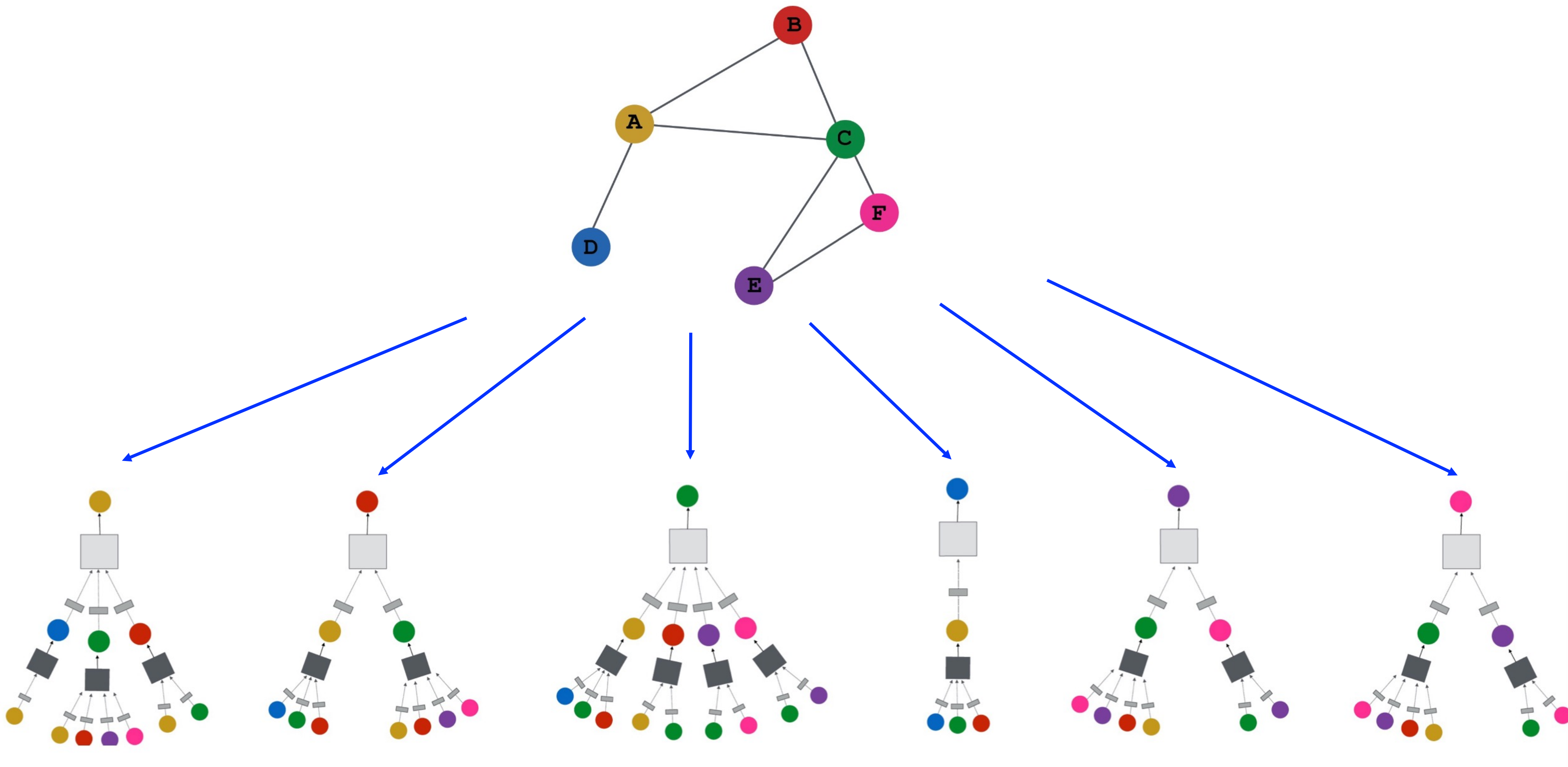
RNN vs GNN

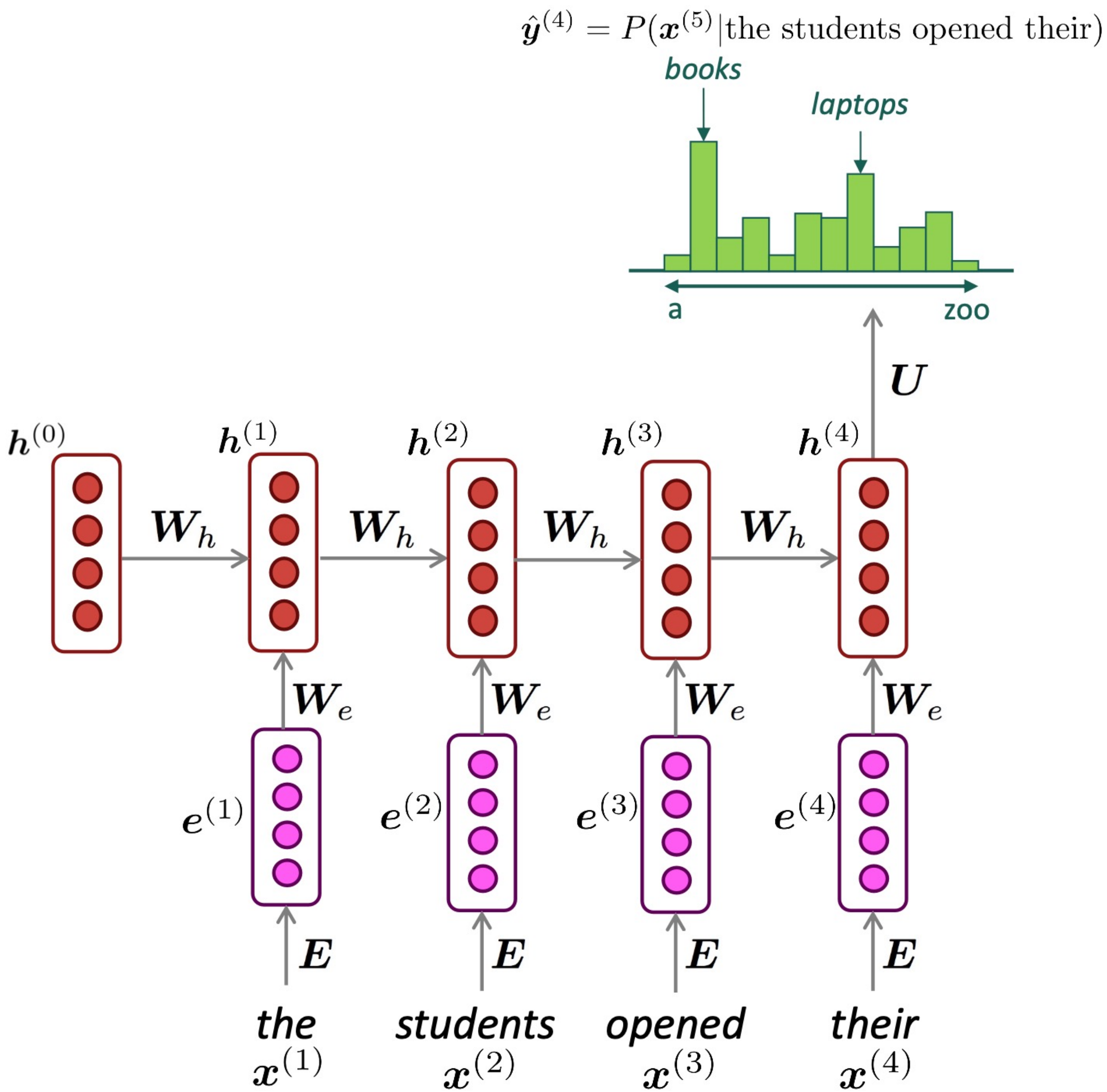
$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)}\sum_{v\in\mathcal{N}(u)}\mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

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

Graph convolutional network



RNN language model



output distribution

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

hidden states

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

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

word embeddings

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

words / one-hot vectors

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

RNN with attention

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

Weighted sum of encoder hidden states based on the attention distribution

Softmax

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

Dot product

$$e^t = [\underbrace{s_t^T h_1}, \underbrace{\ldots}, \underbrace{s_t^T h_N}] \in \mathbb{R}^N$$

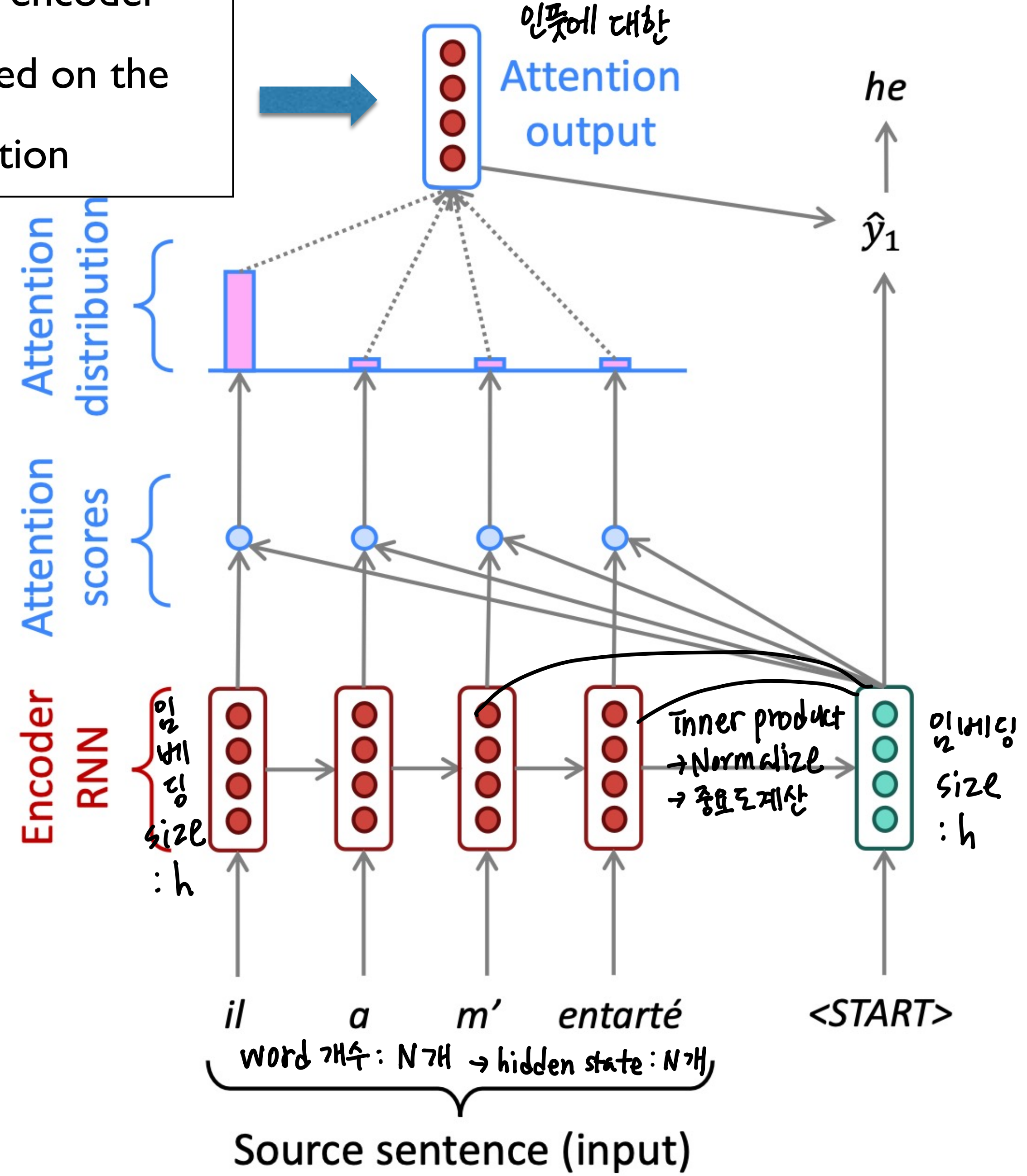
→ vector size : 1×1

Encoder hidden states

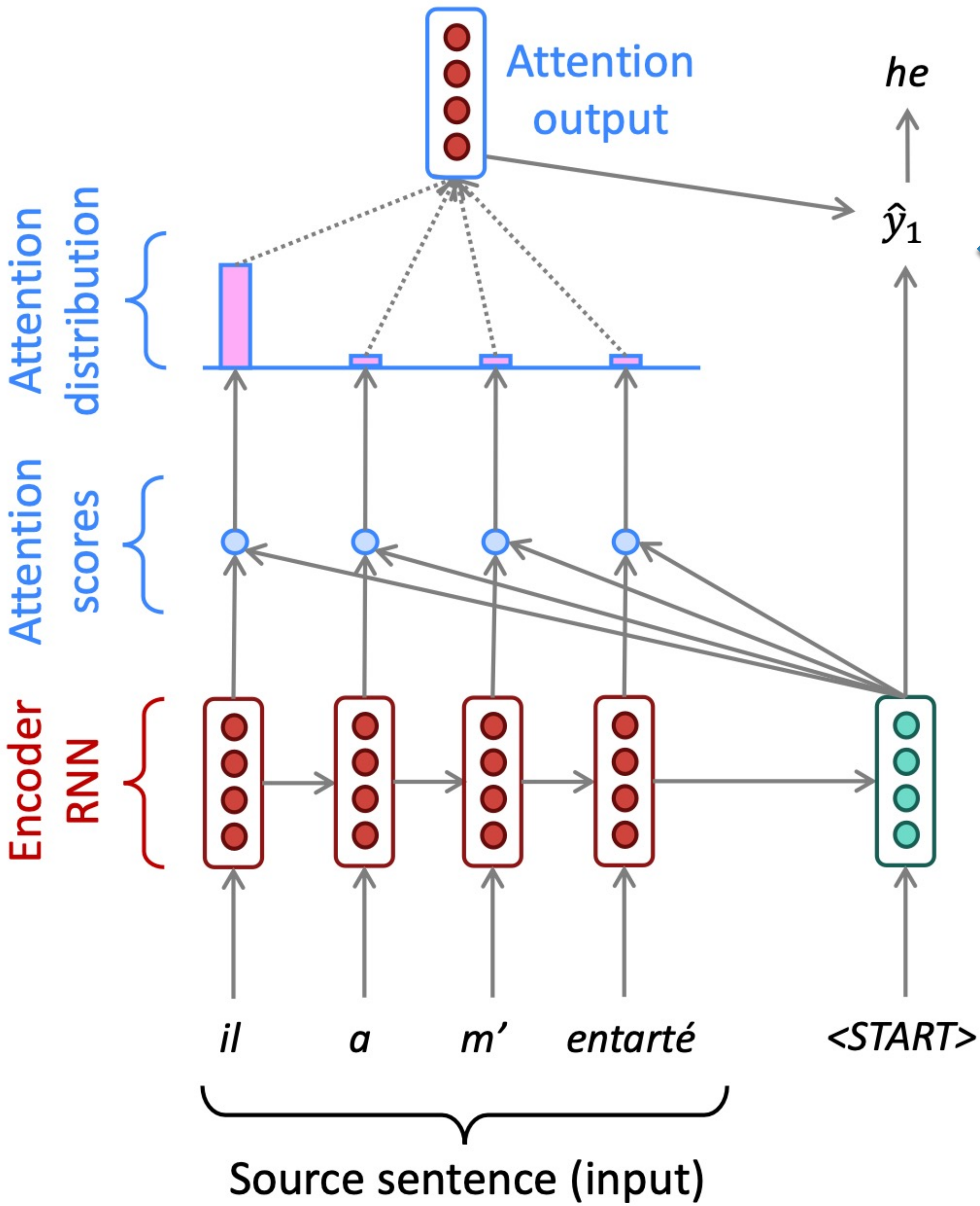
$$h_1, \ldots, h_N \in \mathbb{R}^h$$

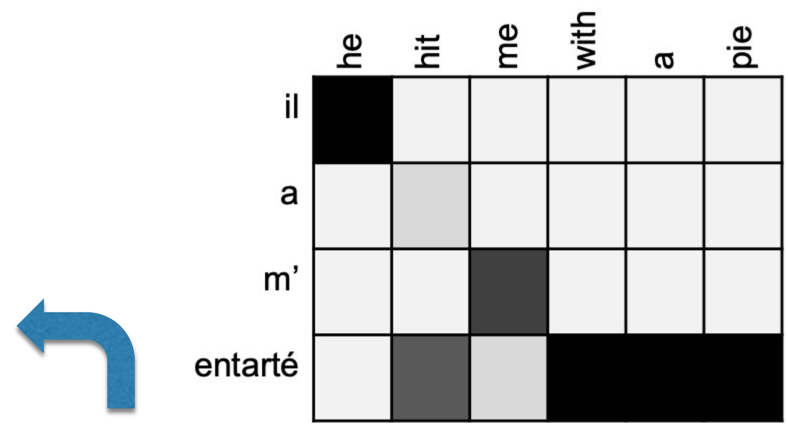
Decoder hidden states

$$s_t \in \mathbb{R}^h$$



RNN with attention





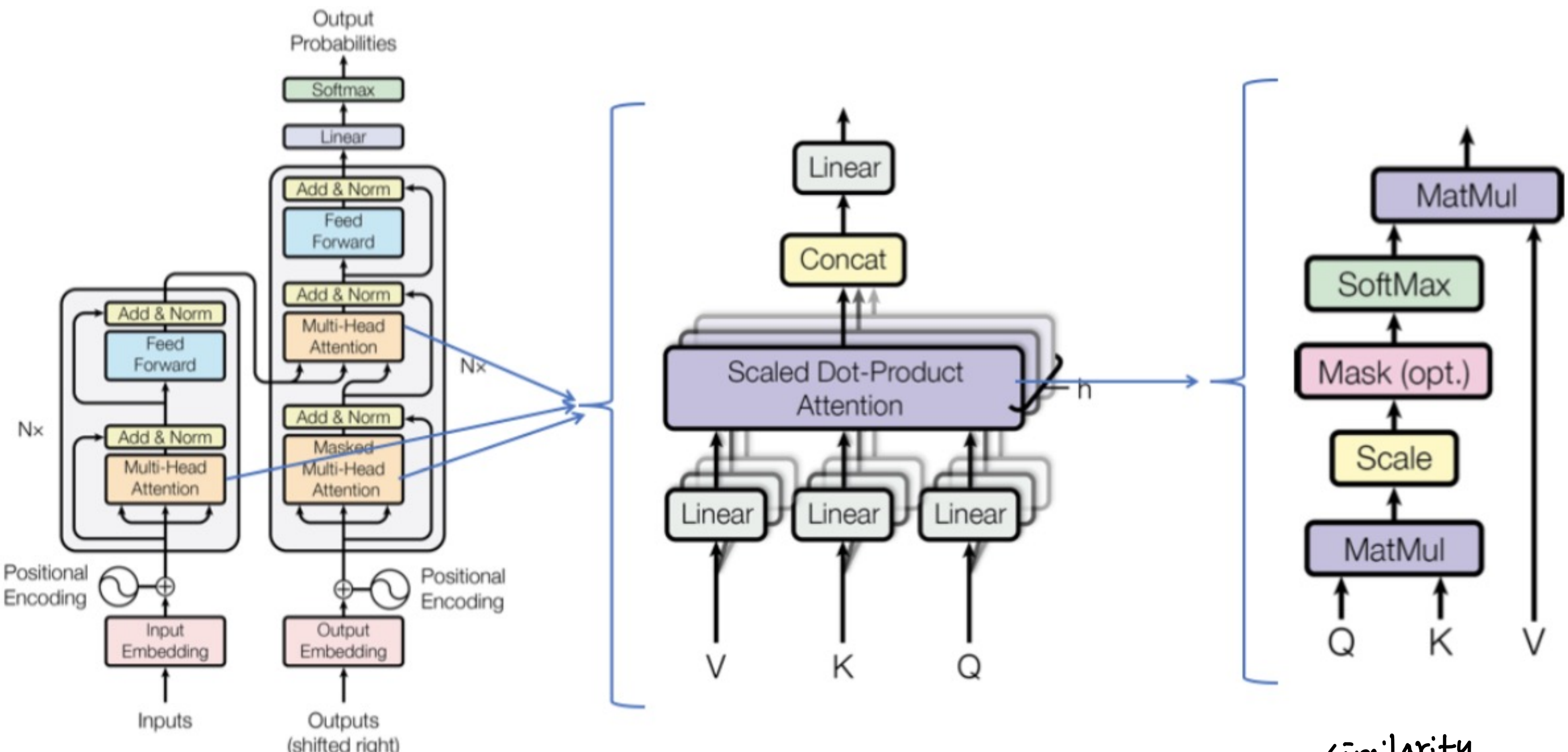
Concatenate attention output and decoder hidden state to compute $\hat{y}_1$

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

RNN with attention

- Attention allows the model to focus on the relevant parts of the input sequence as needed
- Self-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence

Self-attention



similarity

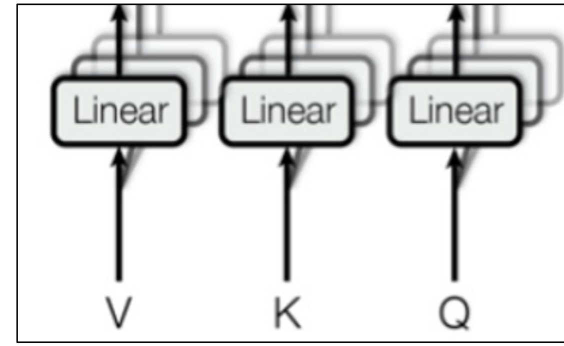
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

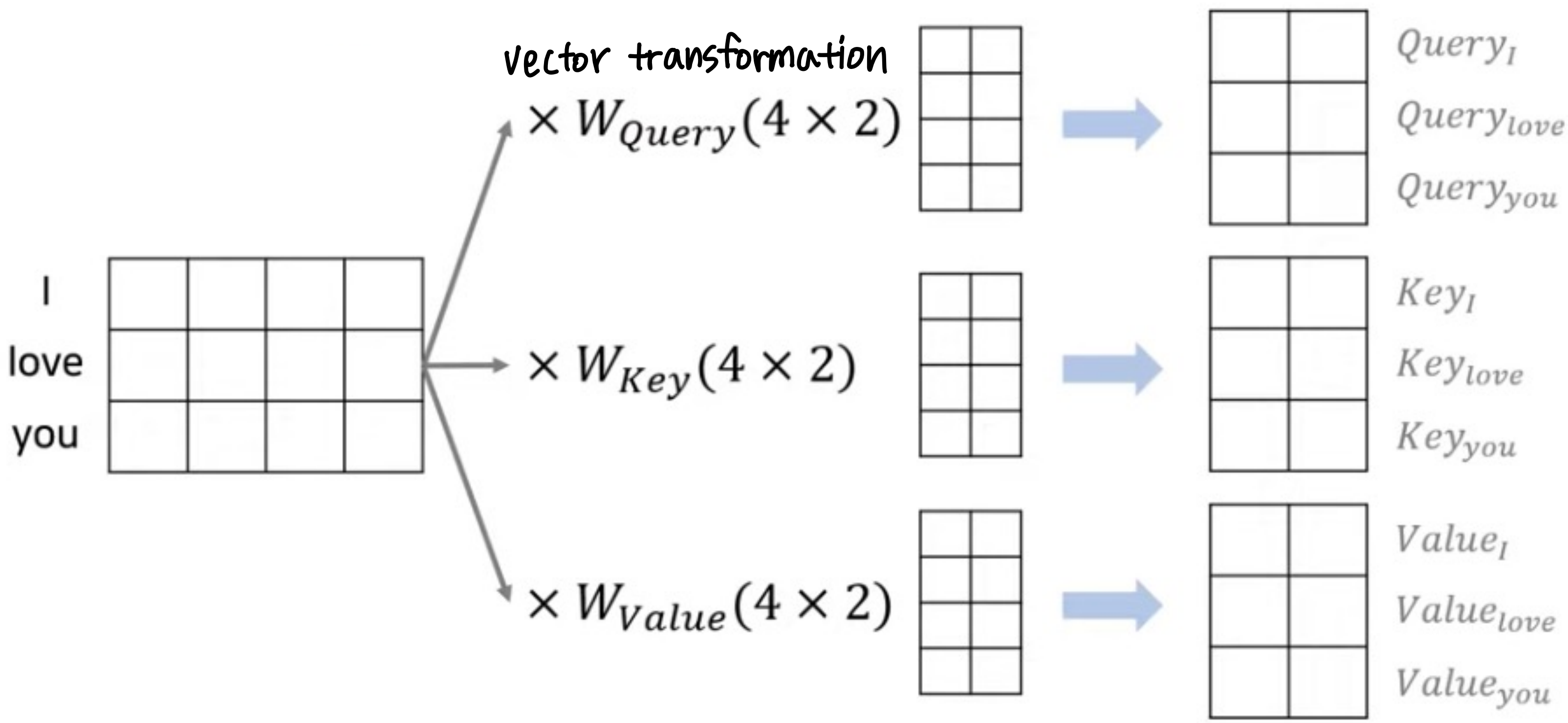
weight계산

https://arxiv.org/abs/1706.03762

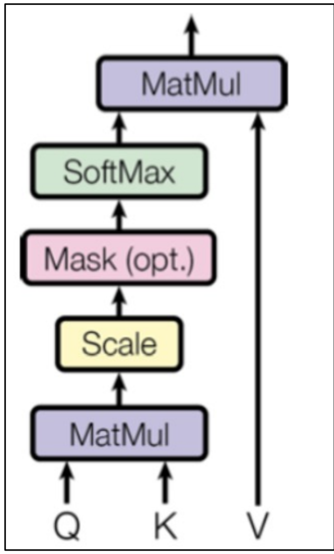
Self-attention

- Transform the embedding to the embedding
  for query, key, and value





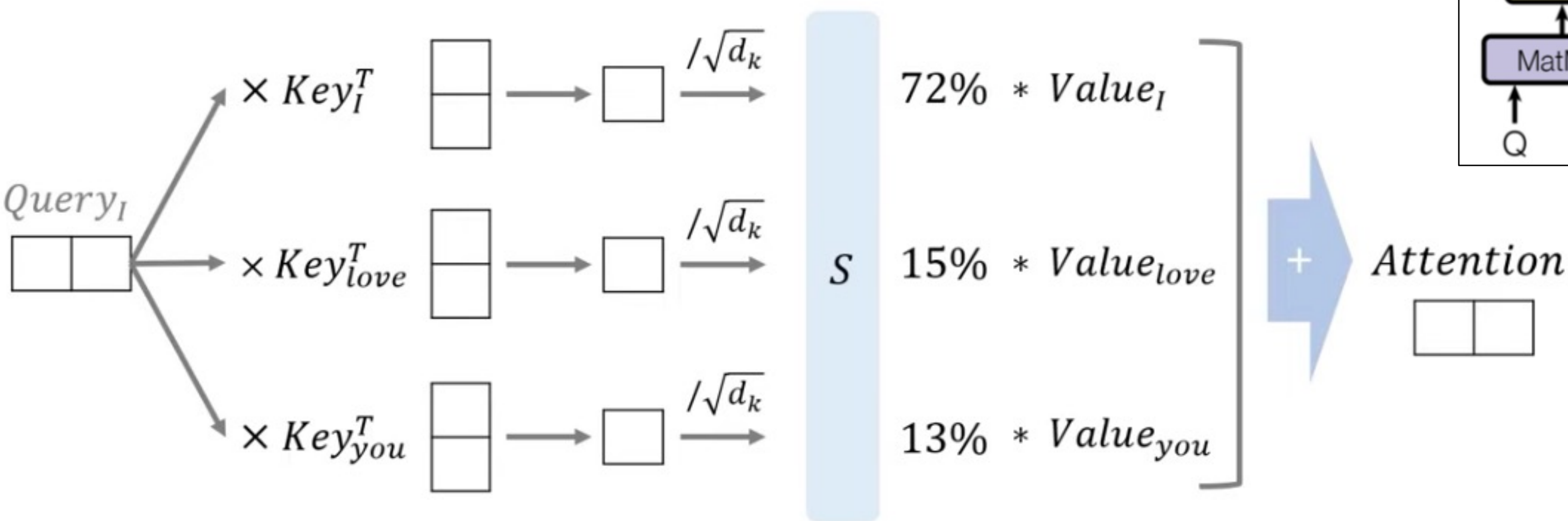
Self-attention



element by element similarity

- $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

Normalization



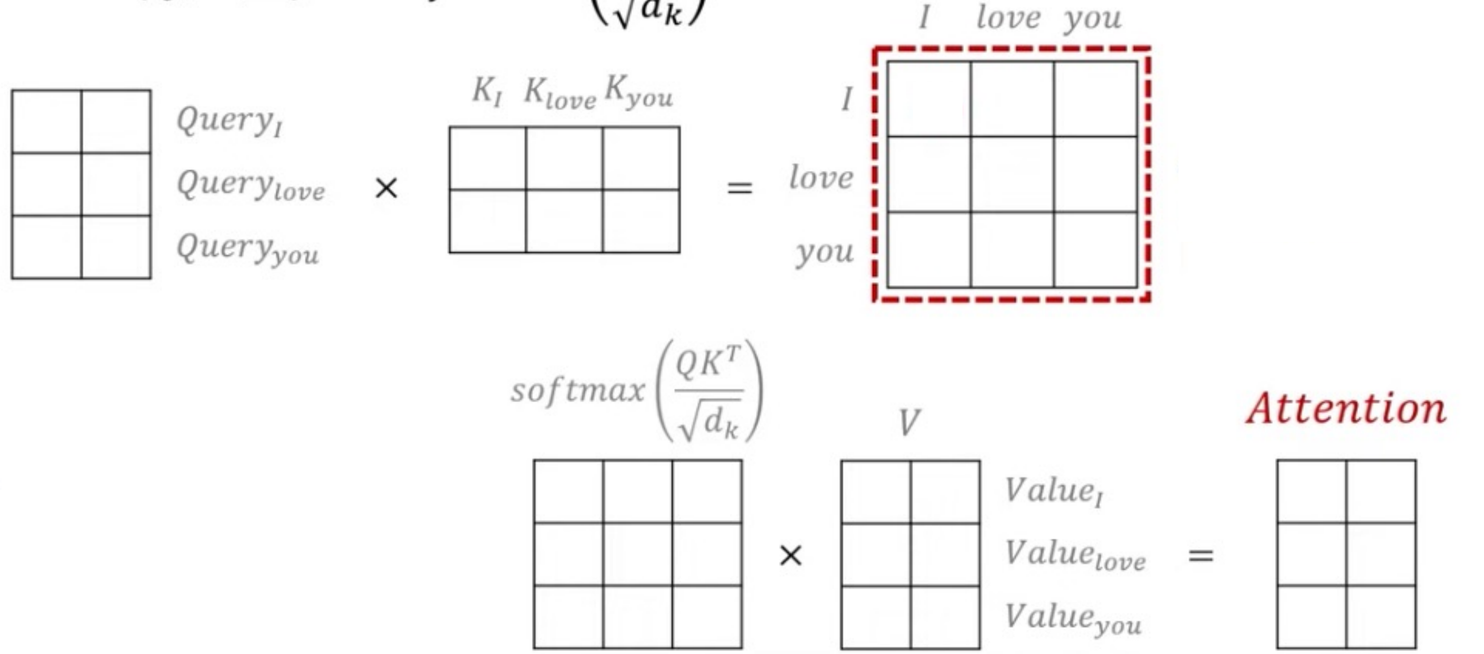
$$e_{ij} = q_i^{\top} k_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})} \qquad \text{output}_i = \sum_j \alpha_{ij} v_j$$

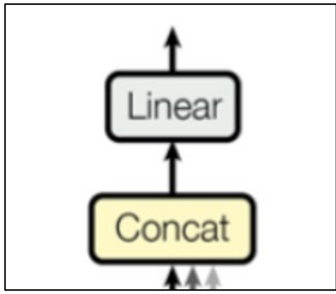
Normalize → coefficient

Self-attention

- $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

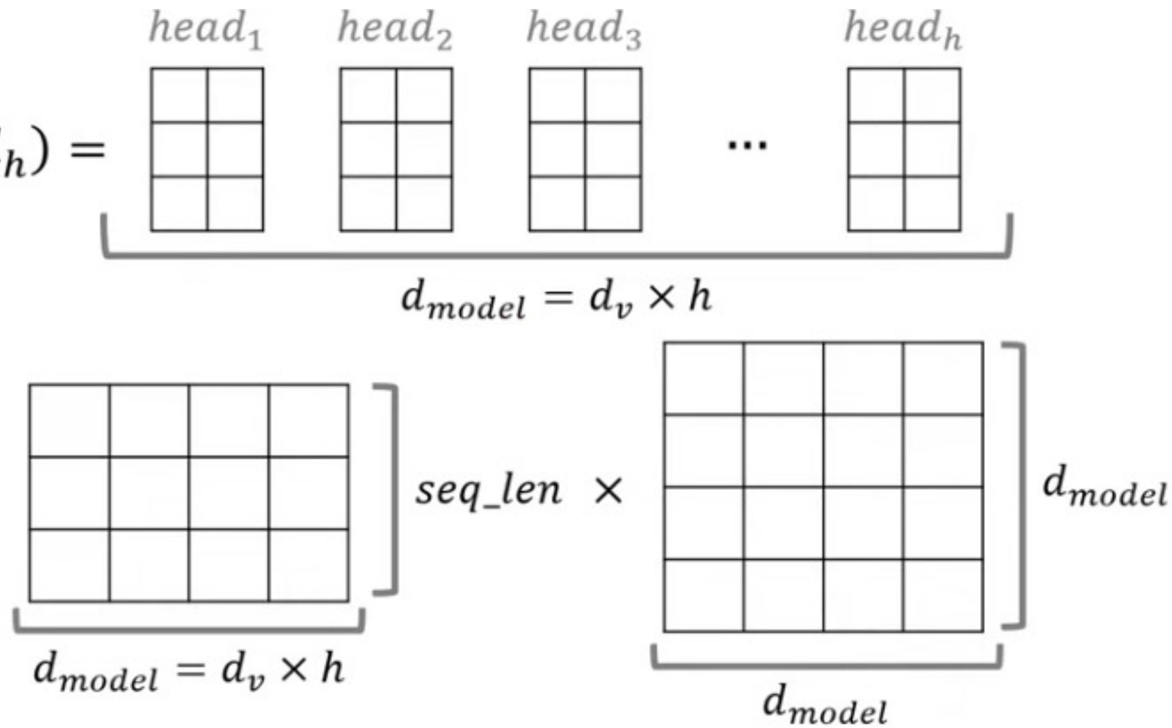


Self-attention

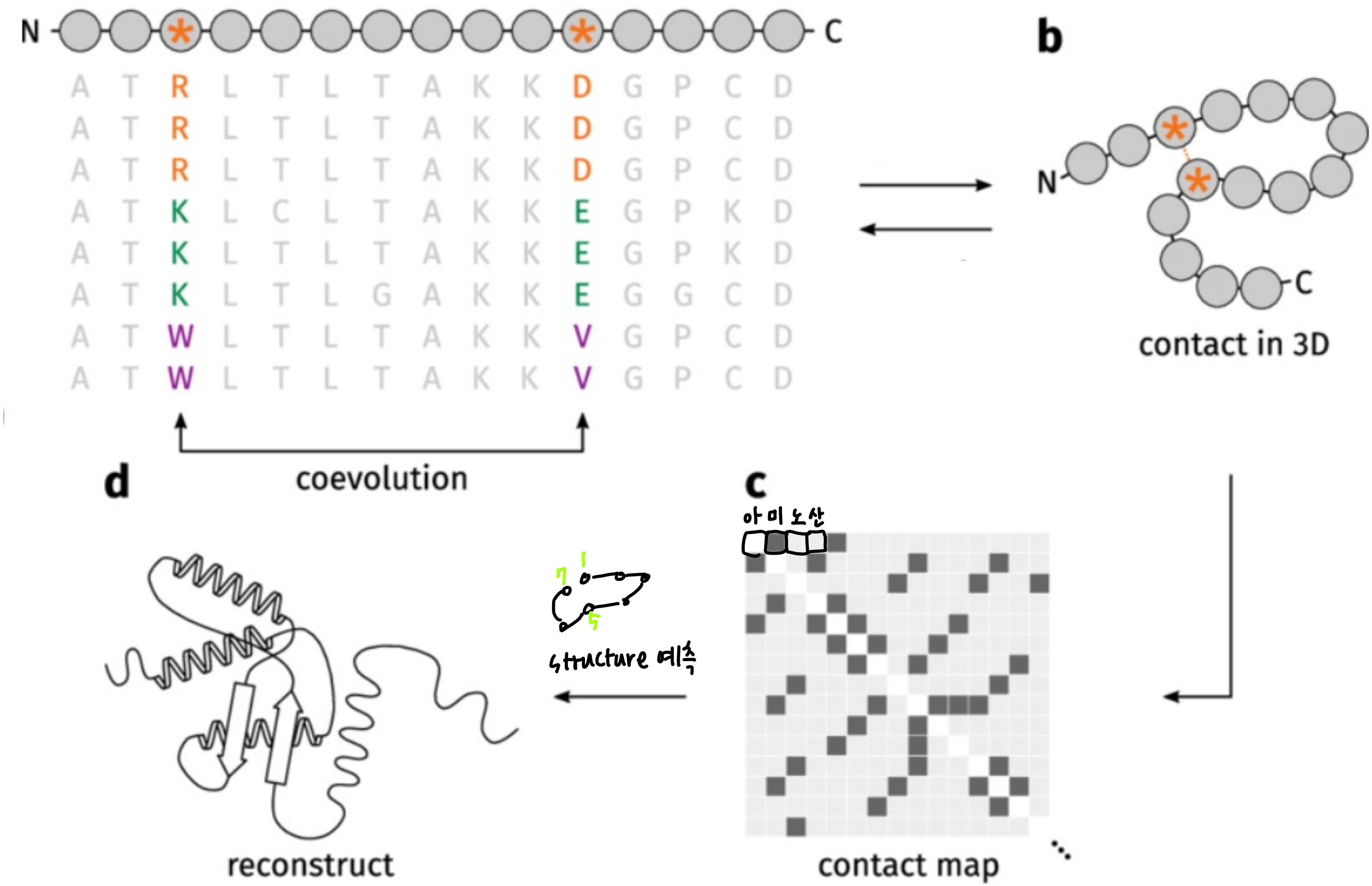


$Concat(head_1, \ldots, head_h) =$

$MultiHead(Q, K, V) =$



Self-attention applied to genomic sequences



https://www.nature.com/articles/s41586-021-03819-2

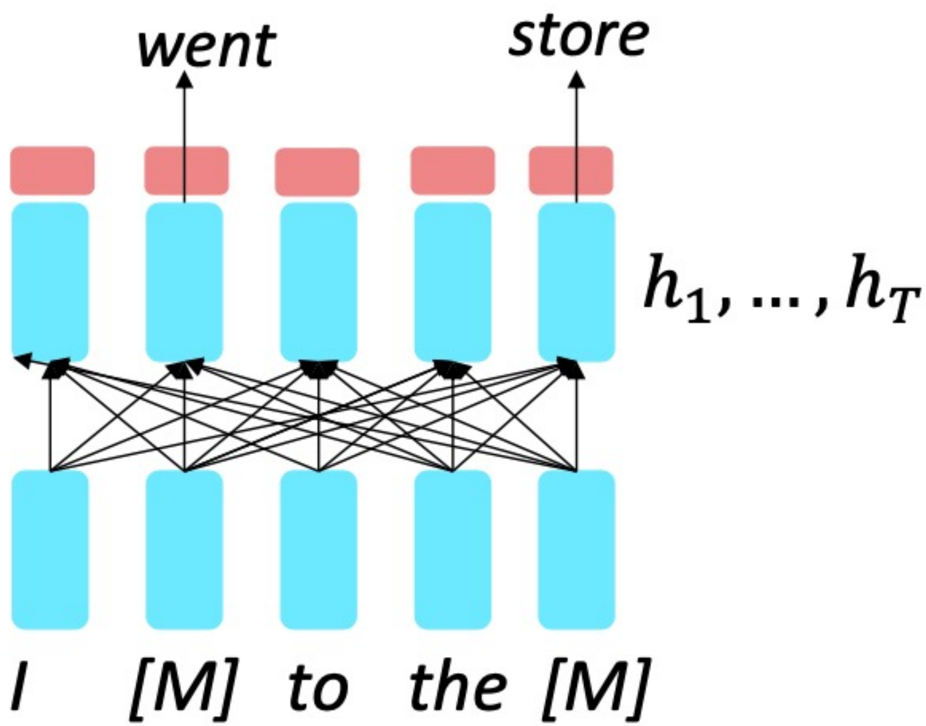
Masked language model

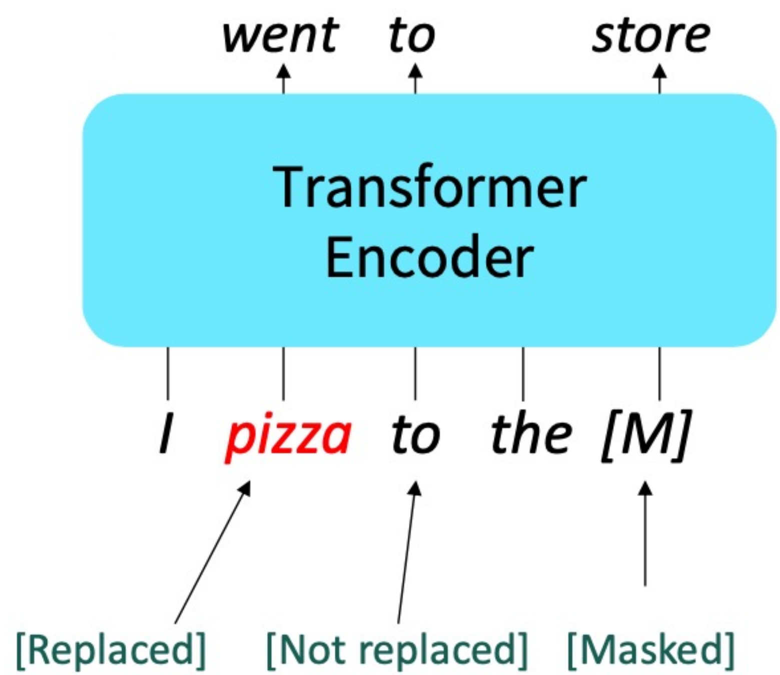
(Bert)

Self-supervised learning
→ 가장 큰 목적 : 레이블된 데이터가 적어도
Input으로 좋은 representation 만들기

$\tilde{x}$ is the masked version of x, we're learning $p(x|\tilde{x})$

- replace some fraction of words in the input with a special [MASK] token
- predict these words.

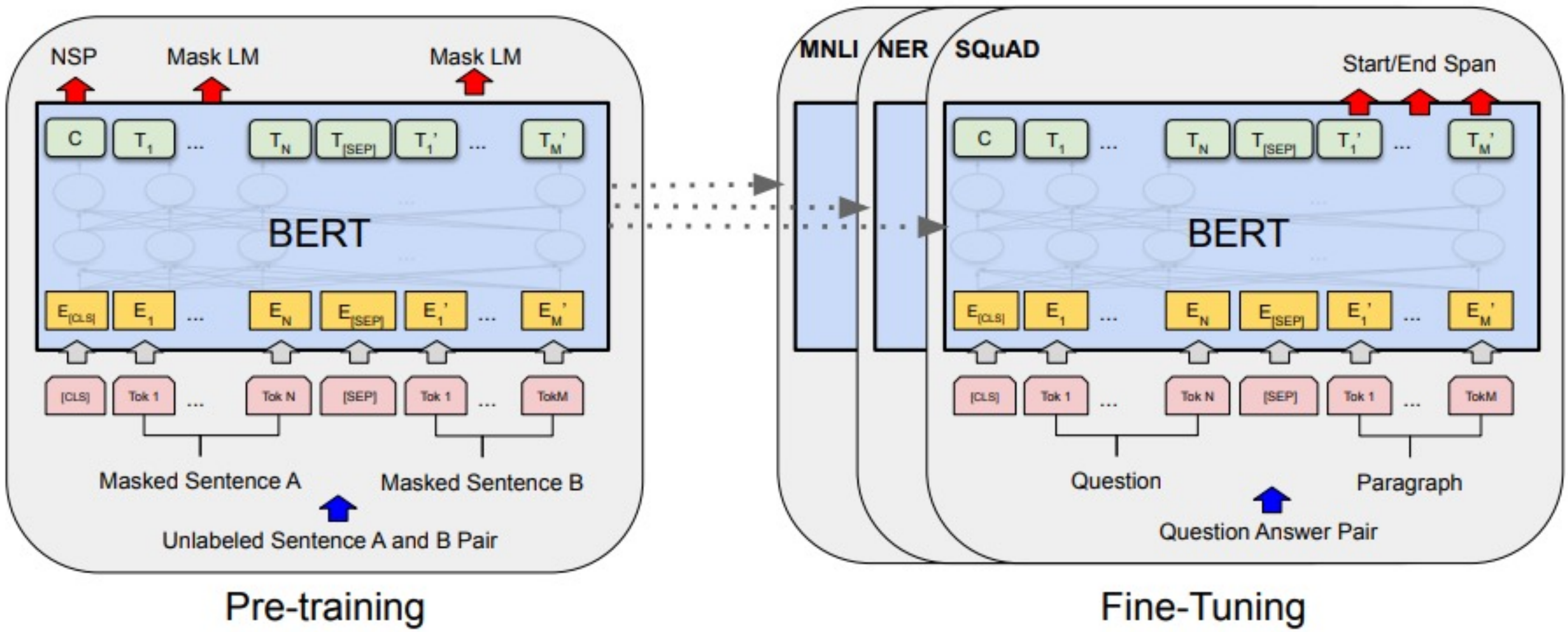




https://arxiv.org/pdf/1810.04805.pdf

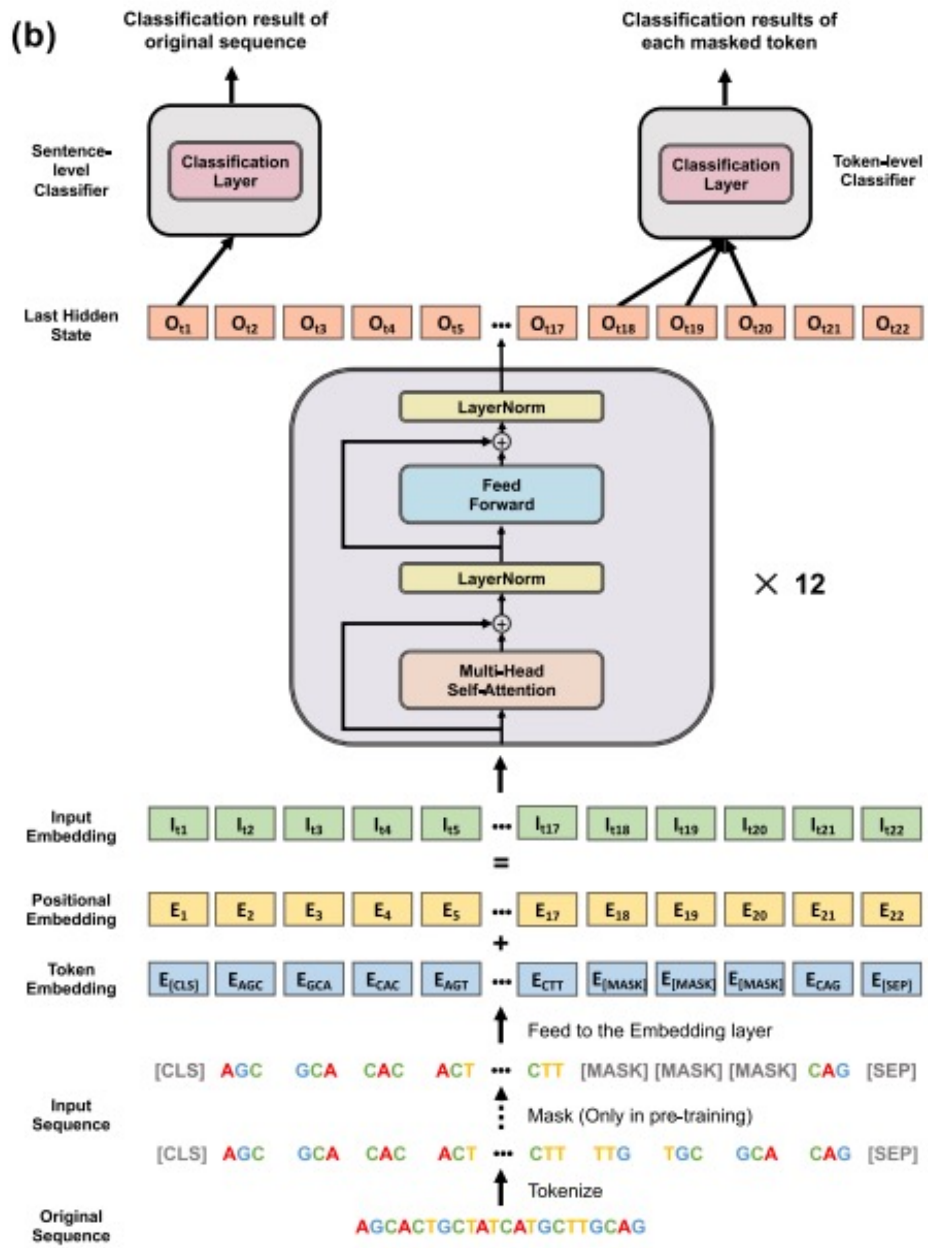
Bidirectional Encoder Representation

Input: word peice



- Apart from output layers, the same architectures are used in both pre-training and fine-tuning
- The same pre-trained model parameters are used to initialize models for different down-stream tasks
- During fine-tuning, all parameters are fine-tuned.

BERT applied to genomic sequences



https://academic.oup.com/bioinformatics/article/37/15/2112/6128680