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)

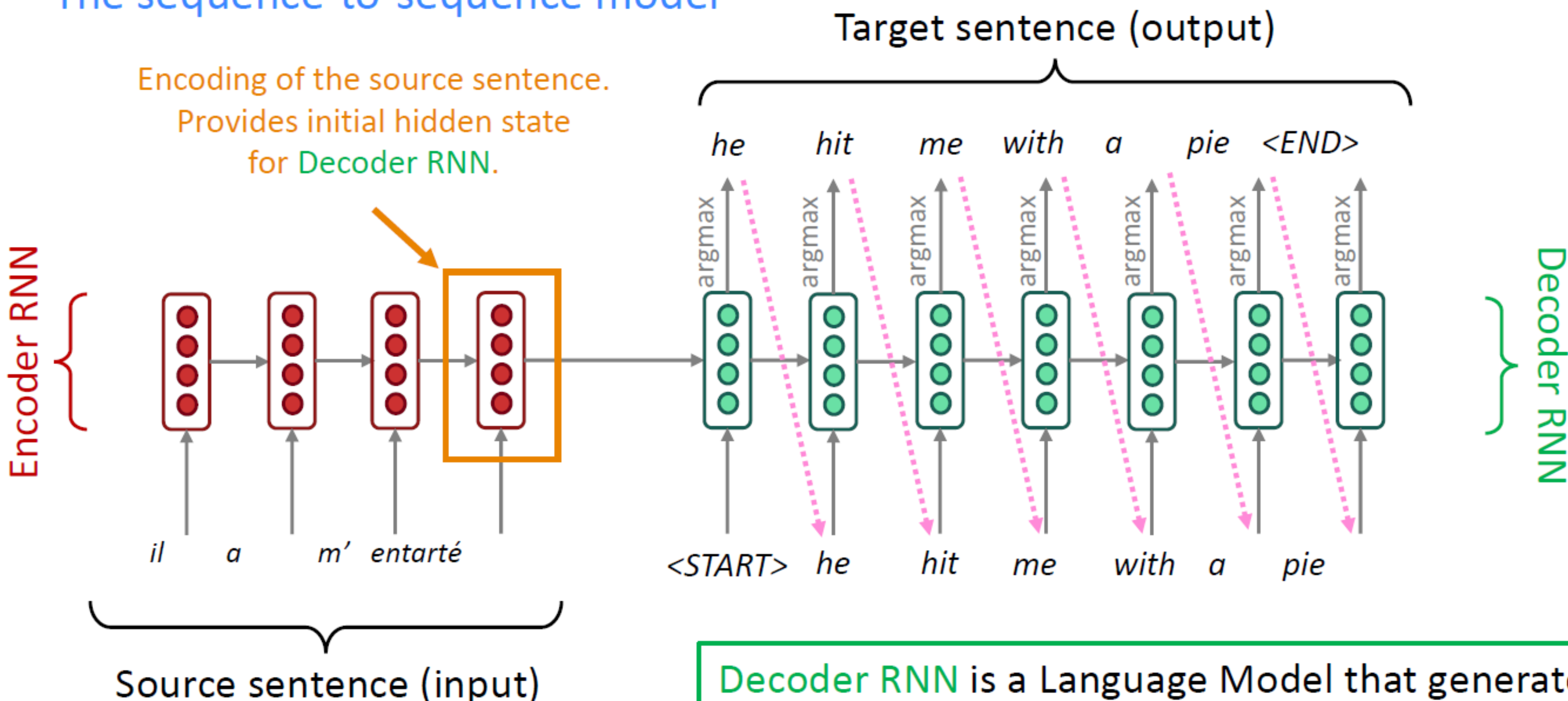
Sequence to Sequence Model

- 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

The sequence-to-sequence model



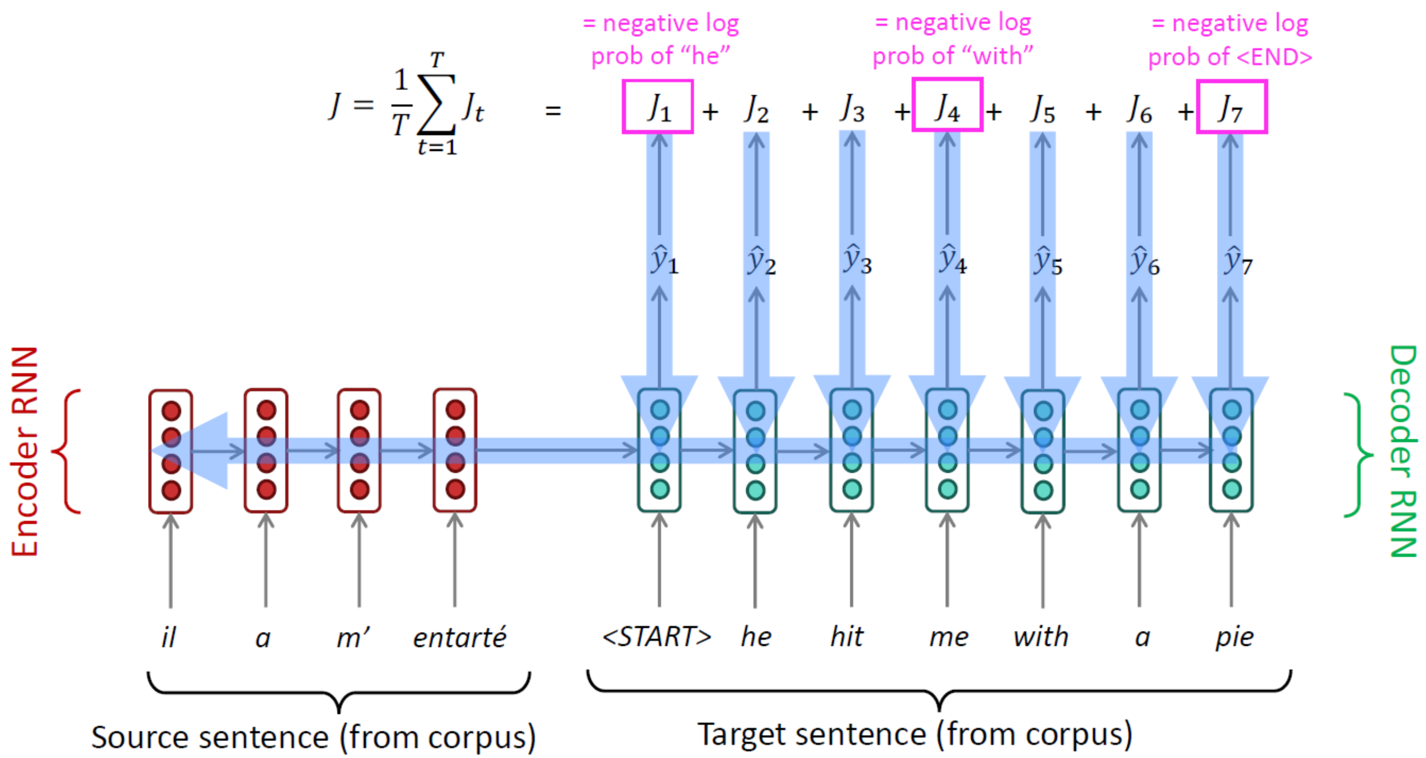
Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

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

Training Sequence to Sequence Model

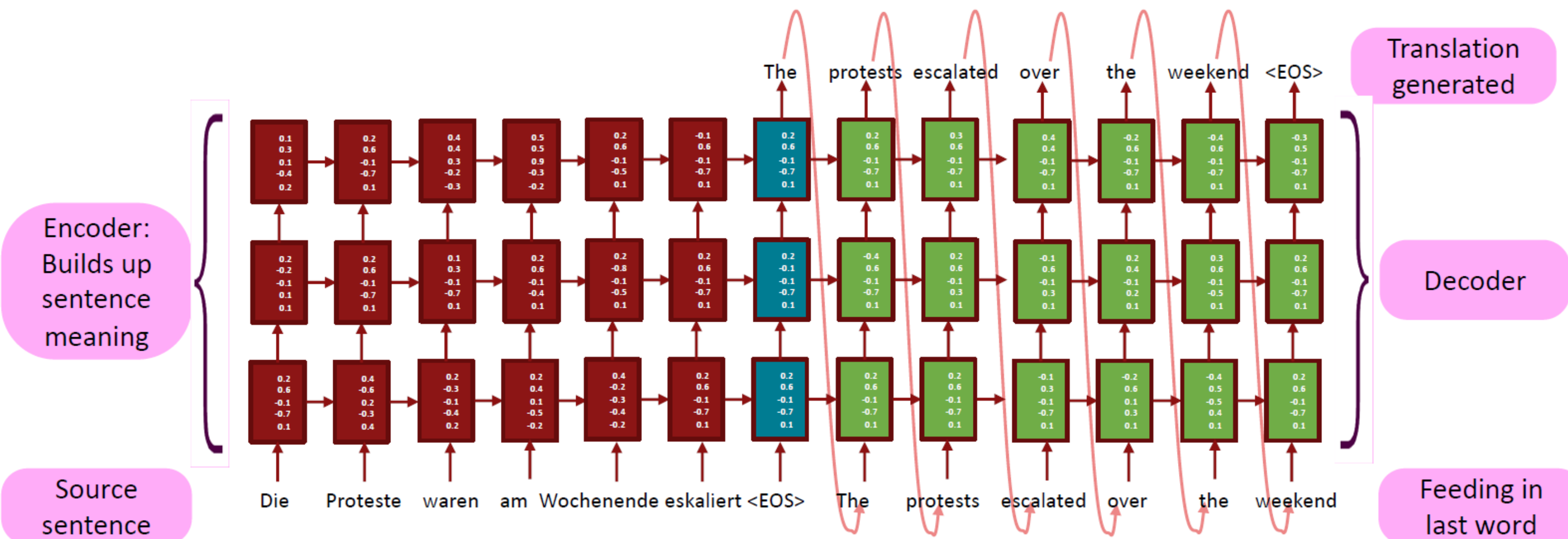
- Collect a huge paired dataset and train it end-to-end via BPTT
- Loss induced by MLE $P(Y|X) = P(Y|f_{enc}(X))$



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

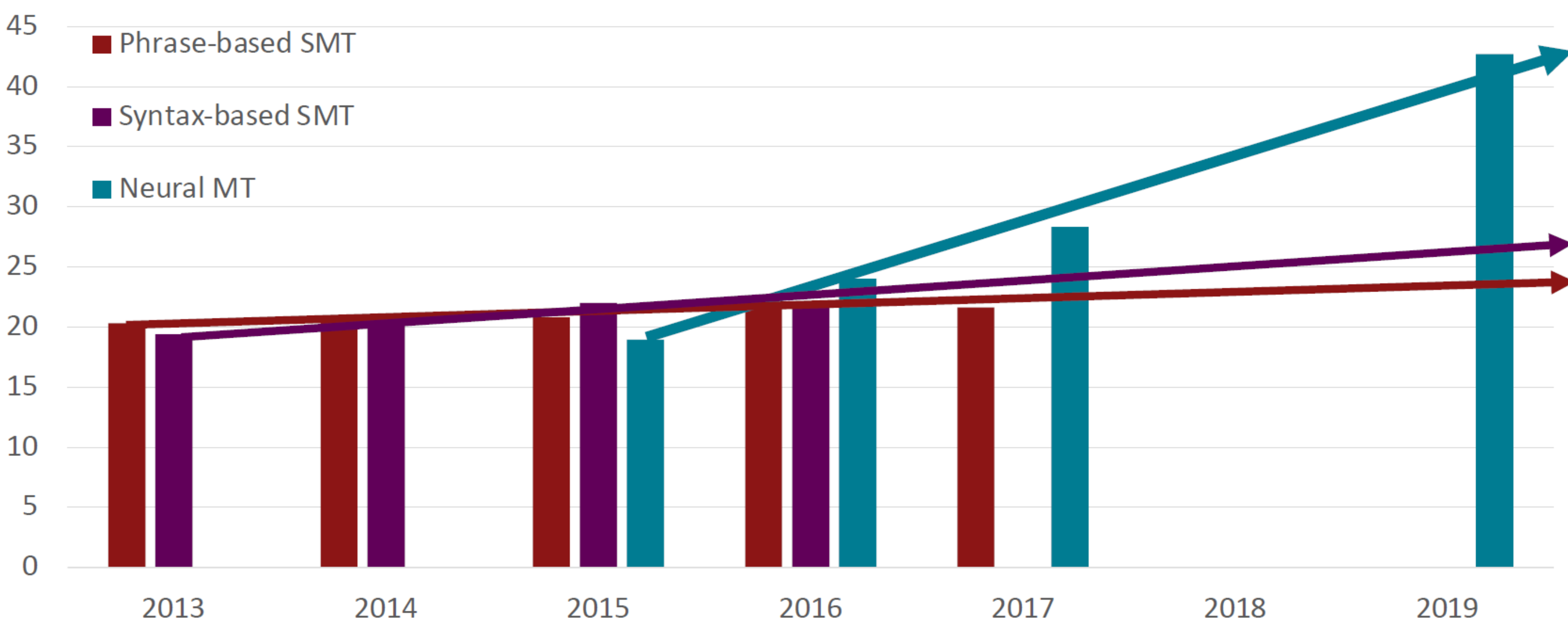
Deep Sequence to Sequence Model

- Stacked seq2seq model



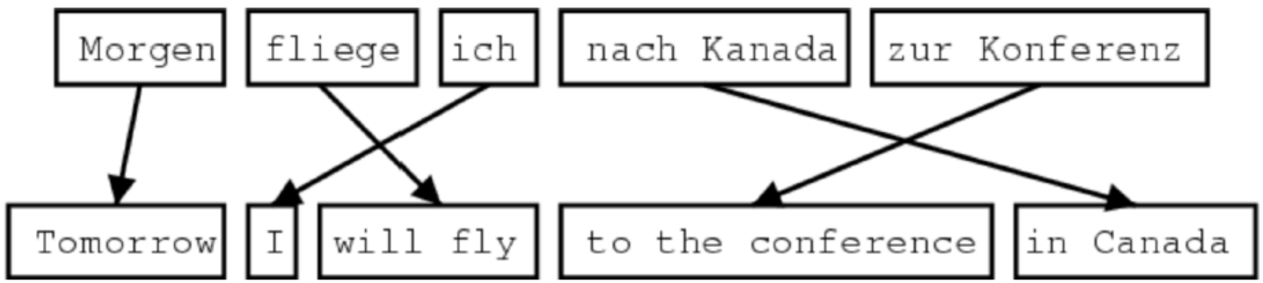
Machine Translation

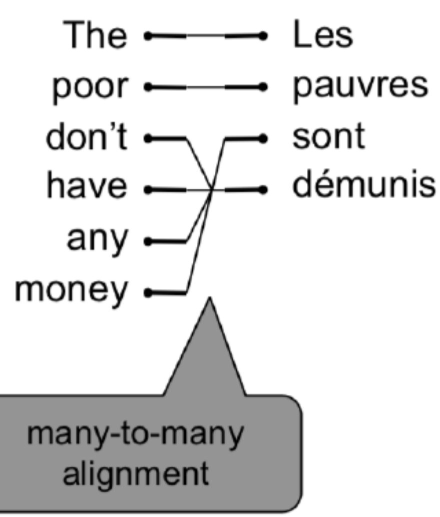
- 2016: Google switched Google Translate from SMT to NMT

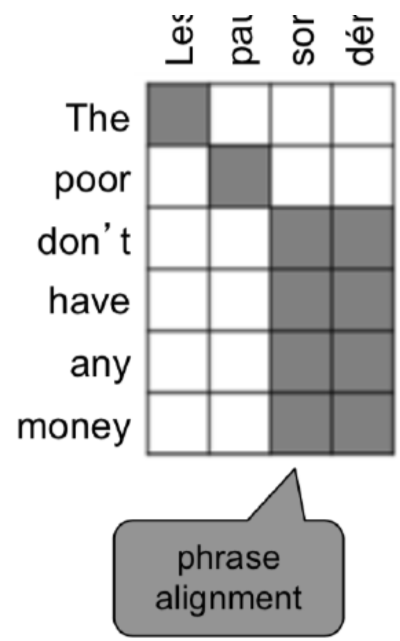


Alignment

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

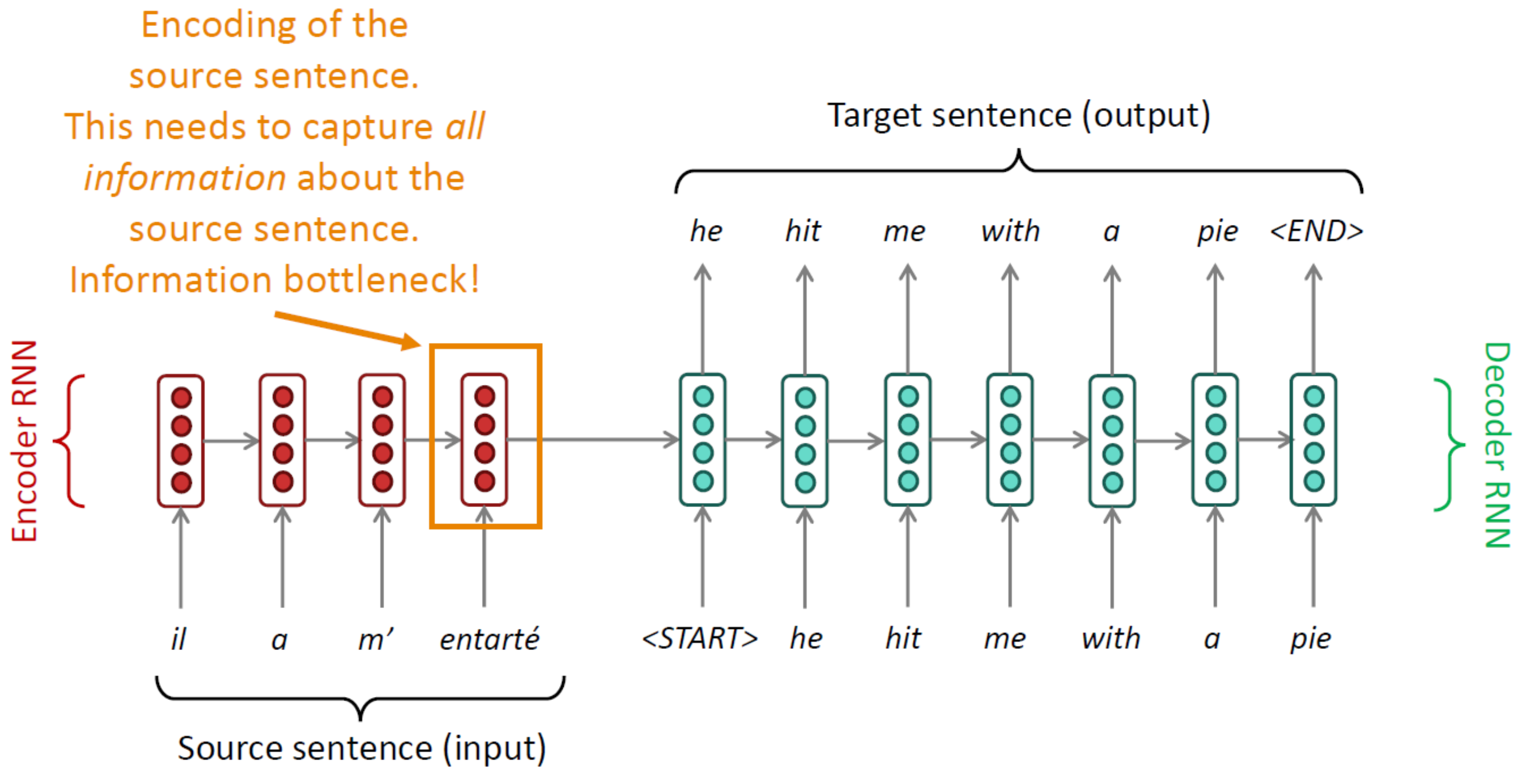






Issue in Seq2Seq

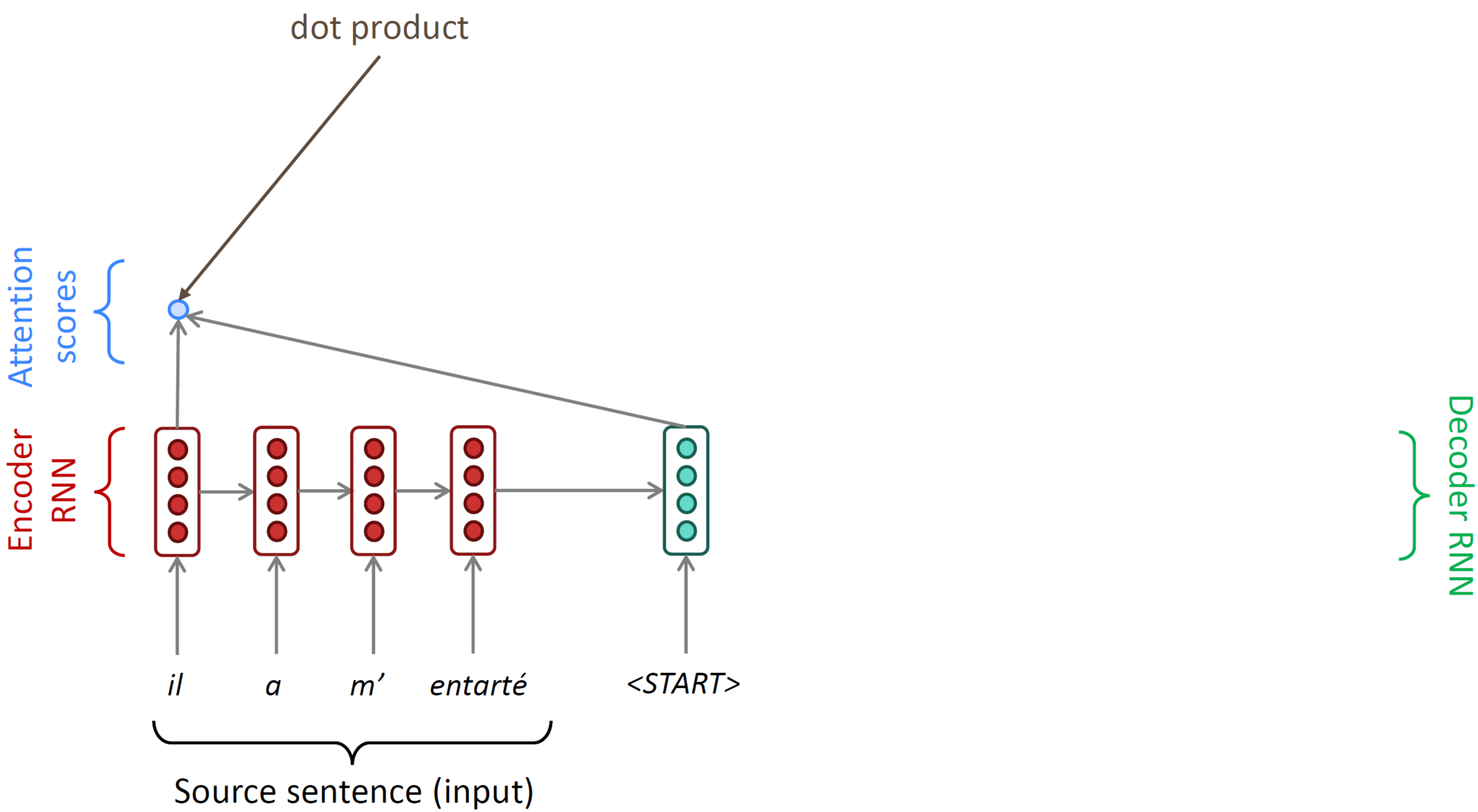
- Alignment: the word-level correspondence between X and Y
  - The information bottleneck due to the hidden state h
  - We want each Y_t to also focus on some X_i that it is aligned with



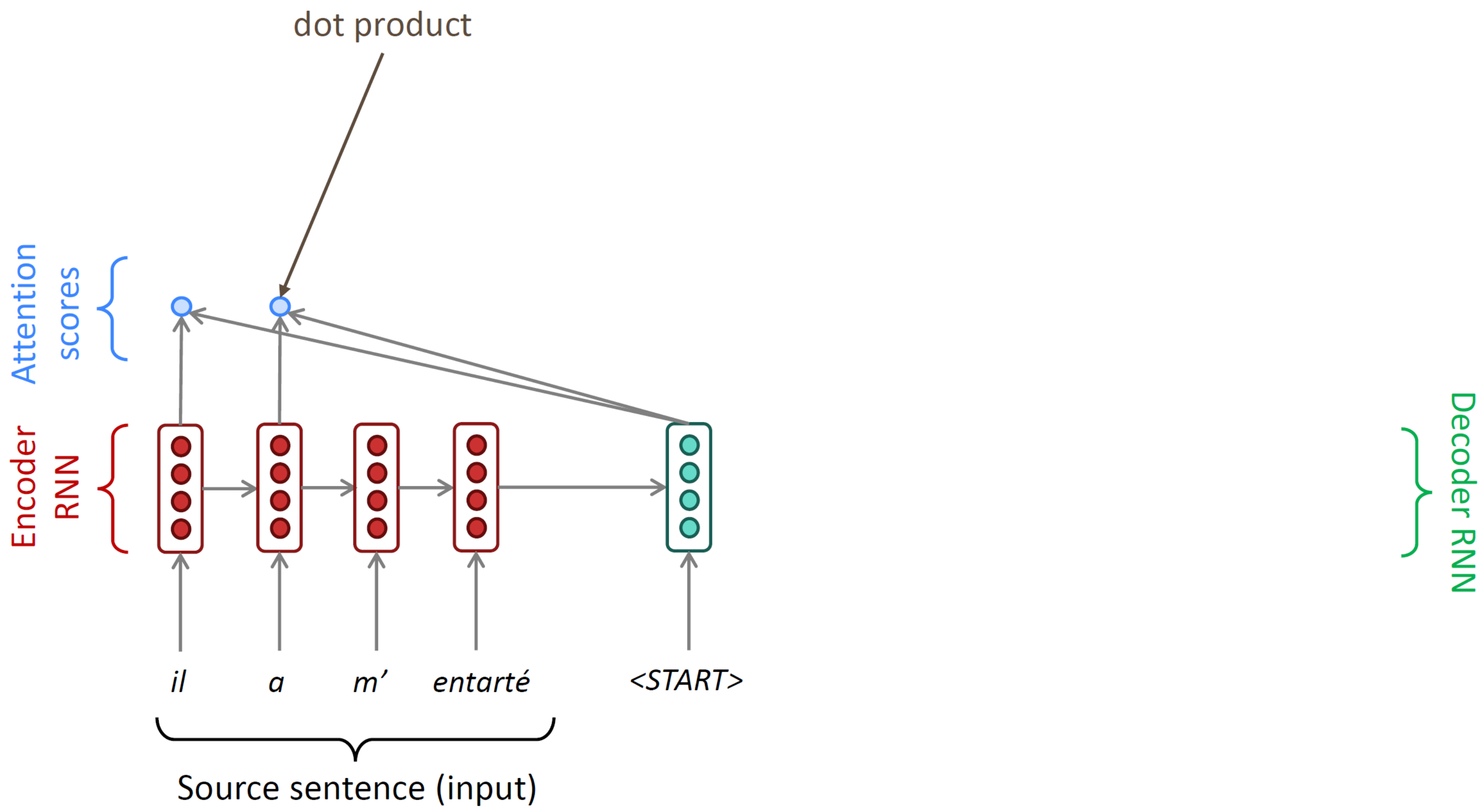
Seq2Seq with Attention

- 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

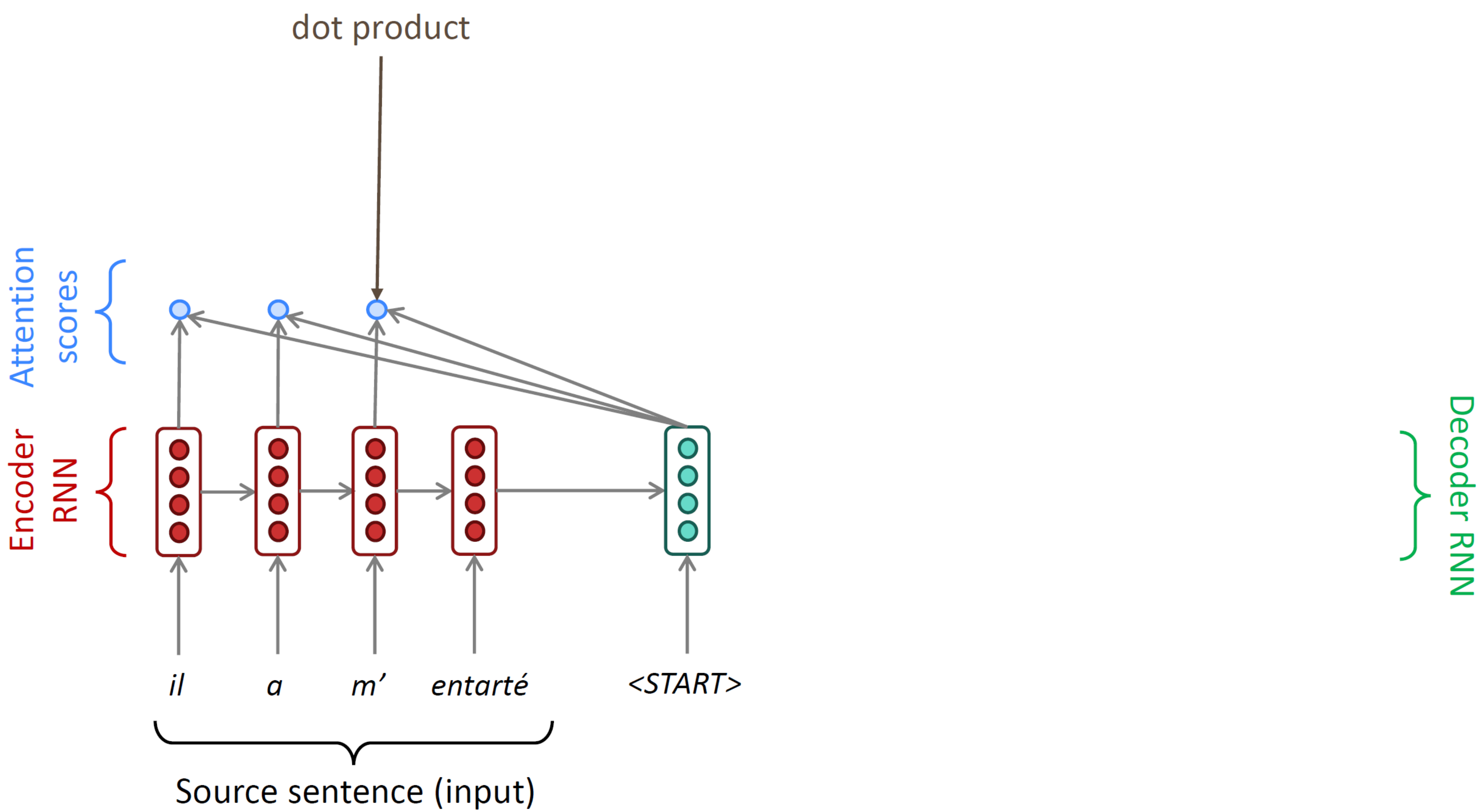
Seq2Seq with Attention



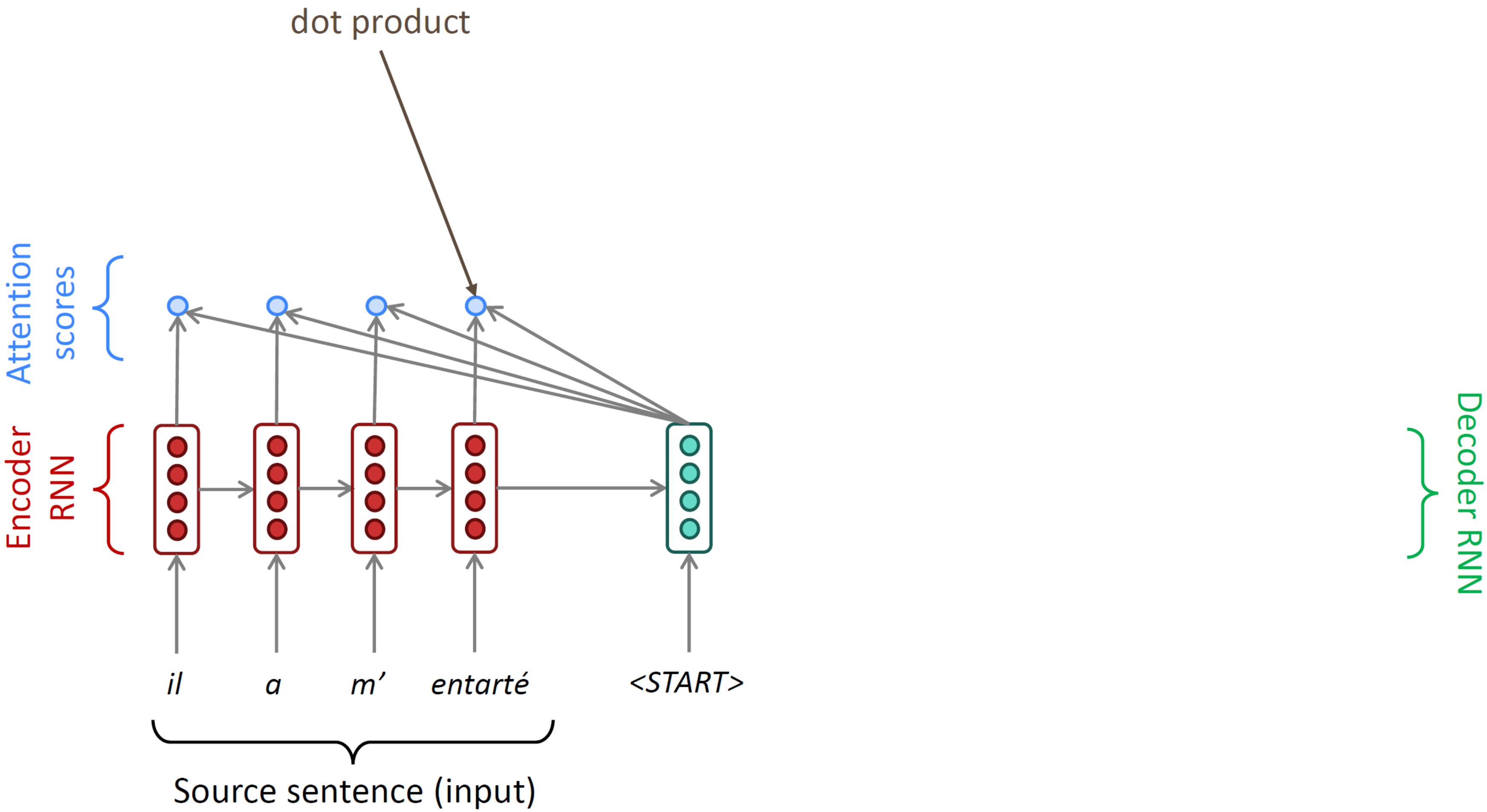
Seq2Seq with Attention



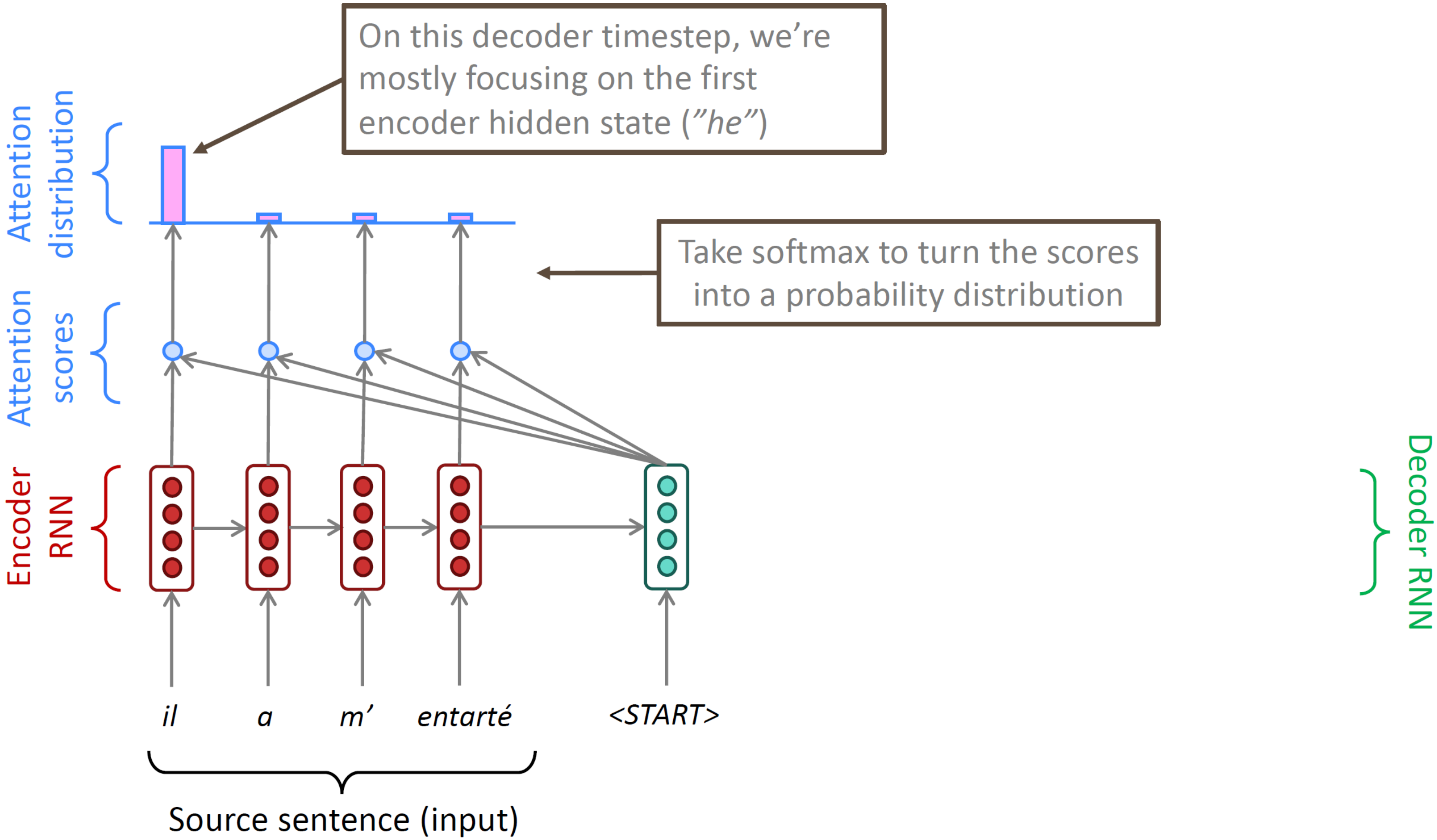
Seq2Seq with Attention



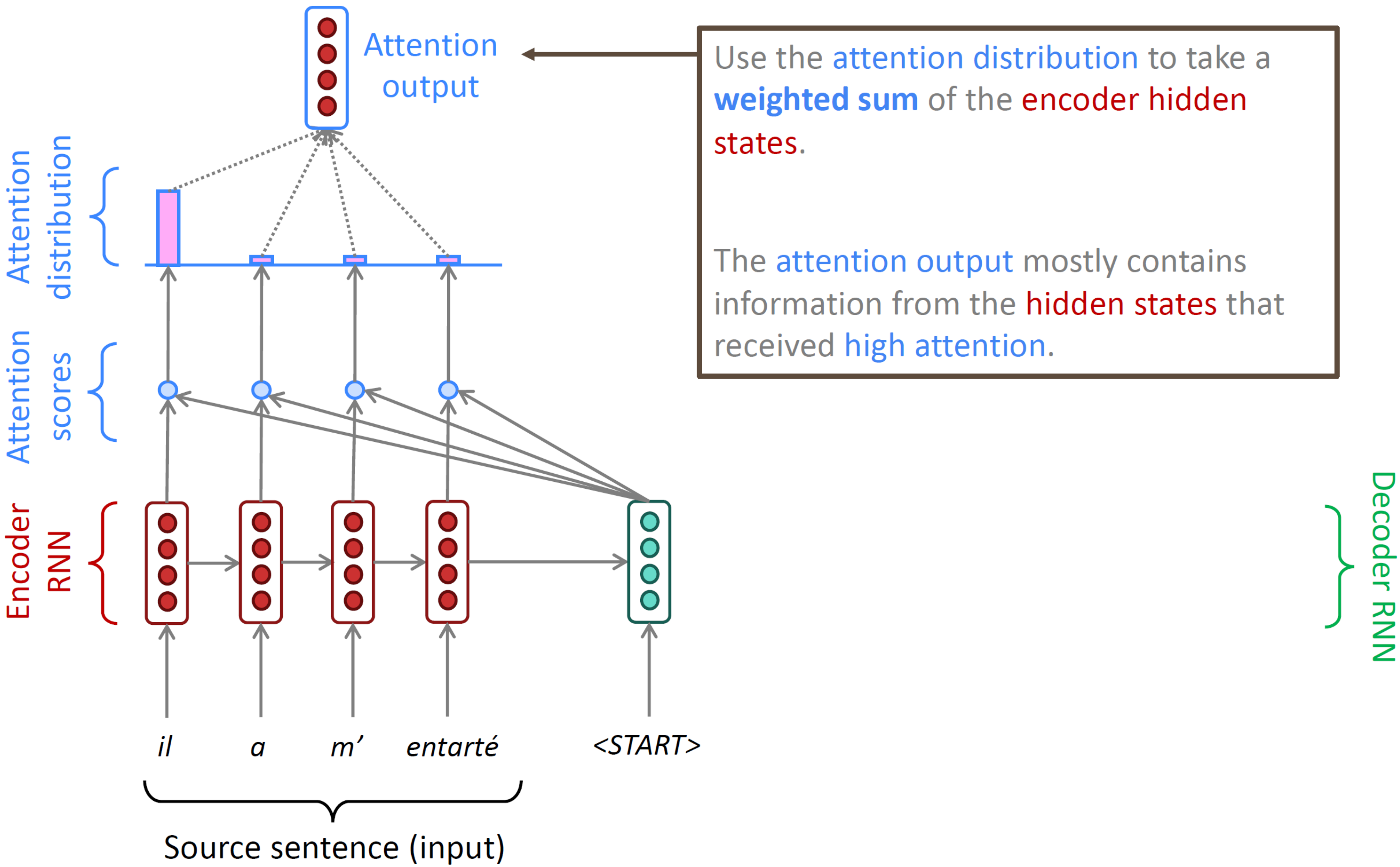
Seq2Seq with Attention



Seq2Seq with Attention



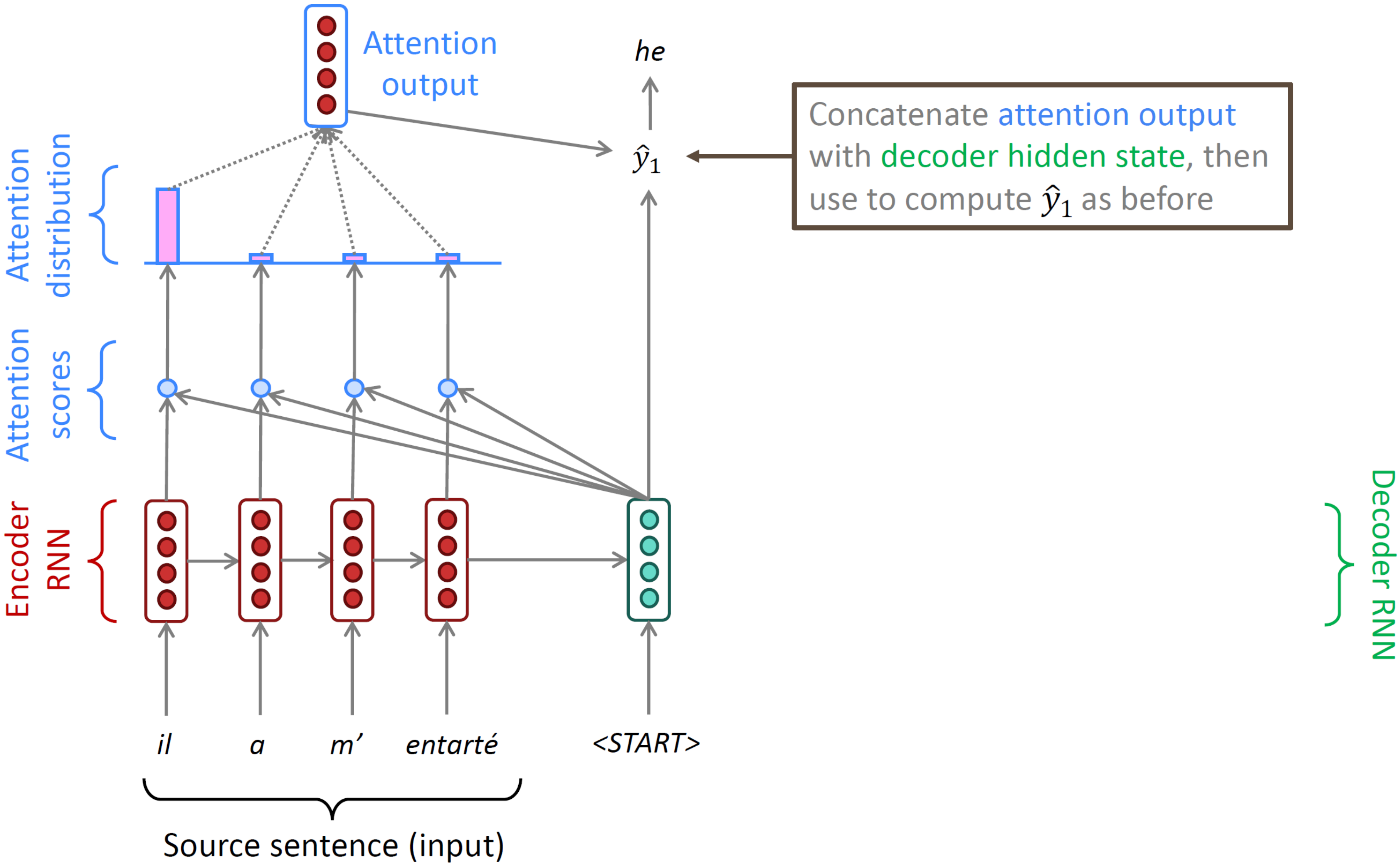
Seq2Seq with Attention



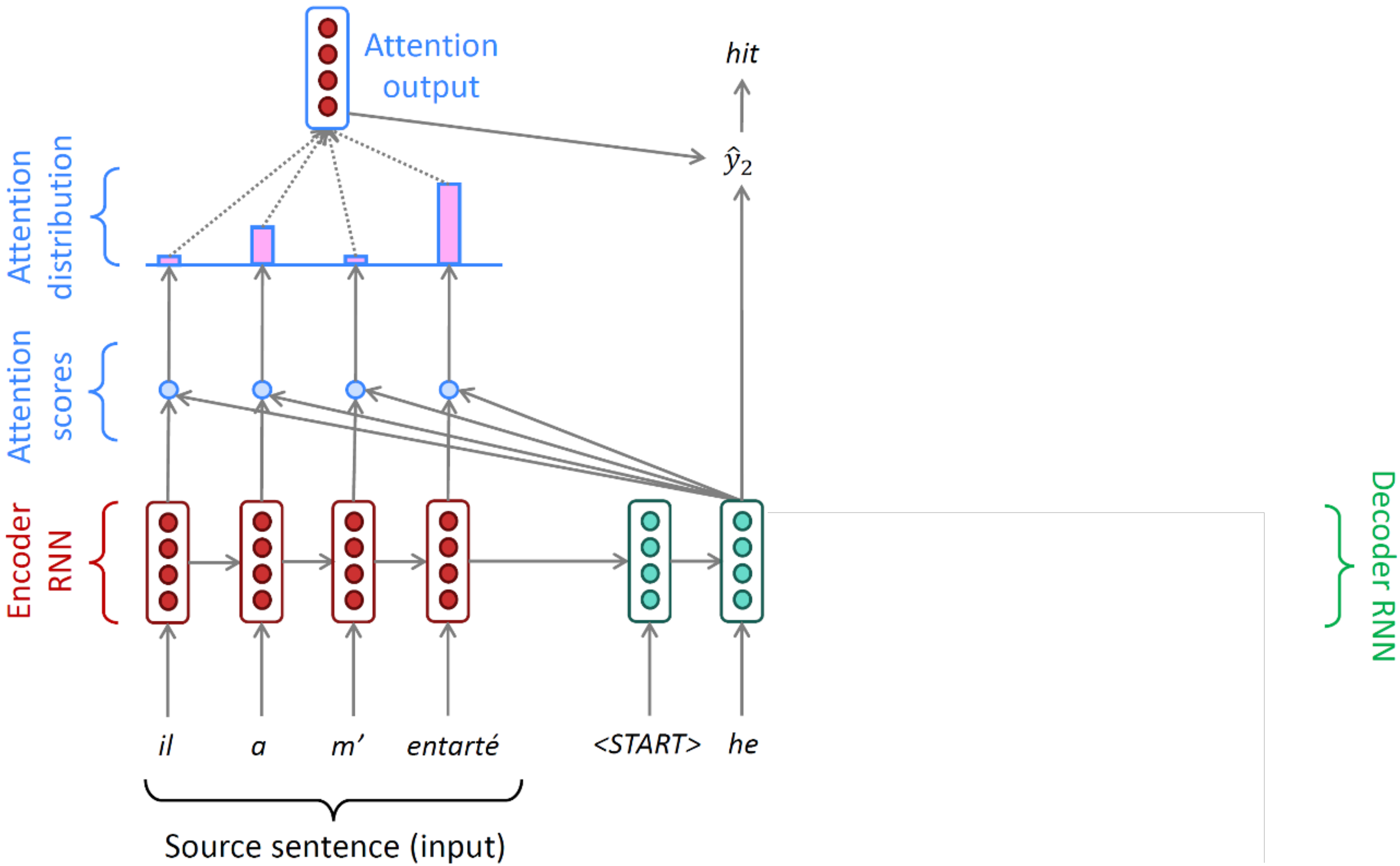
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

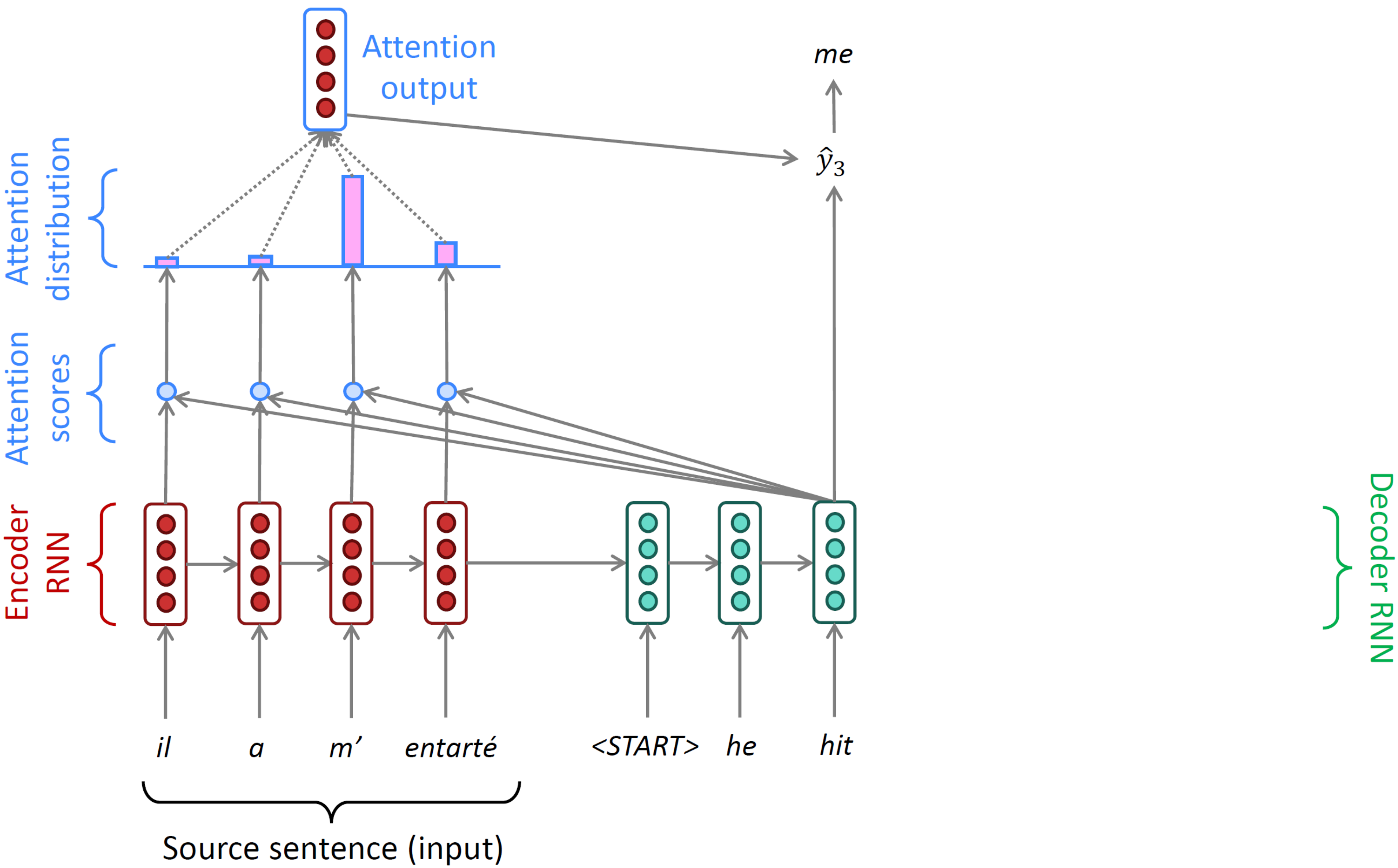
Seq2Seq with Attention



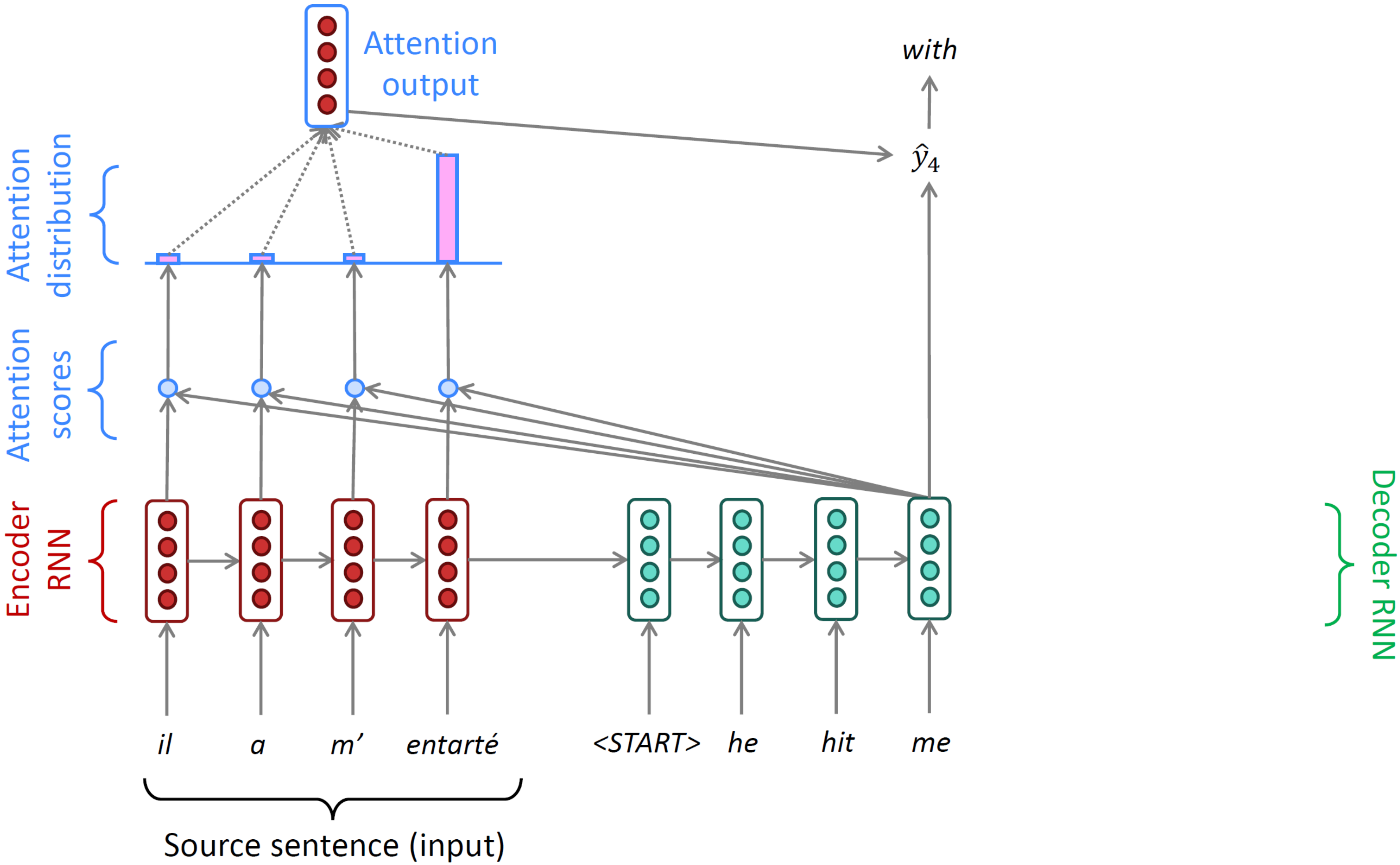
Seq2Seq with Attention



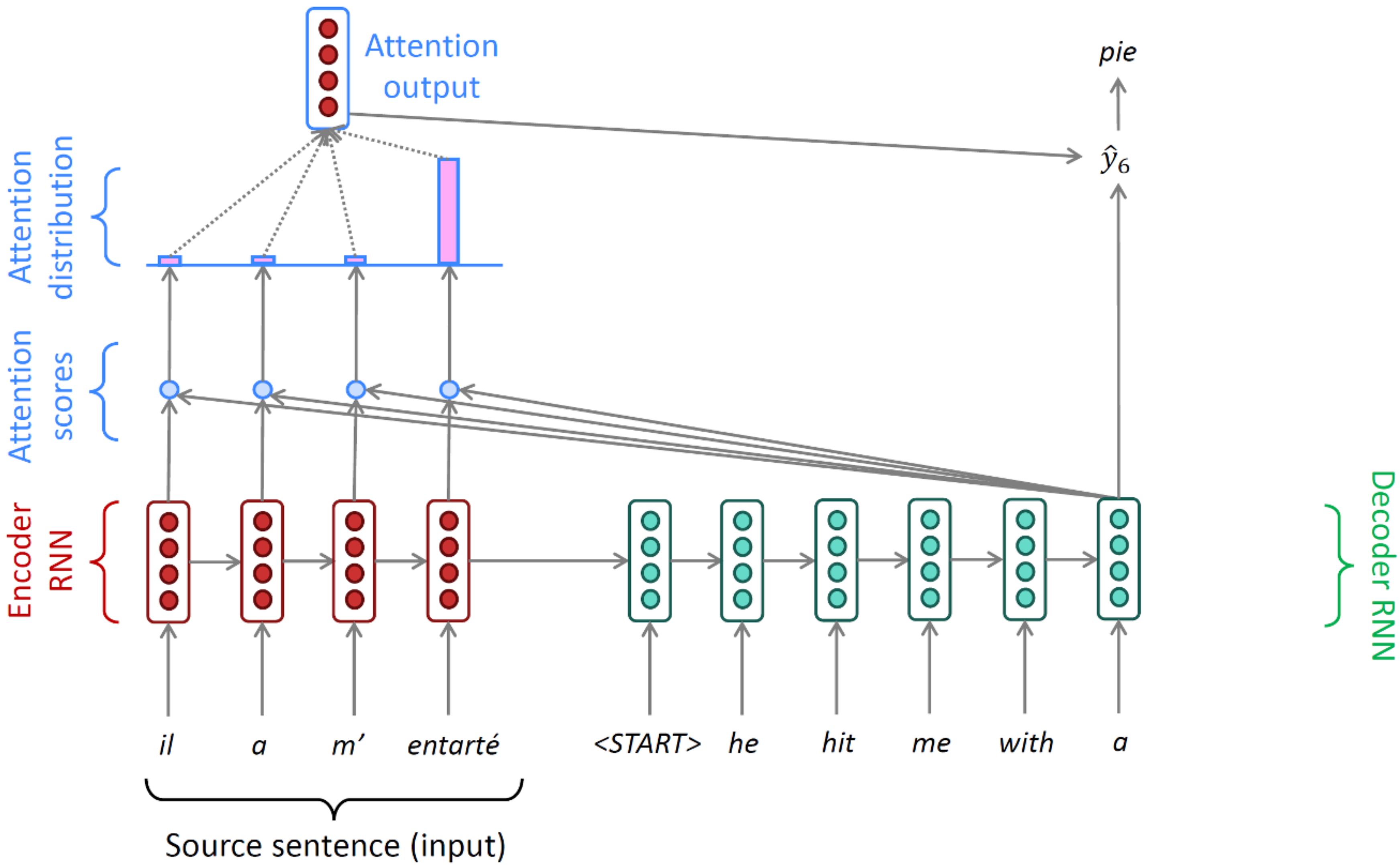
Seq2Seq with Attention



Seq2Seq with Attention



Seq2Seq with Attention



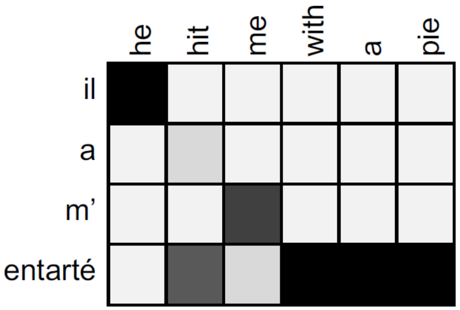
Seq2Seq with Attention

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)

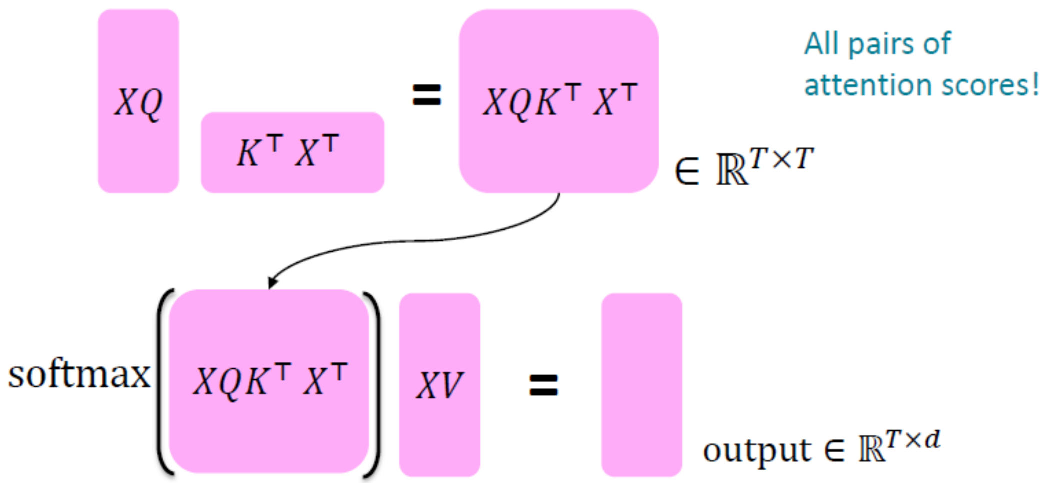
Attention

- 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: $f(v_i, q) = u^\top \tanh(W_1 v_i + W_2 q)$

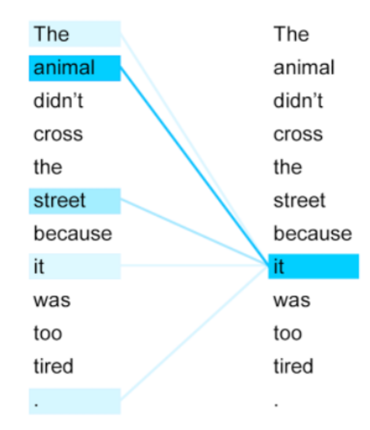
Key-query-value attention

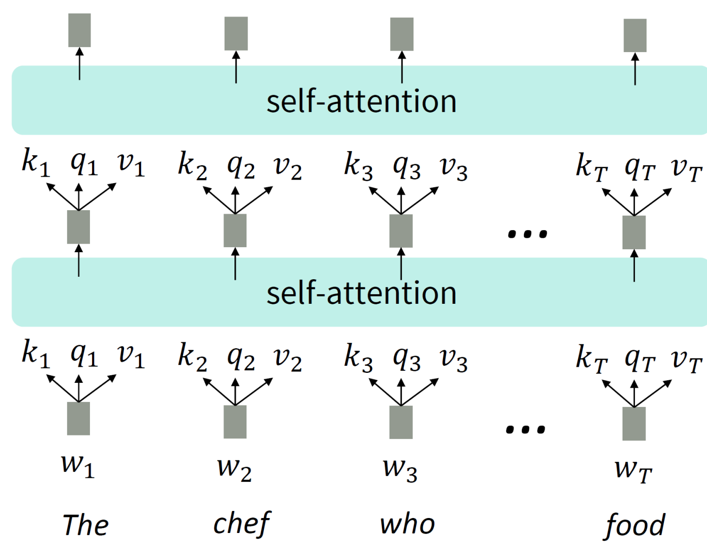
- Obtain q_t, v_t, k_t from X_t
- $q_t = W^q X_t; v_t = W^v X_t; k_t = W^k X_t$ (position encoding omitted)
  - W^q, W^v, W^k are learnable weight matrices
- $\alpha_{i,j} = \text{softmax}(q_i^\top k_j); out_i = \sum_k \alpha_{i,j} v_j$
- Intuition: key, query, and value can focus on different parts 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
  - $(k_t, v_t, q_t) = g_1([X_t, p_t]; \theta)$

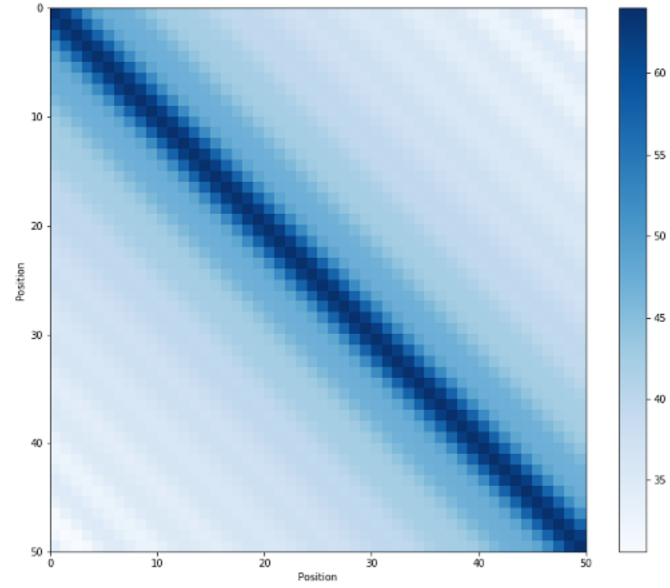
- In practice: Additive is sufficient: $k_t \leftarrow \tilde{k}_t + p_t, q_t \leftarrow \tilde{q}_t + p_t, v_t \leftarrow \tilde{v}_t + p_t;$ $(\tilde{k}_t, \tilde{v}_t, \tilde{q}_t) = g_1(X_t; \theta)$

- 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

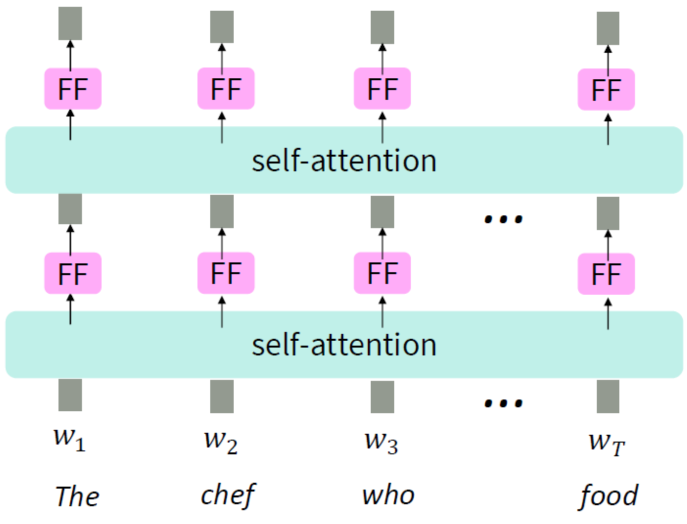
p_t design 2: **Learned representation**

- 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

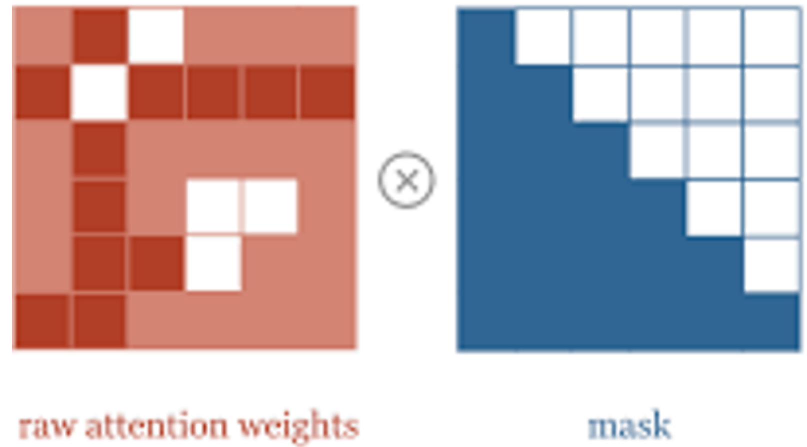
- Fix:
  - Add an MLP to process out_i
  - $m_i = MLP(out_i) = W_2 \text{ReLU}(W_1 out_i + b_1) + b_2$
  - Usually do not put activation layer before softmaax

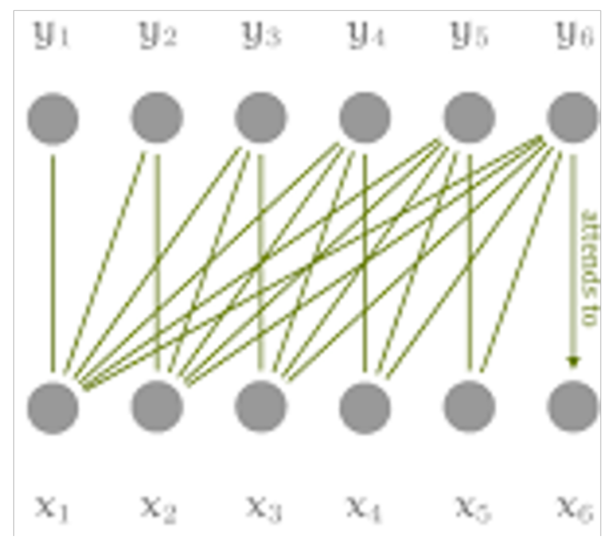


Masked Attention

- In language model decoder: $P(Y_t | X_{i<t})$
  - out_t cannot look at future $X_{i>t}$

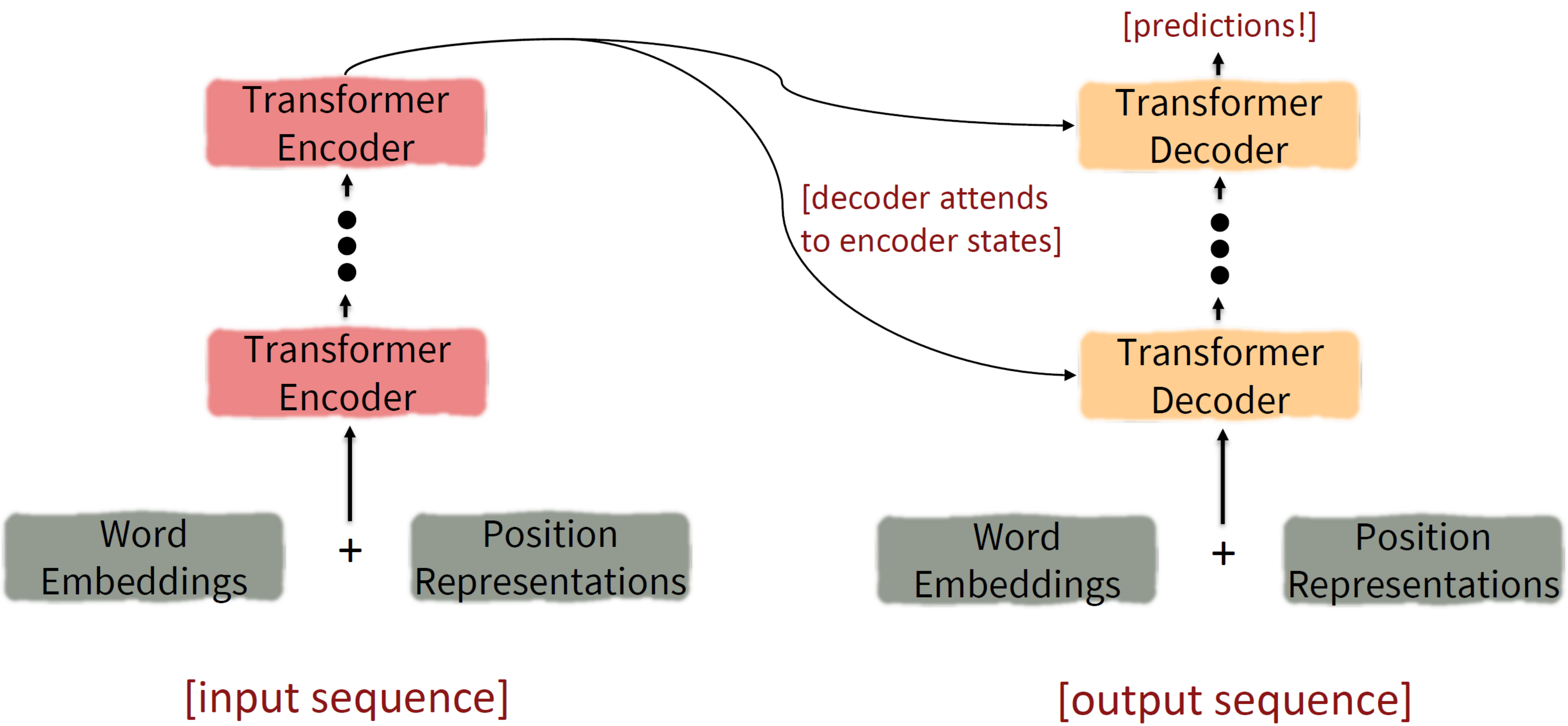
- Masked attention
  - Compute $e_{i,j} = q_i^\top k_j$ as usuall
  - Mask out $e_{i>j}$ by setting $e_{i>j} = -\infty$
    - $e \odot (1 - M) \leftarrow -\infty$
    - M is a fixed 0/1 mask matrix
  - Then compute $\alpha_i = \text{softmax}(e_i)$
  - Remarks:
    - $M = 1$ for full self-attention
    - Set M for arbitrary dependency ordering





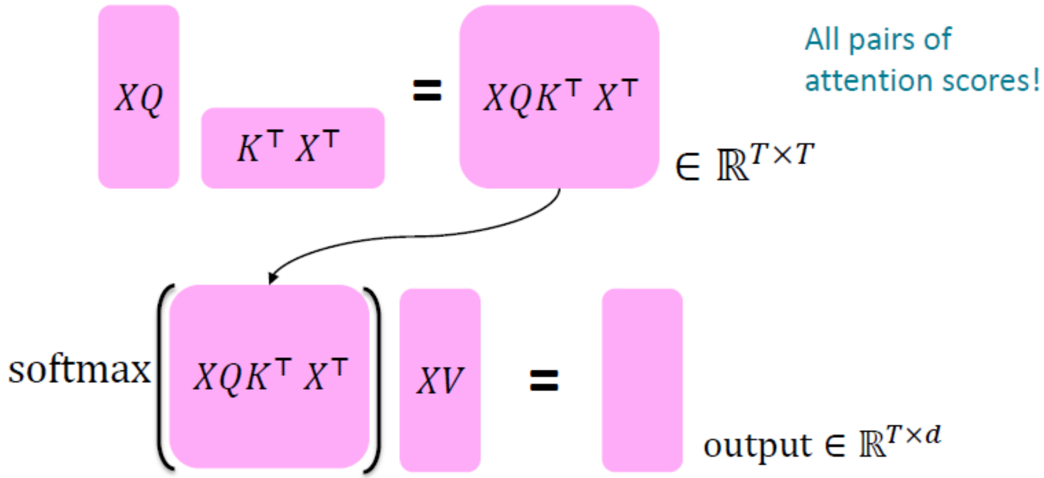
Transformer

Transformer-based sequence-to-sequence modeling



Key-query-value attention

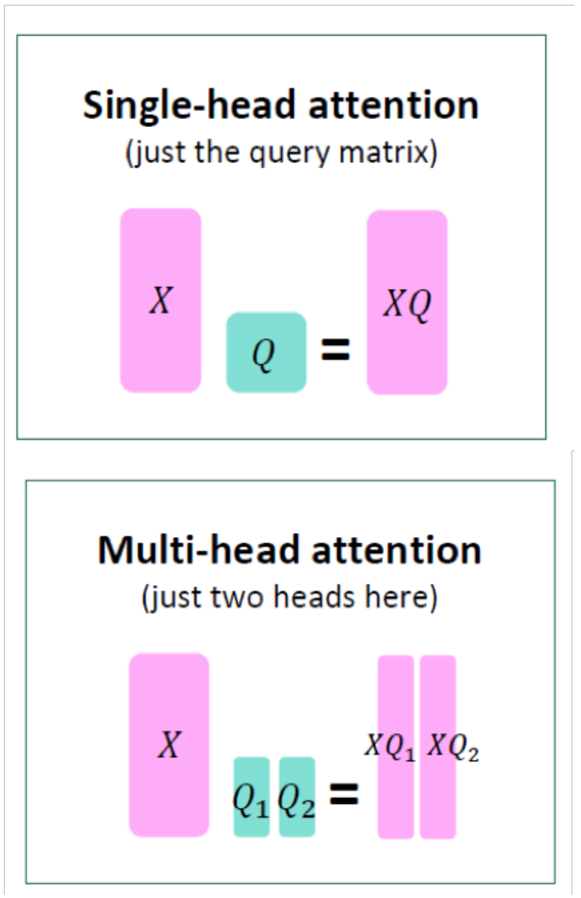
- Obtain q_t, v_t, k_t from X_t
- $q_t = W^q X_t; v_t = W^v X_t; k_t = W^k X_t$ (position encoding omitted)
  - W^q, W^v, W^k are learnable weight matrices
- $\alpha_{i,j} = \text{softmax}(q_i^\top k_j); out_i = \sum_k \alpha_{i,j} v_j$
- Intuition: key, query, and value can focus on different parts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



Multi-headed attention

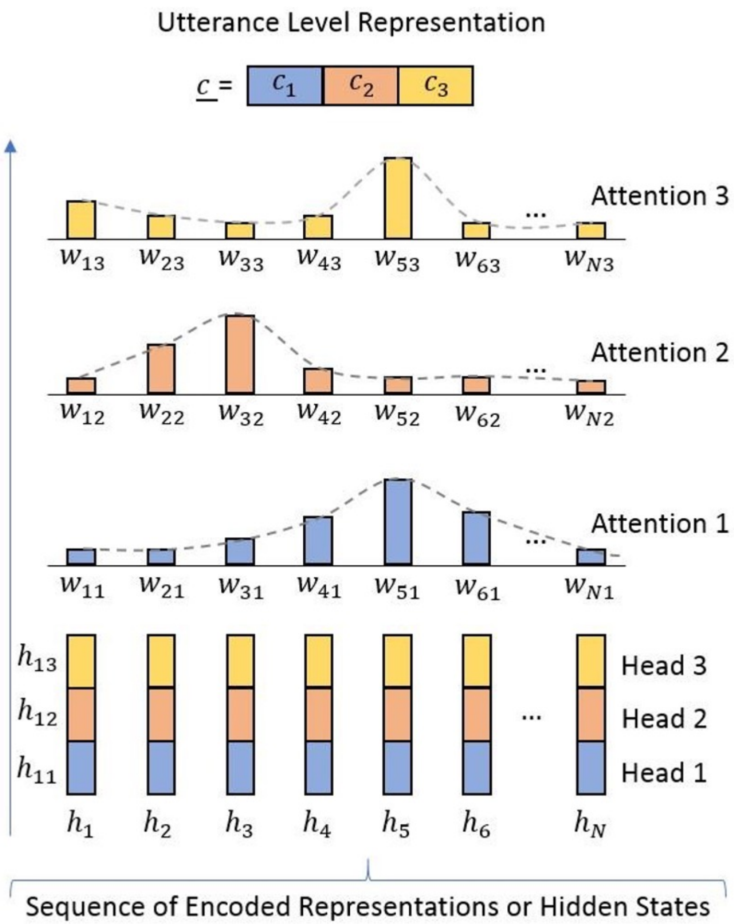
- Standard attention: single-headed attention
  - $X_t \in \mathbb{R}^d$, $Q, K, V \in \mathbb{R}^{d \times d}$
  - We only look at a single position j with high $\alpha_{i,j}$
  - What if we want to look at different j for different reasons?
- Idea: define h separate attention heads
  - h different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

#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 $1 \leq \ell \leq h$
  - $\alpha_{i,j}^\ell = \text{softmax}((q_i^\ell)^\top k_j^\ell); out_i^\ell = \sum_j \alpha_{i,j}^\ell v_j^\ell$

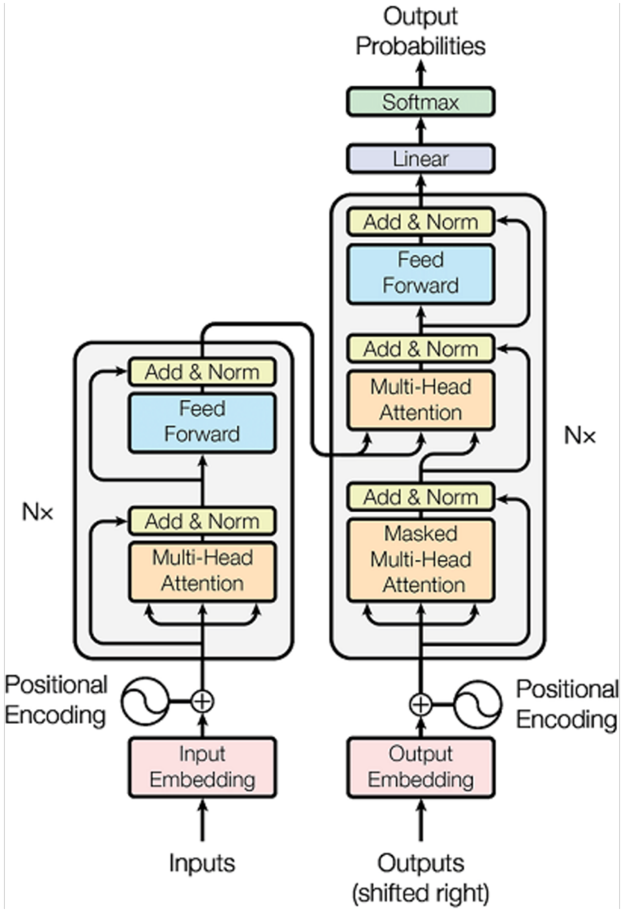


Transformer

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

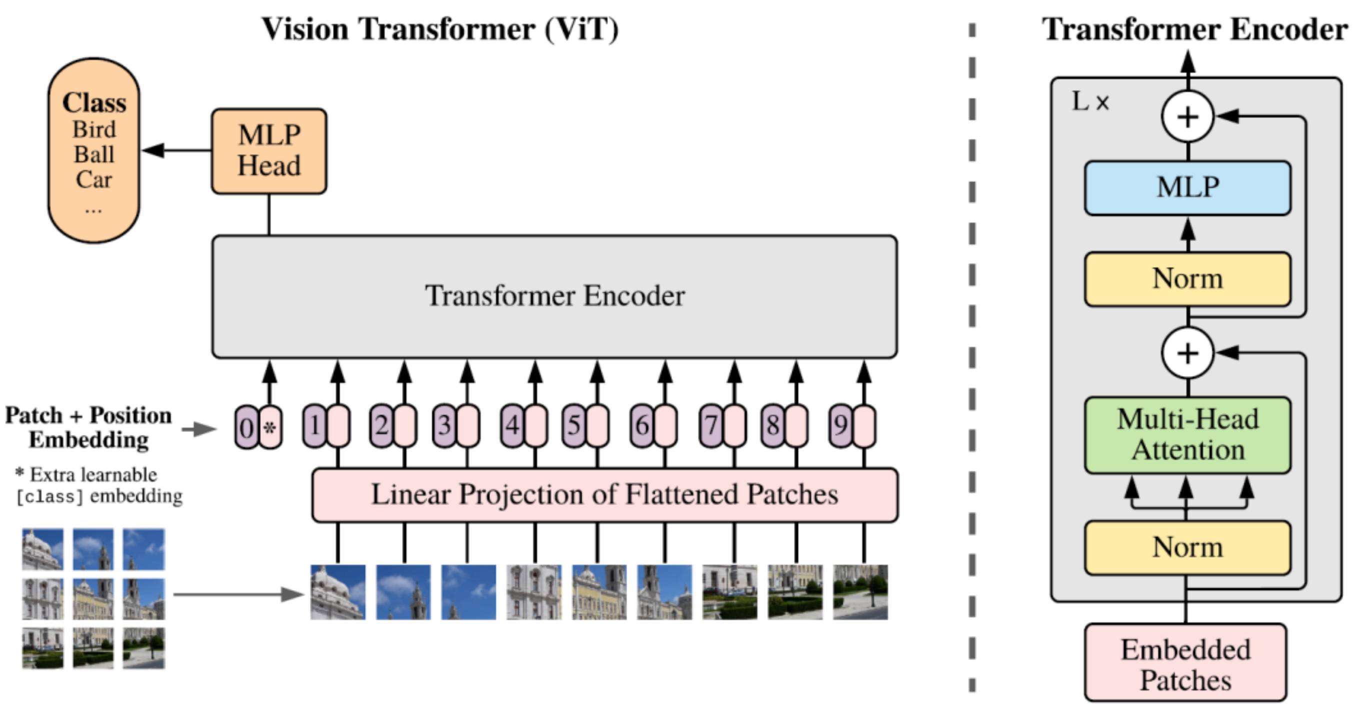
Transformer

- 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)

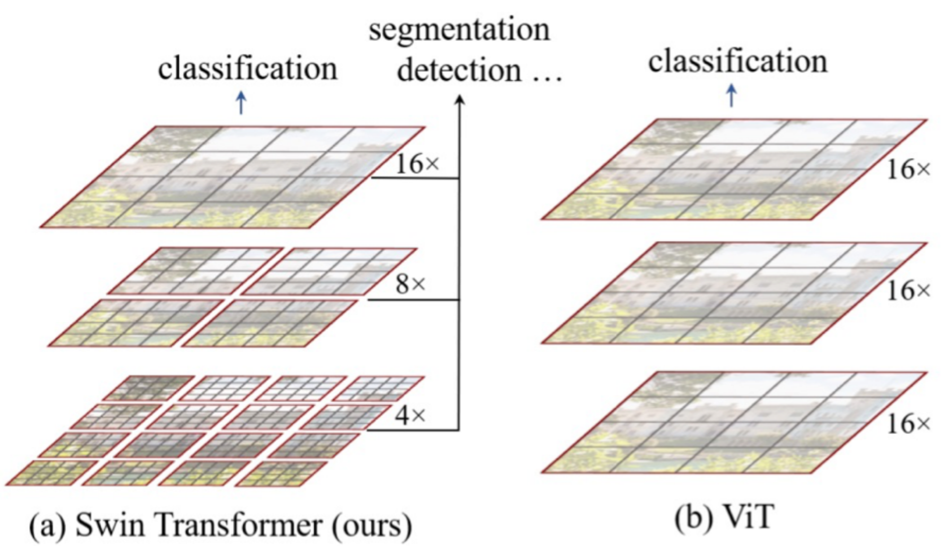
Transformer for Images

- 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



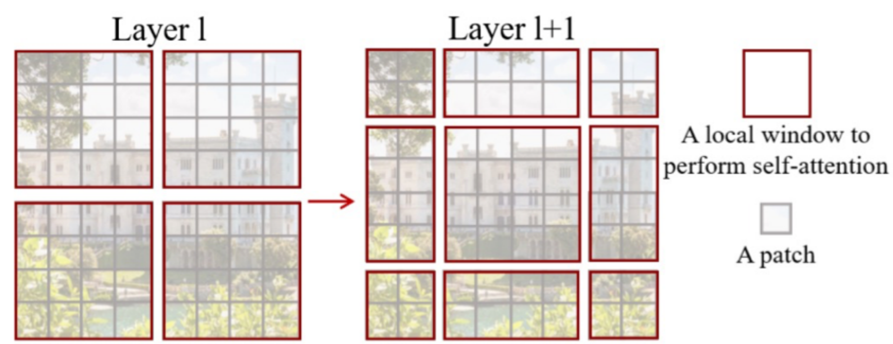
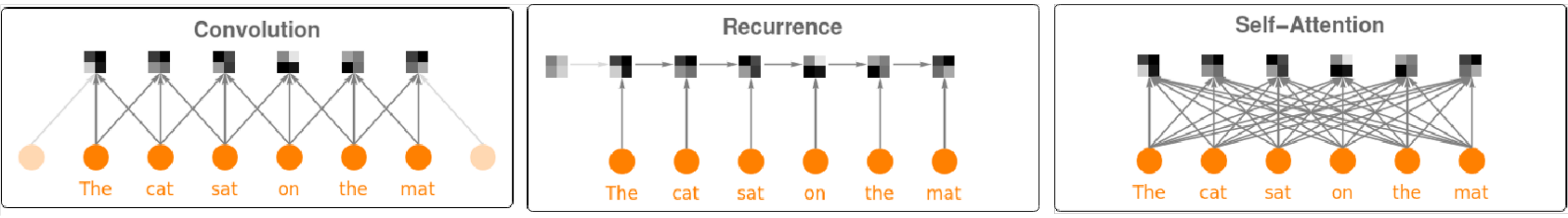


Figure 2. An illustration of the *shifted window* approach for com-

CNN vs. RNN vs. Attention



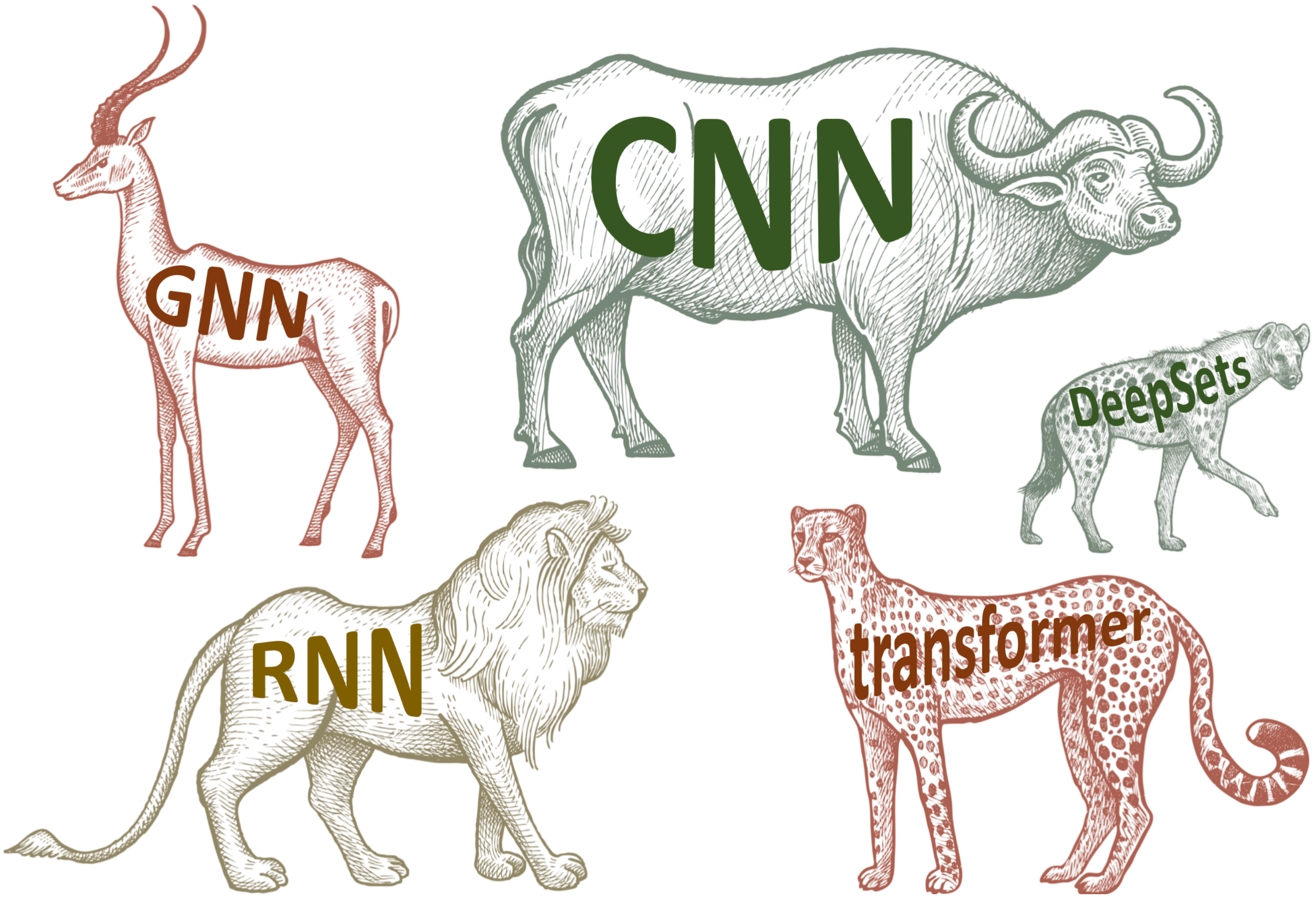
Summary

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

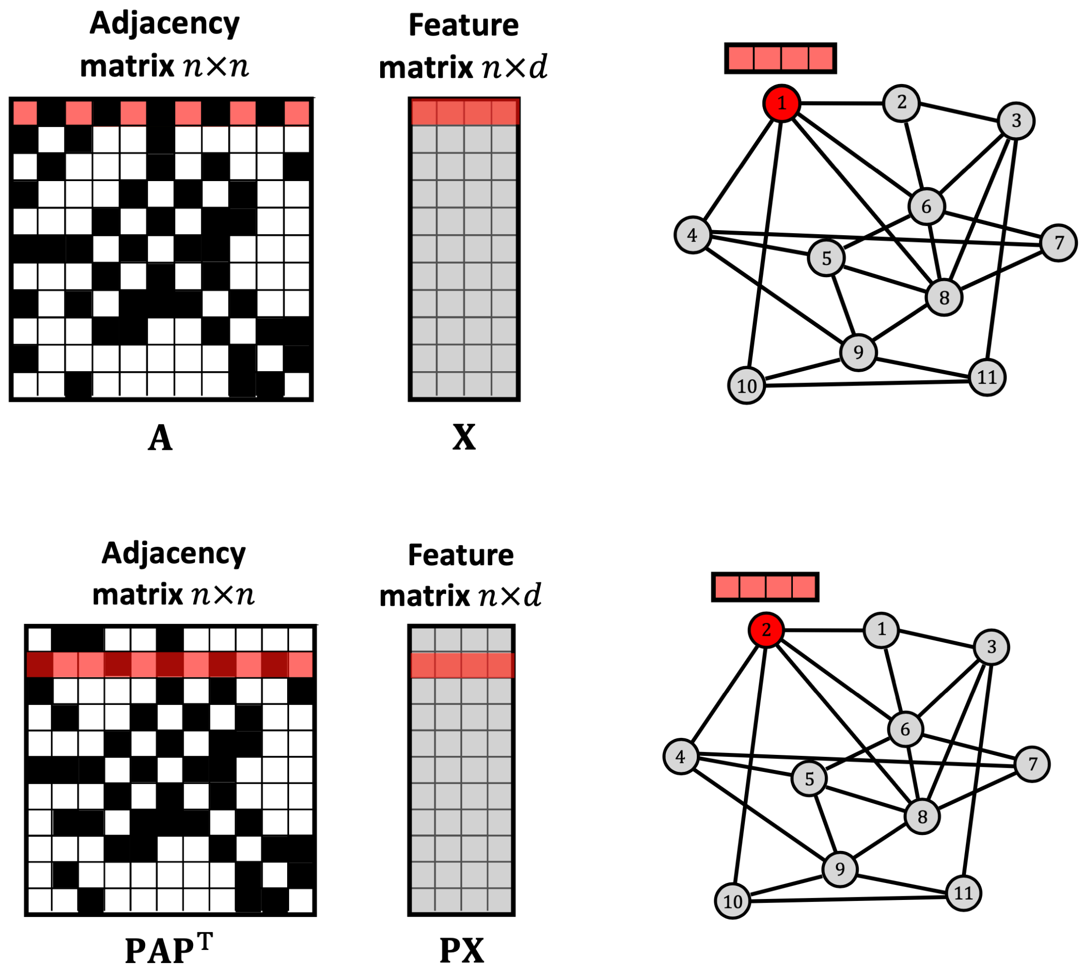
- Attention mechanism
  - So far the most successful idea for sequence data in deep learning
  - A scale/order-invariant representation
  - Transformer: a fully attention-based architecture for sequence data
  - Transformer + Pretraining: the core idea in today's NLP tasks

- LSTM is still useful in lightweight scenarios

Other architectures



Graph Neural Networks

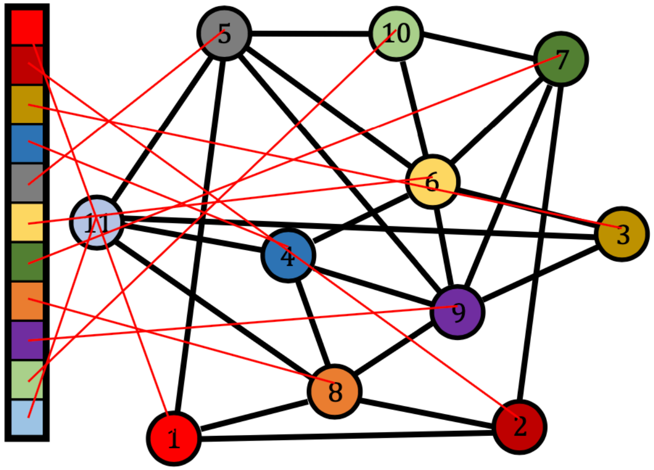


arbitrary ordering of nodes

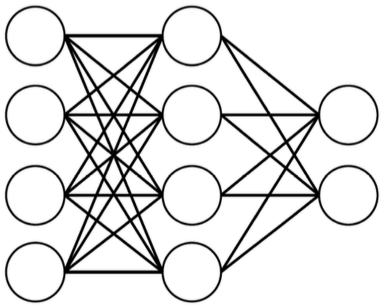
Graph Neural Networks

permutation-equivariant

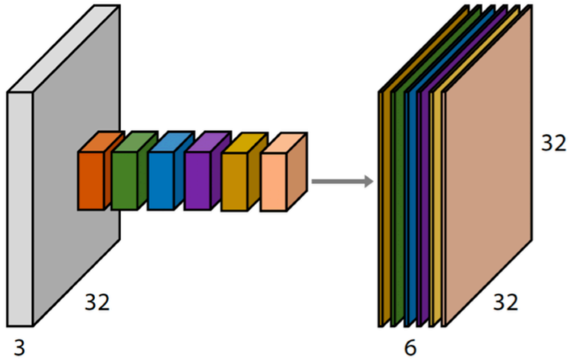
$$\mathbf{F}\left(\mathbf{PX},\mathbf{PAP}^{\top}\right)=\mathbf{PF}(\mathbf{X},\mathbf{A})$$



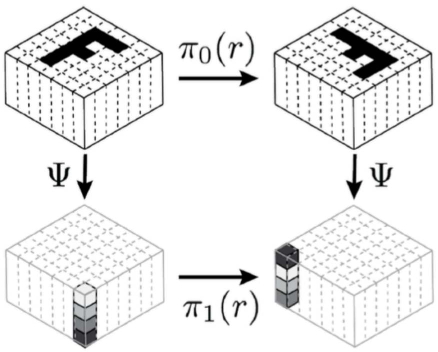
Geometric Deep Learning



Perceptrons
Function regularity

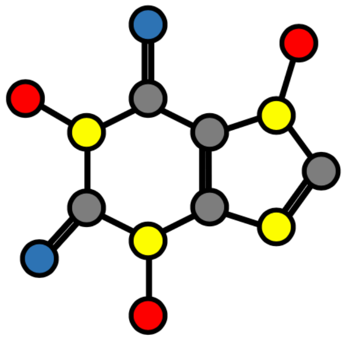


CNNs
Translation



Group-CNNs
Translation+Rotation

DeepSets / Transformers
Permutation



GNNs
Permutation

Intrinsic CNNs
Local frame choice

Supervised Learning Process

Collect a **dataset**

Decide on a **model**

Find the function which fits the data best

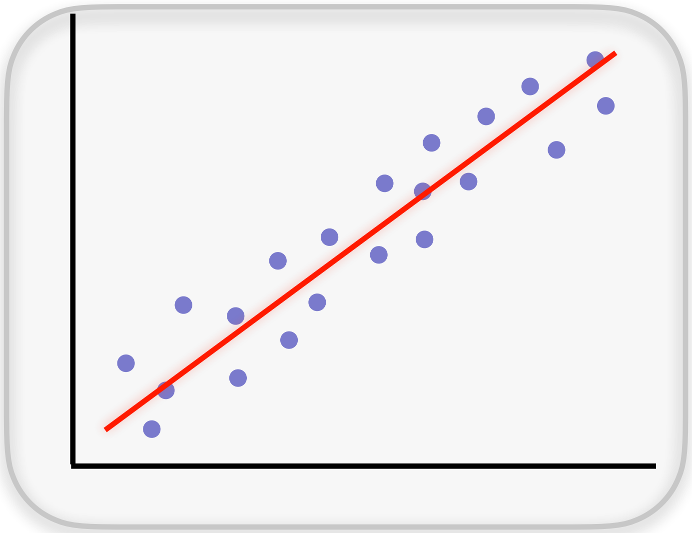
- **Choose a loss function**
- **Pick the function which minimizes loss on data**

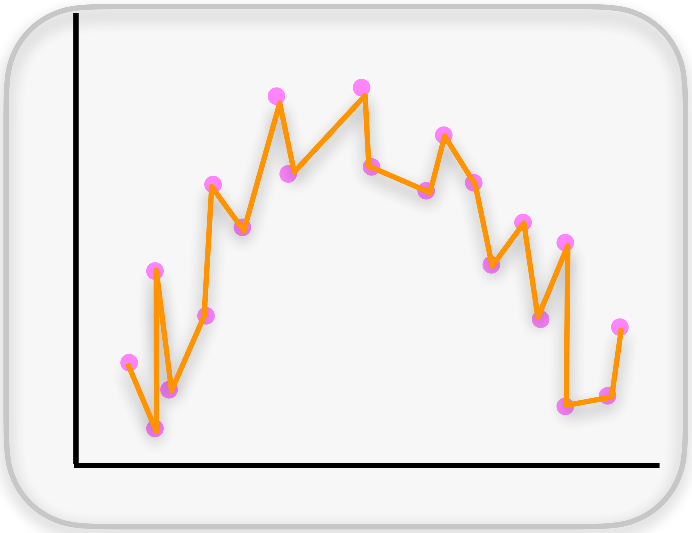
Use function to make prediction on new examples

Framework

Approximation Theory

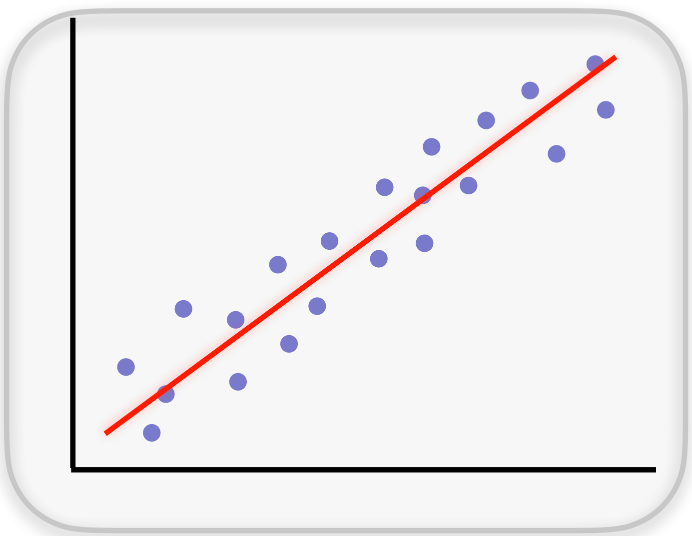
Expressivity / Representation Power





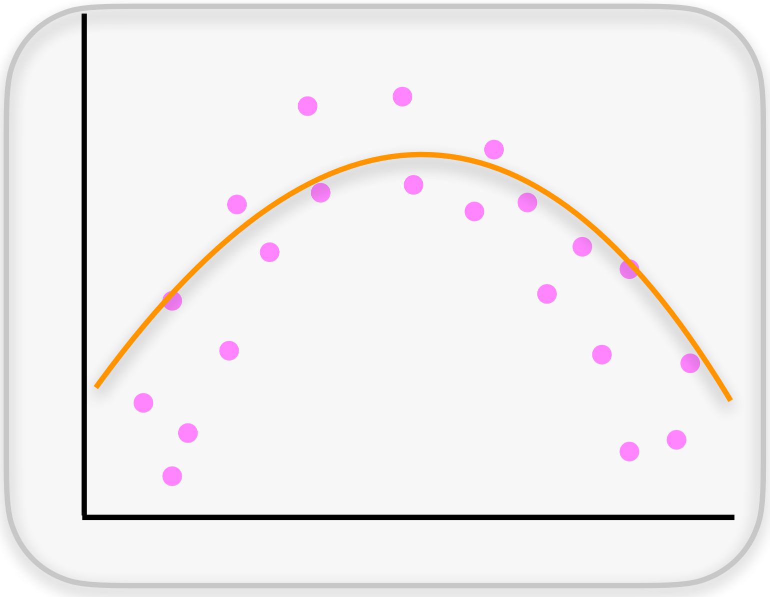
Expressive: Functions in class can represent "complicated" functions.

Linear Function



best linear fit

Review: generalized linear regression



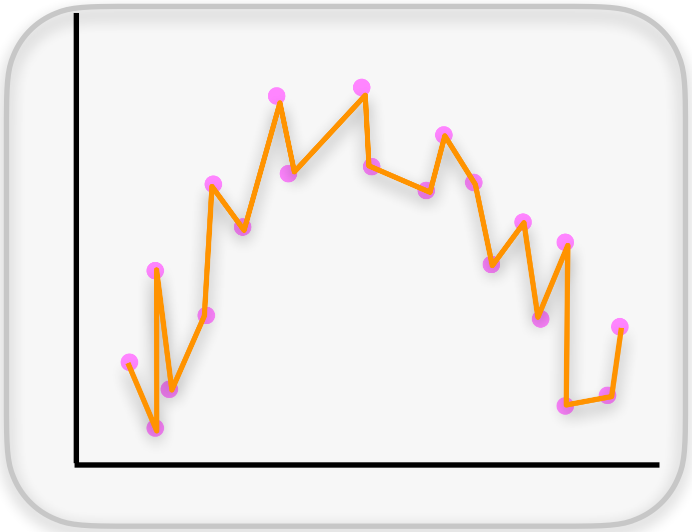
Transformed data:

$$h(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_p(x) \end{bmatrix}$$

Hypothesis: linear in h

$$y_i \approx h(x_i)^T w$$

Review: Polynomial Regression



Approximation Theory Setup

- Goal: to show there exists a neural network that has small error on training / test set.

- Set up a natural baseline:

$$\inf_{f \in \mathcal{F}} L(f) \text{ v.s. } \inf_{g \in \text{ continuous functions}} L(g)$$

Example

Decomposition

Specific Setups

- "Average" approximation: given a distribution μ

$$\|f - g\|_\mu = \int_x |f(x) - g(x)|\, d\mu(x)$$

- "Everywhere" approximation

$$\|f - g\|_\infty = \sup_x |f(x) - g(x)| \geq \|f - g\|_\mu$$

Polynomial Approximation

Theorem (Stone-Weierstrass): for any function f, we can approximate it on any compact set Ω by a sufficiently high degree polynomial: for any $\epsilon > 0$, there exists a polynomial p of sufficient high degree, s.t.,

$$\max_{x \in \Omega} | f(x) - p(x) | \leq \epsilon.$$

Intuition: Taylor expansion!

Kernel Method

Polynomial kernel

Gaussian Kernel

1D Approximation

Theorem: Let $g : [0,1] \to R$, and ρ-Lipschitz. For any $\epsilon > 0, \exists$ 2-layer neural network f with $\lceil \frac{\rho}{\epsilon} \rceil$ nodes, threshold activation: $\sigma(z) : z \mapsto \mathbf{1}\{z \geq 0\}$ such that

$$\sup_{x \in [0,1]} |f(x) - g(x)| \leq \epsilon.$$

Proof of 1D Approximation

Multivariate Approximation

Theorem: Let g be a continuous function that satisfies $\|x - x'\|_\infty \leq \delta \Rightarrow |g(x) - g(x')| \leq \epsilon$ (Lipschitzness). Then there exists a 3-layer ReLU neural network with $O(\frac{1}{\delta^d})$ nodes that satisfy

$$\int_{[0,1]^d} |f(x) - g(x)|\, dx = \|f - g\|_1 \leq \epsilon$$

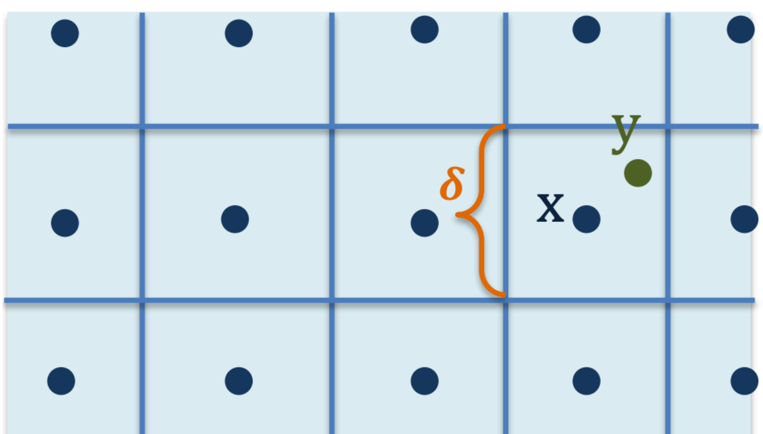
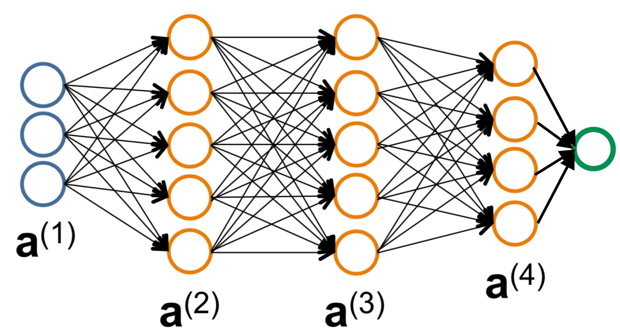


Figure credit to Andrej Risteski



Partition Lemma

Lemma: let g, δ, ϵ be given. For any partition P of $[0,1]^d$, $P = (R_1, \ldots, R_N)$ with all side length smaller than δ, there exists $(\alpha_1, \ldots, \alpha_N) \in \mathbb{R}^N$ such that

$$\sup_{x \in [0,1]^d} |g(x) - h(x)| \leq \epsilon \text{ with } h(x) := \sum_{i=1}^{N} \alpha_i \mathbf{1}_{R_i}(x).$$

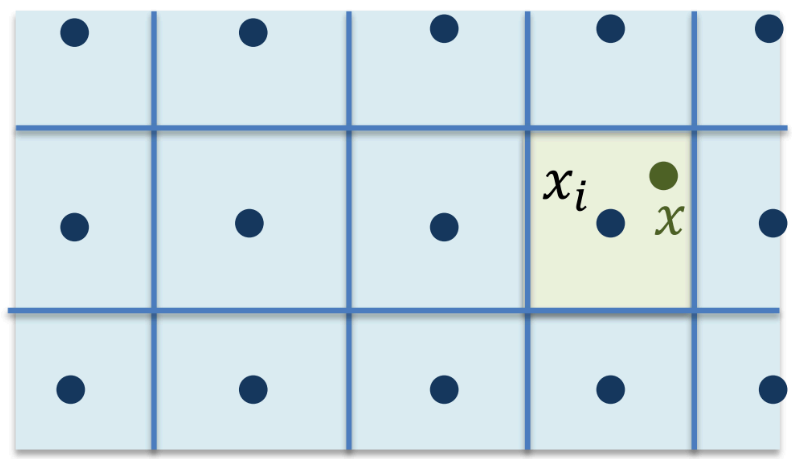


Figure credit to Andrej Risteski

Proof of Partition Lemma

Proof of Multivariate Approximation Theorem

Proof of Multivariate Approximation Theorem

Proof of Multivariate Approximation Theorem

Universal Approximation

Definition: A class of functions $\mathscr{F}$ is universal approximator over a compact set S (e.g., $[0,1]^d$), if for every continuous function g and a target accuracy $\epsilon > 0$, there exists $f \in \mathscr{F}$ such that

$$\sup_{x \in S} |f(x) - g(x)| \leq \epsilon$$

Stone-Weierstrass Theorem

Theorem: If $\mathscr{F}$ satisfies

1. Each $f \in \mathscr{F}$ is continuous.

2. $\forall x, \exists f \in \mathscr{F}, f(x) \neq 0$

3. $\forall x \neq x', \exists f \in \mathscr{F}, f(x) \neq f(x')$

4. $\mathscr{F}$ is closed under multiplication and vector space operations,

Then $\mathscr{F}$ is a universal approximator:

$$\forall g : S \to R, \epsilon > 0, \exists f \in \mathscr{F}, \|f - g\|_\infty \leq \epsilon.$$

Example: cos activation

Example: cos activation

Other Examples

Exponential activation

ReLU activation

Curse of Dimensionality

- Unavoidable in the worse case

Barron's Theory

- Can we avoid the curse of dimensionality for "nice" functions?
- What are nice functions?
  - Fast decay of the Fourier coefficients

- Fourier basis functions:
  $\{e_w(x) = e^{i\langle w,x\rangle} = \cos(\langle w, x\rangle) + i\sin(\langle w, x\rangle) \mid w \in \mathbb{R}^d\}$

- Fourier coefficient: $\hat{f}(w) = \int_{\mathbb{R}^d} f(x)e^{-i\langle w,x\rangle}dx$
- Fourier integral / representation: $f(x) = \int_{\mathbb{R}^d} \hat{f}(w)e^{i\langle w,x\rangle}dw$

Barron's Theorem

Definition: The Barron constant of a function f is:

$$C \triangleq \int_{\mathbb{R}^d} \|w\|_2 |\hat{f}(w)| \, dw.$$

Theorem (Barron '93): For any $g : \mathbb{B}_1 \to \mathbb{R}$ where $\mathbb{B}_1 = \{x \in \mathbb{R} : \|x\|_2 \leq 1\}$ is the unit ball, there exists a 3-layer neural network f with $O(\frac{C^2}{\epsilon})$ neurons and sigmoid activation function such that

$$\int_{\mathbb{B}_1} (f(x) - g(x))^2 dx \leq \epsilon.$$

Examples

- Gaussian function: $f(x) = (2\pi\sigma^2)^{d/2} \exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right)$

- Other functions:
  - Polynomials
  - Function with bounded derivatives

Proof Ideas for Barron's Theorem

Step 1: show any continuous function can be written as an infinite neural network with cosine-like activation functions.
(Tool: Fourier representation.)

Step 2: Show that a function with small Barron constant can be approximated by a convex combination of a small number of cosine-like activation functions.
(Tool: subsampling / probabilistic method.)

Step 3: Show that the cosine function can be approximated by sigmoid functions.
(Tool: classical approximation theory.)

Simple Infinite Neural Nets

Definition: An infinite-wide neural network is defined by a signed measure ν over neuron weights (w, b)

$$f(x) = \int_{w \in \mathbb{R}^d, b \in \mathbb{R}} \sigma(w^\top x + b) d\nu(w, b).$$

Theorem: Suppose $g : \mathbb{R} \to \mathbb{R}$ is differentiable, if $x \in [0,1]$, then $g(x) = \int_0^1 \mathbf{1}\{x \geq b\} \cdot g'(b) db + g(0)$

Step 1: Infinite Neural Nets

The function can be written as

$$f(x) = f(0) + \int_{\mathbb{R}^d} |\hat{f}(w)| (\cos(b_w + \langle w, x \rangle) - \cos(b_w)) dw.$$

Step 2: Subsampling

Writing the function as the expectation of a random variable:

$$f(x) = f(0) + \int_{\mathbb{R}^d} \frac{|\hat{f}(w)|\|w\|_2}{C} \left(\frac{C}{\|w\|_2} (\cos(b_w + \langle w, x \rangle) - \cos(b_w)) \right) dw.$$

Sample one $w \in \mathbb{R}^d$ with probability $\frac{|\hat{f}(w)|\|w\|_2}{C}$ for r times.

Step 3: Approximating the Cosines

Lemma: Given $g_w(x) = \frac{C}{\|w\|_2}(\cos(b_w + \langle w, x \rangle) - \cos(b_w))$, there exists a 2-layer neural network f_0 of size $O(1/\epsilon)$ with sigmoid activations, such that $\sup_{x \in [-1,1]} |f_0(y) - h_w(y)| \leq \epsilon$.

Depth Separation

So far we only talk about 2-layer or 3-layer neural networks.

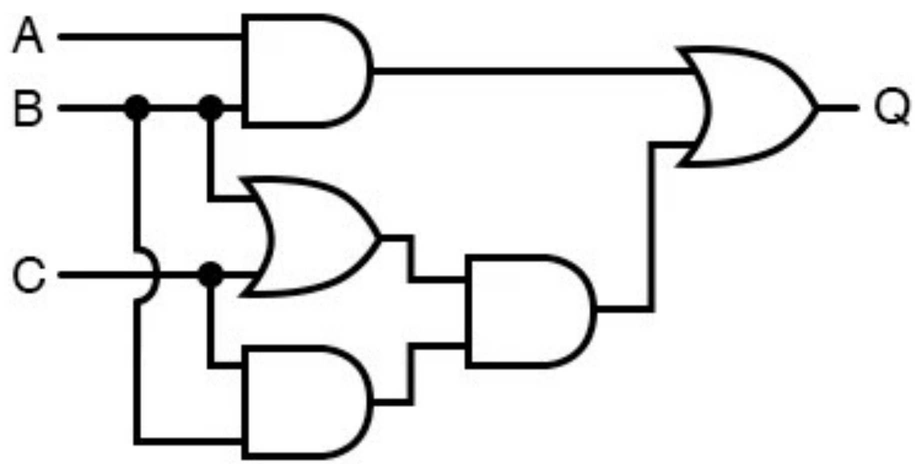
Why we need **Deep** learning?

Can we show deep neural networks are **strictly** better than shallow neural networks?

A brief history of depth separation

Early results from theoretical computer science

Boolean circuits: a directed acyclic graph model for computation over binary inputs; each node ("gate") performs an operation (e.g. OR, AND, NOT) on the inputs from its predecessors.



A brief history of depth separation

Early results from theoretical computer science

Boolean circuits: a directed acyclic graph model for computation over binary inputs; each node ("gate") performs an operation (e.g. OR, AND, NOT) on the inputs from its predecessors.

Depth separation: the difference of the computation power: shallow vs deep Boolean circuits.

Håstad ('86): parity function cannot be approximated by a small constant-depth circuit with OR and AND gates.

Modern depth-separation in neural networks

- **Related architectures / models of computation**
  - Sum-product networks [Bengio, Delalleau ’11]
- **Heuristic measures of complexity**
  - Bound of number of linear regions for ReLU networks [Montufar, Pascanu, Cho, Bengio ‘14]
- **Approximation error**
  - A small deep network cannot be approximated by a small shallow network [Telgarsky ’15]

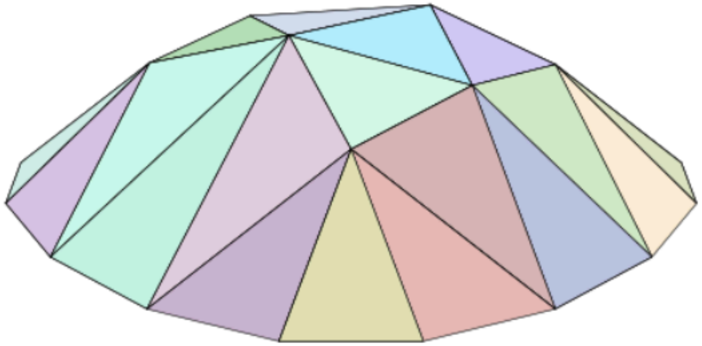
Shallow Nets Cannot Approximate Deep Nets

Theorem (Telgarsky '15): For every $L \in \mathbb{N}$, there exists a function $f : [0,1] \to [0,1]$ representable as a network of depth $O(L^2)$, with $O(L^2)$ nodes, and ReLU activation such that, for every network $g : [0,1] \to \mathbb{R}$ of depth L and $\leq 2^L$ nodes, and ReLU activation, we have

$$\int_{[0,1]} |f(x) - g(x)|\, dx \geq \frac{1}{32}.$$

Intuition

A ReLU network f is piecewise linear, we can subdivide domain into a finite number of polyhedral pieces $(P_1, P_2, \ldots, P_N)$ such that in each piece, f is linear: $\forall x \in P_i, f(x) = A_i x + b_i$.



Deeper neural networks can make exponentially more regions than shallow neural networks.

Make each region has different values, so shallow neural networks cannot approximate.

Benefits of depth for smooth functions

Theorem (Yarotsky '15): Suppose $f : [0,1]^d \to \mathbb{R}$ has all partial derivatives of order r with coordinate-wise bound in $[-1,1]$, and let $\epsilon > 0$ be given. Then there exists a $O(\ln \frac{1}{\epsilon})$ - depth and $\left(\frac{1}{\epsilon}\right)^{O(\frac{d}{r})}$ -size network so that $\sup_{x \in [0,1]^d} |f(x) - g(x)| \leq \epsilon$.

Remarks

- All results discussed are existential: they prove that a good approximator exists. Finding one efficiently (e.g., using gradient descent) is the next topic (optimization).
- The choices of non-linearity are usually very flexible: most results we saw can be re-proven using different non-linearities.
- There are other approximation error results: e.g., deep and narrow networks are universal approximators.
- Depth separation for optimization and generalization is widely open.

Recent Advances in Representation Power

- Analyses of different architectures
  - Graph neural network
  - Attention-based neural network
- Separation between transformers and RNNs (especially for programming tasks)
- Finite data approximation
- In-context learning for specific tasks
- Chain-of-thought
- …