

DeepIntoDeep

Attention Mechanism (w/ Machine Translation)

발표자: 전성후

공지

- 프로그래밍 과제 1 : 오늘 자정까지 제출
- 프로그래밍 과제 2 : 공지 참조 (LSTM + Transformer Encoder (5강 강의 예정))

Attention Mechanism and Machine Translation

전성후

Artificial Intelligence in Korea University(AIKU)

Department of Computer Science and Engineering, Korea University

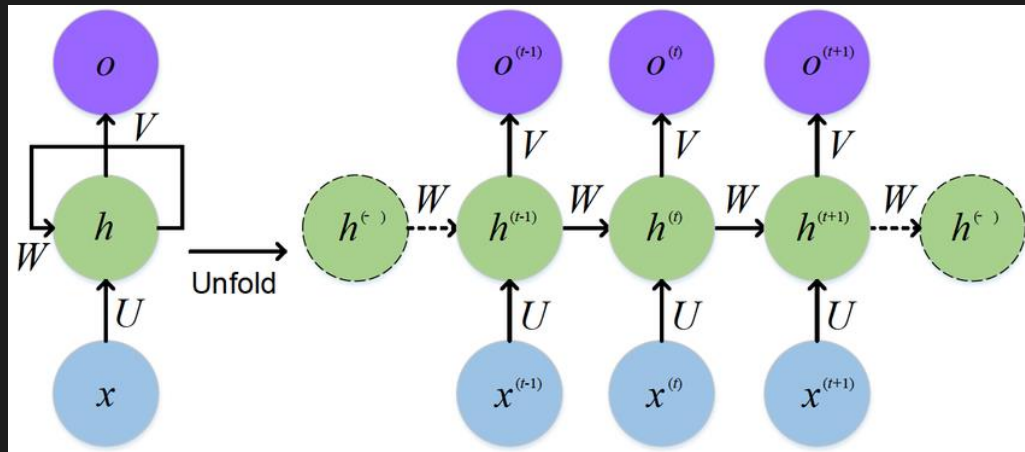
목차

- (Recap) Recurrent Neural Networks
- Encoder-Decoder Architecture
- Neural Machine Translation
 - Seq2Seq
 - Decoding
 - Sampling
- Attention
 - Self-attention
 - Solving NMT with Seq2Seq + attention
 - RNN vs. Attention
 - Permutation Invariance

강의자 소개

- 이름 - 전성후
- 경력
 - 고려대학교 정보대학 컴퓨터학과 22학번
 - AIKU 0기 (22.07.~), 3기 학회장
 - CVLAB 학부 인턴 (23.12.~)
 - GDSC KU AI Core (22.09.~)
 - 2023 정보대학 주최 데이터톤 “iNThon” 대상
- 관심분야
 - Representation Learning
 - Computer Vision
 - Pixel Tracking
 - Video Diffusion

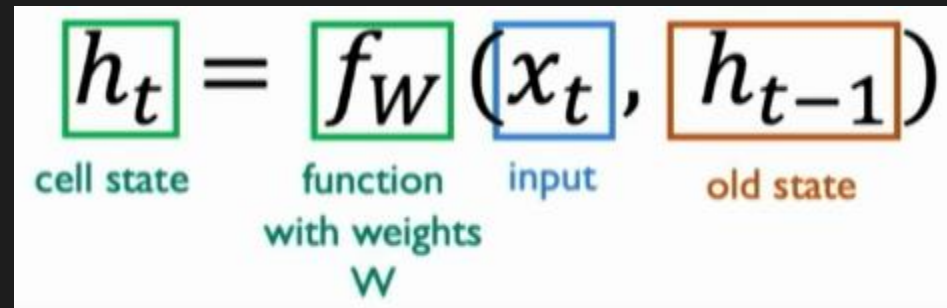
(Recap) Recurrent Neural Networks



- How can we handle “Sequential data”?
 - 언어, 소리, 주식, 악보 ...
 - 순서 정보가 중요하다
 - Input의 길이가 달라질 수 있다
 - Long-term과 short-term의 dependency를 모두 고려할 수 있어야 한다.
- RNNs' Design Criteria
 - Handle variable-length sequences
 - Track long-term dependencies
 - Maintain information about order
 - Share parameters across the sequence

(Recap) Recurrent Neural Networks

- Apply a recurrence relation at every time step to process a sequence:

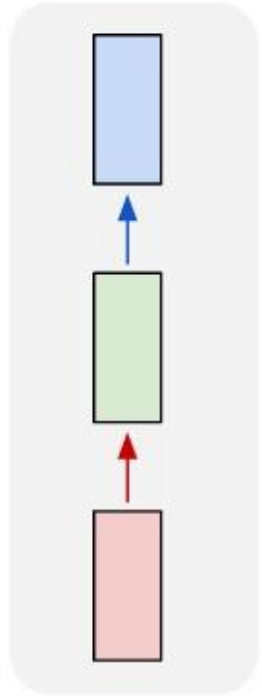

$$\boxed{h_t} = \boxed{f_W}(\boxed{x_t}, \boxed{h_{t-1}})$$

cell state function with weights W input old state

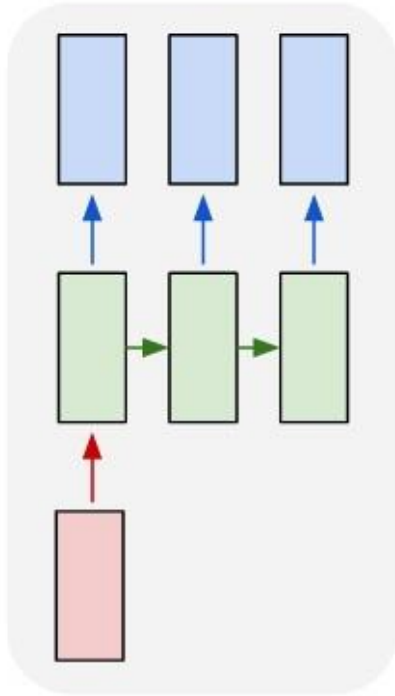
- Function and set of parameters are shared at every time step.
- RNNs have a state h_t , that is updated at each time step as a sequence is processed

(Recap) Recurrent Neural Networks

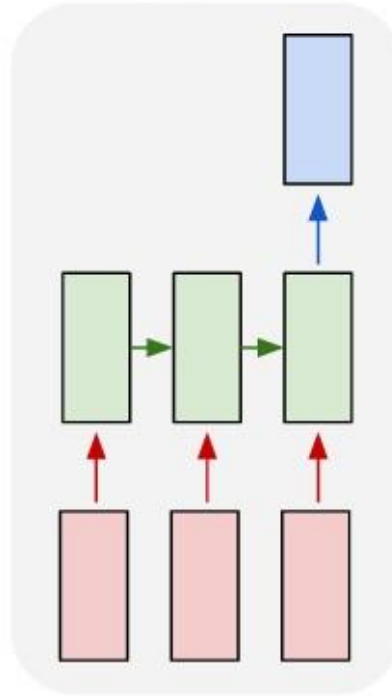
one to one



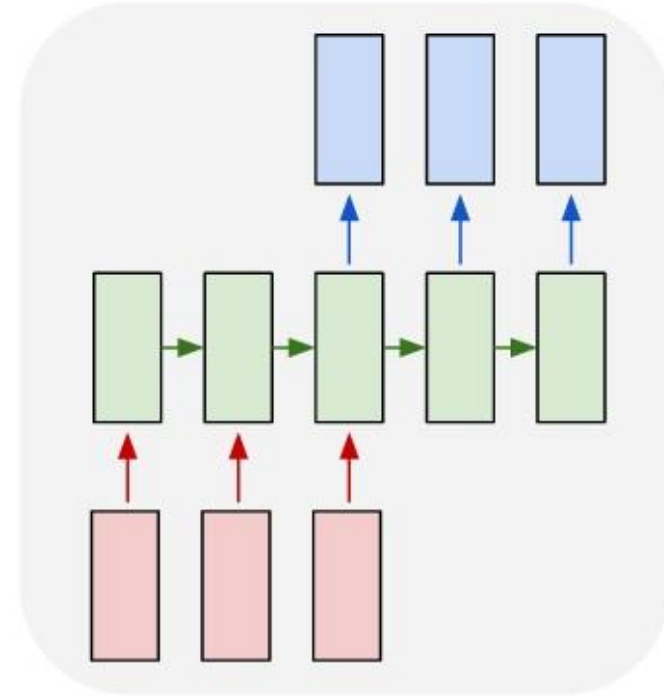
one to many



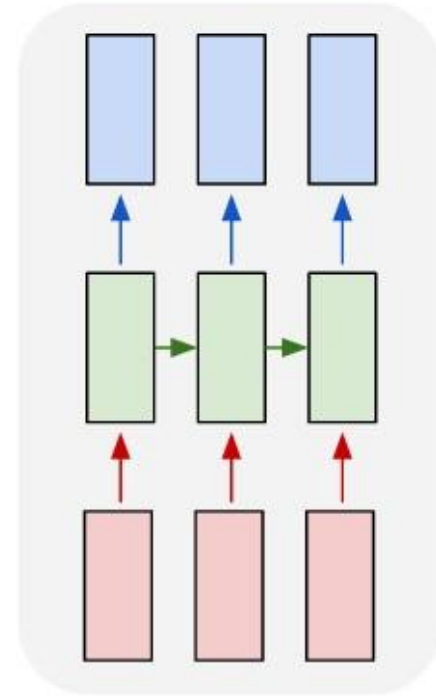
many to one



many to many

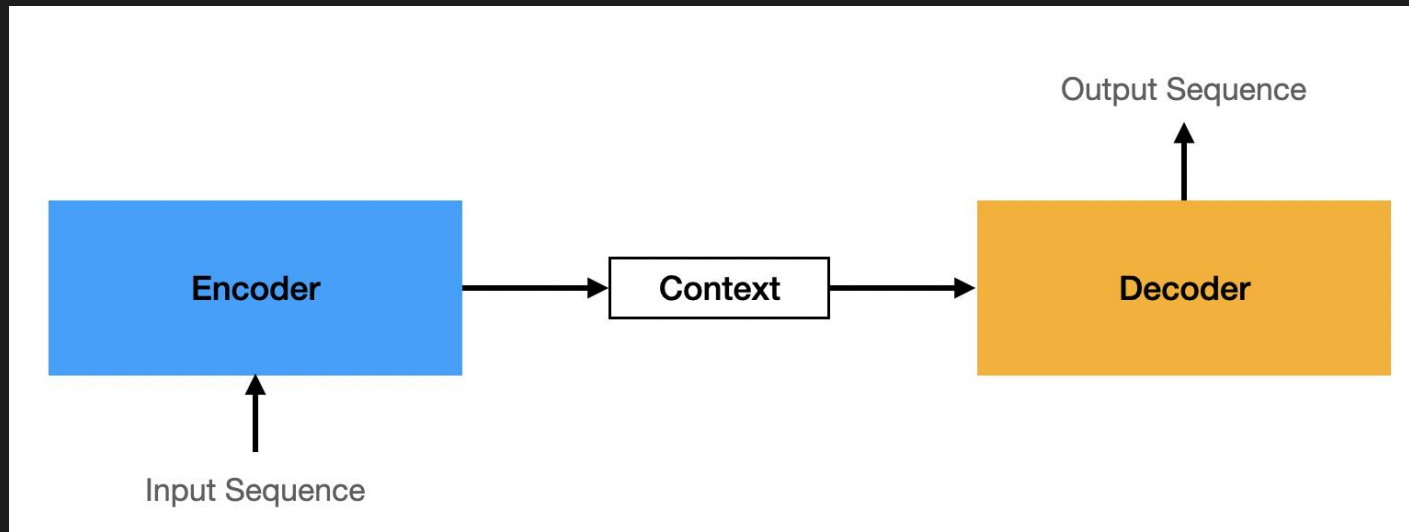


many to many



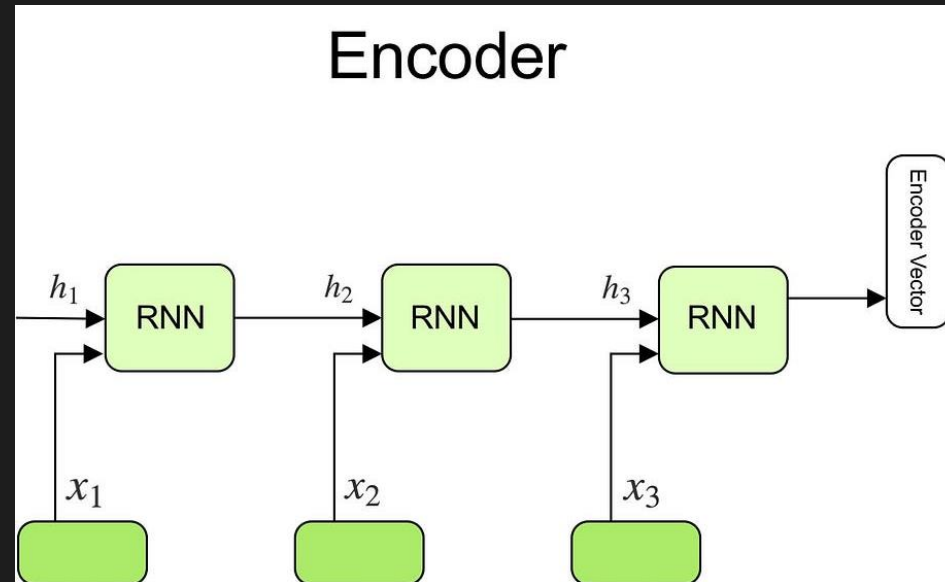
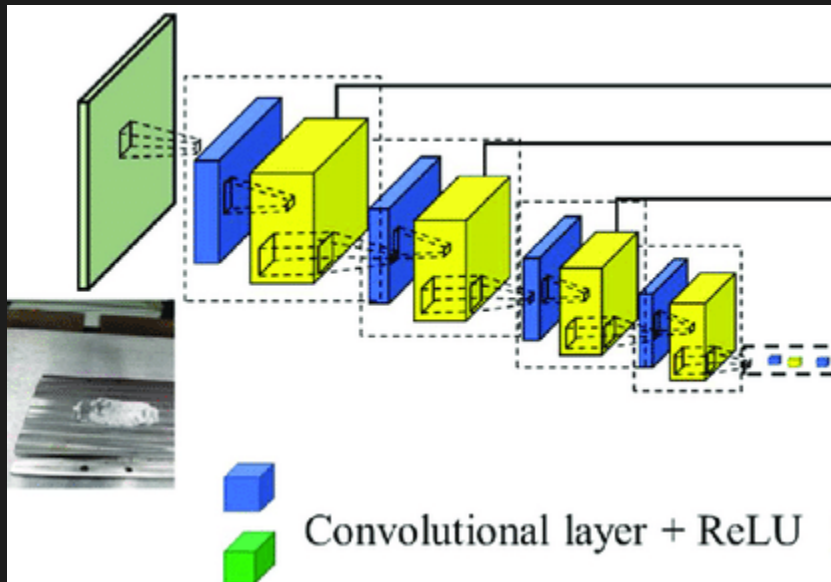
Encoder-Decoder Architecture

- General Model / Perspective of Architecture for Deep/Machine Learning
- **Encoder** : Input을 압축하여 context(feature, representation)으로 만든다.
- **Decoder** : context(feature, representation)을 다시 output으로 변환시킨다.



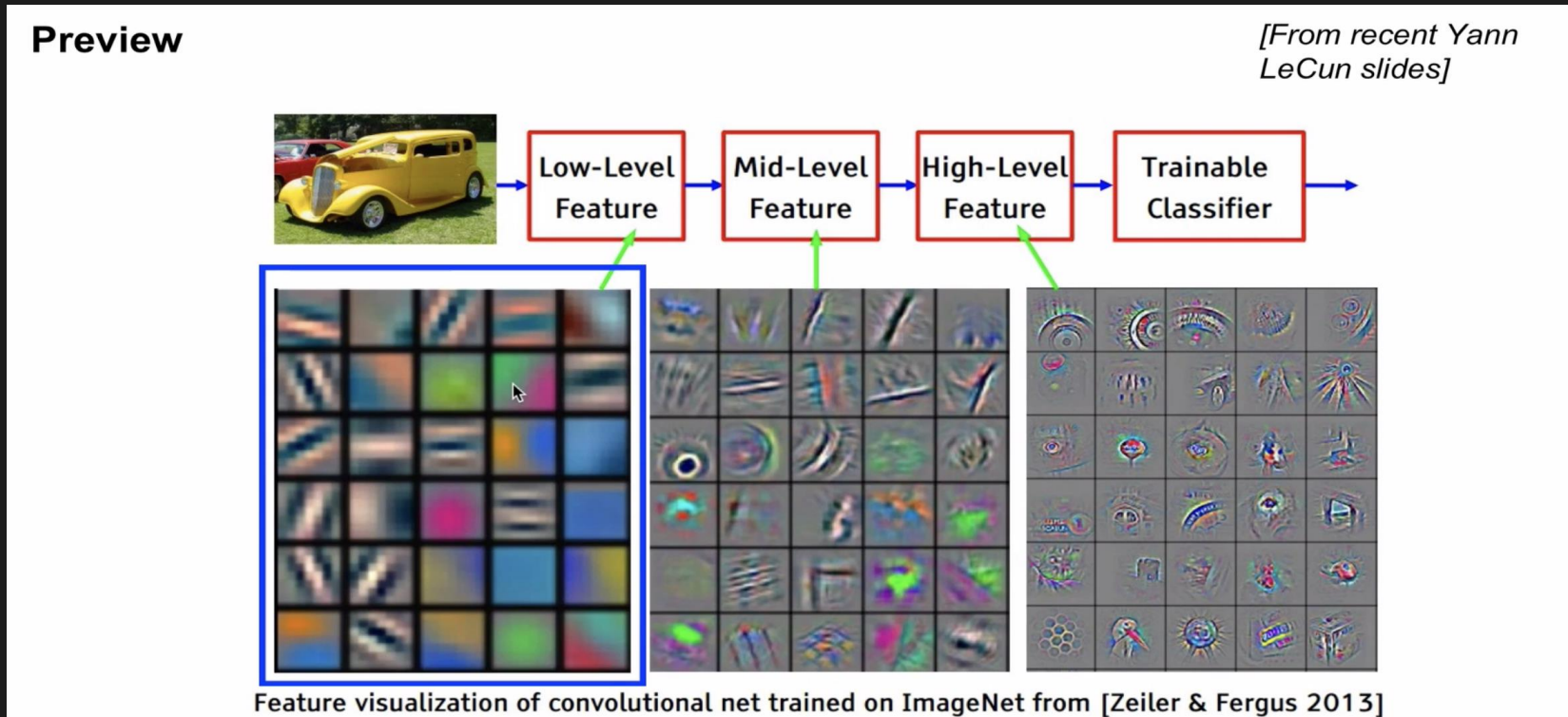
Encoder-Decoder Architecture

- **Encoder** : Input을 압축하여 context(feature, representation, embedding)으로 만든다.



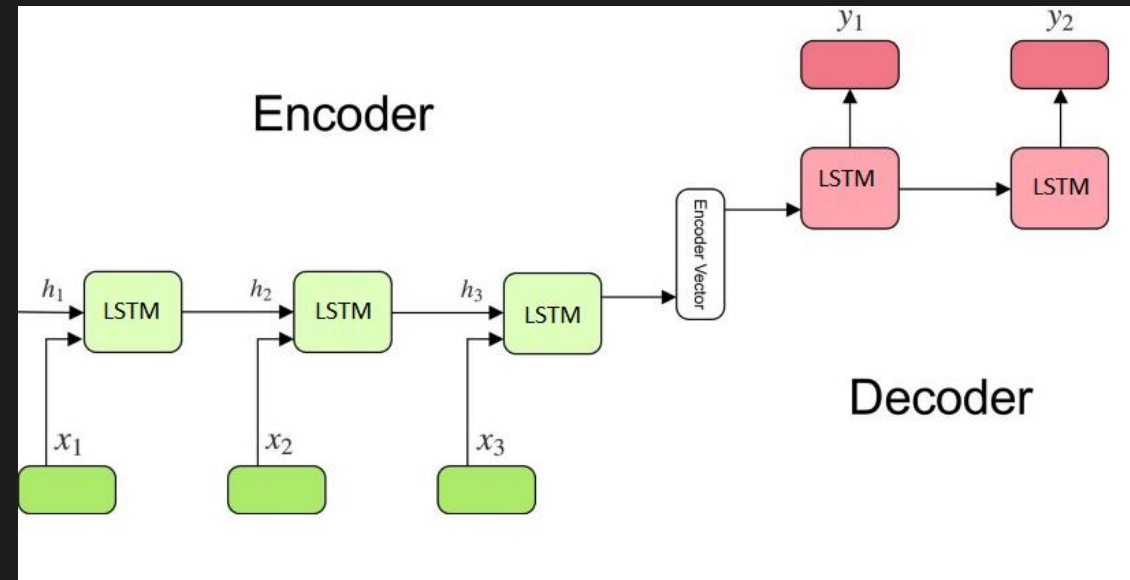
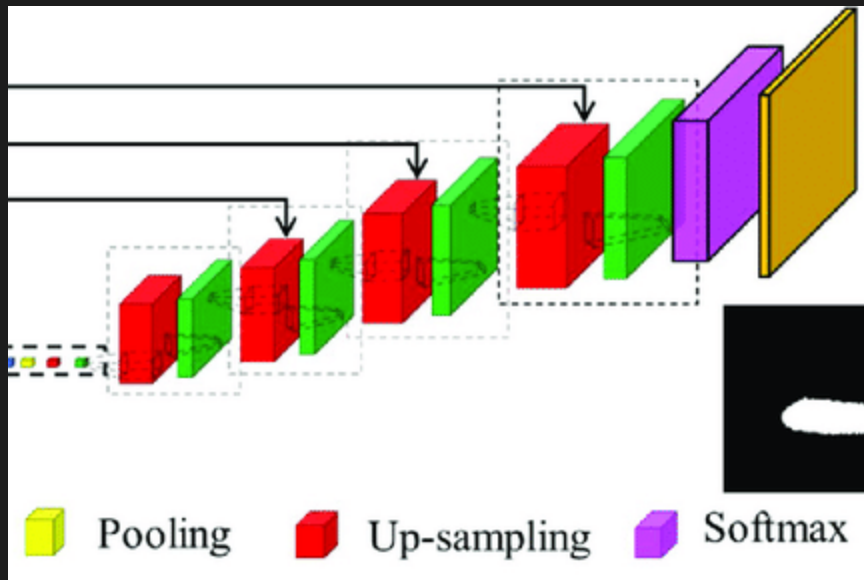
Encoder-Decoder Architecture

- **Encoder** : Input을 압축하여 context(feature, representation, embedding)으로 만든다.
 - **Hierarchical feature** : input의 정보를 압축하되, 정보의 손실을 최소화하는 방향으로 feature를 뽑아냄 -> 이전 layer의 정보를 결합하며, 자연스럽게 High-level feature를 출력



Encoder-Decoder Architecture

- **Decoder** : context(feature, representation, embedding)을 다시 output으로 변환한다.
 - Deconvolution(Transposed Convolution), Upsampling, LSTM Decoder, ...
 - Symmetric architecture is useful and neat, but they don't have to be

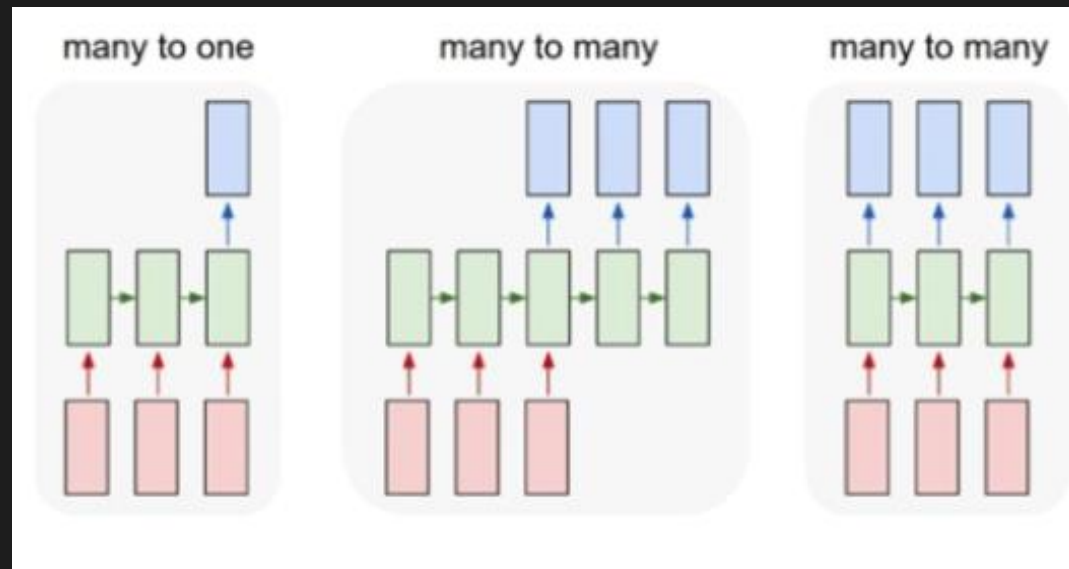


Encoder-Decoder Architecture

- Encoder-Decoder Architecture in NLP

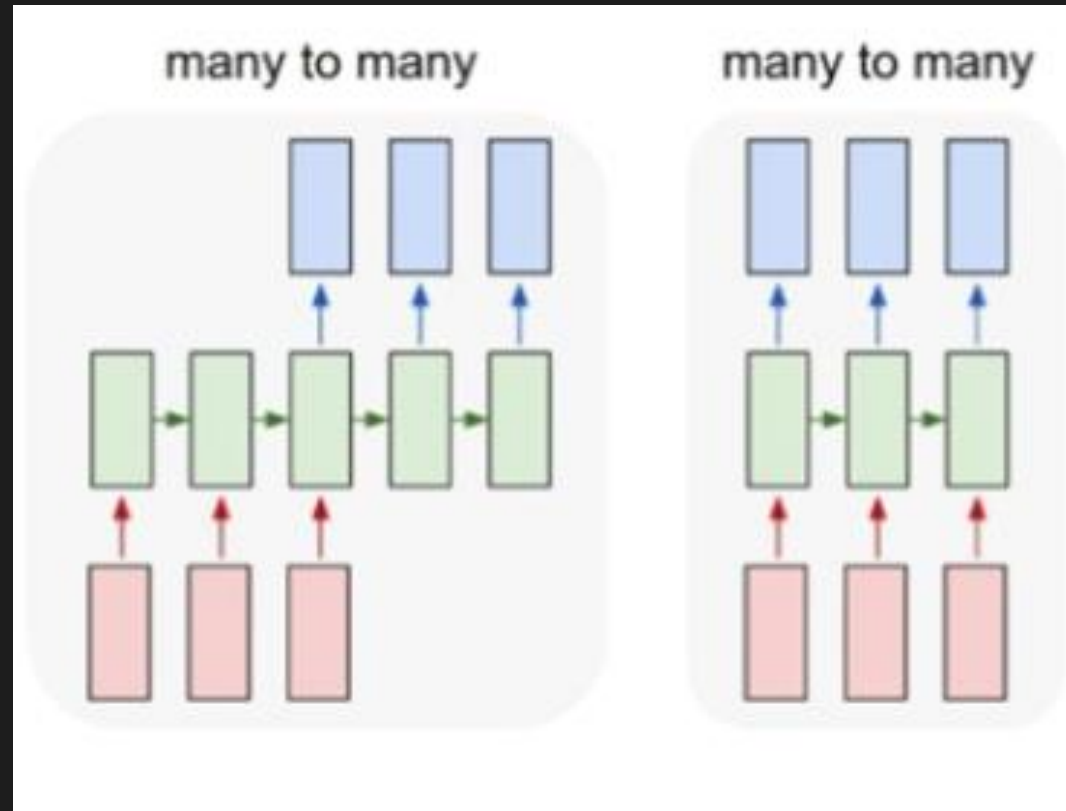
- Many-to-one

- Input으로 들어온 단어들의 정보를 계속 Context vector에 저장하다가 (encoding)
- 마지막 timestep의 context vector를 이용하여 하나의 output (e.g. class distribution)을 내뱉는다 (decoding)



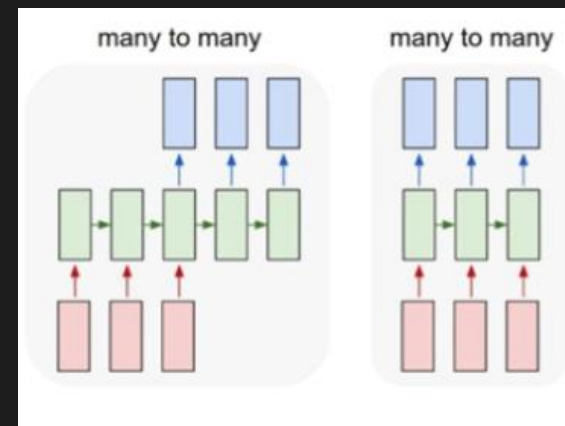
Encoder-Decoder Architecture

- 둘의 차이점은?



Encoder-Decoder Architecture

- 둘의 차이점은?
 - 왼쪽
 - Input으로 들어온 단어들의 정보를 **전부** Context vector에 저장하고 (**encoding**)
 - **마지막** timestep의 context vector를 이용하여 output을 **하나씩** 내뱉는다 (**decoding**)
 - 오른쪽
 - Input으로 들어온 단어들의 정보를 **그 timestep까지** Context vector에 저장하고 (**encoding**)
 - **해당** timestep의 context vector를 이용하여 output을 **하나씩** 내뱉는다 (**decoding**)
- 오른쪽 Model은 현재 이후의 timestep을 볼 수 없다.
- 왼쪽 모델은 전체 timestep을 볼 수 있다.
- 그렇다면, 항상 왼쪽 모델이 더 나은 성능을 낼 것인가?



Machine Translation

- Machine Translation(MT)

- source language로 표현되어 있는 문장 x 를 target language로 표현된 문장 y 로 번역하는 task

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



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

Statistical Machine Translation (90s - 10s)

- 예전에는 MT를 수행하기 위해 data를 기반으로 한 probabilistic model로 접근
- 주어진 French sentence x 에 대해서 가장 좋은 English sentence y 를 찾자!

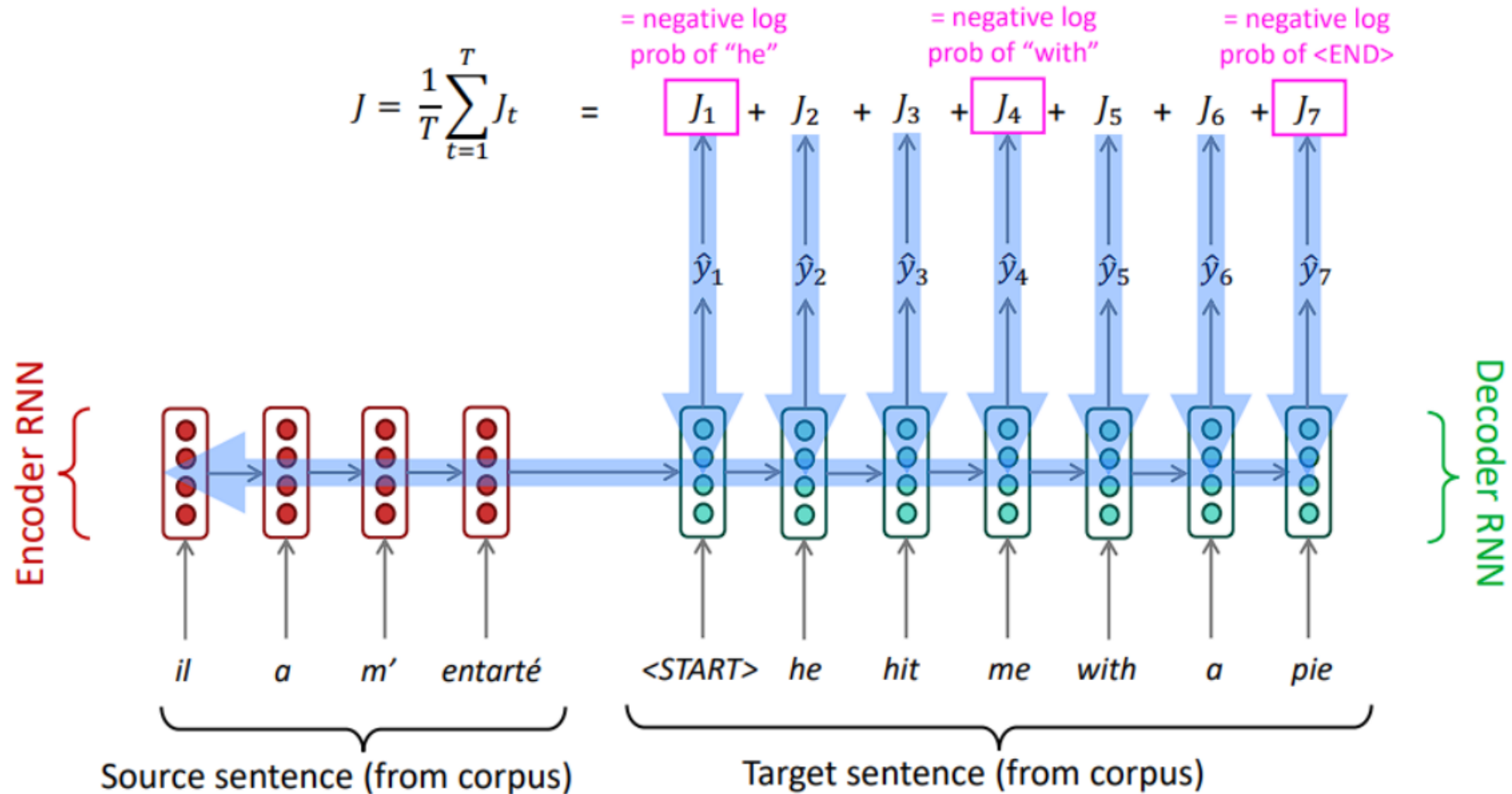
$$\bullet \operatorname{argmax}_y P(y|x) = \operatorname{argmax}_y P(x|y)P(y) \text{ (using Bayesian Rule)}$$

- $P(x|y)$: Translation Model
 - 단어나 구절이 어떻게 번역되어야 하는가
 - Parallel data를 통해 학습
- $P(y)$: Language Model
 - Target language에 대한 지식, 자연스러운 문장이 어떤 구조인가
 - Monolingual data를 통해 학습

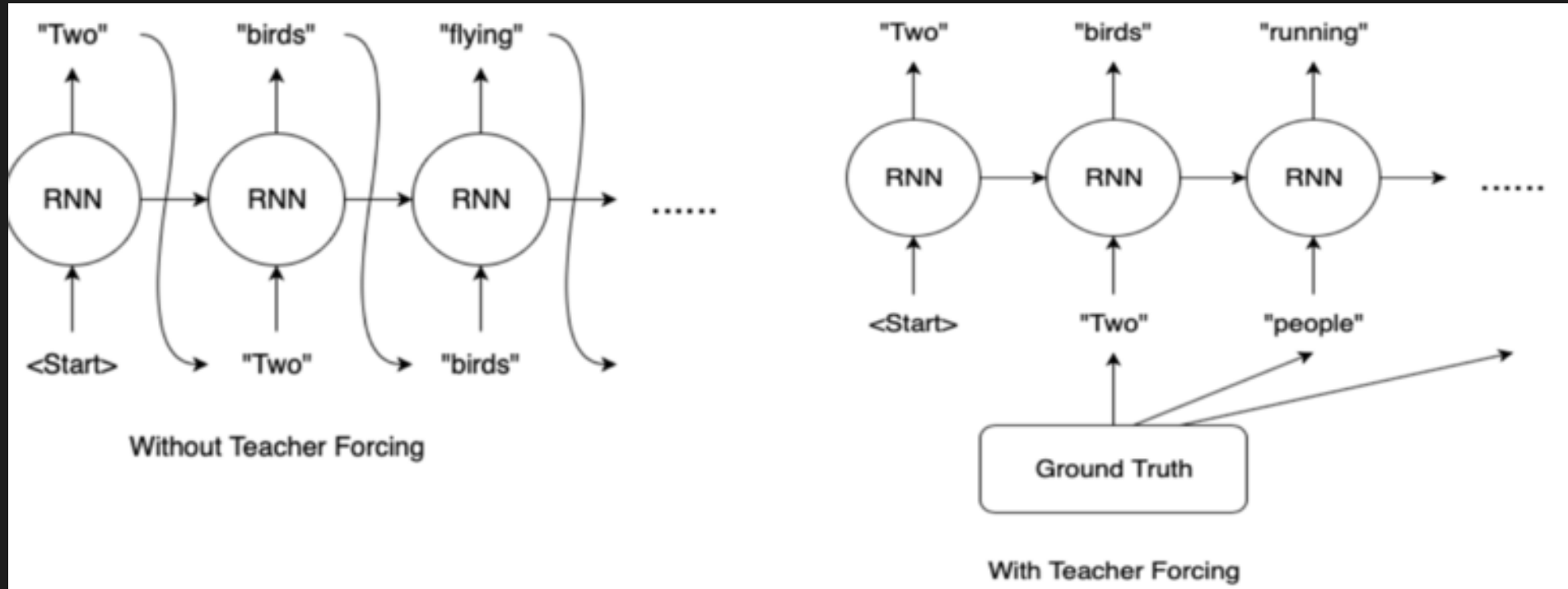
Neural Machine Translation

- Neural Machine Translation(NMT)
 - MT task를 end-to-end neural network를 통해 해결해보자.
 - Neural network architecture는 **sequence-to-sequence(seq2seq)** 모델이라고 부르며, 2개의 RNNs으로 이루어져 있다.
 - Encoder RNN
 - Decoder RNN

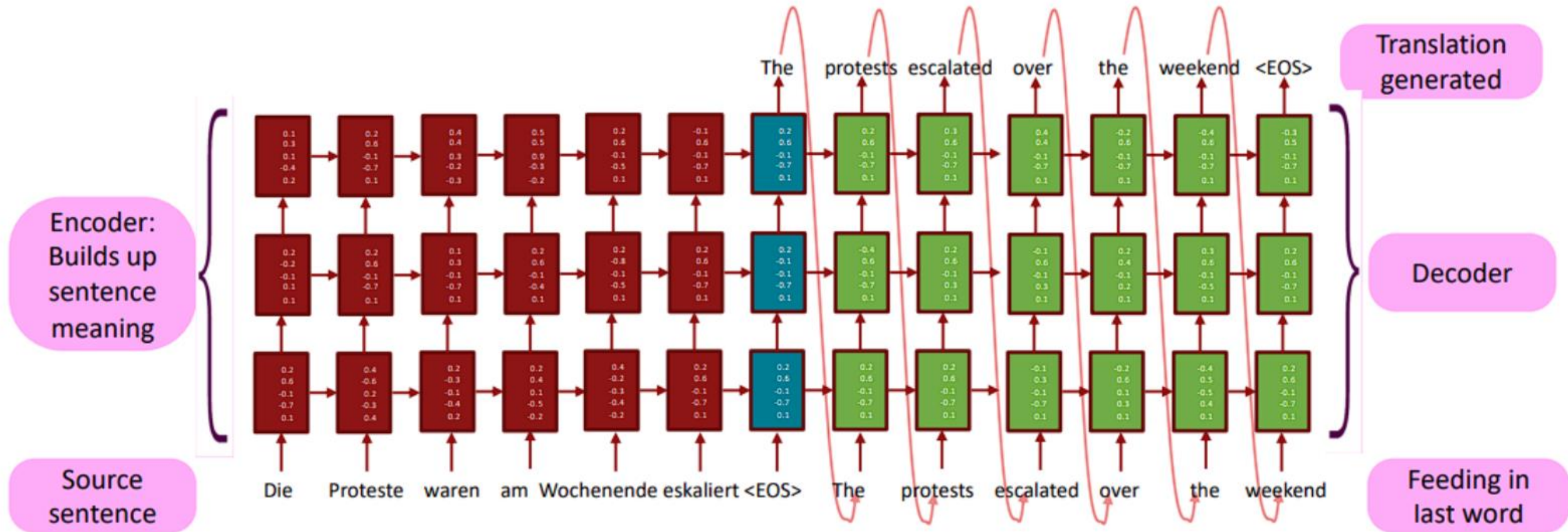
Neural Machine Translation



Teacher Forcing



Multi-layer RNNs



Neural Machine Translation

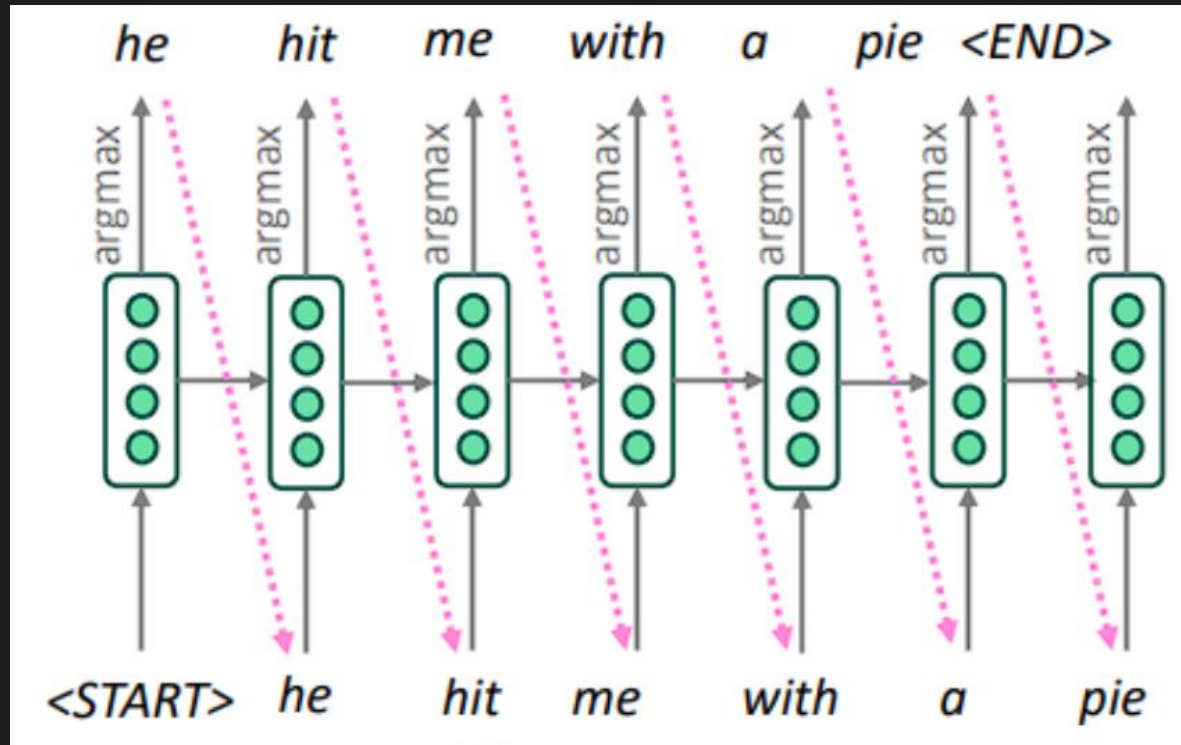
- Seq2Seq model은 Conditional Language Model이다.
 - Decoder가 **source sentence**와 **지금까지 예상한 target sentence**를 바탕으로 다음 단어를 예측하기 때문이다.

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)P(y_3|y_1, y_2, x) \dots P(y_T|y_1, \dots, y_{T-1}, x)$$

- Seq2Seq model은 MT외에도 여러가지 task를 수행할 수 있다.
 - Summarization
 - Dialogue
 - Parsing
 - Code generation

Greedy decoding

- 매 step마다 가장 확률이 높은 단어를 선택
 - 문제점은? 'he' 다음에 올 단어는?

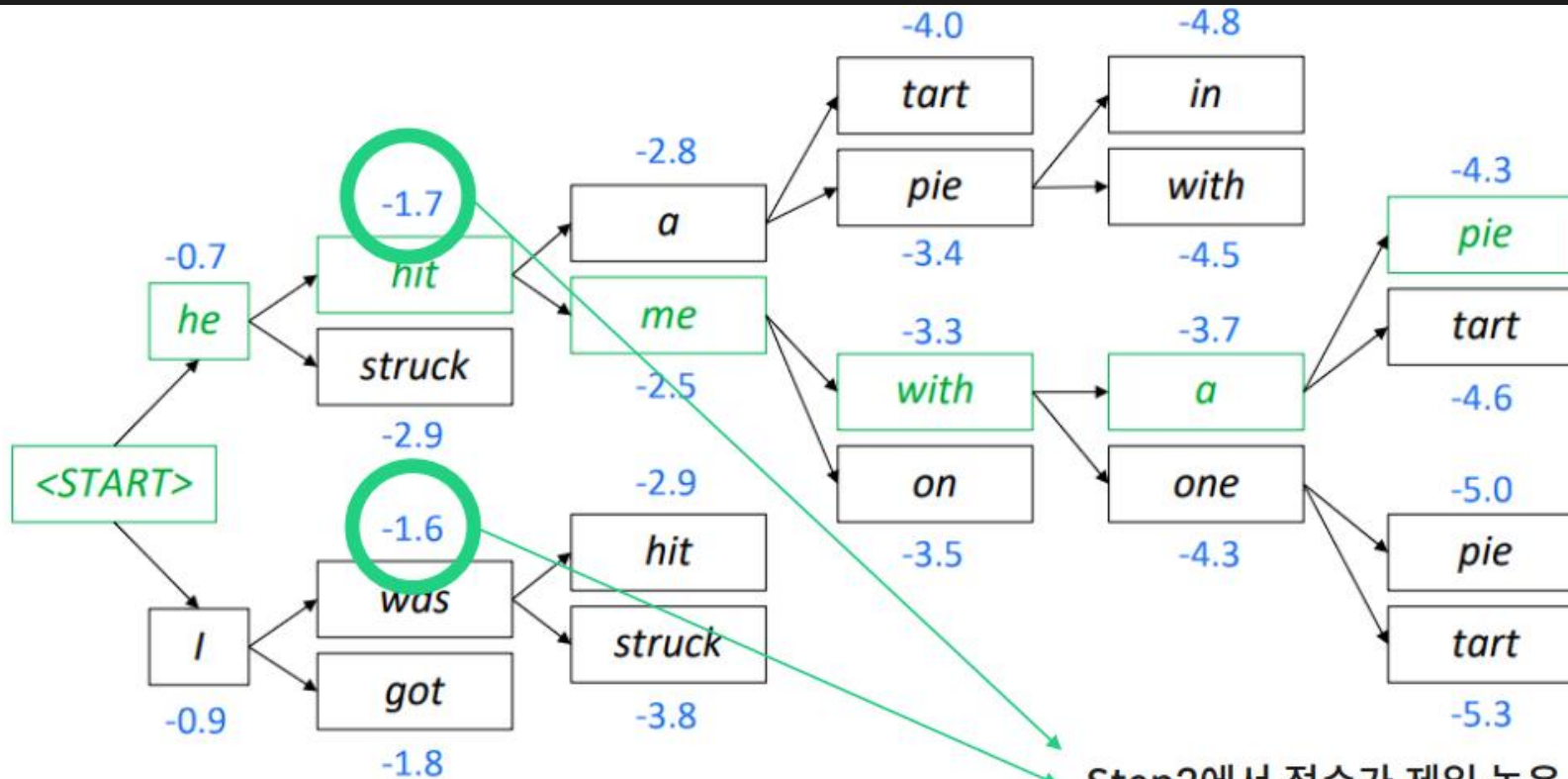


Exhaustive search decoding

- 모든 가능한 y 를 계산해보자
 - 단어의 개수가 V 개, $\text{step}(\text{sequence의 길이})$ 가 t 라면 총 경우의 수는 V^t 개
- V^t 개의 경우를 모두 계산하는 건 시간과 비용 측면에서 좋지 않다.

Beam search decoding

- 각 step마다 확률이 가장 높은 k개만 남겨둔다.
 - Optimal solution을 찾는다는 보장은 없지만 exhaustive search보다 훨씬 효율적



출처: CS224n lecture 7

Step2에서 점수가 제일 높은 2개에 대해서만
step3를 진행

N-gram Penalty

- 언어모델의 고질적인 문제점 중 하나는 동일한 말을 계속 반복한다는 것
- n-gram 단위의 시퀀스가 두 번 이상 등장할 일이 없도록 확률을 0으로 만드는 전략
 - n-gram : n개의 연속적인 단어 나열

Definition:

N-grams are a sequence of n tokens from a sample of text.

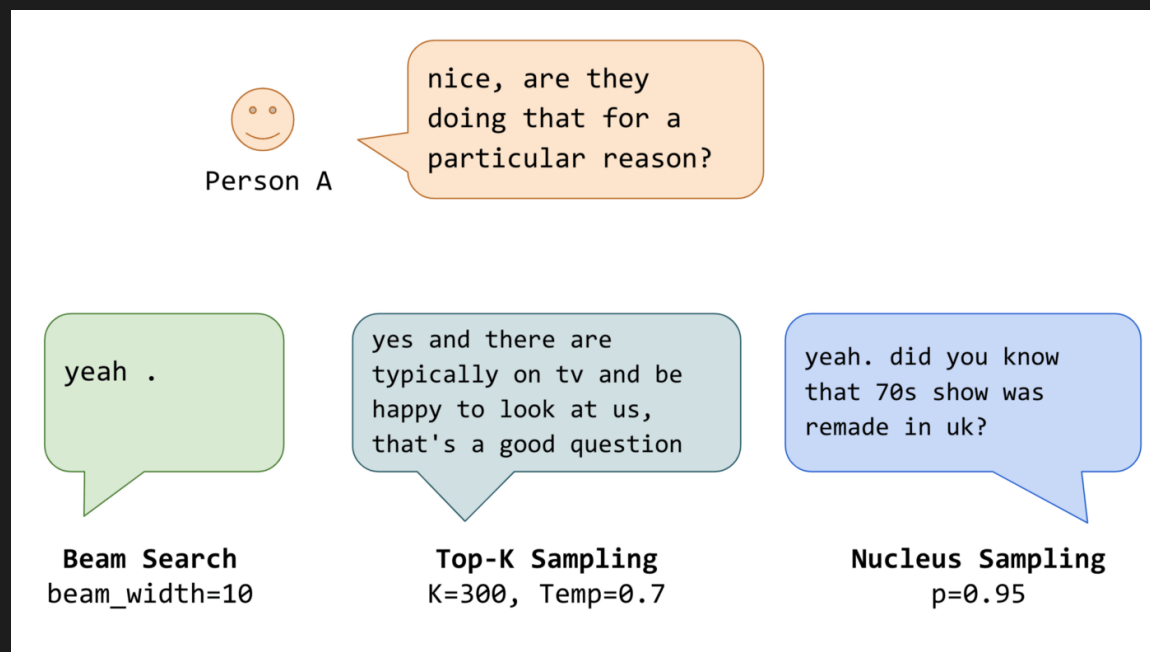
green eggs and ham 2-gram

green eggs and ham 3-gram

green eggs and ham 4-gram

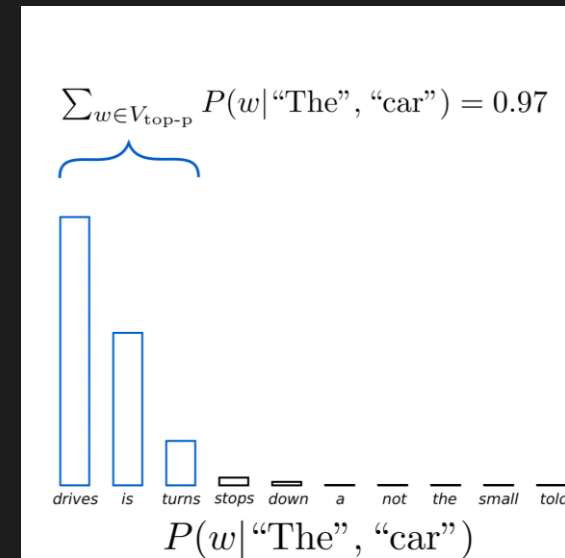
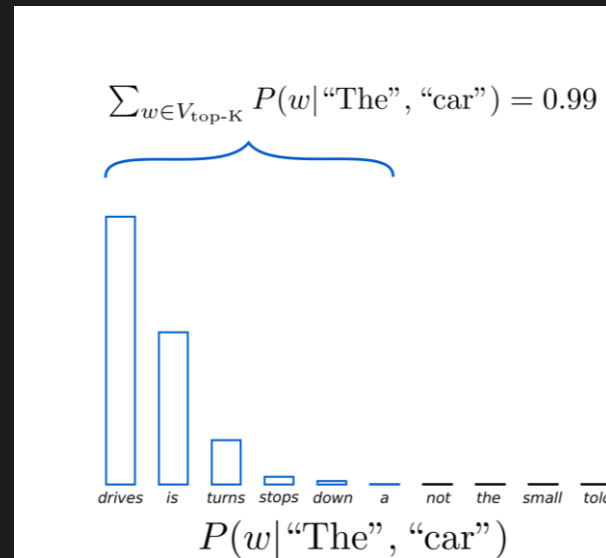
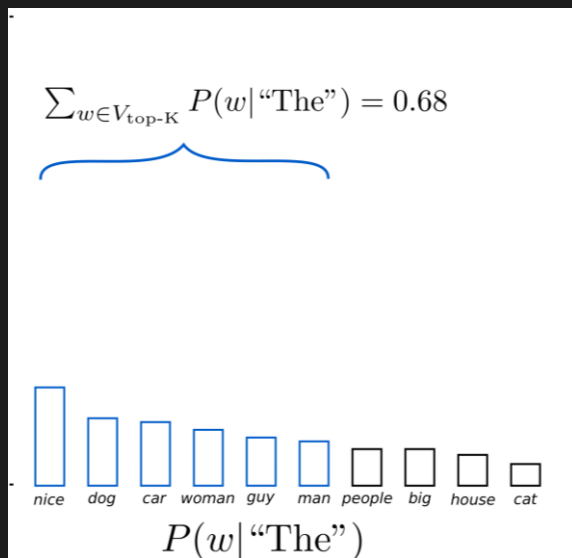
Problem : Beam Search is Boring

- 요약, 번역과 같이 답이 어느정도 정해져 있는 Task에서는 Beam Search가 효과적이다.
- Dialog/Story Generation과 같은 open generation task에는 적절하지 않다. (less surprising)
- Solution : “Sampling”의 도입



Top-k/p Sampling

- Top-k Sampling
 - 다음 단어를 확률이 가장 높은 k개 중에서 랜덤하게 sample한다.
- Top-p Sampling
 - 다음 단어를 누적 확률이 p 이상이 되는 최소한의 집합 안에서 랜덤하게 sample한다.



Neural Machine Translation

- Advantages

- SMT보다 더 좋은 성능
- 하나의 end-to-end 모델
- 인간의 노력이나 보수 작업이 필요하지 않음

- Disadvantage

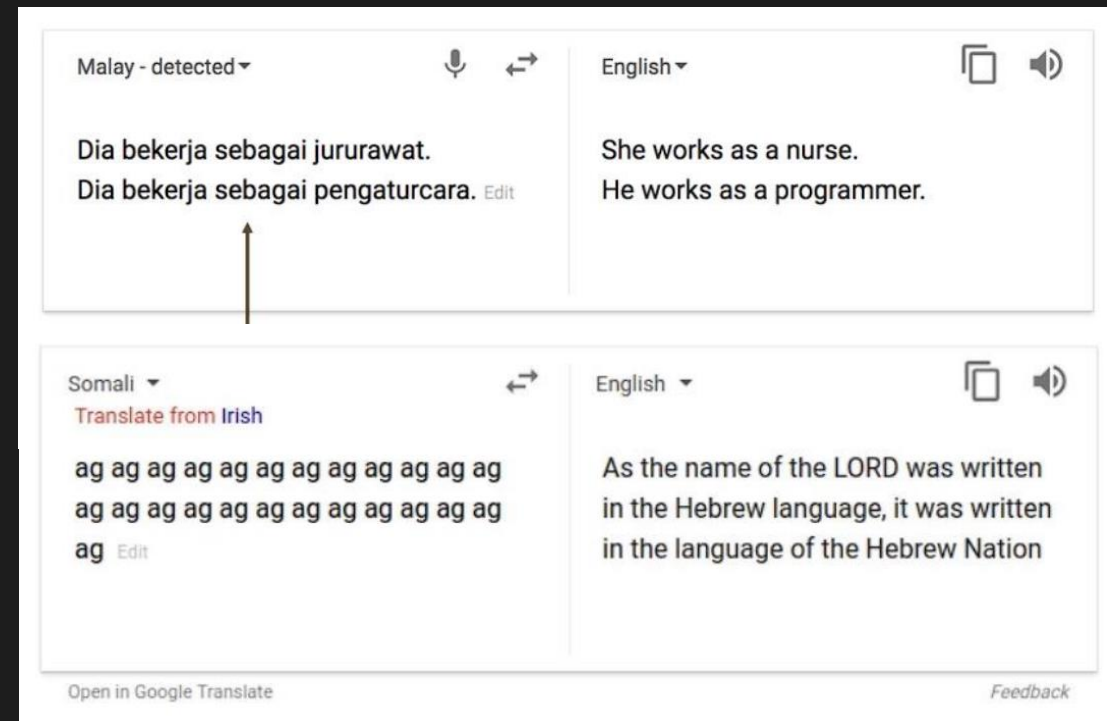
- Black box 모델
- 디버깅이 힘들다
- 모델이 어떠한 규칙으로 번역을 하는지 알 수 없다
- Bias의 문제

Evaluation of Machine Translation

- BLEU (Bilingual Evaluation Understudy)
 - 모델이 번역한 결과와 사람이 직접 번역한 결과가 얼마나 비슷한지 n-gram을 기준으로 비교
 - 두 문장 사이에 겹치는 n-gram이 몇 개인지 세기
 - 같은 의미를 가진 좋은 번역이더라도 BLEU 점수는 낮을 수 있다
 - Example) 예쁘다 vs 아름답다
- R1: The cat is on the mat
- R2: There is a cat on the mat
- C1: The cat and the dog
- C2: The The The The The.
- C3: There is a cat on the mat.
- C4: Mat the cat is on a there.

So, is Machine Translation solved?

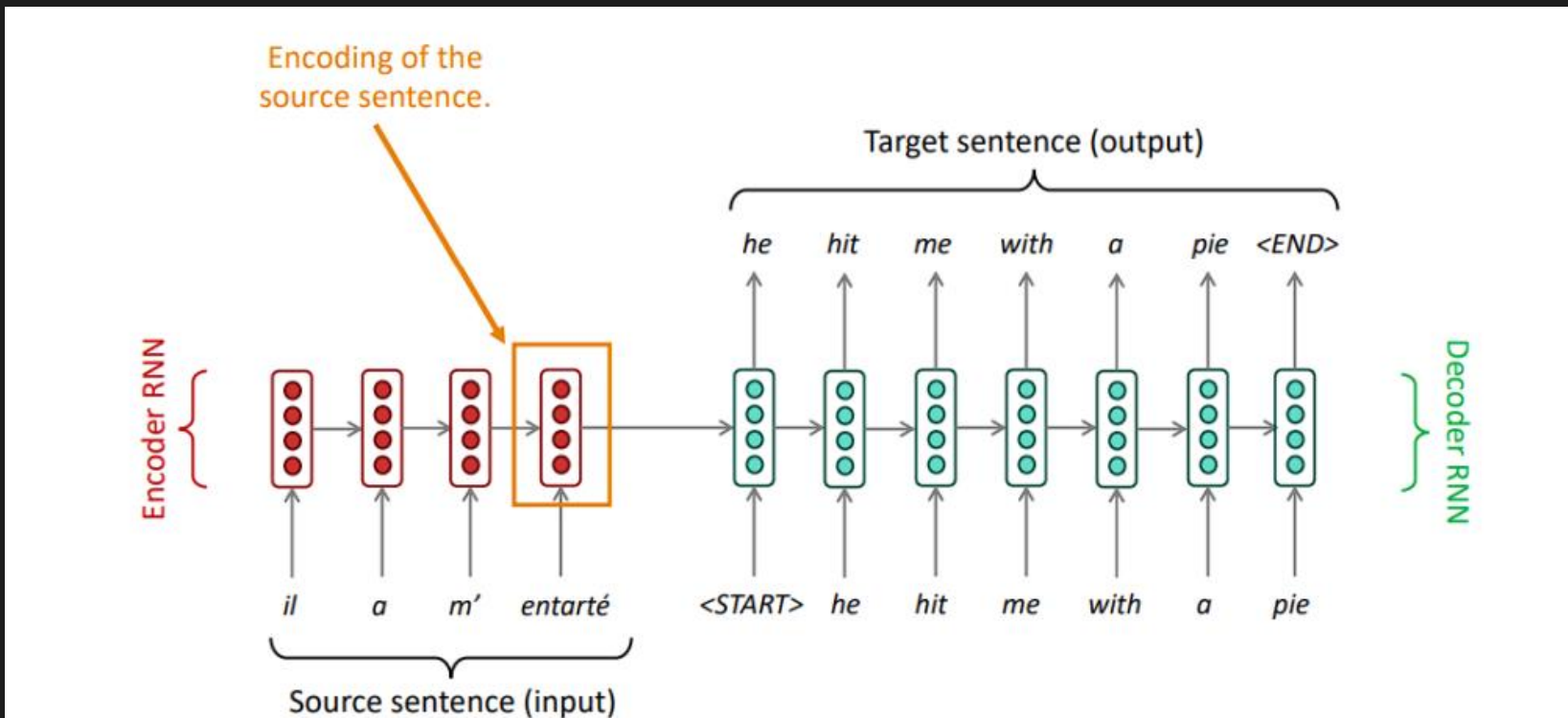
- 관용구를 잘 해석하지 못한다.
- 편향된 지식이 그대로 학습된다.
- Black box 모델이기 때문에 이상한 번역을 해도 이유를 알 수 없다.



Attention

Bottleneck problem

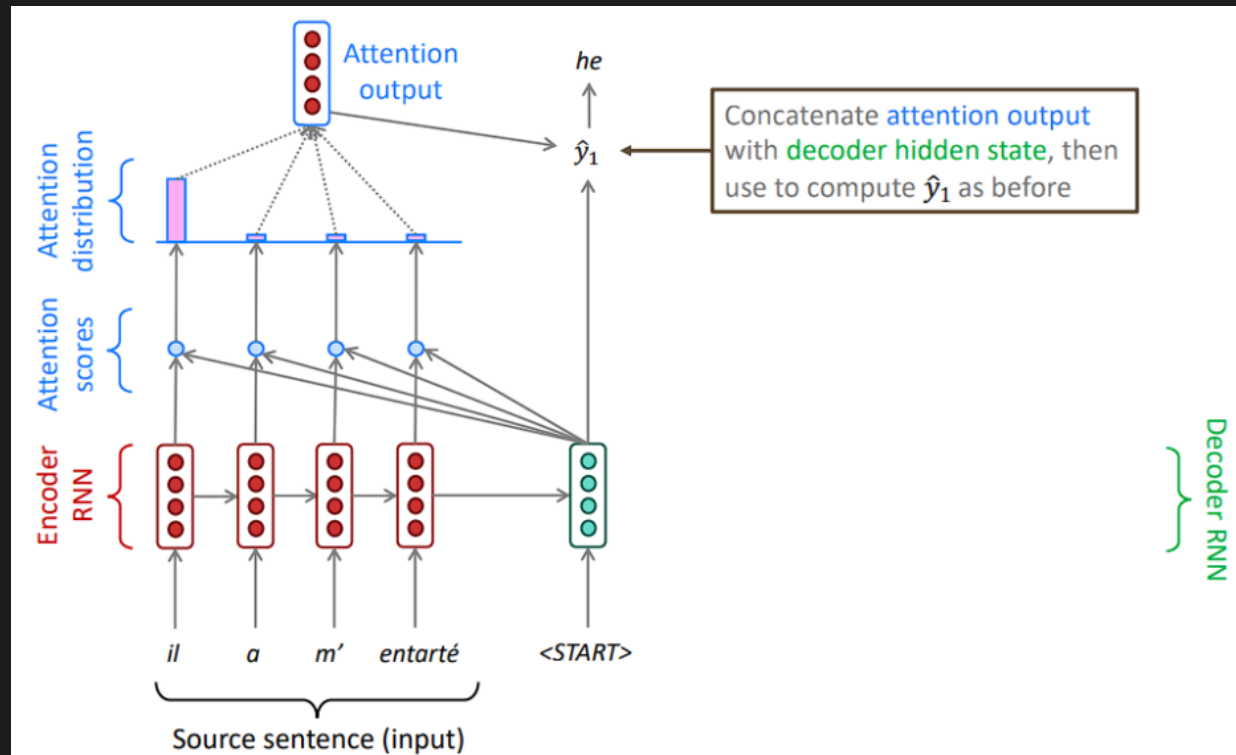
- Encoder의 **마지막 hidden state**를 통해 source sentence를 표현한다
 - 하나의 hidden state로 source sentence의 모든 정보를 담을 수 있을까?
 - 문장이 길어질수록 더 힘들 것이다.



How to Solve Bottleneck problem?

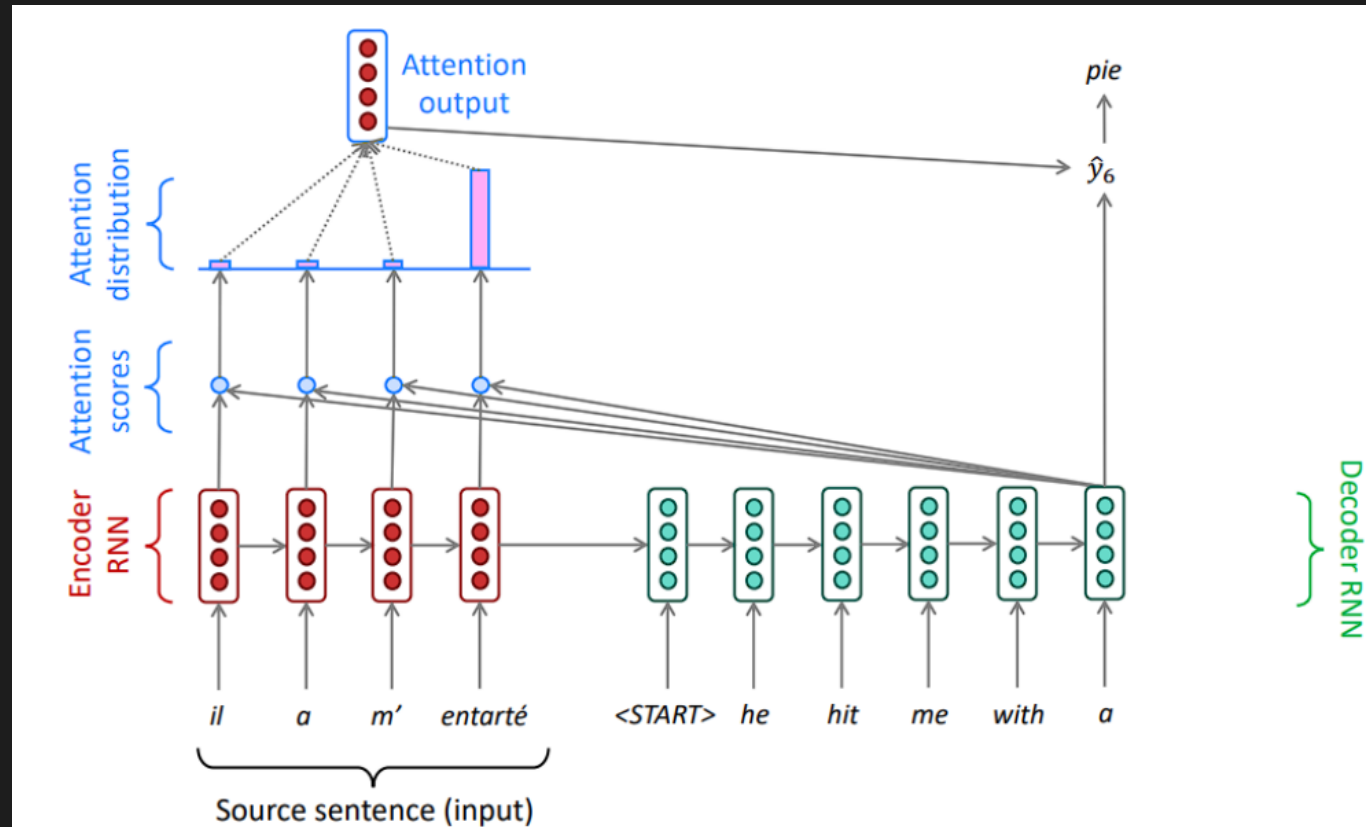
- Attention

- Attention output과 Decoder의 hidden state를 Concatenation한 벡터를 다시 softmax 함수에 통과시켜 Output words를 예측



How to Solve Bottleneck problem?

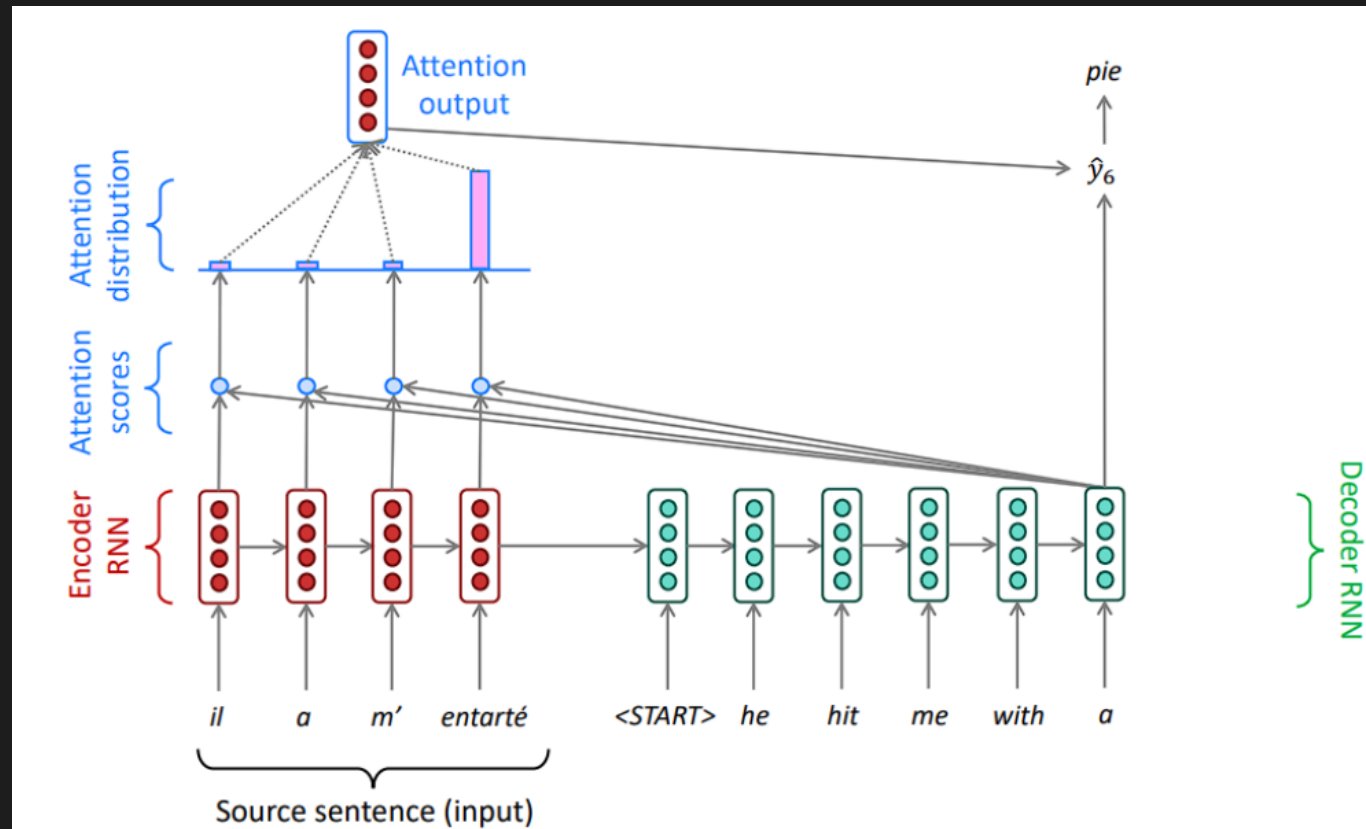
- Attention
 - 지금까지의 과정을 Decoder의 매 step마다 반복



How to Solve Bottleneck problem?

- Attention

- Attention Output은 Input 중 “어느 부분에 집중 할 것인가?” 에 대한 정보를 줄 수 있음.

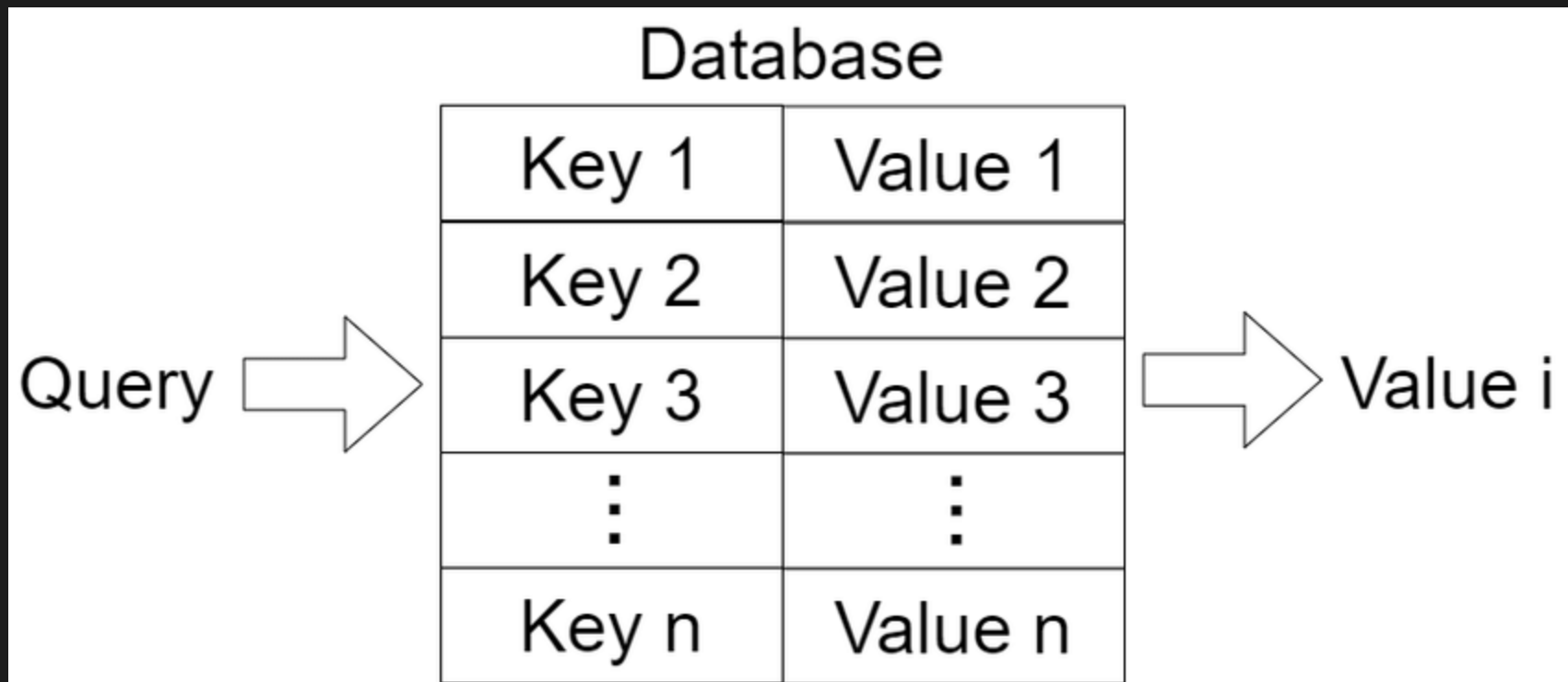


Attention is great

- Attention으로 인해 NMT의 성능이 더 좋아졌다.
- Attention의 작동 원리는 사람이 번역을 하는 방법과 유사하다.
 - Source sentence를 모두 외워서 번역하는 것이 아니라 매 step마다 source sentence를 다시 확인하면서 번역하기 때문이다.
- Attention을 통해 Bottleneck / Vanishing Gradient problem을 해결했다.
 - Decoder의 각 step에서 Encoder의 hidden state로 이어지는 shortcut을 제공하기 때문이다.
- Attention은 Interpretability를 제공해준다.
 - 왜 그 부분을 주목하는지는 아직 알 수 없지만 최소한 Decoder의 각 step이 어느 부분을 주목해서 번역하는지, attention distribution을 통해 확인할 수 있다.

Attention to “Attention”

- Key, Query, and Value
- Similar to Database Design



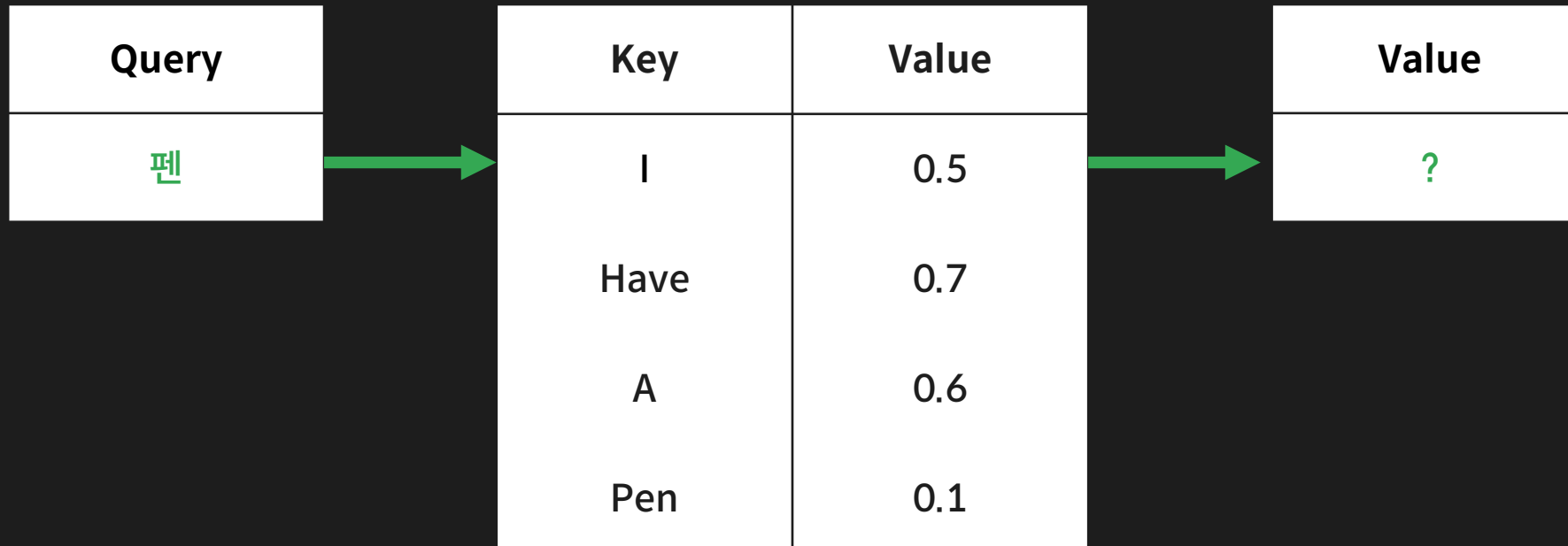
Attention to “Attention”

- Key, Query, and Value
- Similar to Database Design



Attention to “Attention”

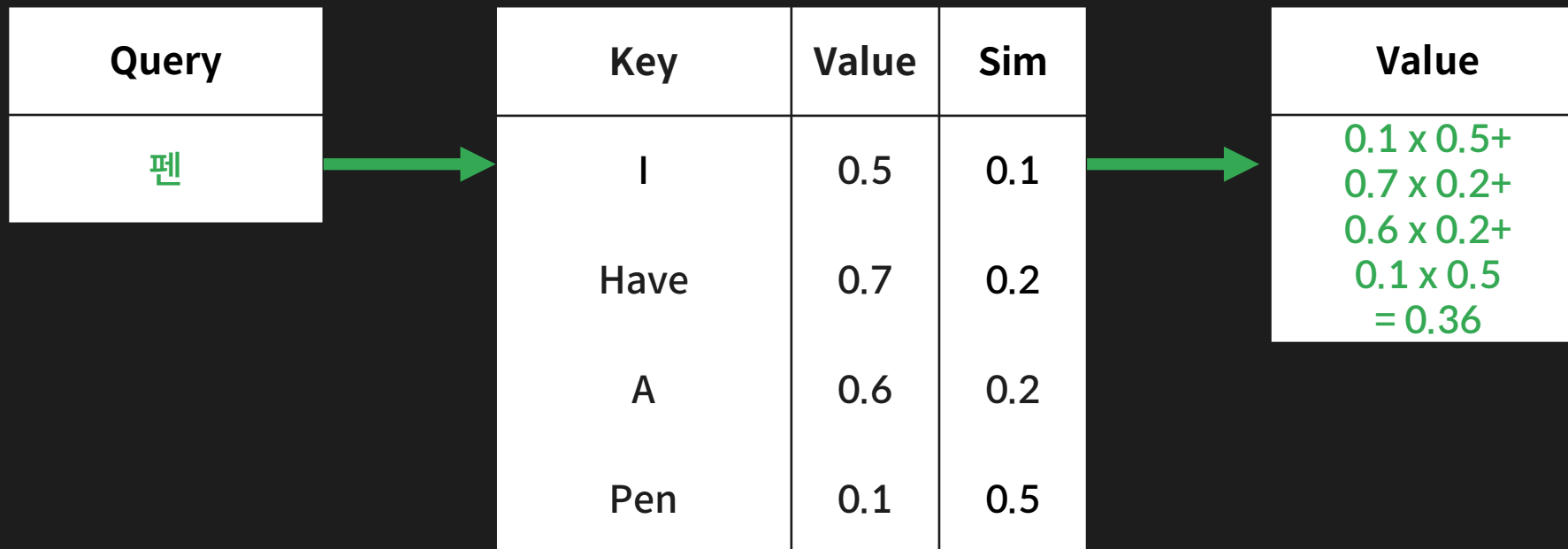
- Key, Query, and Value
- Similar to Database Design



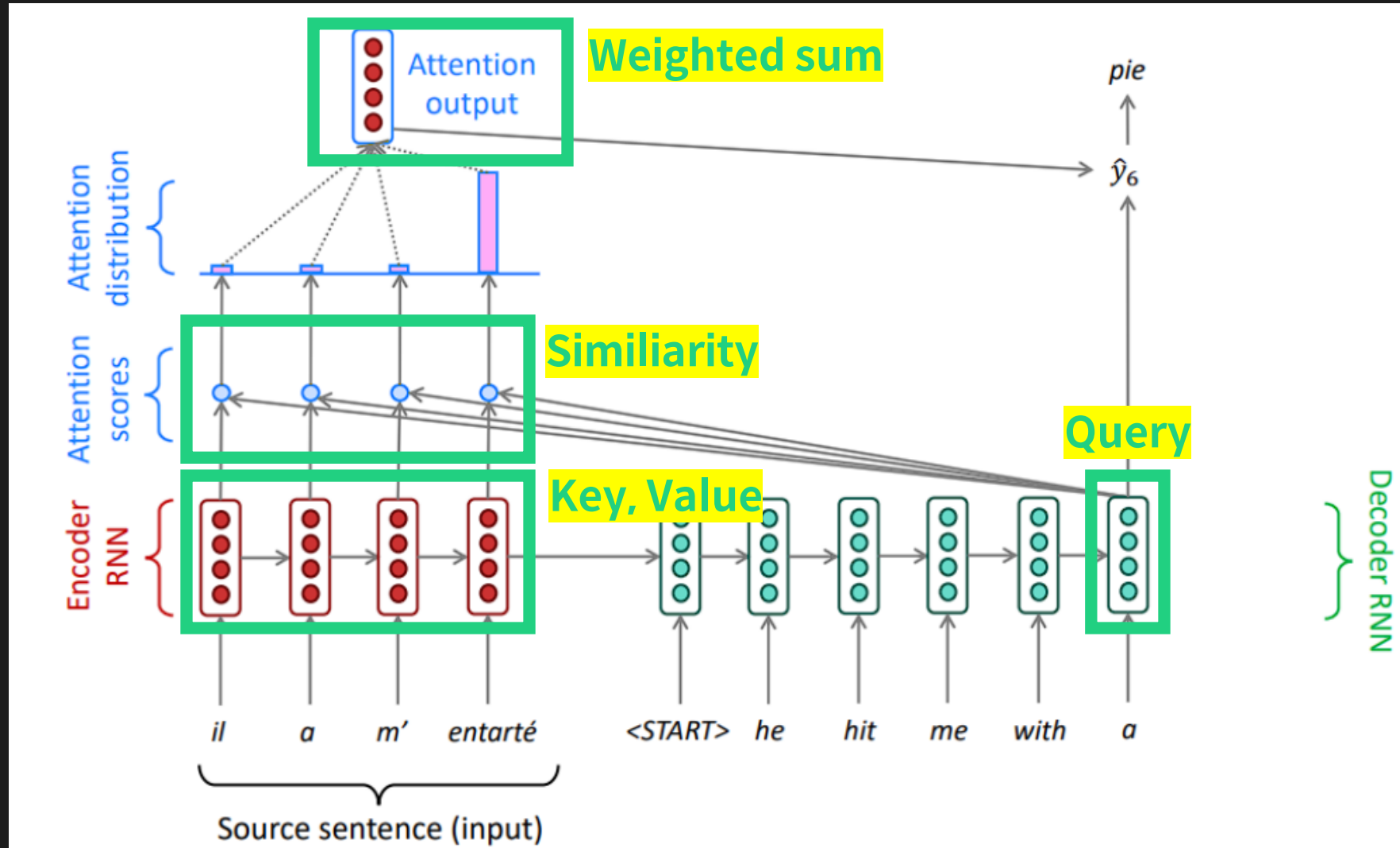
- How similar “펜” and “Pen”?

Attention to “Attention”

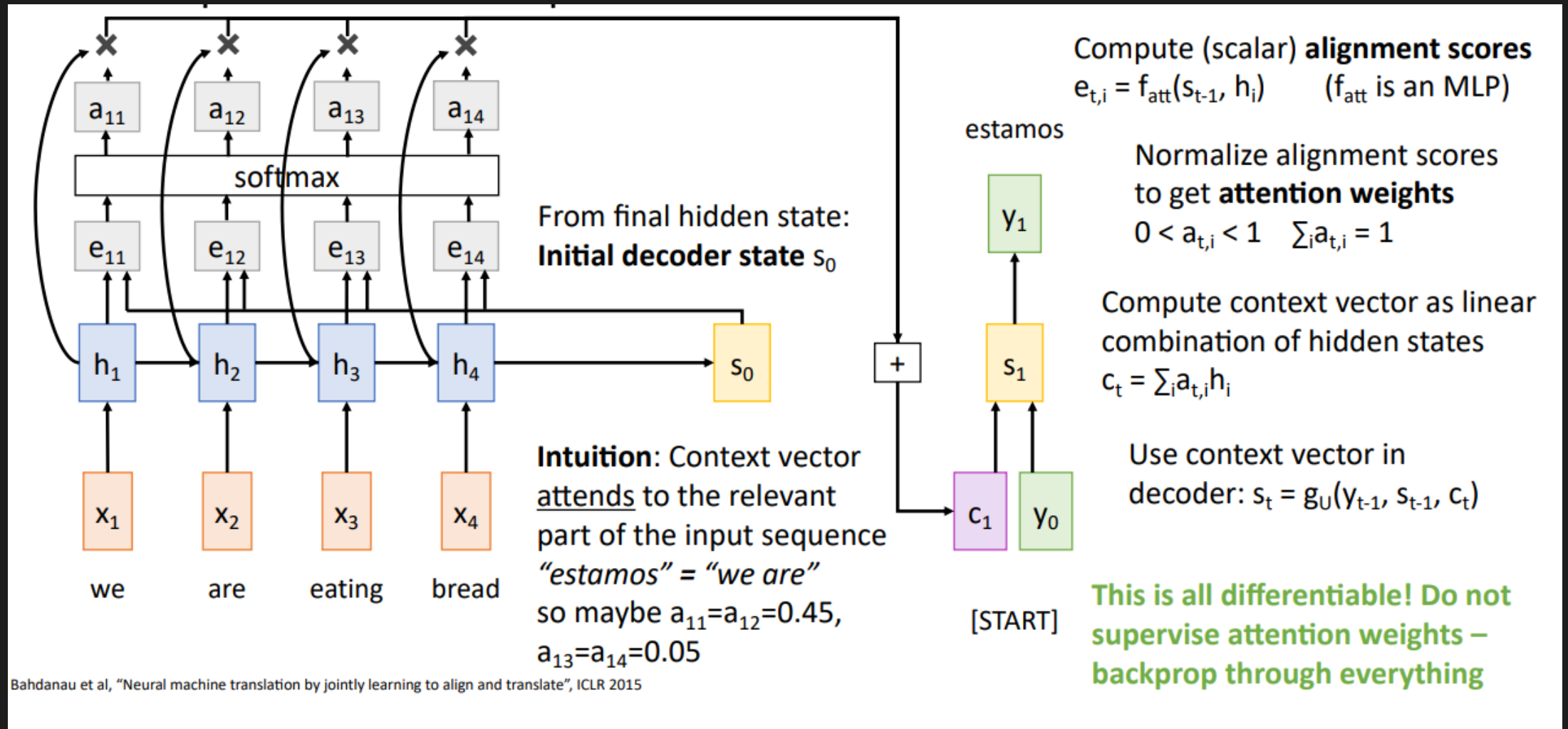
- How similar “펜” and “Pen”? :
 - Query와 Key의 Simliarity를 구할 수 있으면, value에 대한 weighted sum은 Query의 value라고 생각할 수 있다.



Attention to “Attention”



Attention to “Attention”



Visualizing Attention

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Output: “L'accord sur la zone économique européenne a été signé en août 1992.”

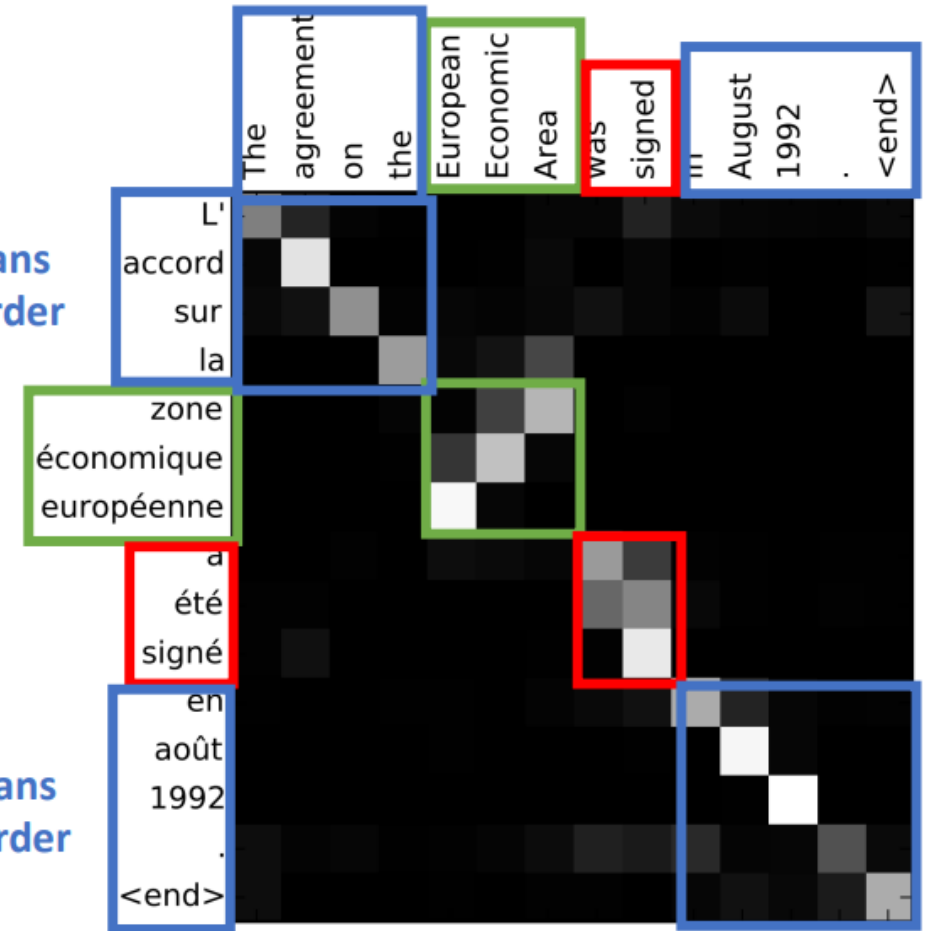
Diagonal attention means words correspond in order

Attention figures out different word orders

Verb conjugation

Diagonal attention means words correspond in order

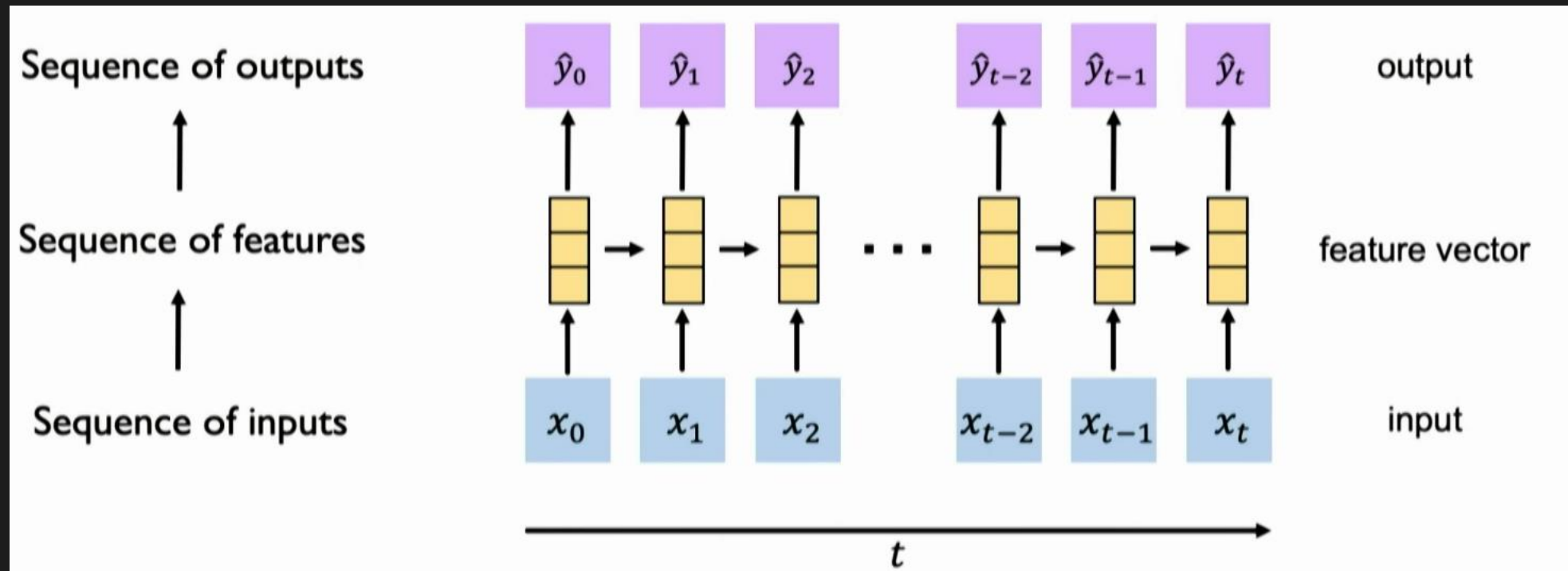
Visualize attention weights $a_{t,i}$



Janau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Recurrent vs. Attention

- Desired Capabilities of Sequential Modeling
 - Continuous stream
 - Parallelization
 - Long memory

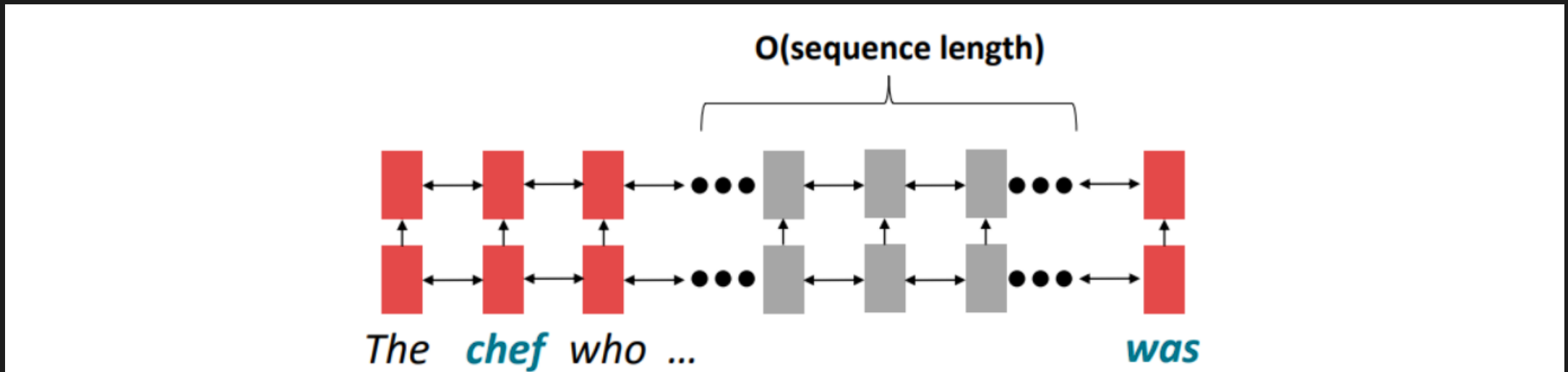


Recurrent vs. Attention

- Issues with recurrent models

- 1. Linear interaction distance

- RNN은 왼쪽에서 오른쪽으로 차례차례 연결되는 구조
 - 떨어진 거리만큼 가야 서로 영향을 줄 수 있다
 - 멀리 떨어진 단어 사이의 의존성을 모델이 잘 반영하지 못한다. (Vanishing Gradient)
 - 반면, Attention은 멀리 떨어진 단어라도 서로의 의존성을 모델이 잘 학습할 수 있다.

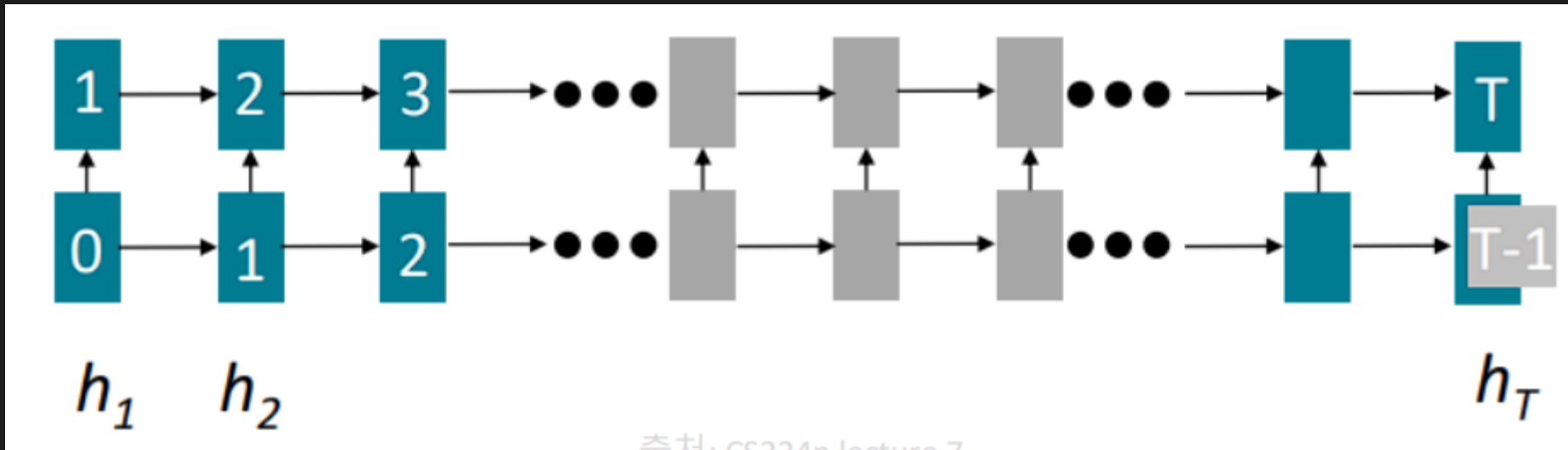


Recurrent vs. Attention

- Issues with recurrent models

- 2. Lack of parallelizability

- 다음 timestep의 hidden state는 그 이전 timestep에 의존한다.
 - 즉, 병렬처리를 할 수 없고, 시간이 오래걸린다. ($O(\text{len}(x))$)
 - 데이터셋이 커질수록, 학습 속도가 매우 오래 걸린다.
 - 반면, Attention은 몇 번의 Matrix Multiplication과 Dot product로 구현되므로, 병렬화가 가능하다.



Self-Attention as an NLP building block

- Issues with recurrent models

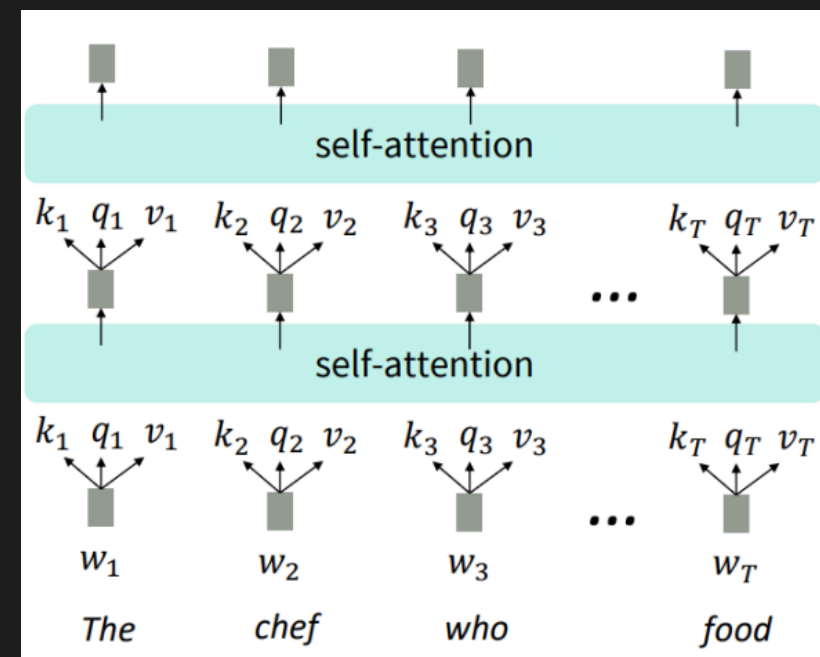
⇒ RNN에 Attention을 달지 말고, 그냥 Attention만 써보자!

- Self-Attention

- 자기 자신에 attend하기, 같은 문장 안에서도 다른 부분을 봐야만 이해할 수 있는 단어들이 있다.
- Self-attention에서는 Query, Key, Value를 모두 같은 값으로부터 가져온다.
- 즉, 자기 자신에게 Query를 날린다.

- Recurrent module을 제거하고,
Self-Attention을 쌓아 NLP Building Block으로 사용할 수 있다.

- Problem?



(Generalized) Attention Layer

Attention Layer

Inputs:

Query vectors: \mathbf{Q} (Shape: $N_Q \times D_Q$)

Input vectors: \mathbf{X} (Shape: $N_X \times D_X$)

Key matrix: \mathbf{W}_K (Shape: $D_X \times D_Q$)

Value matrix: \mathbf{W}_V (Shape: $D_X \times D_V$)

Computation:

Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

X_1

X_2

X_3

Q_1

Q_2

Q_3

Q_4

(Generalized) Attention Layer

Attention Layer

Inputs:

Query vectors: \mathbf{Q} (Shape: $N_Q \times D_Q$)

Input vectors: \mathbf{X} (Shape: $N_X \times D_X$)

Key matrix: \mathbf{W}_K (Shape: $D_X \times D_Q$)

Value matrix: \mathbf{W}_V (Shape: $D_X \times D_V$)

Computation:

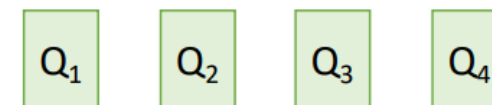
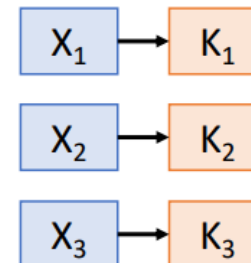
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



(Generalized) Attention Layer

Attention Layer

Inputs:

Query vectors: \mathbf{Q} (Shape: $N_Q \times D_Q$)

Input vectors: \mathbf{X} (Shape: $N_X \times D_X$)

Key matrix: \mathbf{W}_K (Shape: $D_X \times D_Q$)

Value matrix: \mathbf{W}_V (Shape: $D_X \times D_V$)

Computation:

Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

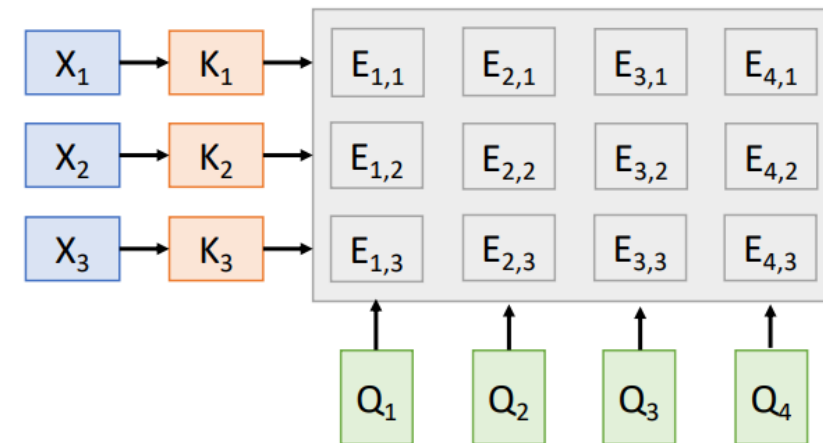
Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

이름	스코어 함수	Defined by
<i>dot</i>	$score(s_t, h_i) = s_t^T h_i$	Luong et al. (2015)
<i>scaled dot</i>	$score(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	Vaswani et al. (2017)
<i>general</i>	$score(s_t, h_i) = s_t^T W_a h_i$ // 단, W_a 는 학습 가능한 가중치 행렬	Luong et al. (2015)
<i>concat</i>	$score(s_t, h_i) = W_a^T \tanh(W_b[s_t; h_i]), score(s_t, h_i) = W_a^T \tanh(W_b s_t + W_c h_i)$	Bahdanau et al. (2015)
<i>location - base</i>	$\alpha_t = \text{softmax}(W_a s_t)$ // α_t 산출 시에 s_t 만 사용하는 방법.	Luong et al. (2015)



(Generalized) Attention Layer

Attention Layer

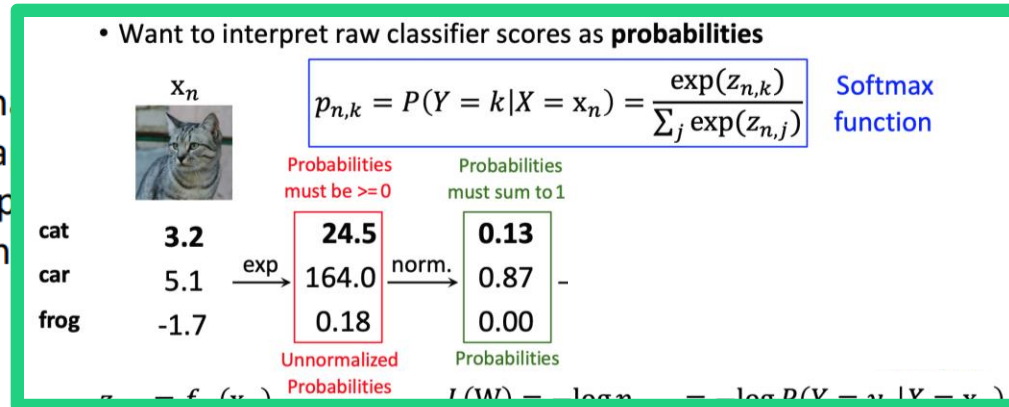
Inputs:

Query vectors: Q (Shape: $N_Q \times D_Q$)

Input vectors: X (Shape: $N_X \times D_X$)

Key matrix: W_K (Shape: $D_X \times D_K$)

Value matrix: W_V (Shape: $D_X \times D_V$)



Computation:

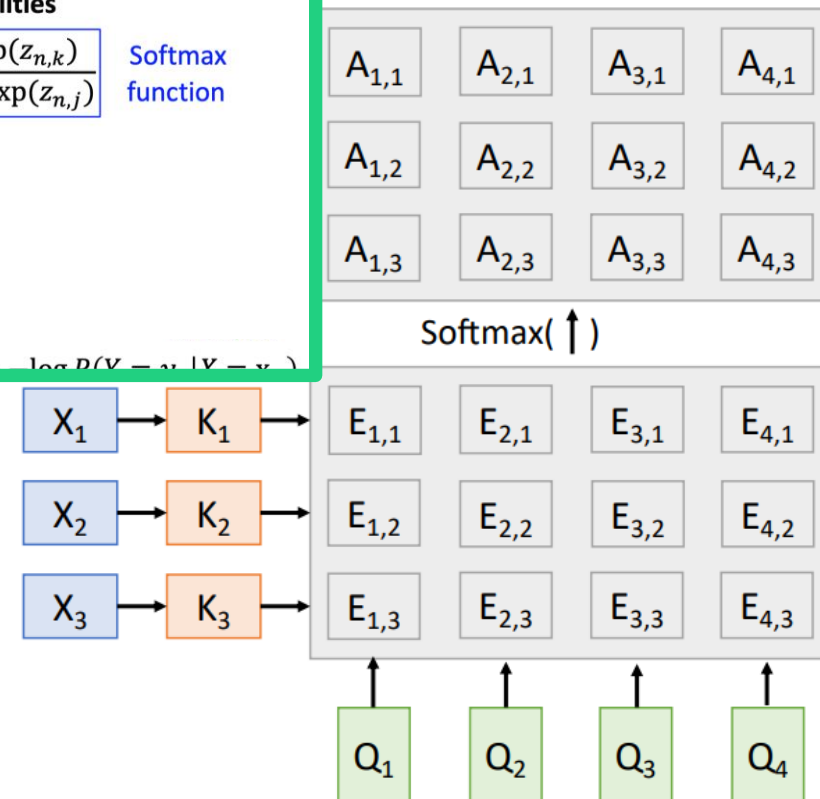
Key vectors: $K = XW_K$ (Shape: $N_X \times D_K$)

Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)

Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



(Generalized) Attention Layer

Attention Layer

Inputs:

Query vectors: \mathbf{Q} (Shape: $N_Q \times D_Q$)

Input vectors: \mathbf{X} (Shape: $N_X \times D_X$)

Key matrix: \mathbf{W}_K (Shape: $D_X \times D_Q$)

Value matrix: \mathbf{W}_V (Shape: $D_X \times D_V$)

Computation:

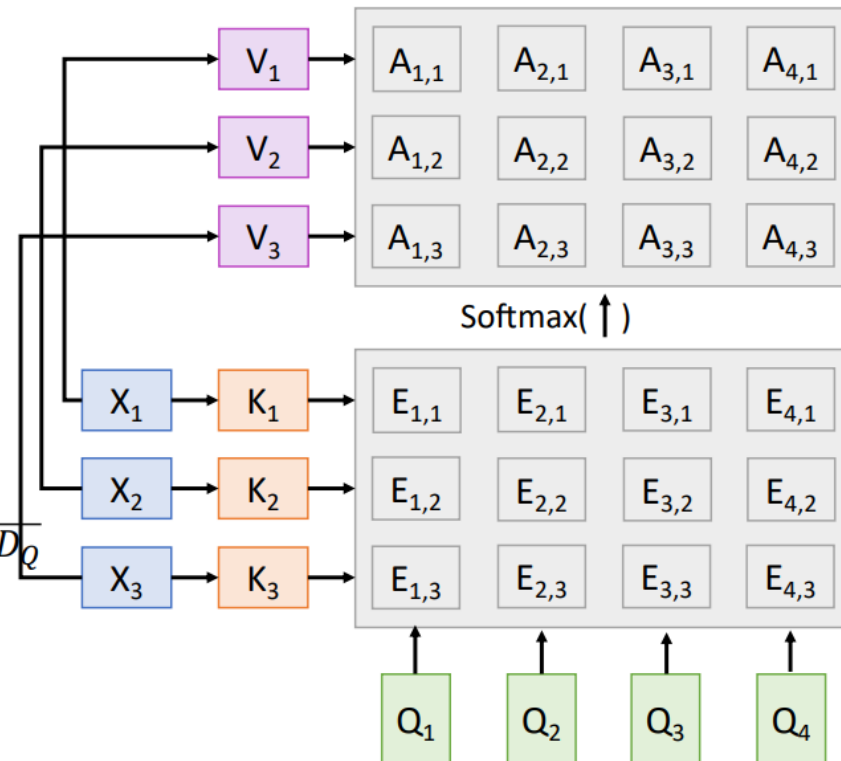
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



(Generalized) Attention Layer

Attention Layer

Inputs:

Query vectors: \mathbf{Q} (Shape: $N_Q \times D_Q$)

Input vectors: \mathbf{X} (Shape: $N_X \times D_X$)

Key matrix: \mathbf{W}_K (Shape: $D_X \times D_Q$)

Value matrix: \mathbf{W}_V (Shape: $D_X \times D_V$)

Computation:

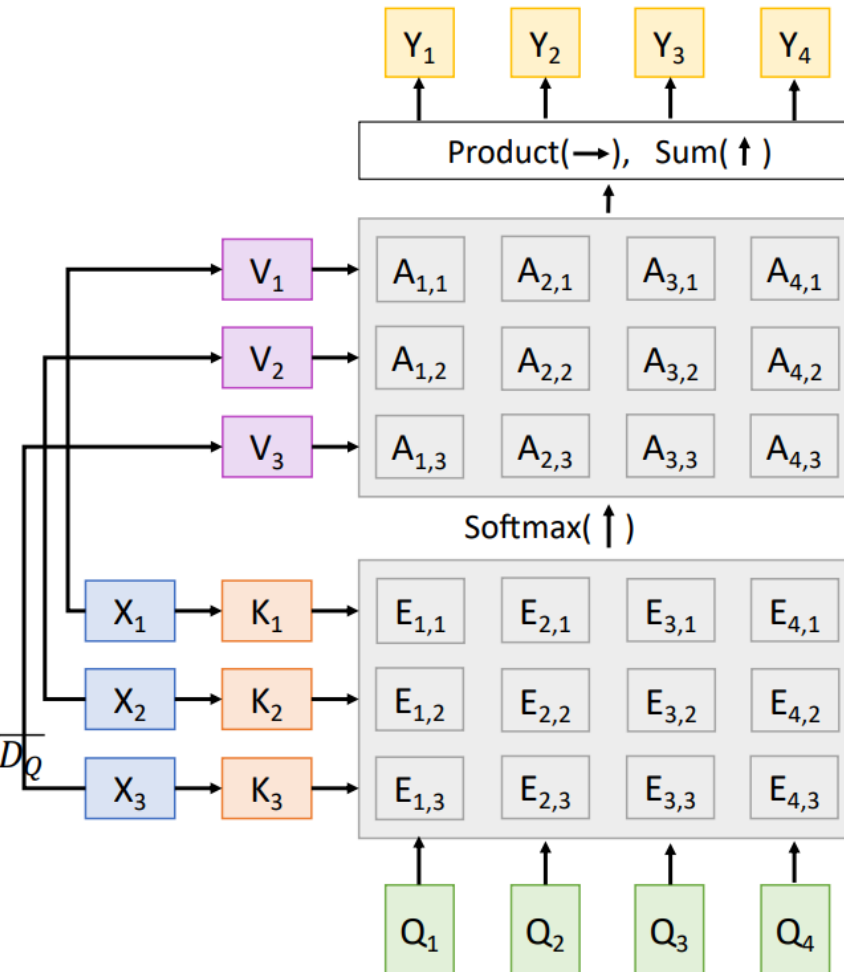
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_X \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_Q \times N_X$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_Q \times N_X$)

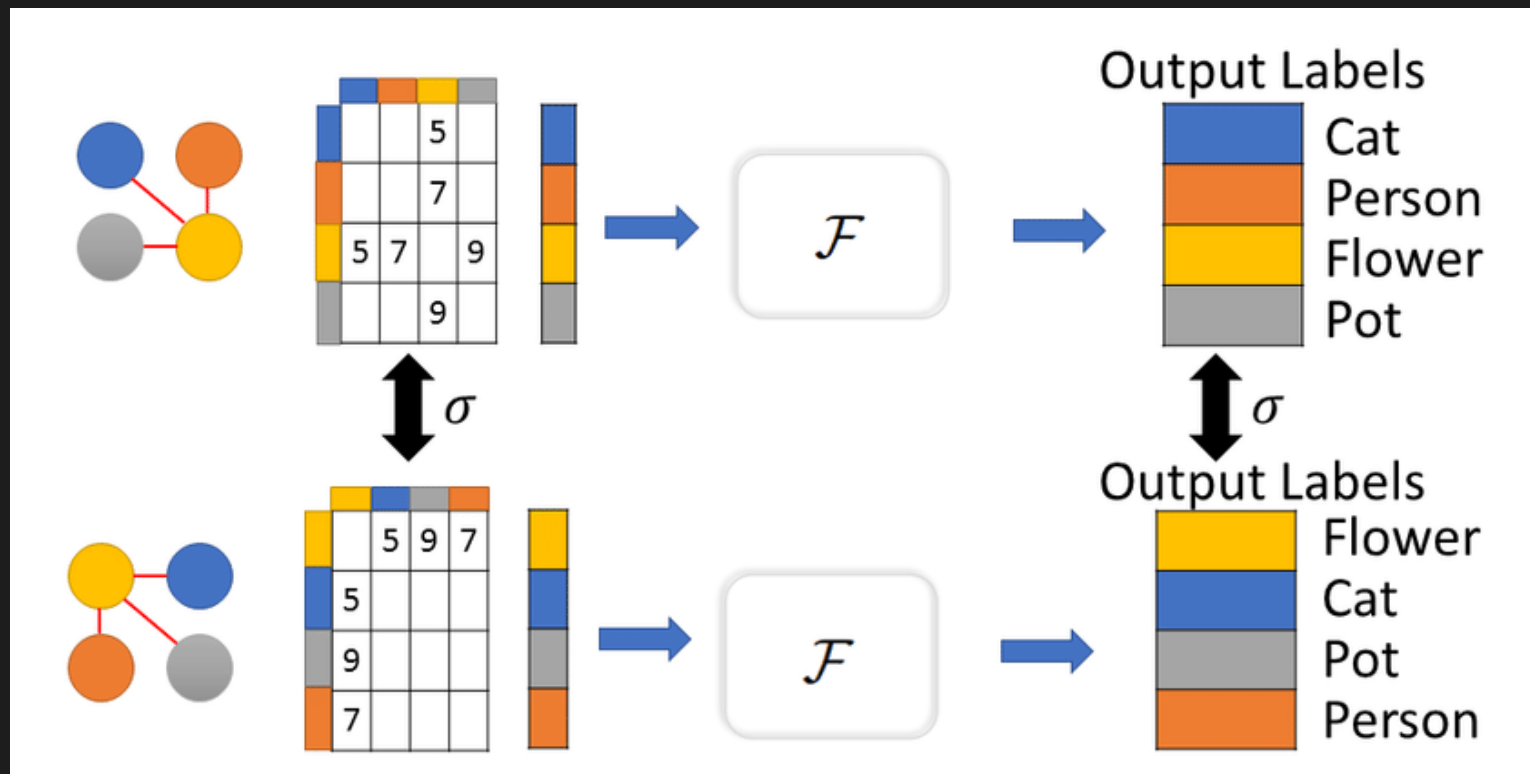
Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Permutation Invariance

- Permutation Invariance

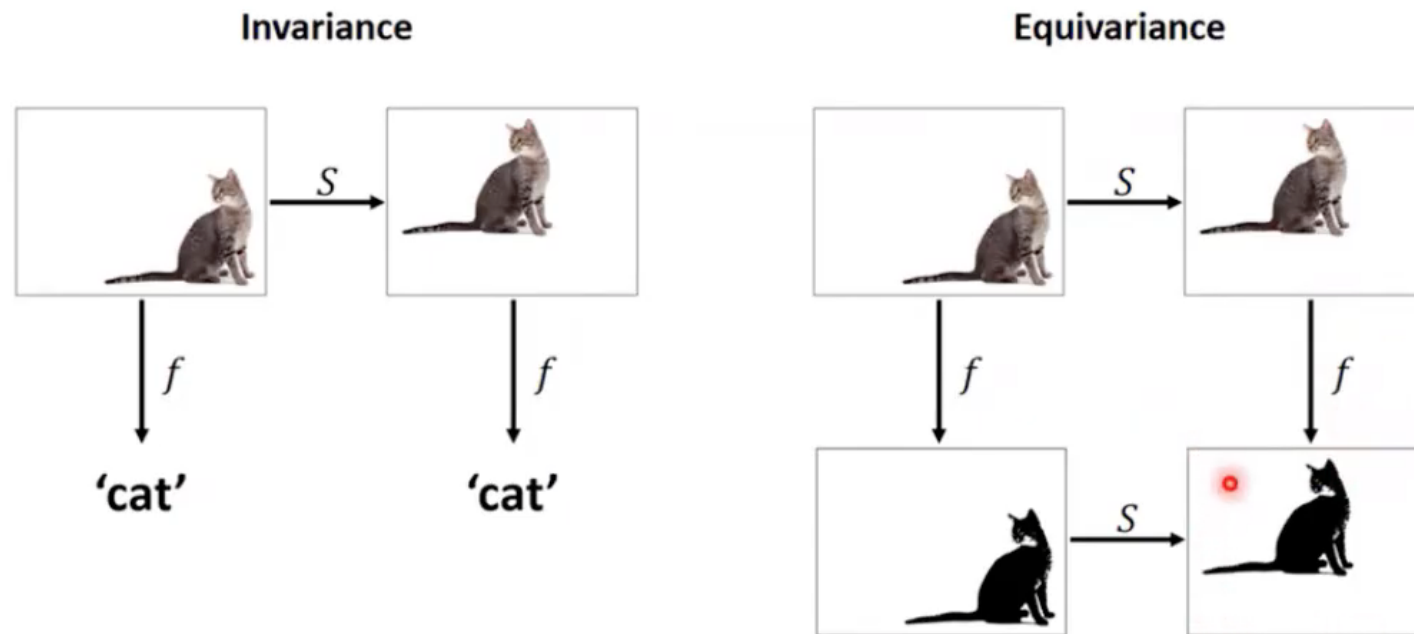
- 함수 (Network, module, layer, etc...)가 가질 수 있는 특성.
- Input이 들어오는 순서가 달라지면, Output의 순서만 바뀔 뿐 값은 동일하게 나오는 함수를 의미한다.



(Note) Spatial Invariance

- Translation Invariance vs. Equivariance

Invariance vs equivariance



디스커션 과제

- 개인 질문

- RNN과 비교하여, Attention만으로 Sequential Modeling을 할 때 생길 수 있는 문제점은 무엇이 있을까요?
어떻게 그 문제를 해결할 수 있을까요? 두 가지 이상 생각해봅시다.

- 팀 질문

- CNN, RNN, Attention을 Permutation / Spatial Invariance 관점에서 생각해봅시다.
 - 각각의 Network/layer는 Permutation / Spatial invariance 중 어떤 특성을 가질까요? 그 이유는 무엇일까요?
 - Attention을 CV에서 사용할 수 있을까요? 사용한다면 CNN과 비교해 어떤 이점이 있을까요?

감사합니다.