Part 2. Generative AI

2.1: Generative AI/LLM overview

7/26/2024, Ver 0.4

AI learning modules

Part 1 AI/ML foundation

1.1 AI/ML overview

Recorded on 7/23

Part 2 Generative AI

2.1: Generative AI/LLM overview

2.2: LLM applications

7/26, We're here

Part 3 AI/ML systems

3.1: AI/ML systems overview

3.2: AI/ML systems optimization

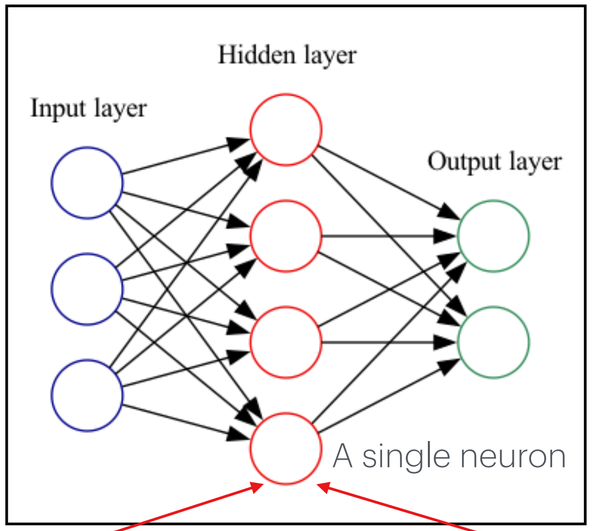
7/30, scheduled

Review

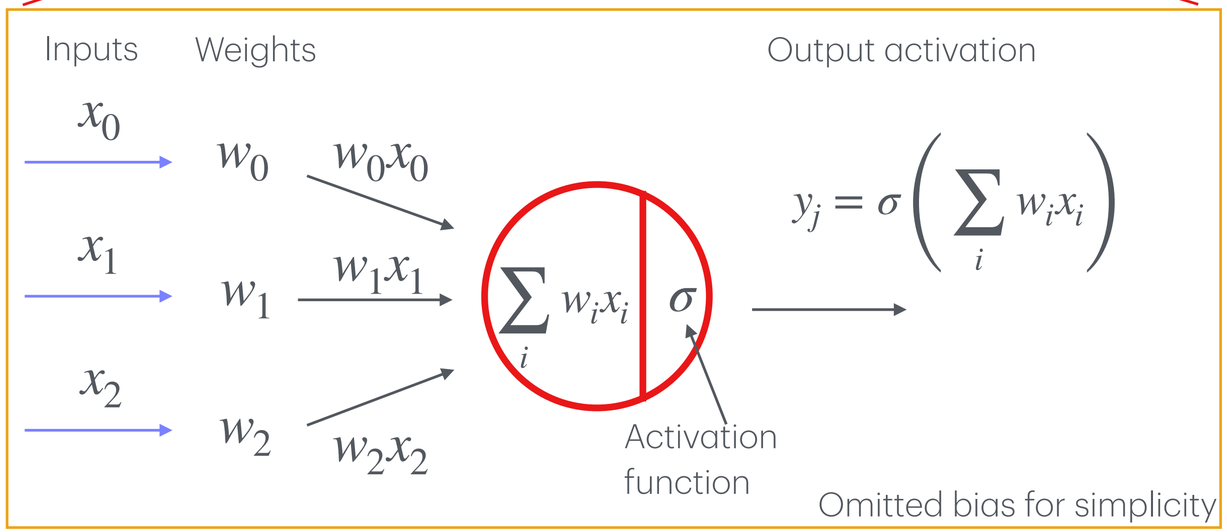
- Machine Learning (ML) learns how to solve problems through data and statistical methods without explicit programming
- Three key ingredients in ML: hypothesis (model), loss (defines the goodness of the model), and optimization (how to find the parameters that minimize the loss).
- A neural network is a basic building block of deep learning and consists of fully connected layers with non-linear activation functions that can learn non-linear boundaries.

ℹ️ Check out the part 1 slide for references, books and neural network code

2 layers neural network with 1 hidden layer



A single neuron internal view



2. Generative AI

Contents

2.1: Generative AI/LLM overview

- Generative model and Transformer based LLM overview

  NLP (natural language processing)

  LLM pre-training

  LLM post-training (fine-tuning, instruction fine-tuning, RLHF)

Motivations?

Large language models are widely used, ChatGPT and Github Copilot, many copilots, and more...

Knowing how language models work will be useful and be interesting

2.1: Generative AI/LLM overview

Plan for today

- Word embedding
- Language model
- RNN
- Sequence to sequence model
- Attention
- Transformer
- Pre-training language model
- Generation in language model

Natural language processing (NLP)

Representation of words

Words can be represented as discrete symbols, such as *one-hot encodings*:

- An example on the right, number of words in the vocabulary (~ 100,000+)

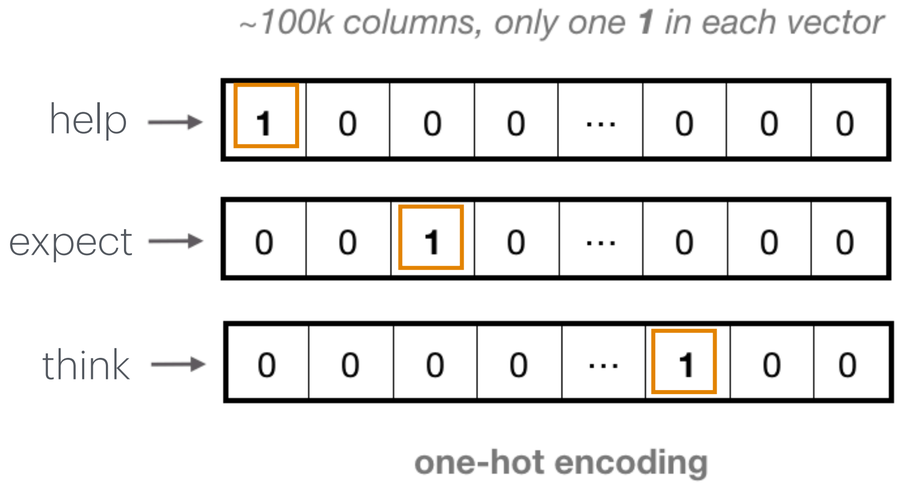


Image source: link

Representation of words

Words can be represented as discrete symbols, such as *one-hot encodings*:

- An example on the right, number of words in the vocabulary (~ 100,000+)

Problem: how to find the similarity of words?

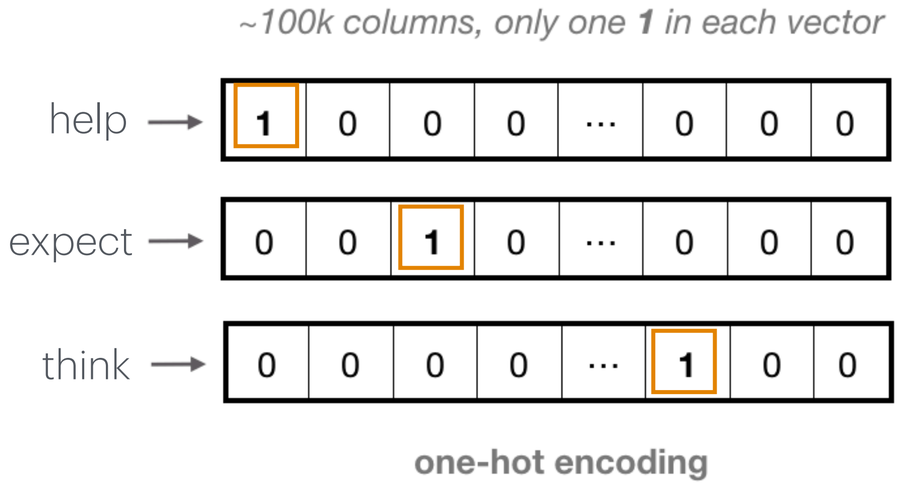


Image source: link

Representation of words

Words can be represented as discrete symbols, such as *one-hot encodings*:

- An example on the right, number of words in the vocabulary (~ 100,000+)

Problem: how to find the similarity of words?

Solution: use a model to learn to encode similarity in the vector themselves. So that vectors of similar meaning will be close to each other.

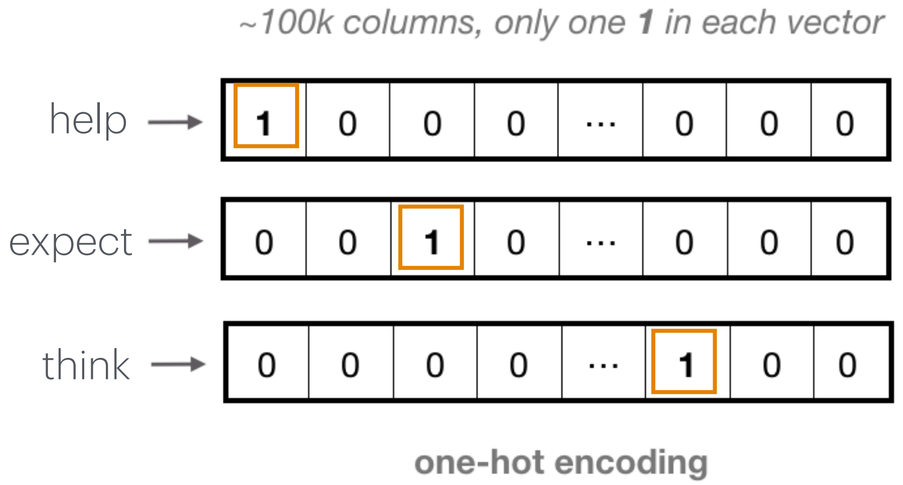


Image source: link

Word embedding

Word2vec: a model that trained to converts *words* to *vector* (word embeddings)

It *learns* the word embedding by training the model to predict probability of given word and its contexts.

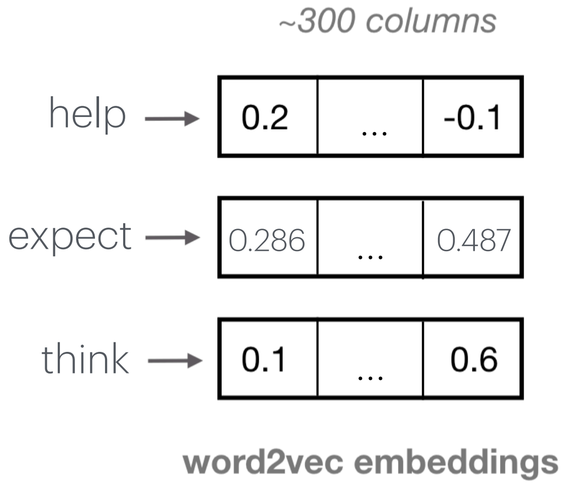


Image source: link

Word embedding

Word2vec: a model that trained to converts *words* to *vector* (word embeddings)

It *learns* the word embedding by training the model to predict probability of given word and its contexts.

Similar words are clustered each other. In the example high dimensional vectors are projected to 2 dim for visualization.

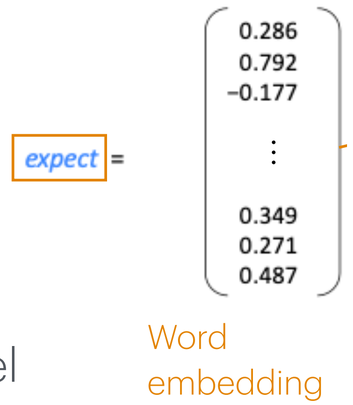


Image source: [5]

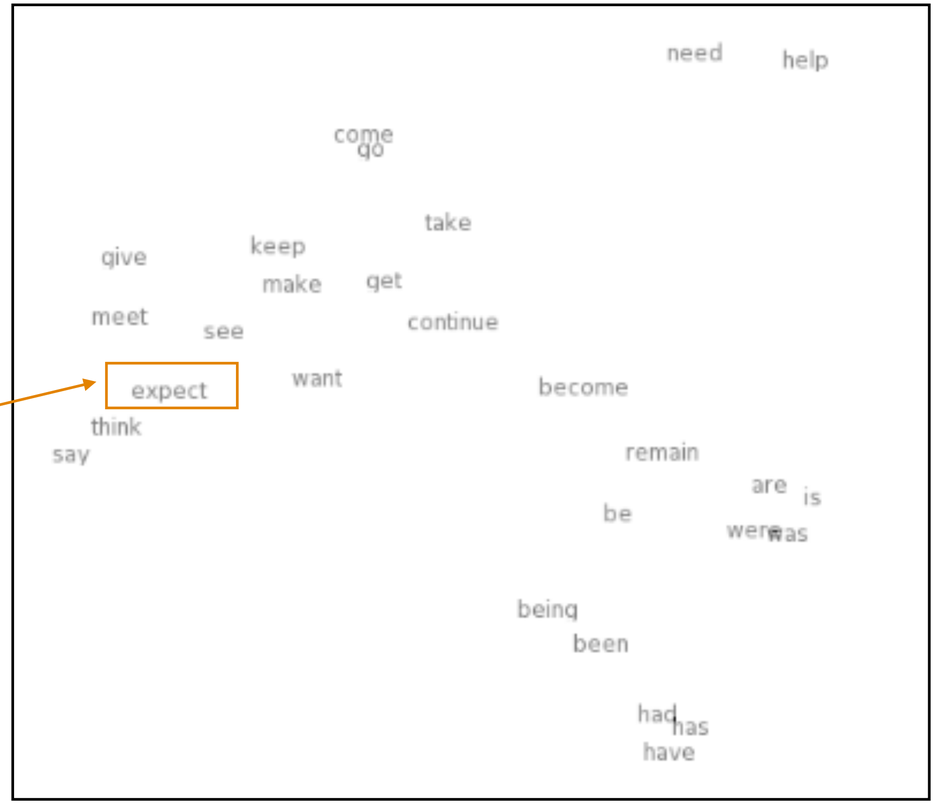
Word embedding

Word2vec: a model that trained to converts *words* to *vector* (word embeddings)

expect =

0.286
0.792
−0.177
⋮
0.349
0.271
0.487

Word embedding



It *learns* the word embedding by training the model to predict probability of given word and its contexts.

Similar words are clustered each other. In the example high dimensional vectors are projected to 2 dim for visualization.

Image source: [5]

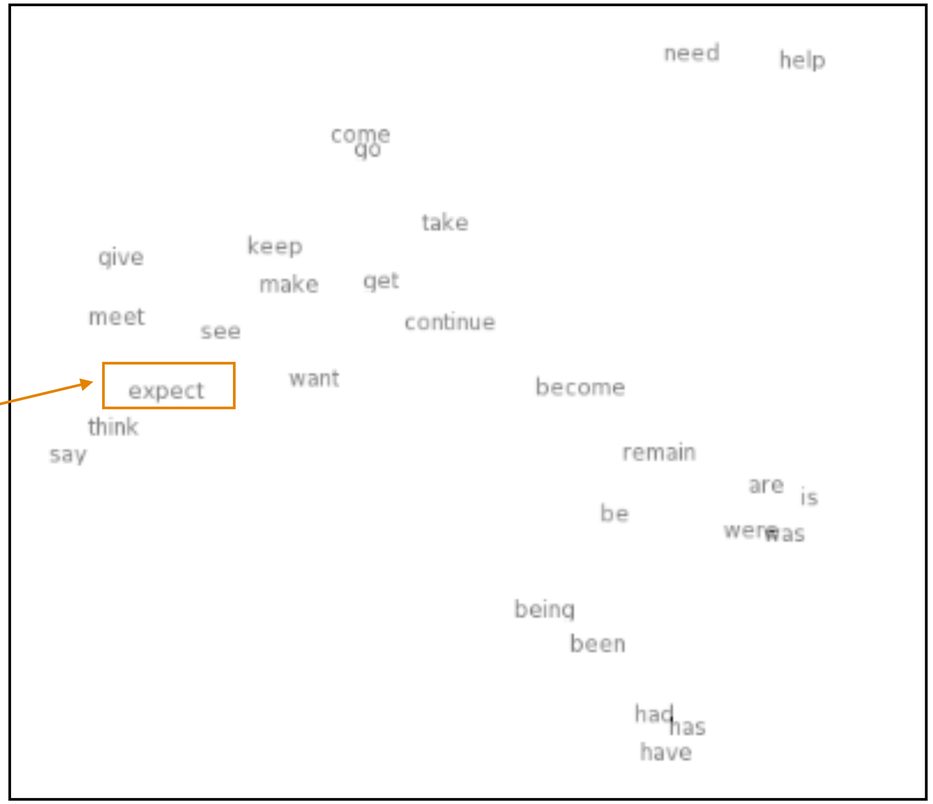
Word embedding

Word2vec: a model that trained to converts *words* to *vector* (word embeddings)

expect =

0.286
0.792
−0.177
⋮
0.349
0.271
0.487

Word embedding



It *learns* the word embedding by training the model to predict probability of given word and its contexts.

Similar words are clustered each other. In the example high dimensional vectors are projected to 2 dim for visualization.

Relationship of words also captured by word embedding. In the example, an answer for ”man:woman=king:?” can be with cosine distance in the embedding space.

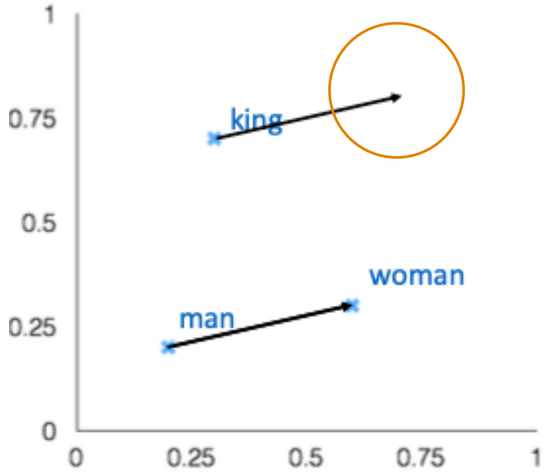


Image source: [5]

Language model

LM (language model): *predicts* next word given input

- Input: text
- Output: predict the next word by sampling

Ref. : [5]

Language model

LM (language model): *predicts* next word given input

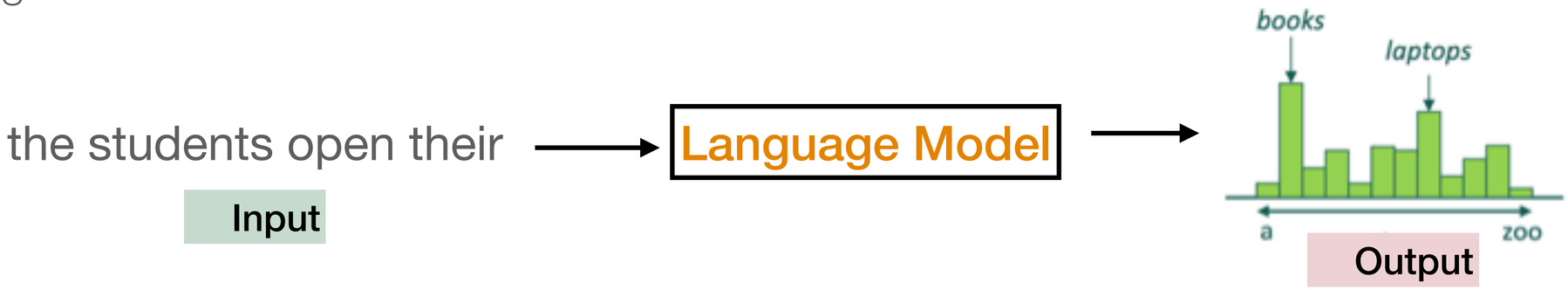
- Input: text
- Output: predict the next word by sampling

Ref. : [5]

Language model

LM (language model): predicts next word given input

- Input: text
- Output: predict the next word by sampling



Ref. : [5]

Language model

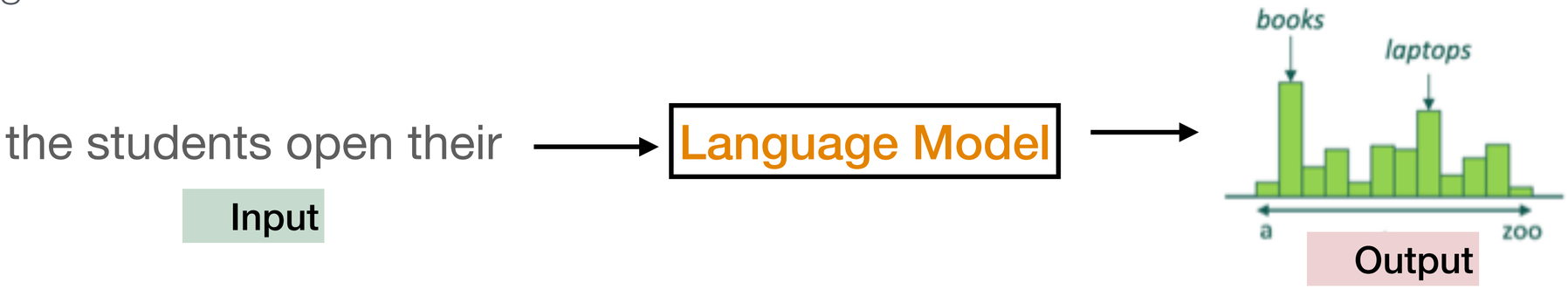
LM (language model): predicts next word given input

- Input: text
- Output: predict the next word by sampling

Formal definition: compute the probability distribution of next word given a sequence of previous words, $x^1, x^2, \ldots, x^t$

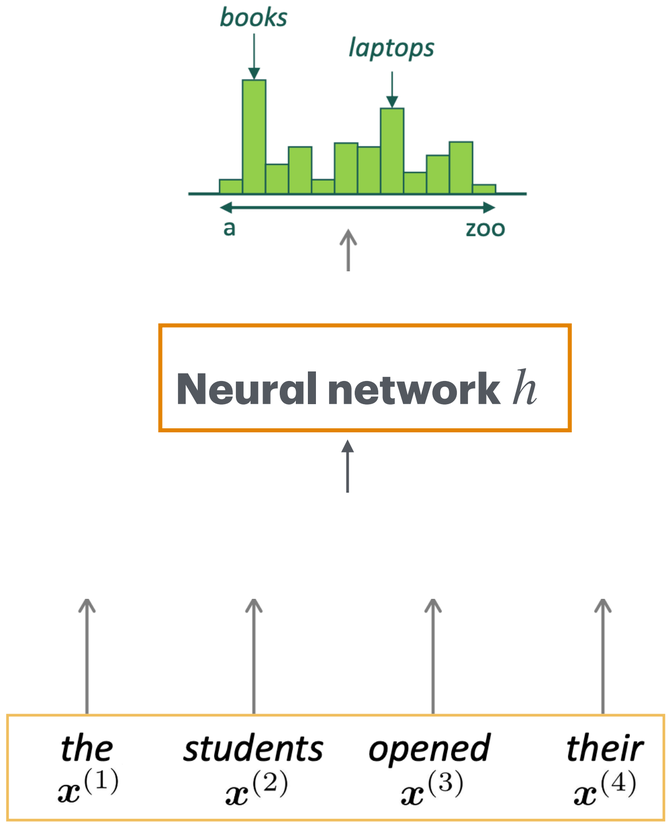
$$P(x^{(t+1)} \mid x^1, \ldots, x^t)$$

where $x^{(t+1)}$ can be a word in the dictionary $V = \{w_1, \ldots w_{|V|}\}$



Ref. : [5]

Neural language model w/ fixed input window



Fixed window sized input, such as 4 words

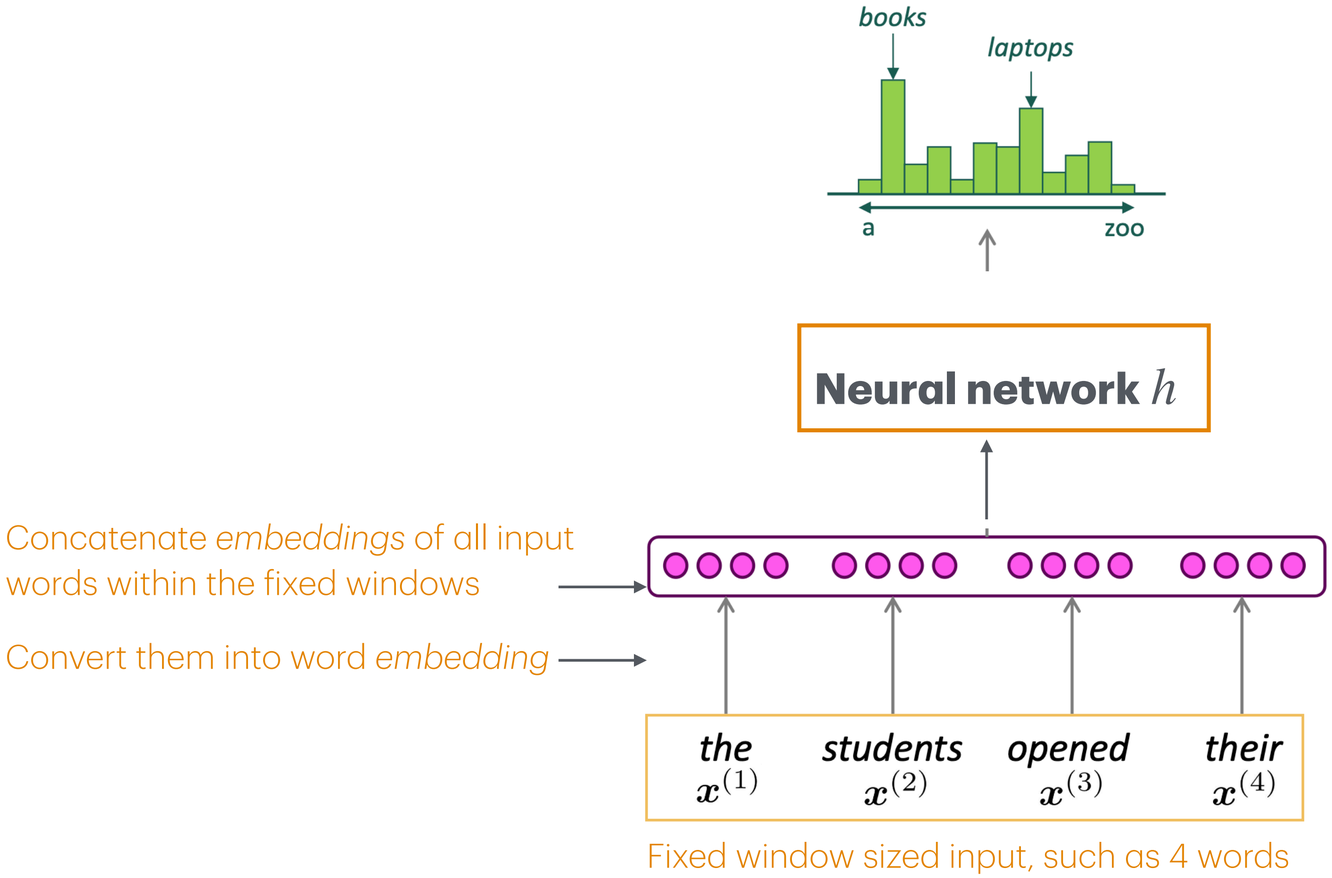
Use neural network to predict next word given input words.

Fixed window neural network based LM

- Keep input within fixed window, and feed to a neural network.
- Neural network based language model predicts the probability of the next word.

Ref. and Image source: [5]

Neural language model w/ fixed input window



Use neural network to predict next word given input words.

Fixed window neural network based LM

- Keep input within fixed window, and feed to a neural network.
- Neural network based language model predicts the probability of the next word.

Ref. and Image source: [5]

Neural language model w/ fixed input window

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

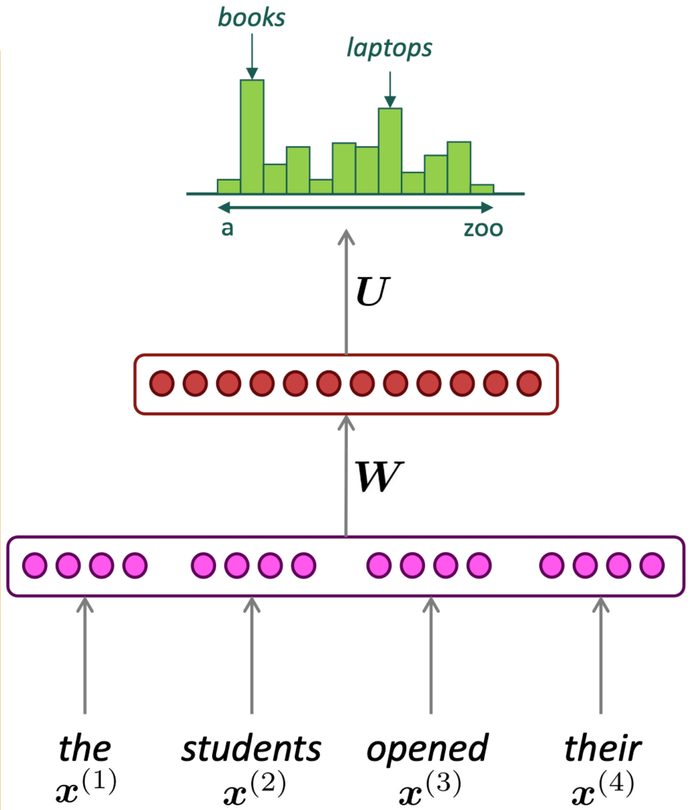
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Fixed window sized input, such as 4 words

Use neural network to predict next word given input words.

Fixed window neural network based LM

- Keep input within fixed window, and feed to a neural network.
- Neural network based language model predicts the probability of the next word.

Problems with this approach?

Ref. and Image source: [5]

Neural language model w/ fixed input window

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

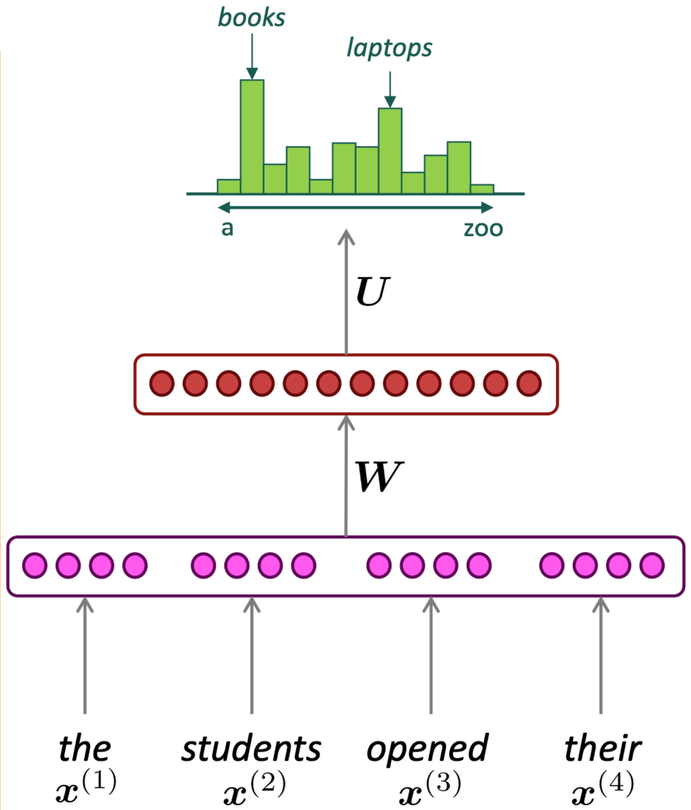
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Fixed window sized input, such as 4 words

Use neural network to predict next word given input words.

Fixed window neural network based LM

- Keep input within fixed window, and feed to a neural network.
- Neural network based language model predicts the probability of the next word.

Problems with this approach?

- Small fixed window
- Larger window means larger model

We need a model that can process any length of input

Ref. and Image source: [5]

RNN (Recurrent neural network)

NN that handles variable length sequential input

Use the same weight $\boldsymbol{W}$ for processing each input.

Allows *stateful* computation since it uses hidden state from previous time $(t-1)$

Any input sequence lengths can be processed

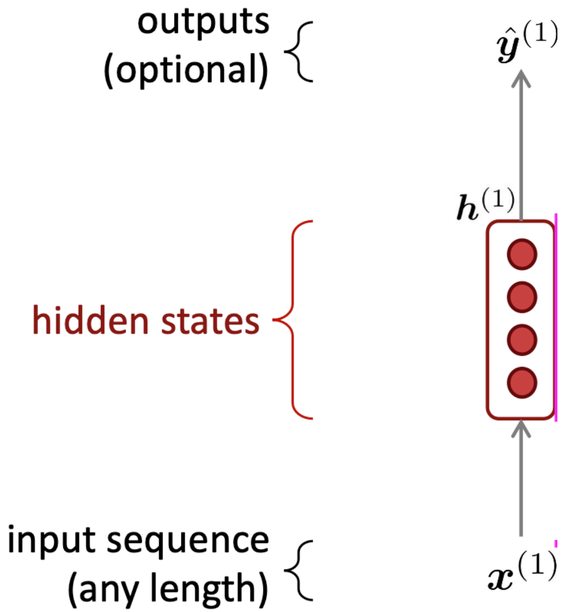


Image source: [5]

RNN (Recurrent neural network)

Use the same weight W for processing each input.

Allows *stateful* computation since it uses hidden state from previous time $(t-1)$

Any input sequence lengths can be processed

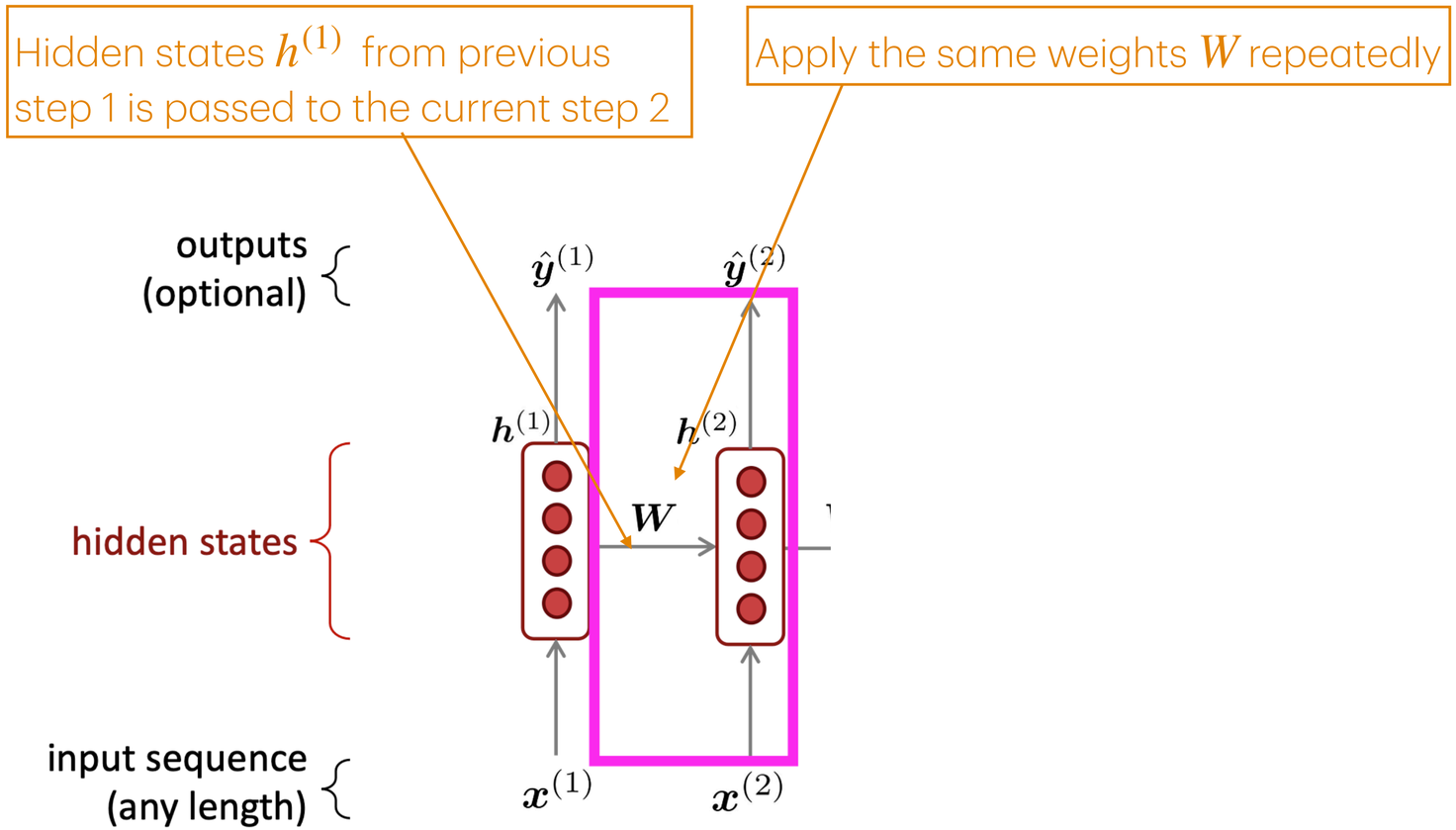


Image source: [5]

RNN (Recurrent neural network)

Use the same weight W for processing each input.

Allows *stateful* computation since it uses hidden state from previous time $(t-1)$

Any input sequence lengths can be processed

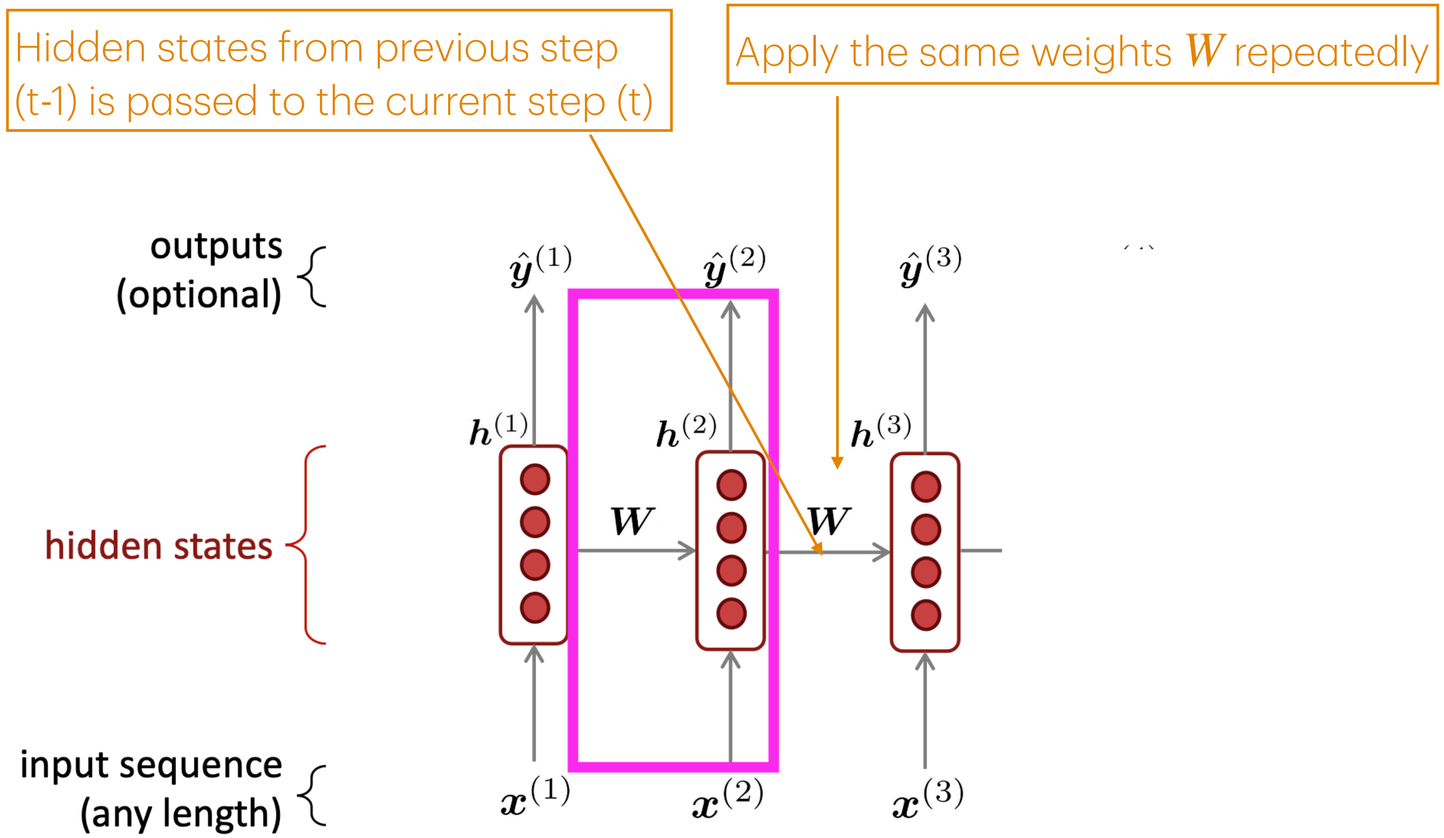


Image source: [5]

RNN (Recurrent neural network)

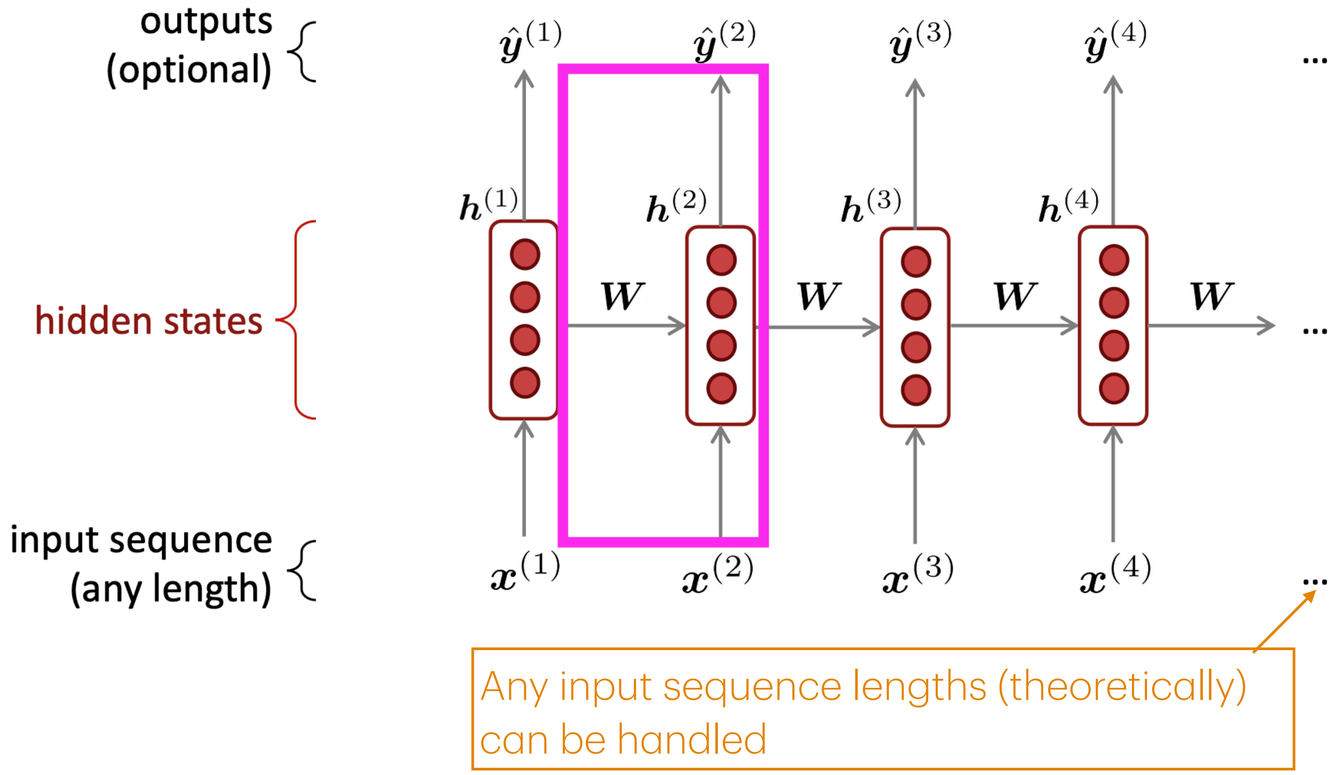
Hidden states from previous step (t-1) is passed to the current step (t)

Apply the same weights W repeatedly

Use the same weight W for processing each input.

Allows *stateful* computation since it uses hidden state from previous time $(t-1)$

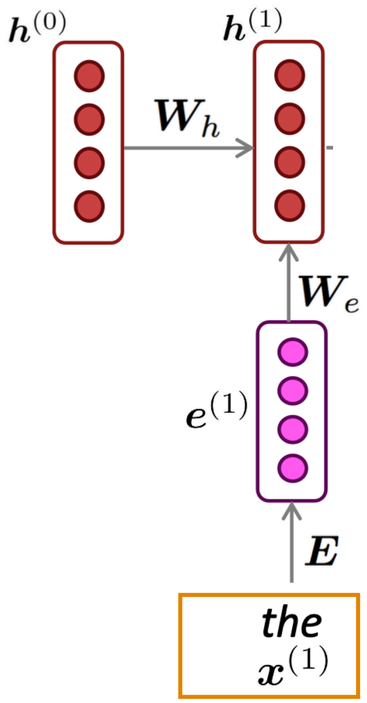
Any input sequence lengths can be processed



Any input sequence lengths (theoretically) can be handled

Image source: [5]

RNN based Language model

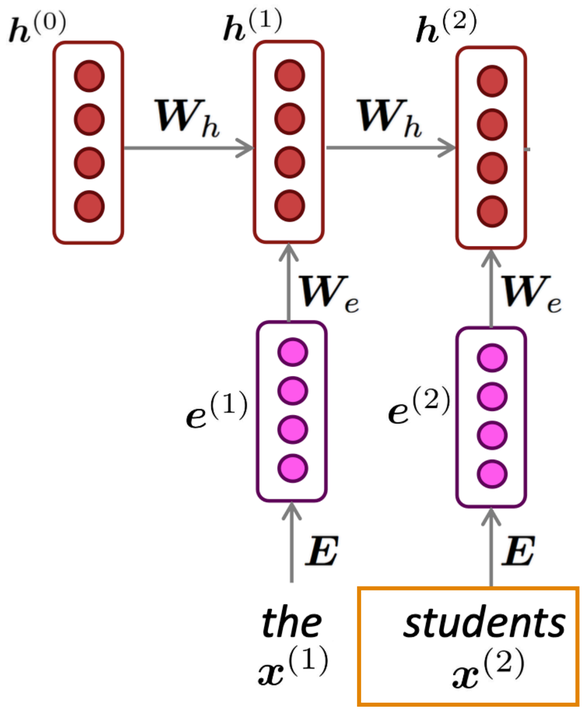


Compared to previous neural network based LM, RNN based LM can handle much *longer input sequence* now.

Sequential states are processed using hidden state from the previous step.

Image source: [5]

RNN based Language model

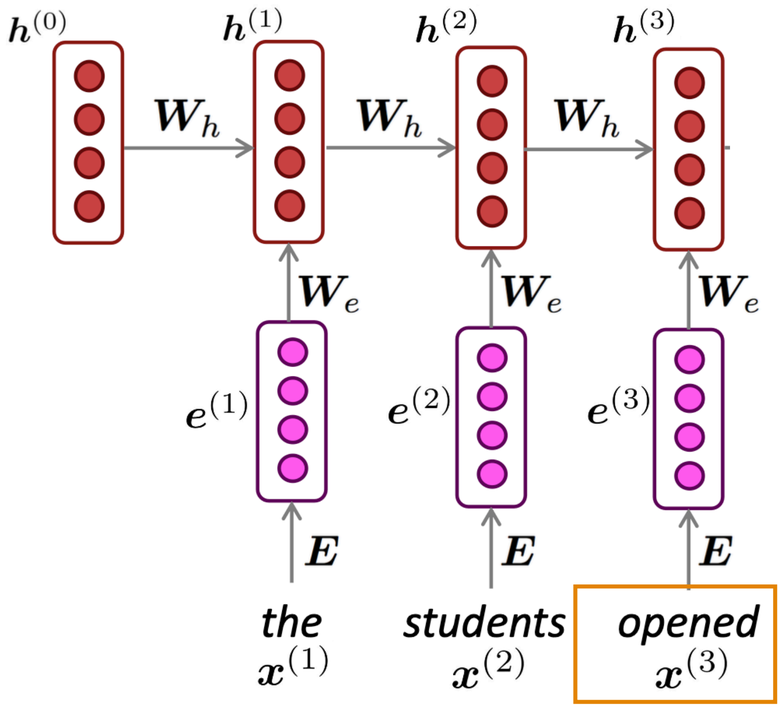


Compared to previous neural network based LM, RNN based LM can handle much *longer input sequence* now.

Sequential states are processed using hidden state from the previous step.

Image source: [5]

RNN based Language model

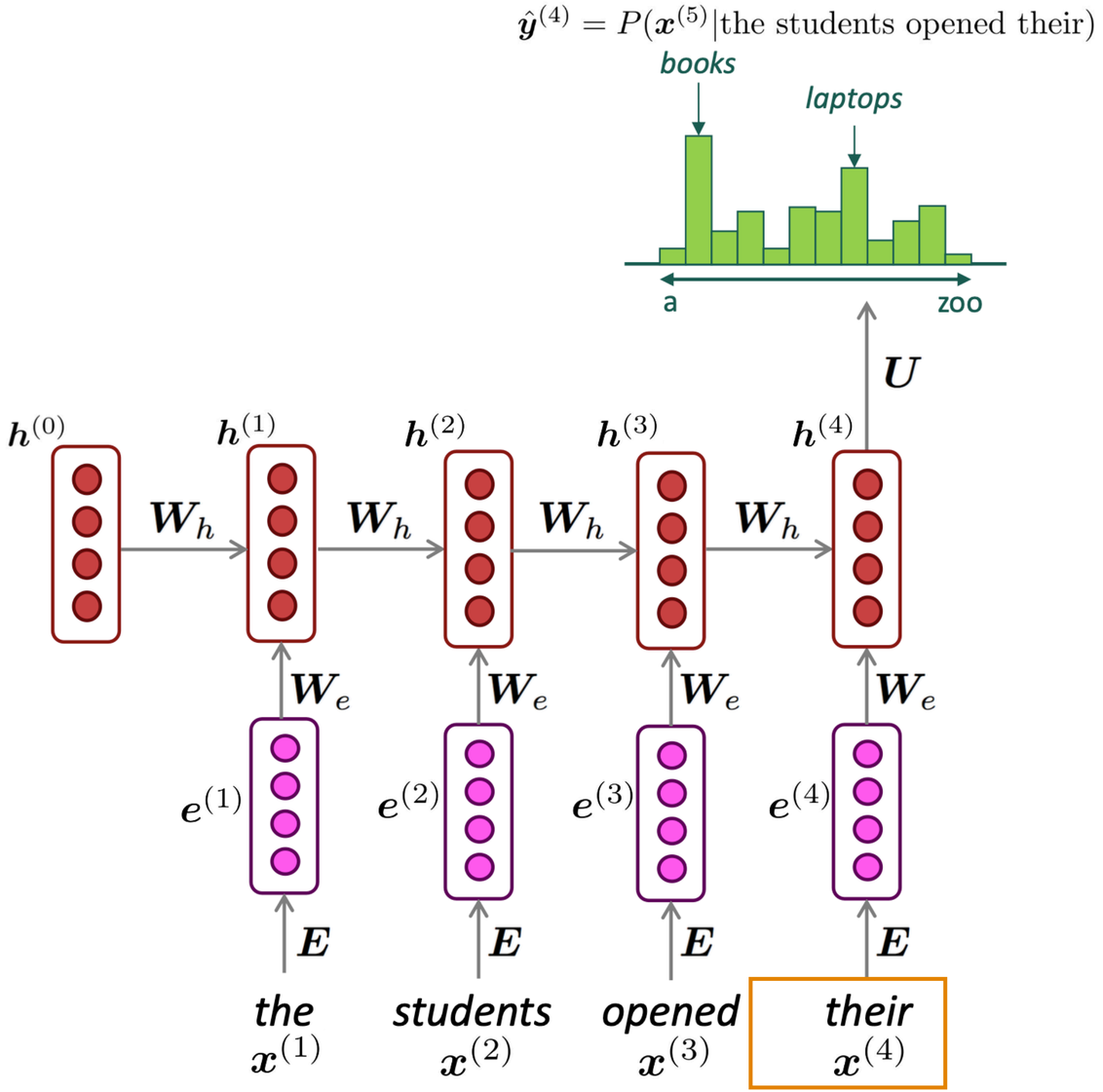


Compared to previous neural network based LM, RNN based LM can handle much *longer input sequence* now.

Sequential states are processed using hidden state from the previous step.

Image source: [5]

RNN based Language model



Compared to previous neural network based LM, RNN based LM can handle much *longer input sequence* now.

Sequential states are processed using hidden state from the previous step.

Image source: [5]

RNN based Language model

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

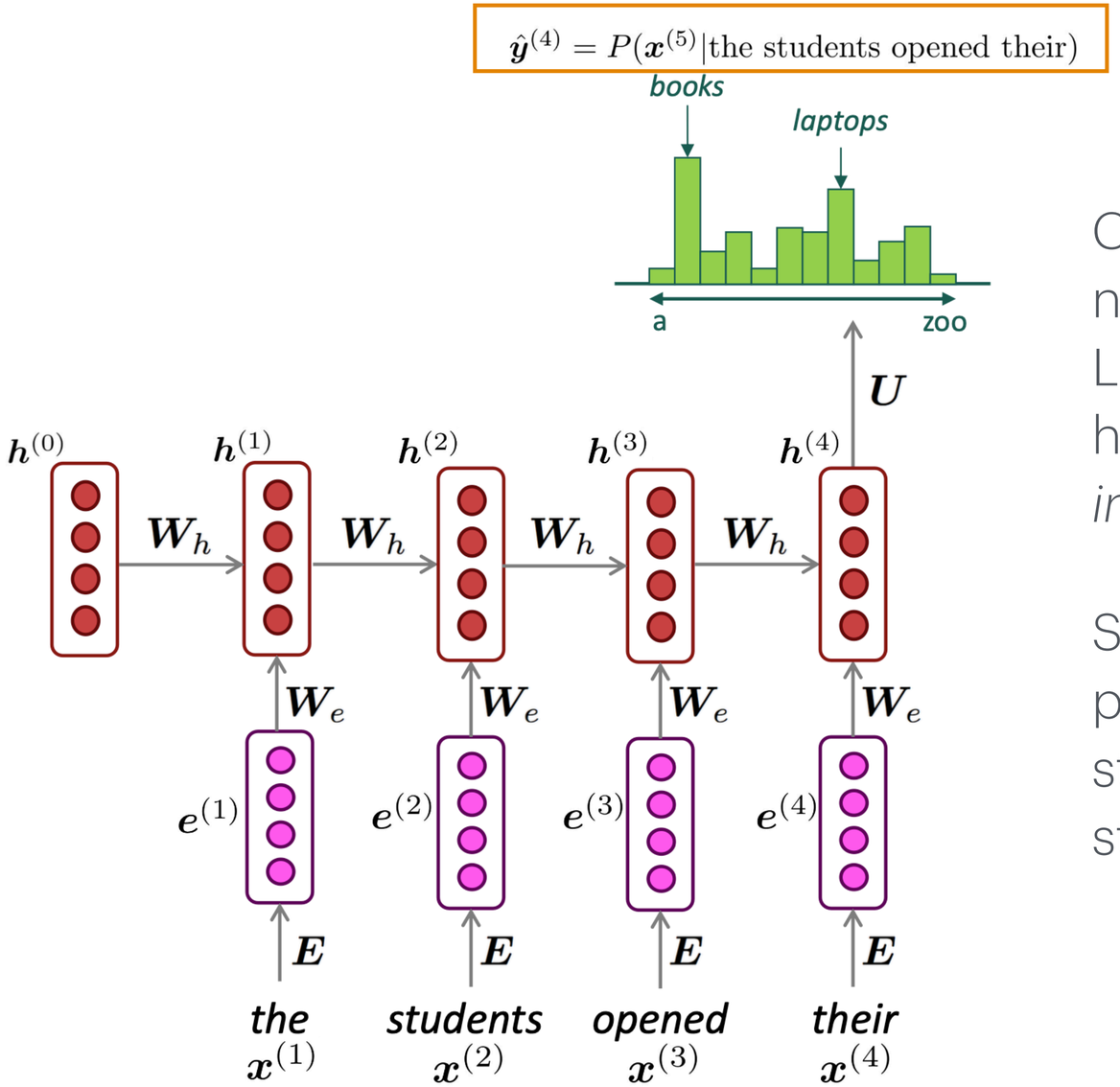
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



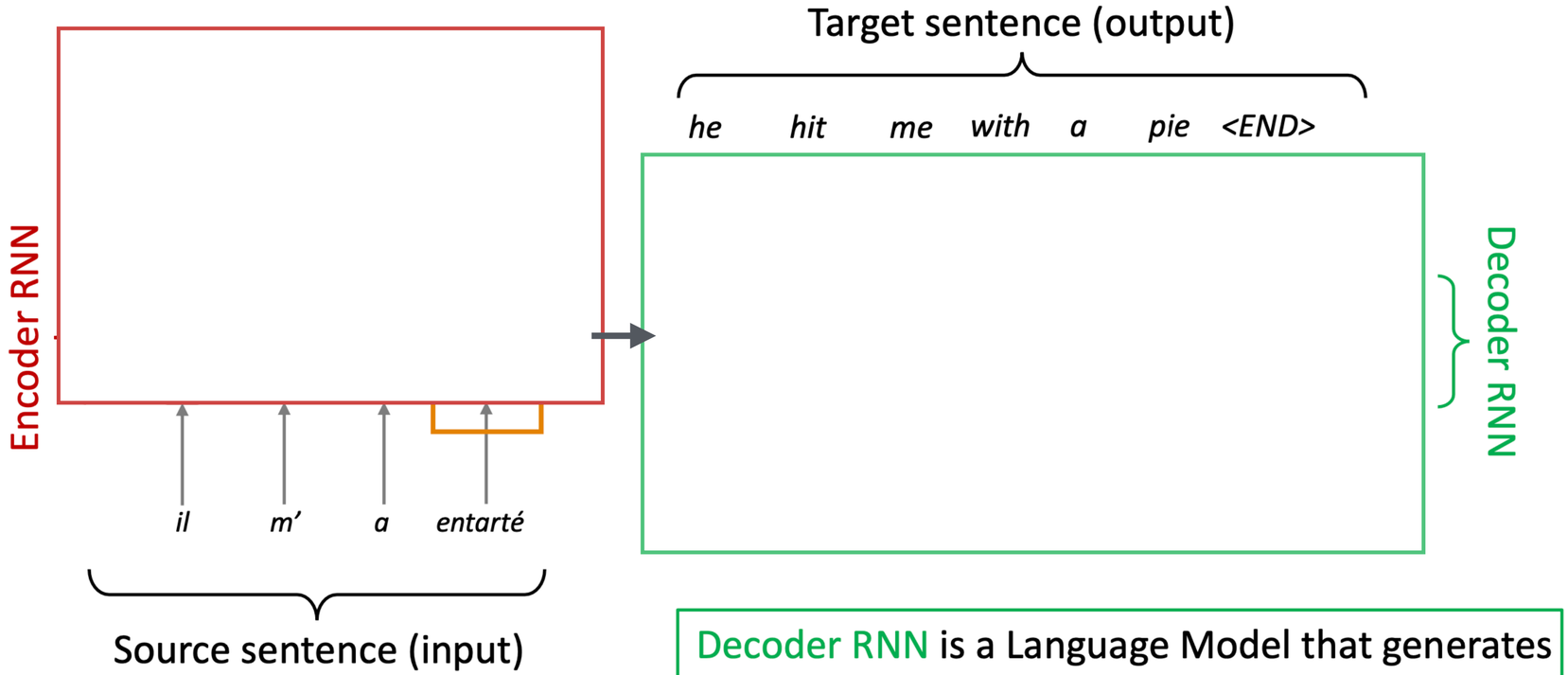
Compared to previous neural network based LM, RNN based LM can handle much *longer input sequence* now.

Sequential states are processed using hidden state from the previous step.

Image source: [5]

Neural Machine Translation (NMT)

The sequence-to-sequence model



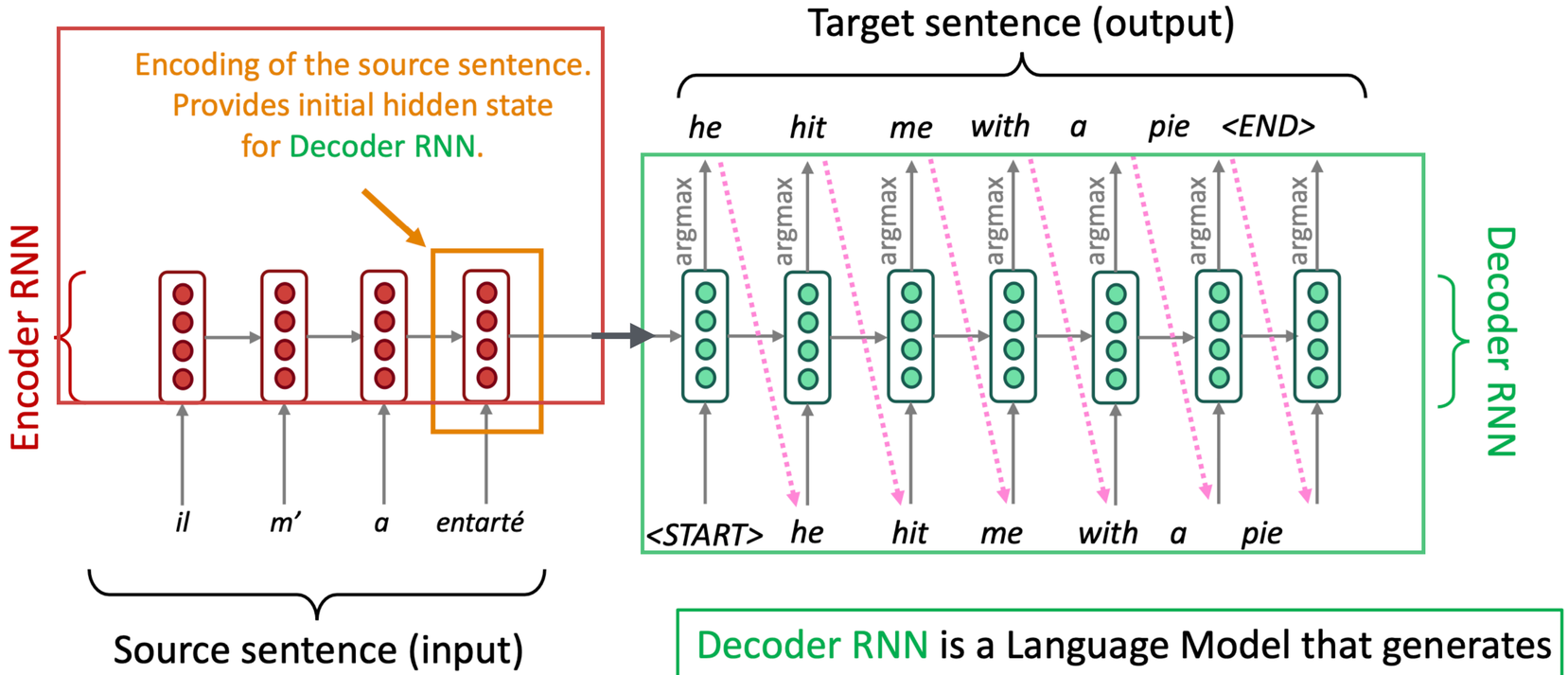
Neural machine translation model consist of encoder part and decoder part

- *Encoder*: produces an encoding of the source sentence
- *Decoder*: a Language Model that generates target sentence, conditioned on encoding

Slide source: [5]

Neural Machine Translation (NMT)

The sequence-to-sequence model



Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Note: This diagram shows **test time** behavior: decoder output is fed in ⋯→ as next step's input

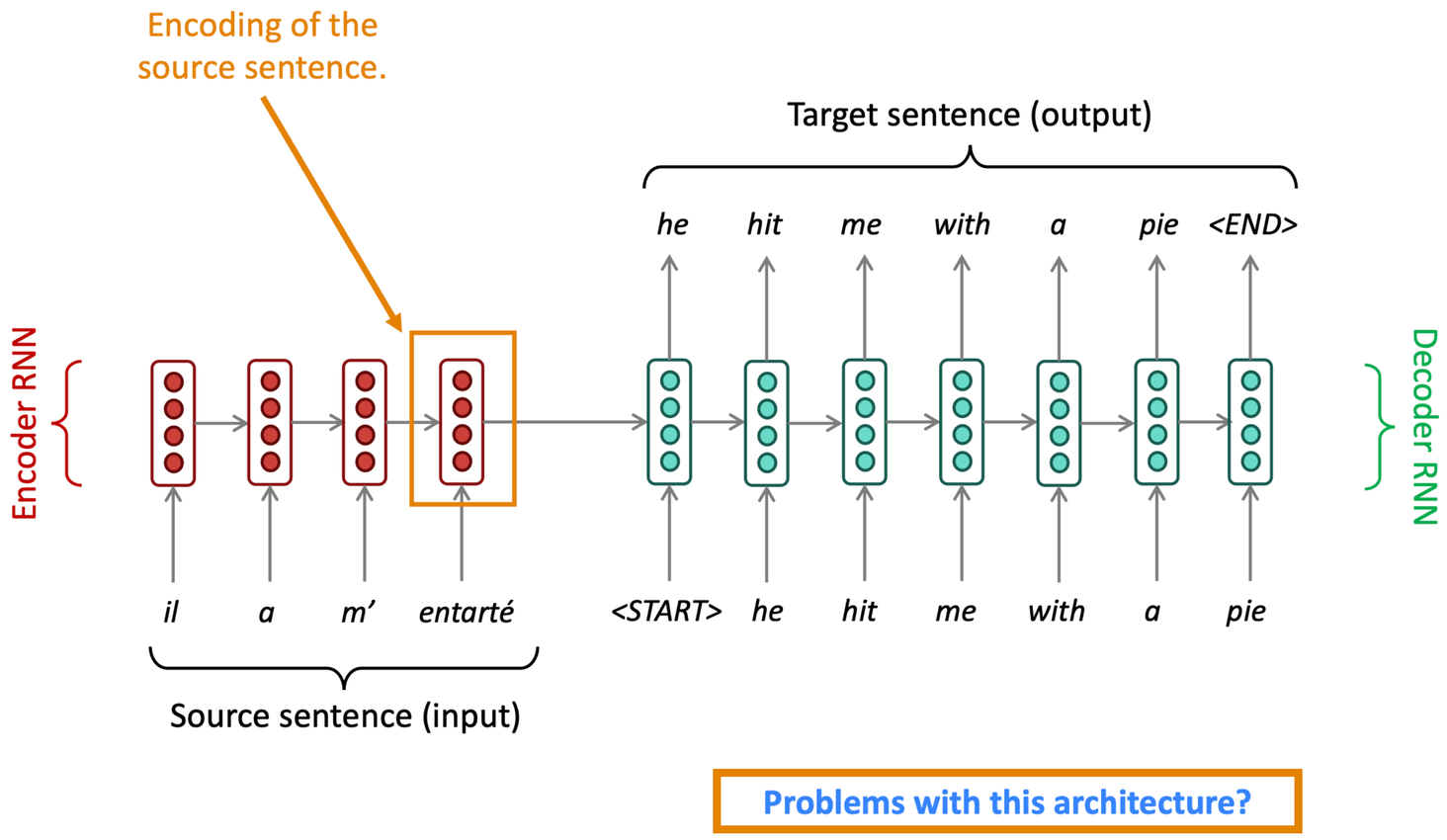
Neural machine translation model consist of encoder part and decoder part

- *Encoder*: produces an encoding of the source sentence
- *Decoder*: a Language Model that generates target sentence, conditioned on encoding

Slide source: [5]

Attention to solve information bottleneck

Sequence to sequence model has a bottleneck problem

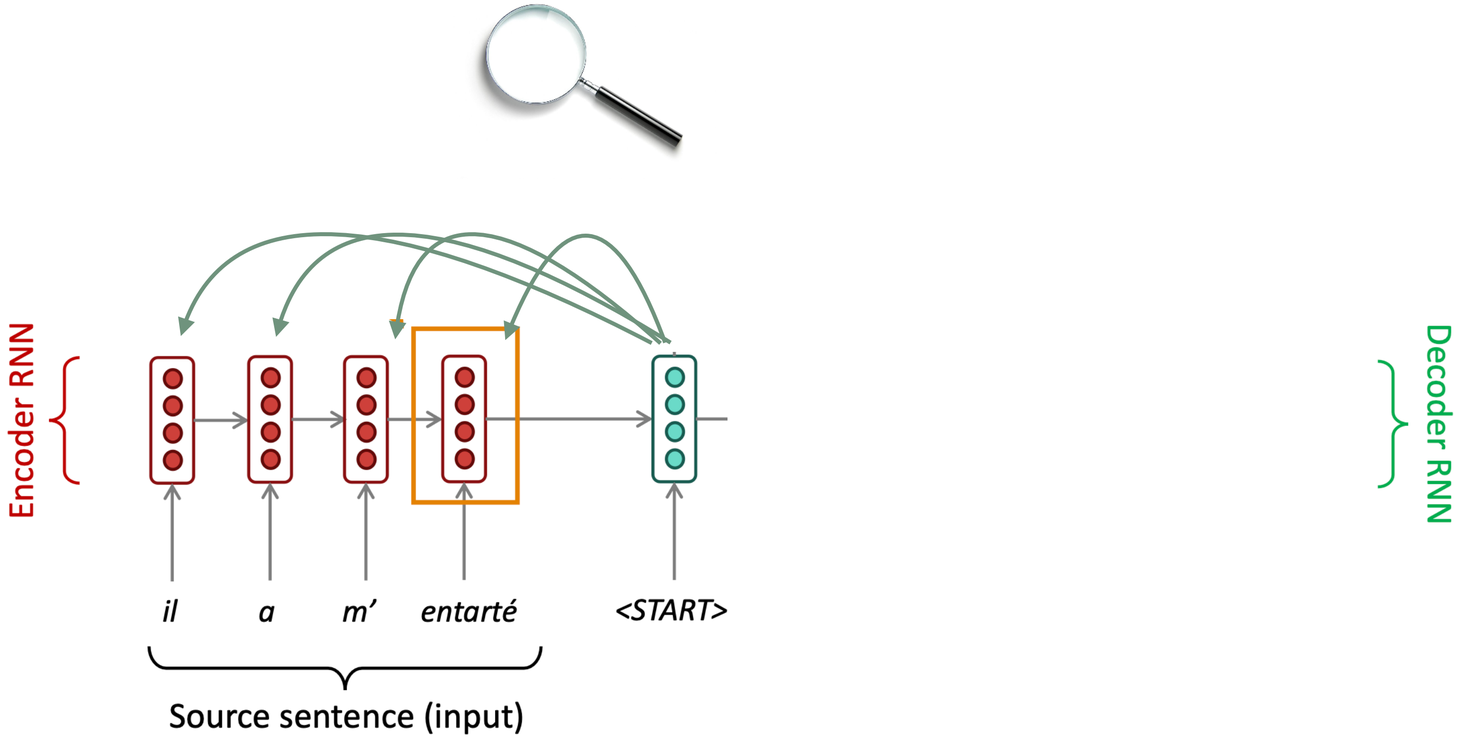


Information bottleneck: encoding of the source sentence needs to capture all information about the input text

Image source: [5]

Attention to solve information bottleneck

Sequence to sequence model has a bottleneck problem



Information bottleneck: encoding of the source sentence needs to capture all information about the input text

Attention is a mechanism to solve the bottleneck. On each step on the decoder, directly compute the similarities with all input and focus on the specific part of the input text.

Image source: [5]

Summary: Key technical concepts

1. Word embedding: represents words as dense vector in a high-dimensional space that captures semantic relationships and similarities between them

2. Language model: a model predicts the probability distribution of next word given previous words

3. RNN: a type of neural network that processes sequential data by maintaining a hidden state that captures information from previous time steps.

4. Sequence to sequence model: a model transforms a sequence of data into another, which consists of encoder and decoder. Neural machine translation uses this model

5. Attention: it enables the models to focus on specific parts of the input sequence and improving in handling long-context in sequential data.

Does RNN model solve NLP problems?

Slow training due to the sequential nature of hidden state propagation

Hard to parallelize training:

- Forward and backward propagation takes O(seq len) and not parallelizable while GPUs can perform many parallel tasks at once

Hard to learn long-distance dependencies

- It takes O(seq len) steps to model the interaction between two words

Does RNN model solve NLP problems?

Slow training due to the sequential nature of hidden state propagation

Hard to parallelize training:

- Forward and backward propagation takes O(seq len) and not parallelizable while GPUs can perform many parallel tasks at once

Hard to learn long-distance dependencies

- It takes O(seq len) steps to model the interaction between two words

"Attention is all you need" proposed a solution to the above problems:

- Transformer architecture: a parallelizable model that can effectively attend to all sequence

Transformer

Transformer model

Transformer architecture/model improves the issues with RNN, such as providing attention paths, enabling parallel computations.

It achieved great success in NLP (spoiler alert, GPT uses decoder part of this architecture)

It also shows many promises outside of NLP, protein folding, computer vision and sequential data processing.

First transform model consists of *encoder* and *decoder* parts (as was in the sequence to sequence model) for translation task.

Transformer block is a core block for the Transformer architecture, which includes multi-headed self attention and feedforward layer.

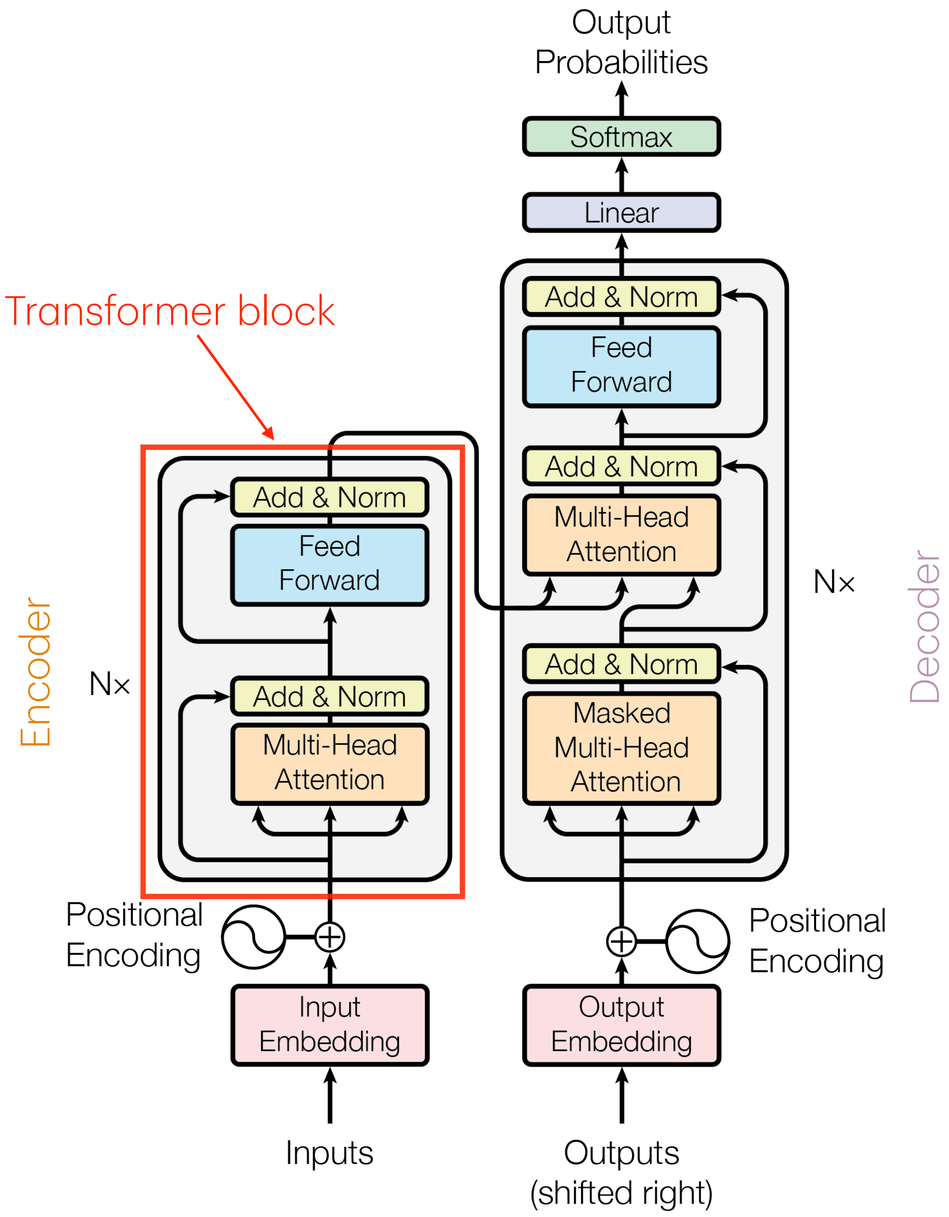


Image source from "Attention is all you need paper", link

Transformer

Transformer architecture uses a layers of *transformer block* (Transformer_block()) which has self-attention* and feedforward network layers to process natural language (sequential data).

$$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$$

Where superscript $i, i+1$th indicate layers, subscript t indicates time step.

For decoder mode language model (an example on the right), self-attention layers attend to all the previous states, which is indicated by red arrow.

*: for simplicity a single self-attention is used for the description; however, there are multiple self attentions, i.e. multi-head attention (MHA).

Ref: [2]

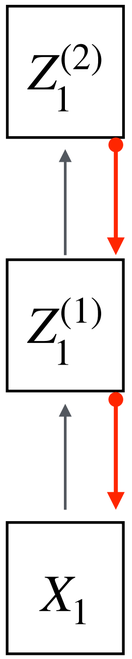
Transformer

Transformer architecture uses a layers of *transformer block* (Transformer_block()) which has self-attention* and feedforward network layers to process natural language (sequential data).

$$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$$

Where superscript $i, i+1$th indicate layers, subscript t indicates time step.

For decoder mode language model (an example on the right), self-attention layers attend to all the previous states, which is indicated by red arrow.



*: for simplicity a single self-attention is used for the description; however, there are multiple self attentions, i.e. multi-head attention (MHA).

Ref: [2]

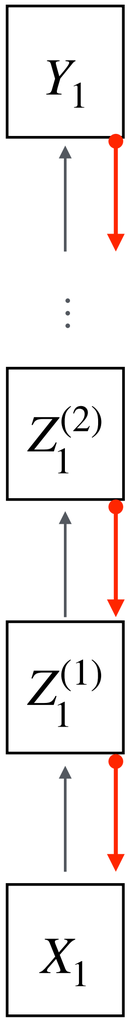
Transformer

Transformer architecture uses a layers of *transformer block* (Transformer_block()) which has self-attention* and feedforward network layers to process natural language (sequential data).

$$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$$

Where superscript $i, i+1$th indicate layers, subscript t indicates time step.

For decoder mode language model (an example on the right), self-attention layers attend to all the previous states, which is indicated by red arrow.



*: for simplicity a single self-attention is used for the description; however, there are multiple self attentions, i.e. multi-head attention (MHA).

Ref: [2]

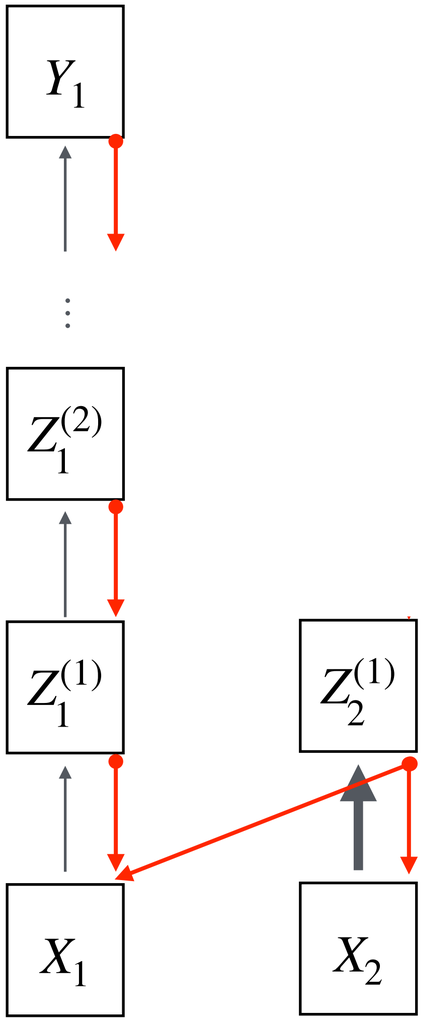
Transformer

Transformer architecture uses a layers of *transformer block* (Transformer_block()) which has self-attention* and feedforward network layers to process natural language (sequential data).

$$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$$

Where superscript $i, i+1$th indicate layers, subscript t indicates time step.

For decoder mode language model (an example on the right), self-attention layers attend to all the previous states, which is indicated by red arrow.



*: for simplicity a single self-attention is used for the description; however, there are multiple self attentions, i.e. multi-head attention (MHA).

Ref: [2]

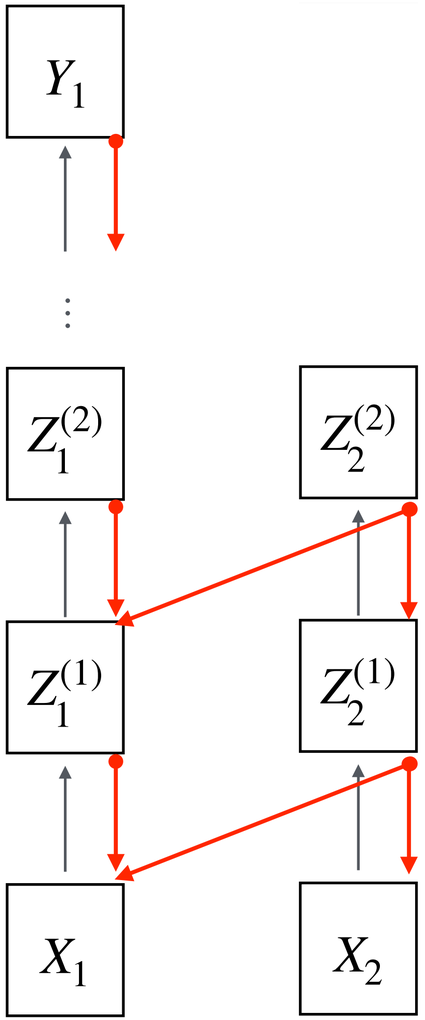
Transformer

Transformer architecture uses a layers of *transformer block* (Transformer_block()) which has self-attention* and feedforward network layers to process natural language (sequential data).

$$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$$

Where superscript $i, i+1$th indicate layers, subscript t indicates time step.

For decoder mode language model (an example on the right), self-attention layers attend to all the previous states, which is indicated by red arrow.



*: for simplicity a single self-attention is used for the description; however, there are multiple self attentions, i.e. multi-head attention (MHA).

Ref: [2]

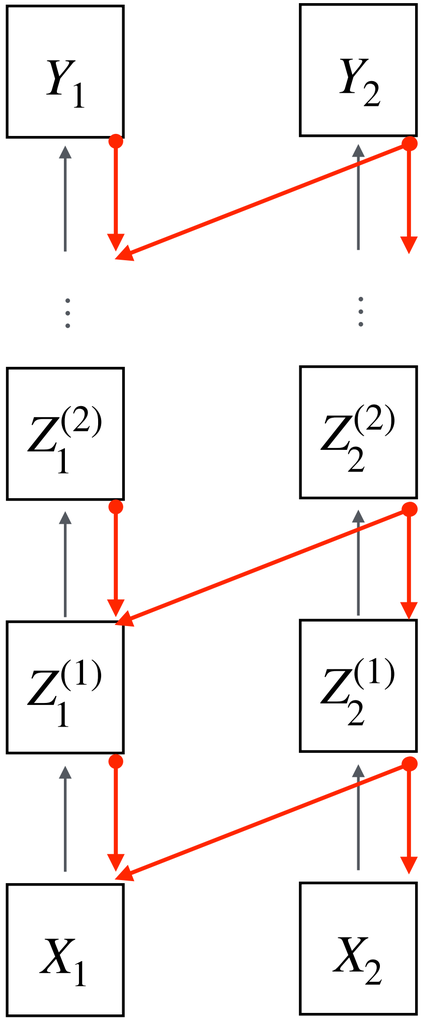
Transformer

Transformer architecture uses a layers of *transformer block* (Transformer_block()) which has self-attention* and feedforward network layers to process natural language (sequential data).

$$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$$

Where superscript $i, i+1$th indicate layers, subscript t indicates time step.

For decoder mode language model (an example on the right), self-attention layers attend to all the previous states, which is indicated by red arrow.



*: for simplicity a single self-attention is used for the description; however, there are multiple self attentions, i.e. multi-head attention (MHA).

Ref: [2]

Transformer

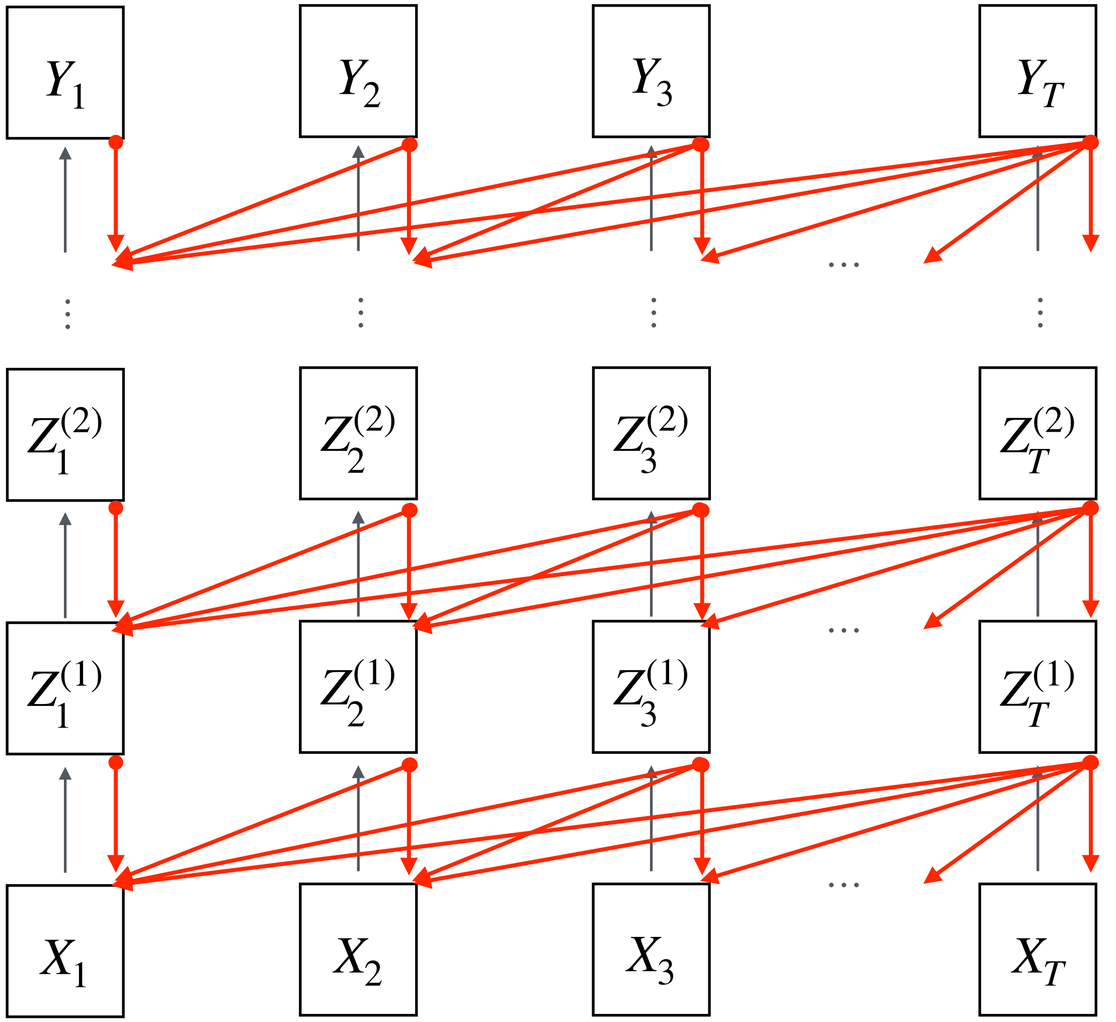
GPT-3 has 96 layers and can handle 2048 time steps

Transformer architecture uses a layers of *transformer block* (Transformer_block()) which has self-attention* and feedforward network layers to process natural language (sequential data).

$$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$$

Where superscript $i, i+1$th indicate layers, subscript t indicates time step.

For decoder mode language model (an example on the right), self-attention layers attend to all the previous states, which is indicated by red arrow.



*: for simplicity a single self-attention is used for the description; however, there are multiple self attentions, i.e. multi-head attention (MHA).

Ref: [2]

Tokens and embedding

Token*: input words maps to sub-word characters called tokens. Sentence piece, byte pair encoding tokenizer. Word will map to one or more token ids.

Tokenizer example: 48 words, 58 tokens. Tokens are shown as a same color characters.

```
OpenAI's large language models (sometimes referred to as GPT's) process
 text using tokens, which are common sequences of characters found in a
 set of text. The models learn to understand the statistical
 relationships between these tokens, and excel at producing the next
 token in a sequence of tokens.
```

Embedding: tokens will map to learned embeddings. It maps token id to embedding vector. This is the actual input to Transformer model. $X \in \mathbb{R}^{N \times d}$, where N is sequence length and d is hidden dimension

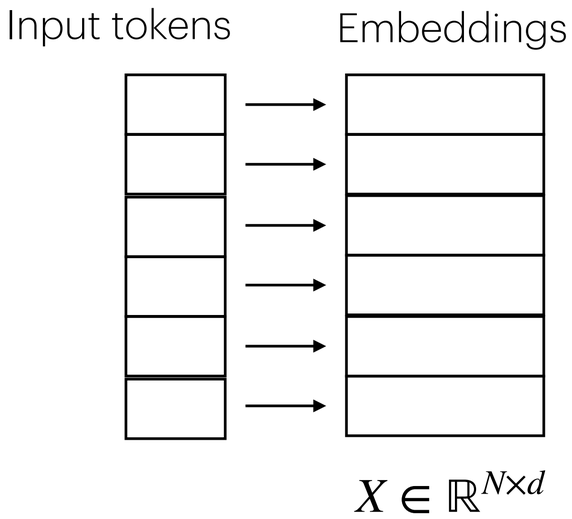


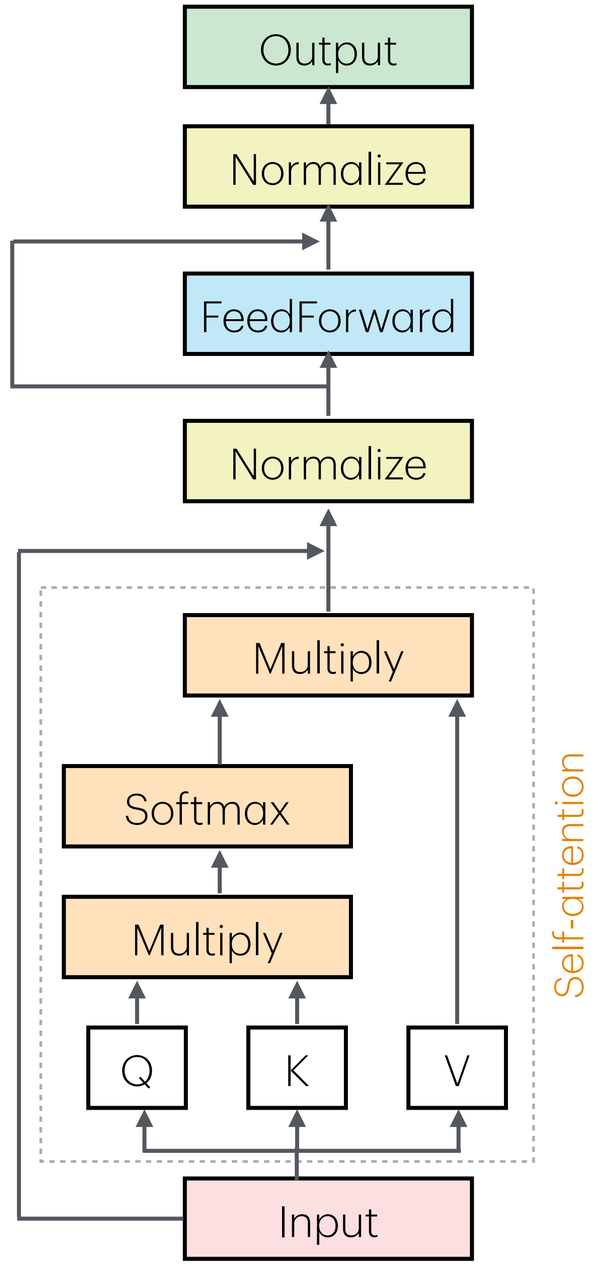
Image source: link

*: in this slide, token and word will be interchangeably used unless we need to use 'token' specifically

Transformer block

Transformer block consists of self-attention and feedforward network.

- Self-attention
- Feedforward network
- Layer norm and residual connection



Self-attention*

Analogous example of the Query-Key-Value design (using YouTube search as an example)

• Query: text prompt in the search bar

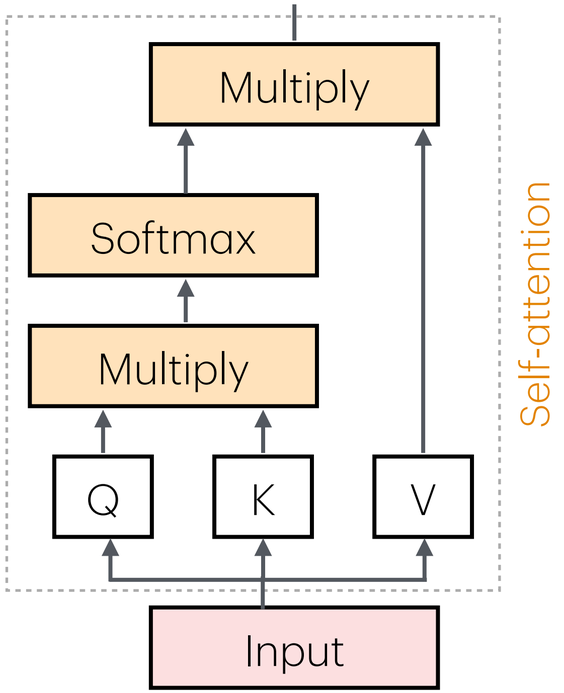
• Key: the titles/descriptions of videos

• Value: the corresponding videos

Input, X, is projected to query, key, values (Q, K, V), where $Q \in \mathbb{R}^{N\times d}$, $K^T \in \mathbb{R}^{d\times N}$, $V \in \mathbb{R}^{N\times d}$, where N is sequence length and d is hidden dimension.

Compute the inner product of Q and K as $S = \frac{(QK^T)}{\sqrt{d}} \in \mathbb{R}^{N\times N}$, then apply softmax row-wise, $A = softmax(S) \in \mathbb{R}^{N\times N}$. A causal mask is applied to attention matrix for the decoder model.

Then, multiply V to attention to get output of the self-attention as $P = AV \in \mathbb{R}^{N\times d}$



*: for simplicity a single self-attention is used for the description; however, there are multiple self attentions, i.e. multi-head attention (MHA).

Content borrowed from link

Feedforward network

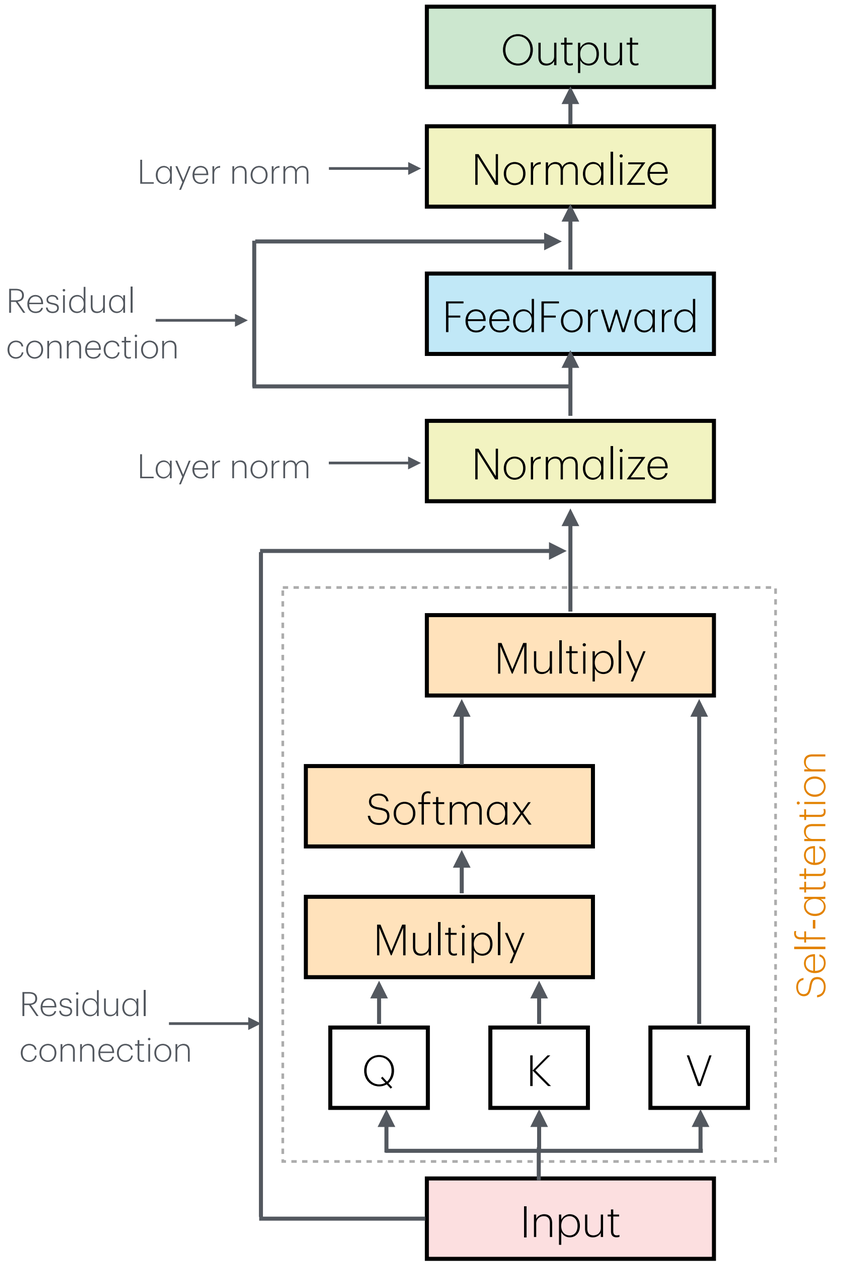
Self-attention layer computes relationship between inputs, but there is *no non-linearity*.

Apply feedforward network (fully connected layer with ReLU). It uses larger hidden size ($4d$), where O is output from the layer norm after the self-attention layer.

$$FFN(O) = \text{ReLU}(OW_1 + b_1)W_2 + b_2$$

Layer normalization and residual connections are used to improve training.

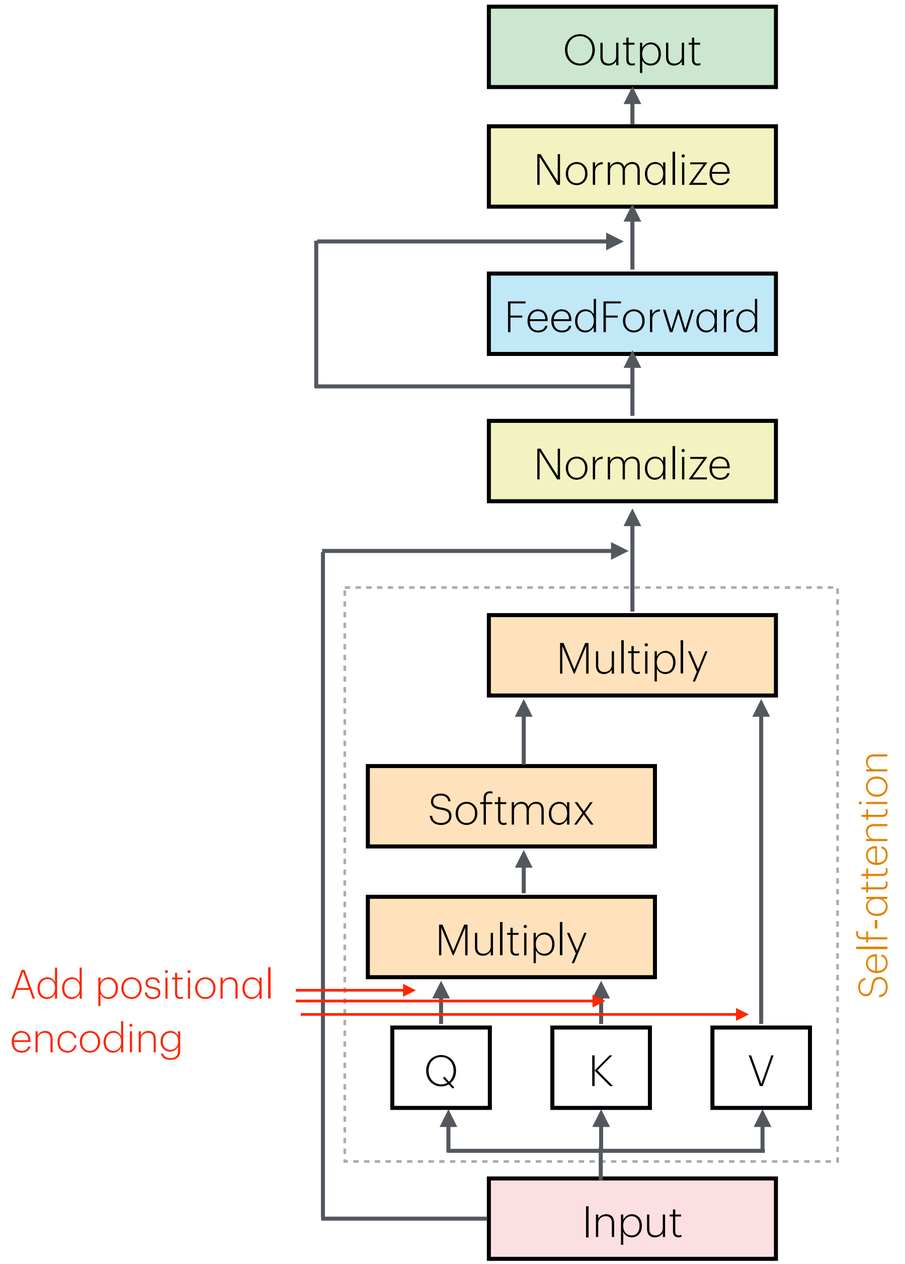
Layer normalization does the normalization for each input, then applied a learnable affine transformation.



Ref: [1]

Positional encoding

Issue: no explicit ordering information, permutation invariant. So we need to encode order to the inputs.



Ref: [2, 5]

Positional encoding

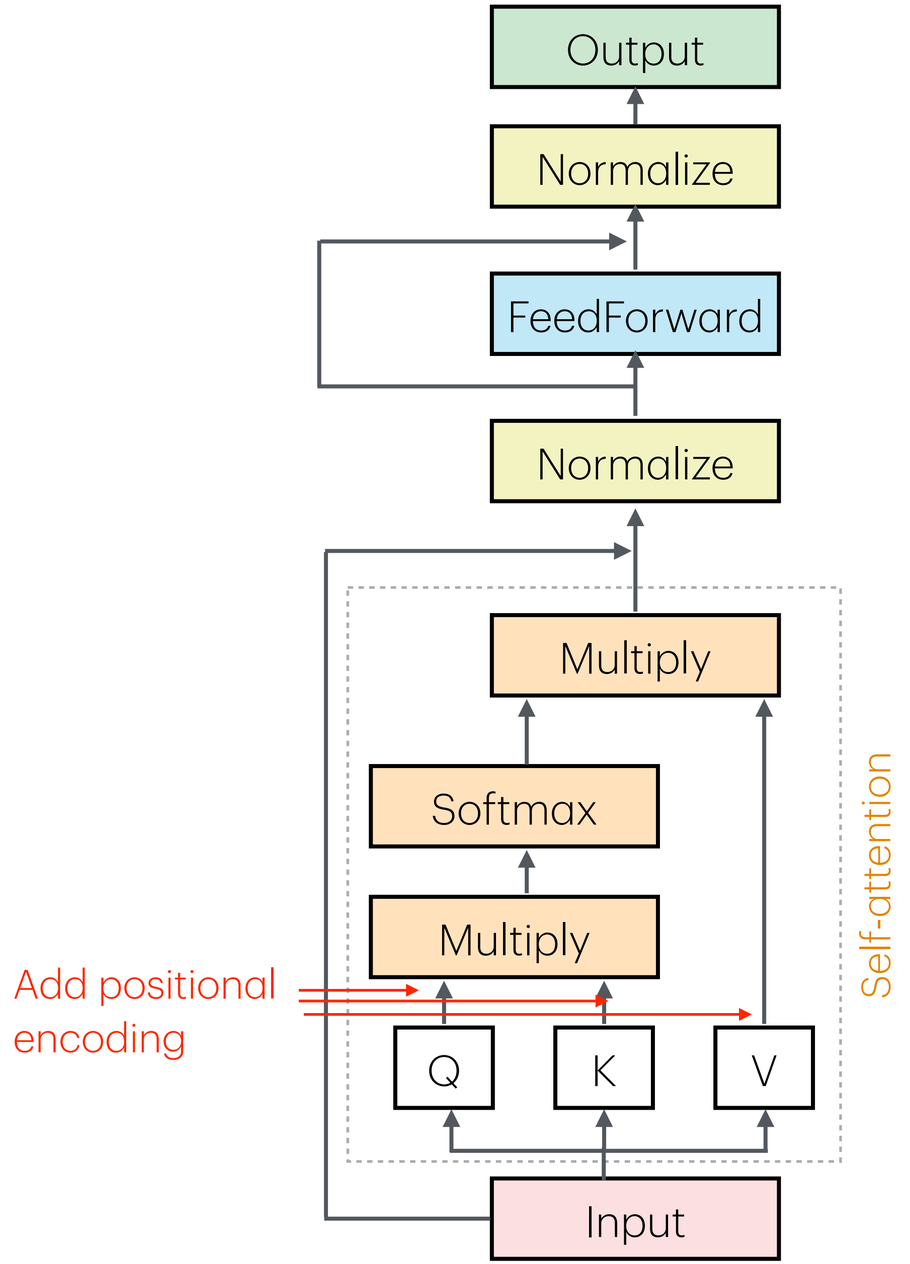
Issue: no explicit ordering information, permutation invariant. So we need to encode order to the inputs.

Add a vector (positional encoding) that encodes positional information such as follows, $p_i \in \mathbb{R}^d$, for $i \in \{1,2,..,N\}$:

$$p_i = \begin{bmatrix} \sin(i/1000^{2*1/d}) \\ \cos(i/1000^{2*1/d}) \\ \cdots \\ \sin(i/1000^{2*\frac{d}{2}/d}) \\ \cos(i/1000^{2*\frac{d}{2}/d}) \end{bmatrix}$$

Let $\tilde{q}_i, \tilde{k}_i, \tilde{v}_i$ be previous values, then we can show update query, key, values with the positional encoding as follows:

$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$
$$v_i = \tilde{v}_i + p_i$$



Ref: [2, 5]

Summary: Key technical concepts

1. Transformer model/architecture: a deep neural network, it is a component of most of the present-day language models, including LLMs.

2. Tokens: a unit used in NLP, a sub-word level tokenization is widely used in language models

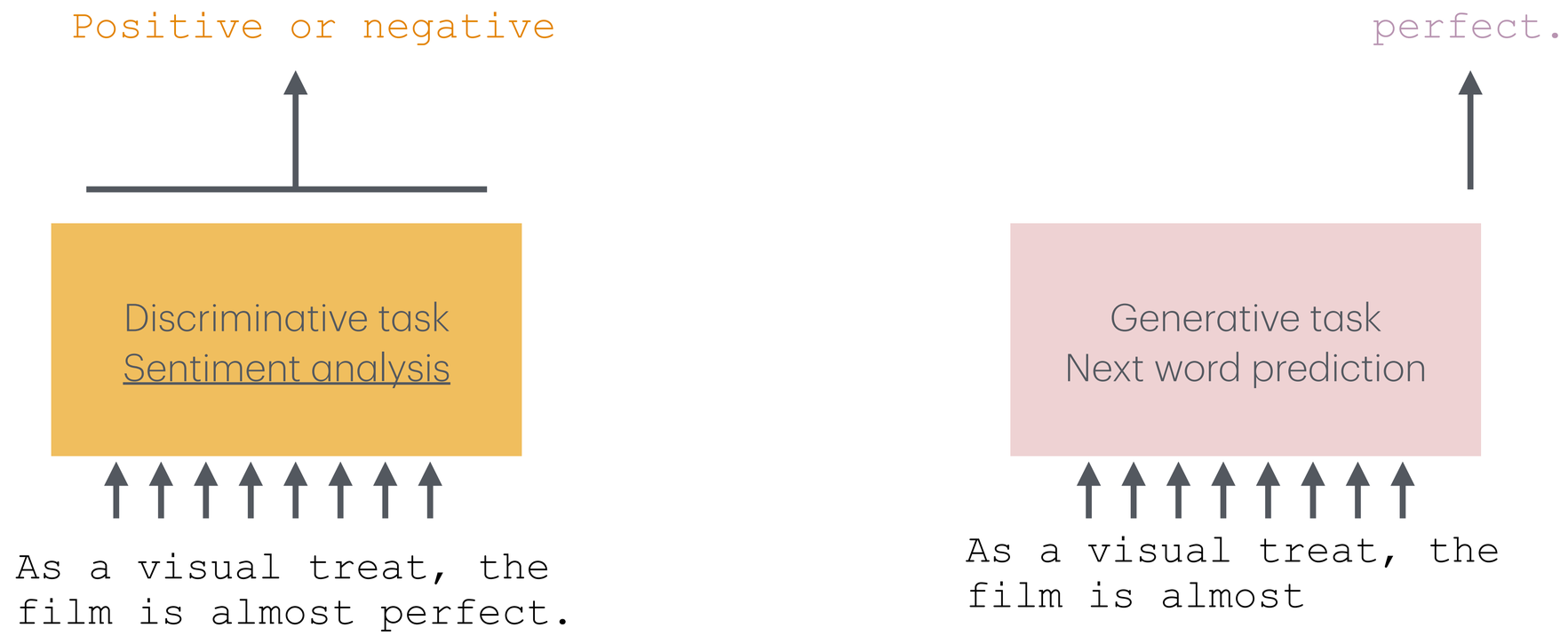
3. Self-attention layer: inputs are projected to query, key, value, and query attends to key to model the similarities of all other inputs.

4. Feed forward network: fully connected layer with ReLU that adds non-linearities to the output of the self-attention layer

5. Positional encoding: additional information added to input sequence that encode the order of the sequence

Discriminative vs Generative tasks in NLP

- Discriminative tasks: classifying input, e.g. sentiment analysis, text classification
- Generative tasks: generate text, language model, summarization, machine translation



Note that in generation of LM, there are two types of generations, i) prompt phase that takes input at once and process together, ii) text generation phase process one token at a time as autoregressive way

Adapted from : [1]

Generative AI/LLM

Generative AI

- Model that learned the probability distribution from the input data and generates new output (text, images, and other modalities) based using the distribution (learned patterns).
- Generative AI: large language model, GAN, diffusion model, VAE

What is a Large Language Model (LLM)?

- An LLM is a model takes text input and generates a high probability continuation of that text output
- The probabilities are trained form massive quantities of text data
- The probabilities are learned from highly abstract matrix operations (self-attention):
  - No symbolic computer code
  - No interpretable database storage
  - No analytical control over behaviors
- It is a prediction mechanism for next text, which is a language model (LM) shown in previous slides.

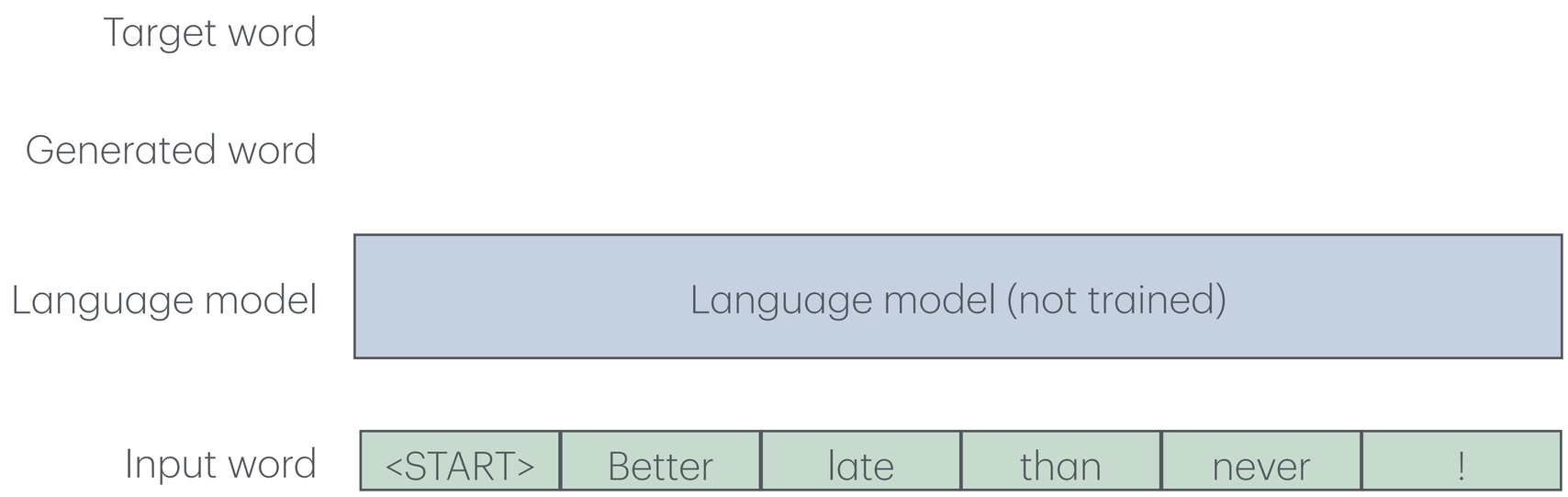
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's *self-supervised* training because there are no explicit labels, but input text is used as both input and target label.



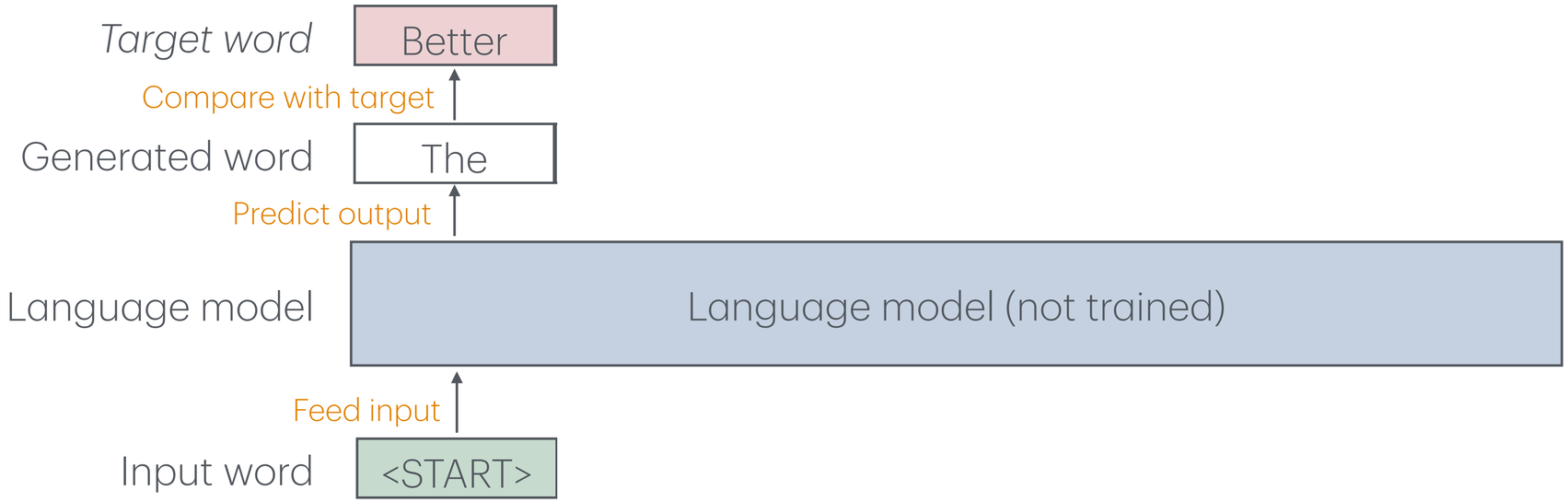
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



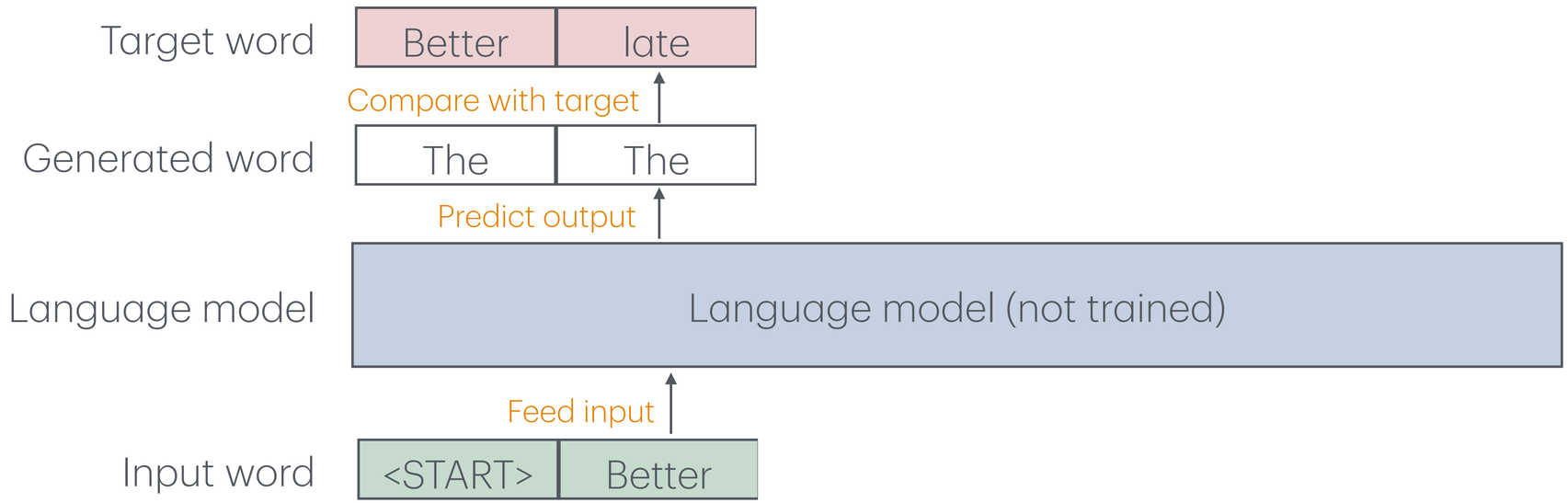
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



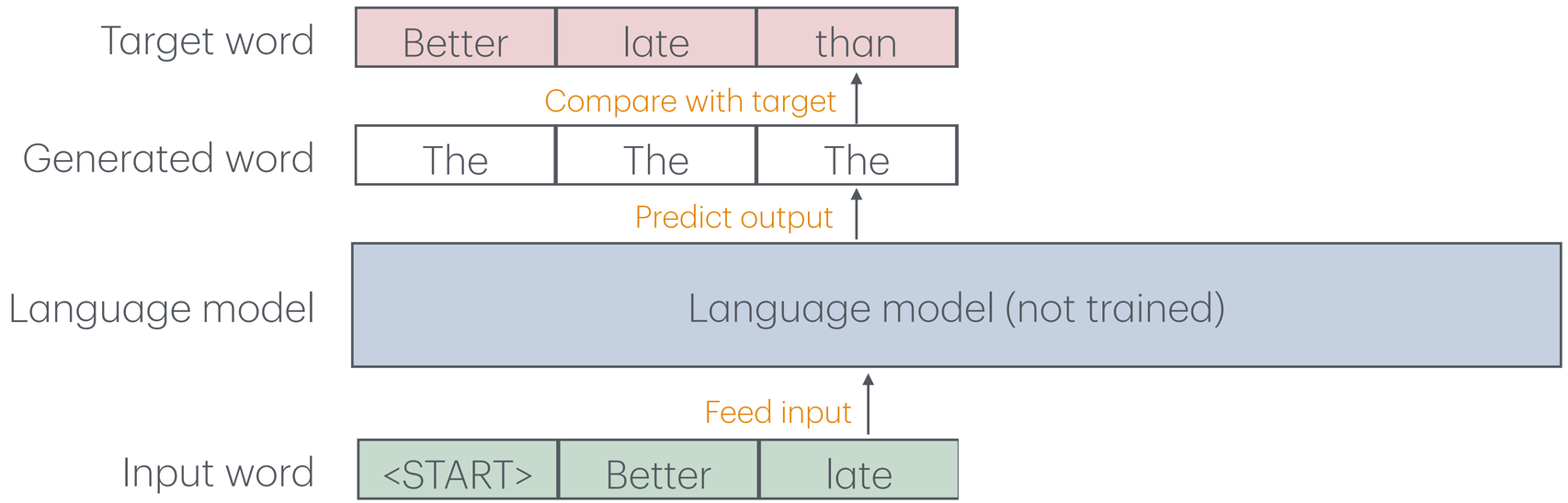
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



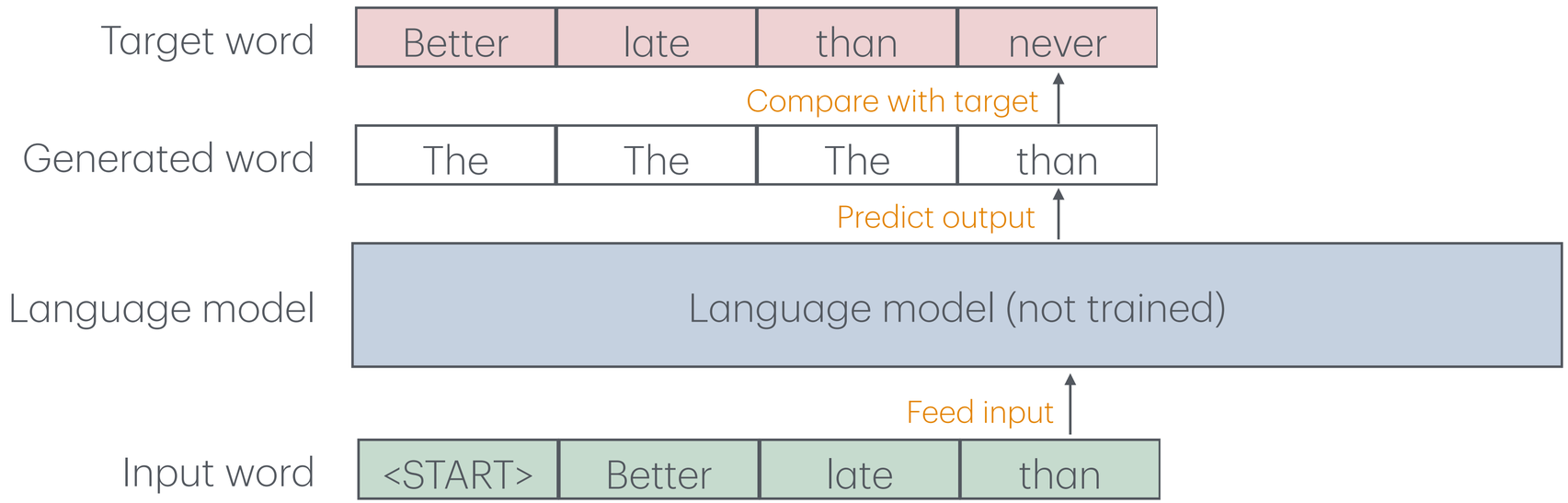
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

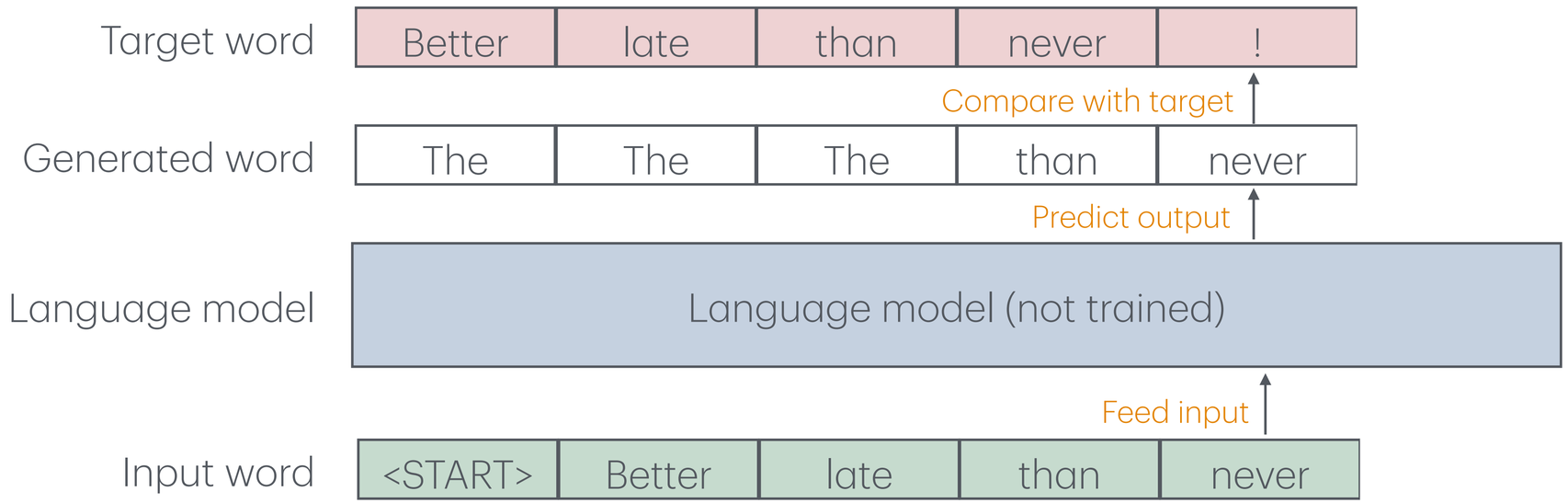


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

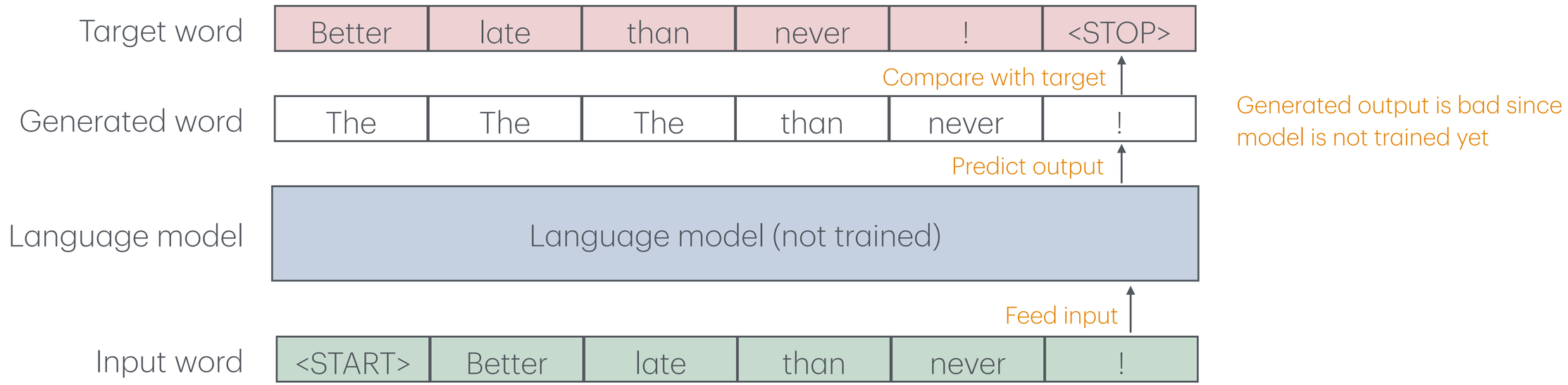


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



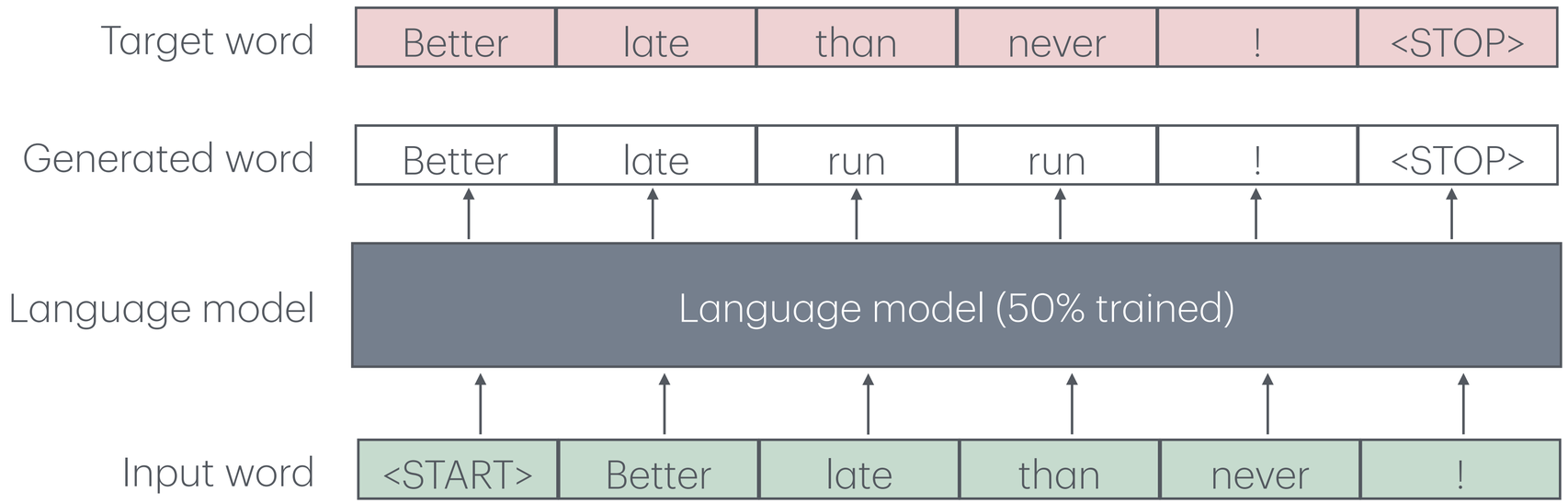
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.



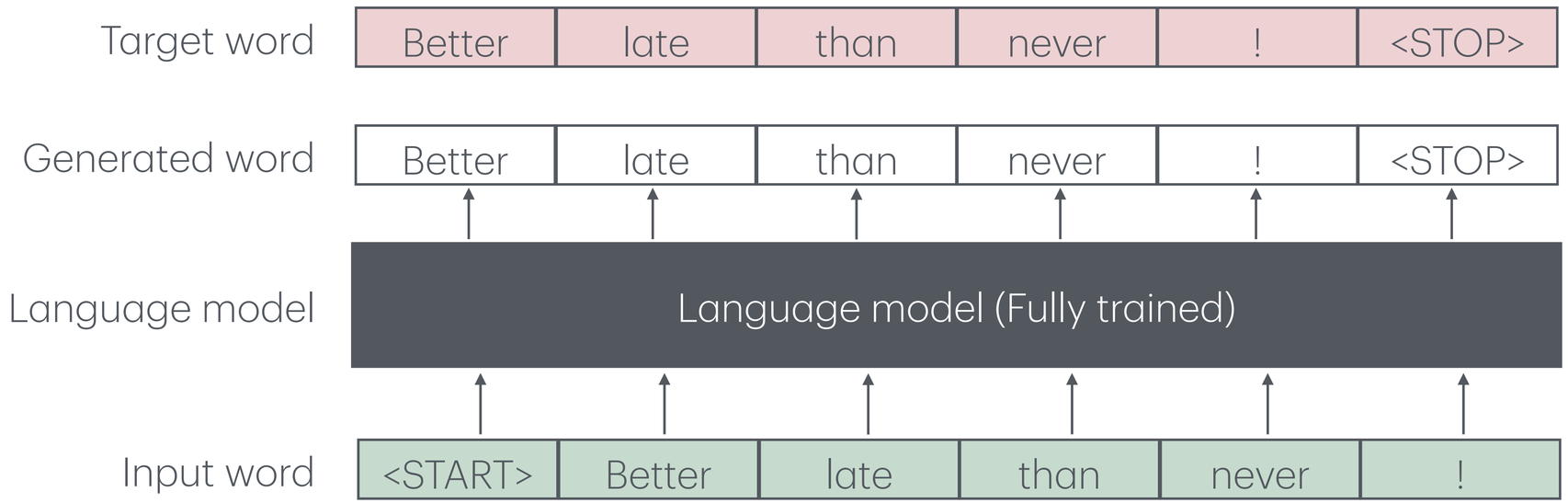
Ref: [4]

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

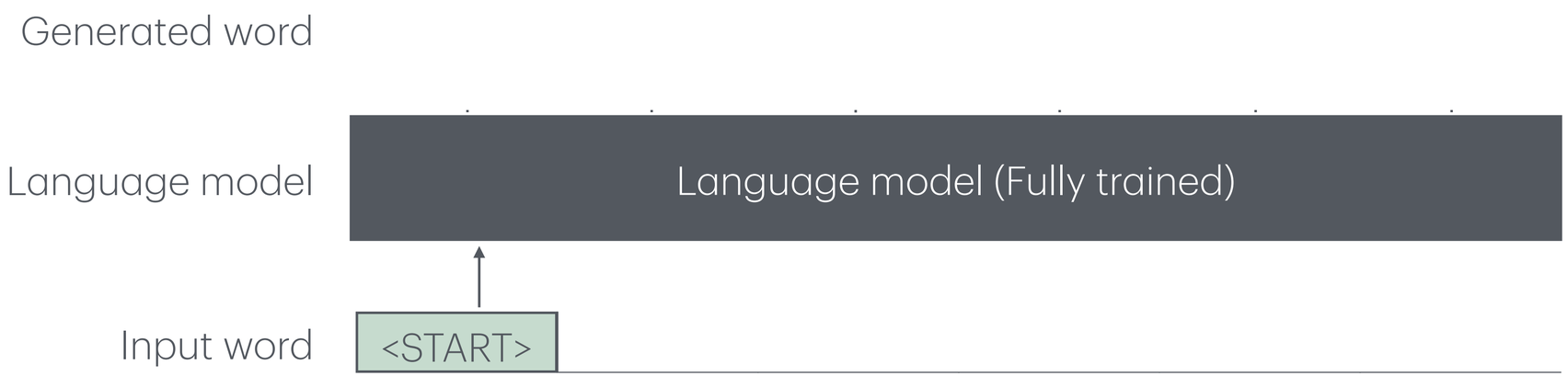


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed as the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

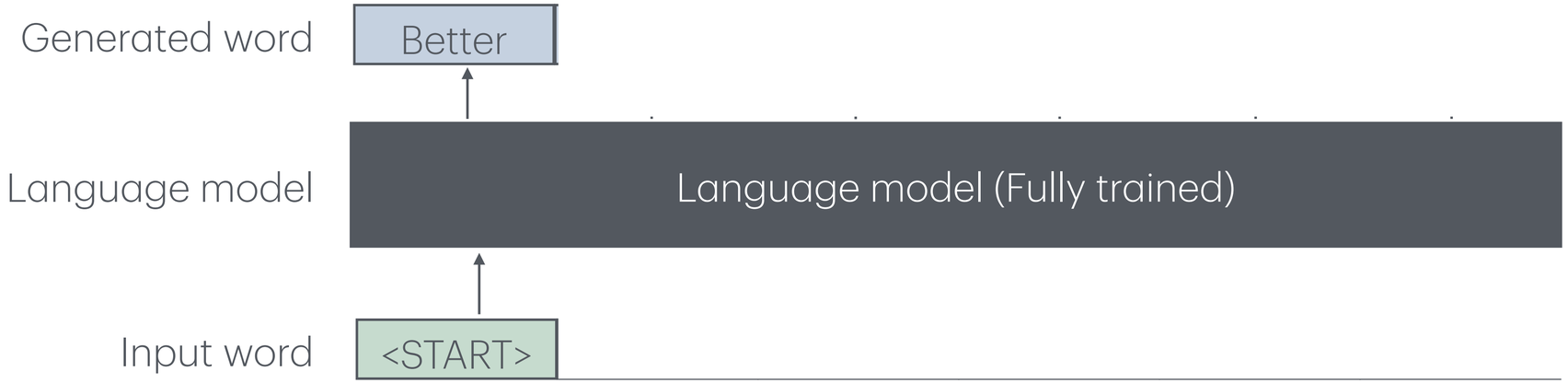


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

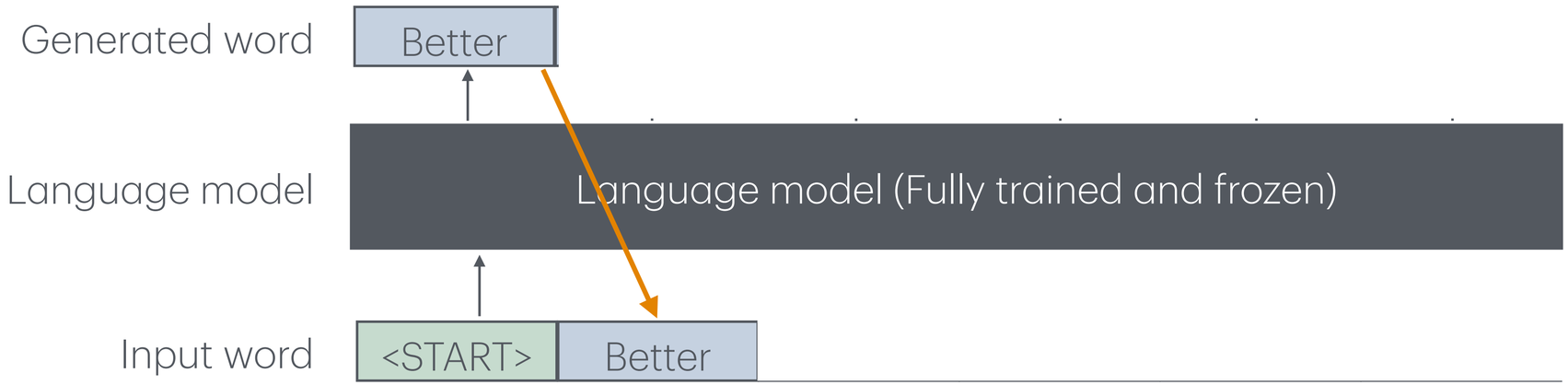


Ref: [4]

LM generation with greedy decoding

In LM *generation (inference)*, previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

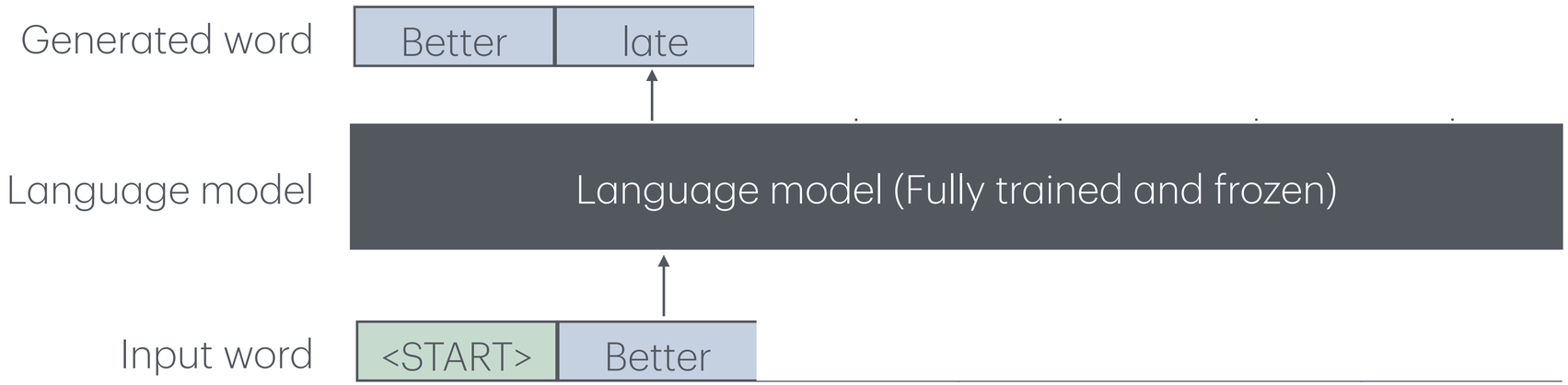


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

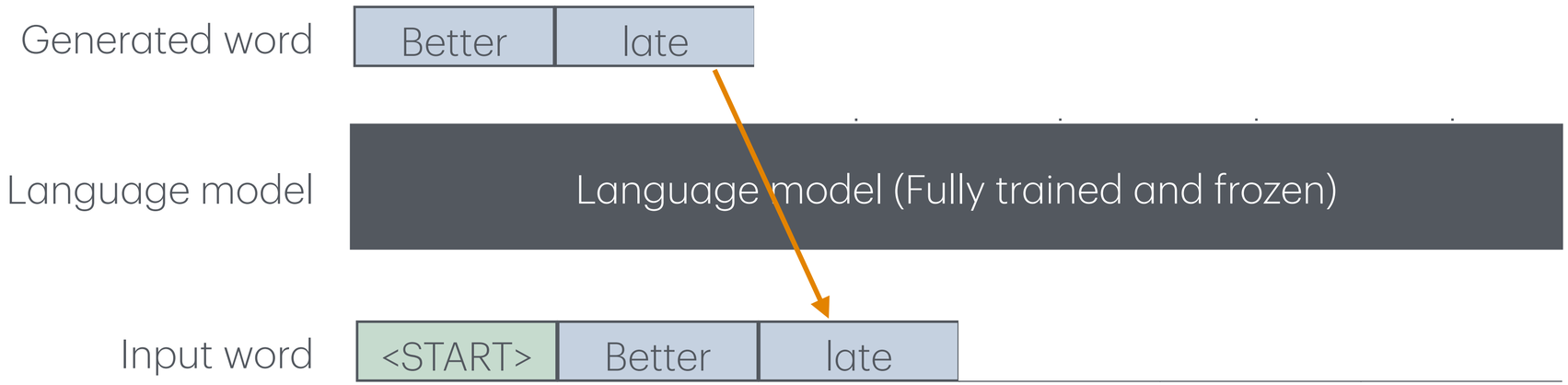


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

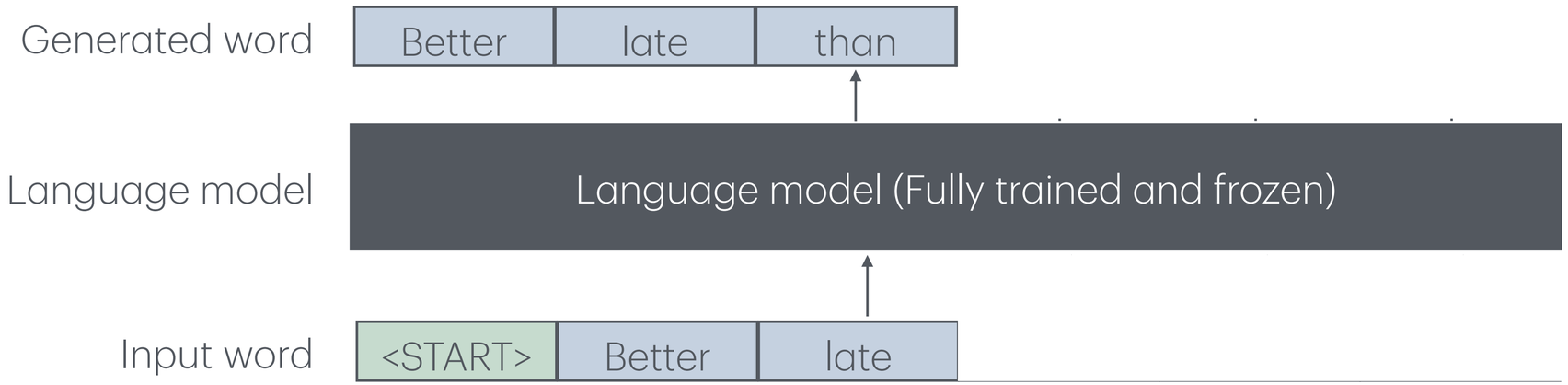


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

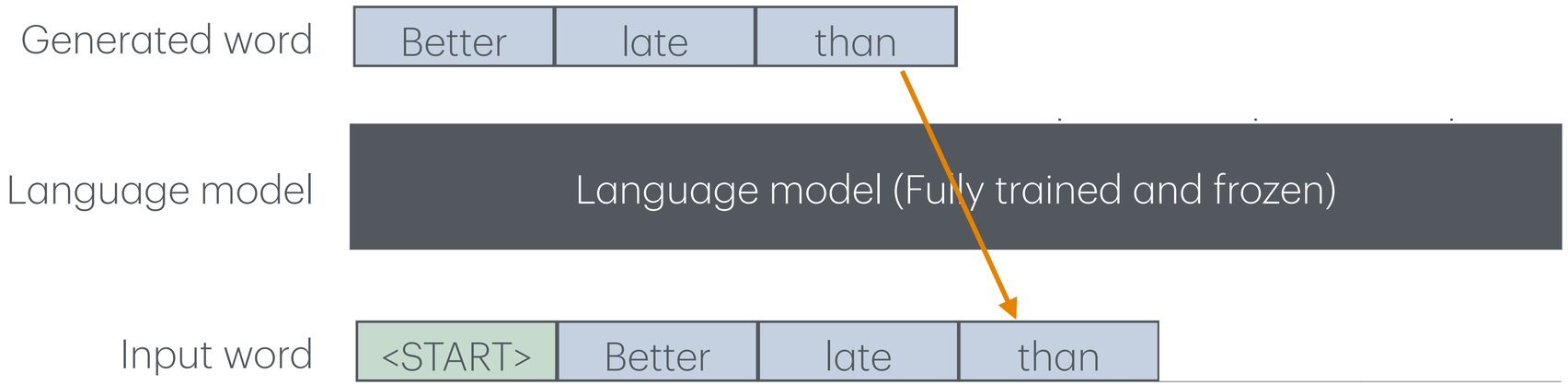


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

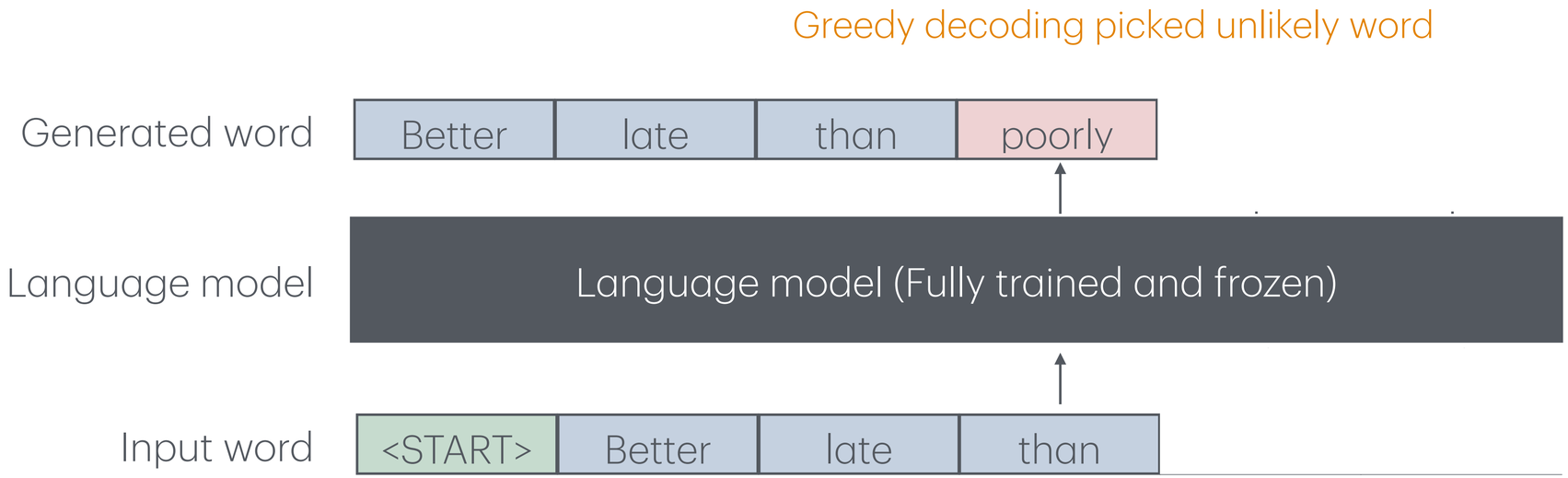


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

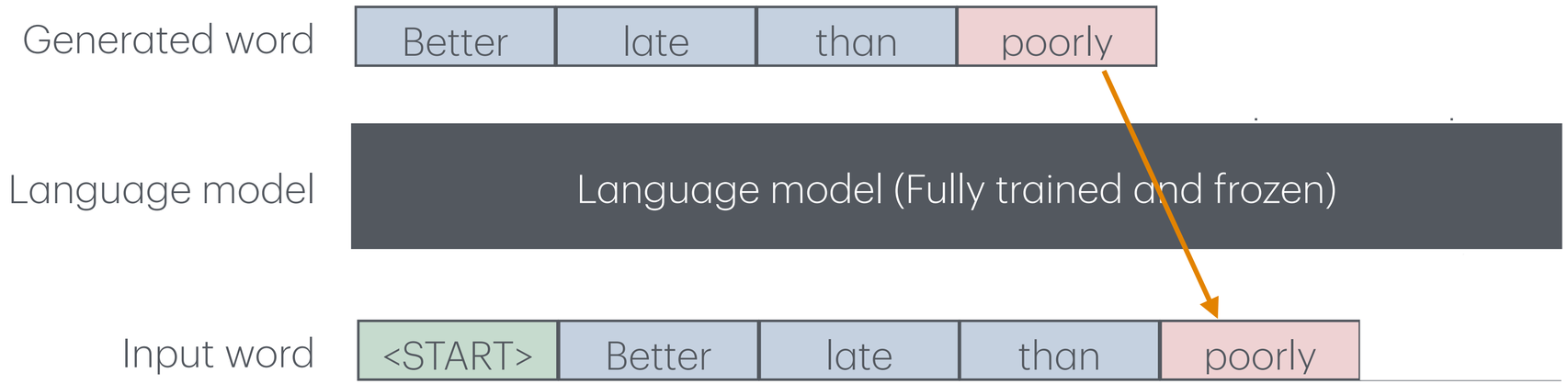


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

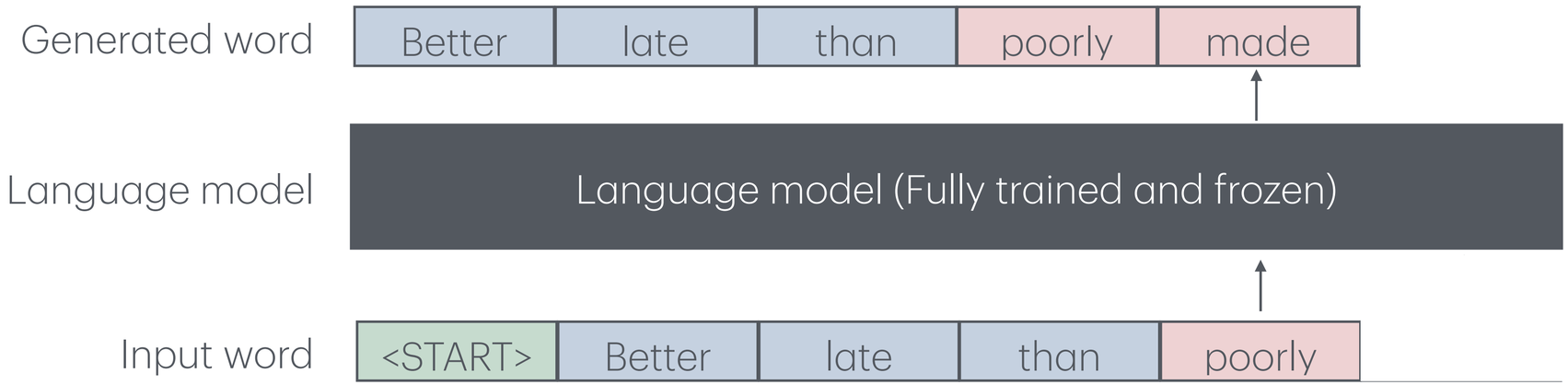


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

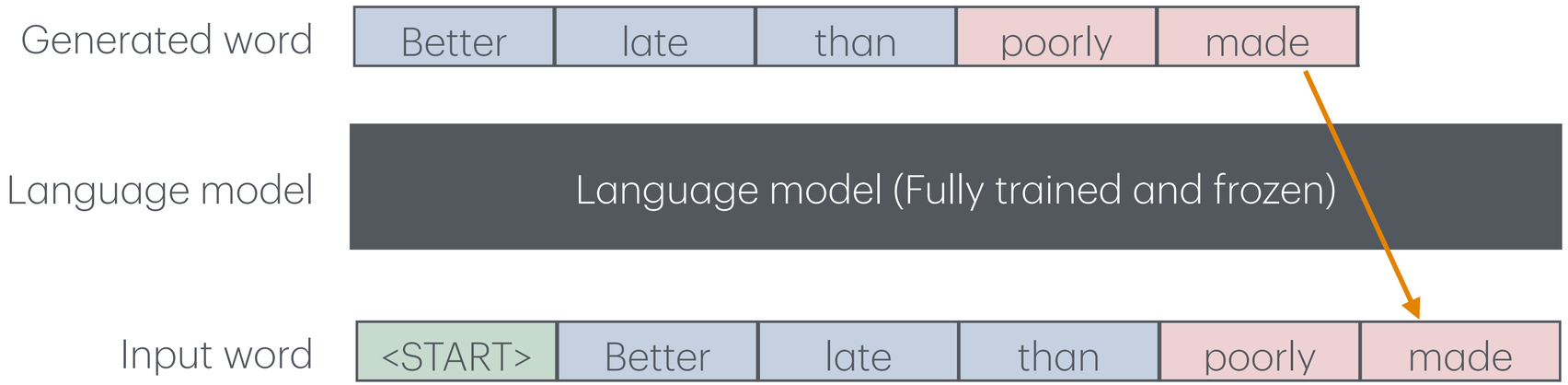


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

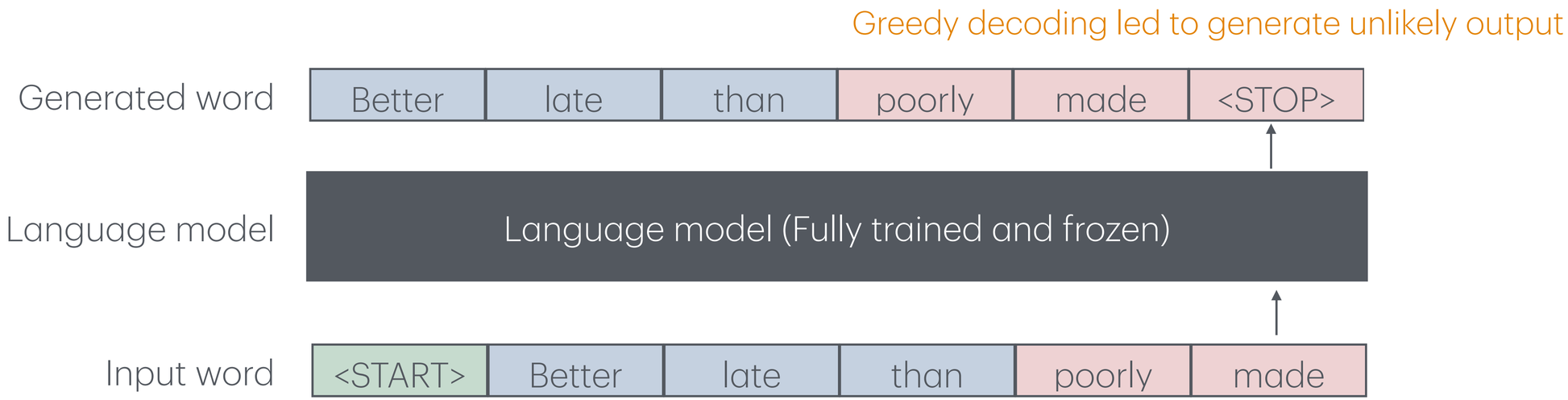


Ref: [4]

LM generation with greedy decoding

In LM generation (inference), previously generated word is fed to the input. Then continuously generate output until it generates <STOP> word.

Generation (LLM inference) is based on greedy decoding, and if one generation word is mis-generated, then the following words will be impacted.

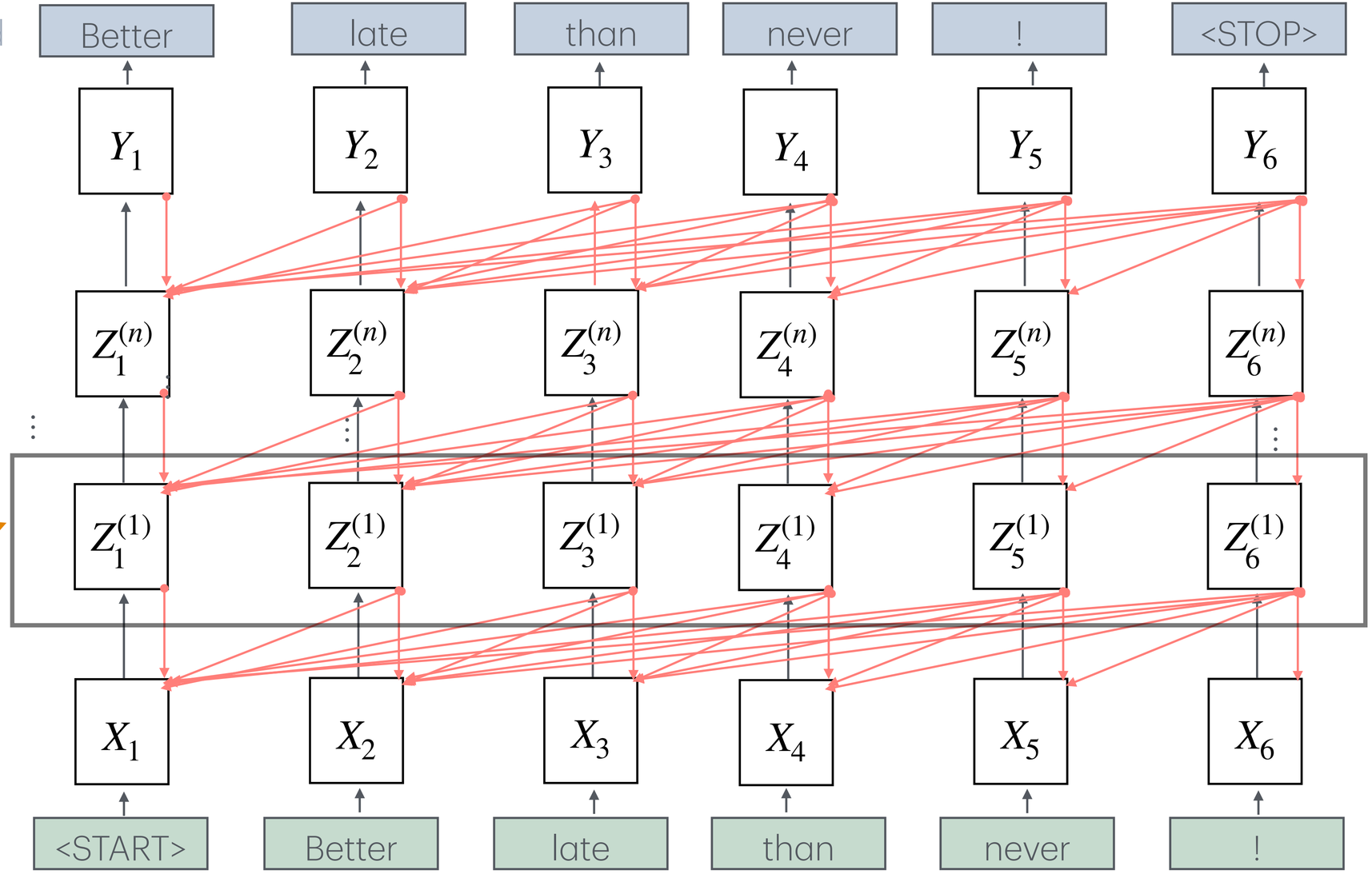


Ref: [4]

Transformer based Language model

Language model is built with transformer blocks

$Z_t^{(i+1)} = \text{Transformer\_block}(Z_t^{(i)})$



Each layer is transformer block with attention connections to preceding states

Image source: [2]

GPT-3: Large language model

- GPT-3 is LLM that uses transformer architecture to generate text based on various tasks prompts.
- # params: 175B
- Training data: 200B tokens of text from books, the internet, and other sources.

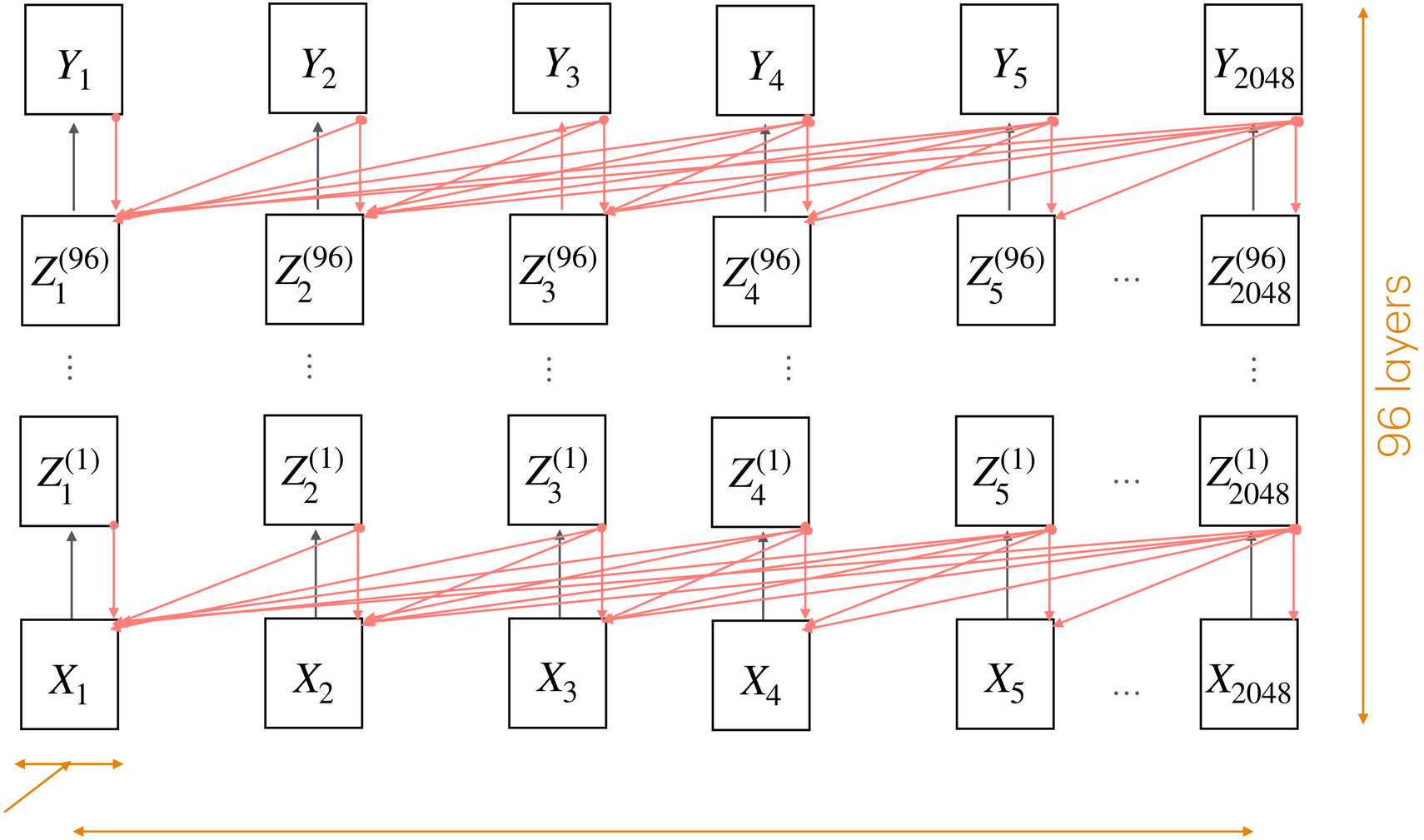


Image source: [2]

How LLM is trained:

1. Pre-training: LLM is trained with large corpus by next token prediction objective (previous slides)
2. Post-training:
    1. Instruction following training adapts pre-trained LLM to follow specific instruction and commands. It is also called as SFT (supervised fine-tuning).
        - It makes the models easier to use.
        - It makes the model to respond in a specific style
    2. RLHF (reinforcement learning with human feedback): method that fine-tunes the model using human preference to align generated behaviors with human values and intentions

A typical instruction template

```
Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.


### Instruction:

{instruction}


### Input:

{input}


### Response:
```

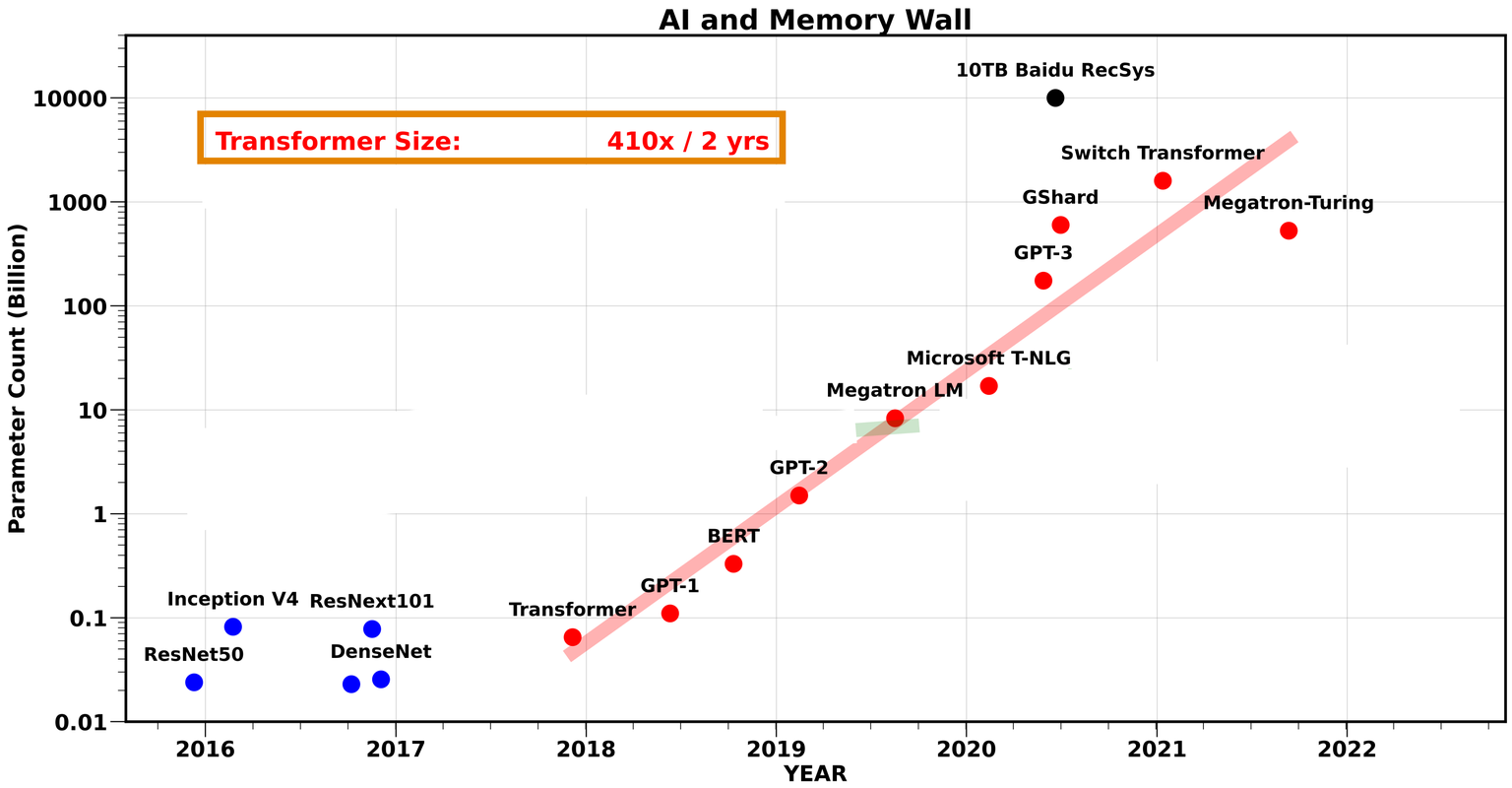
A typical prompt to do tasks using LLM.

Note that there are many ways to prompt LLM, which will be covered in module 2.1

Ref: [4]

Parting thoughts, current LLM trend

Use transformer blocks and growing size



The increasing trend of model size growth, 410x/2 yrs.

Image source: [4]

Summary

1. Self-supervision: language model uses the input text to predict the next word without label. The LM learns to imitate its training examples.

2. Generation (LM inference): the model continues its input (prompt) sequence.

3. (Most of present-day) Language models: consist of Transformer layers

4. Pre-training: teaches the model how the world works in general by using self-supervision training

5. Fine-tuning: customizes it to a specific task to easier to use. There are various ways to fine-tune pre-trained model such as instruction-following fine-tuning and RLHF fine-tuning.

Ref: [4]

References

1. TinyML and Efficient Deep Learning Computing
2. Deep Learning Systems, Algorithms and Implementation
3. Deep Learning for Computer Vision
4. Generative AI: Technology, Business, and Society
5. Natural Language Processing with Deep Learning

Free Books and additional materials

1. Dive into Deep Learning, Interactive deep learning book with code, math, and discussions
2. Understanding Deep Learning, book with Jupyter notebooks
3. Deep Learning, in-depth details theory and math
4. Word2vec visualization: https://projector.tensorflow.org/
5. Good transformer tutorial: https://www.tensorflow.org/text/tutorials/transformer
6. The Illustrated GPT-2 (Visualizing Transformer Language Models)
7. TRANSFORMERS FROM SCRATCH

Additional slides

Sequential problems

There are many applications requires sequential data processing

Sequential data: data at time t has relationship with previous time steps

Various types of input and outputs used in sequential applications

A regular neural network might not use the sequential nature of the input data

A neural network might not generate sequential outputs

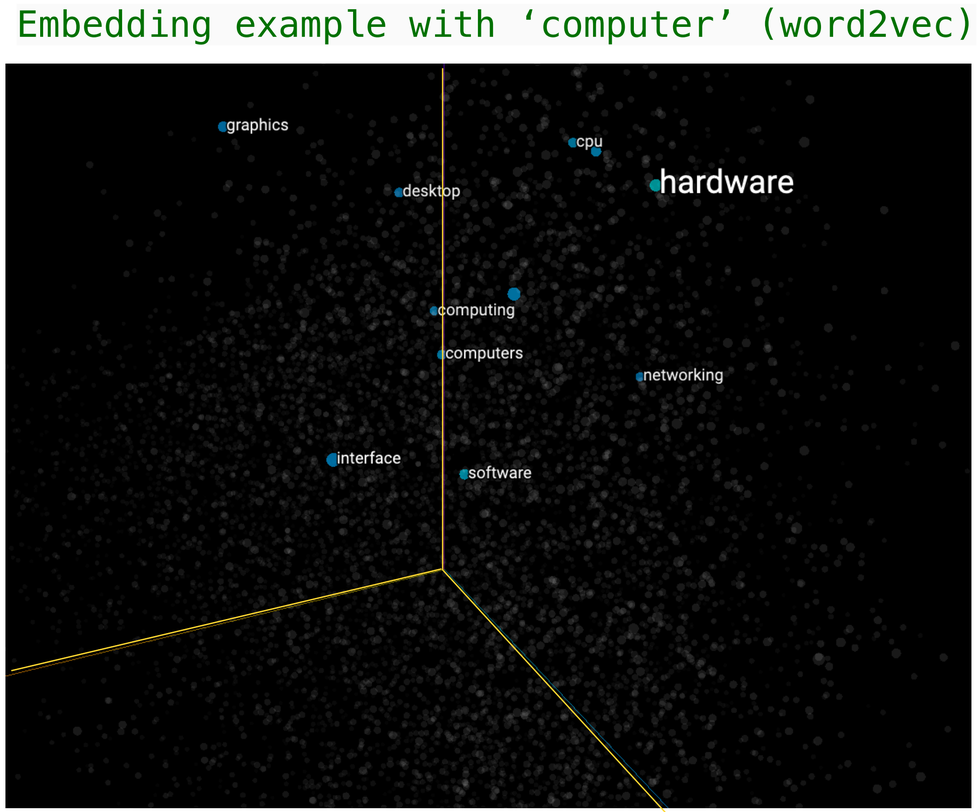
Types	Input	Output
Time series forecasting	Time series 	Next time step prediction
Sentiment analysis	movie review Rotten Tomatoes	Predicted sentiment 
Language translation	English text	日本語 普通话 한국어
Speech recognition		Transcribed text
Document summarization	Document text	Summarized text
Image captioning		Describing scene, planes flying
Question answering	what is the tallest building in the world?	Shanghai Tower

Photo by Anton on Unsplash Photo by Andrey Kremkov on Unsplash

Adapted from : "RNNs" section of Full Stack Deep Learning

Word2vec visualization

- Visualization: https://projector.tensorflow.org



Sequential problems

Problem setup of sequential problems

One to many:

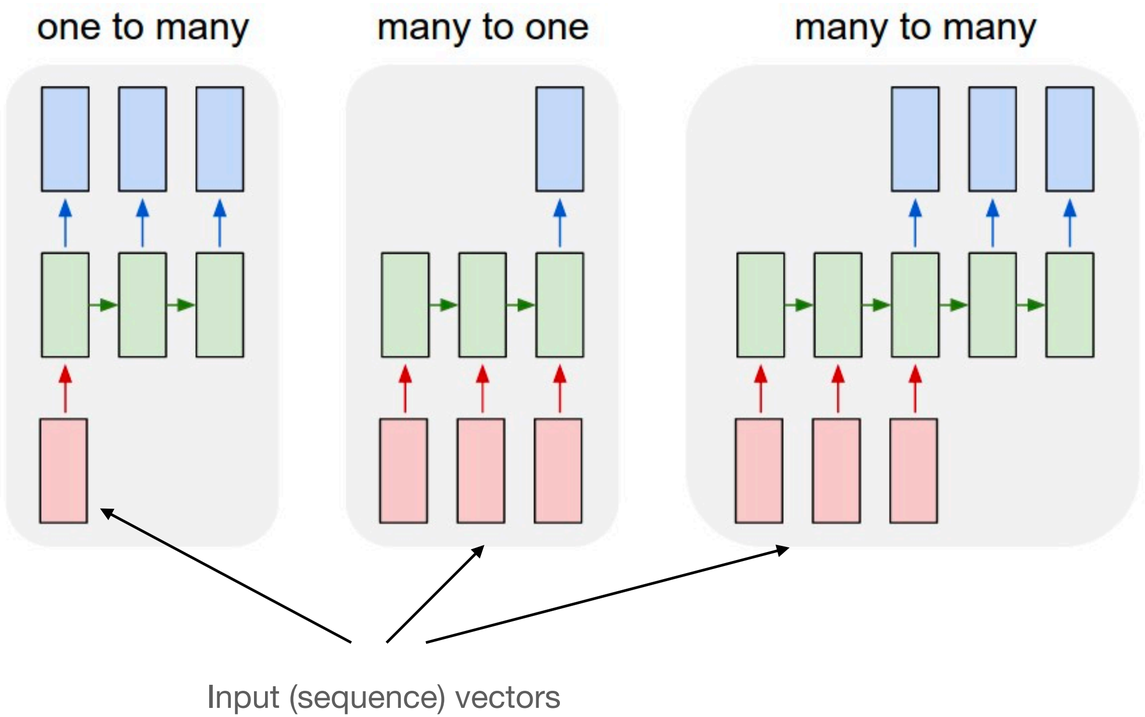
- Image captioning

Many to one:

- Sentiment analysis, time series forecast

Many to many:

- Machine translation



Adapted from : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN in code

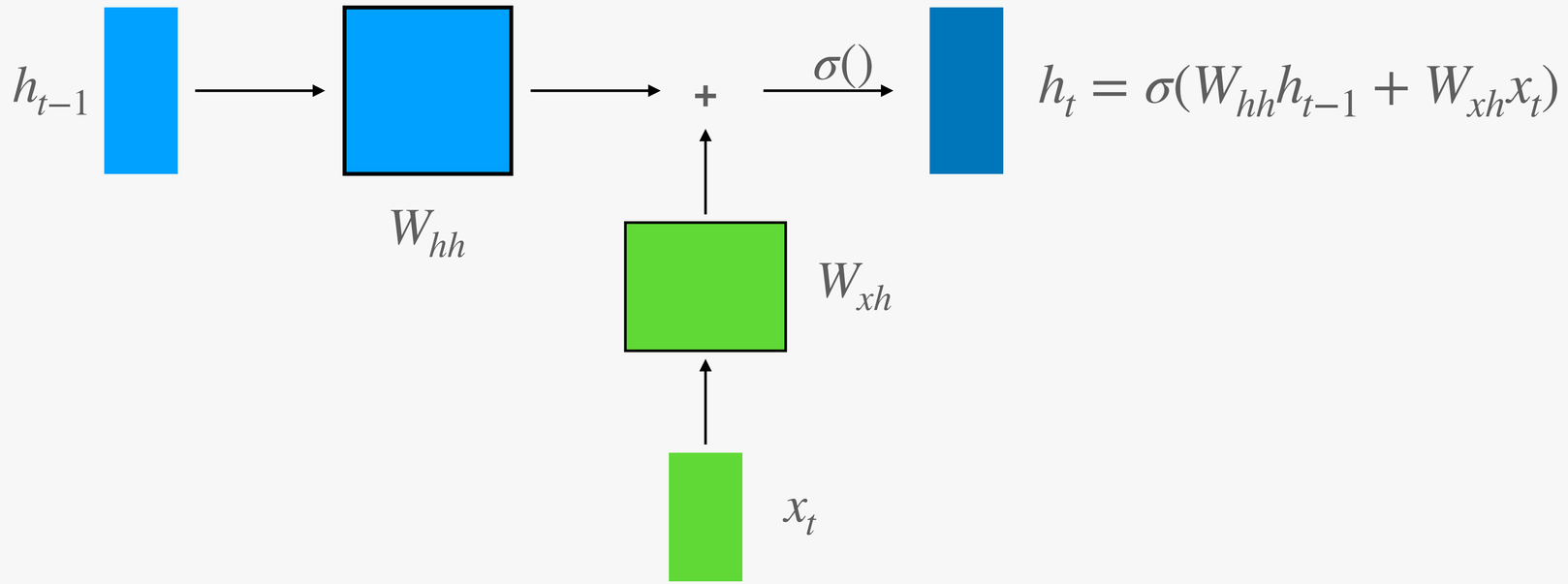
```
class RNN:
  # ...
  def compute_next_h(self, x):
    # Vanilar RNN hidden computation
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    return h
  # ...
  def step(self, x):
    # update the hidden state
    self.h = self.compute_next_h(x)
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t)$$

Adapted from : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN in code 2

```
class RNN:
```

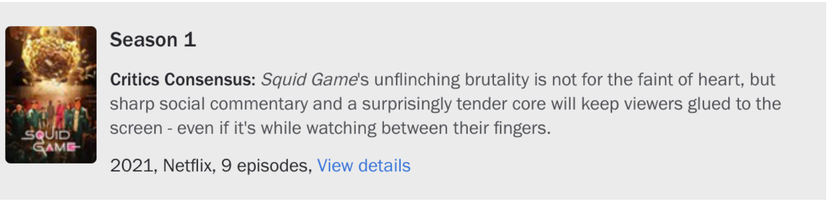


Adapted from : https://drive.google.com/file/d/1bn801i0Brs2FypxDyjBIzJyrNYQof80J/view

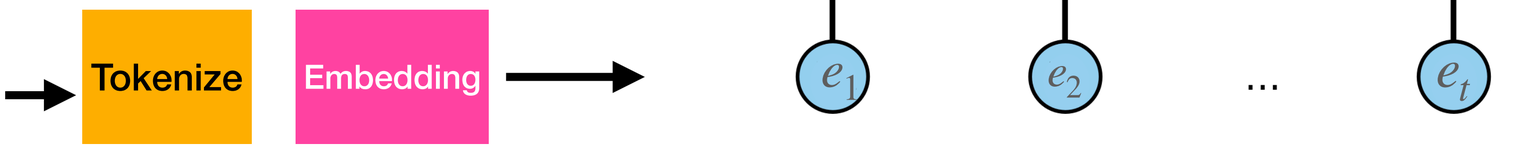
RNN applications

Many to one setup

- Add the fully connected layer at the last hidden state
- Use it as a classification
- RNN is working as encoder, encodes input text
- FC is working as a decoder that use the encoded information from RNN to use it for classification



Input text



Adapted from : https://drive.google.com/file/d/1bn801i0Brs2FypxDyjBIzJyrNYQof80J/view

RNN applications

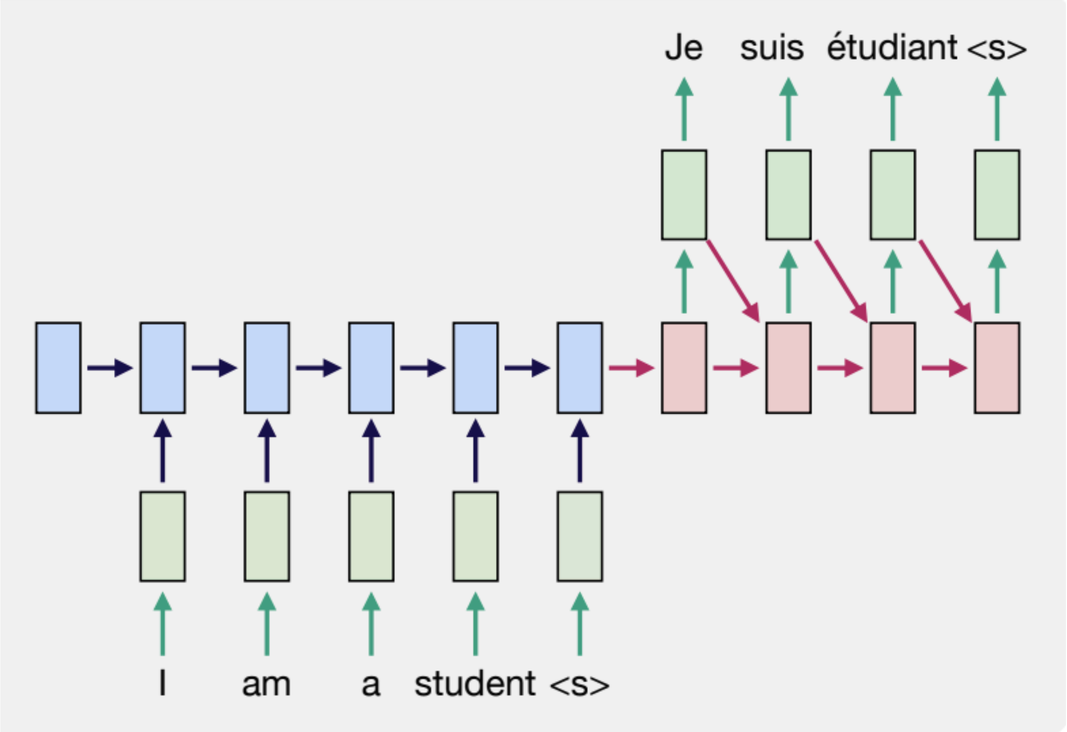
Language translation

Many to many setup consists of encoder RNN and decoder RNN

Encoder decoder architecture

- Also called sequence to sequence (seq2seq)

Dramatically replace all statistical machine translation (SMT) method to RNN based neural machine translation (NMT) in 2014



Note: Another changes in NLP: attention based improvement and transformer type model have been dominating in NLP since 2019

Source: https://drive.google.com/file/d/1bn801i0Brs2FypxDyjBIzJyrNYQof80J/view

Fancy RNNs

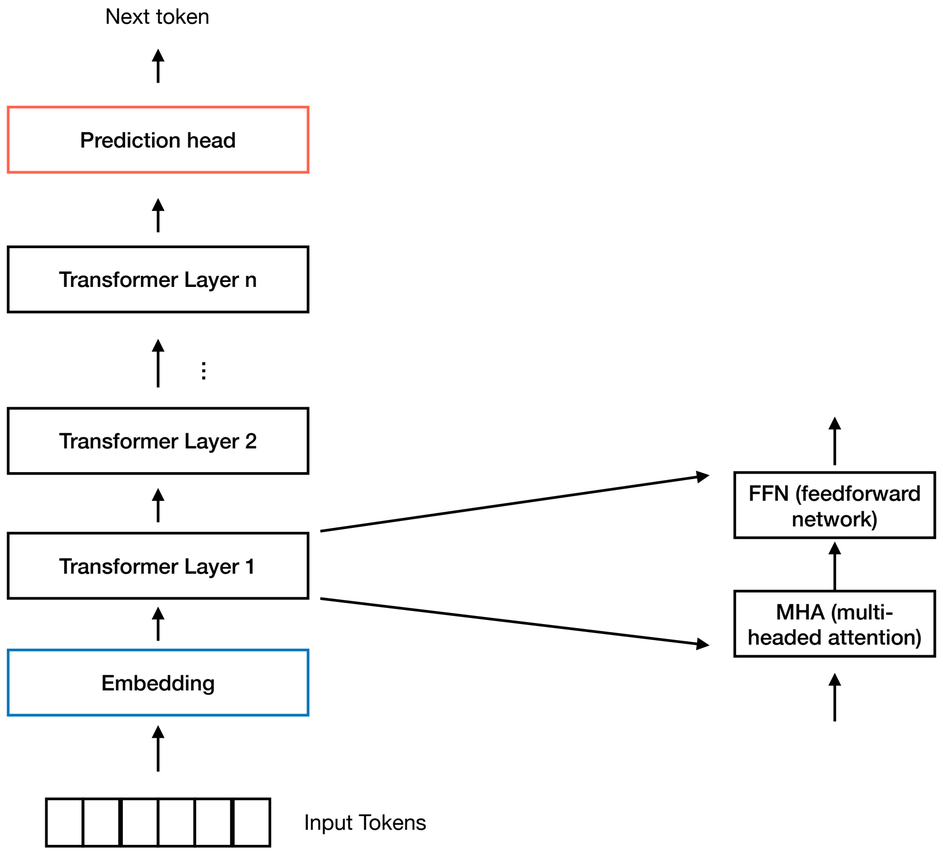
Vanilla RNN has difficulty of capturing long term relations due to vanishing gradient

- Fancy RNN that allows to capture past state better, LSTM (long short term memory), GRU (Gated recurrent unit)

Stacked RNN: stacking RNNs to capture more complex information

Bi-directional RNN: to capture in both forward and backward directional relations

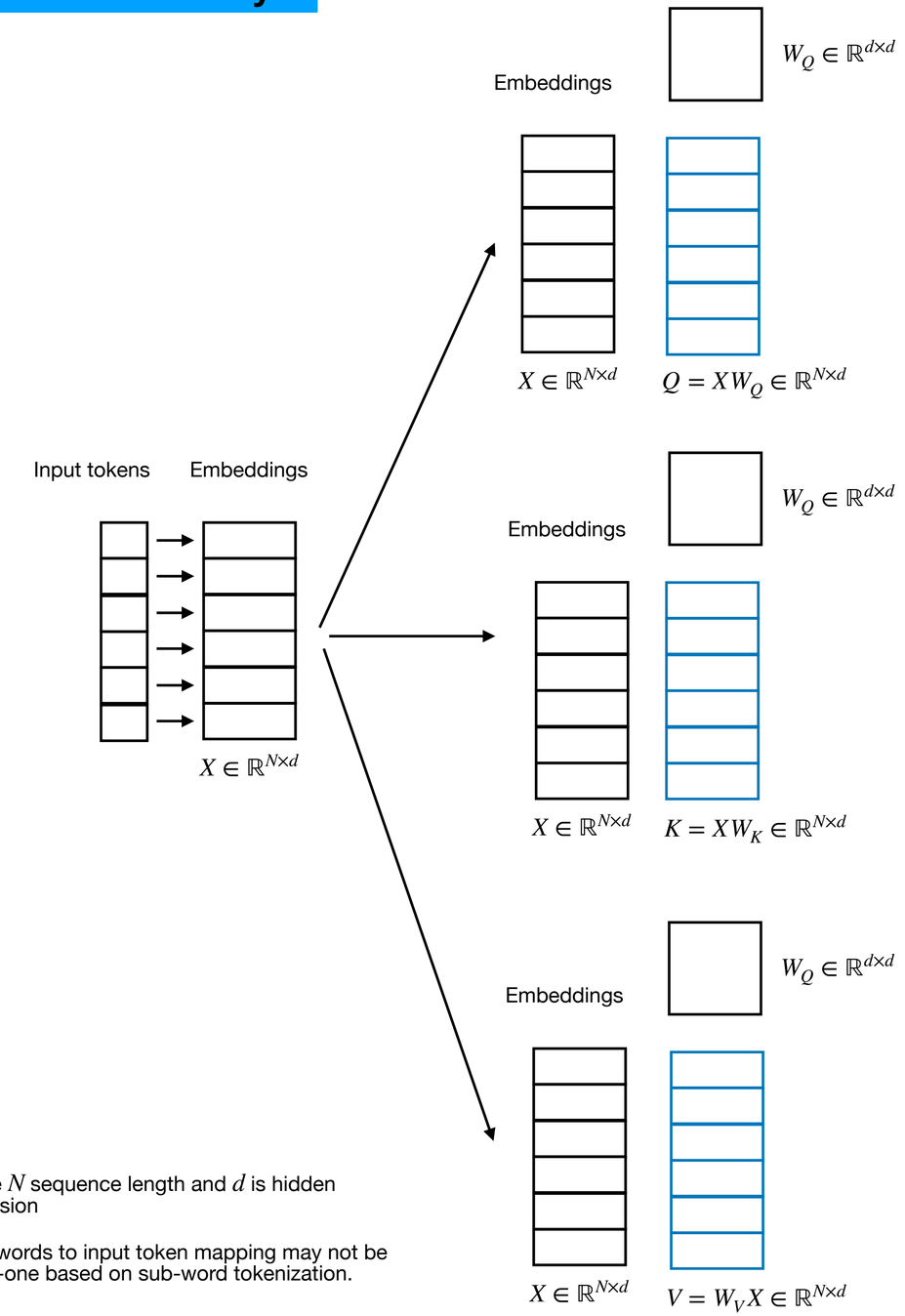
Overview of GPT (decoder) style LLM structure



* Tokenization from Input words to input tokens is omitted for simplicity

** We omit layer norm and residual addition for simplicity

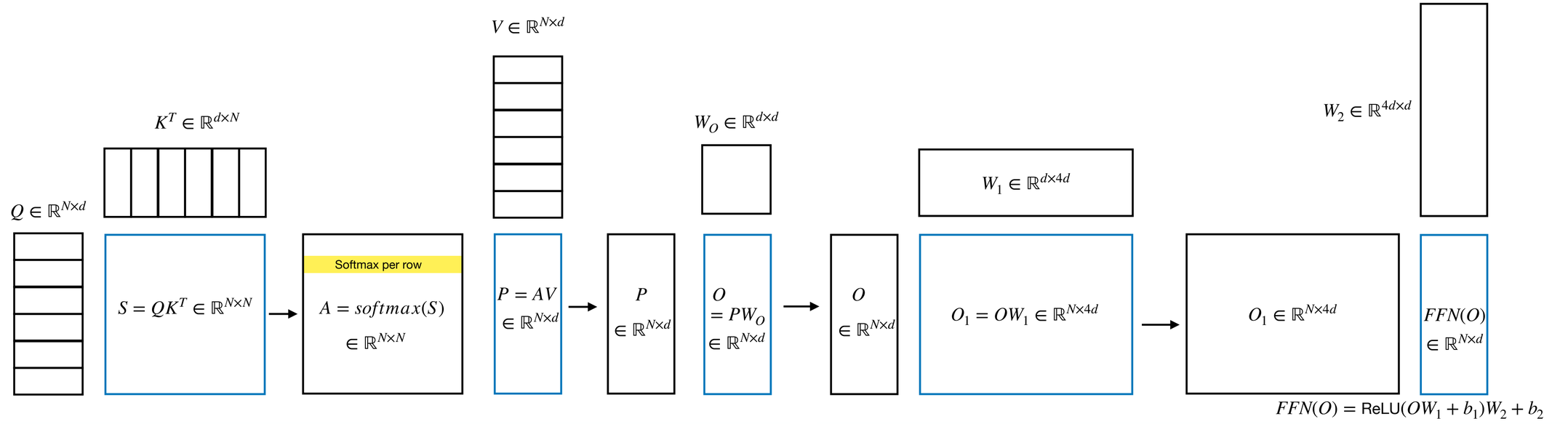
Input tokens to Q, K, V in a single Transformer layer



Where N sequence length and d is hidden dimension

*Input words to input token mapping may not be one-to-one based on sub-word tokenization.

Training: MHA (multi headed attention) and FFN (Feedforward network) in a single transformer layer



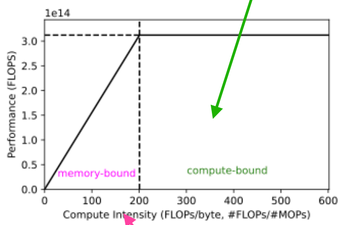
Forward pass compute	$O(N^2d)$	$O(N^2)$	$O(N^2d)$	$O(Nd^2)$	$O(Nd^2)$	$O(Nd^2)$

Where N sequence length and d is hidden dimension

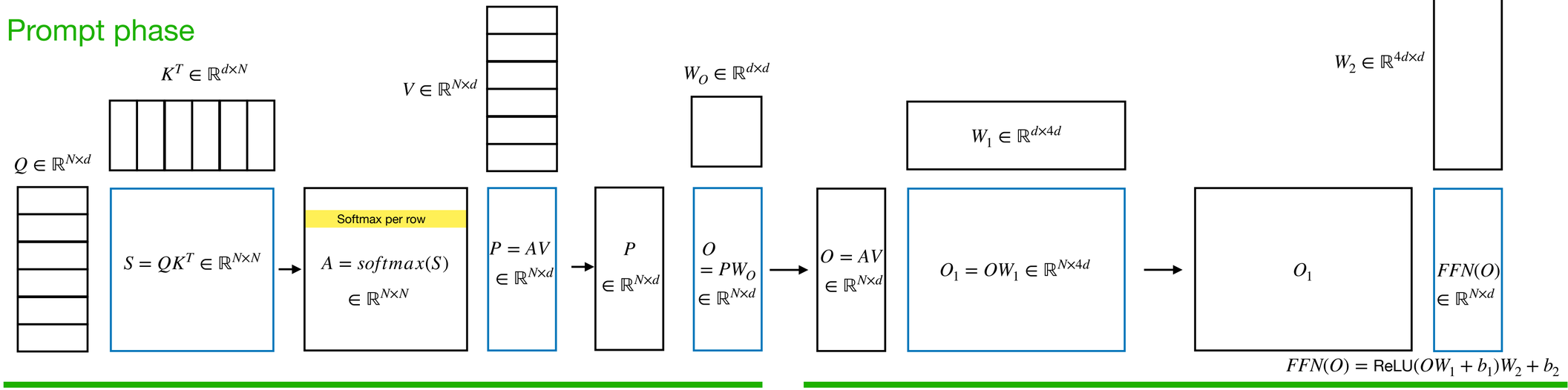
* We omit layer norm, residual addition, and causal masking for simplicity

** We use b, batch size of 1.

Inference: Two phases of inference: MHA (multi headed attention) and FFN



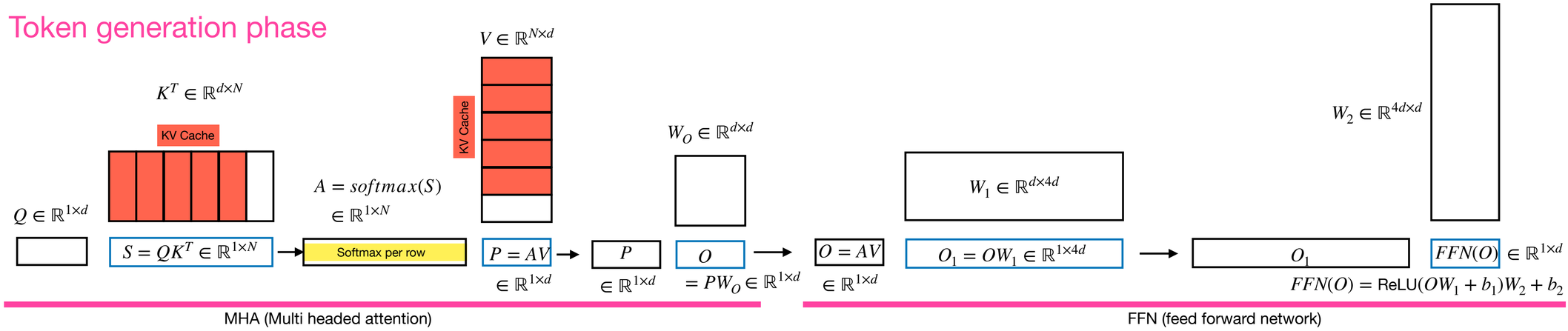
Prompt phase



MHA (Multi headed attention)

FFN (feed forward network)

Token generation phase



MHA (Multi headed attention)

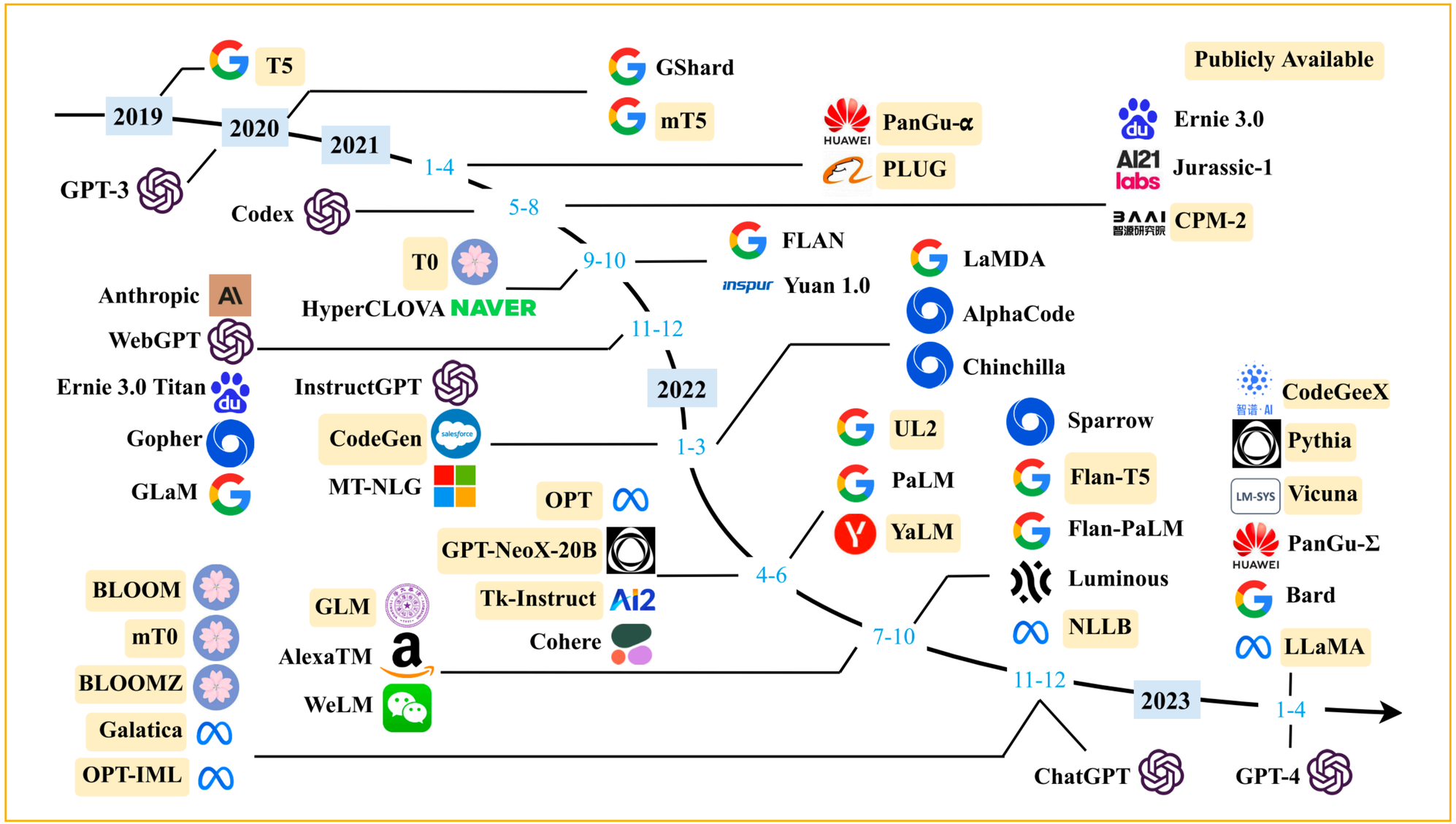
FFN (feed forward network)

Where N sequence length and d is hidden dimension

* We omit layer norm, residual addition, and causal masking for simplicity.

** This roofline model is based on A100 40G HBM

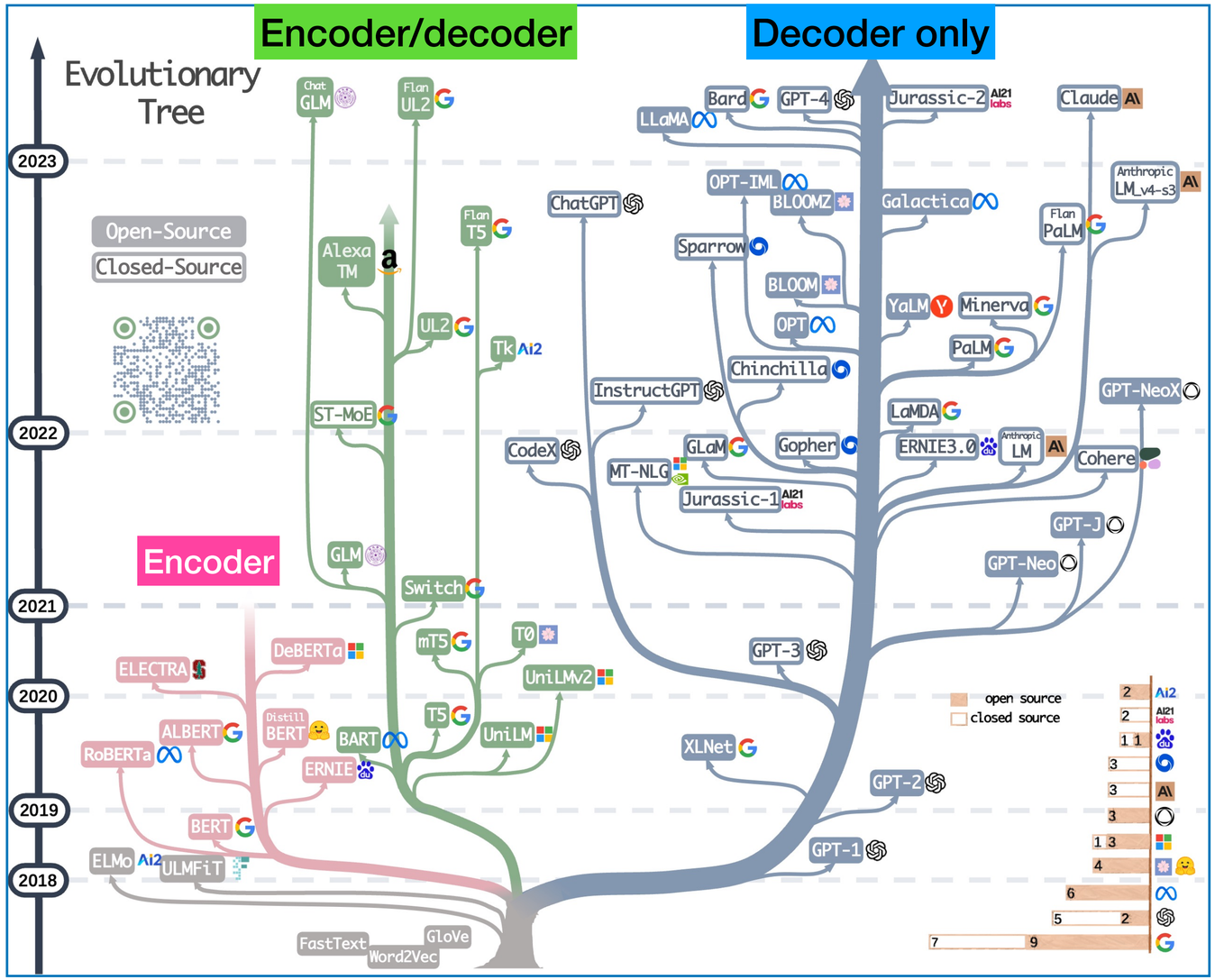
LLM landscape



Ref: A Survey of Large Language Models, https://arxiv.org/abs/2303.18223

LLM landscape

	Model	Release Time	Size (B)	Base Model	Adaptation IT	Adaptation RLHF	Pre-train Data Scale	Latest Data Timestamp	Hardware (GPUs / TPUs)	Training Time	Evaluation ICL	Evaluation CoT
Publicly Available	T5 [73]	Oct-2019	11	-	-	-	1T tokens	Apr-2019	1024 TPU v3	-	✓	-
	mT5 [74]	Oct-2020	13	-	-	-	1T tokens	-	-	-	✓	-
	PanGu-α [75]	Apr-2021	13*	-	-	-	1.1TB	-	2048 Ascend 910	-	✓	-
	CPM-2 [76]	Jun-2021	198	-	-	-	2.6TB	-	-	-	-	-
	T0 [28]	Oct-2021	11	T5	✓	-	-	-	512 TPU v3	27 h	✓	-
	CodeGen [77]	Mar-2022	16	-	-	-	577B tokens	-	-	-	✓	-
	GPT-NeoX-20B [78]	Apr-2022	20	-	-	-	825GB	-	96 40G A100	-	✓	-
	Tk-Instruct [79]	Apr-2022	11	T5	✓	-	-	-	256 TPU v3	4 h	✓	-
	UL2 [80]	May-2022	20	-	-	-	1T tokens	Apr-2019	512 TPU v4	-	✓	✓
	OPT [81]	May-2022	175	-	-	-	180B tokens	-	992 80G A100	-	✓	-
	NLLB [82]	Jul-2022	54.5	-	-	-	-	-	-	-	✓	-
	GLM [83]	Oct-2022	130	-	-	-	400B tokens	-	768 40G A100	60 d	✓	-
	Flan-T5 [64]	Oct-2022	11	T5	✓	-	-	-	-	-	✓	✓
	BLOOM [69]	Nov-2022	176	-	-	-	366B tokens	-	384 80G A100	105 d	✓	-
	mT0 [84]	Nov-2022	13	mT5	✓	-	-	-	-	-	✓	-
	Galactica [35]	Nov-2022	120	-	-	-	106B tokens	-	-	-	✓	✓
	BLOOMZ [84]	Nov-2022	176	BLOOM	✓	-	-	-	-	-	✓	-
	OPT-IML [85]	Dec-2022	175	OPT	✓	-	-	-	128 40G A100	-	✓	✓
	LLaMA [57]	Feb-2023	65	-	-	-	1.4T tokens	-	2048 80G A100	21 d	✓	-
	CodeGeeX [86]	Sep-2022	13	-	-	-	850B tokens	-	1536 Ascend 910	60 d	✓	-
	Pythia [87]	Apr-2023	12	-	-	-	300B tokens	-	256 40G A100	-	✓	-
Closed Source	GPT-3 [55]	May-2020	175	-	-	-	300B tokens	-	-	-	✓	-
	GShard [88]	Jun-2020	600	-	-	-	1T tokens	-	2048 TPU v3	4 d	-	-
	Codex [89]	Jul-2021	12	GPT-3	-	-	100B tokens	May-2020	-	-	✓	-
	ERNIE 3.0 [90]	Jul-2021	10	-	-	-	375B tokens	-	384 V100	-	✓	-
	Jurassic-1 [91]	Aug-2021	178	-	-	-	300B tokens	-	800 GPU	-	✓	-
	HyperCLOVA [92]	Sep-2021	82	-	-	-	300B tokens	-	1024 A100	13.4 d	✓	-
	FLAN [62]	Sep-2021	137	LaMDA-PT	✓	-	-	-	128 TPU v3	60 h	✓	-
	Yuan 1.0 [93]	Oct-2021	245	-	-	-	180B tokens	-	2128 GPU	-	✓	-
	Anthropic [94]	Dec-2021	52	-	-	-	400B tokens	-	-	-	✓	-
	WebGPT [72]	Dec-2021	175	GPT-3	-	✓	-	-	-	-	✓	-
	Gopher [59]	Dec-2021	280	-	-	-	300B tokens	-	4096 TPU v3	920 h	✓	-
	ERNIE 3.0 Titan [95]	Dec-2021	260	-	-	-	-	-	-	-	✓	-
	GLaM [96]	Dec-2021	1200	-	-	-	280B tokens	-	1024 TPU v4	574 h	✓	-
	LaMDA [63]	Jan-2022	137	-	-	-	768B tokens	-	1024 TPU v3	57.7 d	-	-
	MT-NLG [97]	Jan-2022	530	-	-	-	270B tokens	-	4480 80G A100	-	✓	-
	AlphaCode [98]	Feb-2022	41	-	-	-	967B tokens	Jul-2021	-	-	-	-
	InstructGPT [61]	Mar-2022	175	GPT-3	✓	✓	-	-	-	-	✓	-
	Chinchilla [34]	Mar-2022	70	-	-	-	1.4T tokens	-	-	-	✓	-
	PaLM [56]	Apr-2022	540	-	-	-	780B tokens	-	6144 TPU v4	-	✓	✓
	AlexaTM [99]	Aug-2022	20	-	-	-	1.3T tokens	-	128 A100	120 d	✓	✓
	Sparrow [100]	Sep-2022	70	-	-	✓	-	-	64 TPU v3	-	✓	-
	WeLM [101]	Sep-2022	10	-	-	-	300B tokens	-	128 A100 40G	24 d	✓	-
	U-PaLM [102]	Oct-2022	540	PaLM	-	-	-	-	512 TPU v4	5 d	✓	✓
	Flan-PaLM [64]	Oct-2022	540	PaLM	✓	-	-	-	512 TPU v4	37 h	✓	✓
	Flan-U-PaLM [64]	Oct-2022	540	U-PaLM	✓	-	-	-	-	-	✓	✓
	GPT-4 [46]	Mar-2023	-	-	✓	✓	-	-	-	-	✓	✓
	PanGu-Σ [103]	Mar-2023	1085	PanGu-α	-	-	329B tokens	-	512 Ascend 910	100 d	✓	-



Ref: A Survey of Large Language Models, https://arxiv.org/abs/2303.18223

Ref: https://github.com/Mooler0410/LLMsPracticalGuide

Dataset

Statistics of commonly-used data sources.

Corpora	Size	Source	Latest Update Time
BookCorpus [122]	5GB	Books	Dec-2015
Gutenberg [123]	-	Books	Dec-2021
C4 [73]	800GB	CommonCrawl	Apr-2019
CC-Stories-R [124]	31GB	CommonCrawl	Sep-2019
CC-NEWS [27]	78GB	CommonCrawl	Feb-2019
REALNEWs [125]	120GB	CommonCrawl	Apr-2019
OpenWebText [126]	38GB	Reddit links	Mar-2023
Pushift.io [127]	2TB	Reddit links	Mar-2023
Wikipedia [128]	21GB	Wikipedia	Mar-2023
BigQuery [129]	-	Codes	Mar-2023
the Pile [130]	800GB	Other	Dec-2020
ROOTS [131]	1.6TB	Other	Jun-2022

Echo system graph

ecosystem graphs

TABLE GRAPH

dataset

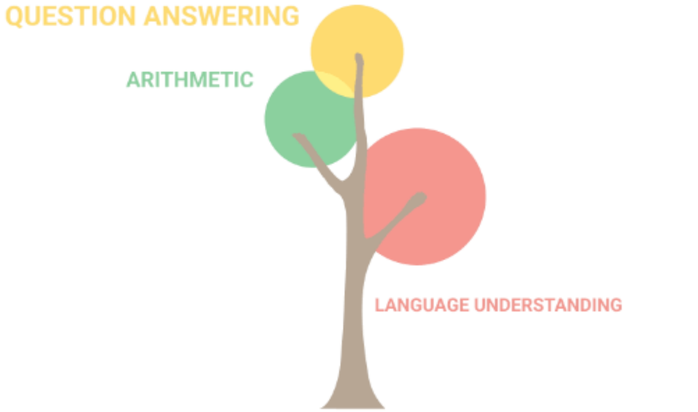
Type	Name	Organization	Created date	Size	Access	Dependencies
dataset	OIG-moderation	Together, LAION, Ontocord	Mar 9, 2023	unknown	open	
dataset	OIG-43M	Together, LAION, Ontocord	Mar 9, 2023	43M instructions	open	P3 | NaturalInstructions-v2 | FLA
dataset	gpt-3.5-turbo dataset	OpenAI	Feb 28, 2023	unknown	limited	
dataset	Rater-SF	Google	Feb 7, 2023	24k captions	closed	MusicCaps
dataset	Rater-LF	Google	Feb 7, 2023	10k captions	closed	MusicCaps
dataset	Noise2Music pseudolabel dataset	Google	Feb 7, 2023	340k hours audio with pseudolabels	closed	Noise2Music audio dataset | Nois pseudolabeler
dataset	Noise2Music audio dataset	Google	Feb 7, 2023	340k hours audio	closed	
dataset	LaMDA-LF	Google	Feb 7, 2023	150k songs	closed	LaMDA
dataset	Phenaki Video-Text Corpus	Google	Jan 31, 2023	15M text-video pairs at 8FPS	closed	
dataset	Flan Collection	Google	Jan 30, 2023	1836 tasks	open	Flan dataset | P3 | NaturalInstruc
dataset	MusicLM dataset	Google	Jan 25, 2023	280K hours audio	closed	
dataset	OpenAI toxicity dataset	OpenAI	Jan 17, 2023	unknown	closed	
dataset	LAION-5B	LAION	Dec 11, 2022	5 billion image-text pairs	open	CommonCrawl

Ref: A Survey of Large Language Models, https://arxiv.org/abs/2303.18223

Ref: data explorer, https://crfm.stanford.edu/ecosystem-graphs

Emergent behavior

- Large size model shows many new capabilities that was not shown smaller size model



Ref: https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html