Deep Learning
lecture 9
Sequence Modeling (2)

Yi Wu, IIIS

Spring 2024

Apr-22

Logistics

- Lecture 10 is happening on Apr 26 (Friday) as usual

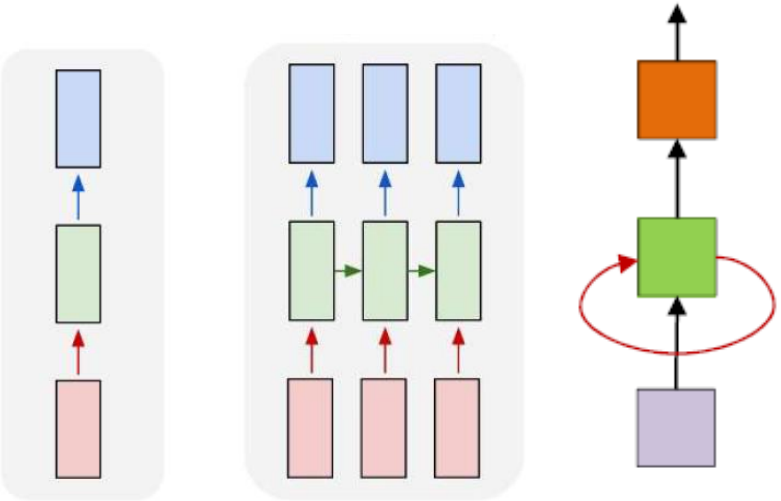
- Coding Project 3 is due this Sunday (Apr. 28)!
- Coding Project 4 will be released later this week.

- Final project proposal instruction will be released tonight!
  - Proposal due on Apr. 29.
    - Topics: AIGC, LM, AI4Science (application driven)
  - Team work (2~3 members encouraged)
  - A very concise 1-page report
    - you need to precisely describe what you are going to do!
    - The goal is to make sure you are on track

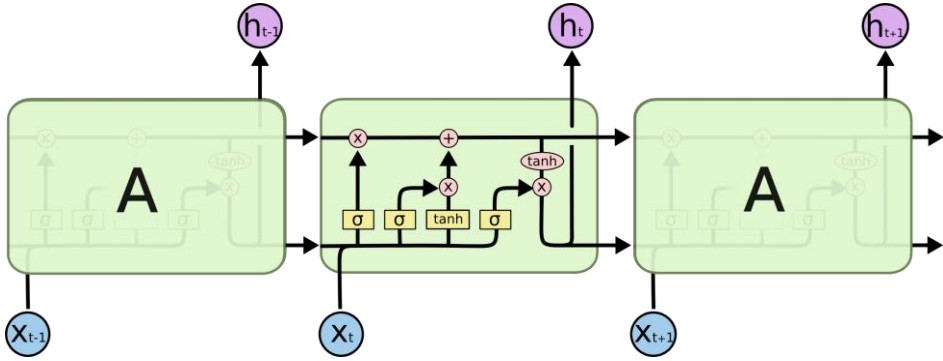
Today’s Topic

- More Techniques for Language Models
- Sequence to Sequence Model and Attention Mechanism
- Modern Language Models and Applications

RNN Recap

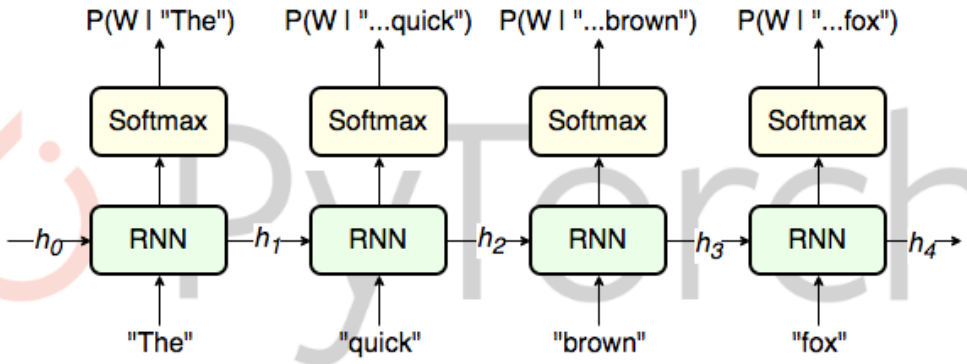


- Recurrent Neural Network
  - Same MLP network over a sequence (i.e., “loops”)
    - Arbitrarily long sequences → fixed-sized vector
  - Training: backpropagation through time (BPTT)
  - Practical Issues
    - Weights/Gradient explosion and saturation
  - A few tricks for gradient explosion
    - Gradient clipping, truncated BPTT, careful initialization

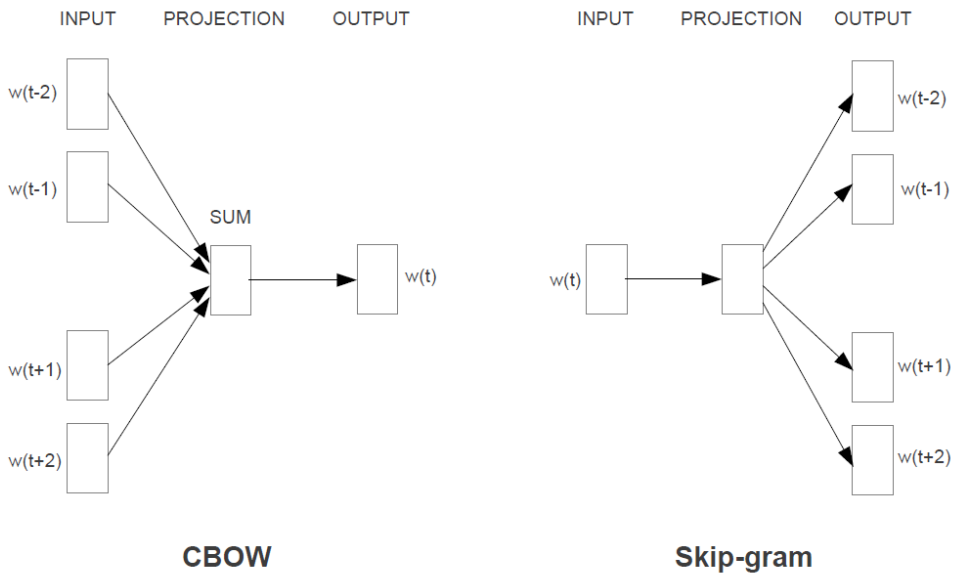


- Long Short-Term Memory (LSTM) Network
  - A specialized RNN for long-term dependency (~100 timesteps)
    - Key ideas: elementary gates
  - Variants: bidirectional LSTM; Peephole LSTM; GRU; etc

RNN Recap

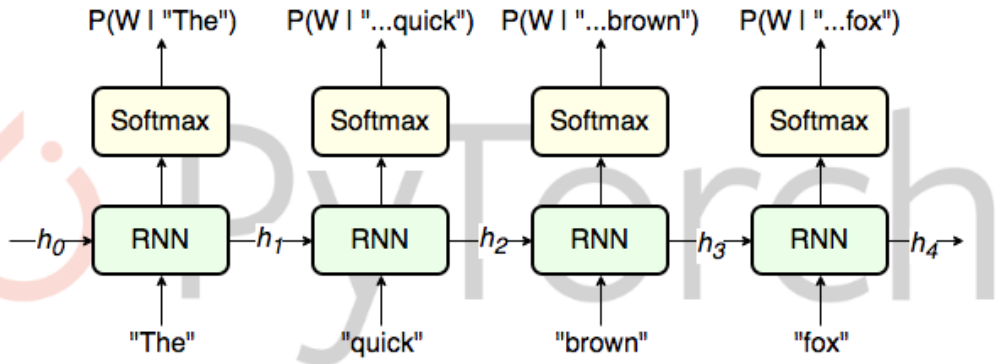


- Autoregressive Language Model
  - Generative model over texts: $P(X) = \prod_t P(X_t | X_{i<t})$
    - LSTM language model: $Y_t, h_t = LSTM(h_{t-1}, X_t)$; $P(X_t | X_{i<t}) = \text{Softmax}(Y_t)$
  - Word Embedding
    - A distributed representation for word semantics
- Word2Vec: a tool for word embedding
  - Objective: from context c to predict word w
    - CBOW and Skip-Gram
    - Negative Sampling
      - Multi-class prediction → binary classification
      - D training corpus; V vocabulary

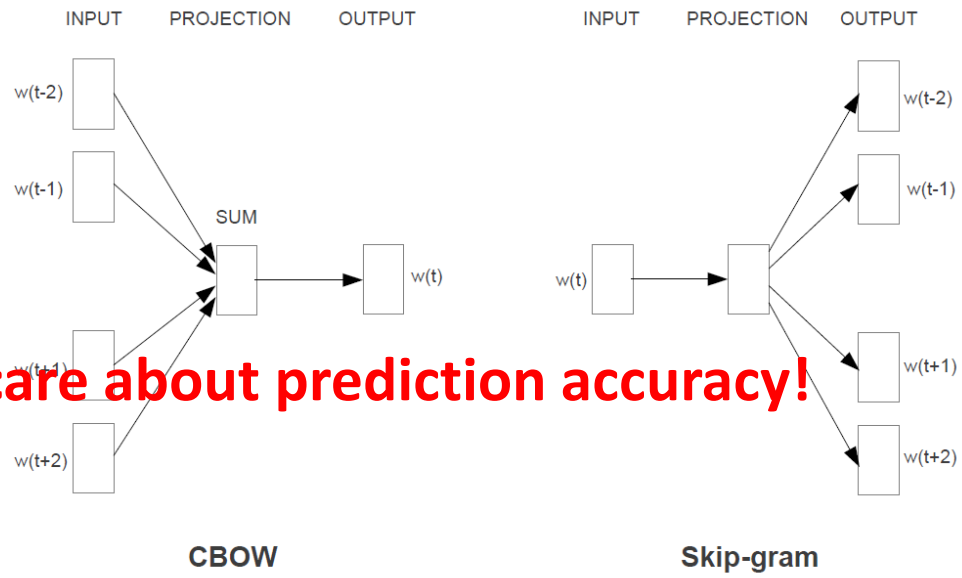


$$L(W, C) = \sum_{(c,w)\in D} \log \frac{1}{\exp(-w^T c) + 1} + \sum_{c\in D, \tilde{w}\sim V} \log \frac{\exp(-\tilde{w}^T c)}{\exp(-\tilde{w}^T c) + 1}$$

RNN Recap



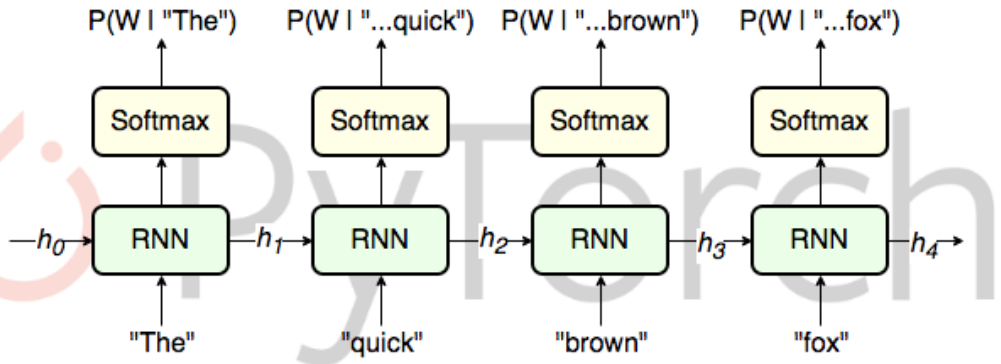
- Autoregressive Language Model
  - Generative model over texts: $P(X) = \prod_t P(X_t | X_{i<t})$
    - LSTM language model: $Y_t, h_t = LSTM(h_{t-1}, X_t)$; $P(X_t | X_{i<t}) = \text{Softmax}(Y_t)$
  - Word Embedding
    - A distributed representation for word semantics
- Word2Vec: a tool for word embedding
  - Objective: from context c to predict word w
    - CBOW and Skip-Gram
    - Negative Sampling
      - Multi-class prediction → binary classification
      - D training corpus; V vocabulary



Word2Vec does not care about prediction accuracy!

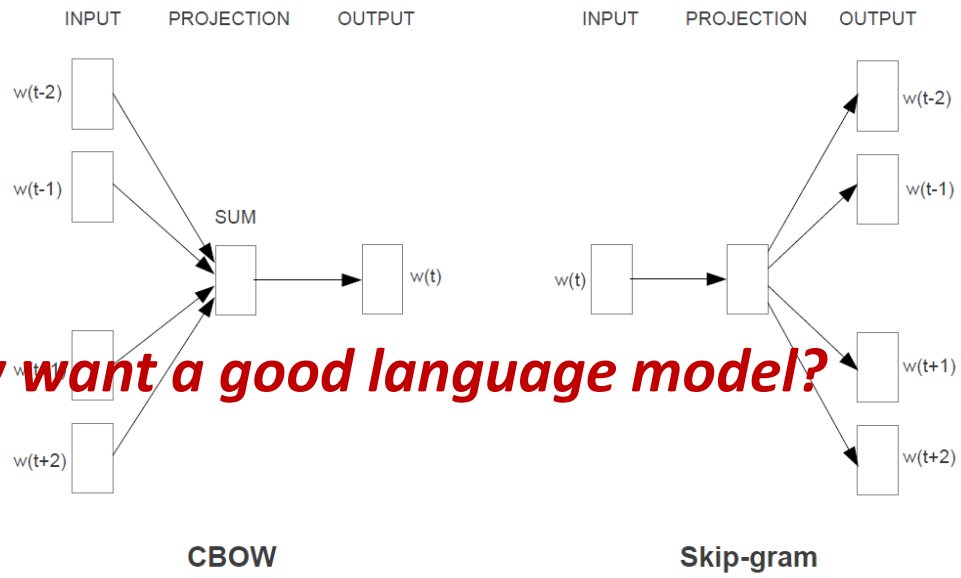
$$L(W, C) = \sum_{(c,w) \in D} \log \frac{1}{\exp(-w^T c) + 1} + \sum_{c \in D, \tilde{w} \sim V} \log \frac{\exp(-\tilde{w}^T c)}{\exp(-\tilde{w}^T c) + 1}$$

RNN Recap



- Autoregressive Language Model
  - Generative model over texts: $P(X) = \prod_t P(X_t | X_{i<t})$
    - LSTM language model: $Y_t, h_t = LSTM(h_{t-1}, X_t)$; $P(X_t | X_{i<t}) = \text{Softmax}(Y_t)$
  - Word Embedding
    - A distributed representation for word semantics
- Word2Vec: a tool for word embedding
  - Objective: from context c to predict word w
    - CBOW and Skip-Gram
    - Negative Sampling
      - Multi-class prediction → binary classification
      - D training corpus; V vocabulary

$$L(W, C) = \sum_{(c,w)\in D} \log \frac{1}{\exp(-w^T c) + 1} + \sum_{c\in D, \widetilde{w}\sim V} \log \frac{\exp(-\widetilde{w}^T c)}{\exp(-\widetilde{w}^T c) + 1}$$



What if we really want a good language model?

Advanced Techniques

- Language Model Learning
  - NCE loss: noise contrastive loss
    - Sampling-based gradient approximation
  - Hierarchical Softmax
    - Non-sampling, tree-based probability computation
  - Remark
    - Use full softmax when possible for the best performance (GPU memory allowed)

Advanced Techniques

- Language Model Inference
  - Goal: find $X^\star = \arg\max_X P(X) = \prod_t P(X_t | X_{i<t})$
  - Greedy Solution:
    - For each t, $X_t^\star = \arg\max_{X_t} P(X_t | X_{i<t}^\star)$ (i.e., keep the best partial candidate)
  - Better Solution: Beam Search
    - Idea: keep top K candidates for each t
    - When to terminate (sequences may have varying lengths)?
      - We typically include a <end> token to indicate a text sequence is ended
      - $L_{\max}$ words reached or n completed sequences obtained (<end> token produced)
    - Which sequence to choose?
      - Issue: longer sequences tend to have lower scores!
      - Adjusted metric: $X^\star = \arg\max_X \frac{1}{L_X} \sum_t \log P(X_t | X_{i<t})$ (normalized by its length)

Advanced Techniques

- Language Model Learning
  - The expensive softmax operator
- Language Model Inference
  - Beam search for the best generated sequence
    - You can also include a temperature parameter in score if you want diverse texts
- Improving the word representation
  - So far, we assume a static (pretrained) embedding
  - Issue: the same word in different contexts may have different meaning
    - Teddy **bear** v.s. I cannot **bear** him any more
    - A nice **weather** v.s. I'm under the **weather** today
    - **苟**富贵，毋相忘 v.s. **苟**全性命于乱世
  - Word embeddings should be context-aware!

Advanced Techniques

- Deep contextualized word representations (EMNLP2018)
  - Idea: a word feature should be related to the whole contexts
    - Including both previous words and future words
  - ELMo
    - (Optional) Use word2vec to pretrain static word embeddings w_s
    - Train a (stacked) bidirectional RNN language model $g_f(w, h)$ and $g_b(w, h)$ use w_s
    - Fix the RNN model g_f and g_b
    - For a sequence for a specific task, for the t-th word
      - Run g_f and g_b on the sentence to get h_t^f and h_t^b
      - Use $[w_t, h_t^f, h_t^b]$ as embedding
  - Remark:
    - Bidirectional LSTM is critical!

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Advanced Techniques

- Language Model Learning
  - The expensive softmax operator
- Language Model Inference
  - Beam search for the best generated sequence
    - You can also include a temperature parameter in score if you want diverse texts
- Contextualized Word Embedding
  - ELMo: use contexts to compute features of a word

- More techniques in your NLP course ☺

LSTM Applications

- Text Classification
  - Supervised learning
- Text Generation
  - $p(X;\theta)$: the probability for X
  - Unconditioned Generation
    - E.g., AI作诗
  - Conditioned generation?
    - E.g., Machine translation

Machine Translation

- A task of translating a sentence from a source language to the target language

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

↓

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

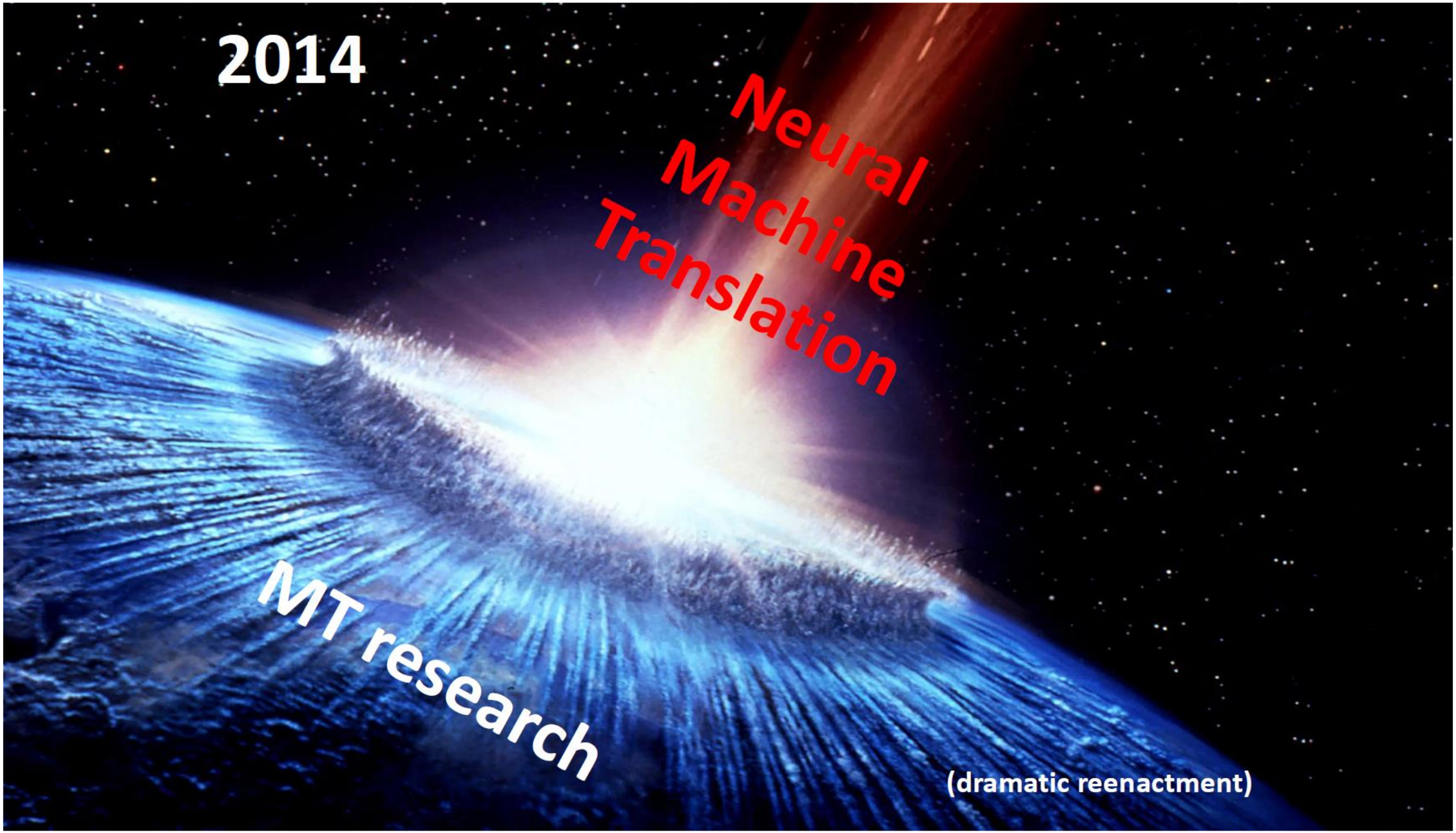
人生而自由，却无往不在枷锁之中。——卢梭《社会契约论》 – Rousseau

Machine Translation

- Before 2014: Statistical Machine Translation
  - Extremely complex systems that require massive human efforts
  - Separately designed components
  - A lot of feature engineering
  - Lots of linguistics domain knowledge and expertise
- Before 2016:
  - Google’s commercial translation product is based on statistical machine translation
- What happened in 2014?
  - A borrowed slide from Stanford CS224

2014

(dramatic reenactment)

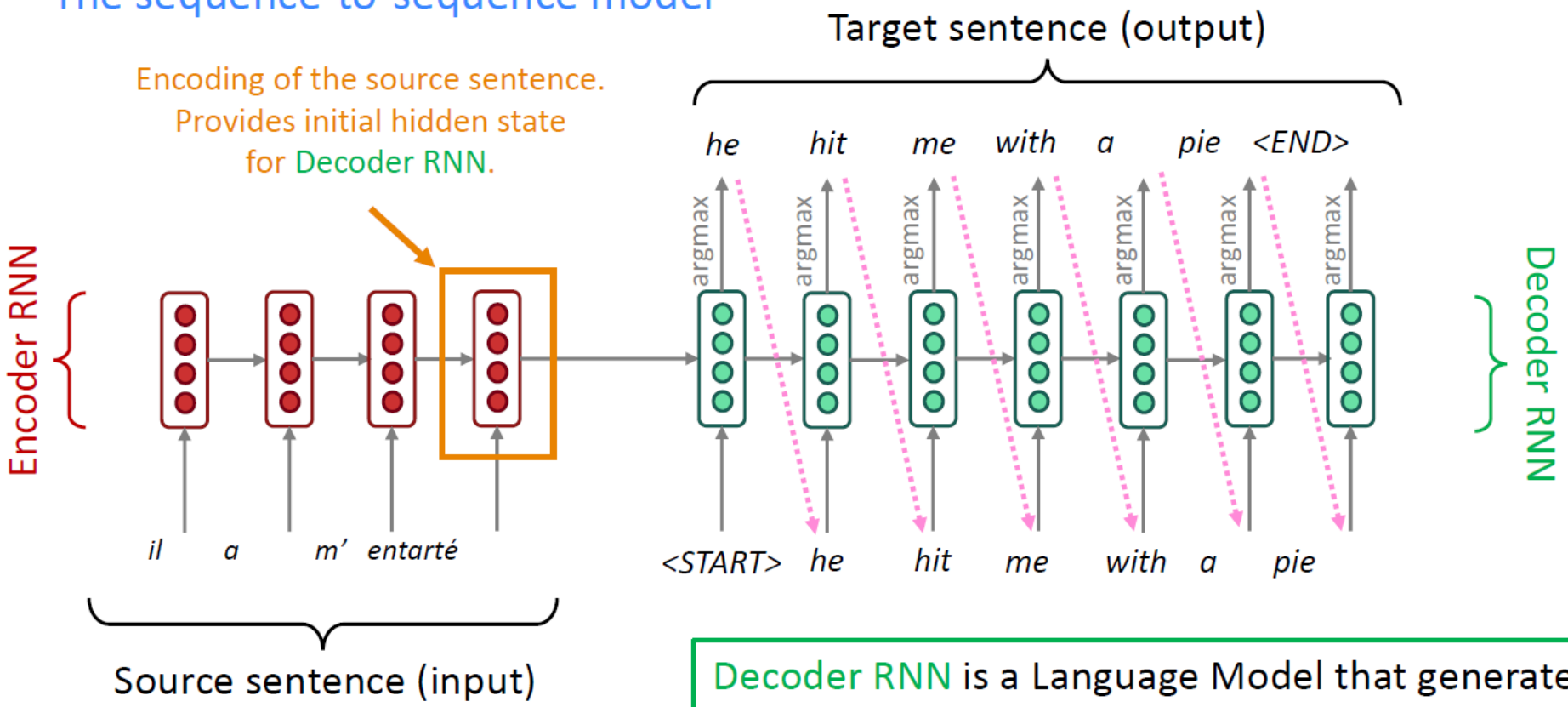

2014
Neural
Machine
Translation
MT research
(dramatic reenactment)

Sequence to Sequence Model

- Neural Machine Translation (NMT)
  - Learning to translate via single end-to-end neural network!
  - Source language X, then $Y = f(X; \theta)$
- Sequence to Sequence Model (Seq2Seq, Sutskever et al, NIPS2014)
  - Two RNNs: f_{enc} and f_{dec}
  - Encoder f_{enc}
    - It takes in X, and produce the initial hidden state for decoder
    - We can use bidirectional RNN
  - Decoder f_{dec}
    - It takes in the hidden state from f_{enc} to generate Y
    - Autoregressive language model

Sequence to Sequence Model

The sequence-to-sequence model



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

Encoder RNN produces an encoding of the source sentence.

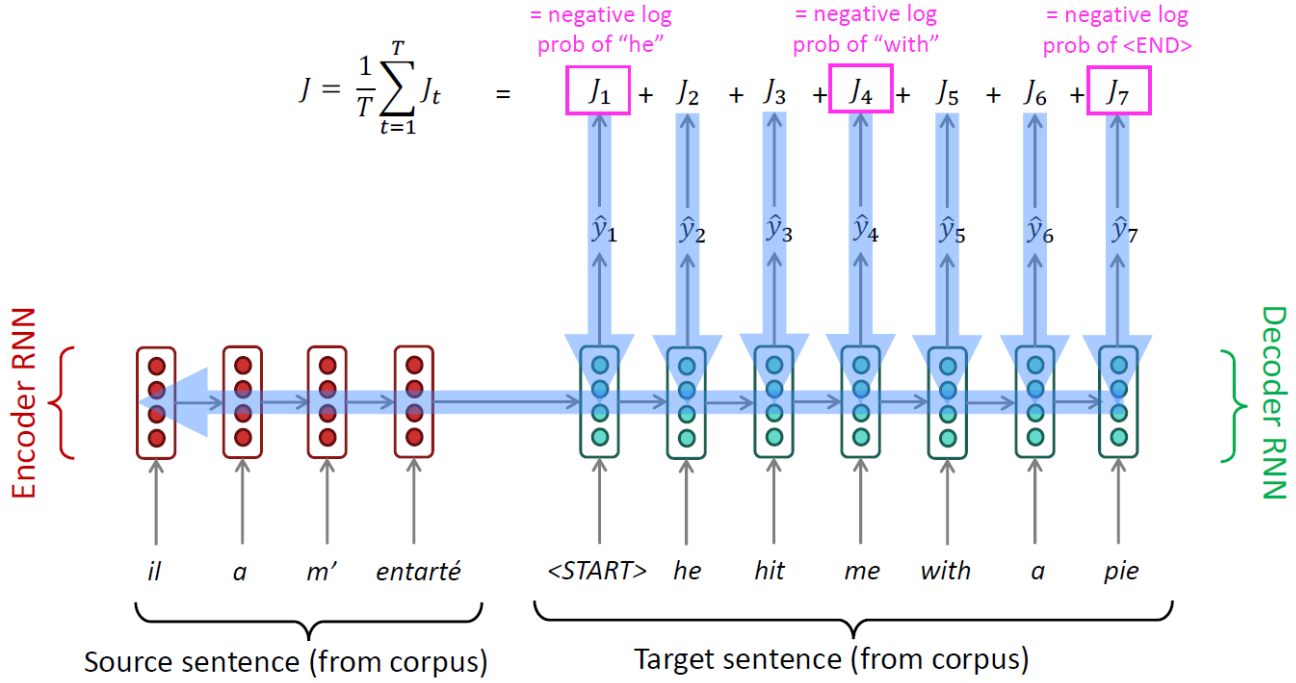
Note: This diagram shows **test time** behavior: decoder output is fed in as next step's input

Sequence to Sequence Model

- Seq2Seq is a conditioned language model
  - $h = f_{enc}(X)$ (final hidden state)
  - $Y = f_{dec}(h)$ (a LM that conditions on the initial hidden state h)
- Seq2Seq model is particularly generic for a lot of applications
  - Summarization (摘要) or Captioning (起标题)
    - Article → abstract/caption
  - Dialogue (对话)
    - Previous utterance → next utterance
  - Code generation
    - Natural language → python
  - VAE-based seq2seq model for text generation with latent variables

Sequence to Sequence Model

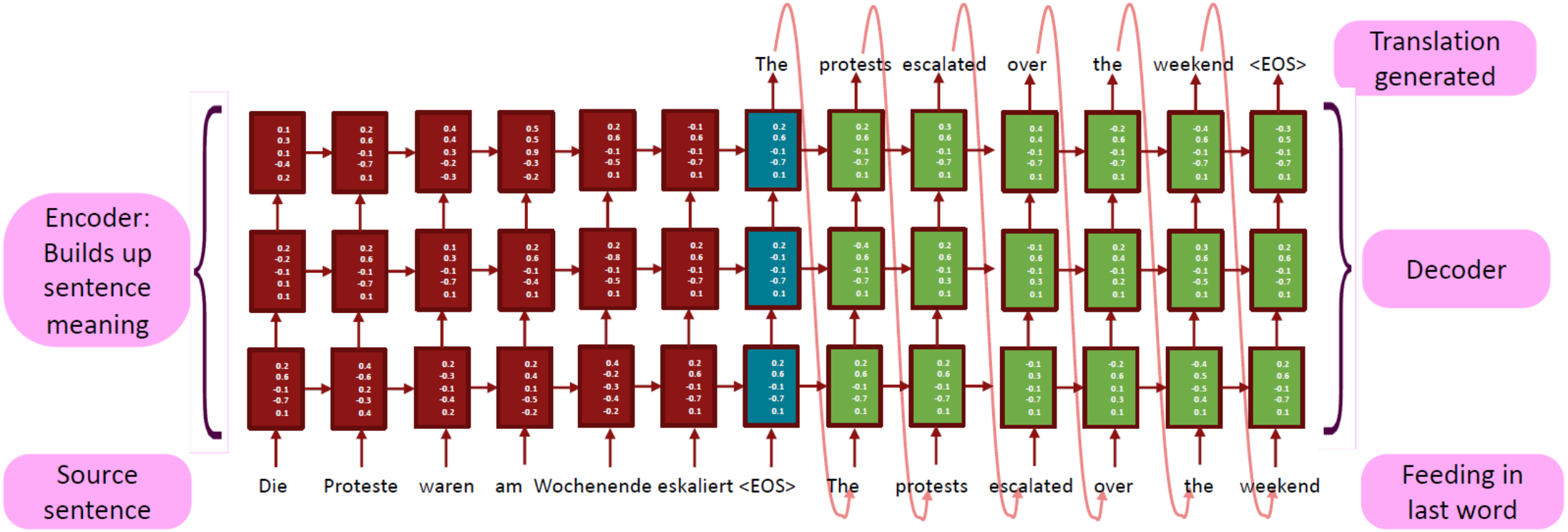
- How to train a seq2seq model?
  - Collect a huge paired dataset and train it end-to-end via BPTT!
  - MLE learning for $P(Y|X) = P(Y|f_{enc}(X))$



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

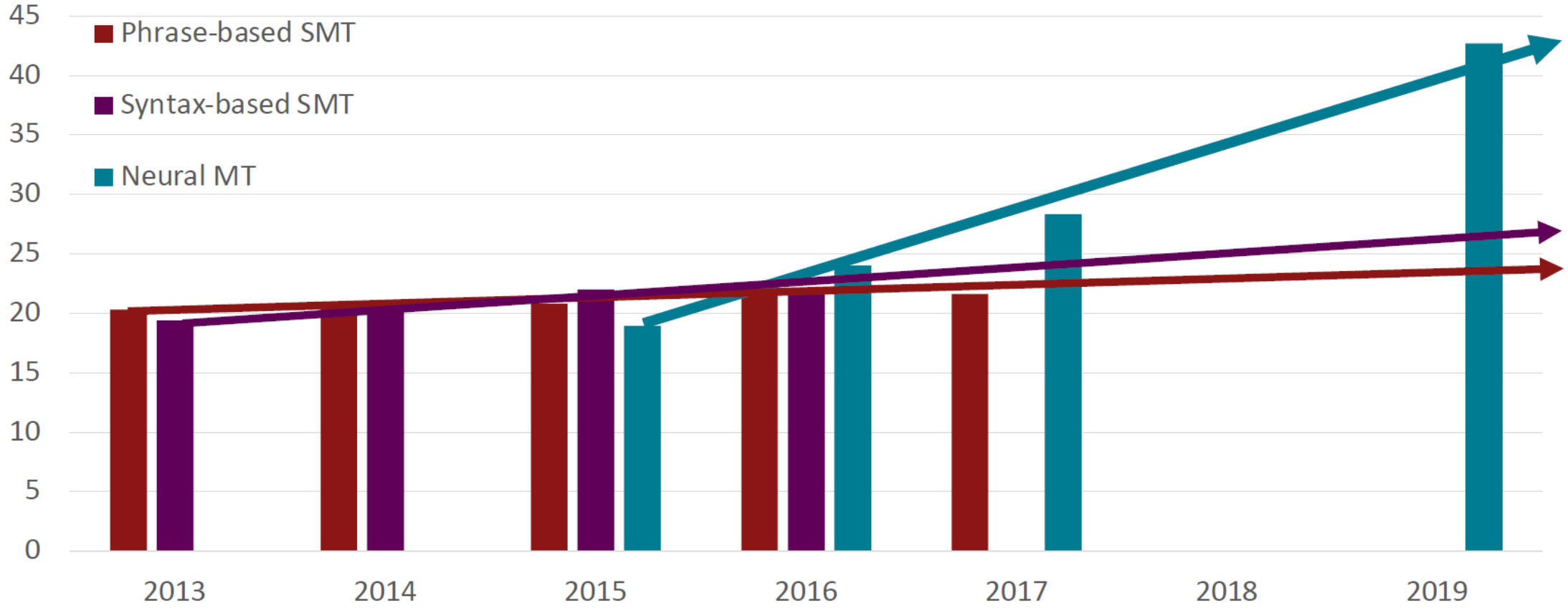
Sequence to Sequence Model

- We can also make the model deeper!
  - Stacked seq2seq model



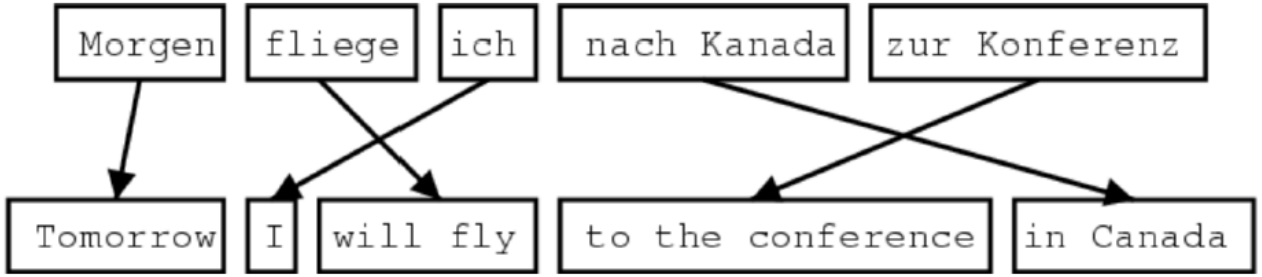
Sequence to Sequence Model

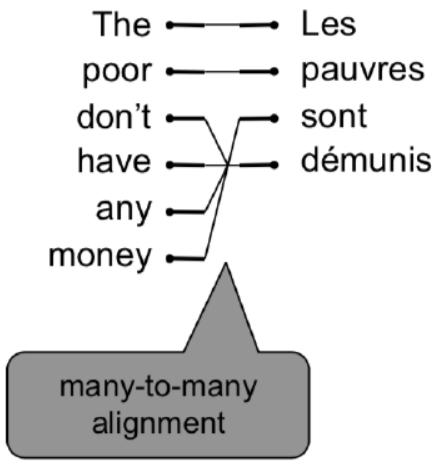
- 2016: Google switch google translate from SMT to NMT
  - Seq2Seq paper has >25.6k citations since 2014

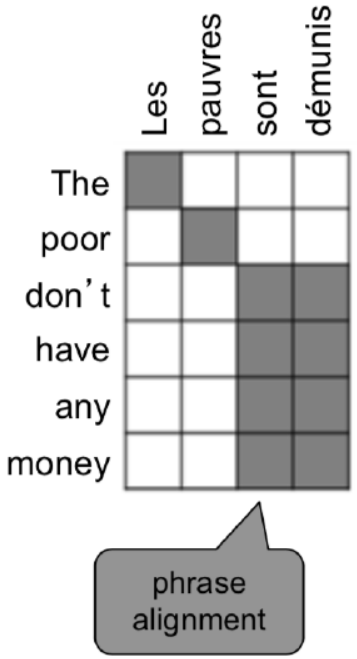


Sequence to Sequence Model

- Issue in the vanilla Seq2Seq model
  - Alignment: the word-level correspondence between X and Y
  - There are complex long-term dependencies

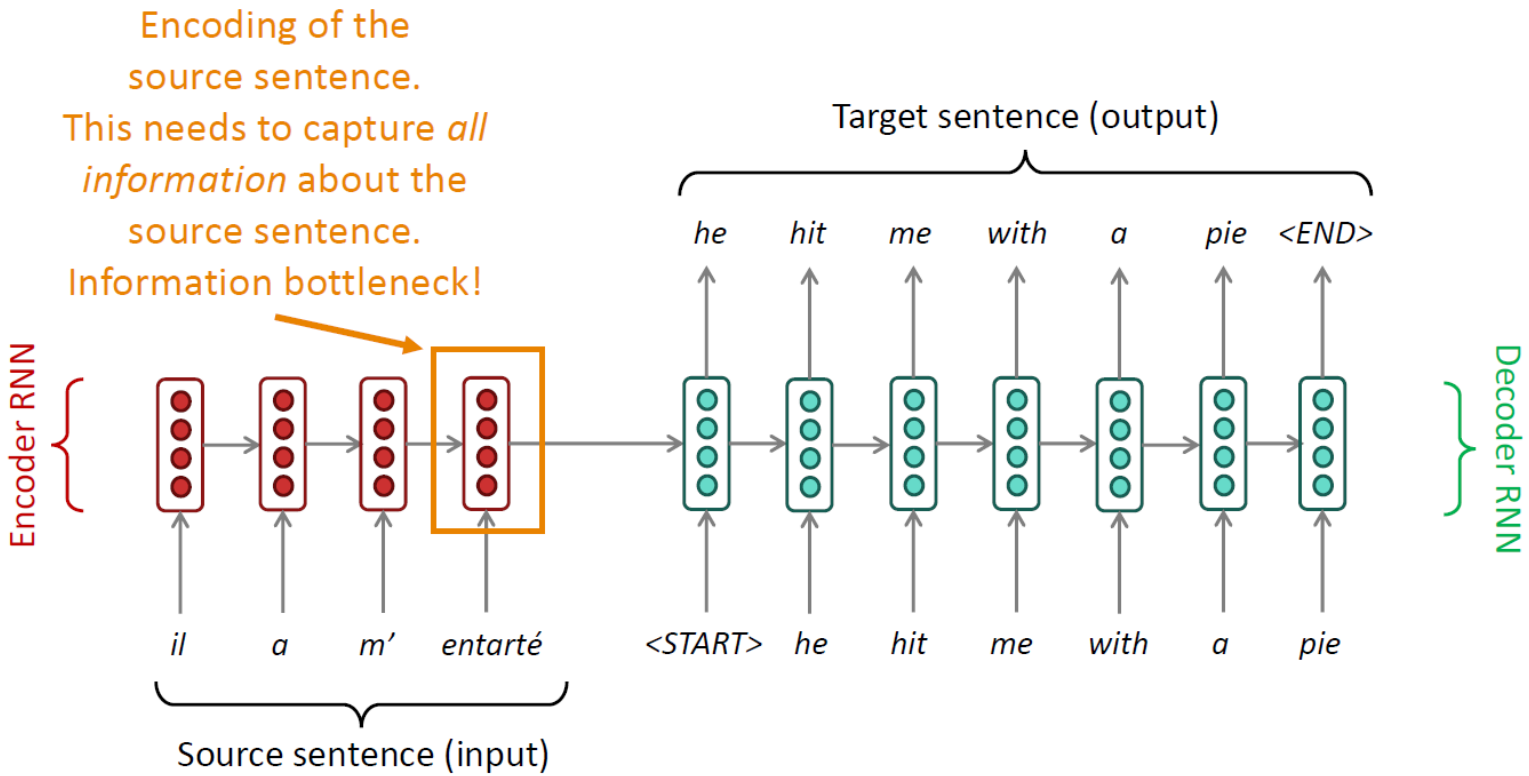






Sequence to Sequence Model

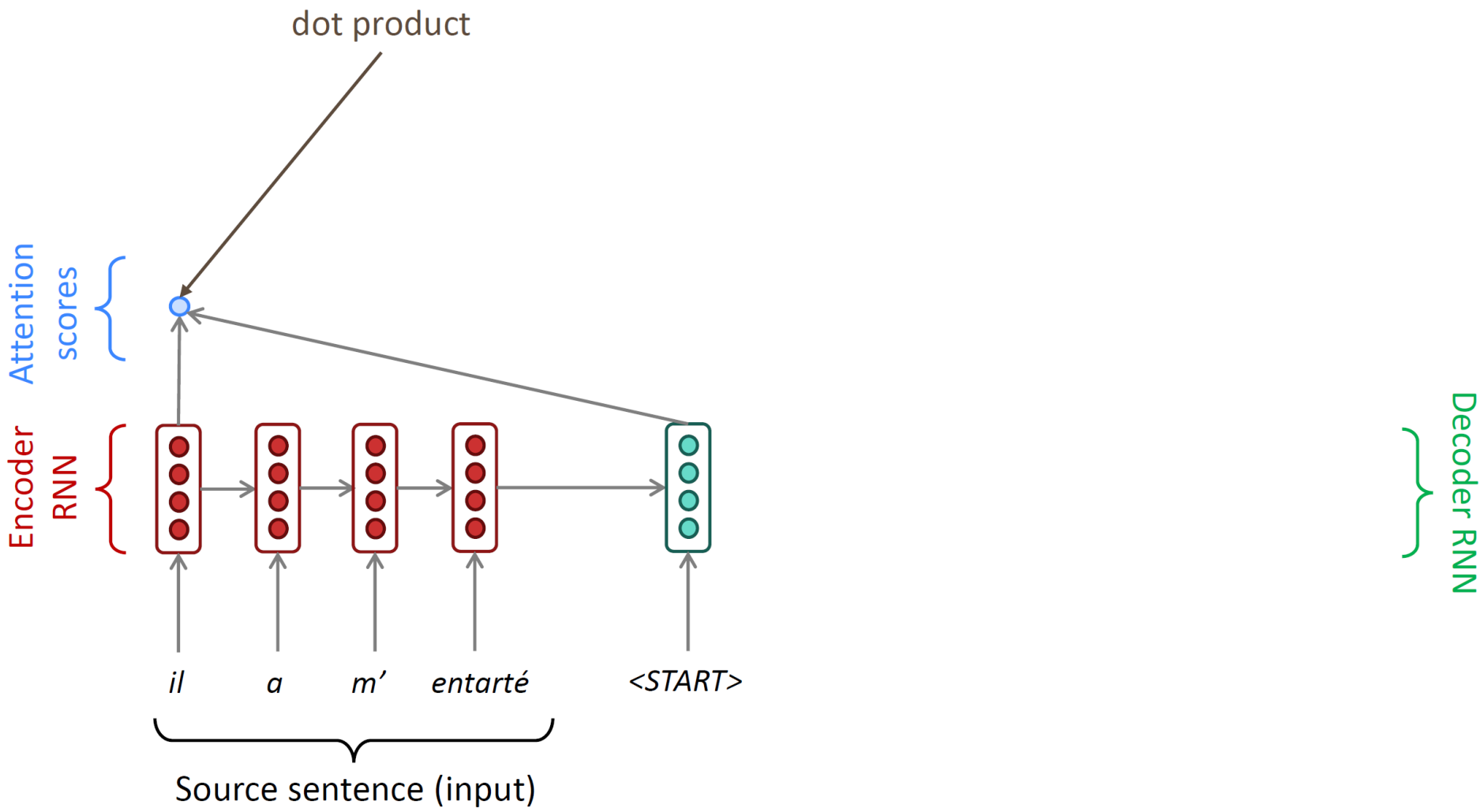
- Issue in the vanilla Seq2Seq model
  - The information bottleneck due to h
  - We want each Y_t to also focus on X_i that it is aligned with



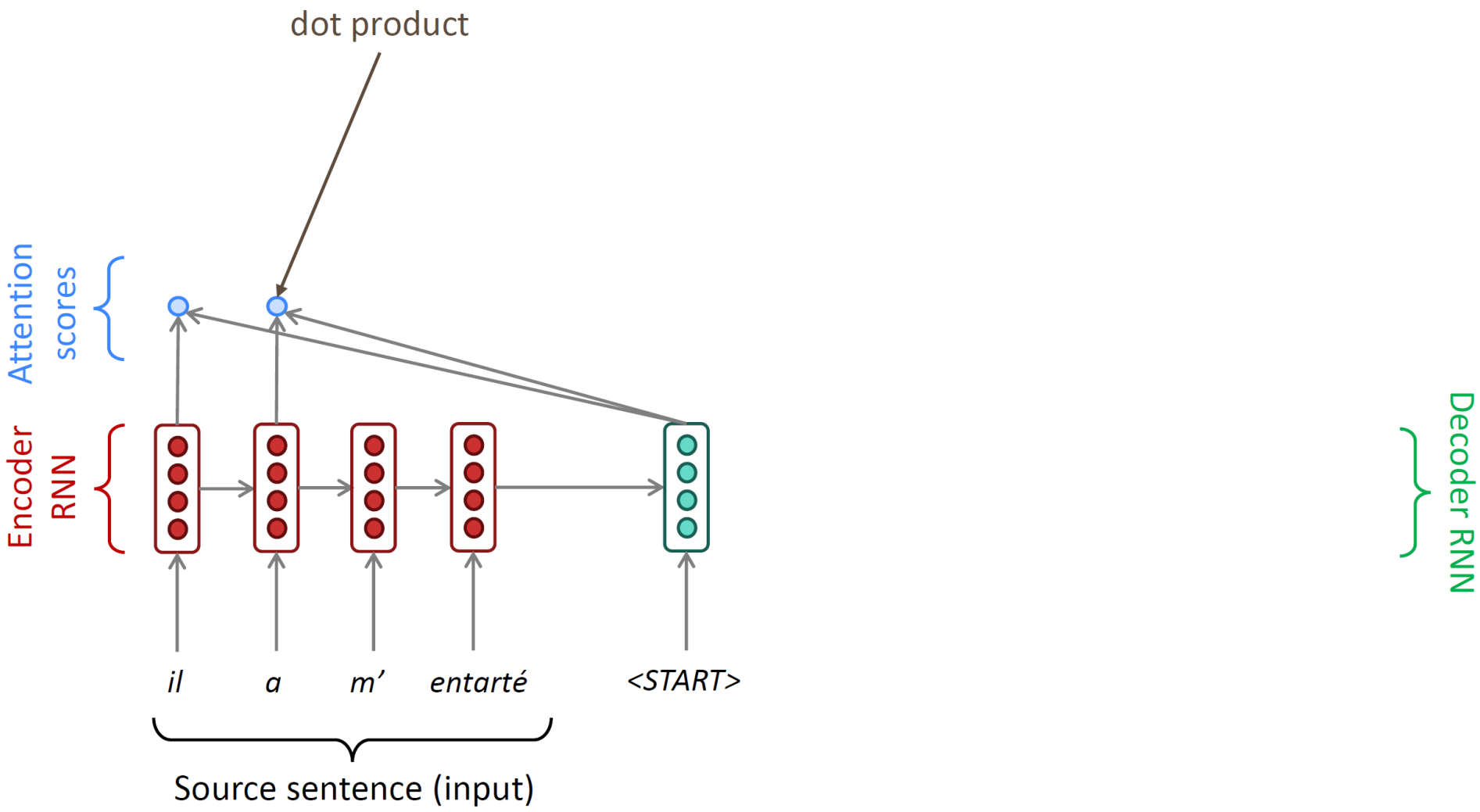
Sequence-to-Sequence with Attention

- NMT by Jointly Learning to Align and Translate
  - Bahdanau, Cho & Bengio, ICLR 2015 (33.9k citation)
  - Core idea:
    - When decoding Y_t, we consider both hidden states and alignments
      - Hidden: h_{t-1} from $Y_{i<t}$, i.e., $h_{t-1} = f_{dec}(Y_{i<t})$
      - Alignment: a direction connection to a portion of X
    - Which portion of X to focus?
      - Learn a softmax weight over X (attention distribution P_{att})
      - $P_{att}(X_i|h_{t-1})$: how much attention you want to put on word X_i
      - attention output $h_{att} = \sum_i f_{enc}\left(X_i \middle| X_{j<i}\right) \cdot P_{att}(X_i|h_{t-1})$
      - Use h_{t-1} and h_{att} to compute Y_t
    - Let's go through the diagram before showing more details

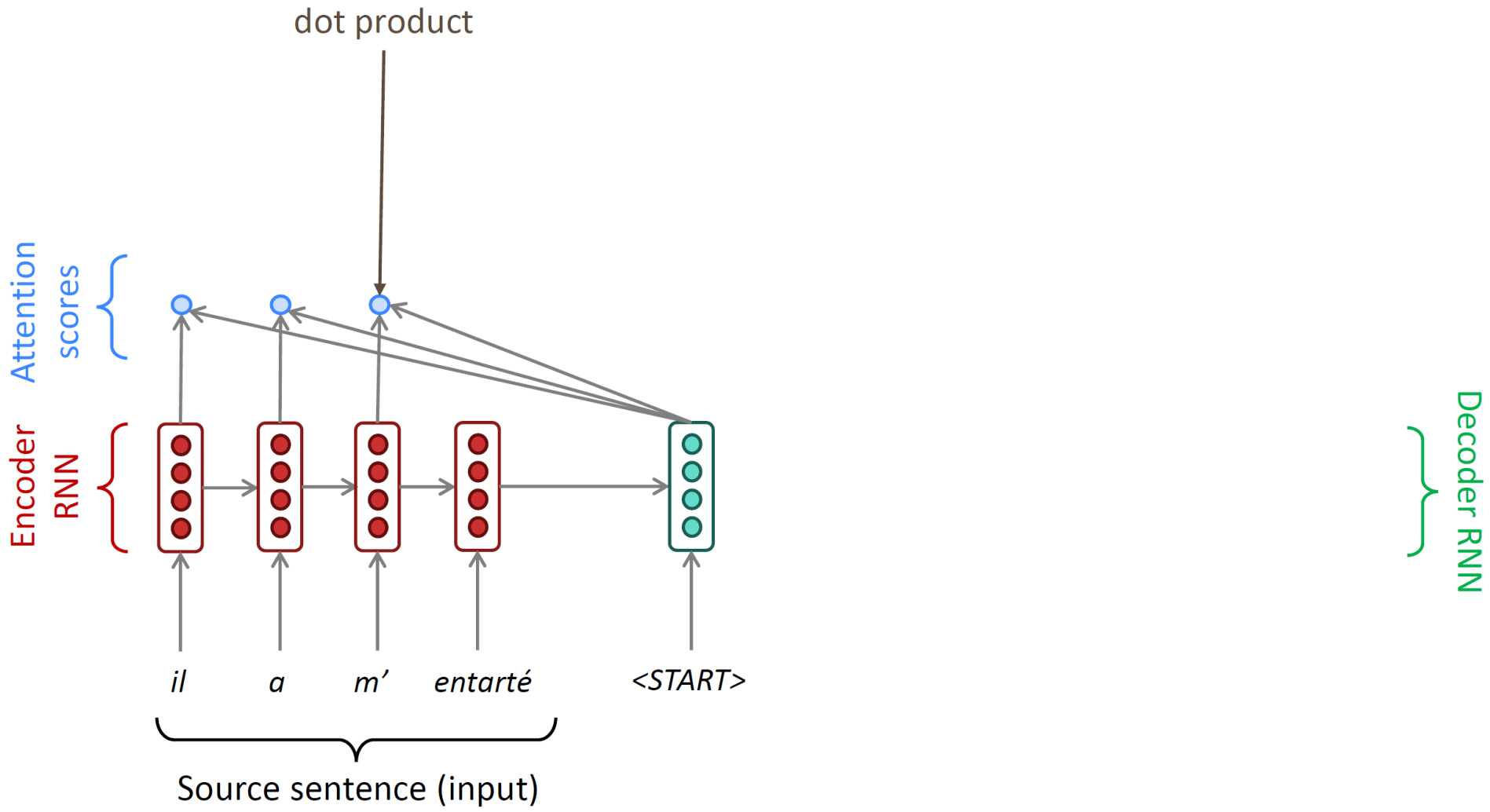
Sequence-to-Sequence with Attention



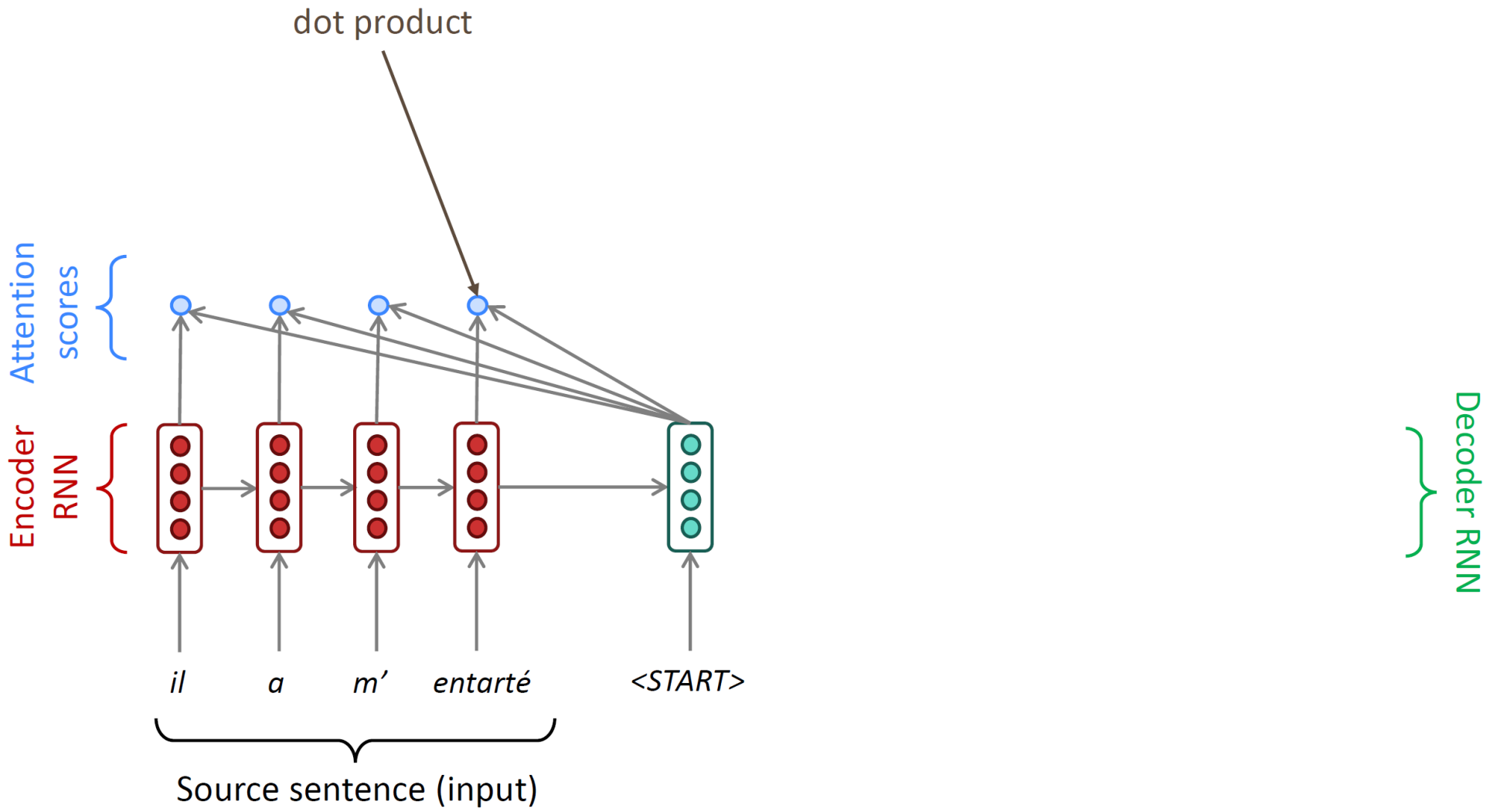
Sequence-to-Sequence with Attention



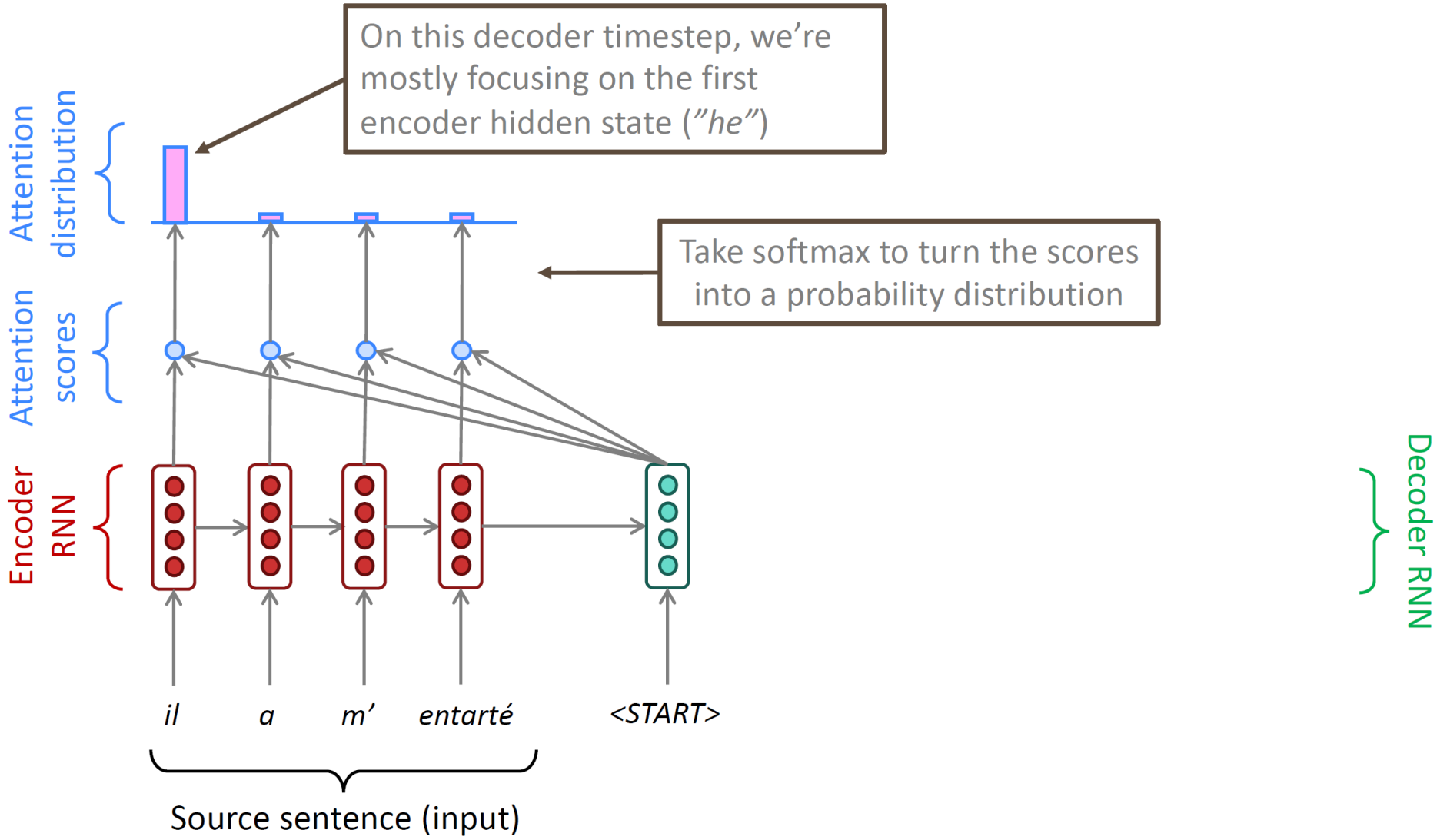
Sequence-to-Sequence with Attention



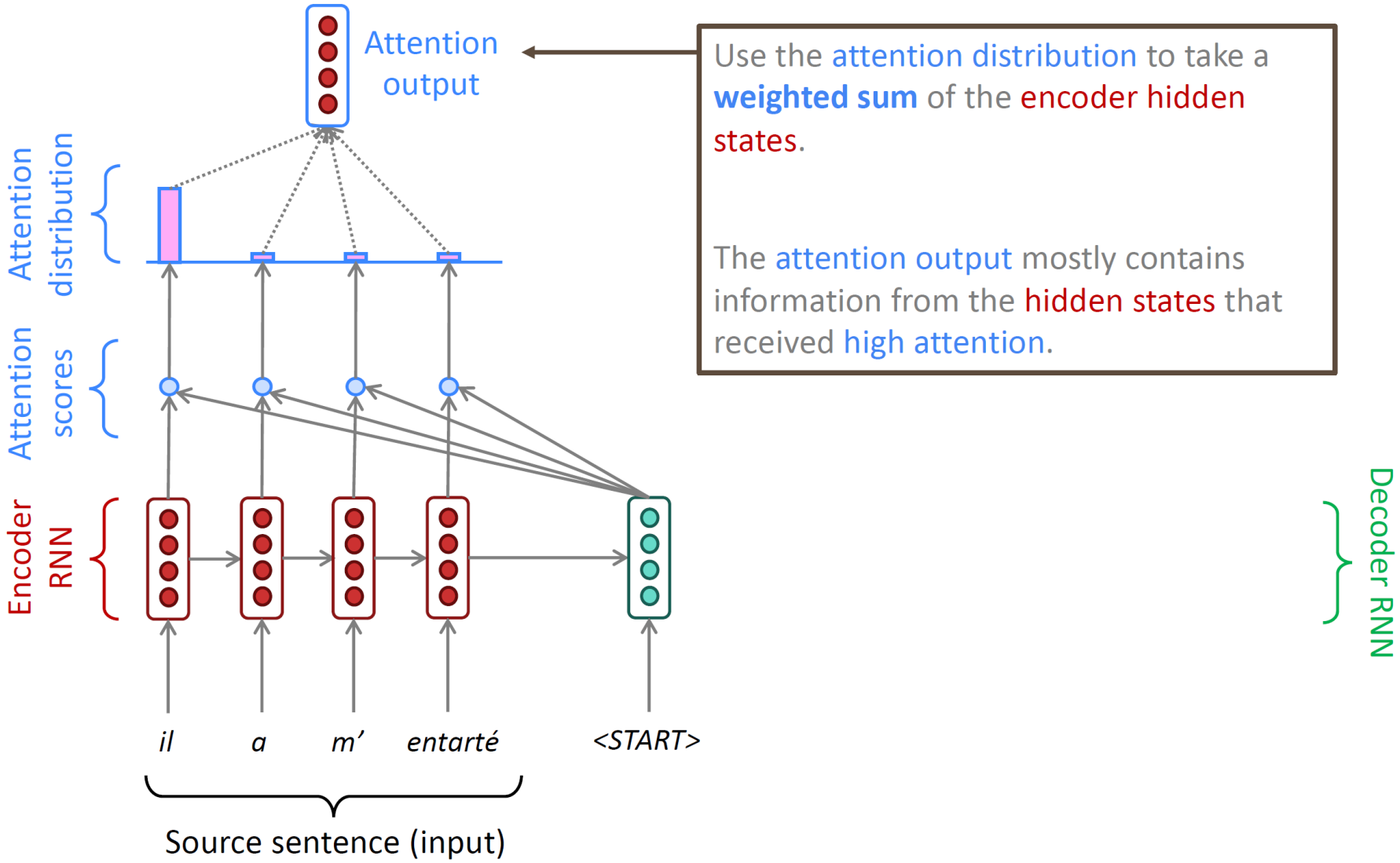
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



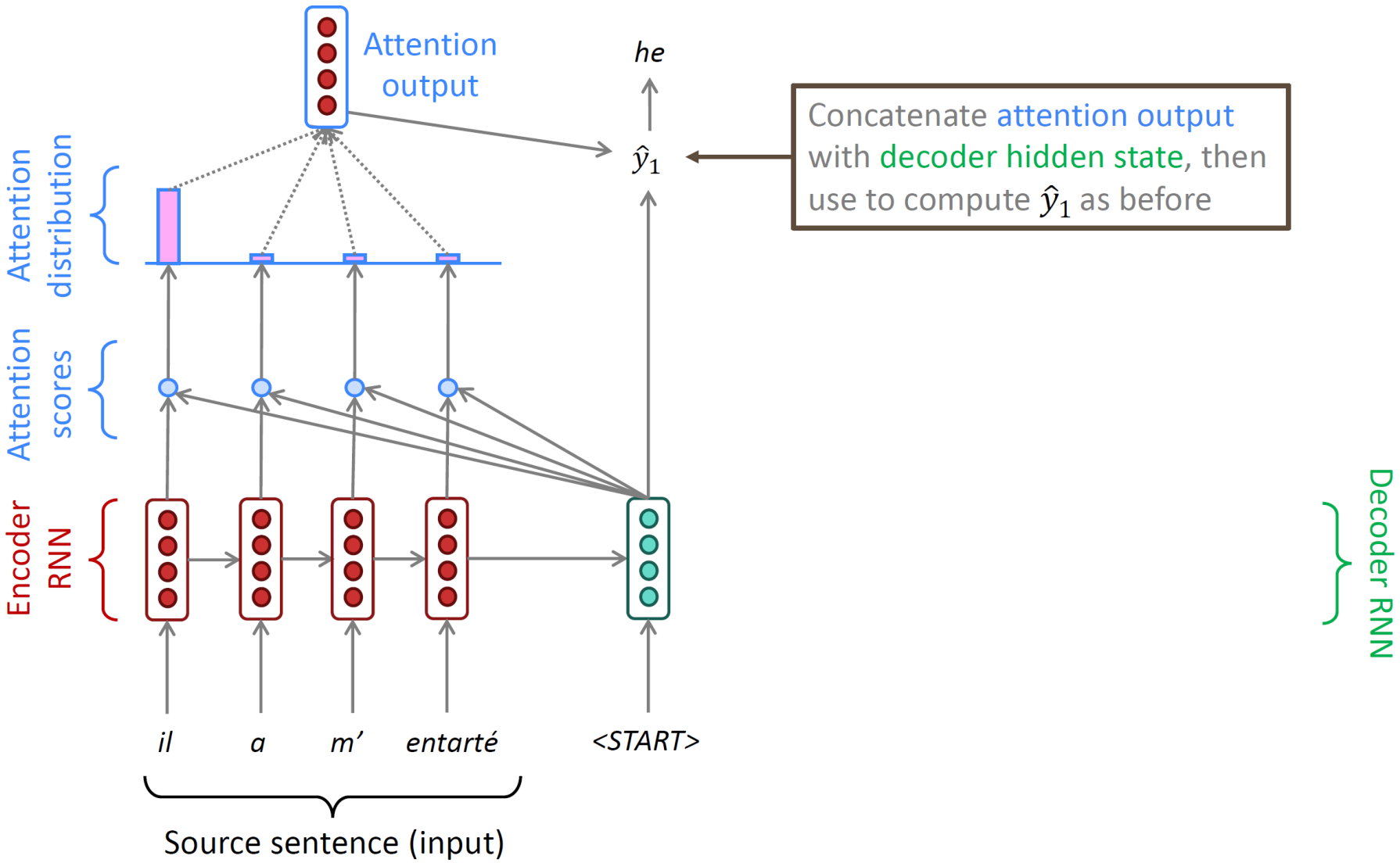
Sequence-to-Sequence with Attention



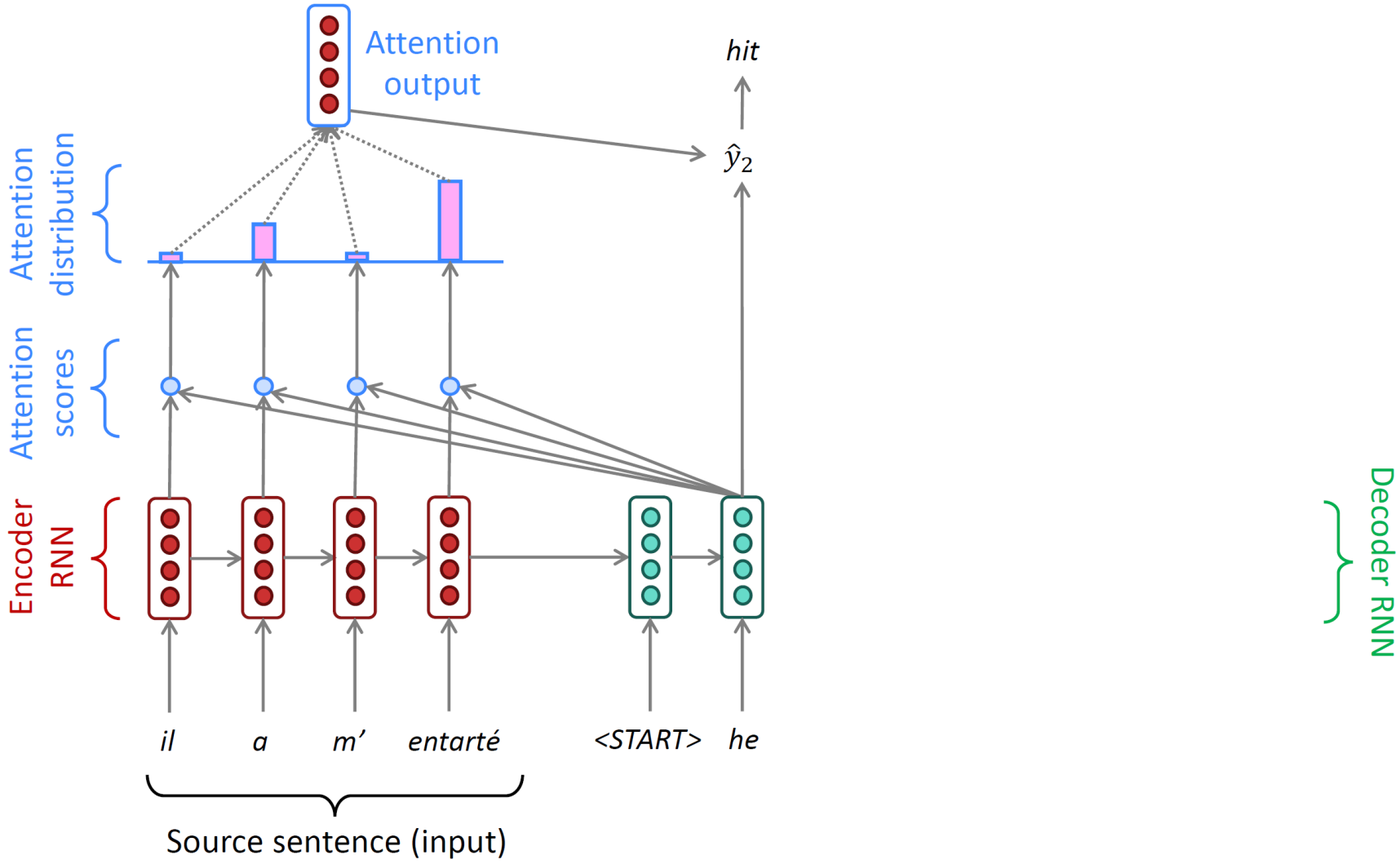
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

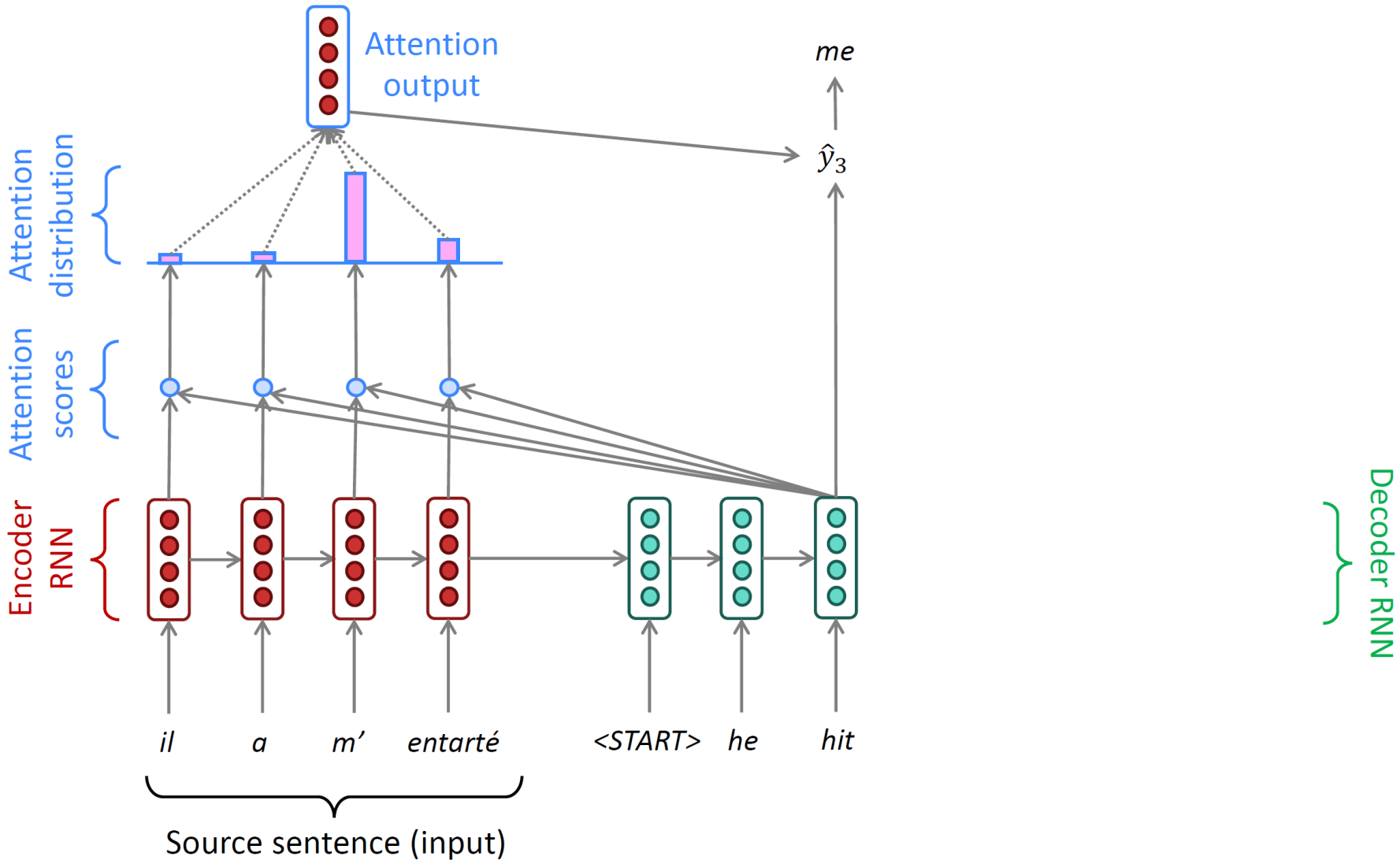
Sequence-to-Sequence with Attention



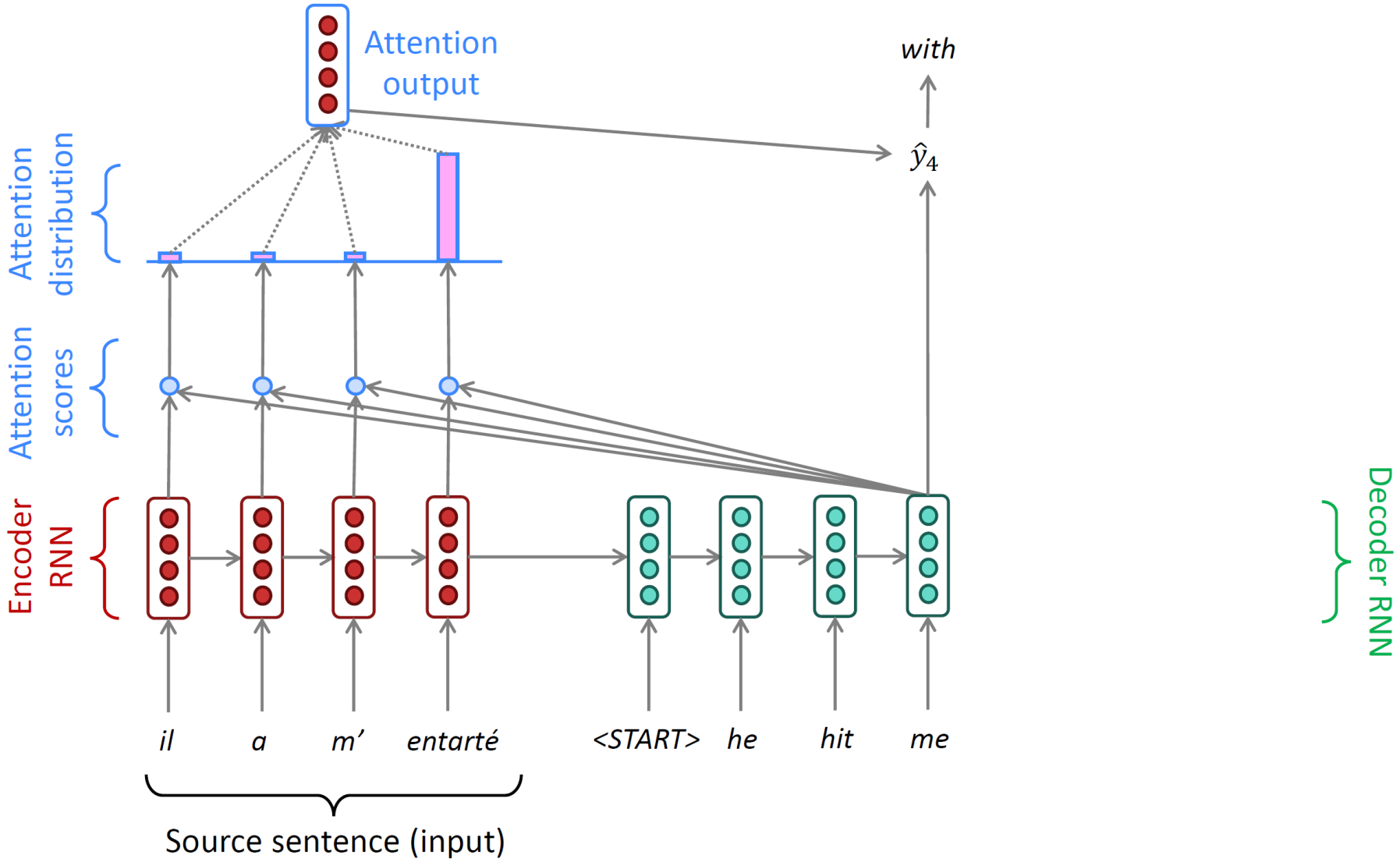
Sequence-to-Sequence with Attention



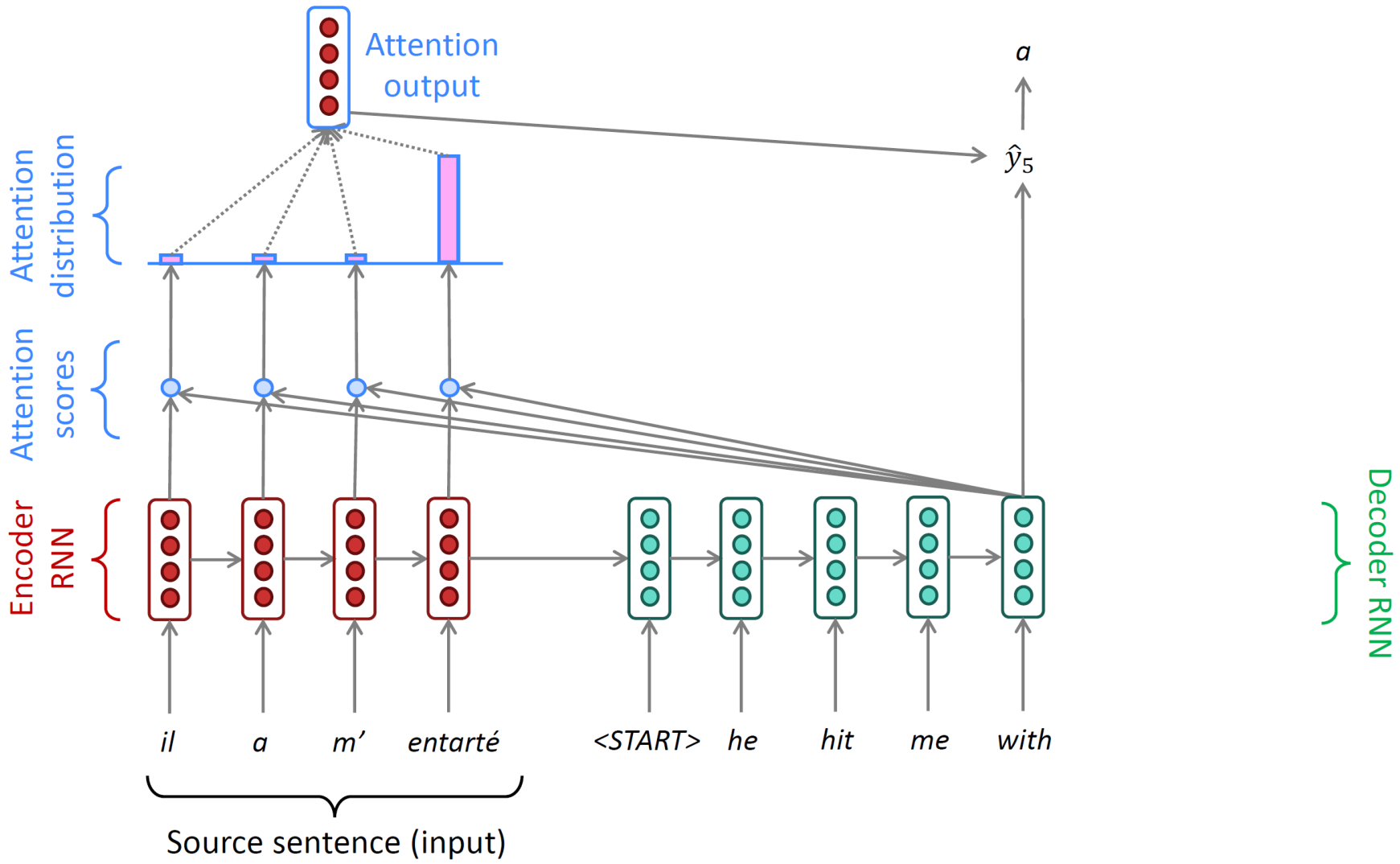
Sequence-to-Sequence with Attention



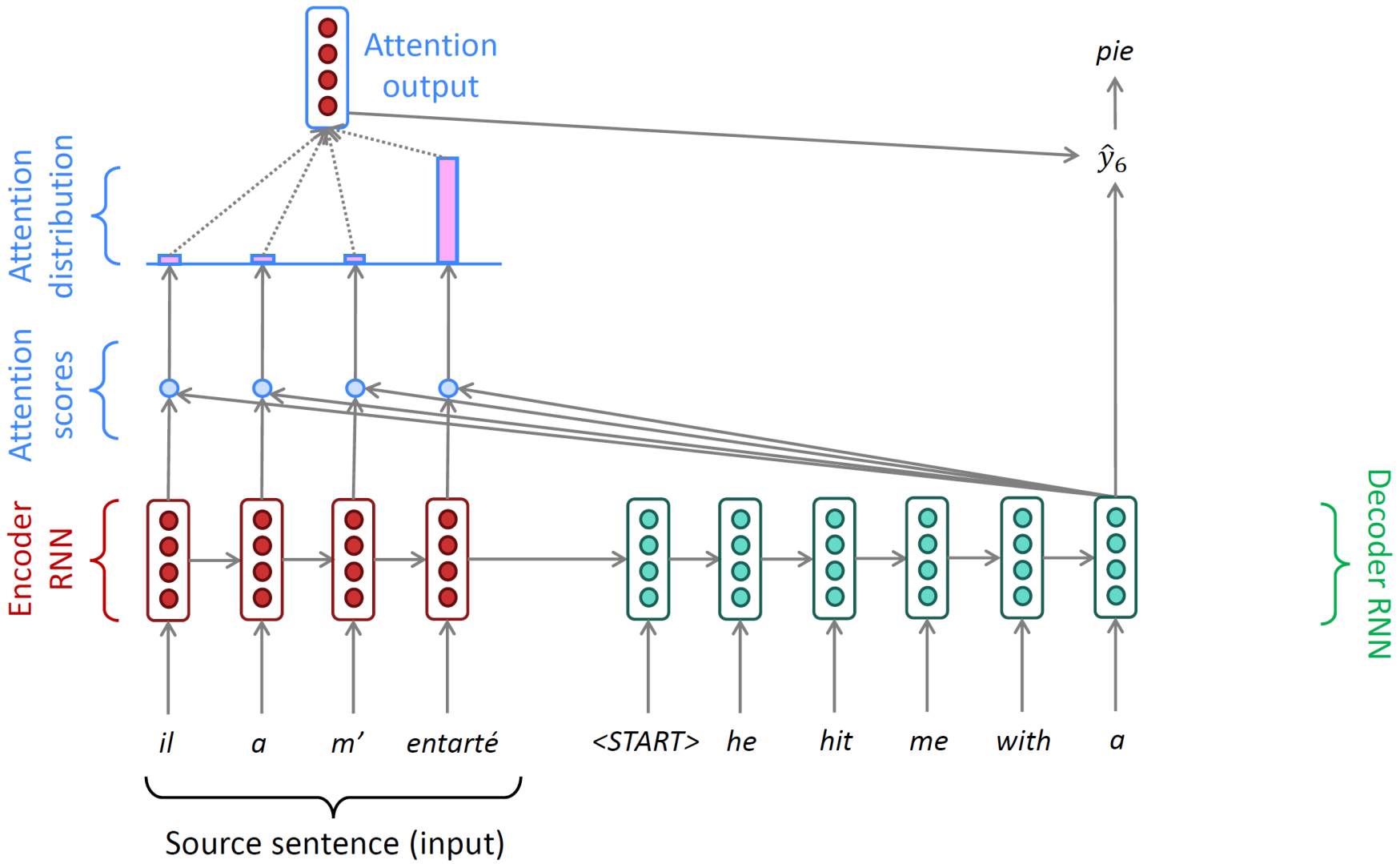
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention

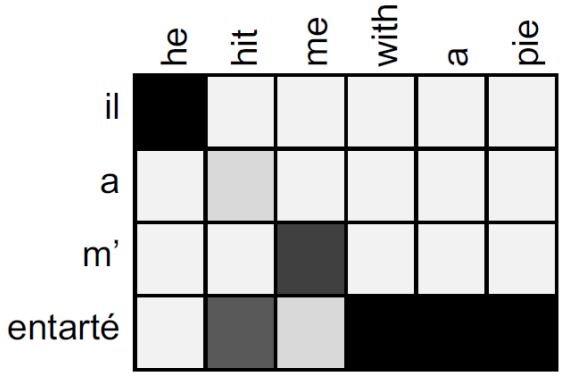
- Attention in equations
  - Input sequence X and encoder f_{enc} and decoder f_{dec}
  - $f_{enc}(X)$ produces hidden states $h_1^{enc}, h_2^{enc}, \dots, h_N^{enc}$
  - On timestep t, we have decoder hidden state h_t
  - Attention score $e_i = h_t^T h_i^{enc}$
  - Attention distribution $\alpha_i = P_{att}(X_i) = \mathrm{softmax}(e_i)$
  - Attention output

$$h_{att}^{enc} = \sum_i \alpha_i h_i^{enc}$$

  - $Y_t \sim g(h_t, h_{att}^{enc}; \theta)$
    - Sample output using both h_t and h_{att}^{enc}

Attention

- Attention is great!
  - It significantly improves NMT!
  - It solves the bottleneck problem and long-term dependency issue
    - Also helps gradient vanishing problem
  - It provides some interpretability
    - We can understand the focus of RNN decoder



- Attention is a general technique
  - Given a set of vector values V_i and a vector query q
  - Attention computes a weighted sum of values depending on q
- Attention can learn a representation of an arbitrary set of vectors $\{v_i\}$ depending on query q

Attention

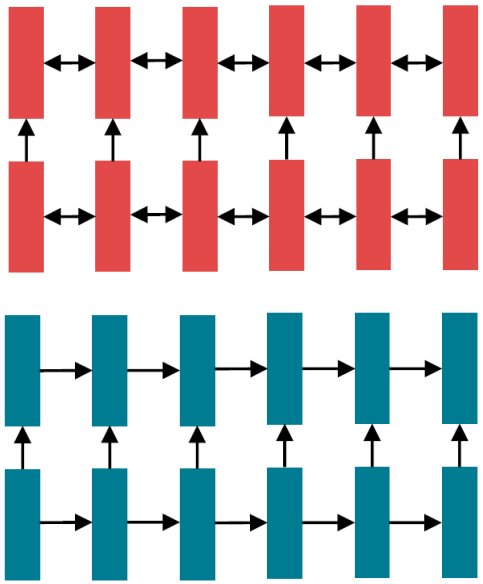
- Attention can learn a representation of an arbitrary set of vectors $\{v_i\}$ depending on query q
  - $\alpha_i = \mathrm{softmax}\big(f(v_i, q)\big)$ (attention distribution)
  - $v_{att} = \sum_i \alpha_i v_i$ (attention output)
  - Attention is *size-invariant* and *order-invariant*
- More use cases
  - E.g., a representation of a set of points (Pointer network, NIPS2015 & Deep Sets, NIPS2017)
  - E.g., include non-local information in CNN (Non-local network, CVPR18; Self-Attention GAN, ICML 19; BigGAN, ICLR 19)

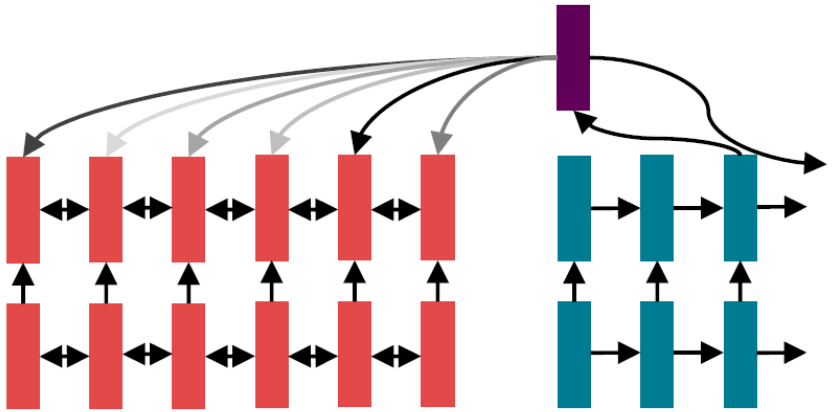
Attention

- Attention can learn a representation of an arbitrary set of vectors $\{v_i\}$ depending on query q
  - $\alpha_i = \mathrm{softmax}\big(f(v_i, q)\big)$ (attention distribution)
  - $v_{att} = \sum_i \alpha_i v_i$ (attention output)
- Attention Variants $f(v_i, q)$
  - Multiplicative attention: $f(v_i, q) = q^T W h_i$
    - W is a weight matrix
  - Additive attention: $f(v_i, q) = u^T \tanh(W_1 v_i + W_2 q)$
    - W_1, W_2 are weight matrices
    - u is a weight vector
  - Expressiveness v.s. efficiency

Sequence to Sequence Model with Attention

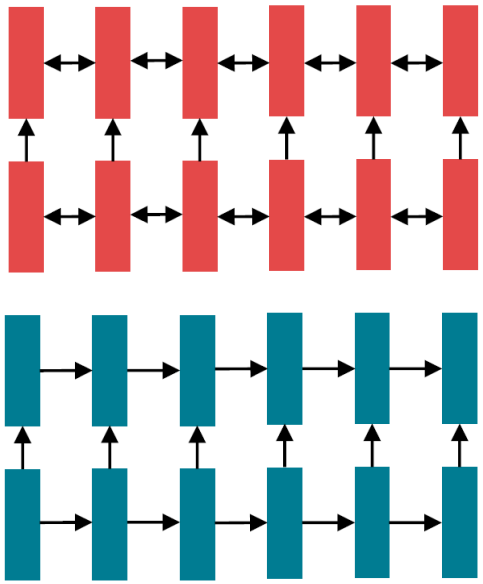
- Attention-Based Seq2Seq Model
  - Use bidirectional LSTMs as your encoder for input data
  - Use stacked LSTMs as your decoder for output data
  - Use attention for long-term dependencies

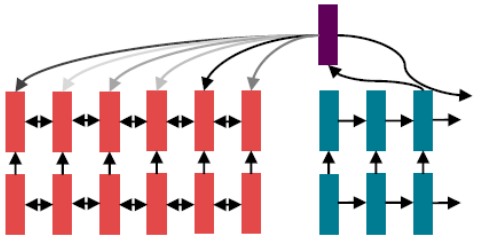




Sequence to Sequence Model with Attention

- Attention-Based Seq2Seq Model
  - Use bidirectional LSTMs as your encoder for input data
  - Use stacked LSTMs as your decoder for output data
  - Use attention for long-term dependencies
  - Most NLP applications!
    - till 2017 ☺





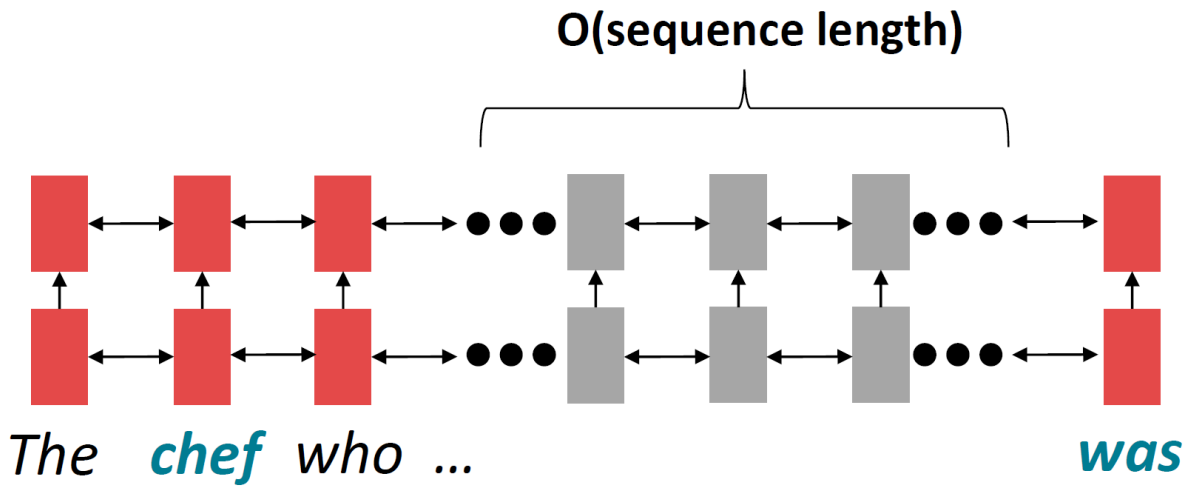
2014-2017ish
Recurrence

Lots of trial
and error

??????

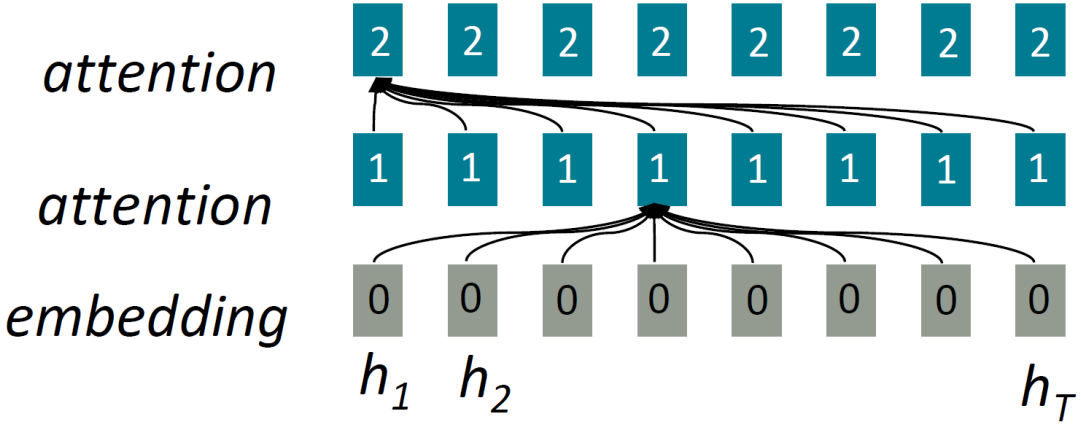
Sequence to Sequence Model with Attention

- Story So Far
  - RNN Models
    - Simple & Generic solution for sequence modeling
    - Issue for long-term dependencies
      - Linear computations for distant words through a single latent state
    - Lack of parallelization
      - Forced sequential computation (contrast with CNN)

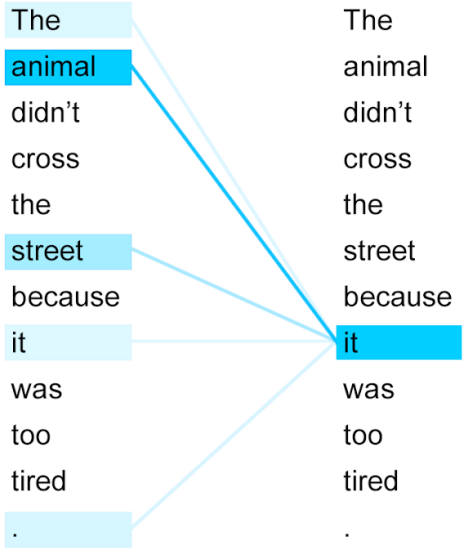


Sequence to Sequence Model with Attention

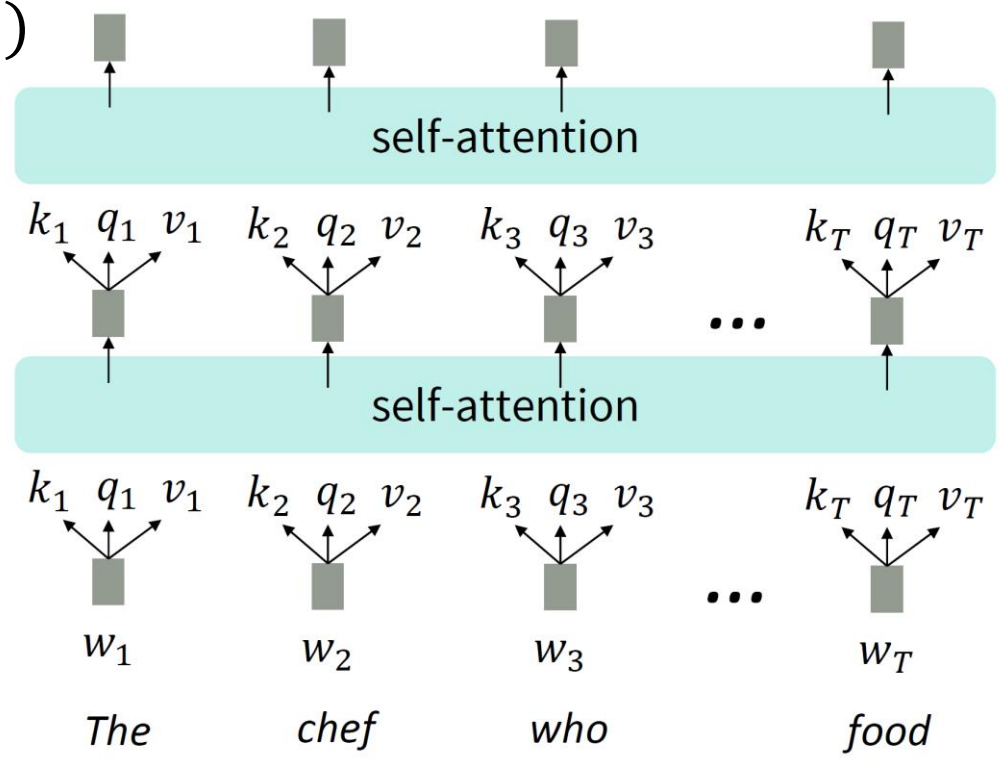
- Story So Far
  - RNN Models
    - Simple & Generic solution for sequence modeling
    - Issue for long-term dependencies
      - Linear computations for distant words through a single latent state
    - Lack of parallelization
      - Forced sequential computation (contrast with CNN)
  - Attention
    - Direct connection to distant words
    - $O(N)$ computation but perfectly parallel!
  - Attention is all we need?



Self-Attention



- Attention is all you need (NIPS2017, Google Brain)
  - A purely attention-base architecture for sequence modeling
    - NO RNN at all
  - Basic component: Self-Attention, $Y = f_{SA}(X;\theta)$
    - Core idea:
      - X_t attends on the entire X sequence
      - Y_t computed from X_t and the attention output
    - Equations for Y_t
      - Key k_t, value v_t, query q_t from X_t
        - $k_t, v_t, q_t = g_1(X_t;\theta)$
      - Attention distribution $\alpha_{t,j} = \mathrm{softmax}(q_t^T k_j)$
      - Attention output $out_t = \sum_j \alpha_{t,j}\, v_j$
      - $Y_t = g_2(out_t;\theta)$
  - Issues of self-attention?



Self-Attention

- Issues of Vanilla Self-Attention
  - Notion of sequence order
    - Attention is order-invariant
  - Lack of non-linearities
    - All the weights are simple linear weighted average
  - Capability of autoregressive modelling
    - In generation tasks, the model cannot “look at the future”
    - E.g., text generation
      - Y_t can only depend on $X_{i<t}$
      - Vanilla self-attention focuses on the entire sequence

Self-Attention

- Issues of Vanilla Self-Attention
  - **Notion of sequence order**
    - Attention is order-invariant
  - Lack of non-linearities
    - All the weights are simple linear weighted average
  - Capability of autoregressive modelling
    - In generation tasks, the model cannot “look at the future”
    - E.g., text generation
      - Y_t can only depend on $X_{i<t}$
      - Vanilla self-attention focuses on the entire sequence

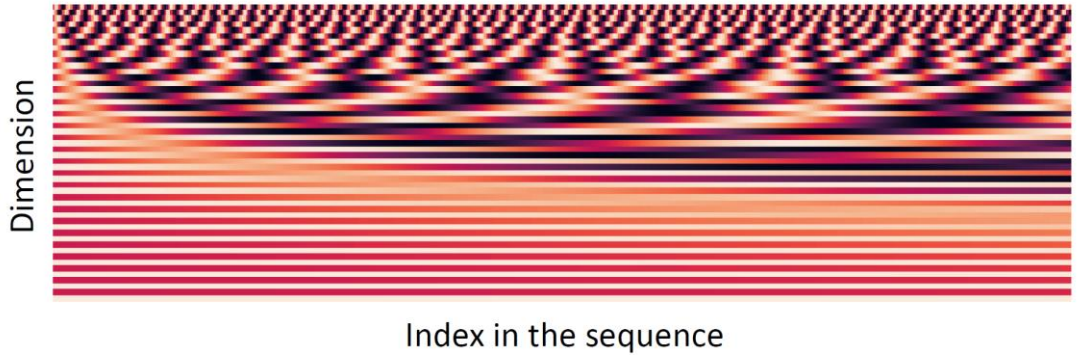
Self-Attention

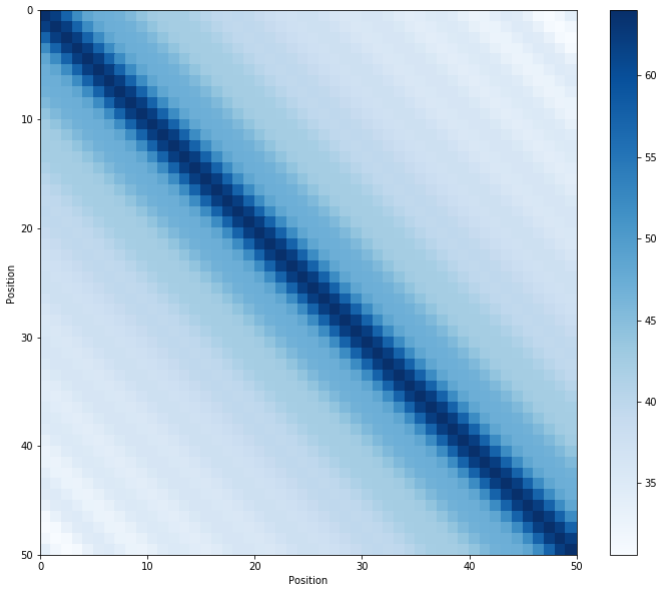
- Notion of Sequence Ordering
  - Vanilla attention
    - $\tilde{\alpha}_{i,j} = \mathrm{softmax}(\tilde{q}_i^T \tilde{k}_j);\ out_i = \sum_j \tilde{\alpha}_{i,j} \tilde{v}_j$
    - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain position information
  - Idea: position encoding
    - p_i: an embedding vector of position i
    - $k_t, v_t, q_t = g_1([X_t, p_t])$ include position features
    - Practical remark:
      - Additive can be sufficient: $k_t \leftarrow \tilde{k}_t + p_t, q_t \leftarrow \tilde{q}_t + p_t, v_t \leftarrow \tilde{v}_t + p_t$
      - p_t is typically only included in the first layer
    - How to design p_i?
      - Note that the length of a sequence can be long

Self-Attention

- Notion of Sequence Ordering
  - Idea: position encoding p_i
    - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain position information
  - Design of p_t
    - **Sinusoidal position representation**
      - Concatenate sinusoidal functions of varying periods

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$





Heatmap of $p_i^T p_j$

      - Pros: simple, naturally modelling “relative position”, easily applied to long sequences
      - Cons: not learnable; generalization poorly to sequences longer than training data

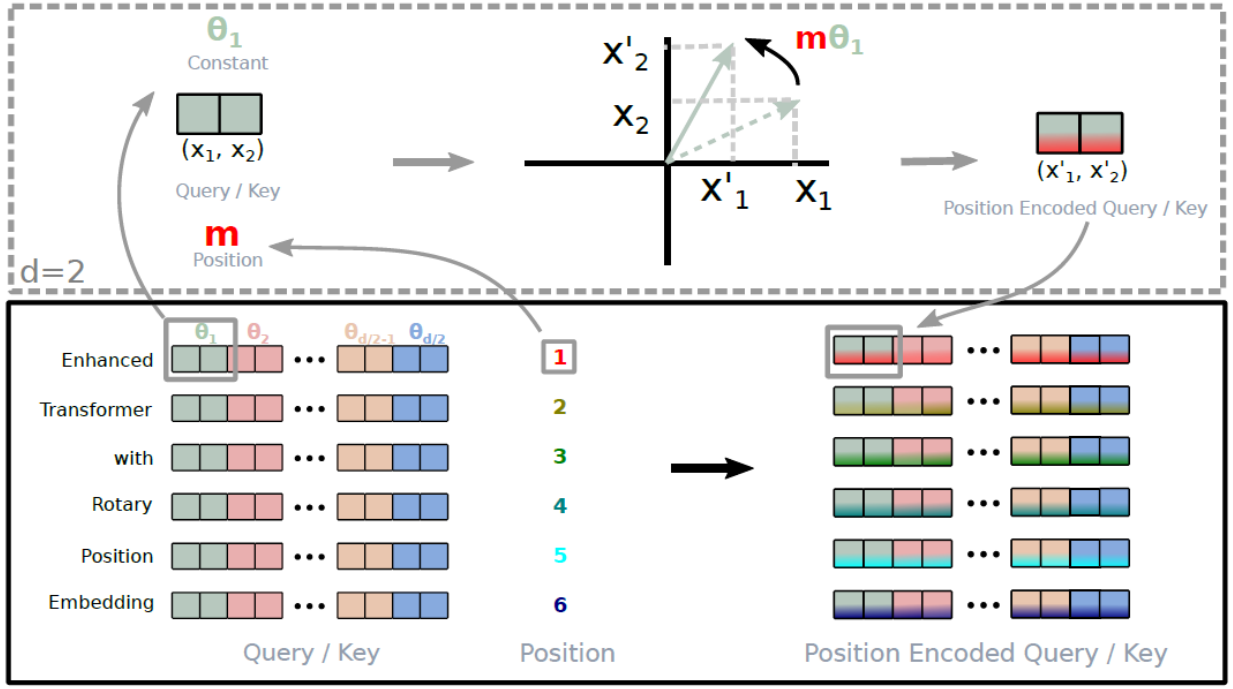
Self-Attention

- Notion of Sequence Ordering
  - Idea: position encoding p_i
    - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain position information
  - Design of p_t
    - Sinusoidal position representation
    - **Learned absolute representation**
      - Let p_t become a learned parameter vector!
        - Assume maximum length L, learn a matrix $p \in \mathbb{R}^{d \times T}$, p_t is a column of p
      - A popular choice in practice!
      - Pros:
        - Flexible and learnable, more powerful
      - Cons:
        - Assume a fixed maximum length L, does not work at all for length above L

Self-Attention

- Notion of Sequence Ordering
  - Idea: position encoding p_i
    - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain position information
  - Design of p_t
    - Sinusoidal position representation
    - Learned absolute representation
    - **Relative position representation (ACL2018, Google)**
      - When computing attention, relative distance is important!
        - $\alpha_{i,j} = \mathrm{softmax}\left(q_i^T\left(k_j + p_{[i-j]}\right)\right)$
        - $out_i = \sum_j \alpha_{i,j}\left(v_j + p_{[i-j]}\right)$
        - Bounded relative distance $p_t = p_{\max(-k,\min(k,t))}$
          - Truncate $t < -k$ to k and $t > k$ to k
      - Pros: learned representation and extrapolate well; More powerful.
      - Cons: computation overhead (refer to the paper for implementation tricks)

Self-Attention



- Notion of Sequence Ordering
  - Idea: position encoding p_i
    - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain p
  - Design of p_t
    - Sinusoidal position representation
    - Learned absolute representation
    - Relative position representation
    - **Rotary position embedding (RoPE, Roformer, 2021)**
      - Relative position but factored computation

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = (\boldsymbol{R}_{\Theta,m}^d \boldsymbol{W}_q \boldsymbol{x}_m)^\intercal (\boldsymbol{R}_{\Theta,n}^d \boldsymbol{W}_k \boldsymbol{x}_n) = \boldsymbol{x}^\intercal \boldsymbol{W}_q R_{\Theta,n-m}^d \boldsymbol{W}_k \boldsymbol{x}_n$$

$$\boldsymbol{R}_{\Theta,n-m}^d = (\boldsymbol{R}_{\Theta,m}^d)^\intercal \boldsymbol{R}_{\Theta,n}^d$$

$$\boldsymbol{R}_{\Theta,m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

Remark:

- Compatible with any dimension and length
- Fast computation
- Effective in practice

Self-Attention

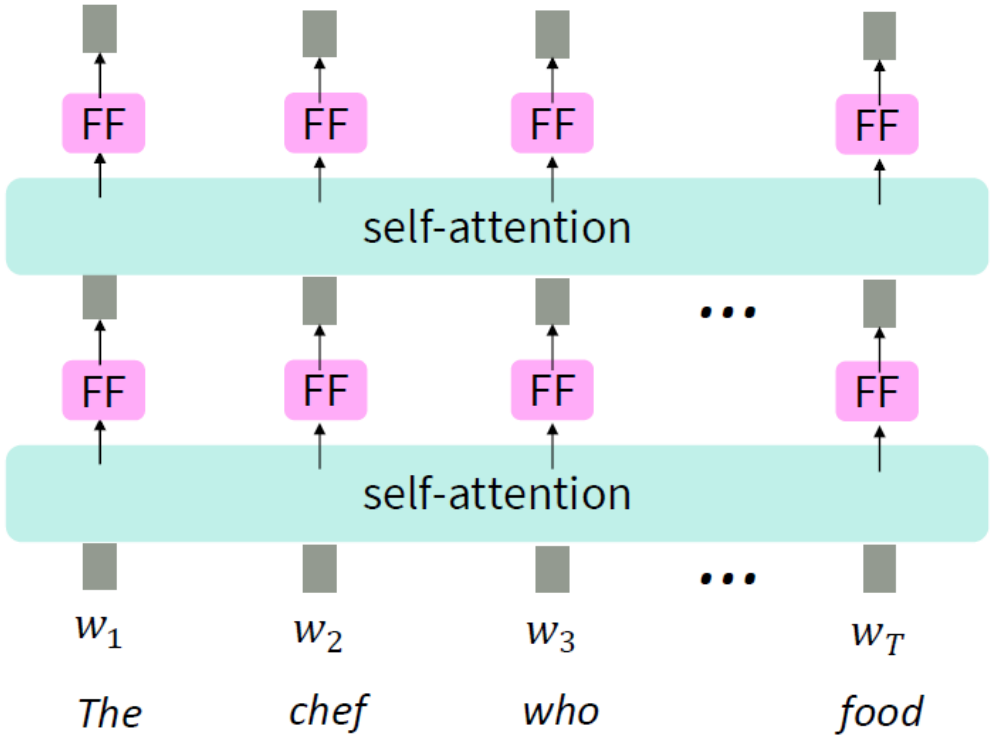
- Issues of Vanilla Self-Attention
  - **Notion of sequence order**
    - Solution: position encoding
  - Lack of non-linearities
    - All the weights are simple linear weighted average
  - Capability of autoregressive modelling
    - In generation tasks, the model cannot “look at the future”
    - E.g., text generation
      - Y_t can only depend on $X_{i<t}$
      - Vanilla self-attention focuses on the entire sequence

Self-Attention

- Issues of Vanilla Self-Attention
  - Notion of sequence order
    - Solution: position encoding
  - **Lack of non-linearities**
    - All the weights are simple linear weighted average
  - Capability of autoregressive modelling
    - In generation tasks, the model cannot “look at the future”
    - E.g., text generation
      - Y_t can only depend on $X_{i<t}$
      - Vanilla self-attention focuses on the entire sequence

Self-Attention

- Combine nonlinearities in self-attention
  - Vanilla self-attention
    - No element-wise activation functions (e.g., ReLU, tanh)
    - Only weighted average and softmax operators
      - Essentially linear transformations of inputs
  - Easy fix:
    - Add an MLP to process out_i
    - $m_i = MLP(out_i)$
    - $= W_2 \cdot \mathrm{ReLU}(W_1 \cdot out_i + b_1) + b_2$
    - Remark
      - we do not put activation layer before softmax



Self-Attention

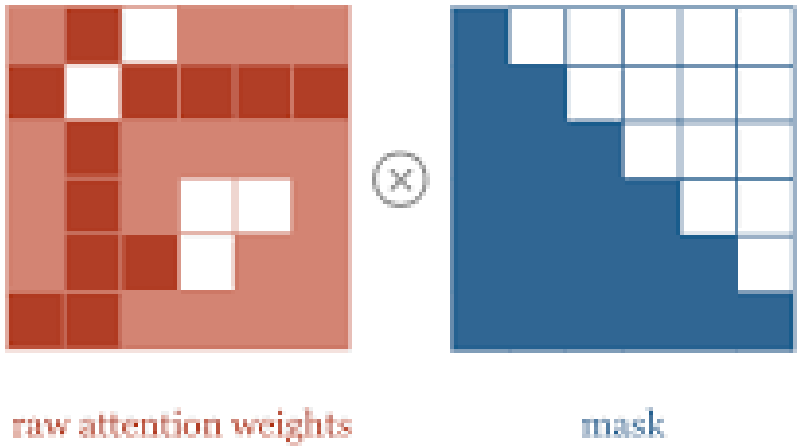
- Issues of Vanilla Self-Attention
  - Notion of sequence order
    - Solution: position encoding
  - **Lack of non-linearities**
    - Solution: post-processing MLP layer
  - Capability of autoregressive modelling
    - In generation tasks, the model cannot "look at the future"
    - E.g., text generation
      - Y_t can only depend on $X_{i<t}$
      - Vanilla self-attention focuses on the entire sequence

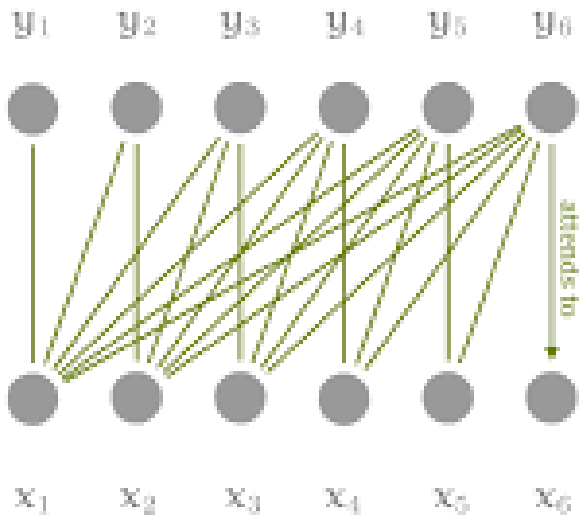
Self-Attention

- Issues of Vanilla Self-Attention
  - Notion of sequence order
    - Solution: position encoding
  - Lack of non-linearities
    - Solution: post-processing MLP layer
  - **Capability of autoregressive modelling**
    - In generation tasks, the model cannot "look at the future"
    - E.g., text generation
      - Y_t can only depend on $X_{i<t}$
      - Vanilla self-attention focuses on the entire sequence

Self-Attention

- Autoregressive Modeling
  - In language mode decoder: model $P(Y_t|X_{i<t})$
    - out_t cannot loot at future $X_{i>t}$
    - Naïve solution:
      - For each t, a varying for-loop only iterating over $i \leq t$
      - Varying for-loop for each t, parallelization unfriendly
  - Masked Attention
    - Compute $e_{i,j} = q_i^T k_j$ as usual (perfect parallel)
    - Mask out $e_{i>j}$ by setting $e_{i>j} = -\infty$ (perfect parallel)
      - $e \odot (1 - M) \leftarrow -\infty$; M is a fixed 0/1 mask matrix
    - Then compute $\alpha_i = \mathrm{softmax}(e_i)$ (perfect parallel)
    - Remark:
      - $M = 1$ for full self-attention
      - Set M for arbitrary dependency ordering





Self-Attention

- Issues of Vanilla Self-Attention
  - Notion of sequence order
    - Solution: position encoding
  - Lack of non-linearities
    - Solution: post-processing MLP layer
  - **Capability of autoregressive modelling**
    - Solution: masked self-attention

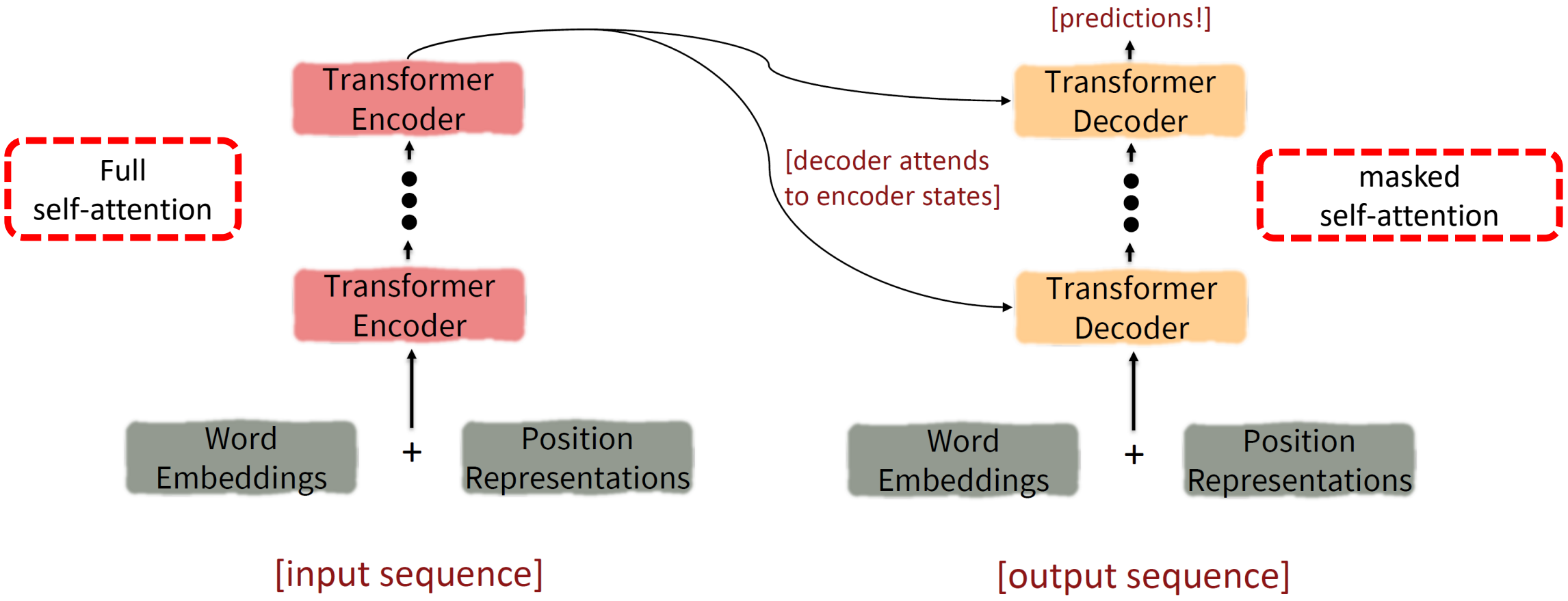
Self-Attention

- Issues of Vanilla Self-Attention
  - Notion of sequence order
    - Solution: position encoding
  - Lack of non-linearities
    - Solution: post-processing MLP layer
  - Capability of autoregressive modelling
    - Solution: masked self-attention

  - Basic building block for the famous “Transformer” model!
    - Attention is all you need (NIPS2017, Vaswani et al, Google)
      - Self-attention + **a few more other enhancements**!
    - A milestone: first pure attention-based model for effective sequence modeling
      - *Originally proposed for NMT; Soon dominates general sequence modeling problems*

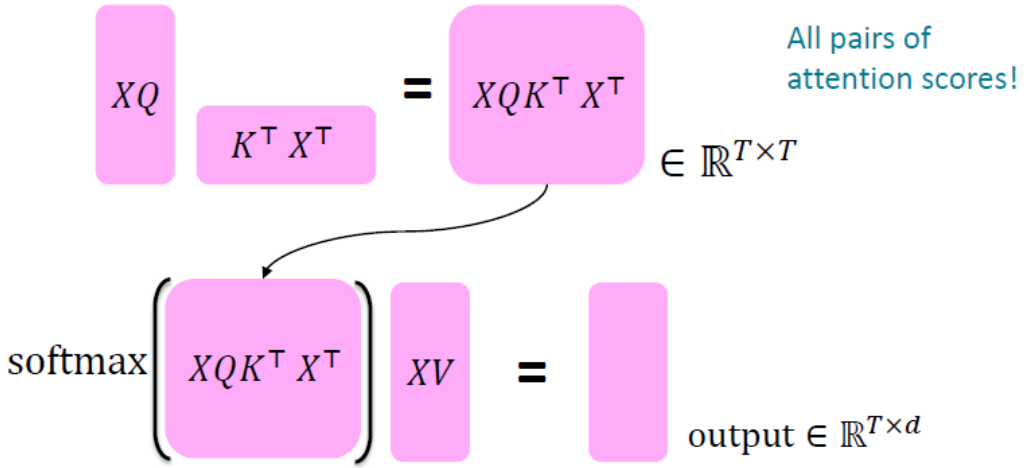
Transformer Model

- Transformer-based sequence-to-sequence modeling



Transformer Model

- Transformer-based sequence-to-sequence modeling
  - Basic building blocks: masked self-attention
  - Enhancements
    - **Key-query-value attention**
      - Obtain q_t, v_t, k_t from X_t
      - $q_t = W^q X_t$; $v_t = W^v X_t$; $k_t = W^k X_t$ (position encoding omitted)
        - W^q, W^v, W^k are learnable weight matrices
      - $\alpha_{i,j} = \mathrm{softmax}(q_i^T k_j)$; $out_i = \sum_j \alpha_{i,j} v_j$
      - Intuition: key, query, and value can focus on different parts of input

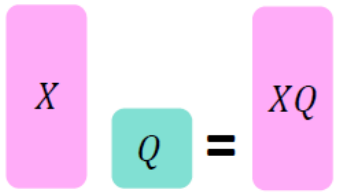


Transformer Model

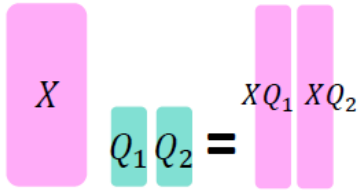
- Transformer-based sequence-to-sequence modeling
  - Basic building blocks: masked self-attention
  - Enhancements
    - Key-query-value attention
    - **Multi-headed attention**
      - Standard attention → single-headed attention
        - $X_t \in \mathbb{R}^d, \quad Q, K, V \in \mathbb{R}^{d \times d}$
        - We only "look at" a single position j with high $\alpha_{i,j}$
        - What if we want to look at different j for different reasons?
      - Idea: define h separate attention heads
        - h different attention distributions, keys, values and queries
        - $Q^l, K^l, V^l \in \mathbb{R}^{d \times \frac{d}{h}}$, for $1 \leq l \leq h$
        - $\alpha_{i,j}^l = \text{softmax}\left(q_i^{l^T} k_j^l\right); out_i^l = \sum_j \alpha_{i,j}^l v_j^l$

#Params Unchanged!

Single-head attention
(just the query matrix)

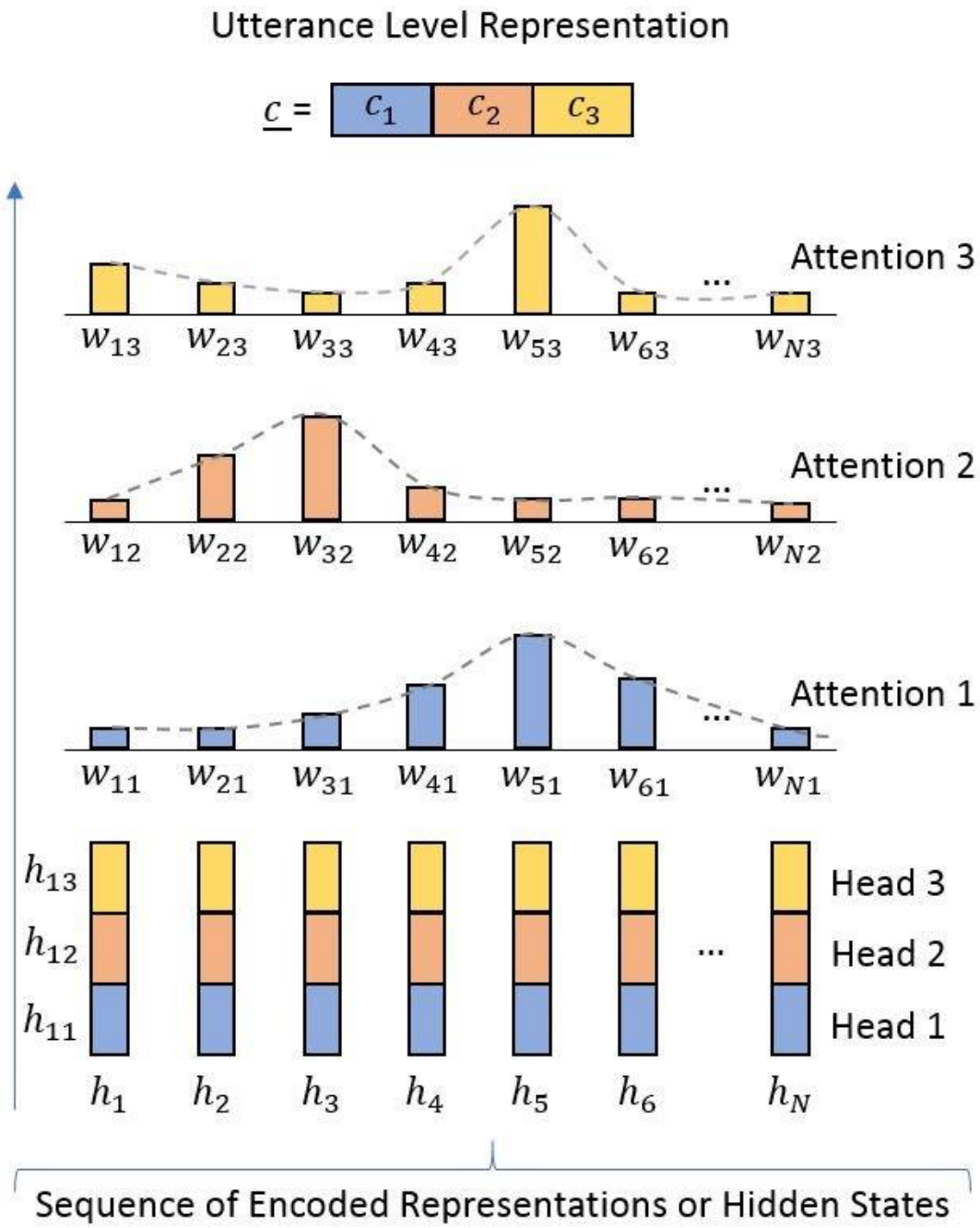


Multi-head attention
(just two heads here)



Transformer Model

- Transformer-based sequence-to-sequenc
  - Basic building blocks: masked self-attention
  - Enhancements
    - Key-query-value attention
    - **Multi-headed attention**
      - Standard attention → single-headed attention
        - $X_t \in \mathbb{R}^d, \quad Q, K, V \in \mathbb{R}^{d \times d}$
        - We only "look at" a single position j with hig
        - What if we want to look at different j for dif
      - Idea: define h separate attention heads
        - h different attention distributions, keys, valu
        - $Q^l, K^l, V^l \in \mathbb{R}^{d \times \frac{d}{h}}$, for $1 \leq l \leq h$
        - $\alpha_{i,j}^l = \text{softmax}\left(q_i^{l^T} k_j^l\right); out_i^l = \sum_j \alpha_{i,j}^l v_j^l$

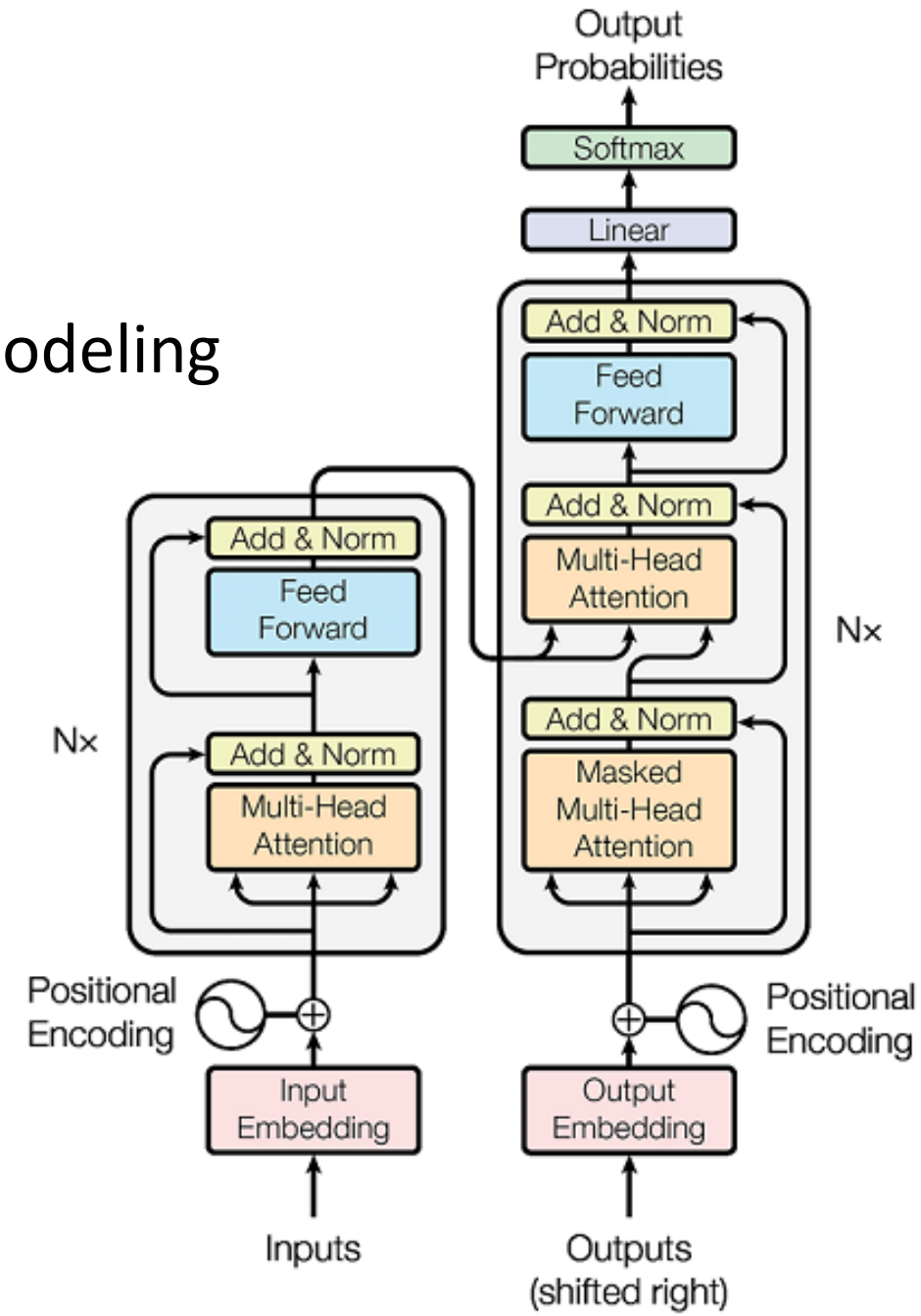


Transformer Model

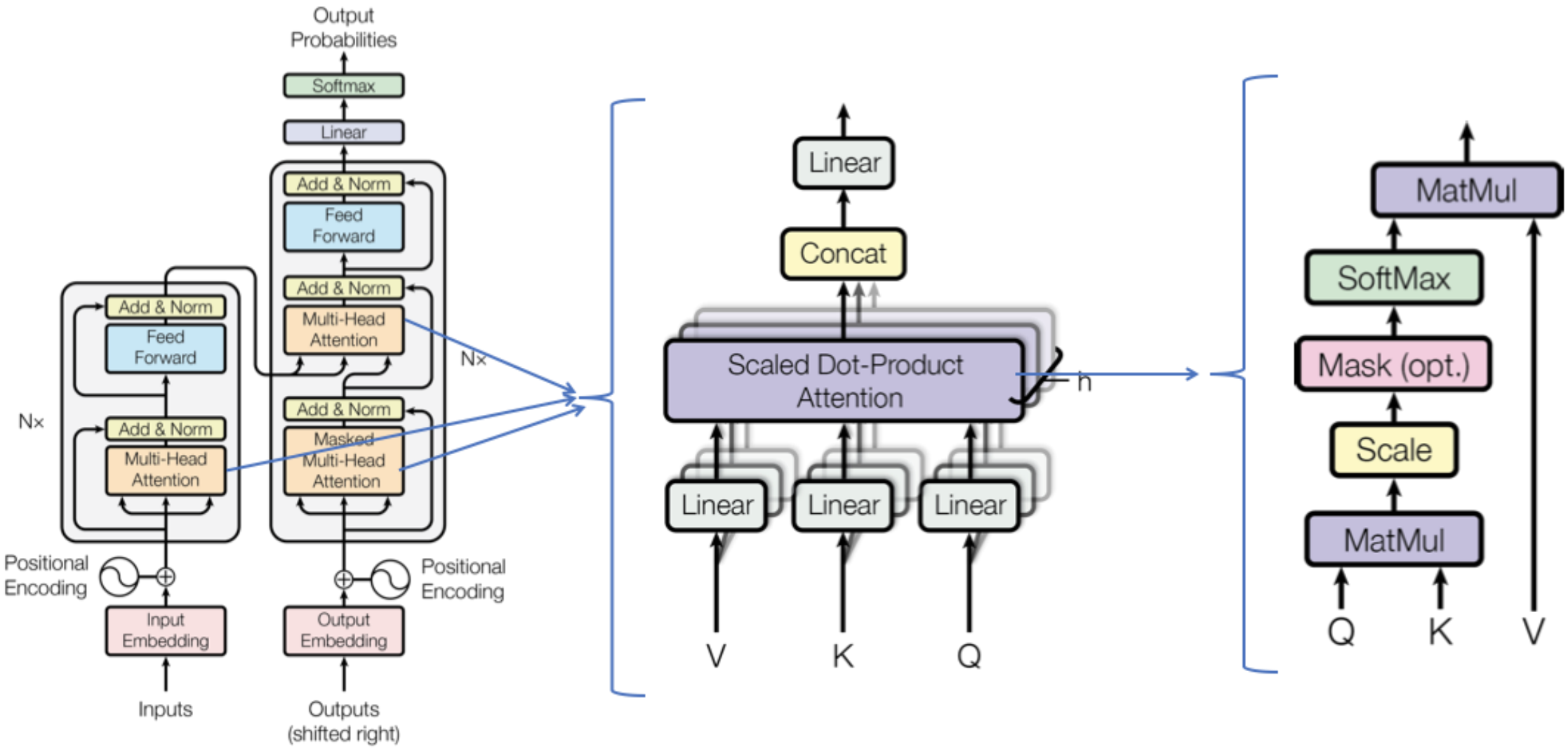
- Transformer-based sequence-to-sequence modeling
  - Basic building blocks: masked self-attention
  - Enhancements
    - Key-query-value attention
    - Multi-headed attention
    - **Architecture modifications**
      - Residual connection
      - Layer normalization
        - $out_t = \mathrm{LN}(f_{SA}(X_t, M) + X_t); m_t = \mathrm{LN}(\mathrm{MLP}(out_t) + out_t)$
      - Scaled dot product
        - Intuition: when dimension d becomes large, $q^T k$ can be large
        - Issue: input to softmax can be large and make gradient small
        - $\alpha_{i,j}^l = \mathrm{softmax}\left(\frac{q_i^{l^T} k_j^l}{\sqrt{d/h}}\right)$

Transformer Model

- Transformer-based sequence-to-sequence modeling
  - Basic building blocks: masked self-attention
    - Position encoding
    - Post-processing MLP
    - Attention mask
  - Enhancements
    - Key-query-value attention
    - Multi-headed attention
    - Architecture modifications
      - Residual connection
      - Layer normalization
      - Scaled dot product



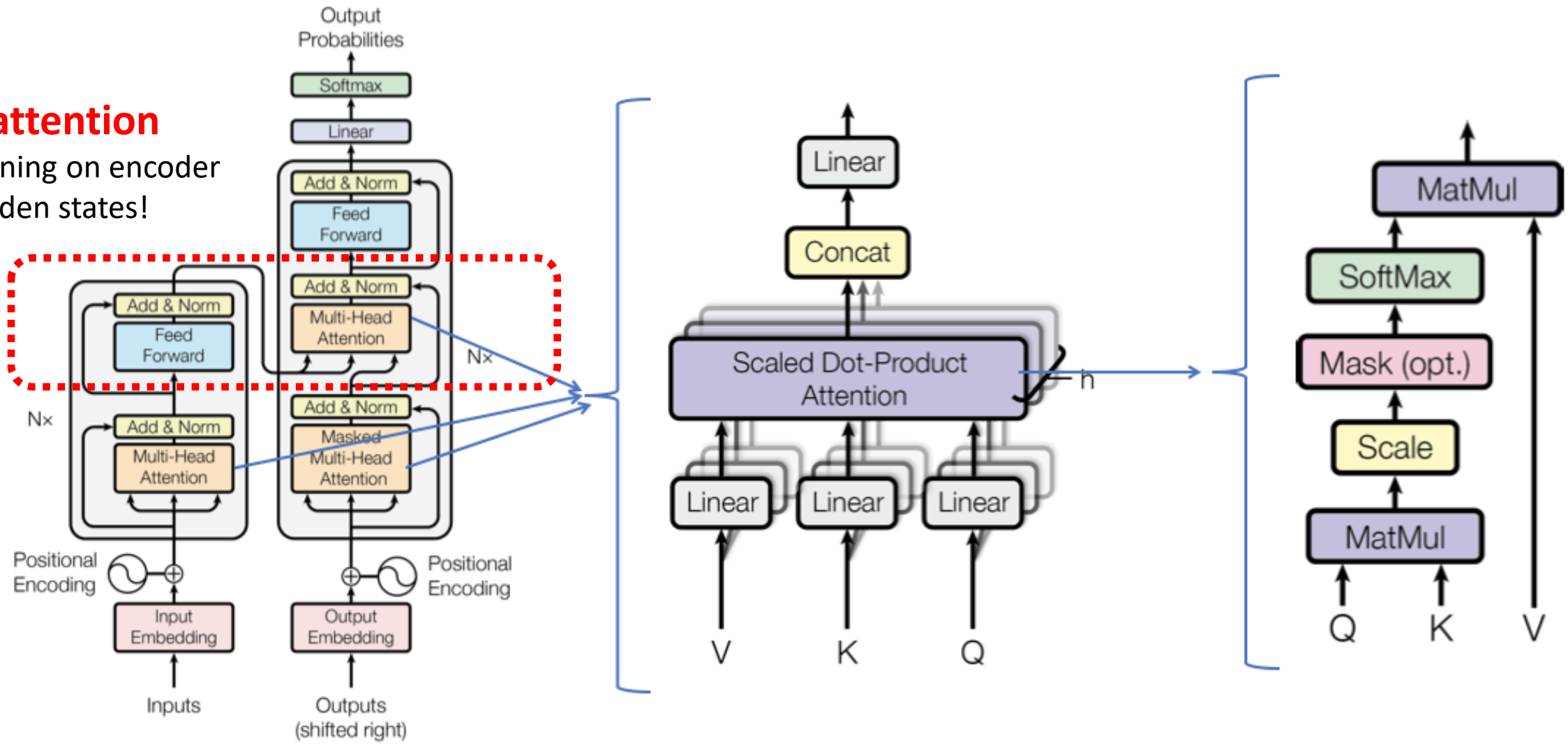
Transformer-Based Seq2Seq Model



Transformer-Based Seq2Seq Model

Cross-attention

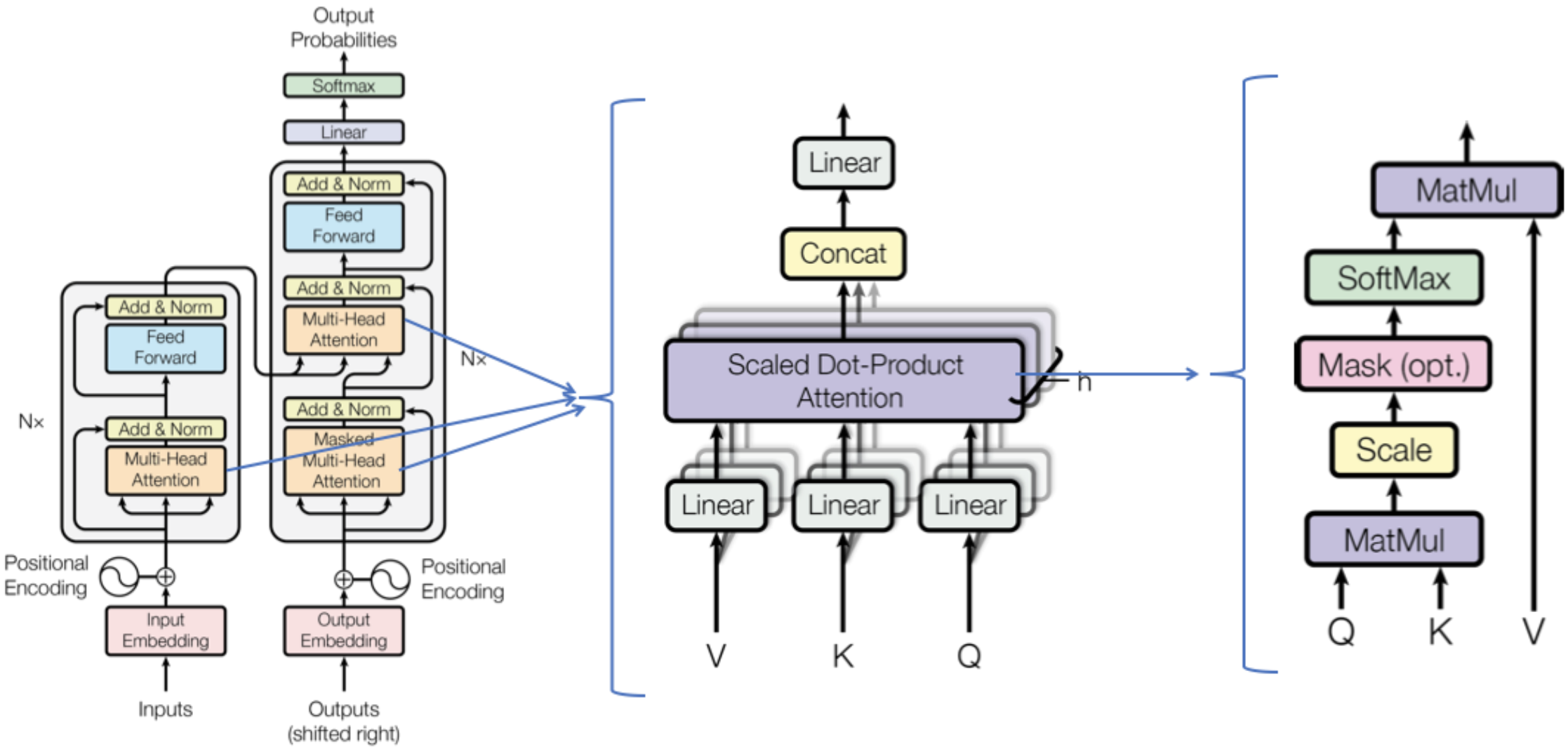
Conditioning on encoder hidden states!



Transformer-Based Seq2Seq Model

- Cross-Attention
  - The conditioning part of transformer
    - Decoder can generate texts conditioning on the input sequence
    - Just like standard attention in RNN seq2seq model
  - z_t: decoder SA module inputs
  - h_t: encoder output hidden states
  - For each decoder out_t, we attend on encoder hiddens
    - Query from decoder: $q_t = W^q z_t$
    - Key and value from encoder: $k_j = W^k h_j$; $v_j = W^v h_j$
    - $\alpha_{ij} = \mathrm{softmax}\left(\frac{q_i^T k_j}{\sqrt{d}}\right)$; $out_i = LN\left(\sum_j \alpha_{ij} v_j + z_i\right)$
      - Many practical variants can be implemented

Transformer-Based Seq2Seq Model



Transformer-Based Seq2Seq Model

- Machine translation with transformer (NIPS2017, Google)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformer-Based Seq2Seq Model

- Generating Wikipedia by Summarizing Long Sequences (ICLR2018, Google)
  - Document generation

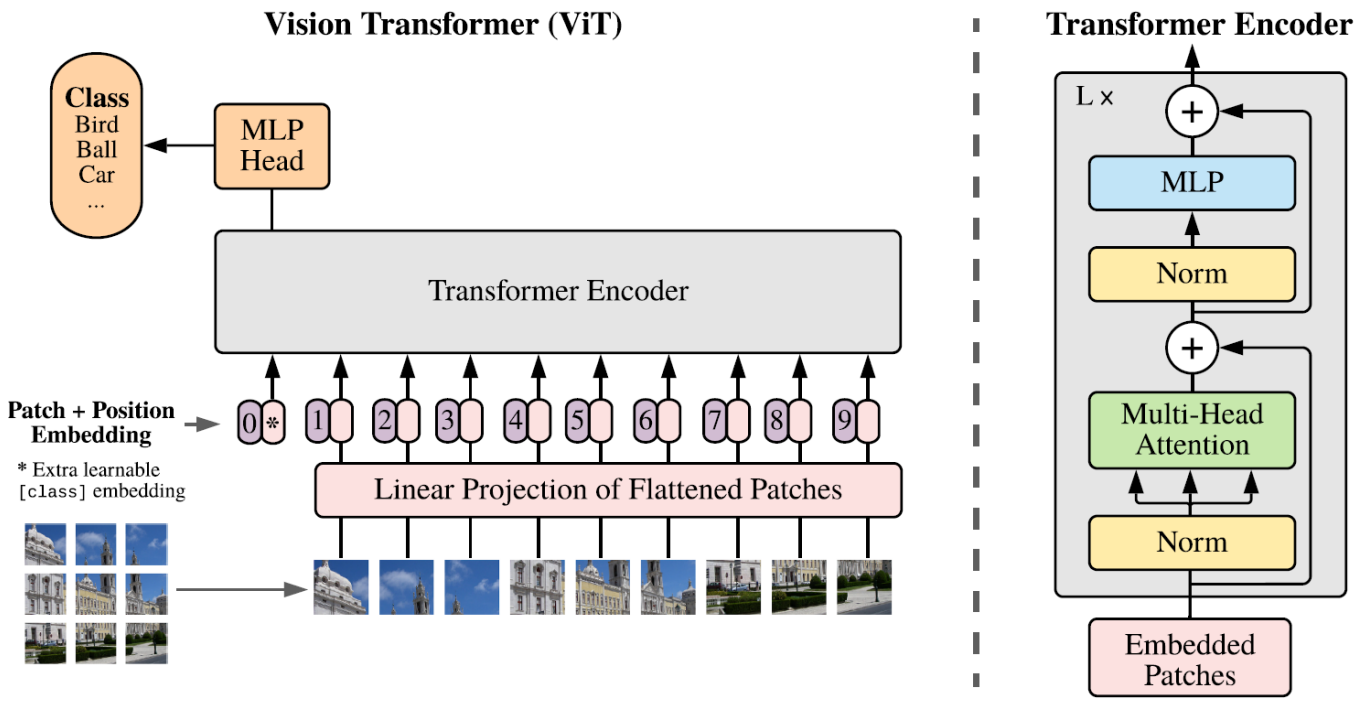
Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

Transformer Model

- Limitation of transformer
  - Quadratic computation cost
    - Linear for RNNs
    - What if we want to model sequence length of, say, $L > 10^4$
  - Follow-ups
    - Large-scale training: transformer-XL; XL-net (Zhilin Yang, et al, Google, 2020)
    - Projection tricks: Linformer (Facebook AI, $O(n)$ computation, 2020)
    - Math tricks: Performer (Google, $O(n)$ computation, 2020)
    - Sparse interactions: Big Bird (Google, 2020)
    - Fast and memory-efficient attention: Flash Attention (Tri Dao, et al, 2022)
    - Deeper transformers: DeepNet (MSRA, scaling transformers to 1000 layers, 2022)
    - Even Parallel RNN (make RNN great again):
      - RWKV RNN (Open-Source, 2023) & Mamba (Gu, Albert and Tri Dao, 2023)
    - More to learn in your NLP course ☺

Transformer Model for Images

- Vision Transformer (ViT, Google Brain, ICLR 2021, 3.3k citation)
  - *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*
  - Decompose an image to 16x16 patches and then apply transformer encoder



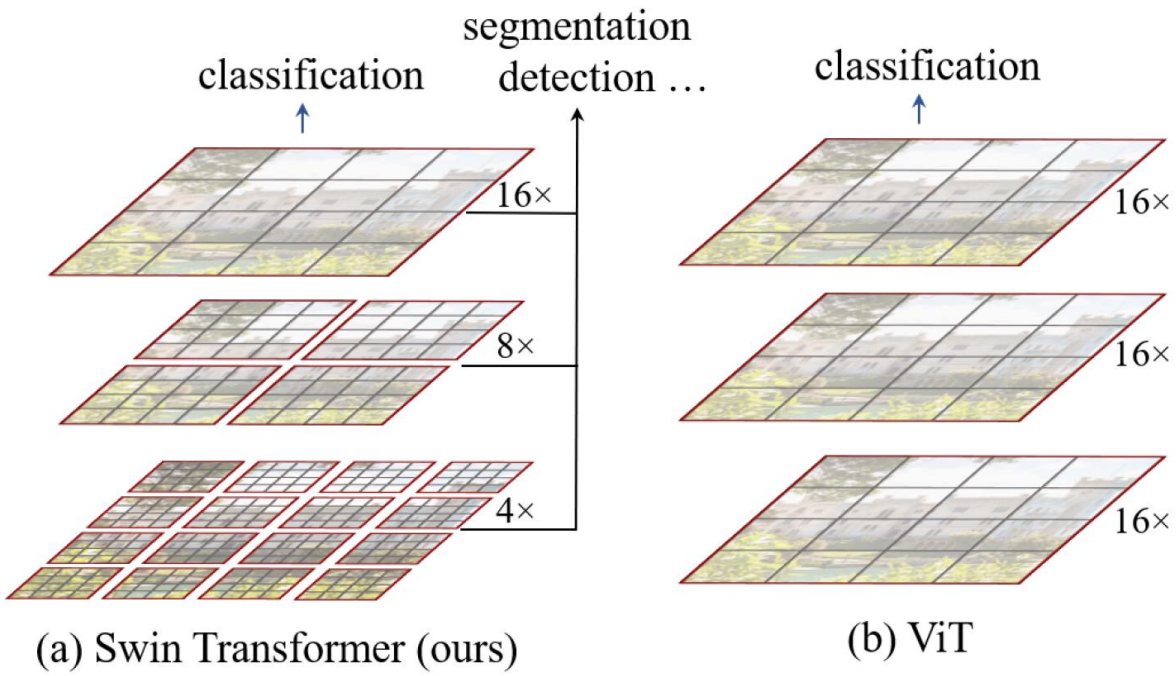
Transformer Model for Images

- Vision Transformer (ViT, Google Brain, ICLR 2021, 14k citation)
  - *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*
  - Decompose an image to 16x16 patches and then apply transformer encoder

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Transformer Model for Images

- Swin Transformer (MSRA, CVPR 2021 best paper)
  - Build hierarchical feature maps at different resolution
    - Self-attention only within each block (linear computation for image size)
    - Shifted block partitions to encode information between blocks



(a) Swin Transformer (ours)

(b) ViT

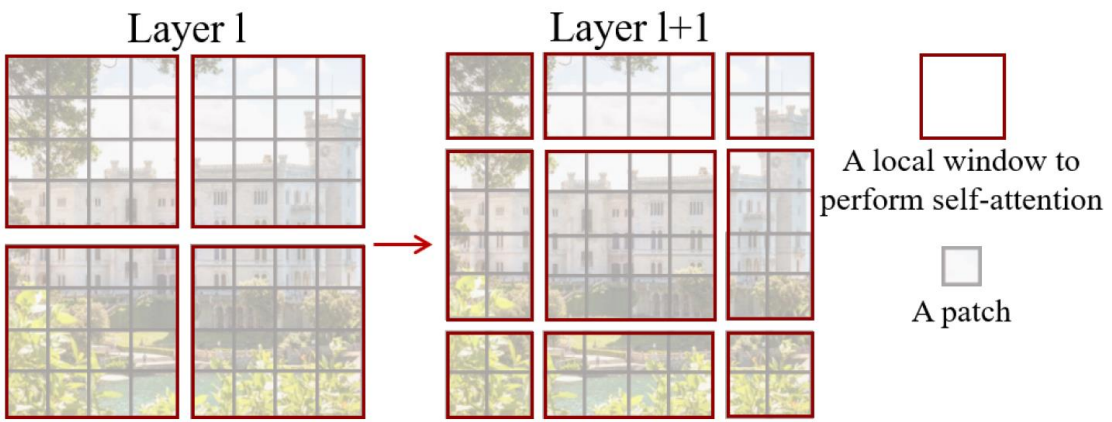


Figure 2. An illustration of the *shifted window* approach for com-

Transformer Model for Images

- Swin Transformer (MSRA, CVPR 2021 best paper)
  - Build hierarchical feature maps at different resolution
    - Self-attention only within each block (linear computation for image size)
    - Shifted block partitions to encode information between blocks

Method	mini-val		test-dev		#param.	FLOPs
	AP^{box}	AP^{mask}	AP^{box}	AP^{mask}		
RepPointsV2* [12]	-	-	52.1	-	-	-
GCNet* [7]	51.8	44.7	52.3	45.4	-	1041G
RelationNet++* [13]	-	-	52.7	-	-	-
SpineNet-190 [21]	52.6	-	52.8	-	164M	1885G
ResNeSt-200* [78]	52.5	-	53.3	47.1	-	-
EfficientDet-D7 [59]	54.4	-	55.1	-	77M	410G
DetectoRS* [46]	-	-	55.7	48.5	-	-
YOLOv4 P7* [4]	-	-	55.8	-	-	-
Copy-paste [26]	55.9	47.2	56.0	47.4	185M	1440G
X101-64 (HTC++)	52.3	46.0	-	-	155M	1033G
Swin-B (HTC++)	56.4	49.1	-	-	160M	1043G
Swin-L (HTC++)	57.1	49.5	57.7	50.2	284M	1470G
Swin-L (HTC++)*	**58.0**	**50.4**	**58.7**	**51.1**	284M	-

Table 2. Results on COCO object detection and instance segmentation. †denotes that additional decovolution layers are used to produce hierarchical feature maps. * indicates multi-scale testing.

ADE20K		val	test	#param.	FLOPs	FPS
Method	Backbone	mIoU	score			
DANet [23]	ResNet-101	45.2	-	69M	1119G	15.2
DLab.v3+ [11]	ResNet-101	44.1	-	63M	1021G	16.0
ACNet [24]	ResNet-101	45.9	38.5	-		
DNL [71]	ResNet-101	46.0	56.2	69M	1249G	14.8
OCRNet [73]	ResNet-101	45.3	56.0	56M	923G	19.3
UperNet [69]	ResNet-101	44.9	-	86M	1029G	20.1
OCRNet [73]	HRNet-w48	45.7	-	71M	664G	12.5
DLab.v3+ [11]	ResNeSt-101	46.9	55.1	66M	1051G	11.9
DLab.v3+ [11]	ResNeSt-200	48.4	-	88M	1381G	8.1
SETR [81]	T-Large‡	50.3	61.7	308M	-	-
UperNet	DeiT-S†	44.0	-	52M	1099G	16.2
UperNet	Swin-T	46.1	-	60M	945G	18.5
UperNet	Swin-S	49.3	-	81M	1038G	15.2
UperNet	Swin-B‡	51.6	-	121M	1841G	8.7
UperNet	Swin-L‡	**53.5**	**62.8**	234M	3230G	6.2

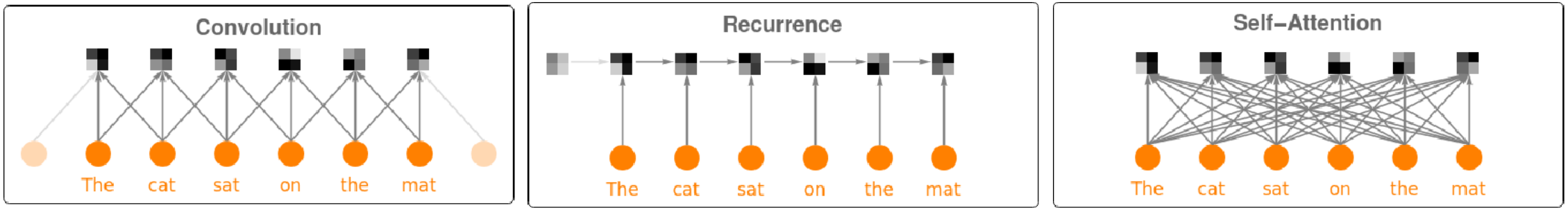
Table 3. Results of semantic segmentation on the ADE20K val and test set. † indicates additional deconvolution layers are used to produce hierarchical feature maps. ‡ indicates that the model is pre-trained on ImageNet-22K.

Story So Far

- Language model techniques
  - Learning: expensive softmax operator (NCE, H-softmax)
  - Inference: beam search
  - Embedding: contextualized embeddings
- Seq2Seq model
  - Generic architecture for conditioned language modeling
- Attention
  - Learning order/scale-invariant representations
  - Capture distant interactions
  - Transformer: attention is all you need for sequence modeling (since 2017)
    - …. and even images (since 2021)

Comparisons

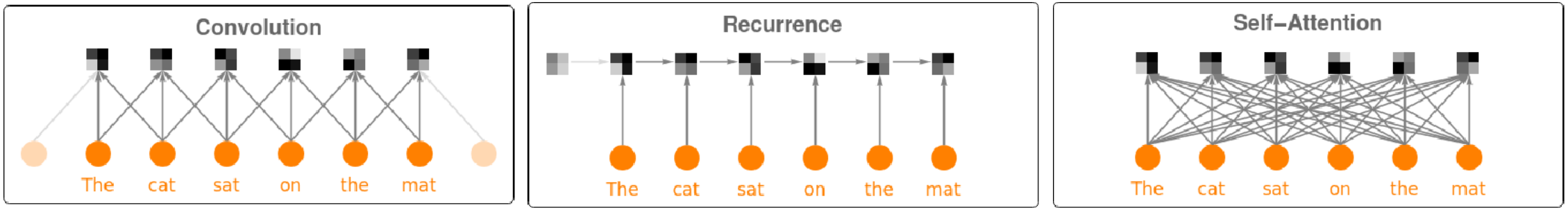
- CNN v.s. LSTM v.s. Transformer



- Contextualized embedding v.s. Transformer
  - Word2vec: static embedding
  - ELMo: pretraining bidirectional LSTMs for contextualized features
  - Transformer encoder: powerful bidirectional sequence encoder

Comparisons

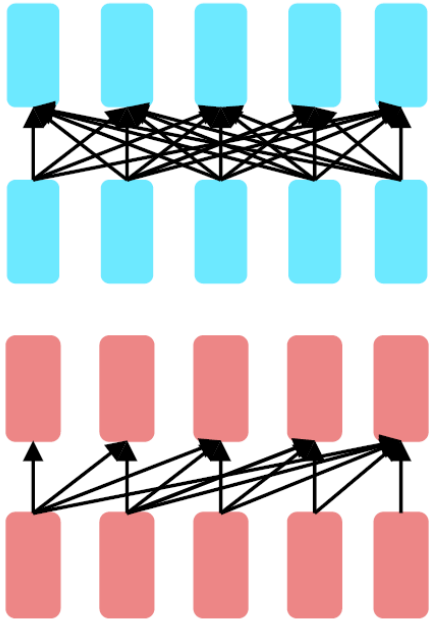
- CNN v.s. LSTM v.s. Transformer



- Contextualized embedding v.s. Transformer
  - Word2vec: static embedding
  - ELMo: pretraining bidirectional LSTMs for contextualized features
  - Transformer encoder: powerful bidirectional sequence encoder
  - Idea: pretraining large transformers!
    - Pretrained transformers also lead to good representations!
    - ***Foundation of most nowadays NLP applications***

Pretraining Transformers

- Collect a large amount of corpus and pretrain a large transformer
- For down-stream tasks, fine-tune the pretrained model
  - Or use the pretrained model to extract features
- How to pretrain a transformer on texts?
  - Pretrain an encoder
    - Bi-directional
  - Pretrain a decoder
    - Auto-regressive
  - Also both encoder and decoder

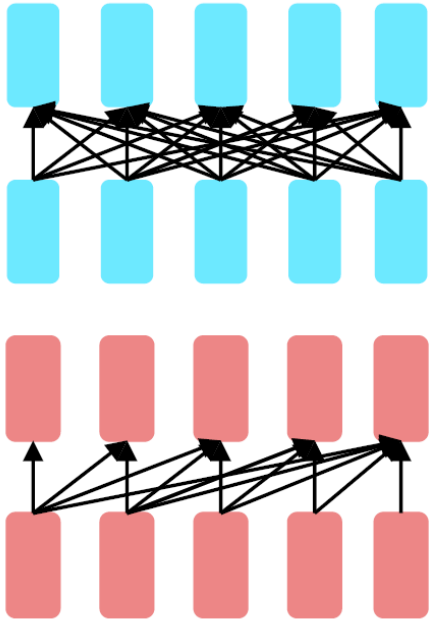


Encoders

Decoders

Pretraining Transformers

- Collect a large amount of corpus and pretrain a large transformer
- For down-stream tasks, fine-tune the pretrained model
  - Or use the pretrained model to extract features
- How to pretrain a transformer on texts?
  - Pretrain an encoder
    - Bi-directional
  - Pretrain a decoder
    - Auto-regressive
  - Also both encoder and decoder

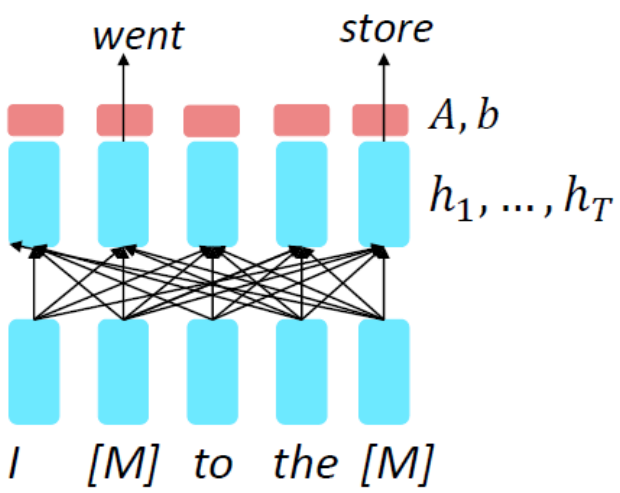


Encoders

Decoders

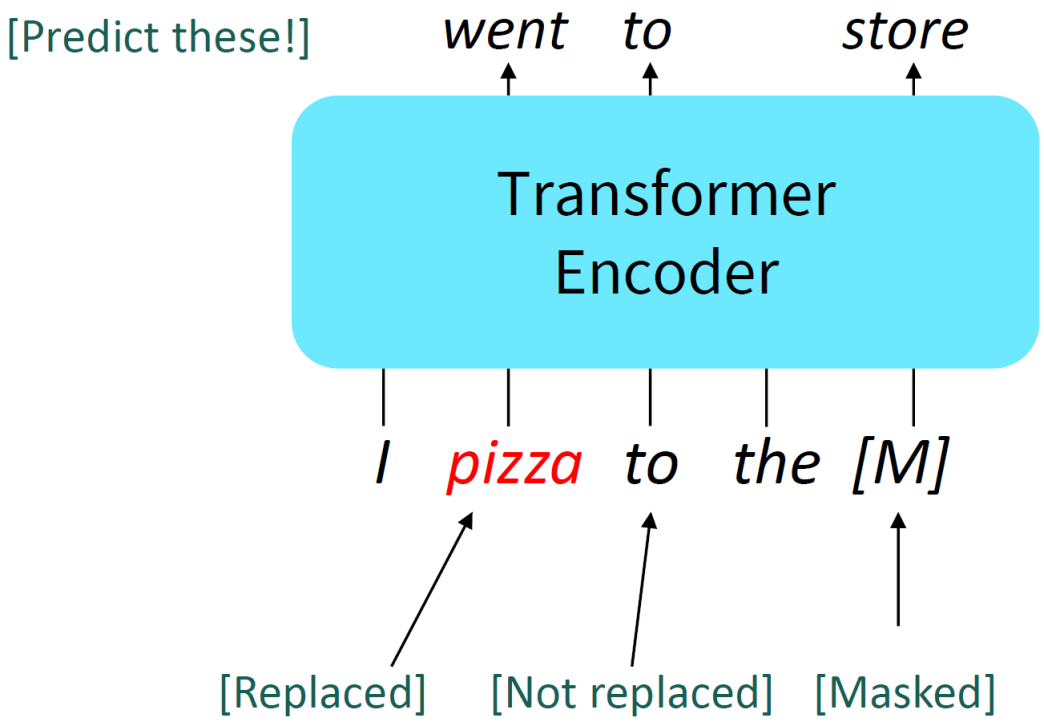
Pretraining Transformer Encoder

- Pretraining a bi-directional encoder
  - We cannot directly adopt language model learning
  - Idea: word prediction given contexts (similar to word2vec)
- Masked Language Model
  - Randomly “masked out” some words
  - Run full transformer encoder
  - Predict the words at masked positions
- Designed for feature extraction
  - More suitable for down-stream tasks



Pretraining Transformer Encoder

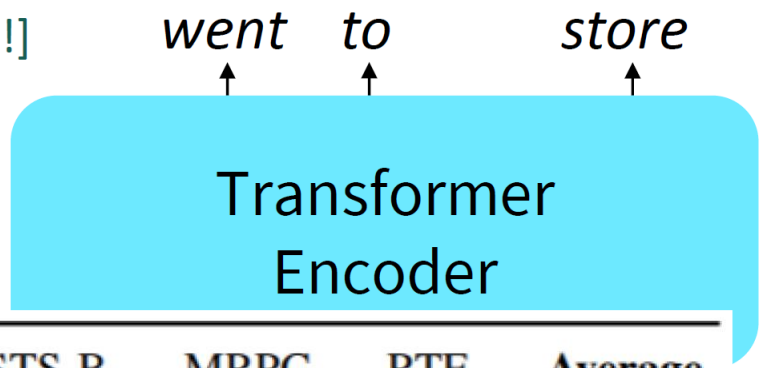
- BERT: Pre-training of Deep Bidirectional Transformers
  - Devlin et al, Google, 2018
    - BERT-base: 12 layers, 110M params
    - BERT-large: 24 layers, 340M params
    - Training on 64 TPUs in 4 days
    - Fine-tuning can be done in a single GPU
  - Masked language model
    - Masked out input words 80% of the time
    - Replace 10% words with random tokens
    - 10% words remain unchanged
    - Predict 15% of word tokens



Pretraining Transformer Encoder

- BERT: Pre-training of Deep Bidirectional Transformers
  - Devlin et al, Google, 2018
    - BERT-base: 12 layers, 110M params
    - BERT-large: 24 layers, 340M params
    - Training on 64 TPUs in 4 days
    - Fine-tuning can be done in a single GPU
  - Masked language model

[Predict these!]

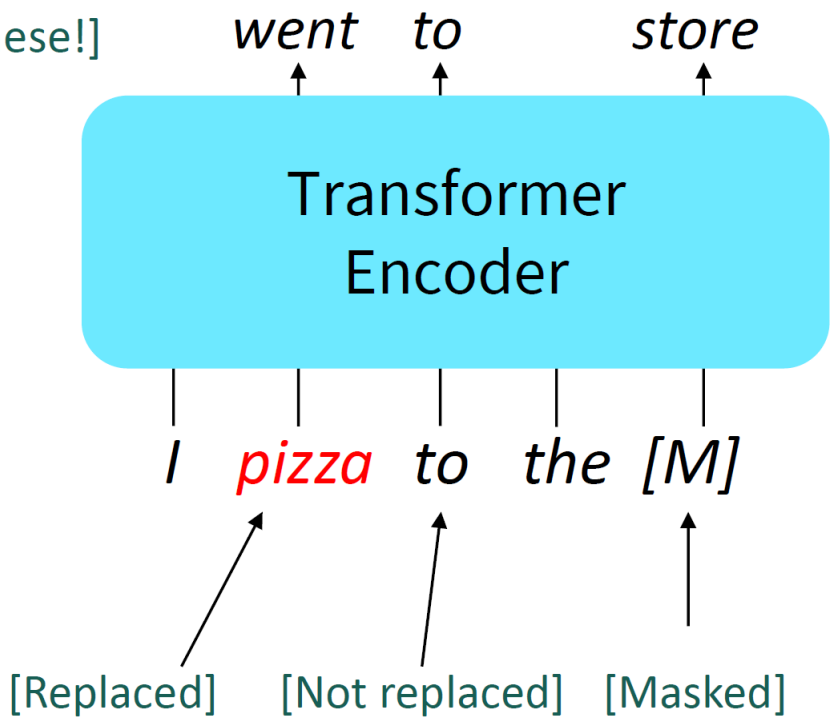


System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Pretraining Transformer Encoder

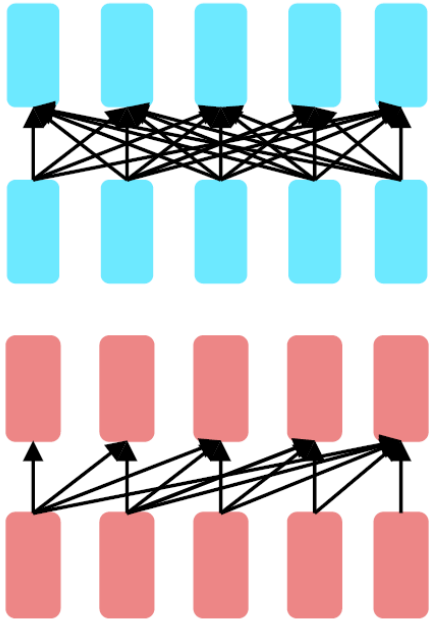
- BERT: Pre-training of Deep Bidirectional Transformers (Google 2018)
- RoBERTa: A Robustly Optimized BERT Pretraining Approach
  - Facebook AI, 2019
  - More compute, data and improved objective

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7



Pretraining Transformers

- Collect a large amount of corpus and pretrain a large transformer
- For down-stream tasks, fine-tune the pretrained model
  - Or use the pretrained model to extract features
- How to pretrain a transformer on texts?
  - Pretrain an encoder
    - Bi-directional (e.g., BERT, RoBERTa)
  - Pretrain a decoder
    - Auto-regressive
  - Also both encoder and decoder

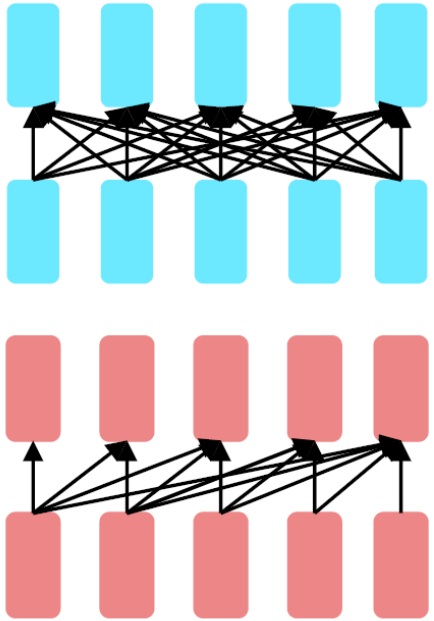


Encoders

Decoders

Pretraining Transformers

- Collect a large amount of corpus and pretrain a large transformer
- For down-stream tasks, fine-tune the pretrained model
  - Or use the pretrained model to extract features
- How to pretrain a transformer on texts?
  - Pretrain an encoder
    - Bi-directional
  - Pretrain a decoder
    - Auto-regressive
  - Also both encoder and decoder

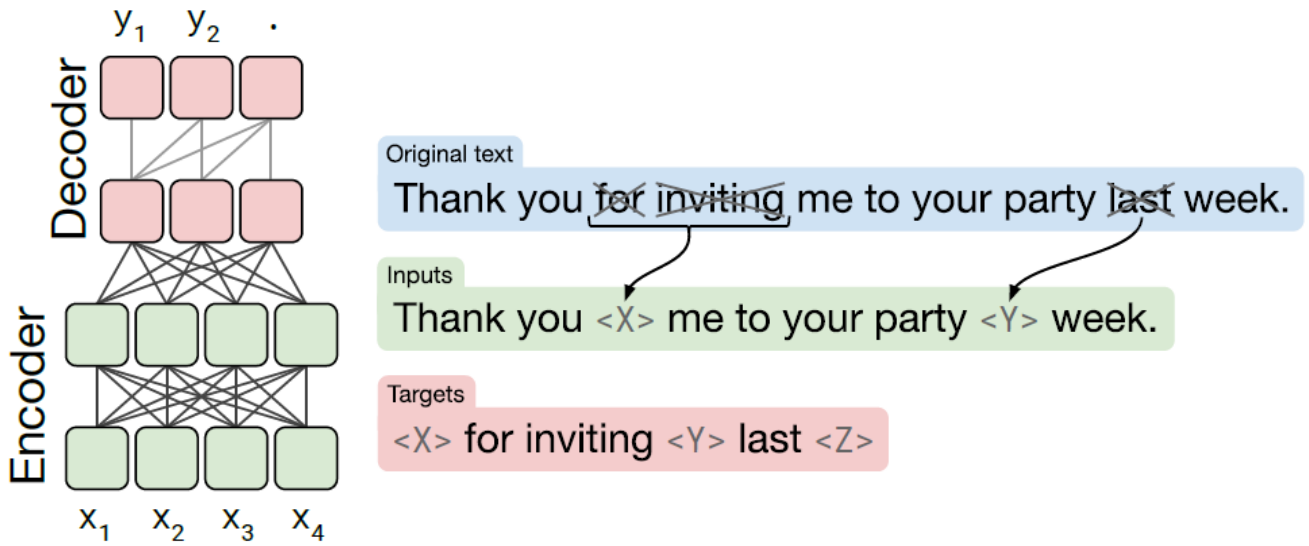


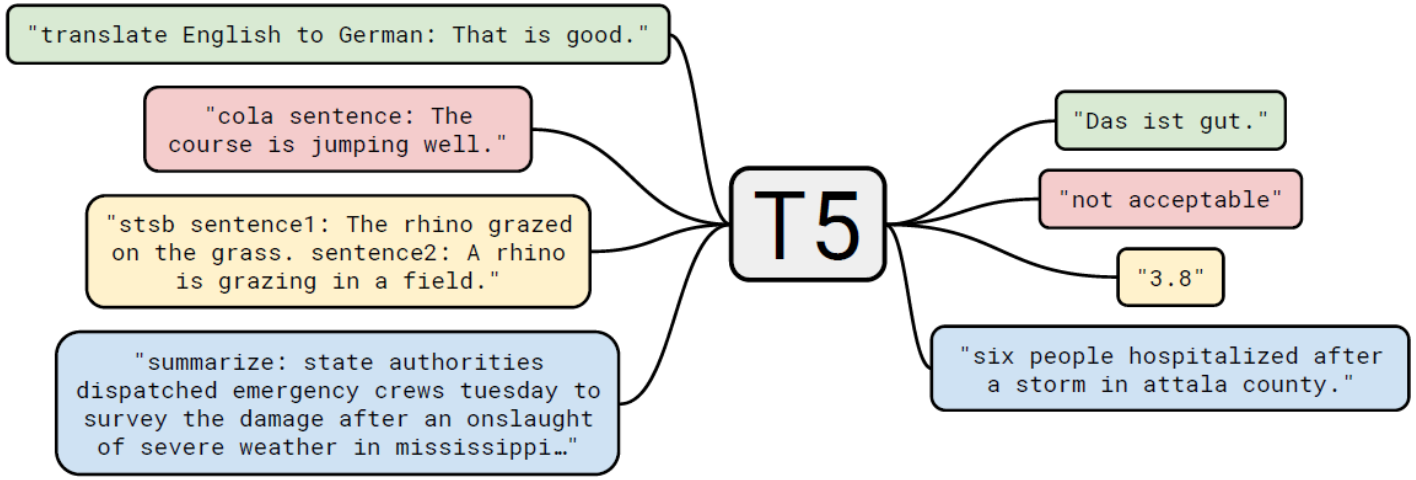
Encoders

Decoders

Pretraining a Multi-Task Encoder-Decoder

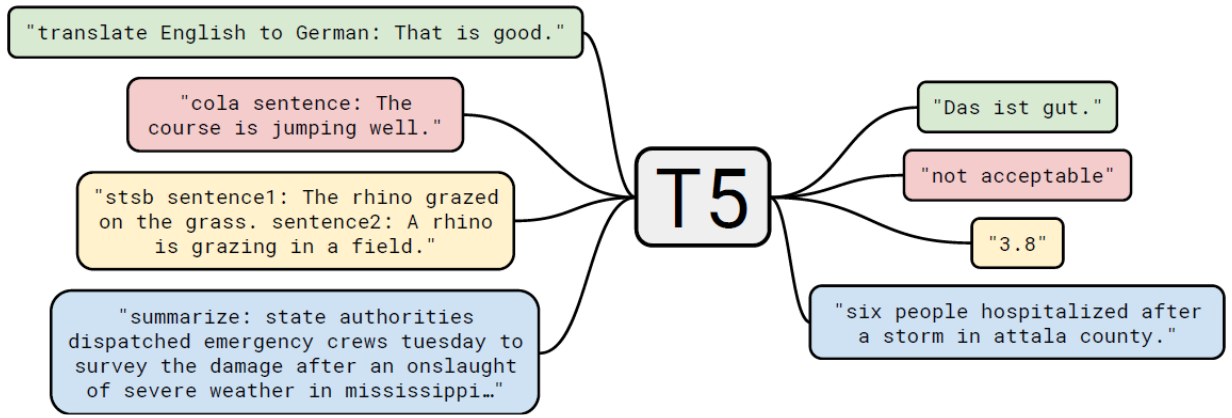
- T5 (Google, 2019)
  - **T**ext-**t**o-**T**ext **T**ransfer **T**ransformer
  - Multi-task training
  - Generalize to general QA

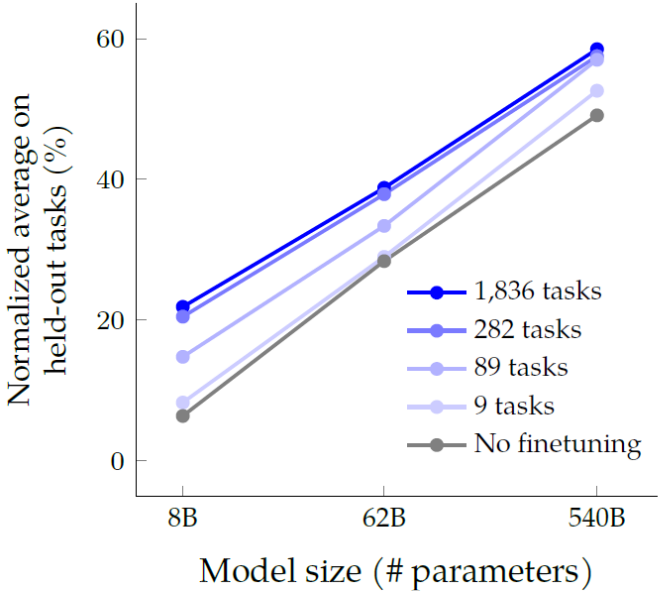


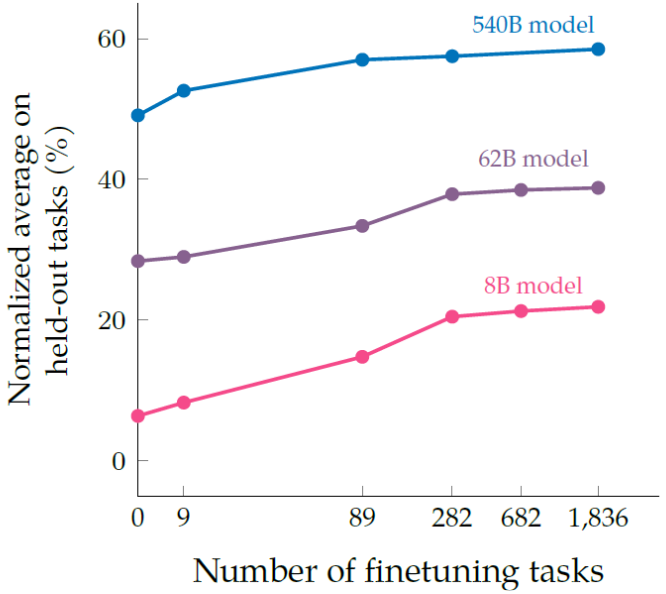


Pretraining a Multi-Task Encoder-Decoder

- T5 (Google, 2019)
  - **T**ext-**t**o-**T**ext **T**ransfer **T**ransformer
  - Multi-task training
  - Generalize to general QA
- FLAN-T5 (Google, 2022)
  - T5 with fine-tuning
  - Large-scale training

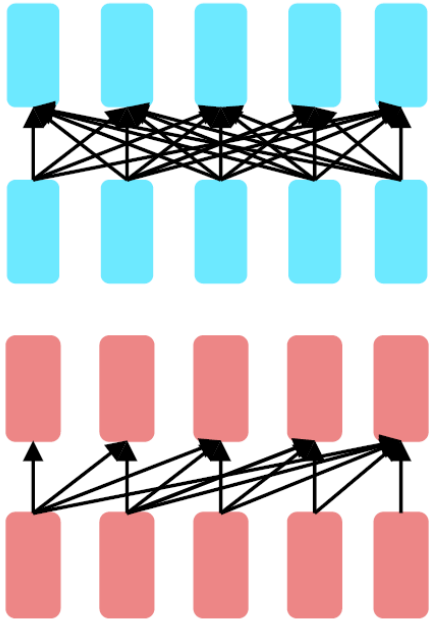






Pretraining Transformers

- Collect a large amount of corpus and pretrain a large transformer
- For down-stream tasks, fine-tune the pretrained model
  - Or use the pretrained model to extract features
- How to pretrain a transformer on texts?
  - Pretrain an encoder
    - Bi-directional
  - Pretrain a decoder
    - Auto-regressive
  - Also both encoder and decoder (T5/FLAN-T5)

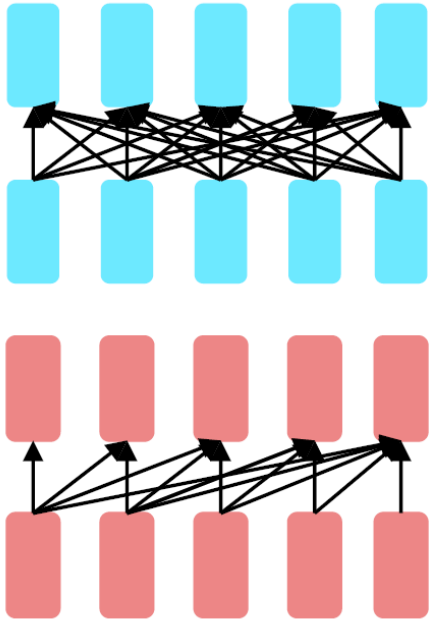


Encoders

Decoders

Pretraining Transformers

- Collect a large amount of corpus and pretrain a large transformer
- For down-stream tasks, fine-tune the pretrained model
  - Or use the pretrained model to extract features
- How to pretrain a transformer on texts?
  - Pretrain an encoder
    - Bi-directional
  - Pretrain a decoder
    - Auto-regressive
  - Also both encoder and decoder

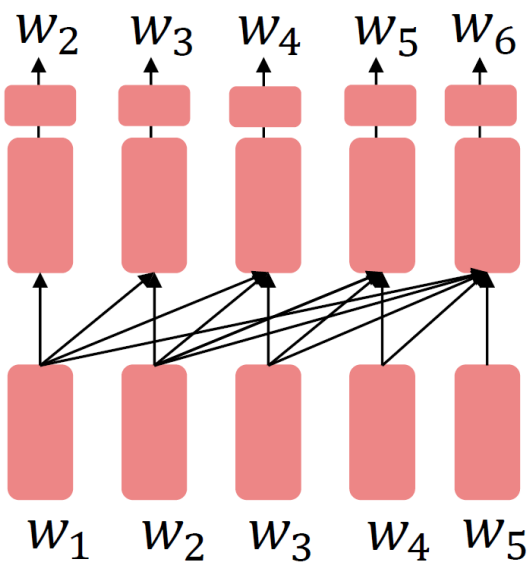


Encoders

Decoders

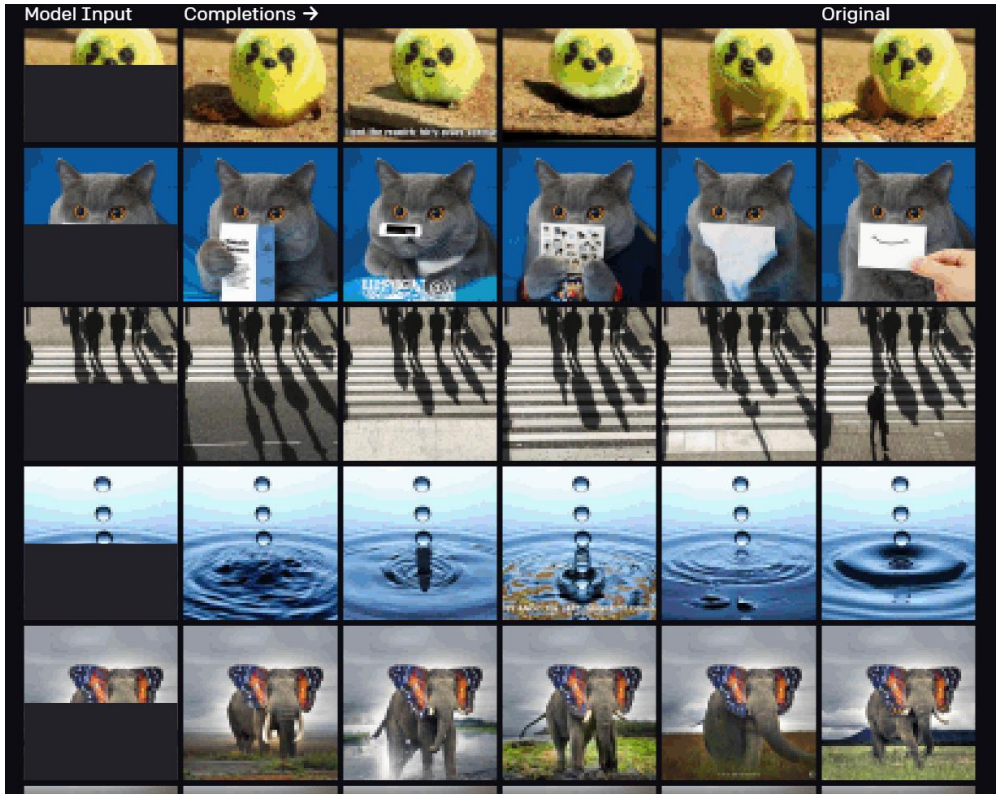
Pretraining Transformer Decoder

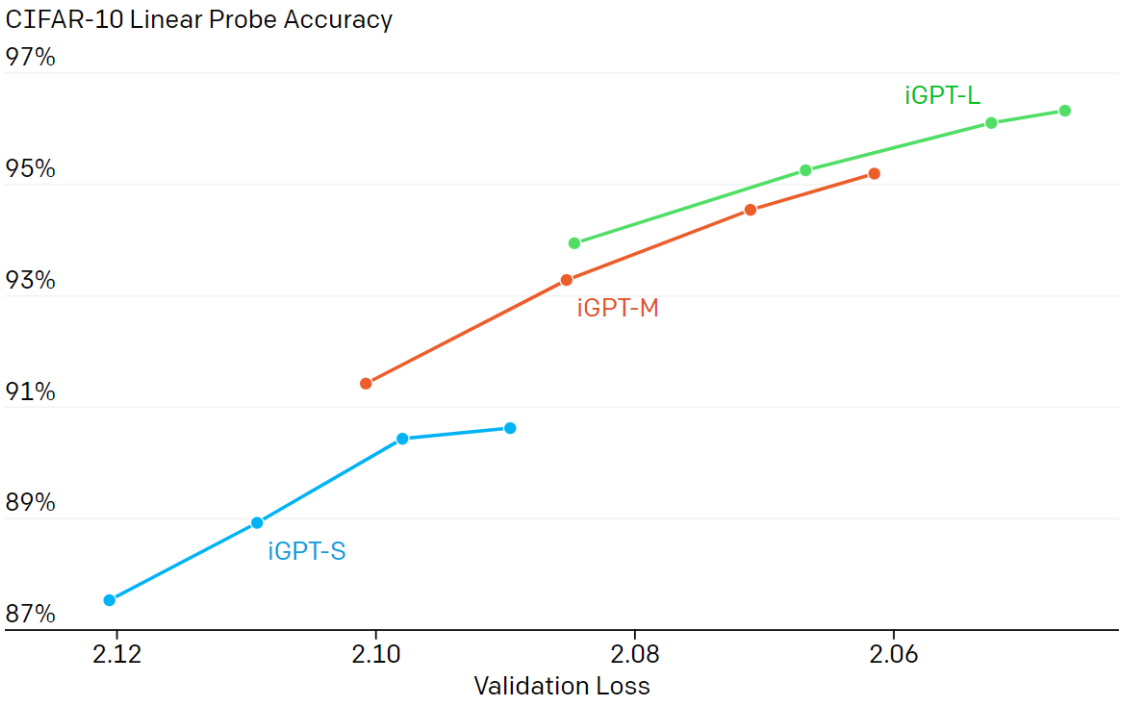
- Decoder Pretraining
  - Just train a language model over the corpus!
  - Great for generative tasks (e.g. text generation) & even beyond
- Generative Pretrained Transformer (GPT, Radford et al, OpenAI 2018)
  - 12-layers transformer, 768-d hidden, 3072-d MLP, BooksCorpus (>7k books)
- GPT-2 (Radford et al, OpenAI 2019.2)
  - 1.5B parameters, 40GB internet texts
- GPT-3 (OpenAI, 2020.5)
  - Language Model are Few-Shot Learners, 175B parameters
- Also ImageGPT (2020.1), ChatGPT (2022.11), GPT-4(2023)



Pretraining Transformer Decoder

- Image GPT (OpenAI, ICML 2020)
  - A large transformer-based generative model over image pixels
  - Learned features allow zero-shot classification (linear probing)





Pretraining Transformer Decoder

- GPT-3: Language models are few-shot learners (OpenAI, NIPS2020)
  - You may not need to fine-tune the model parameters for domain-specific task
    - *New research direction: prompt learning (https://arxiv.org/pdf/2107.13586.pdf)*

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

task description

examples

prompt

Code: `px.line(df.query("continent == 'Europe' and country == 'France'"), x='year', y='gdpPercap', color='country', log_y=False, log_x=False)`

Description: Actually, replace GDP with population

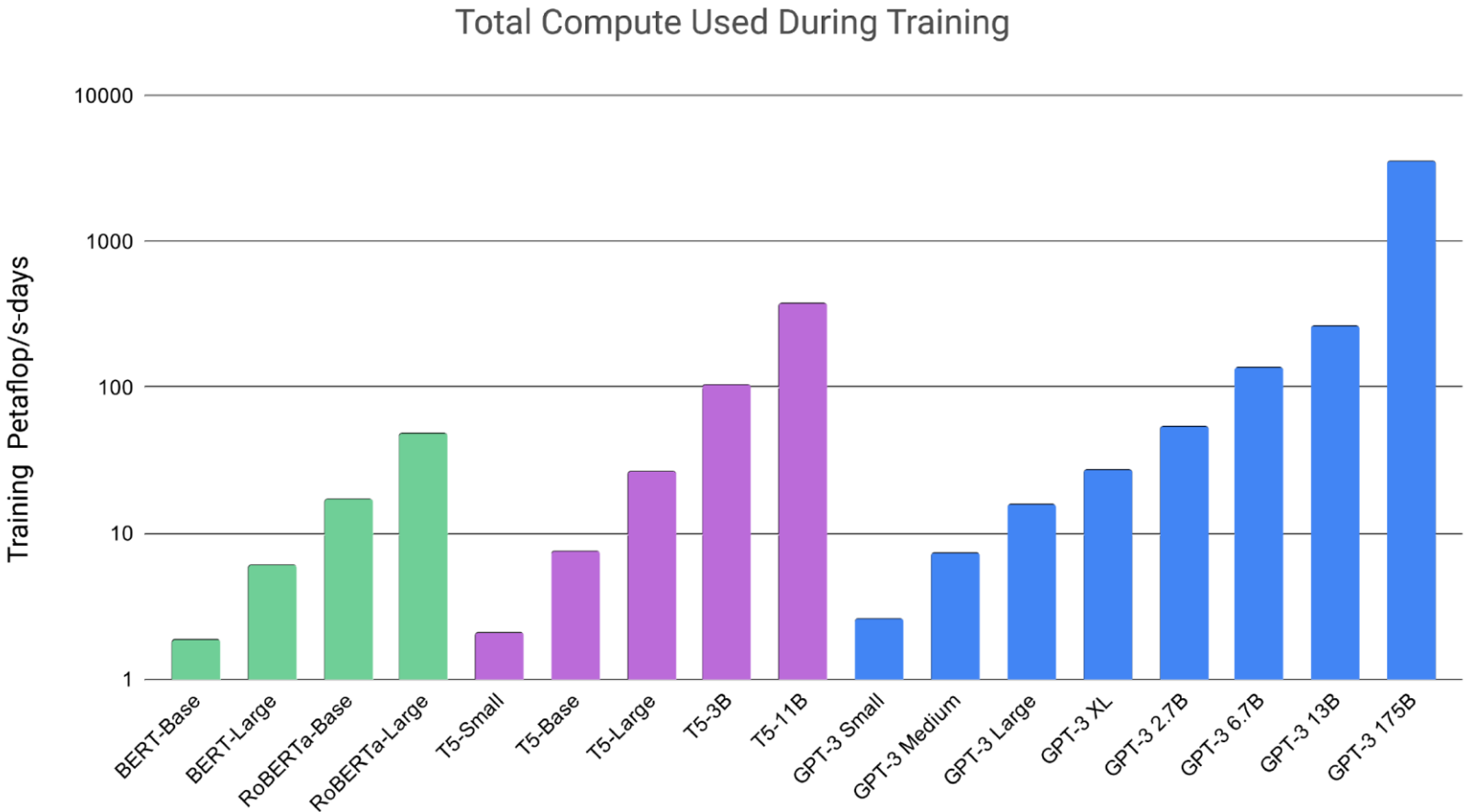
Code: `px.line(df.query("continent == 'Europe' and country == 'France'"), x='year', y='pop', color='country', log_y=False, log_x=False)`

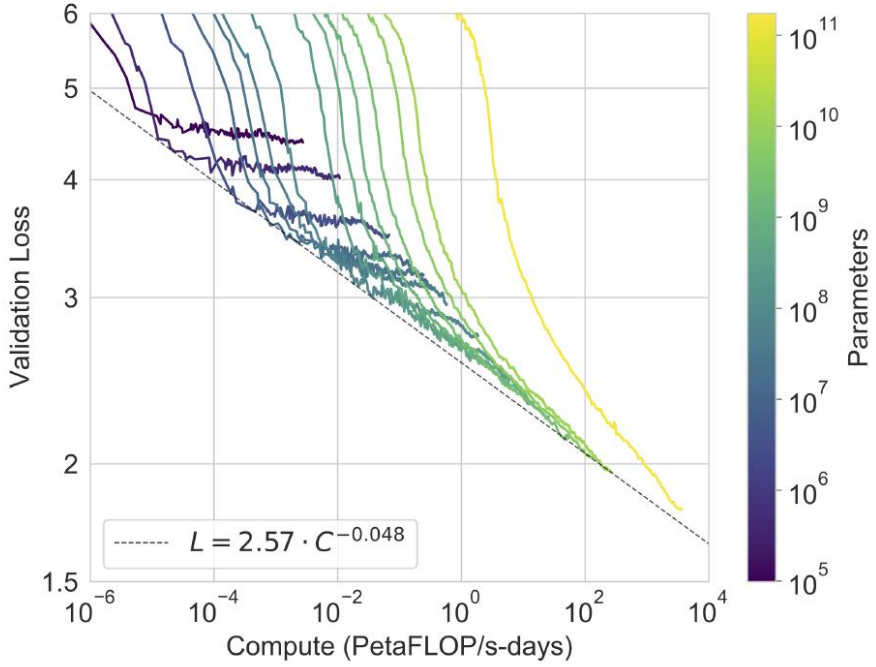
Description: Put y-axis on log scale

Code: `px.line(df.query("continent == 'Europe' and country == 'France'"), x='year', y='pop', color='country', log_y=True, log_x=False)`

Pretraining Transformer Decoder

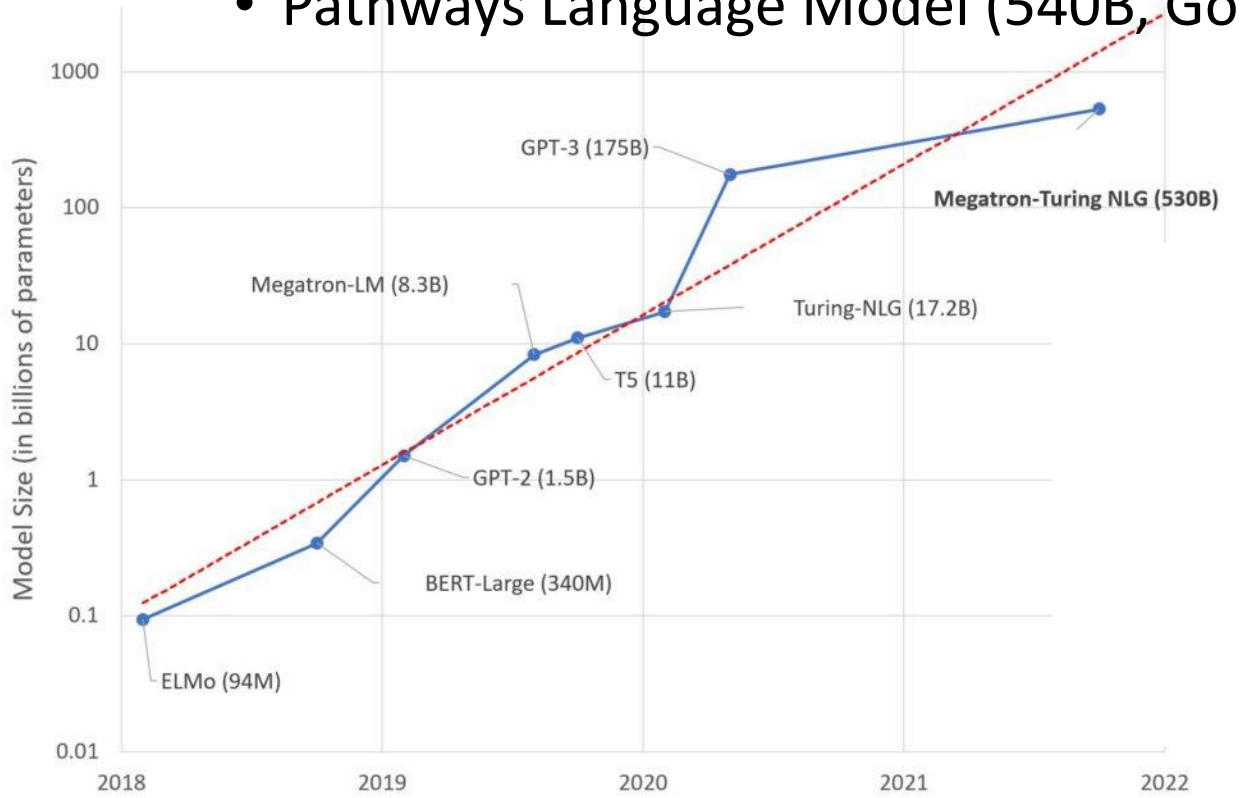
- Computation used by GPT-3

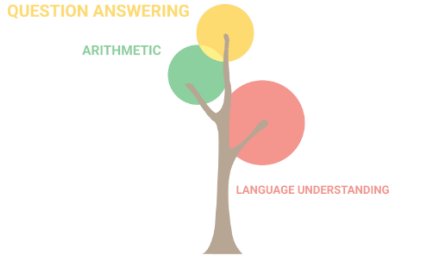




Pretraining Transformer Decoder

- A big ongoing race on training even larger language models
  - Megatron-Turing NLG (530B, Microsoft, 2021.10)
  - Pathways Language Model (540B, Google, 2022.4)

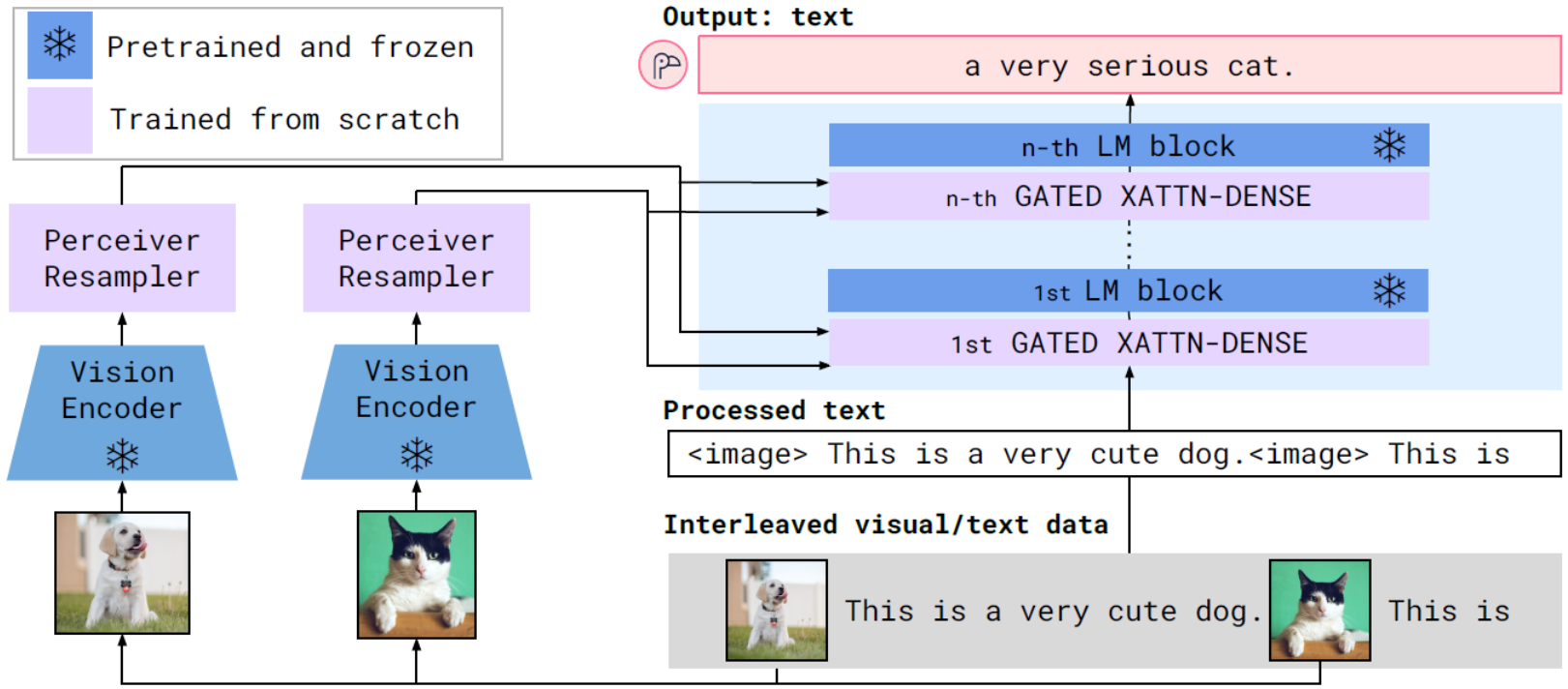




8 billion parameters

Pretraining Transformer Decoder

- Multi-Modal GPT to Unify Image and Text
  - Flamingo (DeepMind, 2022)



Pretraining Transformer Decoder

- Multi-Modal GPT to Unify Image and Text
  - Flamingo (DeepMind, 2022)
  - GPT-4 (OpenAI, 2023.3)

User What is unusual about this image?

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

User Below is part of the InstuctGPT paper. Could you read and summarize it to me?

Training language models to follow instructions with human feedback

Long Ouyang* Jeff Wu* Xu Jiang* Diogo Almeida* Carroll L. Wainwright* Pamela Mishkin* Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens Amanda Askell† Peter Welinder Paul Christiano*† Jan Leike* Ryan Lowe*

OpenAI

arXiv:2203.02155v1 [cs.CL] 4 Mar 2022

Abstract

Making language models bigger does not inherently make them better at following a user's intent. For example, large language models can generate outputs that are untruthful, toxic, or simply not helpful to the user. In other words, these models are not *aligned* with their users. In this paper, we show an avenue for aligning language models with user intent on a wide range of tasks by fine-tuning with human feedback. Starting with a set of labeler-written prompts and prompts submitted through the OpenAI API, we collect a dataset of labeler demonstrations of the desired model behavior, which we use to fine-tune GPT-3 using supervised learning. We then collect a dataset of rankings of model outputs, which we use to further fine-tune this supervised model using reinforcement learning from human feedback. We call the resulting models *InstructGPT*. In human evaluations on our prompt distribution, outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3, despite having 100x fewer parameters. Moreover, InstructGPT models show improvements in truthfulness and reductions in toxic output generation while having minimal performance regressions on public NLP datasets. Even though InstructGPT still makes simple mistakes, our results show that fine-tuning with human feedback is a promising direction for aligning language models with human intent.

1 Introduction

Large language models (LMs) can be "prompted" to perform a range of natural language processing (NLP) tasks, given some examples of the task as input. However, these models often express unintended behaviors such as making up facts, generating biased or toxic text, or simply not following user instructions (Bender et al. [2021], Bommasani et al. [2021], Kenton et al. [2021], Weidinger et al. [2021], Tamkin et al. [2021], Gehman et al. [2020]). This is because the language modeling objective

*Primary authors. This was a joint project of the OpenAI Alignment team. RL and JL are the team leads. Corresponding author: lowe@openai.com.

†Work done while at OpenAI. Current affiliations: AA: Anthropic, PC: Alignment Research Center.

Figure 1: Human evaluations of various models on our API prompt distribution, evaluated by how often outputs from each model were preferred to those from the 175B SFT model. Our InstructGPT models (PPO-ptx) as well as its variant trained without pretraining mix (PPO) significantly outperform the GPT-3 baselines (GPT, GPT prompted); outputs from our 1.3B PPO-ptx model are preferred to those from the 175B GPT-3. Error bars throughout the paper are 95% confidence intervals.

used for many recent large LMs—predicting the next token on a webpage from the internet—is different from the objective "follow the user's instructions helpfully and safely" (Radford et al. [2019], Brown et al. [2020], Fedus et al. [2021], Rae et al. [2021], Thoppilan et al. [2022]). Thus, we say that the language modeling objective is *misaligned*. Averting these unintended behaviors is especially important for language models that are deployed and used in hundreds of applications.

We make progress on aligning language models by training them to act in accordance with the user's intention (Leike et al. [2018]). This encompasses both explicit intentions such as following instructions and implicit intentions such as staying truthful, and not being biased, toxic, or otherwise harmful. Using the language of Askell et al. [2021], we want language models to be *helpful* (they should help the user solve their task), *honest* (they shouldn't fabricate information or mislead the user), and *harmless* (they should not cause physical, psychological, or social harm to people or the environment). We elaborate on the evaluation of these criteria in Section 3.6.

We focus on fine-tuning approaches to aligning language models. Specifically, we use reinforcement learning from human feedback (RLHF; Christiano et al. [2017], Stiennon et al. [2020]) to fine-tune GPT-3 to follow a broad class of written instructions (see Figure 2). This technique uses human preferences as a reward signal to fine-tune our models. We first hire a team of 40 contractors to label our data, based on their performance on a screening test (see Section 3.4 and Appendix B.1 for more details). We then collect a dataset of human-written demonstrations of the desired output behavior on (mostly English) prompts submitted to the OpenAI API[2] and some labeler-written prompts, and use this to train our supervised learning baselines. Next, we collect a dataset of human-labeled comparisons between outputs from our models on a larger set of API prompts. We then train a reward model (RM) on this dataset to predict which model output our labelers would prefer. Finally, we use this RM as a reward function and fine-tune our supervised learning baseline to maximize this reward using the PPO algorithm (Schulman et al. [2017]). We illustrate this process in Figure 2. This procedure aligns the behavior of GPT-3 to the stated preferences of a specific group of people (mostly our labelers and researchers), rather than any broader notion of "human values"; we discuss this further in Section 5.2. We call the resulting models *InstructGPT*.

We mainly evaluate our models by having our labelers rate the quality of model outputs on our test set, consisting of prompts from held-out customers (who are not represented in the training data). We also conduct automatic evaluations on a range of public NLP datasets. We train three model

[2]Specifically, we train on prompts submitted to earlier versions of the InstructGPT models on the OpenAI API Playground, which were trained only using demonstration data. We filter out prompts containing PII.

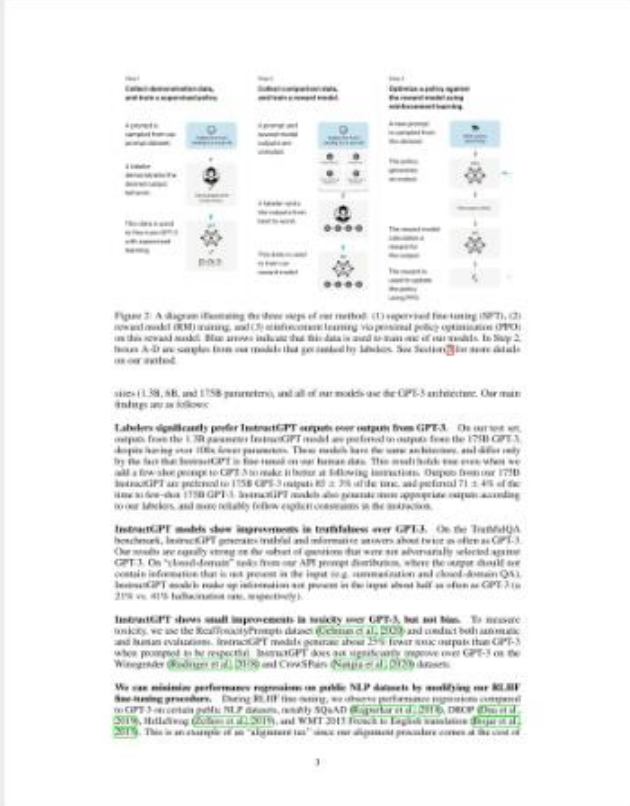


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A–D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

sizes (1.3B, 6B, and 175B parameters), and all of our models use the GPT-3 architecture. Our main findings are as follows:

Labelers significantly prefer InstructGPT outputs over outputs from GPT-3. On our test set, outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3, despite having over 100x fewer parameters. These models have the same architecture, and differ only by the fact that InstructGPT is fine-tuned on our human data. This result holds true even when we add a few-shot prompt to GPT-3 to make it better at following instructions. Outputs from our 175B InstructGPT are preferred to 175B GPT-3 outputs 85 ± 3% of the time, and preferred 71 ± 4% of the time to few-shot 175B GPT-3. InstructGPT models also generate more appropriate outputs according to our labelers, and more reliably follow explicit constraints in the instruction.

InstructGPT models show improvements in truthfulness over GPT-3. On the TruthfulQA benchmark, InstructGPT generates truthful and informative answers about twice as often as GPT-3. Our results are equally strong on the subset of questions that were not adversarially selected against GPT-3. On "closed-domain" tasks from our API prompt distribution, where the output should not contain information that is not present in the input (e.g. summarization and closed-domain QA), InstructGPT models make up information not present in the input about half as often as GPT-3 (a 21% vs. 41% hallucination rate, respectively).

InstructGPT shows small improvements in toxicity over GPT-3, but not bias. To measure toxicity, we use the RealToxicityPrompts dataset (Gehman et al. [2020]) and conduct both automatic and human evaluations. InstructGPT models generate about 25% fewer toxic outputs than GPT-3 when prompted to be respectful. InstructGPT does not significantly improve over GPT-3 on the Winogender (Rudinger et al. [2018]) and CrowS-Pairs (Nangia et al. [2020]) datasets.

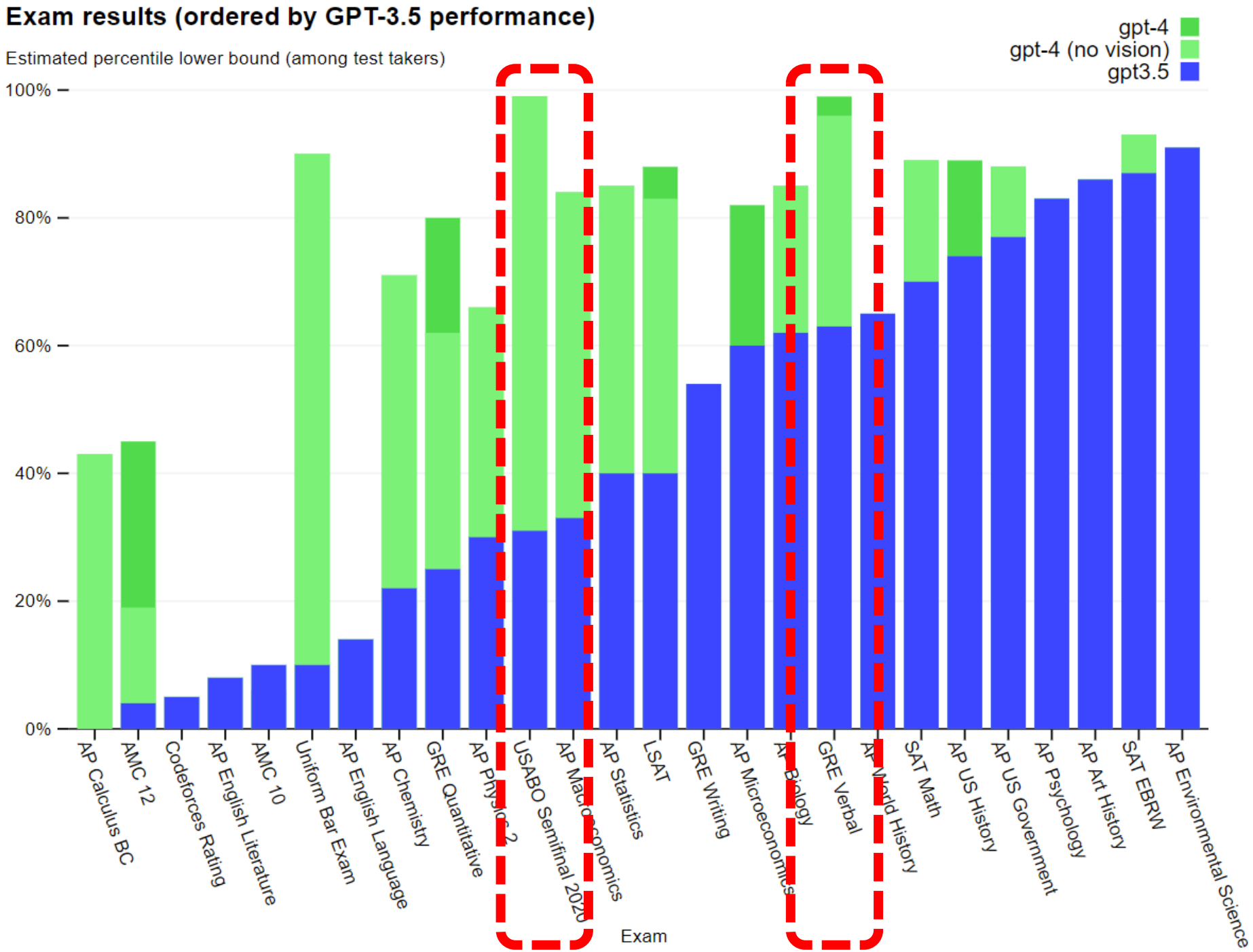
We can minimize performance regressions on public NLP datasets by modifying our RLHF fine-tuning procedure. During RLHF fine-tuning, we observe performance regressions compared to GPT-3 on certain public NLP datasets, notably SQuAD (Rajpurkar et al. [2016]), DROP (Dua et al. [2019]), HellaSwag (Zellers et al. [2019]), and WMT 2015 French to English translation (Bojar et al. [2015]). This is an example of an "alignment tax" since our alignment procedure comes at the cost of

3

GPT-4 The InstructGPT paper focuses on training large language models to follow instructions with human feedback. The authors note that making language models larger doesn't inherently make them better at following a user's intent. Large models can generate outputs that are untruthful, toxic, or simply unhelpful.

Pret

- Mul
  - F
  - G


Exam results (ordered by GPT-3.5 performance)
Estimated percentile lower bound (among test takers)
gpt-4
gpt-4 (no vision)
gpt3.5
100%
80%
60%
40%
20%
0%
AP Calculus BC
AMC 12
Codeforces Rating
AP English Literature
AMC 10
Uniform Bar Exam
AP English Language
AP Chemistry
GRE Quantitative
AP Physics 2
USABO Semifinal 2020
AP Macroeconomics
AP Statistics
LSAT
GRE Writing
AP Microeconomics
AP Biology
GRE Verbal
AP World History
SAT Math
AP US History
AP US Government
AP Psychology
AP Art History
SAT EBRW
AP Environmental Science
Exam

Pretraining Transformer Decoder

- Open-Source GPT (Zoo)
  - Meta – Llama series
    - 2022.2: LLaMA 1 7B, 13B, 33B & 65B
    - 2023.7: LLaMA 2 7B, 13B & 70B
    - 2024.4: LLaMA 3 8B & 70B

Pretraining Transformer Decoder

- Open-Source GPT (Zoo)
  - Meta – Llama series
    - 2022.2: LLaMA 1 7B, 13B, 33B & 65B
    - 2023.7: LLaMA 2 7B, 13B & 70B
    - 2024.4: LLaMA 3 8B & 70B
  - Stanford – Alpaca (2023.3 & 7B)

Pretraining Transformer Decoder

- Open-Source GPT (Zoo)
  - Meta – Llama series
    - 2022.2: LLaMA 1 7B, 13B, 33B & 65B
    - 2023.7: LLaMA 2 7B, 13B & 70B
    - 2024.4: LLaMA 3 8B & 70B
  - Stanford – Alpaca (2023.3 & 7B)
  - LMSYS – Vicuna
    - 2023.3: Vicuna-v1.1-7B & 13B
    - 2023.8: Vicuna-v1.3-7B, 13B & 33B

Pretraining Transformer Decoder

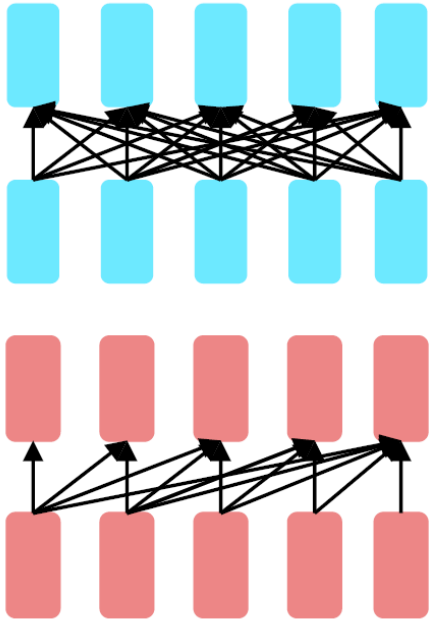
- Open-Source GPT (Zoo)
  - Meta – Llama series
    - 2022.2: LLaMA 1 7B, 13B, 33B & 65B
    - 2023.7: LLaMA 2 7B, 13B & 70B
    - 2024.4: LLaMA 3 8B & 70B
  - Stanford – Alpaca (2023.3 & 7B)
  - LMSYS – Vicuna
    - 2023.3: Vicuna-v1.1-7B & 13B
    - 2023.8: Vicuna-v1.3-7B, 13B & 33B
  - Mistral AI (a French company)
    - 2023.9: Mistral 7B
    - 2023.12: Mixtral 8x7B
    - 2024.4: Mixtral 8x22B
      - The first open-source **moe** GPT

Pretraining Transformer Decoder

- Open-Source GPT (Zoo)
  - Meta – Llama series
    - 2022.2: LLaMA 1 7B, 13B, 33B & 65B
    - 2023.7: LLaMA 2 7B, 13B & 70B
    - 2024.4: LLaMA 3 8B & 70B
  - Stanford – Alpaca (2023.3 & 7B)
  - LMSYS – Vicuna
    - 2023.3: Vicuna-v1.1-7B & 13B
    - 2023.8: Vicuna-v1.3-7B, 13B & 33B
  - Mistral AI (a French company)
    - 2023.9: Mistral 7B
    - 2023.12: Mixtral 8x7B
    - 2024.4: Mixtral 8x22B
      - The first open-source **moe** GPT
- Some Chinese Open-Source GPT
  - 智谱 – GLM Series
    - 2022.8: GLM-130B
    - 2023.3: ChatGLM-6B
    - 2023.6: ChatGLM2-6B
    - 2023.10: ChatGLM3-6B
  - 深度求索 – DeepSeek series
    - 2023.11 DeepSeek-7B & 67B
  - 零一万物 – Yi series
    - 2023.11 Yi-6B & Yi-34B
  - Also a lot of fine-tuned Llama models
    - Good luck with your coding project ☺

Pretraining Transformers

- Collect a large amount of corpus and pretrain a large transformer
- For down-stream tasks, fine-tune the pretrained model
  - Or use the pretrained model to extract features
- How to pretrain a transformer on texts?
  - Pretrain an encoder
    - Bi-directional
  - Pretrain a decoder
    - Auto-regressive (e.g., GPT-X and more)
  - Also both encoder and decoder



Encoders

Decoders

Summary

- Language Model & Sequence to Sequence Model
  - Fundamental ideas and methods for sequence modeling/tasks
- Attention Mechanism
  - So far the most successful idea for sequence data in deep learning
  - A scale/order-invariant representation
  - Transformer: a fully attention-based model for sequence data
  - Transformer + Pretraining: the core idea in today's NLP & AI tasks
- LSTM is still useful in lightweight scenarios

Thanks