Lecture 2: Encoder-Decoder Models

Yulan He
King's College London

AthNLP 2025

AI Landscape

Artificial Intelligence

- John McCarthy – "*the science and engineering of making intelligent machines*".
- Tasks include *perception, learning, reasoning, problem-solving, decision-making*.

Machine Learning

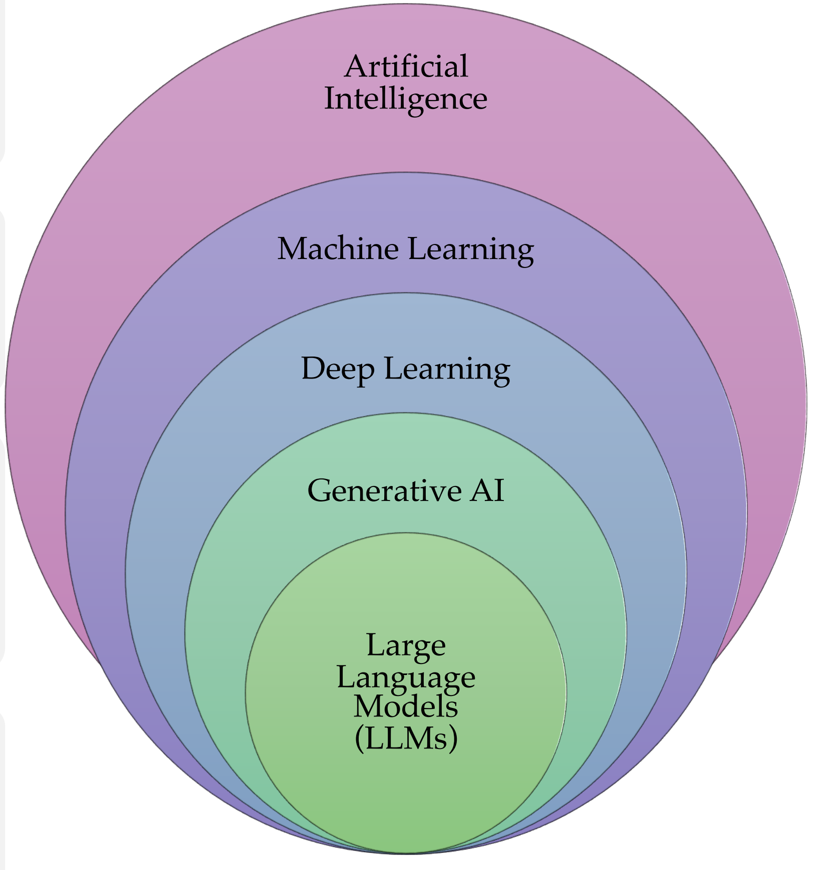
- Algorithms and models *learn from data*.
- Learning approaches include *supervised, unsupervised, semi-supervised, and reinforcement learning*.

Deep Learning

- Utilises *deep artificial neural networks*.
- *Learns representations* of data through multiple layers.
- Effective for tasks such as image recognition, natural language processing, etc.

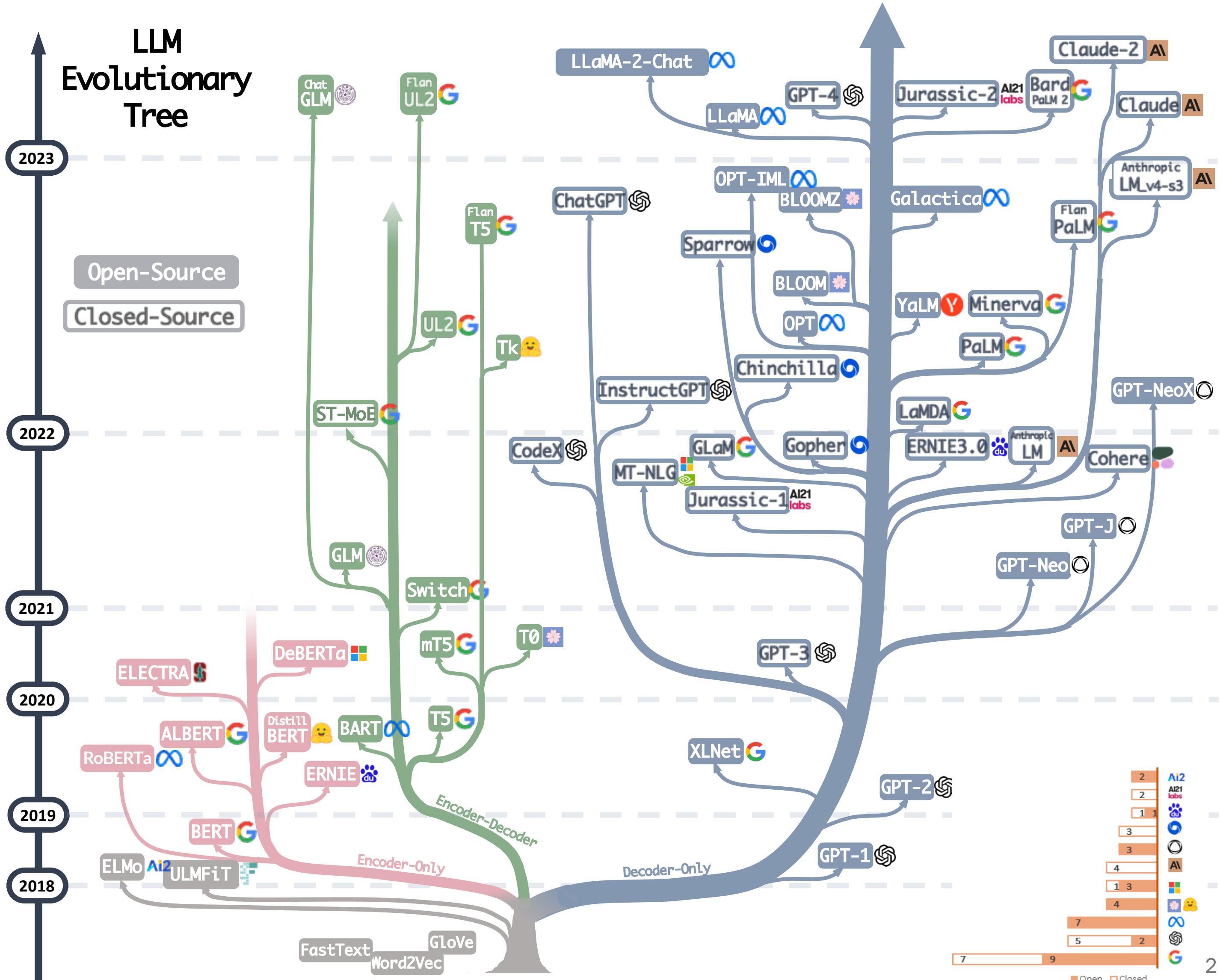
Generative AI

- Focus on creating models to *generate new data*.
- Examples include Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), Large Language Models (LLMs)



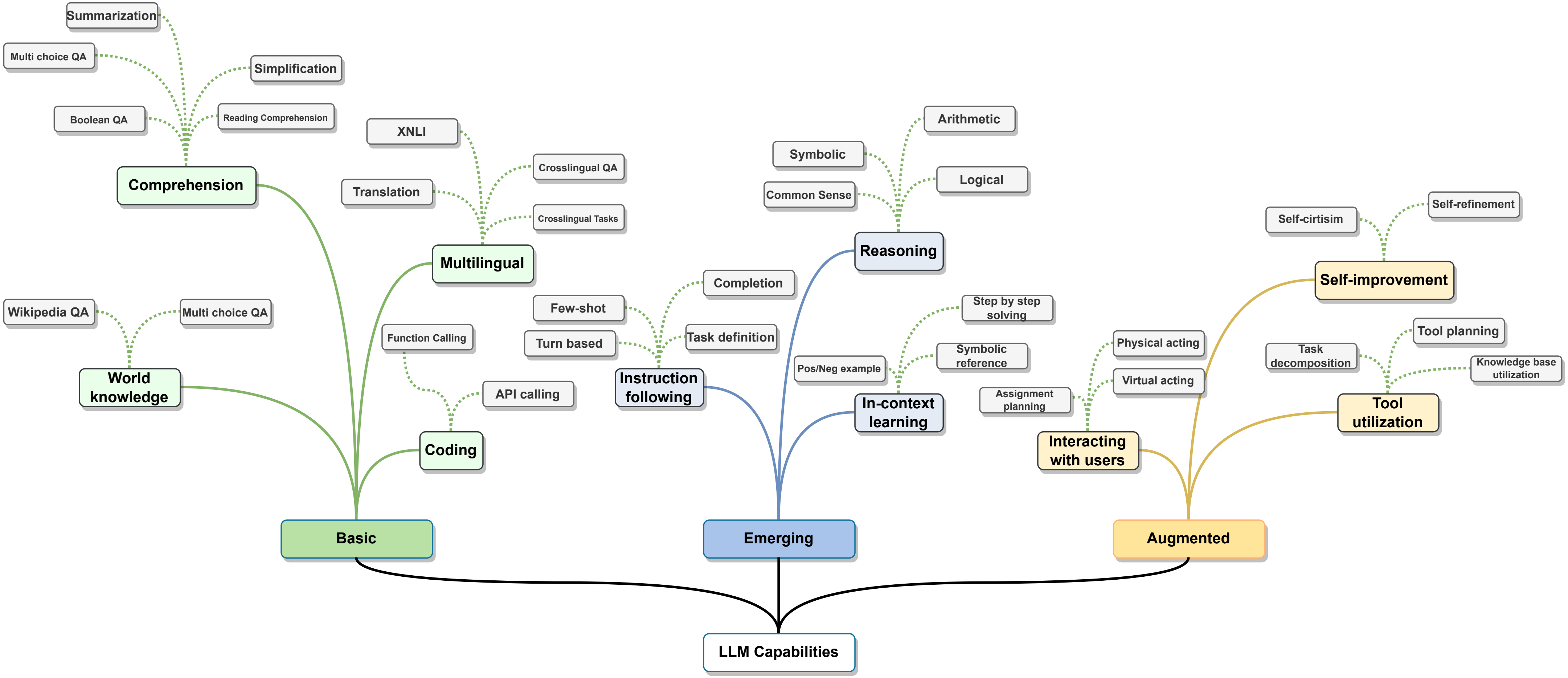
Generative AI

- **Large Language Models**



Yang et al., Harnessing the power of LLMs in practice: A survey on ChatGPT and beyond. *ACM Transactions on KDD*, 18(6), pp.1-32, 2024.

LLM Capabilities



Minaee et al., Large language models: A survey. arXiv preprint arXiv:2402.06196, *2024*.

Generative AI Applications

Art Generation: GAN, VAE and stable diffusion models can create artworks such as paintings and music.

Text Generation: LLMs can generate text such as stories, poetry, and dialogues.

Image Editing: Tools like StyleGAN can be used for photo editing and realistic image synthesis.

Content Creation: Generative AI can assist in content creation for various media, including video games, movies, and advertising, by generating characters, scenes, and scenarios.

Drug Discovery: generate novel molecular structures with desired properties, potentially speeding up drug discovery processes.

Design Assistance: AI can assist designers by generating design suggestions for products, architecture, etc., based on specified criteria and constraints.

Simulation and Prediction: Generative models can simulate real-world scenarios and predict outcomes, useful in fields like climate science, economics, and epidemiology.

Data Augmentation: create synthetic data to augment existing datasets for training ML models.

Outline

- **Part I: Fundamentals**
  - Language Models
  - N-grams
  - Feedforward Neural Network (FFNN) Language Models
- **Part II: RNNs and Attentions**
  - Recurrent Neural Networks (RNNs / LSTMs / GRUs)
  - Sequence-to-Sequence learning
  - Attentions
- **Part III: Transformer and LLMs**
  - The Transformer Architecture
  - Language Models Built on Transformer
  - LLM Training Paradigms
  - LLM Evaluation

Part I: Fundamentals

- Language Models
- N-grams
- Feedforward Neural Network (FFNN) Language Models

What is a Language Model (LM)?

- A model of computing either of the following is called a **Language Model**:
  - the probability of **a sequence of words**:

    p(An NLP summer school happens in Athens)=? ?

    $p(W) = p(w_1, w_2, \ldots, w_n)$

  - the probability of **the upcoming word**:

    p(Athens | An NLP summer school happens in)=? ?

    $p(w_i | w_1, w_2, \ldots, w_{i-1})$

Language Model

- How to estimate the probability $p(W) = p(w_1, w_2, \ldots, w_n)$?

- We can rely on the **Chain Rule of Probability**

$$p(W) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2) \ldots$$

$$= p(w_1) \sum_{i=2}^{n} p(w_i|w_1, \ldots, w_{i-1})$$

Computing $p(W)$ using the chain rule

p(An NLP summer school happens in Athens)=

p(An)×
p(NLP | An)×
p(summer | An NLP)×
p(school | An NLP summer)×
p(happens | An NLP summer school)×
p(in | An NLP summer school happens)×
p(Athens | An NLP summer school happens in)×

How do we compute probabilities?

- Based on the number of occurrences?

$$p(\text{Athens} \mid \text{An NLP summer school happens in}) =$$

$$\frac{\text{count(An NLP summer school happens in Athens)}}{\text{count(An NLP summer school happens in)}}$$

- **Problem:** there are so many different sequences, we won't observe enough instances in our data!

Markov Assumption

- **Approximate** the probability **by simplifying** it:
  - 1st order Markov assumption

    $$p(\text{Athens} \mid \text{An NLP summer school happens in}) \approx p(\text{Athens} \mid \text{in})$$

  - 2nd order Markov assumption

    $$p(\text{Athens} \mid \text{An NLP summer school happens in}) \approx p(\text{Athens} \mid \text{happens in})$$

- It's much **more likely** that we'll observe "*in Athens*" or "*happens in Athens*" in our training data.

Markov Assumption

- Which we can generalise as ***k*^th^-order Markov assumption:**

$$p(w_i|w_1, w_2, \dots, w_{i-1}) \approx p(w_i|w_{i-k}, w_{i-k+1}, \dots, w_{i-1})$$

i.e., we will only look at the **last *k* words**

N-grams

- **N-gram:** sequence of *n* words

- e.g. I want to go to the cinema
  - 2-grams (**bigrams**): I want, want to, to go, go to, to the,…
  - 3-grams (**trigrams**): I want to, want to go, to go to,…
  - 4-grams: I want to go, want to go to, to go to the,…
  - …

Computing n-gram Probabilities

- Let's say we have the following sentences to learn our language models:

  see what I found

  you found a penny

  it has been found

  the book you found

  you came yesterday

What is the probability of the **bigram** "you found"?

With the **1st-order Markov assumption**:

$$P(you, found) = P(found \mid you)$$

Computing n-gram Probabilities

- Let's say we have the following sentences to learn our language models:

  see what I found

  you found a penny

  it has been found

  the book you found

  you came yesterday

What is the probability of the **bigram** "you found"?

With the **1st-order Markov assumption**:

$$P(you, found) = P(found \mid you) = \frac{\text{count}(you\ found)}{\text{count}(you)} = \frac{2}{3}$$

Language Models

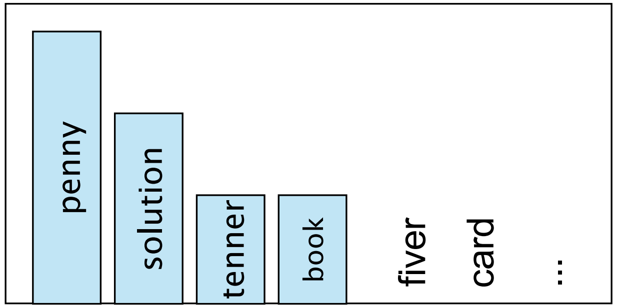
- We can go with unigram, bigrams, trigrams, 4-grams,…
  - **Unigram LM**: $p(w_1, w_2, \ldots, w_n) = \sum_{i=1}^{n} p(w_i)$
  - **Bigram LM**: $p(w_1, w_2, \ldots, w_n) = p(w_1) \sum_{i=2}^{n} p(w_i | w_{i-1})$
  - **trigram LM**: $p(w_1, w_2, \ldots, w_n) = p(w_1) p(w_2 | w_1) \sum_{i=3}^{n} p(w_i | w_{i-2}, w_{i-1})$

- Note: the longer the length:
  - The **more detailed** our language model
    i.e. long sequences will capture more grammar than short sequences
  - But **the more sparse** our counts
    i.e. many observations only seen once

The Intuition of Smoothing

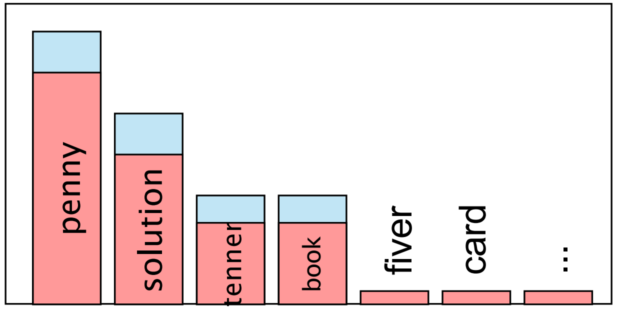
- We have **sparse** statistics:

  P(w | "found a")
  3 → penny
  2 → solution
  1 → tenner
  1 → book
  7 → **Total count**



- We'd like to improve the distribution:

  P(w | "found a")
  3 → penny → 2.5
  2 → solution → 1.5
  1 → tenner → 0.5
  1 → book → 0.5
  Other → 2
  7 → **Total count**



Smoothing

- Relocate probability mass to make generalisation better

- Laplace smoothing (add-one smoothing)
  - Pretend we saw **each word one more time** than we actually did.
  - Just add one to all counts, and adjust normalization
  - MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

  - Add-one estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Evaluation of a Language Model

- We want to evaluate **whether our language model is good**.
  - i.e. does our language model prefer good sentences to bad ones?

- i.e. does it assign **higher probability**:
  - to “real” or “frequent” sentences (e.g. **I want to**)
  - than “ungrammatical” or “rarely observed” sentences? (e.g. **want I to**)

Evaluation of a Language Model

- **Evaluation:**
  - Is our language model good in **giving high probabilities** to sentences in our corpus?

- Usually done in a **comparative** way:
  - Train language model 1 (LM1) from corpus 1.
  - Train language model 2 (LM2) from corpus 2.
  - For sentences in corpus 3, which of LM1 and LM2 is giving me higher probabilities?

- We need an **evaluation metric** to determine which of LM1 or LM2 is best.

Evaluation Approaches

- Two different evaluation approaches:
  - **Extrinsic or in-vivo** evaluation
    i.e. Test LMs in some **NLP task** (sentiment analysis, machine translation, spell corrector, etc.).
  - **Intrinsic or in-vitro** evaluation
    i.e. evaluate LMs directly – how good can the model **assign probabilities to real unseen data**?

Intrinsic Evaluation: Perplexity

- **Perplexity:**

  Given a language model, on average:
  How difficult is it to **predict the next word**?

  e.g. I always order pizza with cheese and _____ → ???

Intrinsic Evaluation: Perplexity

- **The Shannon Game:**
  - How well can we predict the next word?

pizza with cheese and ___

- mushrooms 0.1
- pepperoni 0.1
- jalapeños 0.01
- ....
- biscuits 0.000001

- **A better model:** the one that gives **higher** probability to the **actual next word**.

- If the actual sentence is "*pizza with cheese and biscuits*", my model is quite bad.
- If the actual sentence is "*pizza with cheese and mushrooms*", my model is **better**.

Intrinsic Evaluation: Perplexity

- The **best LM** is the one that is the best at predicting the test set → will **give test sentences the highest probability**.
- **Perplexity** is the *inverse probability of the test set*, normalised by the number of words.
  - Given a set of test sentences D with a total of N words:

$$PP(D) = p(w_1, w_2, \dots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{p(w_1, w_2, \dots, w_N)}}$$

  - Lower perplexity is better.

Perplexity as a Branching Factor

- Under a uniform distribution, perplexity will be the **vocabulary size**.
  - Suppose we have sentences consisting of **random digits** [0-9], $|V| = 10$
  - What is the **perplexity** of the data for a model that **assigns the same probability** to each digit?

$$PP(D) = p(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} = \left(\frac{1}{10}^N\right)^{-\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

- **Perplexity** is the **weighted average branching factor** of a language.
  - i.e., **the number of possible next word** that can follow any word.

Limitations of N-gram Language Models

- **Fixed context window**
  - Only looks at the last n– 1 words → **ignores longer dependencies**.
  - E.g., "*The book that I borrowed from the library … was fascinating*"
  - A **bigram/trigram model** struggles to connect "*book … was*".
- **Smoothing is imperfect**
  - Fixes zero probabilities but often **underestimates rare yet valid sequences**.
- **Not semantically aware**
  - Counts surface forms, **not meaning**.
  - E.g., "*He eats a cake*" ≠ "*A cake is eaten by him*".

Neural Language Models (LMs)

Language Modeling: Calculating ***probability of the next word*** in a sequence ***given previous context***.

Traditional approach: N-gram based LMs

Modern approach: Neural LMs (outperform n-grams)

State of the art: Transformer-based models

Key insight: Even simple feed-forward LMs can perform surprisingly well.

Simple Feedforward Neural Language Models

- Previously, we compute $p(W) = p(w_1, w_2, \ldots, w_n)$
- using the **Chain Rule of Probability**

$$p(W) = p(w_1) \sum_{i=2}^{n} p(w_i | w_1, \ldots, w_{i-1})$$

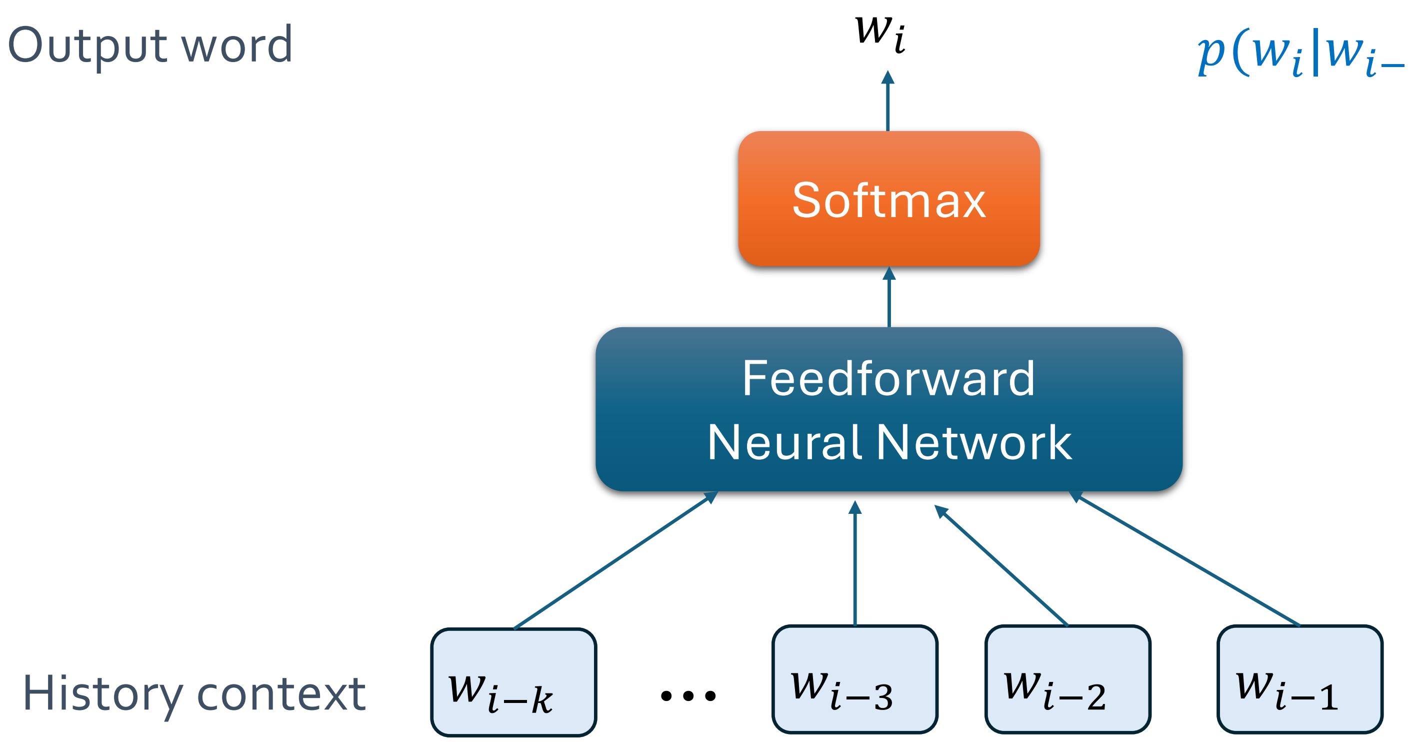
- and make Markov assumption to limit the history

$$p(w_i | w_1, w_2, \ldots, w_{i-1}) \approx p(w_i | w_{i-k}, w_{i-k+1}, \ldots, w_{i-1})$$

- **Task**: predict next word w_i given prior words $w_{i-1}, w_{i-2}, w_{i-3}, \ldots$
- **Solution**: using neural networks for probability estimation

Simple Feedforward Neural Language Models

$p(w_i | w_{i-k}, w_{i-k+1}, \dots, w_{i-1}; \boldsymbol{\theta})$



Simple Feedforward Neural Language Models

- **Problem**: We are dealing with sequences of arbitrary length.
- **Solution**: Sliding windows (of fixed length)

$$p\left(w_i \middle| w_1^{i-1}\right) \approx p(w_i | w_{i-k}^{i-1})$$

A Fixed-window Neural Language Model

output distribution

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

hidden layer

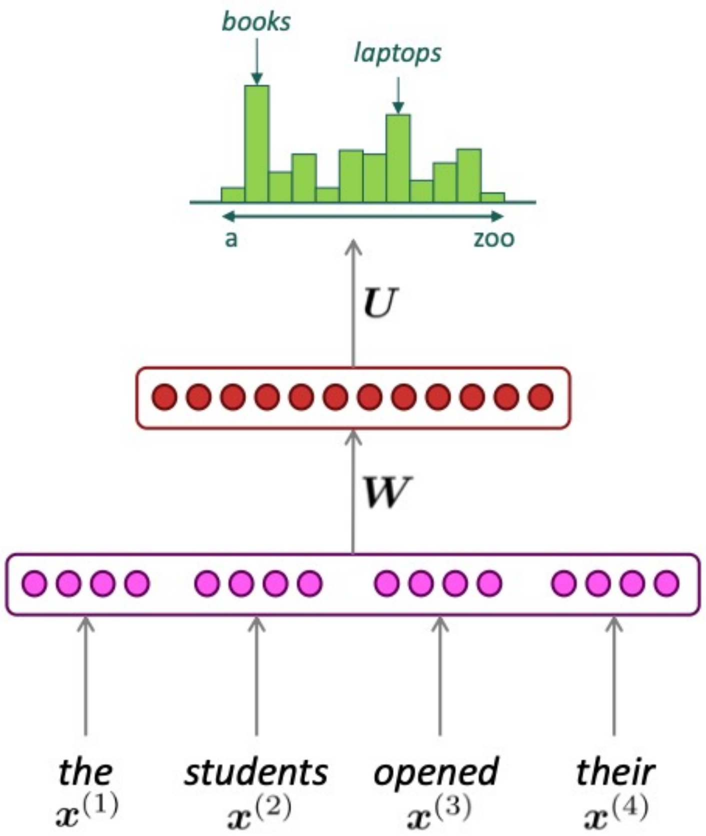
$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$

concatenated word embeddings

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

words / one-hot vectors

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



Bengio et al., 2003. A neural probabilistic language model. Journal of machine learning research, 3(Feb), pp.1137-1155.

Credit: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

A Fixed-window Neural Language Model

- **Improvements over N-gram LMs:**
  - No sparsity problem
  - No need to store all observed n-grams

- **Challenges:**
  - **Context window** is too **small**
    - Increasing window size $\rightarrow$ much larger parameter matrix W
  - Window can never fully capture long-range context
  - **Inputs at different positions** use **different weights** in W
    - $\rightarrow$ No symmetry in how inputs are processed

Can we have a **neural architecture** that can process ***arbitrary length*** input?

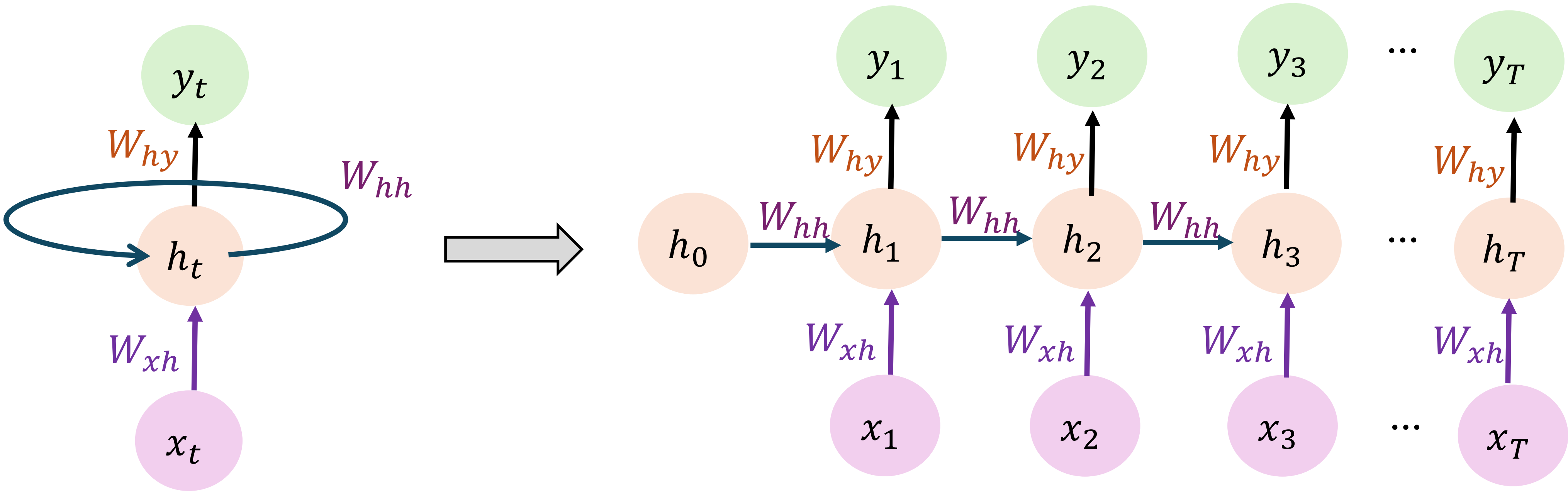
Part II: RNNs and Attentions

- Recurrent Neural Networks (RNNs / LSTMs / GRUs)
- Sequence-to-Sequence learning
- Attentions

Recurrent Neural Networks (RNNs)

- A family of neural networks designed for **sequential data.**
- Handle **variable-length input** naturally.
- Capture **word order.**
- Can model **long-range dependencies** (especially gated variants like LSTMs/GRUs).
- **Do not rely on the Markov assumption** when used as language models.

Recurrent Neural Networks (RNNs)



Recurrent Neural Networks (RNNs)

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

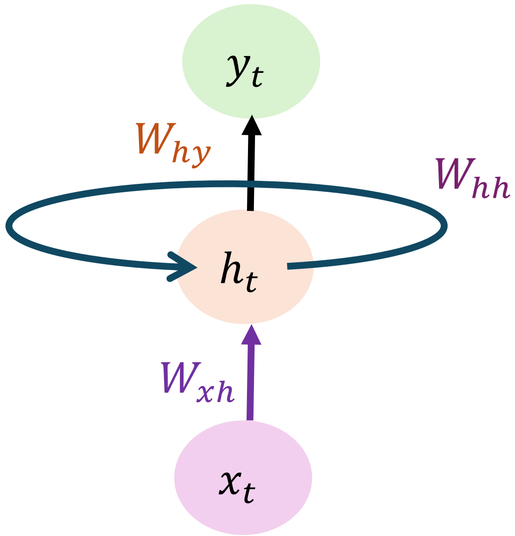
$$h_t = f_W(h_{t-1}, x_t)$$

New state: h_t

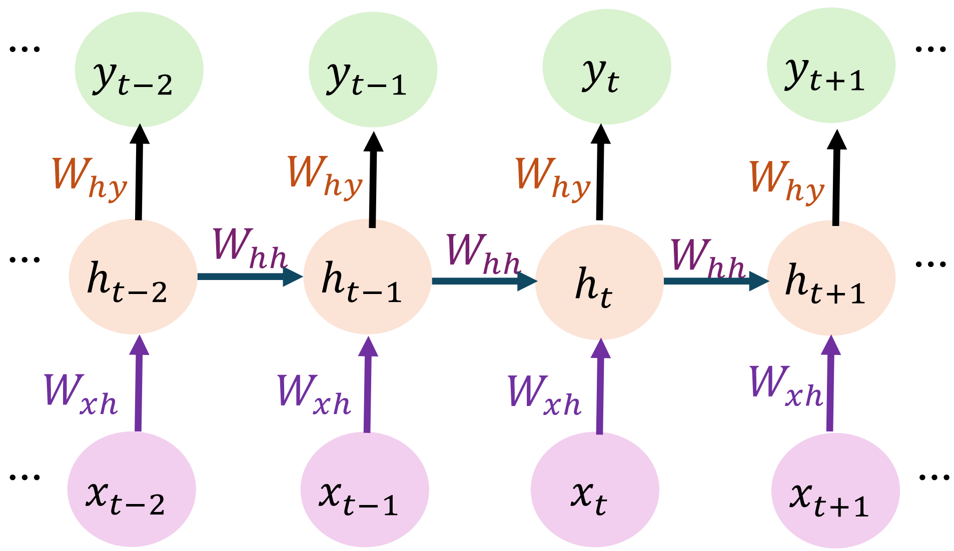
some function with parameters W: f_W

Old state: h_{t-1}

Input vector at some time step: x_t



Recurrent Neural Networks (RNNs)

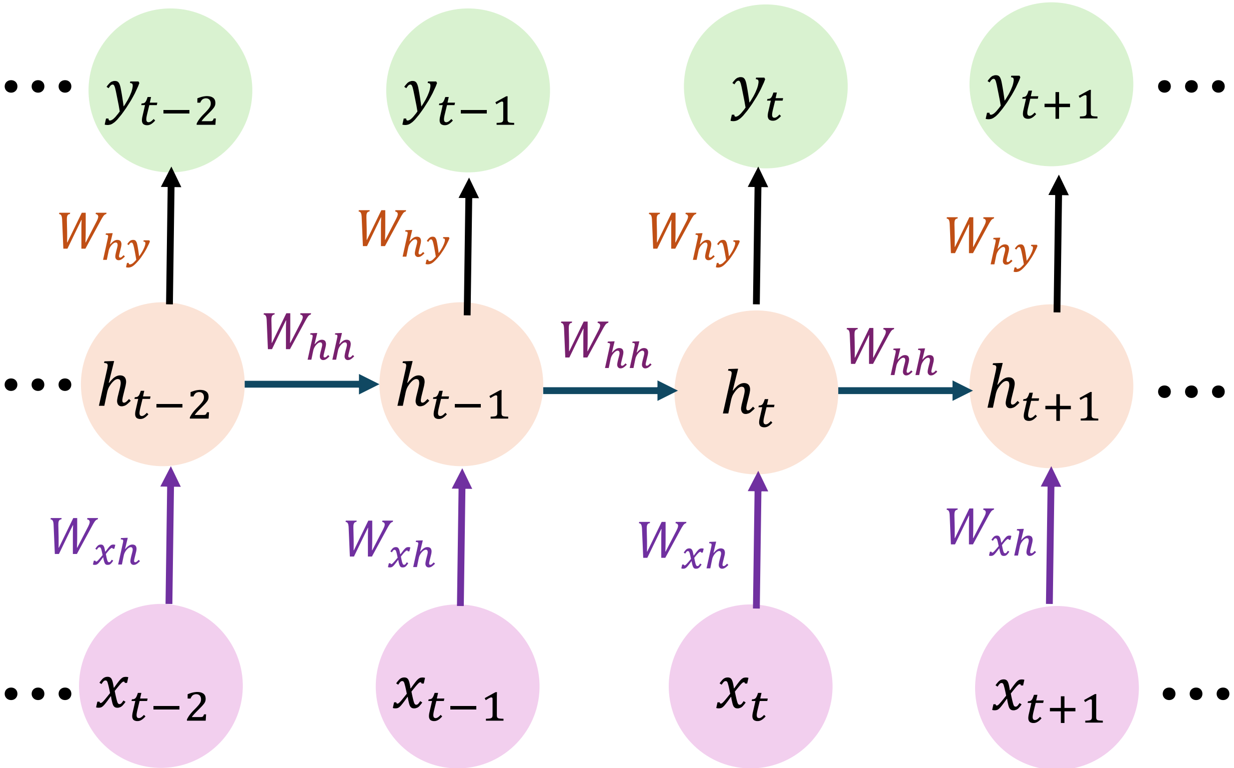


We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

The **same function** and the **same set of parameters** are used at every time step.

(Simple) Recurrent Neural Network

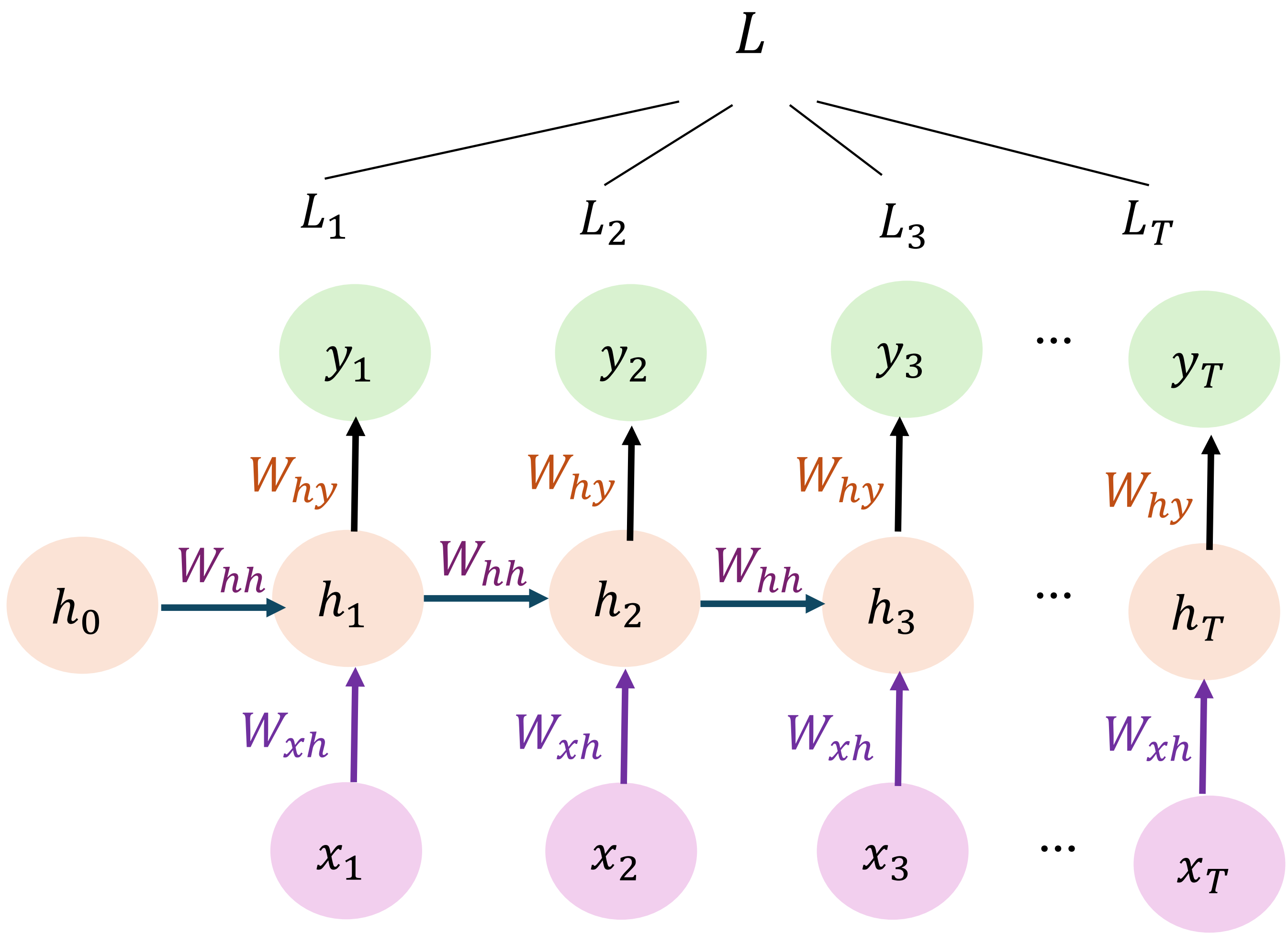


$$y_t = W_{hy} h_t$$

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

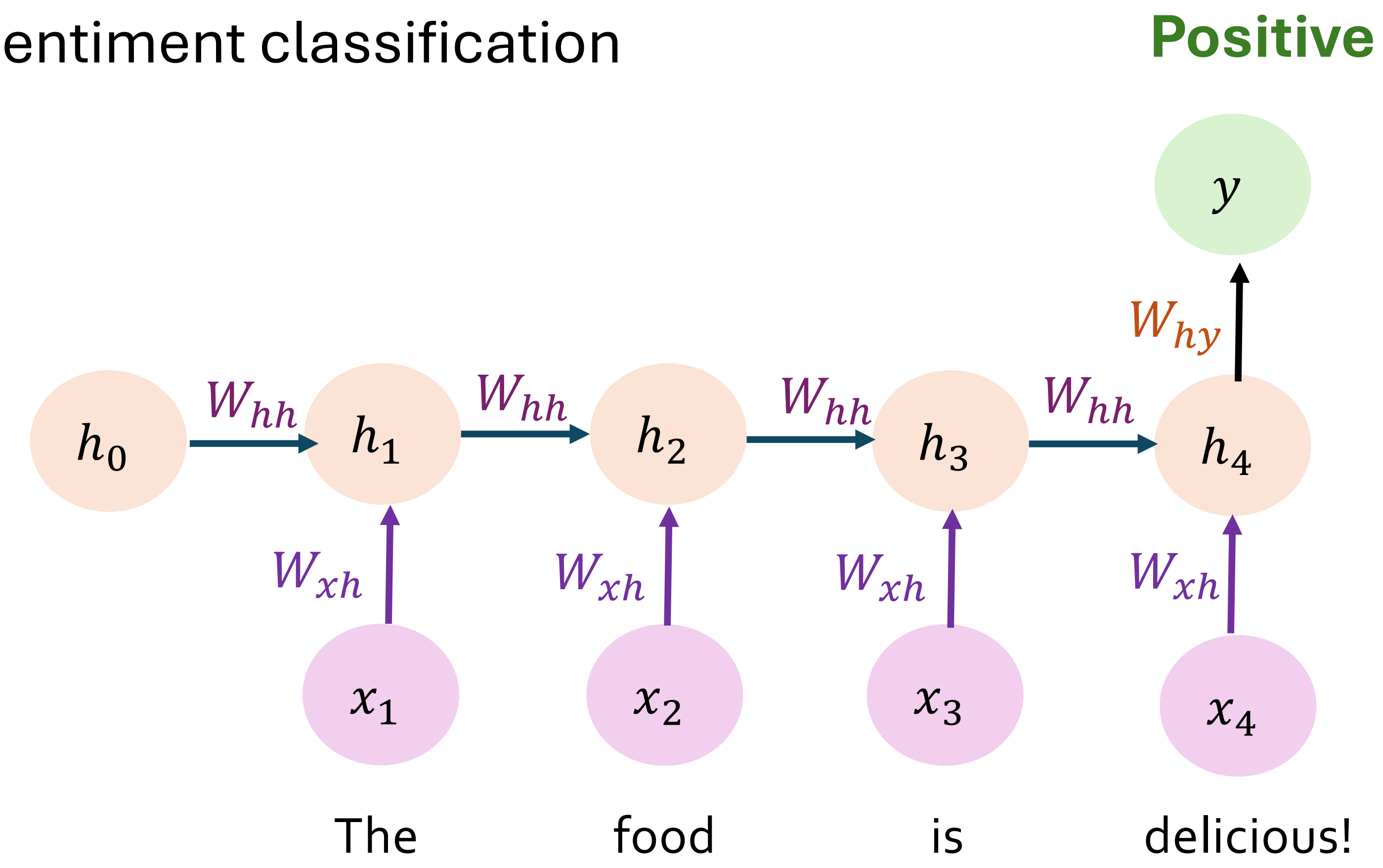
- The state consists of a single *"hidden"* vector **h**
- Re-use the **same weight matrix** at every time-step

RNN: Computational Graph: Many to Many



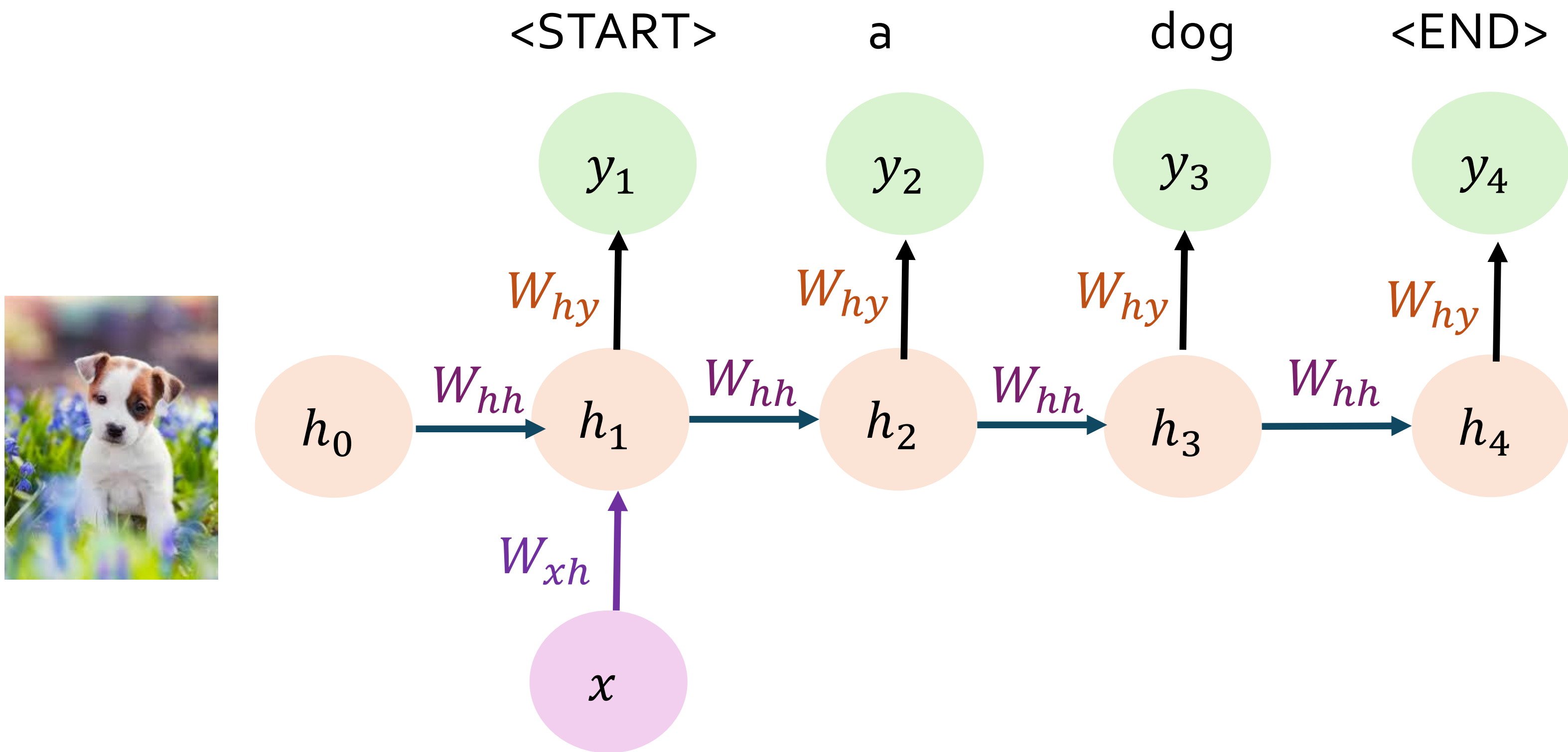
RNN Computational Graph: Many to One

E.g. sentiment classification



RNN Computational Graph: One to Many

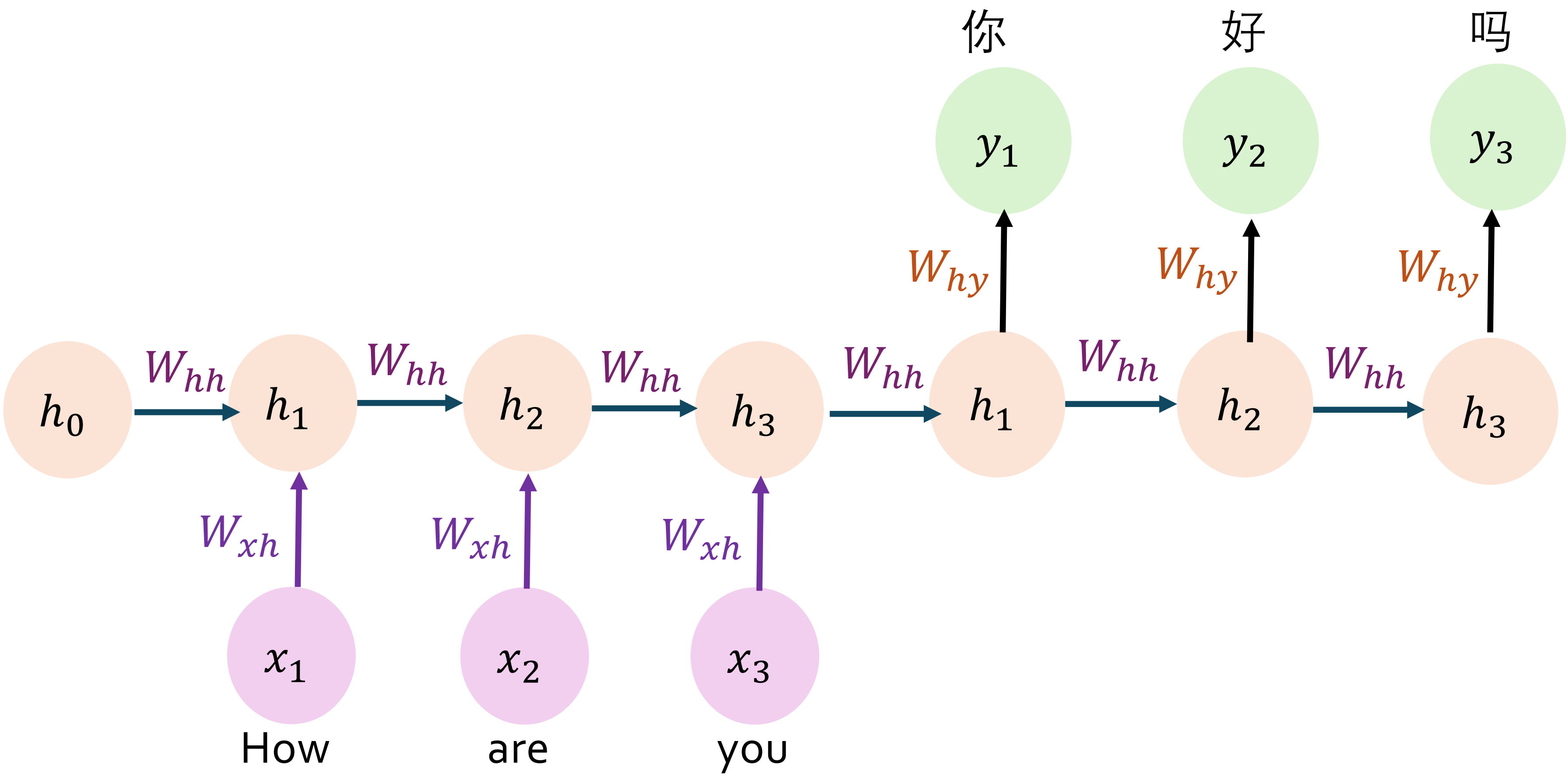
E.g. image captioning



Sequence to Sequence: many-to-one + one-to-many

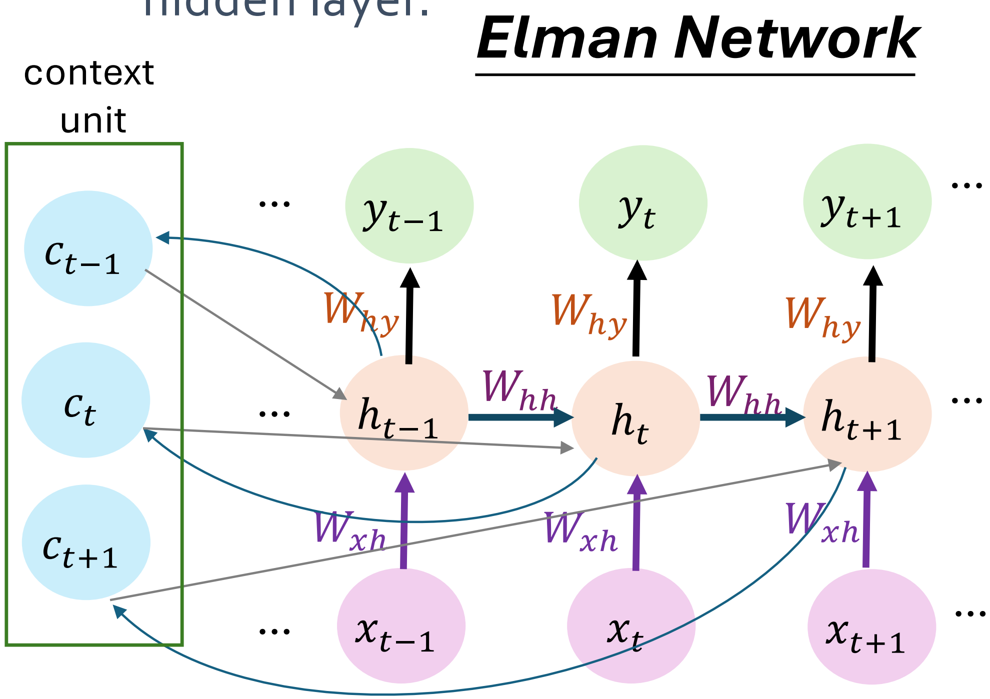
Many to one: Encode input sequence in a single vector

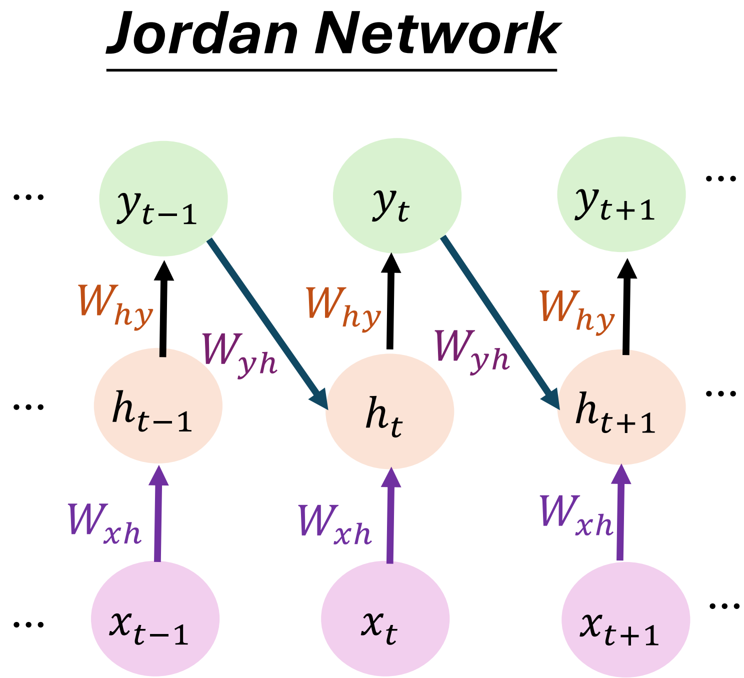
One to many: Produce output sequence from single input vector



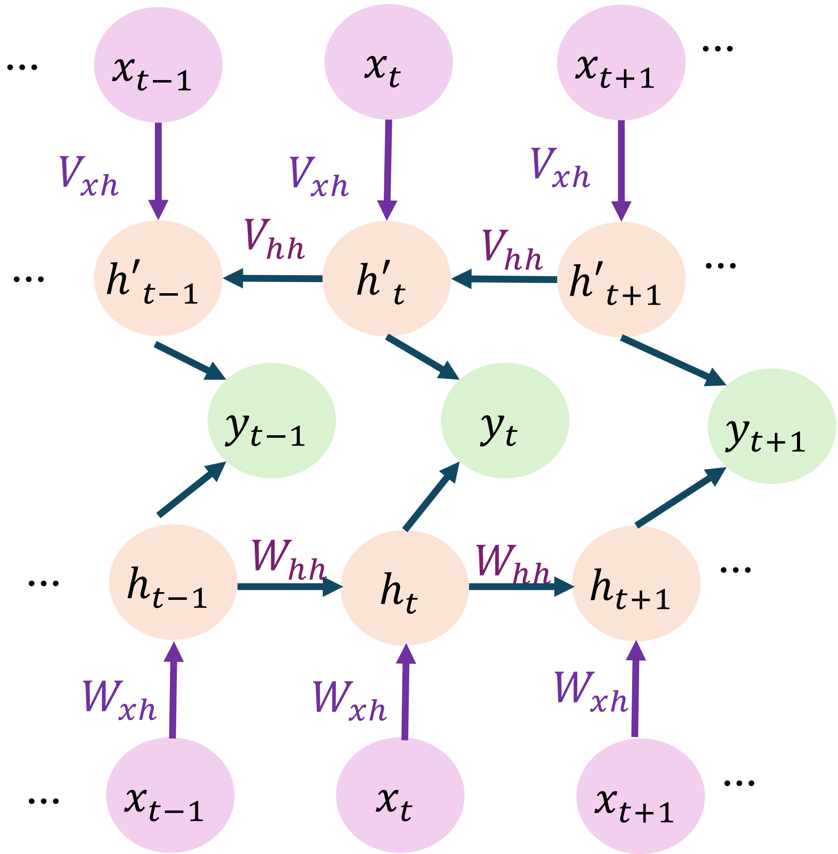
Simple RNN: Elman Network & Jordan Network

- Elman Network – a three-layer network with the addition of a set of "context units" which connects to the hidden layer fixed with a weight of one
- Jordan network – the context units are fed from the output layer instead of the hidden layer.



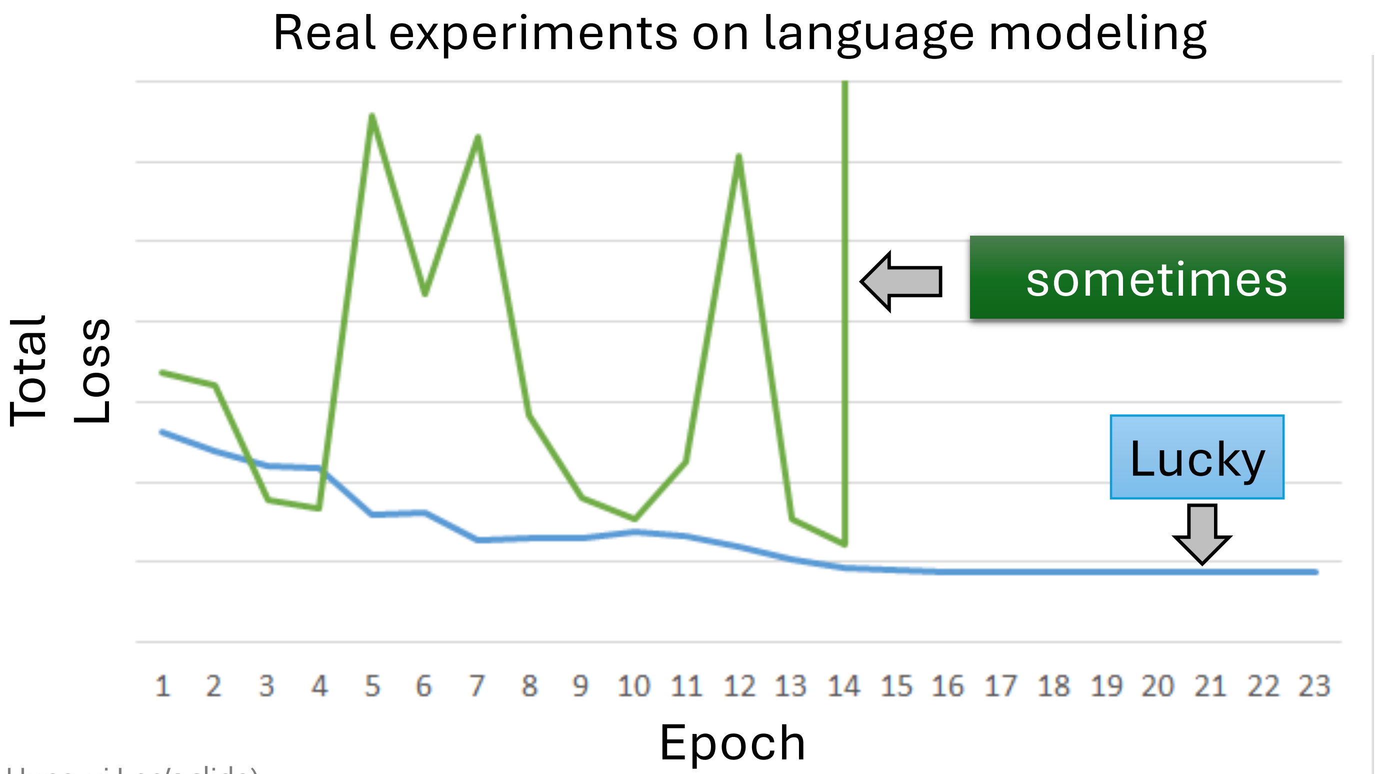


Bidirectional RNN



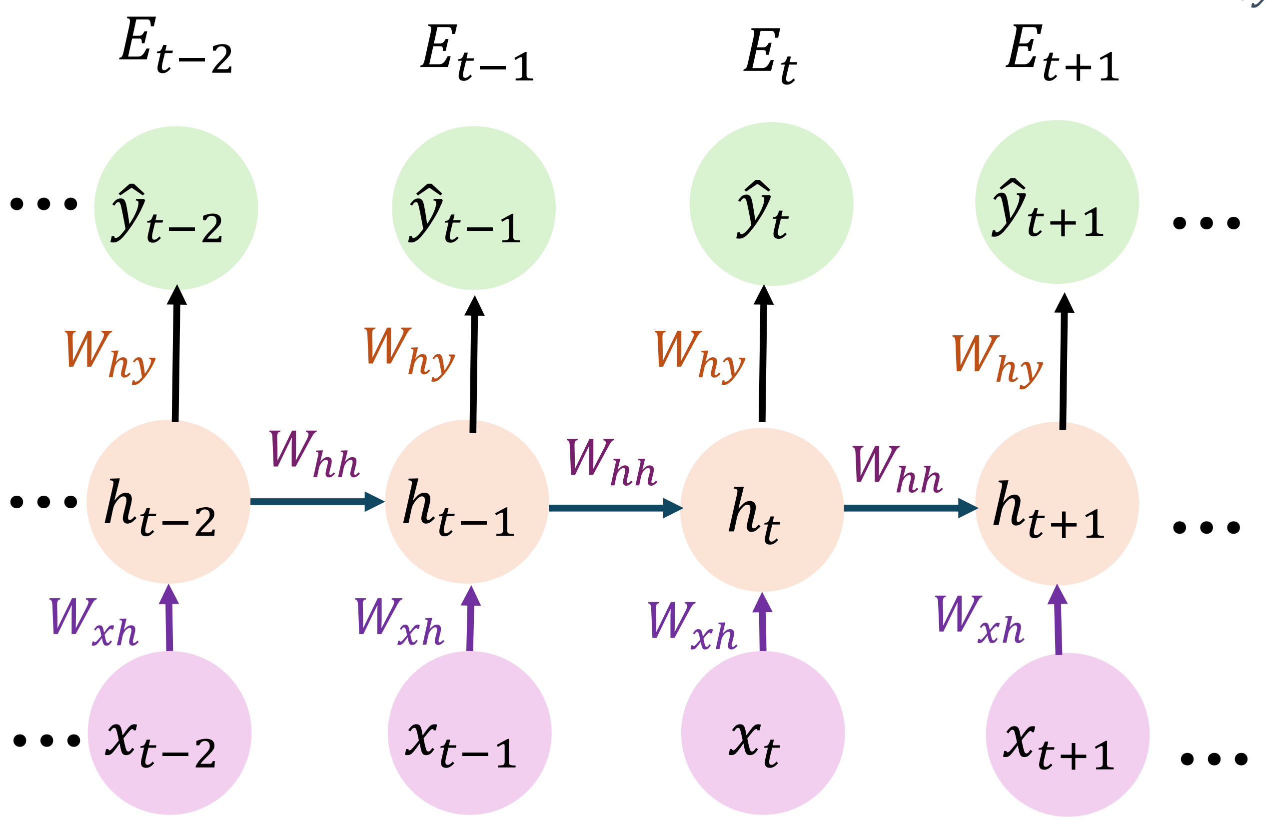
Unfortunately ……

- RNN-based network is not always easy to learn



(Adapter from Hung-yi Lee's slide)

Vanilla RNN Gradient Flow - $\frac{\partial E_t}{\partial W_{hy}}$



$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t \sum_{\mathbb{C}} y_t \log \hat{y}_t$$

$$\hat{y}_t = \text{softmax}(W_{hy} h_t)$$

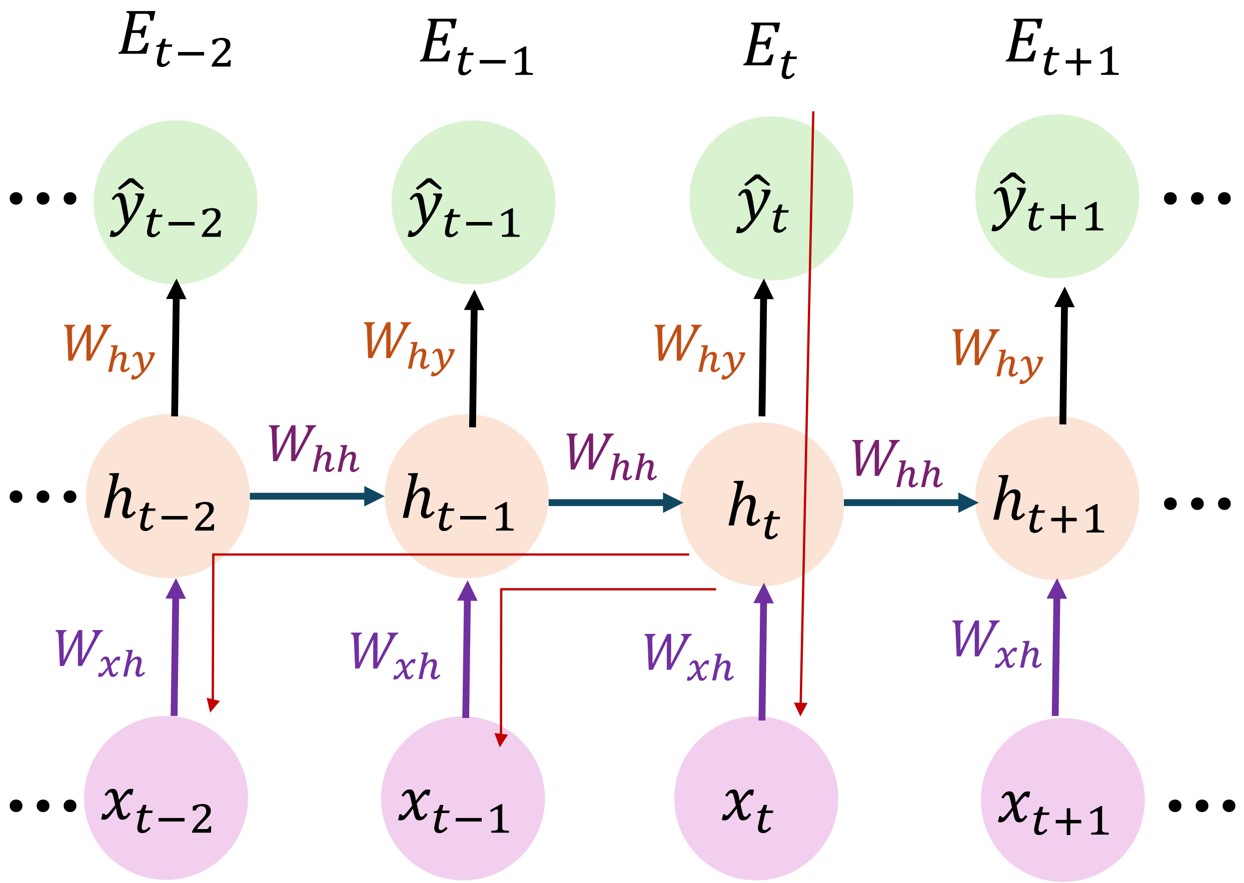
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

For individual cost term

$$\frac{\partial E_t}{\partial W_{hy}} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W_{hy}} = (\hat{y}_t - y_t) h_t^T$$

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Vanilla RNN Gradient Flow - $\frac{\partial E_t}{\partial W_{hh}}$



$$E(y,\hat{y}) = \sum_t E_t(y_t,\hat{y}_t) = -\sum_t \sum_{\mathbb{C}} y_t \log \hat{y}_t$$

$$\hat{y}_t = \text{softmax}(W_{hy} h_t)$$

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

$$\frac{\partial E_t}{\partial W_{hh}} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial W_{hh}}$$

$$= \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\sum_{k=1}^{t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_{hh}}$$

Vanilla RNN Gradient Flow

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

$$\frac{\partial E_t}{\partial W_{hh}} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$

when performing $\frac{\partial h_t}{\partial W_{hh}}$, we need to sum over all intermediate latent nodes, i.e.

$$\frac{\partial h_t}{\partial h_1} \frac{\partial h_1}{\partial W_{hh}} + \frac{\partial h_t}{\partial h_2} \frac{\partial h_2}{\partial W_{hh}} + \ldots + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{hh}}$$

Rewrite $\frac{\partial h_t}{\partial h_k}$ to fill in the gap with chain rule:

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W_{hh}^{\mathrm{T}} \mathrm{diag}(\tanh'(W_{hh} h_{i-1} + W_{xh} x_t))$$

Backpropagation from h_t to h_k multiplies by W_{hh}^{T} many times

Vanilla RNN Gradient Flow

$$\frac{\partial E_t}{\partial W_{hh}} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\sum_{k=1}^{t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} W_{hh}^{\mathrm{T}}\operatorname{diag}(\tanh'(W_{hh}h_{i-1} + W_{xh}x_t))$$

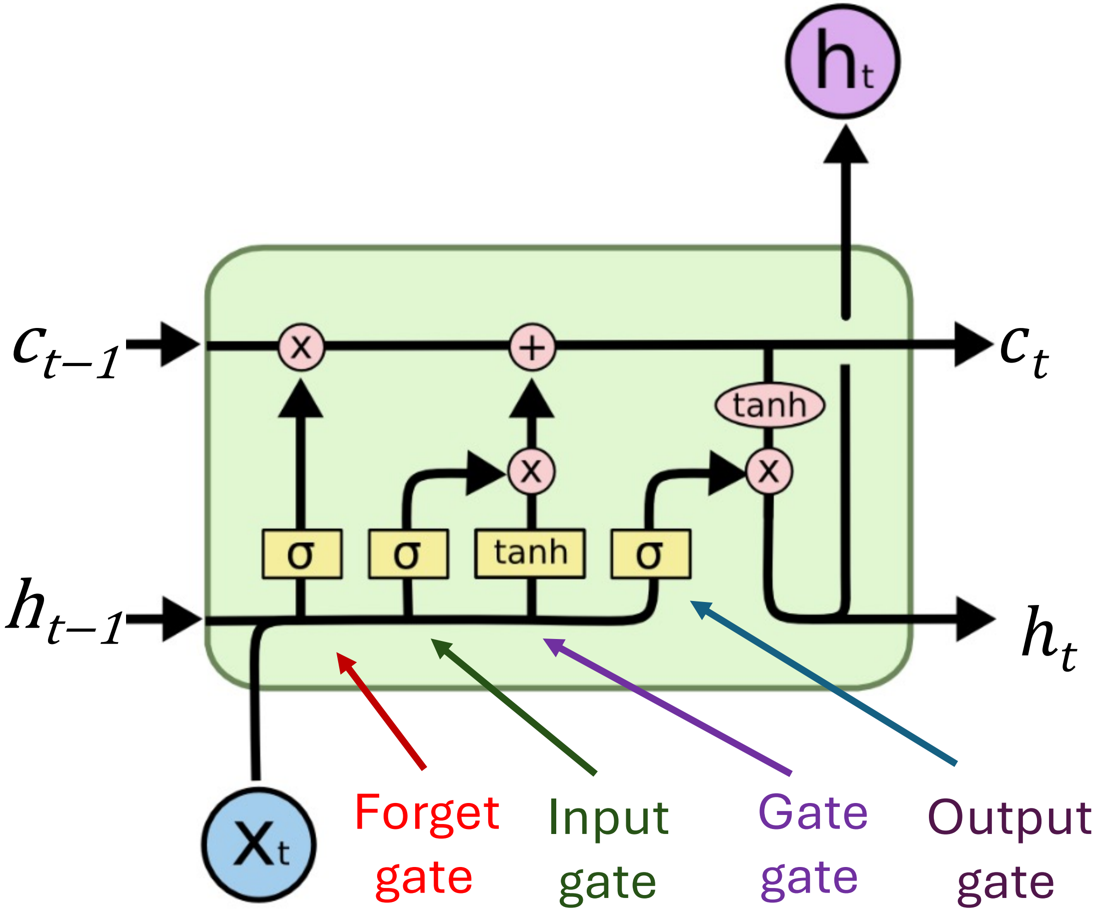
Computing gradient of h_t involves many factors of W_{hh} (and repeated tanh)

W_{hh}^T large:
Exploding gradients → **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

W_{hh}^T small:
Vanishing gradients → Change RNN architecture

Long Short Term Memory (LSTM) [Hochreiter et al., 1997]



i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

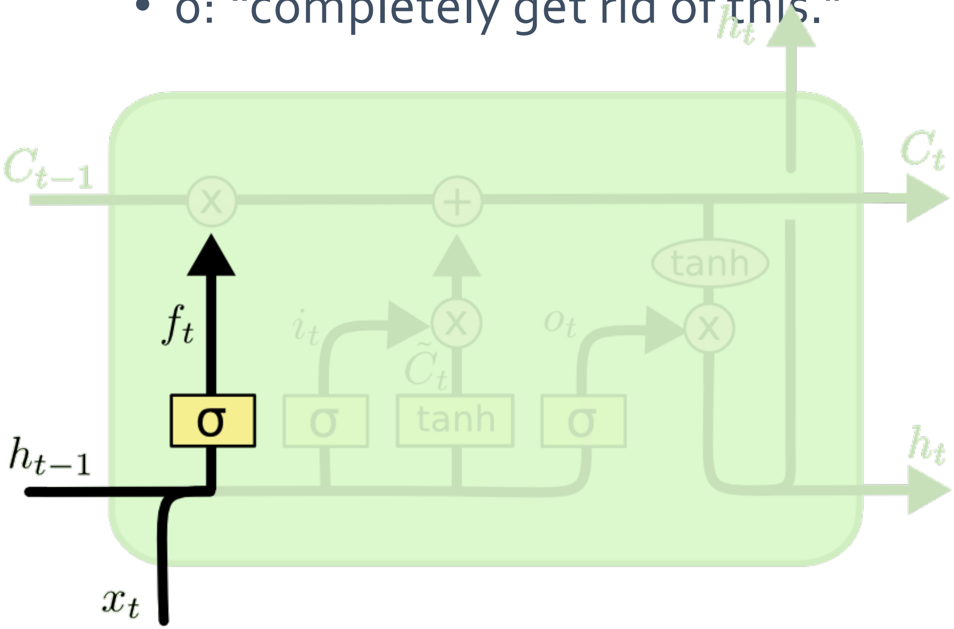
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

This and related figures from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Step-by-Step LSTM Walk Through

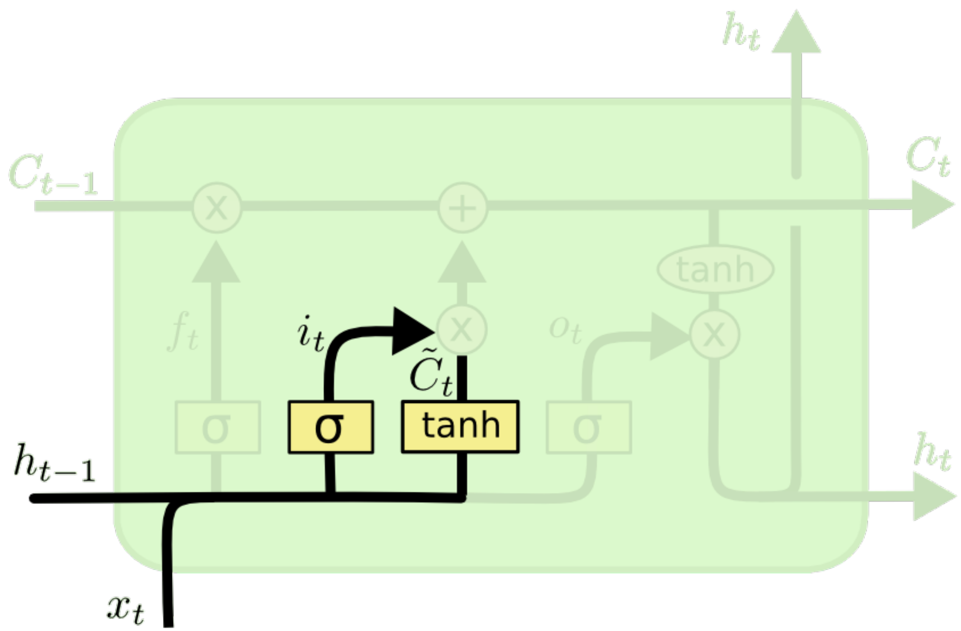
- **Step 1:** what information we're going to throw away from the cell state.
- Forget gate – outputs a number between 0 and 1
  - 1: "completely keep this"
  - 0: "completely get rid of this."



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Step-by-Step LSTM Walk Through…

- **Step 2:** what new information we're going to store in the cell state.
  - **Step 2.1:** input gate – whether to write to cell.
    Gate gate – how much to write to cell

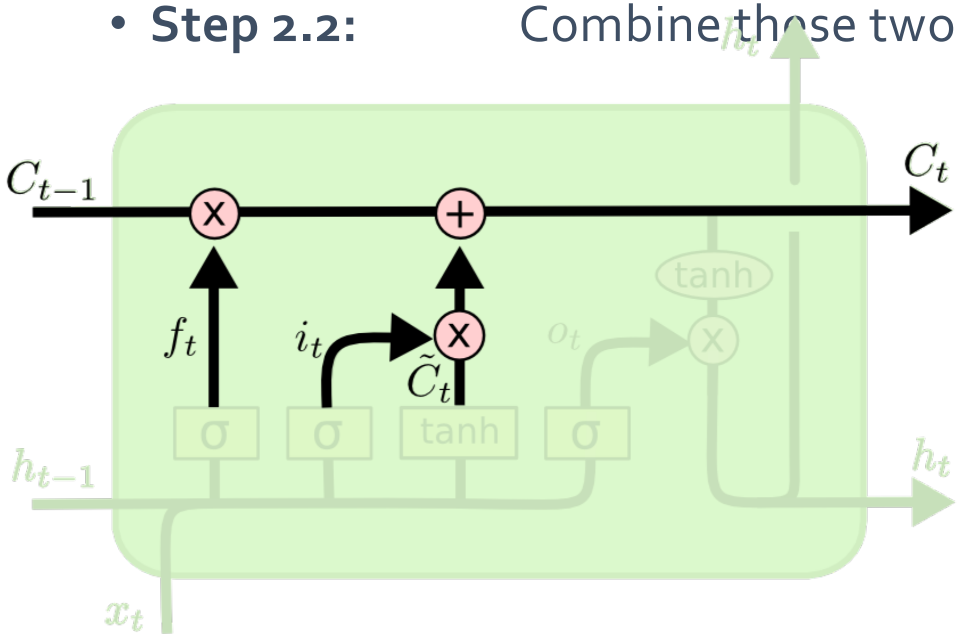


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Step-by-Step LSTM Walk Through…

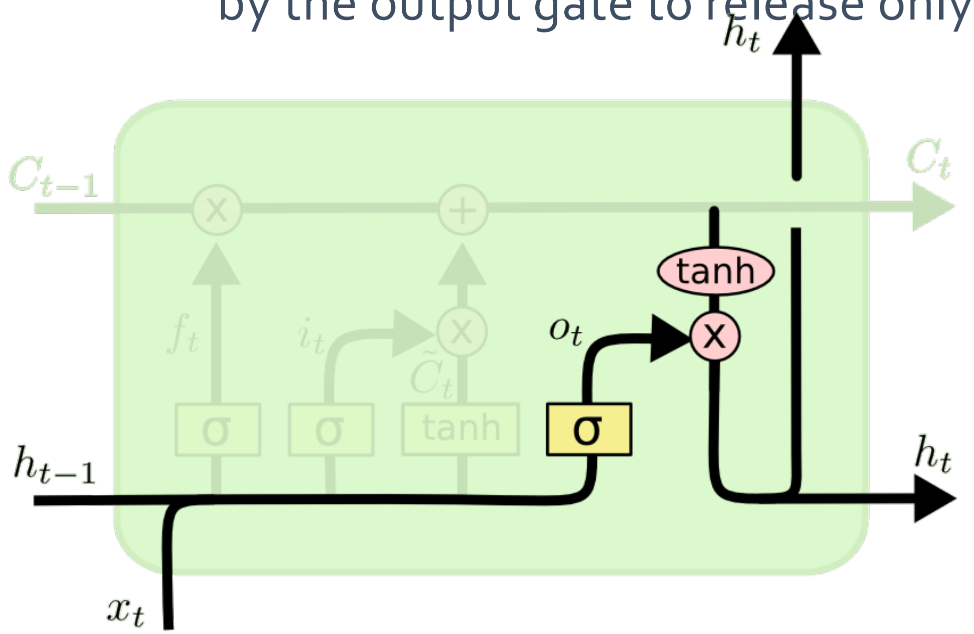
- **Step 2:** what new information we're going to store in the cell state.
  - **Step 2.1:** input gate – whether to write to cell.
    Gate gate – how much to write to cell
  - **Step 2.2:** Combine these two to create an update to the cell.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step-by-Step LSTM Walk Through…

- **Step 3:** what to output based on the cell state
  - **Step 3.1:** output gate – decides what parts of the cell state to output.
  - **Step 3.2:** apply tanh to cell state (to push the values to be in [-1, 1]), then scale by the output gate to release only the chosen parts.



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

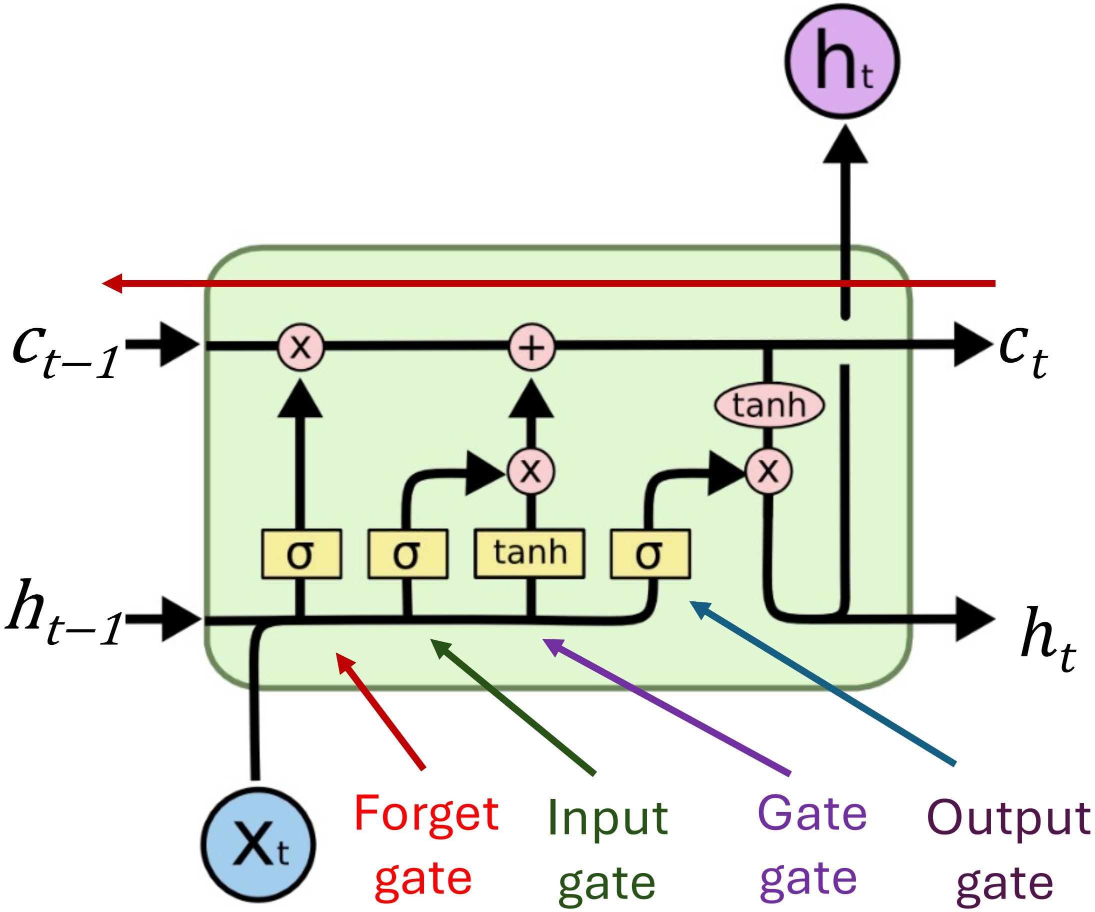
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

LSTM



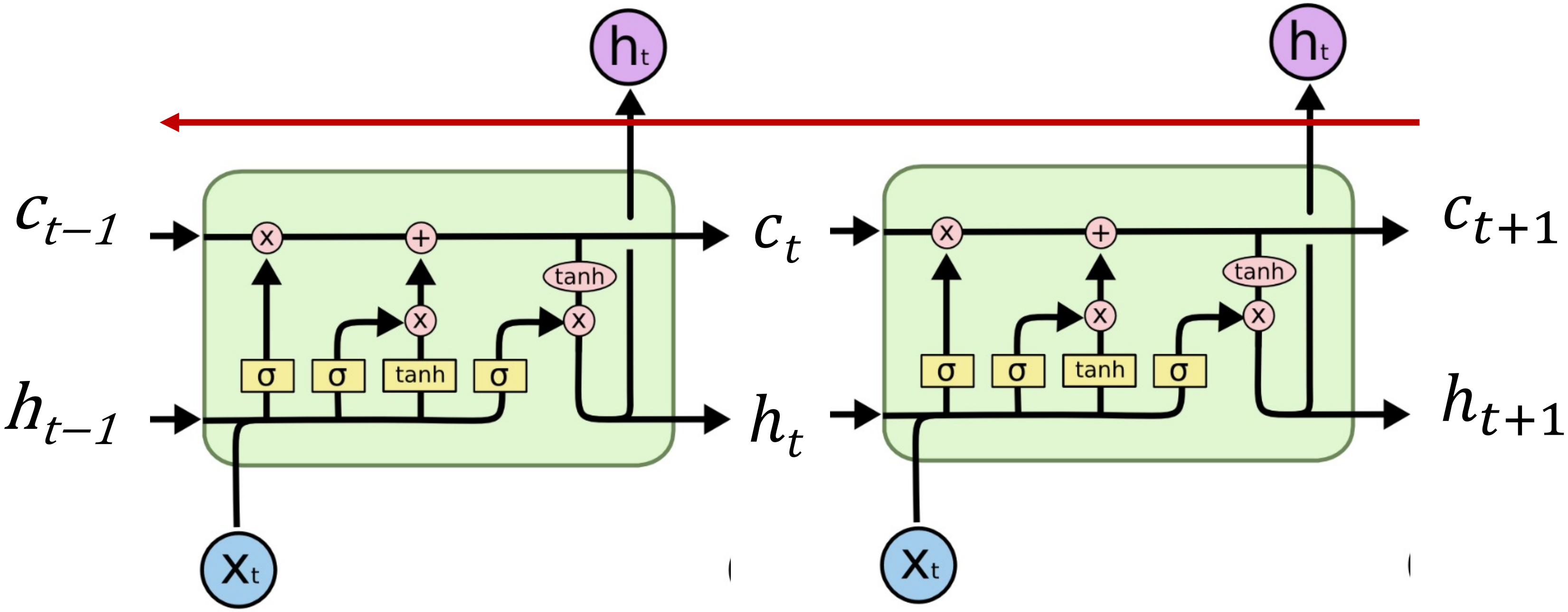
c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

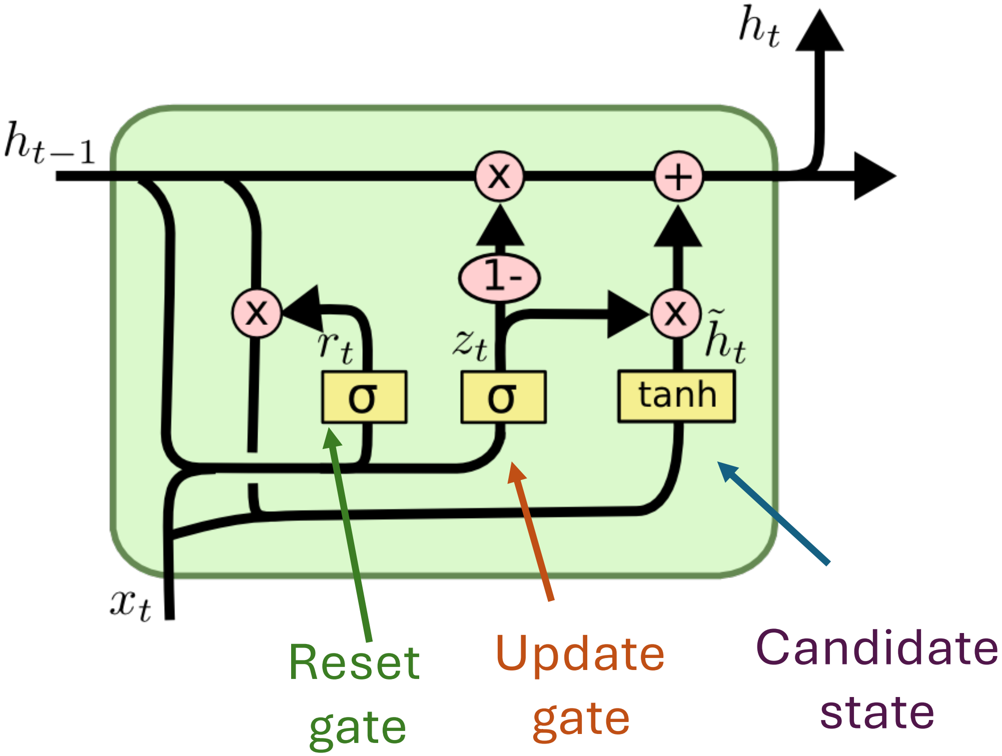
LSTM



Uninterrupted gradient flow!

LSTM Variant - Gated Recurrent Unit (GRU)

- Combines the **forget** and **input gates** into a single "**update gate**"
- Merges the **cell state** and **hidden state**



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$ Update gate

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$ Reset gate

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Cho et al., Learning phrase representations using RNN encoder-decoder for statistical machine translation, *arXiv:1406.1078*, 2014.

Interim Summary

- RNN is good at dealing with **sequence input and/or output**.

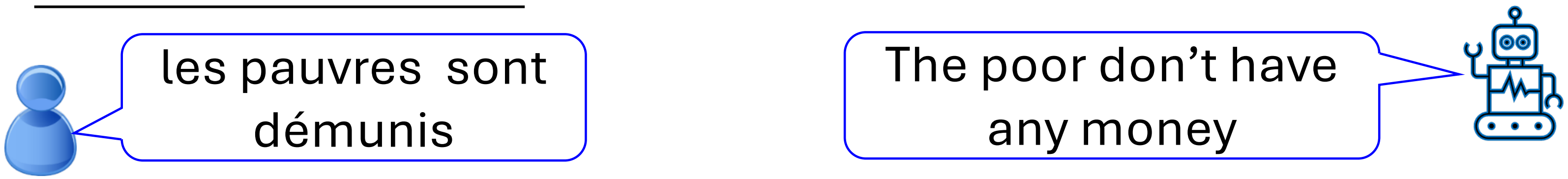
- Vanilla RNNs – suffer from **gradient vanishing/explosion** problem.
  - Exploding is controlled with **gradient clipping**.
  - Vanishing is controlled with additive interactions (**LSTM** or **GRU**).

- Next topics to cover:
  - Sequence-to-sequence learning
  - Attention mechanism

Sequence-to-Sequence (seq2seq) Learning

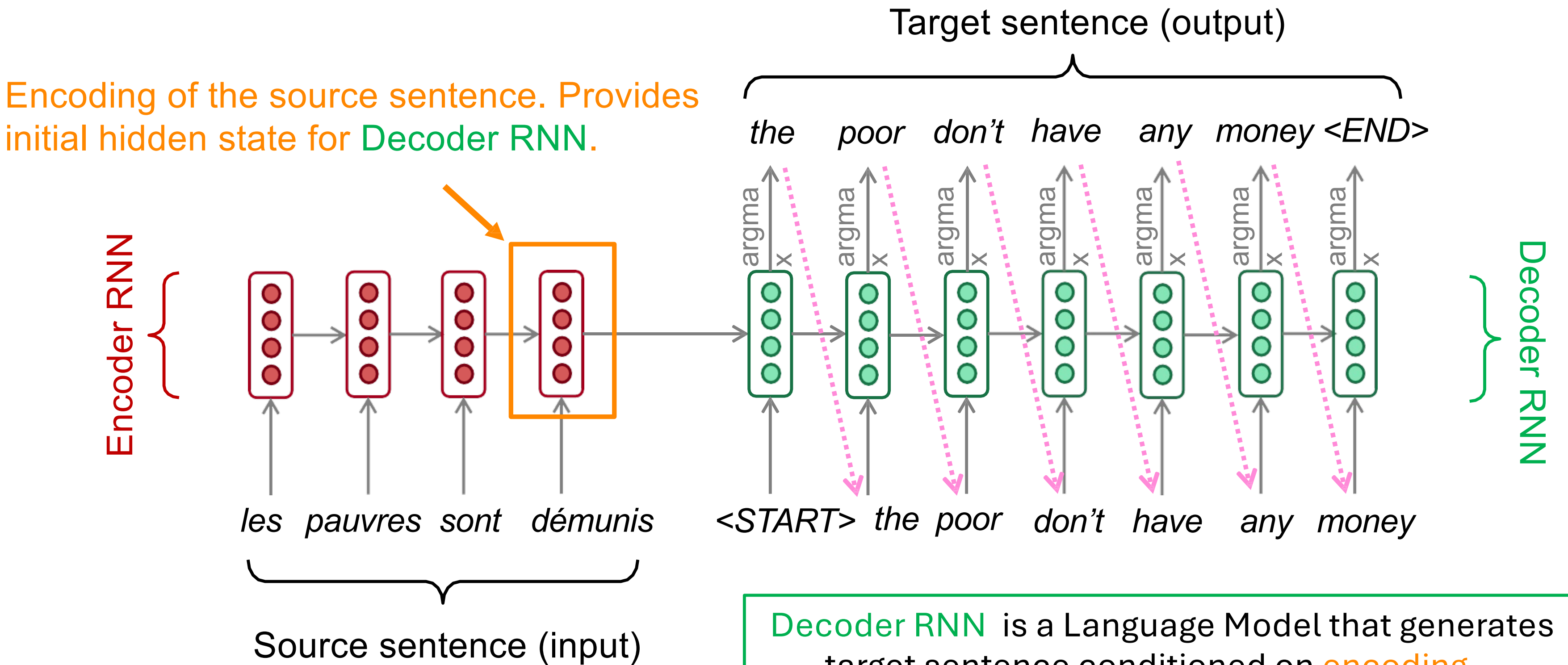
- Seq2seq learning typically involves two Recurrent Neural Networks (RNNs).
- The first RNN is an encoder which encodes the input sequence, and the second RNN is a decoder which generates the output sequence.

Machine Translation



Chatbot

Neural Machine Translation (NMT) – seq2seq Model

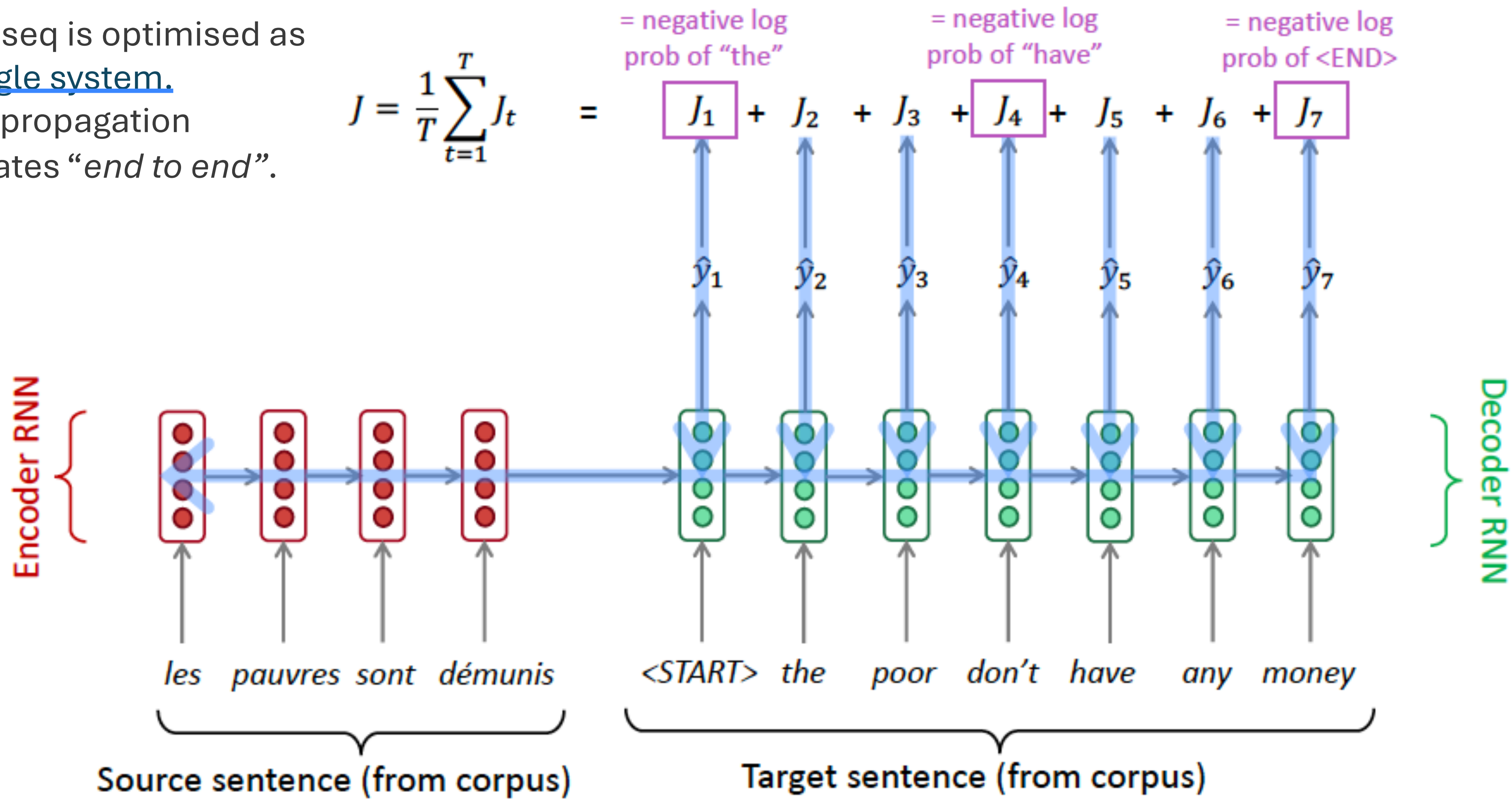


Decoder RNN is a Language Model that generates target sentence conditioned on encoding.

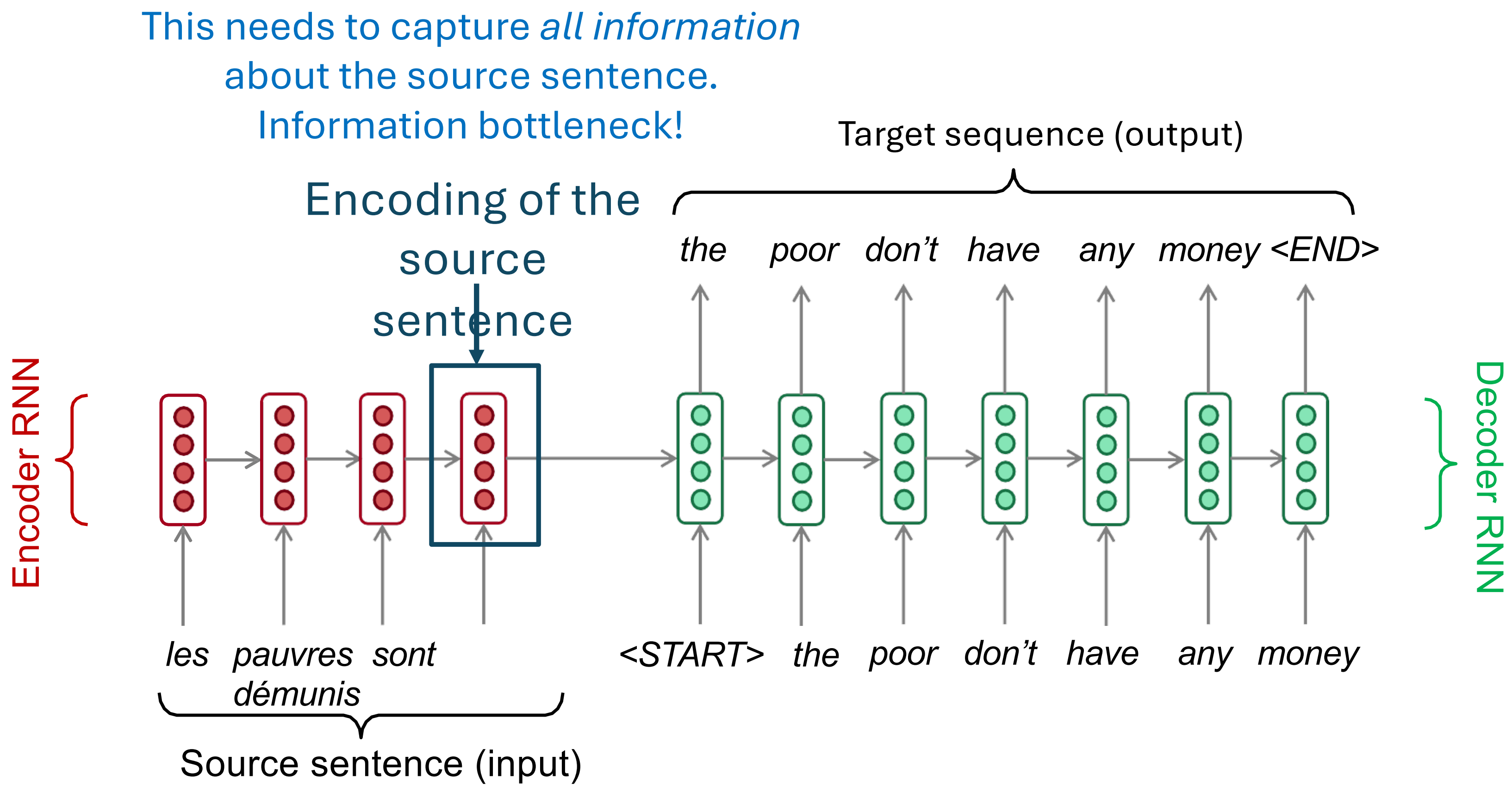
Encoder RNN produces an encoding of the source sentence.

Training a Neural Machine Translation system

Seq2seq is optimised as a single system. Backpropagation operates "*end to end*".



Sequence-to-sequence: the bottleneck problem



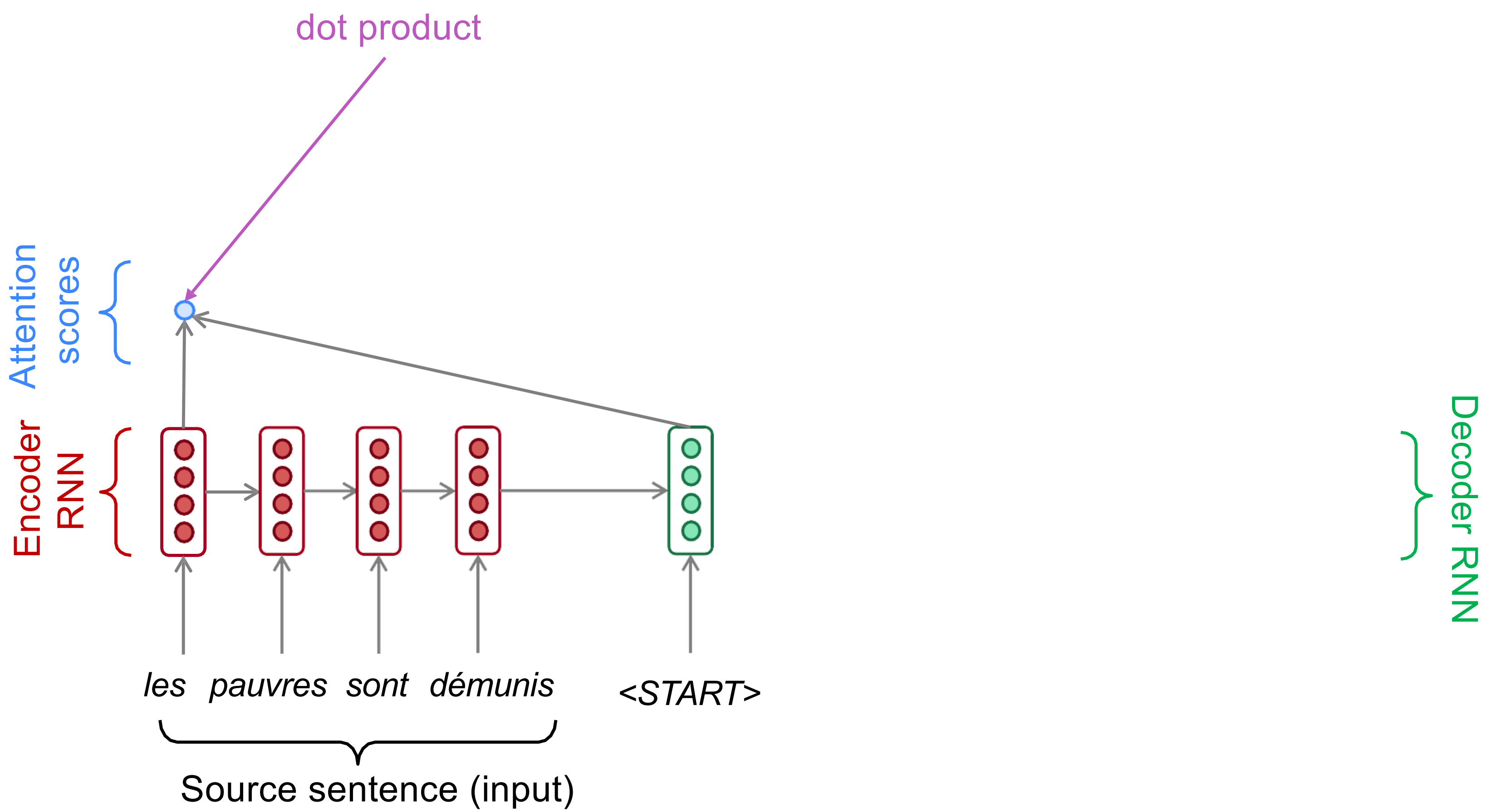
Problems with this architecture?

Attention

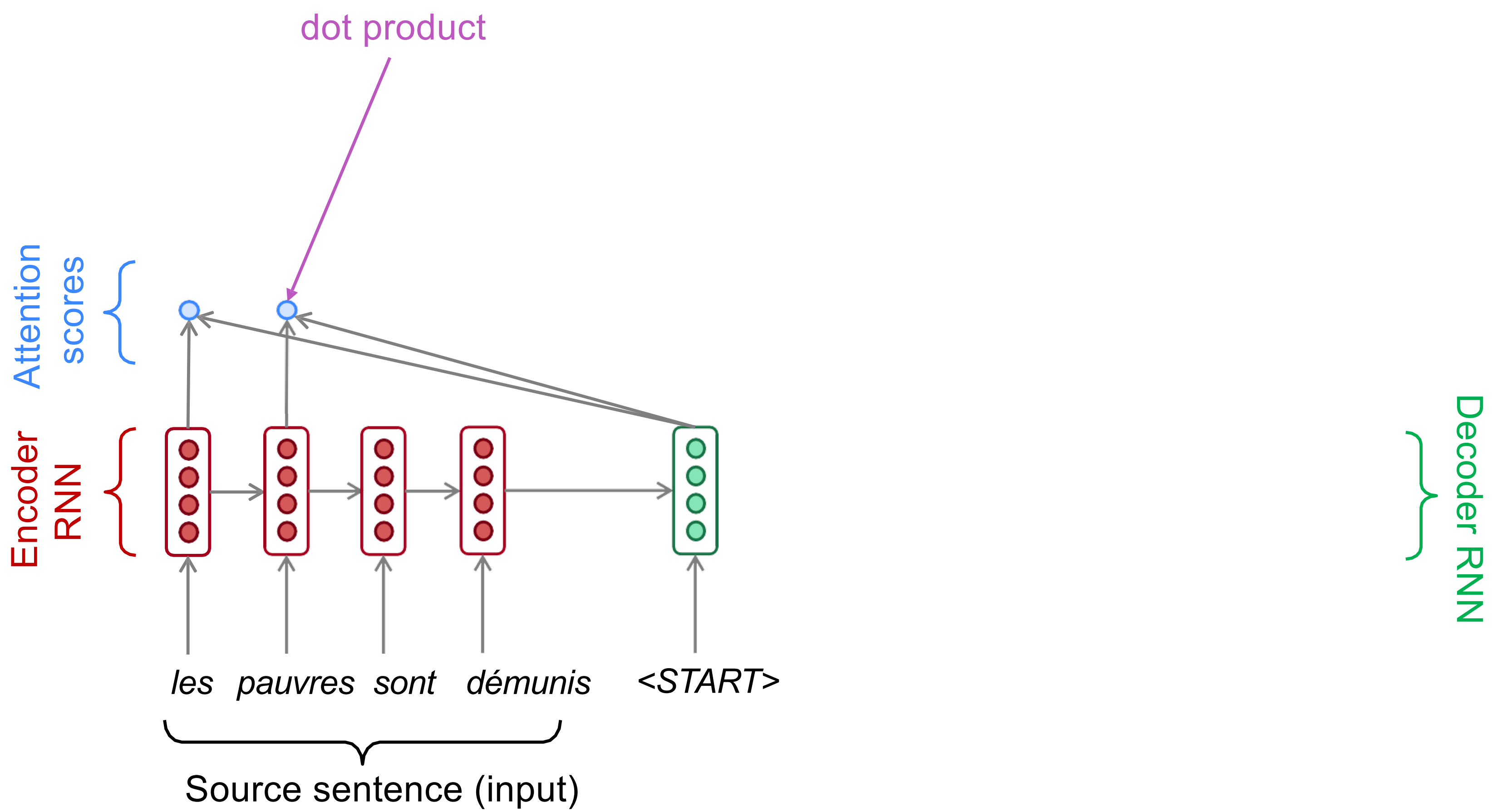
- **Attention** provides a solution to the bottleneck problem.

- Core idea: on each step of the decoder, *focus on a particular part* of the source sequence

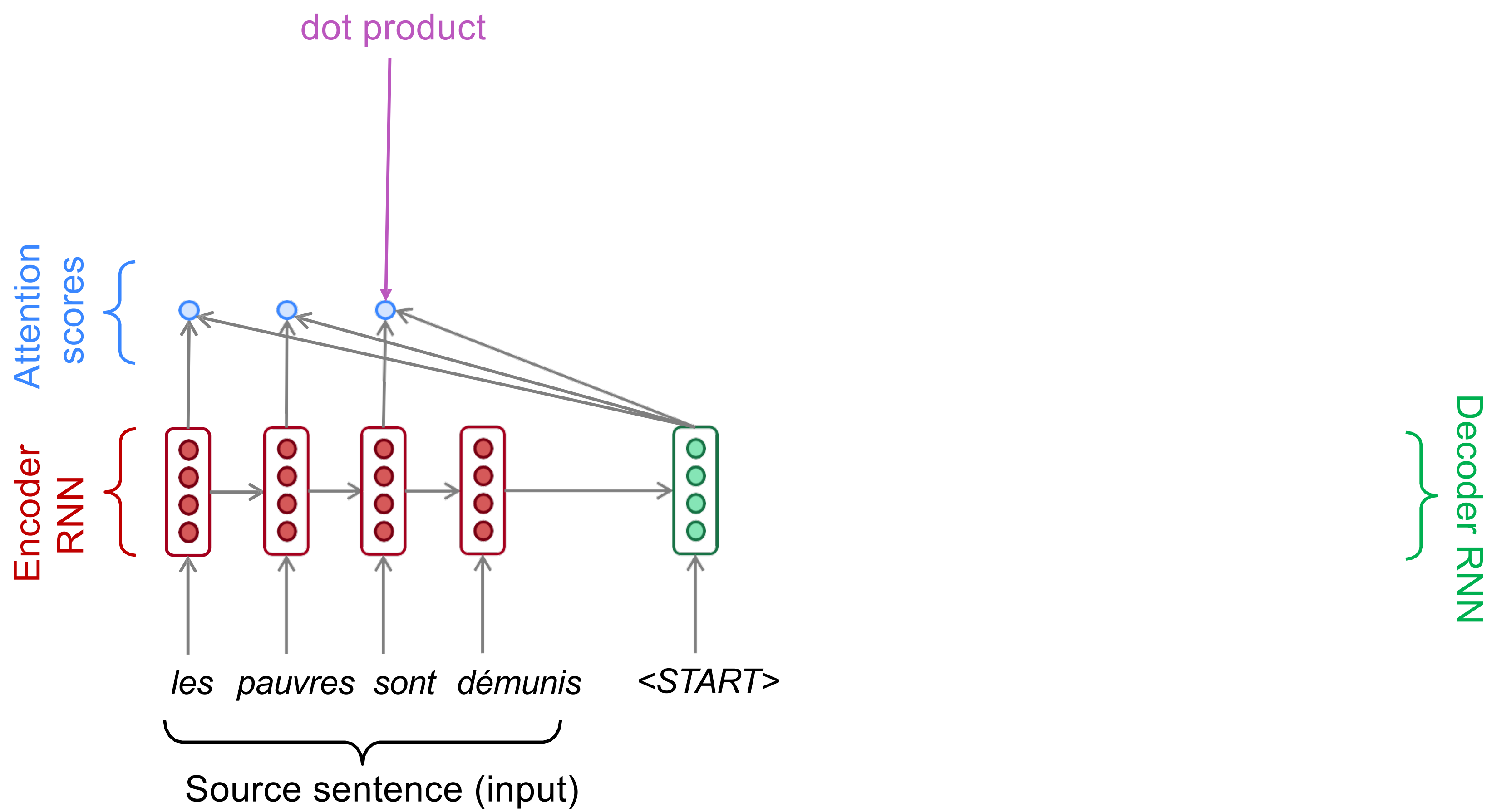
Sequence-to-sequence with attention



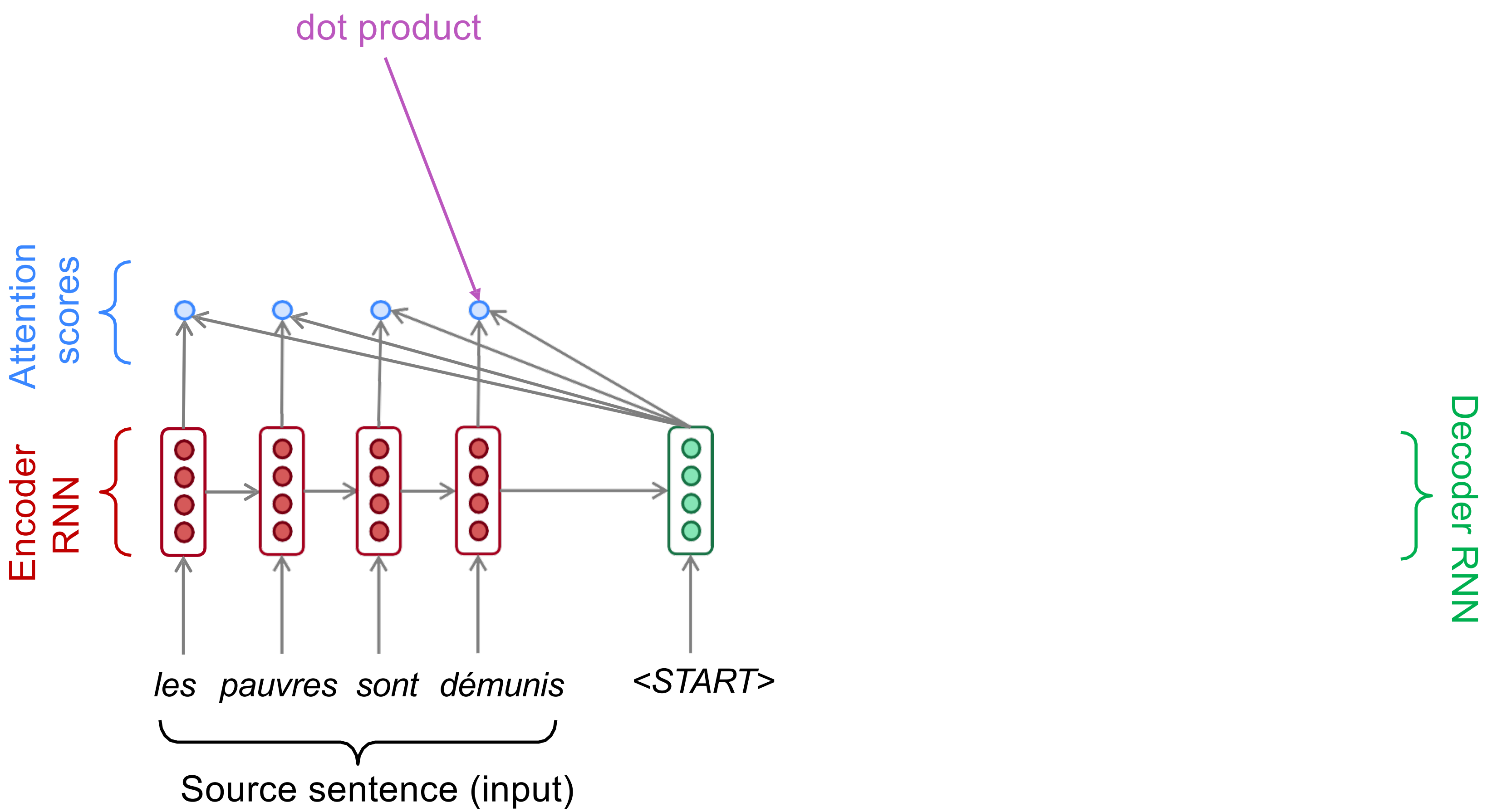
Sequence-to-sequence with attention



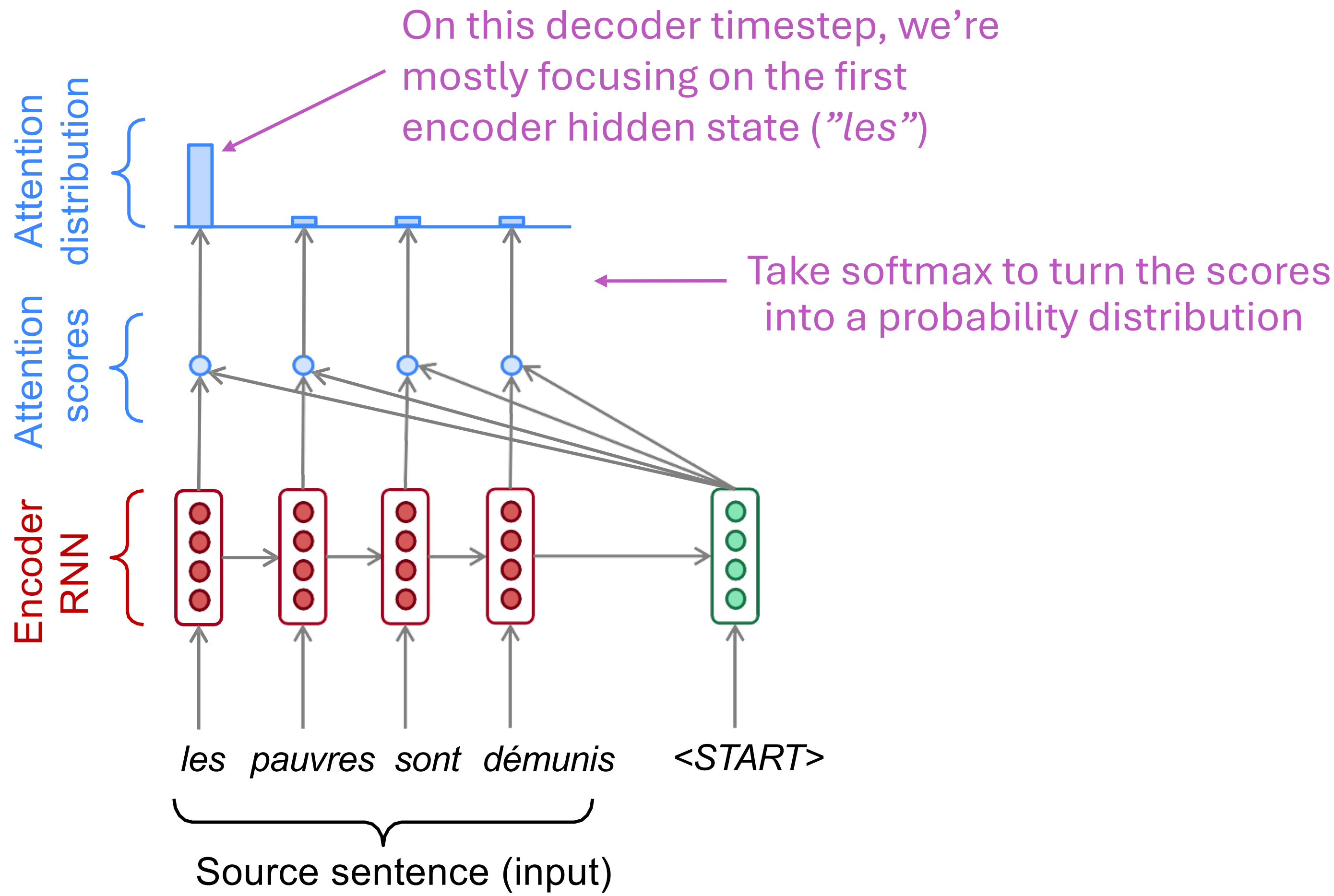
Sequence-to-sequence with attention



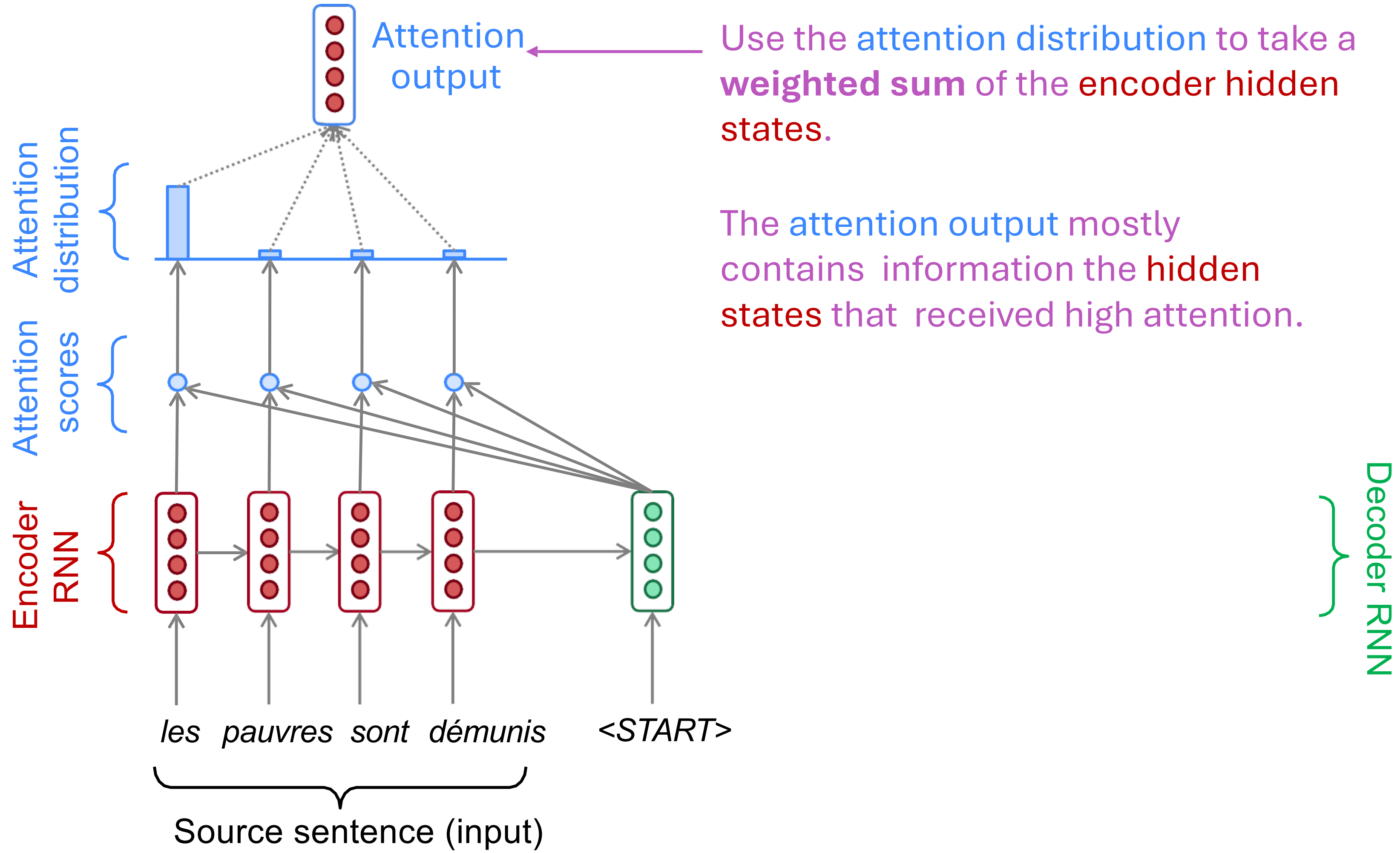
Sequence-to-sequence with attention



Sequence-to-sequence with attention



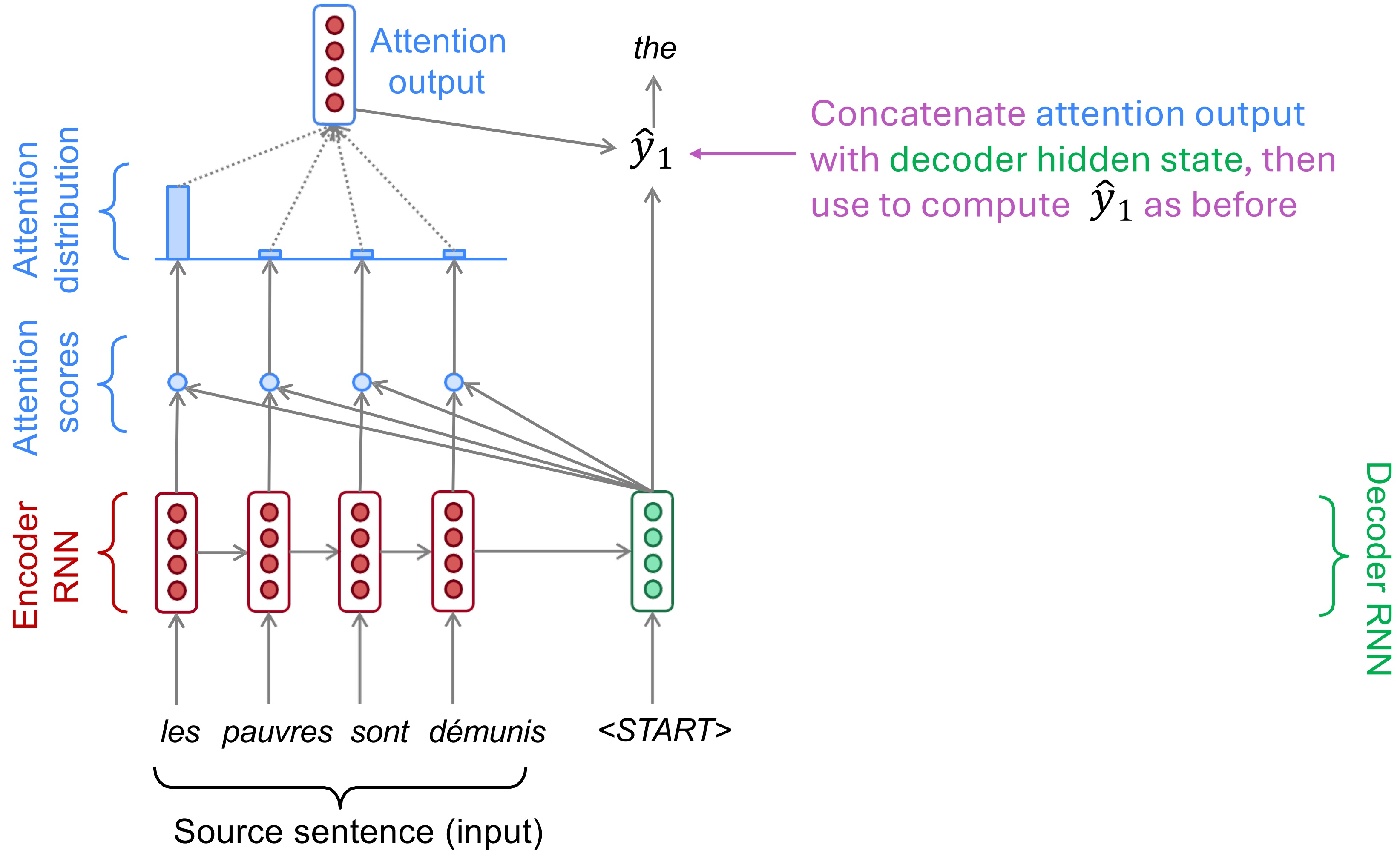
Sequence-to-sequence with attention



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

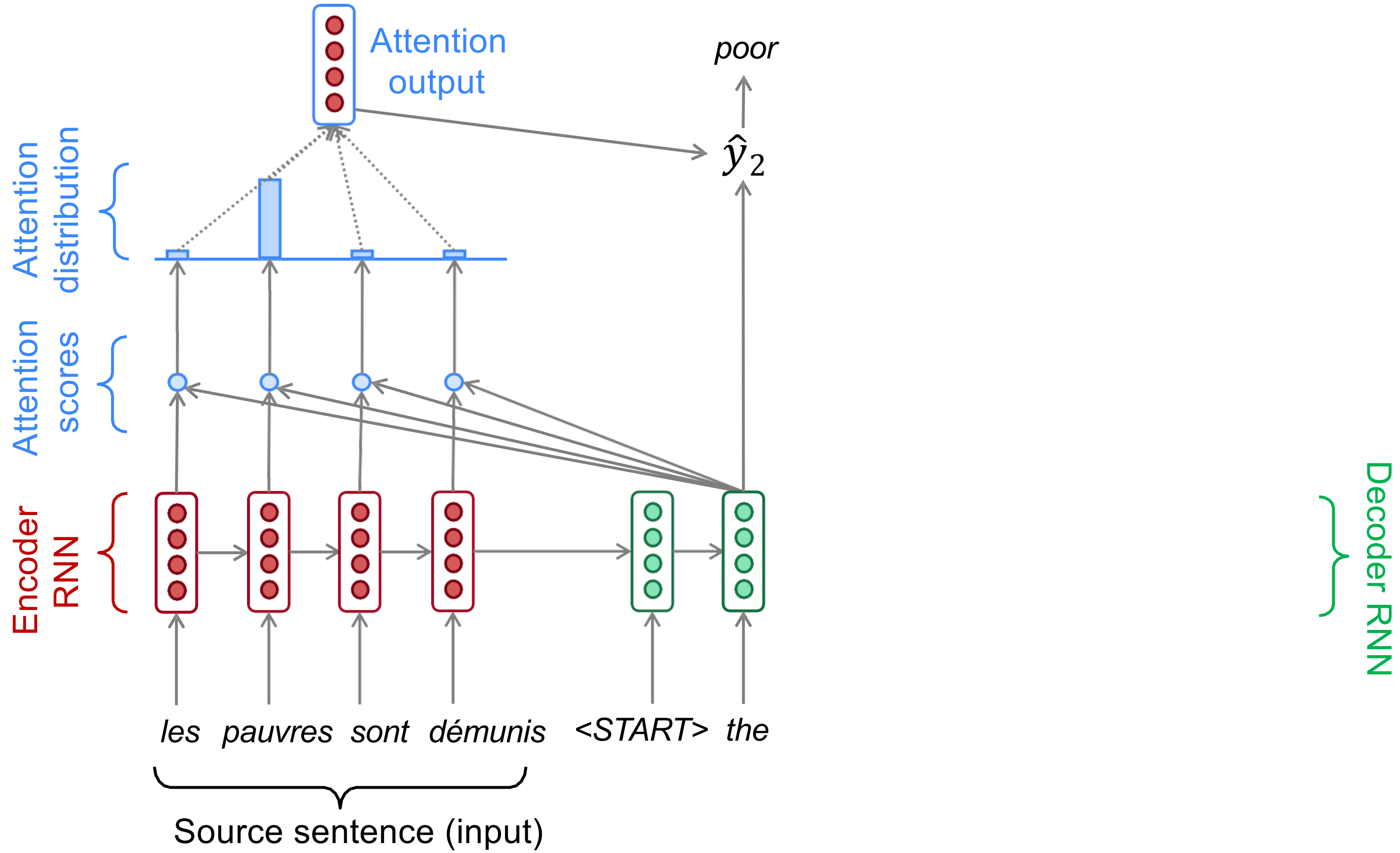
The attention output mostly contains information the hidden states that received high attention.

Sequence-to-sequence with attention

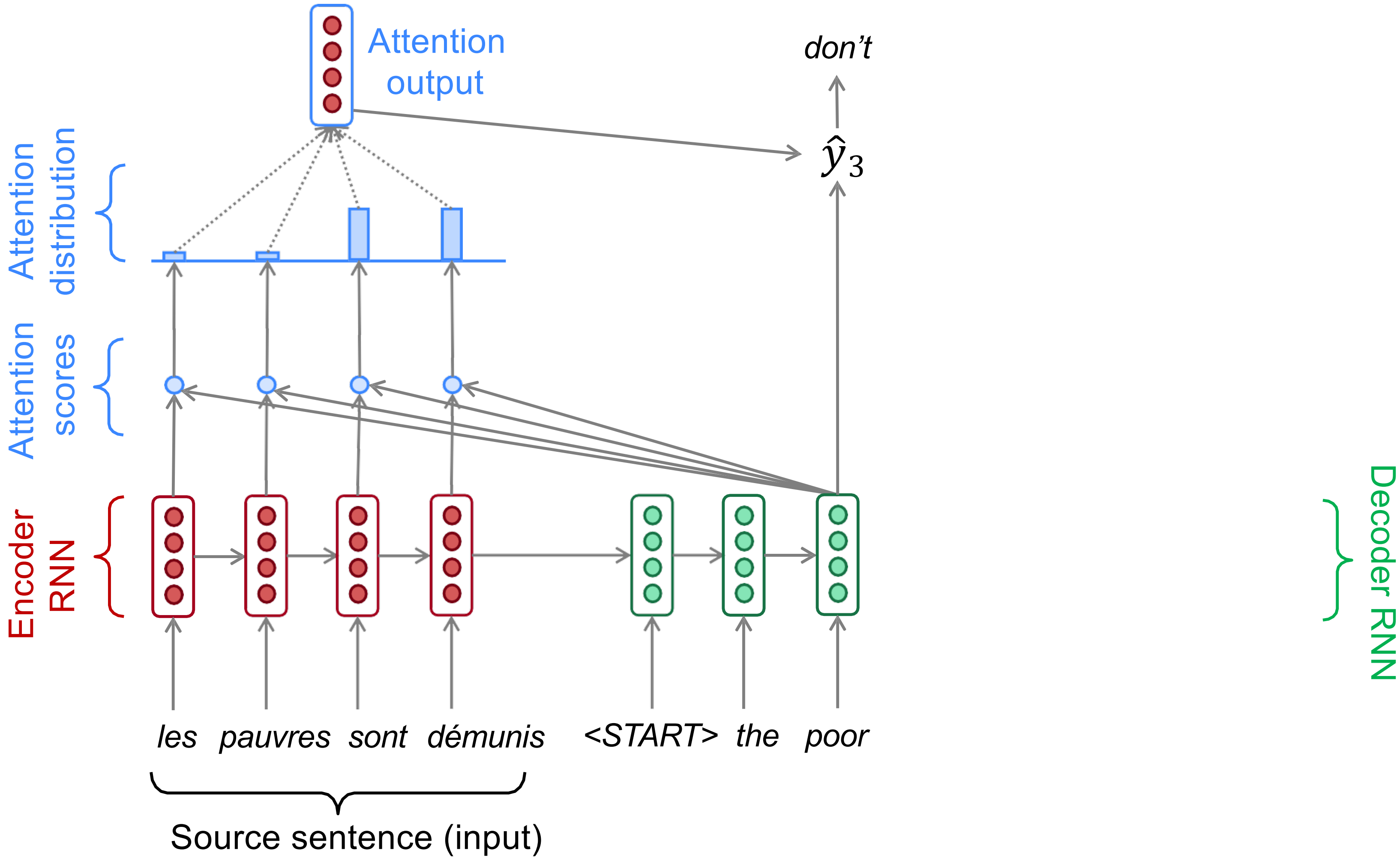


Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

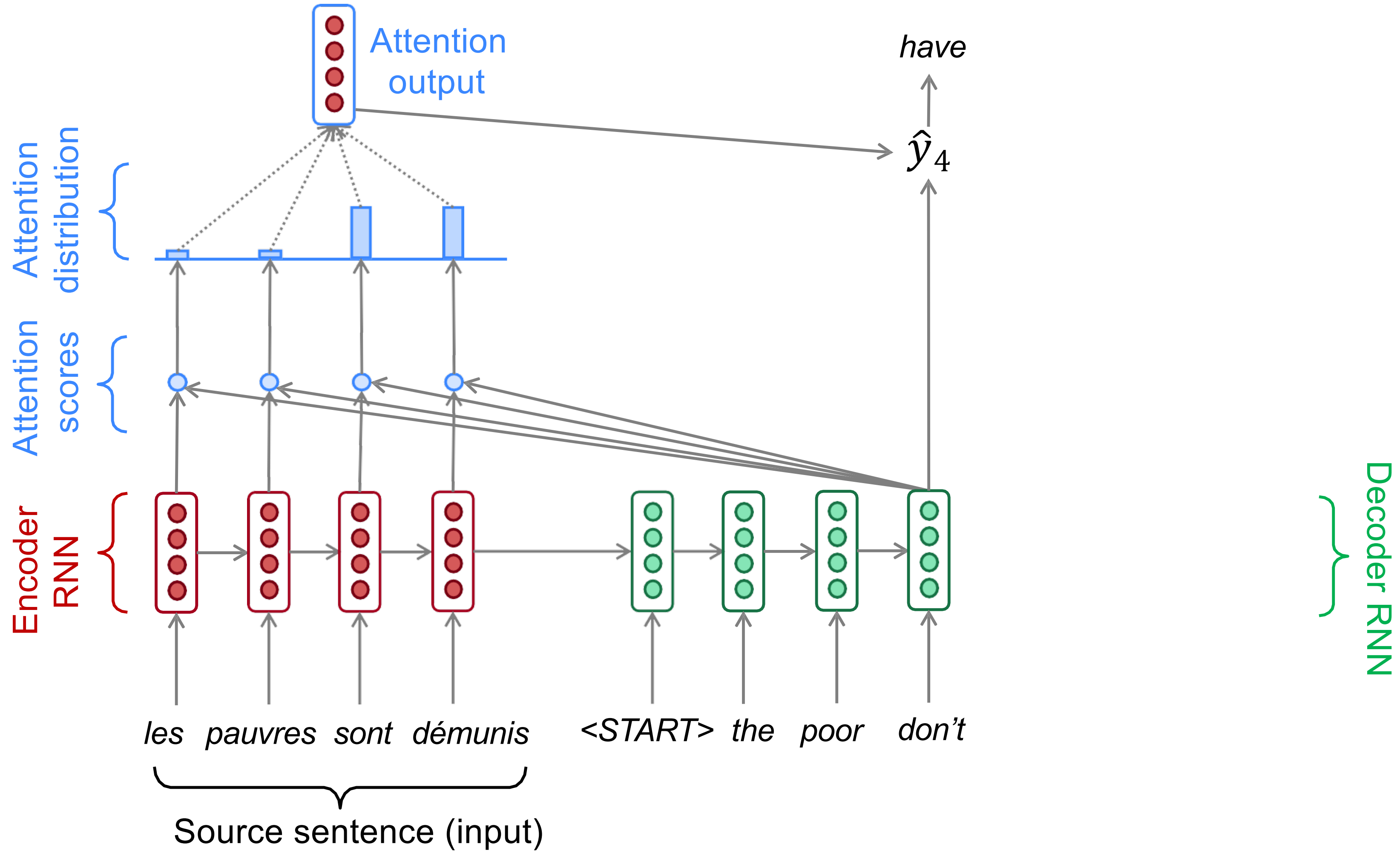
Sequence-to-sequence with attention



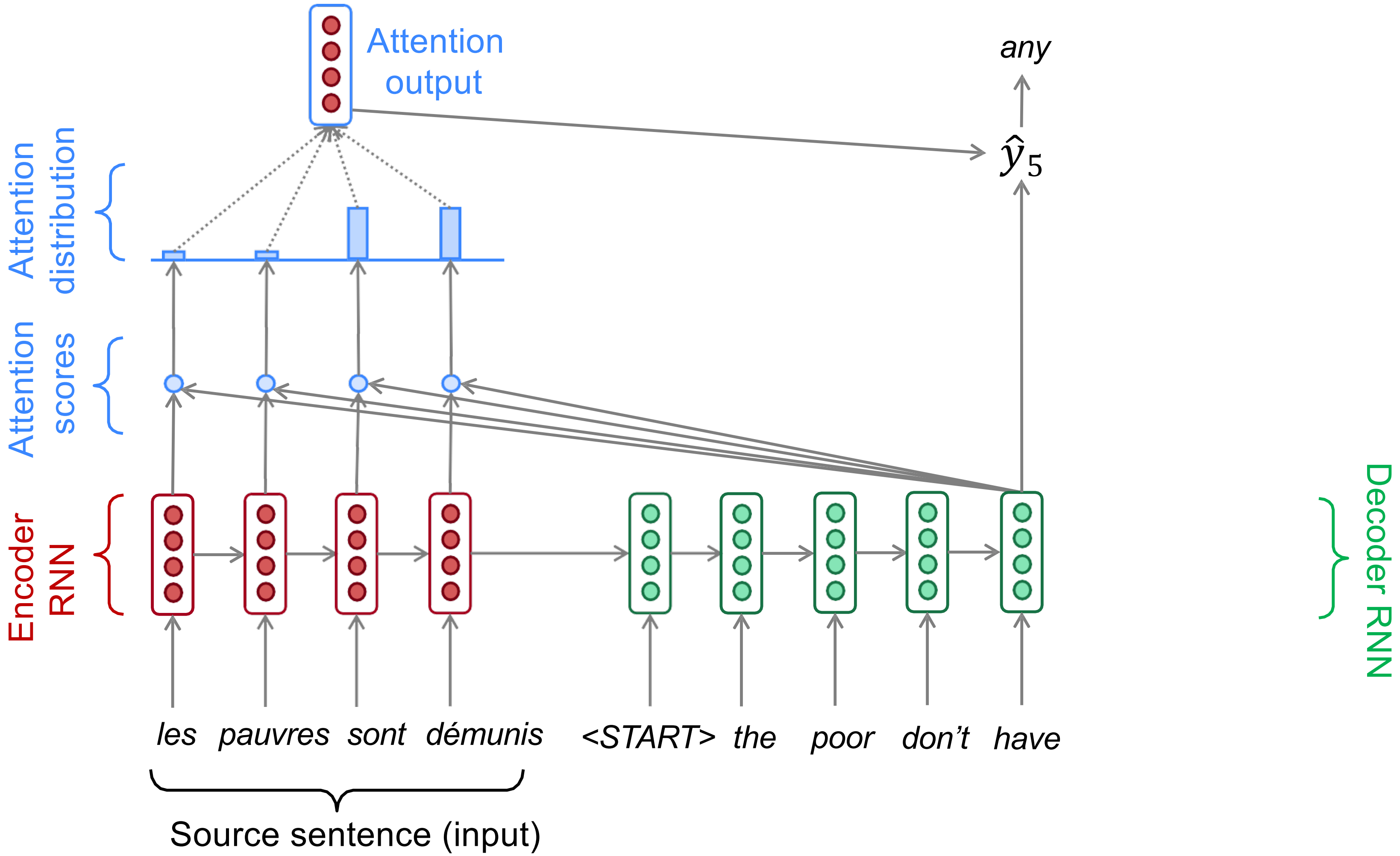
Sequence-to-sequence with attention



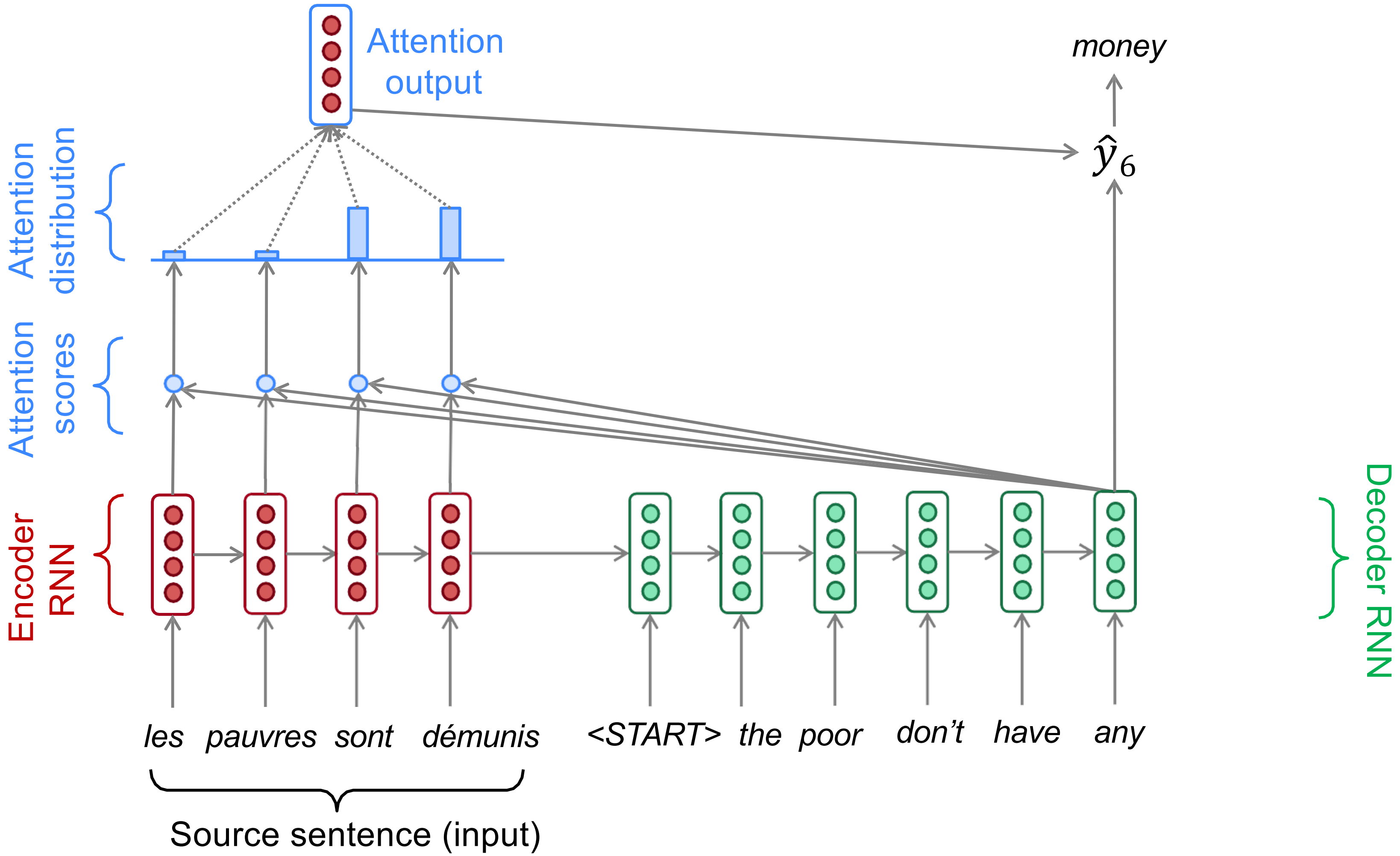
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Attention: in equations

- Encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- Decoder hidden state at timestep t: $s_t \in \mathbb{R}^h$

Step 1: Compute attention scores $\boldsymbol{e}^t$:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

Step2: Normalise into attention weights α_t

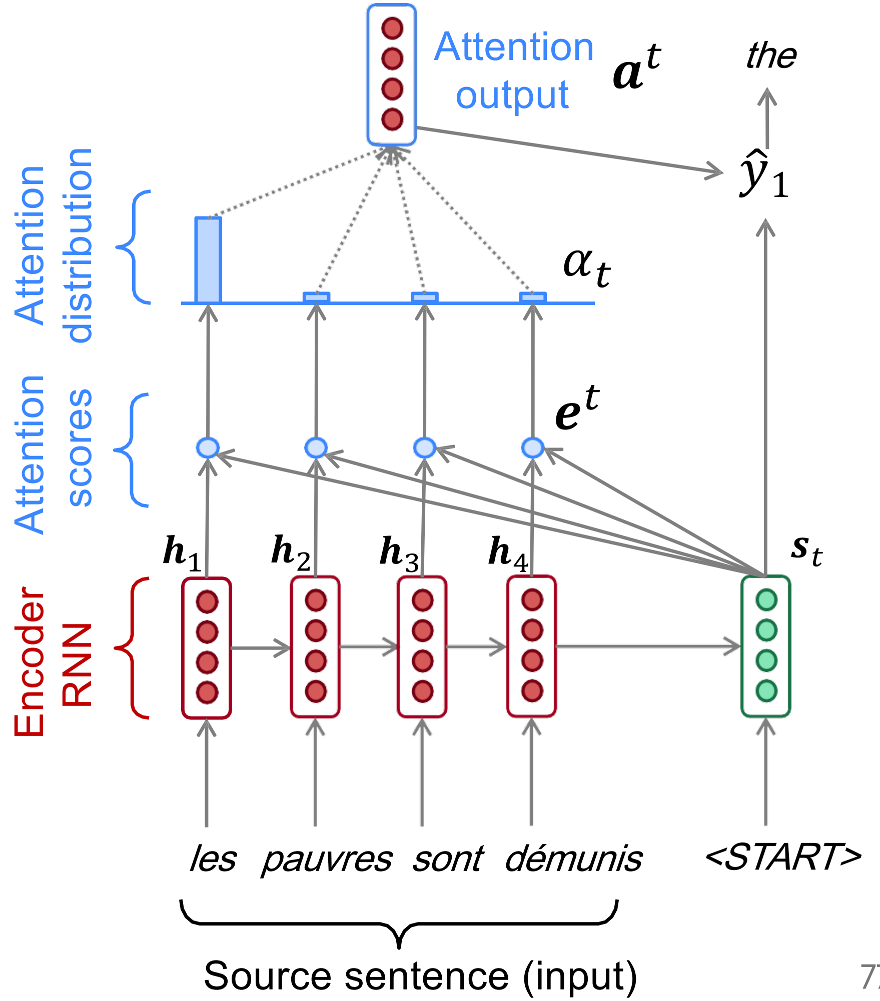
$$\alpha^t = \mathrm{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

Step3: Compute context (attention output) $\boldsymbol{a}^t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

Step 4: Combine with decoder state

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$



Why Attention Matters in seq2seq Learning

- Enables decoder to focus on **the most relevant source words**.

- Decoder can **directly access source states** instead of relying only on a single vector – solves the **bottleneck** problem.

- Provides shortcuts to distant source positions – **mitigates vanishing gradients**.

- Attention weights show which source words the decoder attends to.
  - **Implicit alignment** emerges naturally — no explicit alignment model needed.

General Definition of Attention

- **Definition:** Given a set of **value vectors** and a **query vector**, ***attention*** computes a **weighted sum of the values**, with weights determined by the query.

- **Intuition:**
  - Produces a **selective summary** of the values, guided by the query.
  - Provides **a fixed-size representation of** an arbitrary set of vectors (the **values**), conditioned on another vector (the query).

Mechanics of Attention

- We start with:
  - A set of **values** $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$
  - A **query** $\boldsymbol{s} \in \mathbb{R}^{d_2}$

Step 1: Compute **attention scores** (logits)

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

Step 2: Apply softmax → **attention distribution** (attention weights)

$$\alpha = \text{softmax}(\boldsymbol{e}) \in \mathbb{R}^N \qquad \boldsymbol{e} \in \mathbb{R}^N$$

Step 3: Take the **weighted sum of values** → attention output (**context vector**)

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1} \qquad \boldsymbol{a} \in \mathbb{R}^{d_1}$$

Attention Variants

- **Basic dot-product attention:** $$e_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$$
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier

- **Multiplicative attention:** $$e_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$$
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$ is a weight matrix

- **Additive attention:** $$e_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$$
  - Where $\boldsymbol{W_1} \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W_2} \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $\mathbf{v} \in \mathbb{R}^{d_3}$ is a weight vector

More information: http://ruder.io/deep-learning-nlp-best-practices/index.html#attention

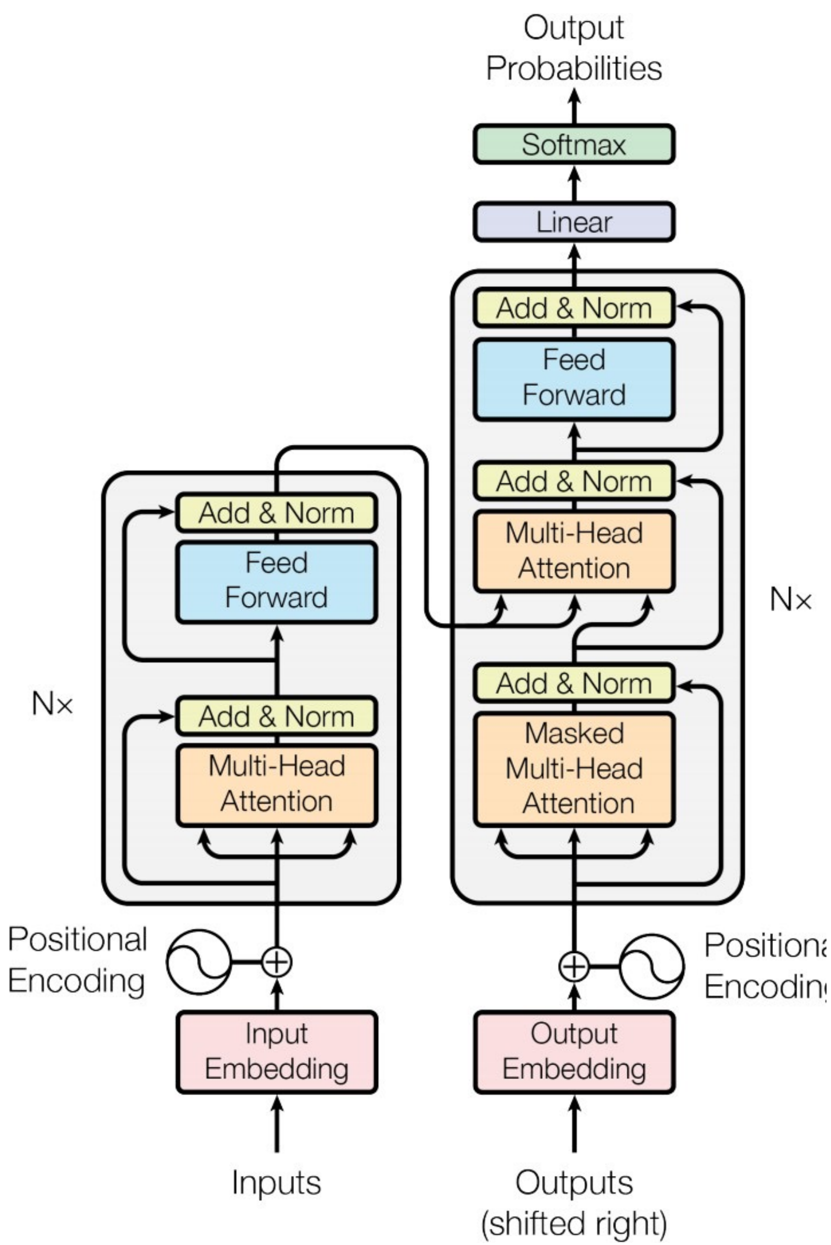
Interim Summary

- Sequence-to-Sequence (**seq2seq**) architecture
  - Two RNNs (encoder-decoder)
  - End-to-end training

- **Attention** mechanism
  - Only attend to a small part of the input sequence when generating the output at each time step
  - Attention variants

Part III: Transformer and LLMs

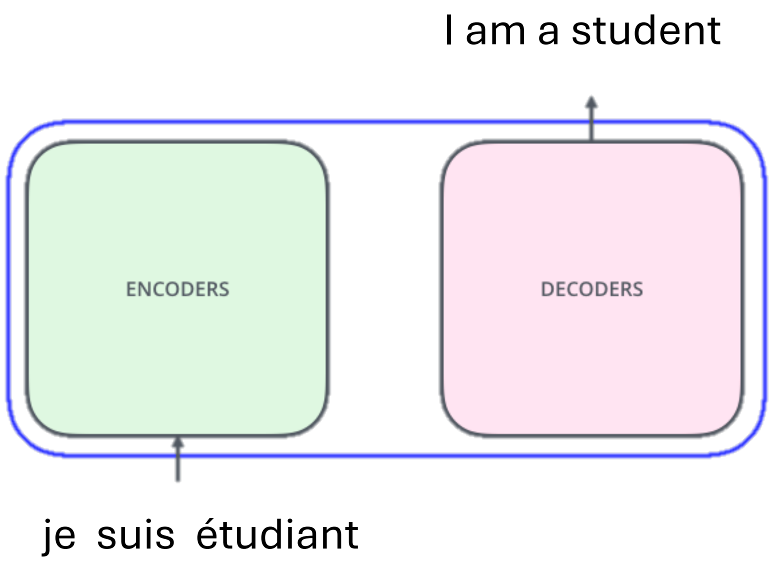
- The Transformer Architecture
- Language Models Built on Transformer
- LLM Training Paradigms
- LLM Evaluation

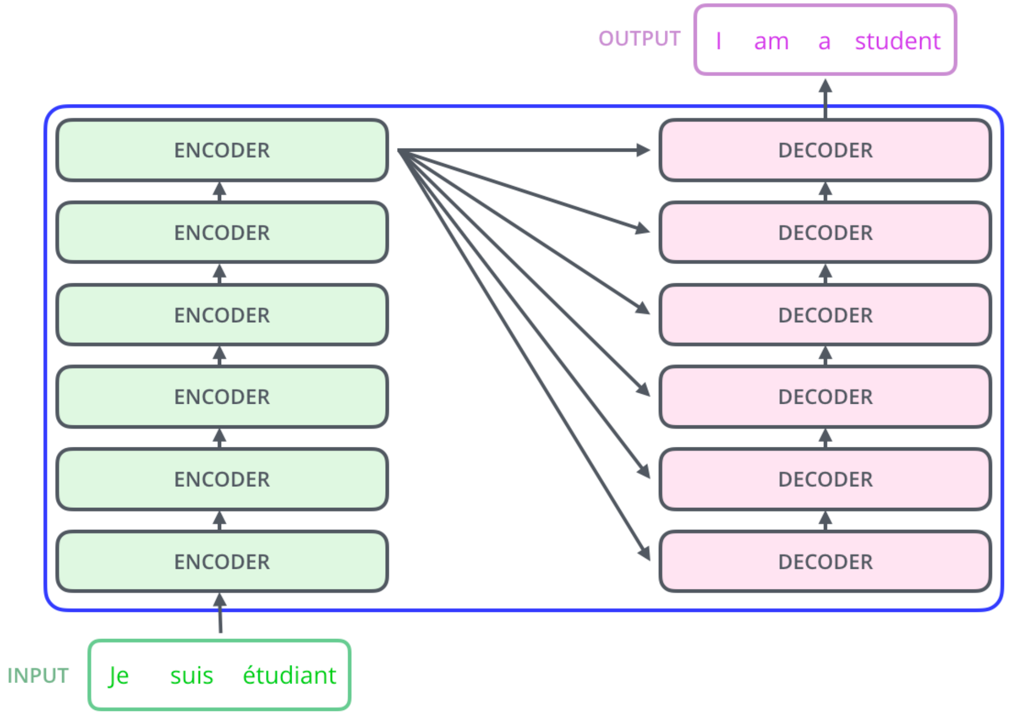


The Transformer Architecture

Vaswani et al., [Attention Is All You Need](). NeurIPS 2017.

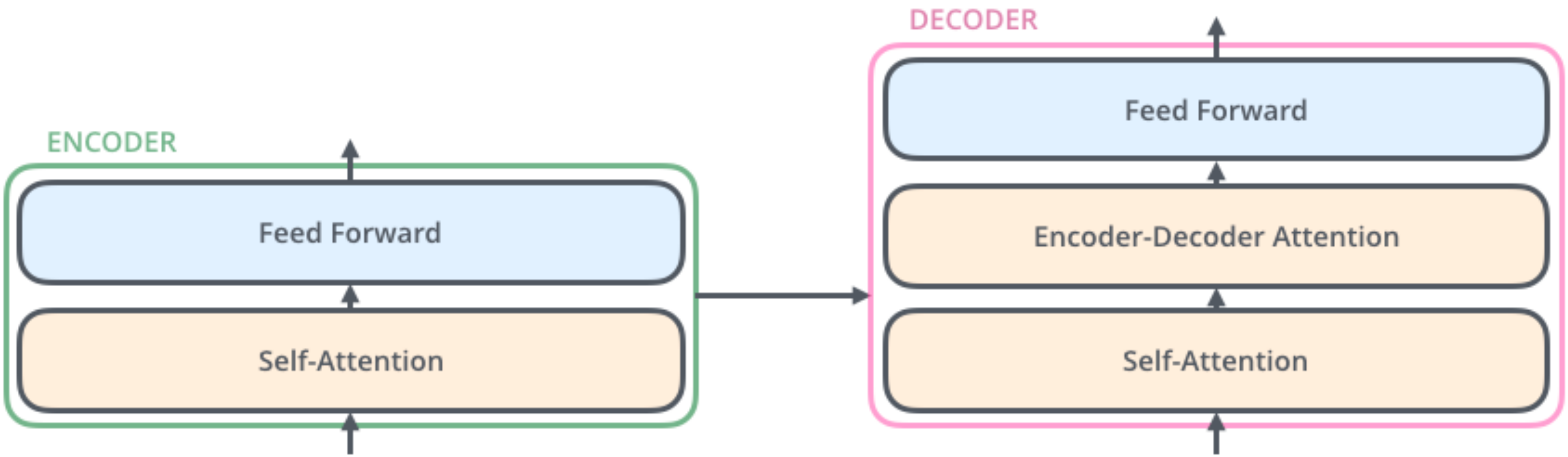
Transformer Overview





https://jalammar.github.io/illustrated-transformer/

Transformer Overview



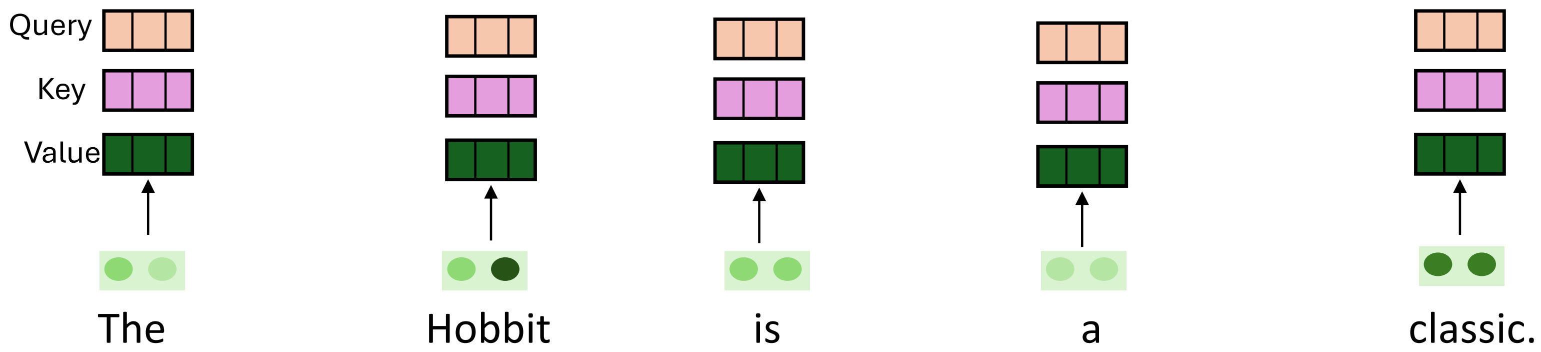
https://jalammar.github.io/illustrated-transformer/

Transformer Basics – Self-Attention Layer

- **Step 1:** create three vectors (Query q, Key k, Value v) from each of the encoder's input vectors x

$$q = W^Q x \qquad k = W^K x \qquad v = W^V x$$

Where $q \in \mathbb{R}^{d_k}$, $k \in \mathbb{R}^{d_k}$, $v \in \mathbb{R}^{d_v}$



Transformer Basics – Self-Attention Layer

- **Step 2:** generate output based on the **dot-product attention**

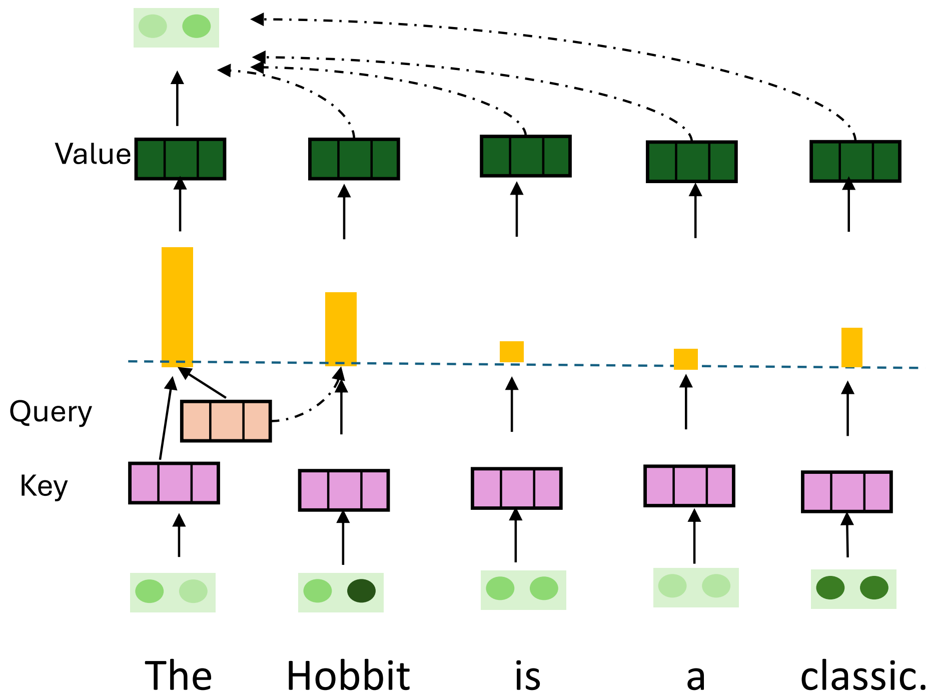
Inputs: a query q and a set of key-value (k-v) pairs in other word positions

Output: weighted sum of values, where weight of each value is computed by an **inner product** of the query and the corresponding key

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

If we have multiple queries q, we stack them in a matrix Q

$$A(Q, K, V) = \mathrm{softmax}(QK^T)V$$

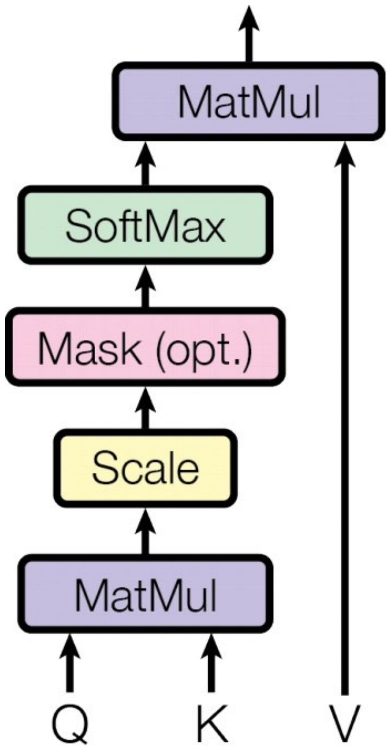


Scaled Dot-Product Attention

- **Problem**: As d_k (dimension of q and k) increases, the variance of $q^{\mathrm{T}}k$ increases. This causes
  - some **values inside the softmax** become **large**, leading to the **softmax** becoming very **peaked**,
  - hence its **gradient** becomes **smaller**.

- **Solution**: Scale by length of query/key vectors:

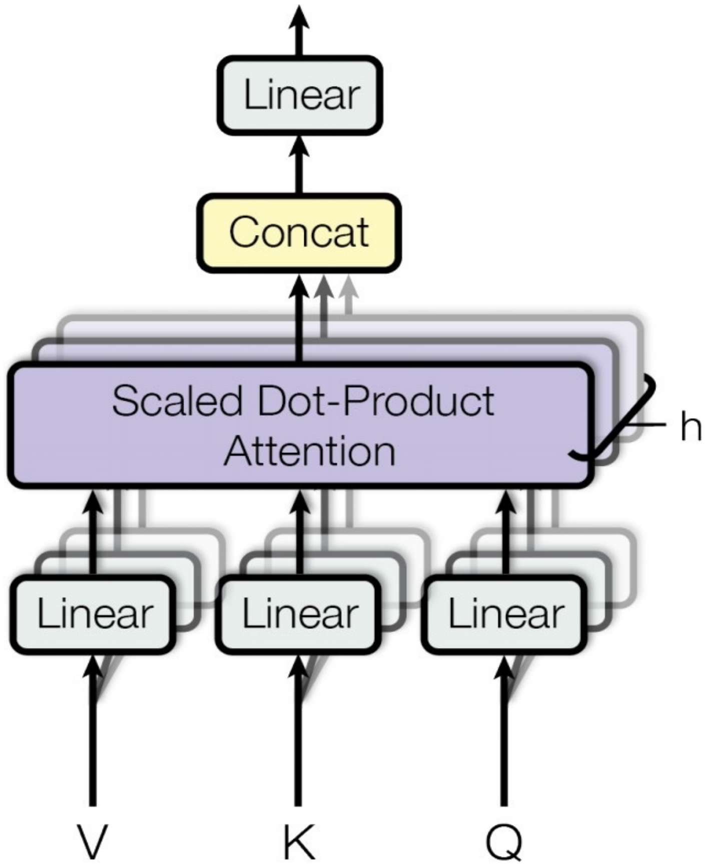
$$A(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



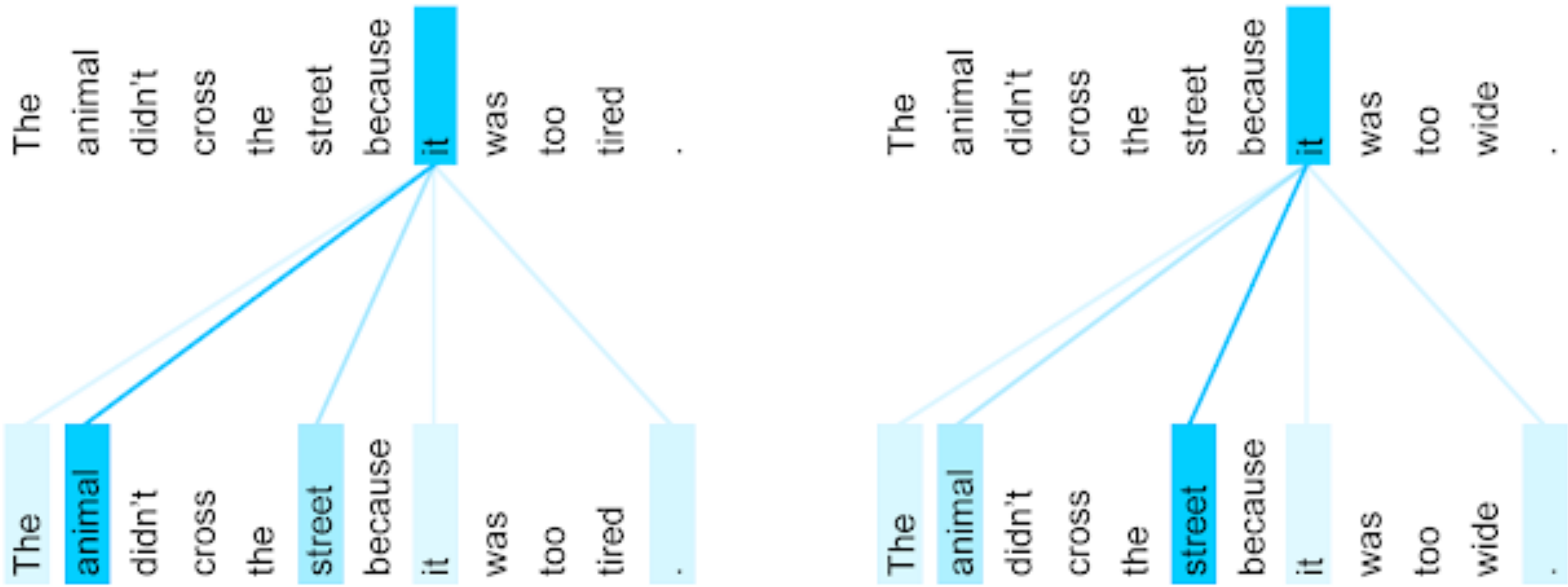
Self-attention and Multi-head attention

- **Problem**: Only one way for words to interact with others
- **Solution**: Multi-head attention
  - First map Q, K, V into h many lower-dimensional spaces via W matrices;
  - Then apply attention, then concatenate outputs and pipe through linear layer.

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



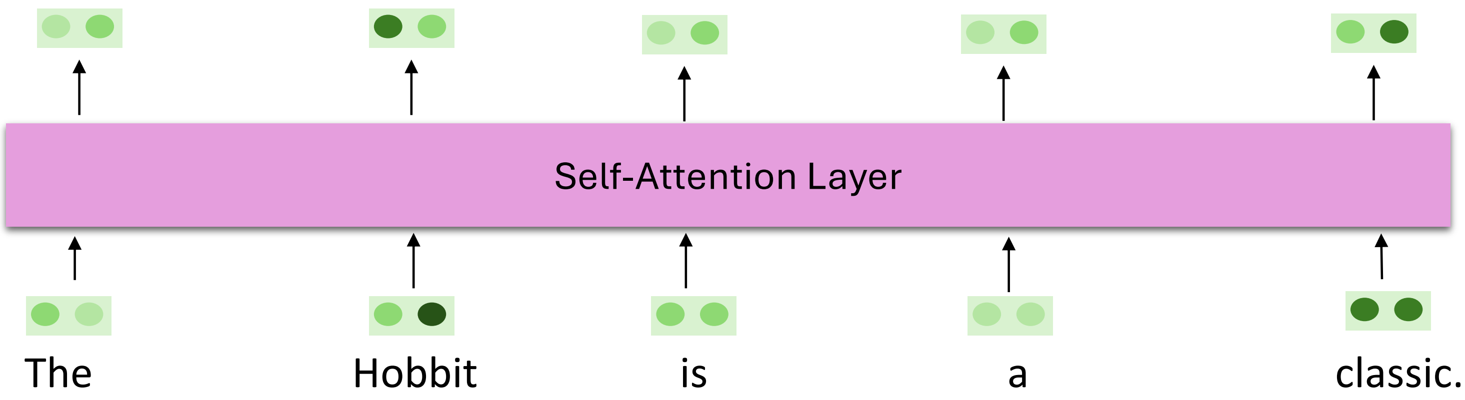
Attention visualisation: Implicit anaphora resolution



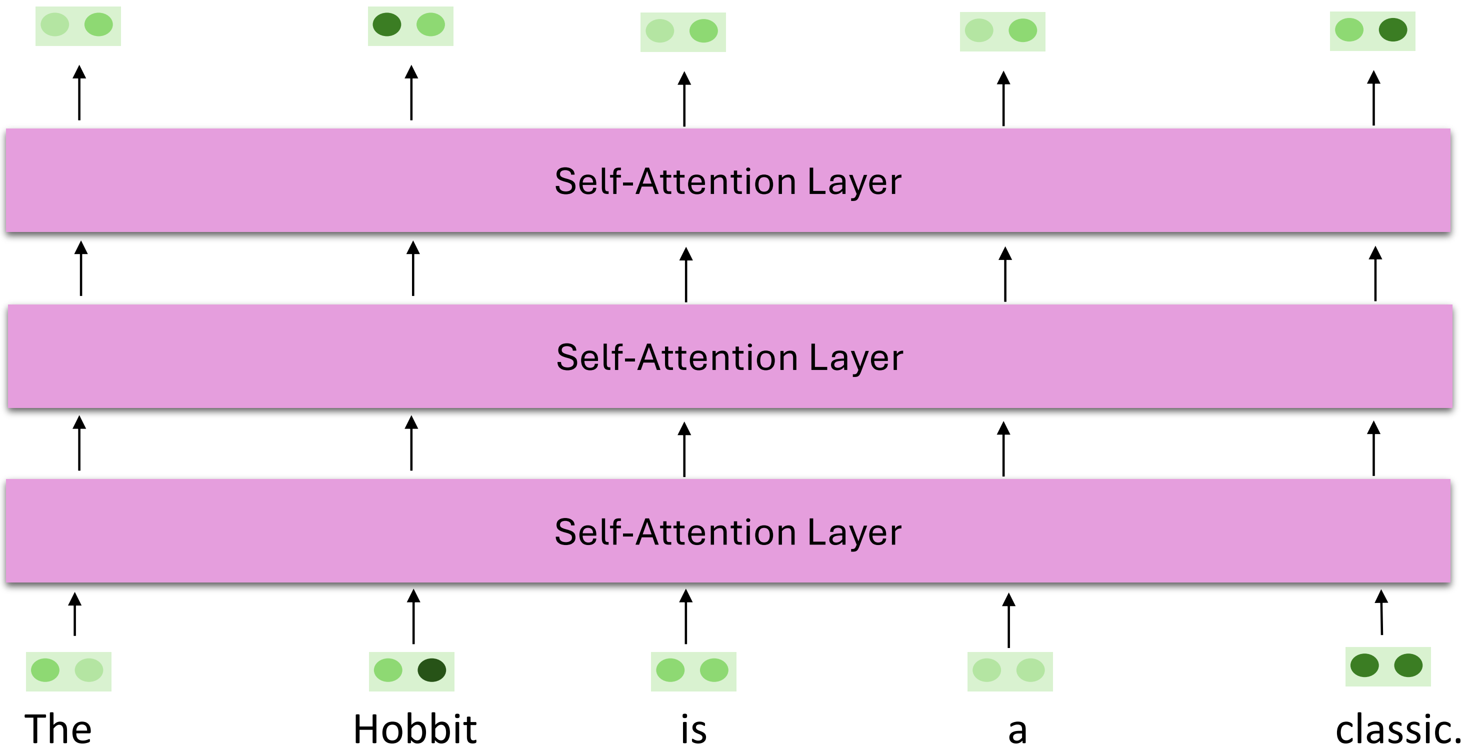
The encoder self-attention distribution for the word "it" from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Transformer Basics – Self-Attention Layer

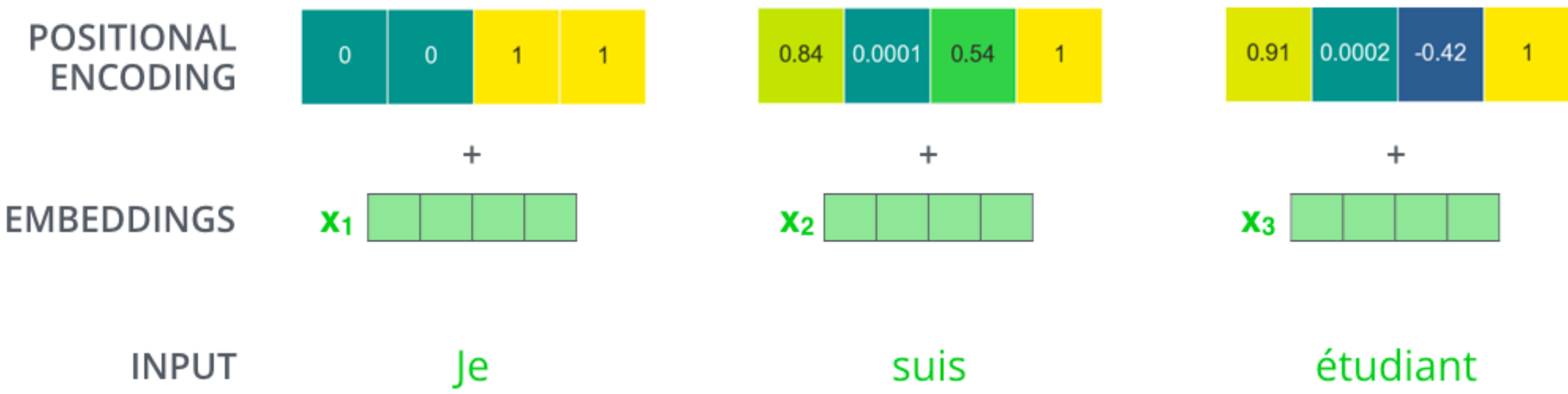


Transformer Basics – Self-Attention Layer



Encoder Input

- Actual word representations are byte-pair encodings
- Also added is a positional encoding so same words at different locations have different overall representations:



https://jalammar.github.io/illustrated-transformer/

Encoder Input…

- **Byte-pair encoding**
  - A simple form of data compression in which *the most common pair of consecutive bytes of data* is **replaced wit**h *a byte that does not occur within that data*.
  - E.g., to encode the data "aaabdaaabac"
    - The byte pair "aa" occurs most often
    - We replace it by a byte that is not used in the data, say, "Z".
    - Now the data become: "ZabdZabac" where Z=aa

- **Positional encoding**

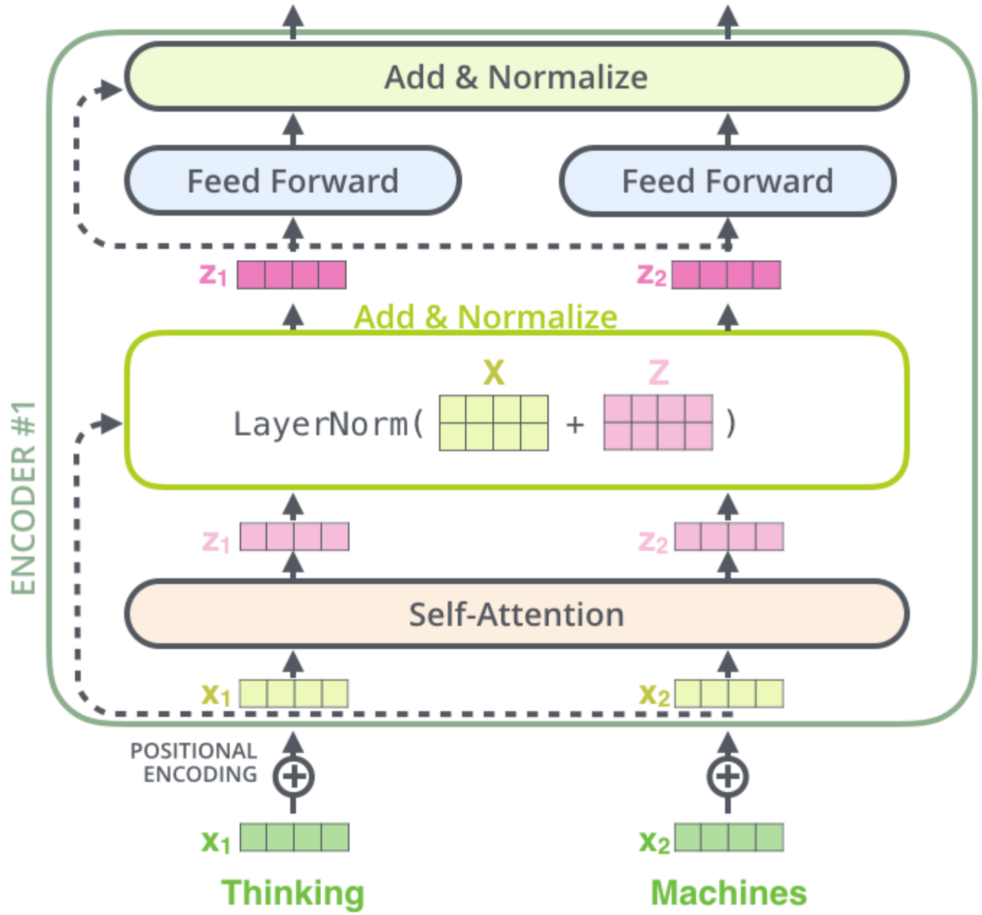
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

  - where *pos* is the position and i is the dimension, d_{model} is the dimension of the word embedding.

Complete the Transformer Block

- Each block has two "sublayers"
  - Multi-head attention
  - 2-layer feed-forward neural network (with Relu)

- Each of these two steps also has:
  - Residual (short-circuit) connection

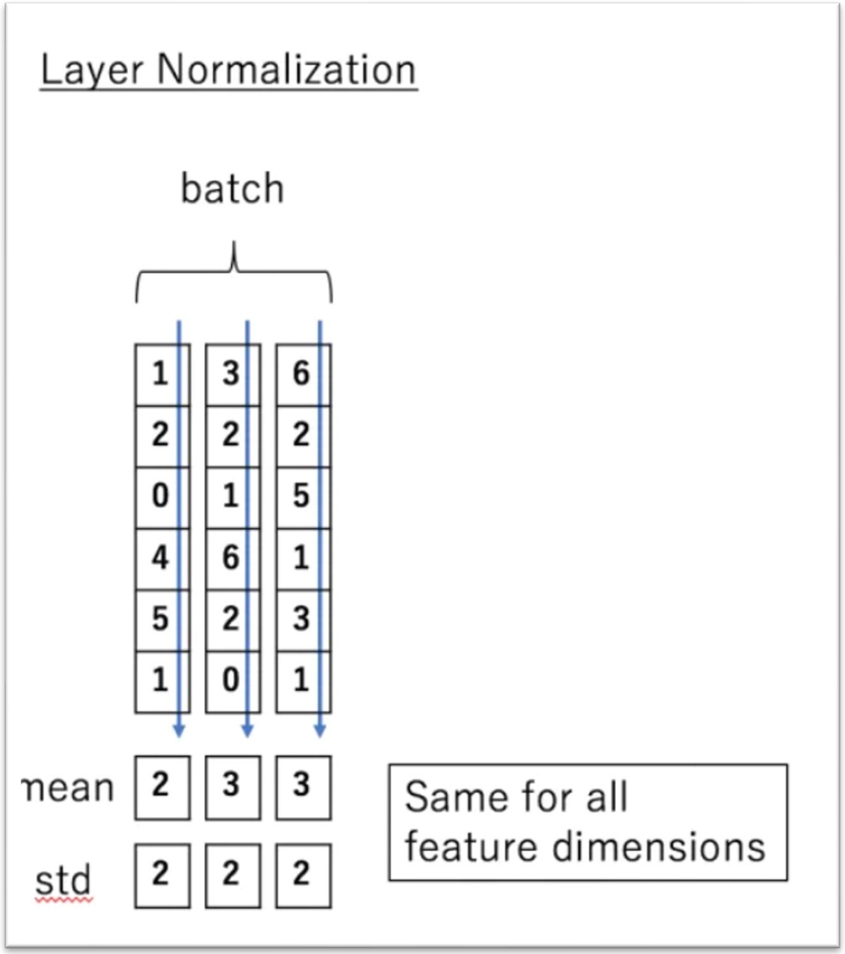


https://jalammar.github.io/illustrated-transformer/

Complete the Transformer Block

- Each block has two "sublayers"
  - Multi-head attention
  - 2-layer feed-forward neural network (with Relu)

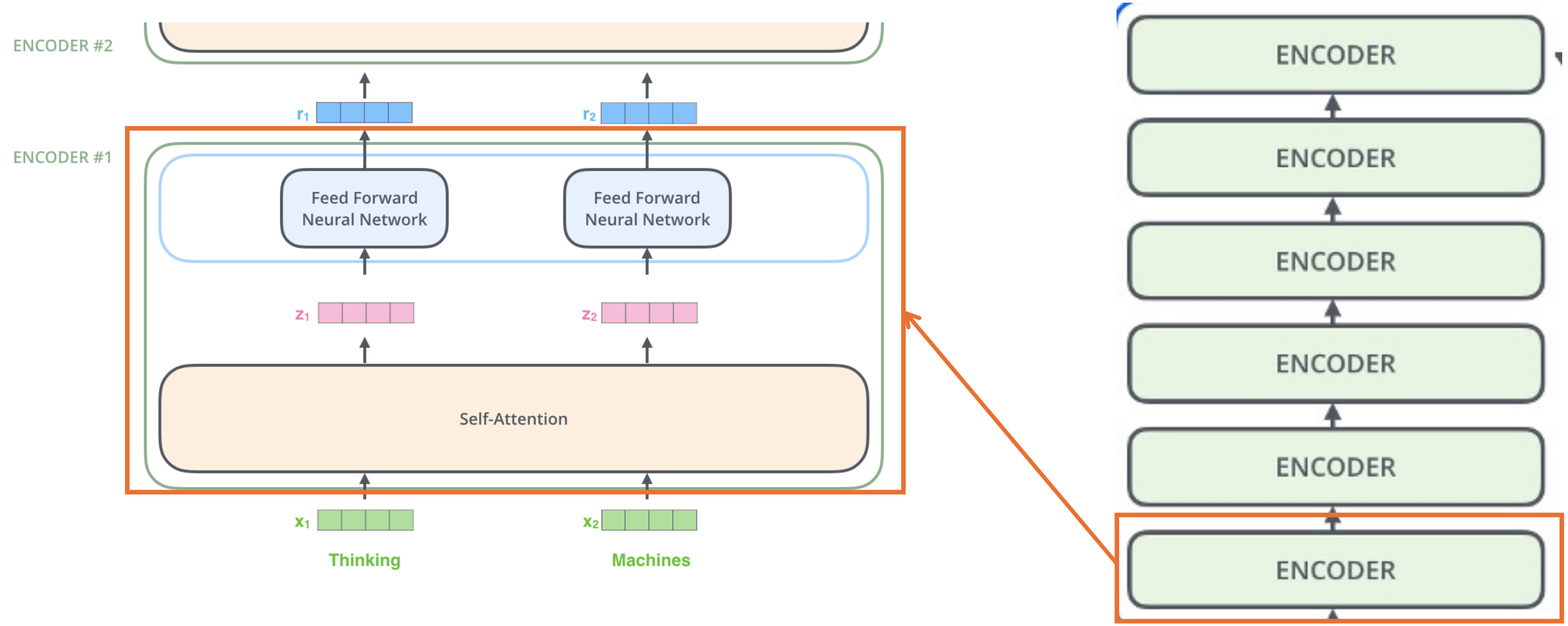
- Each of these two steps also has:
  - Residual (short-circuit) connection
  - LayerNorm: normalizes the inputs across the features to have mean 0 and variance 1



Ba et al., Layer Normalization. arxiv:1607.06450, 2016.

Complete Encoder

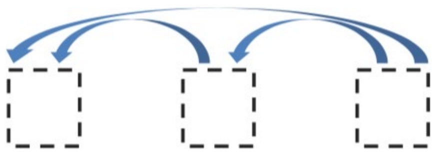
- For encoder, at each block, we use the same Q, K and V from the previous layer
- Blocks are repeated 6 times



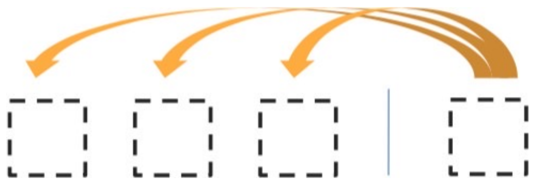
https://jalammar.github.io/illustrated-transformer/

Transformer Decoder

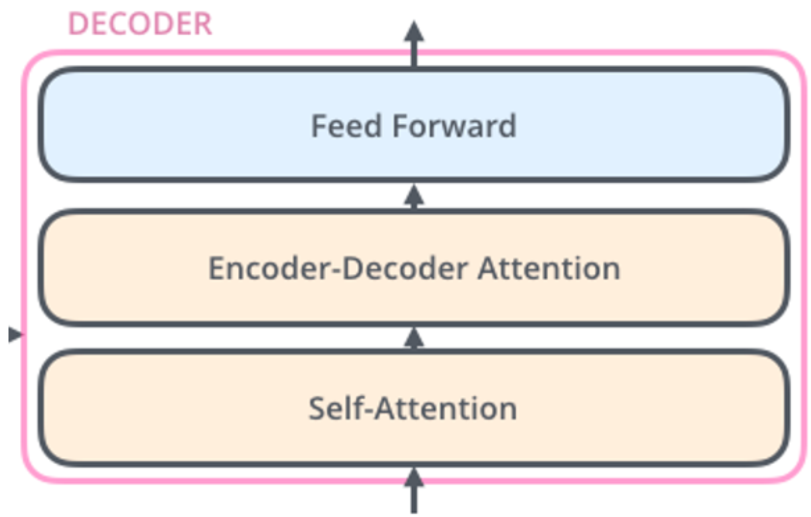
- **Masked decoder self-attention** is only allowed to *attend to earlier positions in the output sequence*. This is done by masking future positions



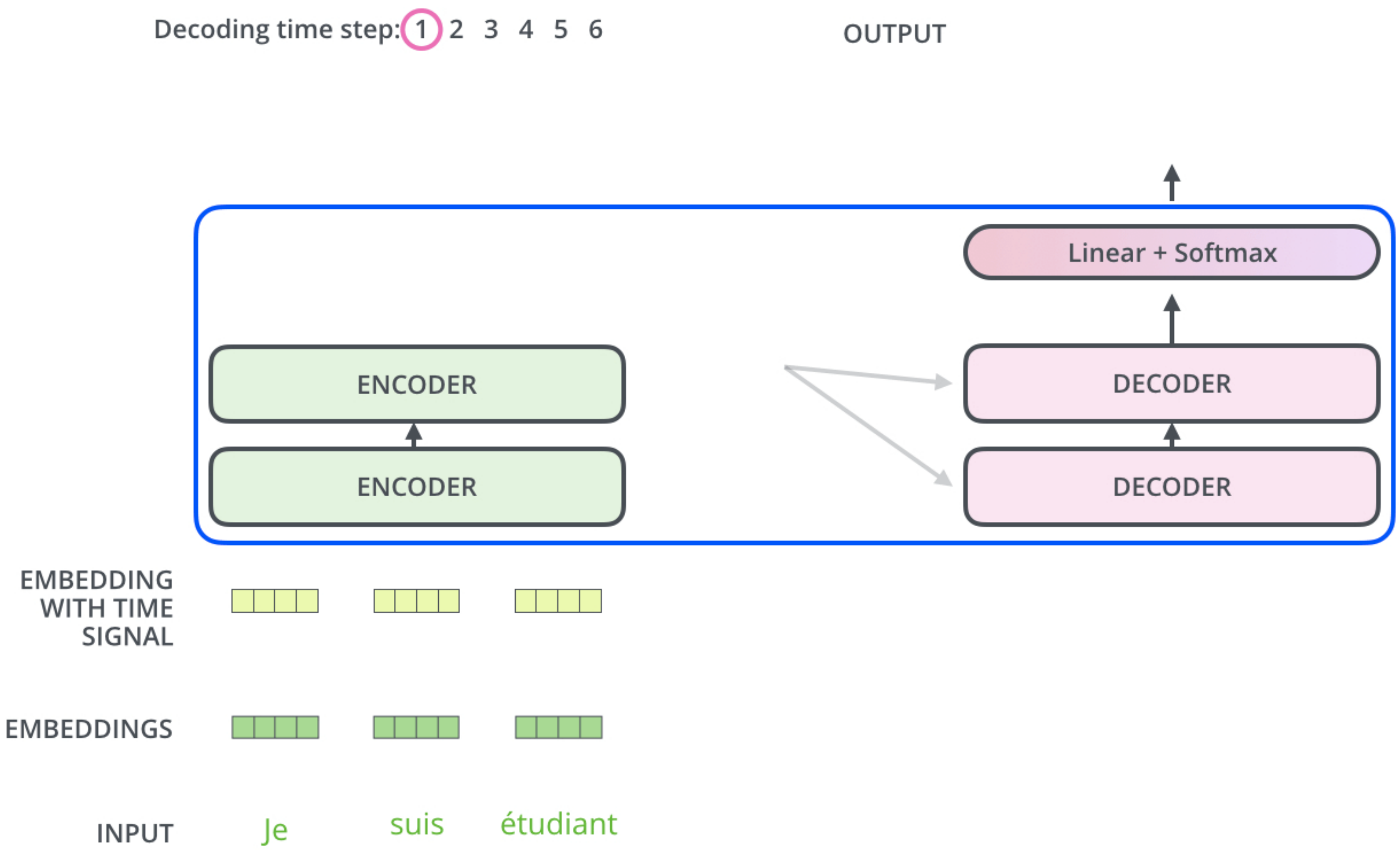
- **Encoder-Decoder Attention**, where *queries* come from *previous decoder layer* and *keys and values* come from *output of encoder*



- Blocks repeated 6 times

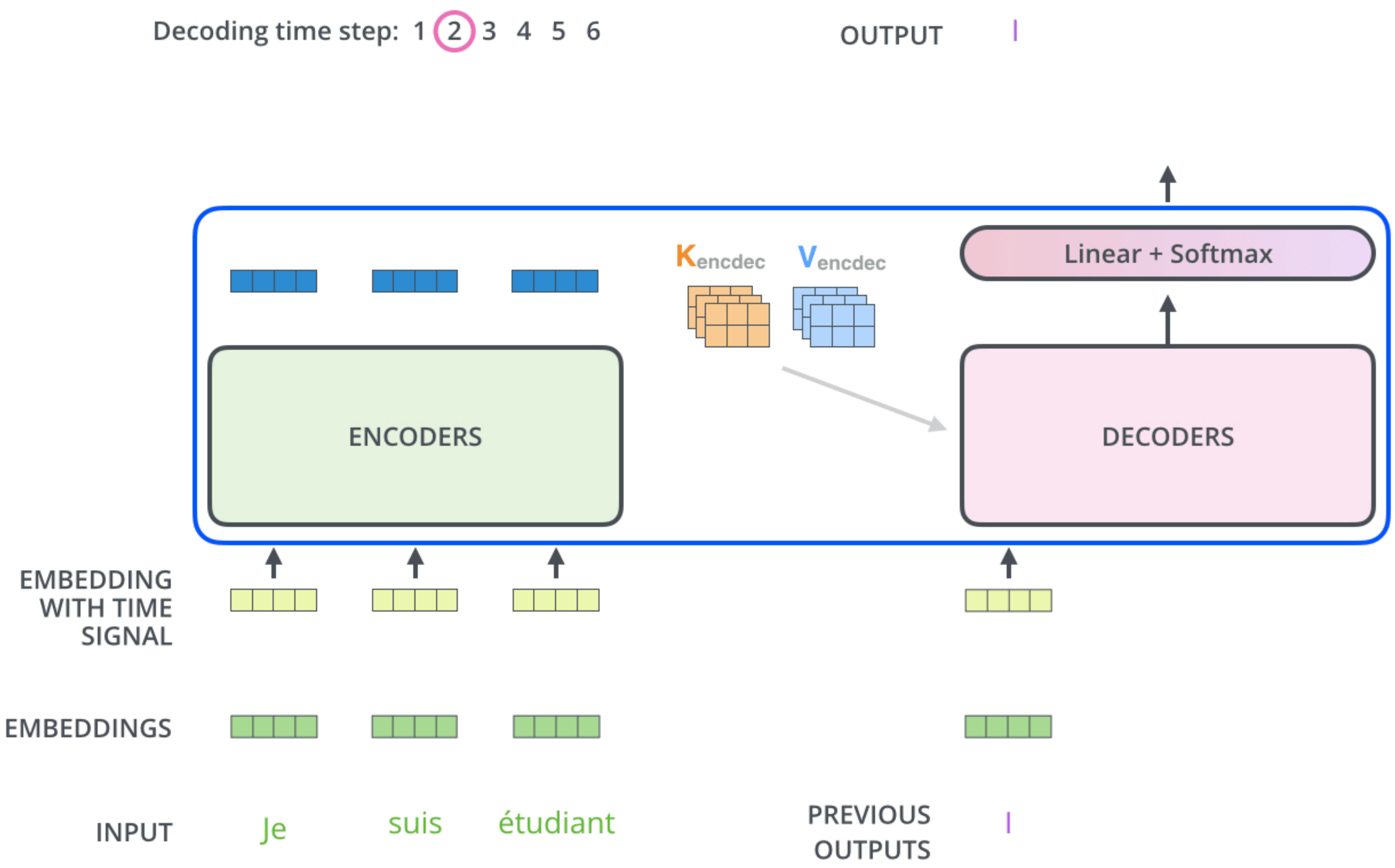


Transformer Decoder



https://jalammar.github.io/illustrated-transformer/

Transformer Decoder



https://jalammar.github.io/illustrated-transformer/

Tips and Tricks of the Transformer

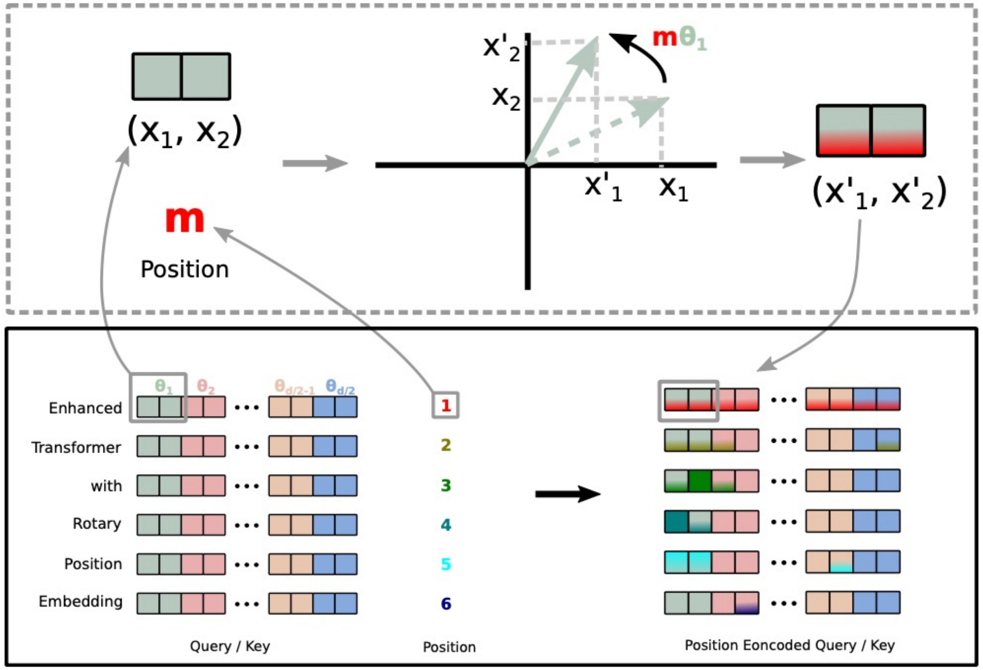
- Details in paper:
  - Byte-pair encodings
  - Checkpoint averaging
  - ADAM optimizer with learning rate changes
  - Dropout during training at every layer just before adding residual
  - Label smoothing
  - Auto-regressive decoding with beam search and length penalties

Improvement on Transformer –
Rotary Position Embedding (RoPE)

- It multiplies the **keys** and **queries** at every attention layer by sinusoidal embeddings.

$$\alpha_{i,j} = \mathrm{softmax}\left(\frac{(\mathbf{R}_i\mathbf{q}_i)^T\mathbf{R}_j\mathbf{k}_j}{\sqrt{D}}\right) = \mathrm{softmax}\left(\frac{\mathbf{q}_i^T(\mathbf{R}_i^T\mathbf{R}_j)\mathbf{k}_j}{\sqrt{D}}\right) = \mathrm{softmax}\left(\frac{\mathbf{q}_i^T\mathbf{R}_{j-i}\mathbf{k}_j}{\sqrt{D}}\right)$$

$$\mathbf{R}_i = \begin{bmatrix} \cos(i\theta_1) & -\sin(i\theta_1) & 0 & 0 & \dots & 0 & 0 \\ \sin(i\theta_1) & \cos(i\theta_1) & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos(i\theta_2) & -\sin(i\theta_2) & \dots & 0 & 0 \\ 0 & 0 & \sin(i\theta_2) & \cos(i\theta_2) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \cos(i\theta_{D/2}) & -\sin(i\theta_{D/2}) \\ 0 & 0 & 0 & 0 & \dots & \sin(i\theta_{D/2}) & \cos(i\theta_{D/2}) \end{bmatrix}$$



- The rotary encoding rotates different representation dimensions by θ_d.
- For two nearby positions, i.e. small distance $i - j$, the rotation R_{i-j} will be small.

Su et al., RoFormer: Enhanced Transformer with Rotary Position Embedding. arxiv:2104.09864, 2021.

Language Models Built on Transformer

Modern Language Models
- mostly built on the Transformer architecture

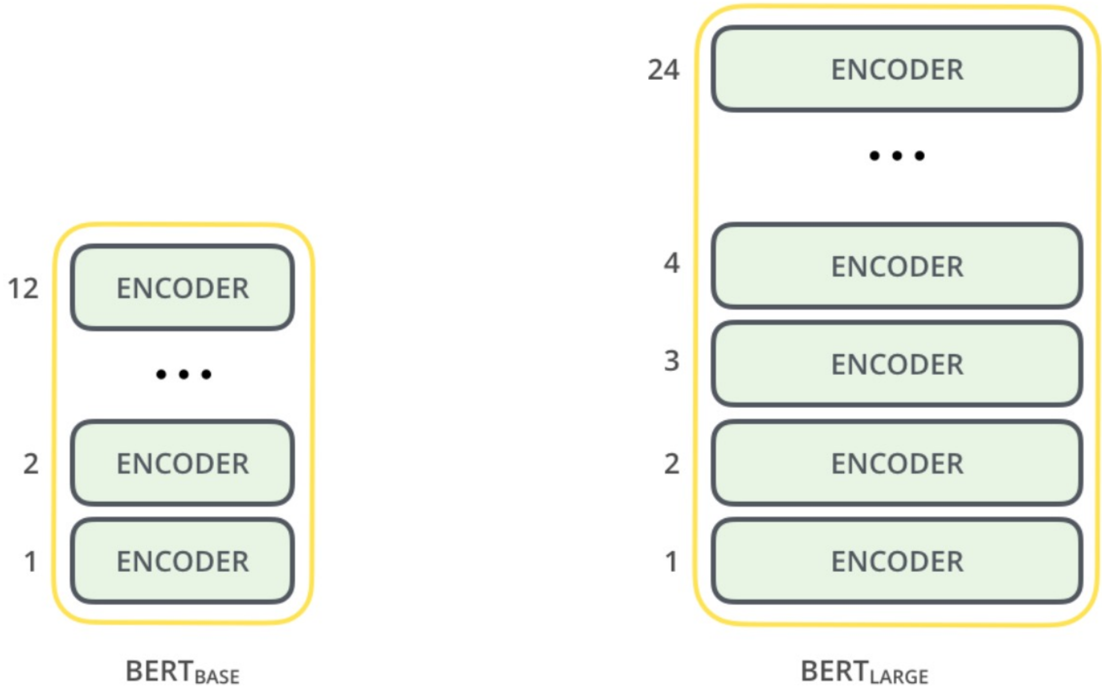
- **Encoder-only** models (e.g., BERT, RoBERTa, ALBERT)
  - Bidirectional attention

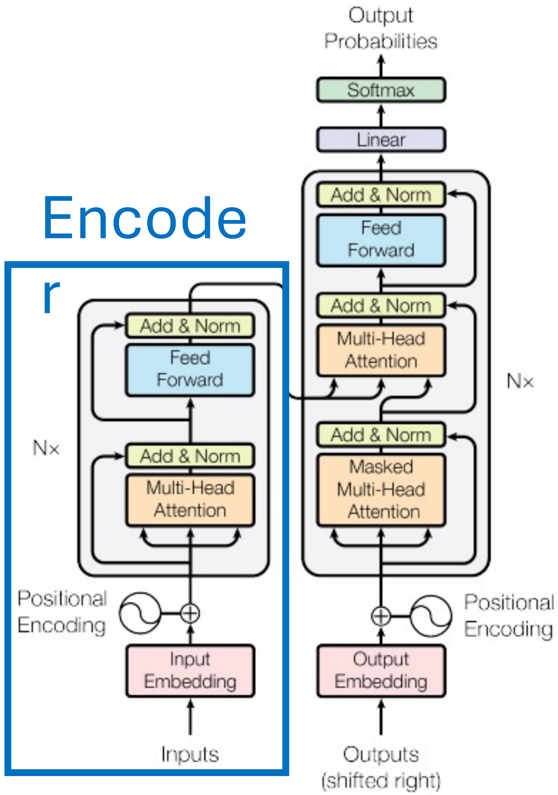
- **Encoder-decoder** models (e.g., T5, BART, Flan-T5)
  - **Encoder**: Bidirectional attention
  - **Decoder**:
    1. Cross-attention to the encoder hidden states
    2. Unidirectional attention mask for sequence generation
       - i.e., each token only attends to the past tokens and itself

- **Decoder-only** models (e.g., GPT-x models, OPT, BLOOM, Gopher)
  - Using the unidirectional attention mask

Bidirectional Encoder Representations from Transformers (BERT)

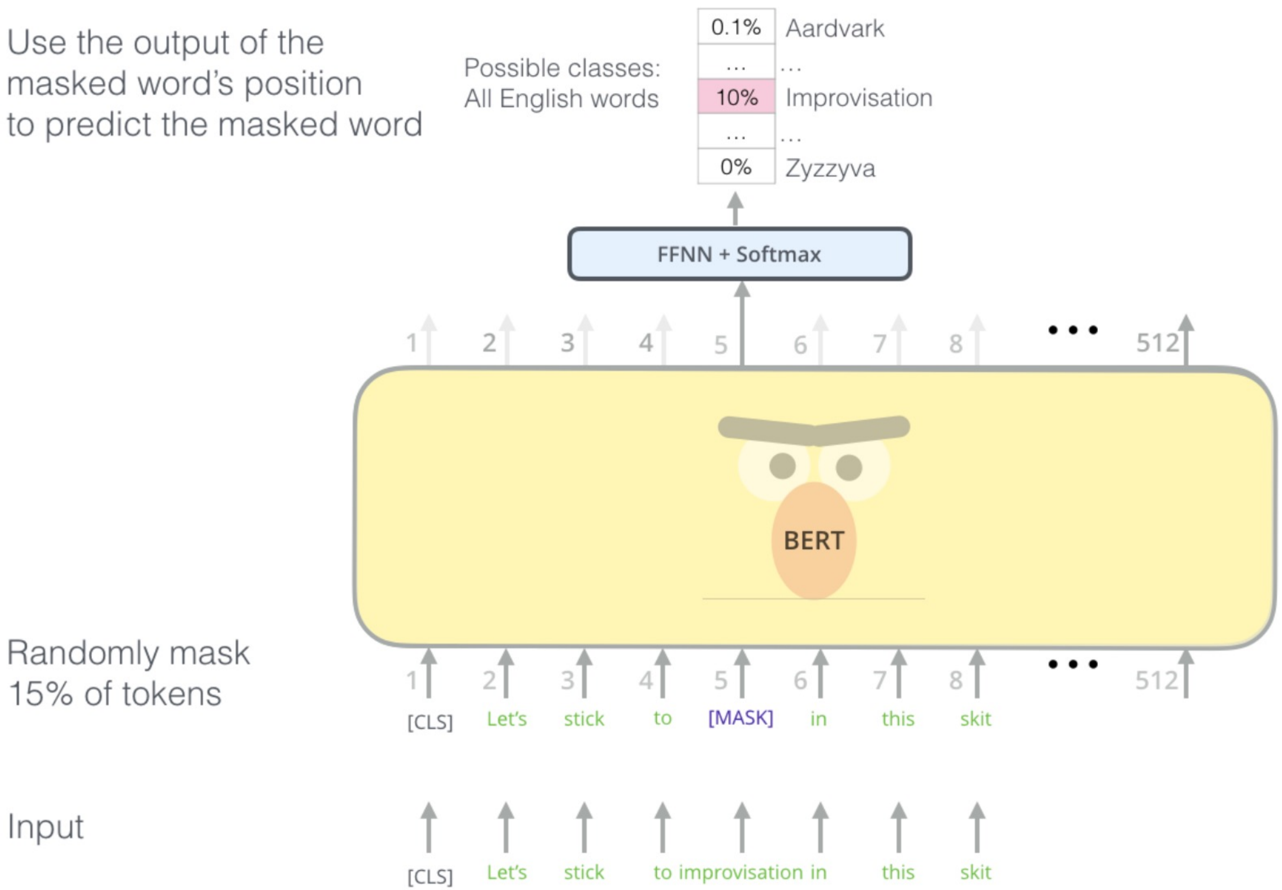
- BERT = Encoder of Transformer
- Learn from a large text corpus without annotation





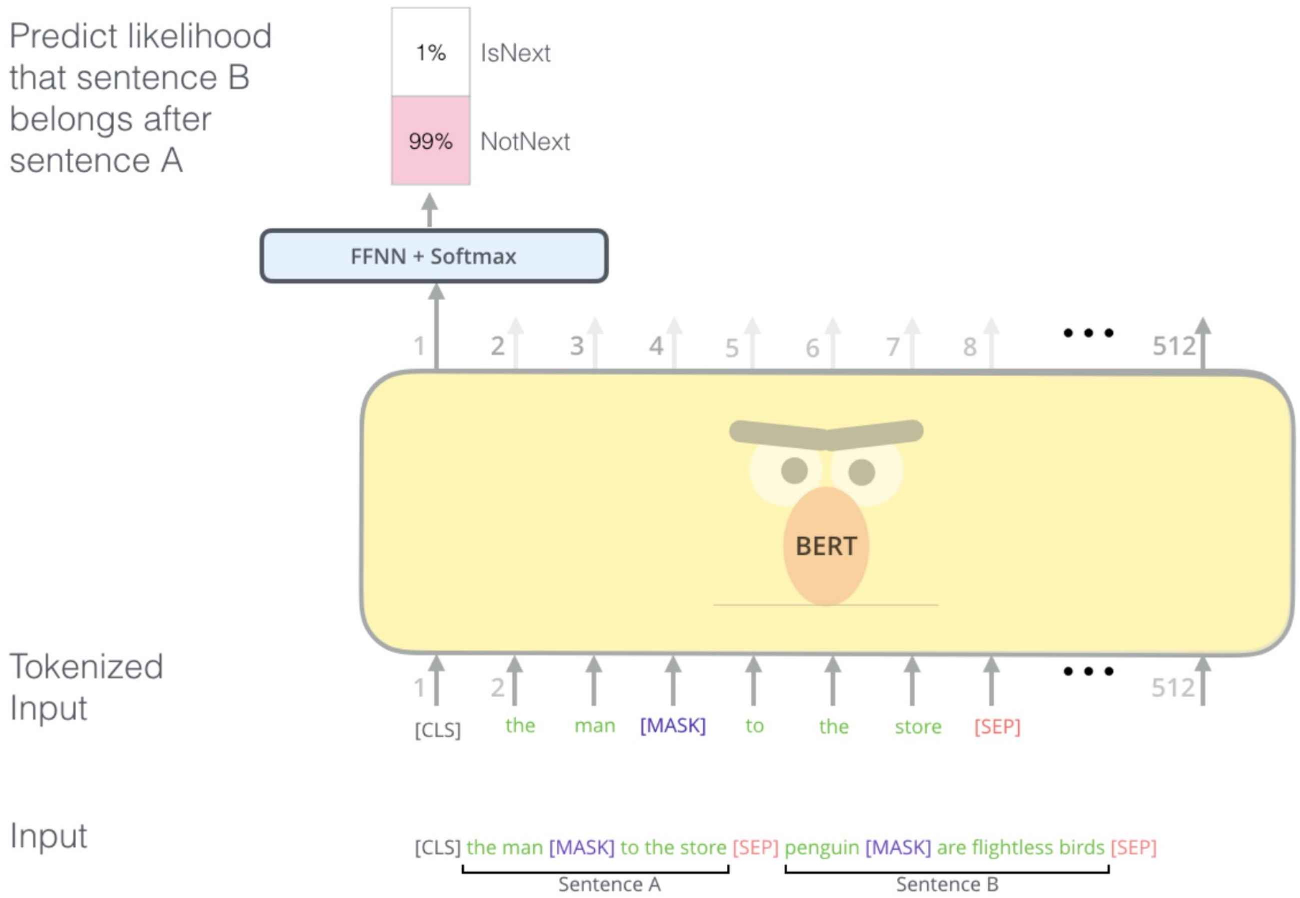
(This and related figures from http://jalammar.github.io/illustrated-bert/)

BERT Training – Masked Language Model



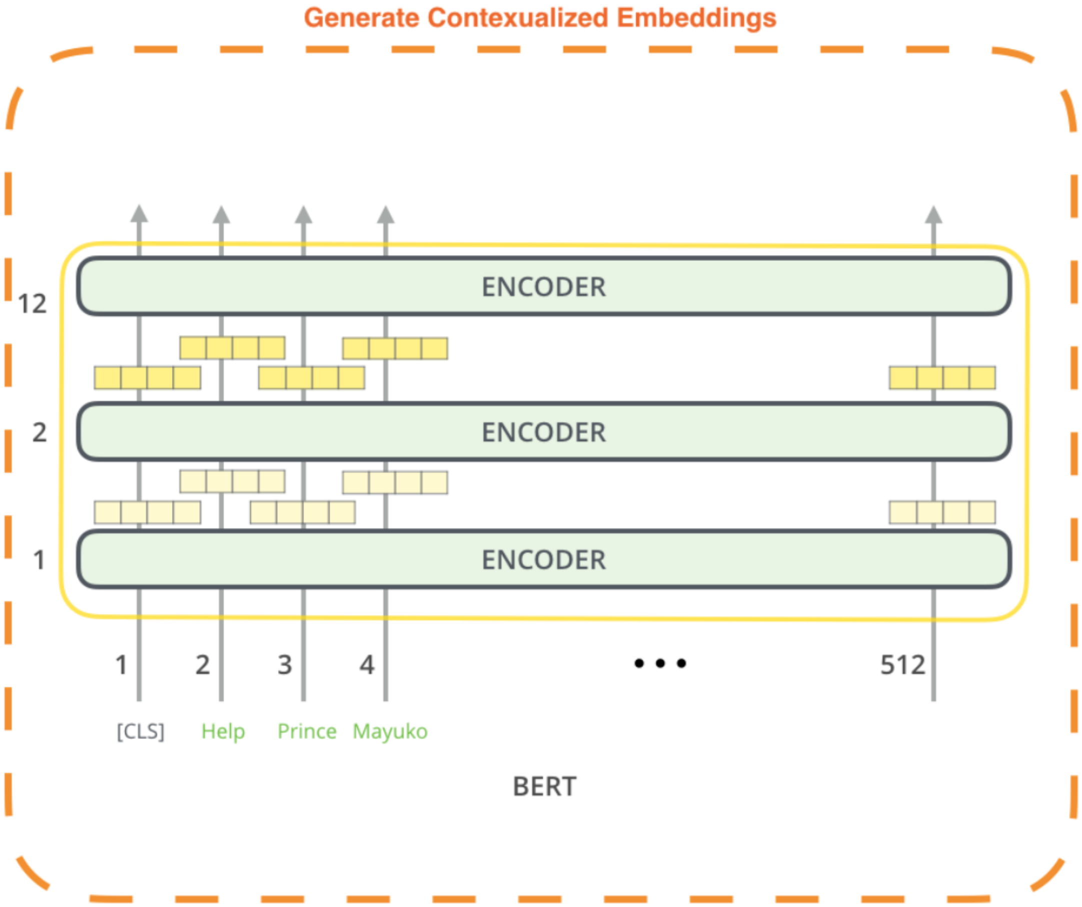
BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.

BERT Training – Two-Sentence Task

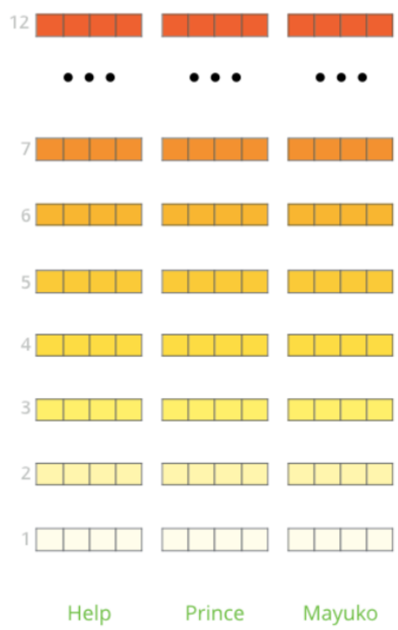


The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.

BERT – Extract contextualised word embeddings

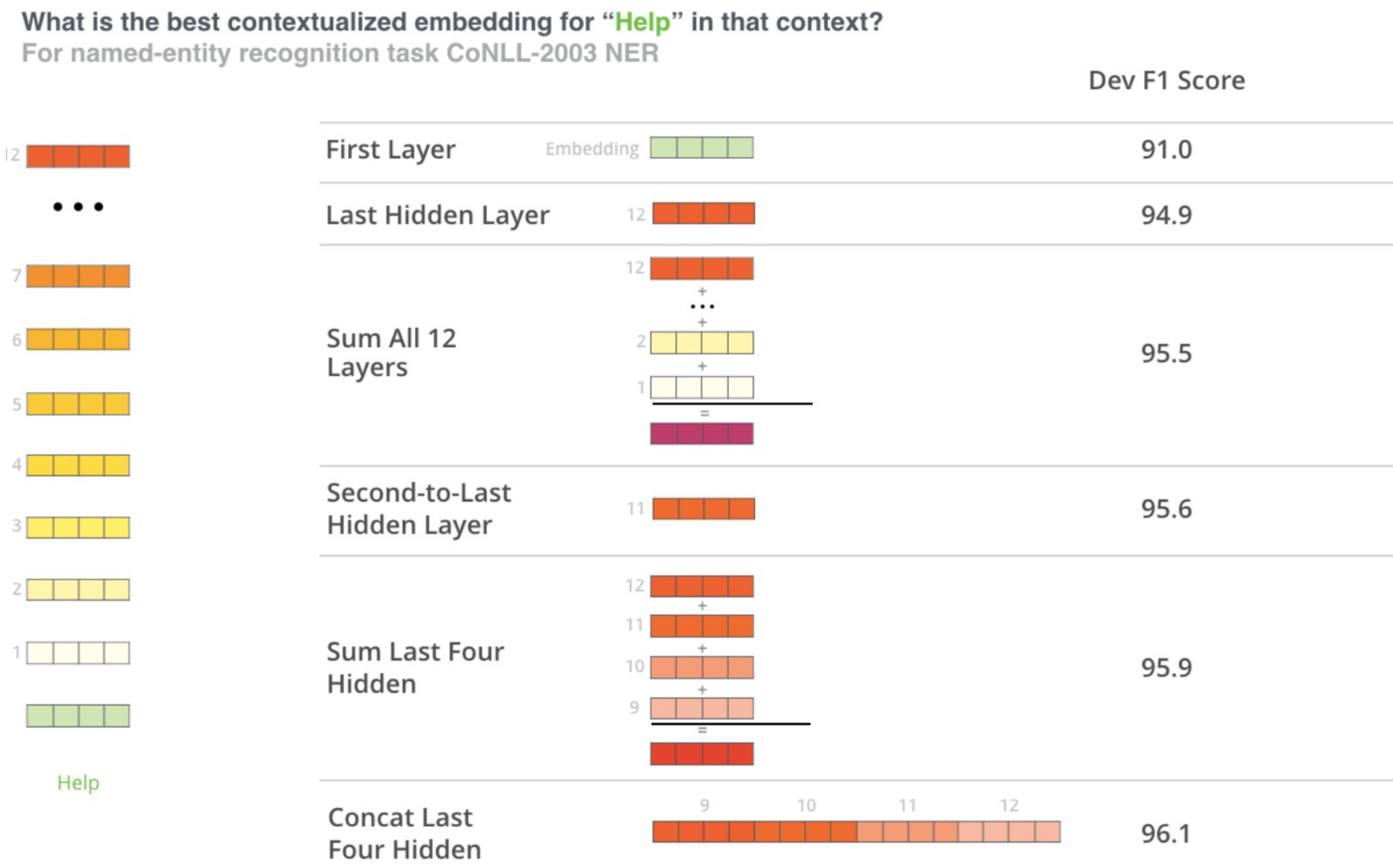


The output of each encoder layer along each token's path can be used as a feature representing that token.

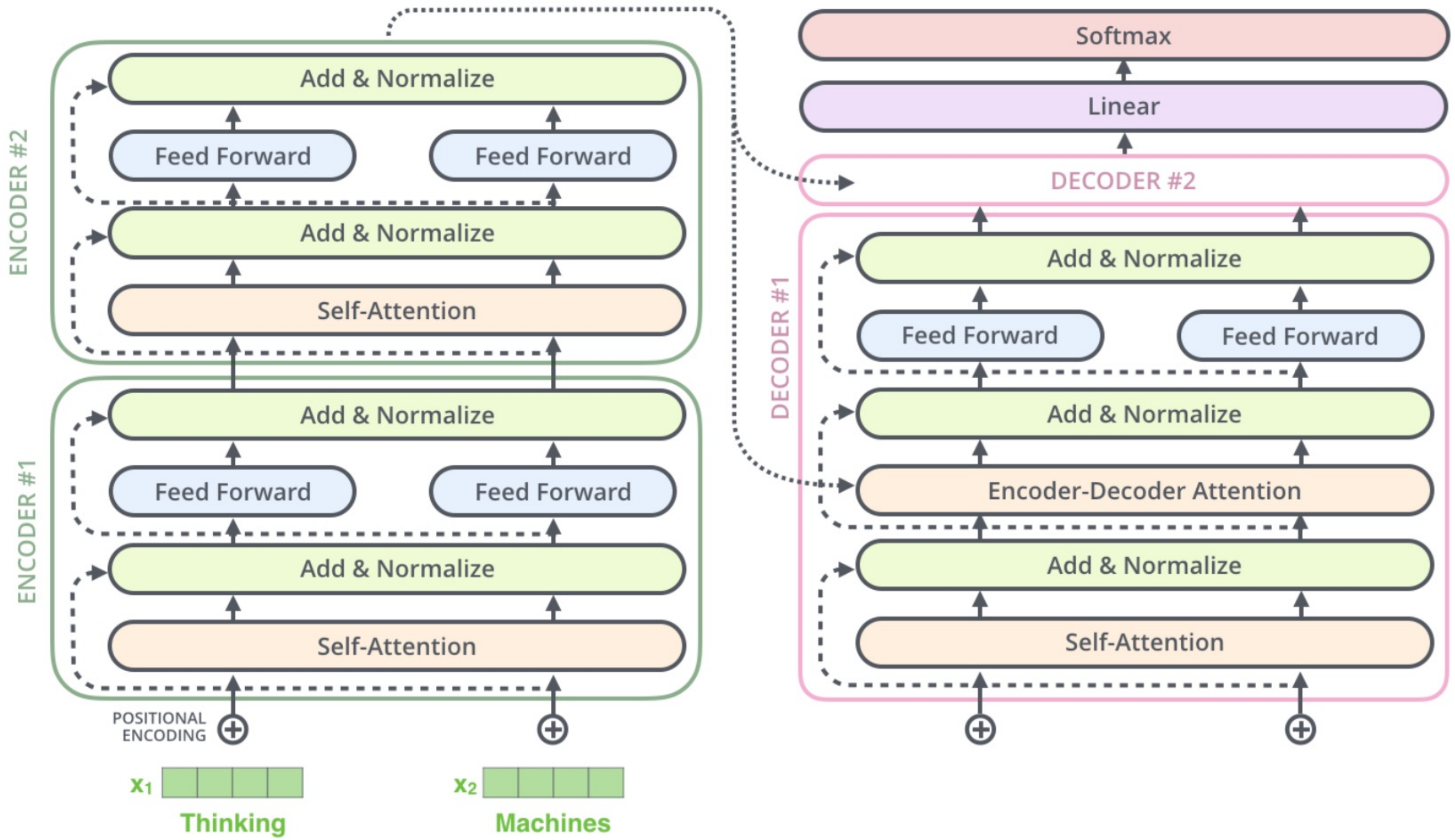


But which one should we use?

BERT – Extract contextualised word embeddings



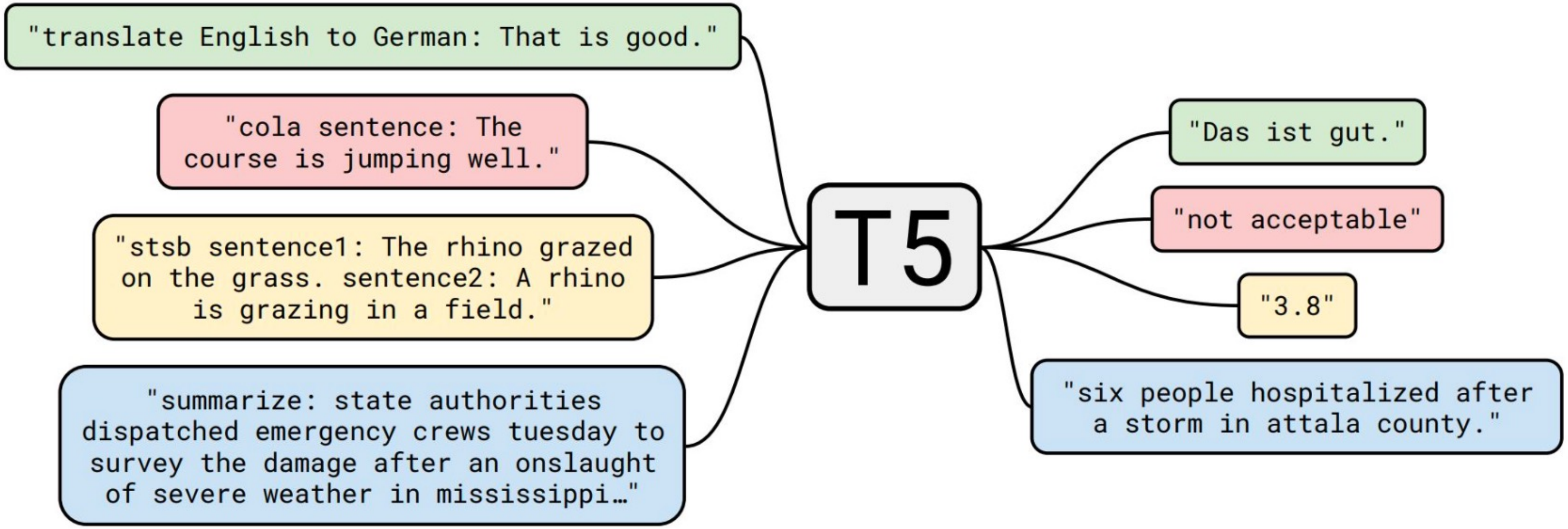
Encoder-Decoder Model: T5



https://jalammar.github.io/illustrated-transformer/

T5: Text-to-Text Transfer Transformer

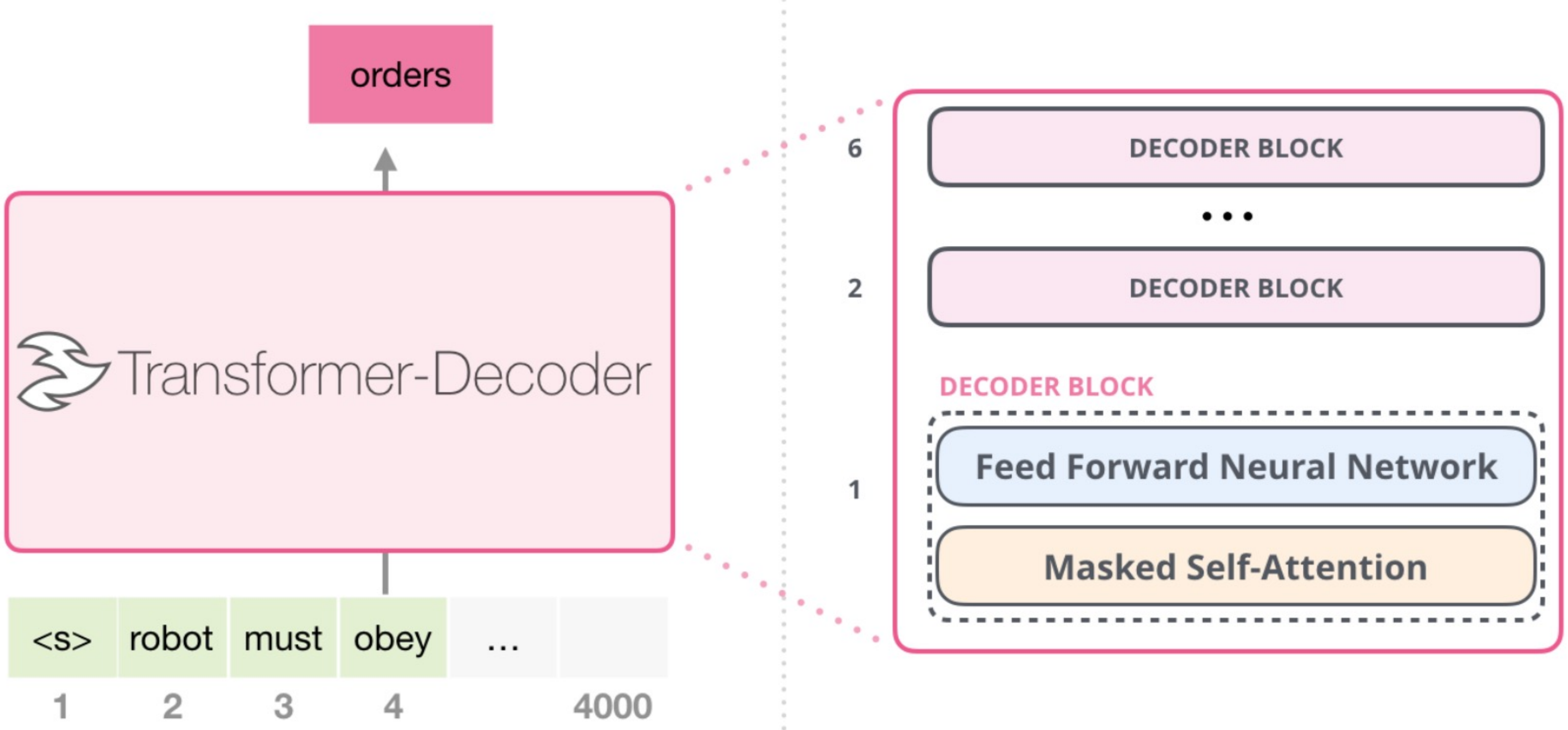
[Task-specific prefix]: [Input text] → T5 → [output text]



Raffel et al., Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arxiv:1910.10683, 2019.

Decoder-only Model: OpenAI's GPT-x

- Use the **decoder layers** from the Transformer architecture.
- **Training objective**: predict the next word using massive (unlabelled) data.

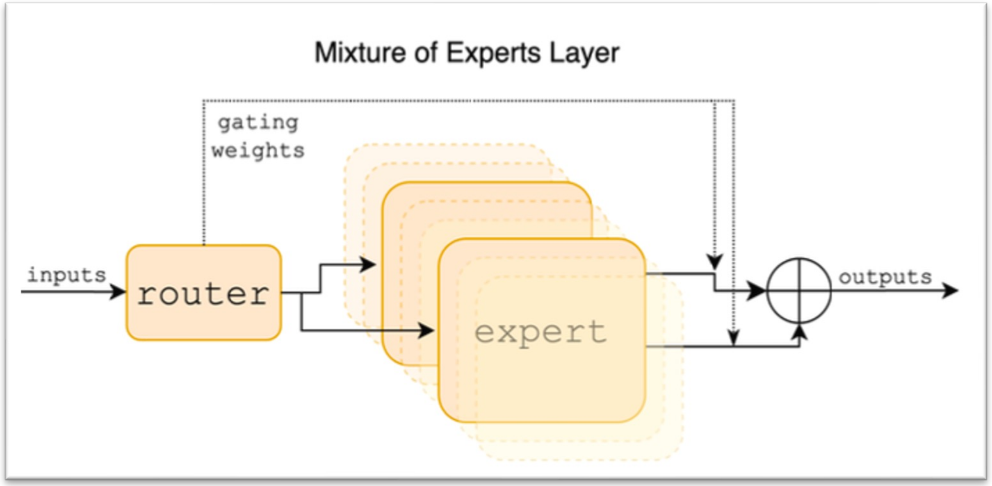


Brown et al., [Language models are few-shot learners](). NeurIPS 2020.

Figure source: https://jalammar.github.io/illustrated-gpt2/

Mixture of Experts (MoE) Models

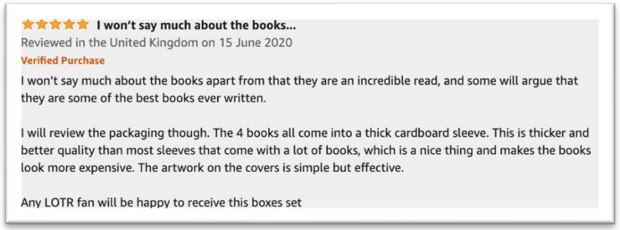
- **Mixtral $8\times7B$** – a Sparse Mixture of Experts language model
  - A **decoder-only** model
  - The feedforward block picks from a set of 8 distinct groups of parameters.
  - At every layer, for every token, a router network chooses two of these groups (the "**experts**") to process the token and combine their output additively.
  - The model only uses a fraction of the total set of parameters per token.



Jiang et al., Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024

LLM Training Paradigms

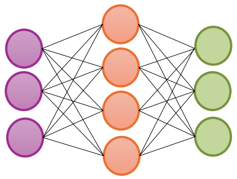
Learning Task-Specific Models


★★★★★ I won't say much about the books...
Reviewed in the United Kingdom on 15 June 2020
Verified Purchase
I won't say much about the books apart from that they are an incredible read, and some will argue that they are some of the best books ever written.

I will review the packaging though. The 4 books all come into a thick cardboard sleeve. This is thicker and better quality than most sleeves that come with a lot of books, which is a nice thing and makes the books look more expensive. The artwork on the covers is simple but effective.

Any LOTR fan will be happy to receive this boxes set

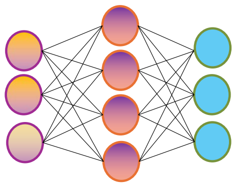
Book reviews



Sentiment Classification

Label: positive

ChatGPT is an AI chatbot developed by OpenAI and released in November 2022.

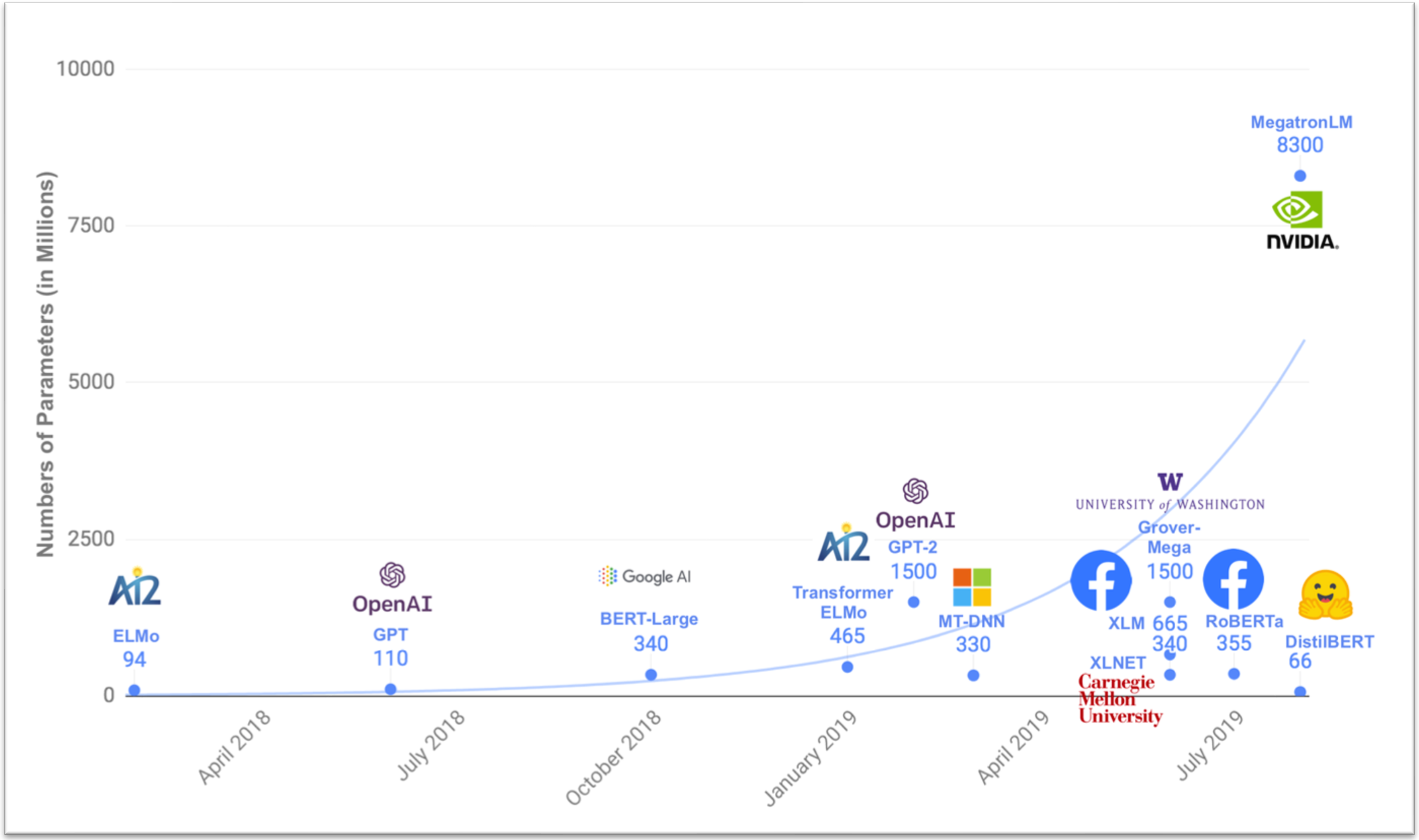


Information Extraction

Product: ChatGPT
Organisation: OpenAI
Date: November 2022

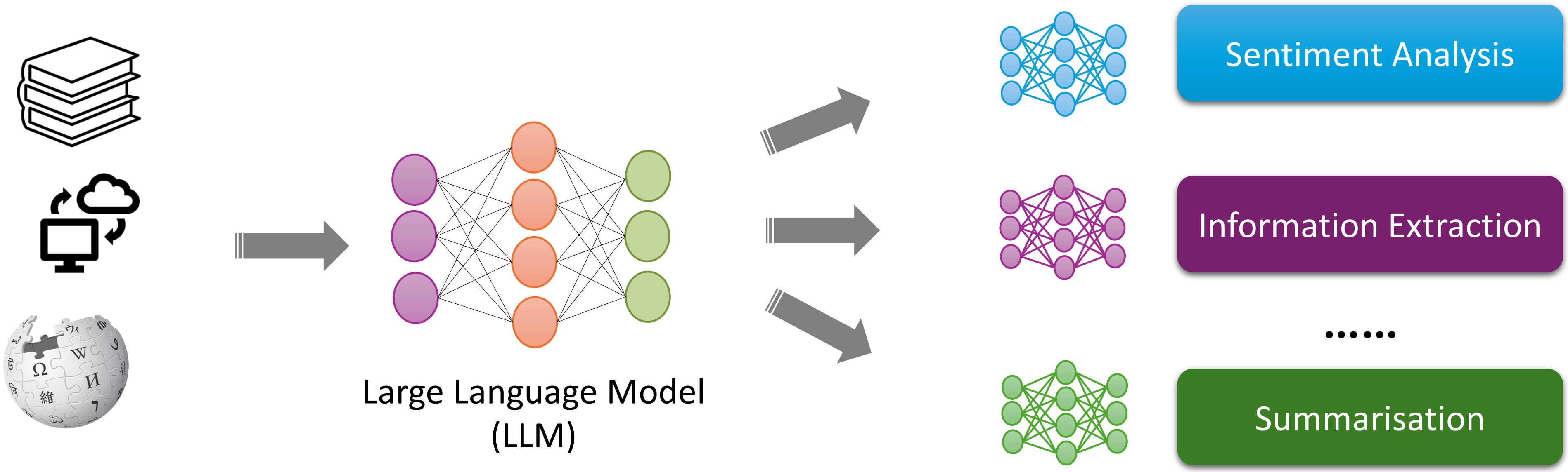
……

Pre-trained Language Models



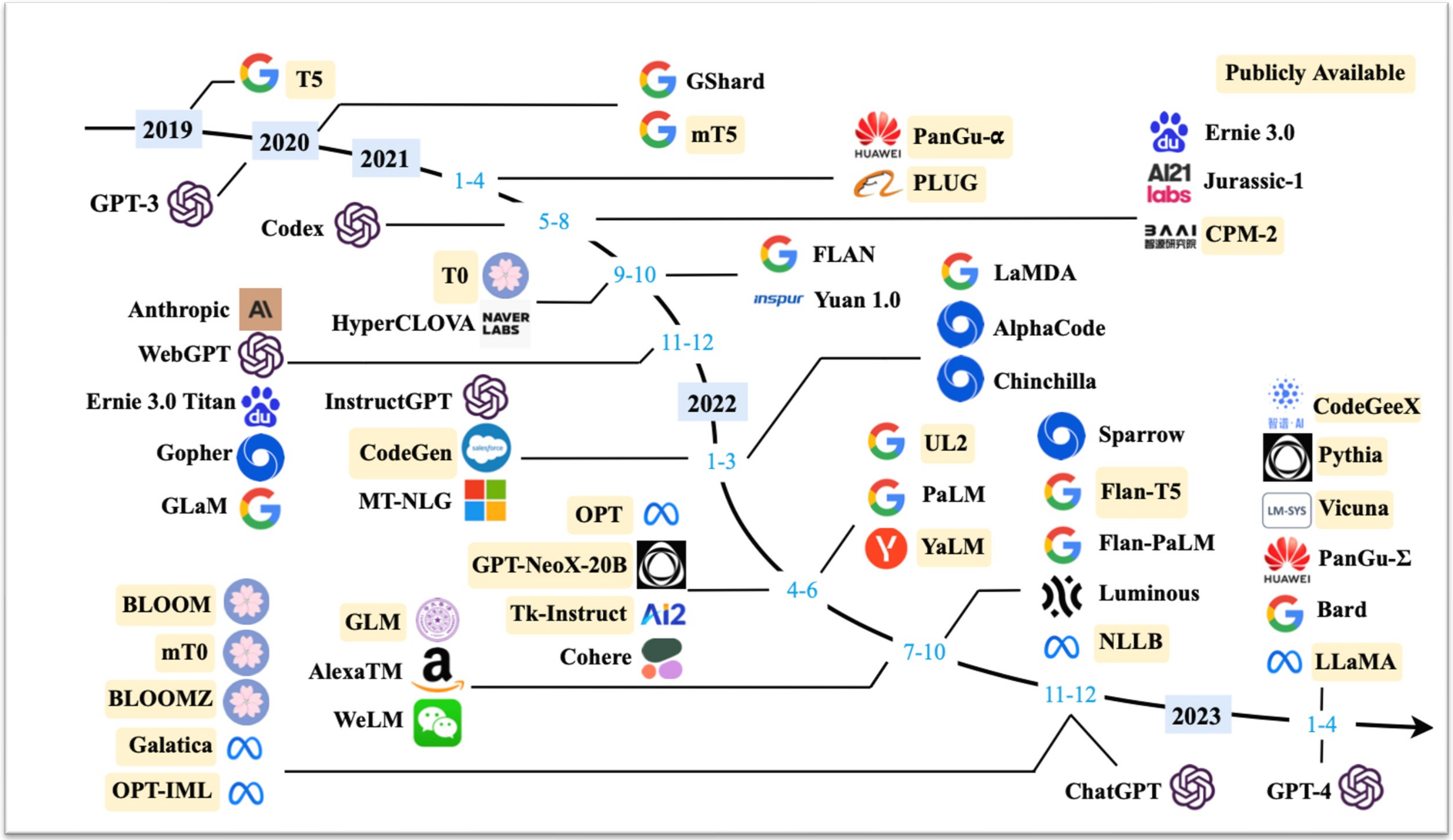
Sanh, V., et al., 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

Pre-training and then Fine-Tuning



(a) Language model pre-training (b) Language model fine-tuning

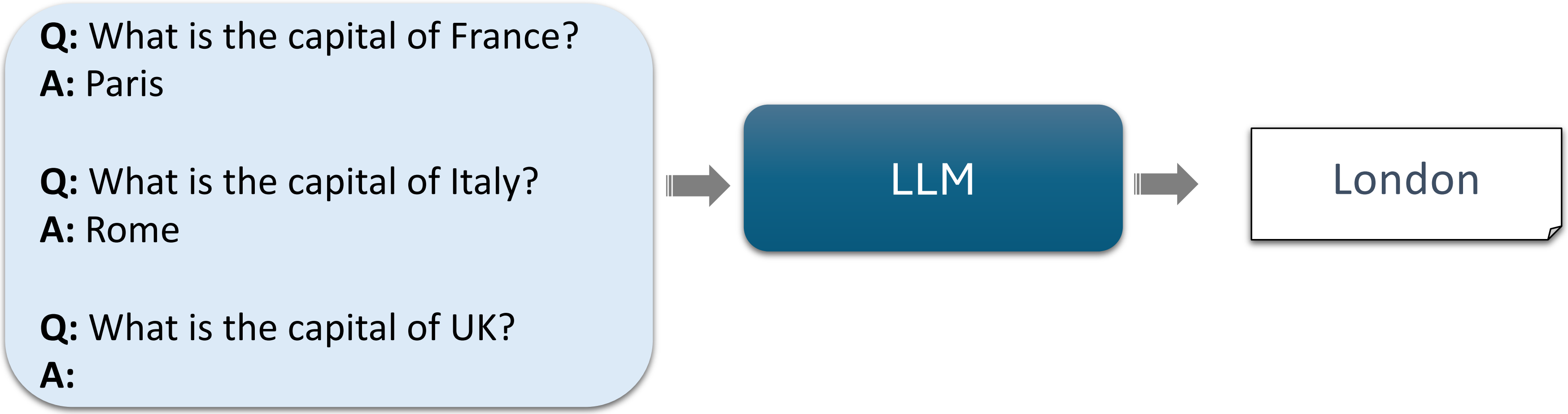
Pre-trained Large Language Models (LLMs)



Zhao, W.X., et al., 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

In-Context Learning

- LLM learns to perform a task during ***inference*** by being given examples or instructions in the input prompt, **without parameter update**.
  - Users provide **examples (few-shot)** or **instructions (zero-shot)** in the prompt.

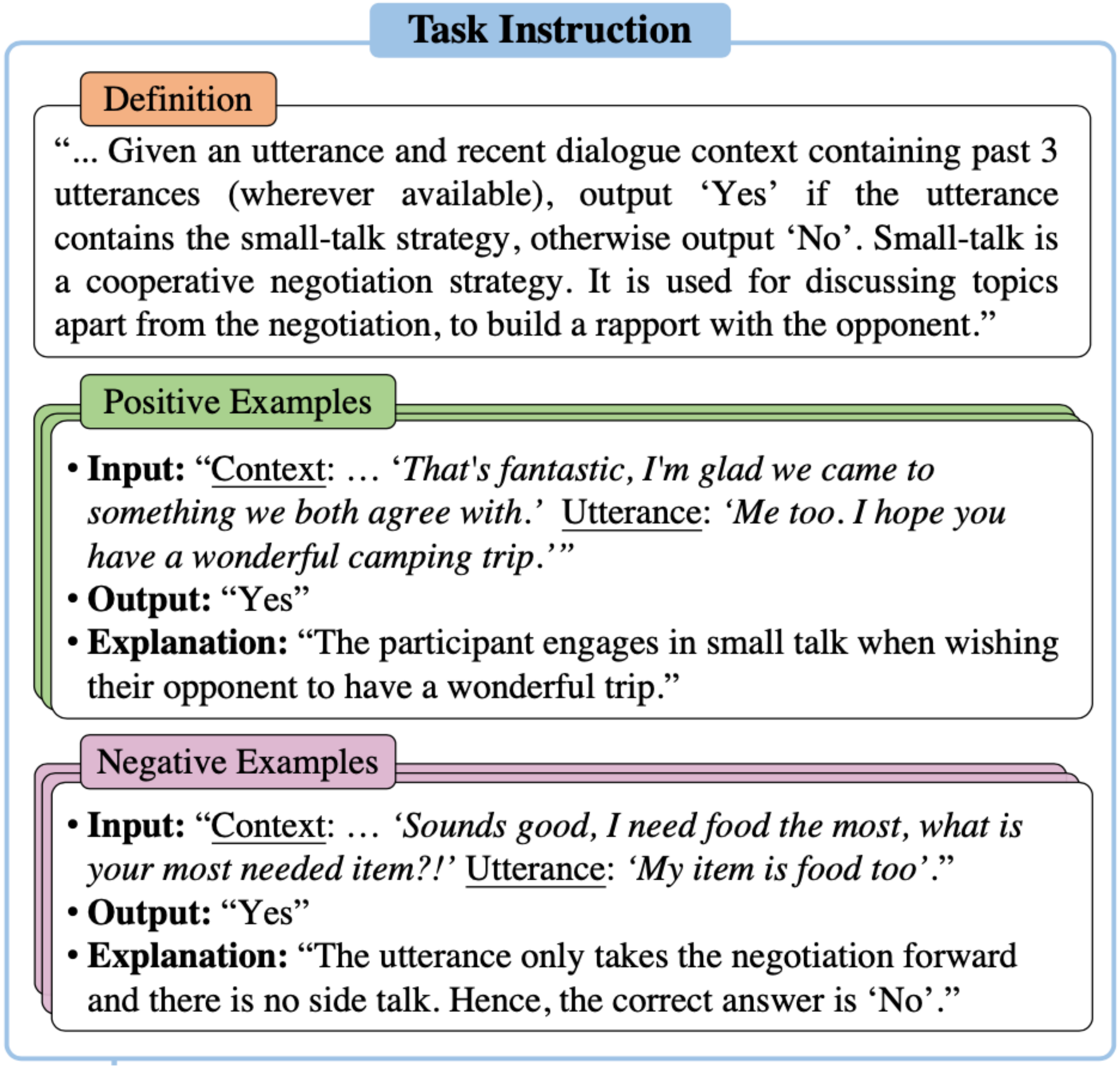


Brown et al., 2020. Language models are few-shot learners. NeurIPS, 33, pp.1877-1901.

Instruction Tuning

Fine-tuning pre-trained LLMs on **formatted task instances**.

- Model learns to **follow instructions** better.
- Improves **zero-shot performance** on unseen tasks.



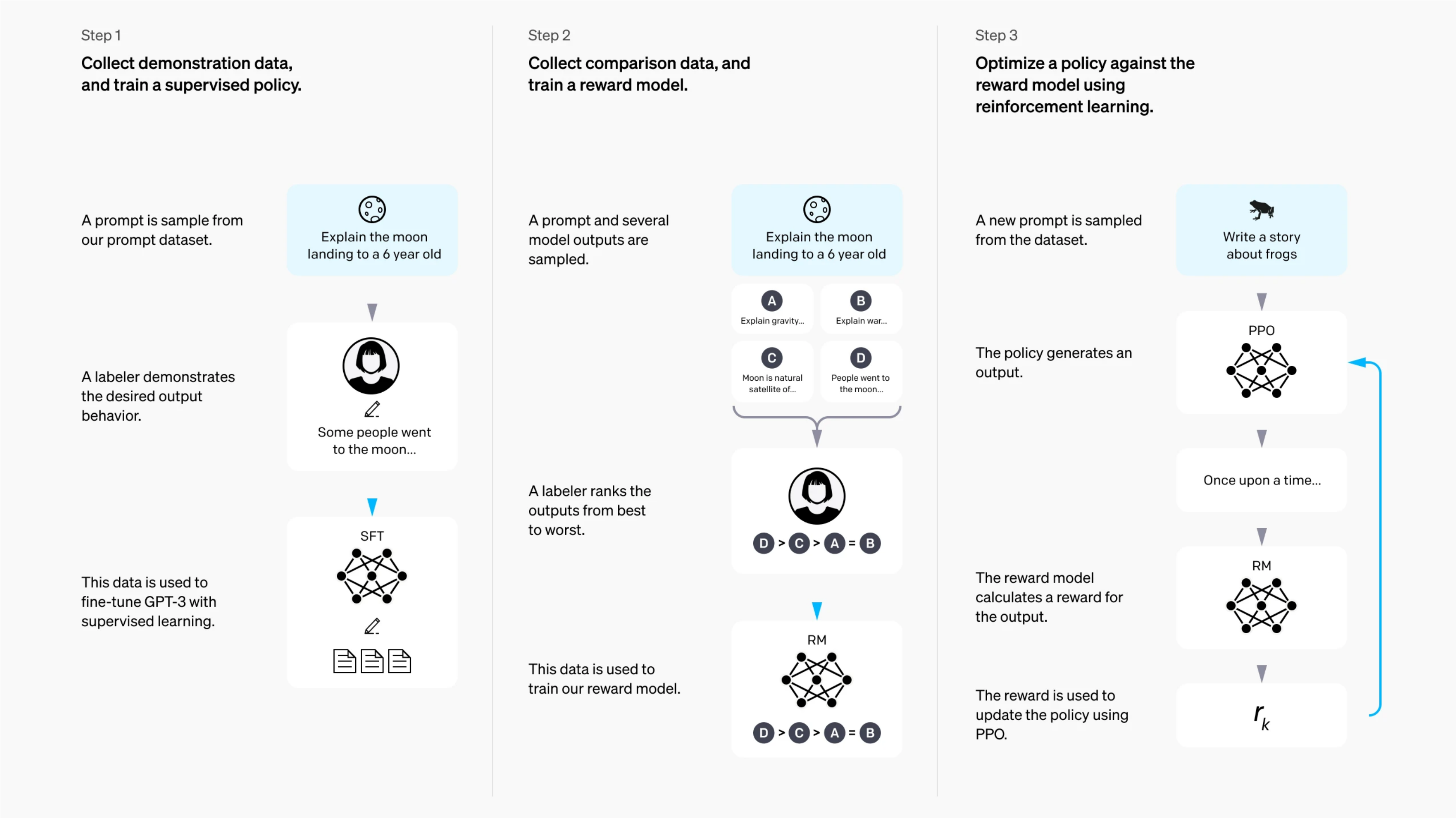
Zhang et al. [Instruction Tuning for Large Language Models: A Survey](). arXiv 2308.10792, 2024.

Alignment Tuning

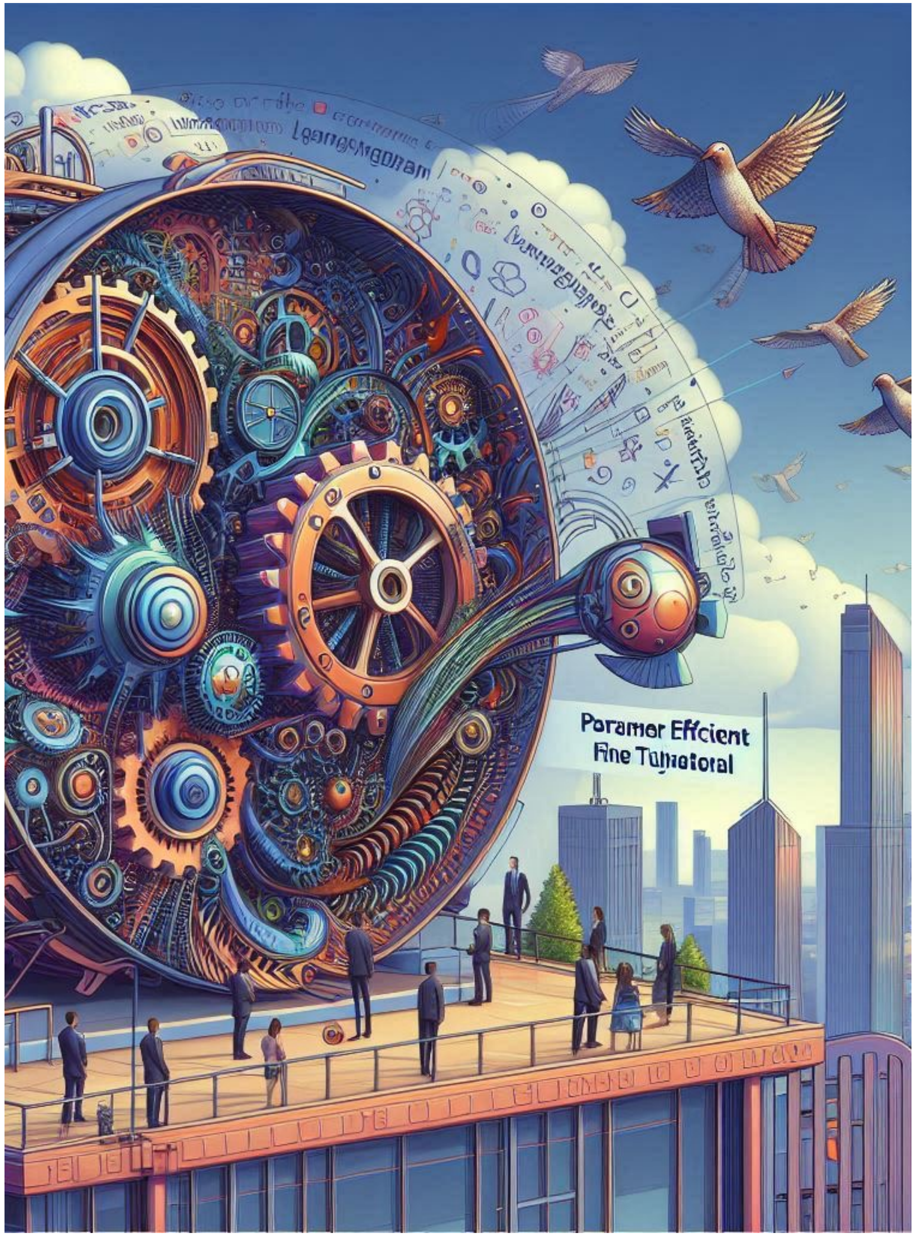
- Adjusting an LLM's behaviour to better align with **human values, intentions, and preferences** (e.g., around helpfulness, honesty, and safety).

- E.g., **Reinforcement Learning from Human Feedback (RLHF)**
  1. Human annotators **rank** different model responses.
  2. A **reward model** is trained to reflect these preferences.
  3. The LLM is then **fine-tuned** using **reinforcement learning (e.g., PPO)** to produce more preferred outputs.

Alignment Tuning



Ouyang et al. Training language models to follow instructions with human feedback. arXiv 2203.02155, 2022.



Parameter-Efficient Fine-Tuning

Parameter-Efficient Fine-Tuning (PEFT)

- LLMs require a lot of memory storage to store, and many high-end GPUs to fine-tune
  - Llama 70B needs 130GB storage and 4 A100-40G to fine-tune.

- Parameter-efficient fine tuning can make LLMs more accessible.
  - Only fine tune **a subset of the parameters** for each task.
  - A 33B model can be fine-tuned on a 24GB consumer GPU in less than 12 hours.

Parameter-Efficient Fine-Tuning (PEFT)

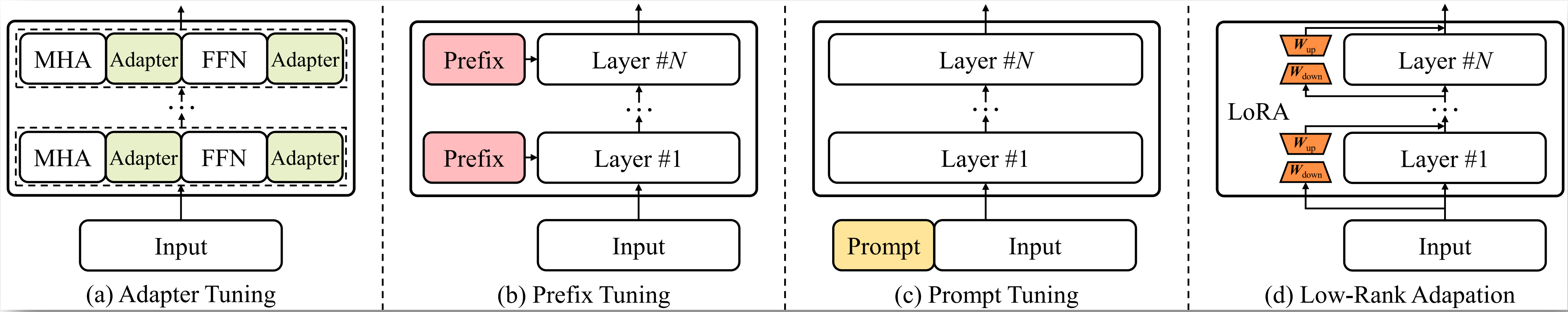


Figure from (Zhao et al., 2023)

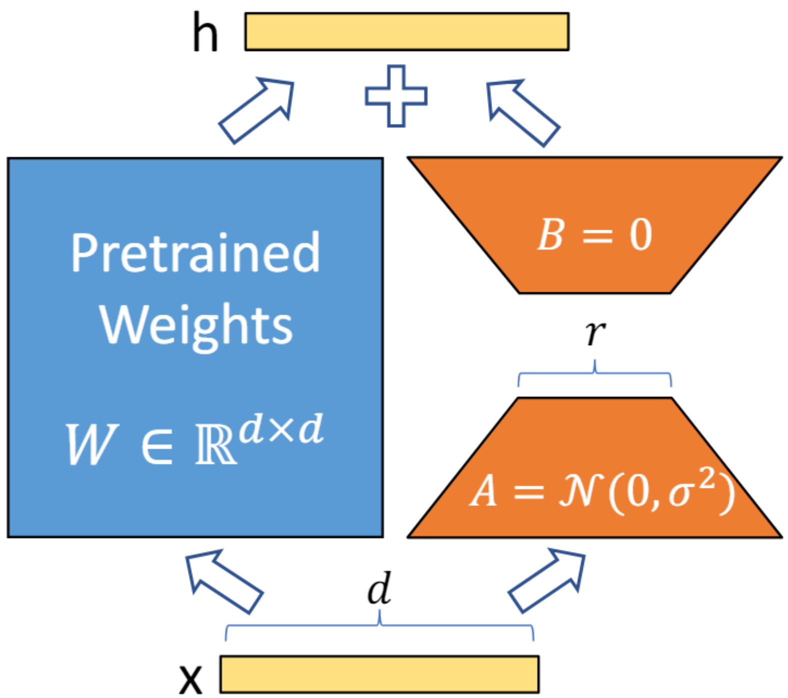
- **Adapter Tuning**
  - Add **adapter layers** in between the transformer layers of a large model.
  - During fine-tuning, only tune the adapter layers.
- **Prefix Tuning**
  - Learns a sequence of **prefixes** that are prepended at each **transformer layer**.
  - Learn an optimal prefix for each task.
- **Prompt Tuning**
  - learns a single **prompt representation** that is prepended to the **embedded input**.

He et al., Towards a unified view of parameter-efficient transfer learning. ICLR 2022.

LoRA: Low-Rank Adaptation

$$h = W_0 x + \Delta W x$$
$$= W_0 x + \mathrm{BA} x$$

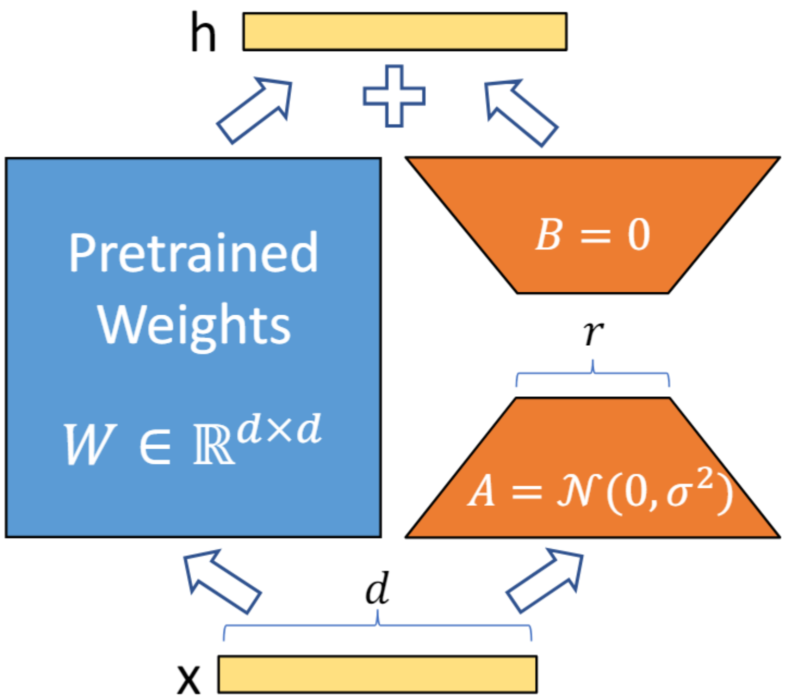
- $W_0 \in R^{d \times k}$ is a weight matrix in the pre-trained model, ΔW is an **adaptor** of the same size.
- W_0 is **frozen**, only ΔW is **updated.**
- $\mathrm{B} \in R^{d \times r}$ and $A \in R^{r \times k}$ are **low rank matrices**, $r \ll \min(d, k)$.
- B is initialised as zero and A uses random Gaussian.



Hu et al., LoRA: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

LoRA – How to adjust the hyperparameters

- **Rank (r)**
  - Lower $r \longrightarrow$ fewer trainable parameters.
  - Little statistical difference between r = 8 and 256 when applied to all layers.
  - Typical values: **8, 16, 32**.
- **Scaling (α)**
  - When adaptors are merged back, original weights are scaled by $\boldsymbol{\alpha}$ **/** $\boldsymbol{r}$.
  - Larger $\alpha \longrightarrow$ stronger adaptor influence (similar to learnir rate).
  - Typical values: **2r, r, 0.5r, 0.25r**.
- **Dropout**
  - Dropout = **0.05** helps smaller models (7B, 13B).



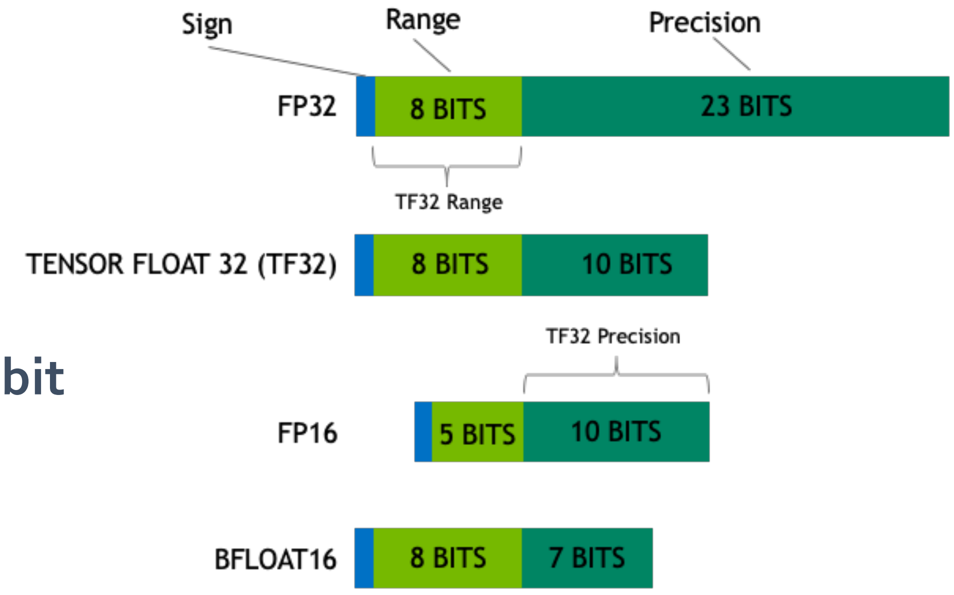
Hu et al., LoRA: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

PEFT - QLoRA

- LoRA – the full LLM still needs to be loaded first which consumes lots of memory.

- QLoRA: Efficient Finetuning of **Quantised** LLMs.

- **Quantisation** – techniques for performing computations and storing tensors at **lower bit width** than floating point precision.



Dettmers et al., QLoRA: Efficient finetuning of quantized LLMs. NeurIPS, 36, 2023.
Frantar et al., GPTQ: Accurate post-training quantization for generative pre-trained transformers. arXiv preprint arXiv:2210.17323, 2022.

PEFT - QLoRA

- QLoRA conducts LoRA fine-tuning based on a quantised model

- Two novel techniques are used:
  1. **4-bit NormalFloat**: Instead of quantising uniformly, it estimates the quantile of the input tensor through the empirical cumulative distribution function.
  2. **Double quantisation**: The quantisation constants are also quantised.

- The forward and backward passes are performed in 16-bit.

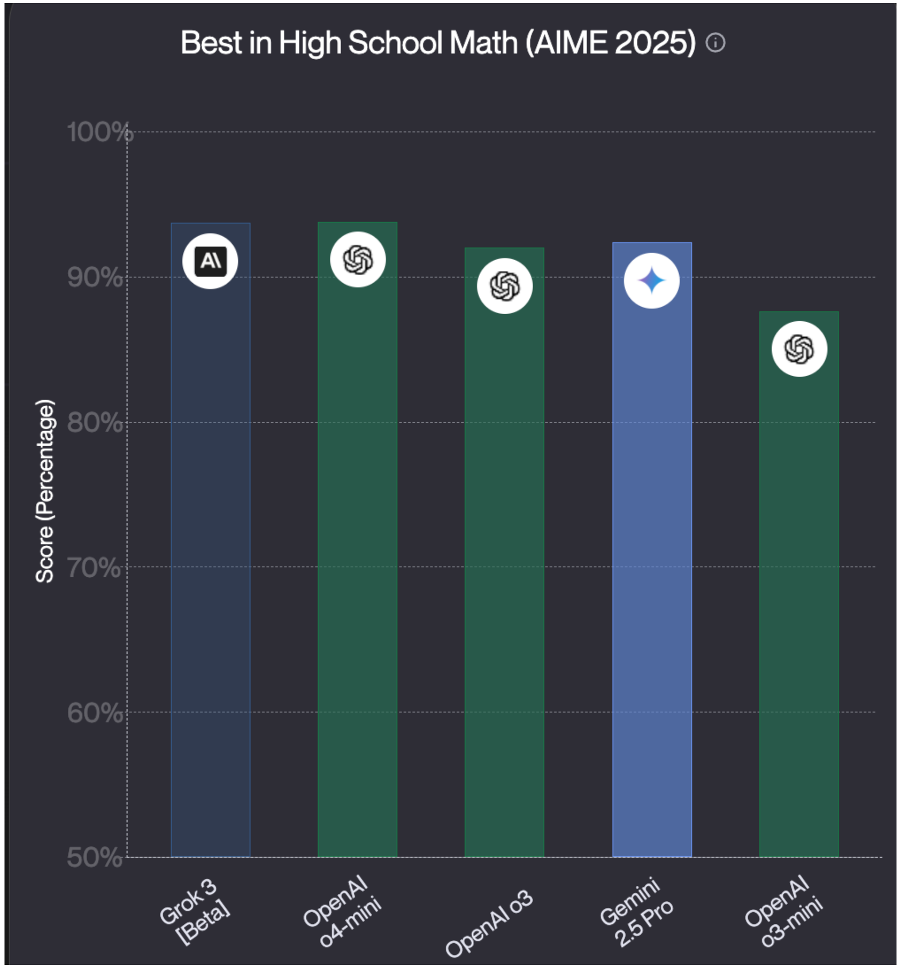
Dettmers et al., QLoRA: Efficient finetuning of quantized LLMs. NeurIPS, 36, 2023.

LLM Evaluation

Mathematical Reasoning – AIME 2025

Question: There is a collection of 25 indistinguishable white chips and 25 indistinguishable black chips. Find the number of ways to place some of these chips in the 25 unit cells of a 5×5 grid such that:

- each cell contains at most one chip all chips in the same row; and
- all chips in the same column have the same colour;
- any additional chip placed on the grid would violate one or more of the previous two conditions.



https://huggingface.co/datasets/Maxwell-Jia/AIME_2024

https://www.vellum.ai/llm-leaderboard

Humanity's Last Exam (HLE)

Mathematics

Question:
The set of natural transformations between two functors $F, G : \mathcal{C} \to \mathcal{D}$ can be expressed as the end

$$\mathrm{Nat}(F,G) \cong \int_A \mathrm{Hom}_{\mathcal{D}}(F(A), G(A)).$$

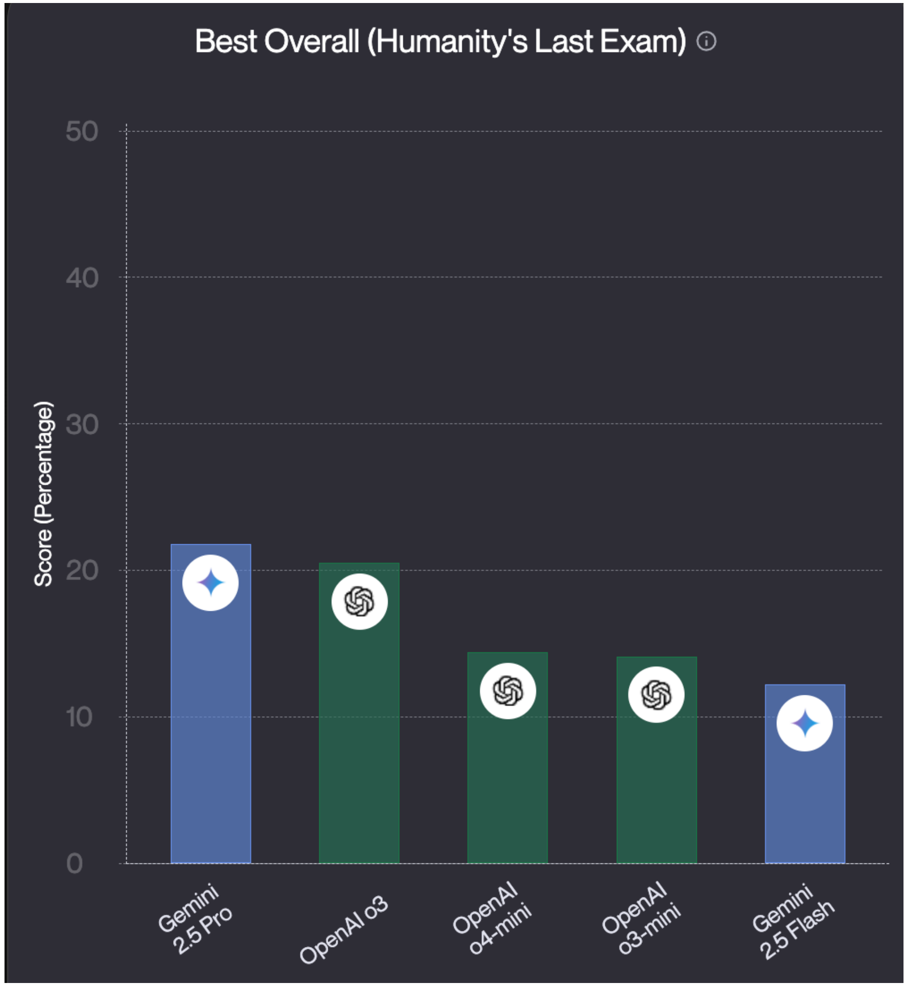
Define set of natural cotransformations from F to G to be the coend

$$\mathrm{CoNat}(F,G) \cong \int^A \mathrm{Hom}_{\mathcal{D}}(F(A), G(A)).$$

Let:
- $F = \mathbf{B}_\bullet(\Sigma_4)_{*/}$ be the under ∞-category of the nerve of the delooping of the symmetric group Σ_4 on 4 letters under the unique 0-simplex $*$ of $\mathbf{B}_\bullet\Sigma_4$.
- $G = \mathbf{B}_\bullet(\Sigma_7)_{*/}$ be the under ∞-category nerve of the delooping of the symmetric group Σ_7 on 7 letters under the unique 0-simplex $*$ of $\mathbf{B}_\bullet\Sigma_7$.

How many natural cotransformations are there between F and G?

https://agi.safe.ai/



https://www.vellum.ai/llm-leaderboard

ChatbotArena

Model A

What are the most critical challenges of LLM evaluation

Evaluating large language models (LLMs) is a complex and evolving challenge. Here are the most critical issues:

1. Lack of Standardized Benchmarks

- Many benchmarks exist (e.g., MMLU, BIG-bench), but no single standard covers all important aspects, especially for new or domain-specific tasks.
- Benchmarks can quickly become "solved" as models improve, making them less useful for differentiation.

2. Evaluation of Factuality and Hallucination

- LLMs frequently generate plausible-sounding but incorrect information ("hallucinations").
- Determining the factual accuracy of generated text is difficult, especially for open-ended or niche queries.

3. Measuring Reasoning and Complex Task Performance

- Many evaluations focus on simple QA or completion, but complex reasoning, multi-hop logic, and real-world problem-solving are harder to test and automate.

Model B

What are the most critical challenges of LLM evaluation

Evaluating Large Language Models (LLMs) is a complex and multifaceted task. Here are some of the most critical challenges associated with LLM evaluation:

1. **Lack of Standardised Metrics**
   - **Issue**: There is no universally accepted set of metrics for evaluating LLMs. Different researchers and organisations use different benchmarks and metrics, making it difficult to compare models.
   - **Impact**: This lack of standardisation hampers the ability to objectively assess and compare the performance of different LLMs.
2. **Benchmark Limitations**
   - **Issue**: Existing benchmarks often do not cover the full range of capabilities and use cases for LLMs. They may focus on specific tasks like question answering, summarisation, or language translation, but miss others.
   - **Impact**: Models may perform well on specific benchmarks but fail in real-world applications that require a broader set of skills.
3. **Bias and Fairness**
   - **Issue**: LLMs can inherit and even amplify biases present in their training data. Evaluating for bias and fairness is challenging because it requires identifying and mitigating biases across multiple dimensions (e.g., gender, race, ethnicity).

https://lmarena.ai

ChatbotArena LLM Leaderboard

Model (206 / 206)	Overall	Hard Prompts	Coding	Math	Creative Writing	Instruction Following	Longer Query	Multi-Turn
gemini-2.5-pro-…	1	1	1	1	1	1	1	1
gemini-2.5-pro-…	2	2	2	1	1	1	1	1
o3-2025-04-16	2	2	2	1	3	3	6	4
chatgpt-4o-late…	4	4	2	6	2	3	1	1
gpt-4.5-preview…	4	4	2	4	3	3	2	1
gemini-2.5-flas…	5	4	4	1	2	3	2	6
claude-opus-4-2…	6	2	2	3	2	3	1	3
gpt-4.1-2025-04…	8	4	5	14	6	7	2	5
gemini-2.5-flas…	8	7	12	5	6	3	7	6
grok-3-preview-…	8	7	5	11	7	9	5	7
claude-sonnet-4…	9	8	3	6	7	7	6	5
o4-mini-2025-04…	9	8	7	1	14	14	15	10
deepseek-v3-0324	10	8	5	14	7	10	8	5

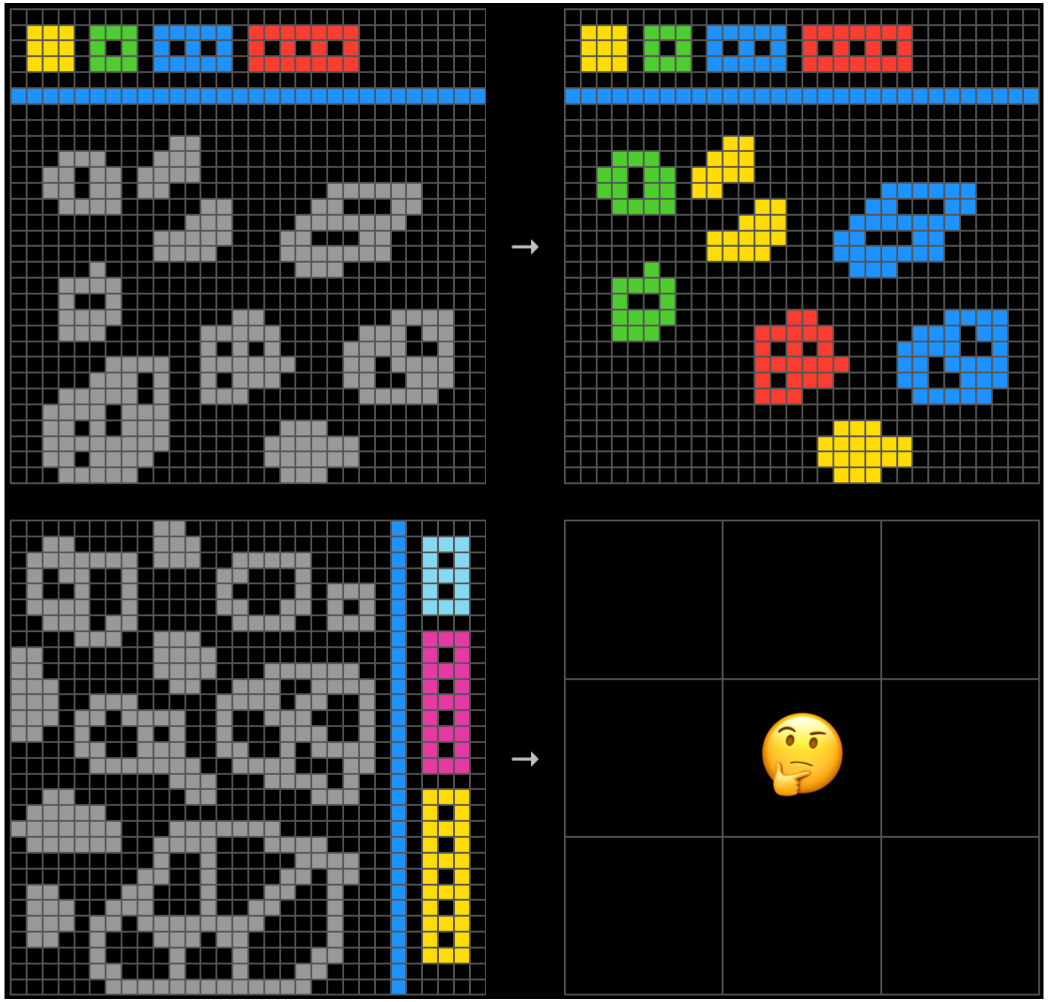
https://lmarena.ai/?leaderboard

ARC-AGI-2 – A Next-Gen Reasoning Benchmark

Evaluate the ***efficiency*** and ***capability*** of state-of-the-art AI reasoning systems.

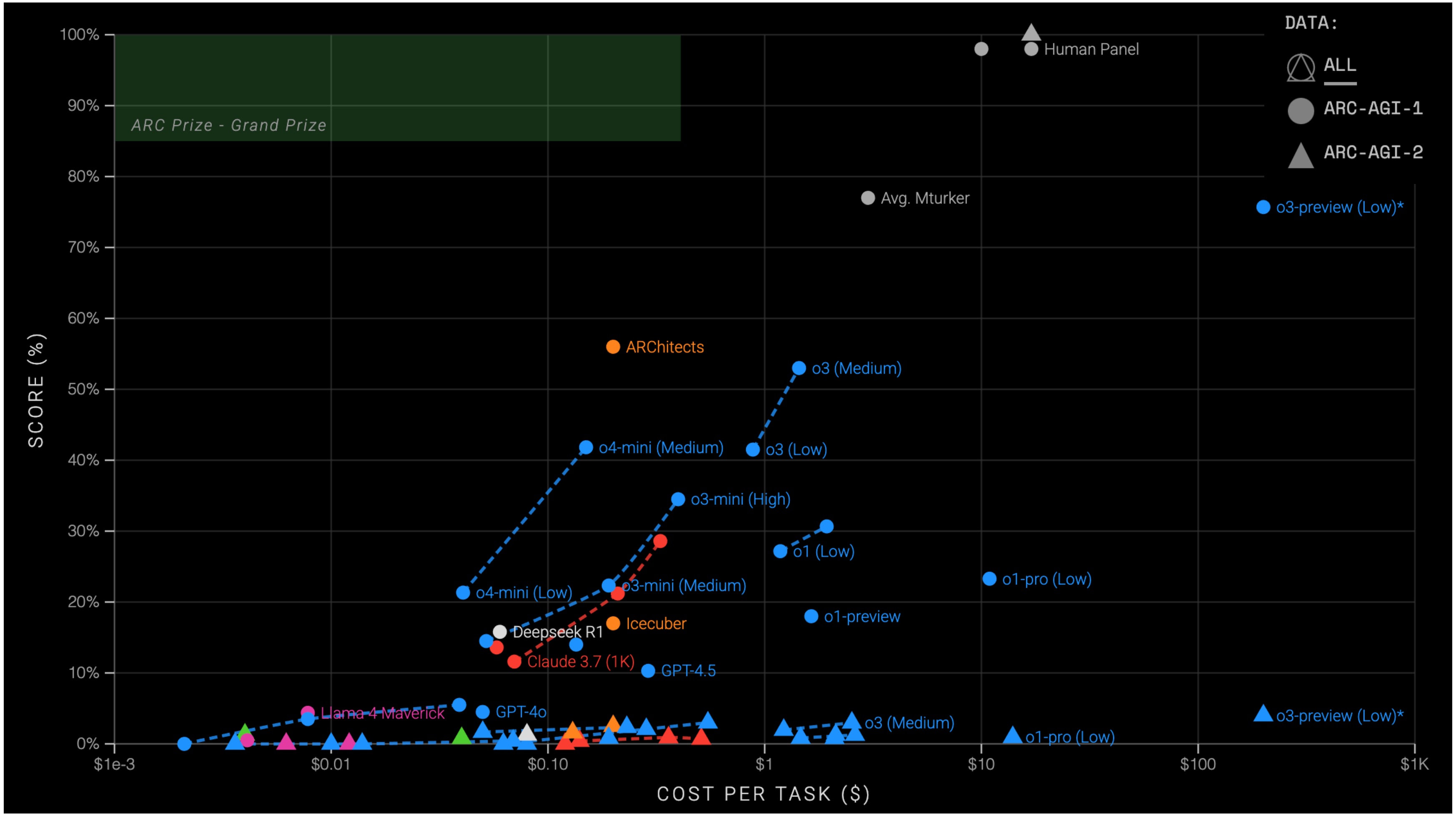
Key Features:

- Multi-step, **abstract reasoning** tasks
- Real-world inspired challenges
- Minimal reliance on superficial cues



https://arcprize.org/

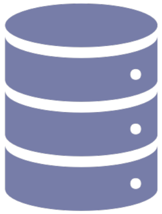
ARC-AGI Leaderboard



https://arcprize.org/

Foundation of LLM Evaluation

What to evaluate?
Evaluation Tasks



Where to evaluate?
Evaluation Benchmarks

How to evaluation?
Evaluation Process

Chang et al., A survey on evaluation of large language models. ACM Transactions on Intelligent Systems and Technology, 2024.

Evaluation Tasks

Task	Details
Language Understanding	• Reading Comprehension, Natural Language Inference (NLI), Summarization, Coreference Resolution, Sentiment Analysis • ***Example Benchmarks***: *GLUE, SuperGLUE, C-Eval*
Knowledge and Reasoning	• General Knowledge, Subject-Specific Knowledge • Common-Sense Reasoning, Mathematical Reasoning... • Fact Verification • ***Example Benchmarks***: *MMLU, BIG-bench, FEVER*
Dialogue and Interaction	• Instruction Following • Helpfulness, Harmlessness, Honesty (HHH) • Dialogue Coherence and Engagement • ***Example Benchmarks***: *MT-Bench, Chatbot Arena, AlpacaEval*
Safety and Robustness	• Toxicity Detection, Bias and Fairness Testing • Value Alignment • Adversarial Robustness • ***Example Benchmarks***: *SafetyBench, TRUSTGPT, AdvBench*

Evaluation Tasks

Multimodal Understanding

- Image + Text Reasoning
- Visual Question-Answering
- Chart/Table Reasoning
- ***Example Benchmarks***: *MMBench, SEED-Bench, MMMU*

Specialised Abilities

- Theory of Mind (ToM) Reasoning
- Emotion Understanding
- Ethical and Moral Reasoning
- Tool Use (API Calls, Planning)
- ***Example Benchmarks***: *ToMi, EmotionBench, API-Bank*

Out-of-Distribution (OOD) and Robustness

- Generalisation to Unseen Data
- Domain Transfer
- Prompt Robustness
- ***Example Benchmarks***: *GLUE-X, BOSS, PromptBench*

Evaluation Benchmarks

General benchmarks

MMLU, C-Eval, OpenLLM, DynaBench, AlpacaEval, HELM, Chatbot Arena, MT-Bench, BIG-bench, PandaLM, BOSS, GLUE-X, KoLA, AGIEval, PromptBench,, LLMEval, GAOKAO-Bench

Specific benchmarks

SOCKET, Choice-75, CUAD, TRUSTGPT, MATH, APPS, CELLO, EmotionBench, CMMLU, API-Bank, M3KE, UHGEval, ARB, MultiMedQA, CVALUES, CMB, MINT, Dialogue CoT, SafetyBench

Multi-modal benchmarks

MMBench, SEED-Bench, M3Exam, ToolBench, MathVista, MM-Vet, LAMM, LVLM-eHub

Chang et al., A survey on evaluation of large language models. ACM Transactions on Intelligent Systems and Technology, 2024.

General Benchmarks

Benchmark	Focus	Notes
MMLU	Multitask knowledge and reasoning	Covers 57 subjects, 15,908 MCQs.
BIG-bench	Diverse task challenges	200+ tasks, multi-domain.
HELM	Holistic performance (accuracy, fairness)	Multi-dimensional evaluation.
OpenLLM	Public model competitions	Leaderboard-style comparisons.
MT-Bench	Multi-turn dialogue	Becoming a general conversational test.
AGIEval	Standardised exam reasoning	SAT, GRE, LSAT-style tasks.
AlpacaEval	Automated NLP task evaluation	Focus on robustness and diversity.
C-Eval	Chinese academic exams (52 subjects)	Big for multilingual/global benchmarks.
GAOKAO-Bench	Advanced reasoning (Gaokao exams)	Very difficult knowledge/reasoning test.
PromptBench	Prompt engineering evaluation	Measures prompt adaptability.
PandaLM	Subjective qualities (clarity, formality)	Human-like model scoring.

Specific Benchmarks

Domain	Benchmark	Focus	Notes
Medical	MultiMedQA	Medical exam QA	Highly specialised in healthcare knowledge.
Law	CUAD	Legal contract review	Extracting and understanding clauses.
Science	ChemBench	scientific reasoning and problem-solving across chemistry subfields.	Evaluate LLMs' ability to understand, reason, and apply knowledge in chemistry.
Emotion	EmotionBench	Understanding and recognising emotions	Focused on emotional intelligence in dialogue.
Theory of Mind (ToM)	OpenToM	Some tasks measure ToM reasoning	Designed tasks simulate ToM scenarios.
Knowledge Reasoning	KoLA	Semantic knowledge inference	Deep reasoning based on general knowledge.
Safety	SafetyBench	Toxicity, bias, adversarial robustness	Evaluates safety issues like bias and toxicity.
Robustness	DynaBench	Adversarial robustness, closed-loop systems	Evaluates performance in real-time, adversarial settings.
Value alignment	TRUSTGPT	Ethics, bias, and value alignment	Evaluates ethical responses and value consistency.

Multimodal Benchmark

Benchmark	Focus	Modalities	Notes
LVLM-eHub	Evaluation of large vision-language models (LVLMs)	Text + Vision (Images)	Targets the integration of vision and language understanding.
MMBench	Visual QA, image understanding, scene reasoning, chart/table interpretation.	Text + science diagrams, infographics, natural scenes	Answering questions based on photos, diagrams, charts, tables, and screenshots.
ToolBench	Multimodal task performance (tools, reasoning)	Text + Images + Other tools (APIs)	Evaluates models on using tools and reasoning with multiple types of input.
VQAv2 (Visual QA)	Visual reasoning via question answering	Text + Images	Tests model performance in answering questions based on images.
GQA	Visual question answering with reasoning	Text + Images	Focuses on reasoning through visual contexts, particularly for logical problem-solving with images.
M3Exam	Multimodal, Multiturn, Multilevel Examination Benchmark	Text + image, tables/graphs	Simulates real-world examination scenarios where multi-step reasoning is needed.
ScienceQA	Science reasoning with text, diagrams	Text + images, diagrams, tables	Especially used for science-based multimodal reasoning.
MathVista	Math + visual understanding	Text + diagrams, graphs, shapes	Combination of visual math reasoning.

Evaluation Process

Automatic evaluation

Accuracy: Exact match, Quasi-exact match, F1 score, ROUGE score
Calibrations: Expected calibration error, Area under the curve
Fairness: Demographic parity difference, Equalised odds difference
Robustness: Attack success rate, Performance drop rate

Human evaluation

Expert assessment rates outputs on dimensions like *accuracy*, *relevance*, and *helpfulness*.

Crowdsourced Evaluation gathers judgments from multiple non-expert evaluators.

Comparative Evaluation presents evaluators with multiple model outputs to rank or choose between.

LLM-as-a-Judge

Single Model Judging uses a strong LLM to evaluate other model outputs.

Multi-Model Consensus employs multiple LLMs as judges and aggregates their scores.

Constitutional AI Evaluation trains models specifically for evaluation tasks .

Evaluation Metrics

1. **Accuracy-Based Metrics**
   - **Exact Match (EM):** % of answers that exactly match the ground truth (used in QA like SQuAD, GSM8K).
   - **Top-k Accuracy:** Whether the correct answer appears in the top *k* predictions.
   - **Pass@k:** Used in generation tasks – likelihood of generating a correct solution in *k* attempts.
2. **Text Overlap Metrics**
   - **BLEU / ROUGE / METEOR**
   - Measure **n-gram overlap** between model output and reference texts.
3. **Semantic Similarity Metrics**
   - **BERTScore, Natural Language Inference (NLI) score**
   - Uses contextual embeddings (e.g., via BERT) to compare **semantic similarity** between generated and reference texts.

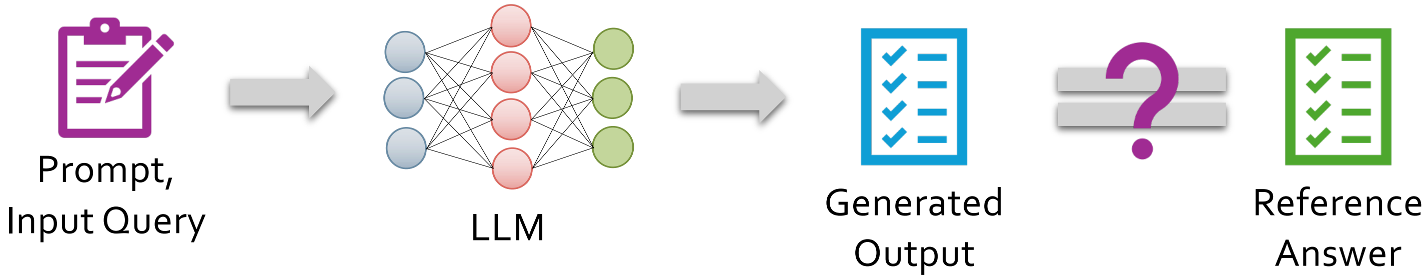
5. **Log-Likelihood / Perplexity**
   - Measures how well the model predicts tokens in a dataset.
   - Common in **pretraining evaluation**, less reliable for downstream task performance.

4. **Win Rate (Arena-Style Comparisons)**
   - **Win Rate:** % of times a model wins in head-to-head matchups.

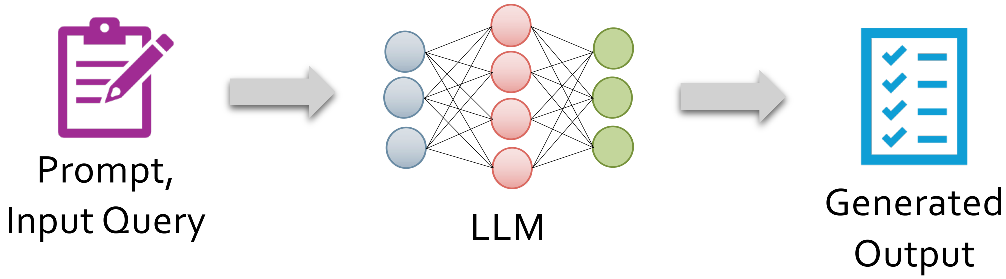
6. **Human Evaluation**
   - Evaluators judge model outputs for:
     - **Helpfulness**
     - **Honesty**
     - **Factuality**
     - **Reasoning quality**
     - **Harmlessness**
     - **…**

Close-Ended vs. Open-Ended Evaluation

- **Close-ended evaluation**



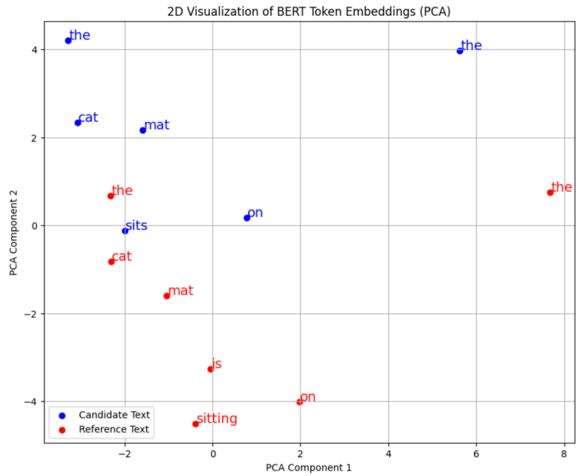
- **Open-ended evaluation**



Close-Ended Evaluation

Candidate Text: The cat sits on the mat.

Reference Text: The cat is sitting on the mat.



Text Overlap Metrics

(e.g., BLEU, ROUGE, METEOR, etc.)

Semantic Similarity Metrics

(e.g., BERTScore, SentenceBERT, BLUERT)

Close-Ended Evaluation

- **Text Overlap Metrics**
  - **Exact Match Accuracy**
  - **Token-Level F1** (Partial token-level overlap between generated and golden answer)
  - **BLEU (Bilingual Evaluation Understudy)**
    - Calculates the **precision** for **each n-gram level**, i.e., the proportion of n-grams in the candidate text that appears in the reference texts.
  - **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**
    - Focuses on **recall-based** evaluation by comparing n-grams, word sequences, and word pairs.
    - **ROUGE-N** (n-gram overlap), **ROUGE-L** (longest common subsequence).
  - **METEOR (Metric for Evaluation of Translation with Explicit ORdering)**
    - Handles **synonyms** and **word-order variations** to improve upon BLEU's limitations.

Candidate Text: The cat sits on the mat.

Reference Text: The cat is sitting on the mat.

Exact Match: 0
Token-F1: Precision: 5/6, Recall: 5/7, F1: 0.77
BLEU: 0.42 (precision-focused, considering n-gram overlap)
ROUGE-1: 0.77 (recall-focused, unigram overlap)
ROUGE-L: 0.77 (longest common subsequence)
METEOR: 0.88 (accounts for precision, recall, synonyms, and word order)

Problem: Ignore semantic similarity between the reference and candidate text.

Close-Ended Evaluation

- Semantic Similarity Metrics
  - **BERTScore**
    - Compares **token embeddings** from a pretrained model like BERT; matches each token in the generated text to the most similar token in the reference.
  - **SentenceBERT**
    - Encodes full sentences and measures **cosine similarit**y between them.
  - **BLUERT**
    - Trains a model to predict human evaluation scores based on **embeddings**; fine-tuned specifically for quality evaluation.

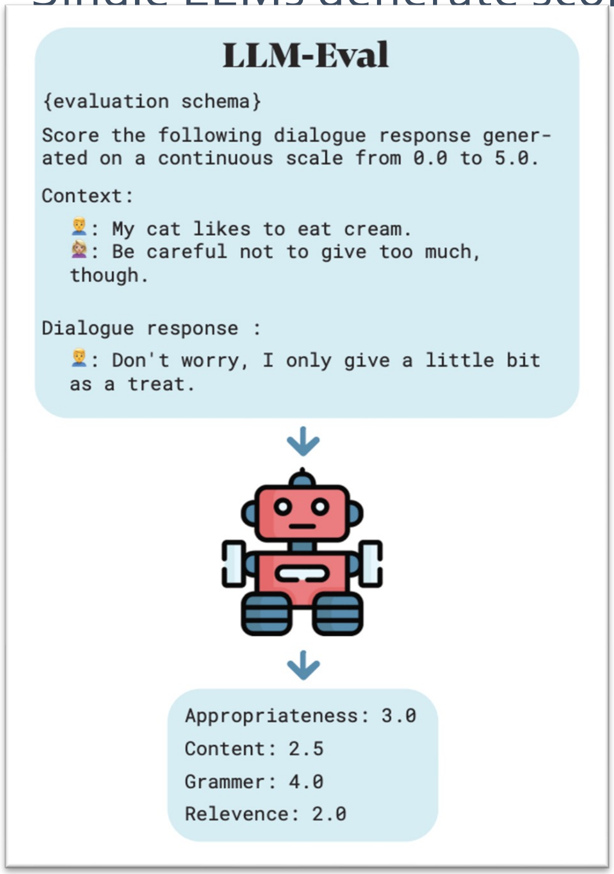
Open-Ended Evaluation

- **No single correct answer**
- **Multiple plausible outputs** can exist
- Focus on evaluating **fluency, coherence, relevance, factuality**, etc.
- **Human judgment** often needed
  - Costly, sometimes inconsistent
- **LLM-as-a-judge**
  - Fast and scalable; Can follow complex evaluation rubrics; Correlates well with human judgment in many cases.
  - Vulnerable if the judging prompt is poorly designed; May reflect training data biases

Example Tasks	Description
Story Writing	Write a short story about space travel
Summarisation	Summarise a news article
Dialogue Response	Continue a conversation naturally
Code Generation	Solve a programming task with multiple valid solutions

Single Model Judging – LLM-EVAL

- Single LLMs generate score for different evaluation dimensions (LLM-EVAL)

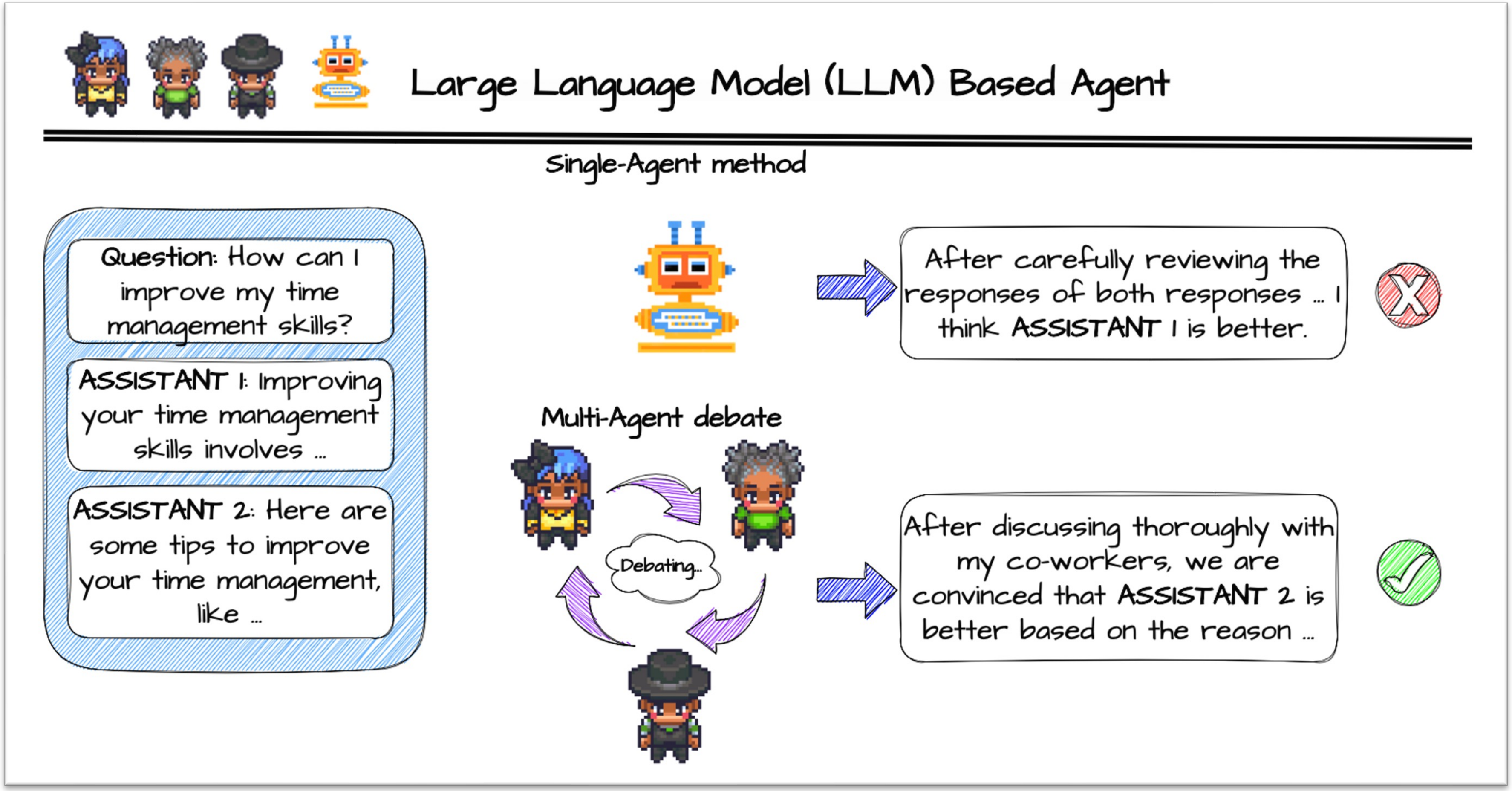


Observations:

- Different scoring ranges, e.g., 0-5, and 0-100
  - Similar performance, overall better than other baselines.
- Different LLMs matter
  - Claude and ChatGPT generally achieve better performance across all dimensions when compared to GPT-3.5.
- Different decoding strategies
  - Greedy decoding generally achieves better performance across all evaluation dimensions.

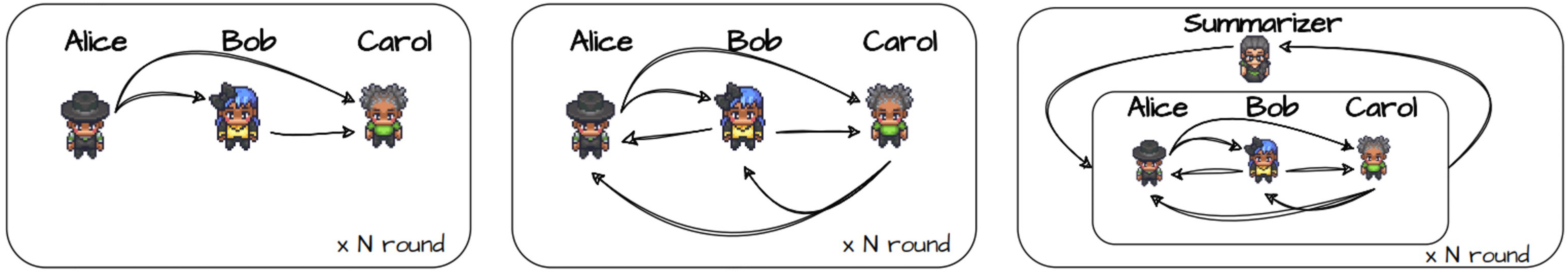
Lin and Chen. LLM-EVAL: Unified Multi-Dimensional Automatic Evaluation for Open-Domain Conversations with Large Language Models. NLP4ConvAI 2023

Multi-Model Consensus – ChatEval



Chan et al., ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate. ICLR 2024.

Multi-Model Consensus – ChatEval



(a) One-by-One (b) Simultaneous-Talk (c) Simultaneous-Talk-with-Summarizer

a) The debater agents **take turns** in a set order to generate their response.
b) The debater agents are prompted to **asynchronously** generate responses.
c) Additionally employ another LLM as a **summarizer** and concatenate this summarization into all debater agents' chat history slots.

Chan et al., ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate. ICLR 2024.

Constitutional AI Evaluation – Prometheus

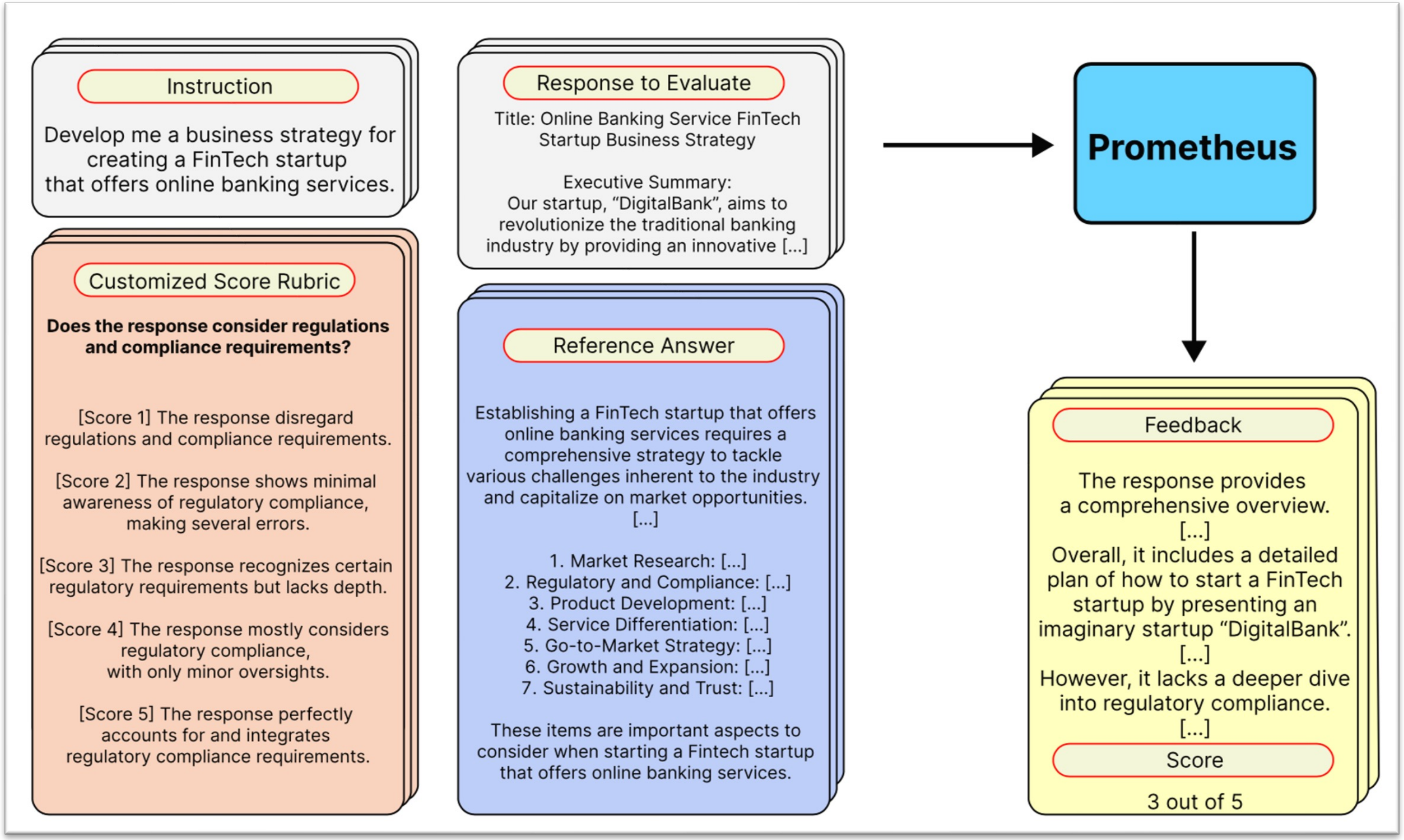
- **Problems** of using proprietary **LLMs** as **an evaluation tool**:
  - A lack of transparency
  - Uncontrolled versioning
  - Prohibitive costs

- PROMETHEUS
  - a 13B LM that aims to induce **fine-grained evaluation capability** of GPT-4, while being open-source, reproducible, and inexpensive.

Kim et al., Prometheus: Inducing Fine-Grained Evaluation Capability in Language Models. ICLR 2024.

Prometheus

- An open-source LM evaluator trained on a dataset containing feedback collections.



Kim et al., [Prometheus: Inducing Fine-Grained Evaluation Capability in Language Models](). ICLR 2024.

Goodhart's Law

"When a measure becomes a target, it ceases to be a good measure."

- *When systems are evaluated based on a specific metric, they often start optimising for that metric directly.*
- *As a result, the metric no longer accurately reflects what it was originally intended to measure.*

- **AI model evaluations:** If a language model is ***optimised to win leaderboard rankings***, it may ***overfit to benchmark tasks*** rather than ***improve general reasoning***.

Put Evaluation into Practice

1

Choose an appropriate benchmark for a given LLM task or domain, justifying the choice against alternatives.

2

Design a small-scale evaluation experiment – select prompts, sampling strategy, and rating protocol that align with study goals.

3

Compute and interpret key metrics (e.g., BERTScore, Win-rate, Pass@k) and articulate their limitations.

4

Critically assess evaluation results – spot statistical noise, annotation bias, or benchmark leakage that may invalidate conclusions.

Interim Summary

- The Transformer architecture
  - Transformer basics – self-attention layer, encoder input, complete encoder, Transformer decoder
  - Improvement on Transformer – Rotary Position Embedding (RoPE)

- Language models built on Transformer
  - Encoder-only models – BERT
  - Encoder-decoder models – T5
  - Decoder-only models – GPT-x
  - Mixture of Experts models – Mixtral 8x7B

Interim Summary

- LLM training paradigms
  - Learning task-specific models
  - Pre-training and then fine-tuning
  - In-context learning
  - Instruction tuning
  - Alignment tuning

- Parameter-efficient fine-tuning
  - Adapter tuning, Prefix tuning, Prompt Tuning, LoRA, QLoRA

- LLM evaluation
  - What to evaluate? **Evaluation Tasks**
  - Where to evaluate? **Evaluation Benchmarks**
  - How to evaluation? **Evaluation Process**

Email: yulan.he@kcl.ac.uk

Twitter: @yulanhe

KCL NATURAL LANGUAGE PROCESSING | NLP

https://kclnlp.github.io/