# Attention Mechanism

# Machine Translation

- Before 2014: Statistical Machine  Translation (SMT)
  - Extremely complex systems that require massive human efforts
  - Separately designed components
  - A lot of feature engineering
  - Lots of linguistic domain knowledge and expertise

- Before 2016:
  - Google Translate is based on statistical machine learning

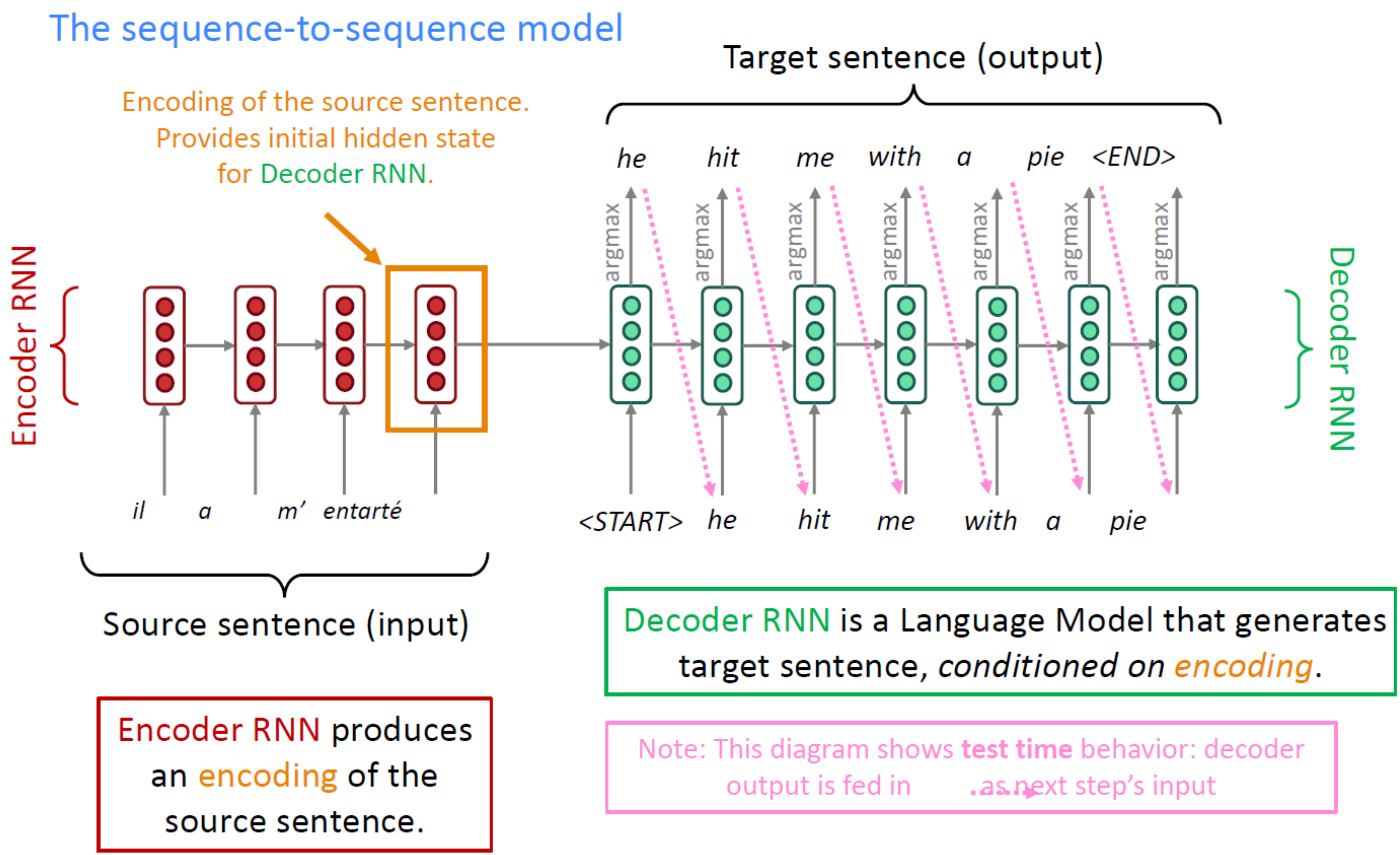
- What happened in 2014?
  - Neural machine translation (NMT)

# Sequence to Sequence Model

- Neural Machine Translation (NMT)
  - Learning to translate via a **single end-to-end** neural network.
  - Source language sentence $X$, target language sentence $Y = f(X; \theta)$

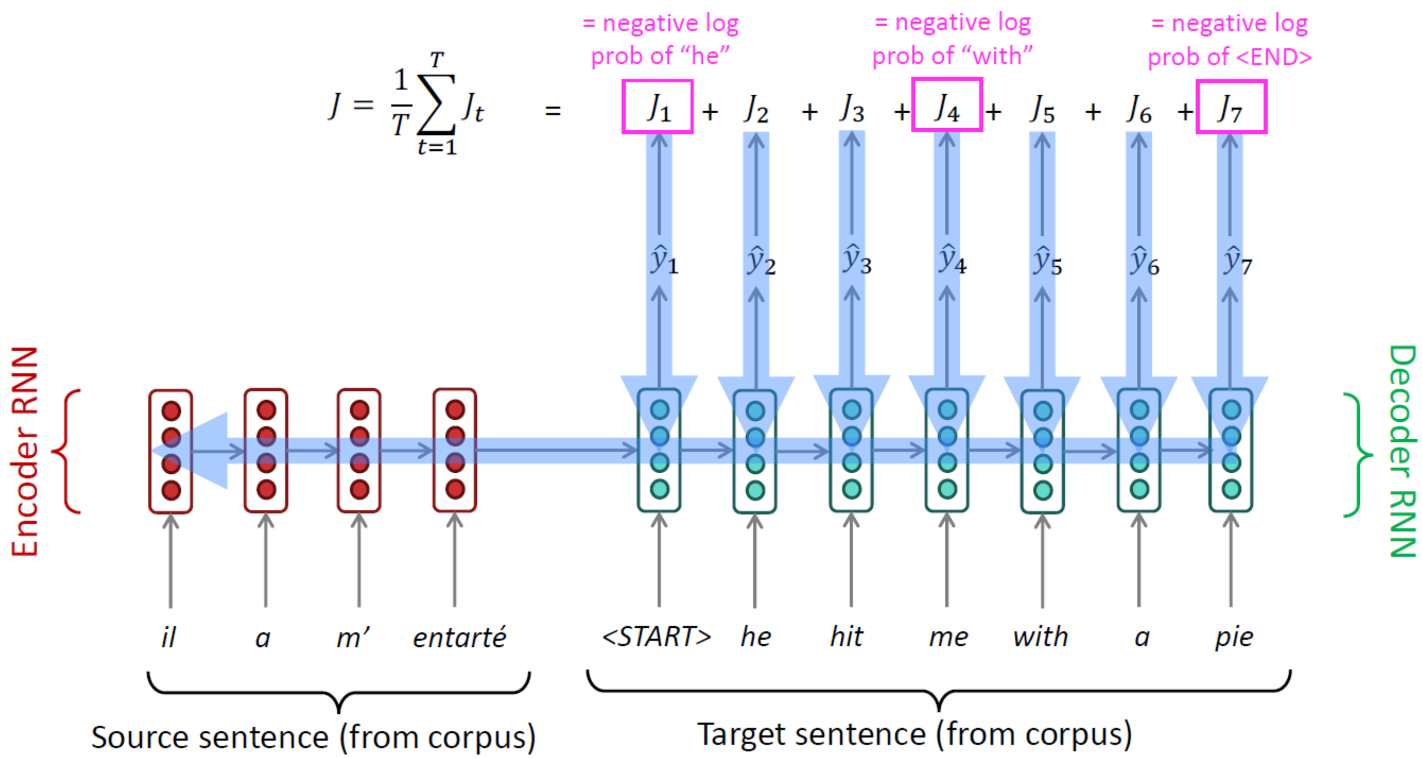
- Sequence to Sequence Model (Seq2Seq, Sutskever et al. , '14)
  - Two RNNs: $f_{enc}$ and $f_{dec}$
  - Encoder $f_{enc}$:
    - Takes $X$ as input, and output the initial hidden state for decoder
    - Can use bidirectional RNN
  - Decoder $f_{dec}$:
    - It takes in the hidden state from $f_{enc}$ to generate $Y$
    - Can use autoregressive language model

# Sequence to Sequence Model



# Training Sequence to Sequence Model

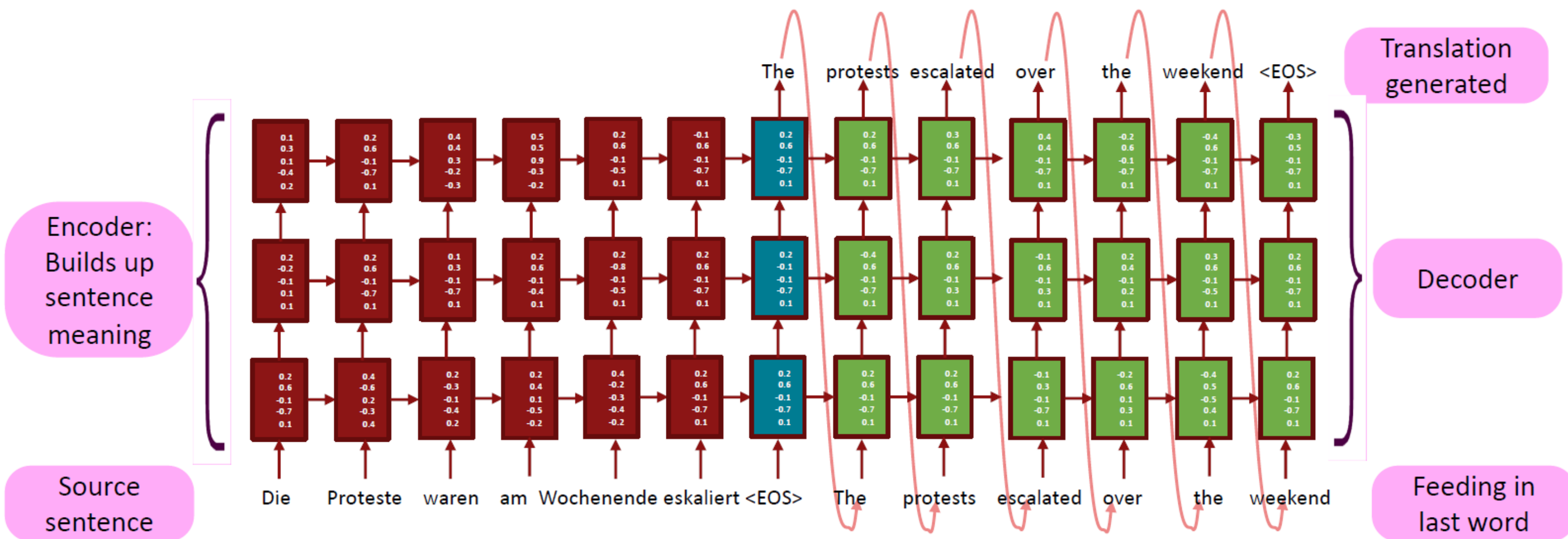
- Collect a huge paired dataset and train it end-to-end via BPTT
- Loss induced by MLE $P(Y \mid X) = P(Y \mid f_{enc}(X))$




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

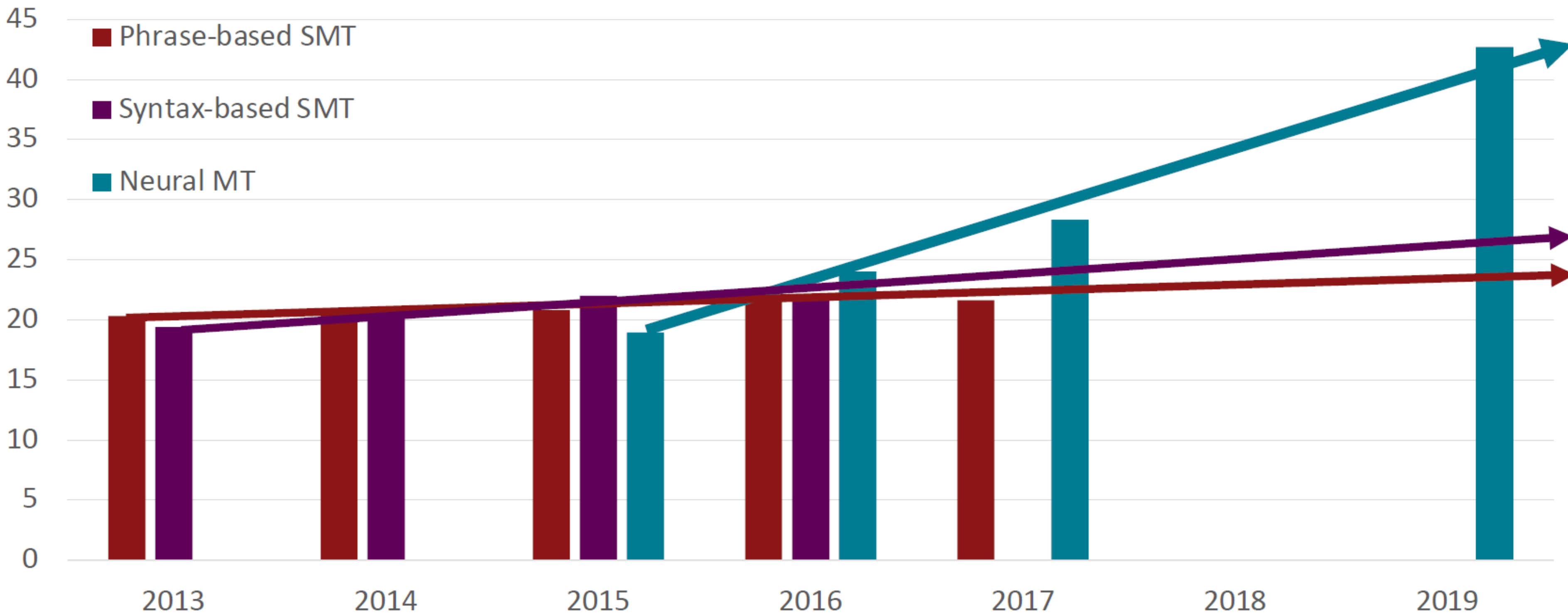
# Deep Sequence to Sequence Model

- Stacked seq2seq model



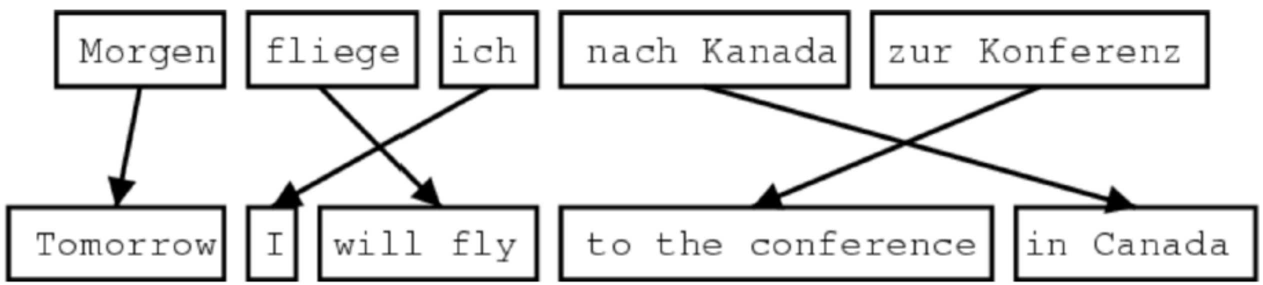
# Machine Translation

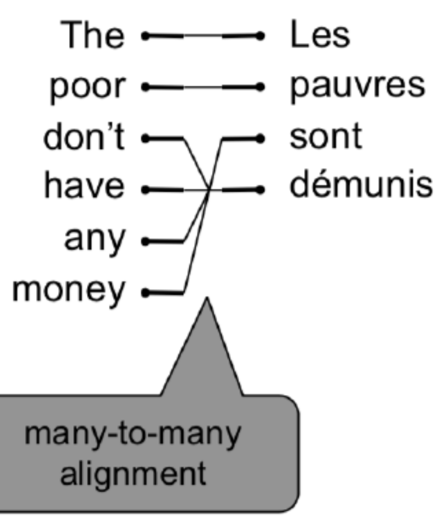
- 2016: Google switched Google Translate from SMT to NMT

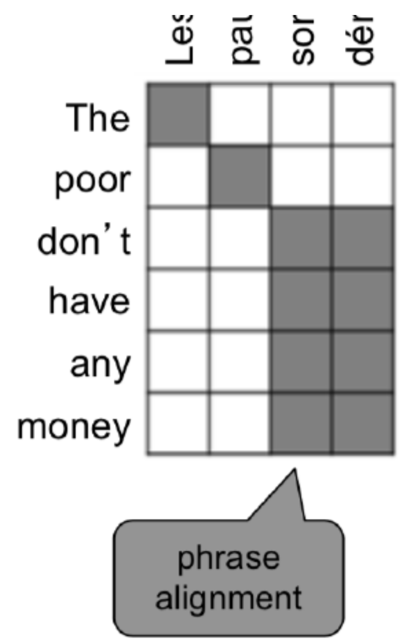


# Alignment

- Alignment: the word-level correspondence between X and Y
- Can have complex long-term dependencies

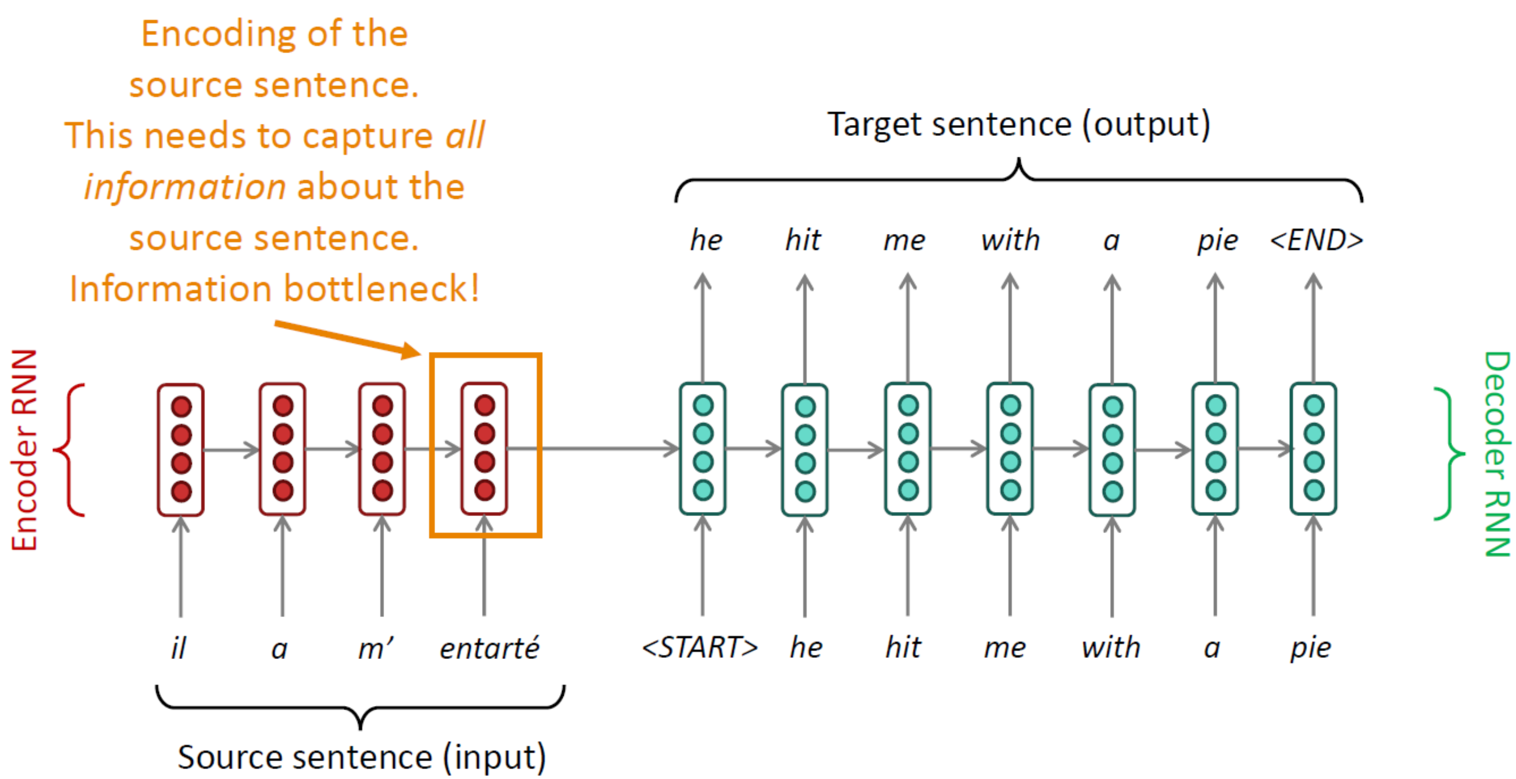


# Issue in Seq2Seq

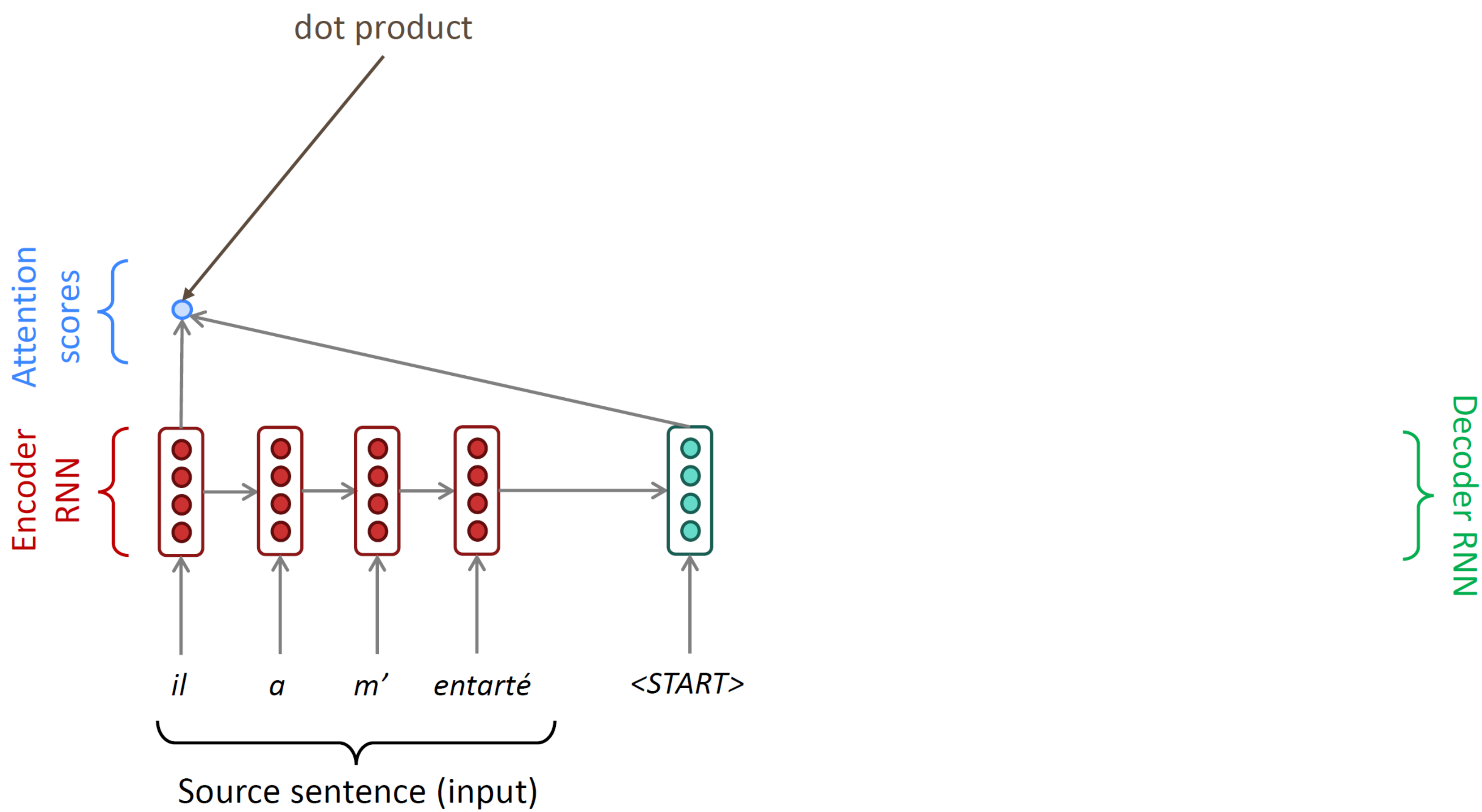
- Alignment: the word-level correspondence between X and Y
  - The information bottleneck due to the hidden state $h$
  - We want each $Y_t$ to also focus on some $X_i$ that it is aligned with



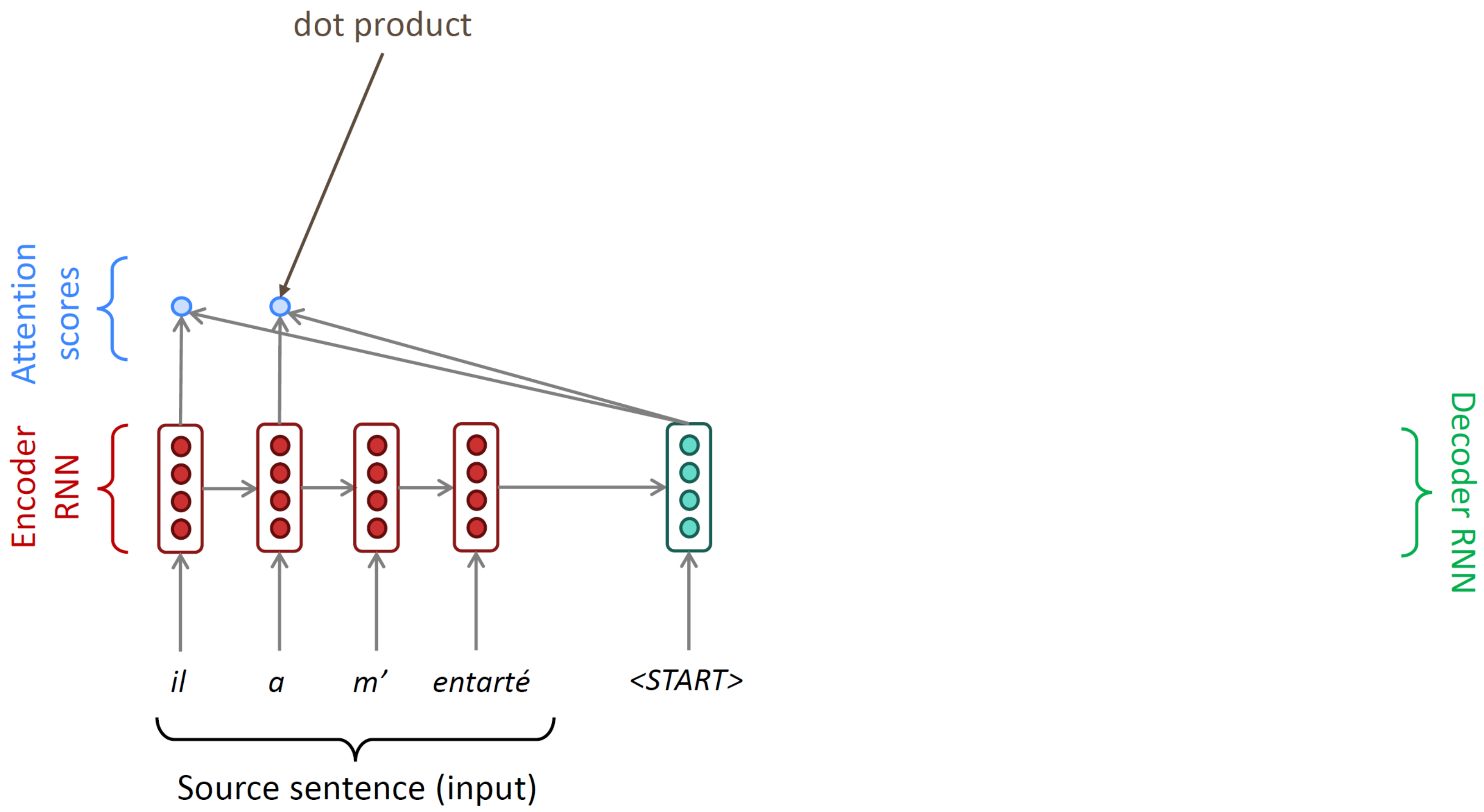
# Seq2Seq with Attention

- NMT by jointly learning to align and translate (Bahdanau, Cho, Bengio, '15)
- Core idea:
  - When decoding $Y_t$, consider both hidden states and alignment:
    - Hidden state: $h_t = f_{dec}(Y_{i<t})$
    - Alignment: connect to a portion of $X$
  - When portion of $X$ to focus on?
    - Learn a softmax weight over $X$: attention distribution $P_{att}$
    - $P_{att}(X_i \mid h_t)$: how much attention to put on word $X_i$
    - Attention output $h_{att} = \sum_i f_{enc}(X_i \mid X_{j<i}) \cdot P_{att}(X_i \mid h_{t-1})$
    - Use $h_{t-1}$ and $h_{att}$ to compute $Y_t$

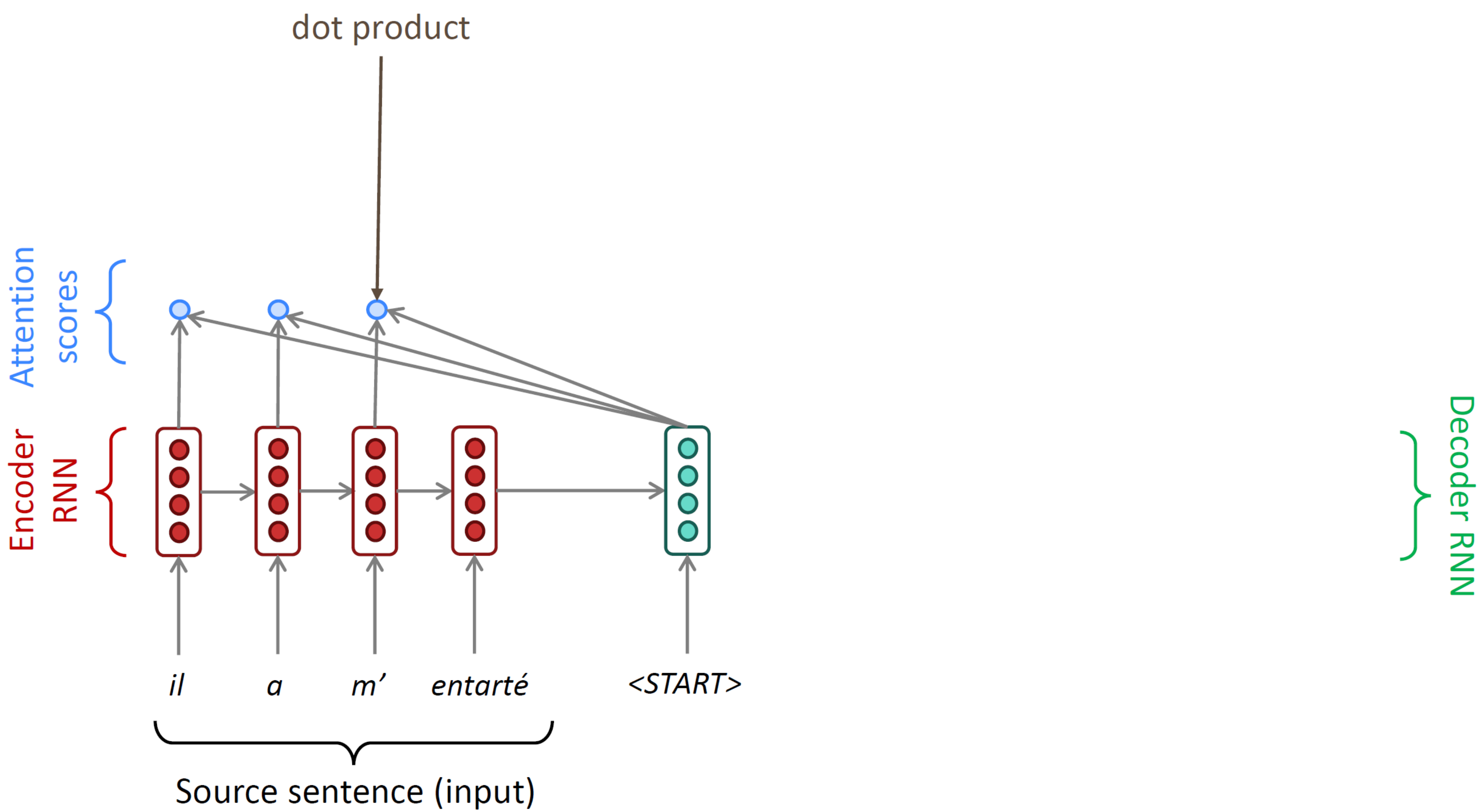
# Seq2Seq with Attention



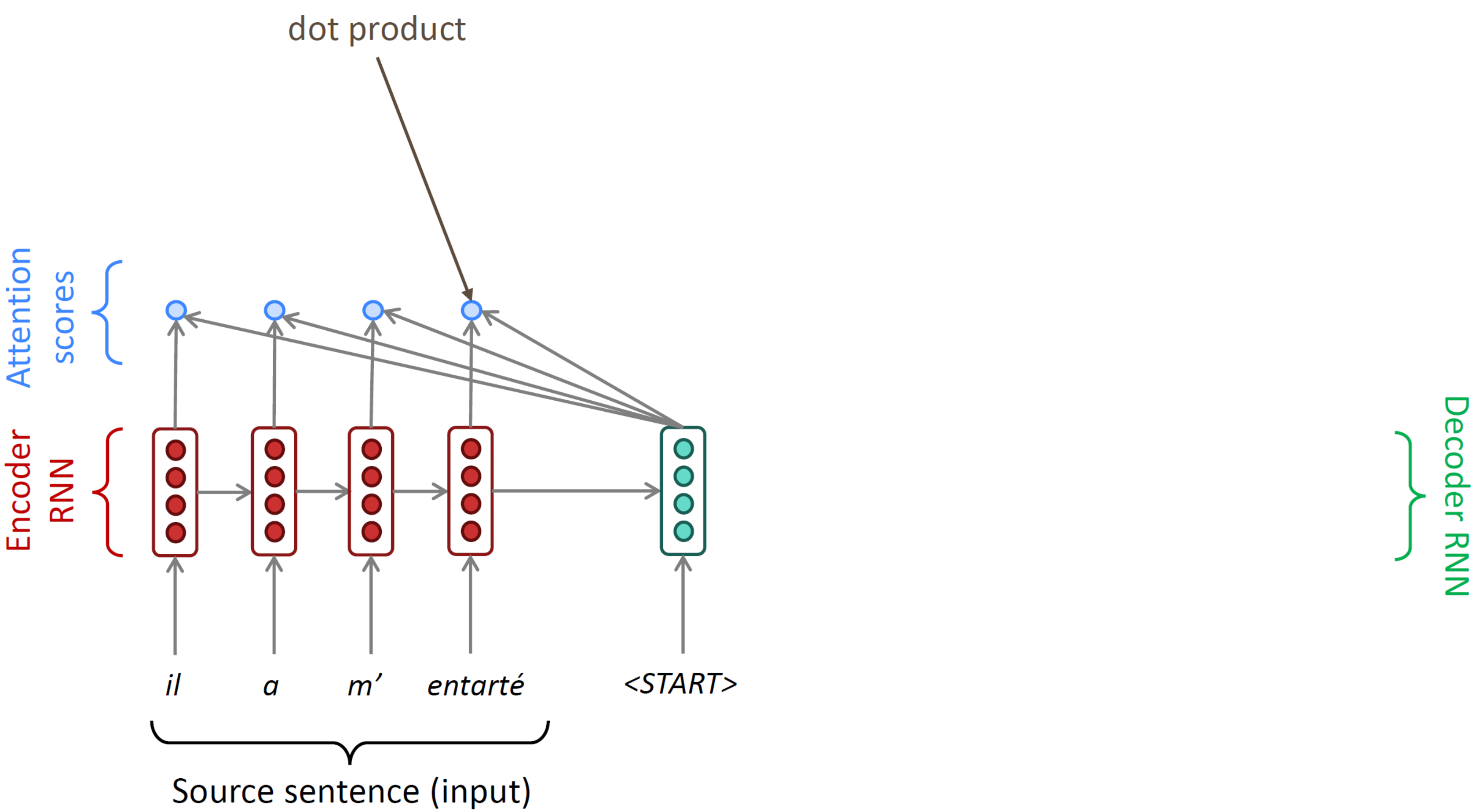
# Seq2Seq with Attention



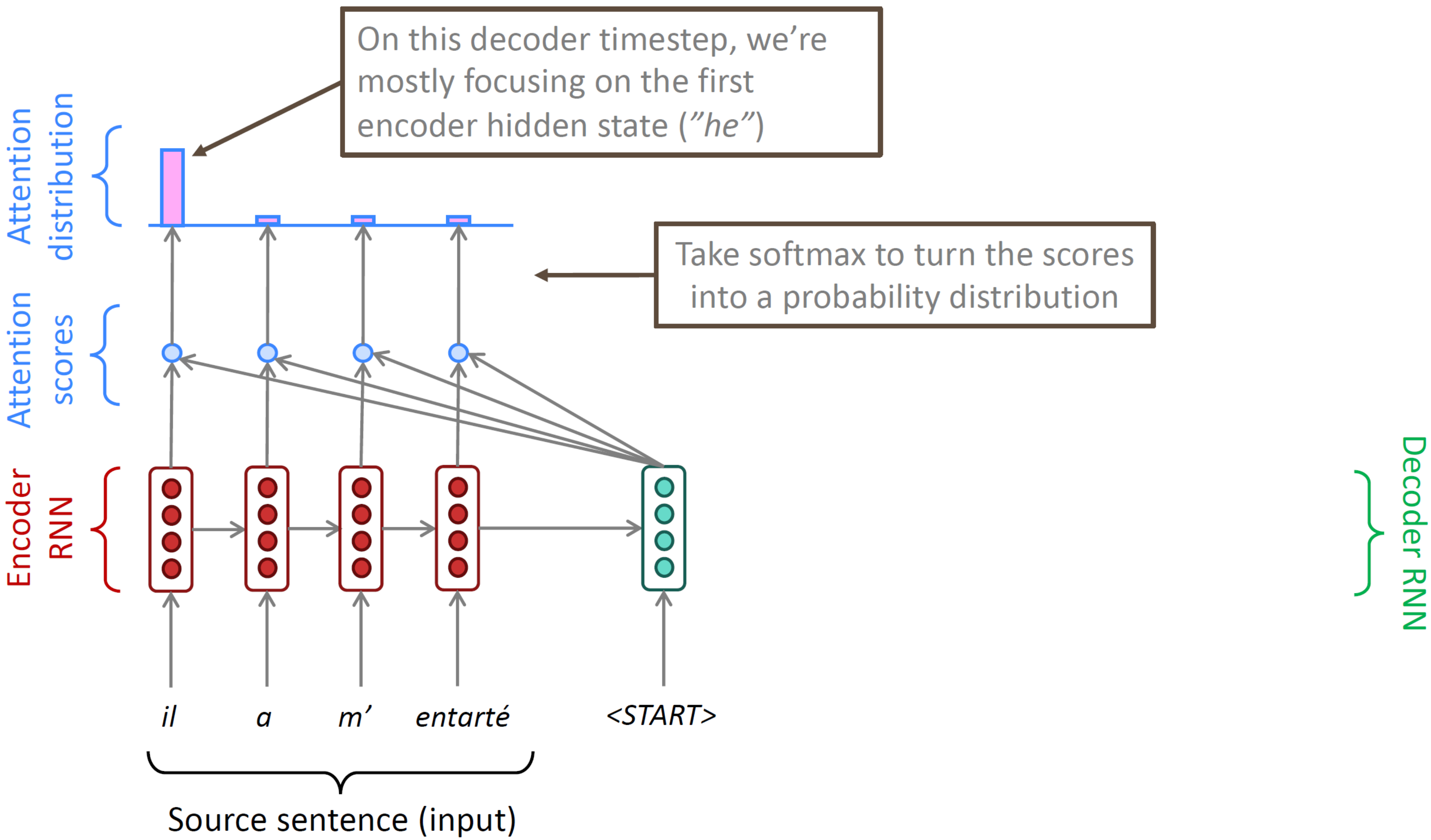
# Seq2Seq with Attention



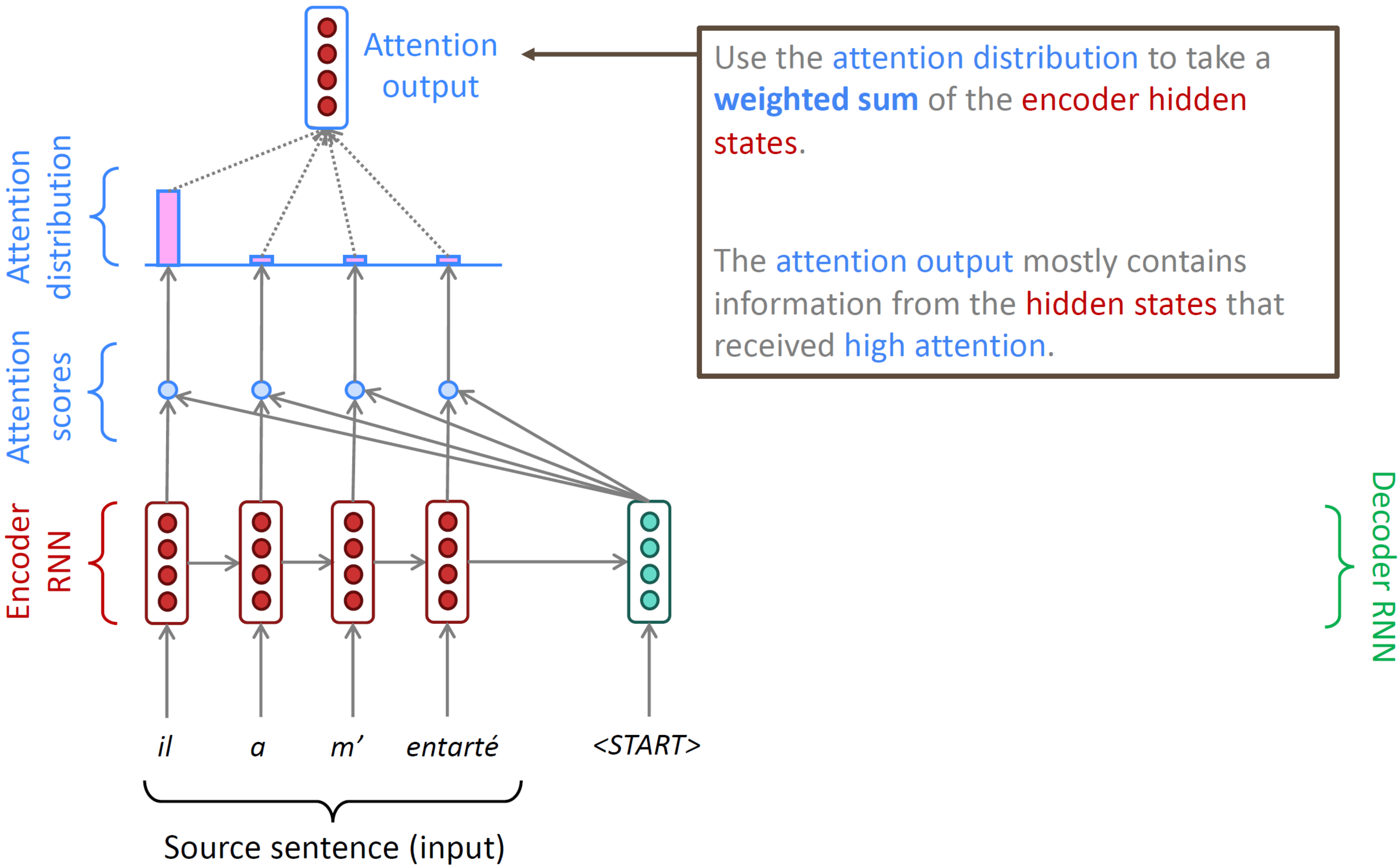
# Seq2Seq with Attention



# Seq2Seq with Attention



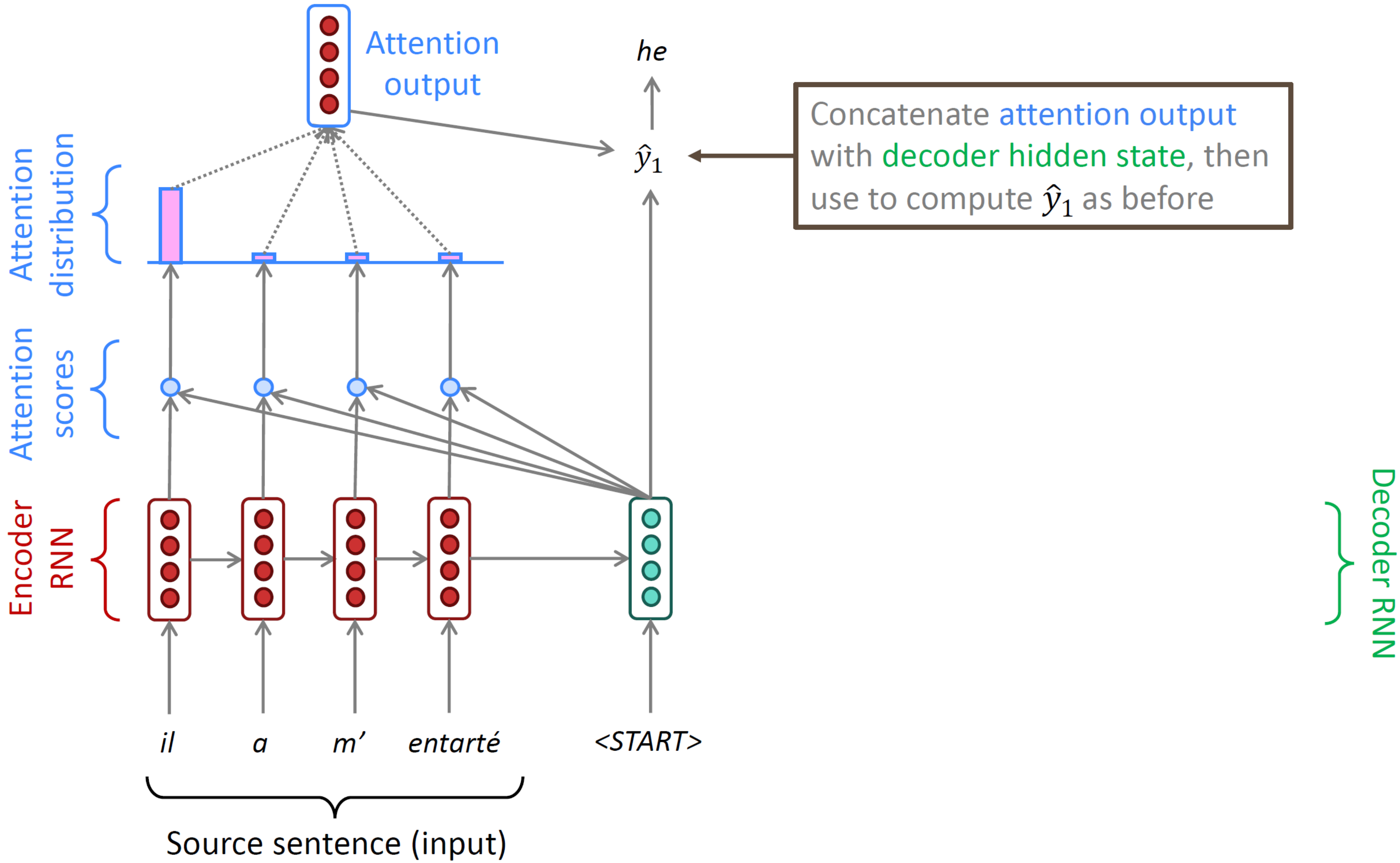
# Seq2Seq with Attention



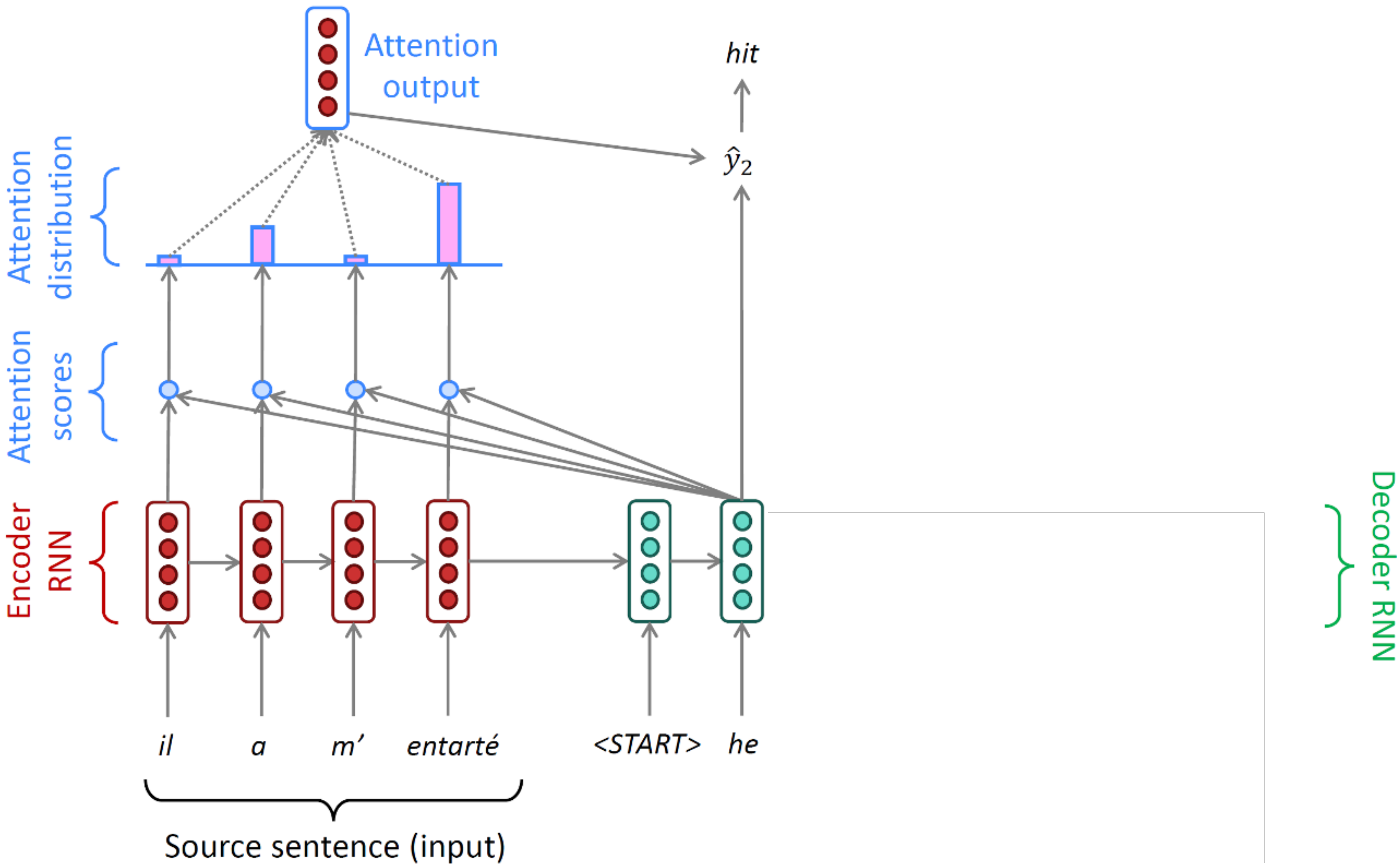

Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

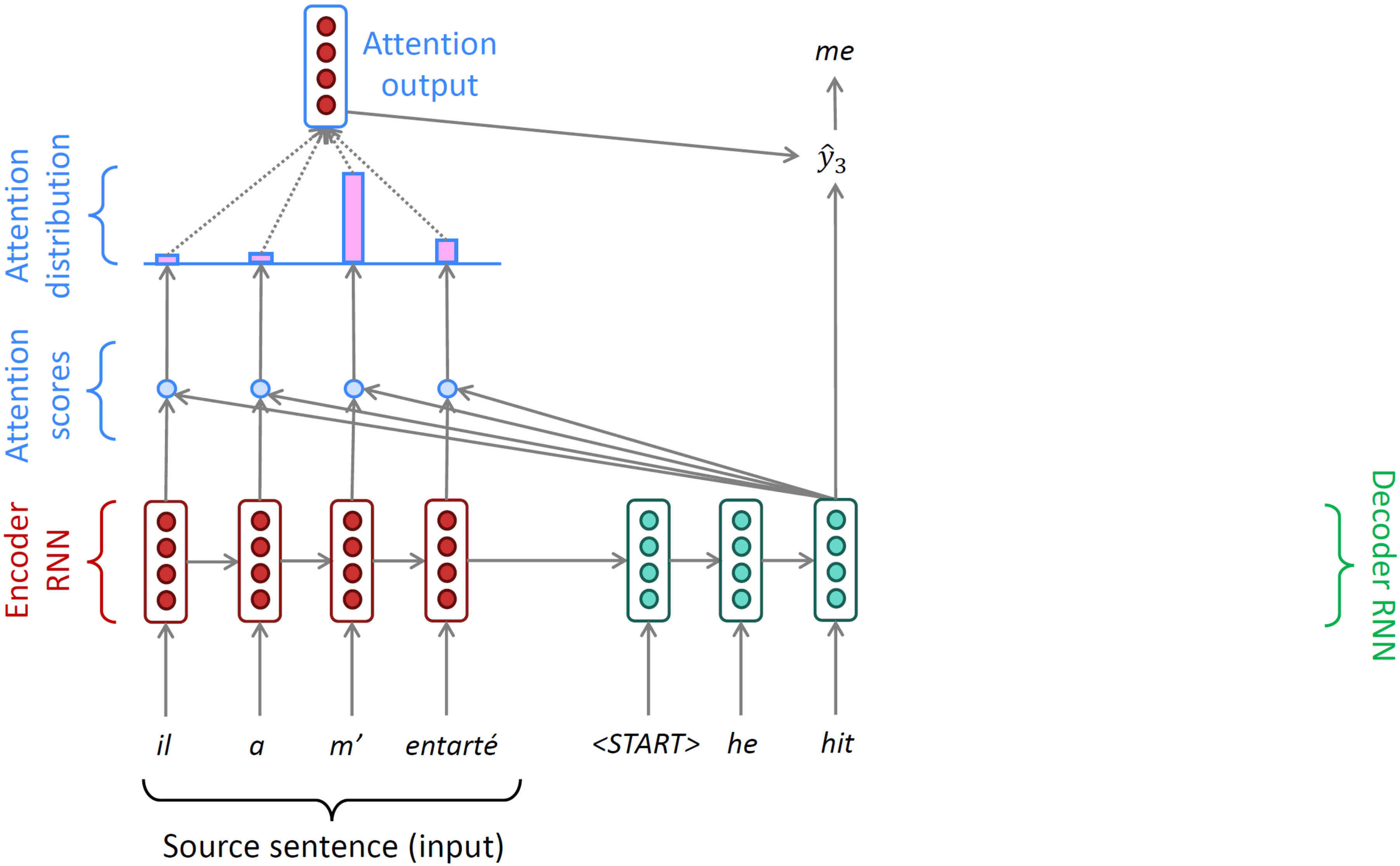
# Seq2Seq with Attention



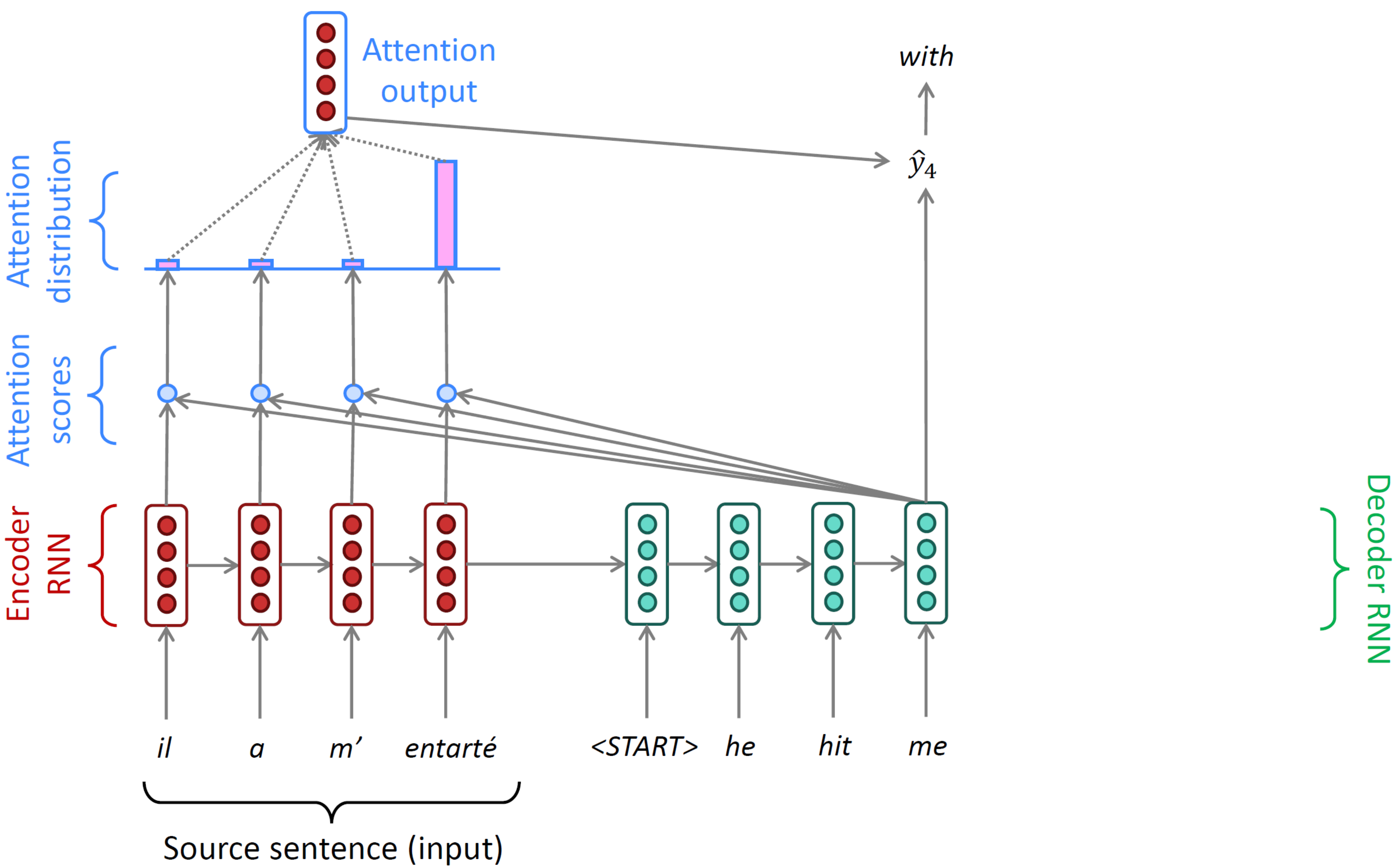
# Seq2Seq with Attention



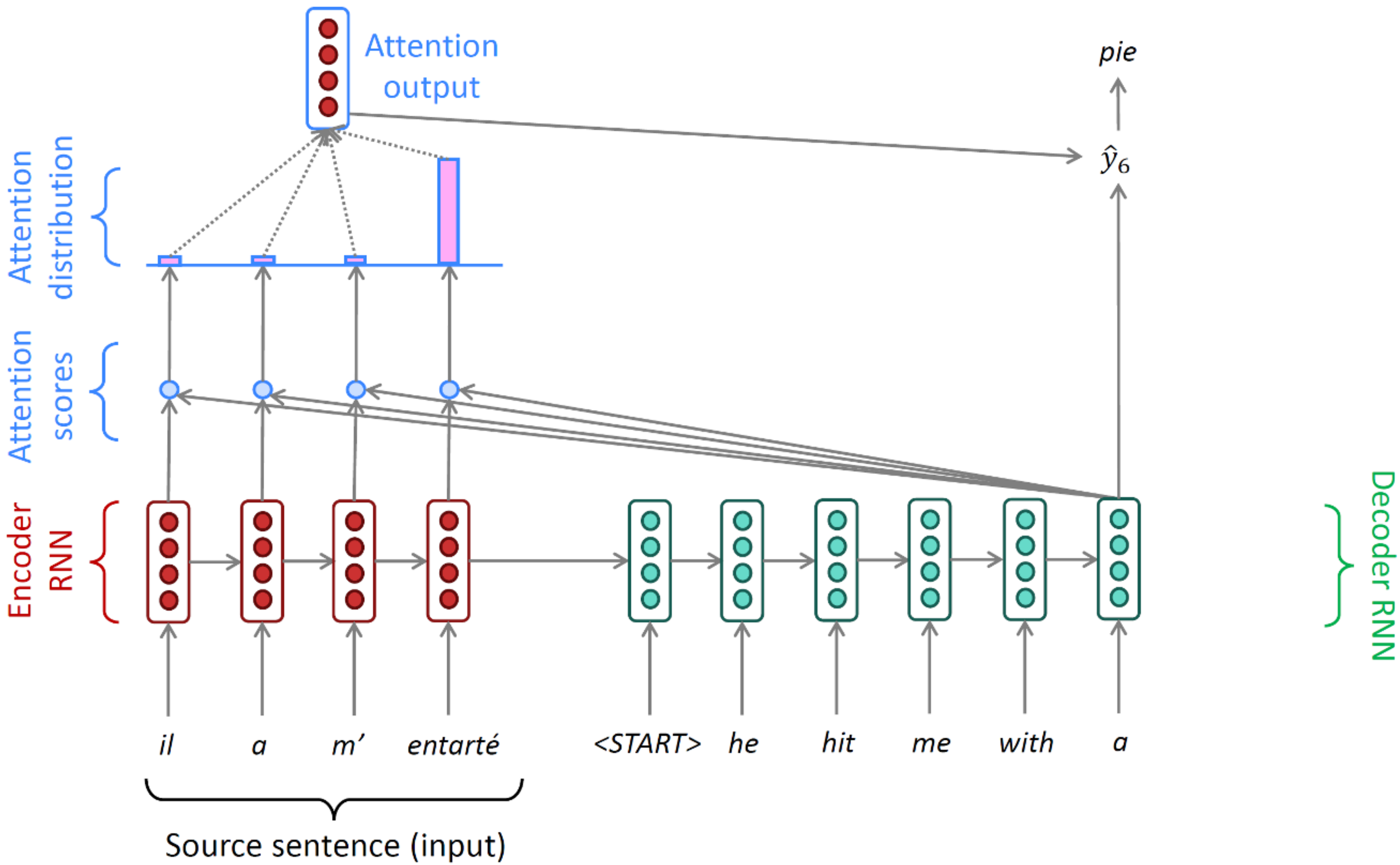
# Seq2Seq with Attention



# Seq2Seq with Attention



# Seq2Seq with Attention



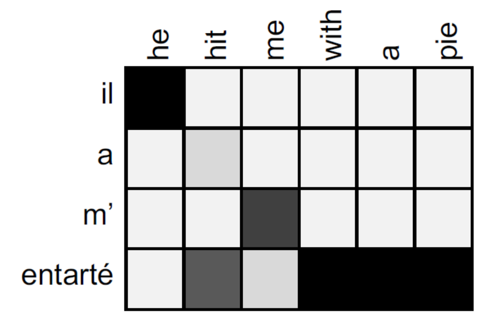
# Seq2Seq with Attention

Summary

- Input sequence $X$, encoder $f_{enc}$, and decoder $f_{dec}$
- $f_{enc}(X)$ produces hidden states $h_1^{enc}, h_2^{enc}, \ldots, h_N^{enc}$
- On time step $t$, we have decoder hidden state $h_t$
- Compute attention score $e_i = h_t^\top h_i^{enc}$
- Compute attention distribution $\alpha_i = P_{att}(X_i) = \text{softmax}(e_i)$
- Attention output: $h_{att}^{enc} = \sum_i \alpha_i h_i^{enc}$
- $Y_t \sim g(h_t, h_{att}^{enc}; \theta)$
    - Sample an output using both $h_t$ and $h_{att}^{enc}$

# Attention

- It significantly improves NMT.
- It solves the bottleneck problem and the long-term dependency issue.
- Also helps gradient vanishing problem.
- Provides some interpretability
  - Understanding which word the RNN encoder focuses on




- Attention is a general technique
  - Given a set of vector values $V_i$ and vector query $q$
  - Attention computes a weighted sum of values depending on $q$

Other use cases:

- Attention can be viewed as a module.
- In encoder and decoder (more on this later)
- A representation of a set of points
  - Pointer network (Vinyals, Forunato, Jaitly ’15)
  - Deep Sets (Zaheer et al., ’17)
- Convolutional neural networks
  - To include non-local information in CNN (Non-local network, ’18)

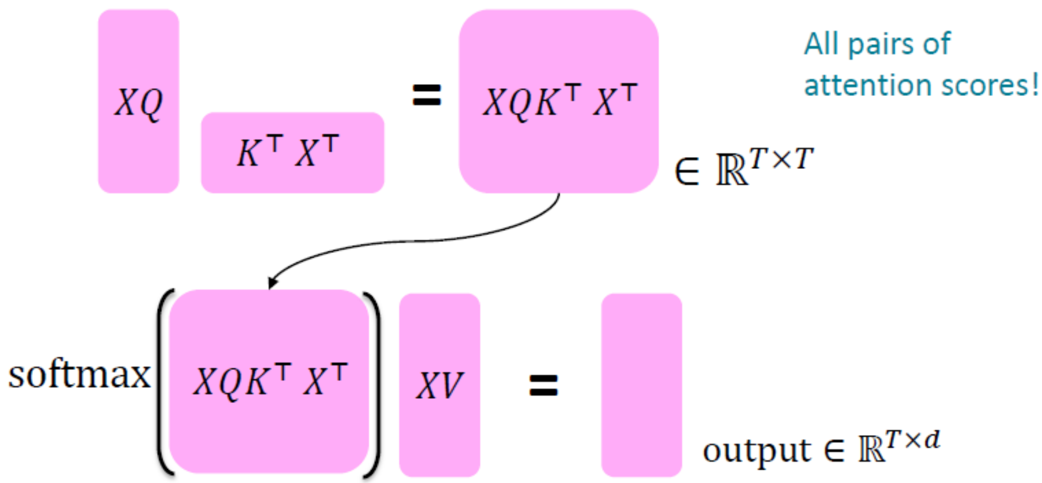
# Attention

- Representation learning:
  - A method to obtain a fixed representation corresponding to a query $q$ from an arbitrary set of representations $\{V_i\}$
  - Attention distribution: $\alpha_i = \text{softmax}(f(v_i, q))$
  - Attention output: $v_{att} = \sum_i \alpha_i v_i$

- Attent variant: $f(v_i, q)$
  - Multiplicative attention: $f(v_i, q) = q^\top W h_i$, $W$ is a weight matrix
  - Additive attention: $f(v_i, q) = u^\top \tanh(W_1 v_i + W_2 q)$

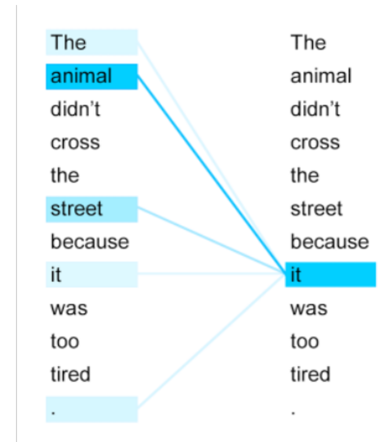
# Key-query-value attention

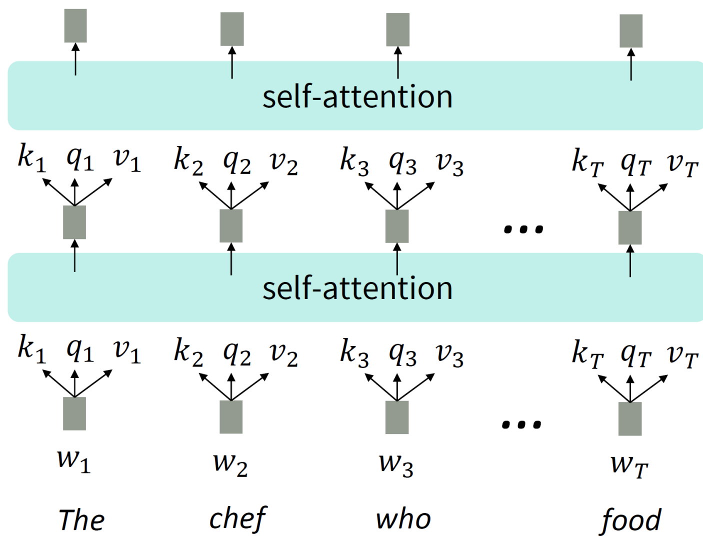
- Obtain $q_t, v_t, k_t$ from $X_t$
- $q_t = W^q X_t$; $v_t = W^v X_t$; $k_t = W^k X_t$ (position encoding omitted)
    - $W^q, W^v, W^k$ are learnable weight matrices
- $\alpha_{i,j} = \text{softmax}(q_i^\top k_j)$; $out_i = \sum_k \alpha_{i,j} v_j$
- Intuition: key, query, and value can focus on different parts of input



# Attention is all you need (Vsawani ’17)

- A pure attention-based architecture for sequence modeling
  - No RNN at all!
- Basic component: self-attention, $Y = f_{SA}(X; \theta)$
  - $X_t$ uses attention on entire $X$ sequence
  - $Y_t$ computed from $X_t$ and the attention output
- Computing $Y_t$
  - 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 $\alpha_{t,j} = \text{softmax}(q_t^\top k_j)$
    - Attention output $out_t = \sum_j \alpha_{t,j} v_j$
    - $Y_t = g_2(out_t; \theta)$



# Issues of Vanilla Self-Attention

- Attention is order-invariant

- Lack of non-linearities
  - All the weights are simple weighted average

- Capability of autoregressive modeling
  - In generation tasks, the model cannot "look at the future"
  - e.g. Text generation:
    - $Y_t$ can only depend on $X_{i<t}$
    - But vanilla self-attention requires the entire sequence

# Position Encoding

- Vanilla self-attention
  - $(k_t, v_t, q_t) = g_1(X_t; \theta)$
  - $\alpha_{t,j} = \text{softmax}(q_t^\top k_j)$
  - Attention output $out_t = \sum_j \alpha_{t,j} v_j$
- Idea: position encoding:
  - $p_i$: an embedding vector (feature) of position $i$
  - $(k_t, v_t, q_t) = g_1([X_t, p_t]; \theta)$

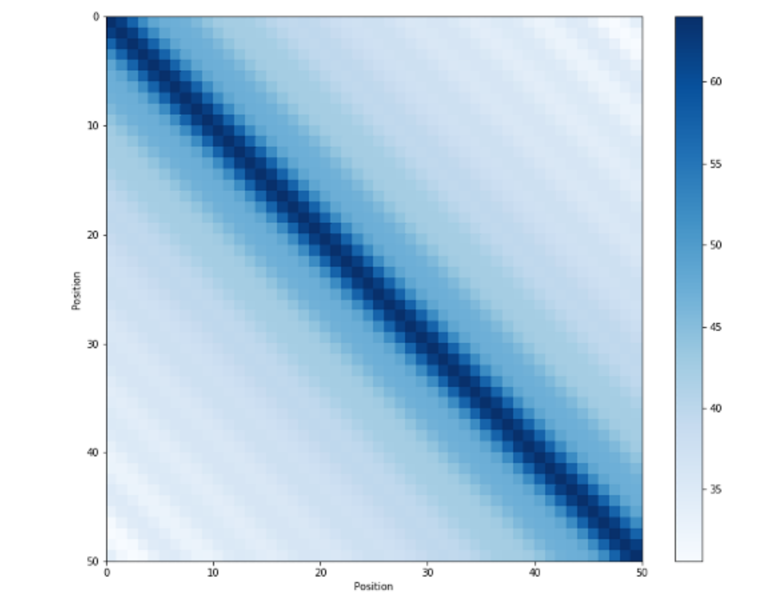
- In practice: Additive is sufficient: $k_t \leftarrow \tilde{k}_t + p_t, q_t \leftarrow \tilde{q}_t + p_t, v_t \leftarrow \tilde{v}_t + p_t;$
  $(\tilde{k}_t, \tilde{v}_t, \tilde{q}_t) = g_1(X_t; \theta)$

- $p_t$ is only included in the first layer

# Position Encoding

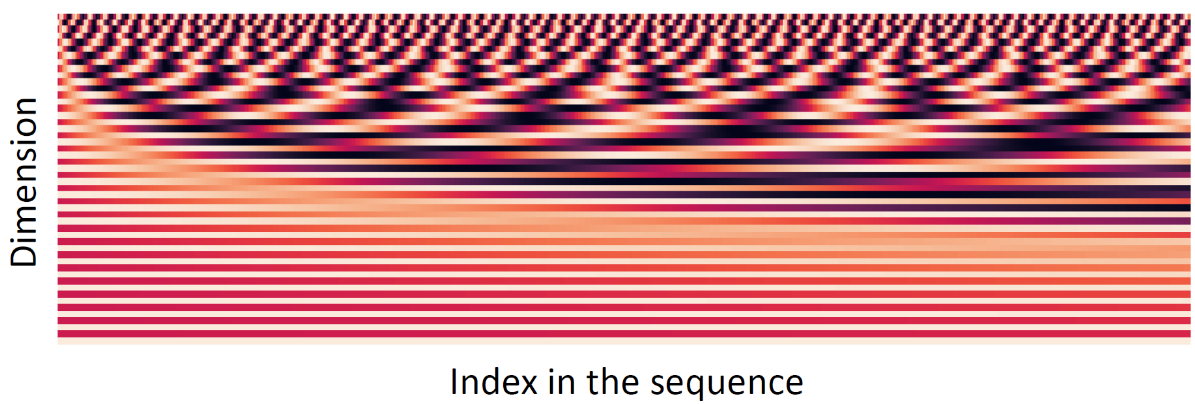
$p_t$ design 1: **Sinusoidal position representation**

- Pros:
  - simple
  - naturally models “relative position”
  - Easily applied to long sequences
- Cons:
  - Not learnable
  - Generalization poorly to sequences longer than training data




Heatmap of $p_i^T p_j$

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



# Position Encoding

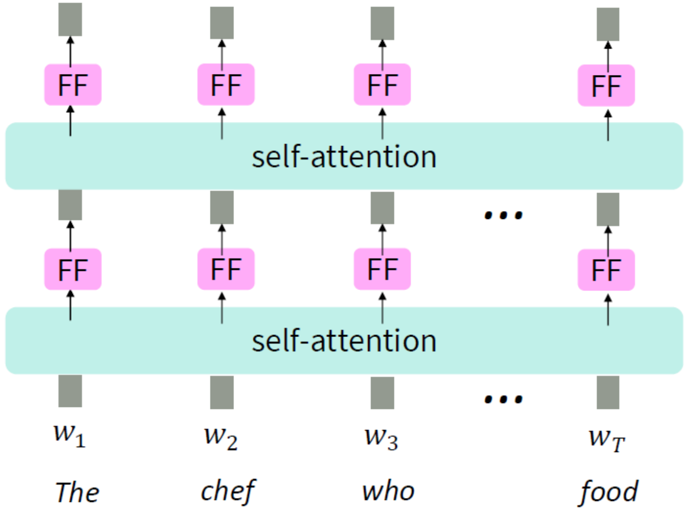
$p_t$ design 2: **Learned representation**

- Assume maximum length $L$, learn a matrix $p \in \mathbb{R}^{d \times T}$, $p_t$ is a column of $p$
- Pros:
  - Flexible
  - Learnable and more powerful
- Cons:
  - Need to assume a fixed maximum length $L$
  - Does not work at all for length above $L$

# Combine Self-Attention with Nonlinearity

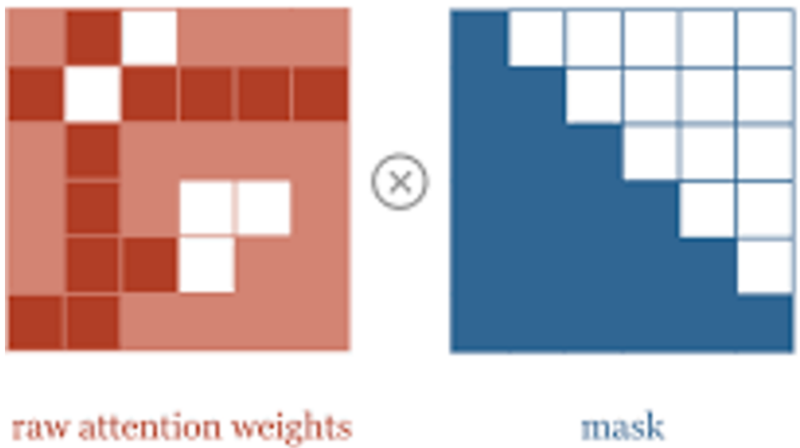
- Vanilla self-attention
  - No element-wise activation (e.g., ReLU, tanh)
  - Only weighted average and softmax operator

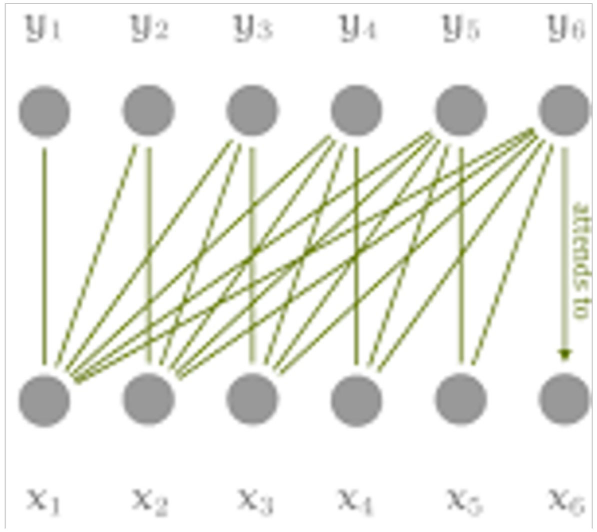
- Fix:
  - Add an MLP to process $out_i$
  - $m_i = MLP(out_i) = W_2\text{ReLU}(W_1 out_i + b_1) + b_2$
  - Usually do not put activation layer before softmaax



# Masked Attention

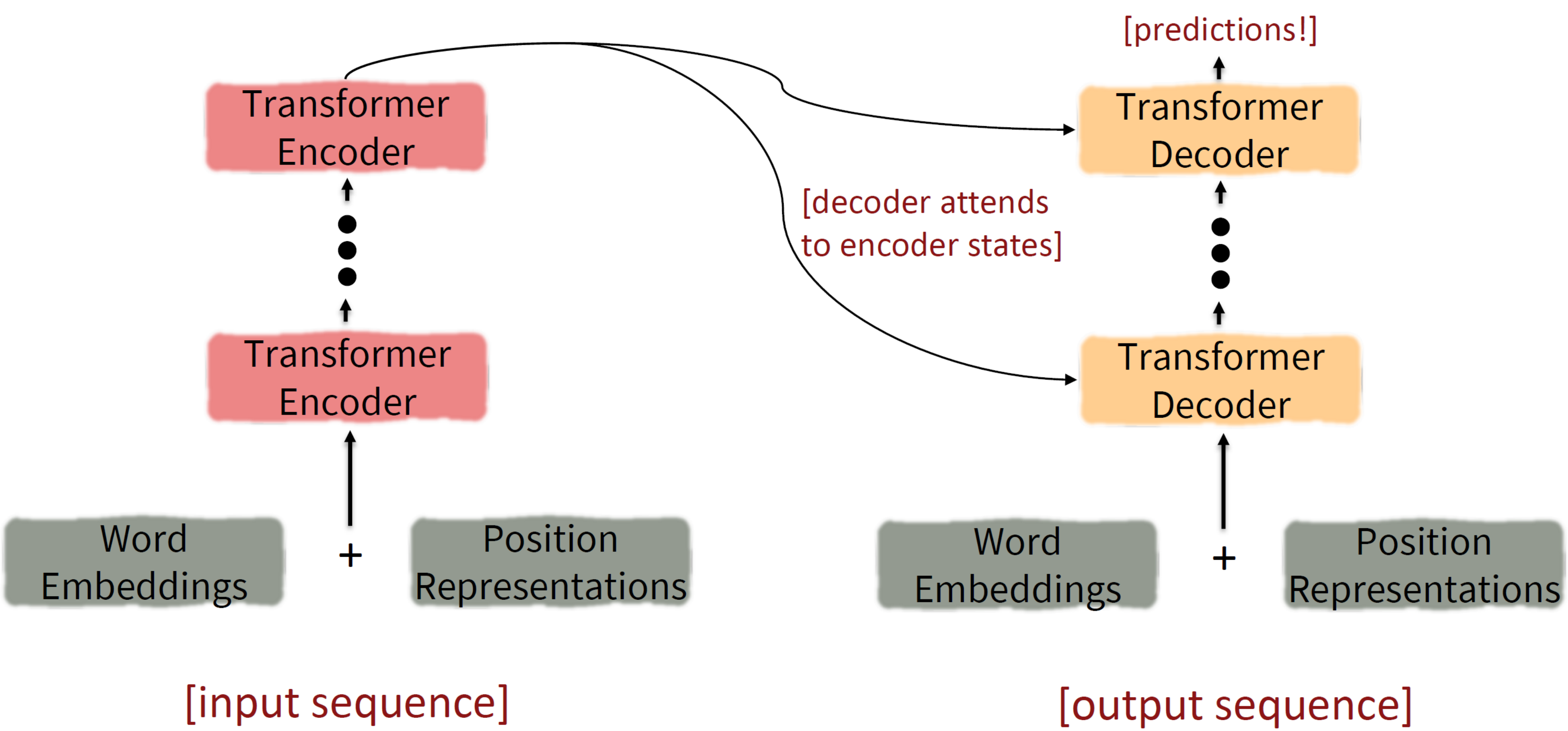
- In language model decoder: $P(Y_t \mid X_{i<t})$
  - $out_t$ cannot look at future $X_{i>t}$
- Masked attention
  - Compute $e_{i,j} = q_i^\top k_j$ as usuall
  - Mask out $e_{i>j}$ by setting $e_{i>j} = -\infty$
    - $e \odot (1 - M) \leftarrow -\infty$
    - $M$ is a fixed 0/1 mask matrix
  - Then compute $\alpha_i = \text{softmax}(e_i)$
  - Remarks:
    - $M = 1$ for full self-attention
    - Set $M$ for arbitrary dependency ordering



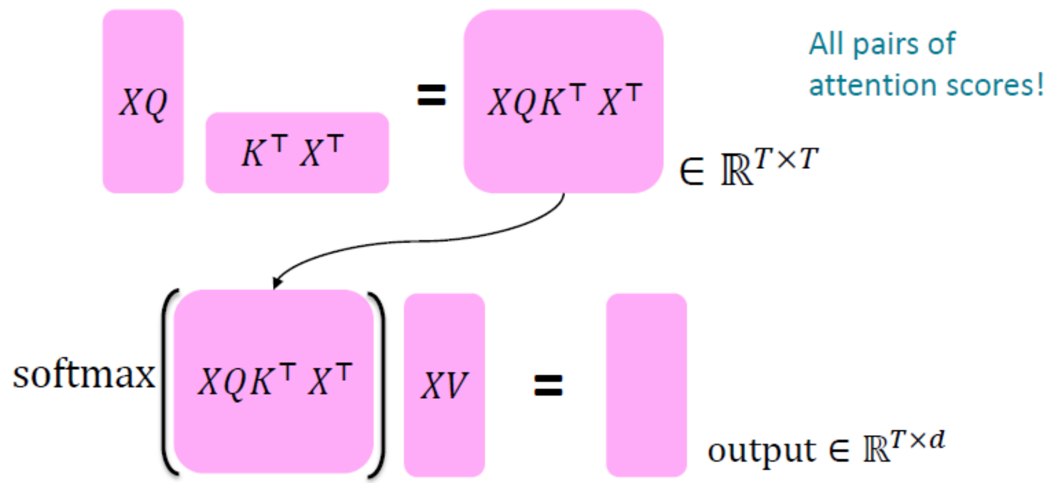
# Transformer

Transformer-based sequence-to-sequence modeling



# Key-query-value attention

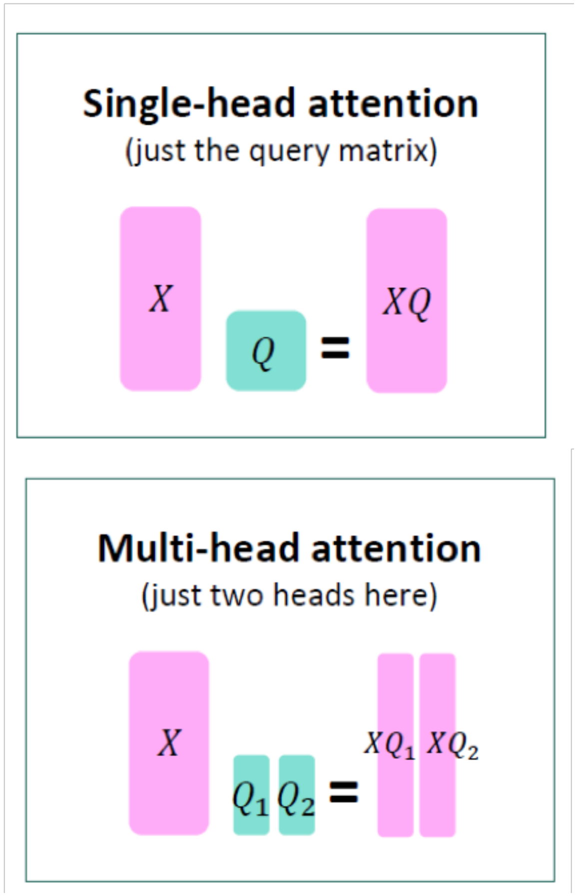
- Obtain $q_t, v_t, k_t$ from $X_t$
- $q_t = W^q X_t$; $v_t = W^v X_t$; $k_t = W^k X_t$ (position encoding omitted)
  - $W^q, W^v, W^k$ are learnable weight matrices
- $\alpha_{i,j} = \text{softmax}(q_i^\top k_j)$; $out_i = \sum_k \alpha_{i,j} v_j$
- Intuition: key, query, and value can focus on different parts of input



# Multi-headed attention

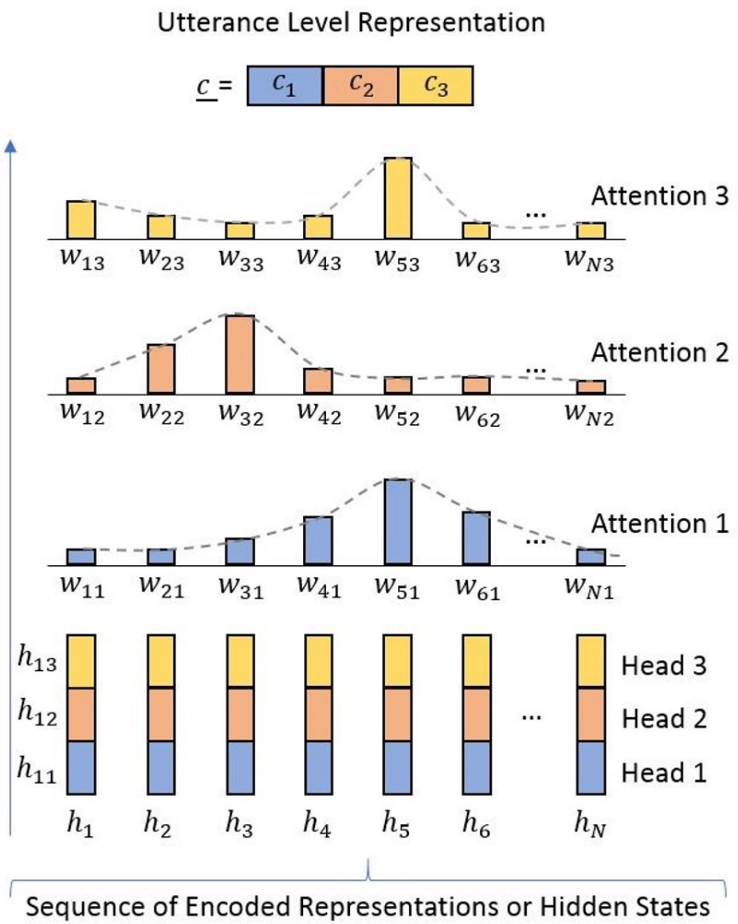
- Standard attention: single-headed attention
  - $X_t \in \mathbb{R}^d, Q, K, V \in \mathbb{R}^{d \times d}$
  - We only look at a single position $j$ with high $\alpha_{i,j}$
  - What if we want to look at different $j$ for different reasons?
- Idea: define $h$ separate attention heads
  - $h$ different attention distributions, keys, values, and queries
  - $Q^\ell, K^\ell, V^\ell \in \mathbb{R}^{d \times \frac{d}{h}}$ for $1 \le \ell \le h$
  - $\alpha_{i,j}^\ell = \text{softmax}((q_i^\ell)^\top k_j^\ell); out_i^\ell = \sum_j \alpha_{i,j}^\ell v_j^\ell$

**#Params Unchanged!**



# Multi-headed attention

- Standard attention: single-headed attention
  - $X_t \in \mathbb{R}^d, Q, K, V \in \mathbb{R}^{d \times d}$
  - 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
  - $Q^\ell, K^\ell, V^\ell \in \mathbb{R}^{d \times \frac{d}{h}}$ for $1 \leq \ell \leq h$
  - $\alpha_{i,j}^\ell = \text{softmax}((q_i^\ell)^\top k_j^\ell); out_i^\ell = \sum_j \alpha_{i,j}^\ell v_j^\ell$

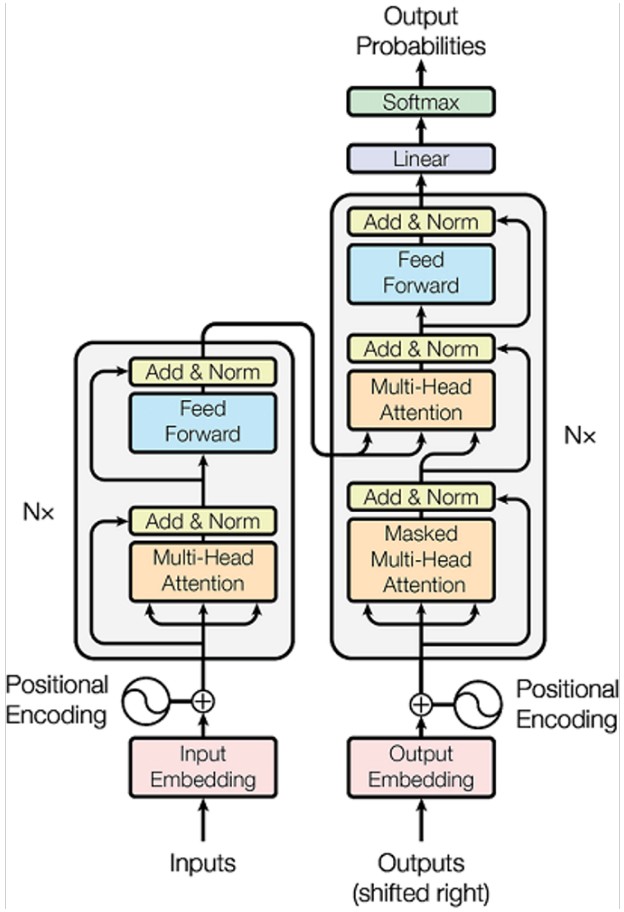


# Transformer

Transformer-based sequence-to-sequence model

- Basic building blocks: self-attention
  - Position encoding
  - Post-processing MLP
  - Attention mask

- Enhancements:
  - Key-query-value attention
  - Multi-headed attention
  - Architecture modifications:
    - Residual connection
    - Layer normalization



# Transformer

Machine translation with transformer

| 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** | $2.3 \cdot 10^{19}$ | |

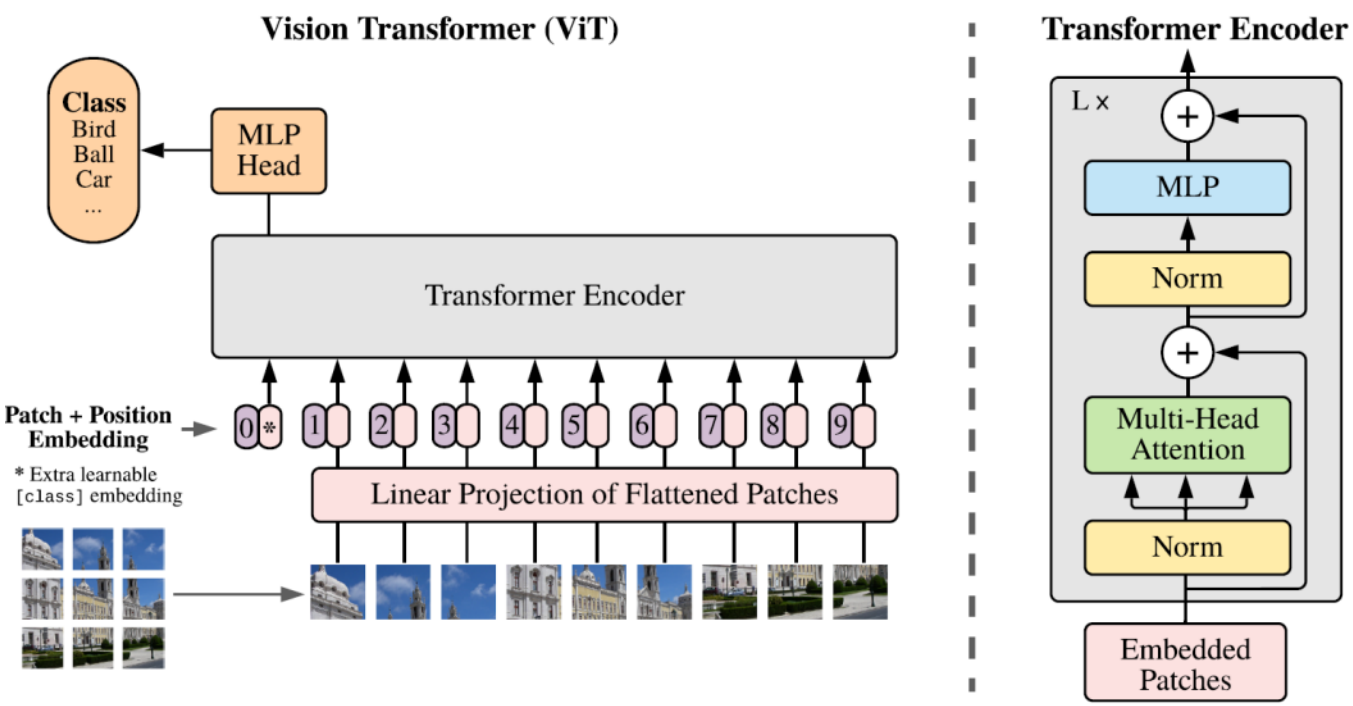
# Transformer

- Limitations of transformer: Quadratic computation cost
  - Linear for RNNs
  - Large cost for large sequence length, e.g., $L > 10^4$

- Follow-ups:
  - Large-scale training: transformer-XL; XL-net (‘20)
  - Projection tricks to $O(L)$: Linformer ('20)
  - Math tricks to $O(L)$: Performer (‘20)
  - Sparse interactions: Big Bird (‘20)
  - Deeper transformers: DeepNet (’22)

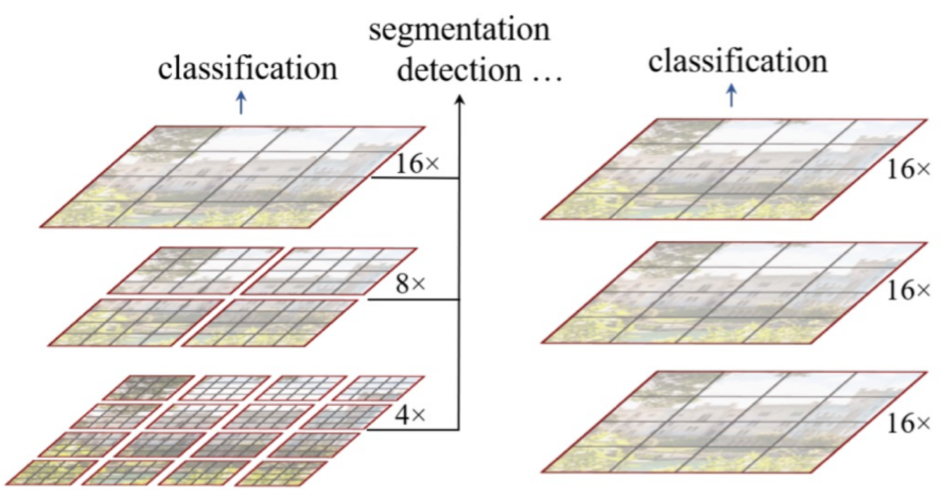
# Transformer for Images

- Vision Transformer ('21)
  - Decompose an image to 16x16 patches and then apply transformer encoder



# Transformer for Images

- Swin Transformer ('21)
  - Build hierachical feature maps at different resolution
    - Self-attention only within each block
    - Shifted block partitions to encode information between blocks




(a) Swin Transformer (ours)

(b) ViT

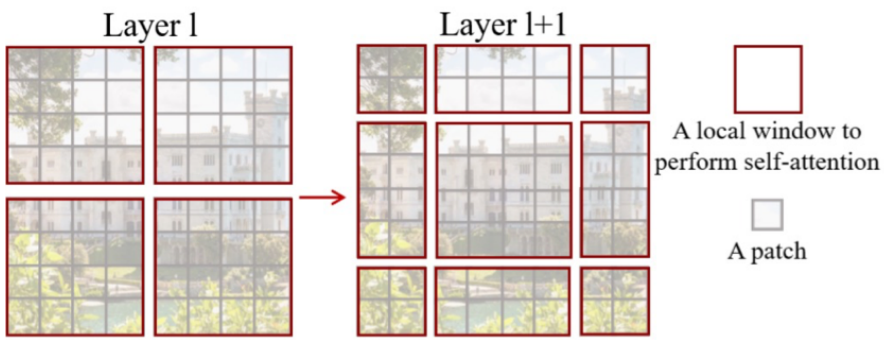
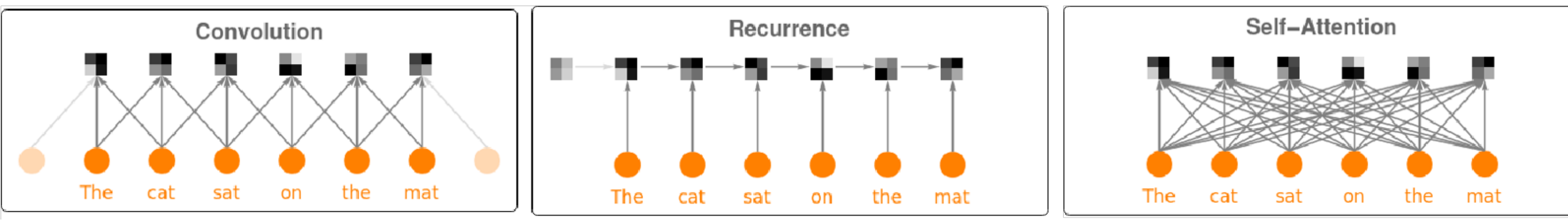



Figure 2. An illustration of the *shifted window* approach for com-

# CNN vs. RNN vs. Attention



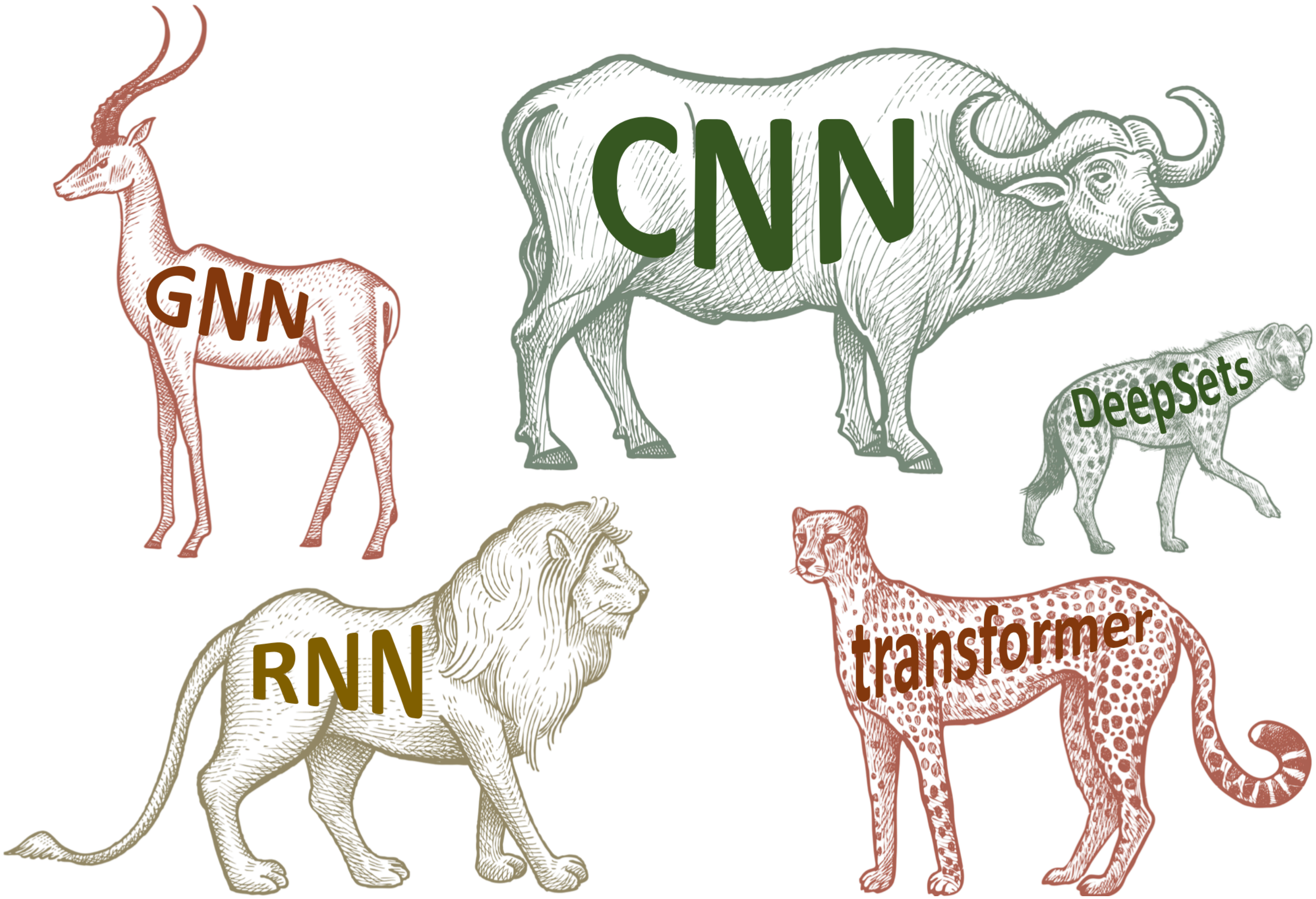
# Summary

- Language model & sequence to sequence model:
  - Fundamental ideas and methods for sequence modeling

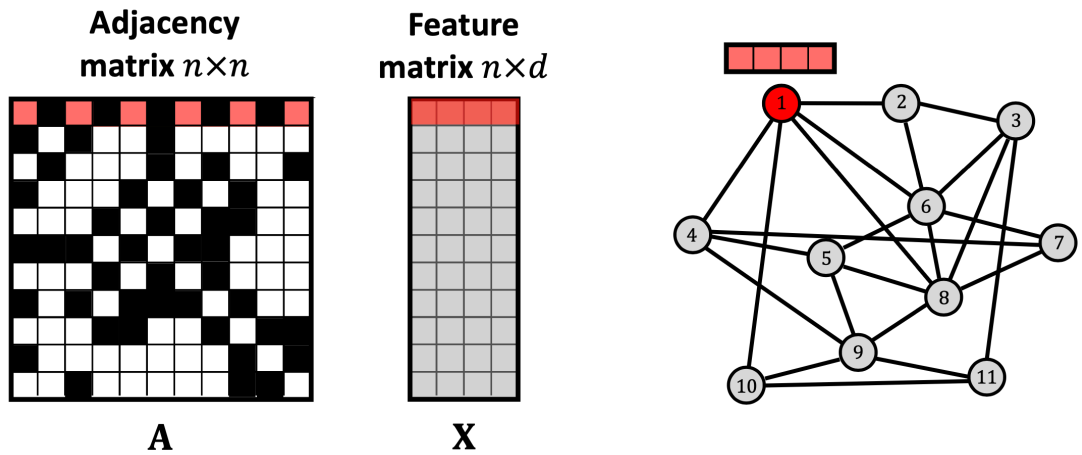
- Attention mechanism
  - So far the most successful idea for sequence data in deep learning
  - A scale/order-invariant representation
  - Transformer: a fully attention-based architecture for sequence data
  - Transformer + Pretraining: the core idea in today's NLP tasks

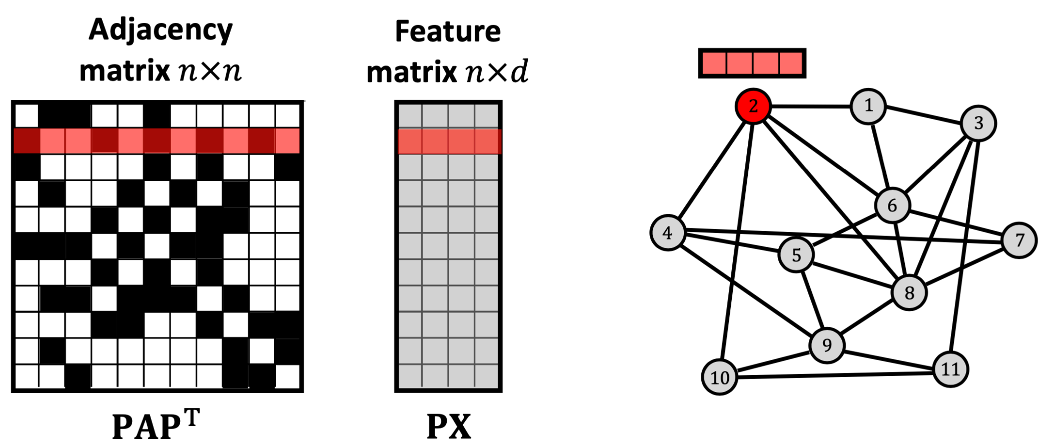
- LSTM is still useful in lightweight scenarios

# Other architectures



# Graph Neural Networks



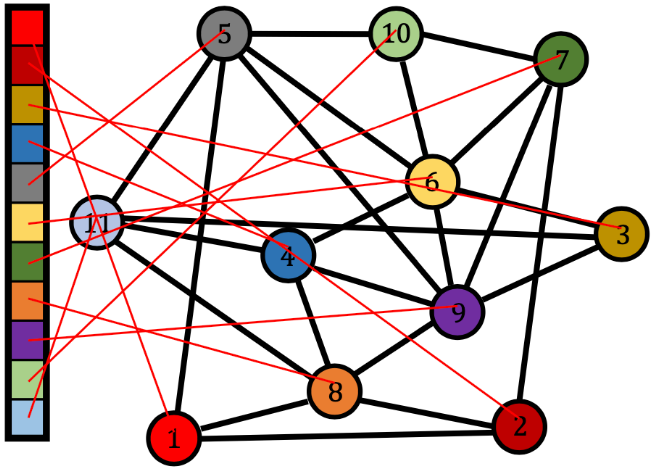




arbitrary ordering of nodes

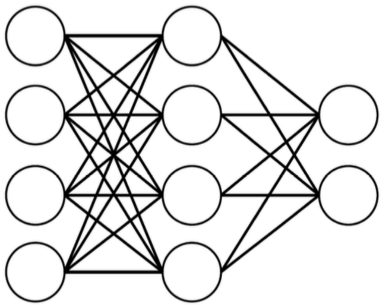
# Graph Neural Networks

permutation-equivariant

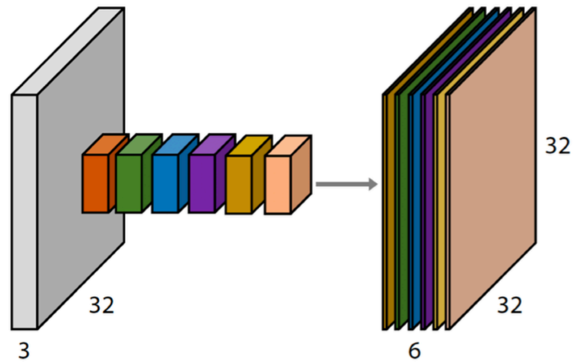
$$\mathbf{F}\left(\mathbf{PX},\mathbf{PAP}^{\top}\right)=\mathbf{PF}(\mathbf{X},\mathbf{A})$$



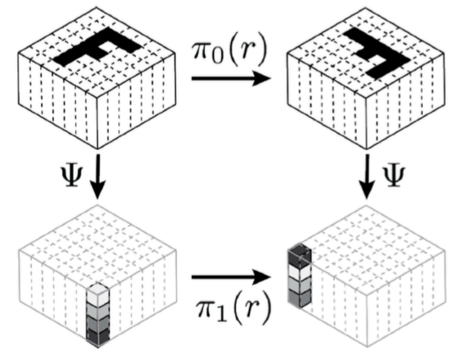
# Geometric Deep Learning



**Perceptrons**
Function regularity

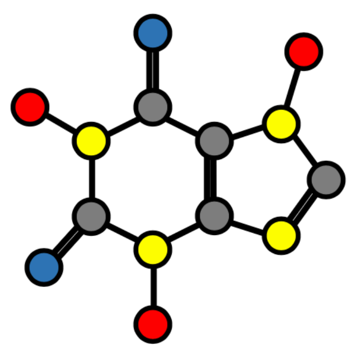



**CNNs**
Translation




**Group-CNNs**
Translation+Rotation

**DeepSets / Transformers**
Permutation



**GNNs**
Permutation

**Intrinsic CNNs**
Local frame choice