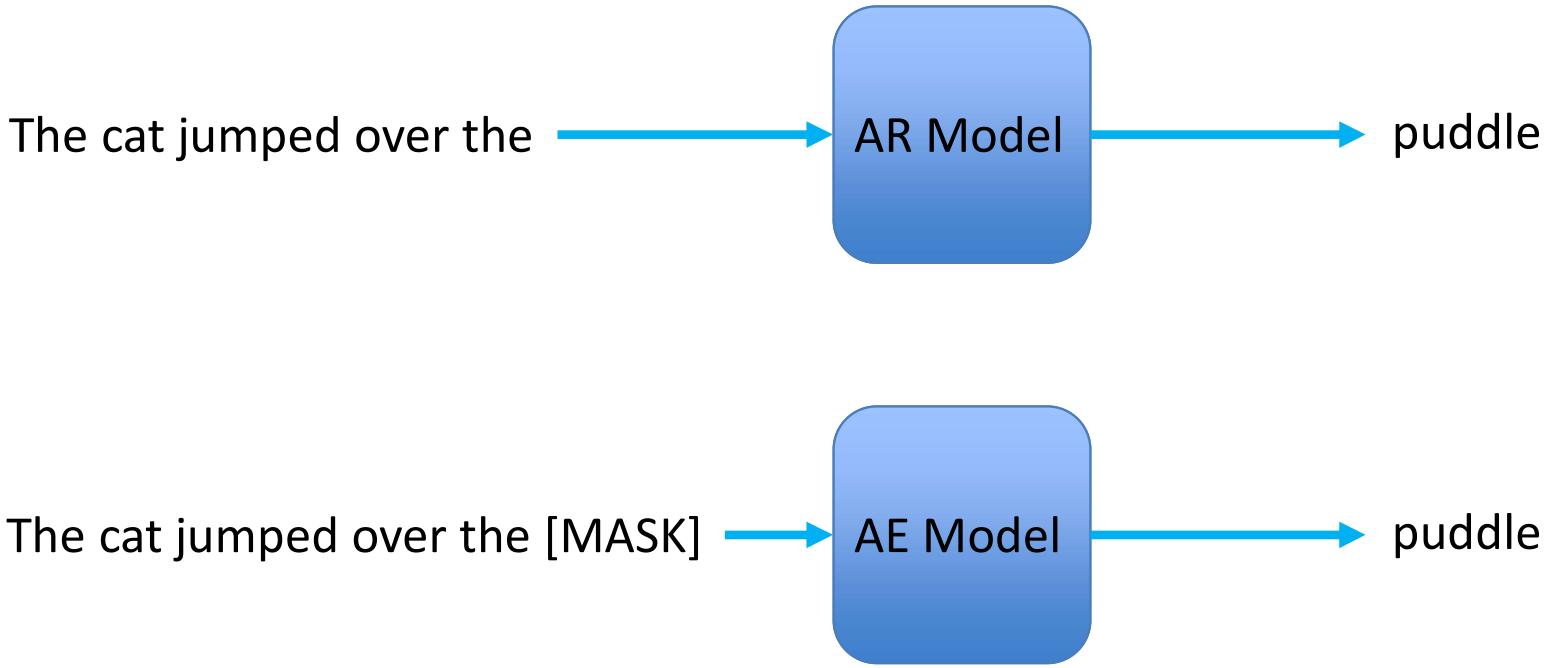
- LMs typically fall under one of two categories:
  - **Autoregressive** (AR) → Estimate probability distribution of next word/segment by conditioning on the history (Seq2Seq, ELMo, Transformer-XL, XLNET)



# Language Modeling and Machine Translation

## Language Modeling

- LMs typically fall under one of two categories:
  - **Autoregressive** (AR) → Estimate probability distribution of next word/segment by conditioning on the history (Seq2Seq, ELMo, Transformer-XL, XLNET)
  - **Autoencoding** (AE) → Reconstruct the original data from a corrupted input; e.g., by randomly masking input words (MaskGAN, BERT)



# Language Modeling and Machine Translation

## Language Modeling

- Other seminal works
  - **Class-Based *n*-gram Models of Natural Language** → Introduced *n*-gram LMs
  - **A Neural Probabilistic Language Model** → First model to jointly learn a distributed representation of words and a probability distribution for word sequences

# Language Modeling and Machine Translation

## Machine Translation

# Language Modeling and Machine Translation

## Machine Translation

- **Machine translation** is a major use case of a neural network architecture called **sequence-to-sequence*** and is improved by a technique called **attention**

*Now, it's probably Transformers

# Language Modeling and Machine Translation

## Machine Translation

- **Machine translation** is a major use case of a neural network architecture called **sequence-to-sequence*** and is improved by a technique called **attention**
- **Machine translation** is the task of translating a Sentence *x* from one domain to a Sentence *y* in another domain

*x:* *L'homme est né libre, et partout il est dans les fers*



*y:* *Man is born free, but everywhere he is in chains*

*Now, it's probably Transformers

# Language Modeling and Machine Translation

## Machine Translation

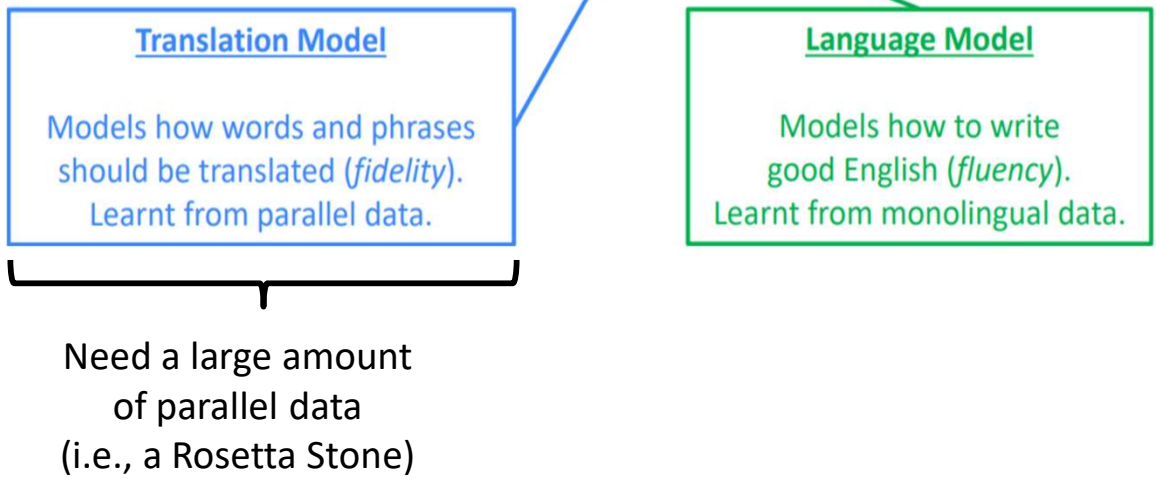
- 1950s: Systems were rule-based → used bilingual dictionary to map words from one language to another

# Language Modeling and Machine Translation

## Machine Translation

- 1950s: Systems were rule-based → used bilingual dictionary to map words from one language to another
- 1990s – 2010s: Statistical Machine Translation (SMT) → Learn a probabilistic model from data → very complex models

Given French sentence *x*, find the best English sentence *y* $\text{argmax}_y P(x|y)P(y)$



# Language Modeling and Machine Translation

## Machine Translation

- 2014: Neural Machine Translation (NMT) → Machine Translation with a single neural network
- First successful architecture is the Seq2Seq model which involves two RNNs (an encoder and a decoder)
- Seq2Seq models are ***conditional*** LMs
  - Language model → Decoder RNN predicts next word of the target sentence
  - Conditional → Predictions are *conditioned* on (representation of) source sentence
- Seq2Seq models are versatile since many NLP tasks can be viewed as seq2seq
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

# Language Modeling and Machine Translation

## Machine Translation

- Other “minor” details related to NMT
  - Decoding → Greedy vs. Beam-search
  - **How to interpret/debug results?**
  - **Model bias (he vs. she)**
  - Can’t easily specify rules or guidelines for translation like in SMT
  - **Model evaluation → BLEU, ROUGE, METEOR, F1, classification accuracy**
  - **Domain mismatch between training and test/fine-tune data**
  - Maintaining context over long sequences
  - **Low-resource language pairs**

# Recurrent Neural Networks for Sequence Learning

## “Sequence to Sequence Learning with Neural Networks”

# Recurrent Neural Networks for Sequence Learning

## "Sequence to Sequence Learning with Neural Networks"

- Before Seq2Seq models, translation systems were probabilistic models based on a combination of:
  - **Translation model**: tells us what a sentence/phrase in a source language most likely translates into
  - **Language model**: tells us how likely a given sentence/phrase is overall
- Most successful translation models were phrase-based
  - Consists of many small, sub-components that are tuned separately
  - Long-term dependencies were difficult to capture
- Seq2Seq models use LSTMs/Gated Recurrent Units (GRUs)
  - Generate *arbitrary* length output sequences after seeing the *entire* input
  - Single end-to-end trainable model that is made up of 2 RNNs
  - With attention, they can focus on specific parts of the input
  - Also referred to as "Encoder-Decoder" models

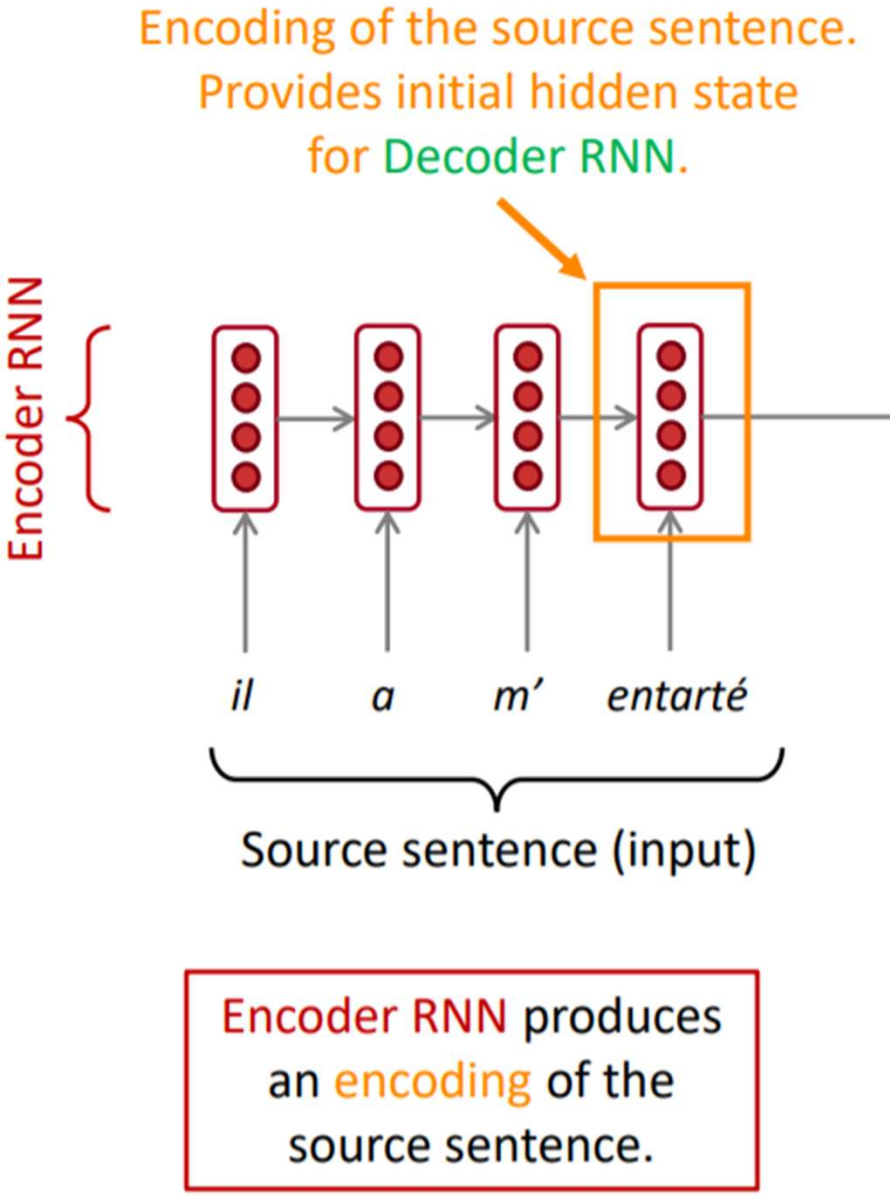
# Recurrent Neural Networks for Sequence Learning

## "Sequence to Sequence Learning with Neural Networks"

- One of two seminal papers to demonstrate an end-to-end approach to sequence learning using a domain-independent method
- Encoder-Decoder architecture using multi-layered LSTM cells
  - Encoder → Map input sequence to a vector of fixed dimensionality
  - Decoder → Conditional LM that decodes target sequence from fixed vector
- Translations tend to be paraphrases of the source sentences
  - By learning to map a variable-length input sentence into a fixed representation, the model learns to find sentence *representations* that capture their meaning → similar meaning sentences are "closer" to one another
- Reversing order of words in *source* sentences improved model performance
  - Reverse order allowed decoder to maintain short-term dependencies between source and target sentences (made optimization easier)
  - "jumped cat the" → el gato saltó

# Recurrent Neural Networks for Sequence Learning

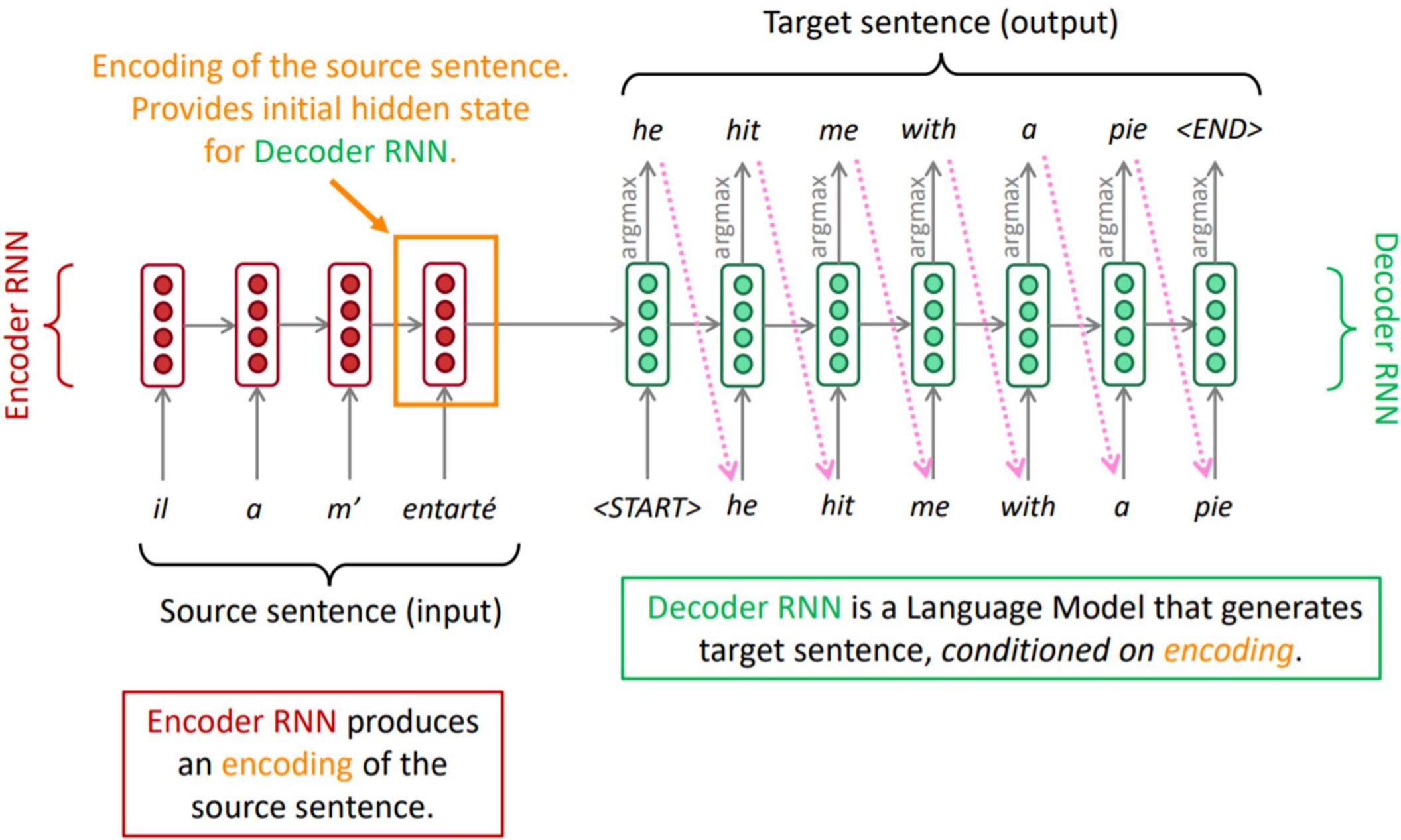
## "Sequence to Sequence Learning with Neural Networks"




Encoder RNN produces an encoding of the source sentence.

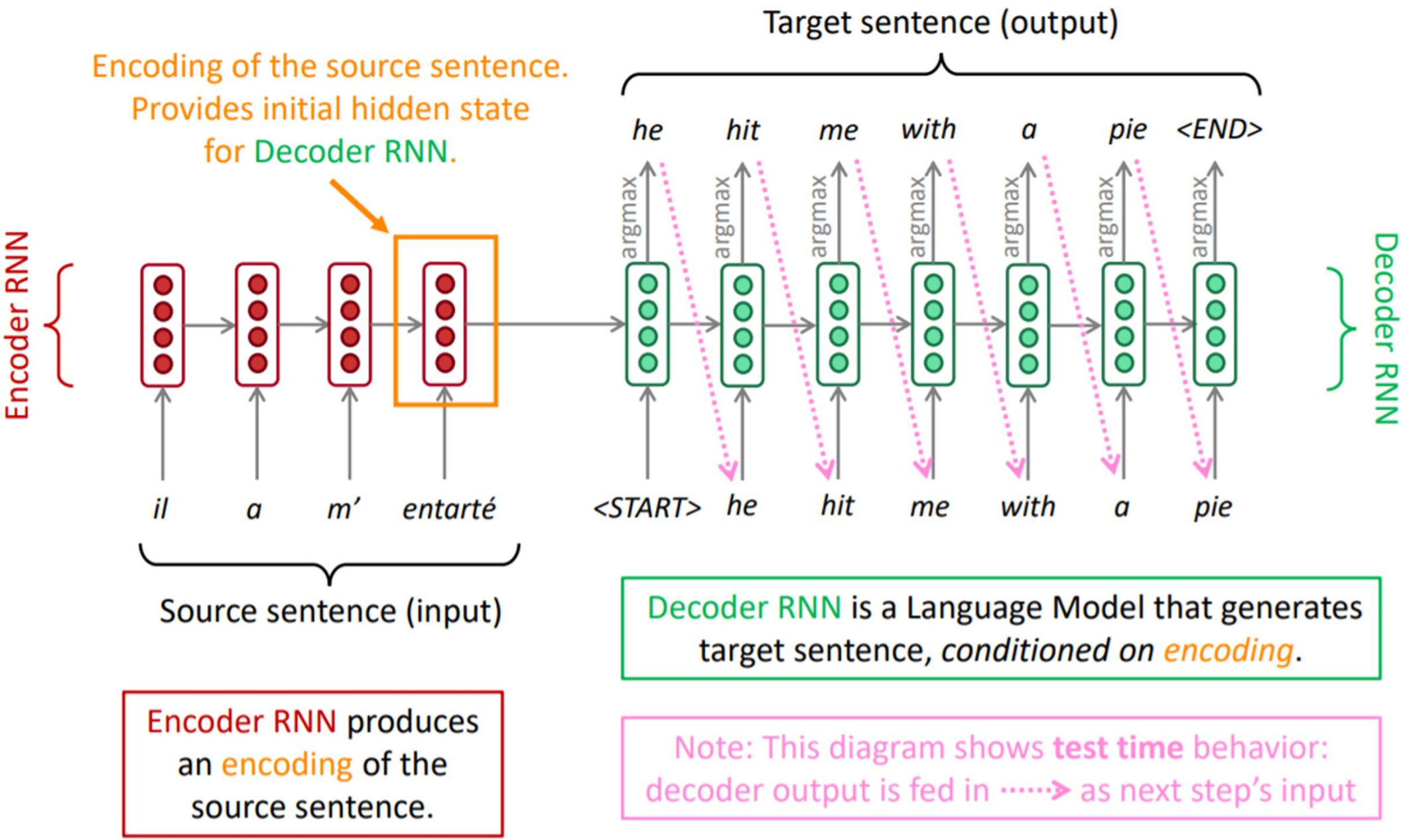
# Recurrent Neural Networks for Sequence Learning

## "Sequence to Sequence Learning with Neural Networks"



# Recurrent Neural Networks for Sequence Learning

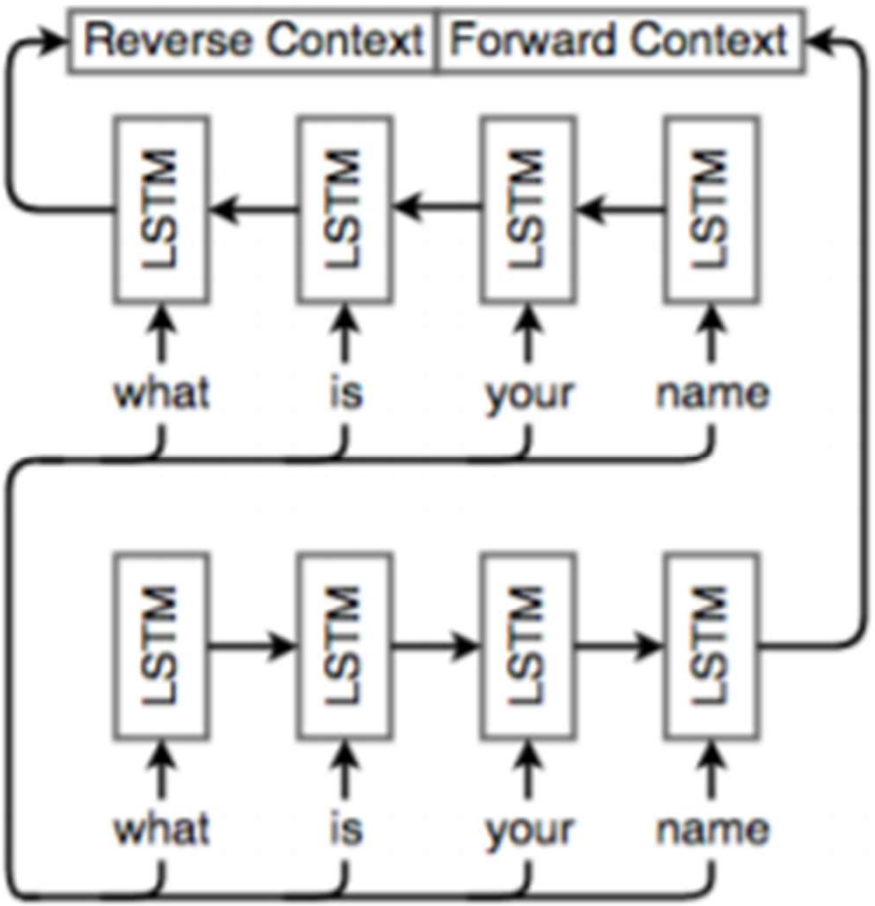
## "Sequence to Sequence Learning with Neural Networks"



# Recurrent Neural Networks for Sequence Learning

## "Sequence to Sequence Learning with Neural Networks"

- Seq2Seq models employ bi-directional RNNs (encoder only) so that the context of a word is learned from both directions
  - A word can have a dependency on another word before *or* after it




- Add another layer that traverses the input in reverse
- Concatenate the last hidden states of the forward and backward layers to use as the context vector for the decoder

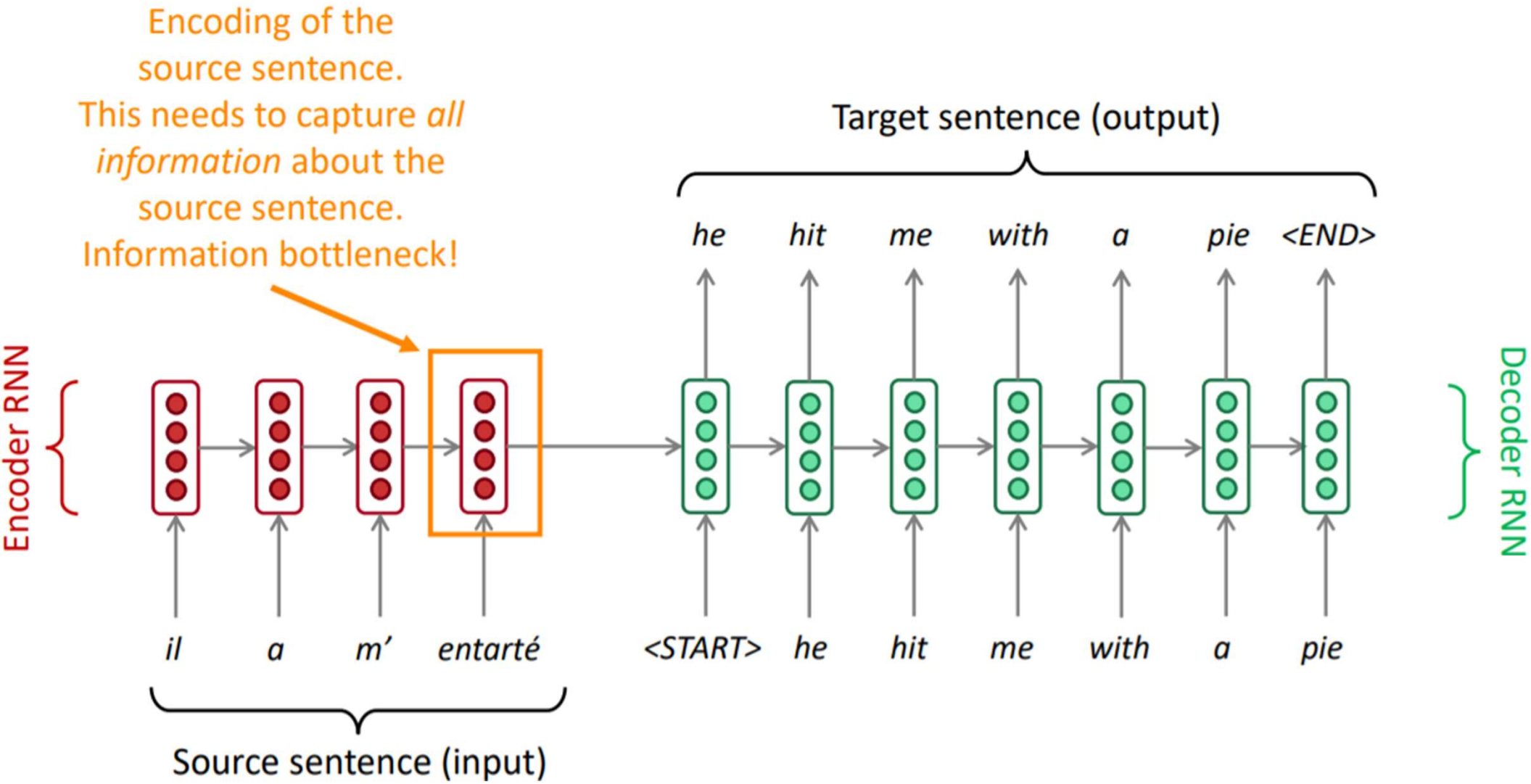
# Recurrent Neural Networks for Sequence Learning

## Limitations of Seq2Seq Models

# Recurrent Neural Networks for Sequence Learning

## Limitations of Seq2Seq Models

### Sequence-to-sequence: the bottleneck problem



# Recurrent Neural Networks for Sequence Learning

## Limitations of Seq2Seq Models

- Information bottleneck → unrealistic to expect a single vector to summarize the meaning of an arbitrary length sequence
- Seq2Seq models still have difficulty processing long sequences even with LSTM/GRU cells
- Seq2Seq models are not parallelizable
- No way to assign importance to words in the sequence
  - "the **ball** is **on** the **field**" → "**ball**", "**on**", and "**field**" are more "important" to the reader than "the" and "is"
  - Encoder → different parts of input have different levels of significance
  - Decoder → different parts of output may consider different parts of input "important"
- Attention mechanism addresses these issues
  - Provides the decoder with a view of the *entire input sequence* at **every** decoding step
  - Decoder can then decide which input words are important at any timestep

# Attention

## Introduction

# Attention

## Introduction


- Attention is a general deep learning technique
  - In early days (~1990s), used primarily in the field of visual imaging
  - Became “trendy” after Google DeepMind published “Recurrent Models of Visual Attention” in 2014 (application of Attention mechanism to RNN model for image classification)
- Basic idea for NLP
  - When model predicts a word, it only uses parts of input where the most relevant information is concentrated, instead of the entire input
  - Model only *pays attention to* some input words
- More formal definition
  - Given a set of vector ***Values*** and a vector ***Query***, **attention** is a technique to compute a weighted sum of the ***Values***, dependent on the ***Query***
  - ***Query*** *attends to* the ***Values*** (explained in detail coming up)

# Attention

## Introduction

- Attention mechanism for NLP was introduced to tackle the issue of memorizing long input sequences in NMT
- Attention is great
  - **Improves NMT performance** → Majority of SOTA results in NMT all achieved with Transformer-based models using Attention mechanisms
  - **Alleviates information bottleneck problem** → Decoder can directly look at all of input sequence
  - **Helps with vanishing and exploding gradient problems** → Provides shortcut to faraway input sequence states
  - **Offers model interpretability** → Attention matrix shows what decoder focuses on while generating output (model learns soft-alignment between input and output sequences for free)

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

# Attention

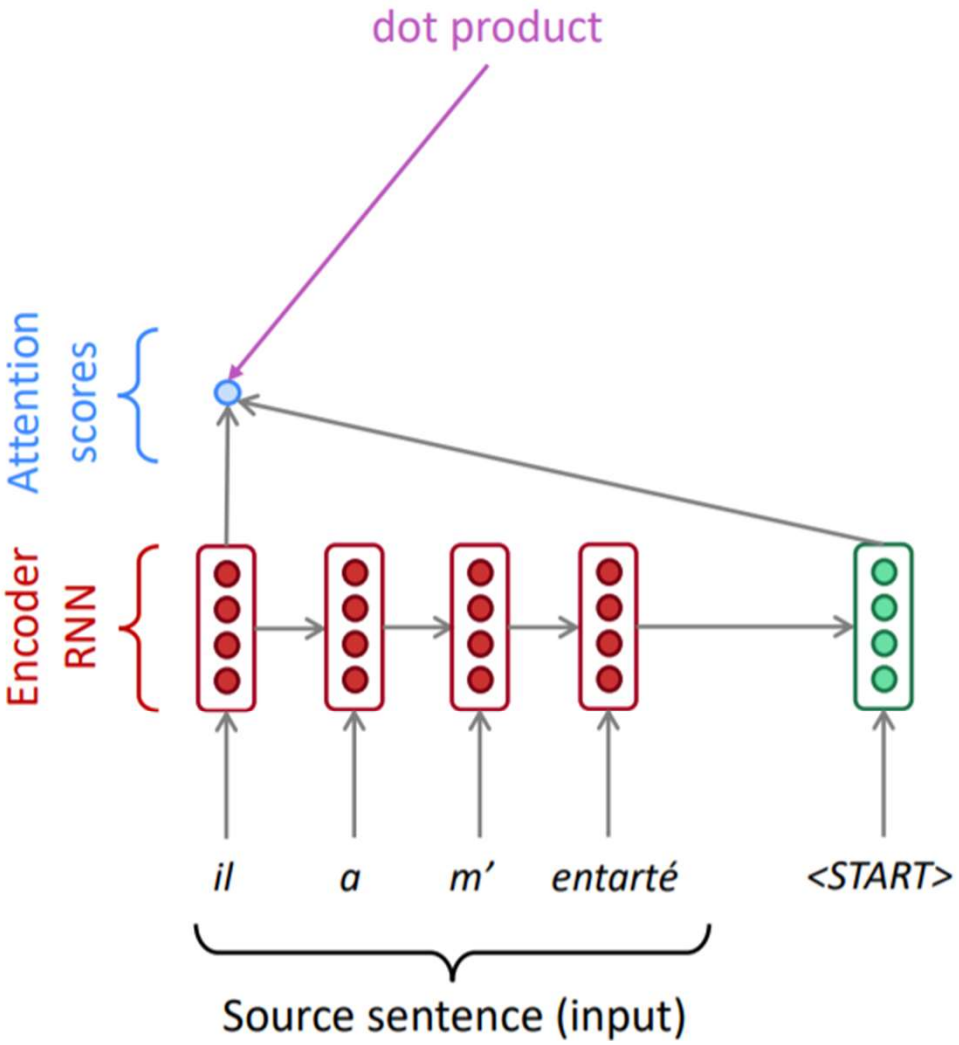
"Neural Machine Translation by Jointly Learning to Align and Translate"

**Sequence-to-sequence with attention**

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention




- For each decoder input, we compute the dot product between the previous hidden state of the decoder and each of the hidden state vectors from the encoder

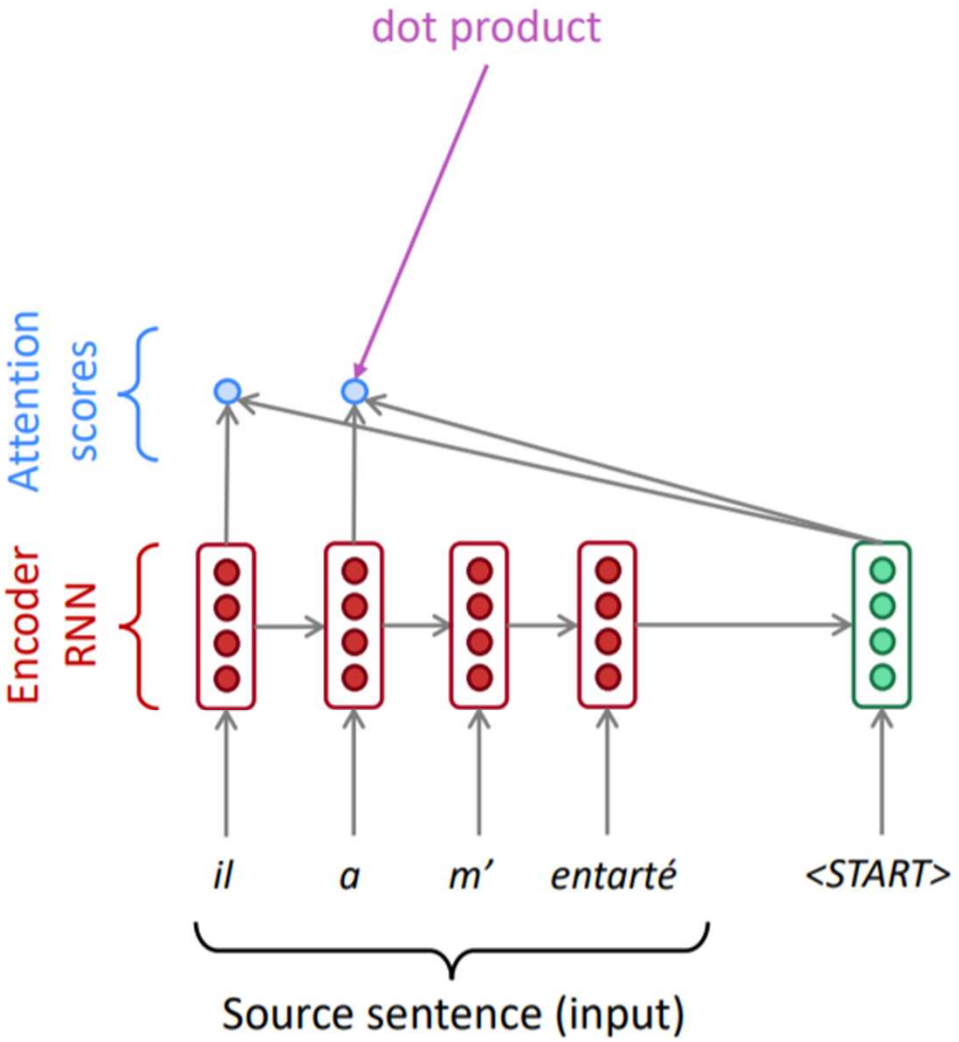
$$e_{ij} = a(s_{i-1}, h_j)$$

- $s_{i-1}$: Previous hidden state of decoder
- $h_j$: Hidden state $j$ from encoder
- *a:* Alignment model which scores how well the inputs around position *j* and the output at position *i* match
- $e_{ij}$: Attention score between input at position *j* and output at position *i*

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention




- For each decoder input, we compute the dot product between the previous hidden state of the decoder and each of the hidden state vectors from the encoder

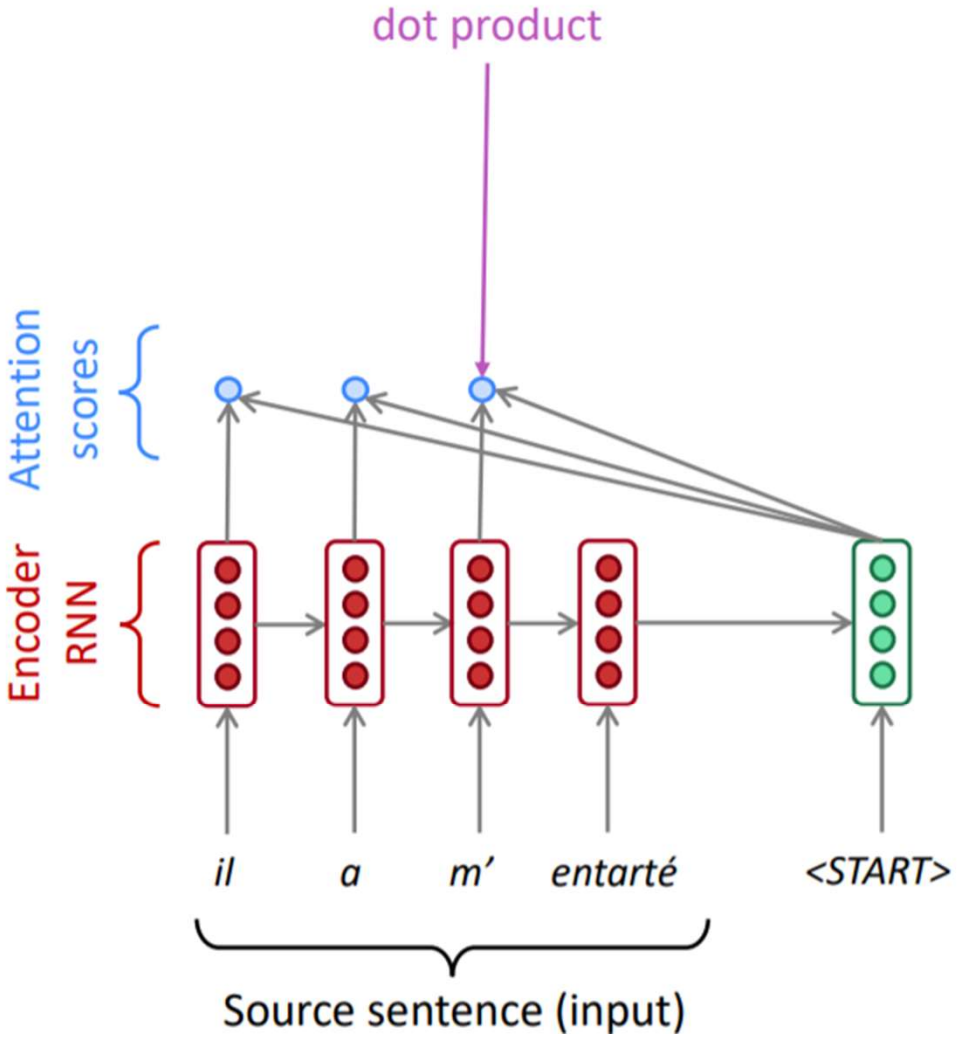
$$e_{ij} = a(s_{i-1}, h_j)$$

- $s_{i-1}$: Previous hidden state of decoder
- $h_j$: Hidden state $j$ from encoder
- *a:* Alignment model which scores how well the inputs around position *j* and the output at position *i* match
- $e_{ij}$: Attention score between input at position *j* and output at position *i*

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention




- For each decoder input, we compute the dot product between the previous hidden state of the decoder and each of the hidden state vectors from the encoder

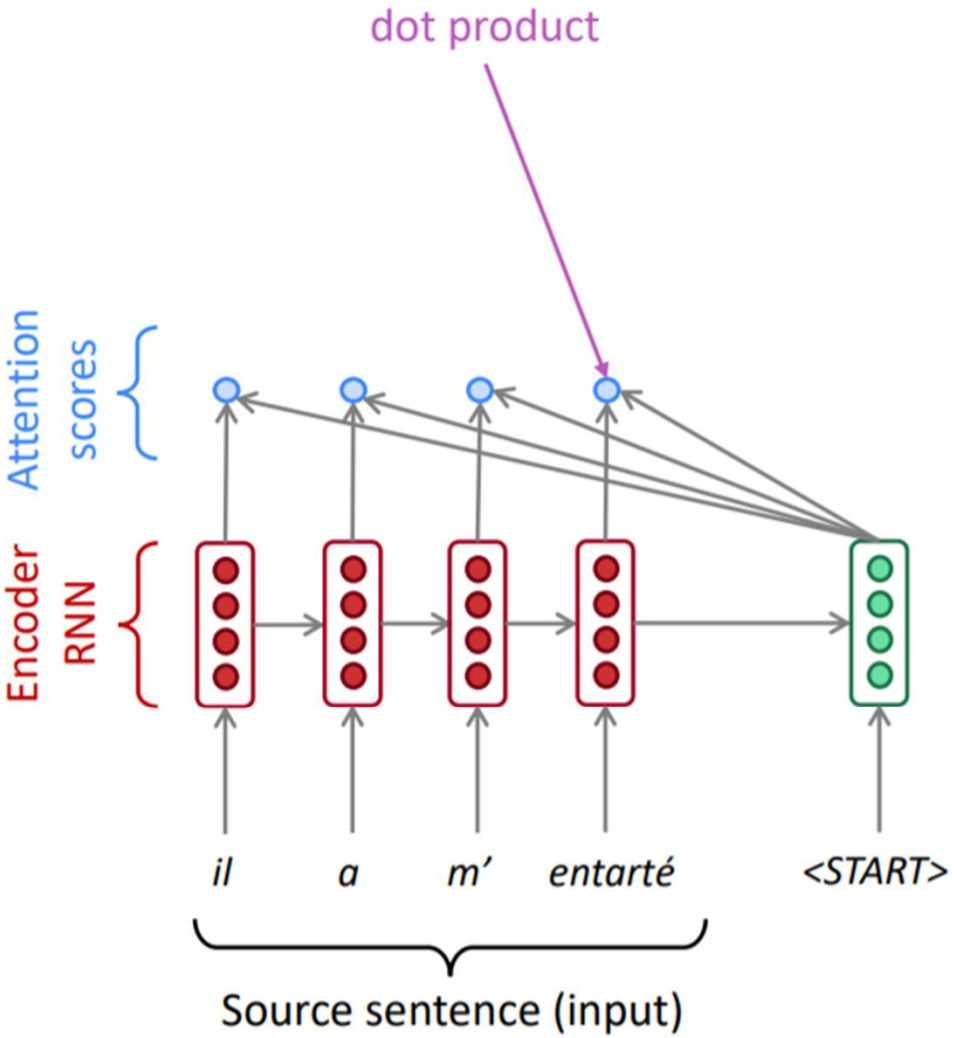
$$e_{ij} = a(s_{i-1}, h_j)$$

- $s_{i-1}$: Previous hidden state of decoder
- $h_j$: Hidden state $j$ from encoder
- *a:* Alignment model which scores how well the inputs around position *j* and the output at position *i* match
- $e_{ij}$: Attention score between input at position *j* and output at position *i*

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention




- For each decoder input, we compute the dot product between the previous hidden state of the decoder and each of the hidden state vectors from the encoder

$$e_{ij} = a(s_{i-1}, h_j)$$

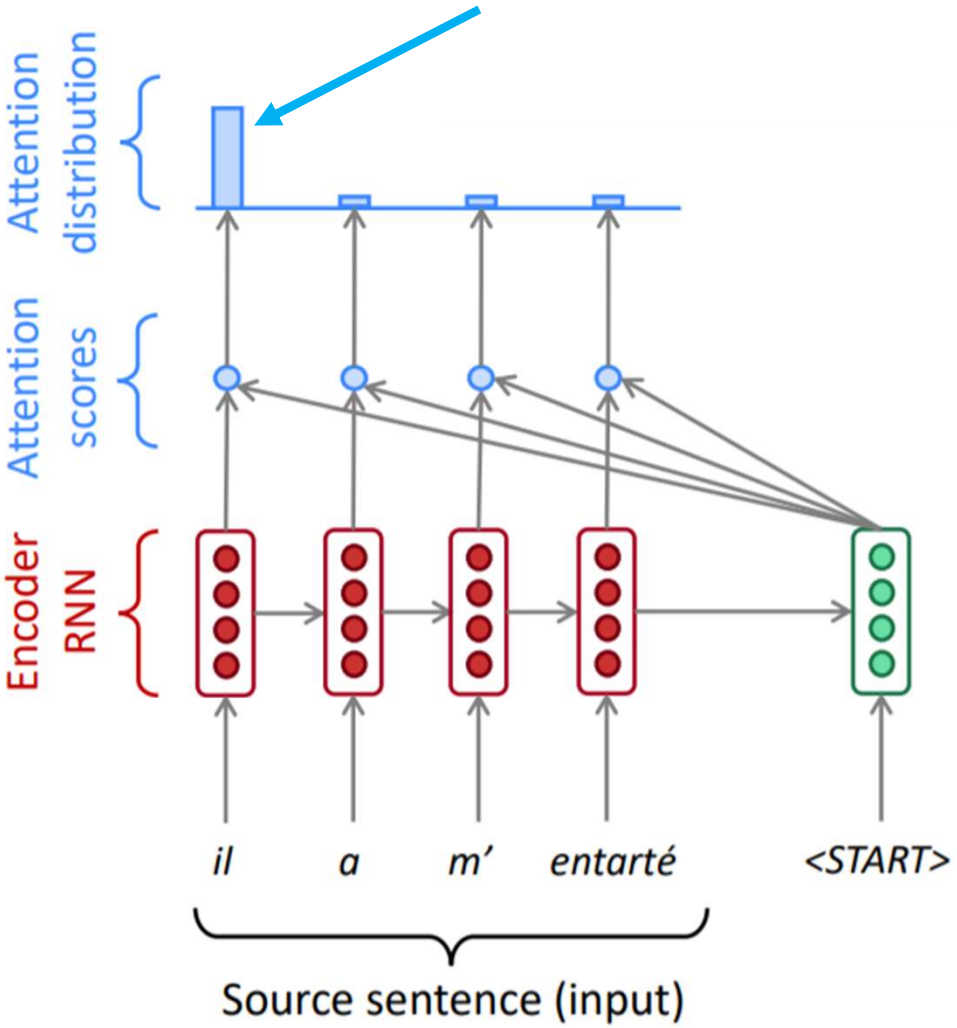
- $s_{i-1}$: Previous hidden state of decoder
- $h_j$: Hidden state $j$ from encoder
- *a:* Alignment model which scores how well the inputs around position *j* and the output at position *i* match
- $e_{ij}$: Attention score between input at position *j* and output at position *i*

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention

- At this decoder step, the model pays the most attention to the first position in the input




- Softmax over all attention scores to get attention probabilities

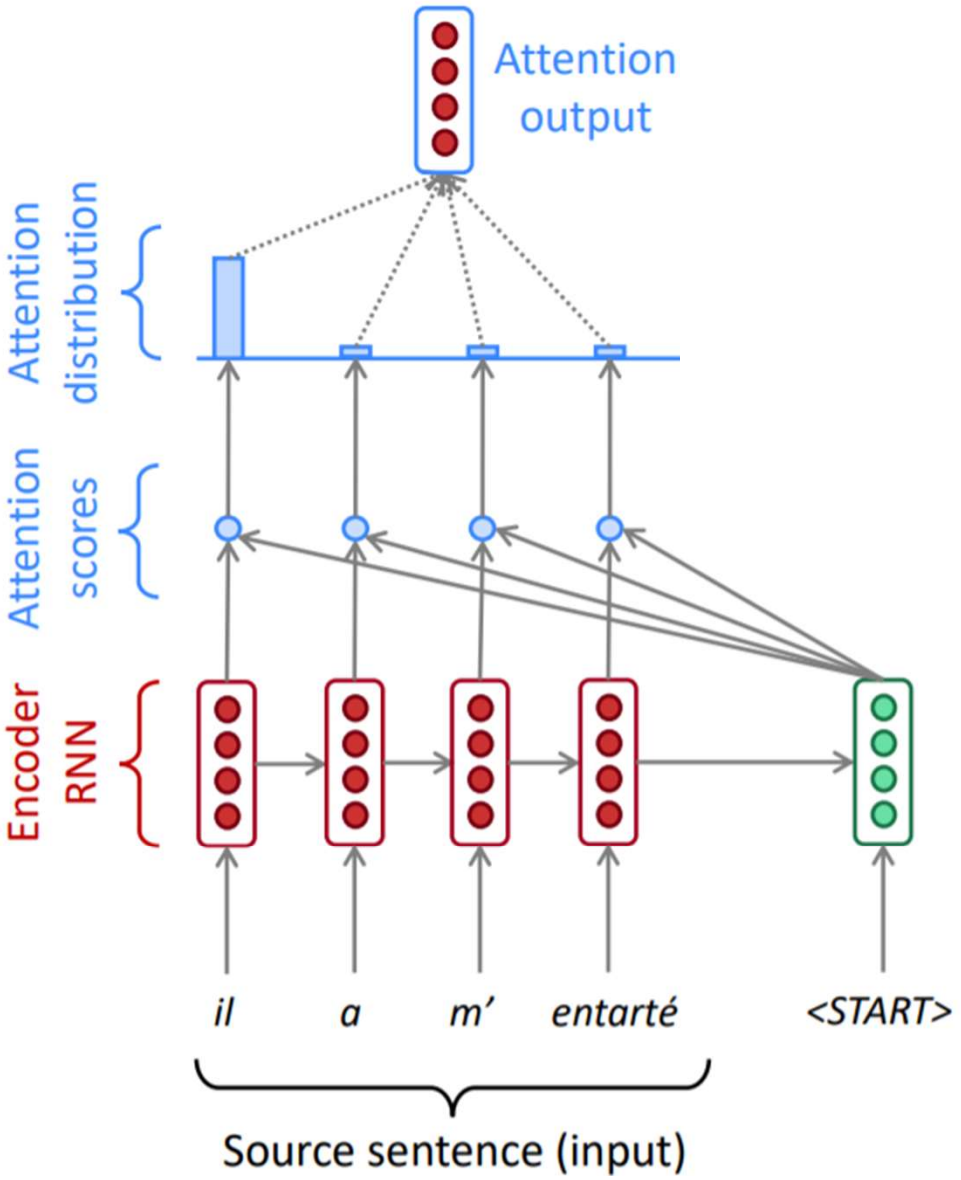
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

- $T_x$: Length of input sequence
- $\alpha_{ij}$: Attention probability between input at position *j* and output at position *i*

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention




- Context vector is now a weighted sum of the encoder hidden state vectors
- Attention output will mostly contain information from the encoder hidden states that received high attention

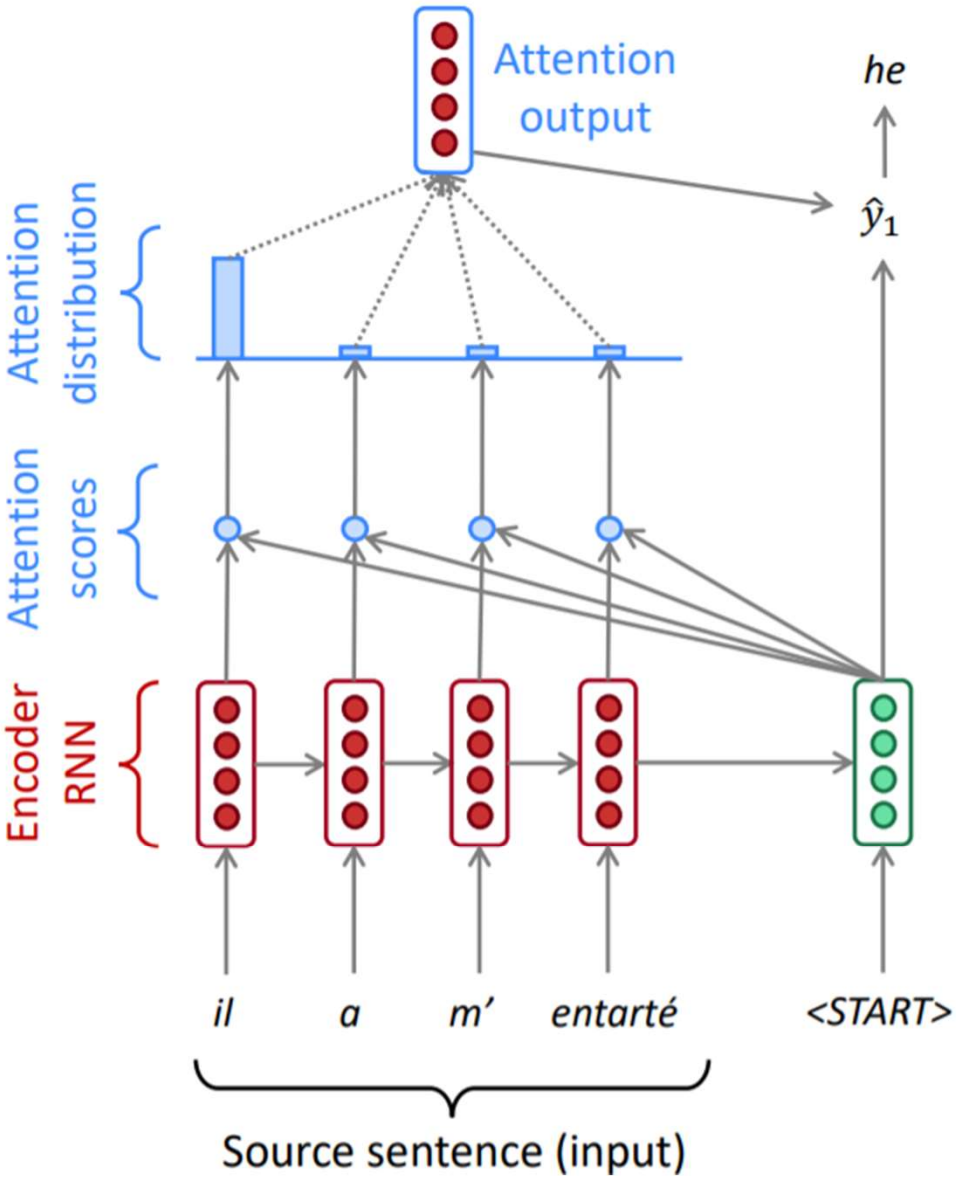
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

- $c_i$: Context vector for output position *i* as a function of each input position *j*, weighted by the agreement between *i* and *j*

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention




- Concatenate context vector with previous decoder hidden state, then compute prediction for current decoder step as usual

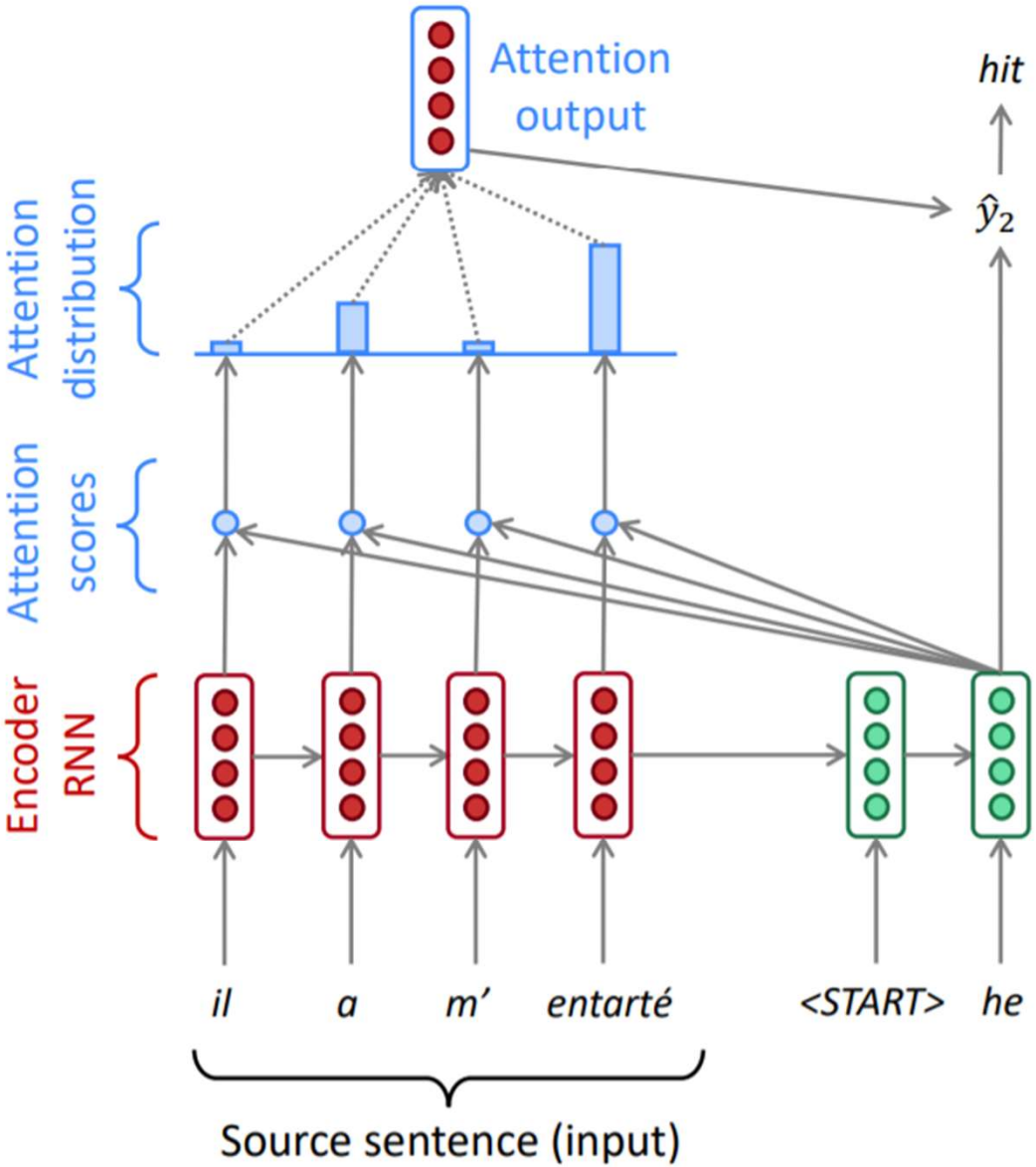
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

- $s_{i-1}$: Decoder hidden state from previous step
- $y_{i-1}$: Decoder prediction from previous step
- $f$: Some nonlinear functions (e.g., LSTM cell)
- $s_i$: Decoder hidden state at current step

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

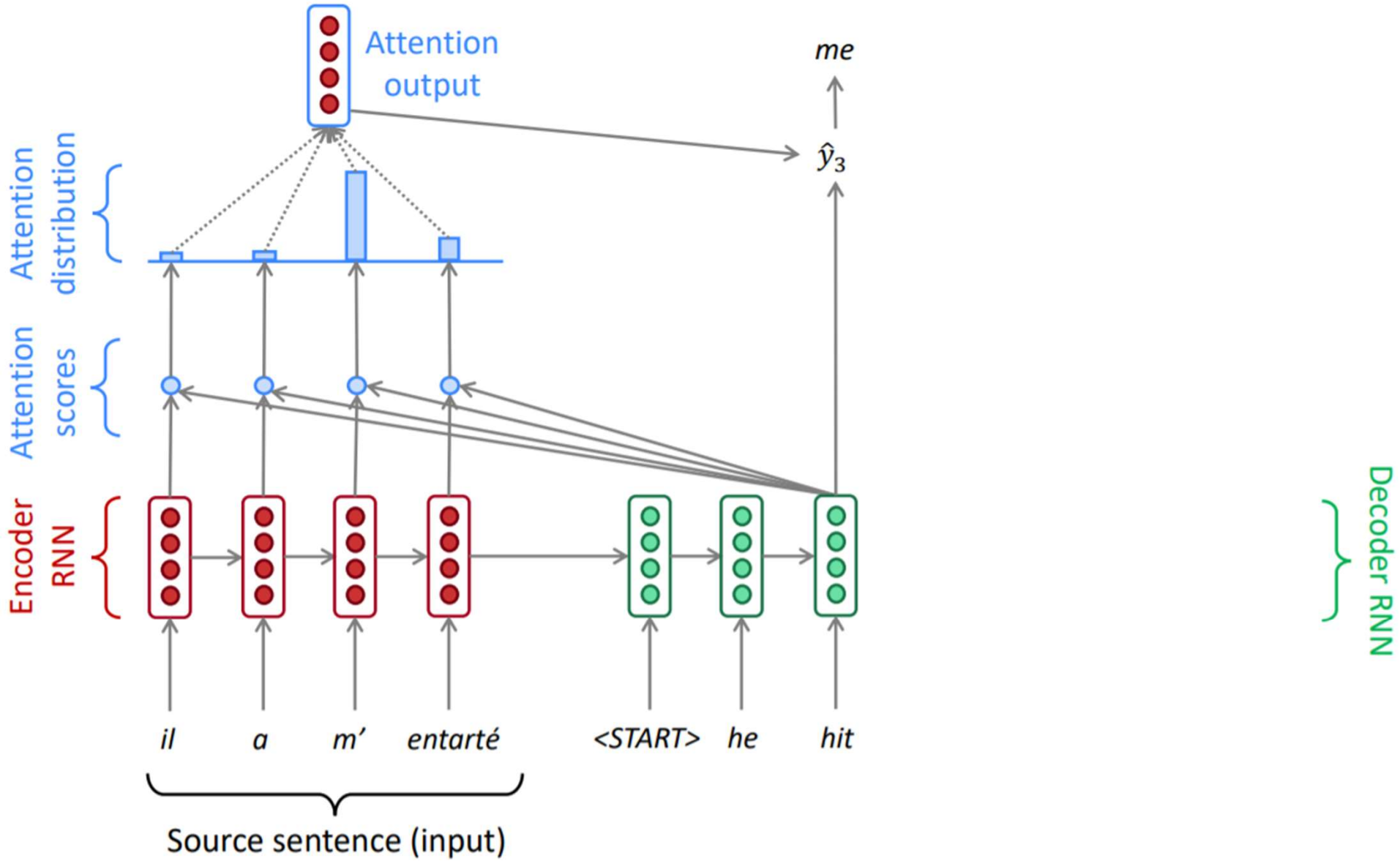
## Sequence-to-sequence with attention



# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

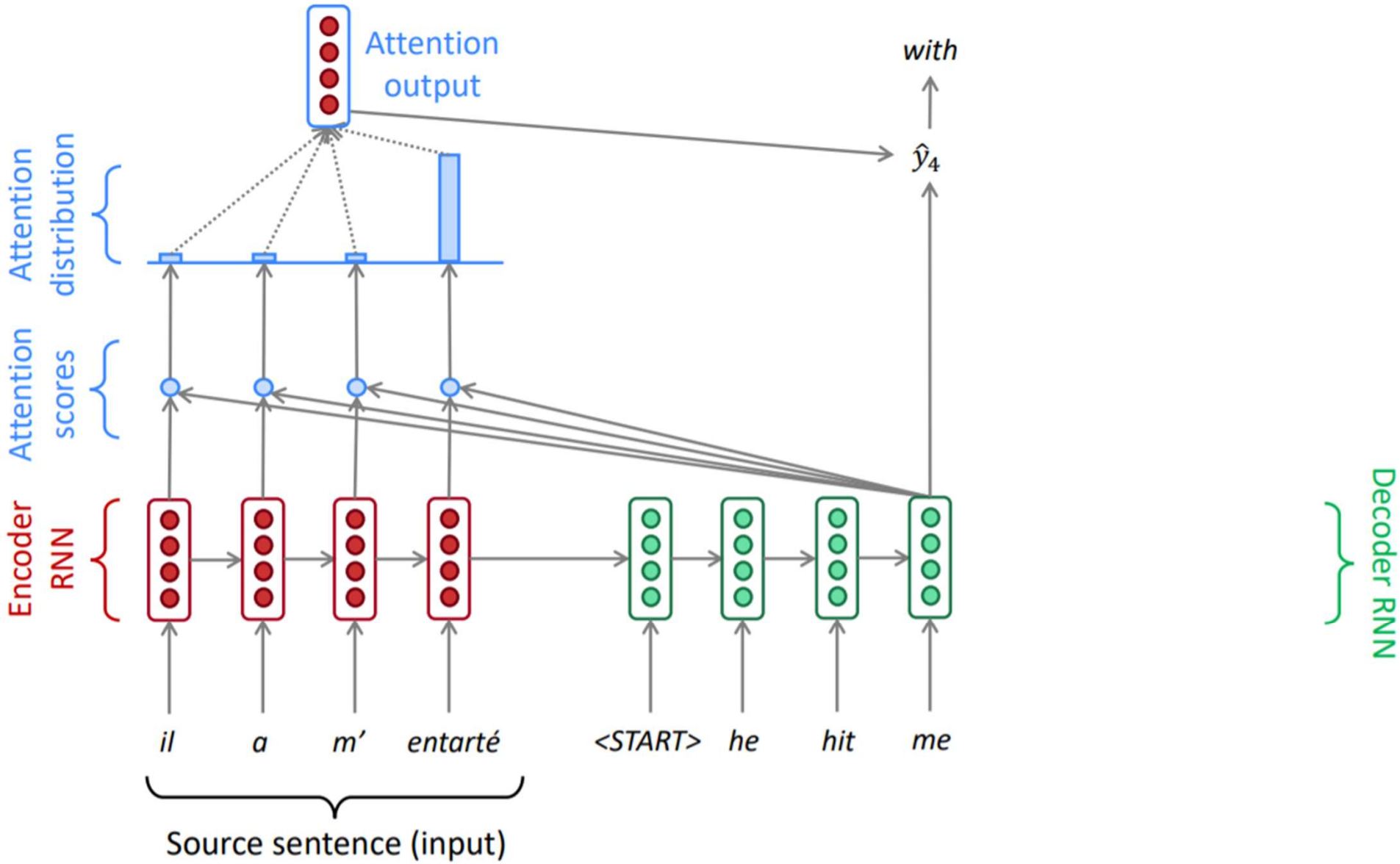
## Sequence-to-sequence with attention



# Attention

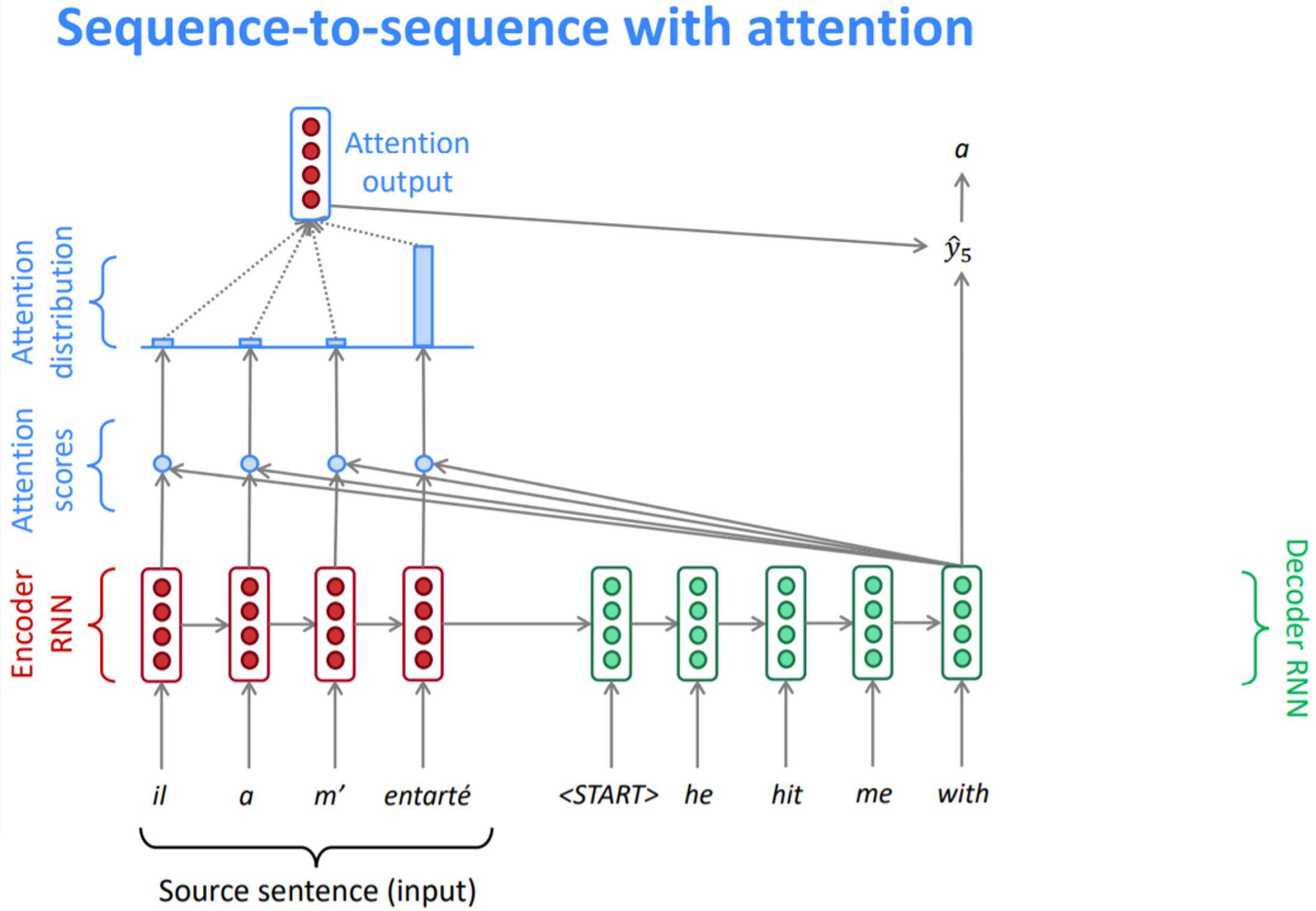
"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention



# Attention

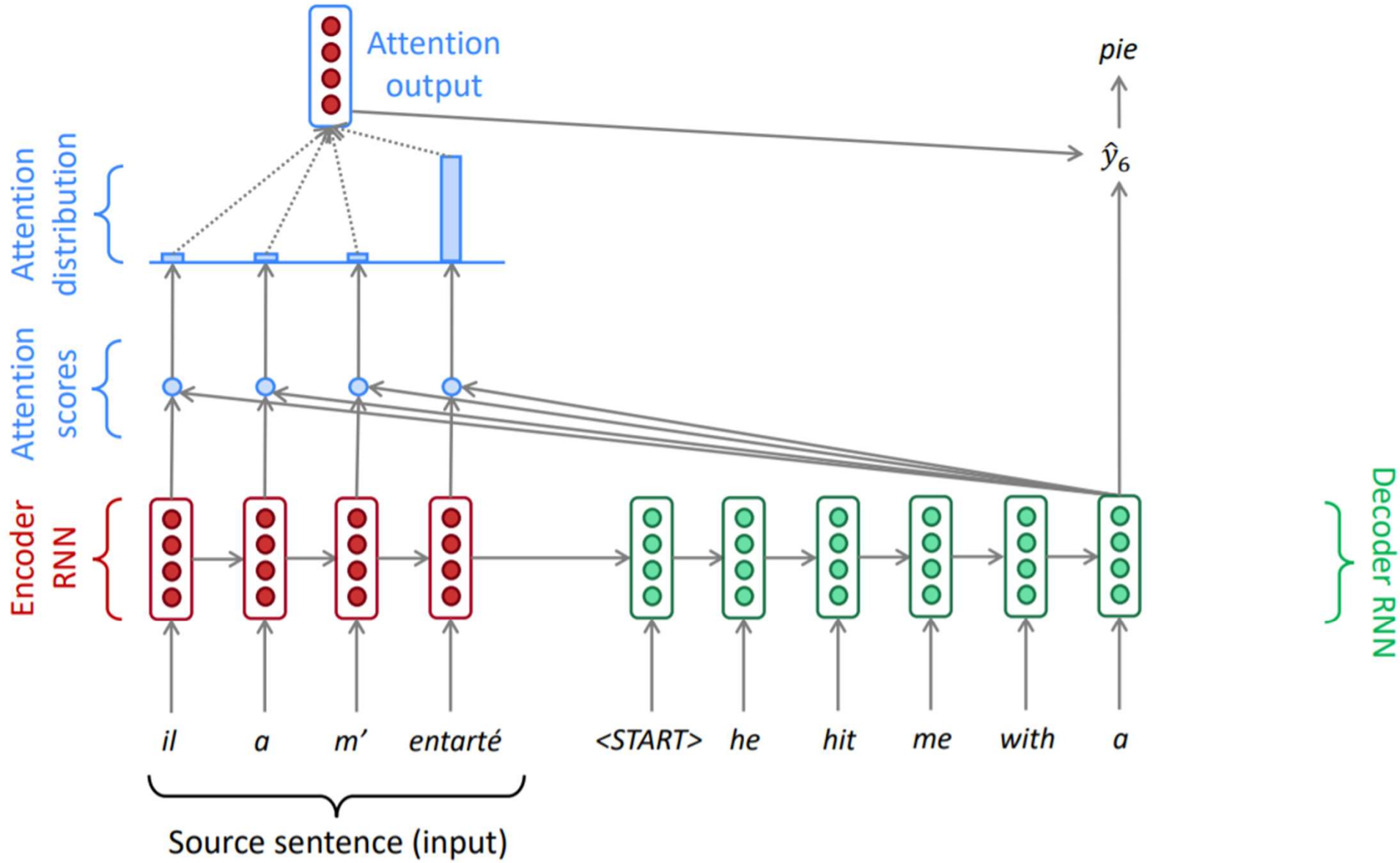
"Neural Machine Translation by Jointly Learning to Align and Translate"



# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

## Sequence-to-sequence with attention



# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

- Given a set of vector ***Values*** and a vector ***Query***, **attention** is a technique to compute a weighted sum of the ***Values***, dependent on the ***Query*** → ***Query*** *attends to* the ***Values***
- In Seq2Seq models with Attention, each decoder hidden state (***Query***) *attends to* all of the encoder hidden states (***Values***)
- Intuition
  - **Query** represents what kind of information we are looking for
  - **Keys** represent the actual contents of the inputs
  - **Values** represent the relevance of the **Key** to the **Query**

# Attention

"Neural Machine Translation by Jointly Learning to Align and Translate"

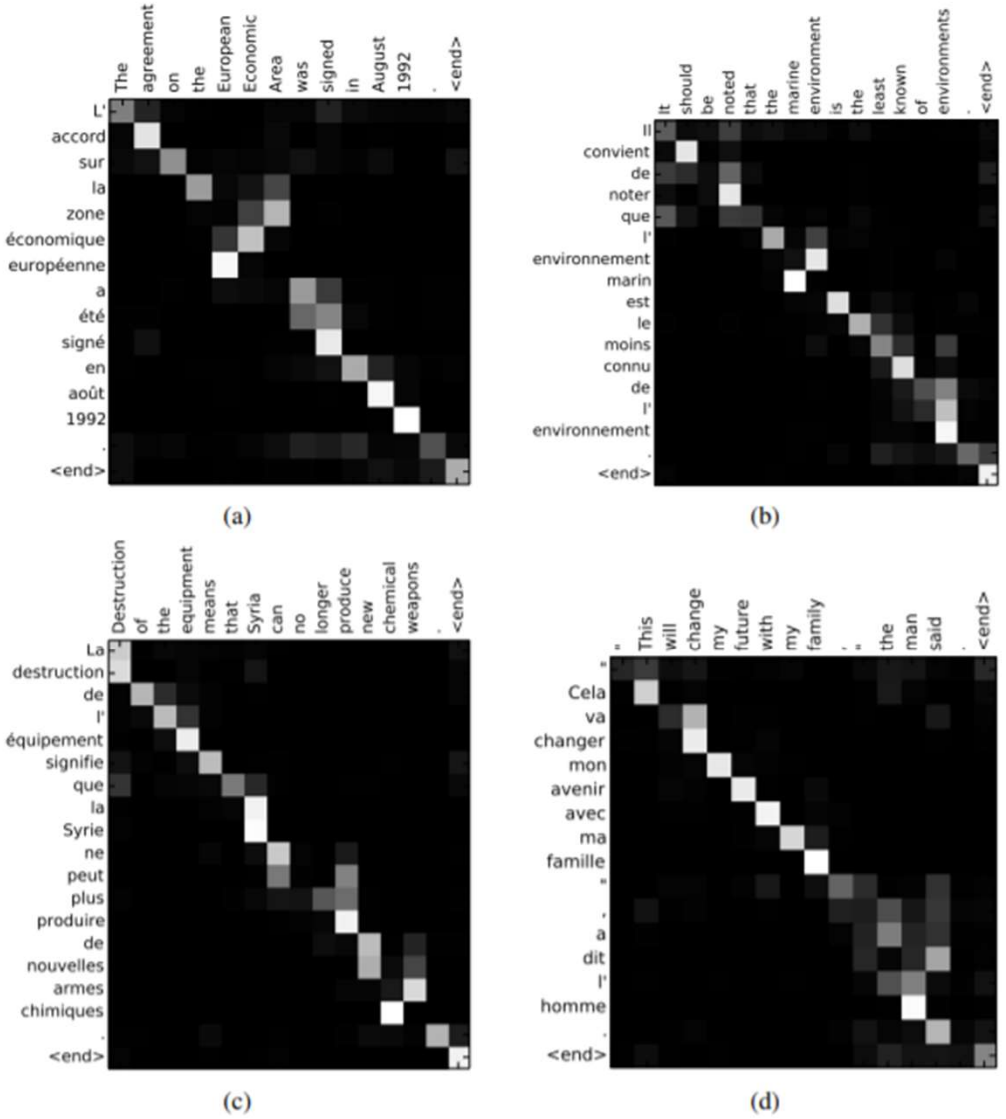



Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight $\alpha_{ij}$ of the annotation of the $j$-th source word for the $i$-th target word (see Eq. (6)), in grayscale (0: black, 1: white). (a) an arbitrary sentence. (b–d) three randomly selected samples among the sentences without any unknown words and of length between 10 and 20 words from the test set.

- Alignment model computes a soft alignment between the input and output sequence words
- For English-French translation, alignment is largely monotonic (high values along diagonal)

# Attention

## Types of Scoring Functions

# Attention

## Types of Scoring Functions

| Name | Alignment score function | Citation |
|---|---|---|
| Content-base attention | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$ | Graves2014 |
| Additive(*) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$ | Bahdanau2015 |
| Location-Base | $\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$<br>Note: This simplifies the softmax alignment to only depend on the target position. | Luong2015 |
| General | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$<br>where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer. | Luong2015 |
| Dot-Product | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$ | Luong2015 |
| Scaled Dot-Product(^) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$<br>Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state. | Vaswani2017 |

# Attention

## Types of Attention Mechanisms

# Attention

## Types of Attention Mechanisms

| Name | Definition | Citation |
|---|---|---|
| Self-Attention(&) | Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence. | Cheng2016 |
| Global/Soft | Attending to the entire input state space. | Xu2015 |
| Local/Hard | Attending to the part of input state space; i.e. a patch of the input image. | Xu2015; Luong2015 |

- **Global attention** → use all of the encoder hidden states when computing context vectors at each decoder step
  - Can be computationally expensive since model has to attend to all words in the input at each decoder step
- **Local attention** → Choose a position in the input sequence and only attend to a window of words centered at that position
  - Monotonic alignment → Assume input and output sequences are roughly monotonically aligned when choosing aligned position for attention
  - Predictive alignment → Model predicts an aligned position for attention

# Attention

## Types of Attention Mechanisms

- **Two-way attention** → Attention from input to output and output to input
  - Useful for textual entailment where task is to understand whether a premise entails the hypothesis)
- **Co-attention** → Compute alignment matrix on *all* pairs of context and query words
  - Seq2Seq Attention computes an alignment matrix using a single summary vector of the query to attend to the context
  - Useful for Q&A tasks
- **Hierarchical attention (Attention-over-Attention)** → Two levels of attention, one at word level and one at sentence level
  - Place a second attention mechanism over the primary attentions to indicate the "importance" of each attention
  - Texts tend to have hierarchical structure
- **Attention flow** → allow attention information to "flow" from one layer to the next in a model at each step; reduces information loss caused by early summarization
  - Attention network is decoupled from prediction network

# Attention

## Types of Attention Mechanisms

- **Self (Intra)-Attention** → Relates different positions of a *single* sequence to *itself* in order to compute attention
  - Useful for tasks like sentiment analysis where there is only a single sequence as input
  - Used as the main mechanism in Transformer-type models

# Transformers

## “Attention is All You Need”

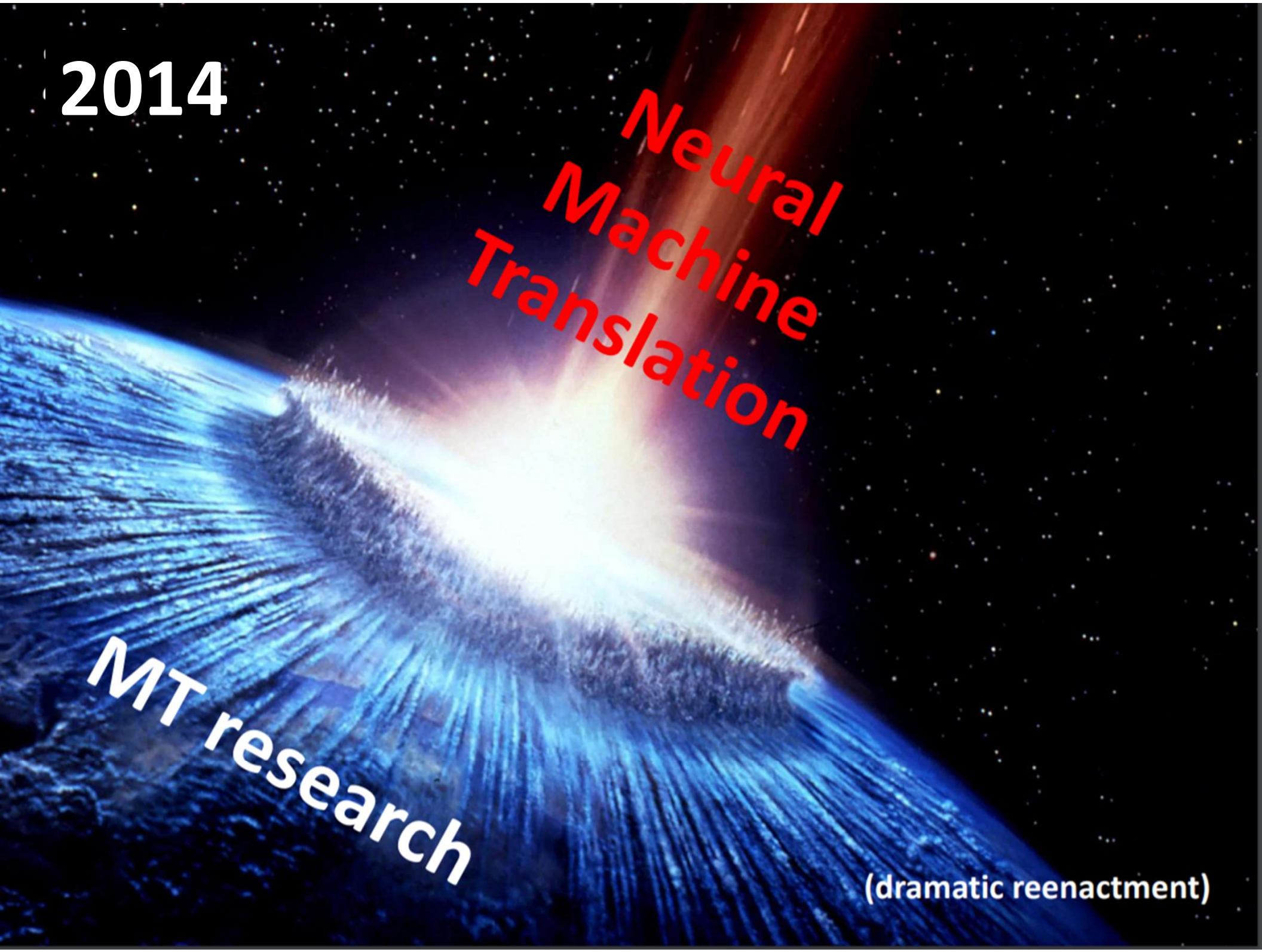
- Introduction
- Self-Attention Mechanism
- Model Architecture

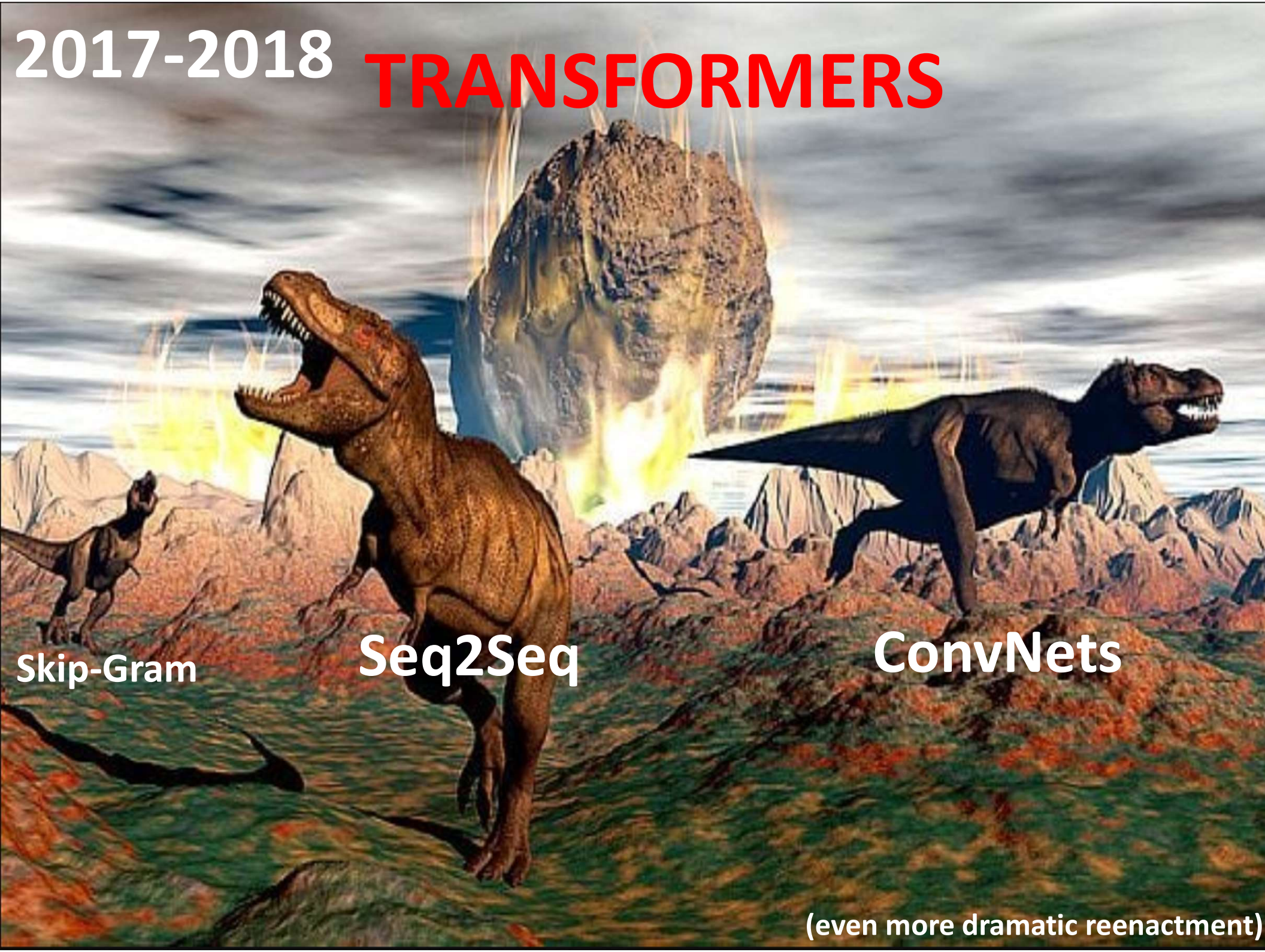
# Transformers

## “Attention is All You Need”: Introduction

# Pre-2014

(dramatic reenactment)


2014
Neural
Machine
Translation
MT research
(dramatic reenactment)


2017-2018
TRANSFORMERS
Skip-Gram
Seq2Seq
ConvNets
(even more dramatic reenactment)

# Transformers

## “Attention is All You Need”: Introduction

- Based solely on attention mechanisms, no recurrence or convolutions
  - Use positional encoding to maintain order of words in sequence
- Parallelizable, significantly less training time (~hrs – ~days vs. ~weeks)
- Uses attention in 3 different ways: 1) Encoder-Decoder, 2) Encoder Self-Attention, 3) Masked Decoder Self-Attention
- Dependencies between different input positions reduced to constant number of operations instead of linear (ConvS2S) or logarithmic (ByteNet)

# Transformers

## "Attention is All You Need": Introduction

- Based solely on attention mechanisms, no recurrence or convolutions
  - Use positional encoding to maintain order of words in sequence
- Parallelizable, significantly less training time (~hrs – ~days vs. ~weeks)
- Uses attention in 3 different ways: 1) Encoder-Decoder, 2) Encoder Self-Attention, 3) Masked Decoder Self-Attention
- Dependencies between different input positions reduced to constant number of operations instead of linear (ConvS2S) or logarithmic (ByteNet)

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. $n$ is the sequence length, $d$ is the representation dimension, $k$ is the kernel size of convolutions and $r$ the size of the neighborhood in restricted self-attention.

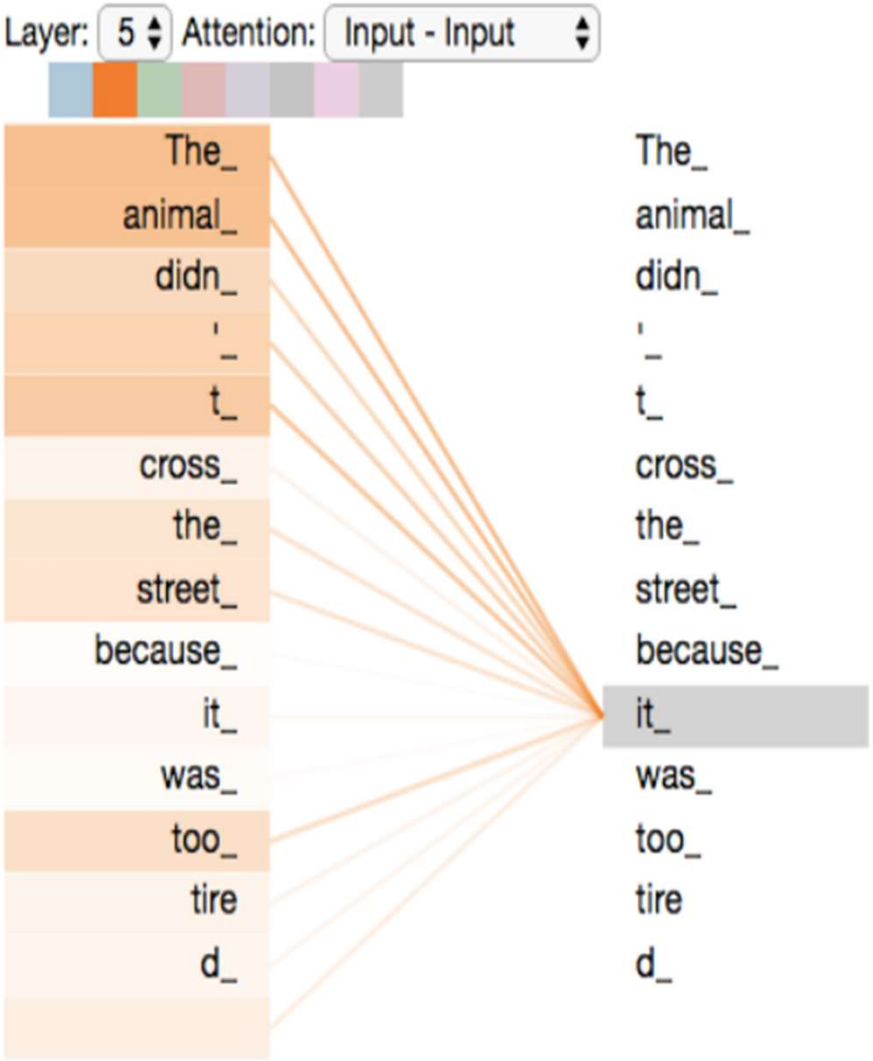
| Layer Type | Complexity per Layer | Sequential Operations | Maximum Path Length |
|---|---|---|---|
| Self-Attention | $O(n^2 \cdot d)$ | $O(1)$ | $O(1)$ |
| Recurrent | $O(n \cdot d^2)$ | $O(n)$ | $O(n)$ |
| Convolutional | $O(k \cdot n \cdot d^2)$ | $O(1)$ | $O(log_k(n))$ |
| Self-Attention (restricted) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

# Transformers

## “Attention is All You Need”: Self-Attention Mechanism

# Transformers

## "Attention is All You Need": Self-Attention Mechanism

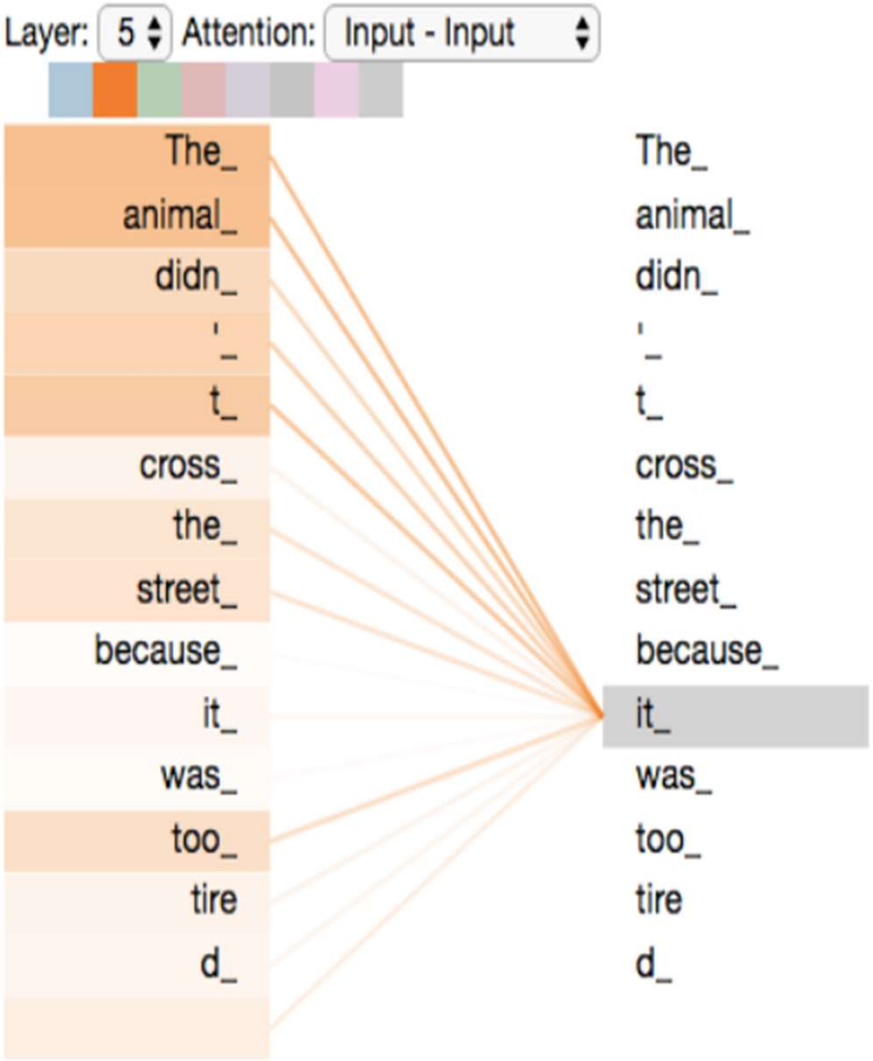



- Self-attention "teaches" the model to associate "it" with "The animal"
- These "associations" result in learned sentence-level embeddings → model learns sentence-level similarity
- Self-attention
  - Processes each word in the input one at a time (**query**)
  - By looking at all other words in the input sequence (**keys**)
  - For clues that can help the model learn a better encoding for the query (**values**)
- Why is it ***self***-attention?
  - There is only one sequence to look at → the encoder (decoder) sequence itself

"The animal didn't cross the street because it was too tired"

# Transformers

## "Attention is All You Need": Self-Attention Mechanism




- Each input word has its own **query**, **key**, and **value** vector
  - Vectors created by multiplying encoder (decoder) input by trainable weight matrices
- More details to follow

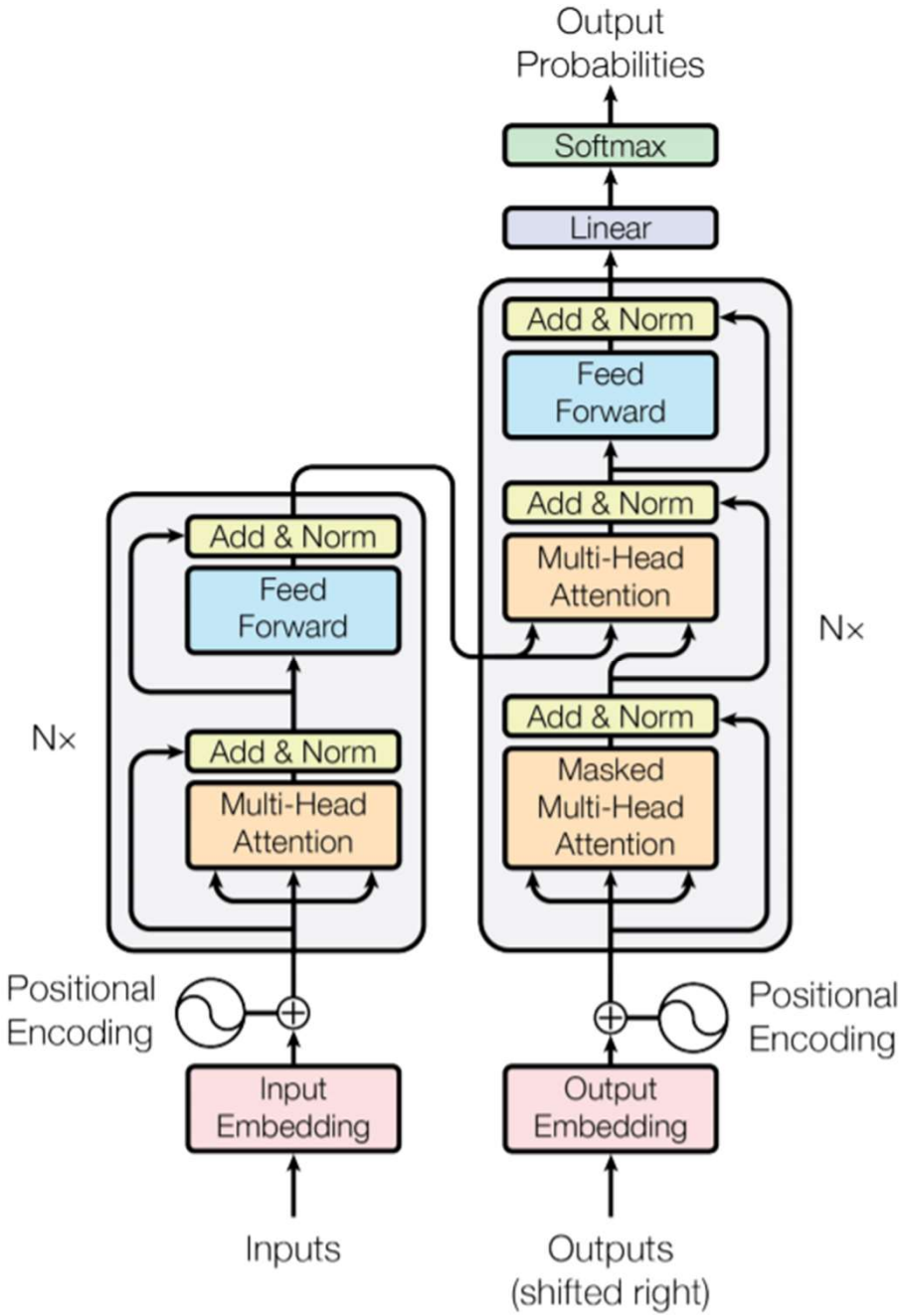
"The animal didn't cross the street because it was too tired"

# Transformers

## “Attention is All You Need”: Model Architecture

# Transformers

## “Attention is All You Need”: Model Architecture

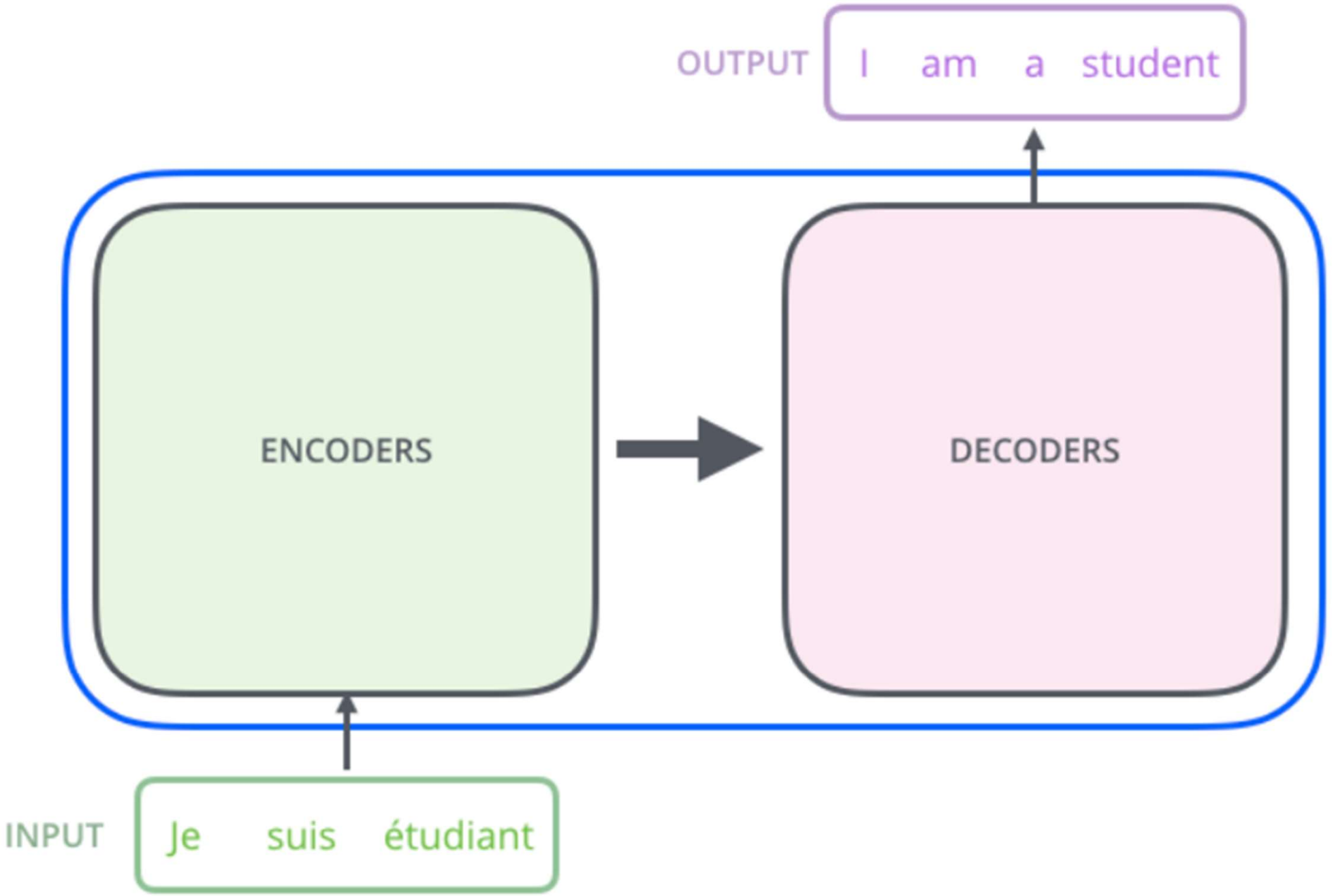


# Transformers

## “Attention is All You Need”: Model Architecture

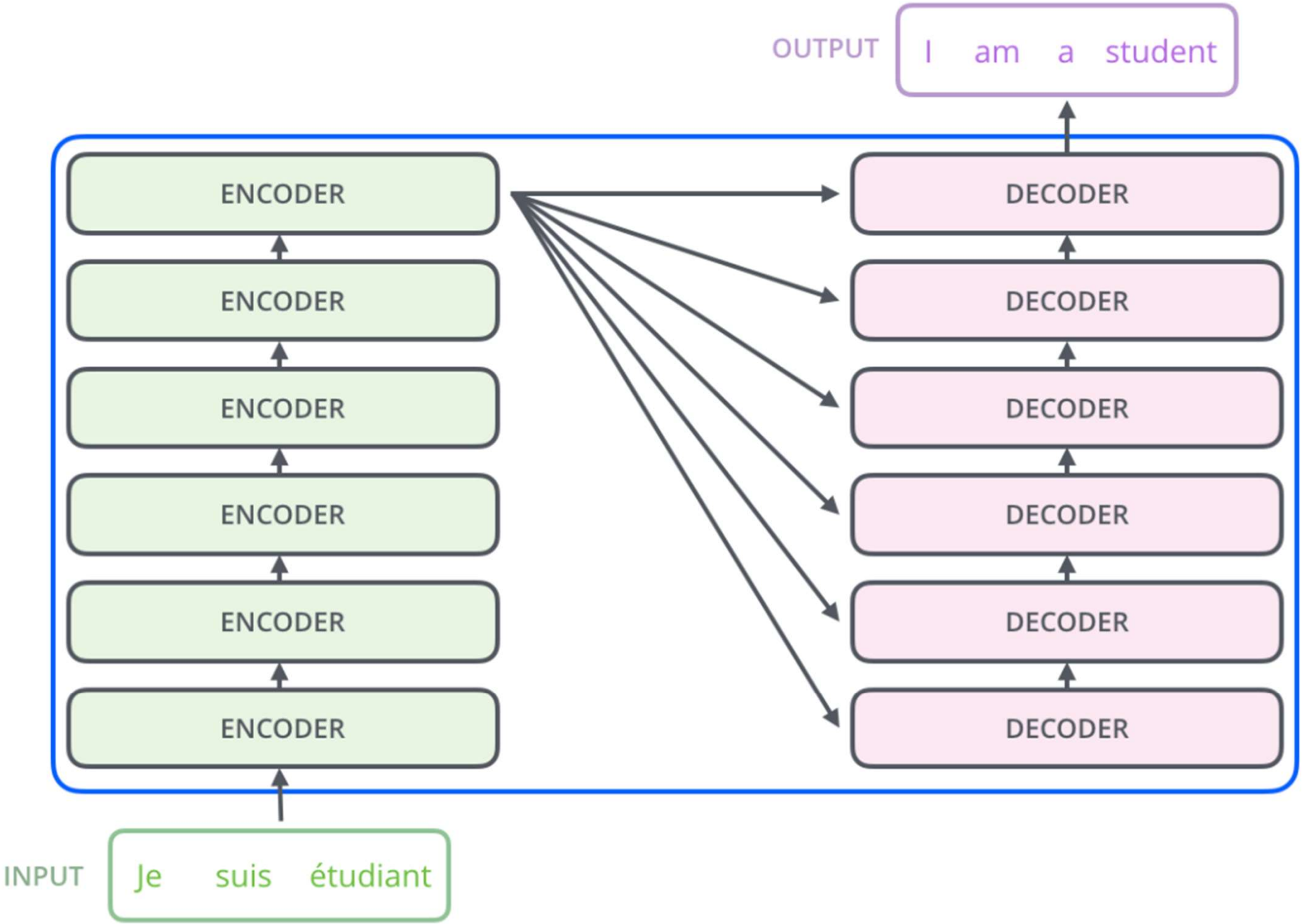
# Transformers

## “Attention is All You Need”: Model Architecture



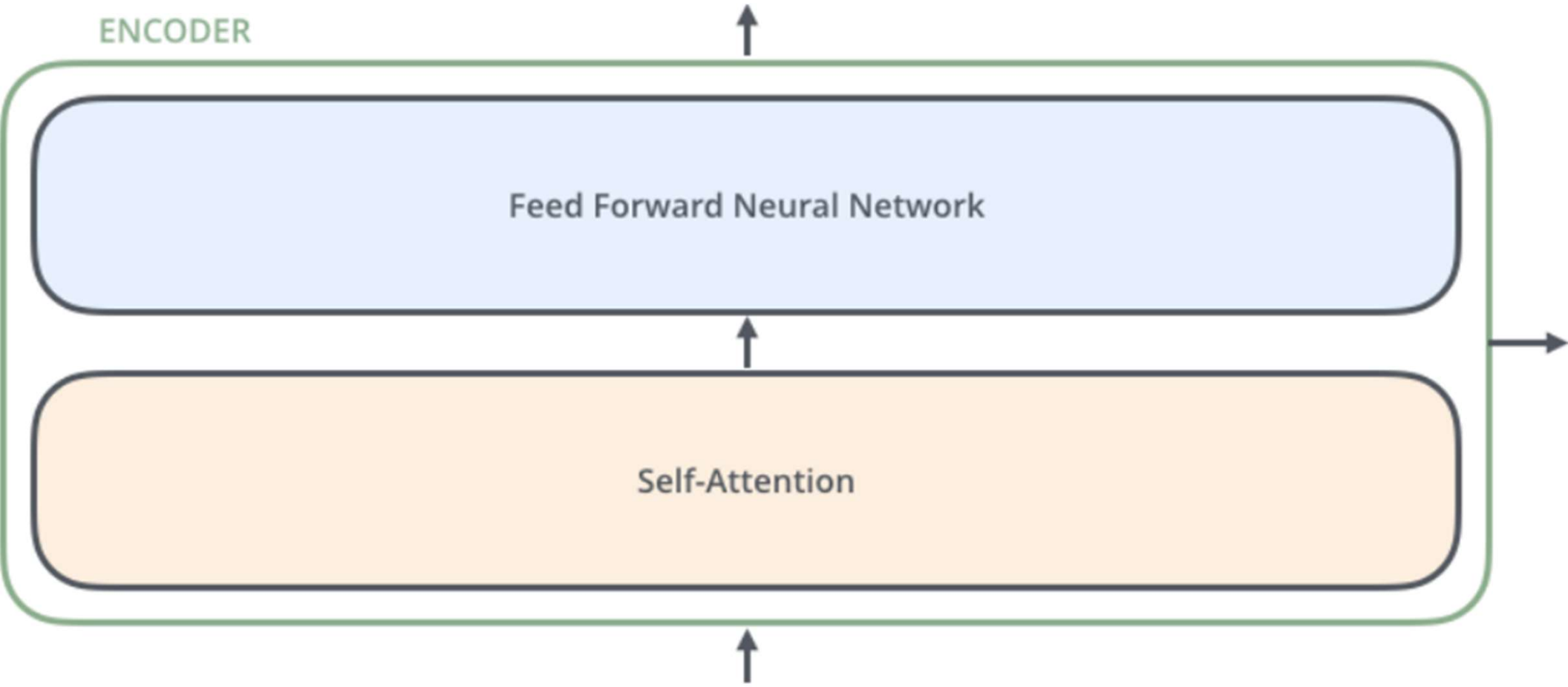
# Transformers

## “Attention is All You Need”: Model Architecture



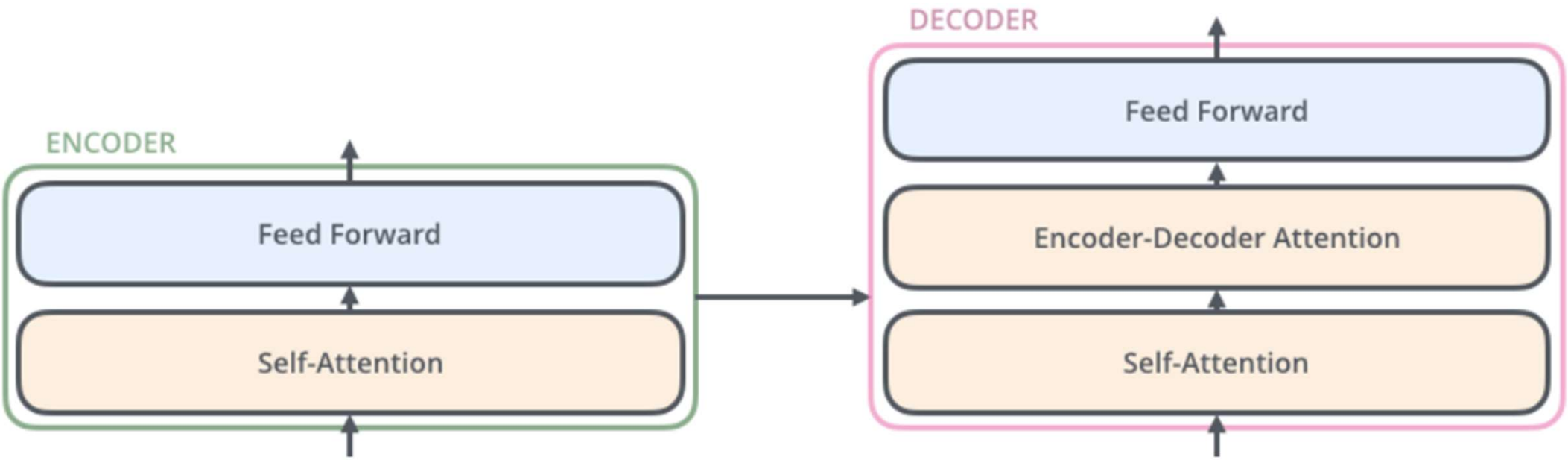
# Transformers

## “Attention is All You Need”: Model Architecture



# Transformers

## “Attention is All You Need”: Model Architecture

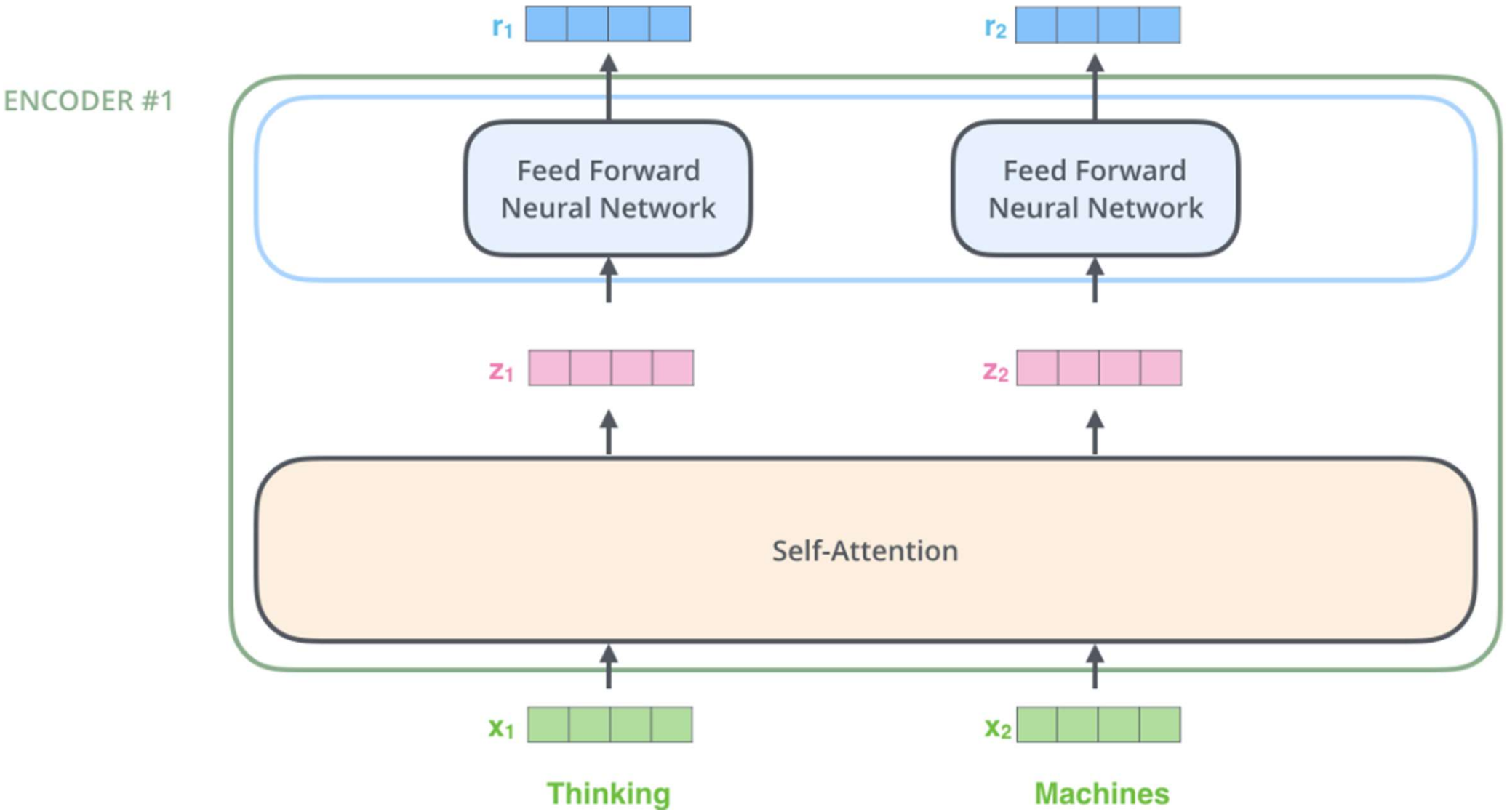


# Transformers

## “Attention is All You Need”: Model Architecture

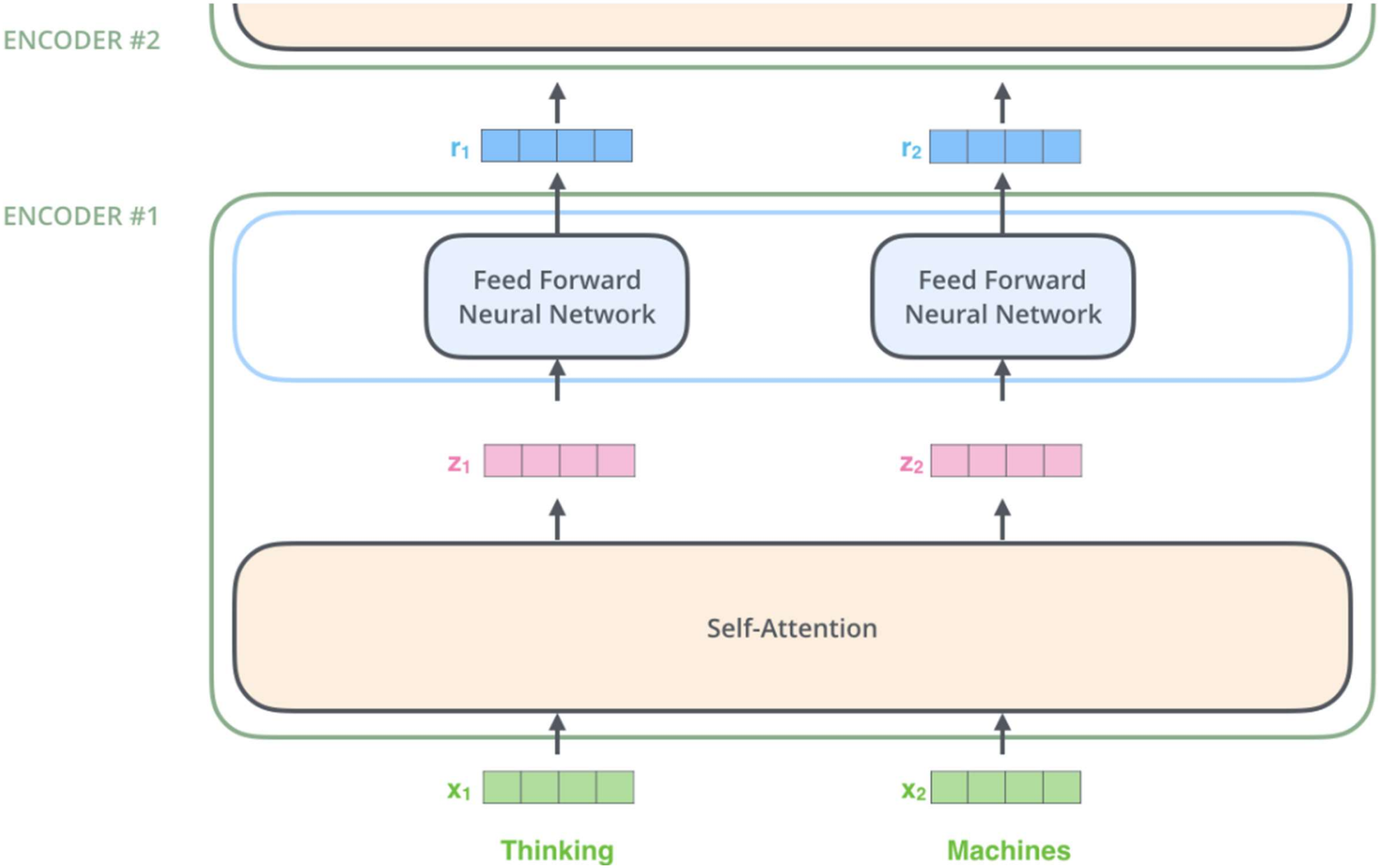
# Transformers

## “Attention is All You Need”: Model Architecture



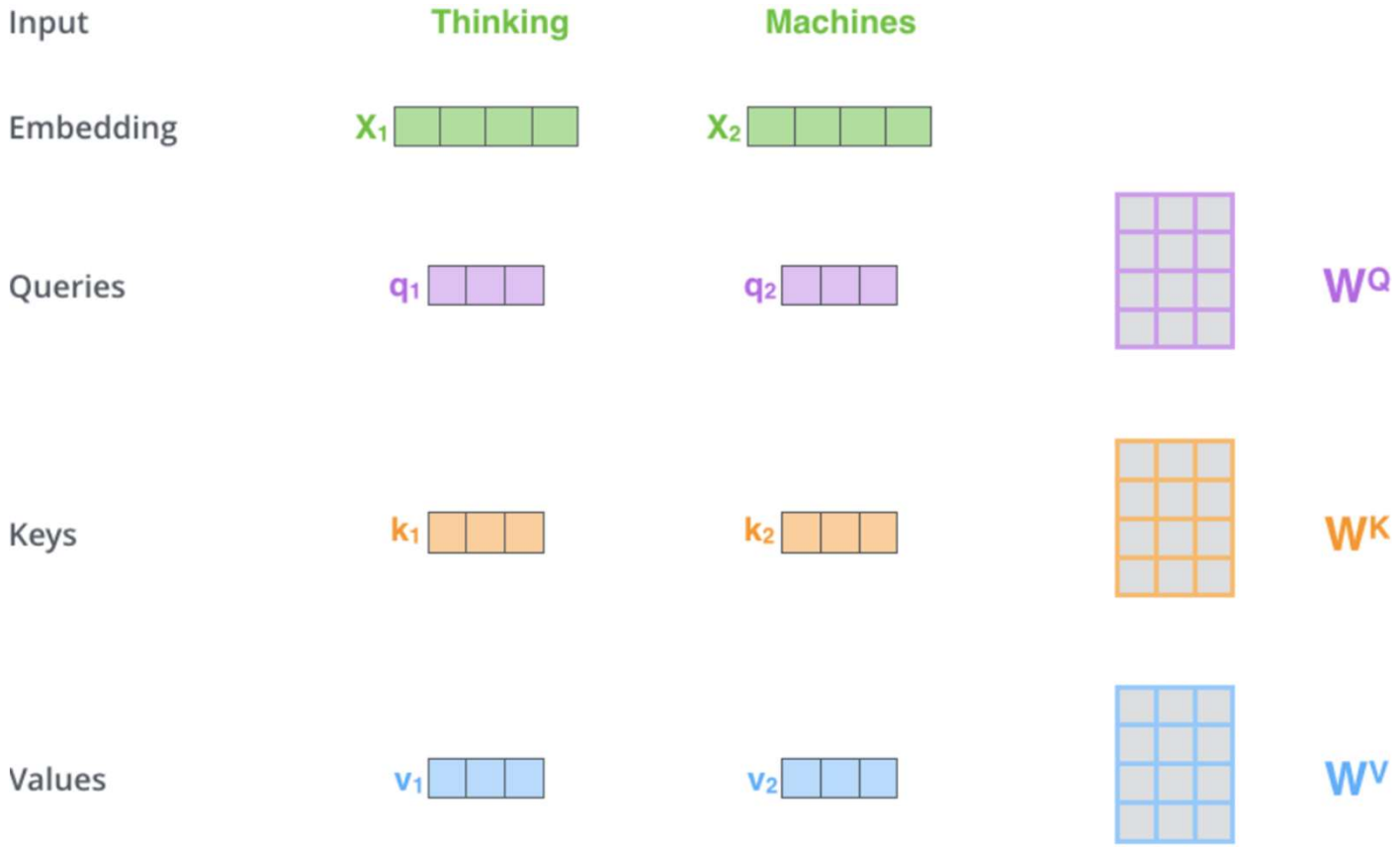
# Transformers

## “Attention is All You Need”: Model Architecture



# Transformers

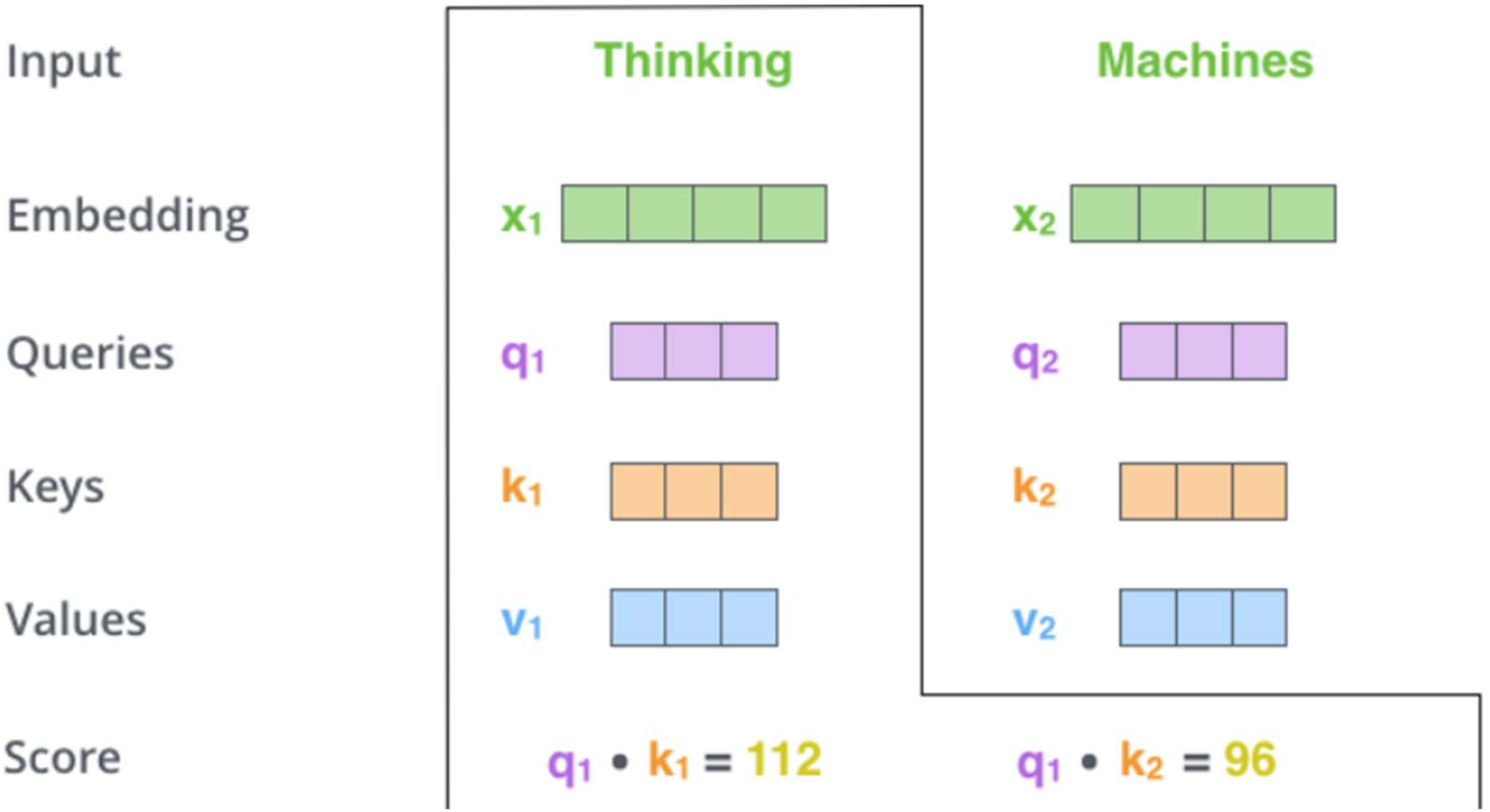
## “Attention is All You Need”: Model Architecture




- Create a Query, Key, and Value vector for each input by multiplying with three weight matrices for each input

# Transformers

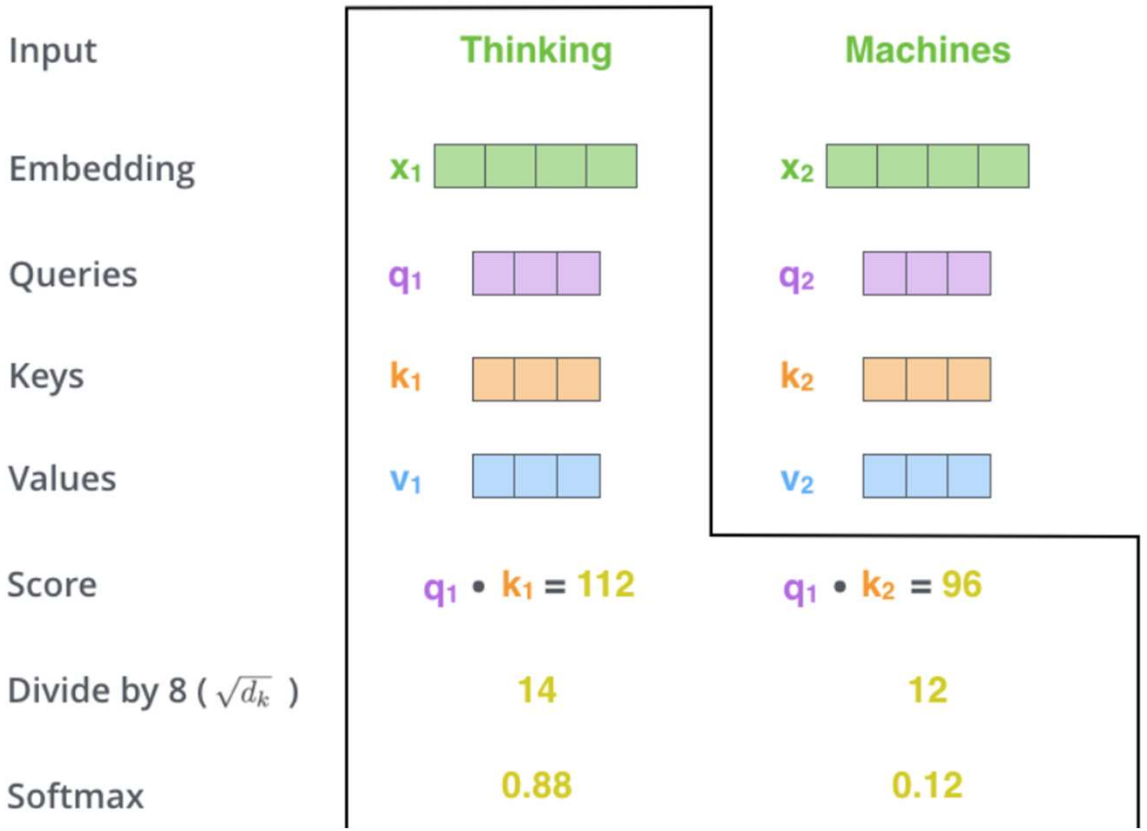
## “Attention is All You Need”: Model Architecture




- Calculate a score between each input word and all other words in input sequence → score determines how much attention to pay to other words as we encode the current input word
- Score = Query x Key

# Transformers

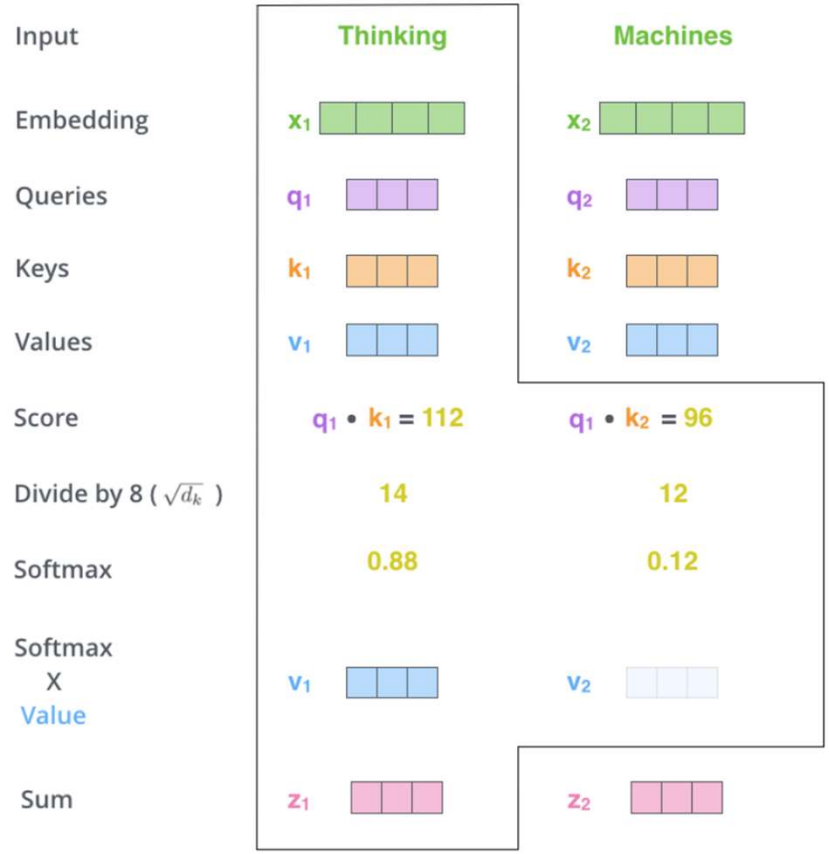
## "Attention is All You Need": Model Architecture




- Divide by sqrt of length of Q/K/V vector (i.e., 8) → helps with training (scaled Softmax)
- Perform Softmax across input sequence for each input → Form a distribution of attention probabilities for each input word over all words in the sequence
- Higher probabilities → Input word should "pay more attention to"

# Transformers

## “Attention is All You Need”: Model Architecture

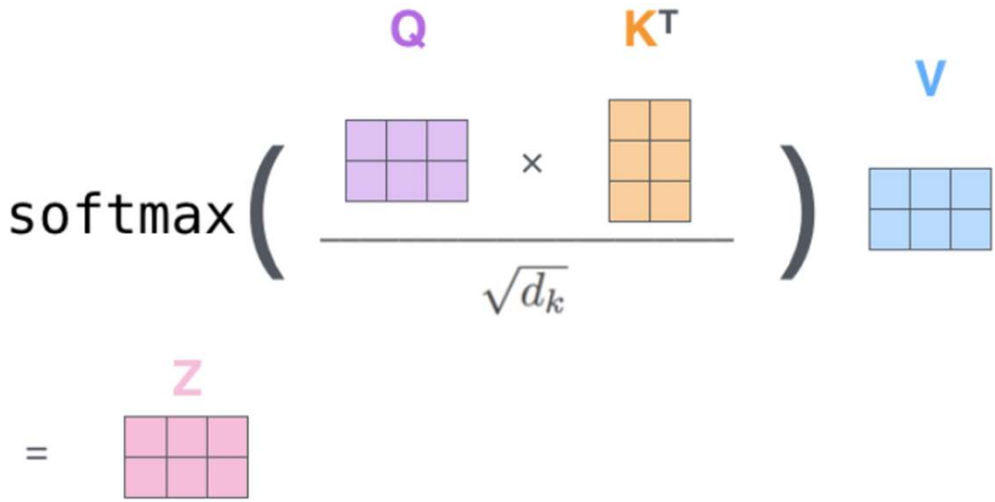



- Multiply each Value vector by the Softmax score
  - Words we want to pay attention to → multiplied by high Softmax scores
  - Irrelevant words → multiplied by low Softmax scores
- Sum up weighted Value vectors → output of self-attention layer → send to Feed Forward layer

# Transformers

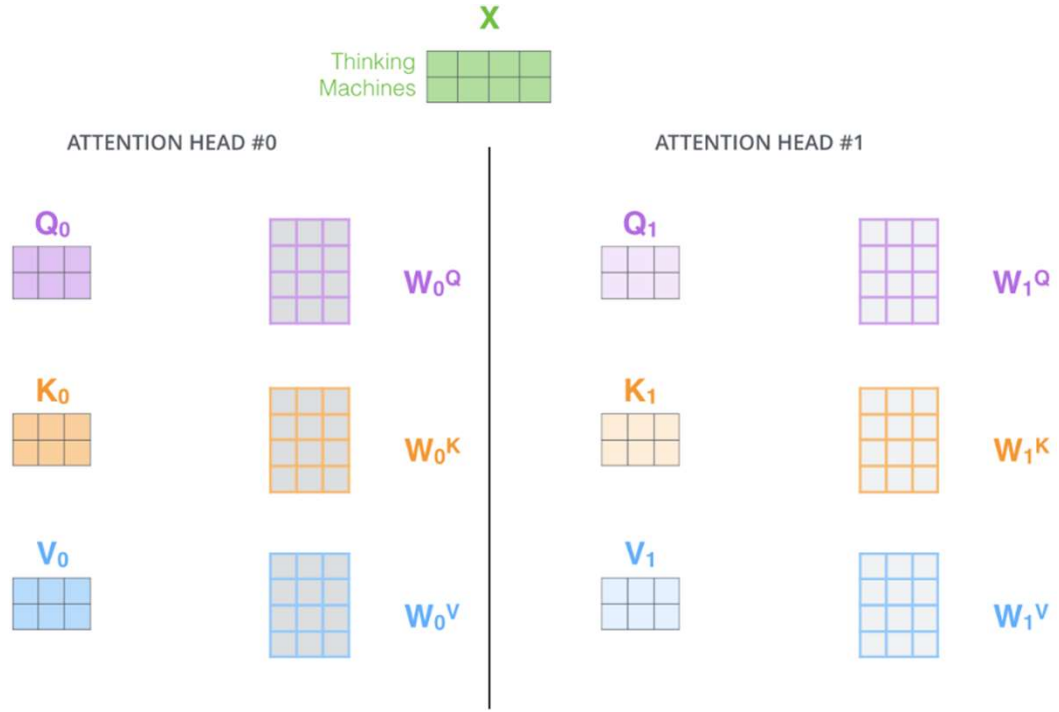
## “Attention is All You Need”: Model Architecture



# Transformers

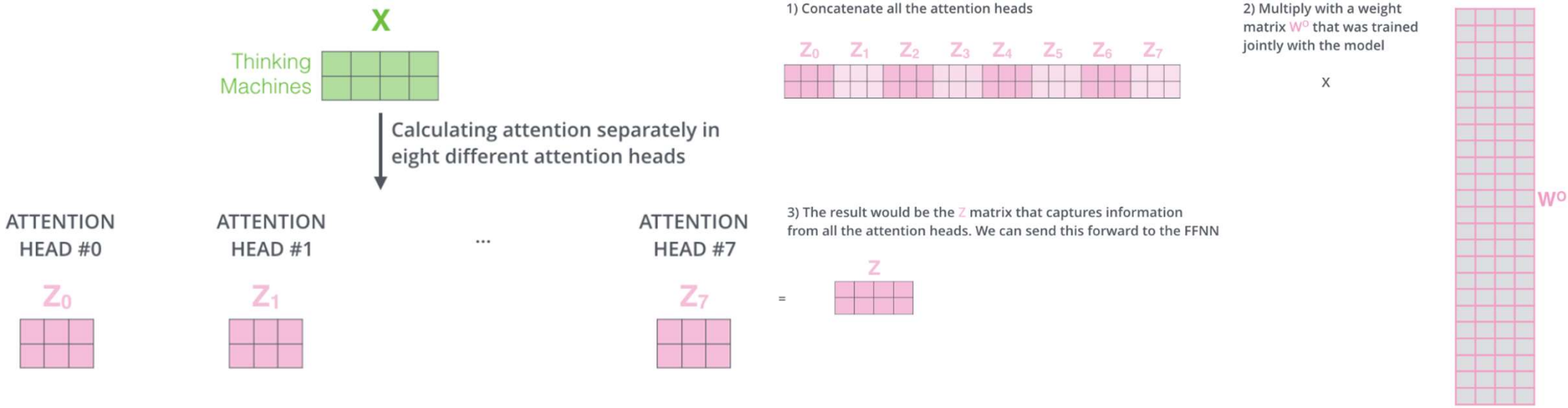
## "Attention is All You Need": Model Architecture




- In Transformers, a single attention head is "equivalent" to a single feature map channel from a CNN
  - Would like to have multiple "feature map channels" or attention heads
- Instead of a single set of Q/K/V matrices for each input, have multiple sets
  - Use 8 attention heads for each encoder (and decoder) self-attention layer
  - Each attention head learns a different representation of the query w.r.t. the input sequence

# Transformers

## “Attention is All You Need”: Model Architecture




- 8 attention heads → 8 *Z* matrices → However, Feed Forward layer expects single matrix
- Concatenate all 8 *Z* matrices → multiply by another matrix to get correct shape for Feed Forward layer
- Computation is constant → instead of 1 attention head with $Z_i$ as a 512-D vector, we have 8 attentions heads with each $Z_i$ as a 64-D vector

# Transformers

## “Attention is All You Need”: Model Architecture

1) This is our input sentence*

Thinking Machines

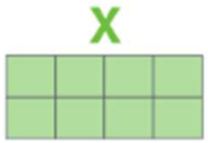
# Transformers

## “Attention is All You Need”: Model Architecture

1) This is our input sentence*

2) We embed each word*

Thinking Machines




* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one

# Transformers

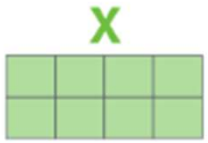
## “Attention is All You Need”: Model Architecture

1) This is our input sentence*

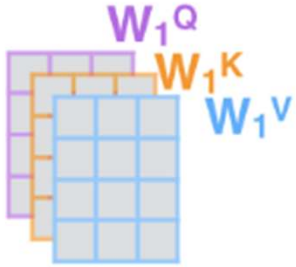
2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

Thinking Machines

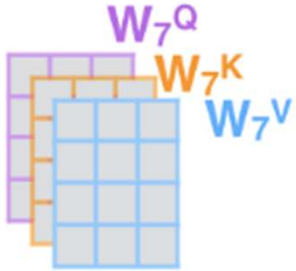







* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one


...



# Transformers

## “Attention is All You Need”: Model Architecture

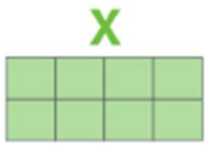
1) This is our input sentence*

2) We embed each word*

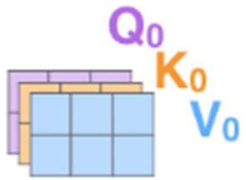
3) Split into 8 heads. We multiply X or R with weight matrices

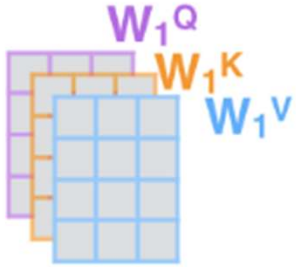
4) Calculate attention using the resulting Q/K/V matrices

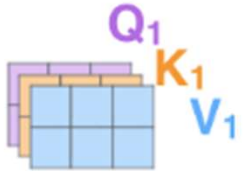
Thinking Machines





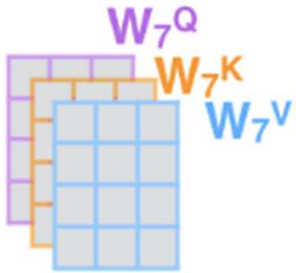


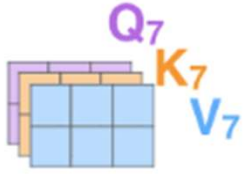




* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one


...




...



# Transformers

## “Attention is All You Need”: Model Architecture

1) This is our input sentence*

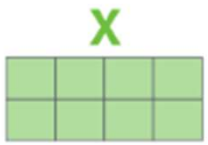
2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

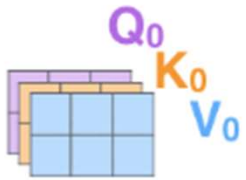
4) Calculate attention using the resulting Q/K/V matrices

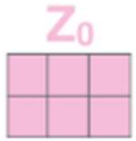
5) Concatenate the resulting Z matrices, then multiply with weight matrix $W^O$ to produce the output of the layer

Thinking Machines

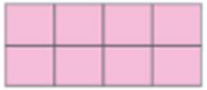


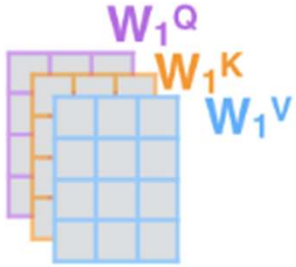


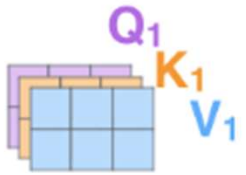


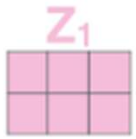





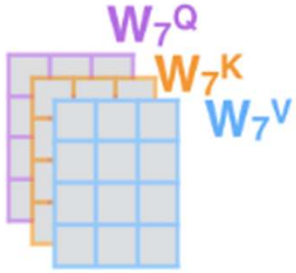


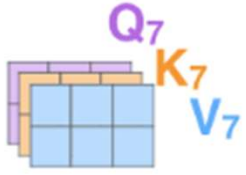



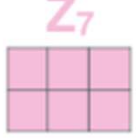




* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one

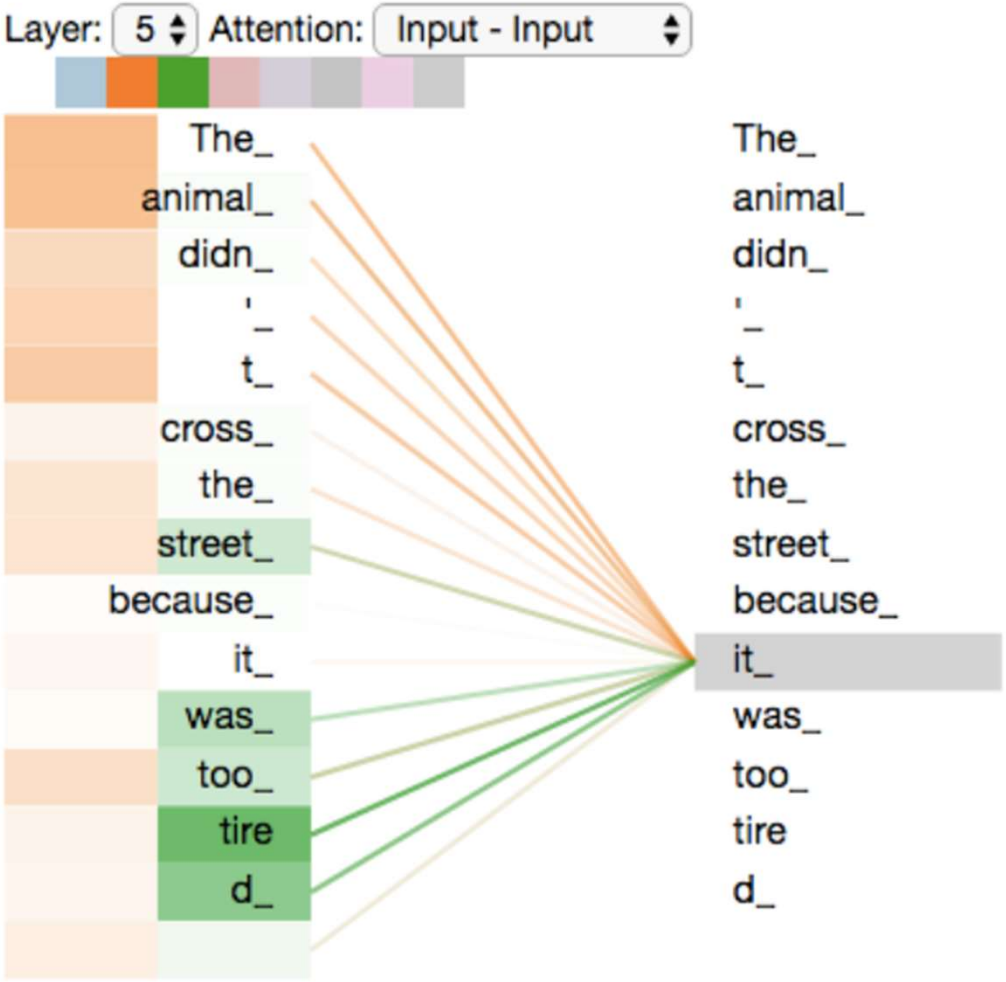



# Transformers

## “Attention is All You Need”: Model Architecture




- Encoding for “it”
  - **Orange** attention head pays attention to “The animal”
  - **Green** attention head pays attention to “too tired”
- Representation for “it” pays attention to “animal” and “tired”

# Transformers

## "Attention is All You Need": Model Architecture

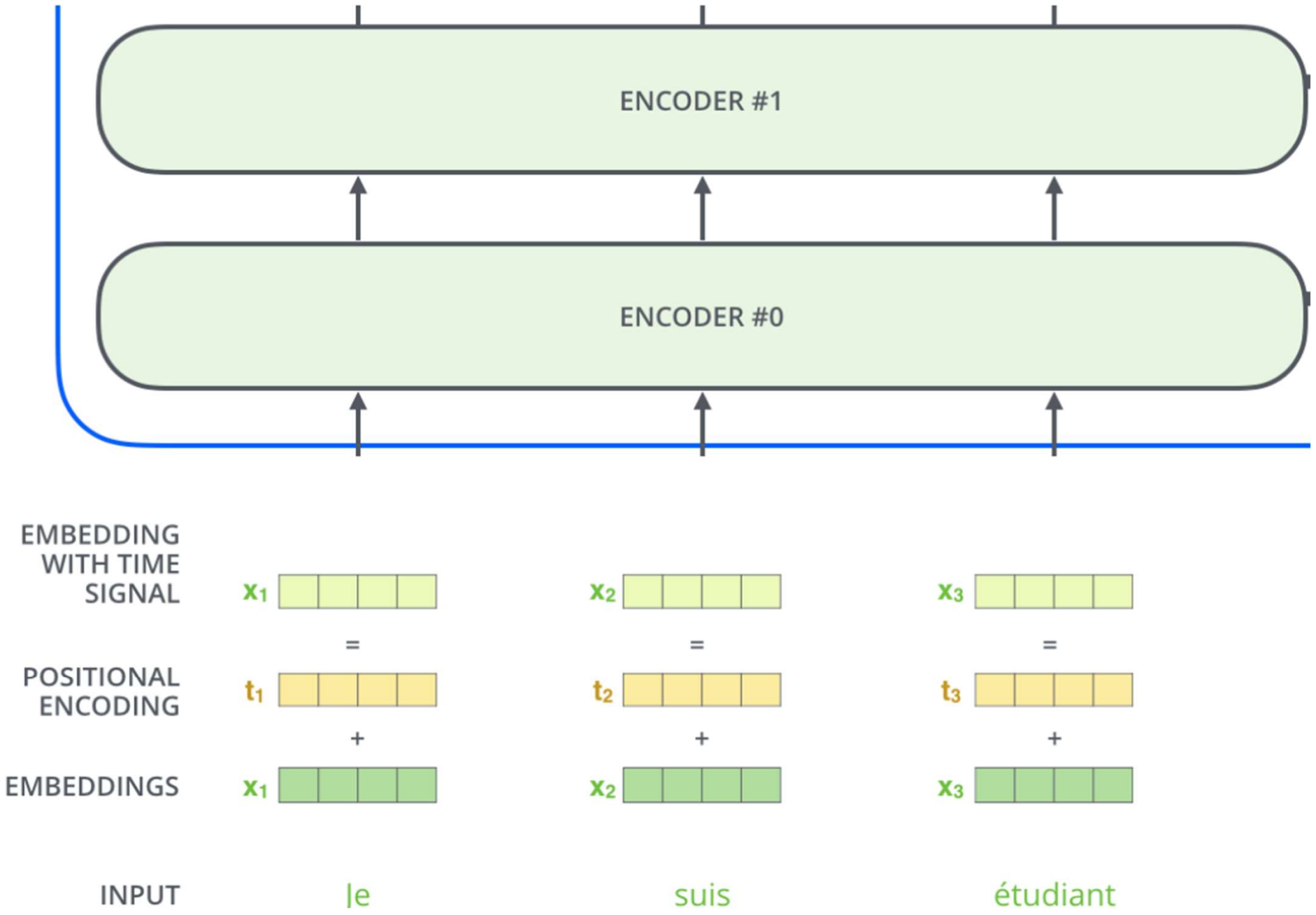
- LMs must have a notion of the *order* of words in a sequence (otherwise, it's a just a Bag-of-Words model)
- Add positional encoding vectors to the word embeddings
  - Positional encoding follows a specific pattern for each unique input position

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

*pos*: Position in input sequence
*i*: Dimension of position vector

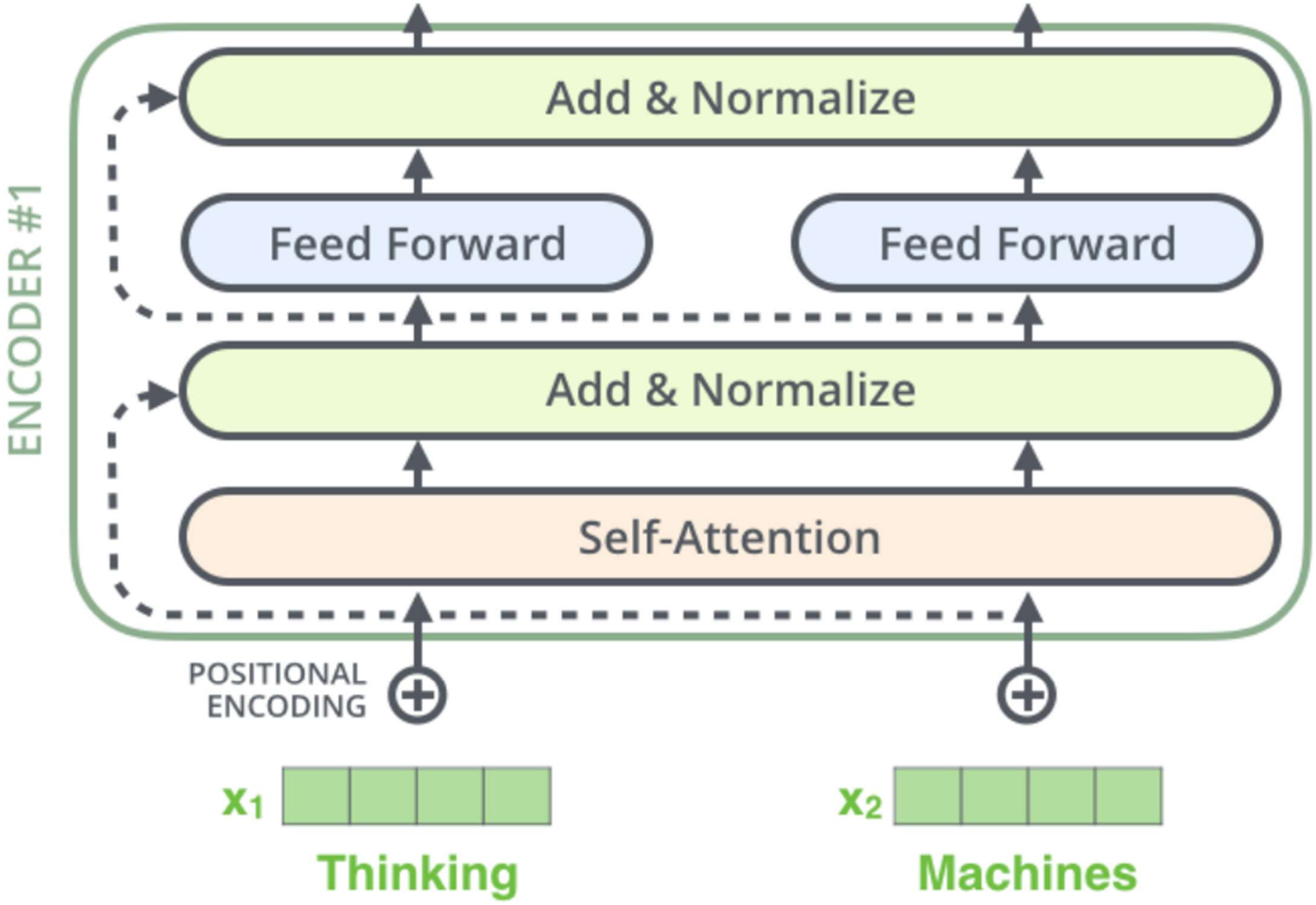
# Transformers

## “Attention is All You Need”: Model Architecture



# Transformers

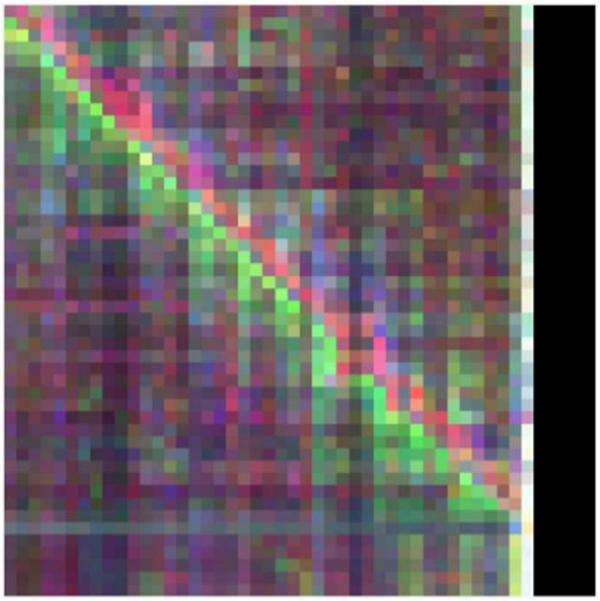
## “Attention is All You Need”: Model Architecture



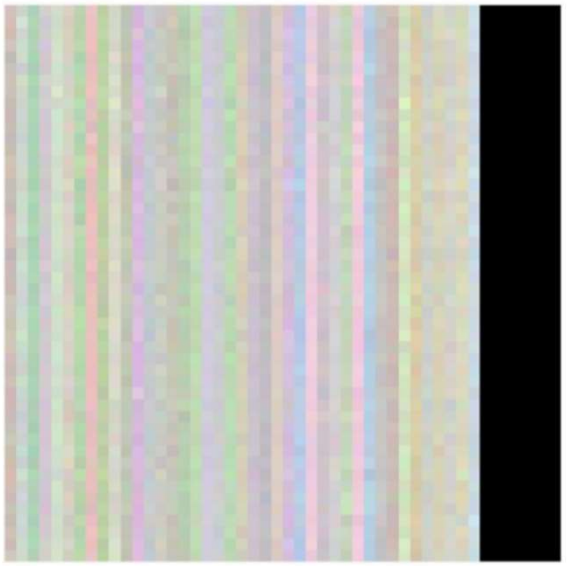
# Transformers

## “Attention is All You Need”: Model Architecture

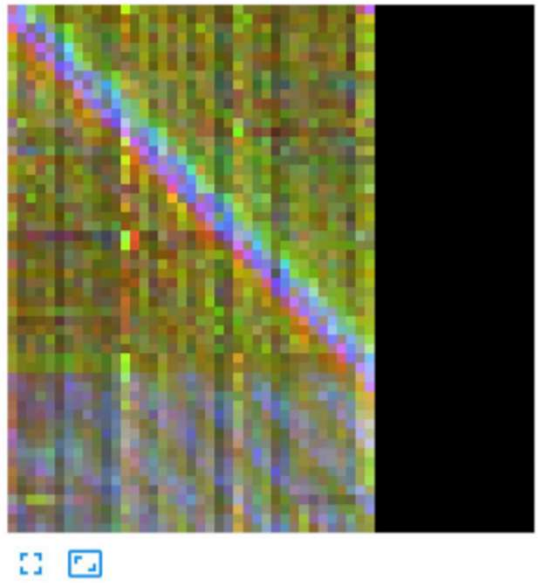
Residuals carry positional information to higher layers, among other information.



With residuals



Without residuals

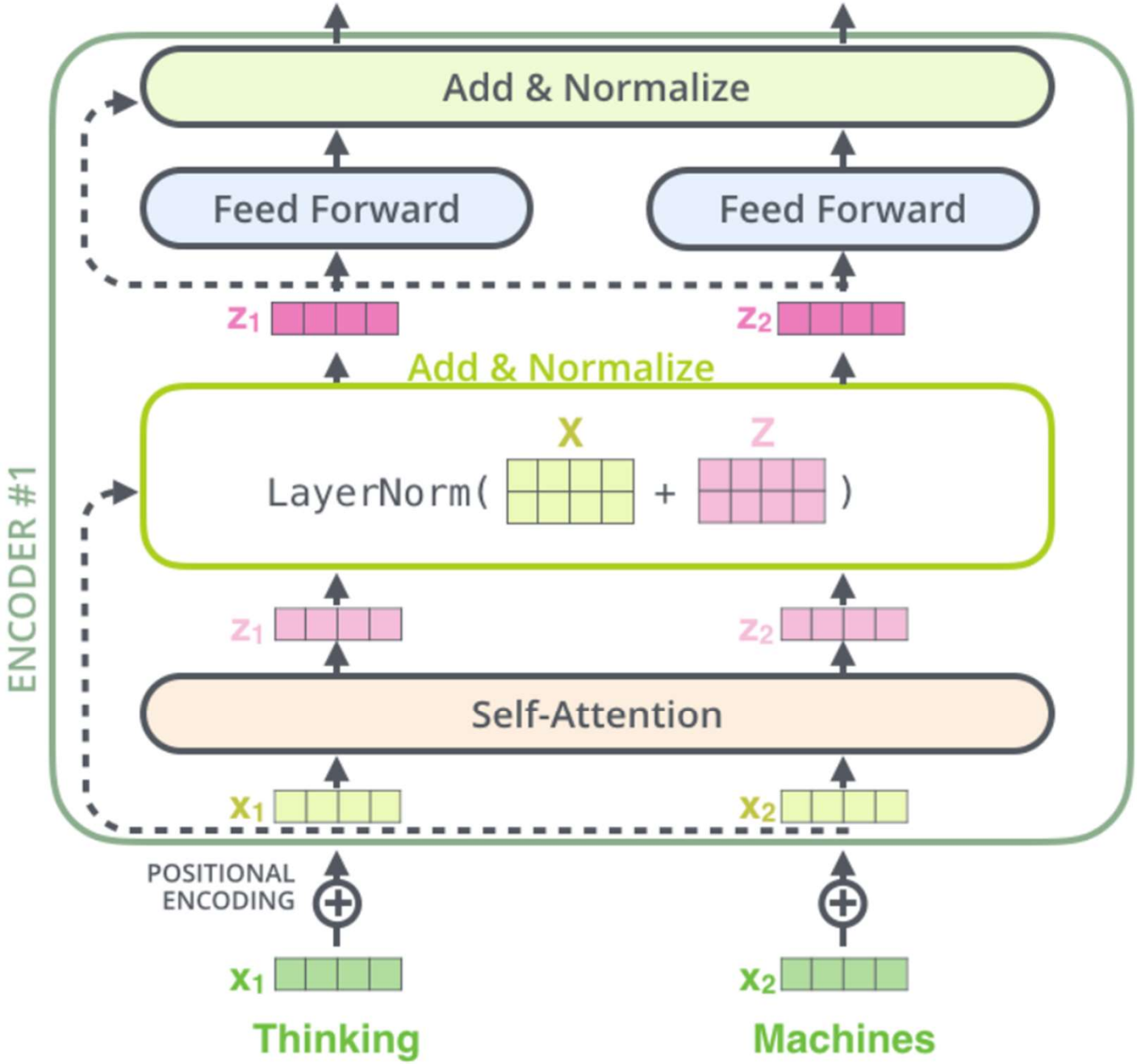


Without residuals, with timing signals

- Images are Encoder-Decoder mapping
- Without residuals, positional information has difficulty propagating to later layers in the model

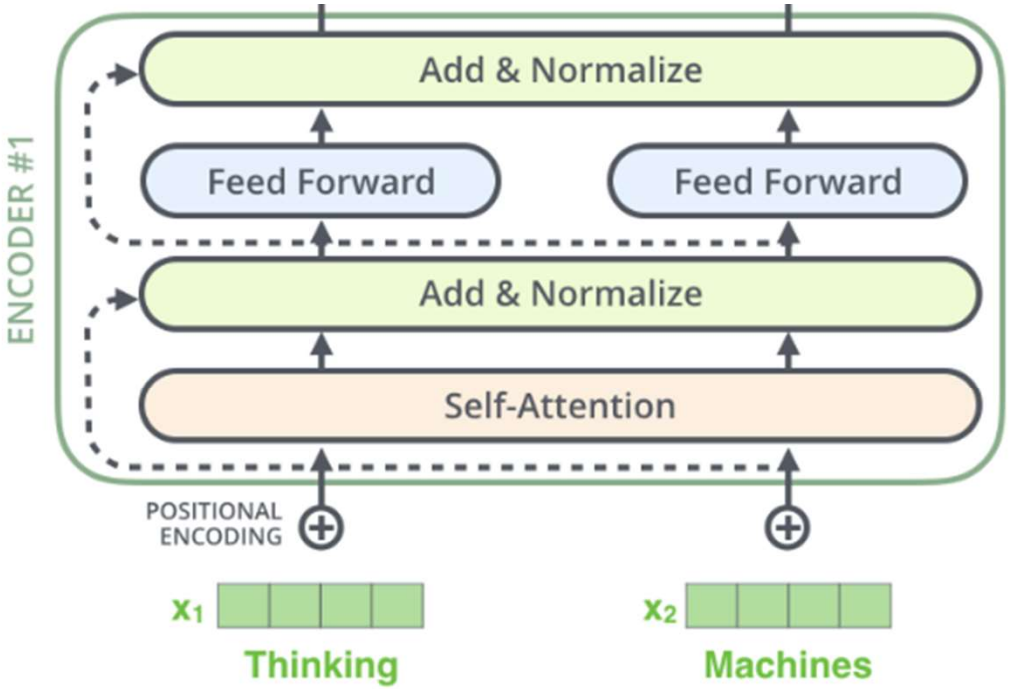
# Transformers

## “Attention is All You Need”: Model Architecture



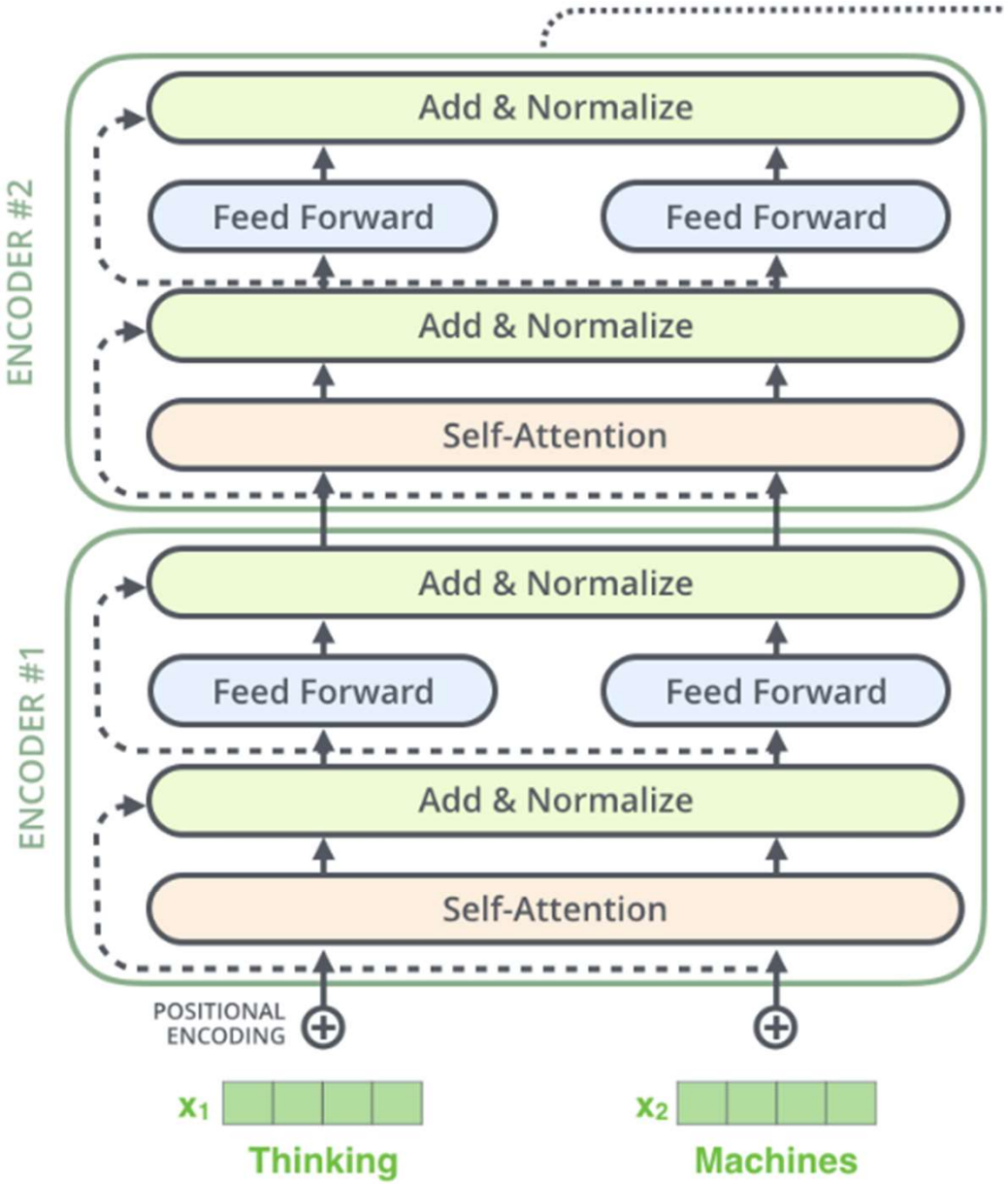
# Transformers

## “Attention is All You Need”: Model Architecture



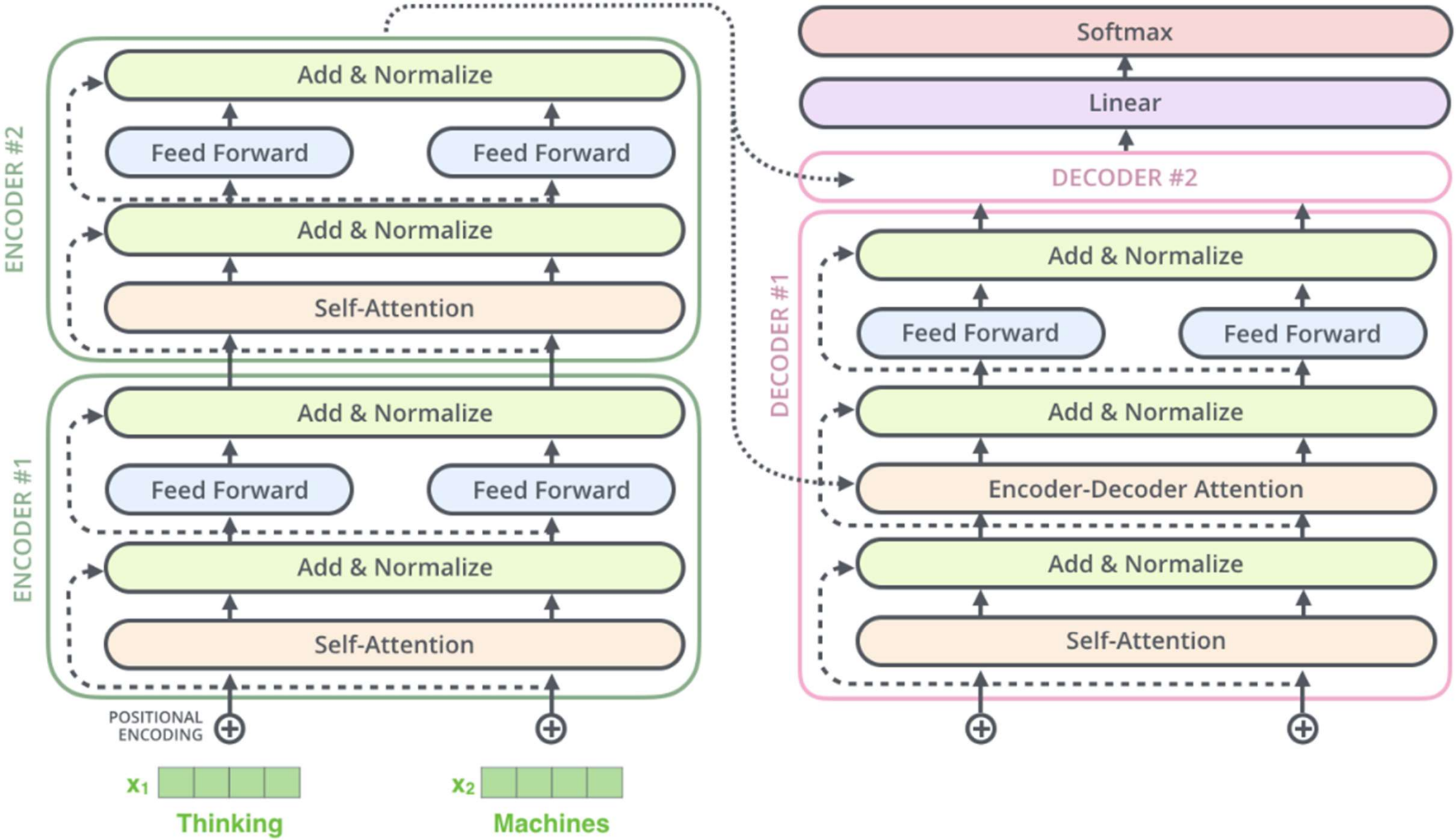
# Transformers

## “Attention is All You Need”: Model Architecture



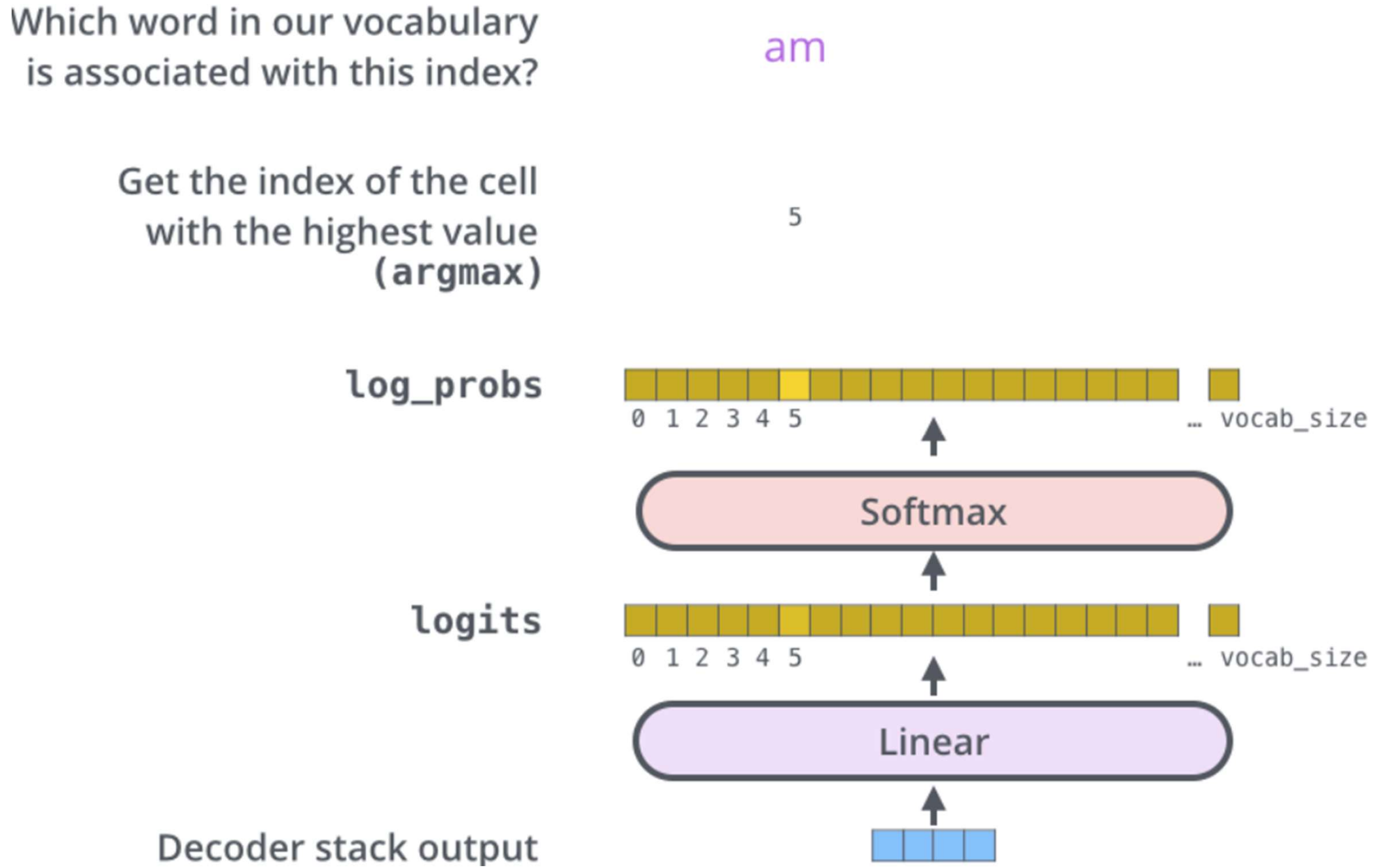
# Transformers

## “Attention is All You Need”: Model Architecture



# Transformers

## “Attention is All You Need”: Model Architecture



# Transformers

"BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

- Introduction
- Model Architecture
- Comparison with OpenAI GPT
- Pretraining BERT
- Fine-tuning BERT
- Feature-based BERT
- Pros and Cons

# Transformers

## BERT: Introduction

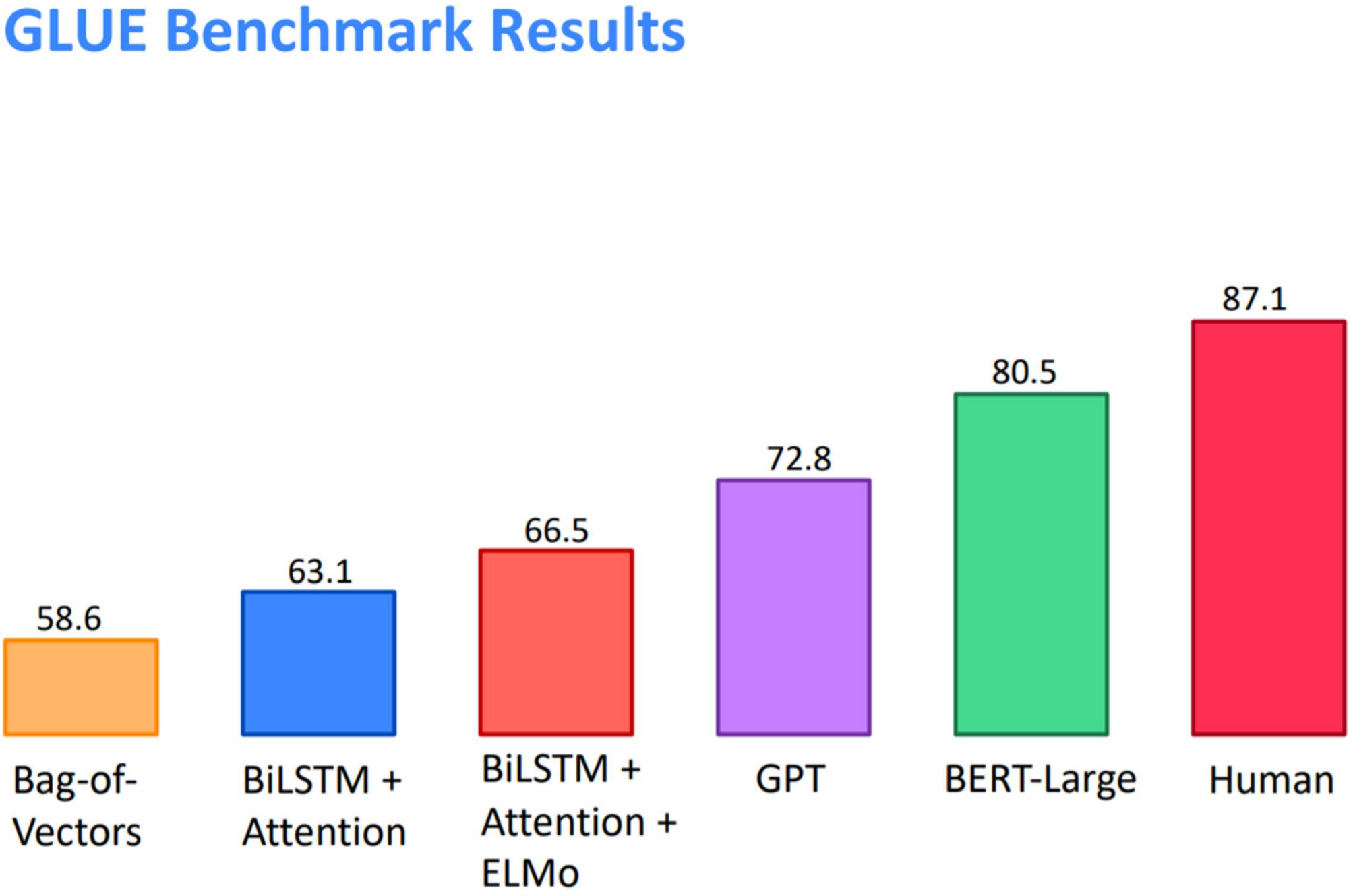
# Transformers

## BERT: Introduction

- BERT is the first fine-tuning based language representation model that achieves SOTA results on a large variety of sentence-level (NLI, PP, etc.) *and* token-level (NER, Q&A, etc.) tasks
  - Uses techniques from previous SOTA models → self-attention, bi-directional context, positional encoding, input autoencoding, etc.
- "BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers."
  - Training objective uses both left and right context → **bidirectional**
  - BERT is basically just the Transformer encoder component → **deep**
- Uses masked LM (MLM) pre-training objective for language modeling
  - Input tokens are randomly masked and objective is to predict the original vocabulary ID of the masked tokens based on the surrounding context
  - Inspired by Cloze task in psychology (Taylor, 1953)
- Also uses a "next sentence prediction" (NSP) objective to pre-train on sequence-pair representations

# Transformers

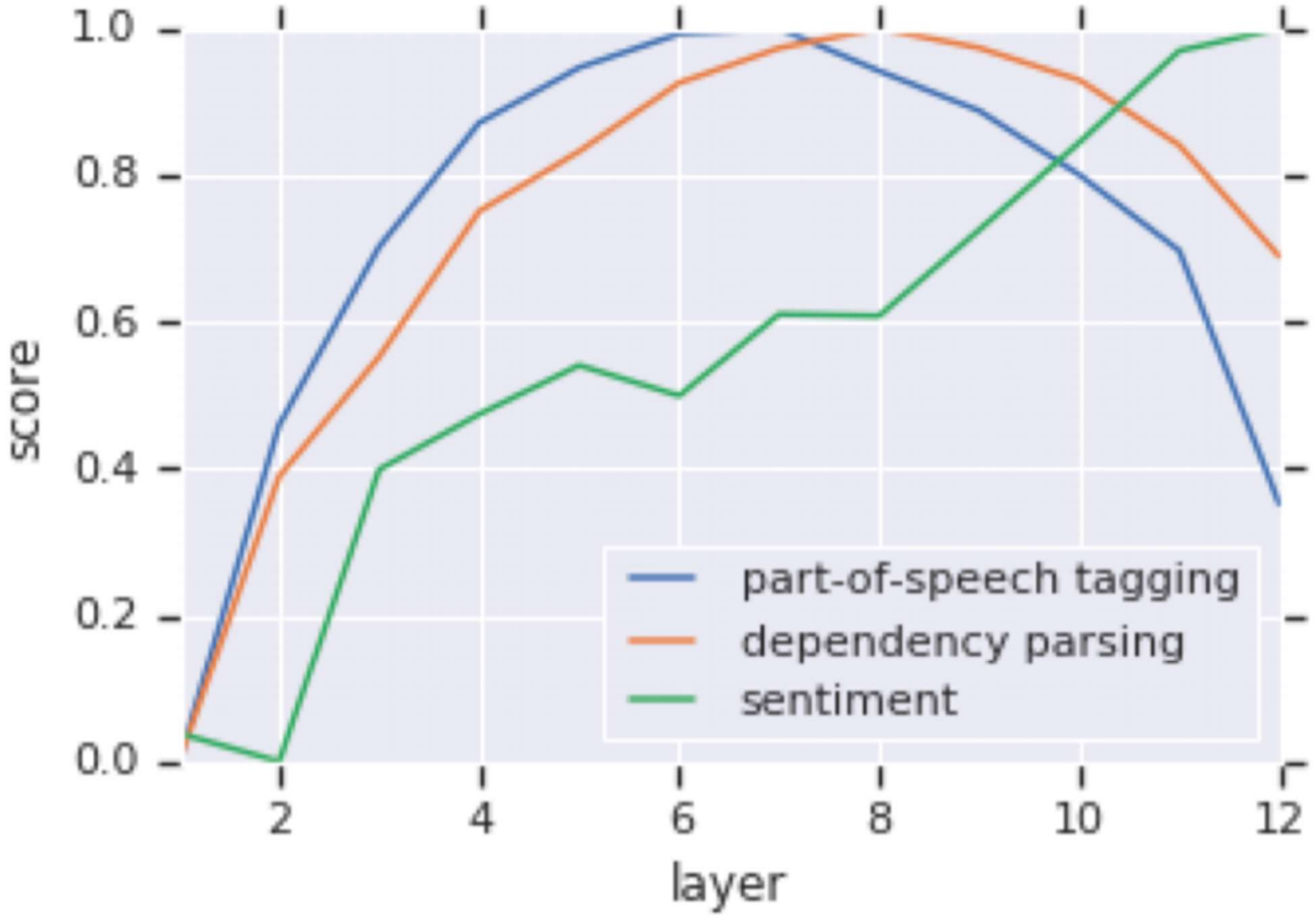
## BERT: Introduction



# Transformers

## BERT: Introduction

- Lower layers of BERT are better at lower-level tasks

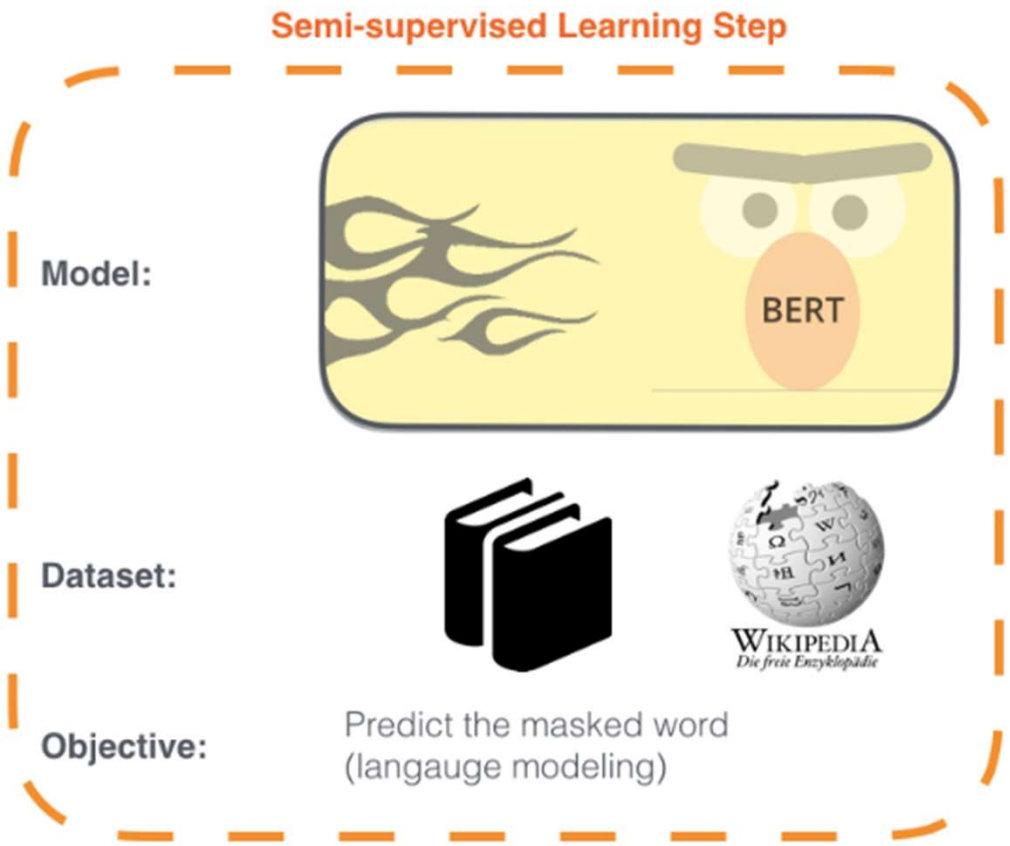


# Transformers

## BERT: Introduction

1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



# Transformers

## BERT: Introduction

1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

**Semi-supervised Learning Step**

**Model:**


**Dataset:**

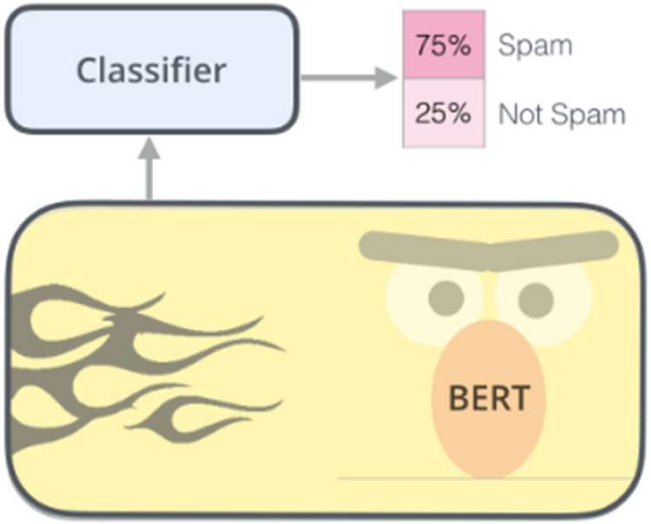

**Objective:**

Predict the masked word (langauge modeling)

2 - Supervised training on a specific task with a labeled dataset.

**Supervised Learning Step**

**Model:** (pre-trained in step #1)




**Dataset:**

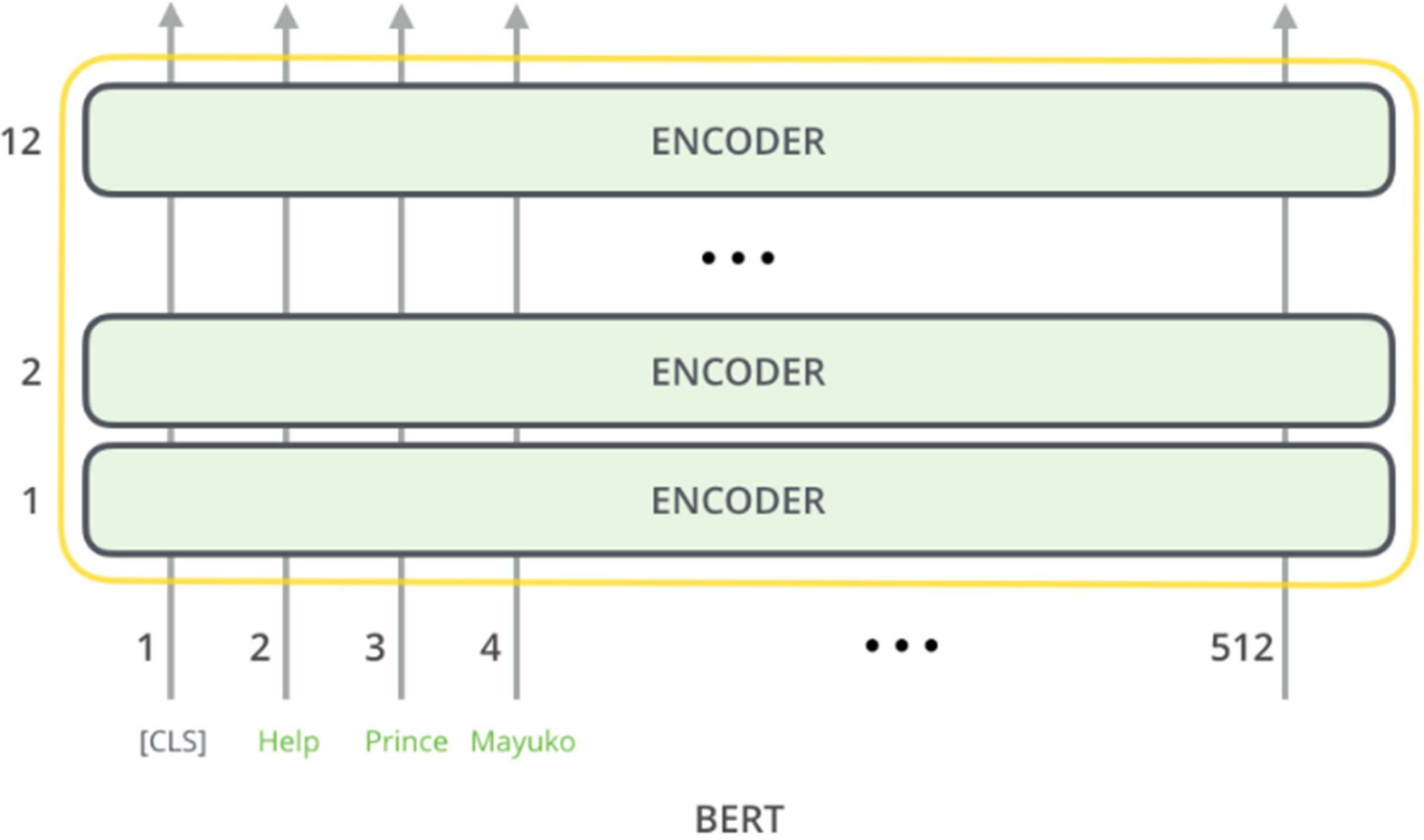
| Email message | Class |
|---|---|
| Buy these pills | Spam |
| Win cash prizes | Spam |
| Dear Mr. Atreides, please find attached… | Not Spam |

# Transformers

## BERT: Model Architecture

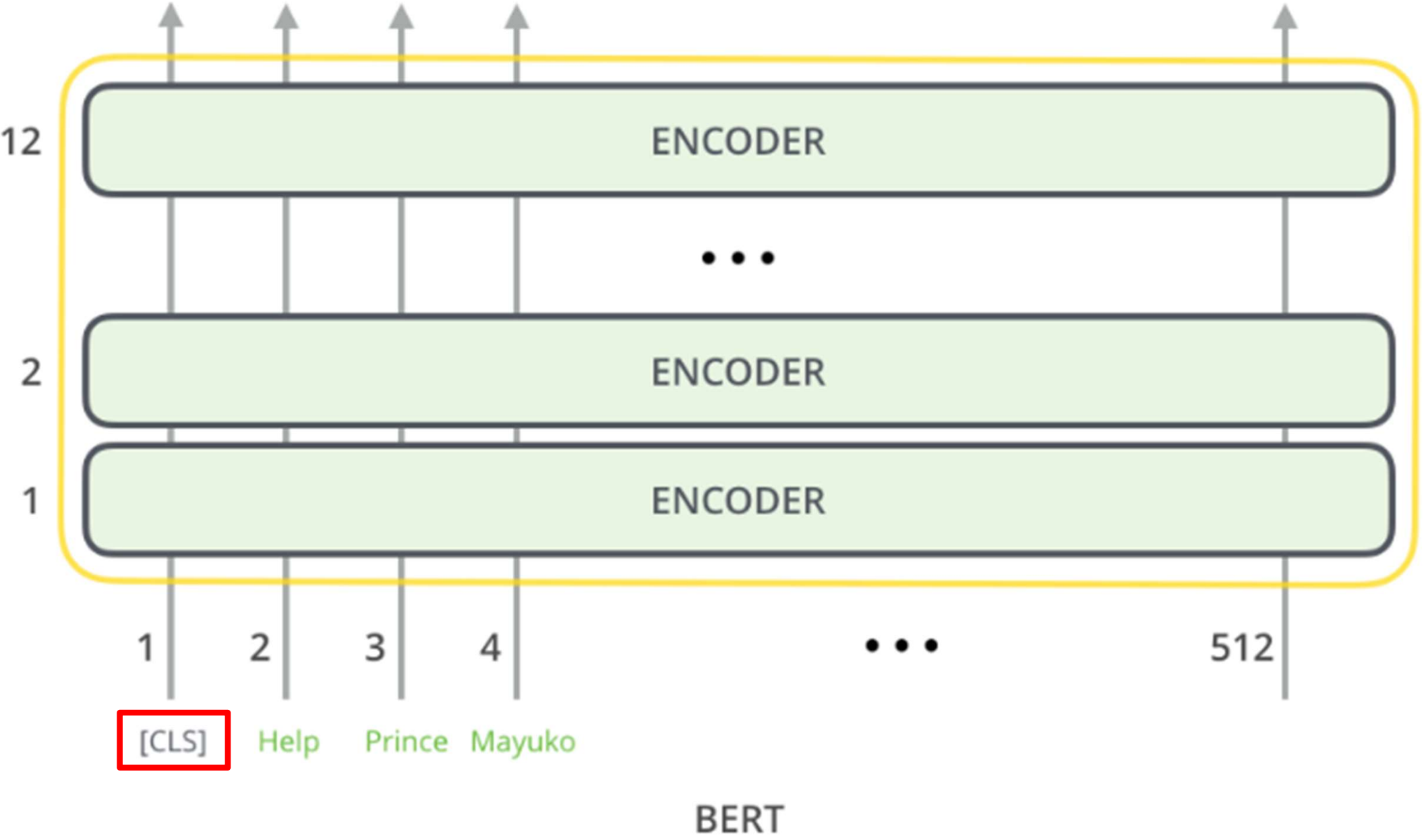
# Transformers

## BERT: Model Architecture



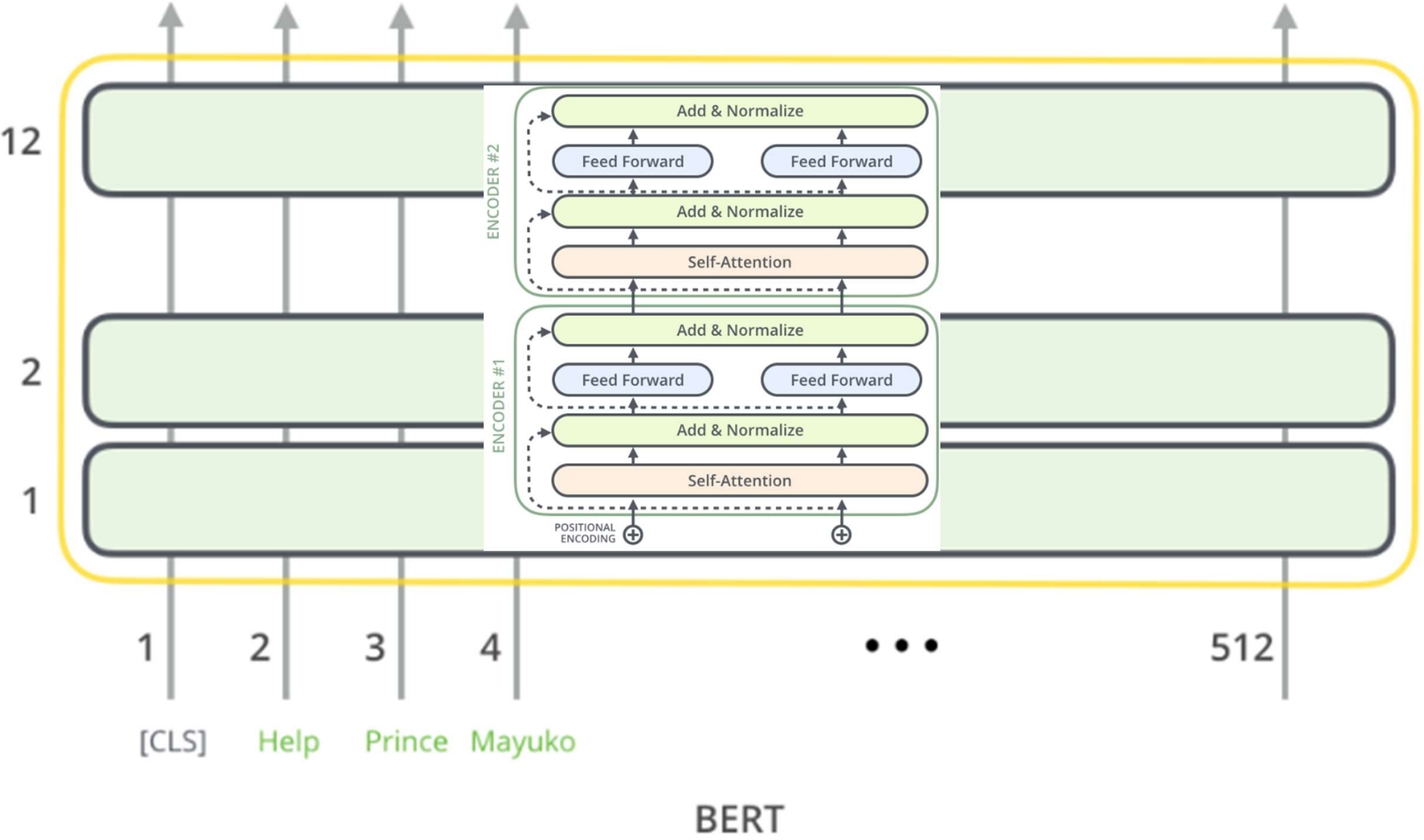
# Transformers

## BERT: Model Architecture



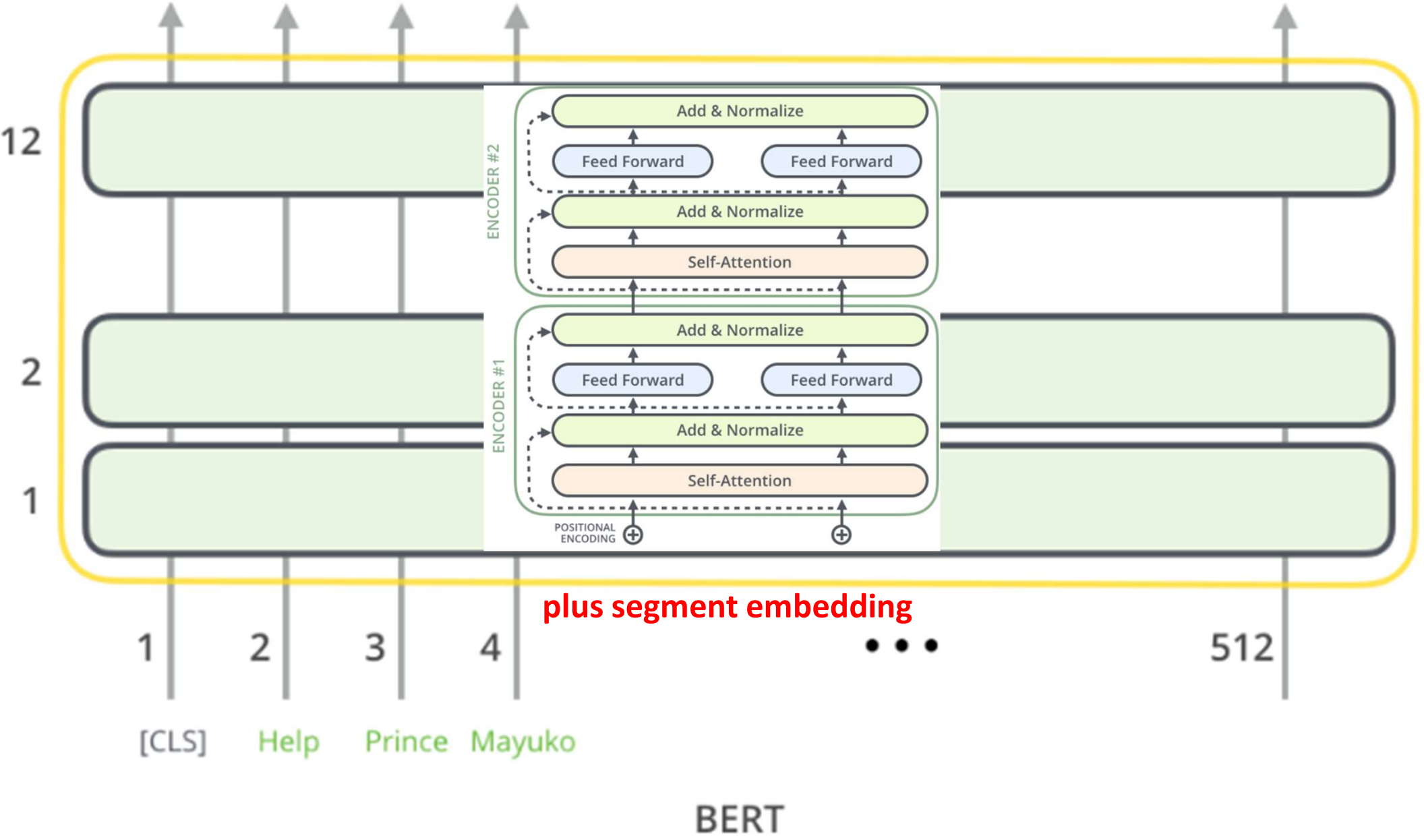
# Transformers

## BERT: Model Architecture



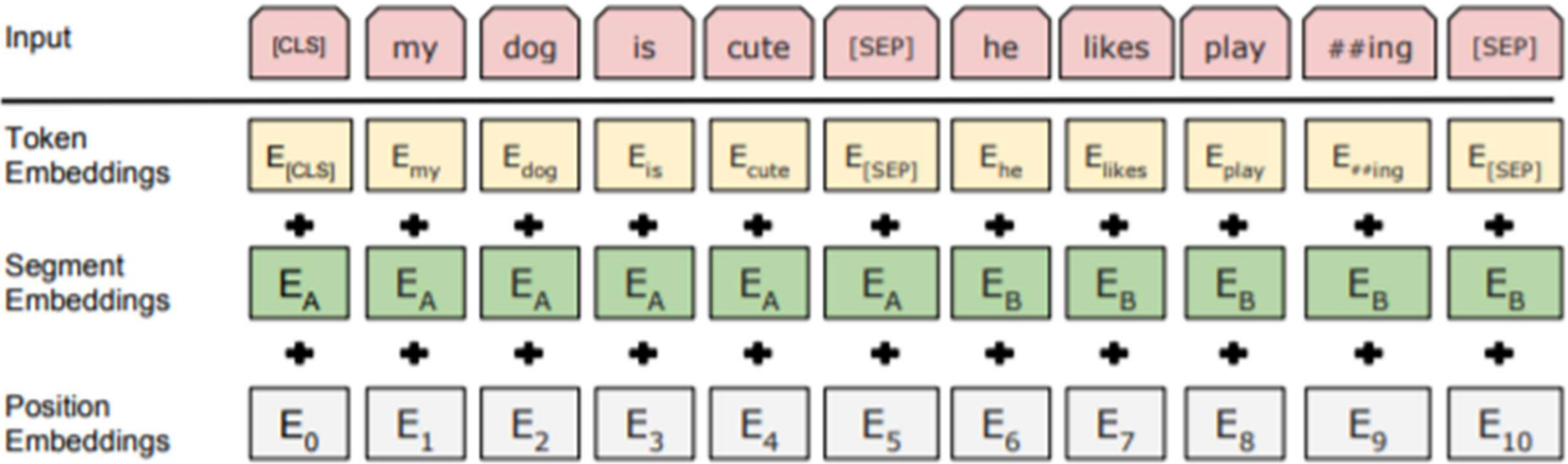
# Transformers

## BERT: Model Architecture



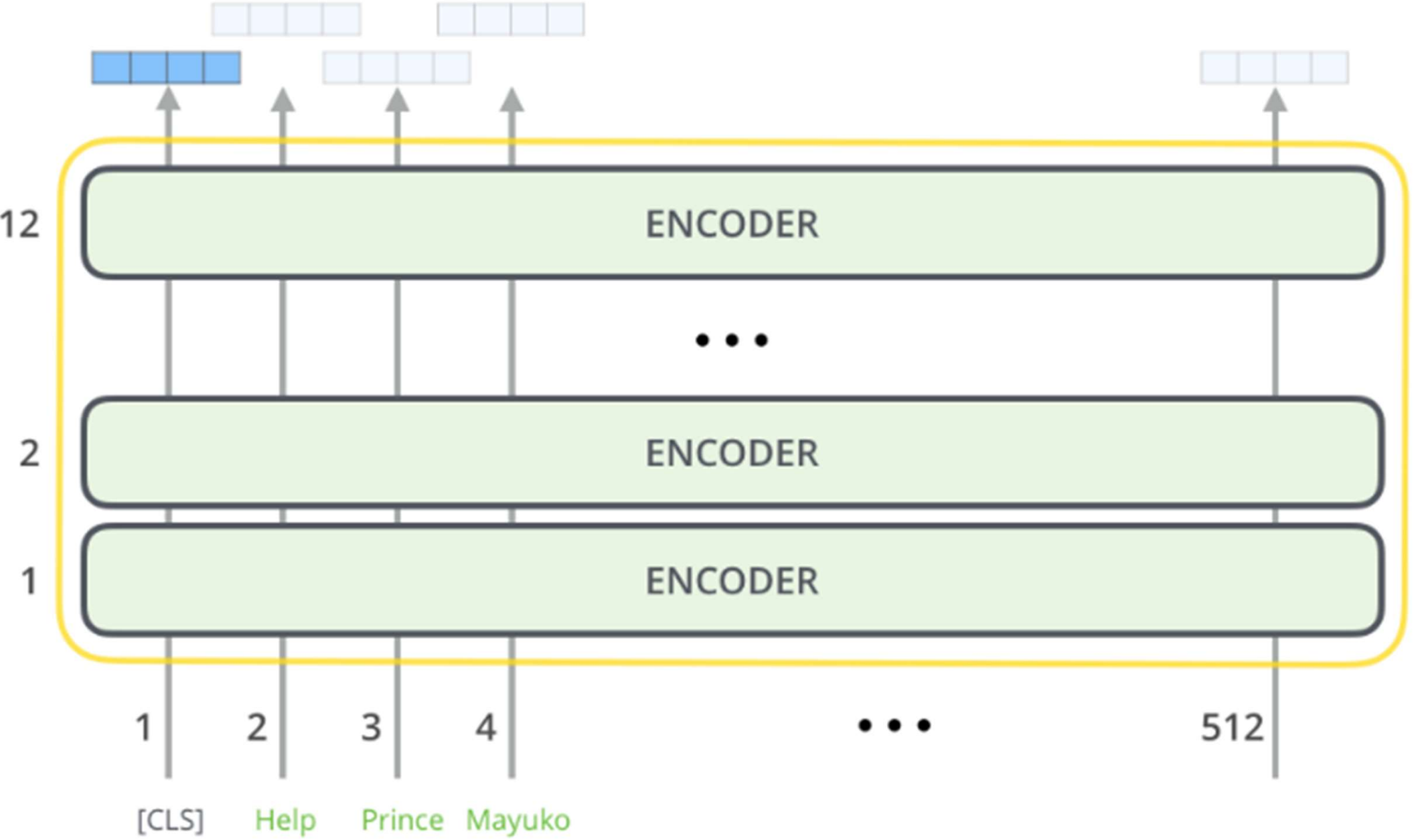
# Transformers

## BERT: Model Architecture



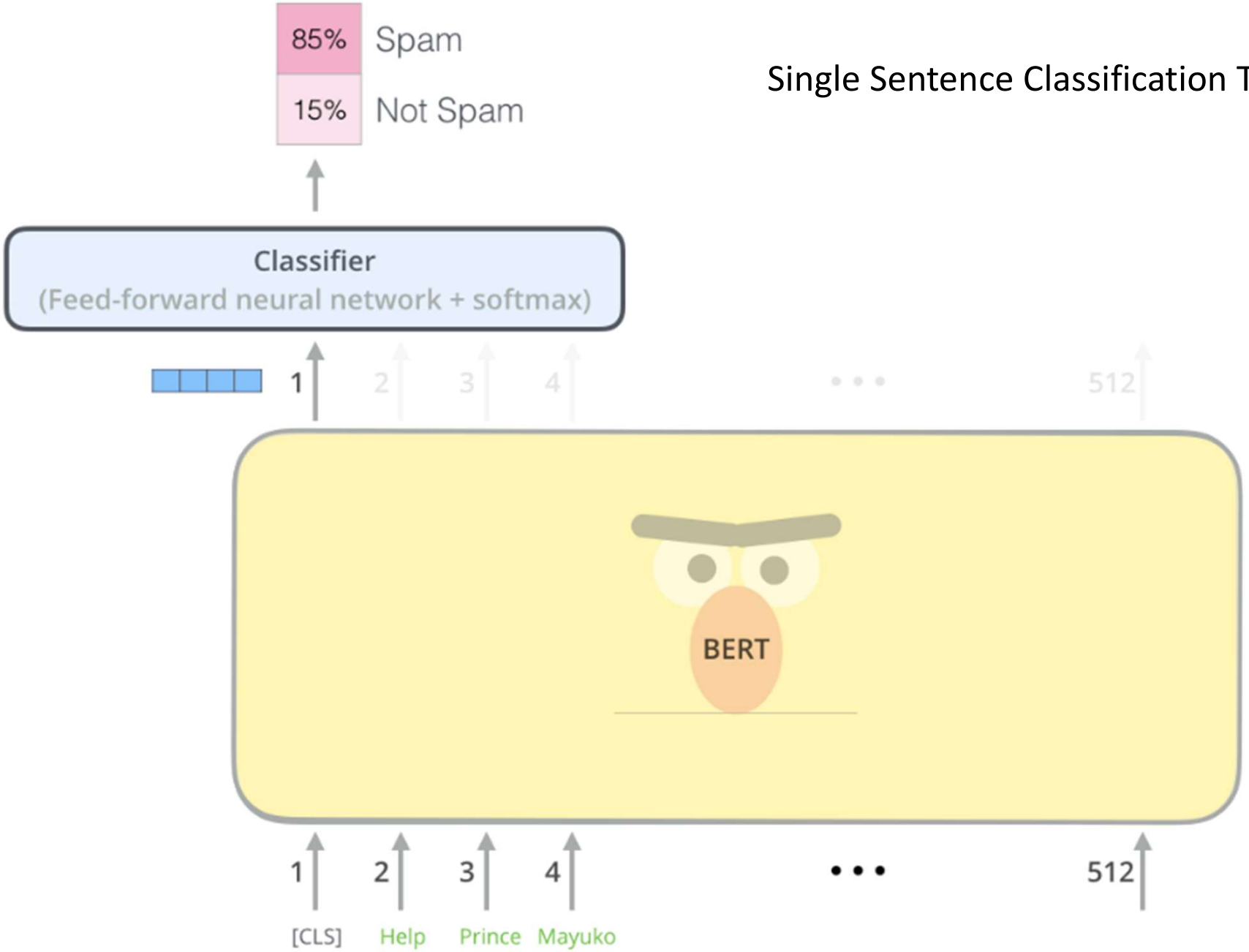
# Transformers

## BERT: Model Architecture



# Transformers

## BERT: Model Architecture




Single Sentence Classification Tasks: SST-2, CoLA

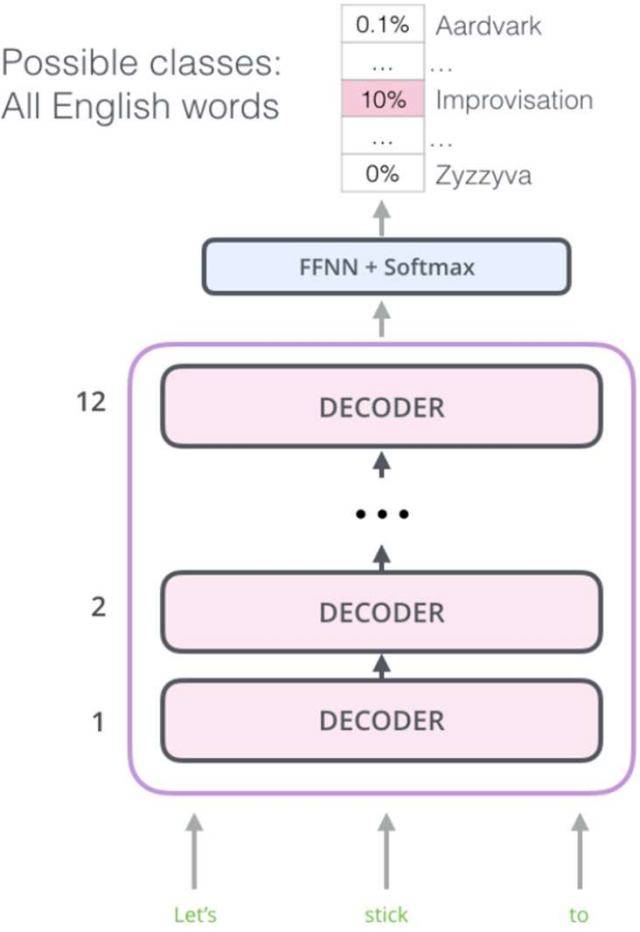
# Transformers

## BERT: Comparison with OpenAI GPT

# Transformers

## BERT: Comparison with OpenAI GPT

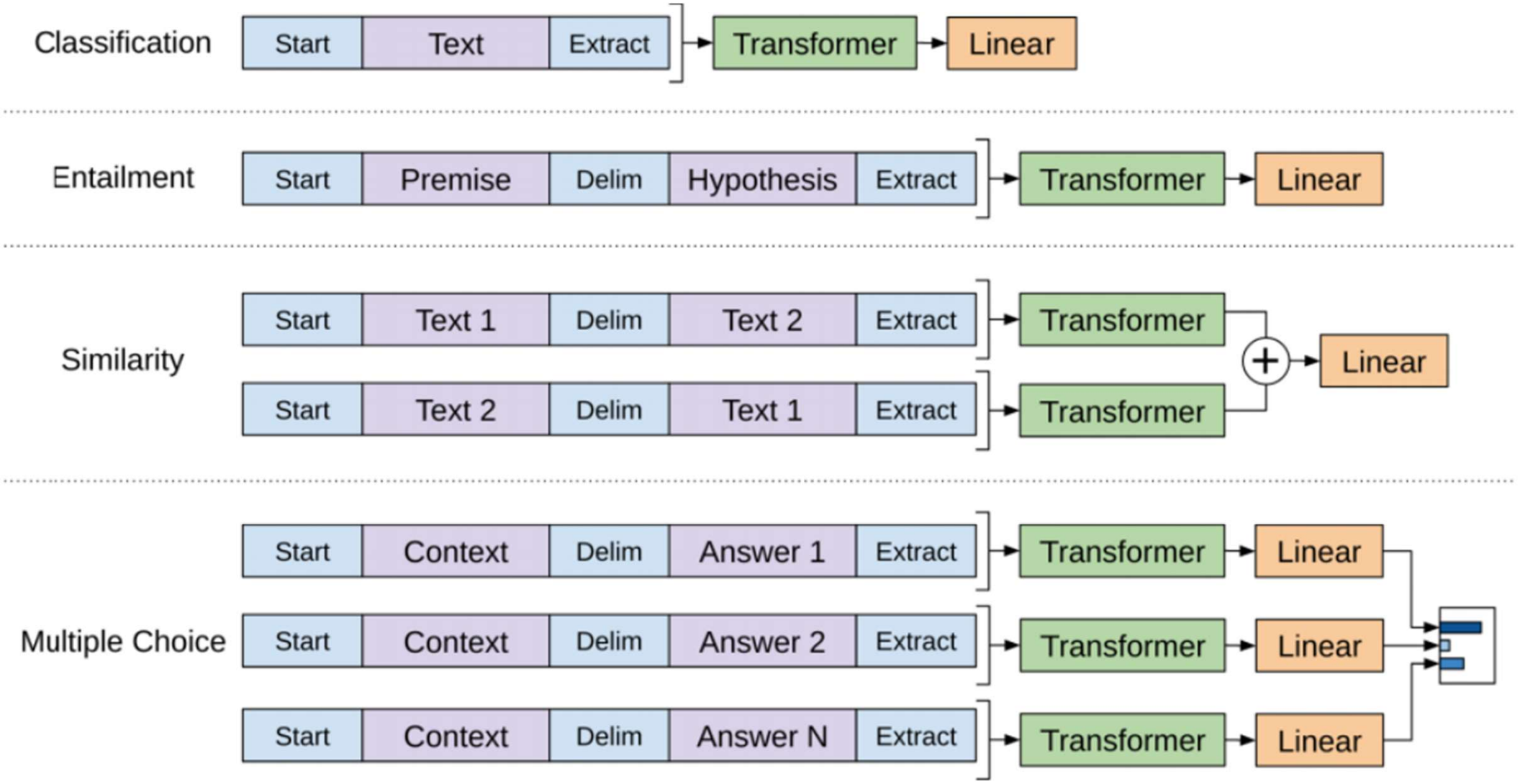





- GPT is basically the Decoder component of the original Transformer model → it is an AR LM
- 12 decoder layers
  - Unidirectional, deep
  - Masked self-attention
  - No encoder-decoder attention (there is no encoder)
  - Residual connection, layer norm

# Transformers

## BERT: Comparison with OpenAI GPT



# Transformers

## BERT: Pretraining BERT

# Transformers

## BERT: Pretraining BERT

- BooksCorpus (800M words) + English Wikipedia (2.5B words, text passages only) → 3.3B words
  - Important to use document-level corpus rather than sentence-level corpus (in order to extract long contiguous sequences)
- WordPiece embedding with 30K vocabulary
  - "he likes playing" → "he likes play ##ing"
- 1st token of each sequence is always [CLS]
- Sequences separated by [SEP] tokens
- Max sequence length of 512 tokens (padded if necessary)
  - First pre-train model with 128 tokens for 90% of total steps, then train the final 10% of steps using 512 tokens → speed up training
  - Self-attention is *O(n^2 x d)*

# Transformers

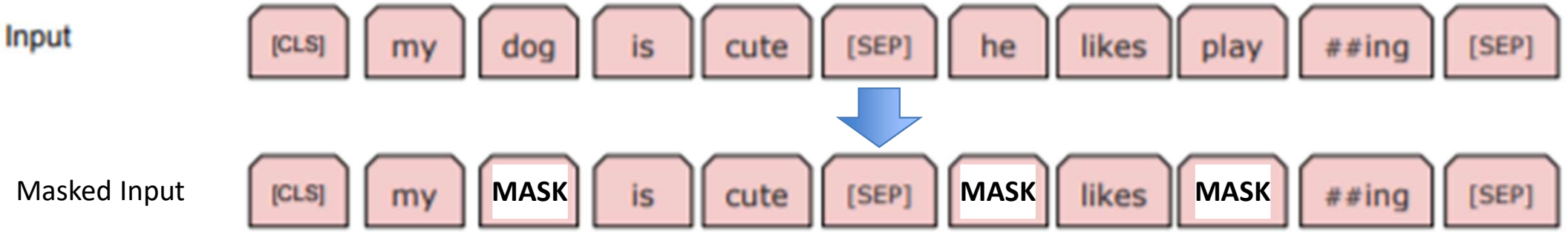
## BERT: Pretraining BERT

- Want the model to learn token representations using bi-directional context
- Want the model to learn relationships between sentences
- Trained using two objectives → Masked LM (MLM) and Next Sentence Prediction (NSP)
  - MLM learns bi-directional context
  - NSP learns relationships between sentences

| Tasks | Dev Set MNLI-m (Acc) | QNLI (Acc) | MRPC (Acc) | SST-2 (Acc) | SQuAD (F1) |
|---|---|---|---|---|---|
| $\text{BERT}_{\text{BASE}}$ | 84.4 | 88.4 | 86.7 | 92.7 | 88.5 |
| No NSP | 83.9 | 84.9 | 86.5 | 92.6 | 87.9 |
| LTR & No NSP | 82.1 | 84.3 | 77.5 | 92.1 | 77.8 |
| + BiLSTM | 82.1 | 84.1 | 75.7 | 91.6 | 84.9 |

# Transformers

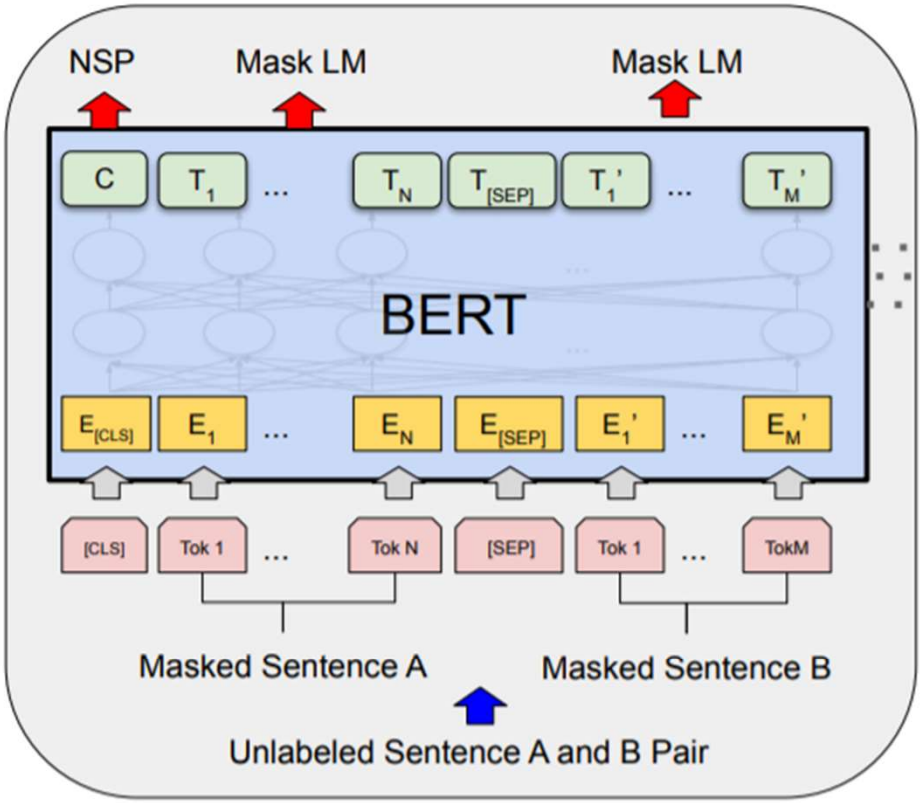
## BERT: Pretraining BERT




- MLM
  - Mask some percentage of the input tokens (i.e., replace with [MASK] token) at random and only predict the masked tokens
  - Creates mismatch between pre-training and fine-tuning since [MASK] token does not appear for fine-tuning tasks
  - 15% of tokens are randomly chosen for masking; of those 15% →
    - 80% receive [MASK] token: my dog is hairy → my dog is [MASK]
    - 10% replaced with a random token: my dog is hairy → my dog is apple
    - 10% unchanged: my dog is hairy → my dog is hairy
  - Learning is slower since MLM only predicts 15% of tokens in each batch → model converges slower but outperforms standard LM almost immediately

# Transformers

## BERT: Pretraining BERT




Pre-training

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

- NSP
  - When choosing Sentences *A* and *B* for each pre-training example → 50% of the time *B* is the actual sentence that follows *A,* 50% of the time *B* is a random sentence
  - Final representation of [CLS] token is used for NSP task
  - [CLS] token is ***not*** a good sequence representation without fine-tuning
    - Could use aggregation of token embeddings instead

# Transformers

## BERT: Pretraining BERT

- Scaling to extreme model sizes leads to large improvements on very small scale tasks (3600 examples in MRPC), **provided that the model has been sufficiently pre-trained**
- Previous works showed the opposite but used a feature-based approached
- BERT argues that fine-tuning an adequately pre-trained model does better since the task-specific model benefits from the expressivity of the pre-trained representations

# Transformers

## BERT: Fine-Tuning BERT

# Transformers

## BERT: Fine-Tuning BERT

- Fine-tuning is cheap, easy, and intuitive
  - Cheap → ~hrs on a single GPU
  - Easy → Slight modifications to input and output for a variety of tasks
  - Intuitive → Self-attention inherently includes bi-directional cross attention between two sequences; thus, no need to independently encode text pairs
- All parameters are fine-tuned end-to-end
  - Large labeled datasets are less sensitive to hyperparameters
- Input Sentence *A* and Sentence *B* from pre-training are:
  - Sentence pairs in paraphrasing
  - Hypothesis-premise pairs in entailment
  - Question-passage pairs in Q&A
  - Text-NULL pairs in text classification/sequence tagging
- Output
  - Individual tokens fed to output layer for tasks such as NER or Q&A
  - [CLS] token is used for classification tasks

# Transformers

## BERT: Feature-Based BERT

# Transformers

## BERT: Feature-Based BERT

- Feature-based approach has certain advantages
  - Downstream task might require task-specific model
  - Computational benefits from using pre-computed features
- Extract activations from one or more layers *without* fine-tuning any parameters of BERT → then use activations as input to a randomly initialized two-layer BiLSTM with a classification layer

| System | Dev F1 | Test F1 |
|---|---|---|
| ELMo (Peters et al., 2018a) | 95.7 | 92.2 |
| CVT (Clark et al., 2018) | - | 92.6 |
| CSE (Akbik et al., 2018) | - | **93.1** |
| Fine-tuning approach | | |
| $\text{BERT}_{\text{LARGE}}$ | 96.6 | 92.8 |
| $\text{BERT}_{\text{BASE}}$ | 96.4 | 92.4 |
| Feature-based approach ($\text{BERT}_{\text{BASE}}$) | | |
| Embeddings | 91.0 | - |
| Second-to-Last Hidden | 95.6 | - |
| Last Hidden | 94.9 | - |
| Weighted Sum Last Four Hidden | 95.9 | - |
| Concat Last Four Hidden | 96.1 | - |
| Weighted Sum All 12 Layers | 95.5 | - |

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

# Transformers

## BERT: Pros and Cons

# Transformers

## BERT: Pros and Cons

- Pros
  - Deep bi-directional context
  - Easy to fine-tune for downstream tasks
  - Feature-based training also produces great results
  - Parallelizable
- Cons
  - Sequences are still short → might need to model longer-term dependencies
  - Fixed-length context → context fragmentation problem

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention"

- Introduction
- Model Architecture
- Training and Auxiliary Losses

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention": Introduction

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention": Introduction

- Character-level LM is challenging
  - Model must learn a large vocabulary of words from scratch
  - Natural text has dependencies over hundreds to thousands of time steps
  - Character sequences are longer than word sequences → more computation
- Typically done with RNNs using truncated back-propagation through time (TBPTT)
  - Training is complicated (batches must be sequential and hidden states must be preserved from batch to batch)
  - Research shows that RNNs don't actually make use of long-term context
- Transformer-based model can perform better
  - Transformers should be able to "quickly" propagate information over arbitrary distances

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention": Model Architecture

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention": Model Architecture

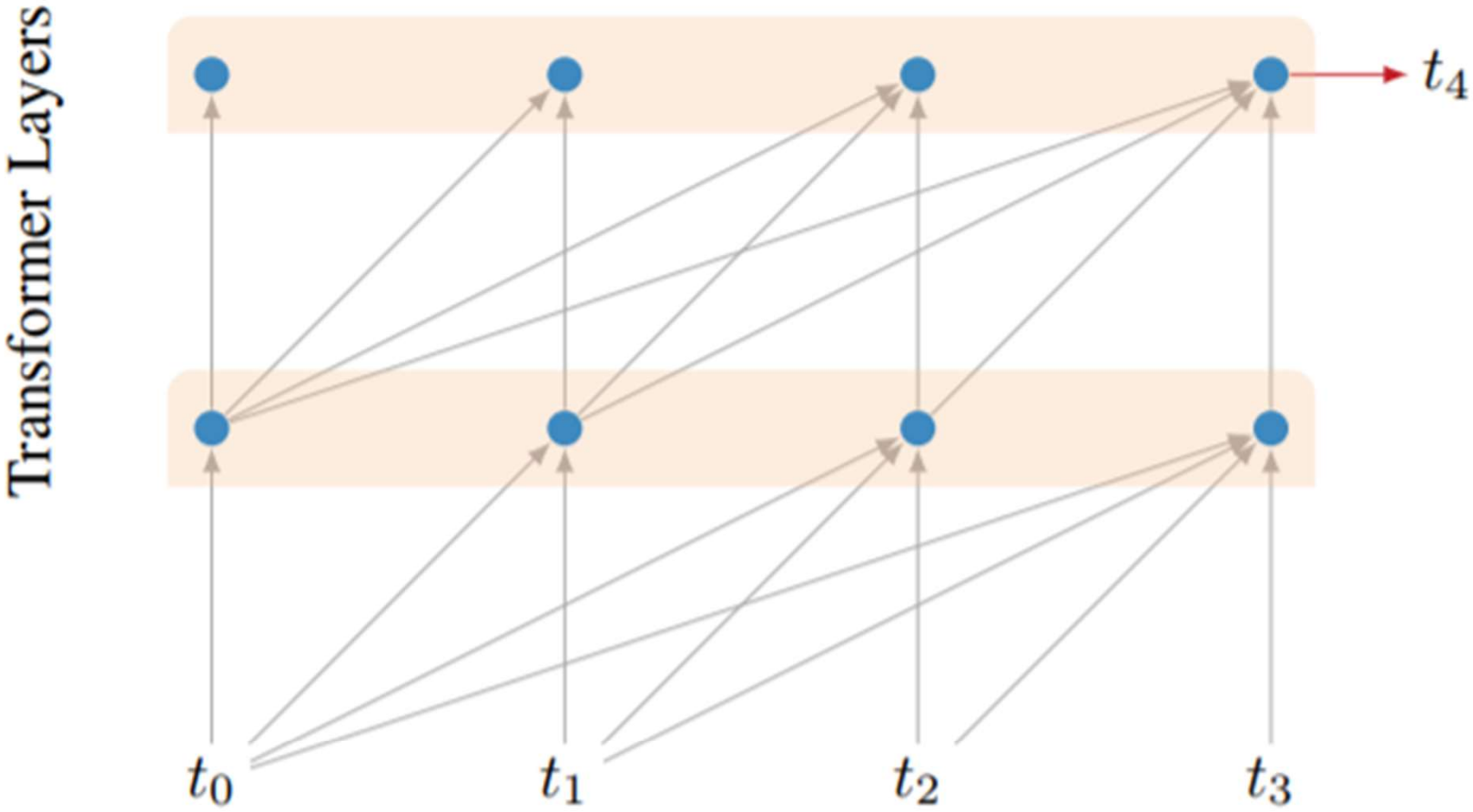



Figure 1: Character transformer network of two layers processing a four character sequence to predict $t_4$. The causal attention mask limits information to left-to-right flow. Red arrows highlight the prediction task the network has to learn.

*Actual model has 64 layers

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention": Training and Auxiliary Losses

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention": Training and Auxiliary Losses

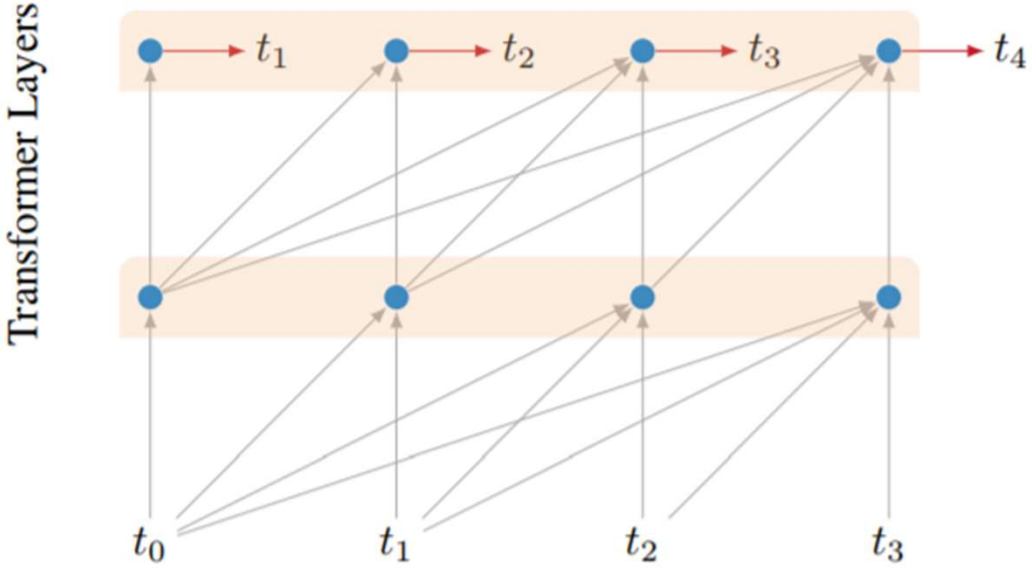



Figure 2: Adding the intermediate positions prediction tasks to our network. Now, we predict the final character $t_4$ and all intermediate characters $t_{0:3}$. $t_3$ has access only to $t_{0:2}$ because of the causal attention masks. All of these losses contribute equally during training.

- Predictions at multiple positions in final layer
  - Similar practice as in RNN models
  - However, model has to make predictions using smaller context (no hidden state is passed between batches)

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention": Training and Auxiliary Losses

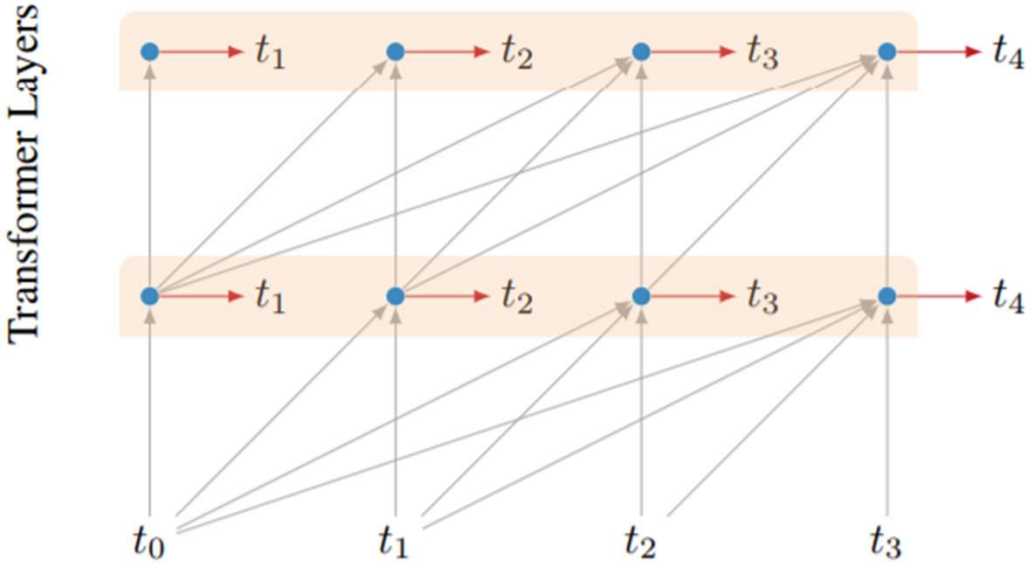



Figure 3: Our network after adding prediction tasks for the intermediate layers. For this example of two layers, the losses of the intermediate layer prediction tasks will be absent after finishing 25% of the training.

- Predictions at intermediate layers
  - Lower layers contribute less and less to loss as training progresses
  - Intermediate layer losses eventually become 0

# Transformers

"Character-Level Language Modeling with Deeper Self-Attention": Training and Auxiliary Losses

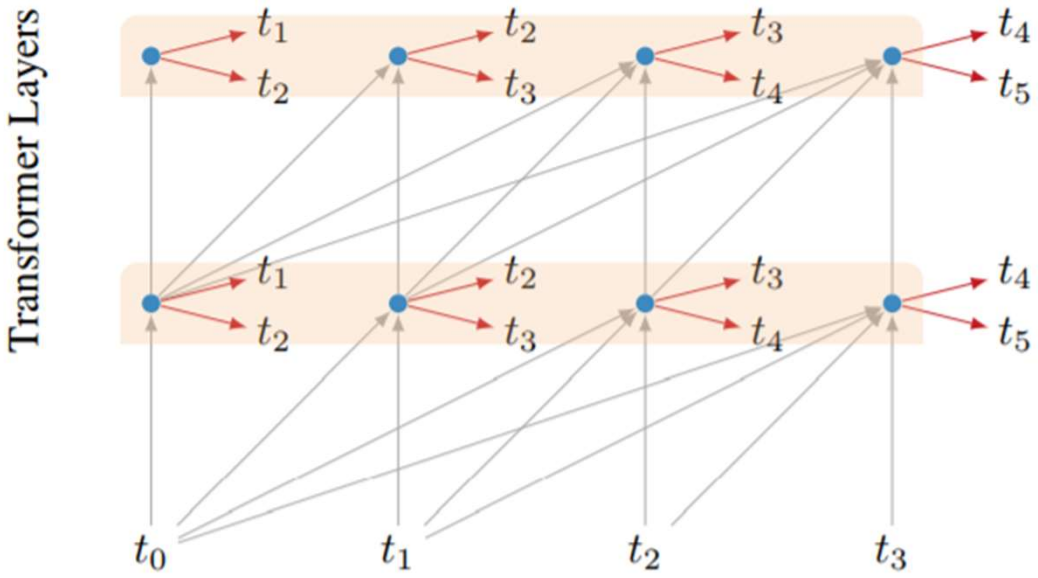



Figure 4: Our example network after adding two predictions per position.

- Predictions at future positions
  - At each position in the sequence, model makes 2 (or more) predictions of future characters

# Transformers

"Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context"

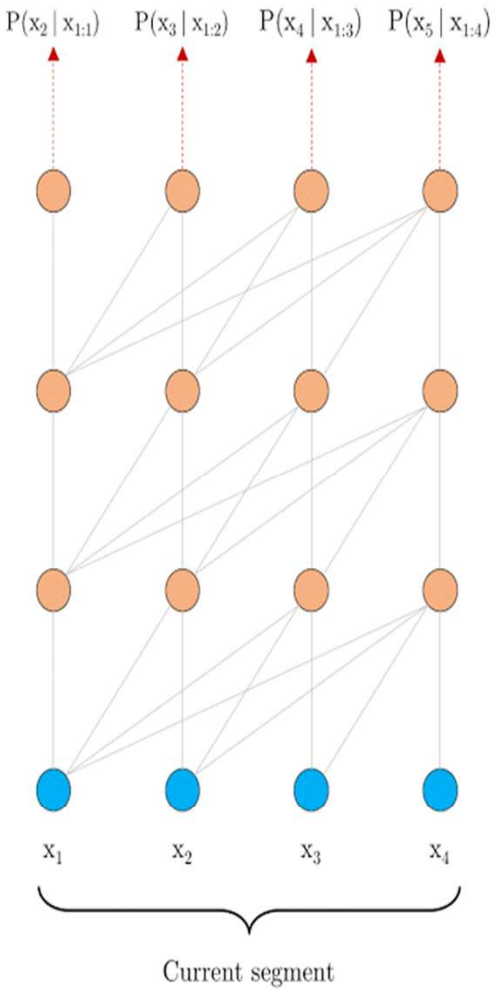
- Introduction
- Model Architecture
- Relative Positional Encodings

# Transformers

## Transformer-XL: Introduction

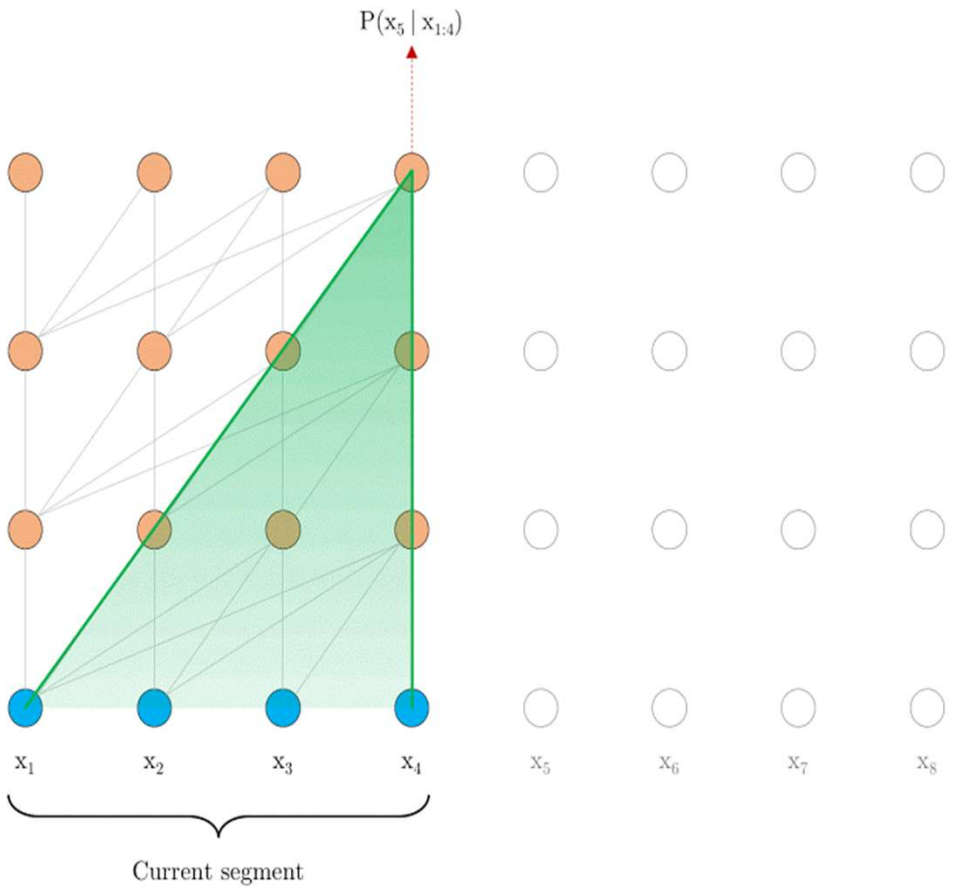
# Transformers

## Transformer-XL: Introduction



# Transformers

## Transformer-XL: Introduction



# Transformers

## Transformer-XL: Introduction

- Transformer-XL is Transformers with Recurrence
  - Combines ideas from RNNs with ideas from Transformers
  - Consists of a segment-level recurrence mechanism and a new positional encoding scheme
  - Enables learning context without using a fixed-length window and without losing temporal coherence between segments
- Can generate looooooooooooooooong sequences (XL = Extra Long)
  - 80% longer than RNNs
  - 450% longer than vanilla Transformers
  - Measured using *Relative* Effective Context Length (RECL)
- Fast
  - 1800+ times faster than vanilla Transformers during evaluation
  - Measured with RECL using per-token time on one GPU

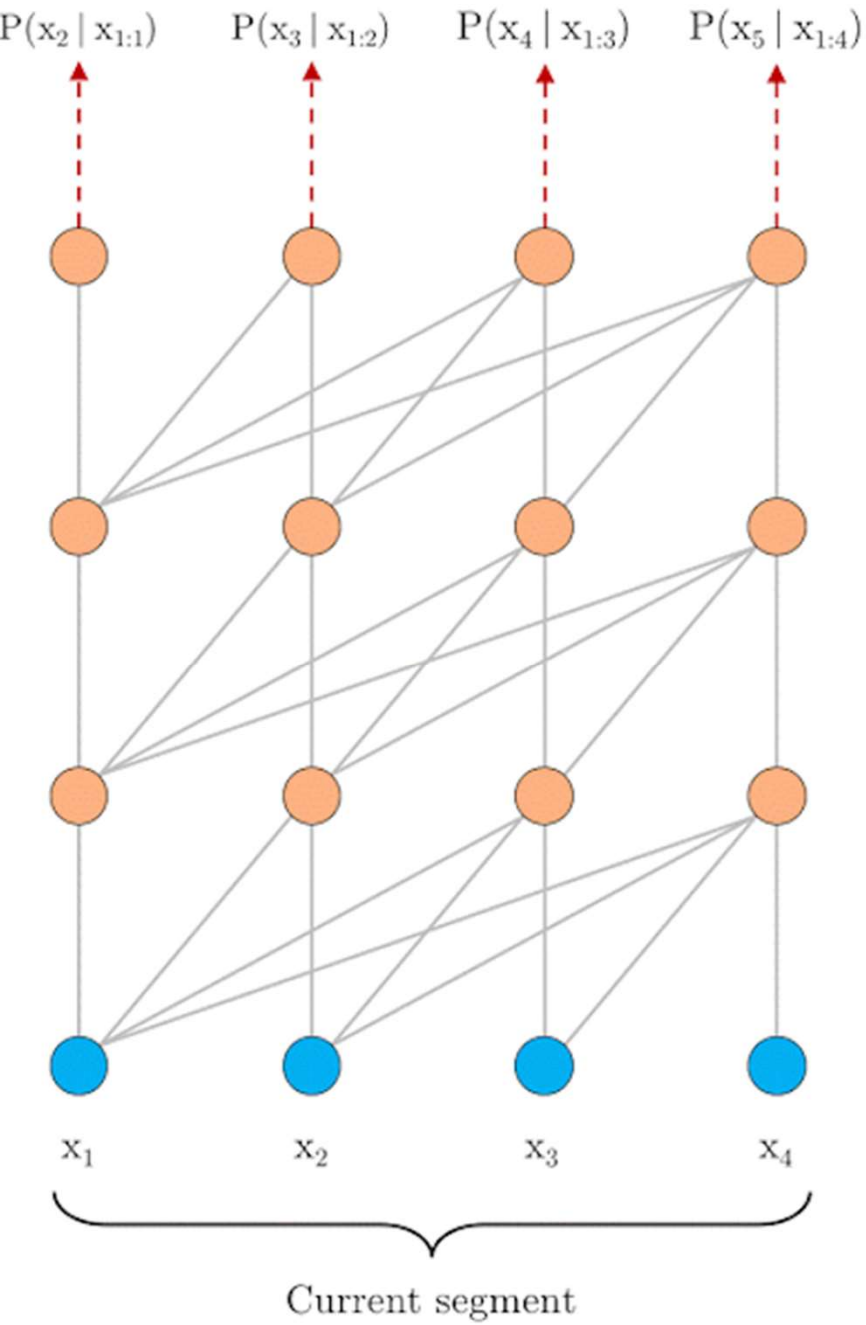
# Transformers

## Transformer-XL: Introduction

- Recurrent segment memory
  - Information is propagated from one segment to the next via recurrent connections → addresses context fragmentation problem
  - Hidden states (i.e., layer outputs) from previous segments are reused → they serve as a "memory" for the current segment
- Relative positional encodings
  - Vanilla Transformers use absolute positional encodings → position is referenced to start of input sequence; **does not allow hidden state reuse**
  - ***Relative*** encodings → position is relative to each input sequence; **model can generalize to attention lengths longer than the one observed during training**
- Need both technical ideas to make Transformer-XL work

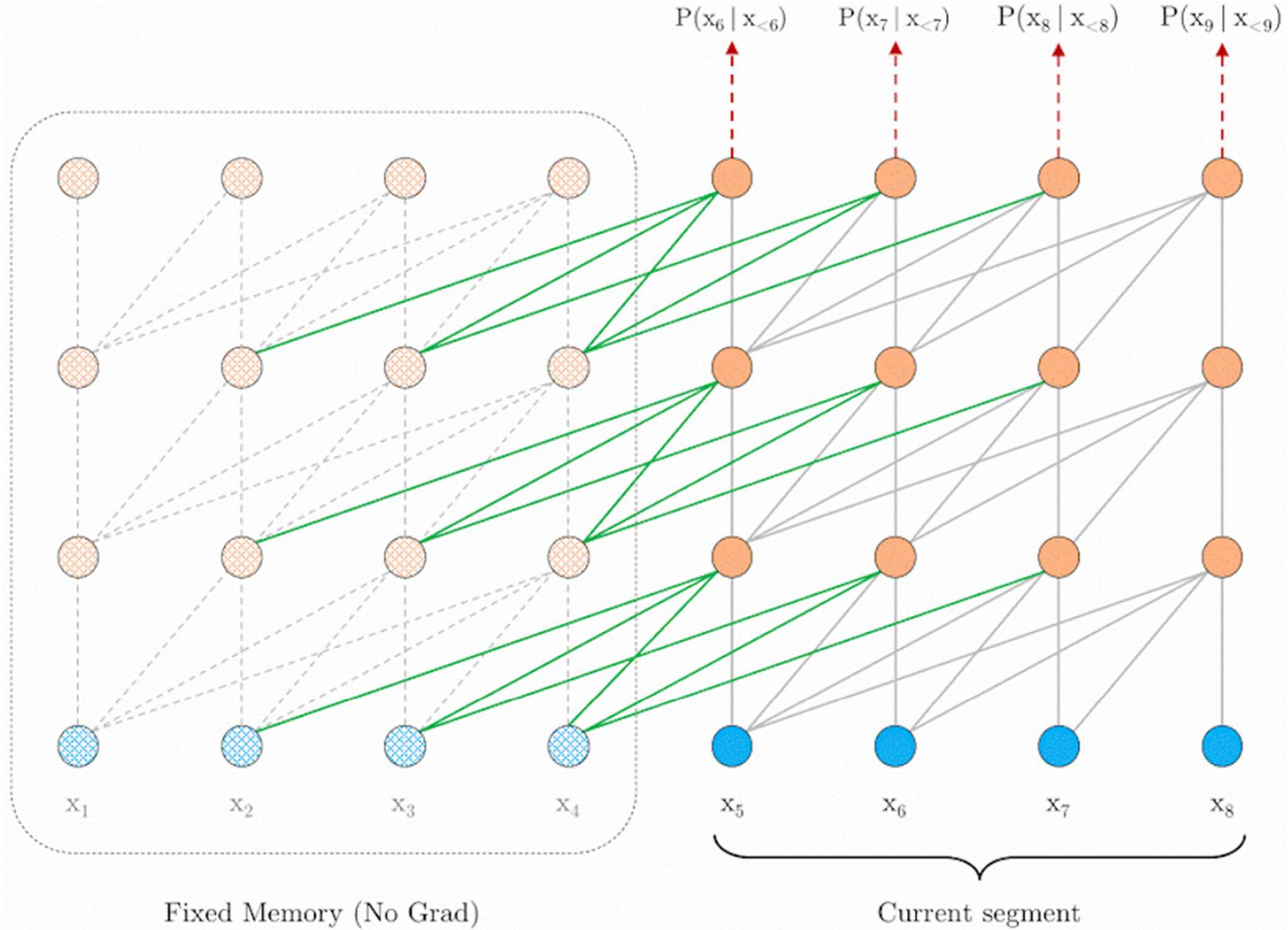
# Transformers

## Transformer-XL: Model Architecture



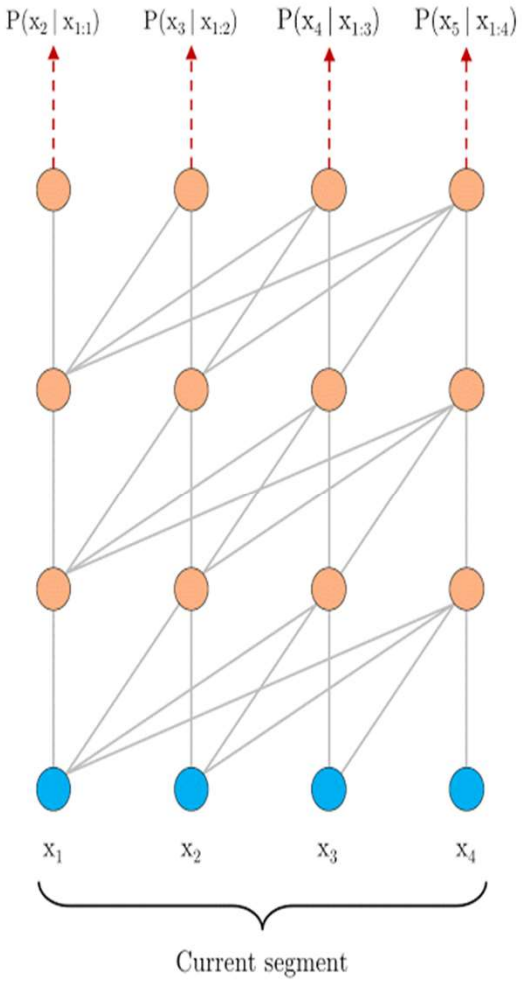
# Transformers

## Transformer-XL: Model Architecture



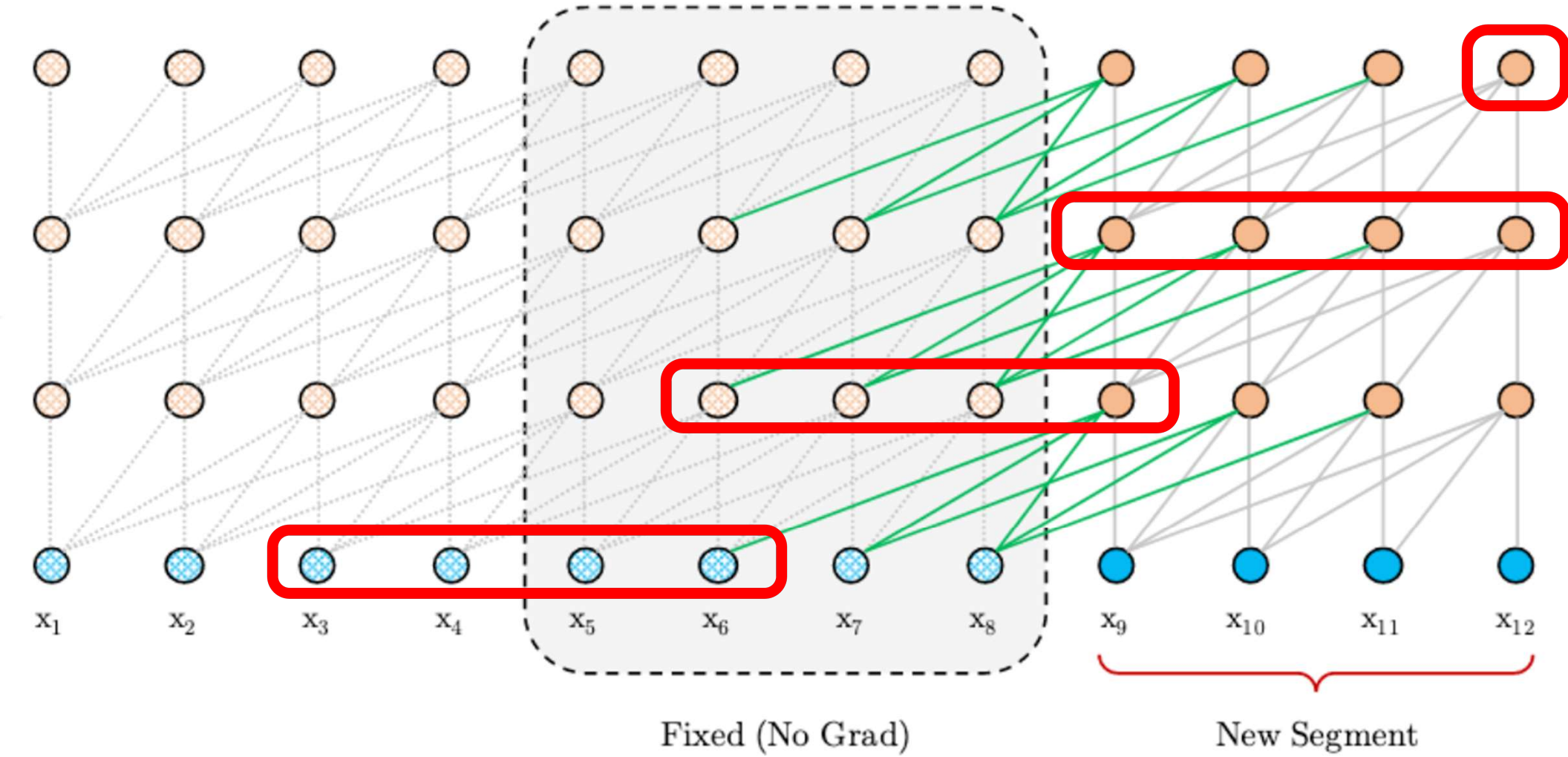
# Transformers

## Transformer-XL: Model Architecture



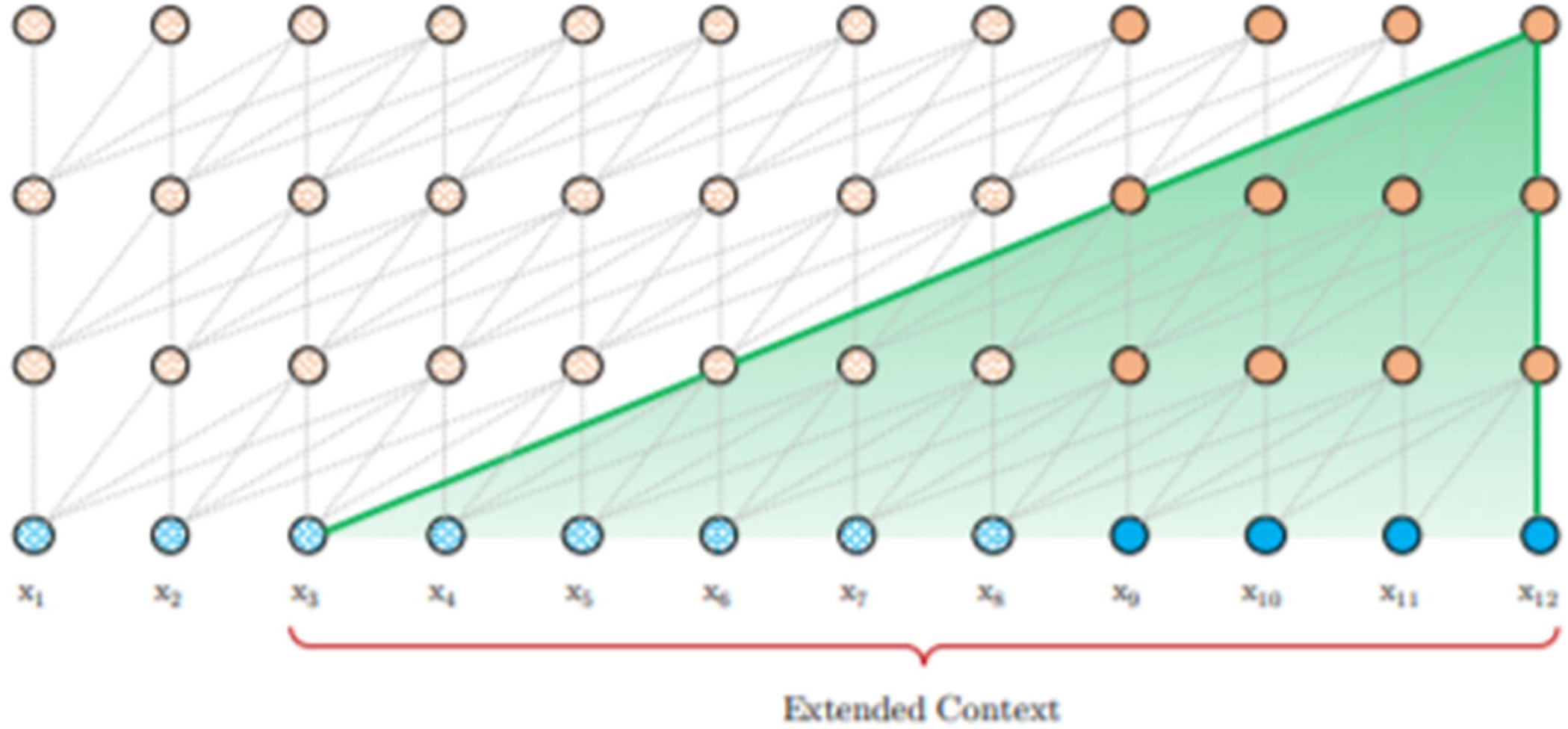
# Transformers

## Transformer-XL: Model Architecture



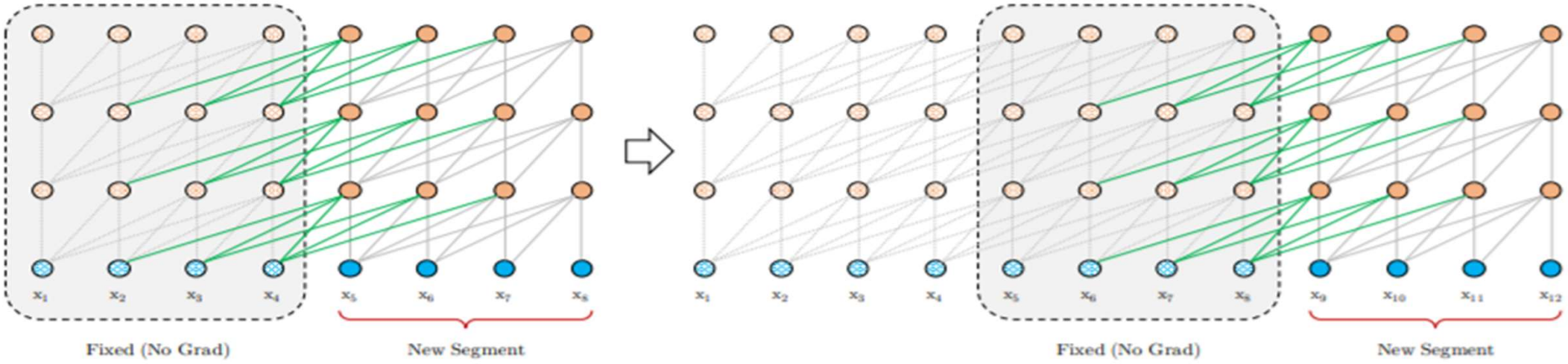
# Transformers

## Transformer-XL: Model Architecture



# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

- *n*-th layer
- τ-th segment
- *d*: hidden dimension
- $h^n\_\tau$: Hidden state for τ-th segment and *n*-th layer
- **W**: model weights

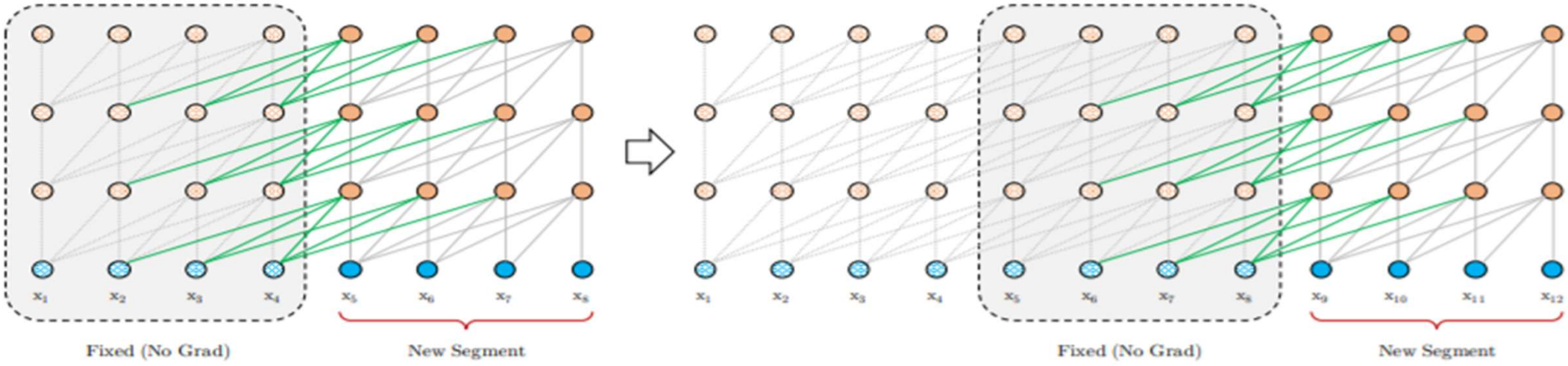
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

No gradient for for hidden states from previous segment

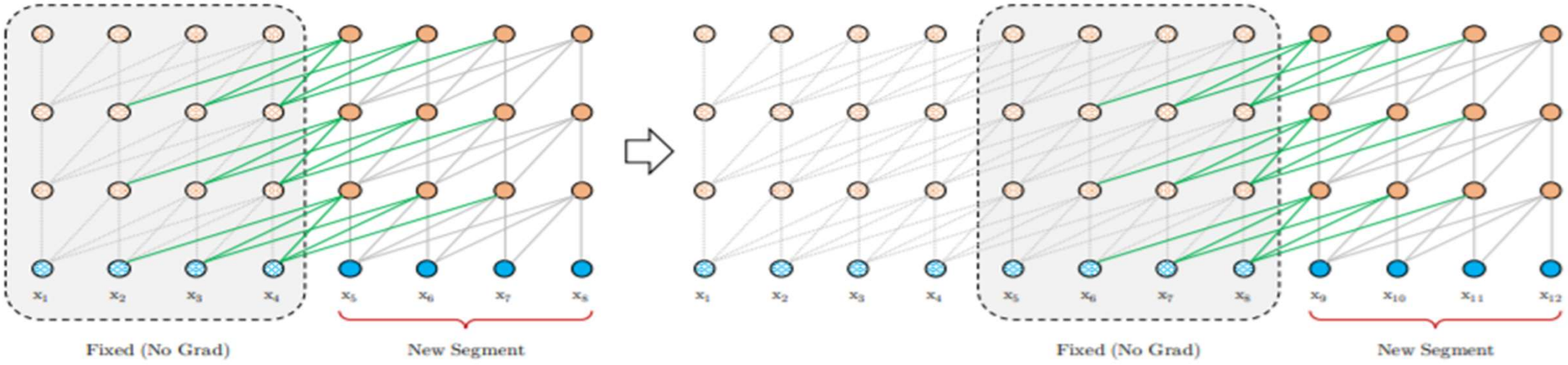
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

*Extended* hidden state for previous layer is concatenation of previous layer hidden states from two segments

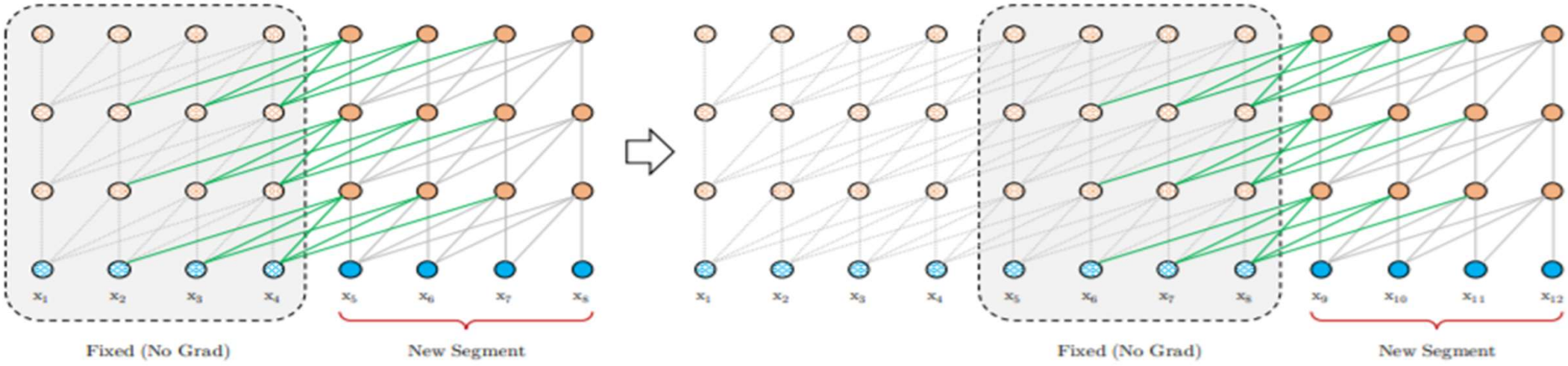
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

Self-attention Q/K/V calculation using extended hidden state

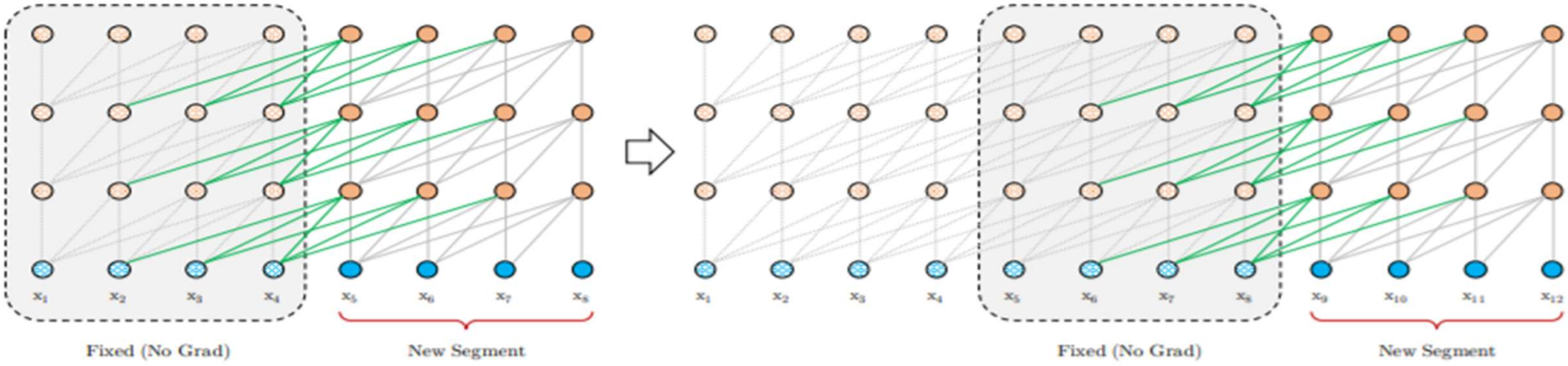
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

Query comes from current segment

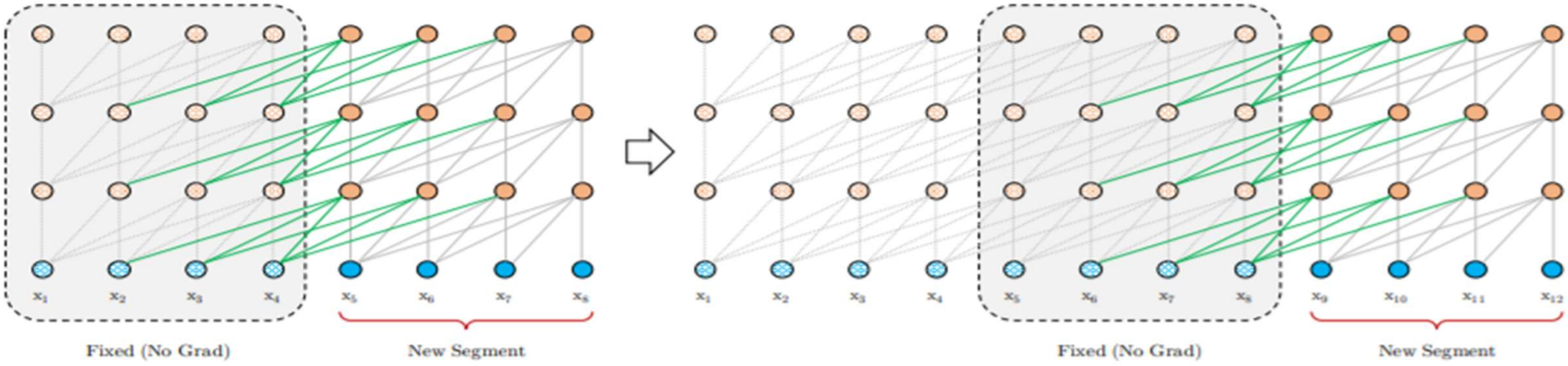
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

Key/Value comes from current and previous segments

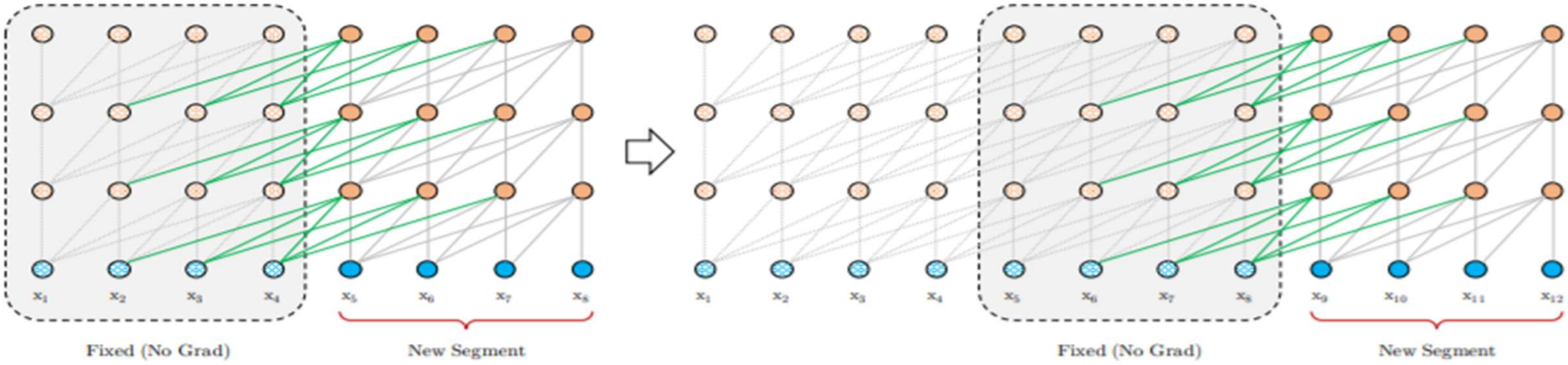
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

Masked-Softmax + Feed Forward + Layer Norm

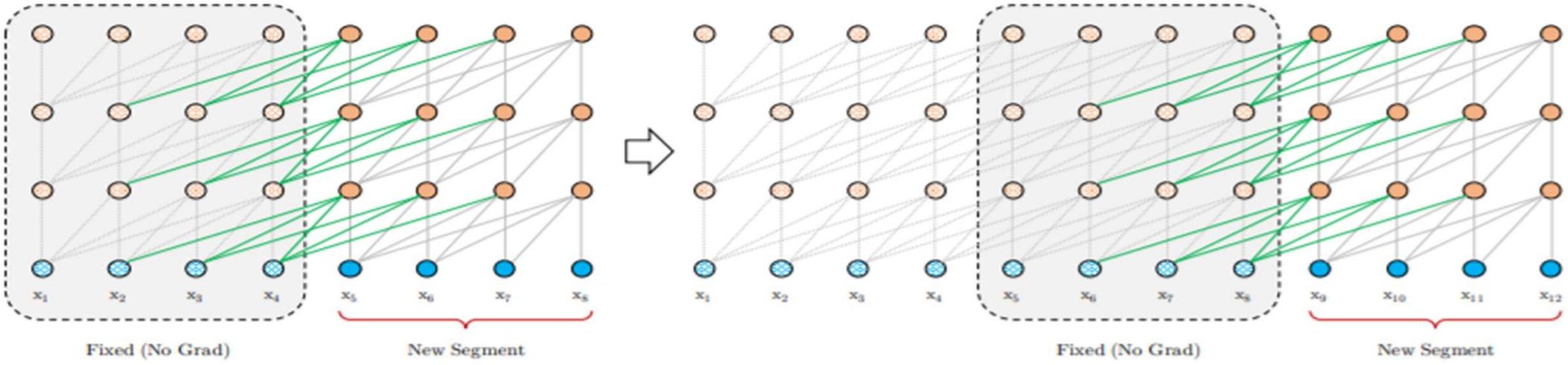
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

Recurrent dependency between two segments shifts one layer downwards per segment

Longest possible dependency length is *O(N x L)* → # of layers (N) and segment length (L)

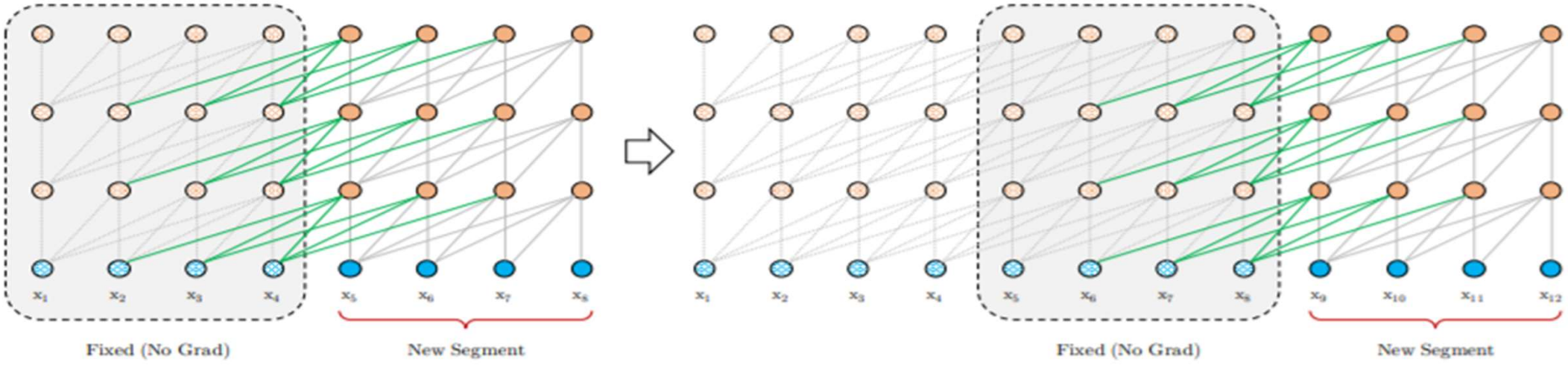
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

## Transformer-XL: Model Architecture




(a) Training phase.

- Can cache and reuse more than just a single previous segment → can use as many as GPU memory allows
- During training → Use hidden states equal to segment length

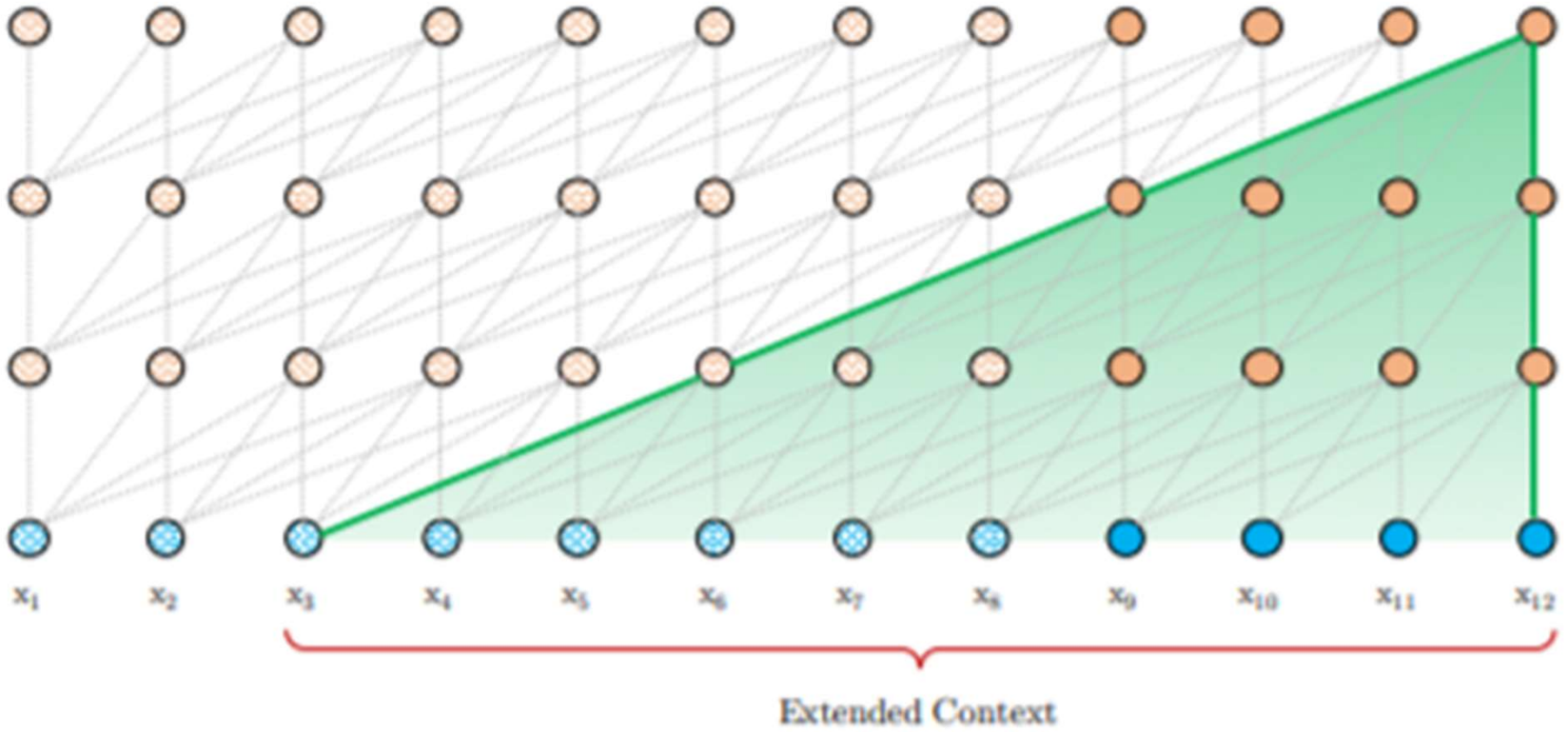
$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

# Transformers

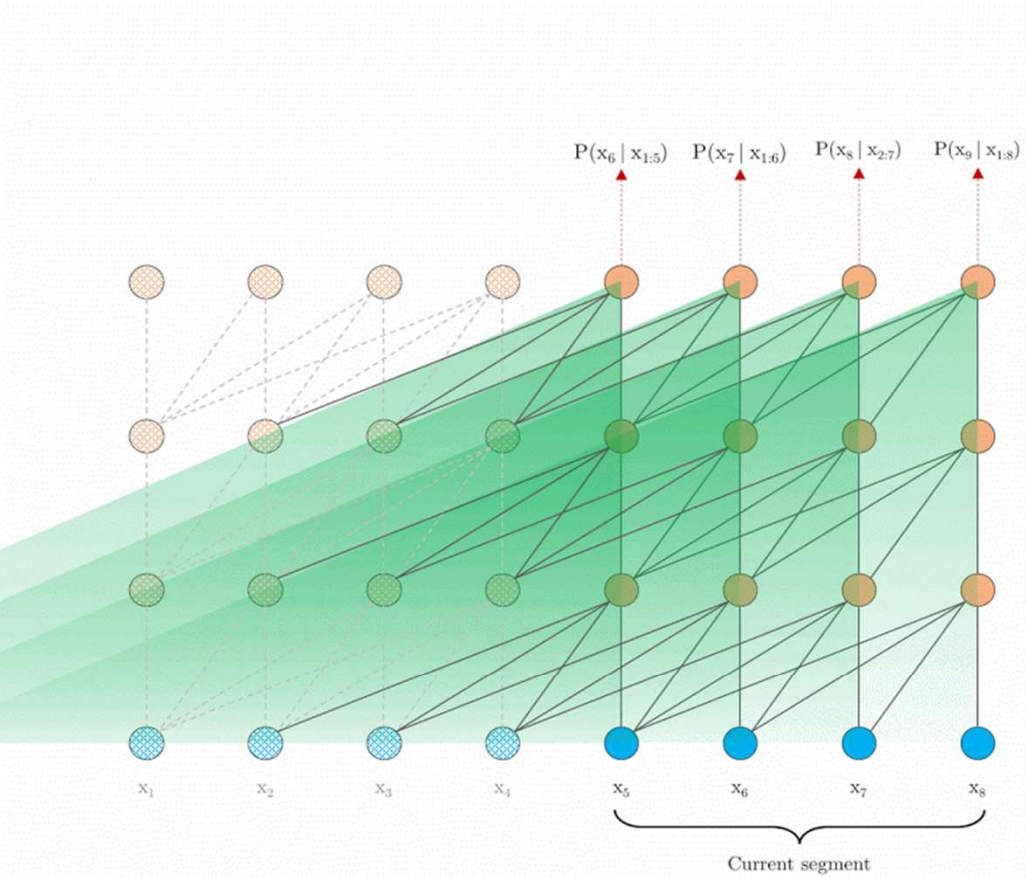
## Transformer-XL: Model Architecture




- During evaluation → use multiple segment lengths for increased context
- In addition, representations from previous segments can be reused instead of computed from scratch

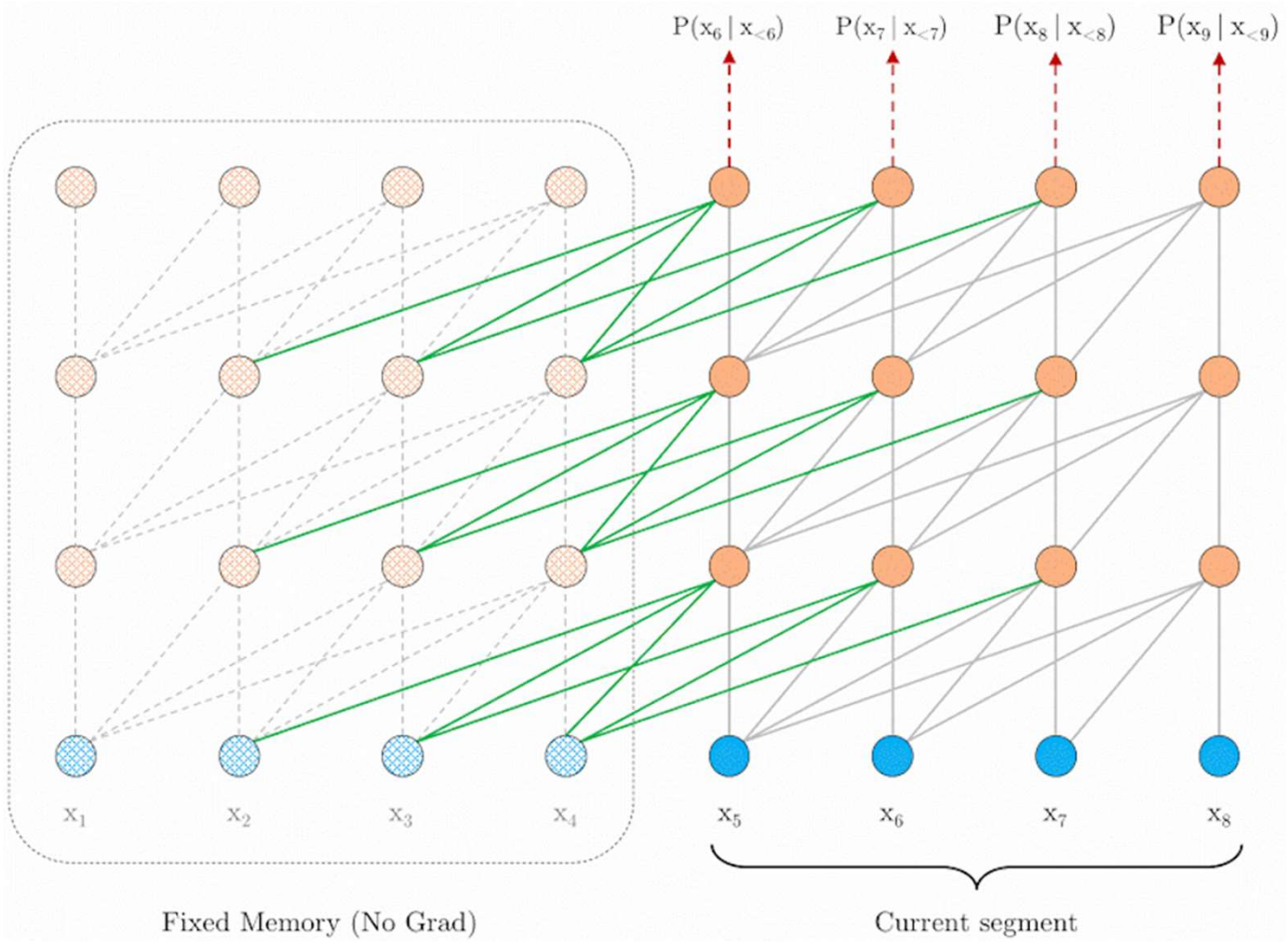
# Transformers

## Transformer-XL: Model Architecture



# Transformers

## Transformer-XL: Relative Positional Encodings




[0, 1, 2, 3]

[0, 1, 2, 3]

# Transformers

## Transformer-XL: Relative Positional Encodings

- Positional encodings (PEs) give the model a temporal clue or “bias” about how information is presented
  - Tells the model how to attend in a sequential manner
  - PEs can be added to the attention score at each layer and learned instead of a one-time summation with the initial embeddings
- Absolute PEs
  - Word embeddings from two segments have same position encoding
  - Model has no way of differentiating tokens at same position in different sequences

$\mathbf{E}_{\mathbf{s}_\tau} \in \mathbb{R}^{L \times d}$ Word embeddings

$\mathbf{U}_{1:L}$ Position wise embeddings

$$\mathbf{h}_\tau = f(\mathbf{h}_{\tau-1}, \mathbf{E}_{\mathbf{s}_\tau} + \mathbf{U}_{1:L})$$

$$\mathbf{h}_{\tau+1} = f(\mathbf{h}_\tau, \mathbf{E}_{\mathbf{s}_{\tau+1}} + \mathbf{U}_{1:L})$$

# Transformers

## Transformer-XL: Relative Positional Encodings

- PEs should be defined in a ***relative*** manner
  - Only care about how one token (Key) is positioned relative to some signifier token (Query), not its absolute position in the overall sequence
- By adding relative PEs to each layer, the Query can distinguish tokens at the same position in different sequences → allows reuse mechanism for recurrent connections
- Model can always recover absolute position recursively from relative positions

# Transformers

## Transformer-XL: Relative Positional Encodings

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

- Q = (E + U)$\mathbf{W_q}$
- K = (E + U)$\mathbf{W_k}$
- QK = [(E + U)$\mathbf{W_q}$][(E + U)$\mathbf{W_k}$]

$$\mathbf{A}^{\text{abs}}_{i,j} = q_i^\top k_j = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}$$

- E_j/U_j: Word embedding/absolute PE for Key vector j
- E_i/U_i: Word embedding/absolute PE for Query vector i

# Transformers

## Transformer-XL: Relative Positional Encodings

$$\mathbf{A}^{\text{abs}}_{i,j} = q_i^\top k_j = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}$$

- (a): Query-Key based addressing → attention from context at query *(i)* and key *(j)* positions
- (b): Query-dependent absolute PE → attention from query at absolute key position
- (c): Key-dependent absolute PE → attention from key at absolute query position
- (d): Global absolute PE → attention from global absolute PE

# Transformers

## Transformer-XL: Relative Positional Encodings

$$\mathbf{A}^{\text{rel}}_{i,j} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}$$

- (a): Query-Key based addressing → attention from context at query *(i)* and key *(j)* positions
- (b): Query-dependent relative PE → attention from query at relative key position
- (c): Global content bias → attention from all key positions
- (d): Global relative PE → attention from all relative PE

# Transformers

## Transformer-XL: Relative Positional Encodings

$$\mathbf{A}^{\text{rel}}_{i,j} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}$$

- *R* is just the sinusoid from original Transformer but with relative position *i* – *j* as input
- Introduce trainable parameters *u* and *v* for dimension matching
- Separate weights for keys, W_k, into W_k_E and W_k_R
  - W_k_E → weights for content-based Key vectors
  - W_k_R → weights for location-based Key vectors

# Transformers

## Transformer-XL: Relative Positional Encodings

$$\mathbf{h}_\tau^0 := \mathbf{E}_{\mathbf{s}_\tau}$$

For $n = 1, \ldots, N$ :

$$\begin{aligned}
\widetilde{\mathbf{h}}_\tau^{n-1} &= \left[\mathrm{SG}(\mathbf{m}_\tau^{n-1}) \circ \mathbf{h}_\tau^{n-1}\right] \\
\mathbf{q}_\tau^n, \mathbf{k}_\tau^n, \mathbf{v}_\tau^n &= \mathbf{h}_\tau^{n-1}\mathbf{W}_q^{n\top}, \widetilde{\mathbf{h}}_\tau^{n-1}\mathbf{W}_{k,E}^{n\top}, \widetilde{\mathbf{h}}_\tau^{n-1}\mathbf{W}_v^{n\top} \\
\mathbf{A}_{\tau,i,j}^n &= \mathbf{q}_{\tau,i}^{n\top}\mathbf{k}_{\tau,j}^n + \mathbf{q}_{\tau,i}^{n\top}\mathbf{W}_{k,R}^n\mathbf{R}_{i-j} + u^\top\mathbf{k}_{\tau,j} + v^\top\mathbf{W}_{k,R}^n\mathbf{R}_{i-j} \\
\mathbf{a}_\tau^n &= \text{Masked-Softmax}(\mathbf{A}_\tau^n)\mathbf{v}_\tau^n \\
\mathbf{o}_\tau^n &= \text{LayerNorm}(\text{Linear}(\mathbf{a}_\tau^n) + \mathbf{h}_\tau^{n-1}) \\
\mathbf{h}_\tau^n &= \text{Positionwise-Feed-Forward}(\mathbf{o}_\tau^n)
\end{aligned}$$

$W_{k,E}^n$ **Content based**

$v^T$ **Global position bias**

$W_{k,R}^n$ **Location based**

$u^T$ **Global content bias**

$R_{i-j}$ **Relative position embedding**

# Transformers

"XLNET: Generalized Autoregressive Pretraining for Language Understanding"

- Introduction
- Review of Autoregressive and Autoencoding Language Models
- Permutation Language Modeling
- Two-Stream Self-Attention
- Model Architecture
- Partial Prediction
- Incorporating Ideas from Transformer-XL
- Relative Segment Encodings
- Pretraining XLNet
- Ablation Studies

# Transformers

## XLNET: Introduction

# Transformers

## XLNET: Introduction

- BERT → models deep ***bi-directional*** context
- Transformer-XL → models variable-length context
- However,
  - By masking input tokens, BERT neglects context dependency between those masked words → **masked tokens are predicted in parallel, so tokens are generated without knowledge of other generated tokens**
  - Transformer-XL does not learn bi-directional context

# Transformers

## XLNET: Introduction

- XLNet → leverage the best of both AR and AE LMs without their limitations
- No fixed forward or backward factorization order as in vanilla AR LMs
  - Instead, maximizes the *expected* log likelihood of a sequence w.r.t **all possible permutations of the factorization order**
  - In *expectation*, each position learns to use contextual information from all positions (i.e., model captures bi-directional context)
- No masking of input → no pretrain-finetune discrepancy
  - AR objective eliminates the independence assumption made in BERT for the [MASK] tokens
- Extends PE from Transformer-XL to relative ***segment*** encodings
- Integrates segment recurrence and relative positional encoding ideas from Transformer-XL

# Transformers

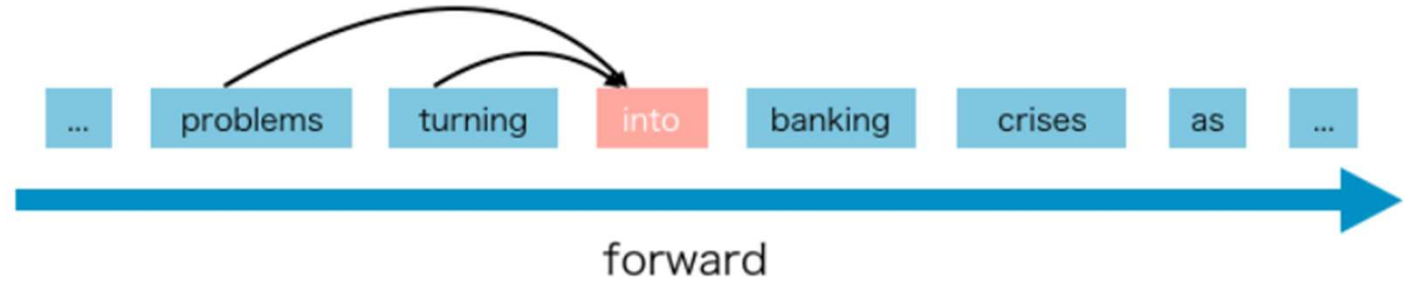
## XLNET: Review of AR and AE LMs

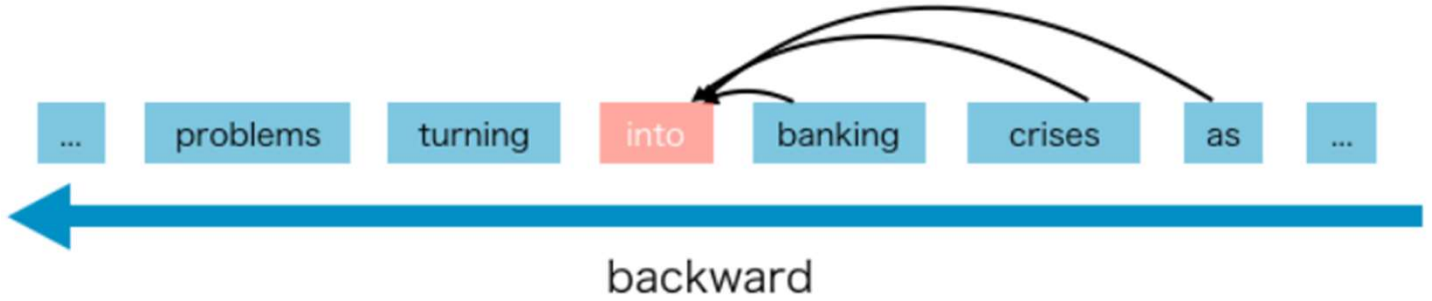
# Transformers

## XLNET: Review of AR and AE LMs

- AR LM → performs pretraining by maximizing likelihood under the forward AR factorization

$$\max_{\theta} \quad \log p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)}$$



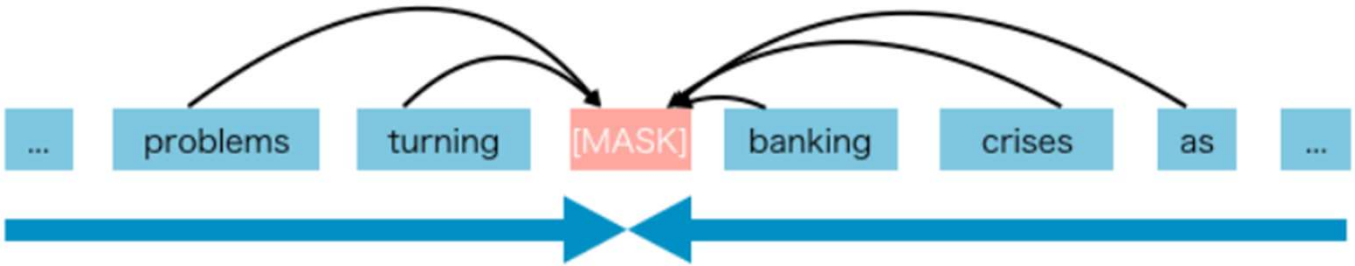



# Transformers

## XLNET: Review of AR and AE LMs

- BERT LM → performs pretraining by via denoising auto-encoding

$$\max_{\theta} \quad \log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)}$$




Bi-directional

# Transformers

## XLNET: Review of AR and AE LMs

- AR LM → performs pretraining by maximizing likelihood under the forward AR factorization

$$\max_{\theta} \quad \log p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)}$$

- BERT LM → performs pretraining by via denoising auto-encoding

$$\max_{\theta} \quad \log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)}$$

# Transformers

## XLNET: Review of AR and AE LMs

- AR LM → performs pretraining by maximizing likelihood under the forward AR factorization

$$\max_{\theta} \boxed{\log p_\theta(\mathbf{x}) =} \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)}$$

- BERT LM → performs pretraining by via denoising auto-encoding

$$\max_{\theta} \boxed{\log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx} \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)}$$

- **Independence assumption**

# Transformers

## XLNET: Review of AR and AE LMs

- AR LM → performs pretraining by maximizing likelihood under the forward AR factorization

$$\max_{\theta} \log p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)}$$

- BERT LM → performs pretraining by via denoising auto-encoding

$$\max_{\theta} \log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)}$$

- **Independence assumption**
- **Input noise**

# Transformers

## XLNET: Review of AR and AE LMs

- AR LM → performs pretraining by maximizing likelihood under the forward AR factorization

$$\max_{\theta} \log p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta(x_t \mid \mathbf{x}_{<t}) = \sum_{t=1}^{T} \log \frac{\exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x_t)\right)}{\sum_{x'} \exp\left(h_\theta(\mathbf{x}_{1:t-1})^\top e(x')\right)}$$

- BERT LM → performs pretraining by via denoising auto-encoding

$$\max_{\theta} \log p_\theta(\bar{\mathbf{x}} \mid \hat{\mathbf{x}}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{\mathbf{x}}) = \sum_{t=1}^{T} m_t \log \frac{\exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x_t)\right)}{\sum_{x'} \exp\left(H_\theta(\hat{\mathbf{x}})_t^\top e(x')\right)}$$

- **Independence assumption**
- **Input noise**
- **Context dependency**

# Transformers

## XLNET: Permutation Language Modeling

# Transformers

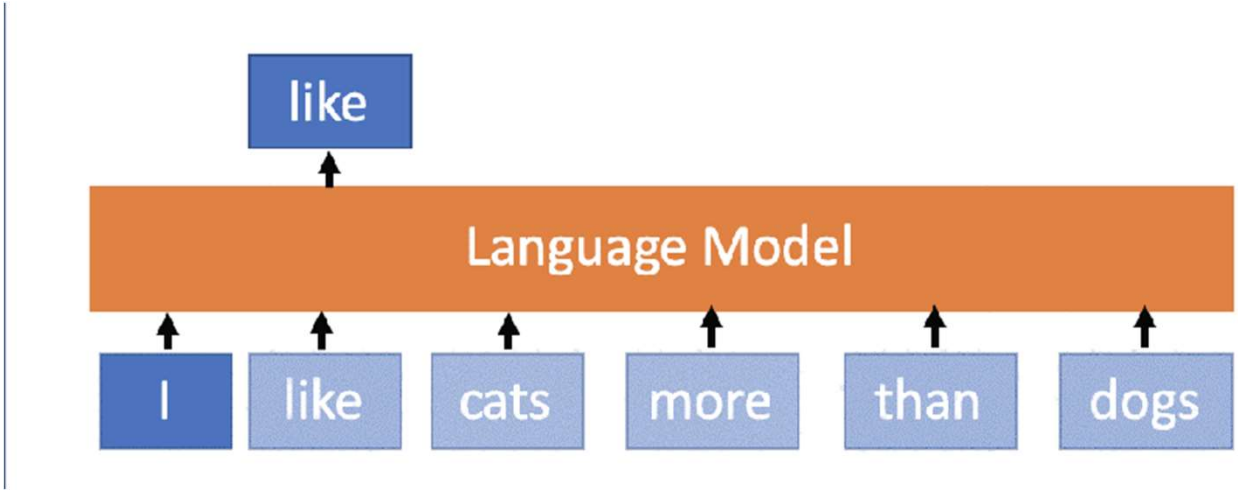
## XLNET: Permutation Language Modeling

- Issue with BERT LM
  - "I went to [MASK] [MASK] and saw the [MASK] [MASK] MASK]."
  - "I went to New York and saw the Empire State building." → valid
  - "I went to San Francisco and saw the Golden Gate bridge." → valid
  - "I went to San Francisco and saw the Empire State building." → not valid
- Permutation LMs
  - Predict one token given preceding context as in traditional AR LMs
  - However, token is predicted in some randomized order
  - "I like fishes because they're so **delicious**" → want to predict "delicious"
  - "**I like fishes because they're so [unk]**" → standard AR model
  - "fishes **because so [unk]** like I they're" → permutation model
  - Benefit → takes forward and backward elements into the context without major architecture overhaul

# Transformers

## XLNET: Permutation Language Modeling

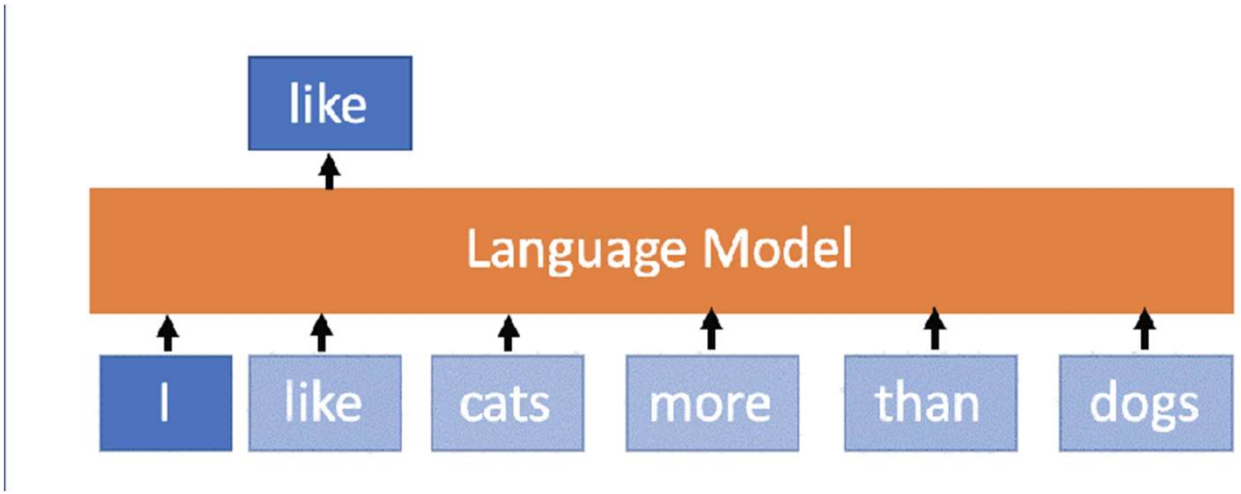
- Traditional AR LM



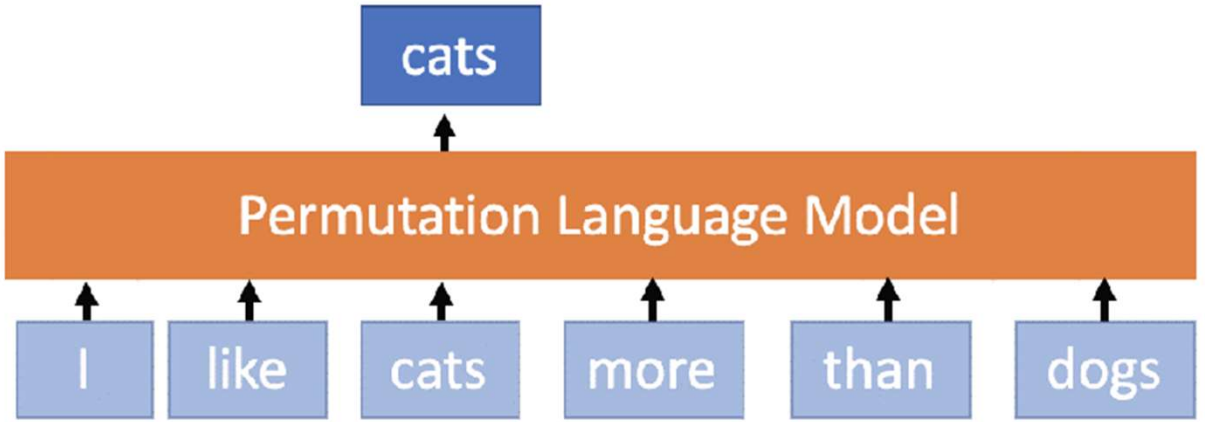
# Transformers

## XLNET: Permutation Language Modeling

- Traditional AR LM



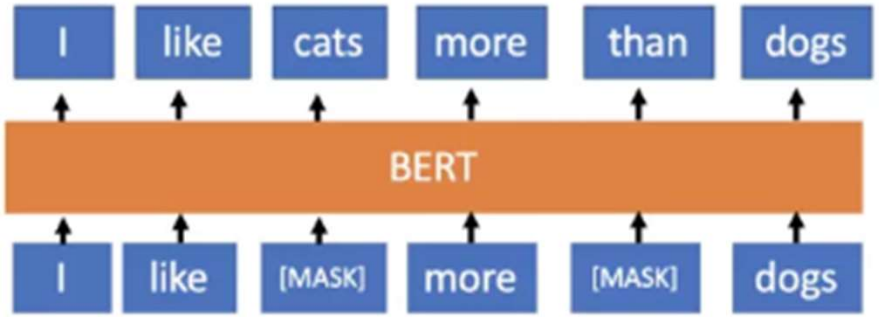

- Permutation AR LM



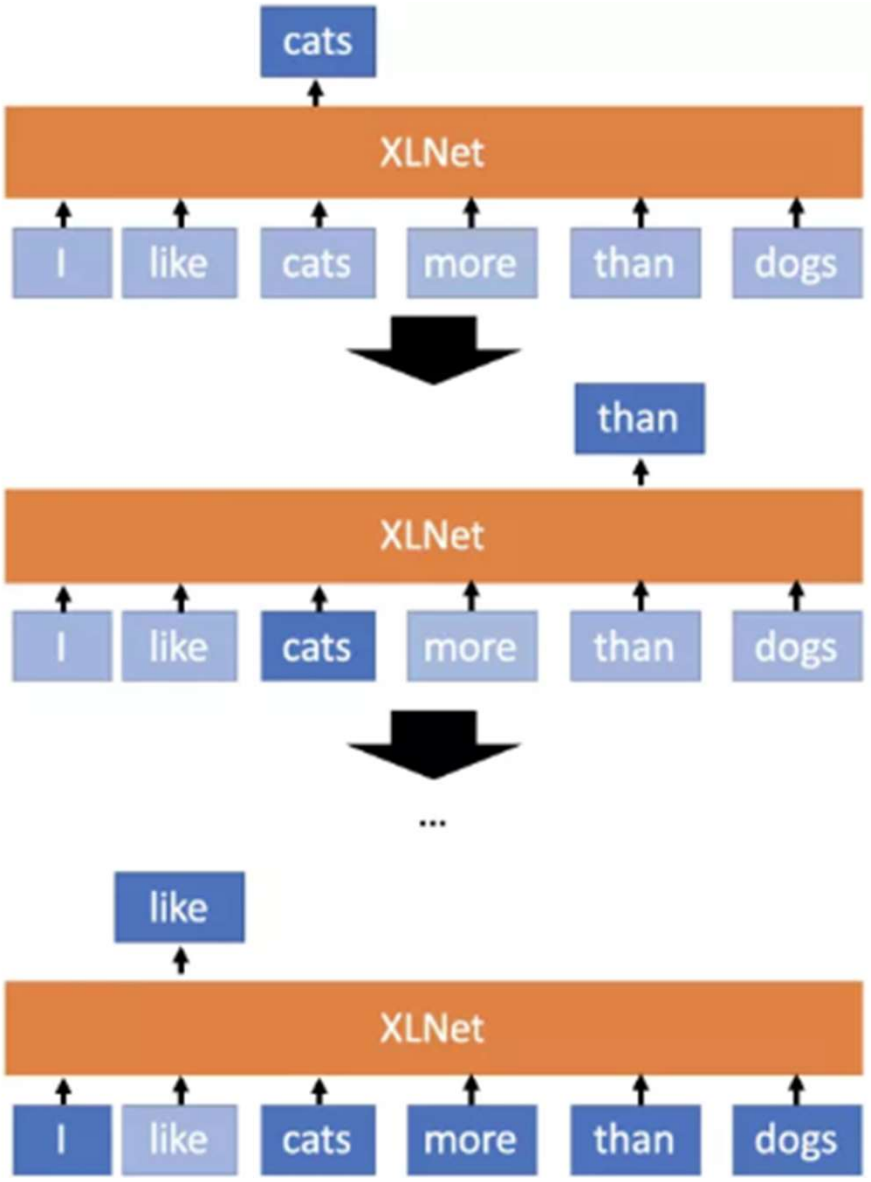
# Transformers

## XLNET: Permutation Language Modeling

**BERT**




**XLNet**



# Transformers

## XLNET: Permutation Language Modeling

- Suppose we have a sequence of 4 words → **# of permutations** = 24
  - [x1, x2, x3, x4]
  - [x1, x2, x4, x3]
  - [x1, x4, x2, x3]
  - …
  - [x4, x3, x2, x1]
- Predict x3 given **x** = [x1, x2, x3, x4] → **# of factorizations** = 4
  - [x3, xx, xx, xx]
  - [xx, x3, xx, xx]
  - [xx, xx, x3, xx]
  - [xx, xx, xx, x3]
- For a sequence **x**, sample a factorization order from the set of all possible permutations and decompose the likelihood according to the factorization order

# Transformers

## XLNET: Permutation Language Modeling

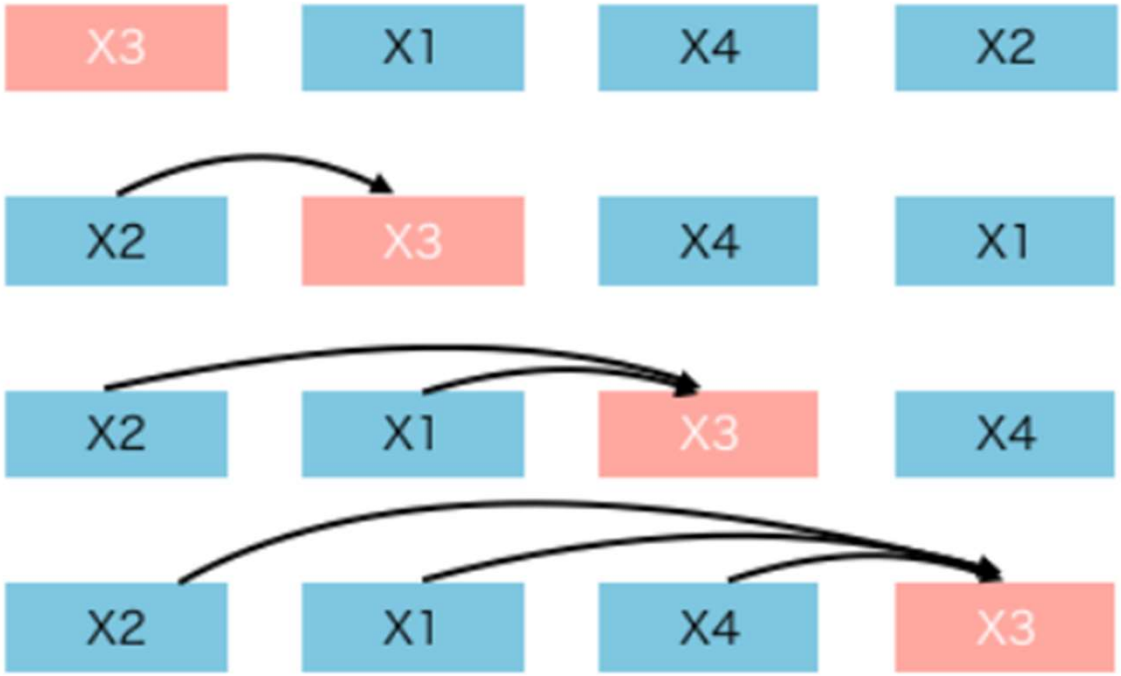
- For a sequence **x**, sample a factorization order from the set of all possible permutations and decompose the likelihood according to the factorization order
- Same model parameters are shared across all factorization orders during pre-training → in expectation, x3 will see every possible element x_i != x3 and bi-directional context is learned

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[ \sum_{t=1}^{T} \log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}<t}) \right]$$

- $Z_T$: Set of all possible permutations
- **z**: One factorization order

# Transformers

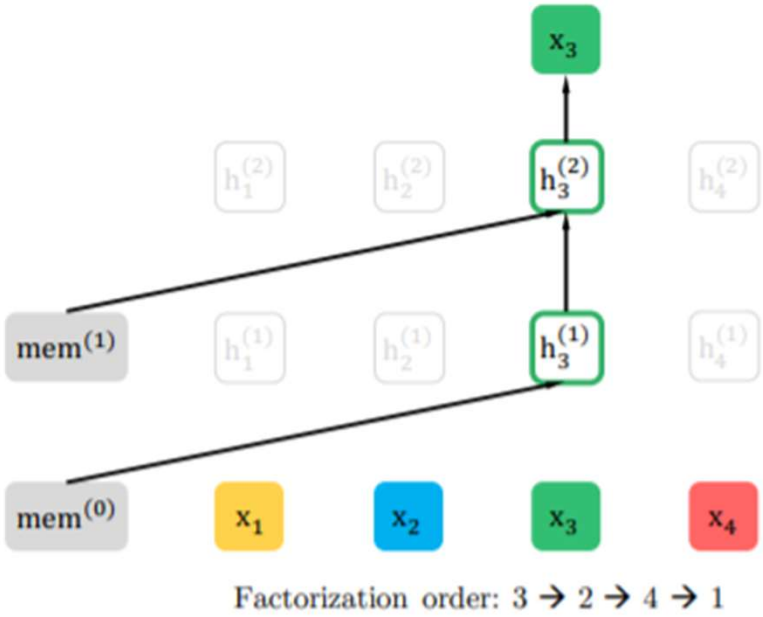
## XLNET: Permutation Language Modeling




- ***Sequence*** order is not changed, only the factorization order
  - Sequence order is kept by using relative PEs corresponding to the original sequence
  - Attention mask is used to achieve permutation of the factorization order
  - Important since text sequences during fine-tuning are in natural order

# Transformers

## XLNET: Permutation Language Modeling

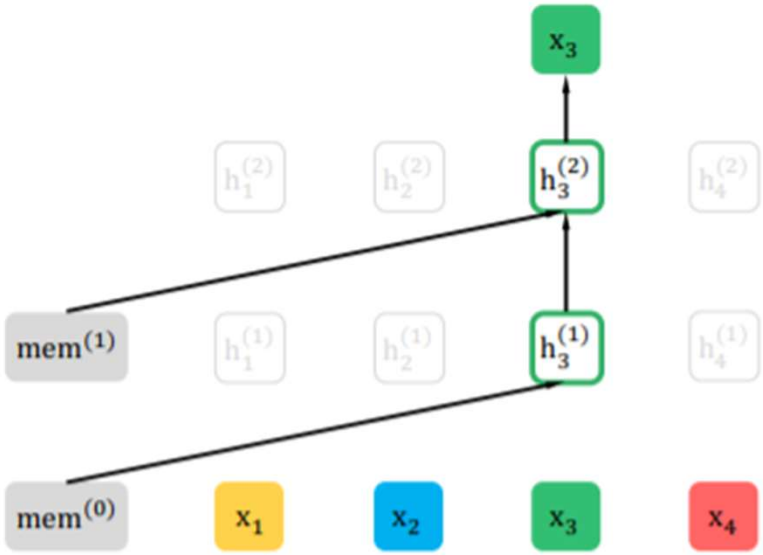


Factorization order: 3 → 2 → 4 → 1

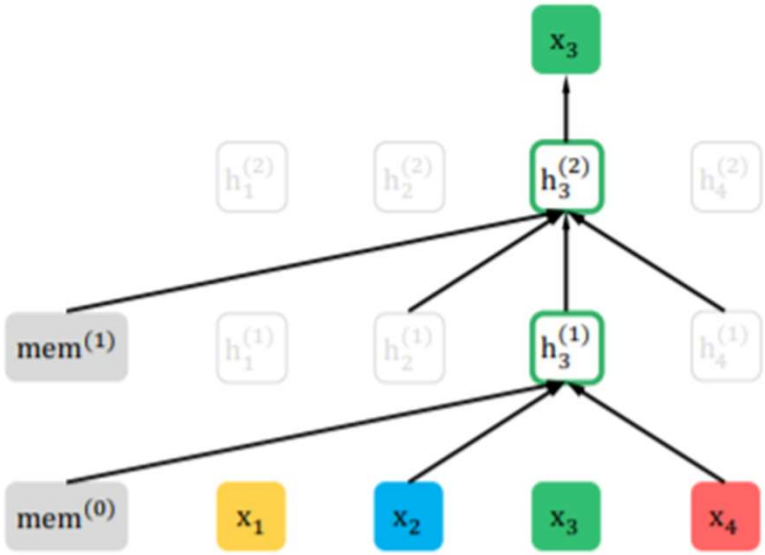
- Predict x3 in this factorization order
  - No other tokens before it → use memory state as context
- Not likely to succeed since there is no context
  - Skip during pre-training

# Transformers

## XLNET: Permutation Language Modeling




Factorization order: 3 → 2 → 4 → 1

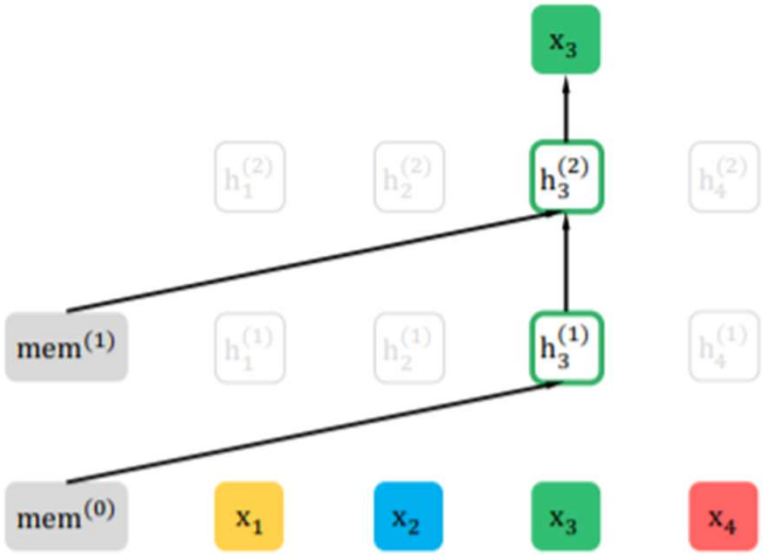



Factorization order: 2 → 4 → 3 → 1

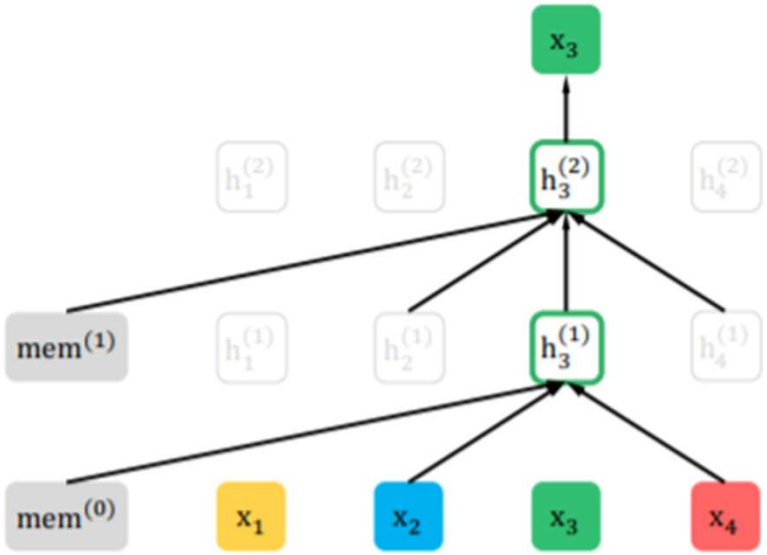
- This permutation has sufficient context for predicting x3
  - Use during pre-training

# Transformers

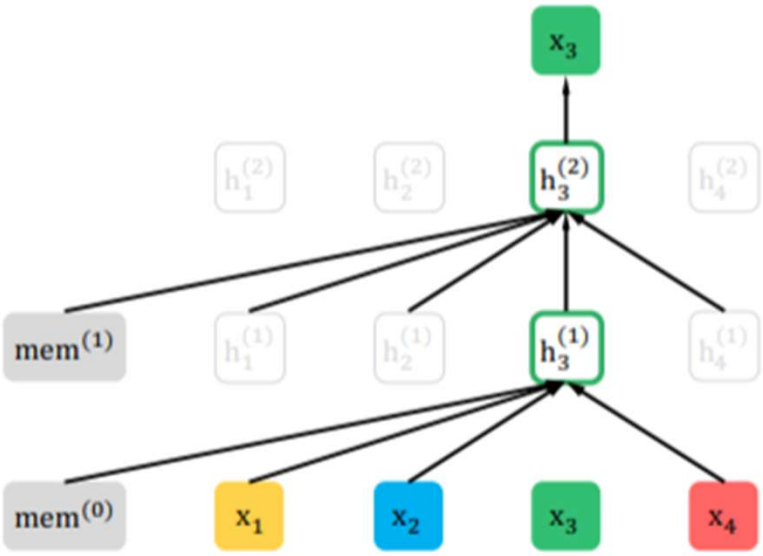
## XLNET: Permutation Language Modeling




Factorization order: 3 → 2 → 4 → 1




Factorization order: 2 → 4 → 3 → 1

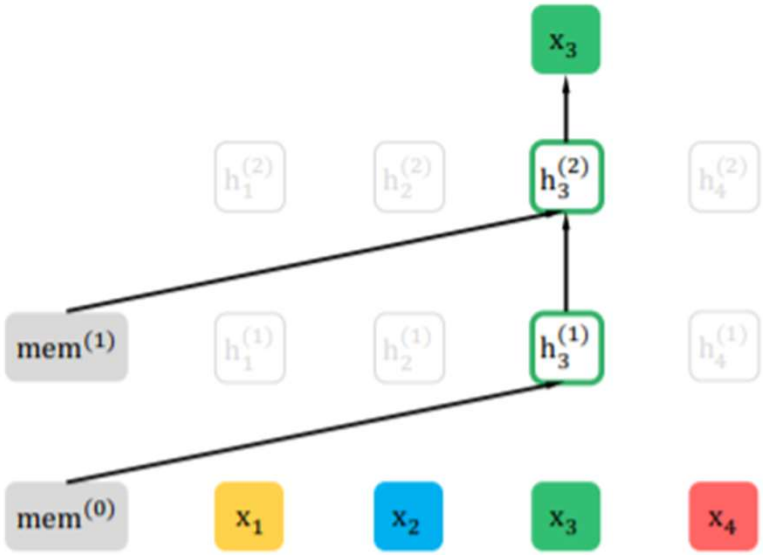



Factorization order: 1 → 4 → 2 → 3

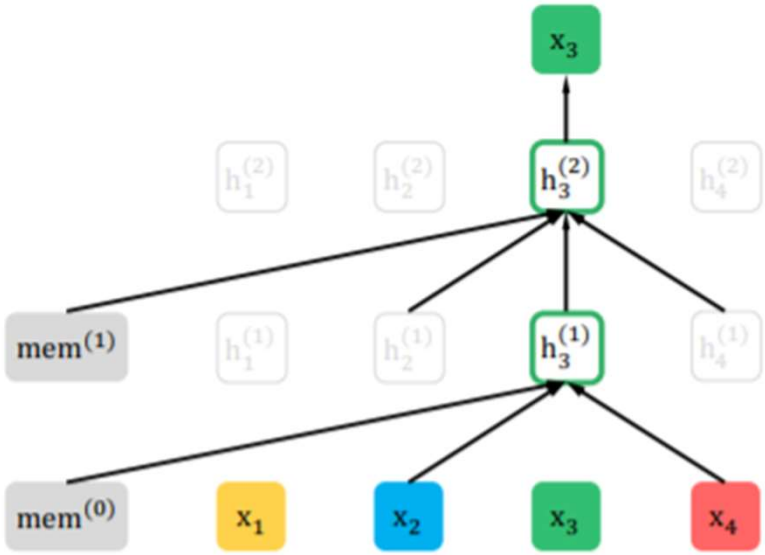
- This permutation also has sufficient context for predicting x3
  - Use during pre-training

# Transformers

## XLNET: Permutation Language Modeling

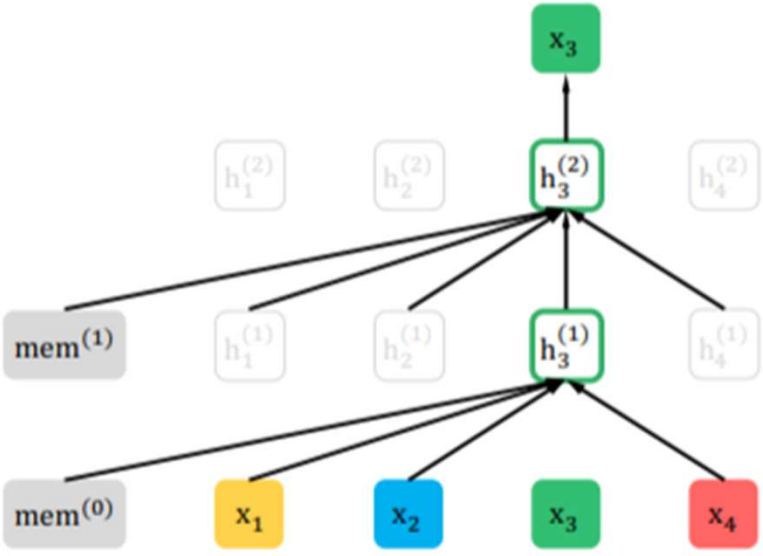



Factorization order: 3 → 2 → 4 → 1

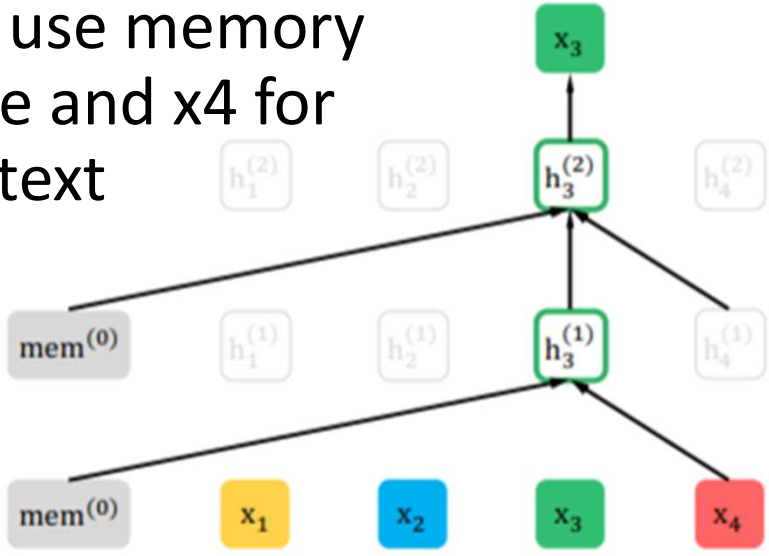



Factorization order: 2 → 4 → 3 → 1

- This permutation can use memory state and x4 for context




Factorization order: 1 → 4 → 2 → 3




Factorization order: 4 → 3 → 1 → 2

# Transformers

## XLNET: Two-Stream Self-Attention

# Transformers

## XLNET: Two-Stream Self-Attention

- Why do we need two streams of self-attention?
- “The cat sat and jumped.”
  - Given “The cat” as context, one permutation can be “sat”|”The cat”
  - Given “The cat” as context, another permutation can be “jumped|The cat”
  - Two different target positions share the same model prediction!
- Technical challenge
  - Each input, [x1, x2, x3, x4], has **position and embedding information added together**
  - However, model can’t use position info. to predict x3 without also using the embedding info. → model will learn to copy trivially
  - Model can’t know the embedding info. of the word it’s trying to predict
  - But it needs to know the position of the word it’s trying to predict…
- So, what do?

# Transformers

## XLNET: Two-Stream Self-Attention

- So, what do? → Let's have another attention stream that just has the positional info.
- **Content Stream**
  - Includes both positional and token embeddings (initialized to the embedding vector for the input word)
  - Provides input to Query Stream
  - Same as regular self-attention except attention mask is permutation dependent
- **Query Stream**
  - Includes positional embeddings and masks token embeddings (initialized to a random vector)
  - Only used for pre-training
  - Takes input from Content Stream
  - Prediction is performed using info. only from Query Stream
- During fine-tuning, only Content Stream is used for text representation

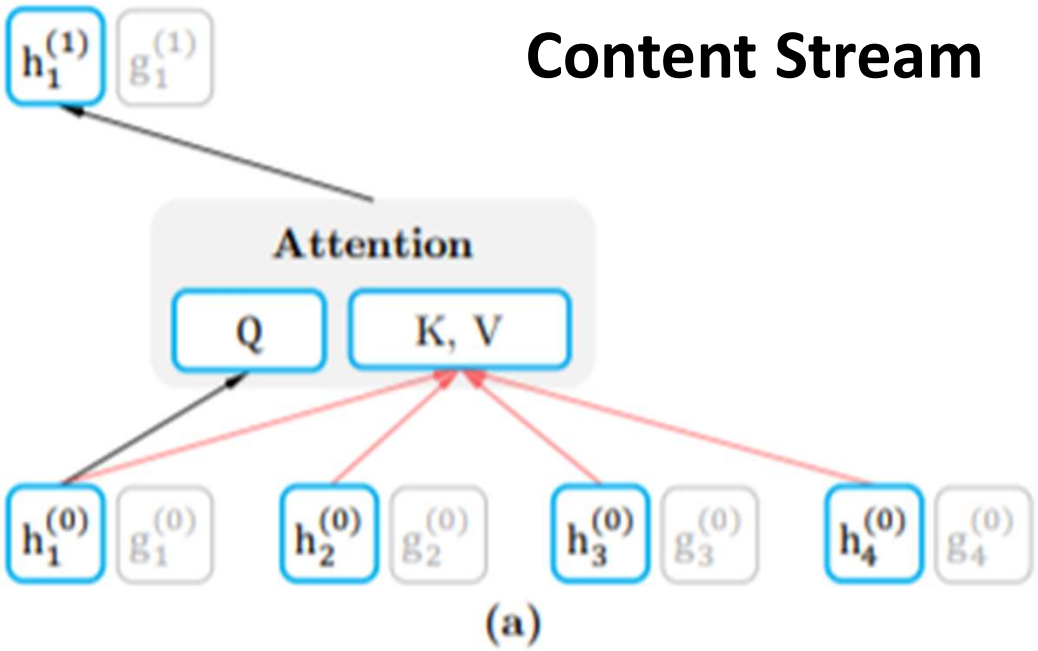
# Transformers

## XLNET: Two-Stream Self-Attention

- “I like cats more than dogs.” → predict “like” given “more” and “dogs”
  - Content Stream → encode position and token info. for “more” and “dogs”
  - Query Stream → encode position info. for “like” and the Content Stream info. for predicting “like”

# Transformers

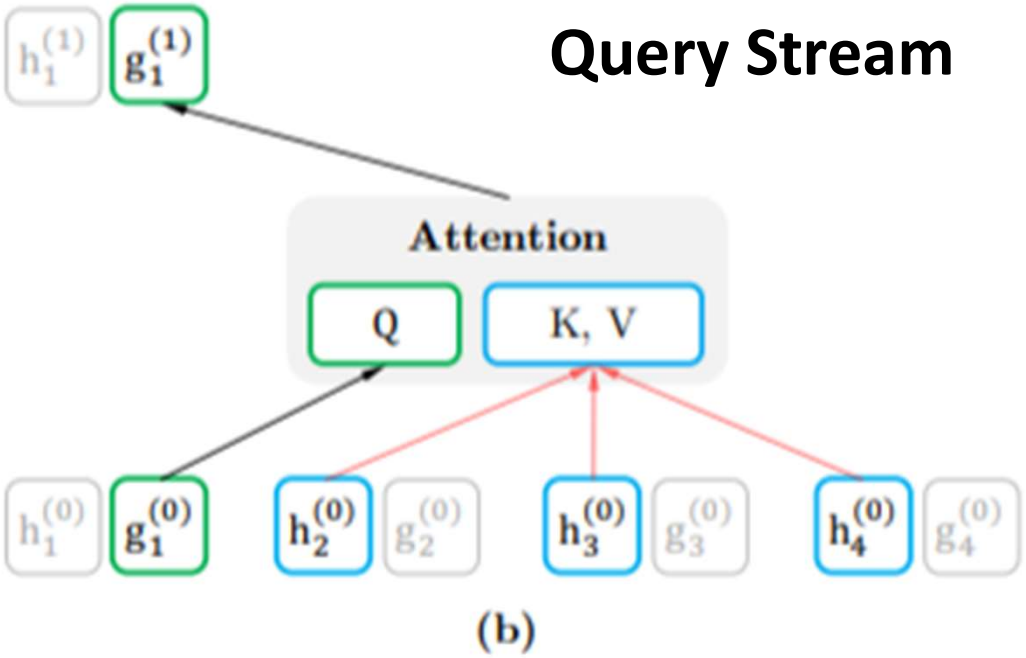
## XLNET: Two-Stream Self-Attention




- Same as standard self-attention
- h1 is the Query, [h1, h2, h3, h4] are the Keys/Values

# Transformers

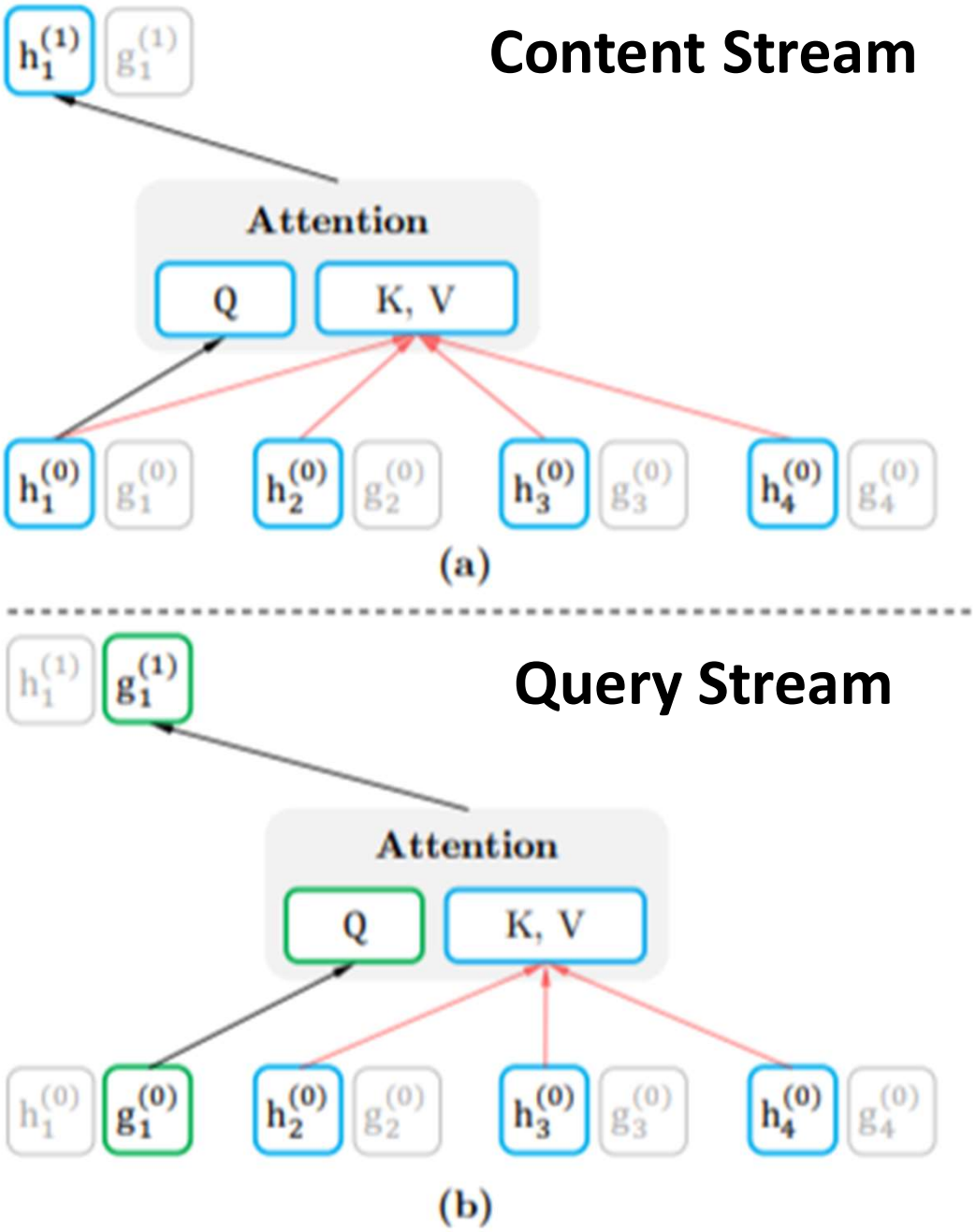
## XLNET: Two-Stream Self-Attention




- Does not have access to h1 for self-attention, only g1
- g1 is the Query and contains only positional info.
- [h2, h3, h4] are the Keys/Values and contain position and token info.

# Transformers

## XLNET: Two-Stream Self-Attention

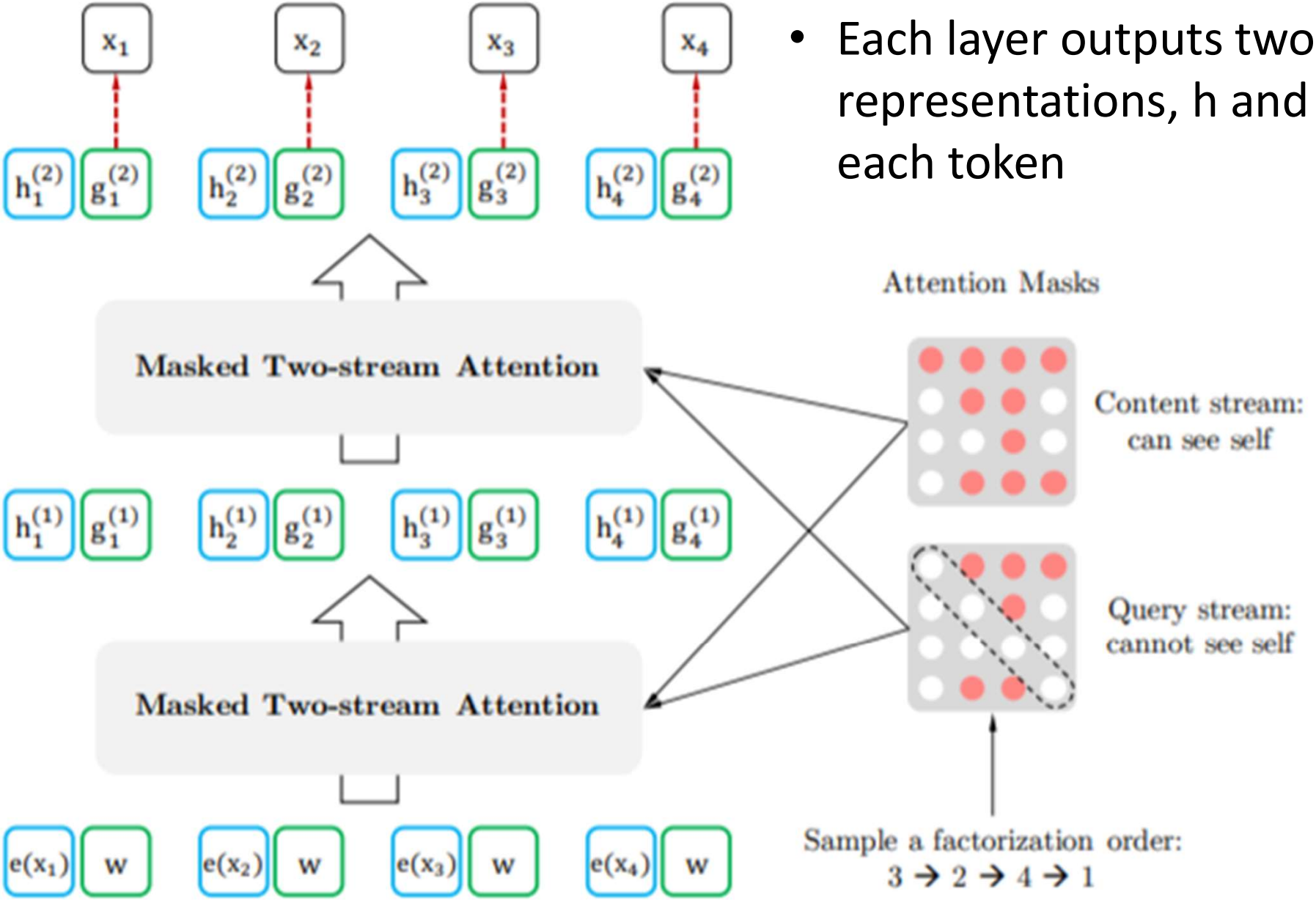



- Each layer outputs two representations, h and g, for each token

# Transformers

## XLNET: Two-Stream Self-Attention

- Each layer outputs two representations, h and g, for each token




(c)

# Transformers

## XLNET: Two-Stream Self-Attention

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = g_{z_t}^{(m-1)}, \text{KV} = \mathbf{h}_{\mathbf{z}_{<t}}^{(m-1)}; \theta), \quad \text{(query stream: use } z_t \text{ but cannot see } x_{z_t})$$

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = h_{z_t}^{(m-1)}, \text{KV} = \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}; \theta), \quad \text{(content stream: use both } z_t \text{ and } x_{z_t}).$$

- Query Stream, g
  - Contains embedding info ***up to t***
  - Only use position info. at t
  - Keys/Values come from Content Stream
- Content Stream, h
  - Contains embedding and position info ***up to and including t***
  - Generates Keys/Values for input to Query Stream
- At model input
  - g is some trainable parameter *w*
  - h is embedding of token (also trainable)

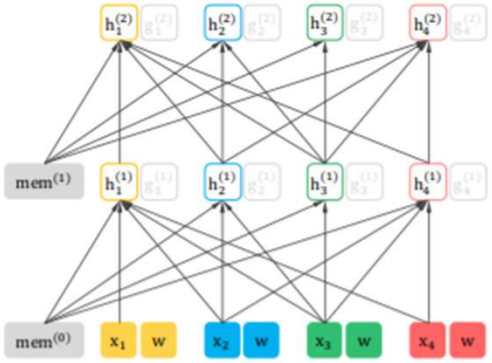
# Transformers

## XLNET: Model Architecture

# Transformers

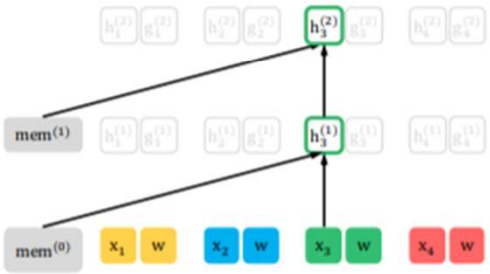
## XLNET: Model Architecture

**Content Stream**

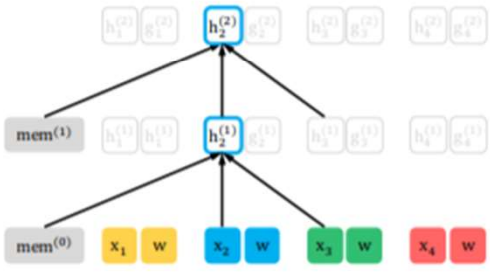



Joint View of the Content Stream
(Factorization order: 3 → 2 → 4 → 1)

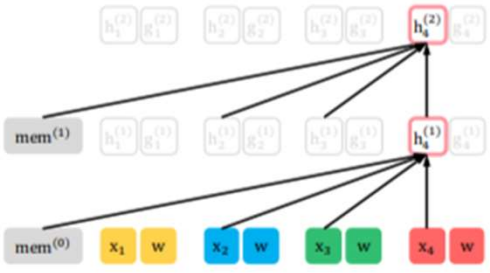
Split View



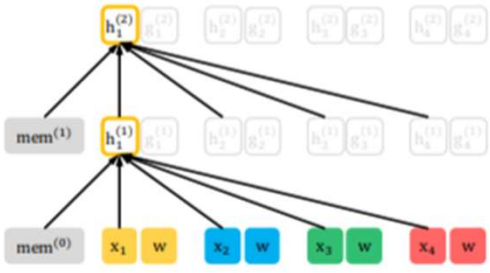

Position-3 View




Position-2 View




Position-4 View



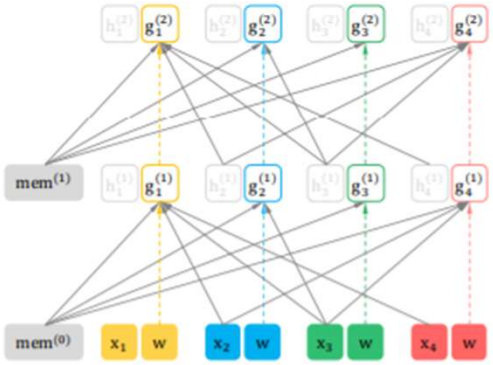

Position-1 View

Split View of the Content Stream
(Factorization order: 3 → 2 → 4 → 1)

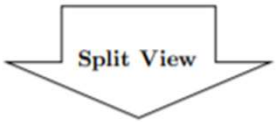
# Transformers

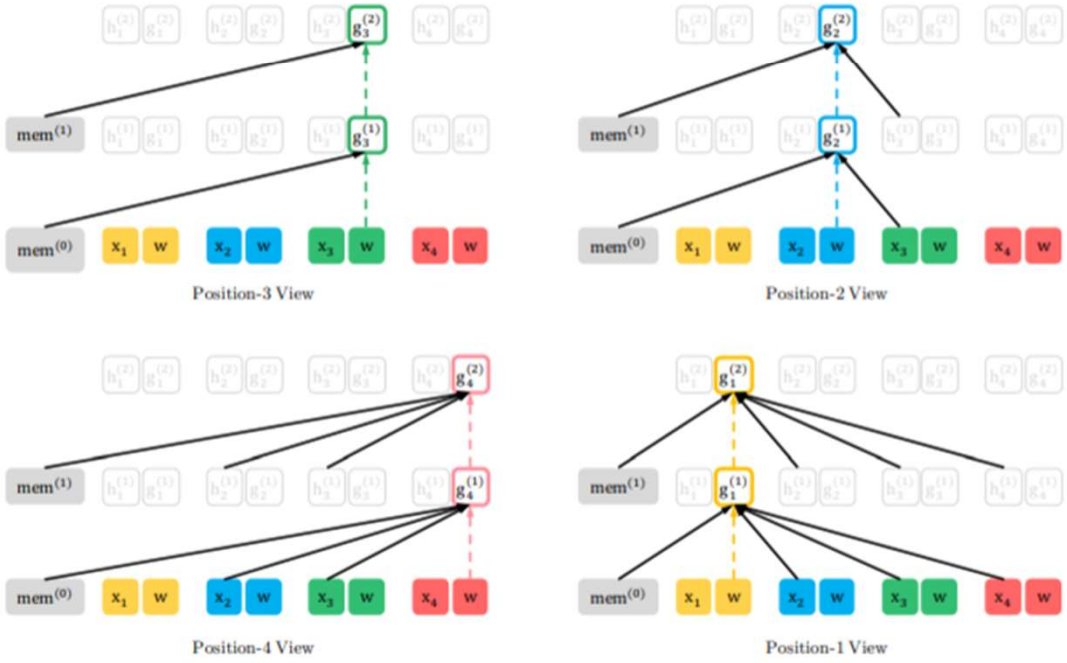
## XLNET: Model Architecture

**Query Stream**




Joint View of the Query Stream
(Factorization order: 3 → 2 → 4 → 1)




Position-3 View

Position-2 View

Position-4 View

Position-1 View

Split View of the Query Stream
(Factorization order: 3 → 2 → 4 → 1)

# Transformers

## XLNET: Partial Prediction

# Transformers

## XLNET: Partial Prediction

- Pretraining is a challenging optimization problem due to permutation
  - Caused slow convergence in preliminary experiments
- To reduce optimization difficulty → only predict the last tokens in a factorization order
- For a permutation sequence **z**, split **z** into a non-target subsequence and a target subsequence

# Transformers

## XLNET: Partial Prediction

- Pretraining is a challenging optimization problem due to permutation
  - Caused slow convergence in preliminary experiments
- To reduce optimization difficulty → **only predict the last tokens in a factorization order since these will have sufficient context**
- For a permutation sequence **z**, split **z** into a non-target subsequence and a target subsequence
  - Modified objective is to maximize log-likelihood of the *target* subsequence conditioned on the non-target subsequence
  - Split point ("c") is a hyperparameter
  - **z**_>c is chosen as target because it contains the longest context in the sequence given a factorization order **z**

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=1}^{T}\log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]$$

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\log p_\theta(\mathbf{x}_{\mathbf{z}_{>c}} \mid \mathbf{x}_{\mathbf{z}_{\leq c}})\right] = \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=c+1}^{|\mathbf{z}|}\log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right]$$

# Transformers

## XLNET: Incorporating Ideas from Transformer-XL

# Transformers

## XLNET: Incorporating Ideas from Transformer-XL

- Integrates two important techniques from Transformer-XL
- Relative positional encoding
  - Required since XLNet is bi-directional
  - Applied to **original** (i.e., not permutated) input sequence
- Segment recurrence mechanism
  - Enables model to reuse hidden states from previous segments

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\text{Q} = h_{z_t}^{(m-1)}, \text{KV} = \left[\tilde{\mathbf{h}}^{(m-1)}, \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}\right]; \theta)$$

  - Process first permutated segment and cache hidden layer vectors for each layer, *m*
  - Attention update is independent of the previous permutation order once the cached hidden layer vectors are computed
  - Permutation order does not have been to known in order to reuse previous segment
  - In expectation, model learns over all permutation orders of the previous segment

# Transformers

## XLNET: Relative Segment Encodings

# Transformers

## XLNET: Relative Segment Encodings

- Recall
  - BERT uses absolute segment embeddings → segment embeddings reset for each new segment
  - XLNet has recurrence → segment embeddings can no longer be absolute
- Relative segment encoding
  - Consider a pair of positions *i* and *j* in a sequence
  - If *i* and *j* are from the same segment (i.e., same context), use one encoding
  - If *i* and *j* are from a different segment (i.e., different context), use another encoding
  - For each attention *head*, these two segment encodings are learnable parameters
  - **Intuition**: only care whether two positions are ***within the same segment*****,** rather than ***which specific segments they are from***

# Transformers

## XLNET: Relative Segment Encodings

- Two benefits of using relative segment encodings
  - Better generalization
  - Allows fine-tuning on tasks that have more than two input segments (not possible using absolute segment encodings)
- Implementation
  - Compute a separate attention score for segment encodings and add to regular attention score

When $i$ attends to $j$, the segment encoding $\mathbf{s}_{ij}$ is used to compute an attention weight $a_{ij} = (\mathbf{q}_i + \mathbf{b})^\top \mathbf{s}_{ij}$, where $\mathbf{q}_i$ is the query vector as in a standard attention operation and $\mathbf{b}$ is a learnable head-specific bias vector. Finally, the value $a_{ij}$ is added to the normal attention weight.

# Transformers

## XLNET: Pretraining XLNet

# Transformers

## XLNET: Pretraining XLNet

- During pre-training
  - Input to model is: [A, SEP, B, SEP, CLS]
  - Randomly sample two segments (either from same context or not)
  - Concatenate two segments as one sequence and perform permutation language modeling
  - Only reuse memory from previous segment if two segments are from the same context
  - XLNet-Large does **not** do NSP task since it does not show consistent improvement in ablation studies
  - Use SentencePiece tokenization

# Transformers

## XLNET: Pretraining XLNet

- During pre-training
  - **After data pruning, trained on 32.89B subword pieces total → 10X larger than BERT (3.3B)***
  - **Used 512 TPUs and took about 2.5 days**
  - XLNet-Large still underfit data (low model capacity) at end of training but more training did not help during fine-tuning (so no practical point in larger models)
  - Bi-directional data input pipeline → decides factorization ***direction*** and allows XLNET to apply idea of "masking" future positions
  - Span-based prediction → Randomly select a consecutive span of $L$ (1 – 5) tokes for prediction, with context of ($KL$) tokens ($K$ = 6)

* However, see https://medium.com/@xlnet.team/a-fair-comparison-study-of-xlnet-and-bert-with-large-models-5a4257f59dc0

# Transformers

## XLNET: Ablation Studies

# Transformers

## XLNET: Ablation Studies

| # | Model | RACE | SQuAD2.0 F1 | SQuAD2.0 EM | MNLI m/mm | SST-2 |
|---|---|---|---|---|---|---|
| 1 | BERT-Base | 64.3 | 76.30 | 73.66 | 84.34/84.65 | 92.78 |
| 2 | DAE + Transformer-XL | 65.03 | 79.56 | 76.80 | 84.88/84.45 | 92.60 |
| 3 | XLNet-Base ($K = 7$) | 66.05 | **81.33** | **78.46** | **85.84/85.43** | 92.66 |
| 4 | XLNet-Base ($K = 6$) | 66.66 | 80.98 | 78.18 | 85.63/85.12 | **93.35** |
| 5 | - memory | 65.55 | 80.15 | 77.27 | 85.32/85.05 | 92.78 |
| 6 | - span-based pred | 65.95 | 80.61 | 77.91 | 85.49/85.02 | 93.12 |
| 7 | - bidirectional data | 66.34 | 80.65 | 77.87 | 85.31/84.99 | 92.66 |
| 8 | + next-sent pred | **66.76** | 79.83 | 76.94 | 85.32/85.09 | 92.89 |

Table 6: Ablation study. The results of BERT on RACE are taken from [39]. We run BERT on the other datasets using the official implementation and the same hyperparameter search space as XLNet. $K$ is a hyperparameter to control the optimization difficulty (see Section 2.3). All models are pretrained on the same data.

- Ablation studies
  - Effectiveness of permutation LM vs. BERT
  - Using Transformer-XL backbone and using segment-level recurrence
  - Span-based prediction, bi-directional input pipeline, and NSP
- TLDR
  - Need everything in order for XLNet to be SOTA except NSP

# What’s Next in NLP?

## (Textual) Q&A Systems

# What's Next in NLP?

## (Textual) Q&A Systems

- Data is growing at an exponential rate in our world
  - Simply returning relevant documents (i.e., web search) is of limited use
- Instead, we want **answers** to our **questions**
  - Digital assistants do this to some extent (e.g., Alexa, Google Assistant, etc.)
- Q&A systems can be factored into two parts
  - Finding relevant documents that might contain the answer → can be handled by traditional information retrieval/web search algorithms
  - Finding an answer in the documents → **reading comprehension** problem

**Stanford Question Answering Dataset (SQuAD)**

(Rajpurkar et al., 2016)

**Question:** Which team won Super Bowl 50?

**Passage**

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California.

100k examples

Answer must be a span in the passage

A.k.a. extractive question answering

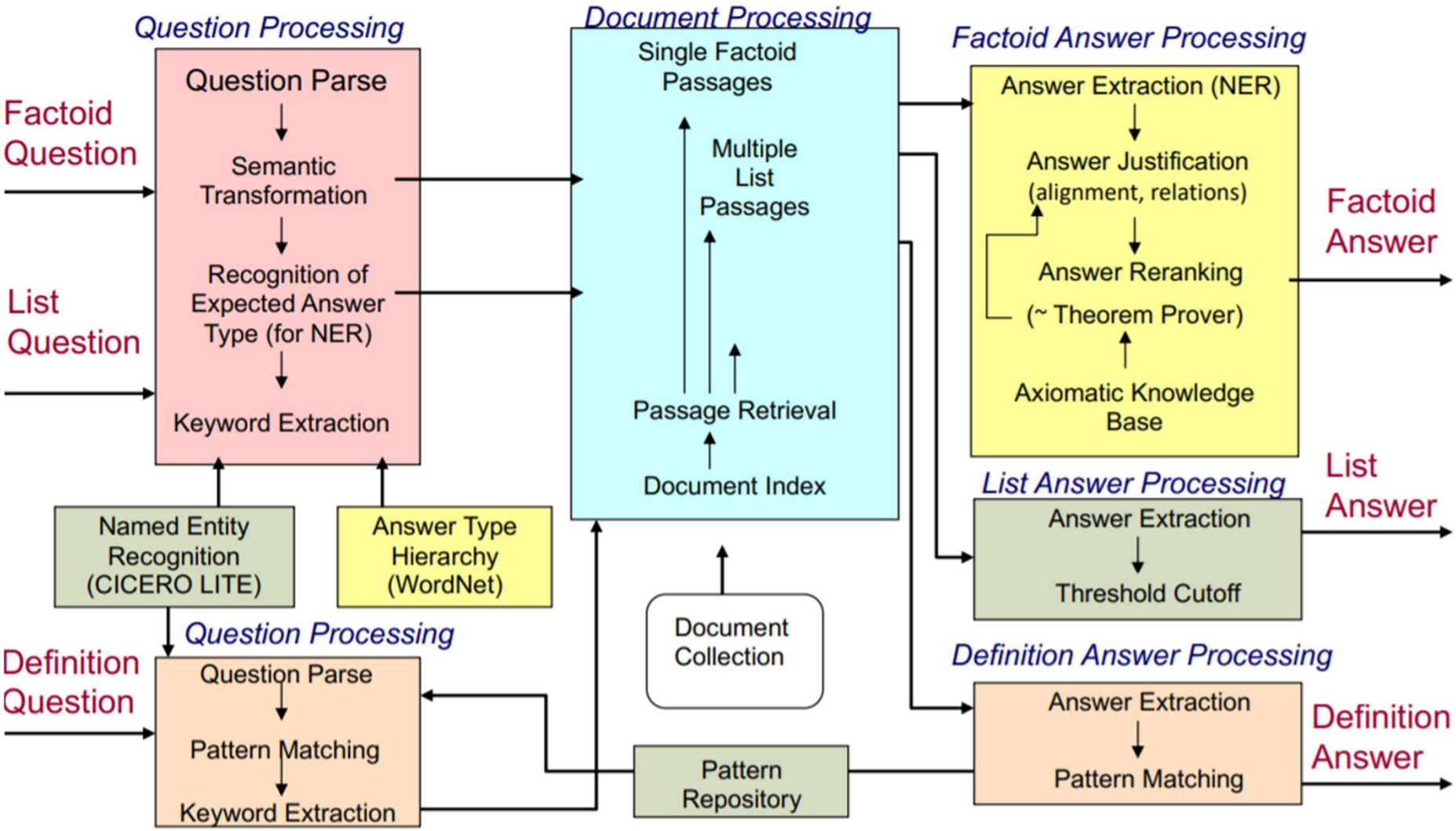
# What's Next in NLP?

## (Textual) Q&A Systems

### Turn-of-the Millennium Full NLP QA:

**[architecture of LCC (Harabagiu/Moldovan) QA system, circa 2003]**

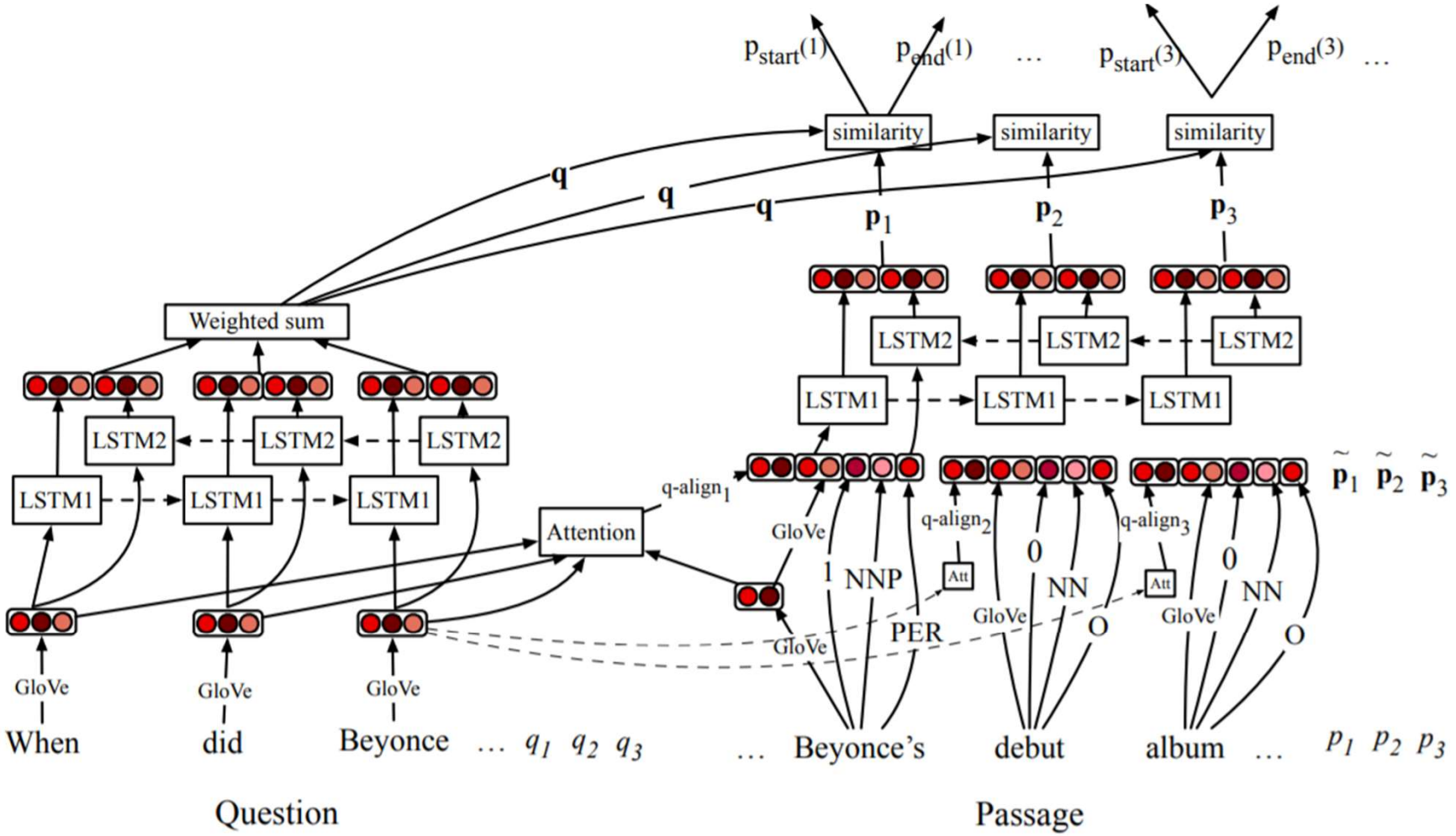
**Complex systems but they did work fairly well on "factoid" questions**



# What's Next in NLP?

## (Textual) Q&A Systems

- Stanford Attentive Reader++ (Bi-directional LSTM with Attention)
  - Minimal, successful architecture for reading comprehension and Q&A




Training objective: $\mathcal{L} = -\sum \log P^{(\text{start})}(a_{\text{start}}) - \sum \log P^{(\text{end})}(a_{\text{end}})$

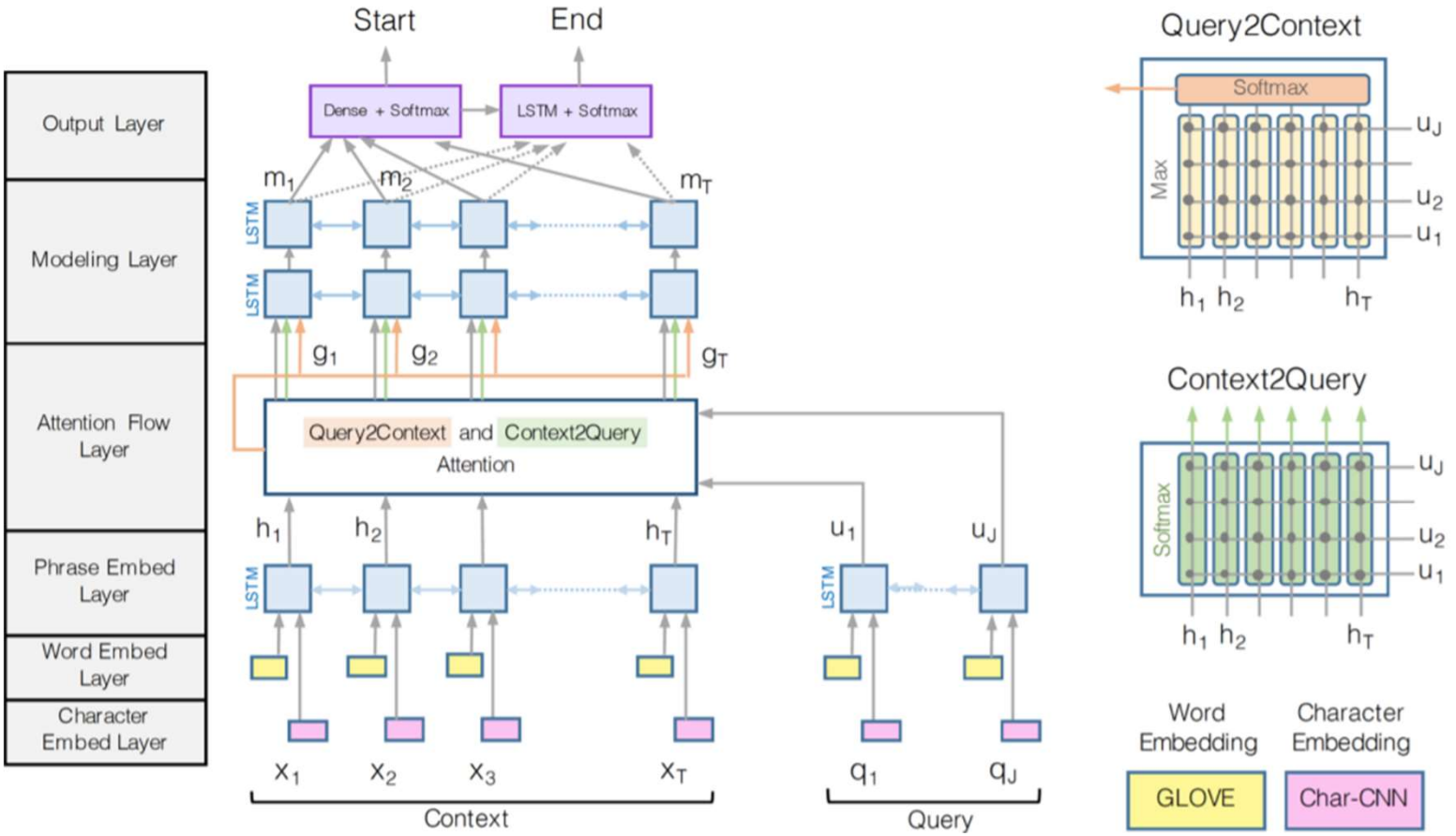
# What's Next in NLP?

## (Textual) Q&A Systems

- BiDAF → uses Attention Flow layer
  - Attention should flow both ways – from the context to the question and from the question to the context

**5. BiDAF: Bi-Directional Attention Flow for Machine Comprehension**

**(Seo, Kembhavi, Farhadi, Hajishirzi, ICLR 2017)**



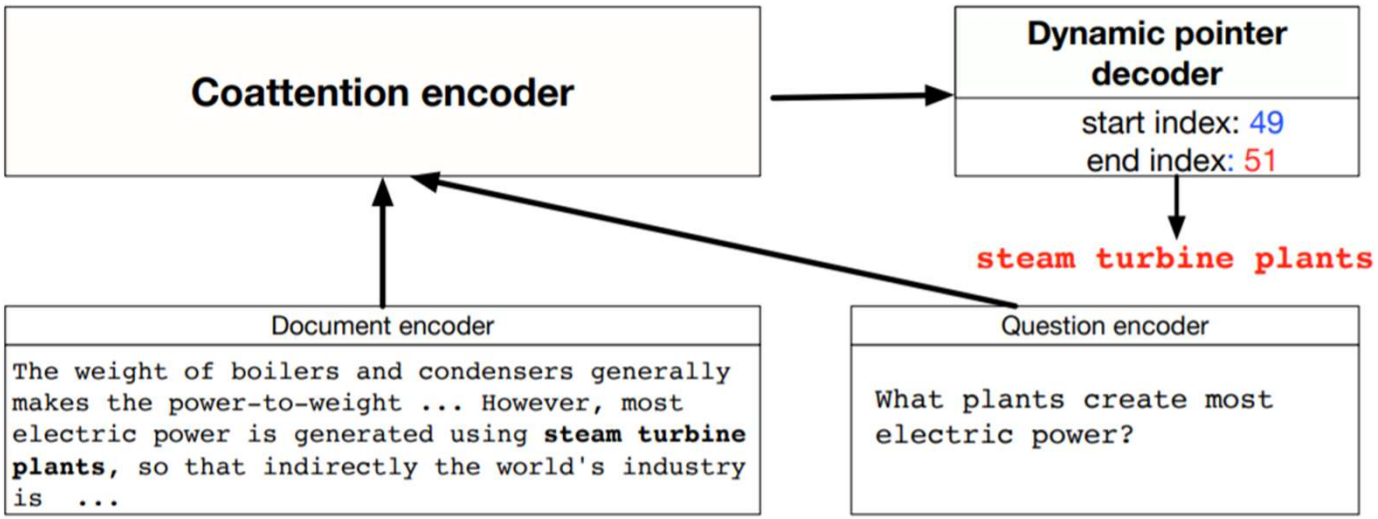
# What's Next in NLP?

## (Textual) Q&A Systems

- Coattention Decoder
  - Two-way attention between context and question as in BiDAF
  - Coattention → Attending over attention representations

**Dynamic Coattention Networks for Question Answering**
**(Caiming Xiong, Victor Zhong, Richard Socher ICLR 2017)**

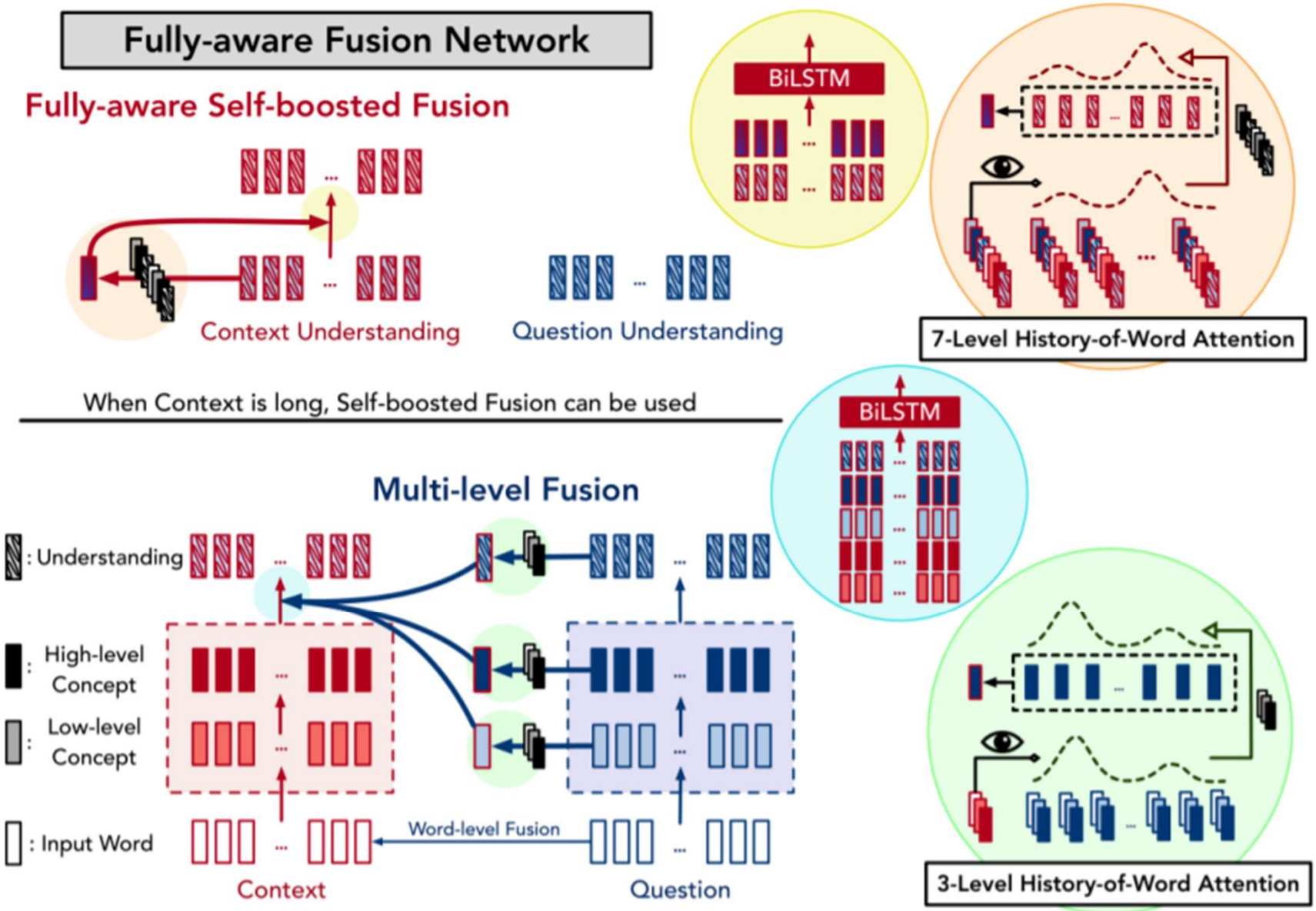
- Flaw: Questions have input-independent representations
- Interdependence needed for a comprehensive QA model



# What's Next in NLP?

## (Textual) Q&A Systems

- FusionNet → combine many forms of attention



# What's Next in NLP?

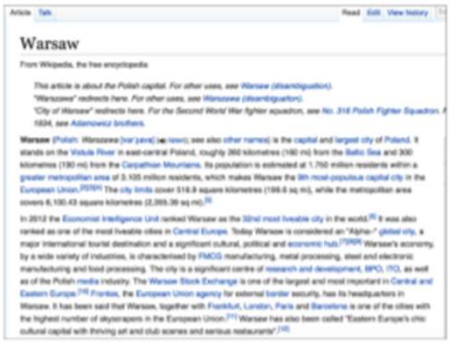
## (Textual) Q&A Systems

## DrQA: Open-domain Question Answering

**(Chen, et al. ACL 2017) https://arxiv.org/abs/1704.00051**

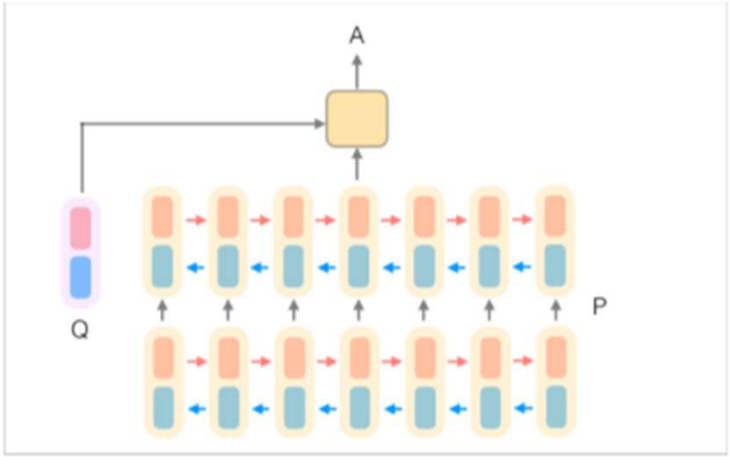
Q: How many of Warsaw's inhabitants spoke Polish in 1933?


**Document Retriever**




**Document Reader**

833,500



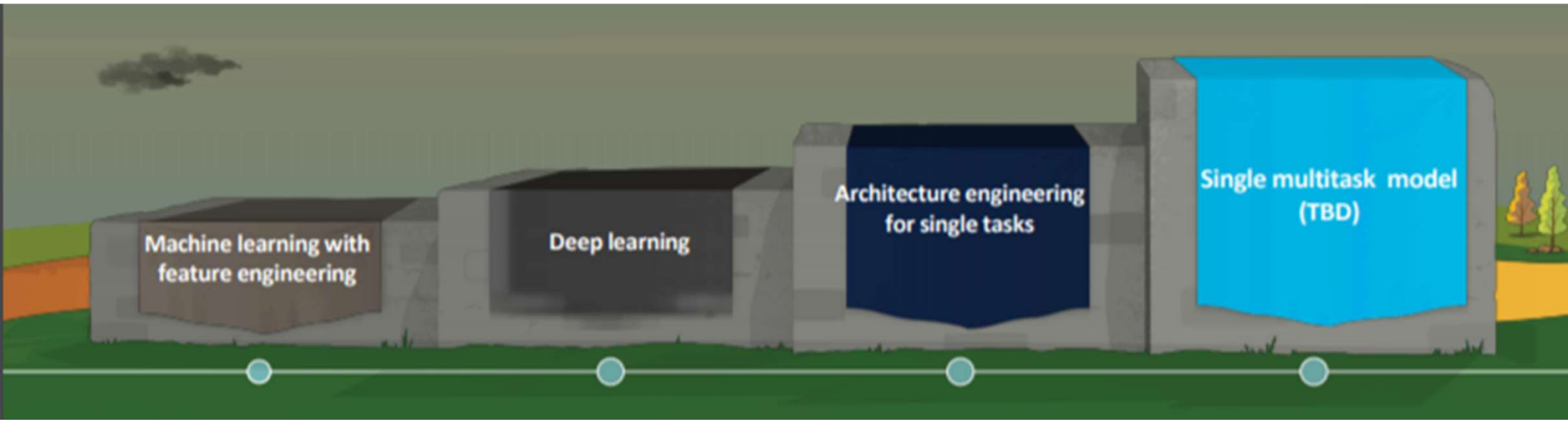
# What’s Next in NLP?

## Multi-Task Learning

# What's Next in NLP?

## Multi-Task Learning

- Limits of Single-Task Learning
  - Great performance improvements in recent years given {dataset, task, model, metric}
  - Can hill-climb to local optima as long as |dataset| > 1000 x Classes
  - For more general AI, need continuous learning in a single model
  - Models typically start from random or are only partly pre-trained



# What's Next in NLP?

## Multi-Task Learning

- Why hasn't weight and model sharing been prevalent in NLP?
  - NLP requires many types of reasoning (logical, linguistic, emotional, visual, etc.)
  - Requires short and long term memory
  - NLP has been divided into intermediate/separate tasks → benchmark chasing in individual communities
- Can a single unsupervised task solve everything? No.
  - Language will require supervision to some extent

# What's Next in NLP?

## Multi-Task Learning

- Why single model for multi-task learning in NLP?
  - Multi-task learning is a blocker for general NLP systems
- Unified models can decide how to transfer knowledge (domain adaptation, weight sharing, transfer and zero-shot learning)
  - More easily adapt to new tasks
  - Make deploying to production simpler
  - Lower the bar for more people to solve new tasks
  - Potentially move towards continual learning

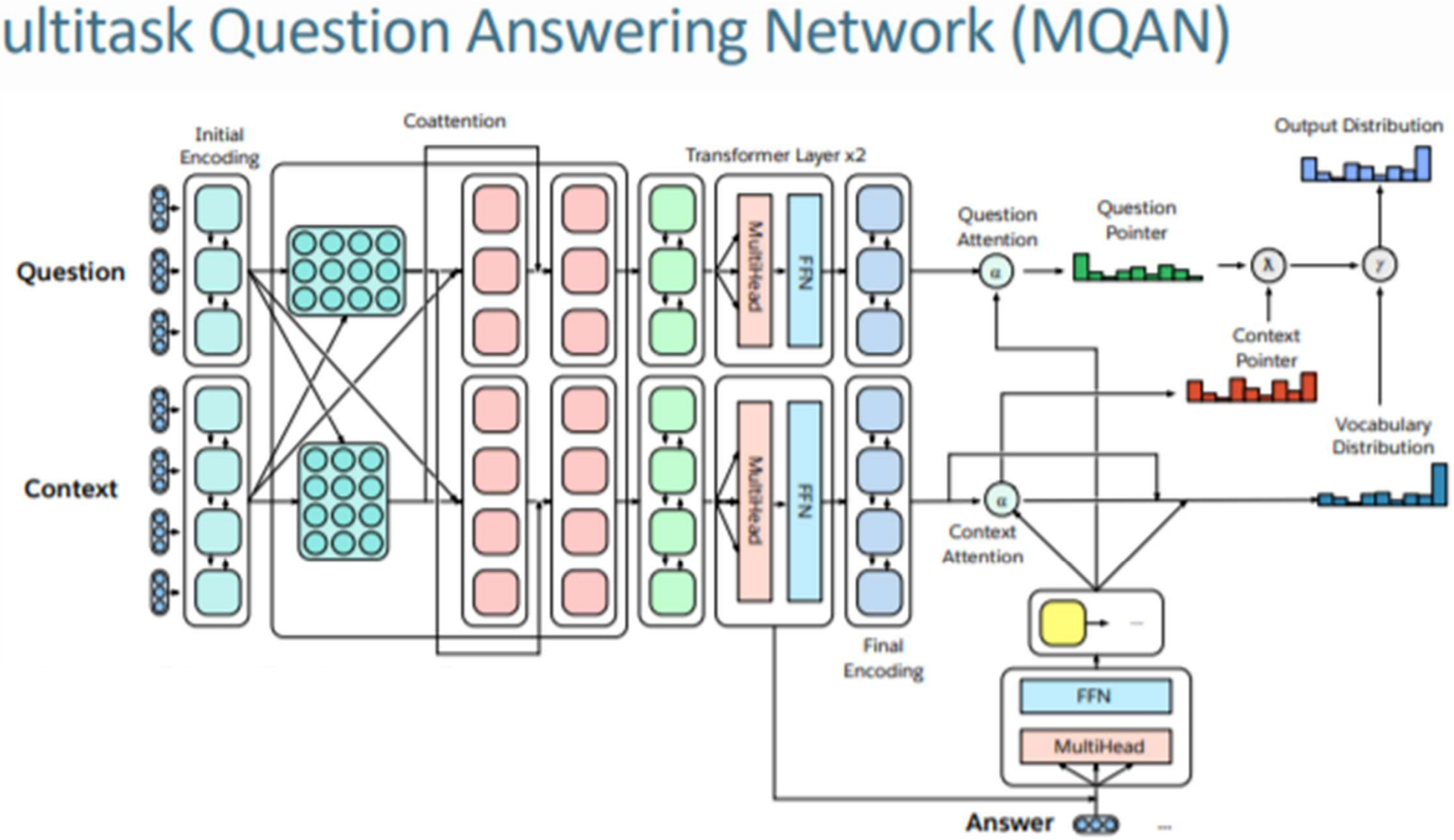
# What's Next in NLP?

## Multi-Task Learning

- NLP consists of three "supertasks"
  - Language Modeling
  - Q&A
  - Dialogue
- We can view multi-task learning as Question Answering
  - Meta-supervised learning: from {x, y} to {x, t, y} (t is the task)
  - Use a question, q, as a natural description of the task, t, to allow the model to use linguistic information to connect tasks
  - y is the answer to q and x is the context necessary to answer q

# What's Next in NLP?

## Multi-Task Learning

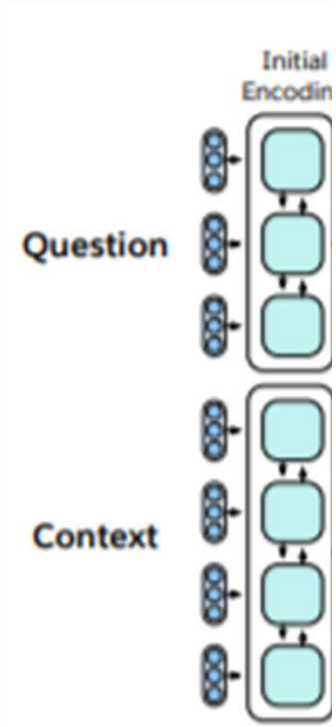


# What's Next in NLP?

## Multi-Task Learning

- Fixed embeddings → linear → shared BiLSTM with skip connection

## Multitask Question Answering Network (MQAN)

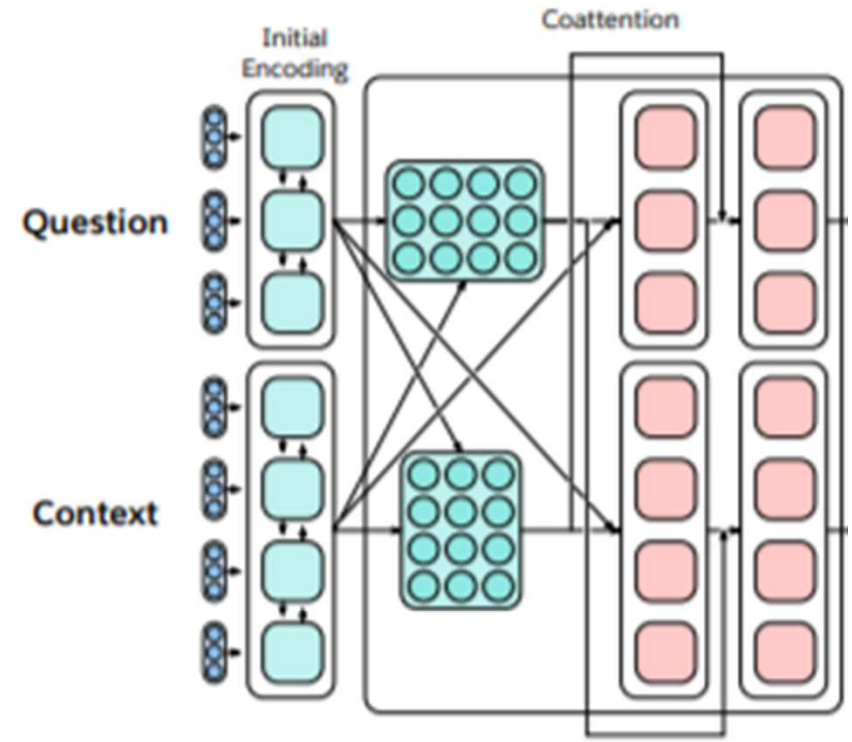


# What's Next in NLP?

## Multi-Task Learning

- Attention summations from one sequence to another and back again (coattention) with skip connections

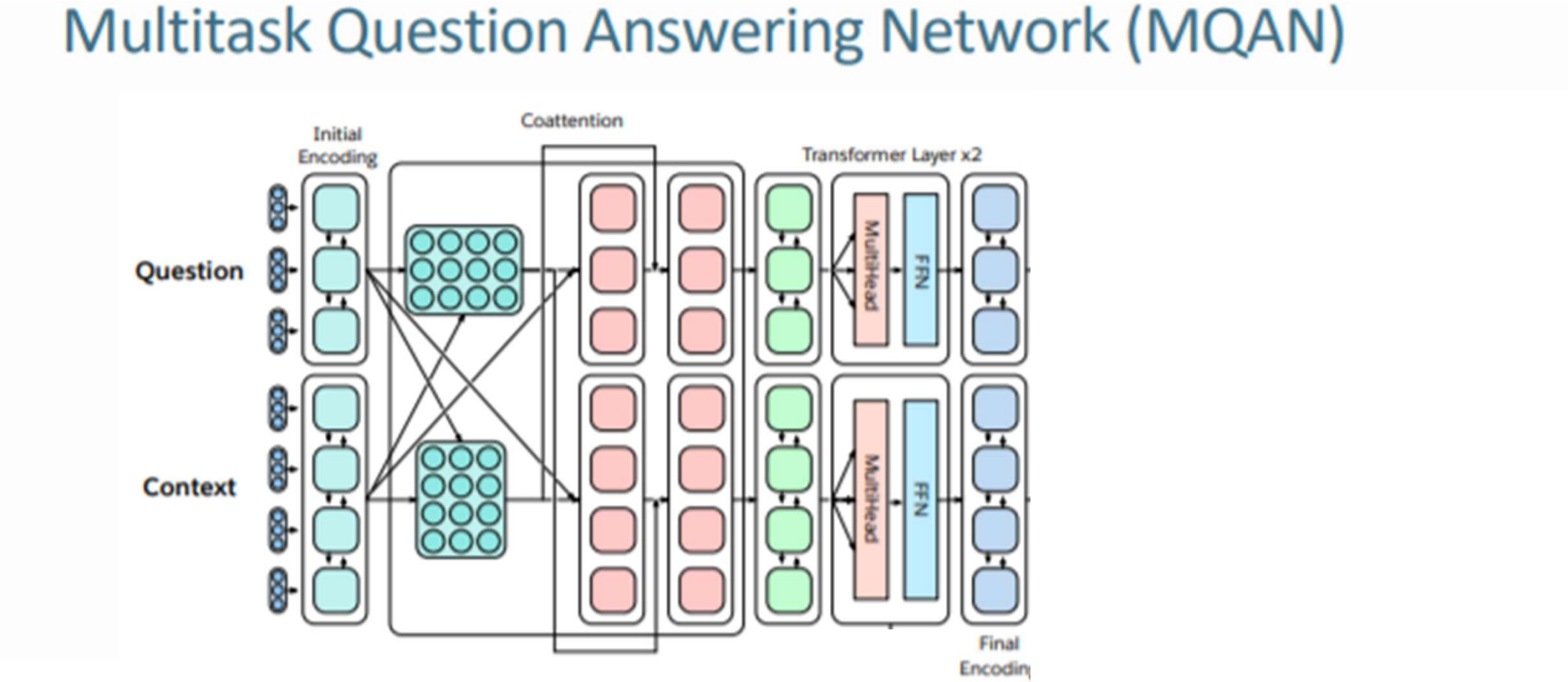
Multitask Question Answering Network (MQAN)



# What's Next in NLP?

## Multi-Task Learning

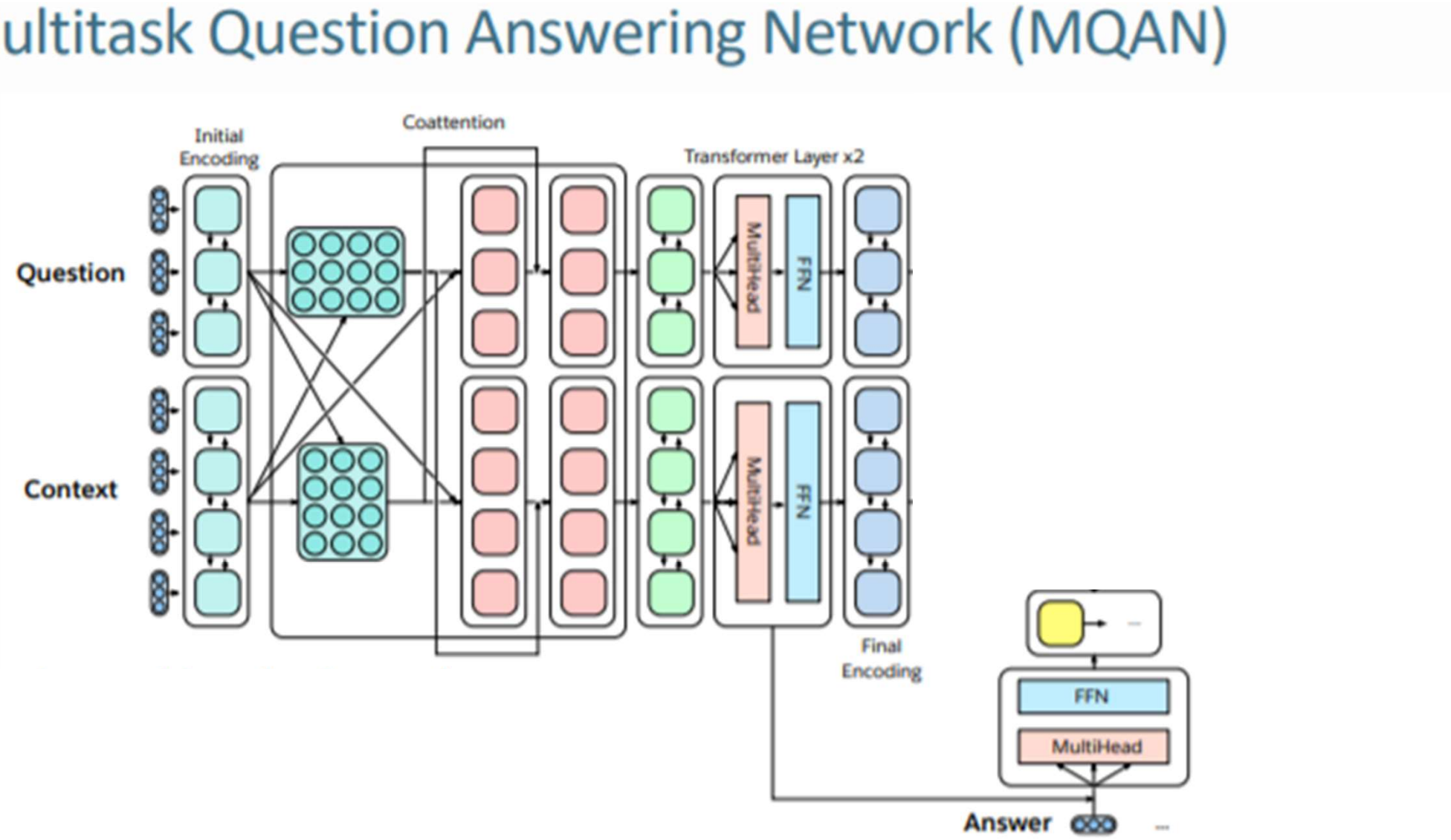
- Separate BiLSTMs to reduce dimensionality, two transformer layers, another BiLSTM



# What's Next in NLP?

## Multi-Task Learning

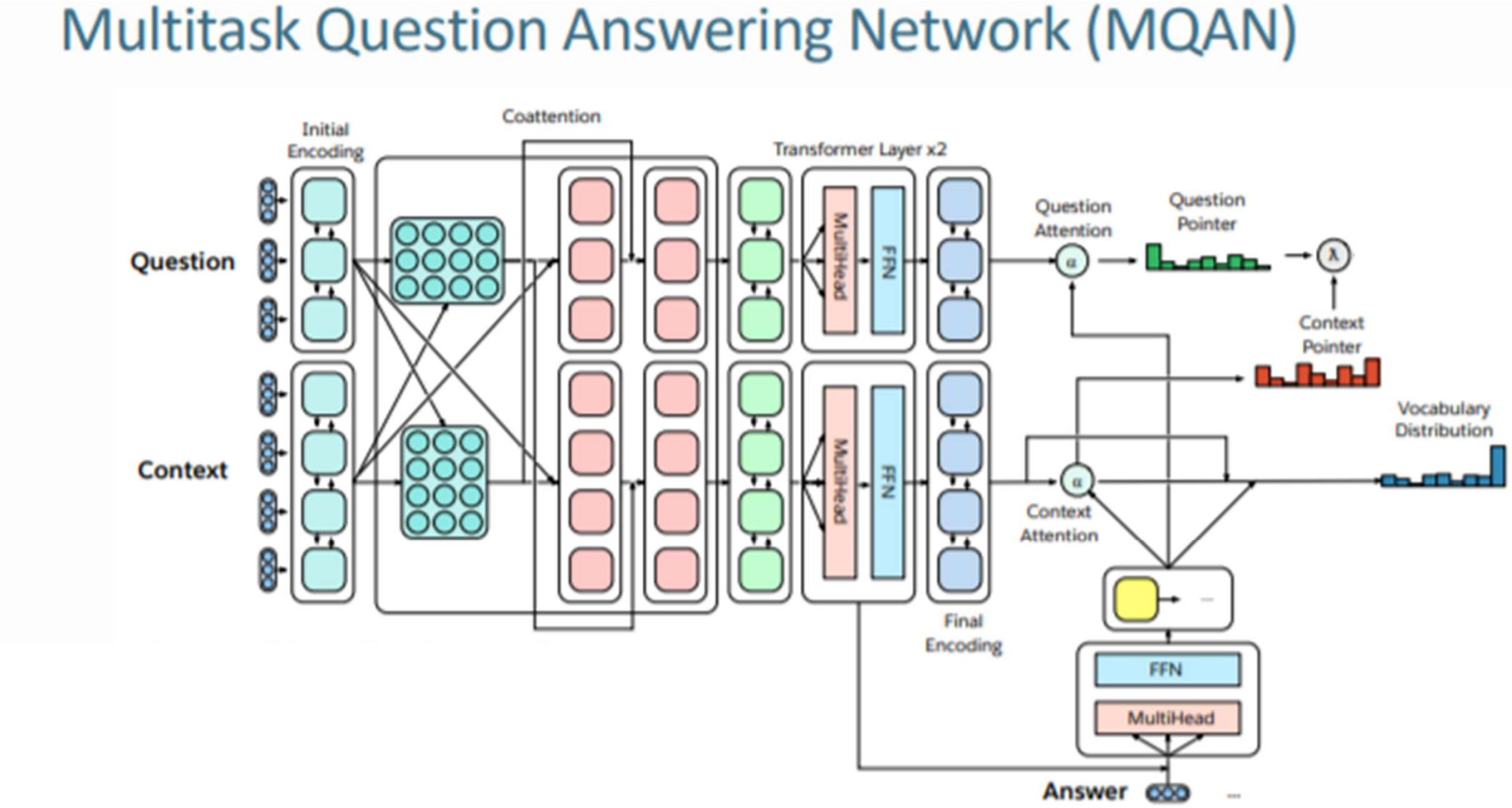
- Auto-regressive decoder uses embeddings, two transformer layers, and an LSTM layer that attend to outputs of last 3 layers of the encoder



# What's Next in NLP?

## Multi-Task Learning

- LSTM decoder state is used to compute attention distributions over the context and question which are used pointers

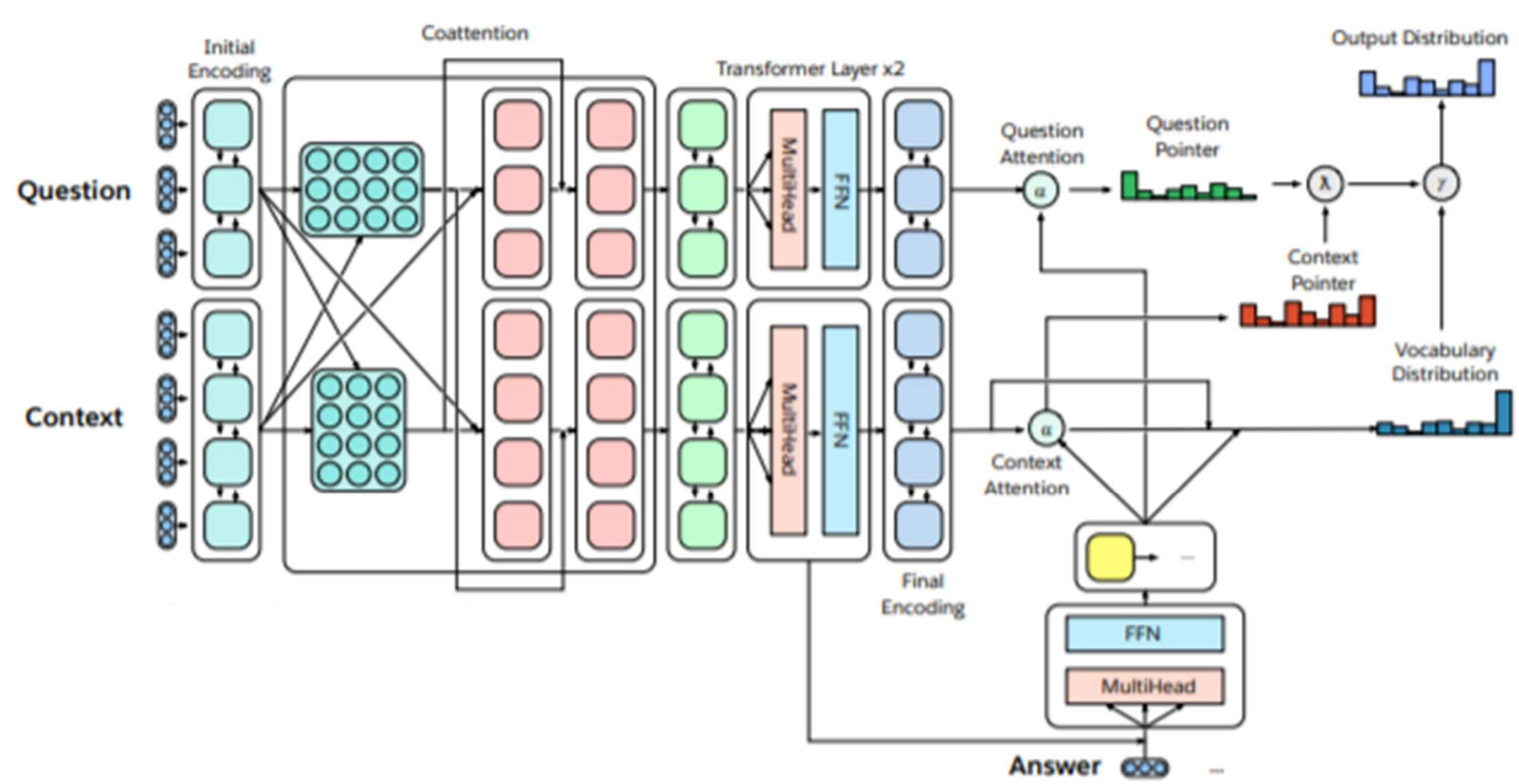


# What's Next in NLP?

## Multi-Task Learning

- Attention over context and question influences two switches
  - Gamma decides whether to copy or select from an external vocabulary
  - Lambda decides whether to copy from context or question

Multitask Question Answering Network (MQAN)



# What’s Next in NLP?

## Unsupervised Neural Machine Translation

# What's Next in NLP?

## Unsupervised Neural Machine Translation

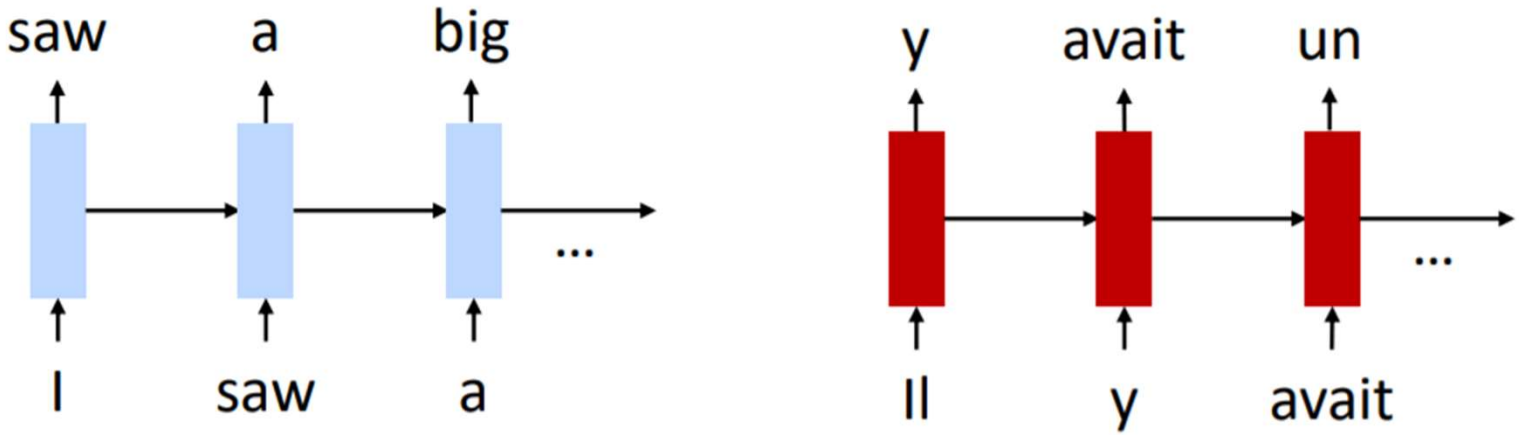
- For English, most tasks have < 100K labeled examples
- Even less data for other languages
- Increasingly popular solution → use **unlabeled** data
- Acquiring translations require human expertise → limits the size and domain of data
- Monolingual text is easier to capture!

# What's Next in NLP?

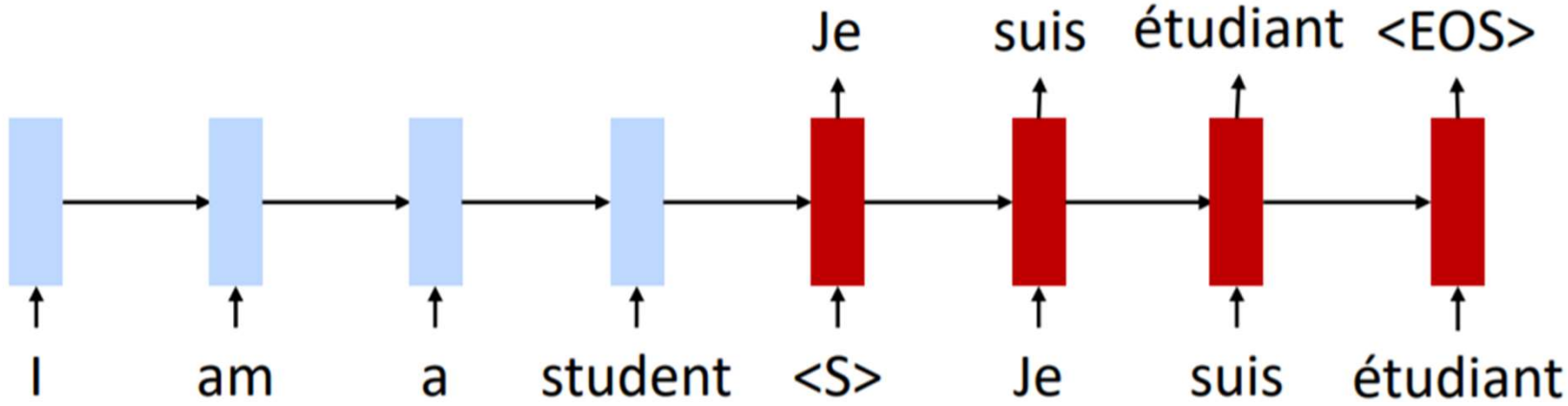
## Unsupervised Neural Machine Translation

### Pre-Training

1. Separately Train Encoder and Decoder as Language Models




2. Then Train Jointly on Bilingual Data

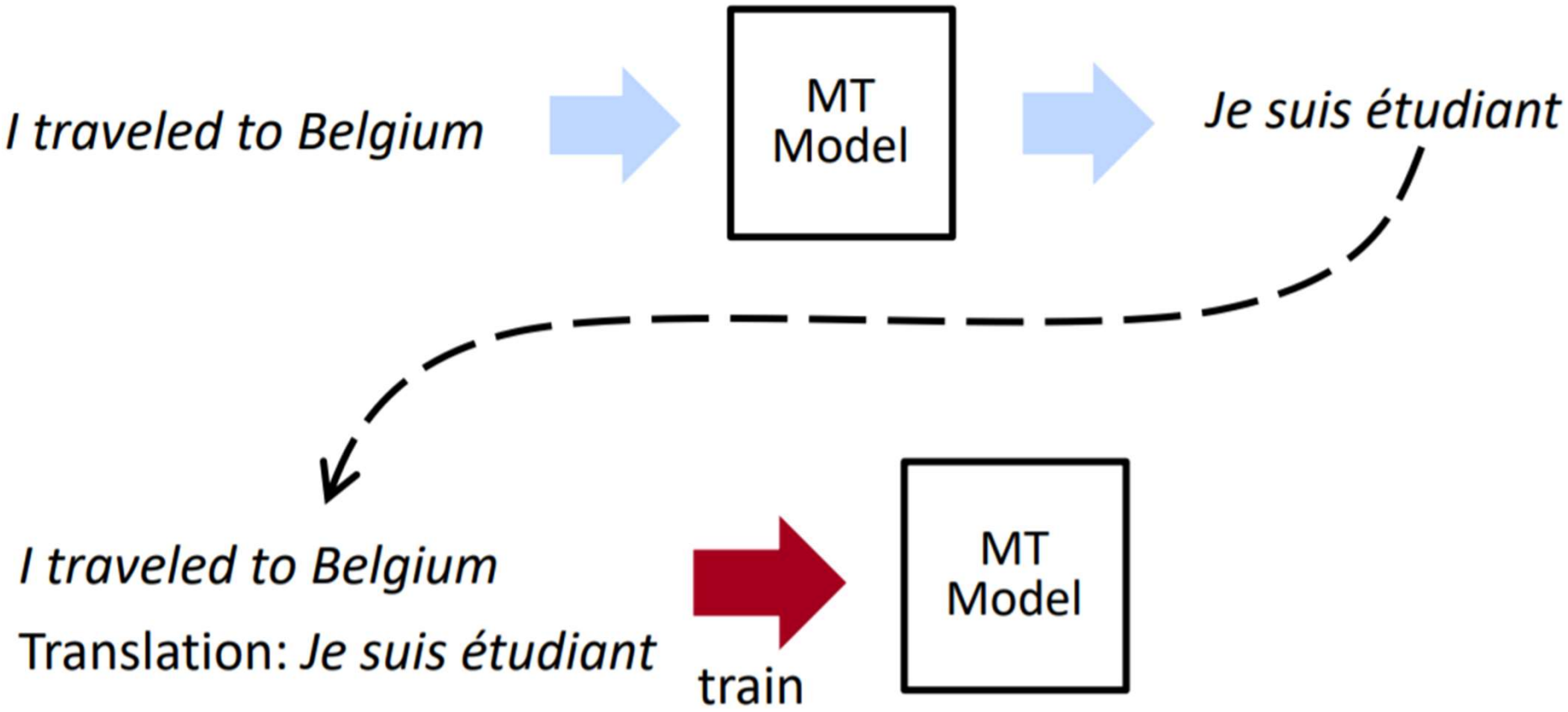


# What's Next in NLP?

## Unsupervised Neural Machine Translation

### Self-Training

- Problem with pre-training: no "interaction" between the two languages during pre-training
- Self-training: label unlabeled data to get noisy training examples

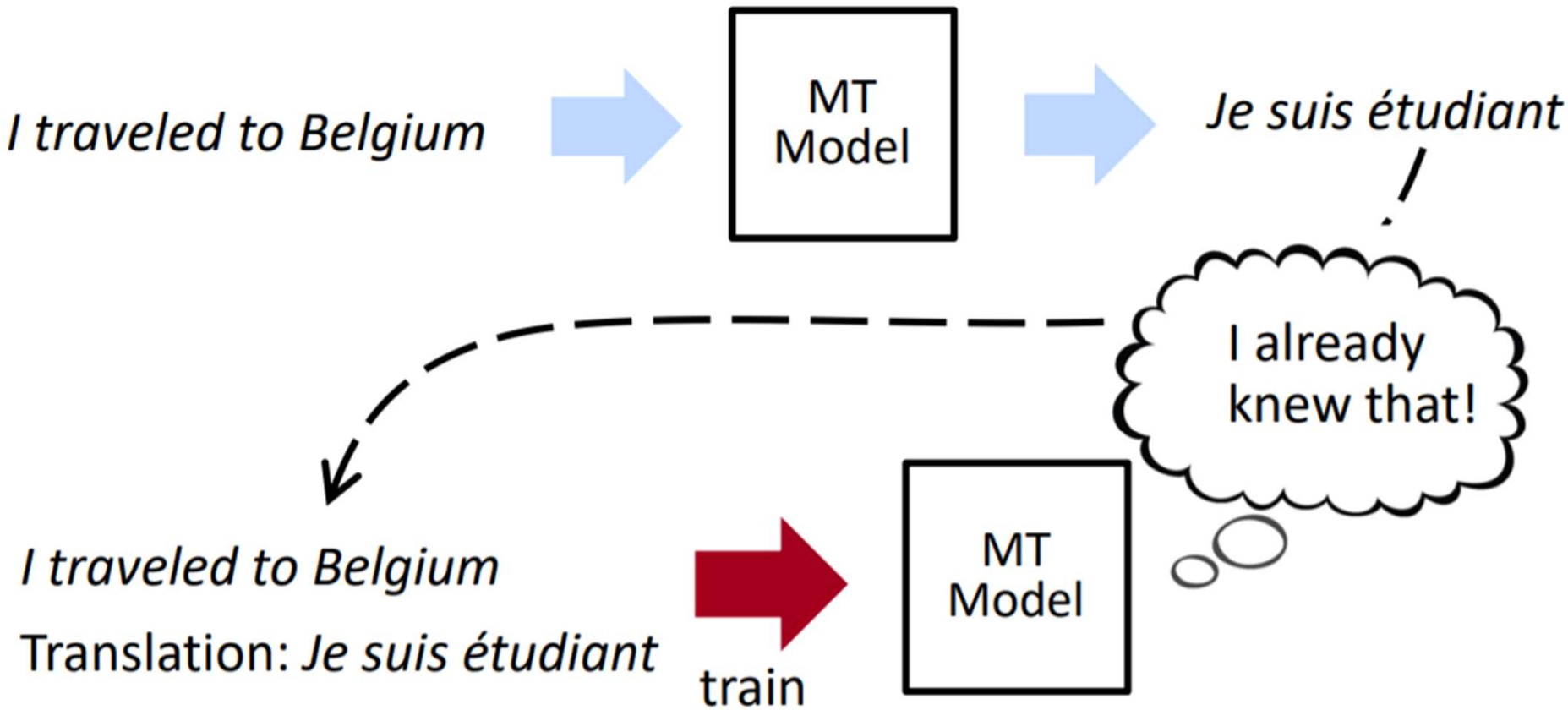


# What's Next in NLP?

## Unsupervised Neural Machine Translation

### Self-Training

- Circular?

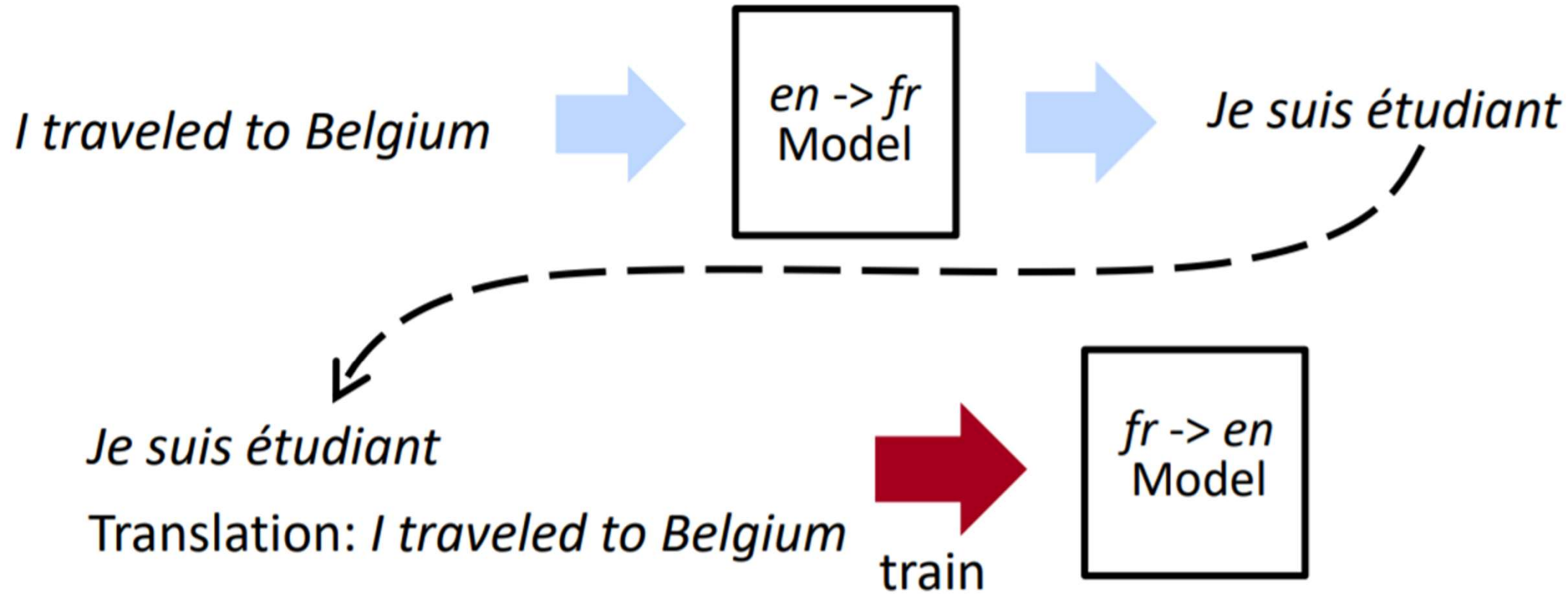


# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Back-Translation

- Have two machine translation models going in opposite directions (*en -> fr*) and (*fr -> en)*

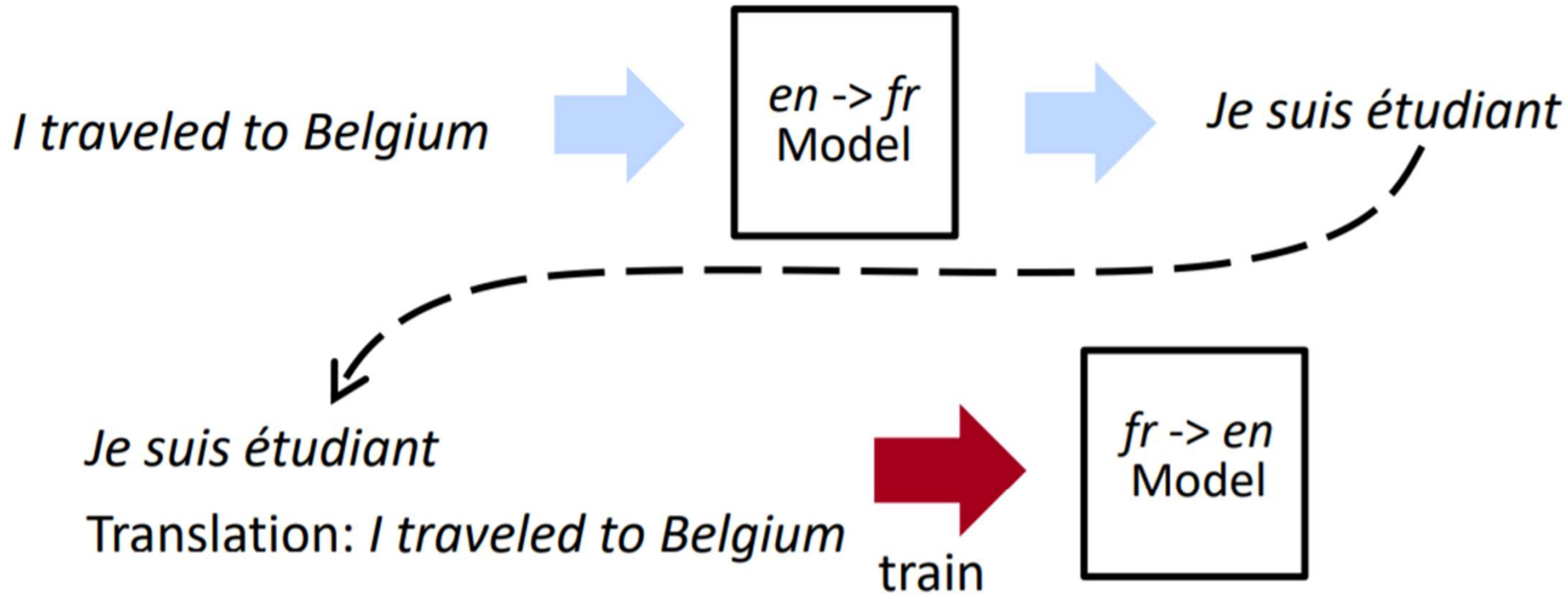


# What's Next in NLP?

## Unsupervised Neural Machine Translation

### Back-Translation

- Have two machine translation models going in opposite directions (*en -> fr*) and (*fr -> en)*

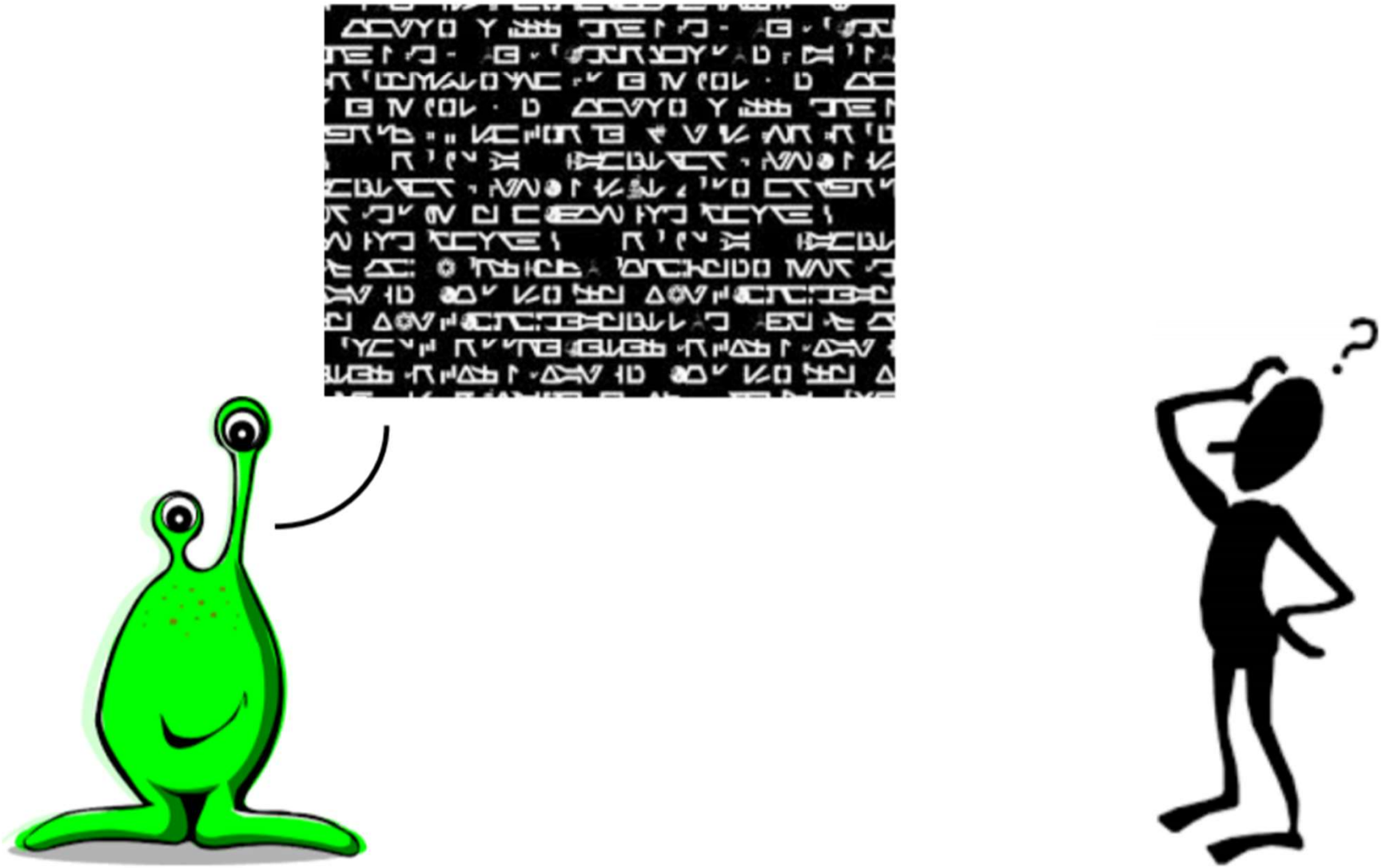



- No longer circular
- Models never see "bad" translations, only bad inputs

# What's Next in NLP?

## Unsupervised Neural Machine Translation

**What if there is no Bilingual Data?**

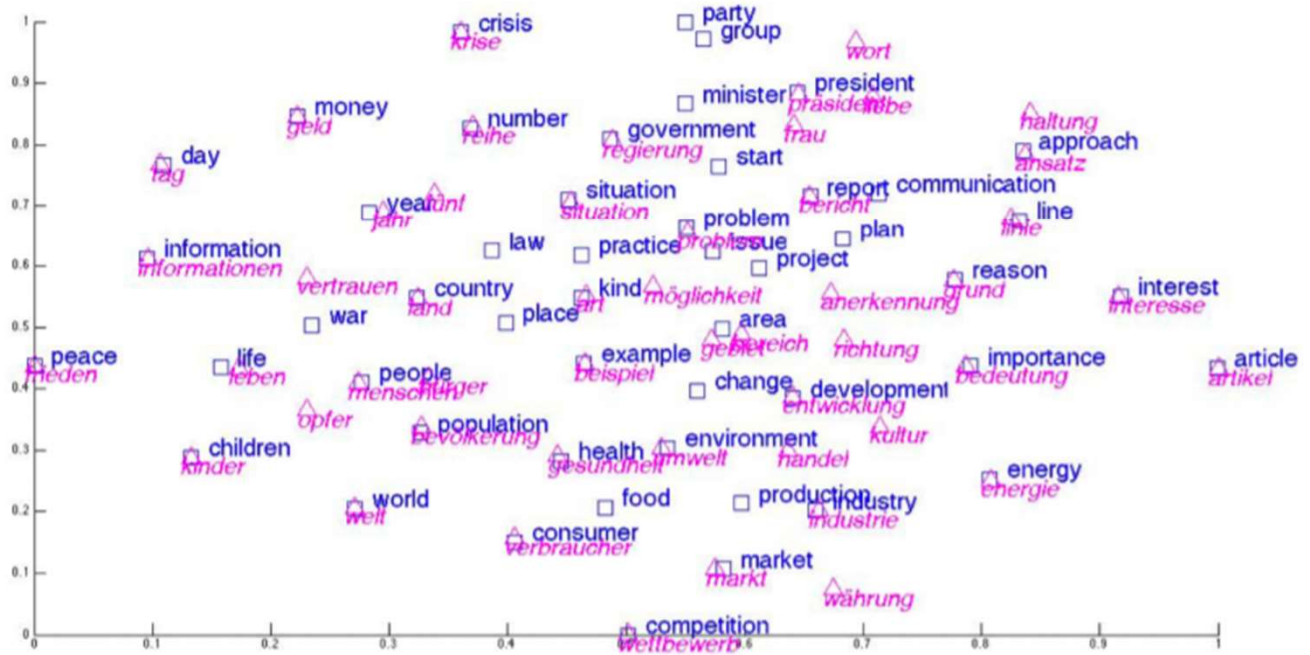


# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Unsupervised Word Translation

- *Cross-lingual word embeddings*
  - Shared embedding space for both languages
  - Keep the normal nice properties of word embeddings
  - But also want words close to their translations
- Want to learn from monolingual corpora

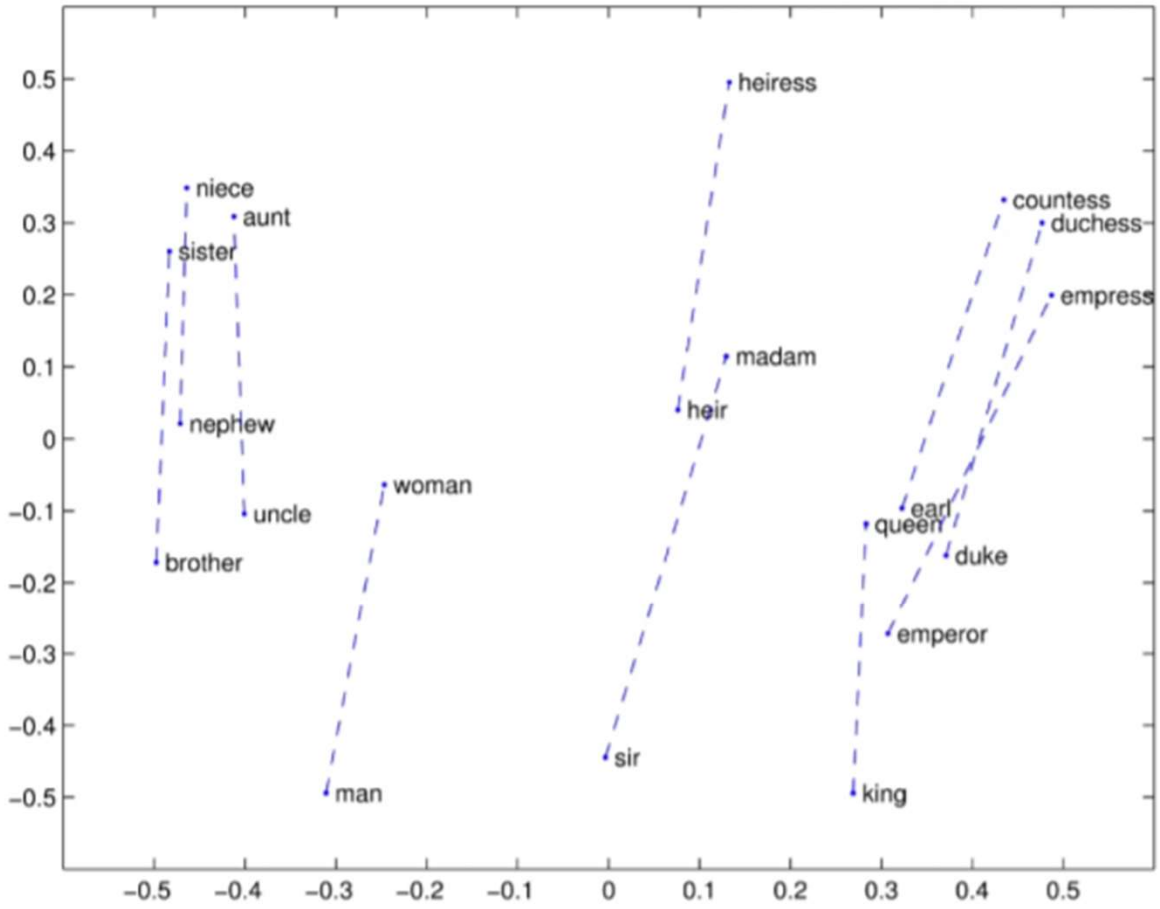


# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Unsupervised Word Translation

- Word embeddings have a lot of structure
- Assumption: that structure should be similar across languages



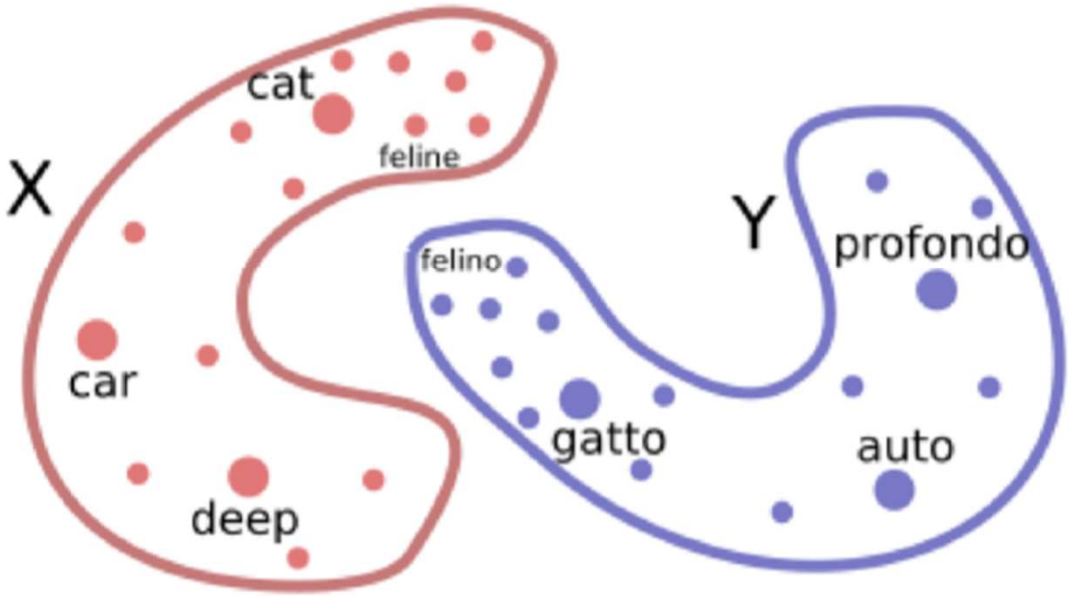
# What's Next in NLP?

## Unsupervised Neural Machine Translation

### Unsupervised Word Translation

- Word embeddings have a lot of structure

- Assumption: that structure should be similar across languages

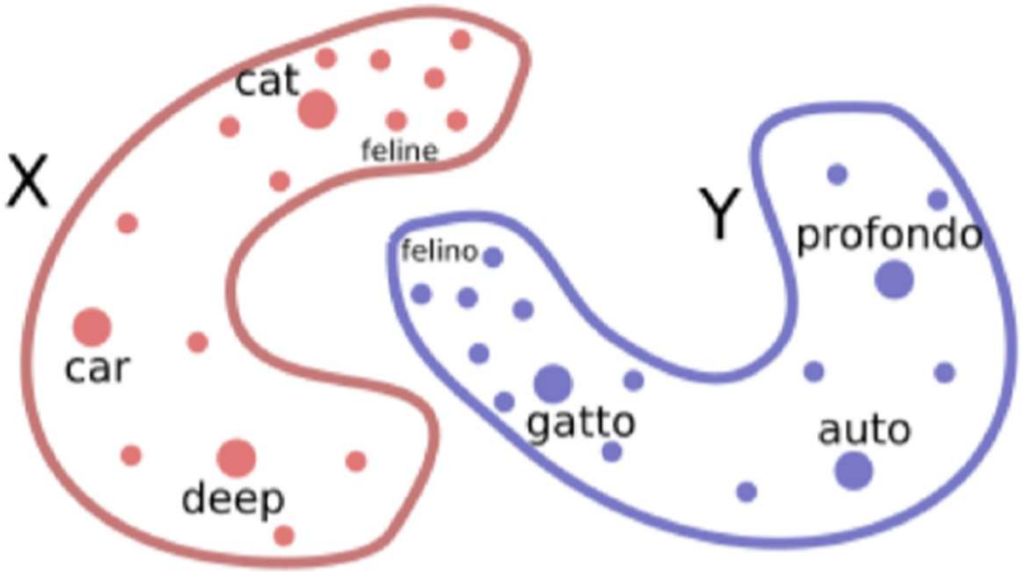


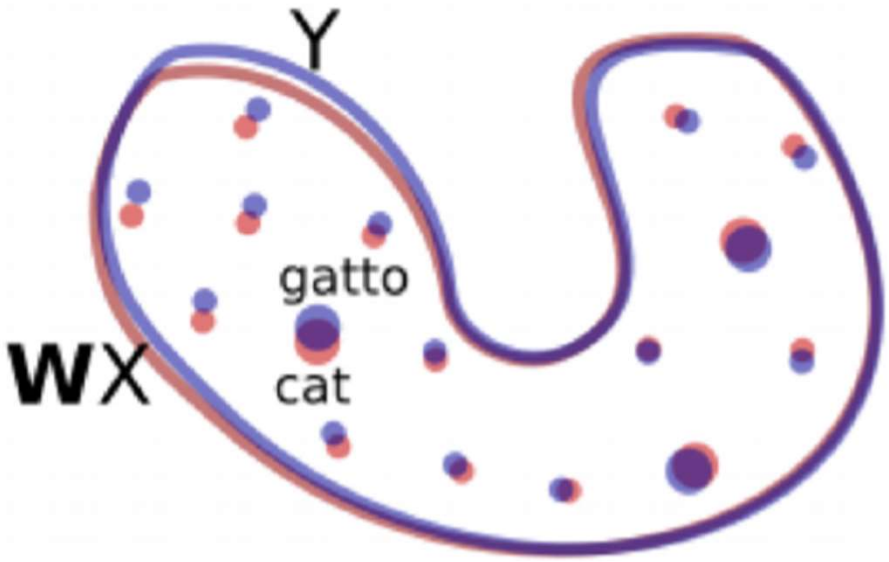
# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Unsupervised Word Translation

- First run word2vec on monolingual corpora, getting words embeddings *X* and *Y*
- Learn an (orthogonal) matrix *W* such that *WX* ~ *Y*



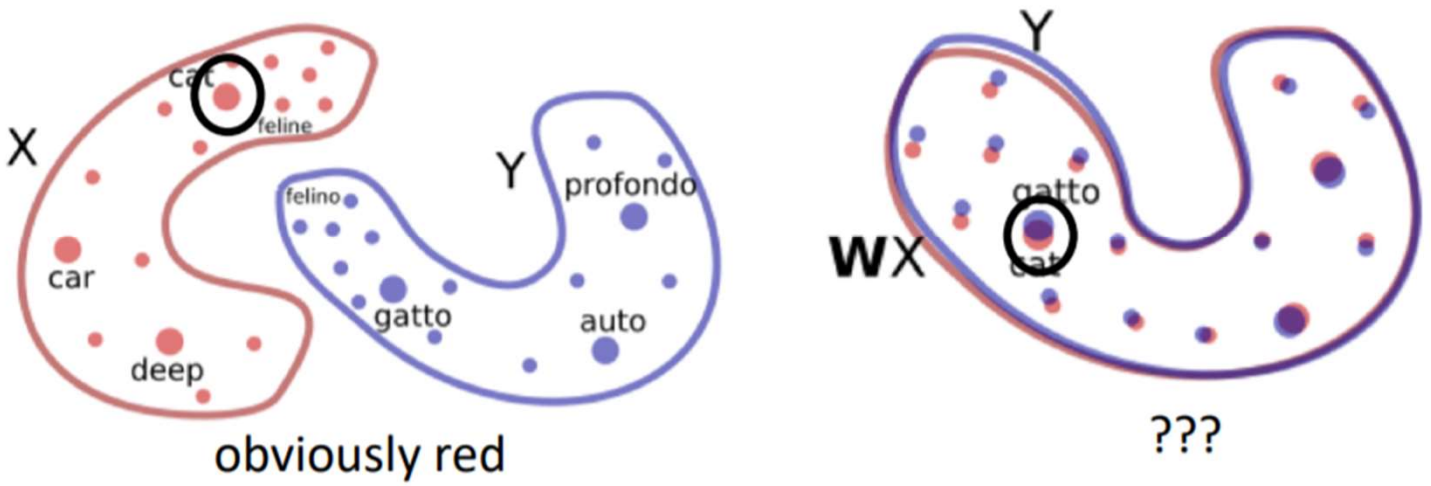
# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Unsupervised Word Translation

- Learn *W* with *adversarial training.*
- Discriminator: predict if an embedding is from *Y* or it is a transformed embedding *Wx* originally from *X.*
- Train *W* so the Discriminator gets "confused"

Discriminator predicts: is the circled point red or blue?



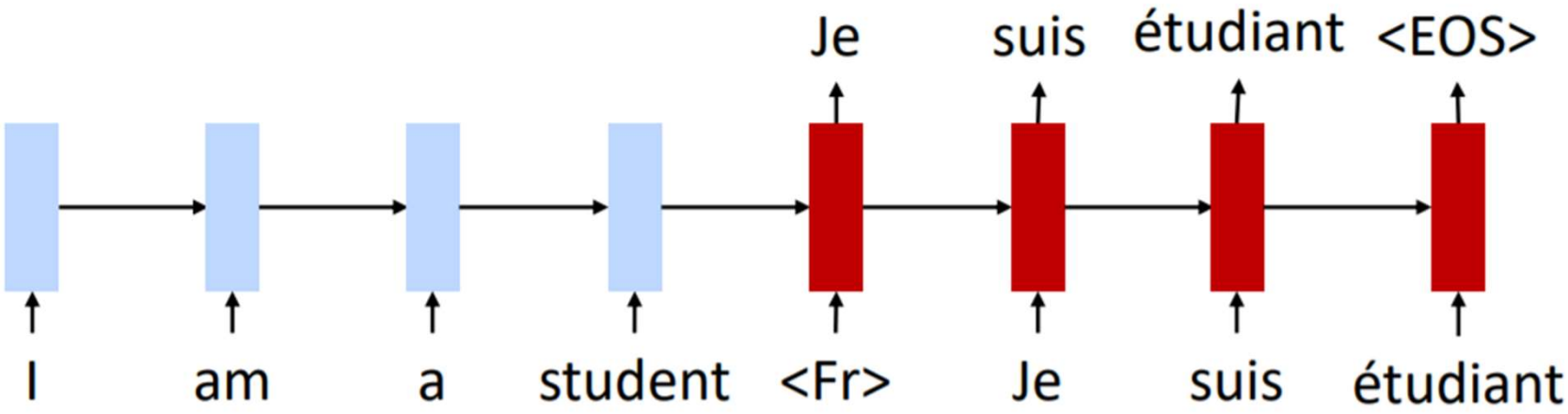

- Other tricks can be used to further improve performance, see Word Translation without Parallel Data

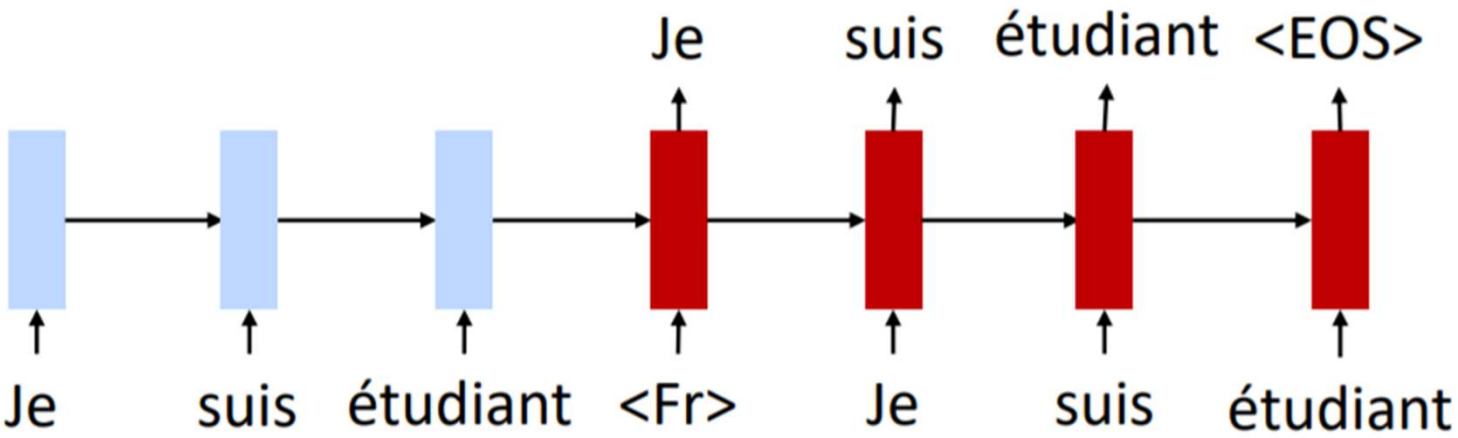
# What's Next in NLP?

## Unsupervised Neural Machine Translation

### Unsupervised Machine Translation

- Model: **same** encoder-decoder used for both languages
  - Initialize with cross-lingual word embeddings



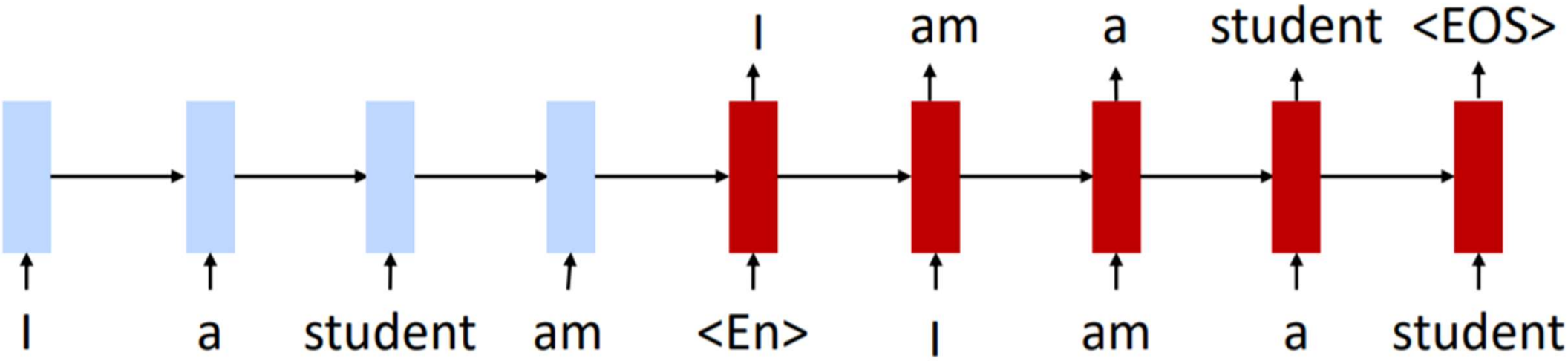



# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Unsupervised Neural Machine Translation

- Training objective 1: de-noising autoencoder

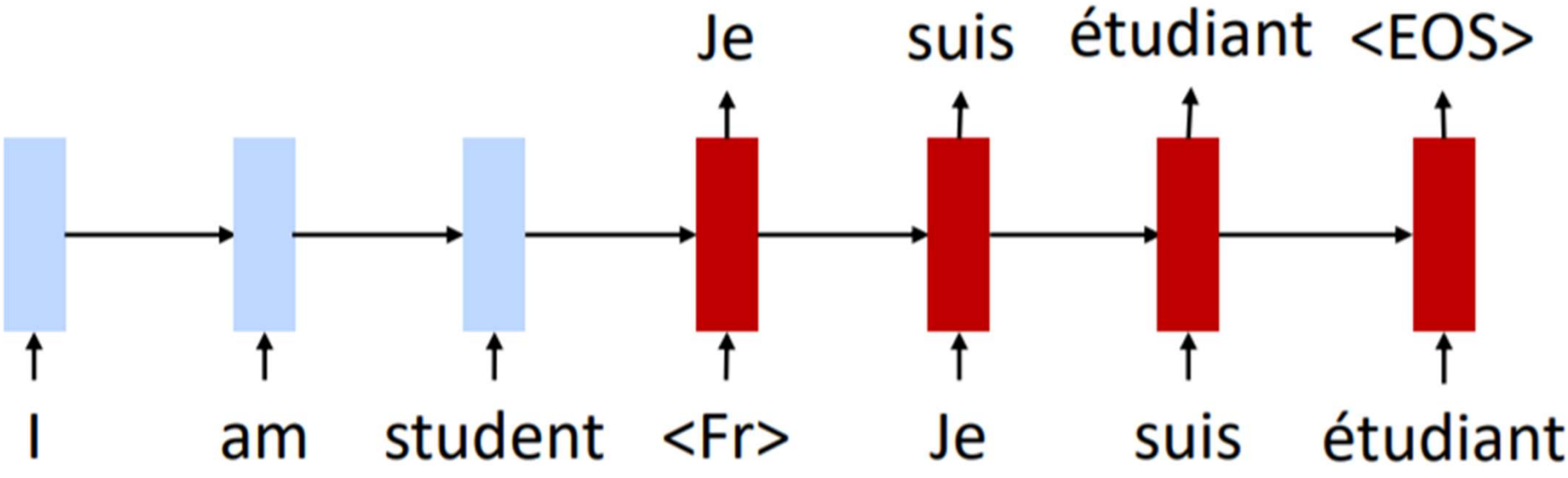


# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Unsupervised Neural Machine Translation

- Training objective 2: back translation
  - First translate *fr -> en*
  - Then use as a "supervised" example to train *en -> fr*

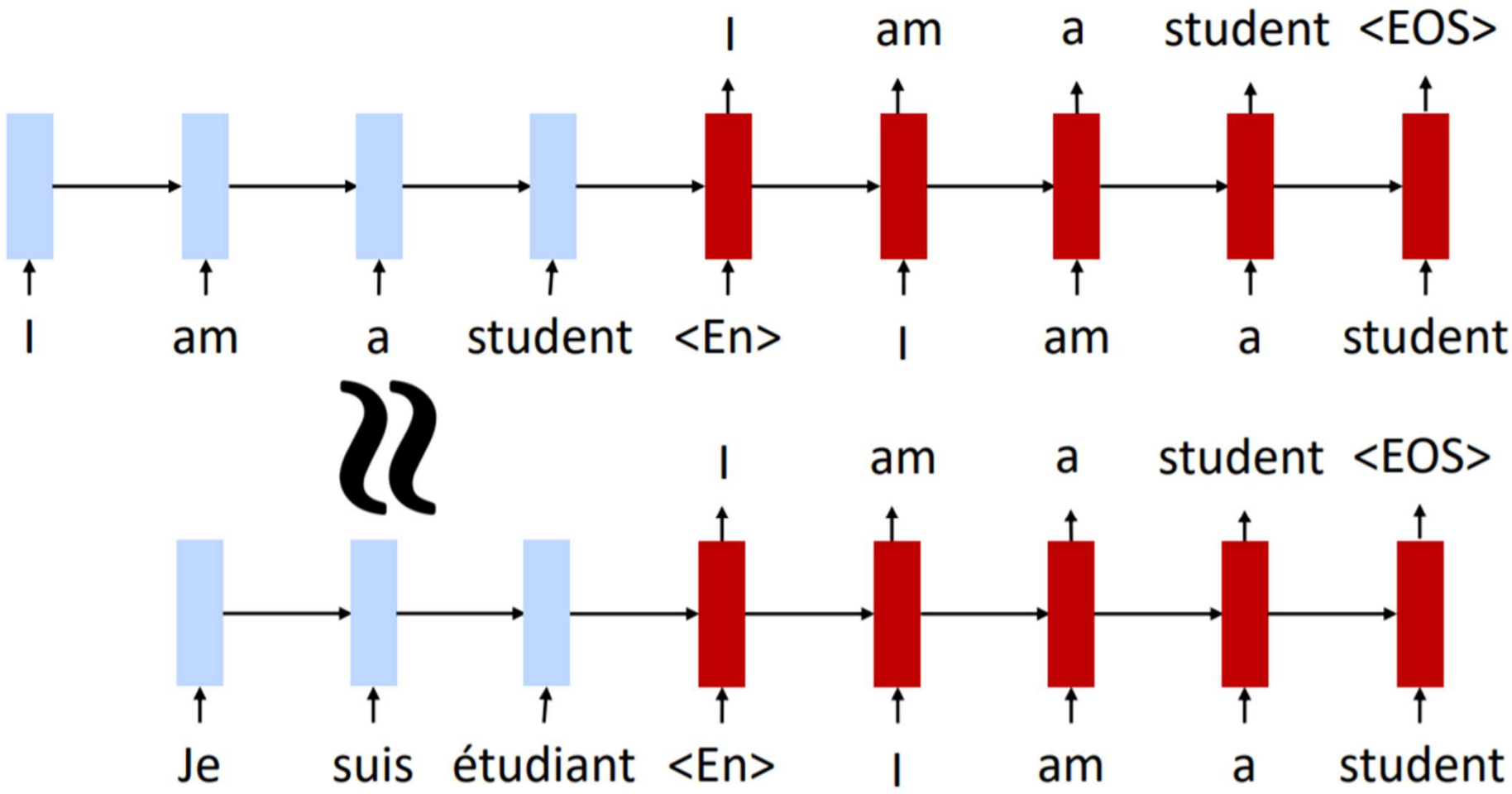


# What's Next in NLP?

## Unsupervised Neural Machine Translation

### Why Does This Work?

- Cross lingual embeddings and shared encoder gives the model a starting point

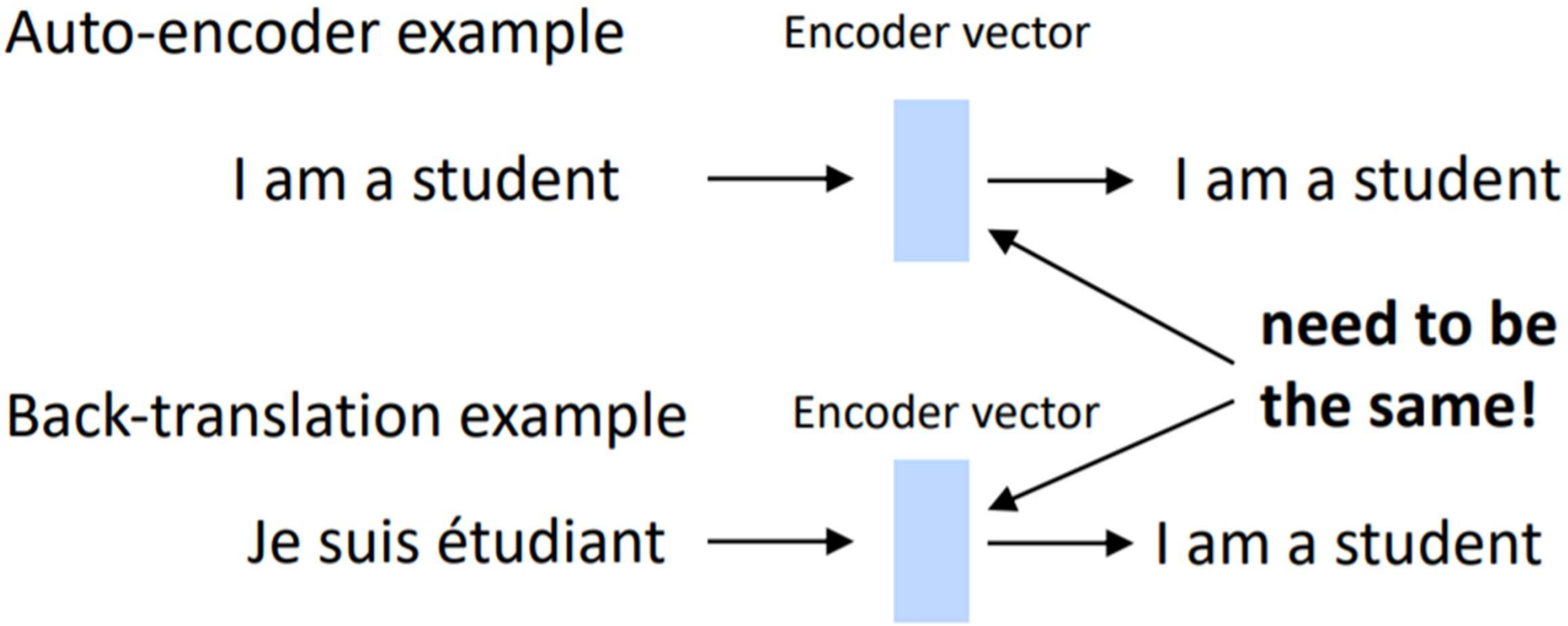


# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Why Does This Work?

- Objectives encourage language-agnostic representation

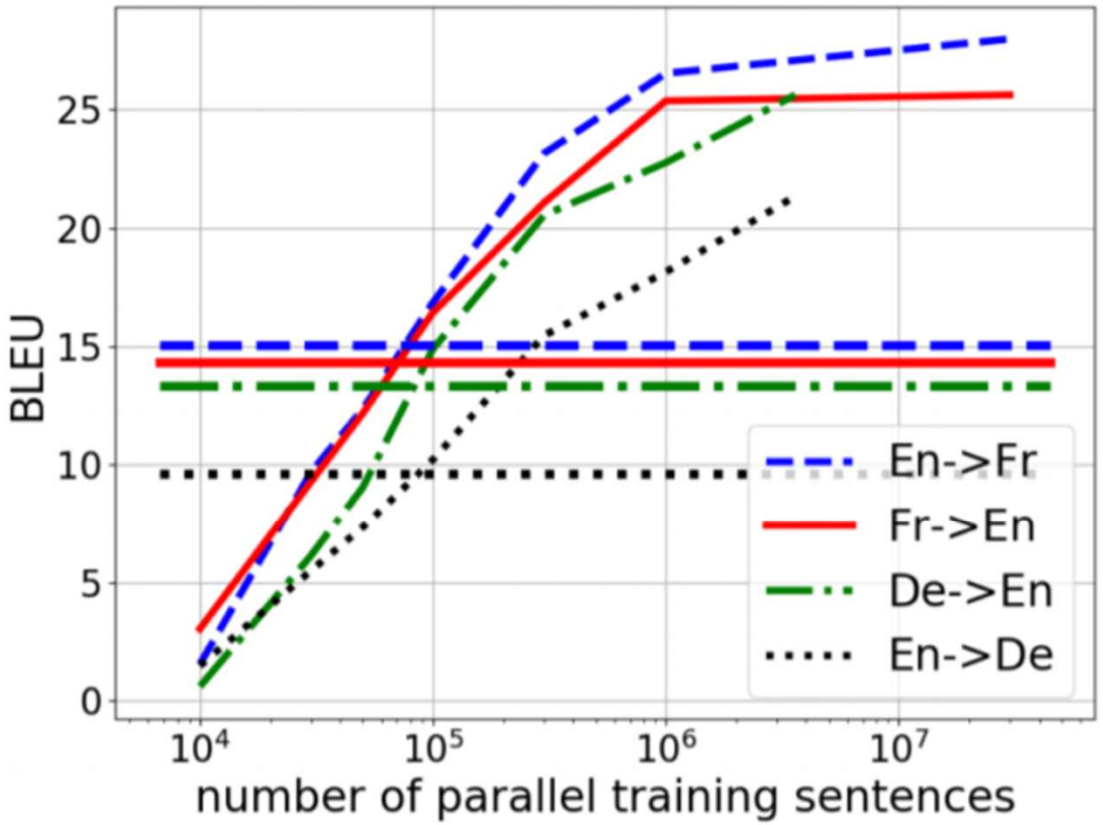


# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Unsupervised Machine Translation

- Horizontal lines are unsupervised models, the rest are supervised




Lample et al., 2018

# What's Next in NLP?

## Unsupervised Neural Machine Translation

### Not so Fast

- English, French, and German are fairly similar

- On very different languages (e.g., English and Turkish)…
  - Purely unsupervised word translation doesn't work very. Need *seed dictionary* of likely translations.
    - Simple trick: use identical strings from both vocabularies

  - UNMT barely works

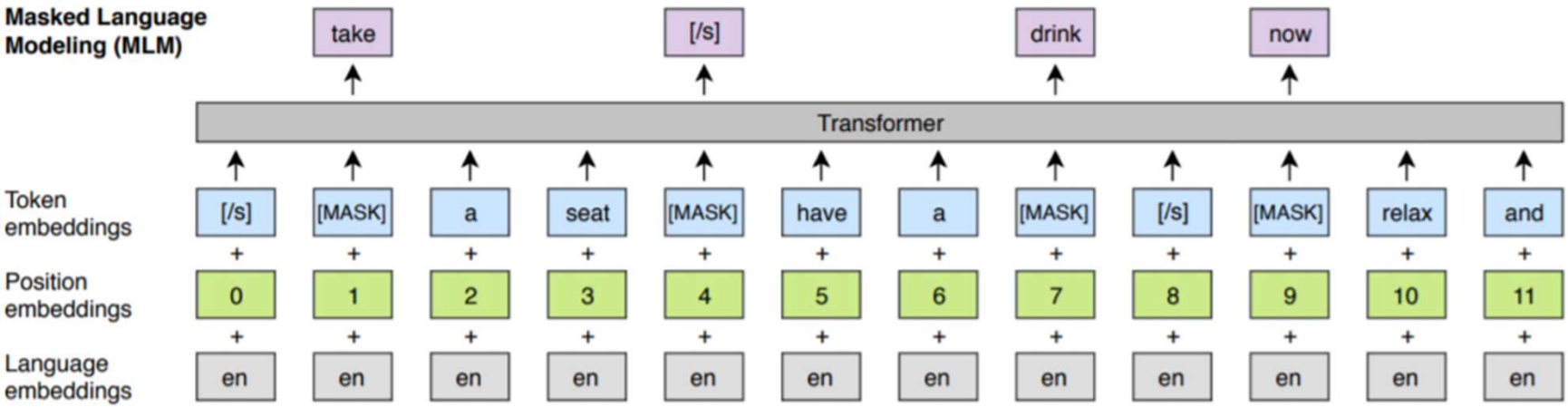
| System | English-Turkish BLEU |
|---|---|
| Supervised | ~20 |
| Word-for-word unsupervised | 1.5 |
| UNMT | 4.5 |

Hokamp et al., 2018

# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Cross-Lingual BERT

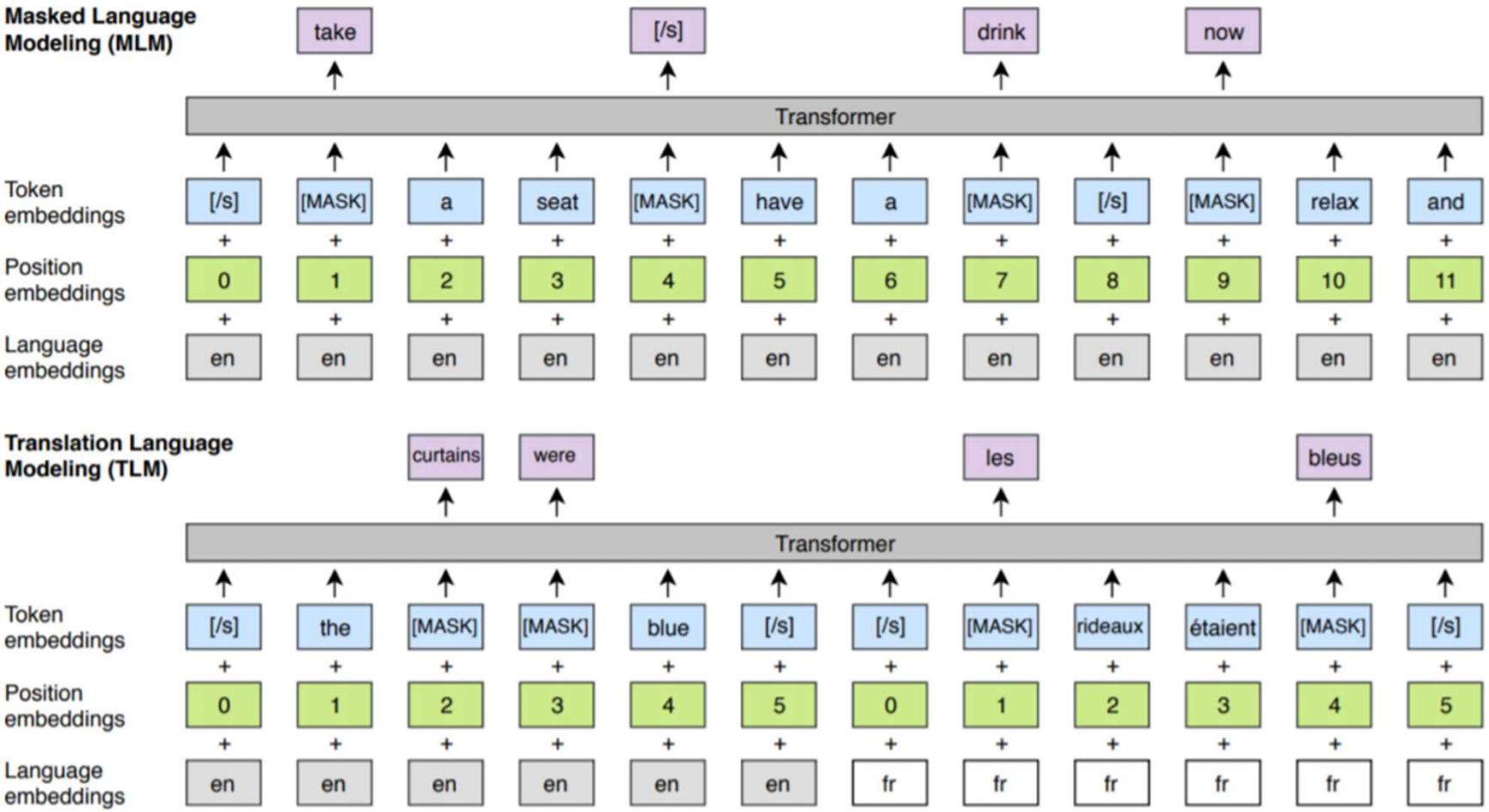



Lample and Conneau., 2019

# What's Next in NLP?

## Unsupervised Neural Machine Translation

## Cross-Lingual BERT




Lample and Conneau., 2019

# What’s Next in NLP?

## Natural Language Understanding

# What's Next in NLP?

## Natural Language Understanding

## QuAC: Question Answering in Context

- Dialogue between a student who asks questions and a teacher who answers
  - Teacher sees Wikipedia article on the subject, student doesn't

**Section:** Daffy Duck, Origin & History

STUDENT: **What is the origin of Daffy Duck?**
TEACHER: ↪ first appeared in Porky's Duck Hunt
STUDENT: **What was he like in that episode?**
TEACHER: ↪ assertive, unrestrained, combative
STUDENT: **Was he the star?**
TEACHER: ↪ No, barely more than an unnamed bit player in this short
STUDENT: **Who was the star?**
TEACHER: ↛ No answer
STUDENT: **Did he change a lot from that first episode in future episodes?**
TEACHER: ↪ Yes, the only aspects of the character that have remained consistent (...) are his voice characterization by Mel Blanc
STUDENT: **How has he changed?**
TEACHER: ↪ Daffy was less anthropomorphic
STUDENT: **In what other ways did he change?**
TEACHER: ↪ Daffy's slobbery, exaggerated lisp (...) is barely noticeable in the early cartoons.
STUDENT: **Why did they add the lisp?**
TEACHER: ↪ One often-repeated "official" story is that it was modeled after producer Leon Schlesinger's tendency to lisp.
STUDENT: **Is there an "unofficial" story?**
TEACHER: ↪ Yes, Mel Blanc (...) contradicts that conventional belief
. . .

Choi et al., 2018

# What's Next in NLP?

## Natural Language Understanding

## QuAC: Question Answering in Context

- Still a big gap to human performance

| Rank | Model | F1 | HEQQ | HEQD |
|---|---|---|---|---|
| | Human Performance<br>(Choi et al. EMNLP '18) | 81.1 | 100 | 100 |
|  <br>March 7, 2019 | BERT w/ 2-context<br>(single model)<br>*NTT Media Intelligence Labs* | **64.9** | 60.2 | **6.1** |
| 2<br>Feb 21, 2019 | GraphFlow (single model)<br>*Anonymous* | **64.9** | **60.3** | 5.1 |
| 3<br>Sep 26, 2018 | FlowQA (single model)<br>*Allen Institute of AI*<br>https://arxiv.org/abs/1810.06683 | 64.1 | 59.6 | 5.8 |
| 4<br>Jan 30, 2019 | BERT + History Answer<br>Embedding (single model)<br>*Anonymous* | 62.4 | 57.8 | 5.1 |
| 5<br>Aug 20, 2018 | BiDAF++ w/ 2-Context<br>(single model)<br>*baseline* | 60.1 | 54.8 | 4.0 |
| 6<br>Aug 20, 2018 | BiDAF++ (single model)<br>*baseline* | 50.2 | 43.3 | 2.2 |

# What's Next in NLP?

## Natural Language Understanding

### HotPotQA

- Designed to require *multi-hop reasoning*
- Questions are over *multiple documents*

**Paragraph A, Return to Olympus:**
[1] *Return to Olympus is the only album by the alternative rock band Malfunkshun.* [2] *It was released after the band had broken up and after lead singer Andrew Wood (later of Mother Love Bone) had died of a drug overdose in 1990.* [3] Stone Gossard, of Pearl Jam, had compiled the songs and released the album on his label, Loosegroove Records.

**Paragraph B, Mother Love Bone:**
[4] *Mother Love Bone was an American rock band that formed in Seattle, Washington in 1987.* [5] The band was active from 1987 to 1990. [6] *Frontman Andrew Wood's personality and compositions helped to catapult the group to the top of the burgeoning late 1980s/early 1990s Seattle music scene.* [7] *Wood died only days before the scheduled release of the band's debut album, "Apple", thus ending the group's hopes of success.* [8] The album was finally released a few months later.

**Q:** What was the former band of the member of Mother Love Bone who died just before the release of "Apple"?
**A:** Malfunkshun
**Supporting facts:** 1, 2, 4, 6, 7

Figure 1: An example of the multi-hop questions in HOTPOTQA. We also highlight the supporting facts in *blue italics*, which are also part of the dataset.

Zang et al., 2018

# What's Next in NLP?

## Natural Language Understanding

## HotPotQA

- Human performance is above 90 F1

| | Model | Code | Ans | |
|---|---|---|---|---|
| | | | EM | $F_1$ |
| 1<br>Nov 21, 2018 | QFE (single model)<br>*NTT Media Intelligence Laboratories* | | 53.86 | 68.06 |
| 2<br>Mar 4, 2019 | GRN (single model)<br>*Anonymous* | | 52.92 | 66.71 |
| 3<br>Mar 1, 2019 | DFGN + BERT (single model)<br>*Anonymous* | | 55.17 | 68.49 |
| 4<br>Mar 4, 2019 | BERT Plus (single model)<br>*CIS Lab* | | **55.84** | **69.76** |
| 5<br>Oct 10, 2018 | Baseline Model (single model)<br>*Carnegie Mellon University, Stanford University, & Universite de Montreal*<br>(Yang, Qi, Zhang, et al. 2018) | | 45.60 | 59.02 |
| -<br>Feb 27, 2019 | DecompRC (single model)<br>*Anonymous* | | 55.20 | 69.63 |

# References

- **Courses**
  - CS224n: Natural Language Processing with Deep Learning, http://web.stanford.edu/class/cs224n/index.html
  - CS231n: Convolutional Neural Networks for Visual Recognition, http://cs231n.stanford.edu/
- **Books**
  - Speech and Language Processing, Daniel Jurafsky & James H. Martin, https://web.stanford.edu/~jurafsky/slp3/3.pdf
  - Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, https://www.deeplearningbook.org/
- **Videos**
  - Transformer XL | AISC Trending Papers, https://www.youtube.com/watch?v=cXZ9YBqH3m0&t=3910s

# References


- **Papers**
  - Class-Based *n*-gram Models of Natural Language, https://www.aclweb.org/anthology/J92-4003
  - word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method, https://arxiv.org/abs/1402.3722
  - word2vec Parameter Learning Explained, https://arxiv.org/abs/1411.2738
  - Efficient Estimation of Word Representations in Vector Space, https://arxiv.org/abs/1301.3781
  - Distributed Representations of Words and Phrases and their Compositionality, https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
  - Distributed Representations of Sentences and Documents, https://arxiv.org/abs/1405.4053
  - Skip-Thought Vectors, https://arxiv.org/pdf/1506.06726.pdf
  - Japanese and Korean Voice Search, https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/37842.pdf
  - GloVe: Global Vectors for Word Representations, https://nlp.stanford.edu/pubs/glove.pdf
  - Deep contextualized word representations, https://arxiv.org/pdf/1802.05365.pdf
  - Neural Machine Translation of Rare Words with Subword Units, https://arxiv.org/abs/1508.07909
  - Contextual Word Represenations: A Contextual Introduction, https://arxiv.org/abs/1902.06006
  - BLEU: a Method for Automatic Evaluation of Machine Translation, https://www.aclweb.org/anthology/P02-1040
  - Sequence to Sequence Learning with Neural Networks, https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf
  - Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, https://arxiv.org/abs/1406.1078
  - Neural Machine Translation by Jointly Learning to Align and Translate, https://arxiv.org/abs/1409.0473
  - Effective Approaches to Attention-based Neural Machine Translation, https://arxiv.org/abs/1508.04025
  - Attention Is All You Need, https://arxiv.org/abs/1706.03762
  - BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, https://arxiv.org/abs/1810.04805
  - Character-Level Language Modeling with Deeper Self-Attention, https://arxiv.org/pdf/1808.04444.pdf
  - Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context, https://arxiv.org/abs/1901.02860
  - XLNet: Generalized Autoregressive Pretraining for Language Understanding, https://arxiv.org/abs/1906.08237

# References


- **Blogposts**
  - A Review of the Recent History of NLP, http://ruder.io/a-review-of-the-recent-history-of-nlp/
  - Paper Dissected: "Glove: Global Vectors for Word Representation" Explained, http://mlexplained.com/2018/04/29/paper-dissected-glove-global-vectors-for-word-representation-explained/
  - Attention? Attention!, https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
  - Attention in NLP, https://medium.com/@joealato/attention-in-nlp-734c6fa9d983
  - Self-Attention Mechanisms in Natural Language Processing, https://dzone.com/articles/self-attention-mechanisms-in-natural-language-proc
  - A Step-by-Step NLP Guide to Learn ELMo for Extracting Features from Text, https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/
  - The Illustrated Transformer, http://jalammar.github.io/illustrated-transformer/
  - The Annotated Transformer, http://nlp.seas.harvard.edu/2018/04/03/attention.html
  - Linear Relationships in the Transformer's Positional Encoding, https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/
  - The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), http://jalammar.github.io/illustrated-bert/
  - BERT Word Embeddings Tutorial, https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/
  - Multi-label Text Classificaiton using BERT – The Mighty Transformer, https://medium.com/huggingface/multi-label-text-classification-using-bert-the-mighty-transformer-69714fa3fb3d
  - BERT: State of the Art NLP Model, Explained, https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
  - [Transformer-XL] Attentive Language Models Beyond a Fixed-Length Context, https://aisc.ai.science/events/2019-02-21/
  - What is XLNet and why it outperforms BERT, https://towardsdatascience.com/what-is-xlnet-and-why-it-outperforms-bert-8d8fce710335
  - Paper Dissected: "XLNet: Generalized Autoregressive Pretraining for Language Understanding" Explained, https://mlexplained.com/2019/06/30/paper-dissected-xlnet-generalized-autoregressive-pretraining-for-language-understanding-explained/

# Extra Slides