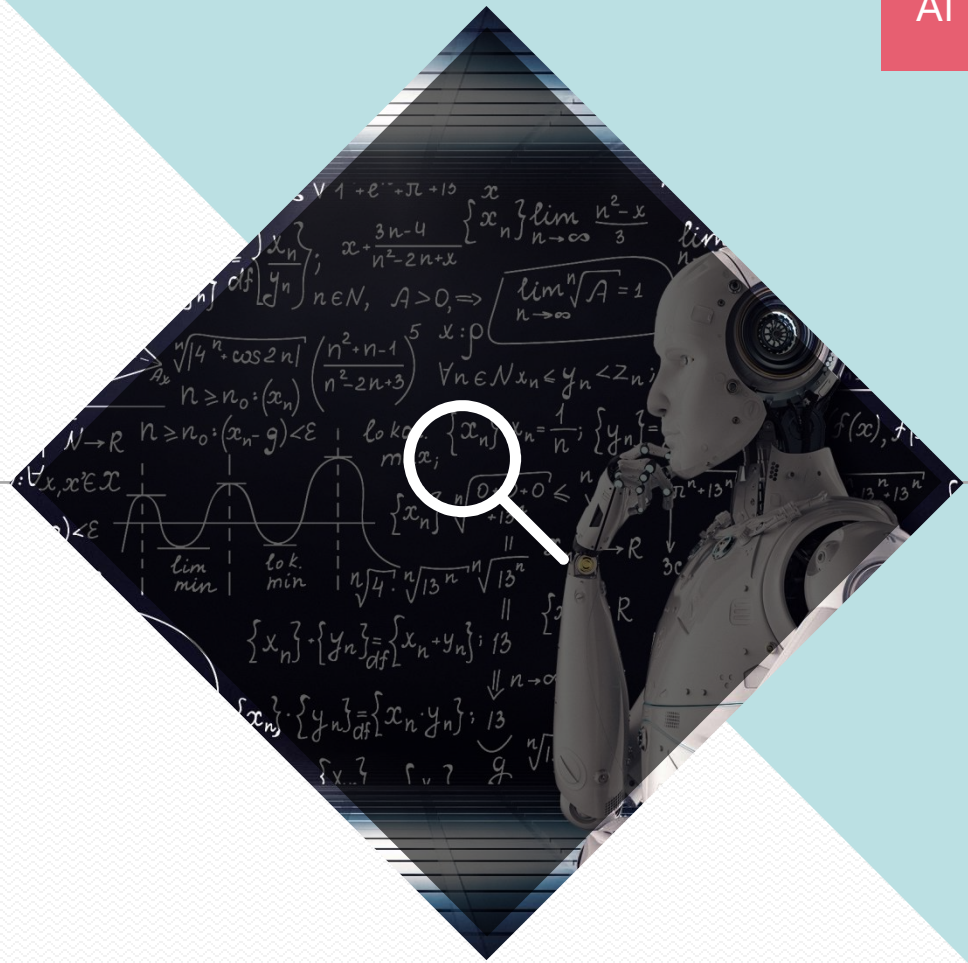
AI TRAINING SERIES



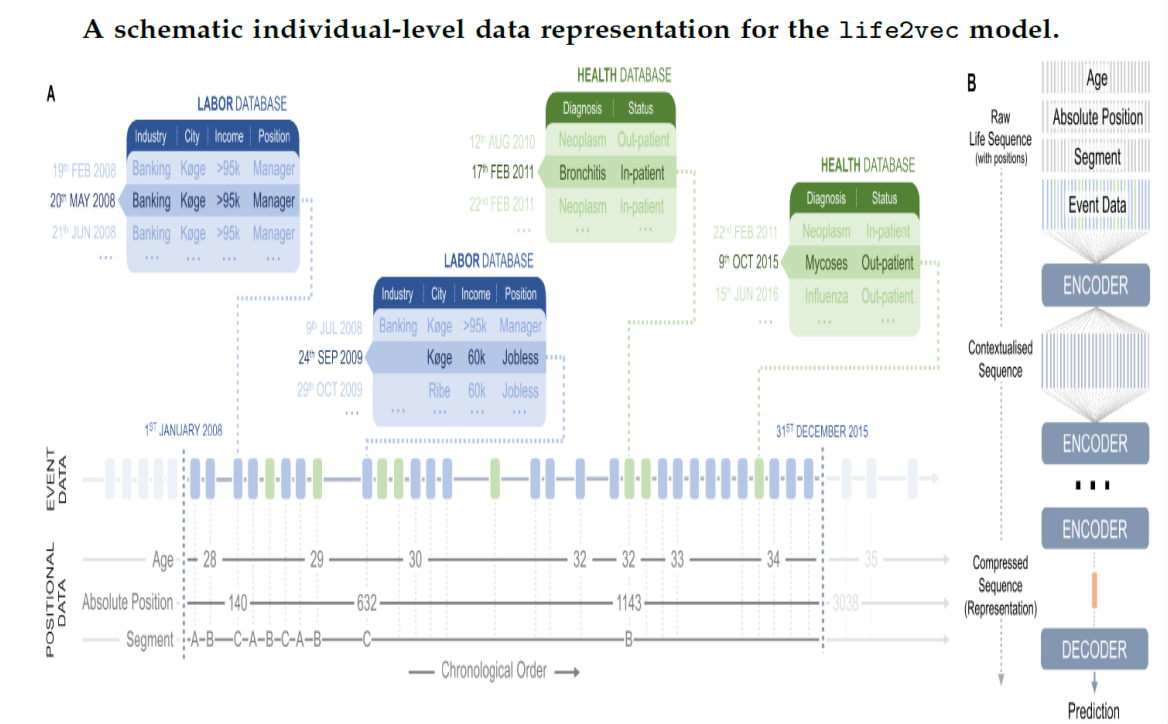
LLM Large Language Model

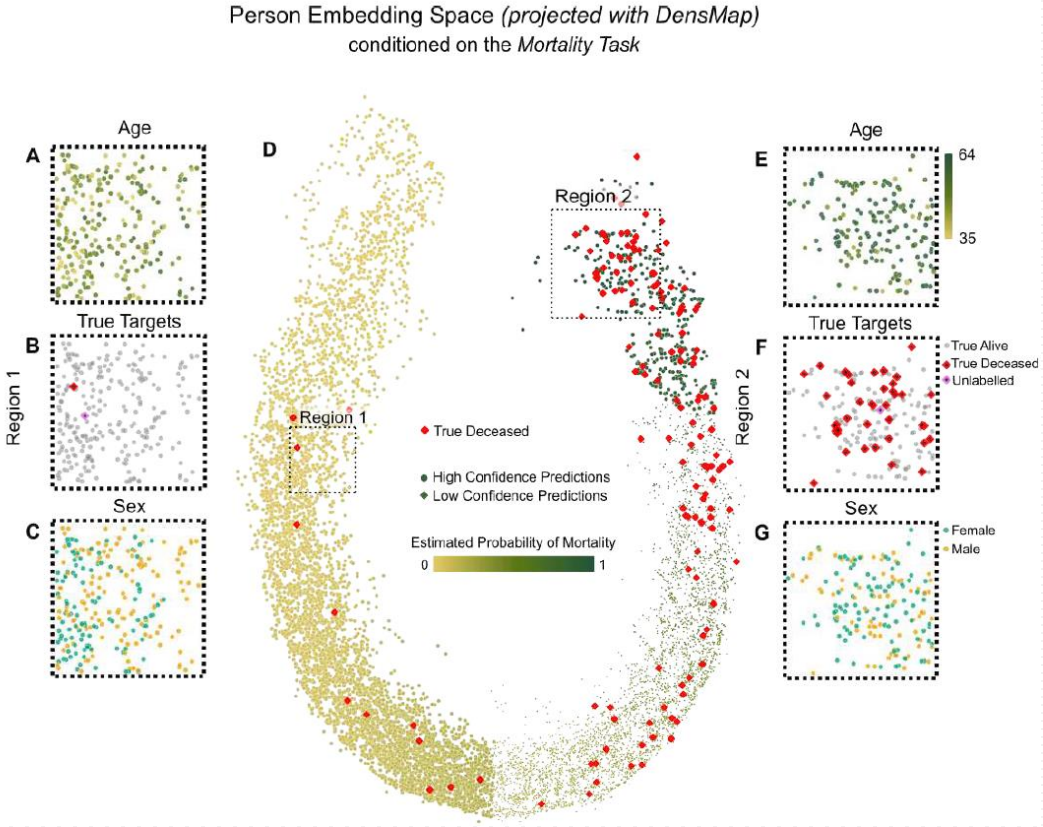
What is Large Language Model? How Does LLM Work?

Contents

AI 타임스

"LLM으로 사람의 미래까지 예측 가능"...덴마크공대, 미래 생성하는 모델 개발





[자율주행 반대 시위자가 라바콘을 임의로 차에 올려놓는 경우]

[뛰어든 개 피하기 위해 뒤따라오던 차량과 충돌]

[자율주행 차량의 예측불가능한 상황의 사고]

미국 캘리포니아 DMV(교통국)에서 제공하는 **165건의 자율주행차 실사고 데이터인 DMV Collision Report 분석 결과**

자율주행차량/일반차량 충돌 형태가 **총 327,502개의 사고 연관규칙 도출. 이는 결론적으로 기존 학습방법으로 모든 사고 예측이 어려움**

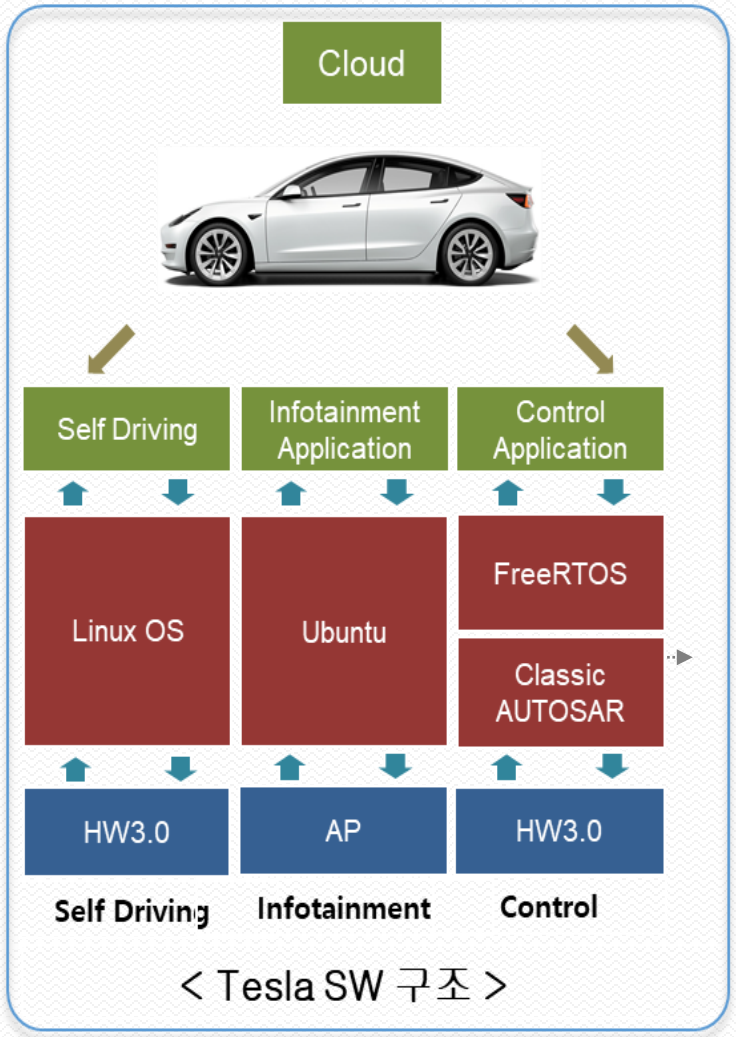
따라서 **새로운 환경이나 예기치 못한 사고**에 대해서는 자율주행차량과 자율협력주행 관제센터에서 **스스로 예측하고 대응할 수 있는 상식 추론 모델의 개발이 필요**

자율주행 소프트웨어에 사람의 언어를 학습시켜 스스로 인지·판단·제어할 수 있는 모델

Vision 자율주행 소프트웨어	Language 초거대AI언어모델	Action 전이학습/미세조정
■ 자율주행 차량SW 모델 적용	■ 상식 추론을 위한 LLM 모델	■ Prompt Engineering 기반 전이학습/미세조정
1) Base Model ① 해외 • RT-2(Google) • NVIDIA • 모빌아이 ② 국내 • 오토노머스에이투지 • ETRI 인지/판단/제어 SW 2) 신뢰성 지원(Illusion:착시) ① 적대적 공격에 대한 강건한 모델 구축	1) Base Model ① 해외 • ChatGPT(OPEN AI) • PaLM-E(Google) • LLaMA2(Meta) ② 국내 • ETRI 엑소브레인 2) 정확성 지원(Hallucination:환각) ① LLM 환각 예방을 위한 지식그래프 모델 구축	1) Prompt Engineering ① LangChain Python Framework 활용 개발 ② 영상과 텍스트 통합 Cross Attention 모델 적용 2) 전이학습/미세조정 (Hallusion : 착시+환각) ① 도로교통법, 자동차관리법 ② 교통안전공단 • 자율주행사고 분석데이터 ③ 자율주행차 실사고 데이터 DMV collision report

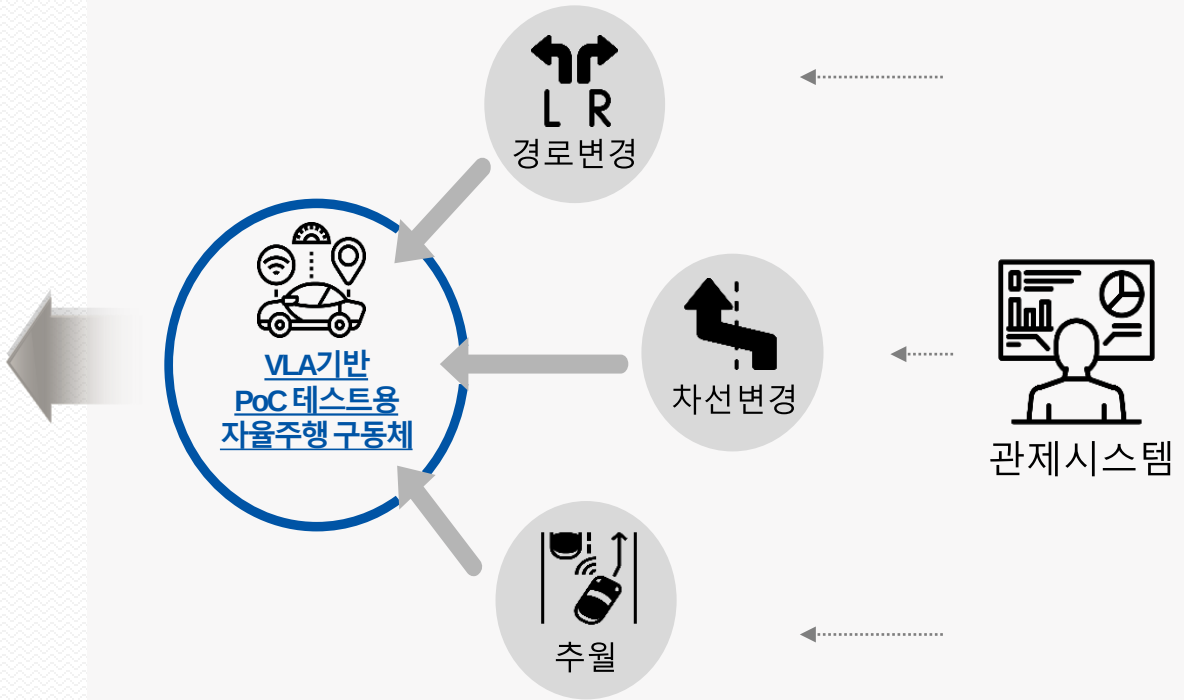
자율주행 차량 플랫폼이 기존 복잡하고 어려운 AUTOSAR에서 개발자가 풍부한 범용 플랫폼으로 전환 중.

구축 장비 구성 SW (구성 사례)



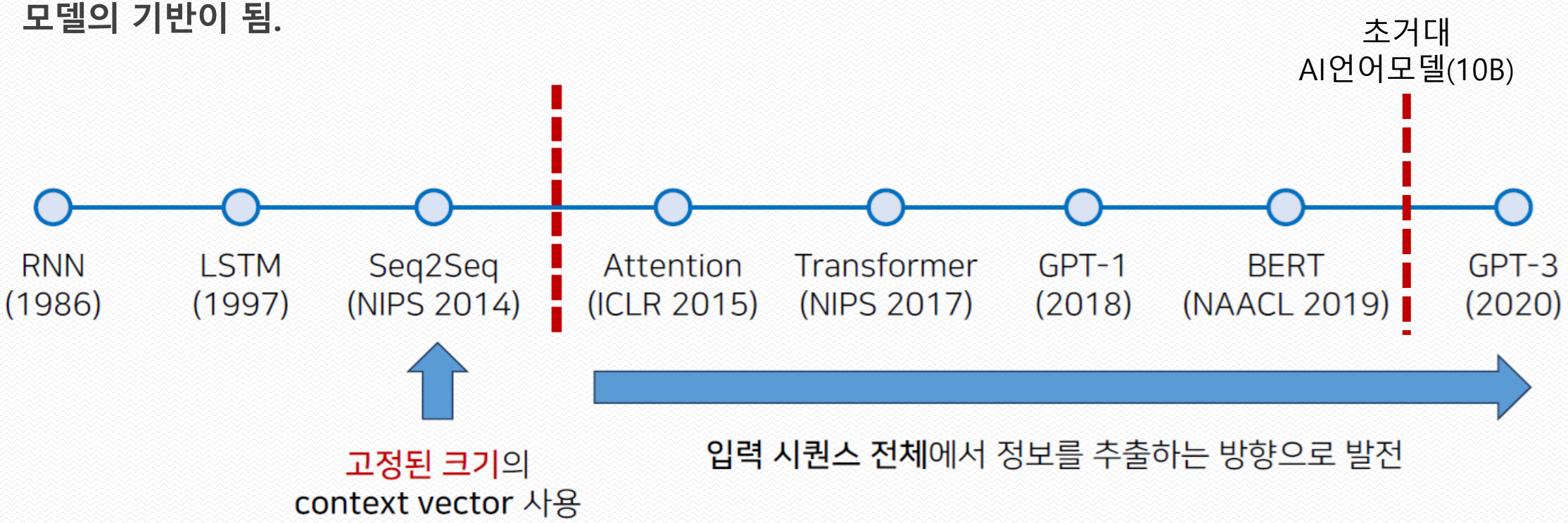
< Tesla SW 구조 >

장비 구축 활용 계획

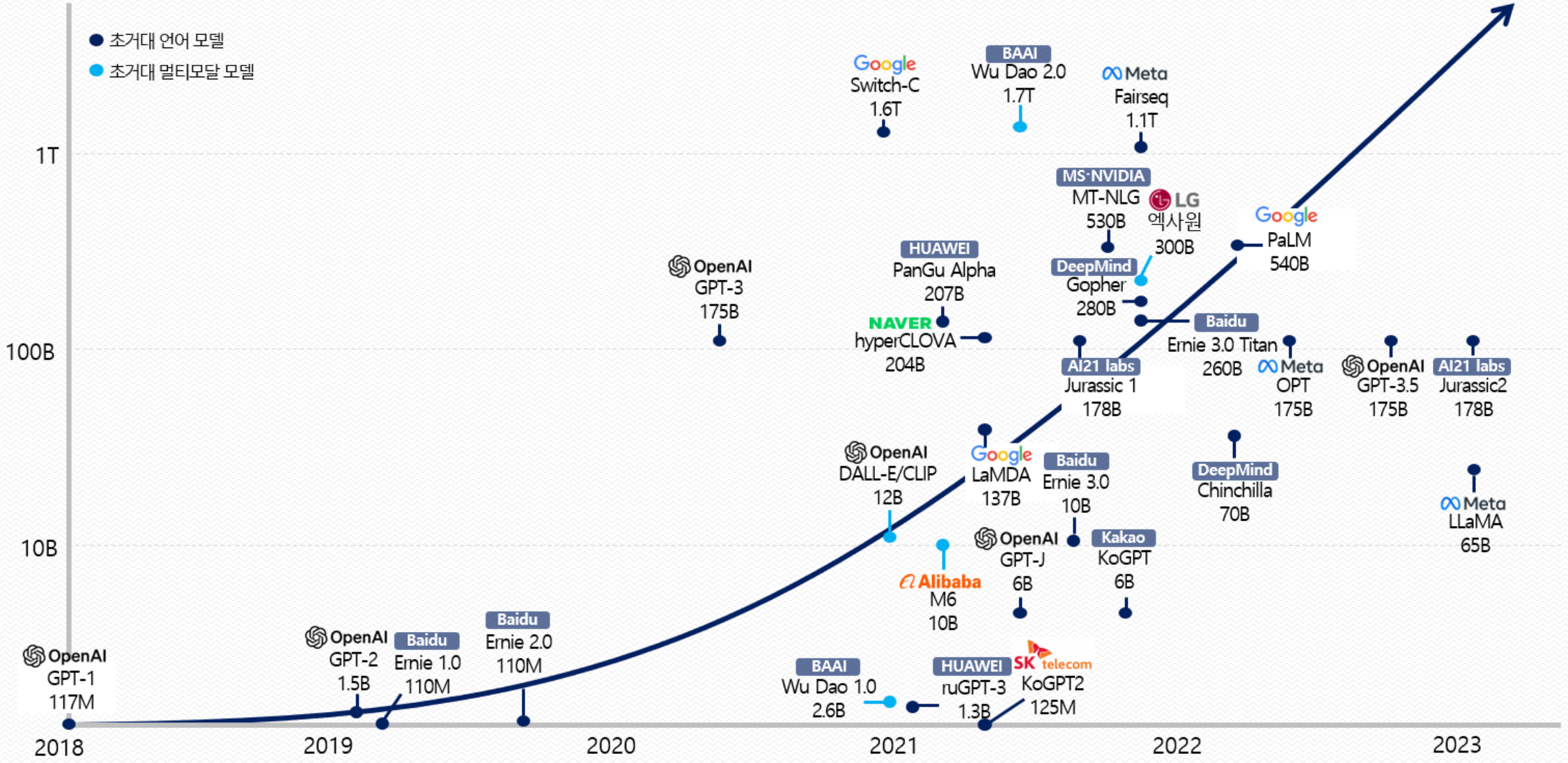


언어 모델의 발전 과정

긴 문장의 이해와 대용량 데이터에 대한 복잡도와 연산속도를 병렬로 처리하여 초거대 AI 모델의 기반이 됨.

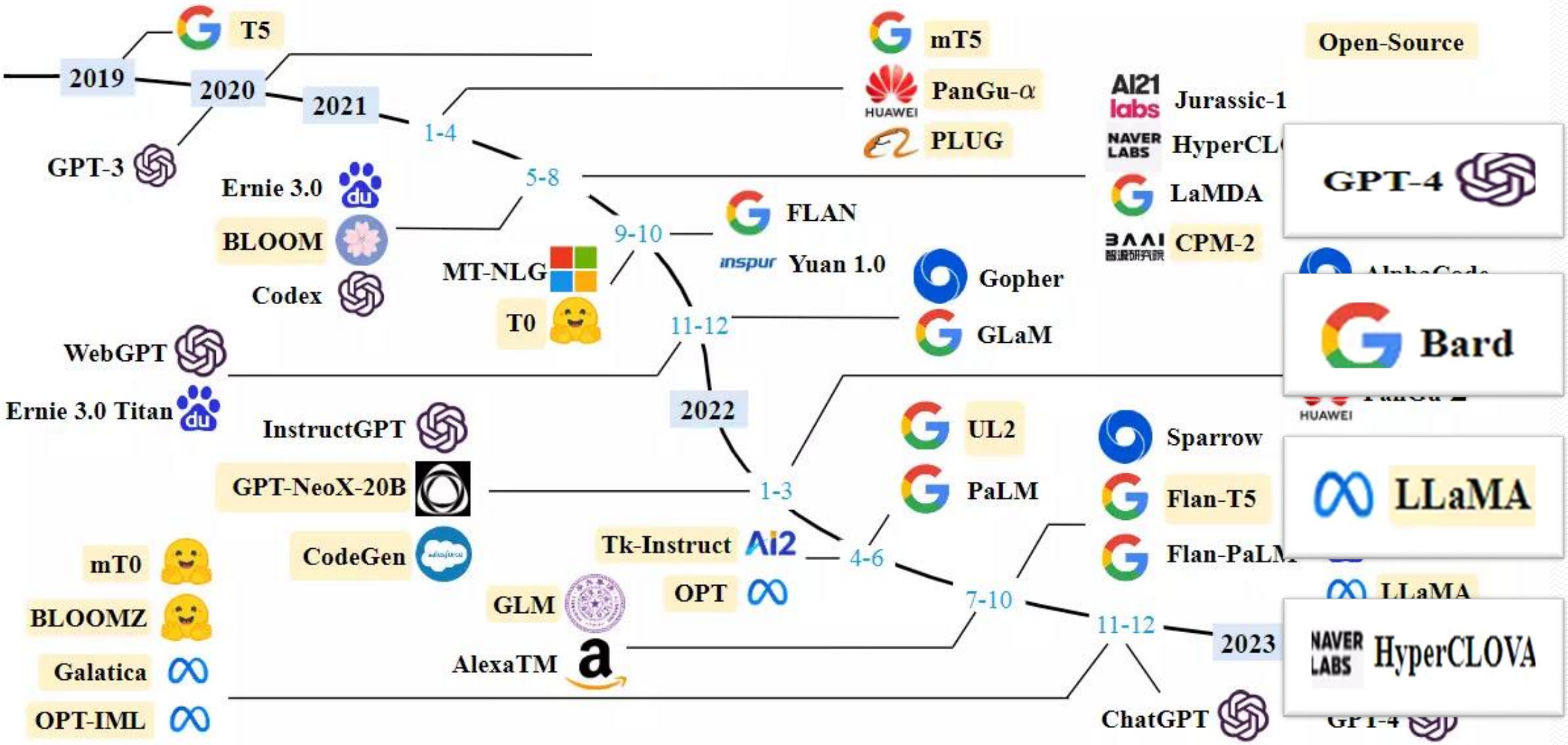


LLM Main models



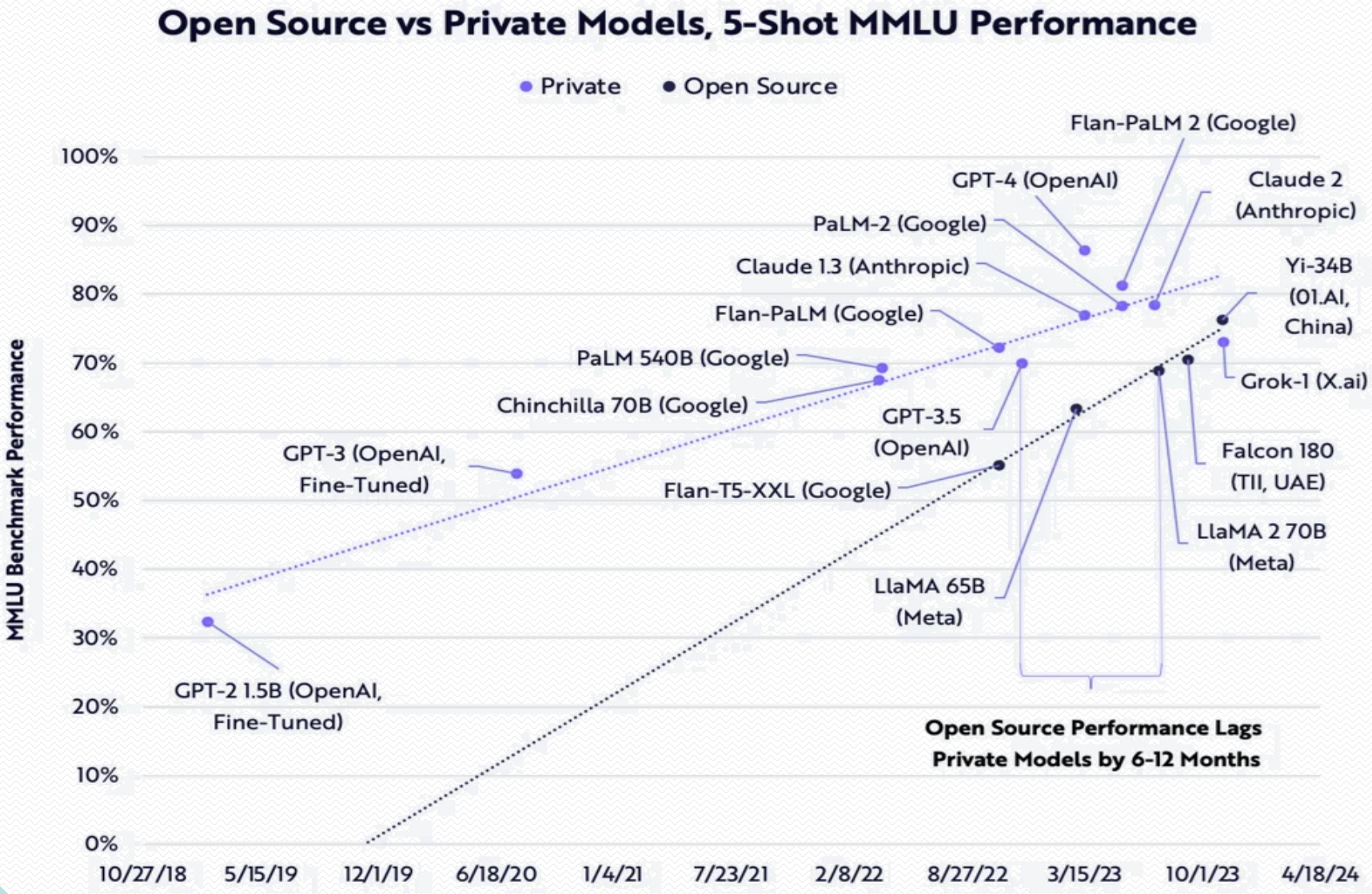
(출처 : https://eiec.kdi.re.kr/publish/reviewView.do?idx=143&fcode=000020003600003&ridx=&pp=10&pg=1)

LLM Main models



A timeline of exisiting large language models (having a size larger than 10B) in recent years. We mark the open source LLMs in yellow color

오픈 소스와 프라이빗 모델의 비교



GPT-4 : 86.5%

Flan-PaLM- 구글의 의료 전문 모델 : **81.2%**

앤트로픽 **'클로드 2 '** : **76.5%**

메타 **'라마2 70B ' : 68.9%**

UAE **'팰컨 180B'** : **70.4%**

일론 머스크 Xai **'그록 '** : **73%**

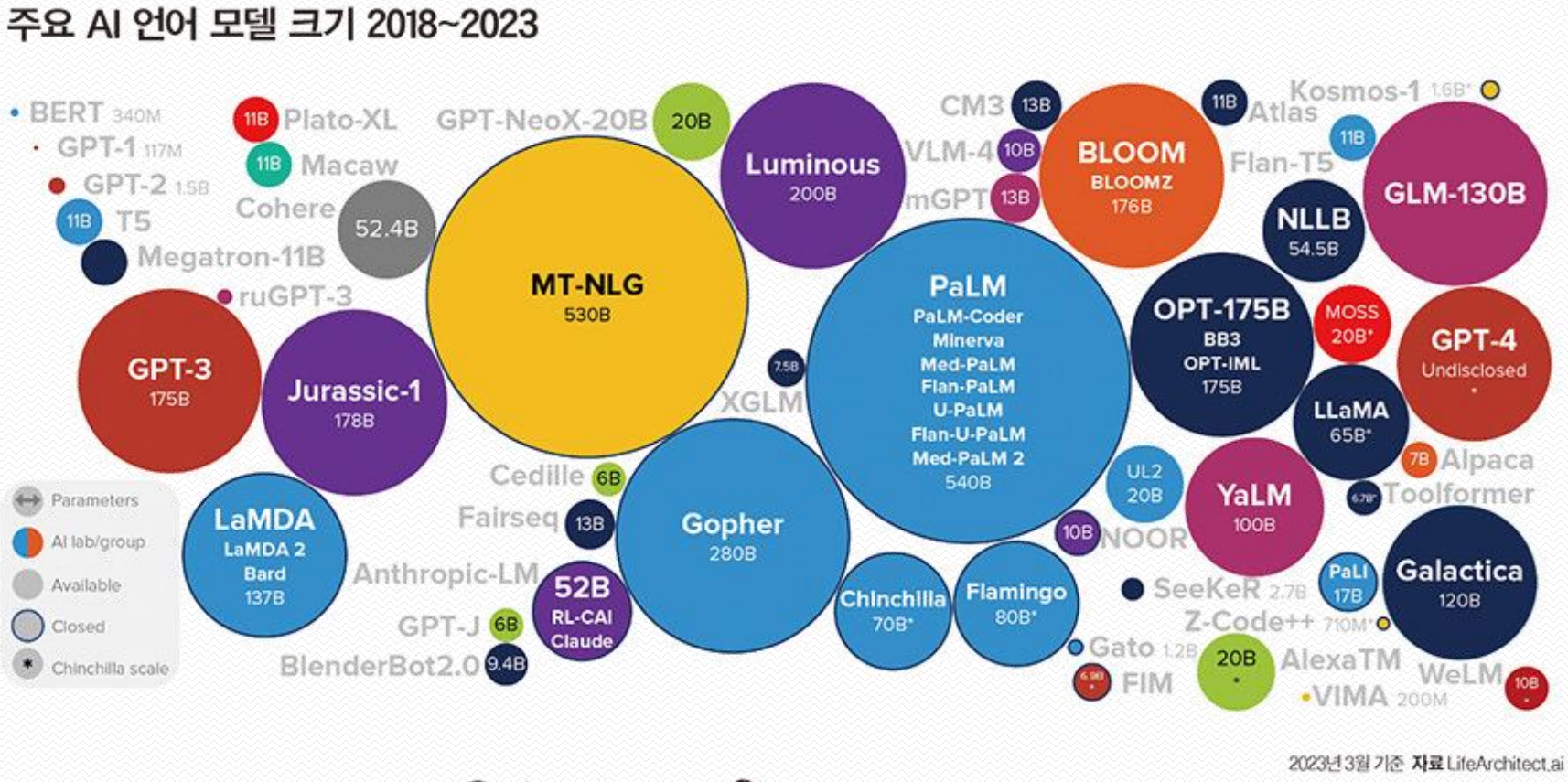
중국 **01.AI : 76.3%**

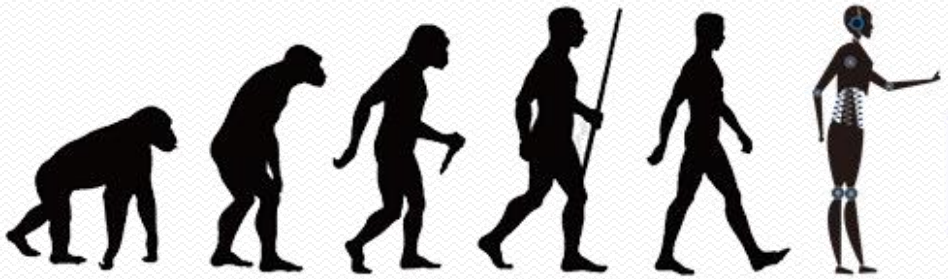
미스트랄 미쿠 70B(miqu-1-70b), 'GPT-4' 성능에 근접

둘의 차이는 이제 불과 몇 개월 차이로 좁혀졌다

MMLU(대규모 멀티태스크 언어 이해) (출처 : https://www.aitimes.com/news/articleView.html?idxno=155196)

LLM Main models





인류가 출판한 총 서적의 단어수
6조5000억 개

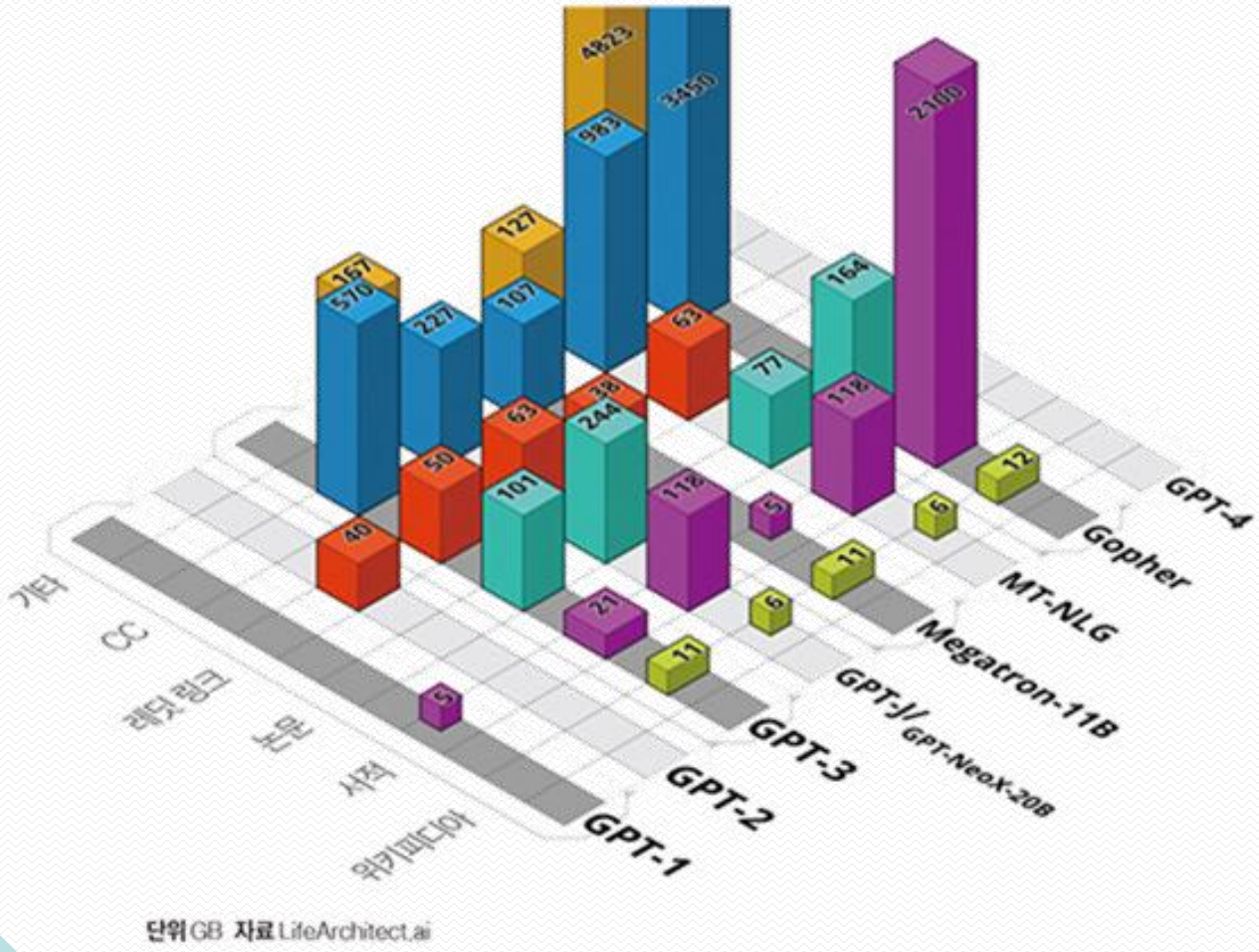
=

챗GPT의
14일 생성 단어수

(출처 : https://eiec.kdi.re.kr/publish/reviewView.do?idx=143&fcode=000020003600003&ridx=&pp=10&pg=1)

LLM Main models

AI 언어 모델별 훈련 데이터세트 크기와 주요 출처



단위 GB 자료 LifeArchitect.ai

구글특허	0.48
뉴욕타임스	0.06
LA타임스	0.06
가디언	0.06
과학공공도서관	0.06
포브스	0.05
허핑턴 포스트	0.05
Patents.com	0.05
스크립드	0.04
기타	99.09

일반 웹문서 크롤

구글	3.4
아카이브	1.3
블로그스팟	1.0
깃허브	0.9
뉴욕타임스	0.7
워드프레스	0.7
워싱턴포스트	0.7
위키아	0.7
BBC	0.7
기타	89.9

레딧 링크

인물정보	27.8
지리학	17.7
문화예술	15.8
역사	9.9
생물학·의약학	7.8
스포츠	6.5
비즈니스	4.8
사회학	4.4
과학·수학	3.5
교육학	1.8

영어 위키피디아

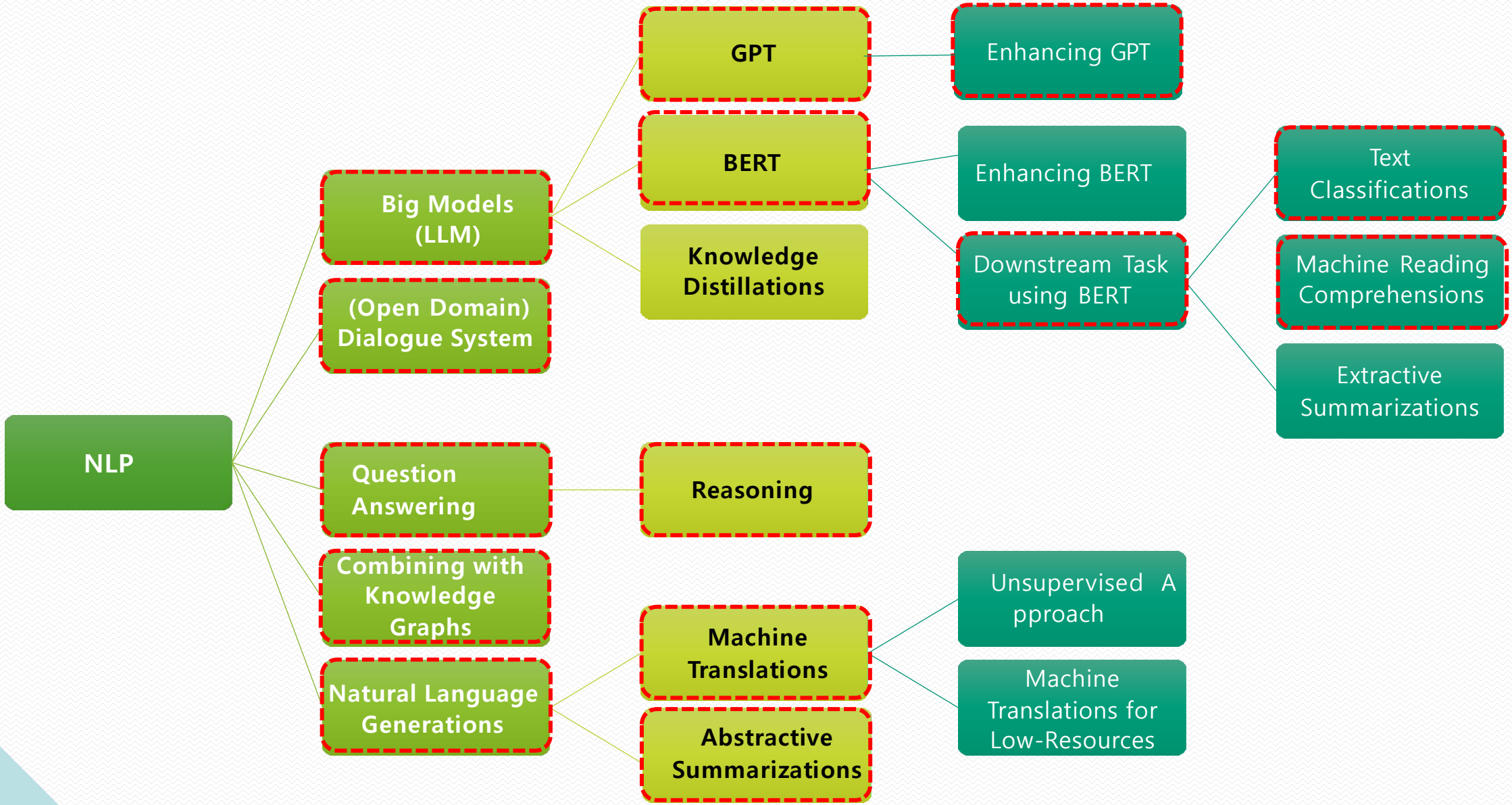
로맨스	26.1
판타지	13.6
공상소설	7.5
뉴 어덜트	6.9
영 어덜트	6.8
스릴러	5.9
미스터리	5.6
뱀파이어	5.4
호러	4.1
기타	18.0

북코퍼스(GPT-1만 해당)

단위 %

(출처 : https://eiec.kdi.re.kr/publish/reviewView.do?idx=143&fcode=000020003600003&ridx=&pp=10&pg=1)

Research 분야


NLP
Big Models (LLM)
(Open Domain) Dialogue System
Question Answering
Combining with Knowledge Graphs
Natural Language Generations
GPT
BERT
Knowledge Distillations
Reasoning
Machine Translations
Abstractive Summarizations
Enhancing GPT
Enhancing BERT
Downstream Task using BERT
Unsupervised Approach
Machine Translations for Low-Resources
Text Classifications
Machine Reading Comprehensions
Extractive Summarizations

Contents

Non-Sequential Data

- **Non-Sequential data : 시간 정보를 포함하여 순차적으로 생성되는 데이터**
  - ✓ 순차 데이터가 아닌 경우 데이터는 N by D 행렬(N : 관측치 수, D : 변수 수)로 표현됨
    - 예) 특정 고객의 금융상품 이용 현황을 바탕으로 대출 상품 추천

      X: 고객별 금융 상품 이용 현황

      Y: 대출 상품 이용 유무(1 : 사용, 0 : 미사용)

	Var 1	Var 2	Var 3	…	Var D
Obs 1					
Obs 2			X		
Obs 3					
…					
…					
…					
…					
Obs N-1					
Obs N					

Y
Y

Sequential Data

- **Sequential Data : 시간 정보를 포함하여 순차적으로 생성되는 데이터**
  - ✓ 순차 데이터의 경우 데이터는 (N) by T by D Tensor (T는 측정 시점 수)로 표현됨
    - 예 1) 특정 설비에서 분 단위로 측정되는 여러 센서 값을 이용한 장비 건강도 추정

      X: 시간별 측정되는 센서 값

      Y: 장비 건강도

Obs 1	Var 1	Var 2	Var 3	…	Var D
Time 1					
Time 2					
Time 3					
…					
…					
…					
…					
Time T-1					
Time T					

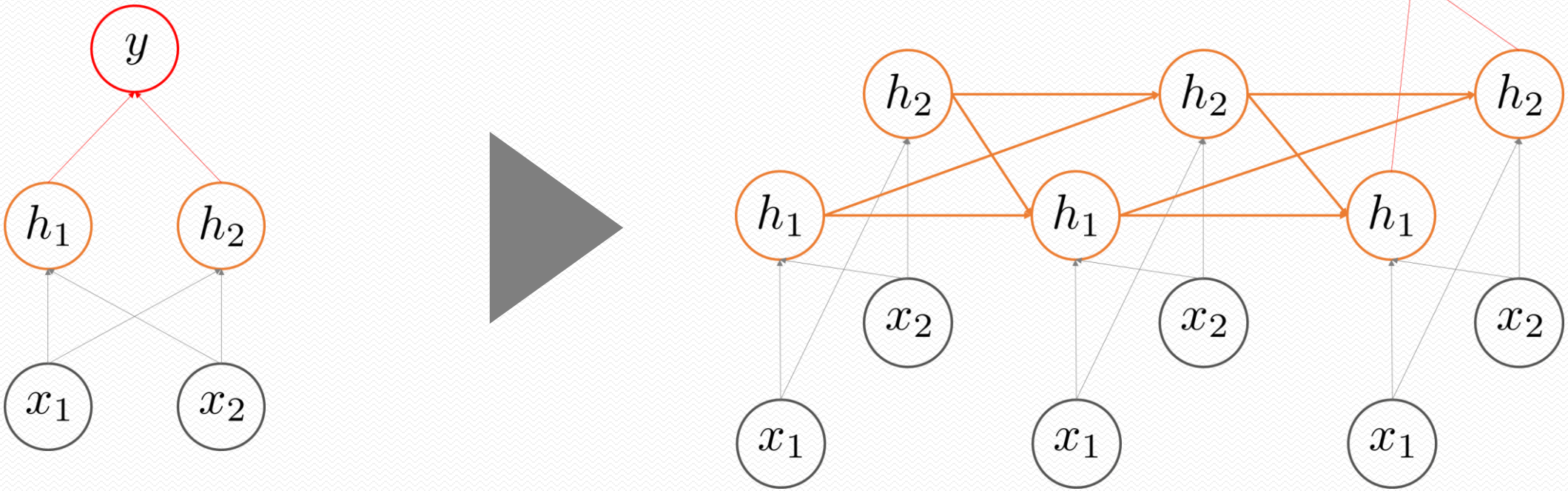
Y
Y

Sequential Data

피드포워드(feed forword) 신경망

- 흐름이 단방향
- 시계열 데이터의 성질(패턴)을 충분히 학습할 수 없음

순환 신경망(Recurrent Neural Network, RNN)의 등장

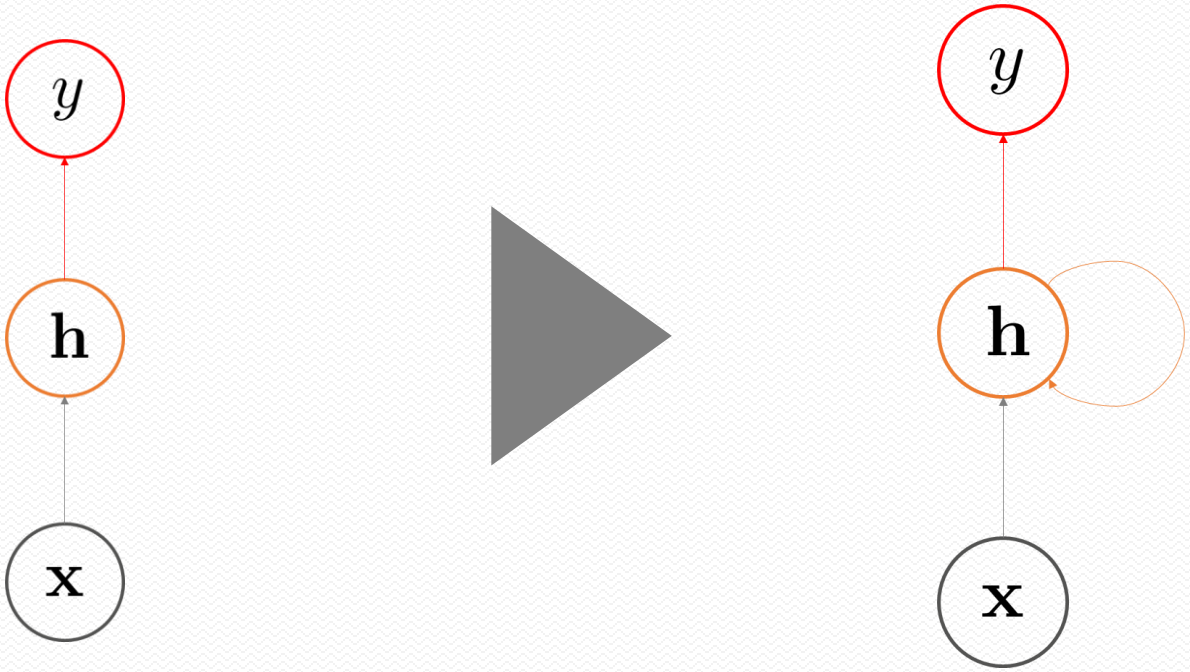


Sequential Data

피드포워드(feed forword) 신경망

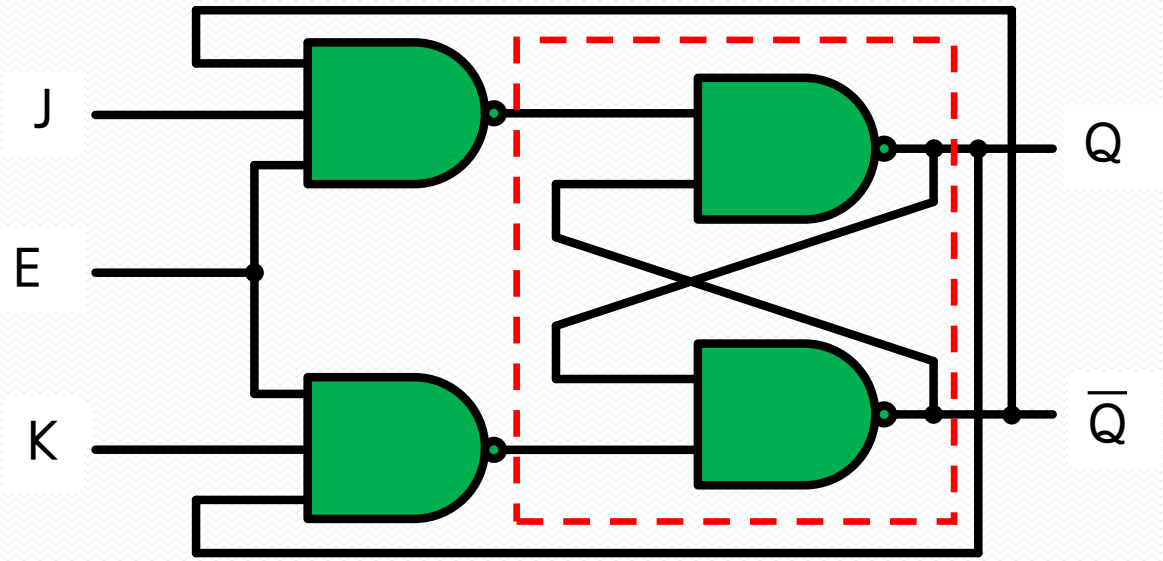
- 흐름이 단방향
- 시계열 데이터의 성질(패턴)을 충분히 학습할 수 없음

순환 신경망(Recurrent Neural Network, RNN)의 등장



Flip-flop (순차회로)

[NAND 게이트를 이용한 J-K 플립플롭]



입력			출력
E	J	K	Q(t+1)
0	×	×	Q(t)(불변)
1	0	0	Q(t)(불변)
1	0	1	0
1	1	0	1
1	1	1	$\overline{Q}$(t)(toggle)

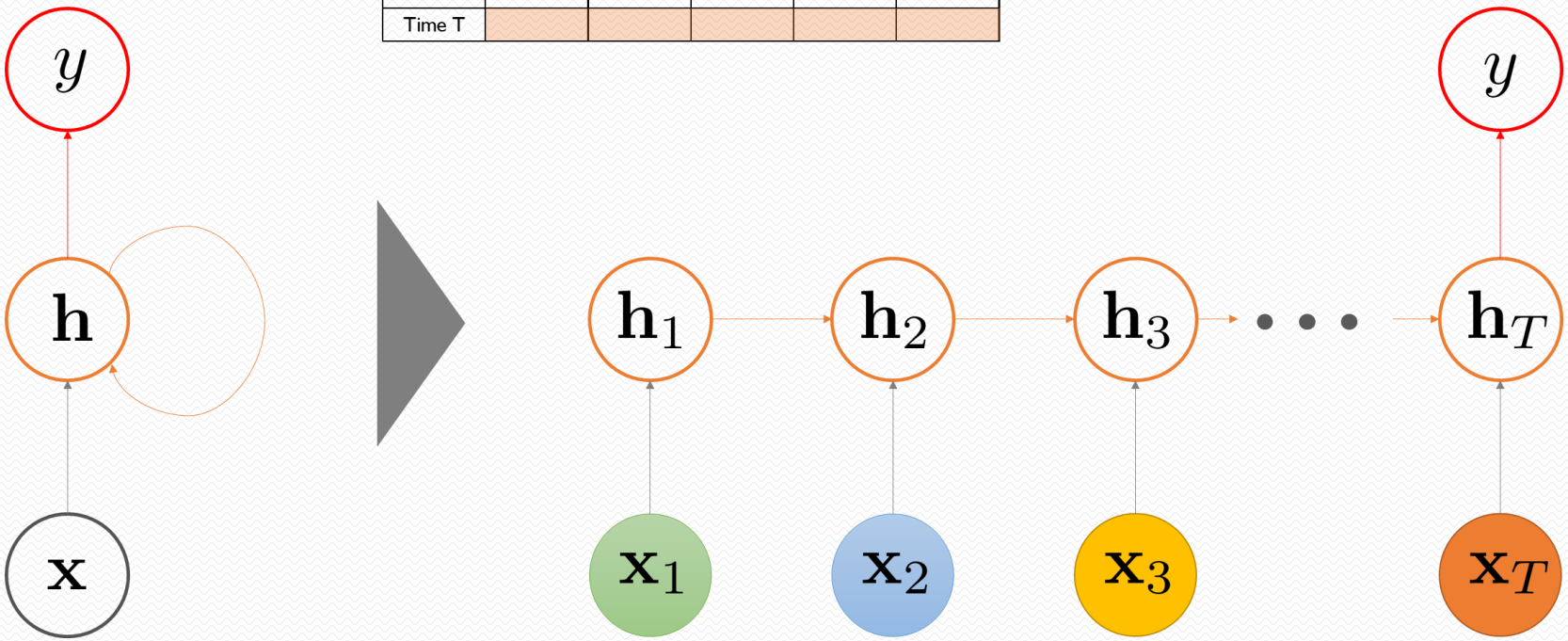
- 순차회로를 이용해 기존의 조합회로에서 구현할 수 없었던 레지스터나 계수기를 구현 가능하게됨.
- RNN을 통해 기존 CNN에서 한계를 드러내었던 시계열 분석에 대한 성능향상이 진행됨.

Sequential Data

- 순서(sequence)가 있는 인공신경망 구조(벡터 표현)

Obs 1	Var 1	Var 2	Var 3	…	Var D
Time 1					
Time 2					
Time 3					
…					
…					
…					
…					
Time T-1					
Time T					

Y



Sequential Data

- **Sequential Data : 시간 정보를 포함하여 순차적으로 생성되는 데이터**
  - ✓ 순차 데이터의 경우 데이터는 (N) by T by D Tensor (T는 측정 시점 수)로 표현됨
    - 예 2) 설비 가동 정상상태 모니터링

      X: 특정 시점에서의 측정되는 센서들의 값

      Y: 다음 시점에서의 동일한 센서들의 값

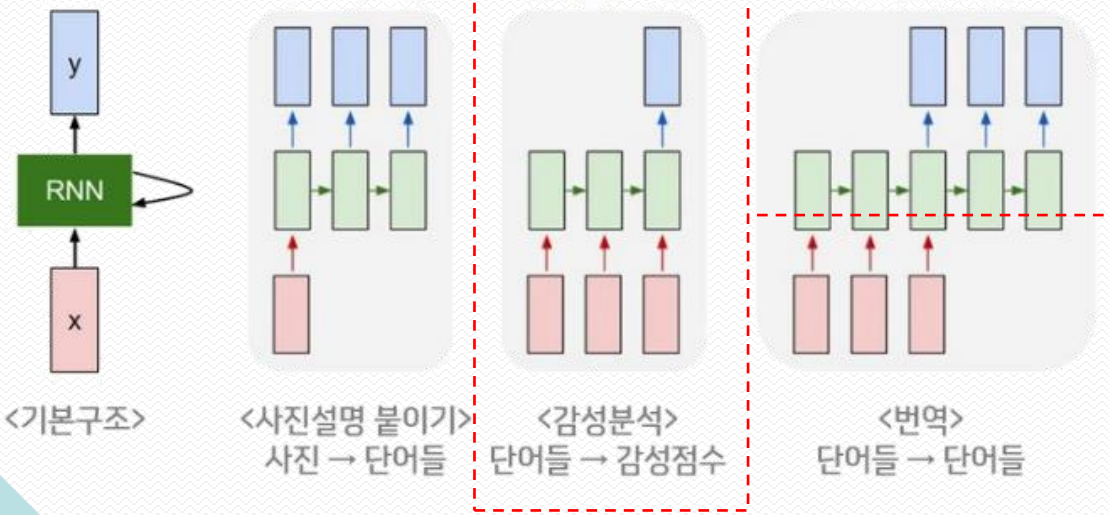
Obs I	Var I	Var 2	Var 3	…	Var D
Time I			X		
Time 2			Y		
Time 3					
…					
…					
…					
…					
Time T-I					
Time T					

순환 신경망(RNN)

시간 개념을 반영할 수 있는 인공 신경망

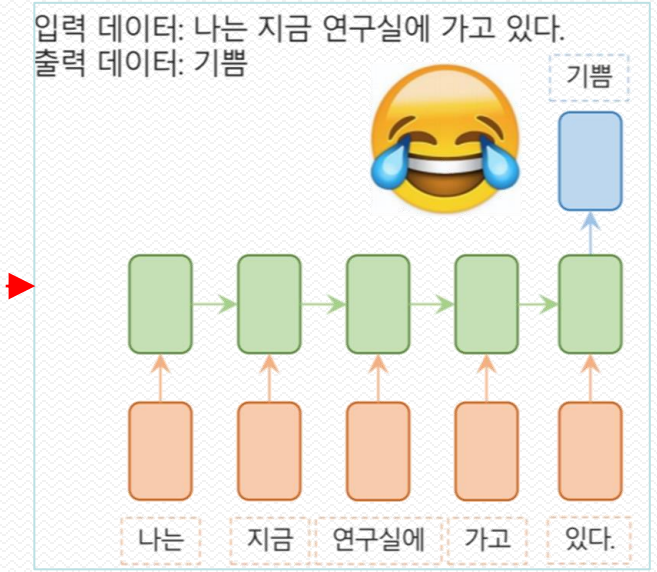
- RNN은 시퀀스 데이터를 처리하기 위해 설계된 신경망으로, 자연어 처리에서 특히 유용
- 시퀀스 데이터는 단어의 시퀀스인 문장, 시간에 따라 변화하는 주식 가격 등과 같이 순서에 의미가 있는 데이터
- RNN은 이전에 처리한 정보를 기억하고, 그 정보를 현재의 결정에 사용할 수 있는 '상태 벡터'를 사용하여 정보를 시간에 따라 전달

RNN기본 구조



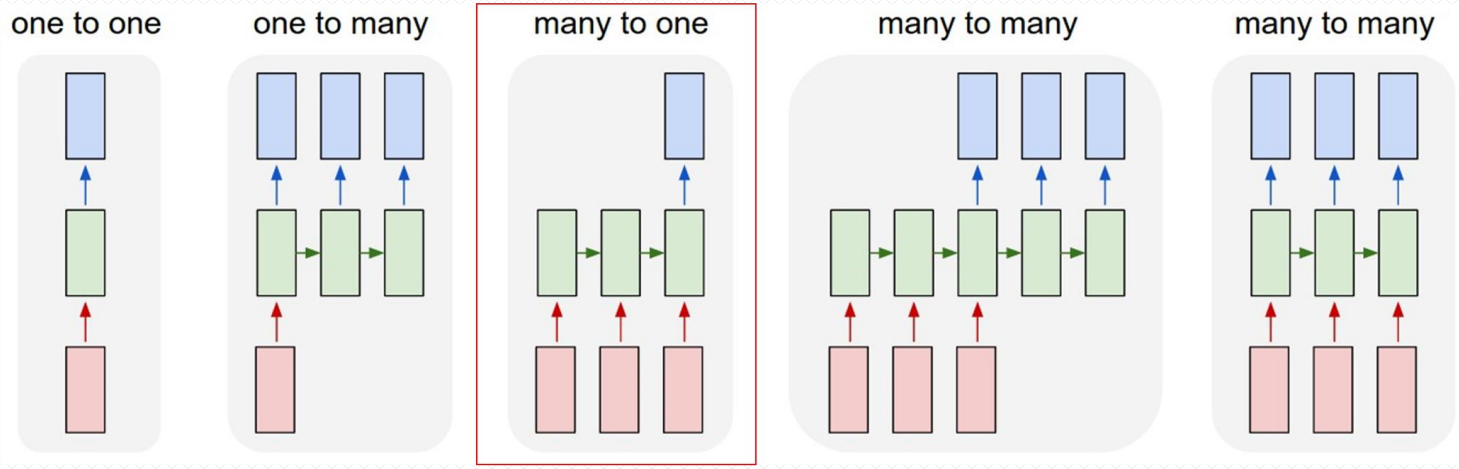
<기본구조>
<사진설명 붙이기>
사진 → 단어들
<감성분석>
단어들 → 감성점수
<번역>
단어들 → 단어들

감성분석 예시

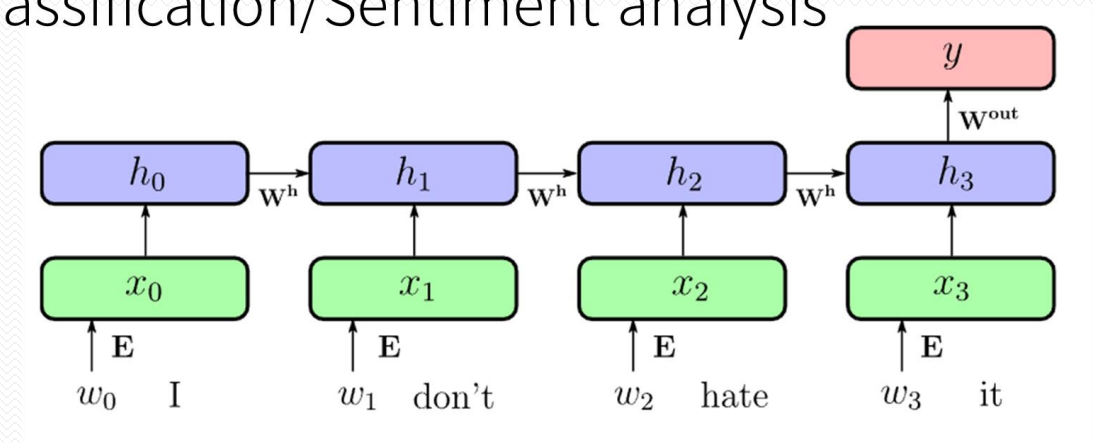


RNN structure

- **순서(sequence)가 있는 인공신경망 구조**
  - ✓ 입력-출력에 따른 활용 사례 : Many-to-One



Text classification/Sentiment analysis



RNN structure

- **순서(sequence)가 있는 인공신경망 구조**
  - ✓ 입력-출력에 따른 활용 사례 : One-to-Many

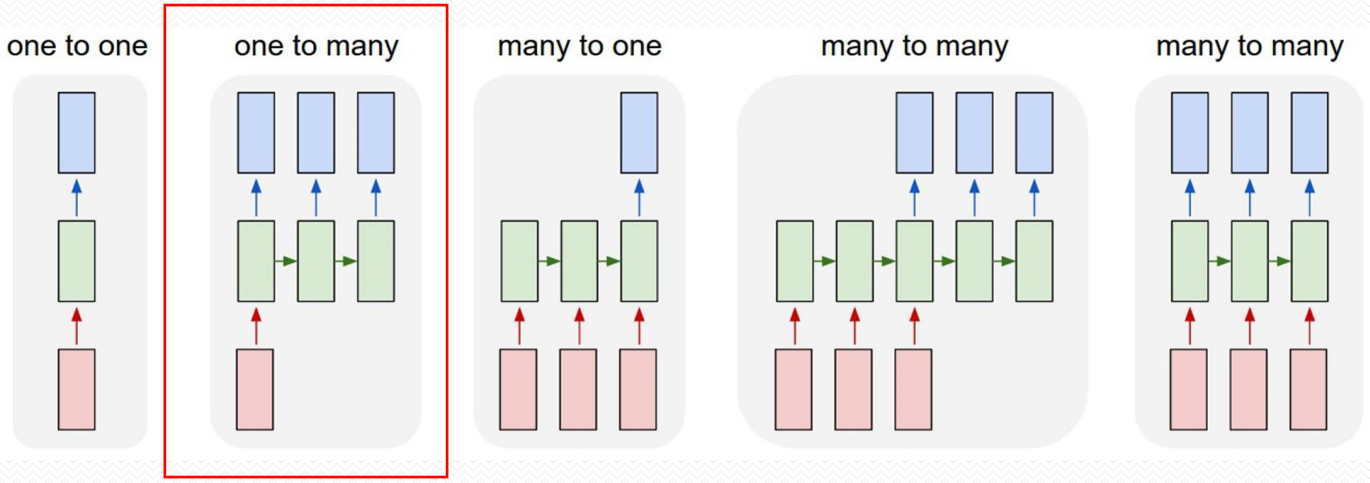
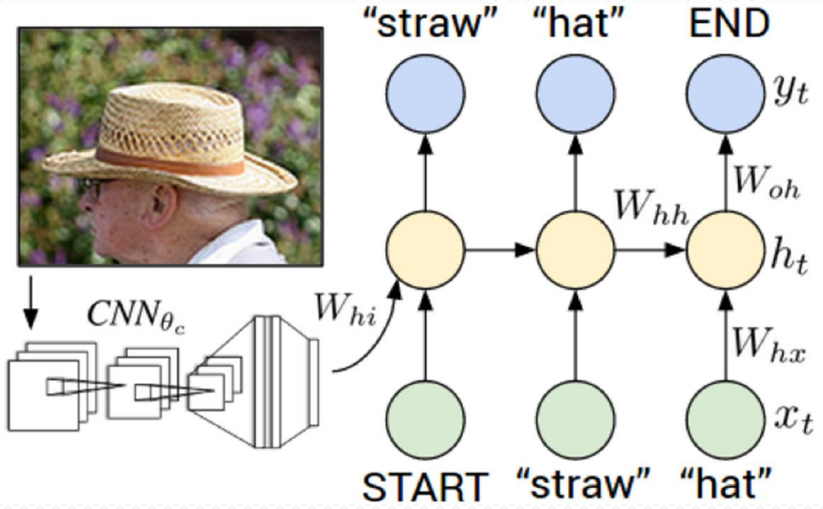
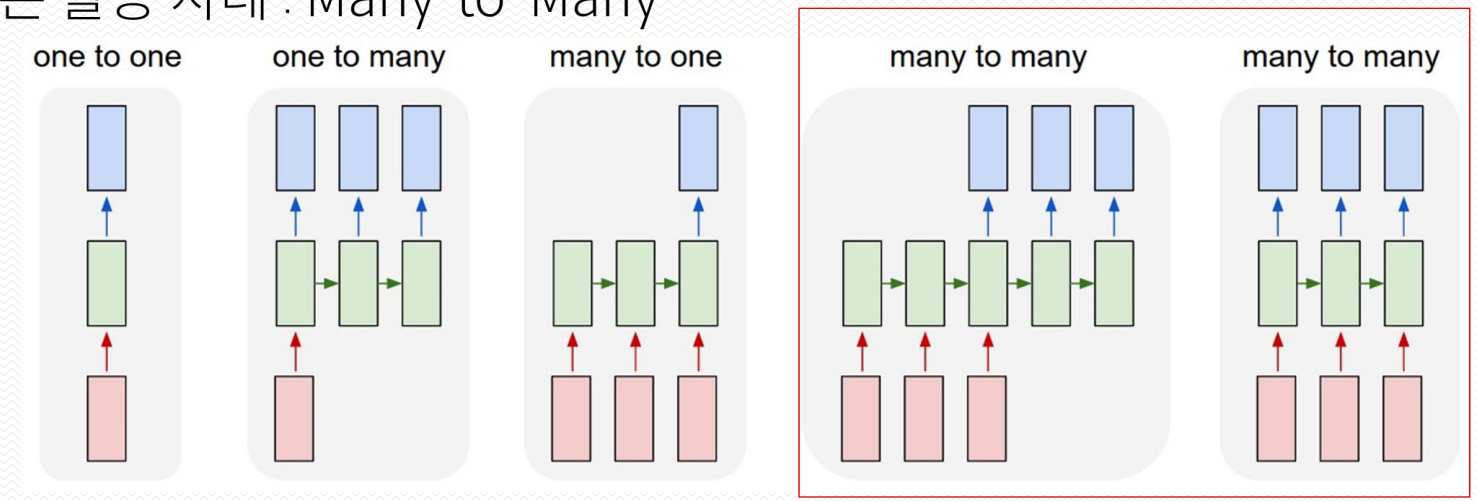


Image captioning

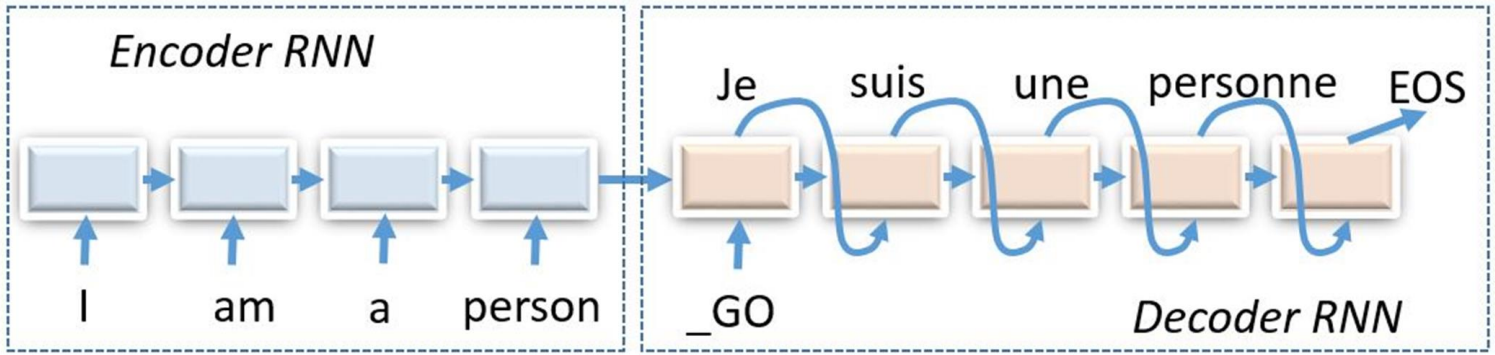


RNN structure

- **순서(sequence)가 있는 인공신경망 구조**
  - ✓ 입력-출력에 따른 활용 사례 : Many-to-Many



Machine translation/Dialog system

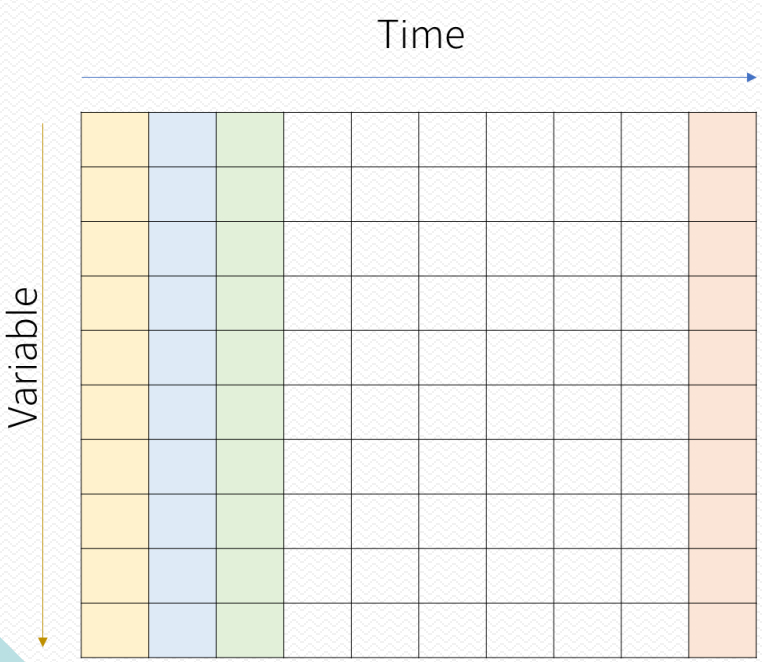


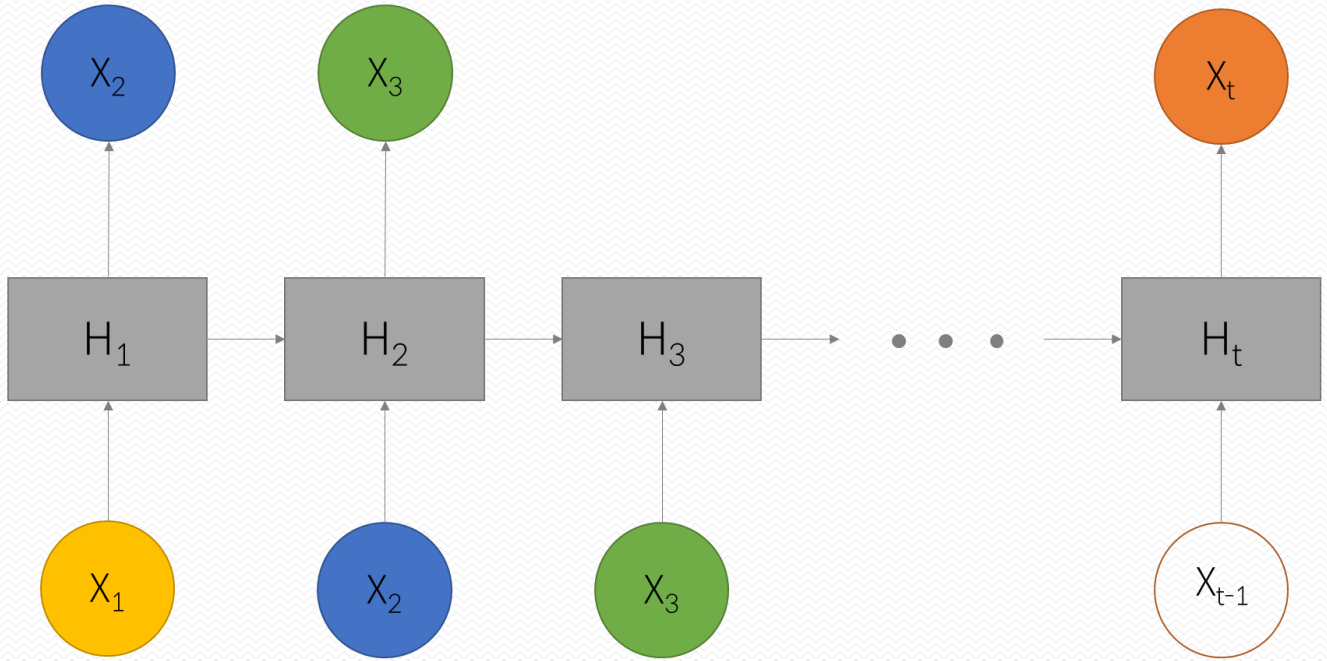
https://esciencegroup.com/2016/03/04/fun-with-recurrent-neural-nets-one-more-dive-into-cntk-and-tensorflow/

RNN structure

- **순서(sequence)가 있는 인공신경망 구조**
  - ✓ 입력-출력에 따른 활용 사례 : Many-to-Many

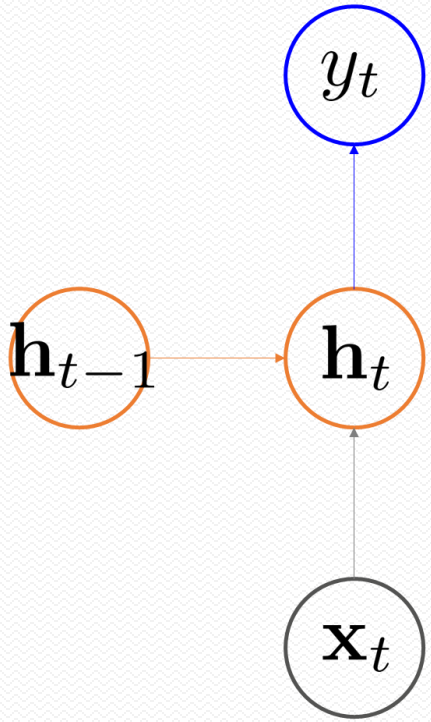
$$\text{Abnormal Score} = \sum_{i=2}^{t} ||\hat{\mathbf{x}}_i - \mathbf{x}_i||_2$$





RNN Basics : Forward Path

- **기본 RNN(Vanilla RNN) 구조에서 정보의 흐름**
  - ✓ f( )와 g( )는 활성 함수



$$y_t = g(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y)$$

$$\mathbf{h}_t = f(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_x)$$

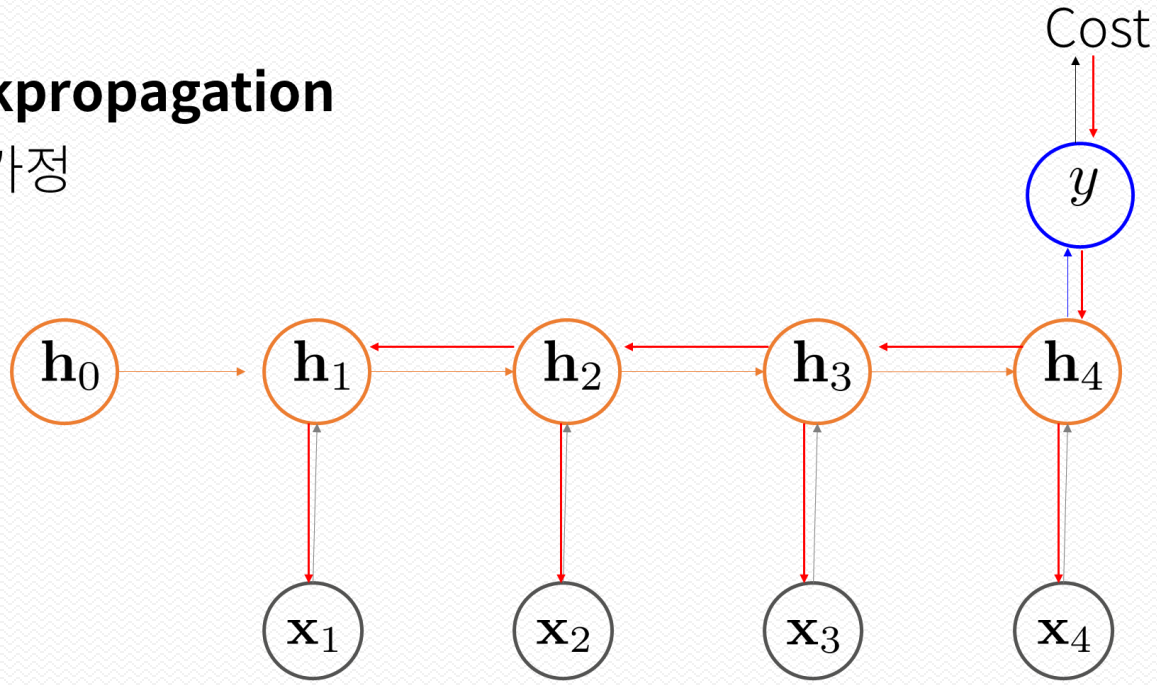
t시점에서의 은닉 노드의 값은

t-1 시점까지 은닉 노드에 저장된 정보와

t 시점에서 새롭게 제공되는 입력 정보에 영향을 받는다

RNN Basics : Gradient Vanishing/Exploding Problem

- **RNN에서의 Backpropagation**
  - ✓ f( )는 Tanh로 가정



$$\frac{\partial Cost}{\partial \mathbf{W}_{xh}} = \frac{\partial Cost}{\partial y} \times \frac{\partial y}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{W}_{xh}} + \frac{\partial Cost}{\partial y} \times \frac{\partial y}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{W}_{xh}}$$

$$+ \frac{\partial Cost}{\partial y} \times \frac{\partial y}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \times \frac{\partial \mathbf{h}_2}{\partial \mathbf{W}_{xh}}$$

$$+ \frac{\partial Cost}{\partial y} \times \frac{\partial y}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \times \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \times \frac{\partial \mathbf{h}_1}{\partial \mathbf{W}_{xh}}$$

RNN Basics : Gradient Vanishing/Exploding Problem

- **RNN에서의 Backpropagation**
  - ✓ 일반적으로

$$\frac{\partial Cost}{\partial \mathbf{W}_{xh}} = \sum_{i=1}^{n} \left(\frac{\partial Cost}{\partial y} \cdot \frac{\partial y}{\partial \mathbf{h}_n} \cdot \left(\prod_{j=i}^{n-1} \frac{\partial \mathbf{h}_{j+1}}{\partial \mathbf{h}_j} \right) \cdot \frac{\partial \mathbf{h}_i}{\partial \mathbf{W}_{xh}} \right)$$

  - ✓ f( )는 Tanh이고

$$\mathbf{h}_t = f(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_x) = tanh(\mathbf{z}_t)$$

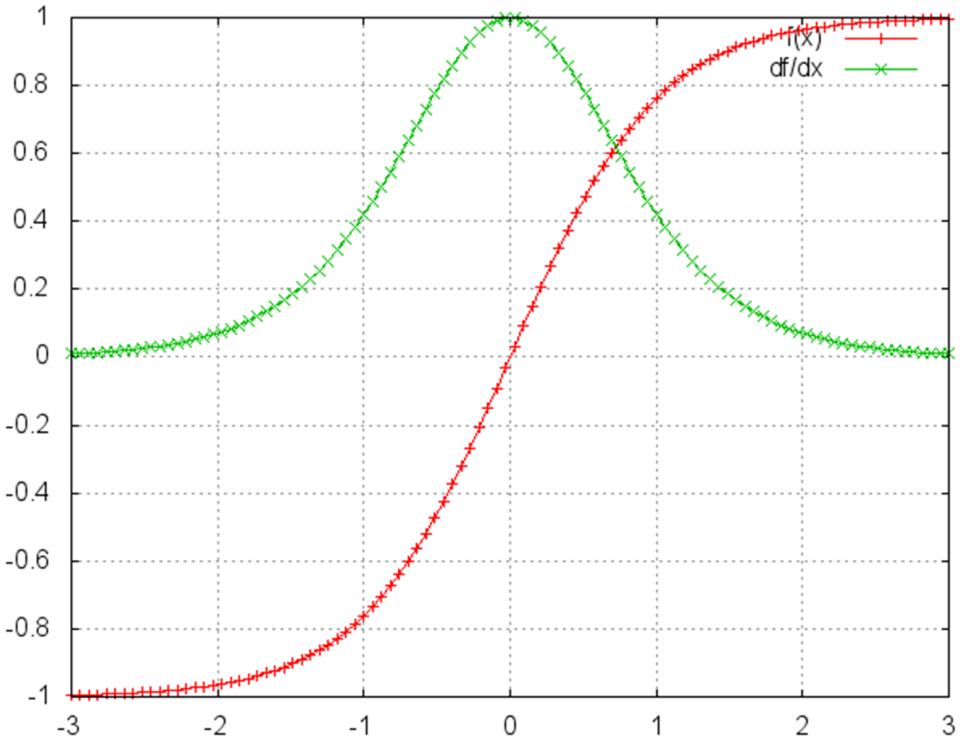
$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \times \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = (1 - tanh^2(\mathbf{z}_t)) \times \mathbf{W}_{hh}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{xh}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \times \frac{\partial \mathbf{z}_t}{\partial \mathbf{W}_{xh}} = (1 - tanh^2(\mathbf{z}_t)) \times \mathbf{x}_t$$

RNN Basics : Gradient Vanishing/Exploding Problem

- **RNN에서의 Backpropagation**

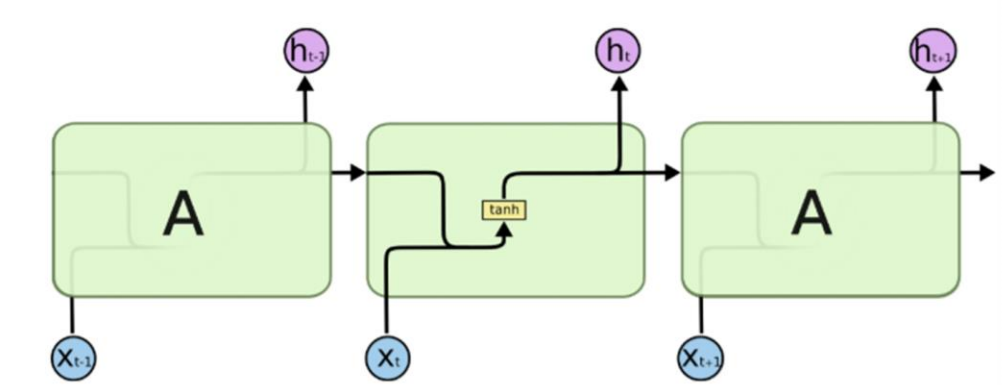
$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \times \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \boxed{(1 - tanh^2(\mathbf{z}_t))} \times \mathbf{W}_{hh}$$



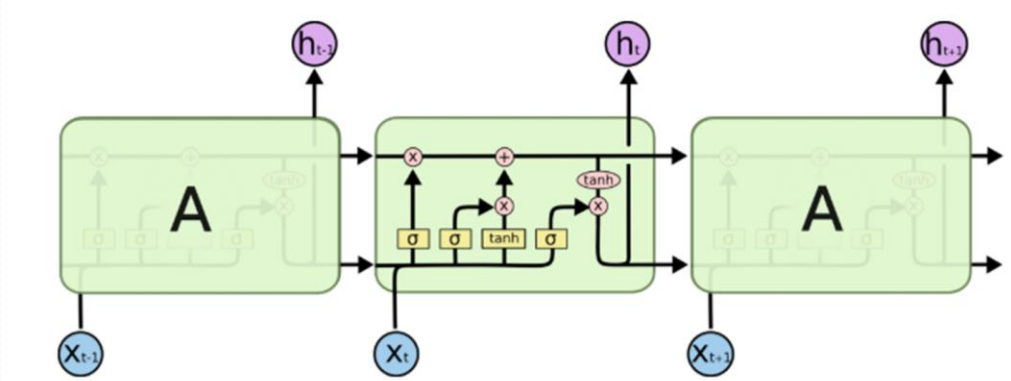
RNN Hidden Unit : LSTM

- **LSTM : Long Short-Term Memory**
  - ✓ Gradient exploding/vanishing 문제를 해결하여 Long-term dependency 학습 가능

Vanilla RNN

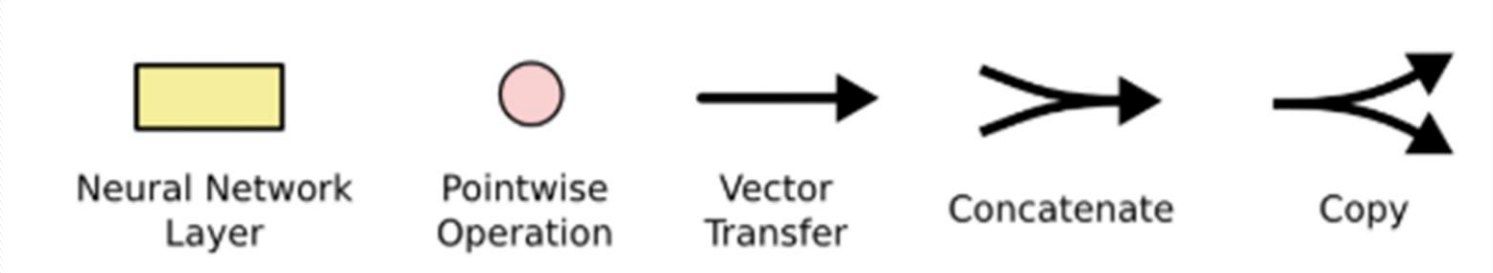


LSTM



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Operation symbols

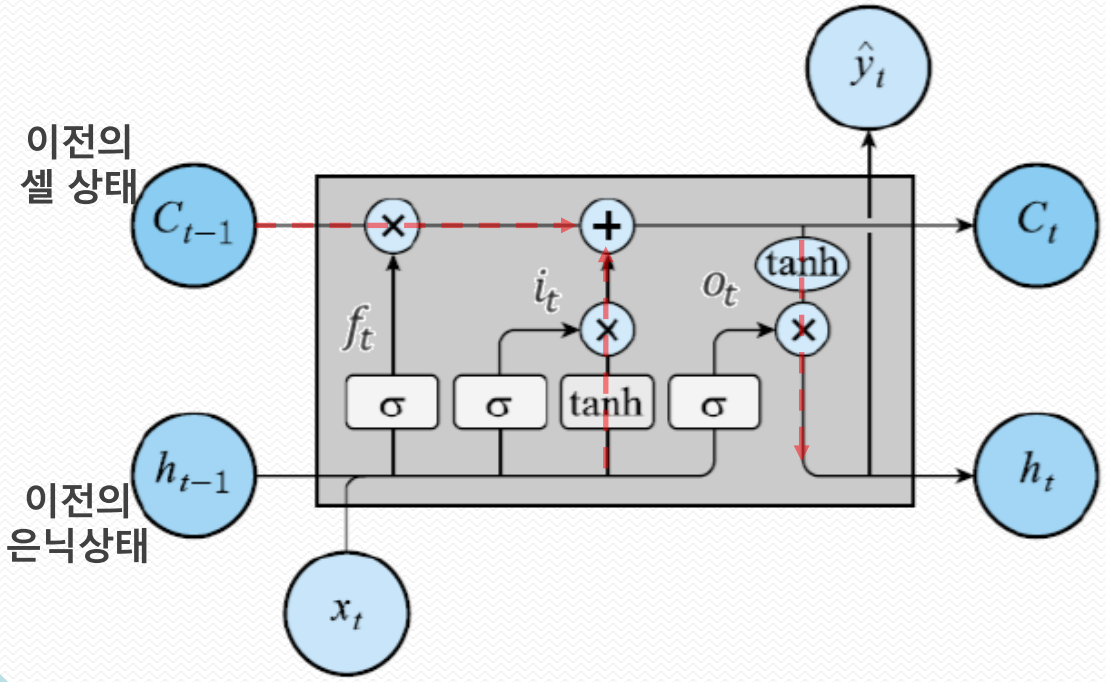


RNN Hidden Unit : LSTM

장기 의존성 문제를 완화한 RNN 개선 모델

- LSTM의 주요 특징은 '게이트(gates)'라는 구조를 도입하여 각 시점(time step)에서 어떤 정보를 기억하고, 삭제하며, 출력할지를 결정.

이 게이트들은 신경망이 필요한 정보를 장기간 기억하는 데 도움을 줌.



수식 구현

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

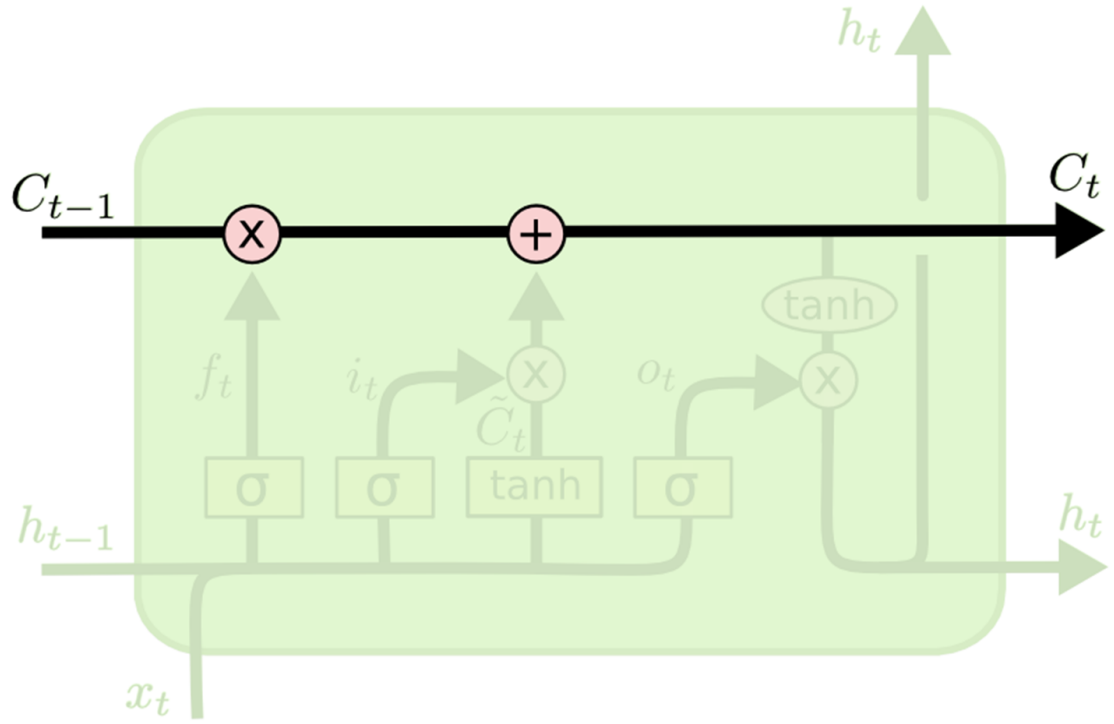
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t * c_{t-1} + i_t * tanh(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t * \tanh(c_t)$$

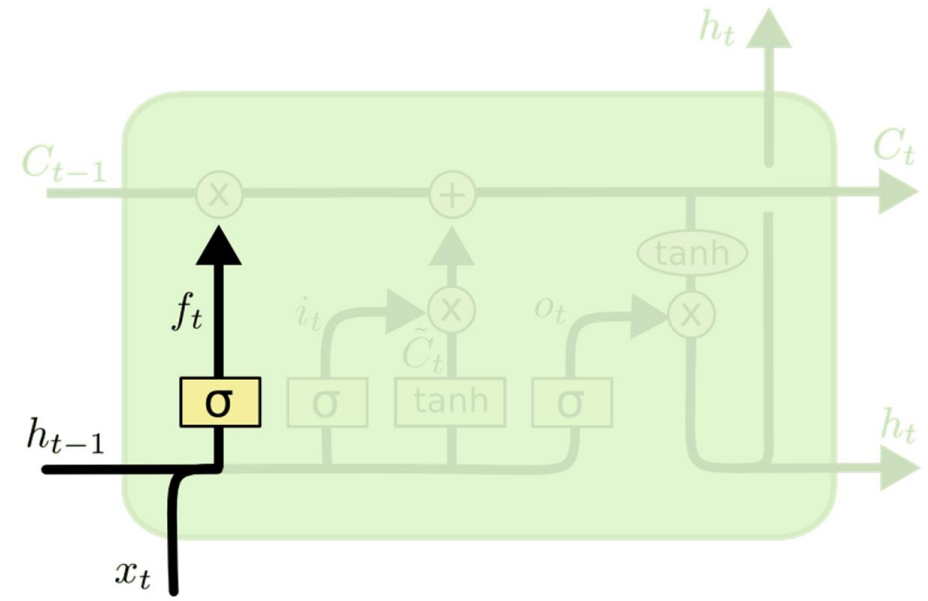
RNN Hidden Unit : LSTM

- **Cell state**
  - ✓ LSTM의 핵심 구성 요소, 아래 그림에서는 다이어그램의 상부를 관통하는 선(line)



RNN Hidden Unit : LSTM

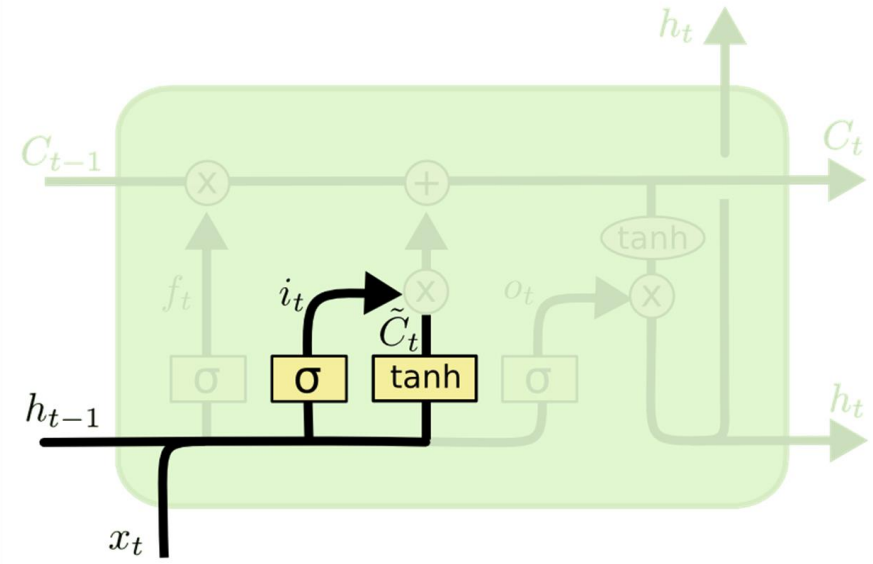
- **Step 1 : 지금까지의 cell state에 저장된 정보 중에서 얼마만큼을 망각(forget)할 것인지 결정**
  - ✓ Forget gate : 이전 단계의 hidden state h_{t-1}와 현 단계의 입력 x_t로부터 0과 1 사이의 값을 출력(Sigmoid 함수 사용)
    - 1: 지금까지 cell state에 저장된 모든 정보를 보존
    - 0 : 지금까지 cell state에 저장된 모든 정보를 무시



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

RNN Hidden Unit : LSTM

- **Step 2 : 새로운 정보를 얼마만큼 cell state에 저장할 것인지를 결정**
  - ✓ Input gate : 어떤 값을 업데이트할 것인지 결정
  - ✓ Tanh layer를 사용하여 새로운 cell state 후보 $\tilde{C}_t$ 를 생성

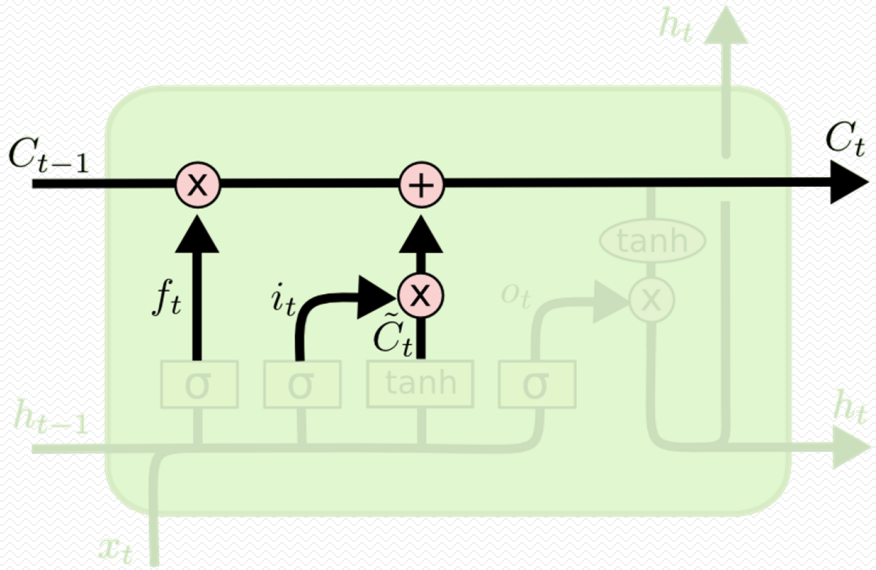


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

RNN Hidden Unit : LSTM

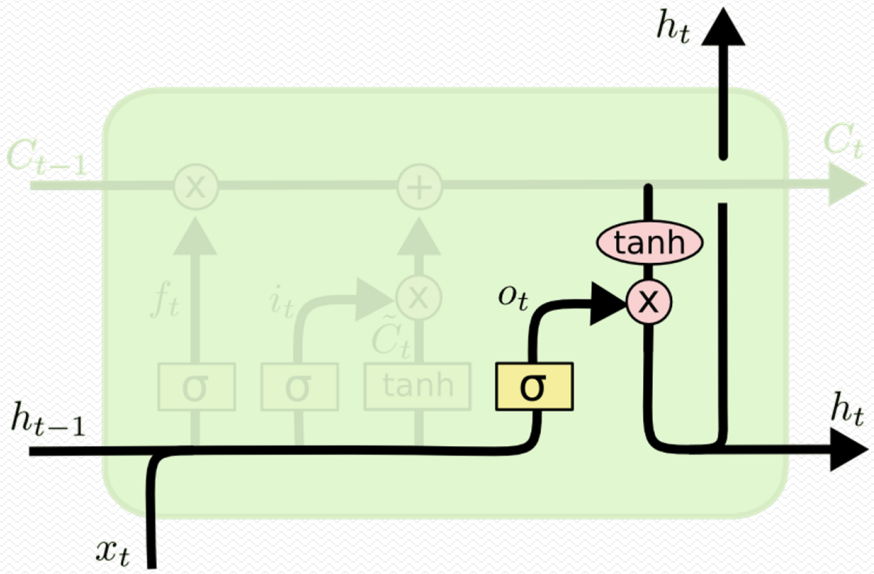
- **Step 3 : 예전 cell state를 새로운 cell state로 업데이트**
  - ✓ 예전 cell state를 얼마만큼 망각할 것인가를 계산한 forget gate 결과값과 곱함
  - ✓ 새로운 cell state 후보와 얼마만큼 보존할 것인가를 계산한 input gate 결과값을 곱함
  - ✓ 두 값을 더하여 새로운 cell state 값으로 결정



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

RNN Hidden Unit : LSTM

- **Step 4 : 출력 값을 결정**
  - ✓ 이전 hidden state 값과 현재의 입력 값을 이용하여 output gate 값을 산출
  - ✓ Output gate 값과 현재의 cell state 값을 결합하여 현재의 hidden state 값을 계산

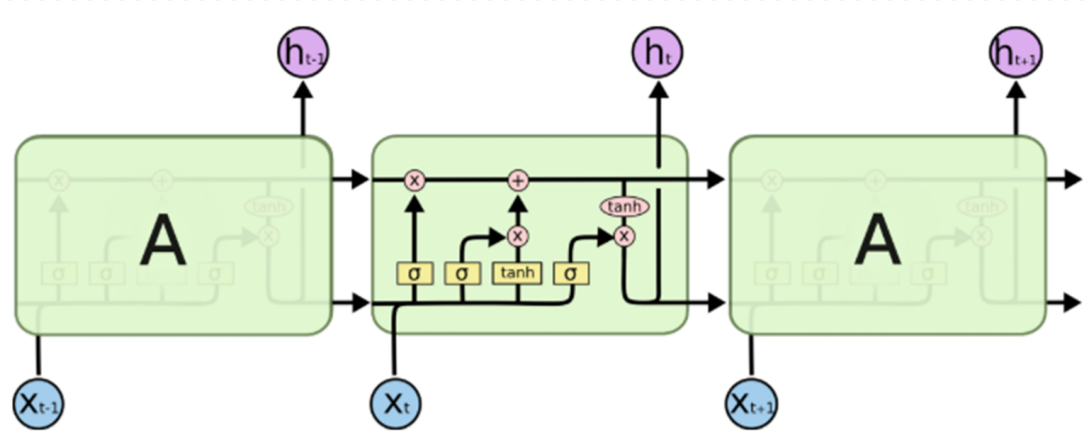


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

RNN Hidden Unit : LSTM

- **LSTM 요약**



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

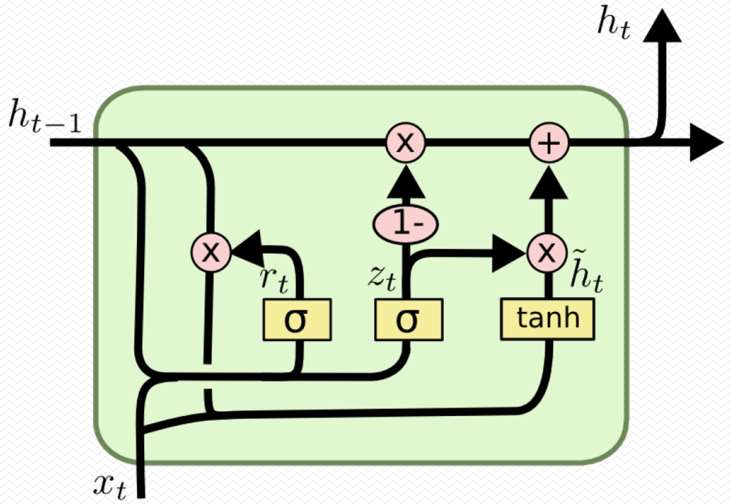
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

RNN Hidden Unit : GRU

- **GRU : Gated Recurrent Unit**



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$ Update gate

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$ Reset gate

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- ✓ LSTM을 단순화한 구조, 실제 활용에서는 LSTM과 GRU의 성능 차이는 미비함
- ✓ 별도의 Cell state가 존재하지 않음
- ✓ LSTM의 forget gate와 input gage를 하나의 update gate로 결합
- ✓ Reset gate를 통해 망각과 새로운 정보 업데이트 정도를 결정

RNN Hidden Unit : GRU

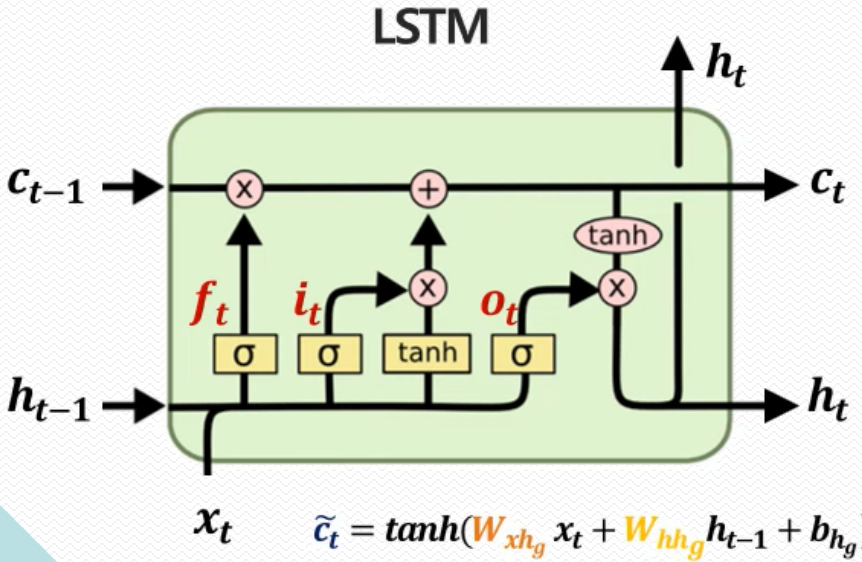
LSTM 구조 개선 모델(단순화 시키고 성능은 비슷)

일반적인 LSTM에서 cell state와 세가지 gate가 정의됨 (forget, input, output)

- Cell state $c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$ 정의하는 과정에서 forget gate와 input gate가 활용
- Hidden state $h_t = o_t \otimes tanh(c_t)$ 로 출력 값 정의 (output gate 활용)

GRU는 LSTM의 구조를 간단하게 개선하여 파라미터 개수를 줄임

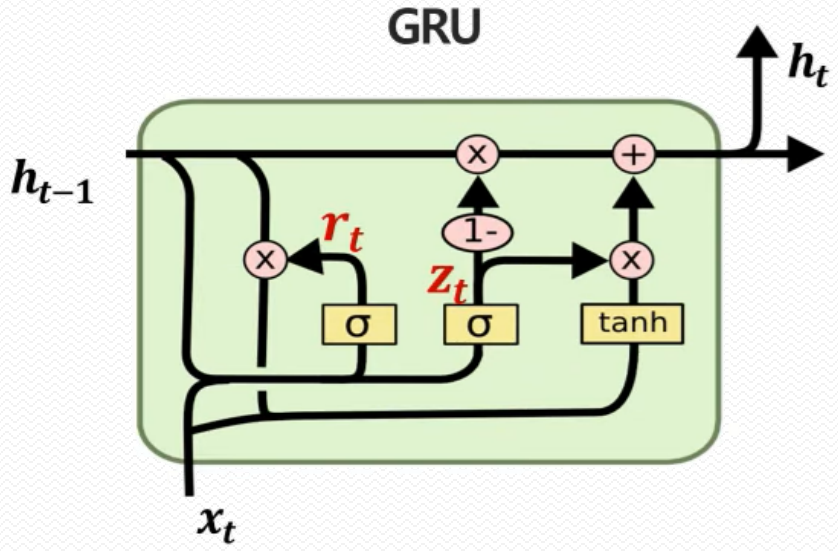
- forget gate, input gate를 update gate(z_t)로 통합, output gate를 없애고 reset gate(r_t)정의
- Cell state, hidden state를 hidden state로 통합



$$\tilde{c}_t = tanh(W_{xh_g} x_t + W_{hh_g} h_{t-1} + b_{h_g})$$

$$c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$$

$$h_t = o_t \otimes tanh(c_t)$$



$$h_t = (1 - z_t) h_{t-1} + z_t \tanh(w_{xh} x_t + b + w_{xh} r_t h_{t-1})$$

Contents

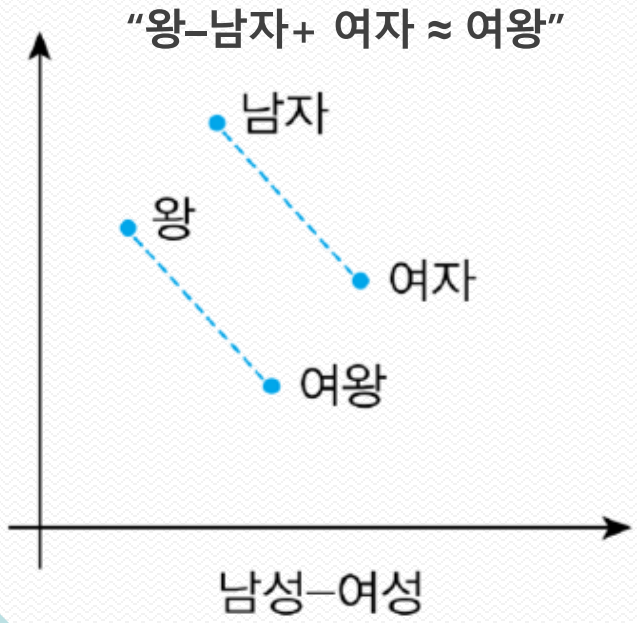
단어 임베딩

목표 : 컴퓨터가 단어의 의미를 이해하고 처리할 수 있도록 단어를 벡터로 변환.

해당 벡터는 다차원 공간에 위치하며, 단어 간의 의미적 관계를 반영함.

벡터 공간 상 단어 벡터들의 특징

- 학습된 단어 벡터들은 단어의 의미가 벡터 공간 상의 다양한 특성들(거리, 길이, 각도 등)로 표현됨
- 단어 벡터에는 문법에서 의존관계에 이르기까지 다양한 언어학적 지식들이 담겨져 있음.
- 단어 임베딩 모델에는 Word2Vec, GloVe, FastText 등이 있음

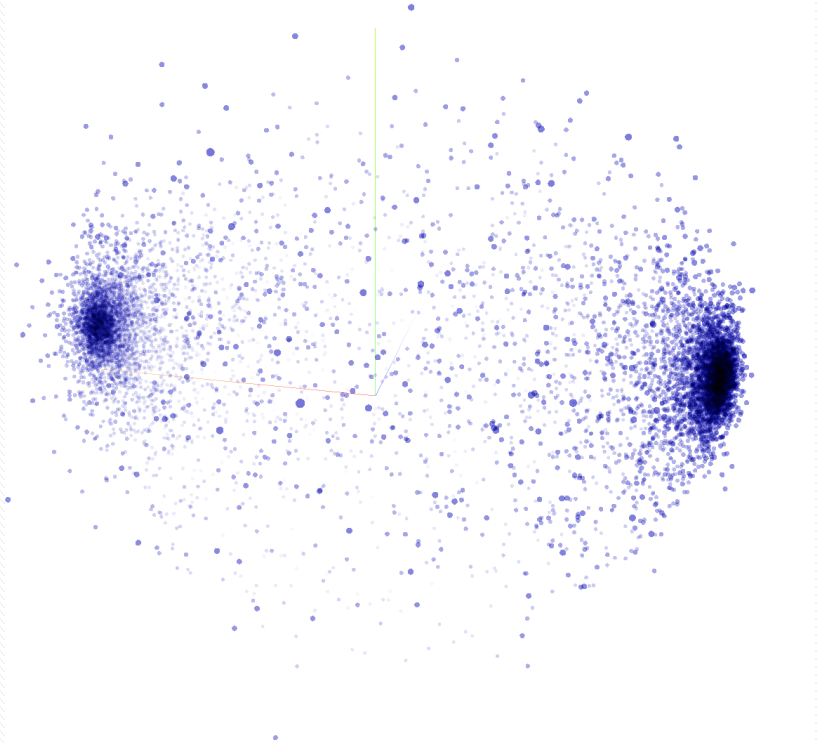


(예시) 5차원 벡터로 표현하면

- 왕(king): [0.125, -0.765, 0.159, 0.256, -0.495]
- 여왕(queen): [0.145, -0.750, 0.140, 0.275, -0.482]
- 남자(man): [0.115, -0.855, 0.169, 0.333, -0.425]
- 여자(woman): [0.135, -0.840, 0.150, 0.352, -0.412]

단어 임베딩

- **'https://ai.stanford.edu/%7Eamaas/data/sentiment/'에서 제공하는 데이터를 이용**
- **Keras 파일 유틸리티를 사용하여 데이터 세트를 다운로드**
- **Tensor의 Embedding을 이용**
- **임베딩 레이어를 생성할 때 임베딩의 가중치는 다른 레이어와 마찬가지로 무작위로 초기화후 학습진행**
- **얻어진 가중치를 저장하고 임베딩 프로젝터를 이용하여 시각화함**



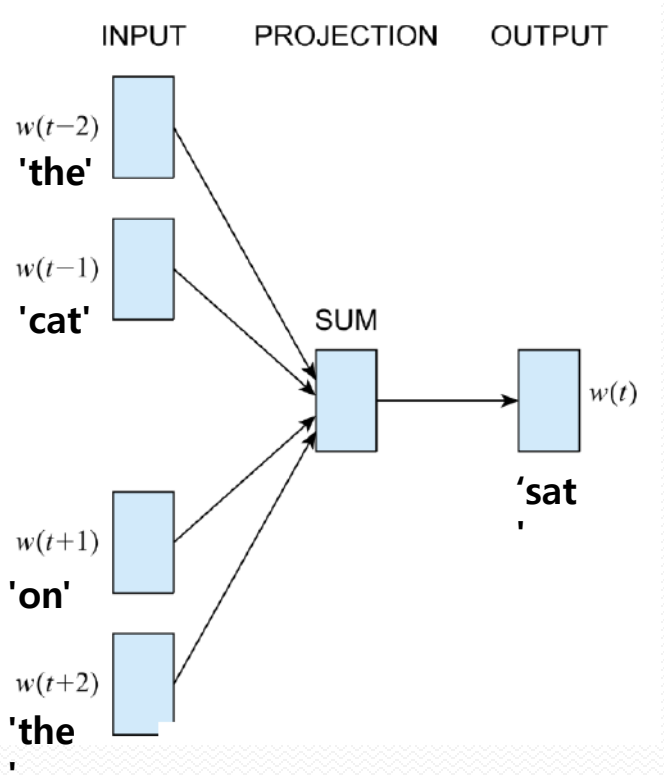
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding
https://projector.tensorflow.org/?hl=ko

Word2Vec

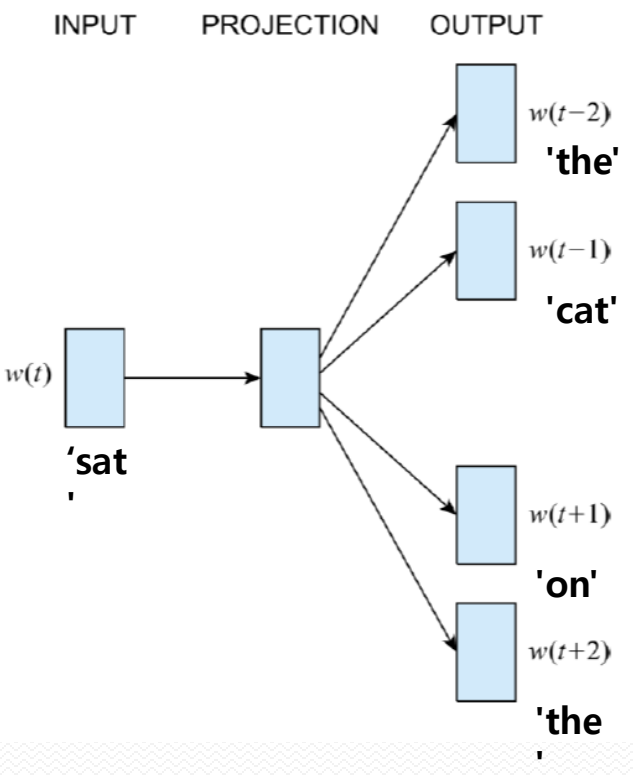
대표적인 두가지 알고리즘

- 두 방법 모두 확률적으로 더 먼 거리의 단어들을 선택함으로써 넓은 범위의 문맥을 단어 벡터 학습에 활용할 수 있도록 함

CBOW (Continuous Bag Of Words) 모델 : 특정 단어를 중심으로 이전 n 개의 단어와 이후 n 개의 단어가 주어졌을 때 , 중심 단어를 예측하는 것을 목표로 학습



Skip gram 모델 : 중심 단어가 주어졌을 때 이전 n 개의 단어와 이후 n 개의 단어를 예측하는 것을 목표로 학습



GloVe

GloVe : Global Vectors for Word Representation (Pennington et 의 2014 년 논문)

- 특정 단어가 문서 내에서 함께 등장하는 동시 발생 행렬 (co-occurrence matrix) 빈도를 기반으로 단어 벡터를 생성
- 단어 간 동시 발생 빈도의 비율이 벡터 공간상 차이로 나타나도록 단어 임베딩을 수행
- 특정 두 단어의 동시 발생 비율을 다른 단어 동시 발생 비율과 비교함으로써 단어 간 의미적 차이를 포착하는 학습

co-occurrence matrix

특정한 두 단어가 주어진 문맥이나 문서에서 함께 나타나는 빈도수를 나타내는 행렬

- 텍스트 마이닝과 자연어 처리에서 단어 사이의 관계와 패턴을 분석하는 데 중요한 도구.
- 각 행렬 요소 C(i,j)는 단어 i가 단어 j와 근접하여 나타나는 횟수를 나타냄
-의미 분석, 주제 모델링, 단어 사이의 맥락적 유사성 이해 등의 작업에 도움

-	apples	are	green	and	red	sweet	oranges	sour
apples	2	2	1	1	2	1	0	0
are	2	3	1	1	2	1	1	1
green	1	1	2	1	1	0	1	1
and	1	1	1	1	1	0	0	0
red	2	2	1	1	2	1	0	0
sweet	1	1	0	0	1	1	0	0
oranges	0	1	1	0	0	0	1	1
sour	0	1	1	0	0	0	1	1

FastText

Facebook Research 에서 2016 년도에 공개

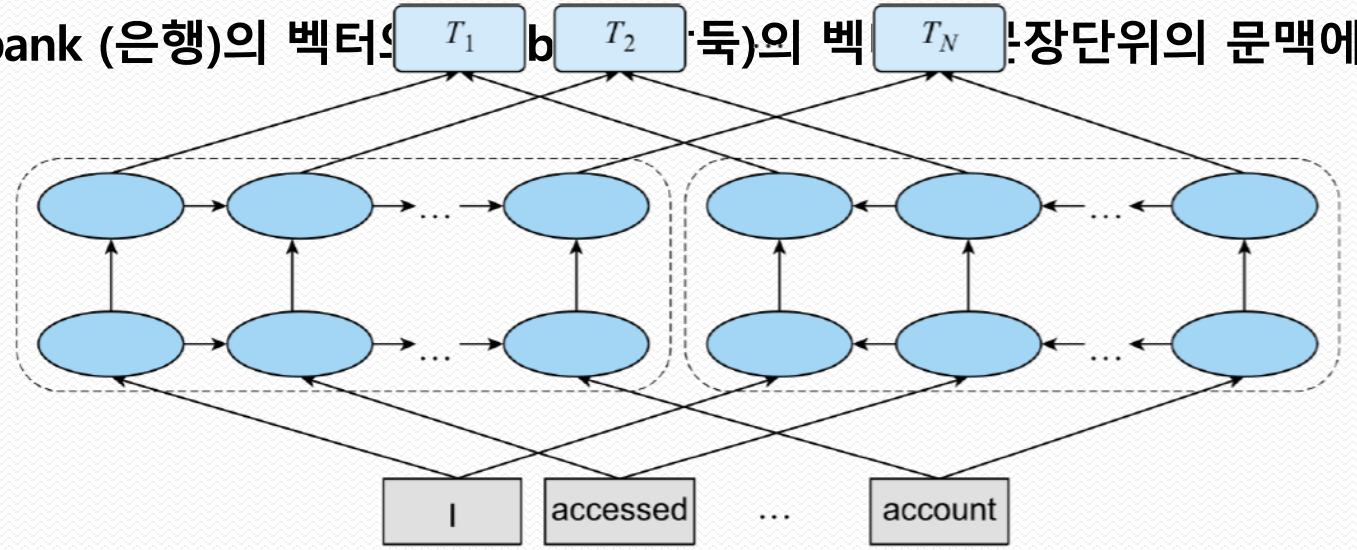
- 단어가 한국어의 형태소와 같이 더 작은 서브워드(subword) 단위로 나뉘어질 수 있다는 점에 착안
- 단어를 여러 n-gram 들의 집합으로 분리한 후 , 각 n-gram 에 대한 벡터를 학습
  - ✓ 2-gram 에서의 예 : “ 공부하였다”⇒ ["<공”, ”공부”, ”부하”, ”하였”, ”였다”, "다>"]
  - ✓ “공부하였다” 의 단어 벡터는 이를 구성하는 n gram 벡터들의 합으로 나타냄
- 모델이 단어의 형태학적인 특성을 학습, 복합 단어나 신조어, 심지어 철자가 틀린 단어까지도 그 의미를 더 잘 이해할 수 있게 함

ELMo-Embeddings from Language Models

2018 년 Peters et al. 의 논문에서 제안된 **문장 단위 임베딩** 모델

- 양방향 LSTM 네트워크를 사용하여 주어진 텍스트에서 앞뒤의 문맥을 모두 고려.
- 여러 층(layer)에 걸쳐 단어의 표현을 학습하며, 각 층은 다른 수준의 의미를 포착,
  예를 들어 낮은 층은 문법적인 특성을, 높은 층은 문맥상의 의미를 반영
- **동적 임베딩** : 각 단어에 대해 각 층의 표현을 결합하여 최종 임베딩을 생성.
  이 과정은 문장 내에서의 단어의 실제 사용 상황을 반영하기 때문에, **같은 단어라도 다른 문맥에서는 다른 임베딩을 가지게 됨.**

예를 들면 the bank (은행)의 벡터와 b(둑)의 벡 문장단위의 문맥에 맞게 다른 임베딩으로 생성



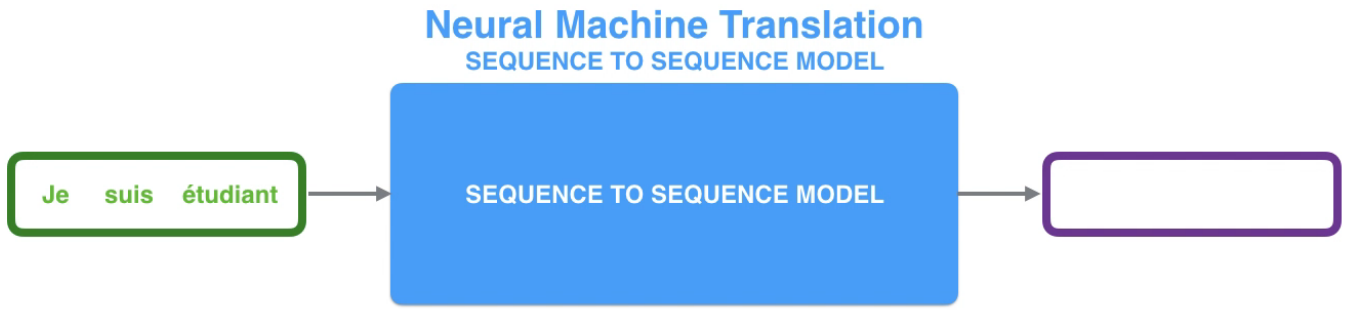
Contents

Seq2seq(Sequence-to-Sequence)

- **A model that takes a sequence of items (words, letters, features of an images, etc.)**
- **Outputs another sequence of items**

· A trained model

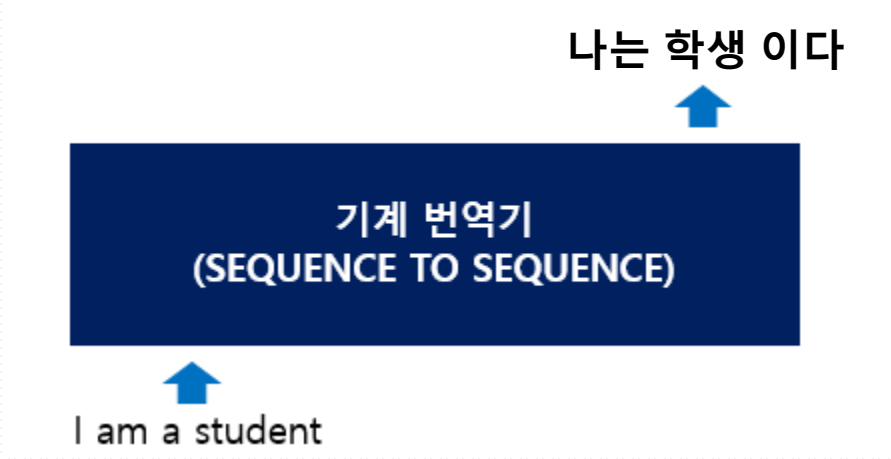
· Neural machine translation

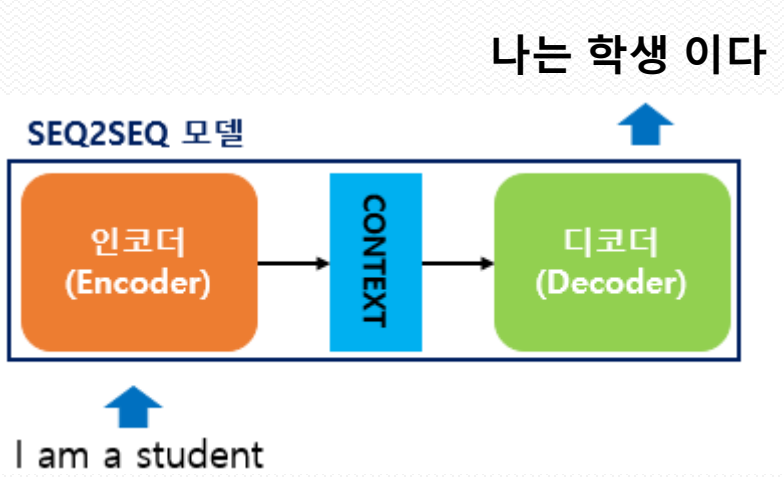


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Seq2seq(Sequence-to-Sequence)

- Encoder-Decoder구조로 RNN의 발전된 모델 아키텍쳐
- Encoder : 입력 데이터의 정보를 압축하는 역할
- Decoder : 인코더의 압축 정보와 출력 데이터를 입력 받아 문장 생성하는 역할
- LSTM, GRU 등의 RNN Cell을 길고 깊게 쌓아 더욱 복잡하고 방대한 시퀀스 데이터를 처리

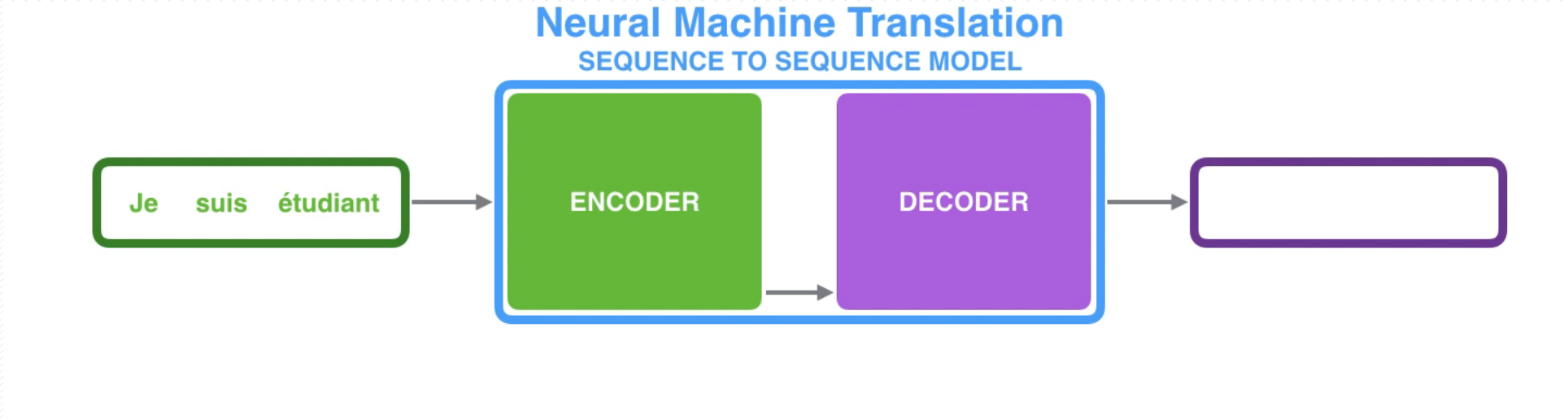




Seq2seq(Sequence-to-Sequence)

· Encoder-Decoder

- The encoder processes each item in the input sequence and compiles the information it captures into a vector (context)
- After processing the entire input sequence, the encoder send the context over to thevdecoder, which begins producing the output sequence item by item

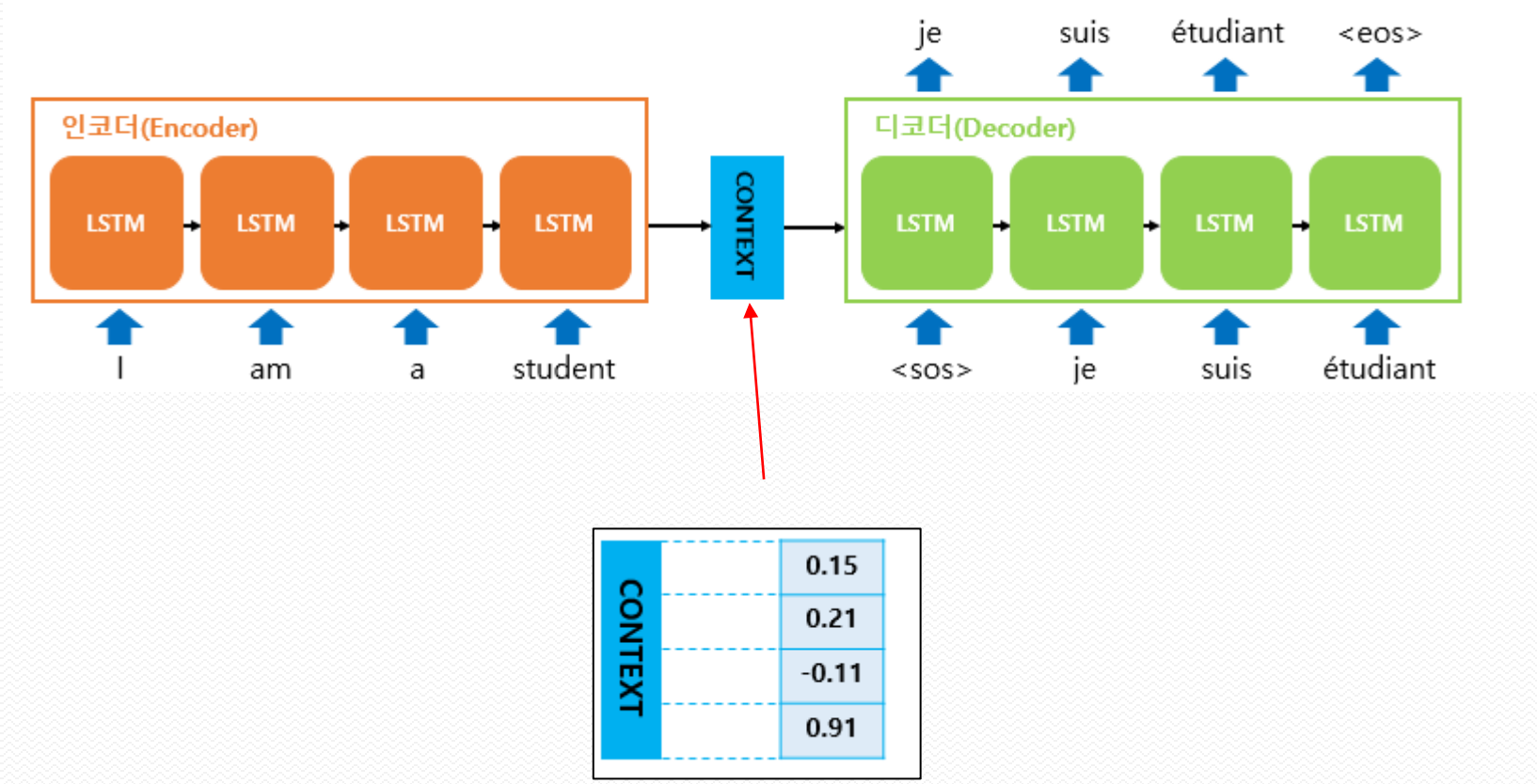


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

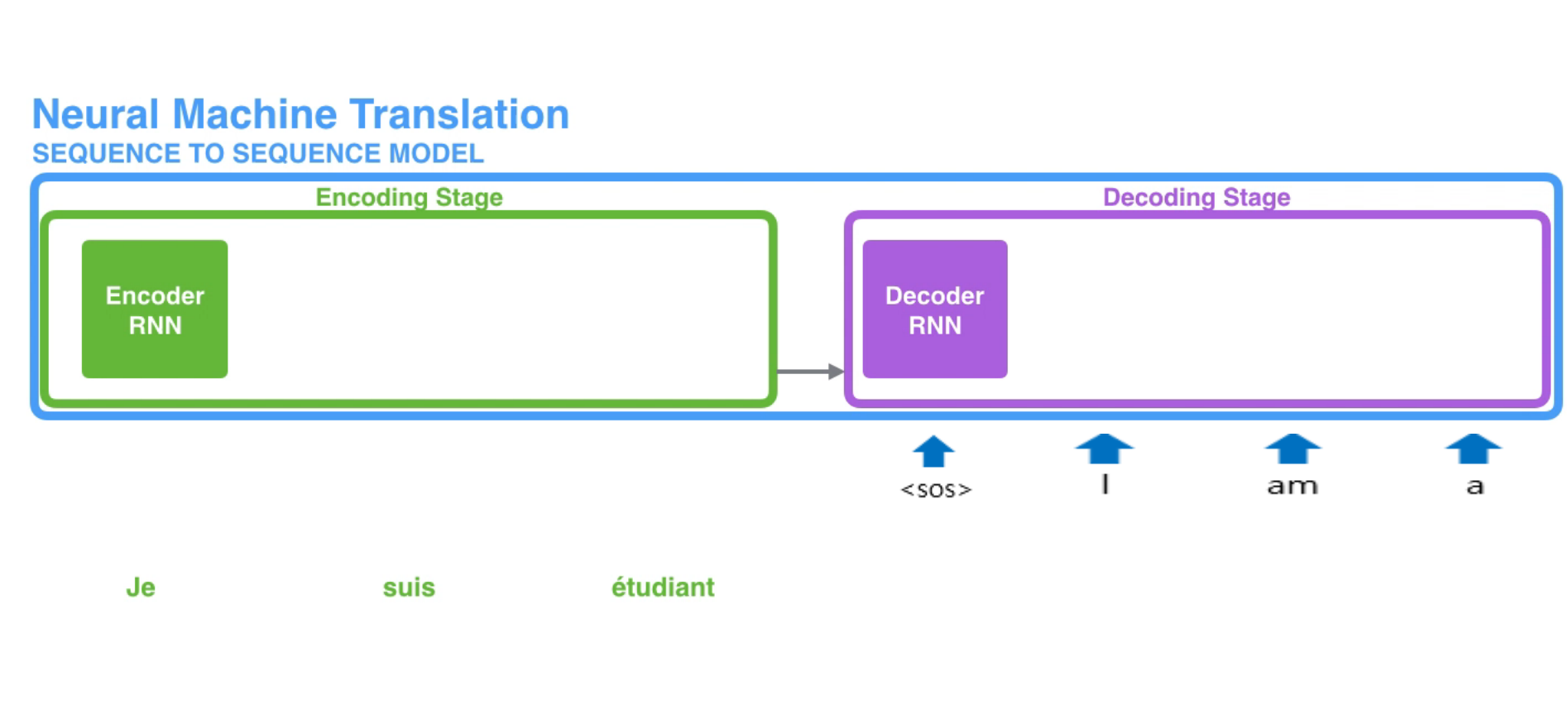
Seq2seq(Sequence-to-Sequence)

· Encoder-Decoder

- The context is a vector in the case of machine translation

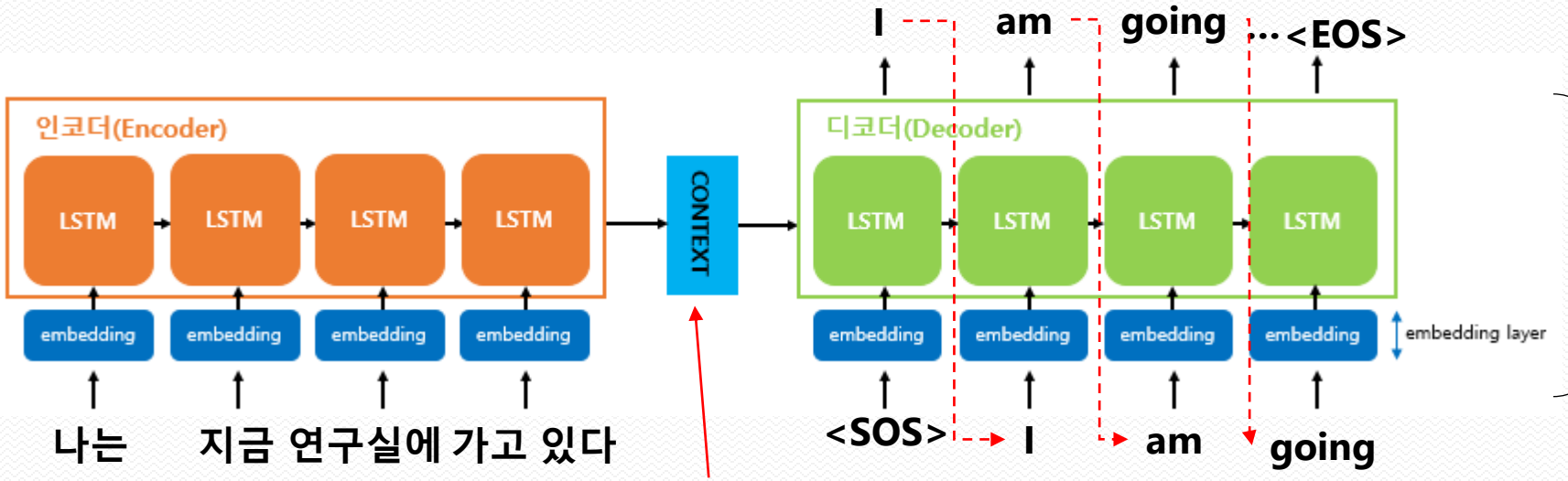


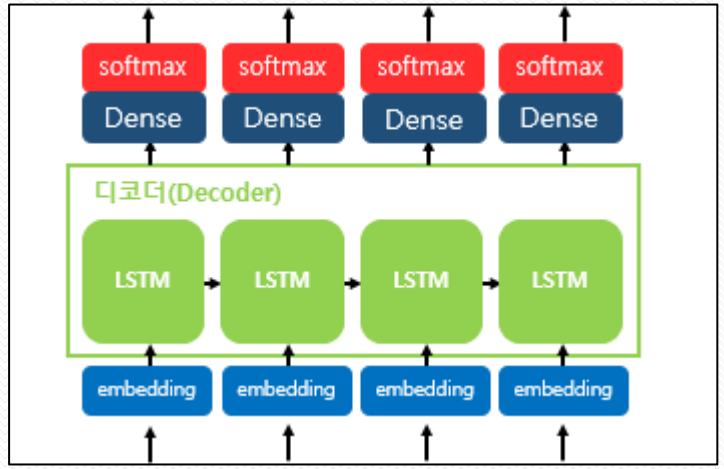
Seq2seq(Sequence-to-Sequence)

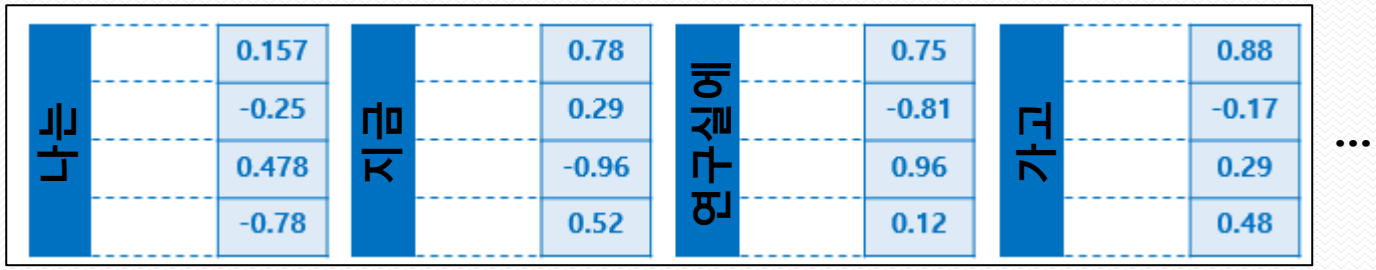


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

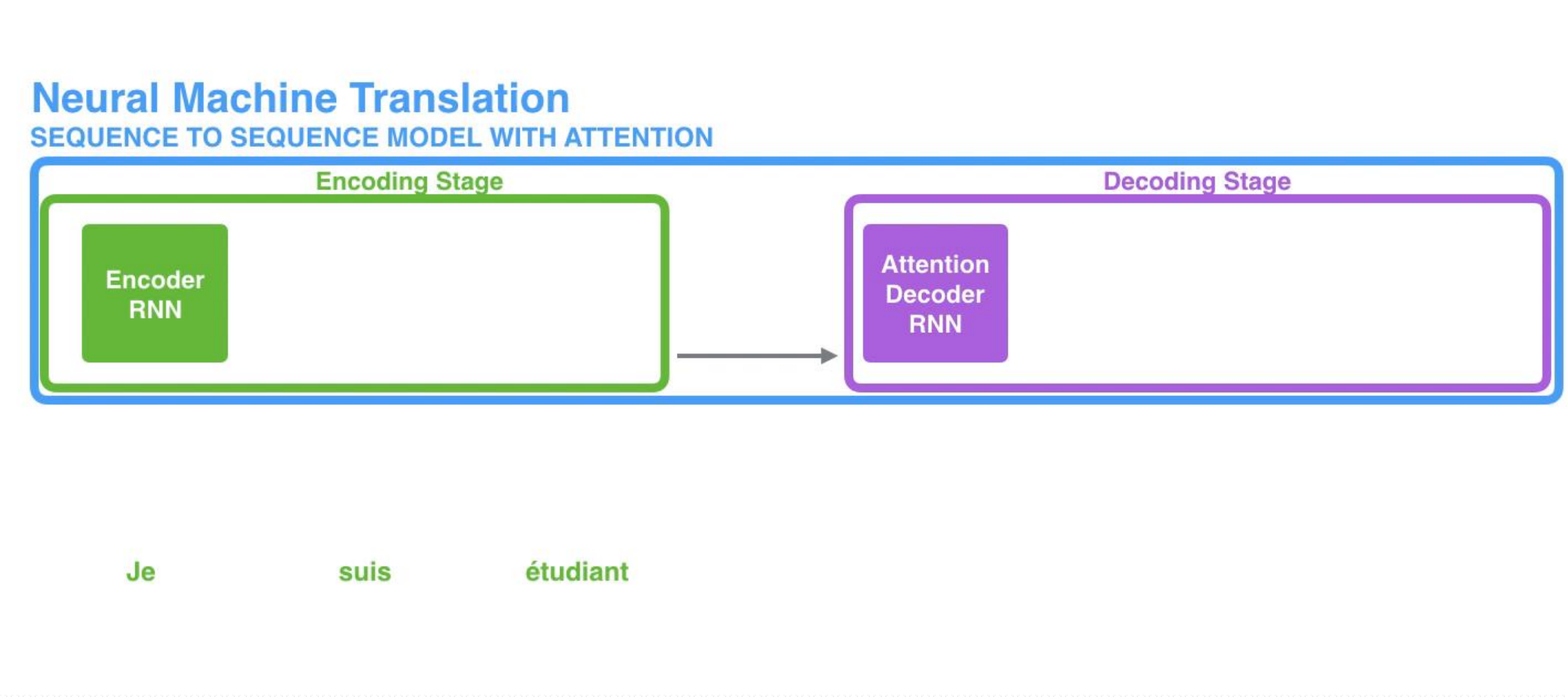
Seq2seq(Sequence-to-Sequence)







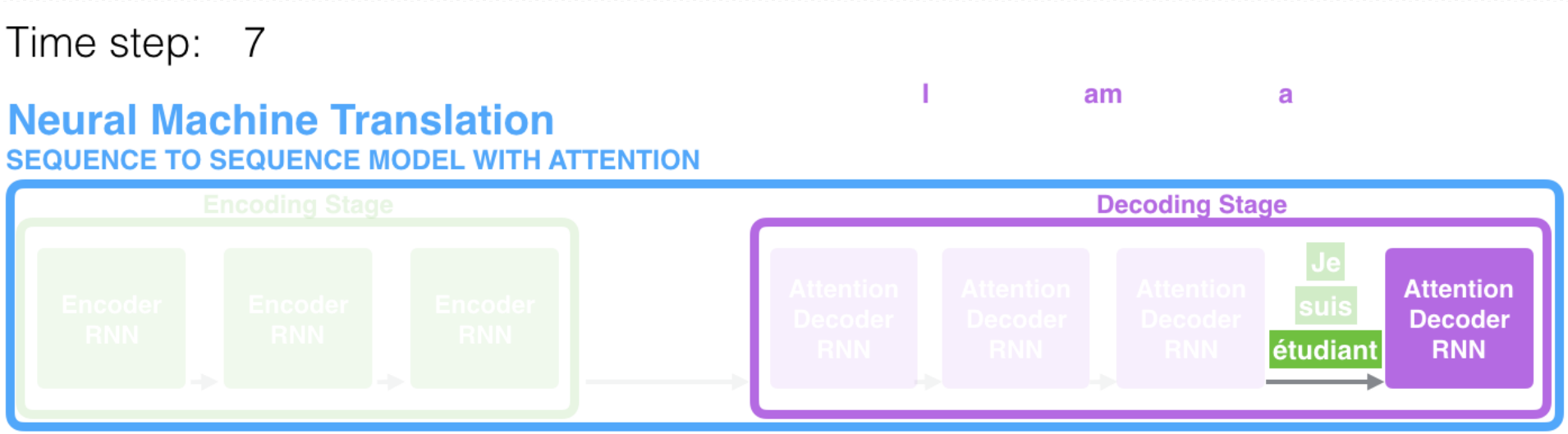
Attention


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION
Encoding Stage
Decoding Stage
Encoder RNN
Attention Decoder RNN
Je
suis
étudiant

Attention

Attention Mechanism

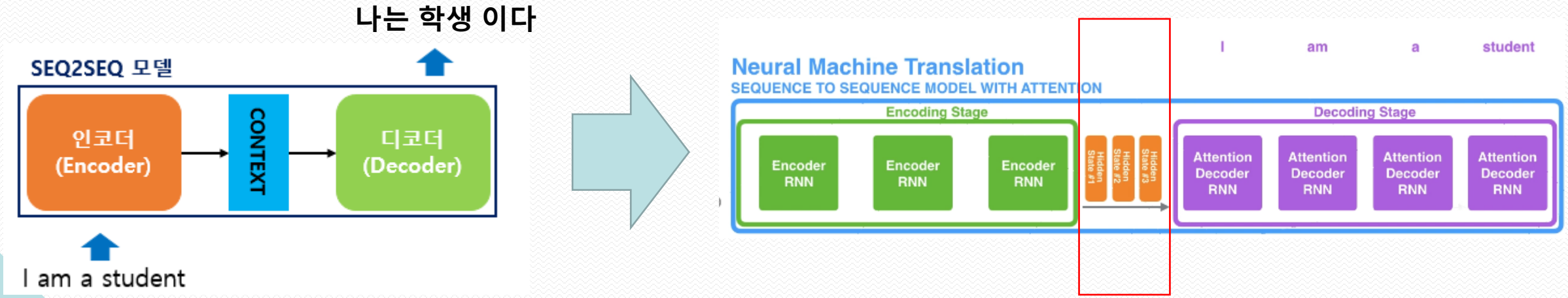
- Seq2Seq모델 Encoder에서 압축한 context vector는 전체 입력 시퀀스 데이터로만 효과적으로 표현할 수 있을까?
- **입력 시퀀스 데이터가 길어질 경우 문장 앞 부분에 대한 정보 손실 발생 (Long term dependency problem)**
- 입력 시퀀스 데이터의 길이가 긴 문장의 기억을 돕기 위한 방법
- Decoder에서 i번째 단어를 예측할 때 사용하는 이전 스텝의 Decoder정보와 인코더의 j번째 정보가 얼마나 유사한지 스코어 산출



Attention

Attention Mechanism

- Attention model differs from a classic Seq2Seq model in two main ways:
- The encoder passes a lot more data to the decoder
- Instead of passing the last hidden state of the encoding stage, the encoder passes all the hidden states to the decoder

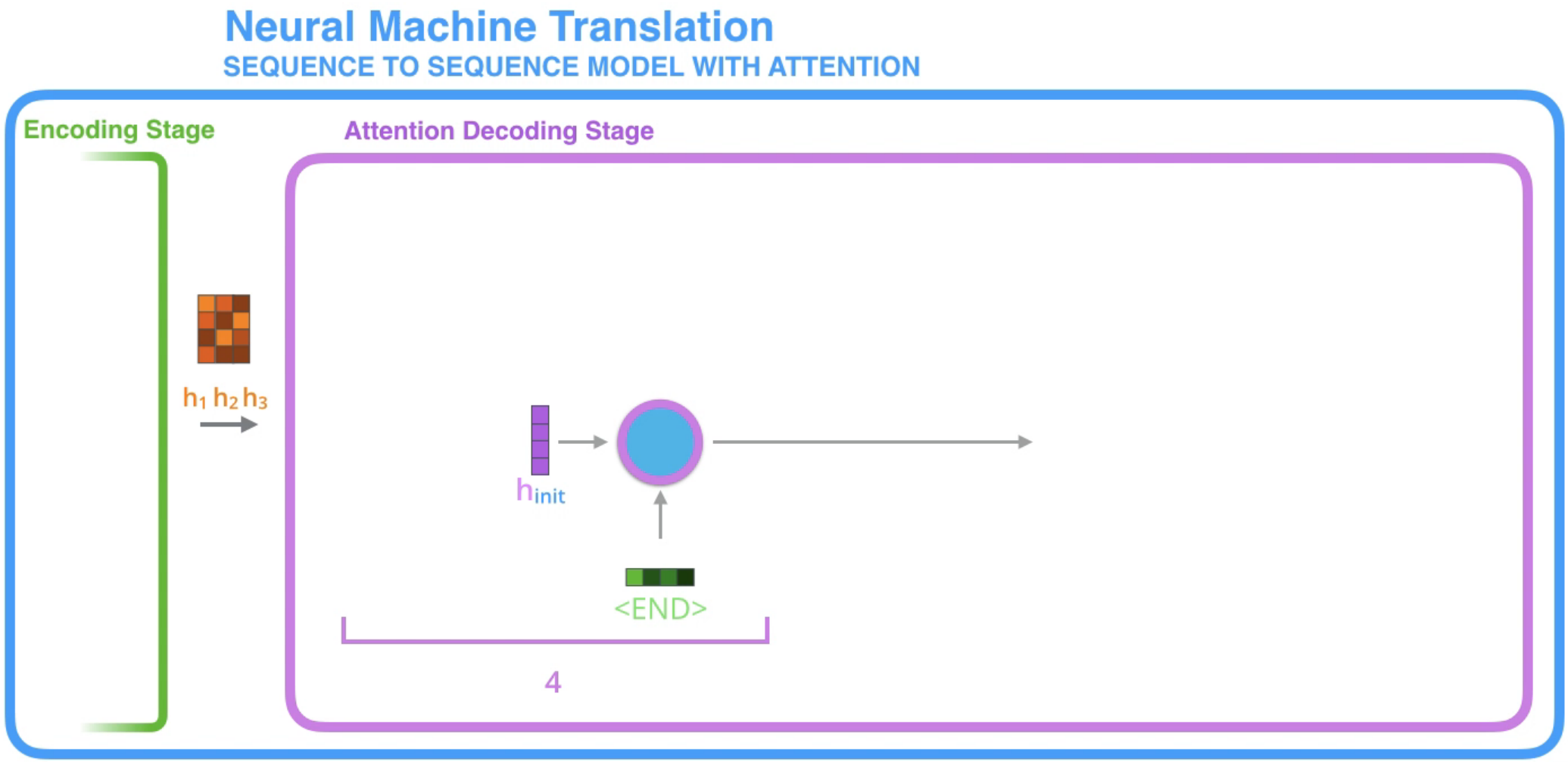


Attention

Attention Mechanism

- An attention decoder does an extra step before producing its output
- Look at the set of encoder hidden states it received
- each encoder hidden states is most associated with a certain word in the input sentence
- Give each hidden states a score.
- Multiply each hidden state by its softmaxed score, this amplifying hidden state with high scores, an drowning out hidden state with low scores

Attention

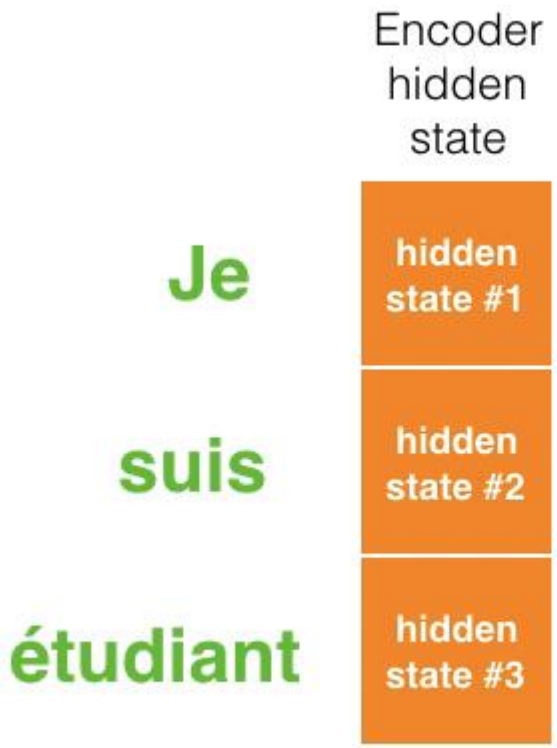

Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION
Encoding Stage
Attention Decoding Stage
h1 h2 h3
hinit
<END>
4

Attention

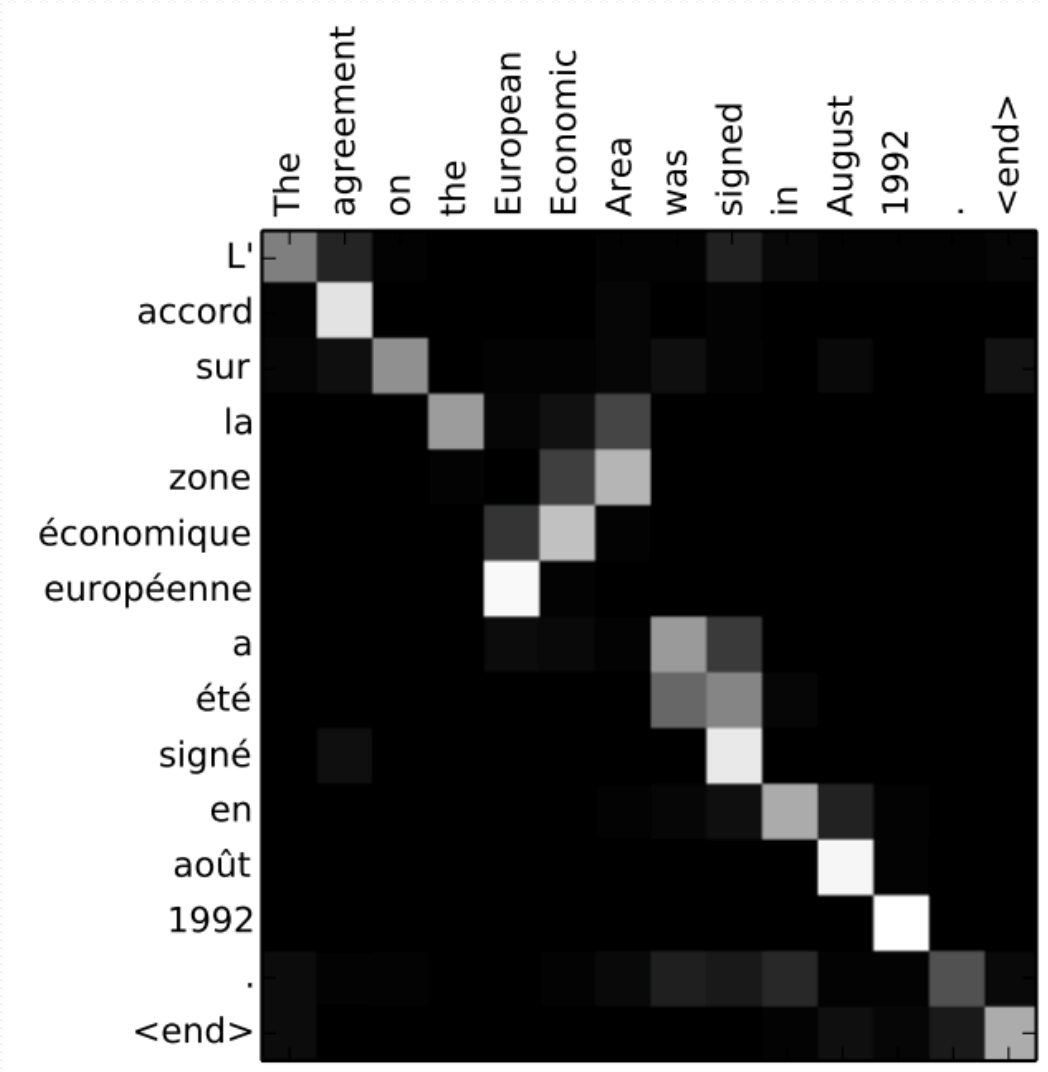
Attention Mechanism

- The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
- The RNN/LSTM processes its inputs, producing an output and a new hidden state vector(h4). The output is discarded.
- Attention Step:
- 1) We use the encoder hidden states and the h4 vector to calculate acontext vector (C4) for this time step.
- 2) We concatenate h4 and C4 into one vector.
- 3) We pass this vector through a feedforward neural network (one trained jointly with the model).
- 4) The output of the feedforward neural networks indicates the output word of this time step.
- Repeat for the next time steps

Attention


Encoder
hidden
state
Je
suis
étudiant
hidden
state #1
hidden
state #2
hidden
state #3

Attention


The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

다차원 Vector의 내적

직교성(Orthogonal & Orthonormal)

$$\vec{x} = x_1\overrightarrow{\phi_1} + x_2\overrightarrow{\phi_2} + \ldots + x_n\overrightarrow{\phi_n}$$

$$\langle \overrightarrow{\phi_n}, \overrightarrow{\phi_m} \rangle = \begin{cases} 1 & n = m \\ 0 & n \neq m \end{cases}$$

Set $\{\overrightarrow{\phi_n}\}$ are Orthonormal

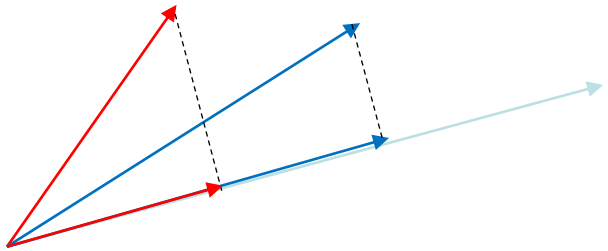
- 신호 $x(t)$에 관한 식의 기저함수를 $\overrightarrow{\phi_n}$라고 할 때, 다른 기저함수 간의 내적이 0일 경우 Orthogonal 이라고 함
- 이때, 동일 기저함수 간의 내적이 1일 경우 Orthonormal 이라고 함

다차원 Vector의 내적

가중치 추출

$$\vec{x} = x_1\overrightarrow{\phi_1} + x_2\overrightarrow{\phi_2} + \ldots + x_n\overrightarrow{\phi_n}$$

$$\overrightarrow{x_n} = \langle \overrightarrow{x_n}, \overrightarrow{\phi_m} \rangle$$

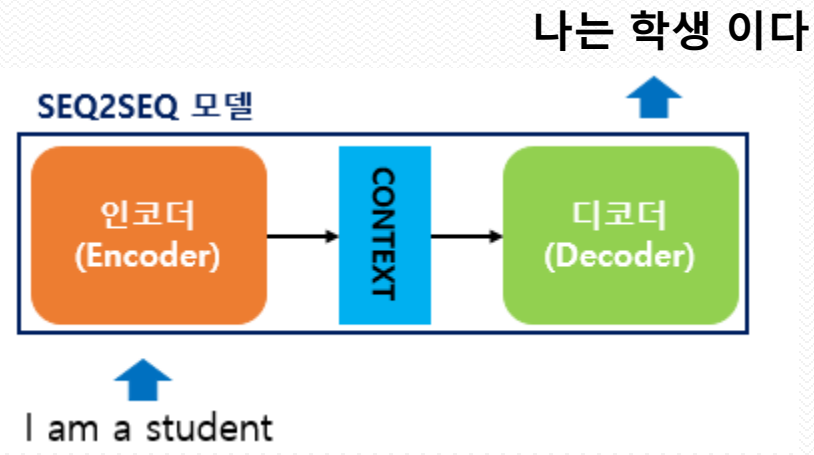


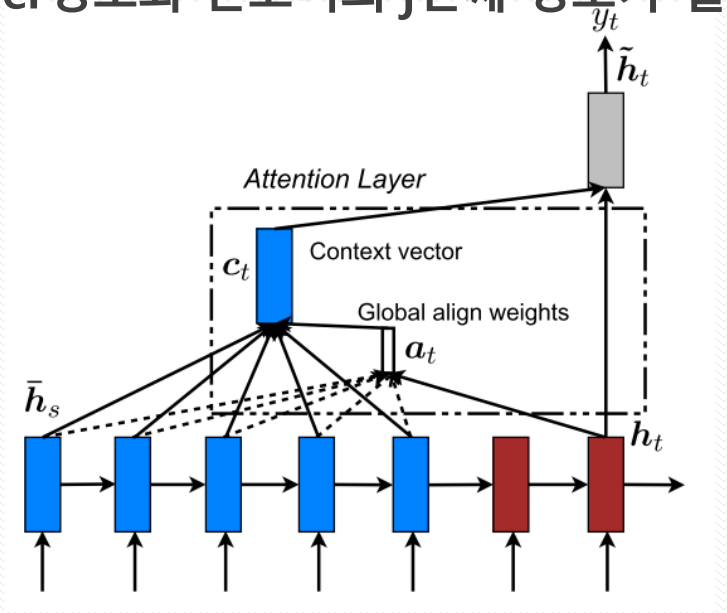
- $\overrightarrow{x_n}$에 관한 기저함수 $\overrightarrow{\phi_n}$ 의 가중치 혹은 계수는$\overrightarrow{x_n}$와 $\overrightarrow{\phi_n}$의 내적으로 추출됨
- $\overrightarrow{x_n}$와 $\overrightarrow{\phi_n}$의 내적은 $\overrightarrow{x_n}$ 의 $\overrightarrow{\phi_n}$ 방향의 정사형 성분으로 $\overrightarrow{\phi_n}$ 방향의 **가중치**로 해석됨

Attention

Attention Mechanism

- Seq2Seq모델 Encoder에서 압축한 context vector는 전체 입력 시퀀스 데이터로만 효과적으로 표현할 수 있을까?
- 입력 시퀀스 데이터가 길어질 경우 문장 앞 부분에 대한 정보 손실 발생 (Long term dependency problem)
- 입력 시퀀스 데이터의 길이가 긴 문장의 기억을 돕기 위한 방법
- Decoder에서 i번째 단어를 예측할 때 사용하는 이전 스텝의 Decoder정보와 인코더의 j번째 정보가 얼마나 유사한지 스코어 산출





Attention

Attention Mechanism

Attention이 고려된 hidden state vector를 통해 최종 예측 수행

$$p(y|\mathbf{x}) = \text{softmax}(\mathbf{W_s}\tilde{\mathbf{h}})$$

Attention이 고려된 hidden state는 context vector와 기존 hidden state를 concatenation한 뒤 선형 및 비선형 변환을 수행

$$\tilde{\mathbf{h}} = tanh(\mathbf{W_c}[\mathbf{c}; \mathbf{h}^*])$$

Context vector는 각 입력 단계의 hidden state의 선형 결합 (가중치 = alpha)

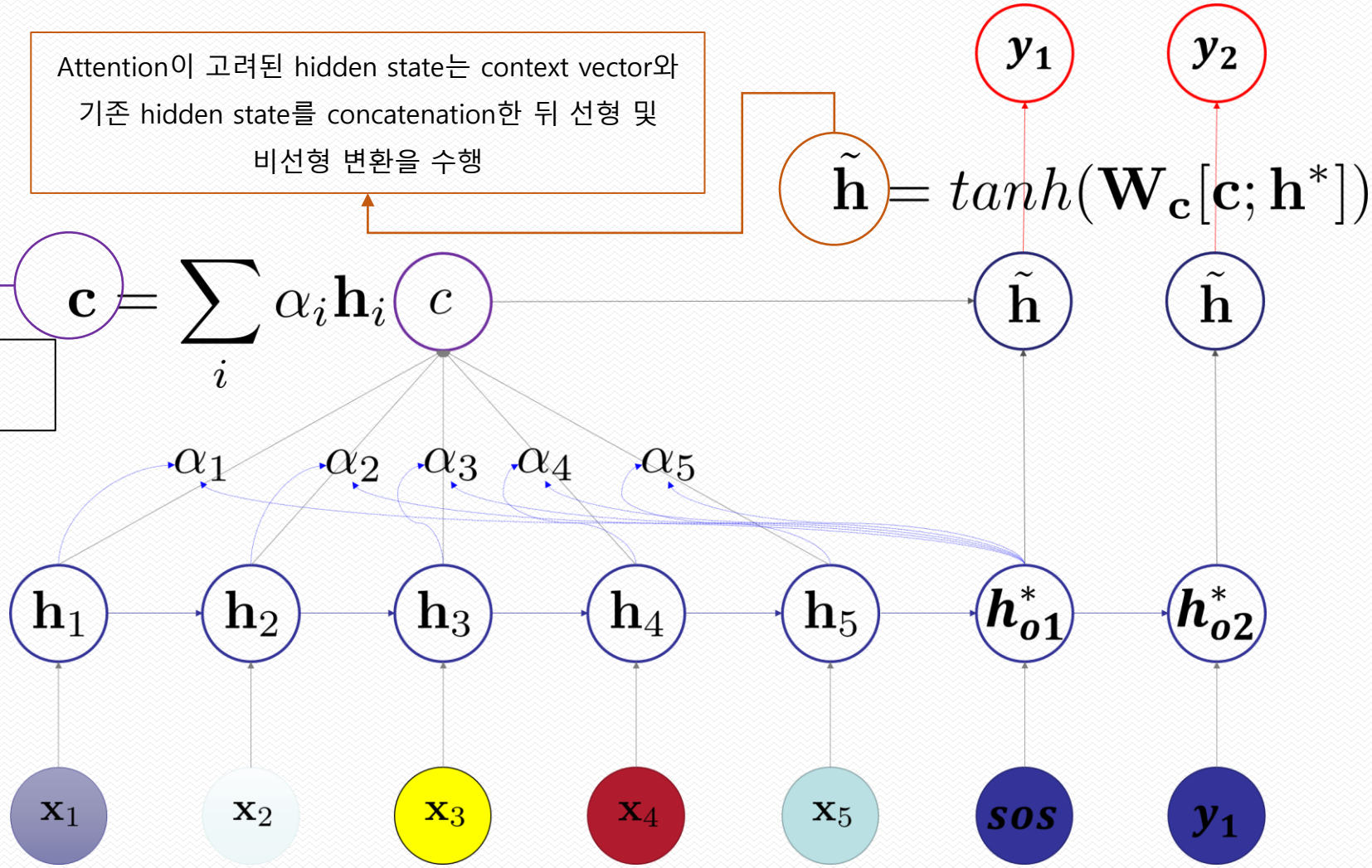
$$\mathbf{c} = \sum_i \alpha_i \mathbf{h}_i$$

i번째 hidden state vector가 context vector 생성에 기여하는 비중

$$\alpha_i = \text{align}(\mathbf{h}^*, \mathbf{h}_i) = \frac{exp\Big(\text{score}(\mathbf{h}^*, \mathbf{h}_i)\Big)}{\sum_{i'} exp\Big(\text{score}(\mathbf{h}^*, \mathbf{h}'_i)\Big)}$$

두 hidden state vector 사이의 score 산출 방식 중 가장 간단한 방식은 두 벡터의 내적을 사용하는 것

$$\text{score}(\mathbf{h}^*, \mathbf{h}_i) = \mathbf{h}^{*T}\mathbf{h}_i$$



Attention

Attention Mechanism

Attention이 고려된 hidden state는 context vector와 기존 hidden state를 concatenation한 뒤 선형 및 비선형 변환을 수행

$$\tilde{\mathbf{h}} = tanh(\mathbf{W_c}[\mathbf{c}; \mathbf{h}^*])$$

Context vector는 각 입력 단계의 hidden state의 선형 결합 (가중치 = alpha)

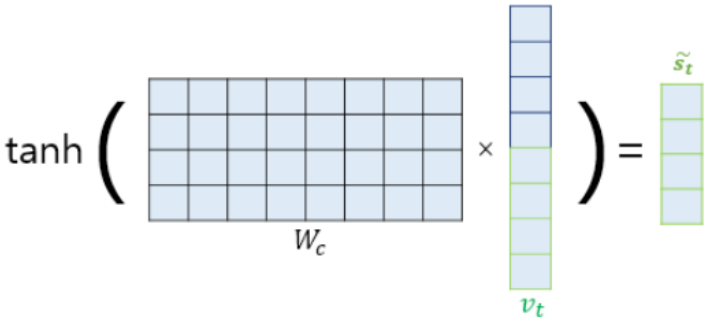
$$\mathbf{c} = \sum_i \alpha_i \mathbf{h}_i$$

i번째 hidden state vector가 context vector 생성에 기여하는 비중

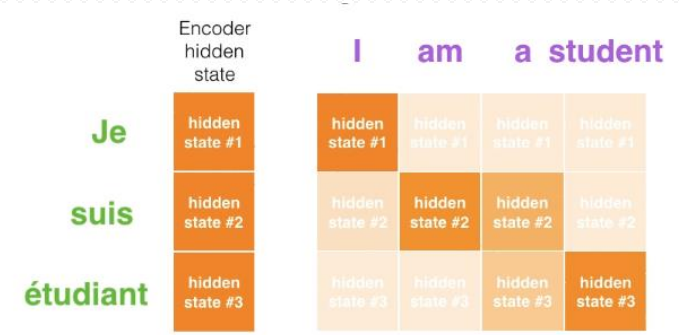
$$\alpha_i = \text{align}(\mathbf{h}^*, \mathbf{h}_i) = \frac{exp\Big(\text{score}(\mathbf{h}^*, \mathbf{h}_i)\Big)}{\sum_{i'} exp\Big(\text{score}(\mathbf{h}^*, \mathbf{h}'_i)\Big)}$$

두 hidden state vector 사이의 score 산출 방식 중 가장 간단한 방식은 두 벡터의 내적을 사용하는 것

$$\text{score}(\mathbf{h}^*, \mathbf{h}_i) = \mathbf{h}^{*T}\mathbf{h}_i$$



$$\boldsymbol{C} = \sum \boldsymbol{softmax}(\langle \boldsymbol{h_i}, \boldsymbol{h^*_{oj}} \rangle) \boldsymbol{h_i}$$



Attention : 닷-프로덕트 어텐션(Dot-Product Attention)

1) 어텐션 스코어(Attention Score)를 구함. $\boxed{\text{score}(\mathbf{h}^*, \mathbf{h}_i) = \mathbf{h}^{*T}\mathbf{h}_i}$

2) 소프트맥스(softmax) 함수를 통해 어텐션 분포(Attention Distribution)를 구함. $\boxed{\alpha_i = \text{align}(\mathbf{h}^*, \mathbf{h}_i) = \frac{exp\Big(\text{score}(\mathbf{h}^*, \mathbf{h}_i)\Big)}{\sum_{i'} exp\Big(\text{score}(\mathbf{h}^*, \mathbf{h}'_i)\Big)}}$

3) 각 인코더의 어텐션 가중치와 은닉 상태를 가중합하여 어텐션 값(Attention Value)을 구함.

$$\boxed{\mathbf{c} = \sum_i \alpha_i \mathbf{h}_i}$$

4) 어텐션 값과 디코더의 t 시점의 은닉 상태를 연결.(Concatenate)

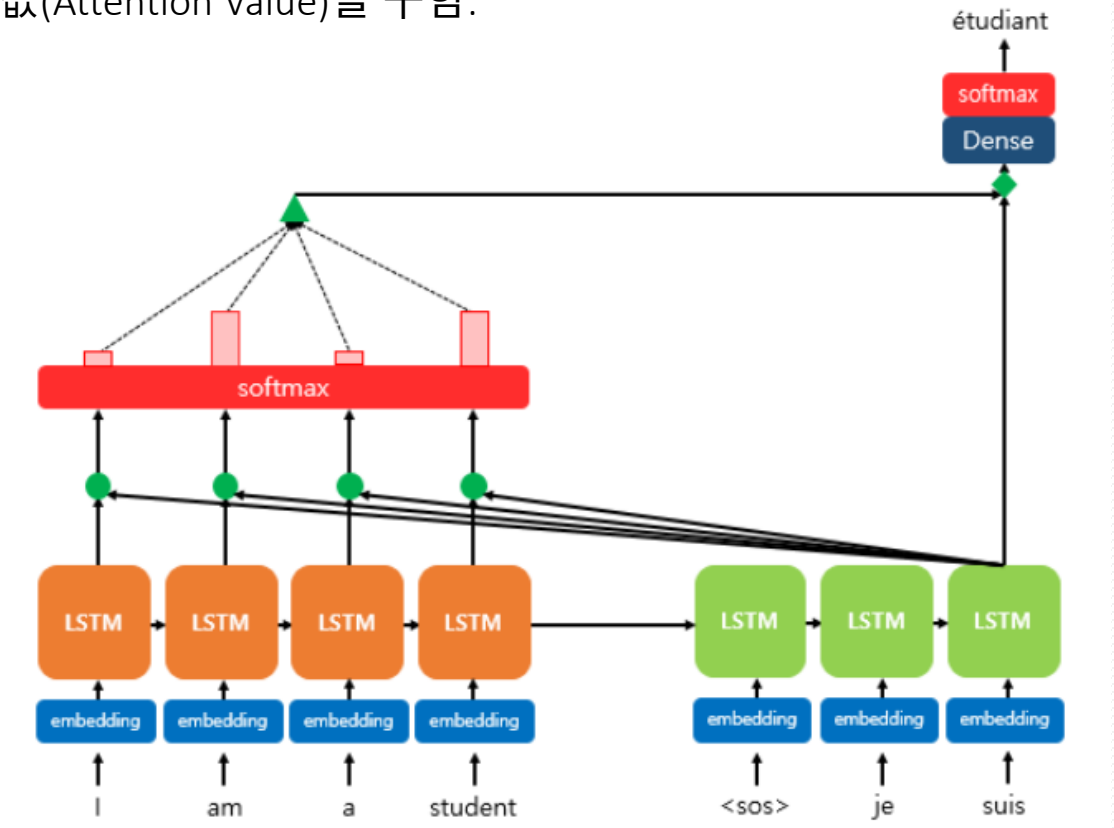
$$[\boldsymbol{c}; \boldsymbol{h}^*]$$

5) 출력층 연산의 입력이 되는 $\tilde{h}$ (or $\tilde{S}_t$)를 계산.

$$\tilde{\boldsymbol{s}}_{\boldsymbol{t}} = \tanh(\boldsymbol{W}_{\boldsymbol{c}}[\boldsymbol{c}; \boldsymbol{h}^*])$$

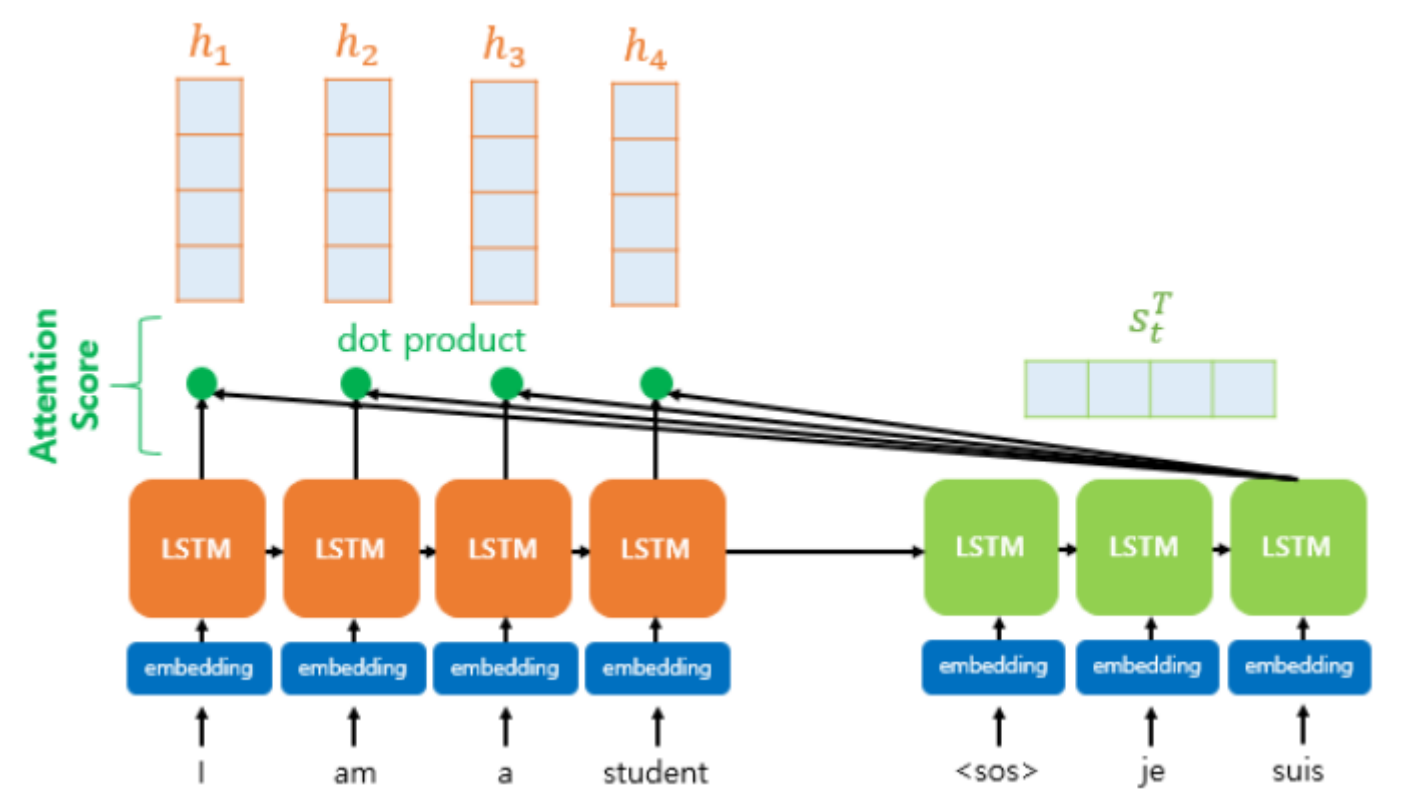
6) $\tilde{S}_t$ 를 출력층의 입력으로 사용하여 예측 벡터를 얻음.

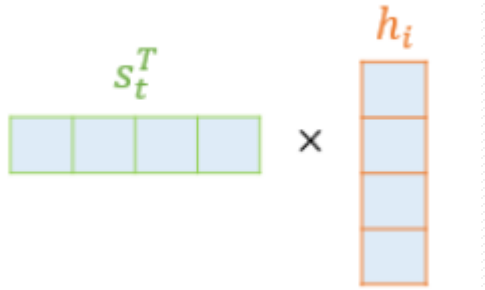
$$\boxed{p(y|\mathbf{x}) = \text{softmax}(\mathbf{W}_\mathbf{s}\tilde{\mathbf{h}})}$$



Attention : 닷-프로덕트 어텐션(Dot-Product Attention)

1) 어텐션 스코어(Attention Score)를 구함.





$$score(s_t,\ h_i) = s_t^T h_i$$

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N]$$

$$e_{ij} = f(s_{i-1}, h_j)$$

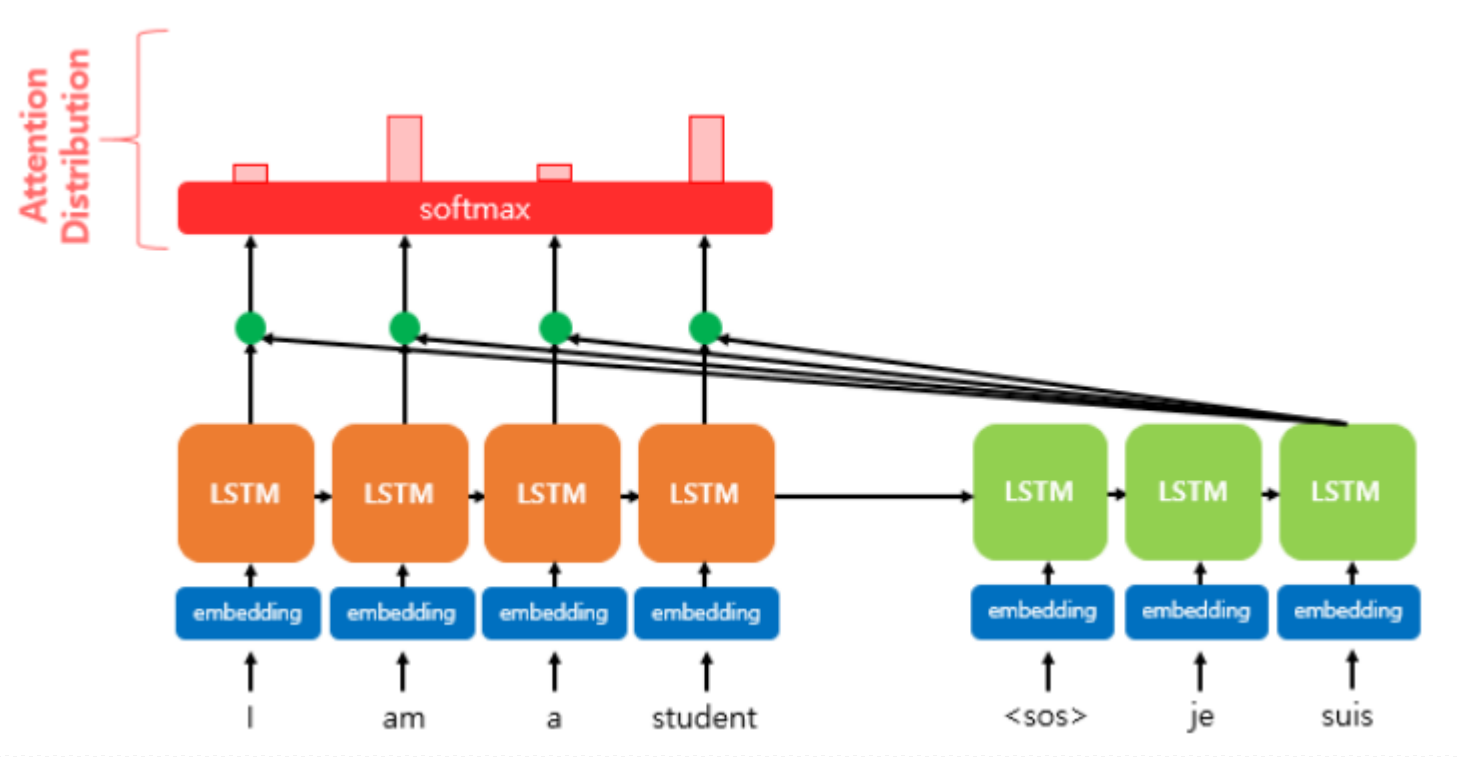
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

$$\alpha^t = softmax(e^t)$$

$$c_t = \sum_i \alpha_i h_i$$

Attention : 닷-프로덕트 어텐션(Dot-Product Attention)

2) 소프트맥스(softmax) 함수를 통해 어텐션 분포(Attention Distribution)를 구함..



$$score(s_t,\ h_i) = s_t^T h_i$$

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N]$$

$$e_{ij} = f(s_{i-1}, h_j)$$

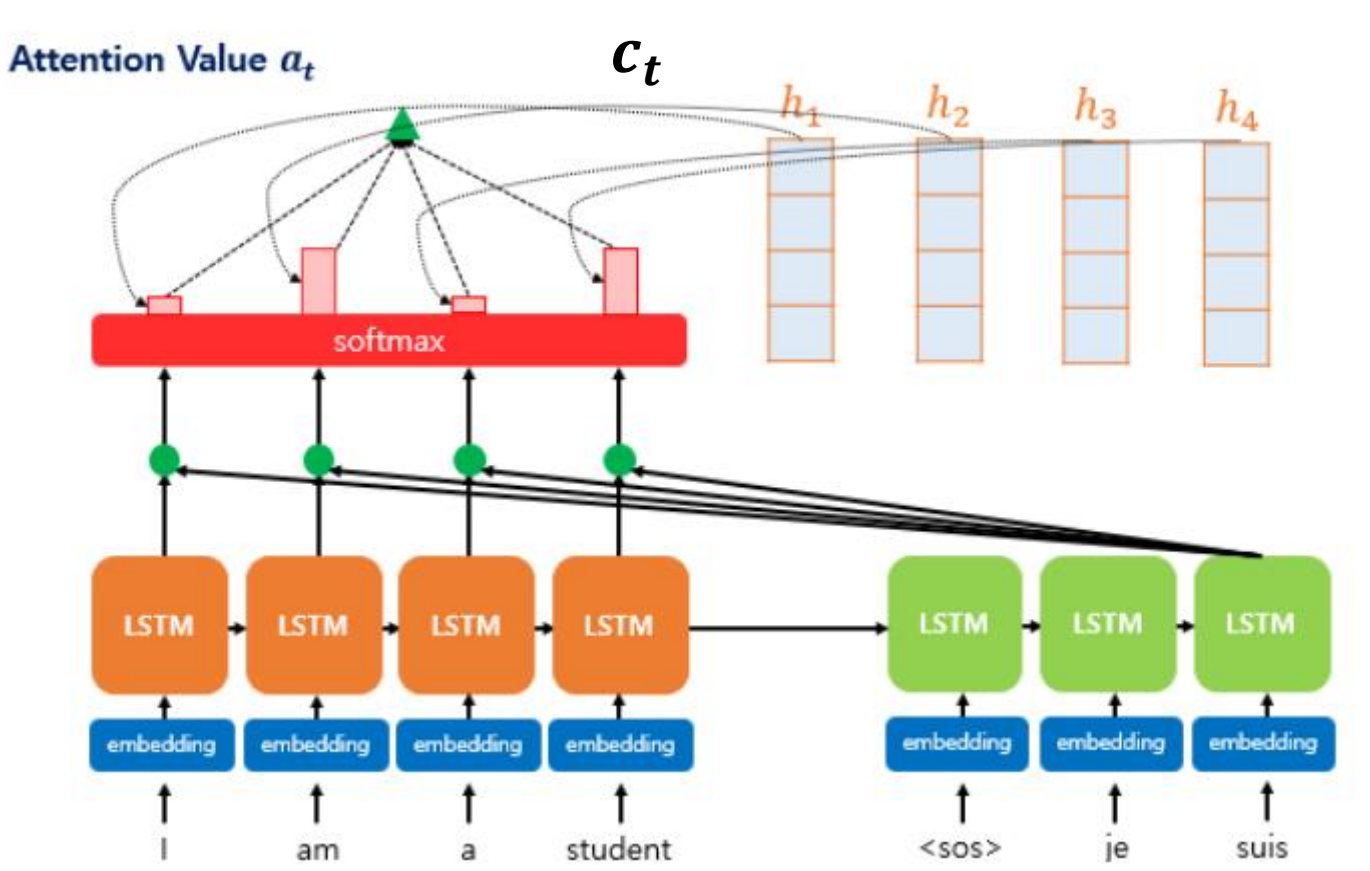
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

$$\alpha^t = softmax(e^t)$$

$$c_t = \sum_i \alpha_i h_i$$

Attention : 닷-프로덕트 어텐션(Dot-Product Attention)

3) 각 인코더의 어텐션 가중치와 은닉 상태를 가중합하여 어텐션 값(Attention Value)을 구함.



$$score(s_t,\ h_i) = s_t^T h_i$$

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N]$$

$$e_{ij} = f(s_{i-1}, h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

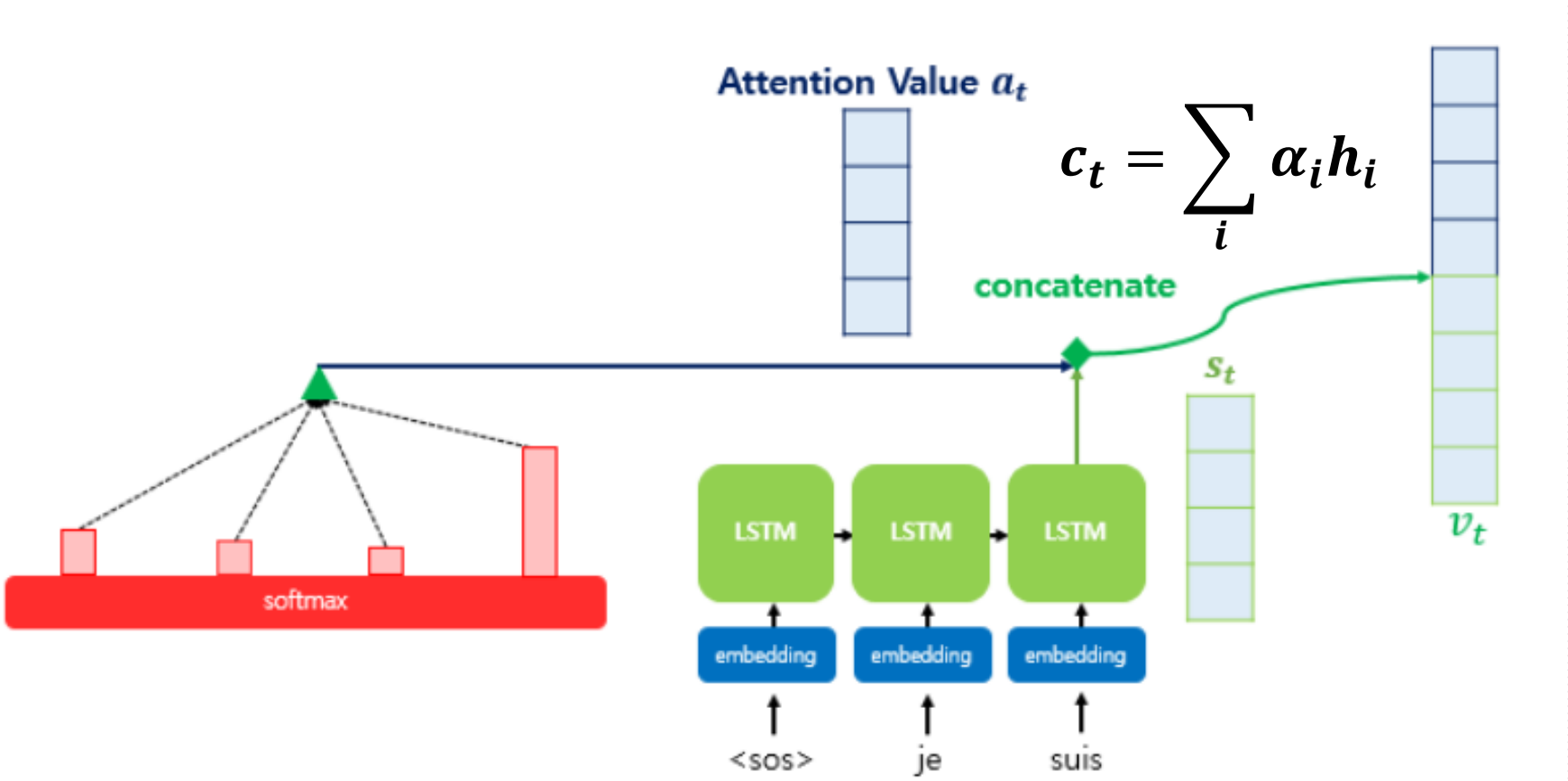
$$\alpha^t = softmax(e^t)$$

Context vector

$$c_t = \sum_i \alpha_i h_i$$

Attention : 닷-프로덕트 어텐션(Dot-Product Attention)

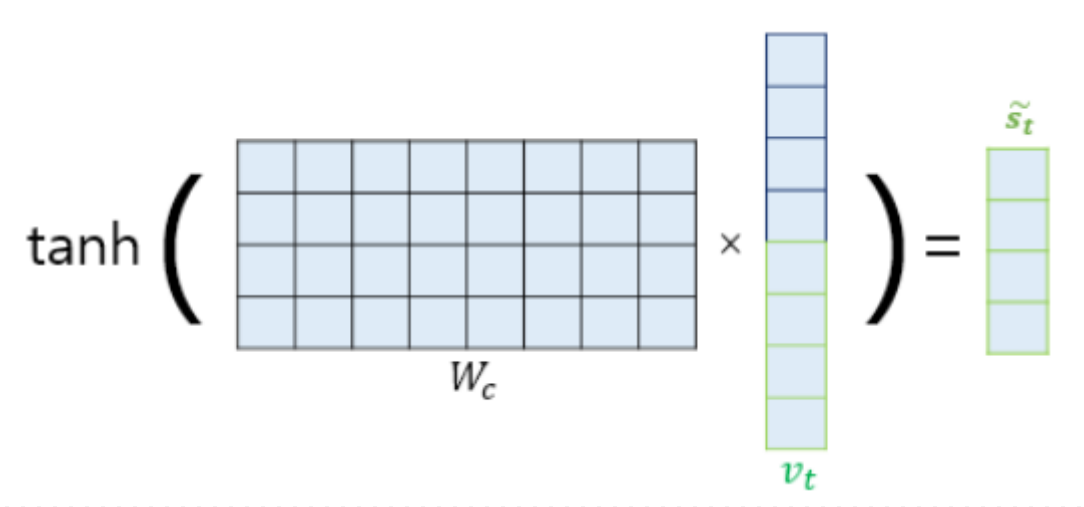
4) 어텐션 값과 디코더의 t 시점의 은닉 상태를 연결.(Concatenate)



$[c_t; s_t]$

Attention : 닷-프로덕트 어텐션(Dot-Product Attention)

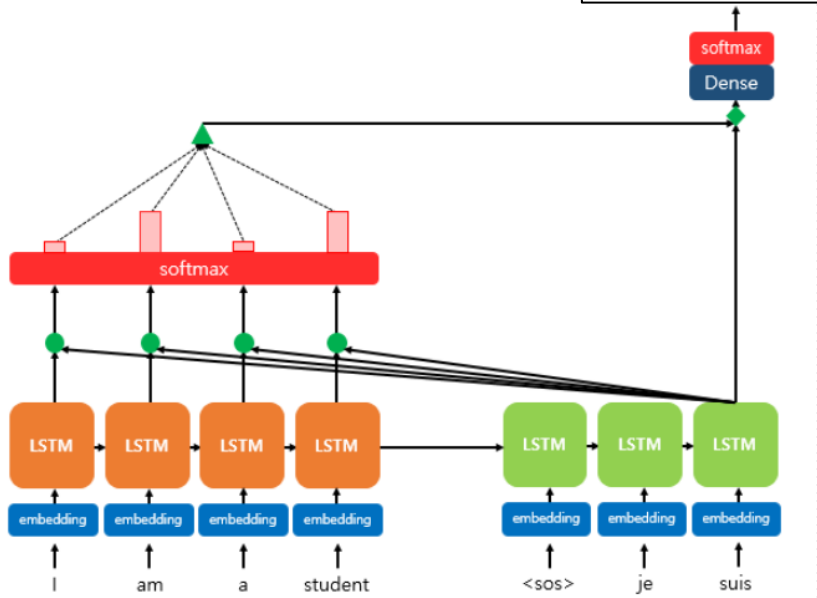
5) 출력층 연산의 입력이 되는 $\tilde{S}_t$ 를 계산.
6) $\tilde{S}_t$ 를 출력층의 입력으로 사용하여 예측 벡터를 얻음.



$$\tilde{\boldsymbol{s}}_{\boldsymbol{t}} = \tanh(\boldsymbol{W}_{\boldsymbol{c}}[\boldsymbol{c}_{\boldsymbol{t}}; \boldsymbol{s}_{\boldsymbol{t}}] + \boldsymbol{b}_{\boldsymbol{c}})$$

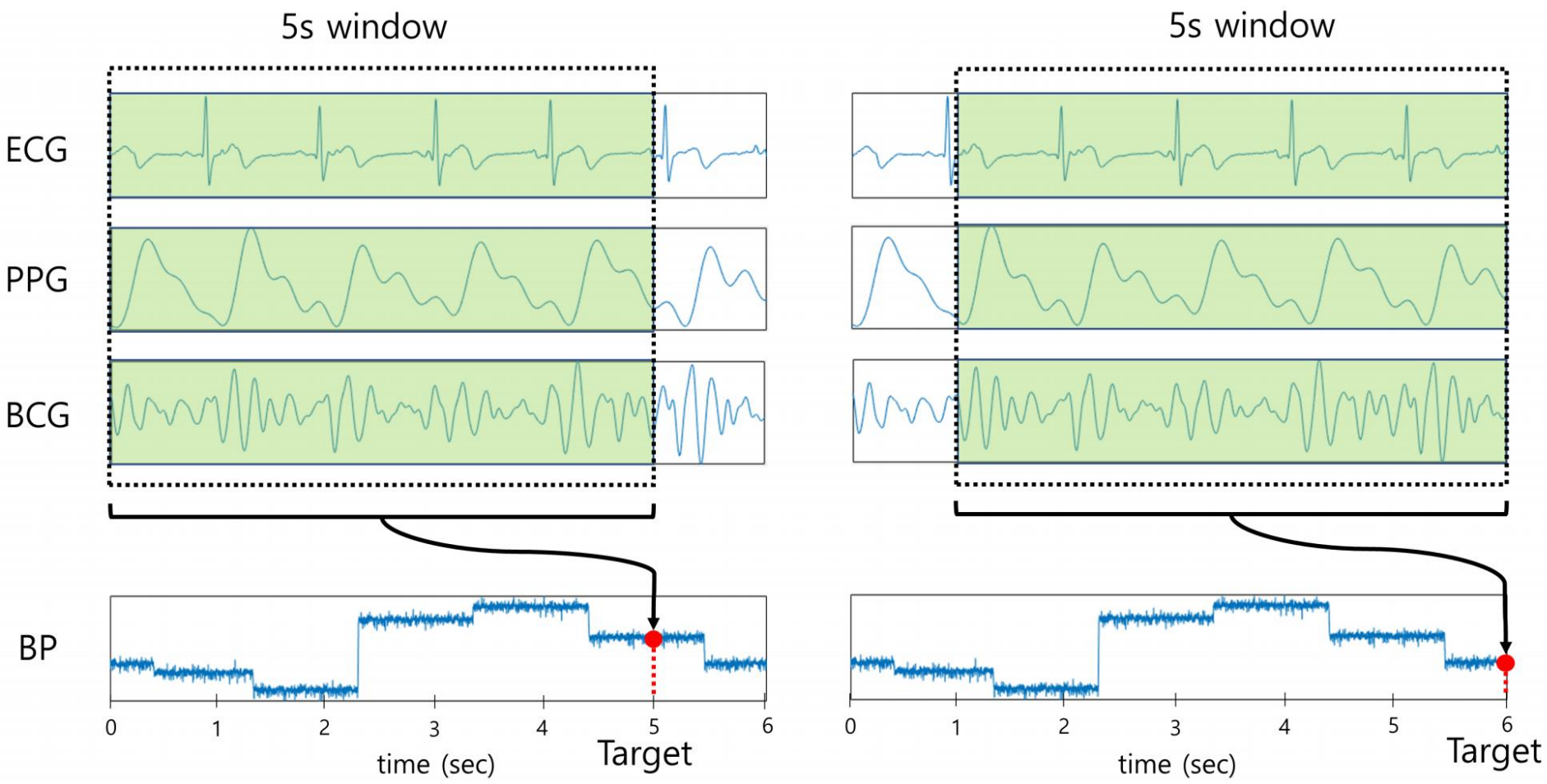
$$\hat{y}_t = \text{Softmax}\,(W_y \tilde{s}_t + b_y)$$

étudiant



Attention

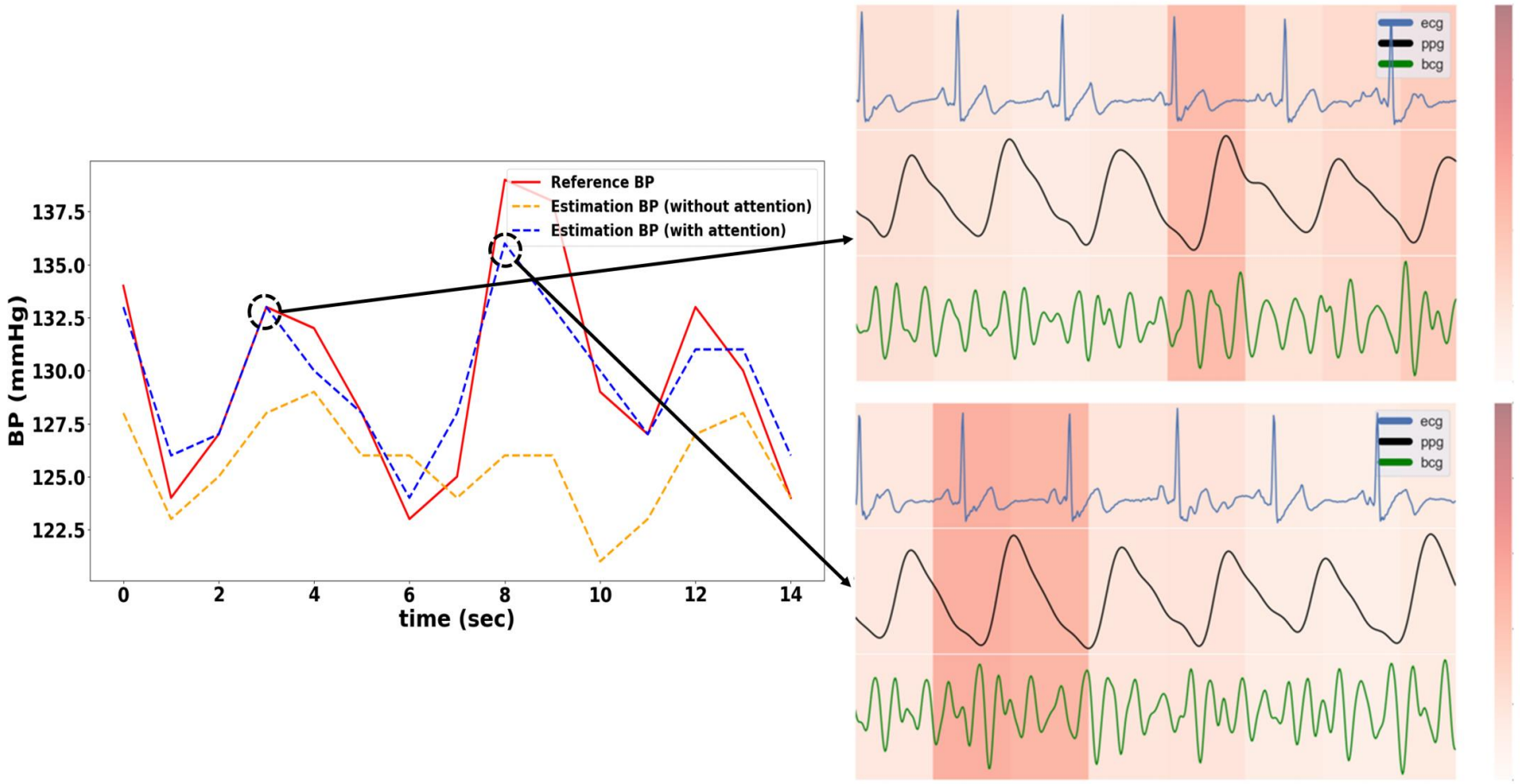
- Blood Pressure



Eom, H., Lee, D., Han, S., Hariyani, Y. S., Lim, Y., Sohn, I., ... & Park, C. (2020). End-to-end deep learning architecture for continuous blood pressure
estimation using attention mechanism. *Sensors*, *20*(8), 2338.

Attention

- Blood Pressure



Eom, H., Lee, D., Han, S., Hariyani, Y. S., Lim, Y., Sohn, I., ... & Park, C. (2020). End-to-end deep learning architecture for continuous blood pressure
estimation using attention mechanism. *Sensors*, *20*(8), 2338.

문제점

기존 Attention Mechanism의 문제점

- 입력 시퀀스 데이터를 순차적으로 처리하기 때문에 병렬 처리 불가능
- 계산 복잡도 , 연산시간
- 해결 방안 : **Self-Attention mechanism**, 입력 시퀀스 데이터를 병렬 처리하여 정보 압축하고 계산 복잡도/연산시간을 줄이자 !

Attention

Attention Mechanism

Attention이 고려된 hidden state는 context vector와 기존 hidden state를 concatenation한 뒤 선형 및 비선형 변환을 수행

$$\tilde{\mathbf{h}} = tanh(\mathbf{W_c}[\mathbf{c}; \mathbf{h}^*])$$

Context vector는 각 입력 단계의 hidden state의 선형 결합 (가중치 = alpha)

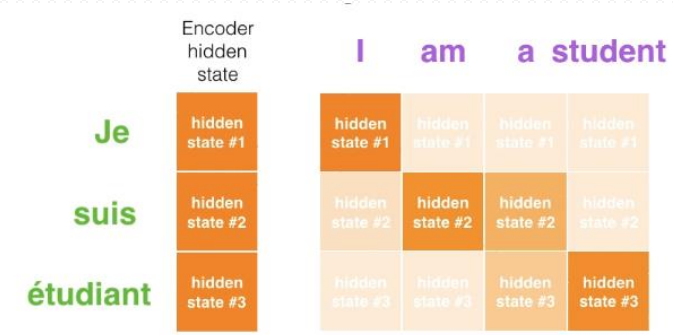
$$\mathbf{c} = \sum_i \alpha_i \mathbf{h}_i$$

i번째 hidden state vector가 context vector 생성에 기여하는 비중

$$\alpha_i = \text{align}(\mathbf{h}^*, \mathbf{h}_i) = \frac{exp\Big(\text{score}(\mathbf{h}^*, \mathbf{h}_i)\Big)}{\sum_{i'} exp\Big(\text{score}(\mathbf{h}^*, \mathbf{h}'_i)\Big)}$$

두 hidden state vector 사이의 score 산출 방식 중 가장 간단한 방식은 두 벡터의 내적을 사용하는 것

$$\text{score}(\mathbf{h}^*, \mathbf{h}_i) = \mathbf{h}^{*T}\mathbf{h}_i$$

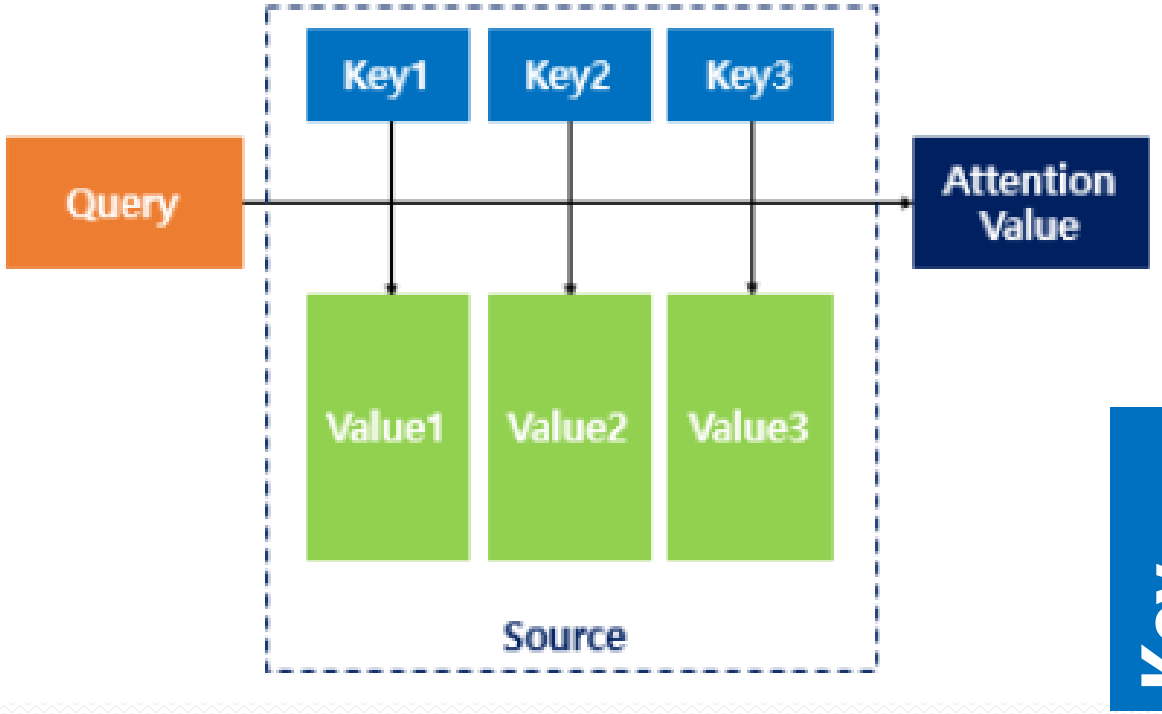


$$\boldsymbol{C} = \sum \langle \boldsymbol{h_i}, \boldsymbol{h^*_{oj}} \rangle \boldsymbol{h_i}$$

Key **Query** **Value**

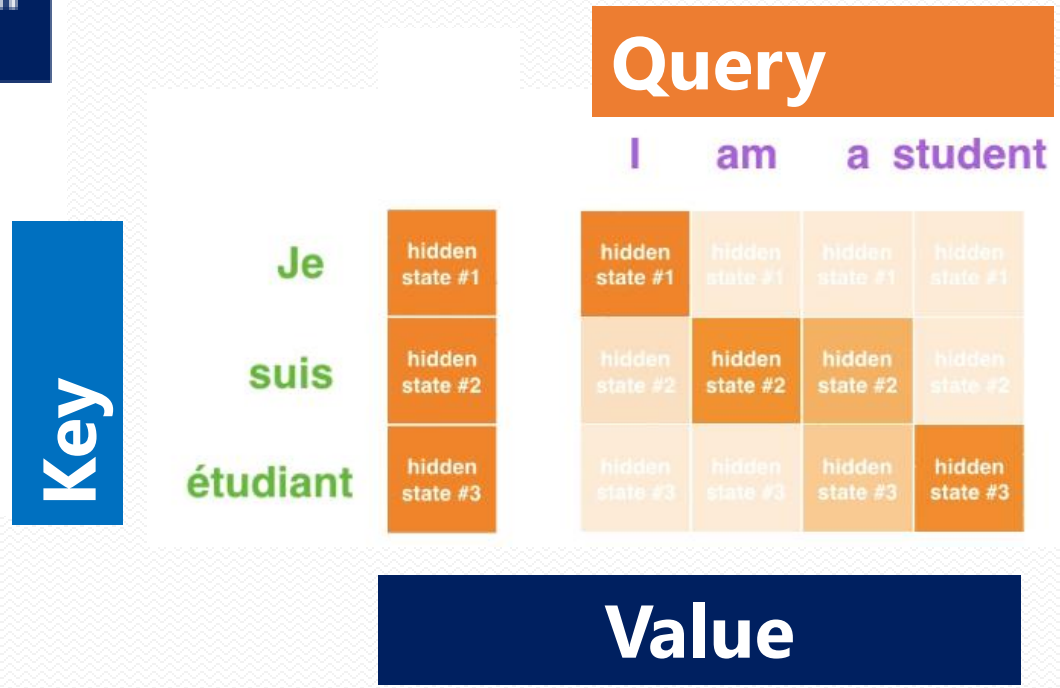
Attention

✓ Attention allows the model to focus on the relevant part of the input sequence as needed



Attention(Q, K, V) = Attention Value

Q = Query : t 시점의 디코더 셀에서의 은닉 상태
K = Keys : 모든 시점의 인코더 셀의 은닉 상태들
V = Values : 모든 시점의 인코더 셀의 은닉 상태들



Contents

Transformer

January 16, 2013

WORD2VEC

Word2Vec Paper by Tomas Mikolov et al

January 2, 2014

GLOVE

GloVe: Global Vectors for Word Representation by Jeffrey Pennington et al.

July 15, 2016

FASTTEXT

Enriching Word Vectors with Subword Information by Piotr Bojanowski et al

February 15, 2018

ELMO

Deep contextualized word representations by Matthew E. Peters et al

June 12, 2017

TRANSFORMER

Attention Is All You Need by Ashish Vaswani et al

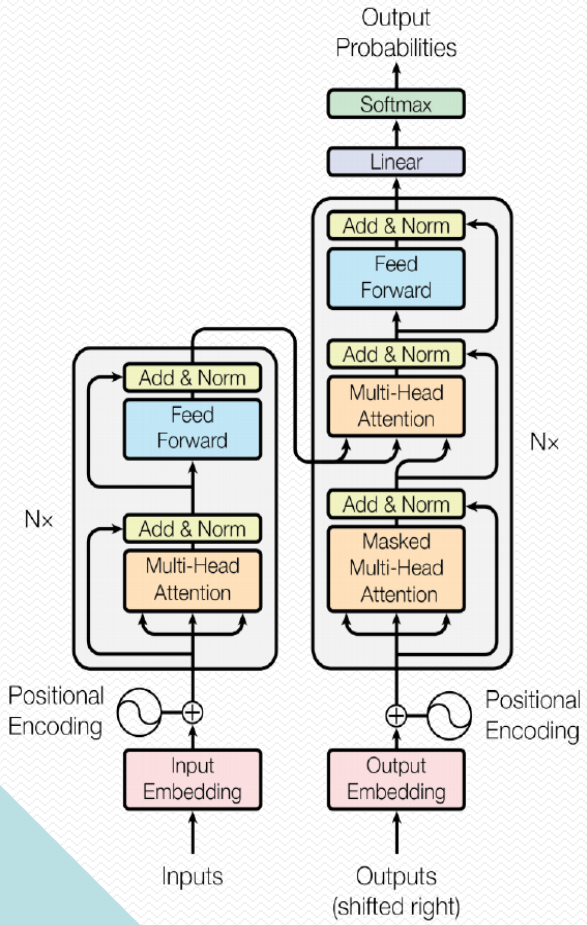
October 11, 2018

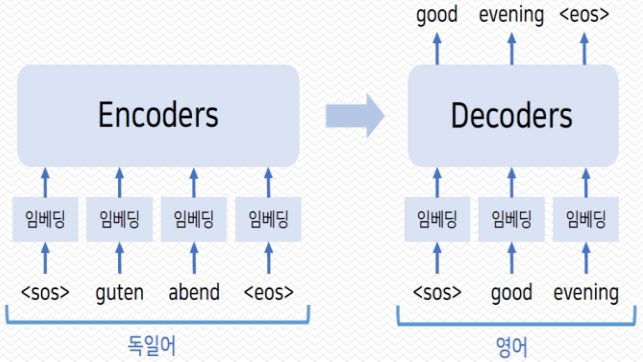
BERT

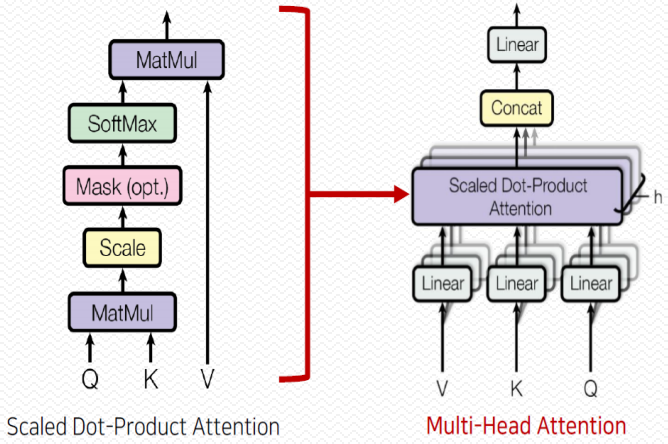
BERT: Pre-training of Deep Bidirectional Transformers for...

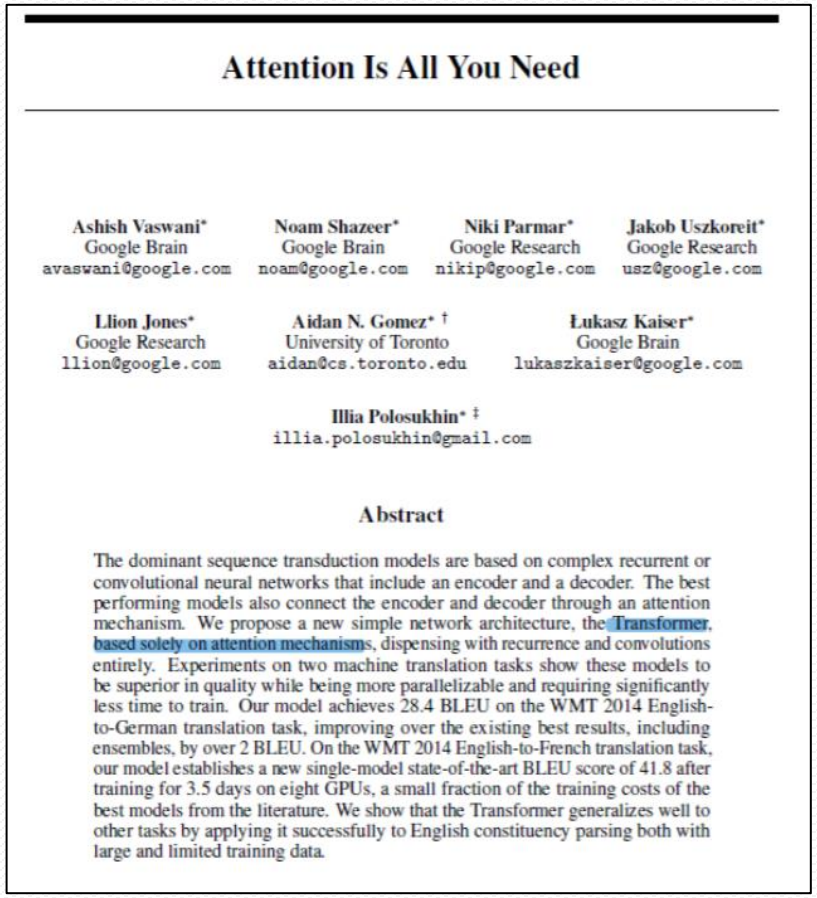
Attention Is All You Need

Attention Is All You Need (Google Brain, Google Research, University of Toronto)
- 2017년 Neural Information Processing Systems(Neural IPS)에서 발표된 논문
- 2024년 2월 현재 106,941회 인용
- A model that uses attention to boost the speed with which these models can be trained and easy to parallelize









Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

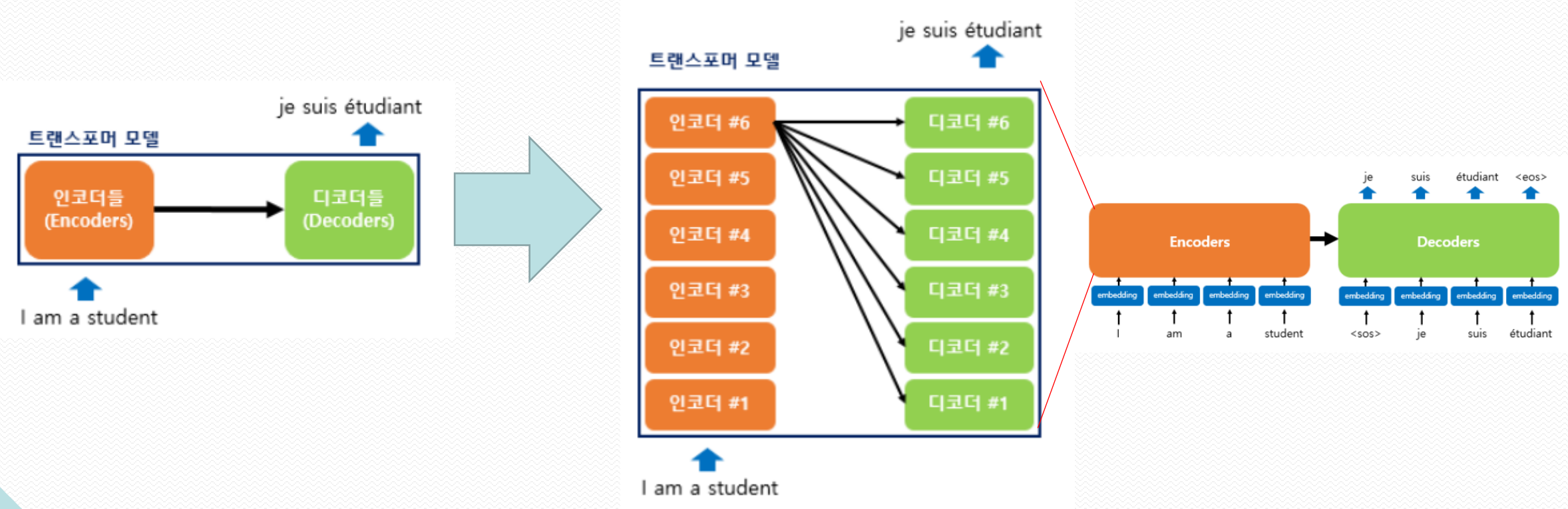
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Transformer

Encoder-Decoder

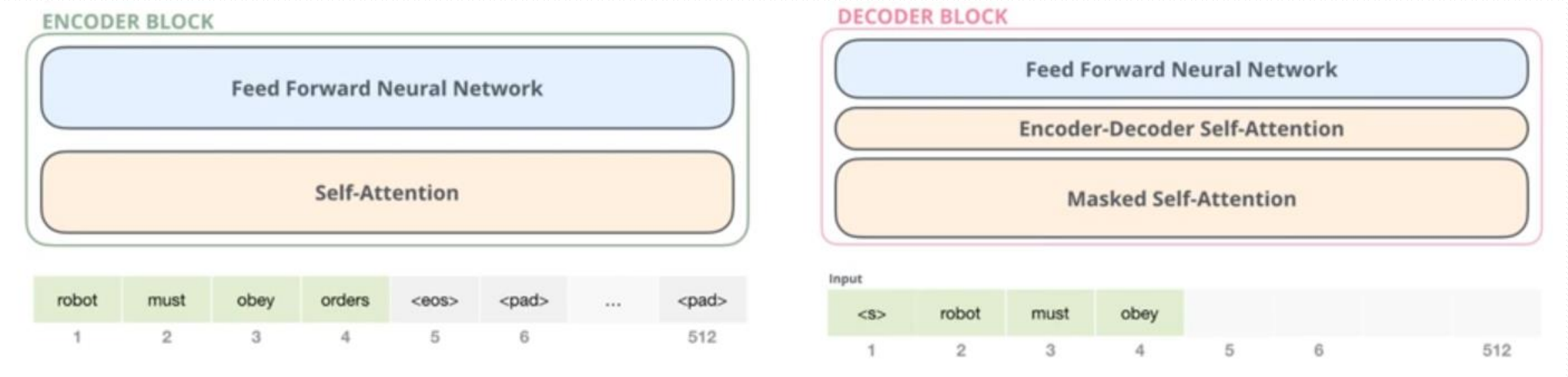
- Transformer는 Encoder-Decoder 구조의 모델 아키텍쳐
- 세부적으로 보면 6개의 Encoder-Decoder로 이루어져 있음.



Transformer

Encoder-Decoder

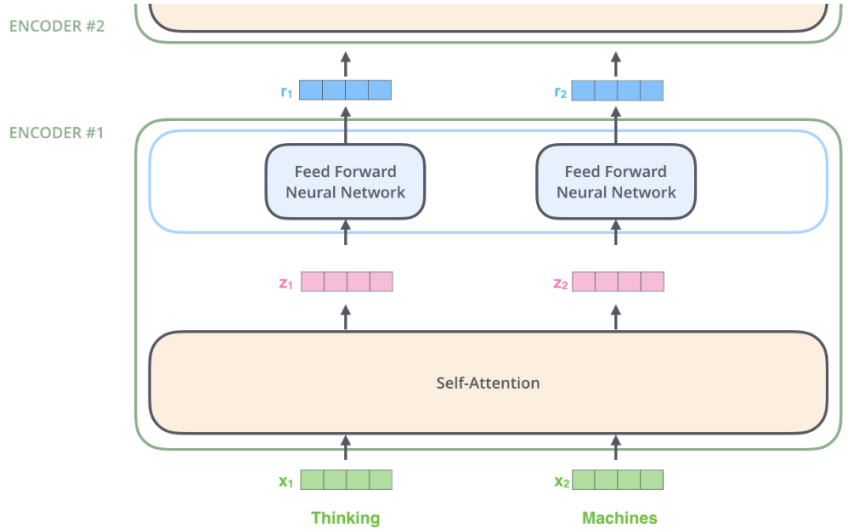
- Encoding block vs. Decoding block = Unmasked vs. Masked



Transformer

The encoder are all identical in structure (does not mean that they share the weights), each of which is broken down into two sub-layers

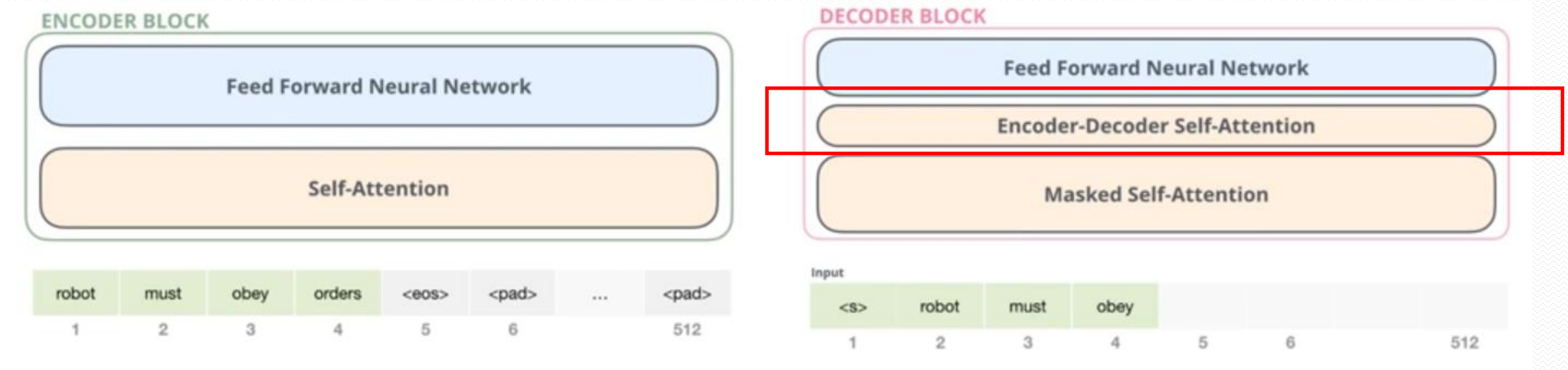
- The encoder's input first flow through a self-attention layer (a layer that helps theencoder look at other words in the input sentence as it encodes a specific word)
- The output of the self-attention layer are fed to a feed-forward neural network
- The exact same feed-forward network is idependently applied to each position



The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

Transformer

The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence



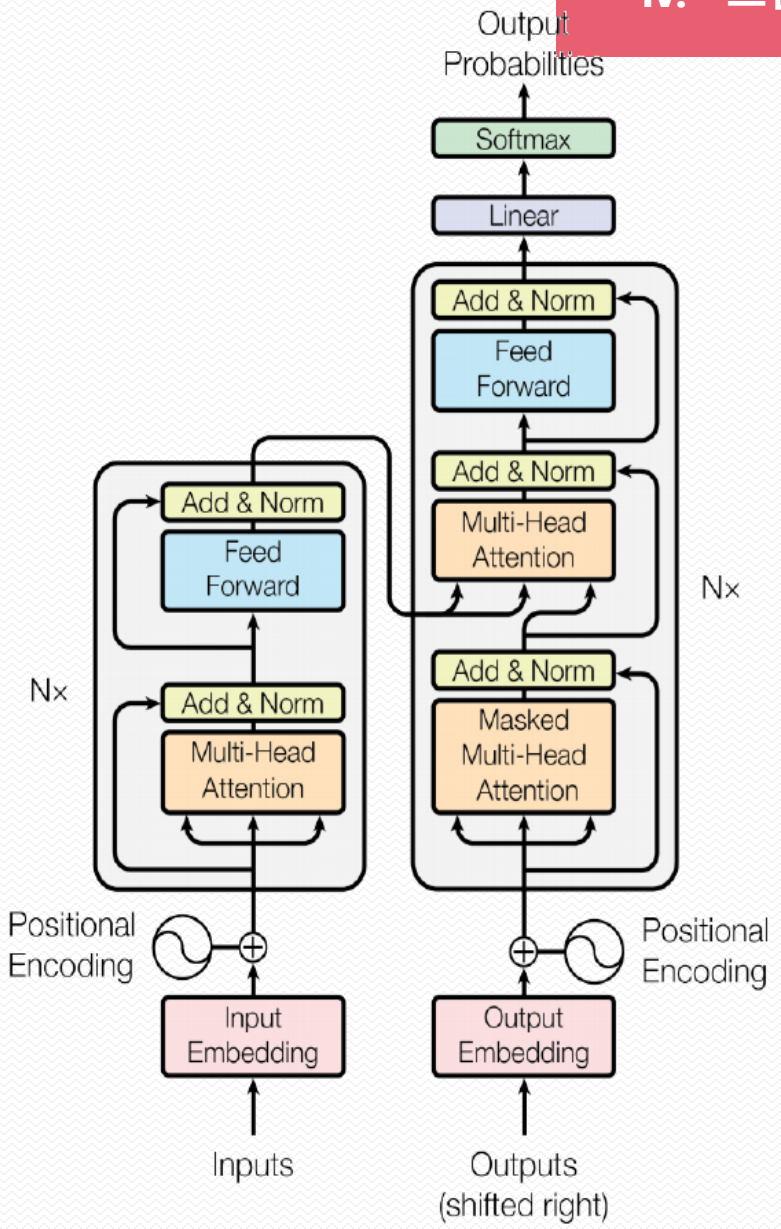
Transformer

Embedding

- The embedding only happens in the bottom-most encoder
- The abstraction that is common to all the encoders is that they receive a list of vectors each of the size 512
- In the bottom encoder that would be the word embeddings, but in other encoders, it would be the output of the encoder that is directly below
- The size of this list is a hyperparameter we can set - basically it would be the length of thelongest sentence in our training dataset

$$d_{model} = 512 \quad \text{num\_heads} = 8$$

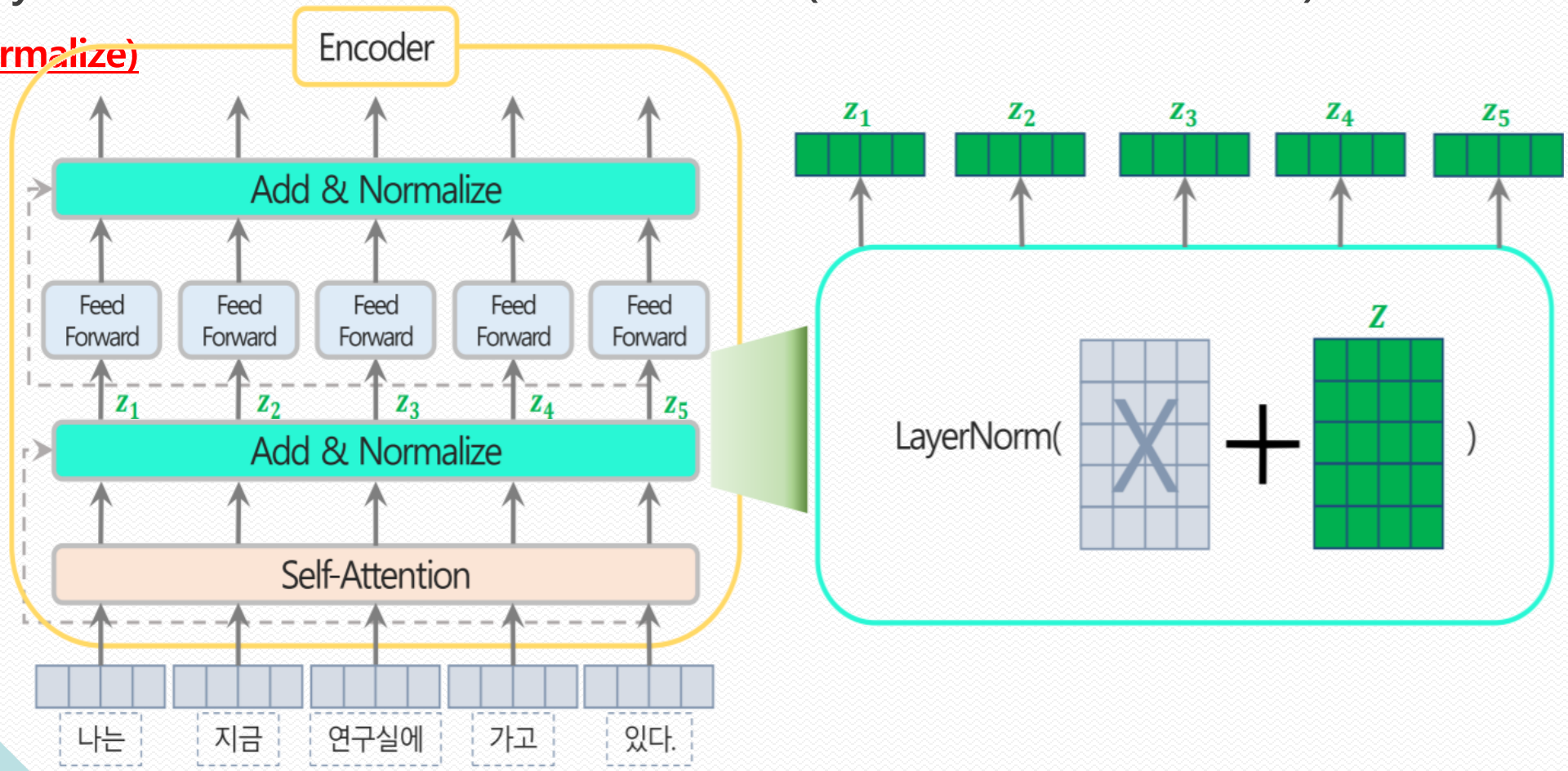
$$\text{num\_layers} = 6 \quad d_{ff} = 2048$$



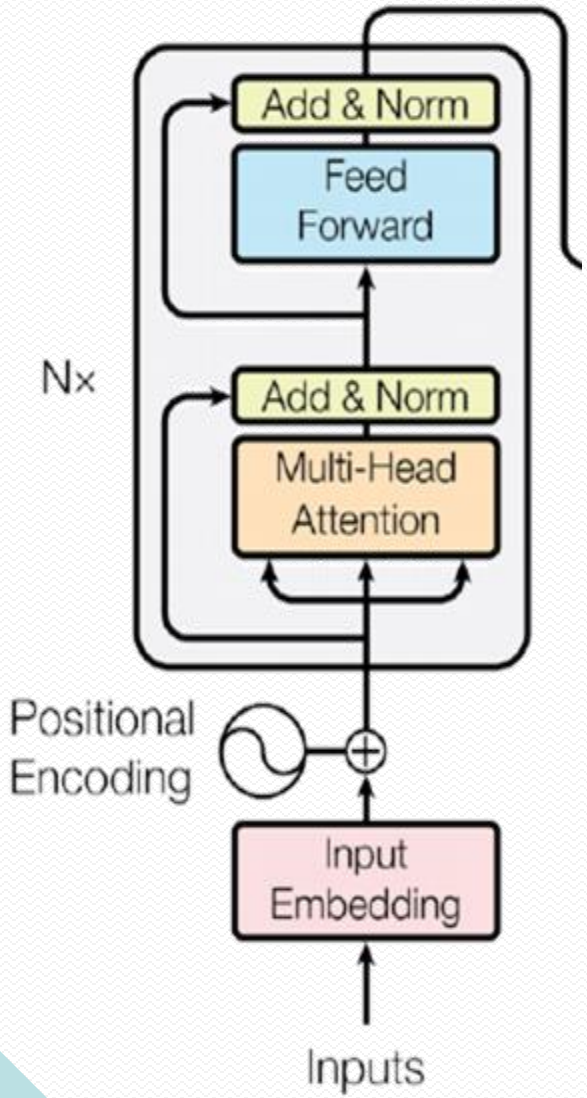
Transformer

Encoder details

- Residual connection : 입력 데이터와 Self-Attention의 결과를 더함 (Add)
- Layer normalization : residual connection 결과 (입력 'x' + context vector 'z')에 대한 정규화 진행 (Normalize)

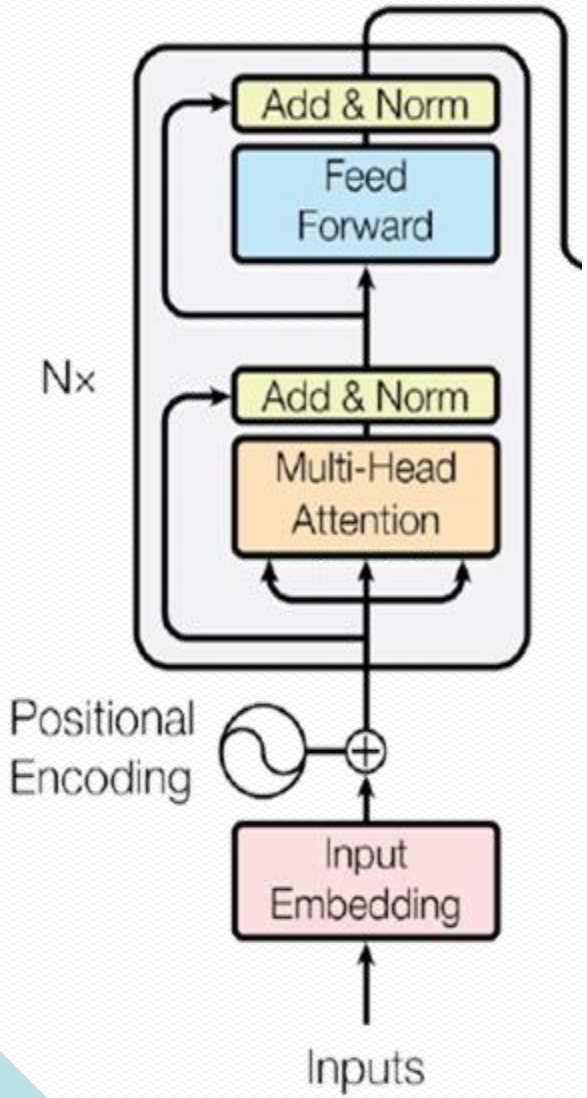


Transformer




```python
def encoder_layer(dff, d_model, num_heads, dropout, name="encoder_layer"):
  inputs = tf.keras.Input(shape=(None, d_model), name="inputs")

  # 인코더는 패딩 마스크 사용
  padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")

  # 멀티-헤드 어텐션 (첫번째 서브층 / 셀프 어텐션)
  attention = MultiHeadAttention(
      d_model, num_heads, name="attention")({
          'query': inputs, 'key': inputs, 'value': inputs, # Q = K = V
          'mask': padding_mask # 패딩 마스크 사용
      })

  # 드롭아웃 + 잔차 연결과 층 정규화
  attention = tf.keras.layers.Dropout(rate=dropout)(attention)
  attention = tf.keras.layers.LayerNormalization(
      epsilon=1e-6)(inputs + attention)

  # 포지션 와이즈 피드 포워드 신경망 (두번째 서브층)
  outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention)
  outputs = tf.keras.layers.Dense(units=d_model)(outputs)

  # 드롭아웃 + 잔차 연결과 층 정규화
  outputs = tf.keras.layers.Dropout(rate=dropout)(outputs)
  outputs = tf.keras.layers.LayerNormalization(
      epsilon=1e-6)(attention + outputs)

  return tf.keras.Model(
      inputs=[inputs, padding_mask], outputs=outputs, name=name)
```

Transformer

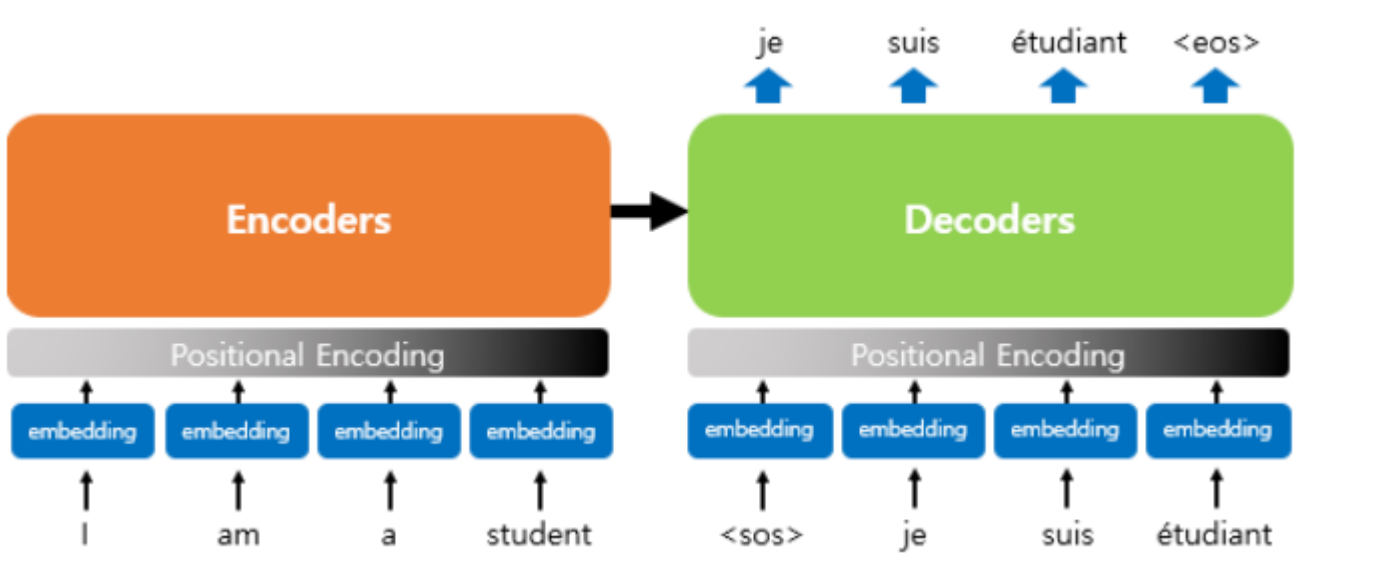


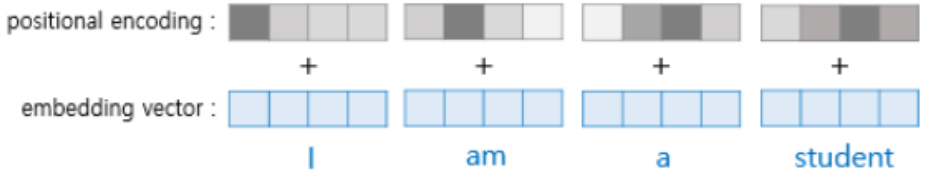

```python
def encoder(vocab_size, num_layers, dff,
            d_model, num_heads, dropout,
            name="encoder"):
  inputs = tf.keras.Input(shape=(None,), name="inputs")

  # 인코더는 패딩 마스크 사용
  padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")

  # 포지셔널 인코딩 + 드롭아웃
  embeddings = tf.keras.layers.Embedding(vocab_size, d_model)(inputs)
  embeddings *= tf.math.sqrt(tf.cast(d_model, tf.float32))
  embeddings = PositionalEncoding(vocab_size, d_model)(embeddings)
  outputs = tf.keras.layers.Dropout(rate=dropout)(embeddings)

  # 인코더를 num_layers개 쌓기
  for i in range(num_layers):
    outputs = encoder_layer(dff=dff, d_model=d_model, num_heads=num_heads,
        dropout=dropout, name="encoder_layer_{}".format(i),
    )([outputs, padding_mask])

  return tf.keras.Model(
      inputs=[inputs, padding_mask], outputs=outputs, name=name)
```

Transformer

Positional Encoding

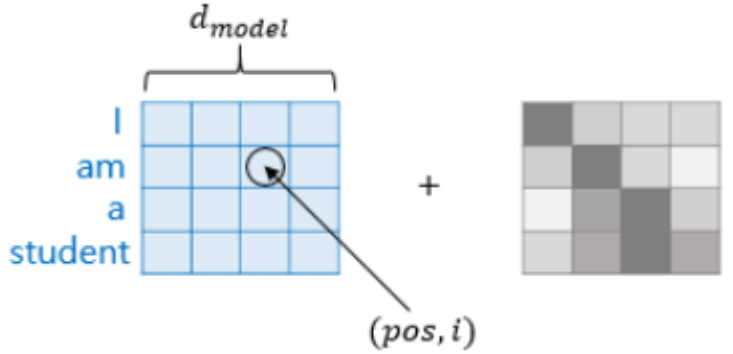
- Self-Attention은 RNN과 달리 입력 시퀀스 데이터를 순차적으로 처리하지 않음.
- Positional Encoding : 입력 시퀀스에서 단어의 순서를 표현하기 위한 임베딩 방법(-1 ~ 1)



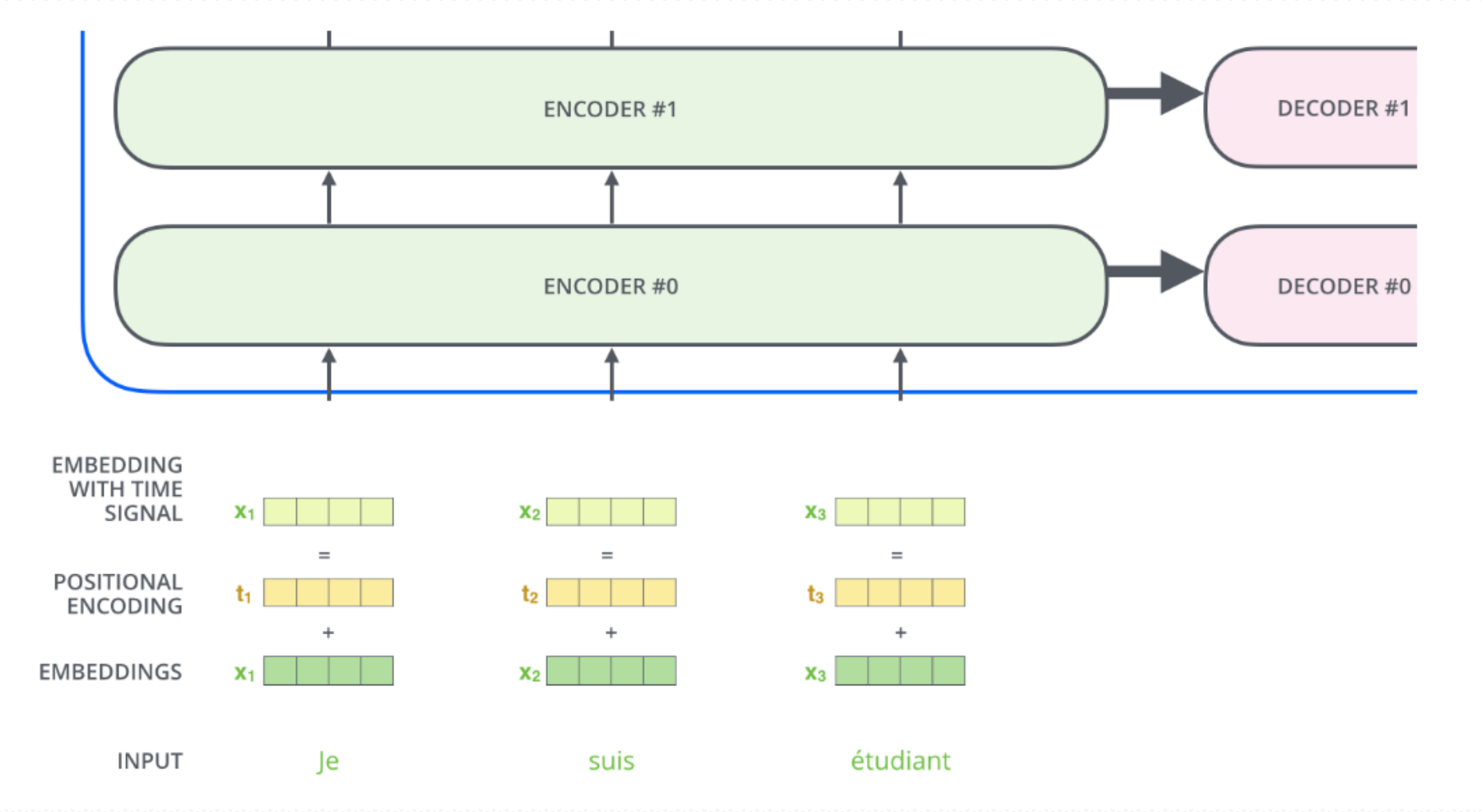


$$PE_{(pos,\,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,\,2i+1)} = cos(pos/10000^{2i/d_{model}})$$



Transformer

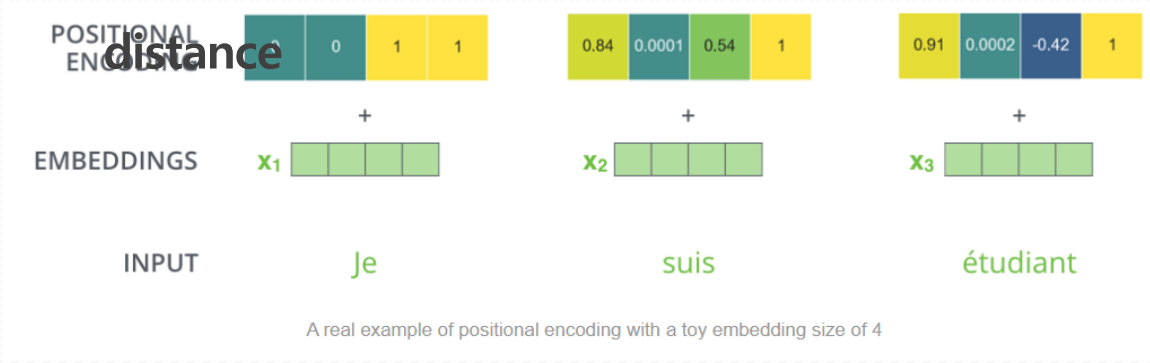


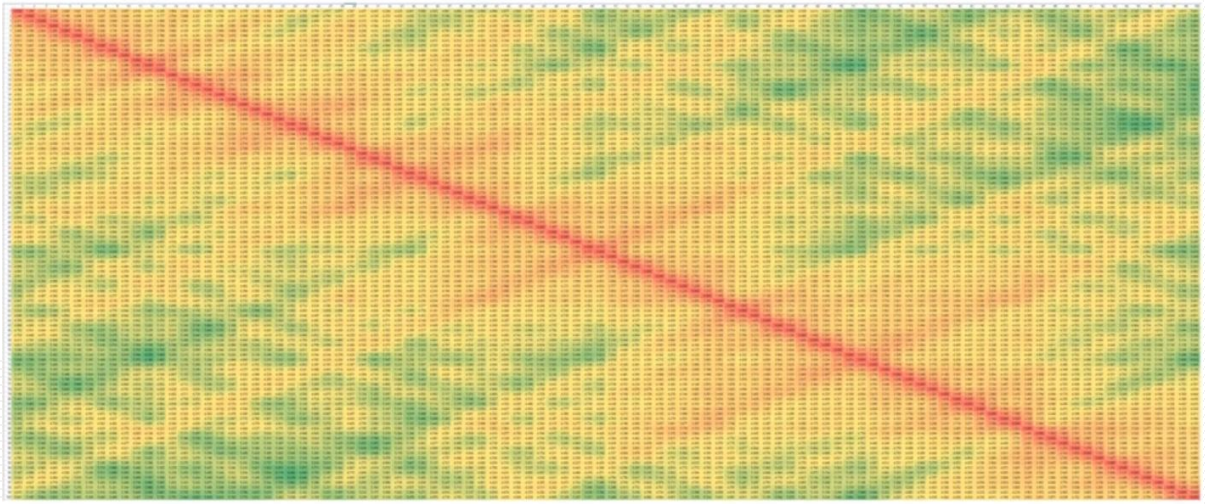
Transformer

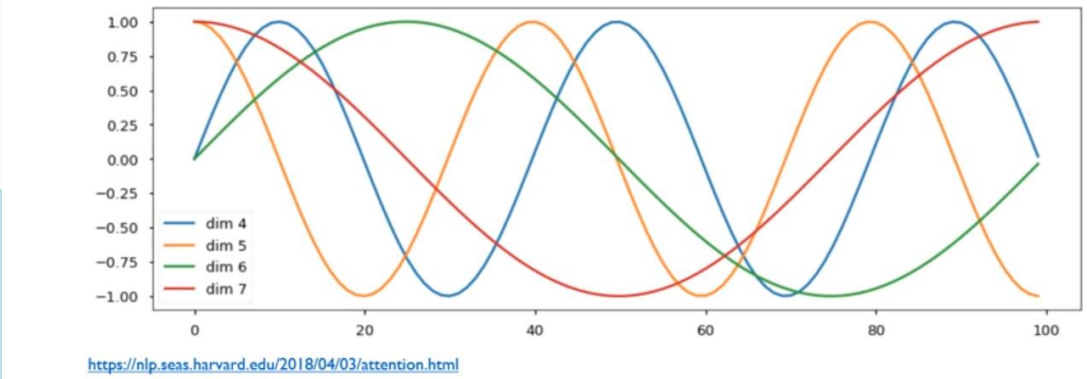
Positional Encoding

- The norm of encoding vector is the same for all positions.

- The further the two positions, the larger the distance

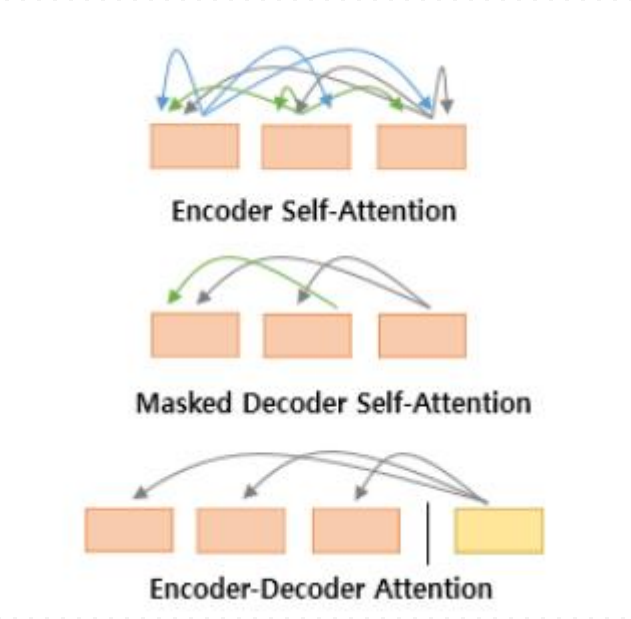


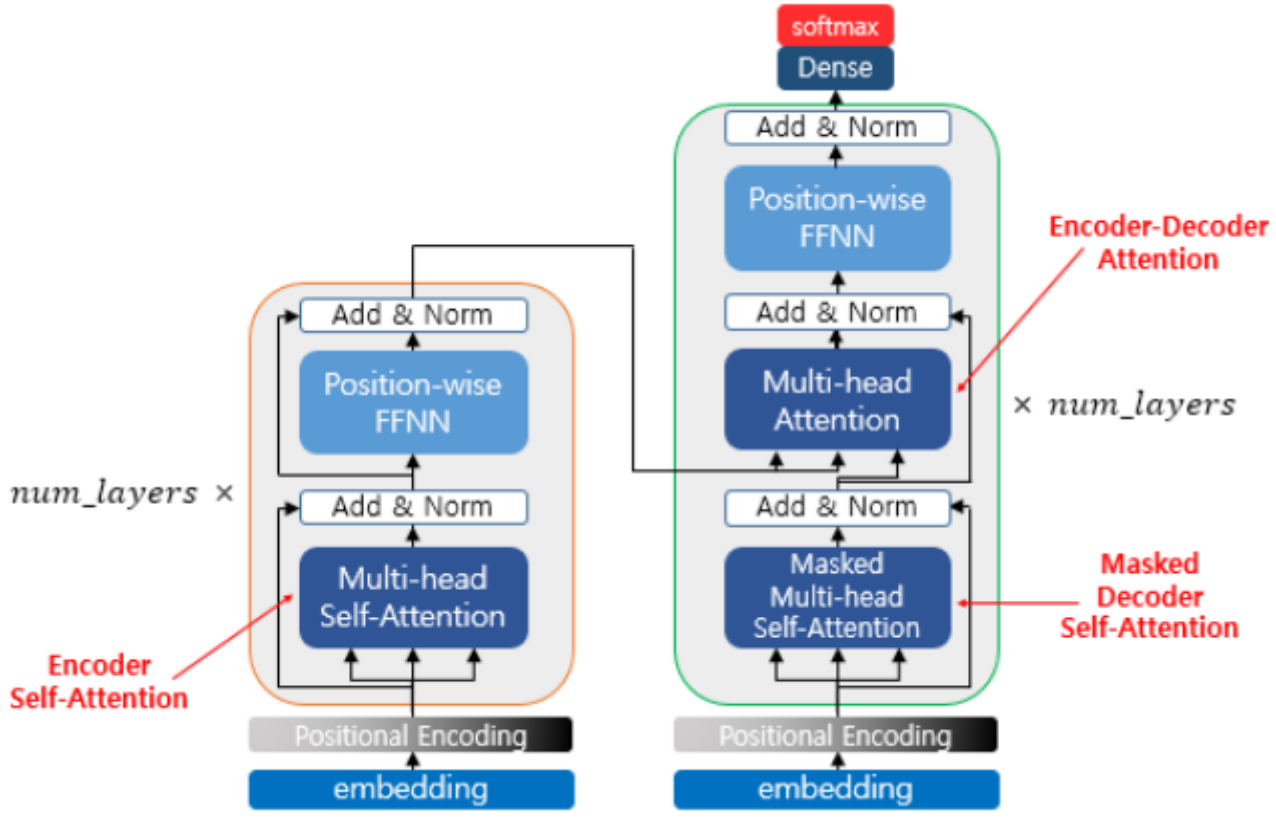




Transformer :

Attention





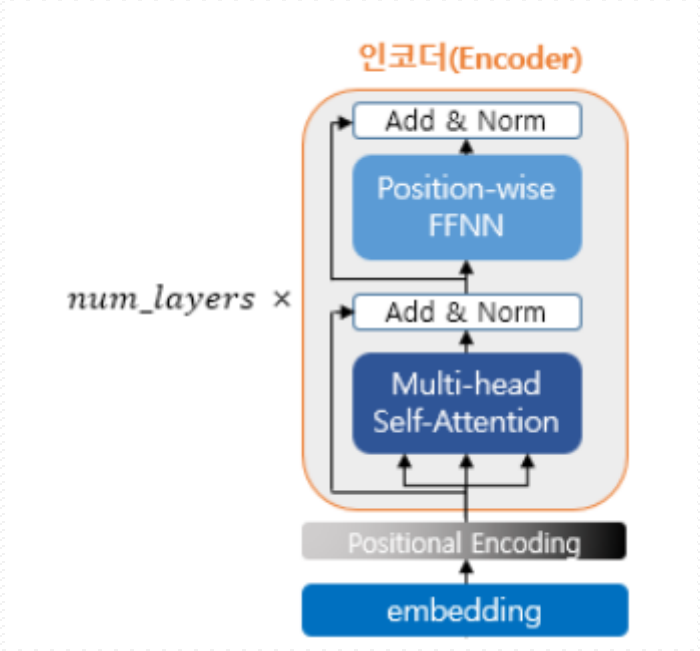
Transformer :

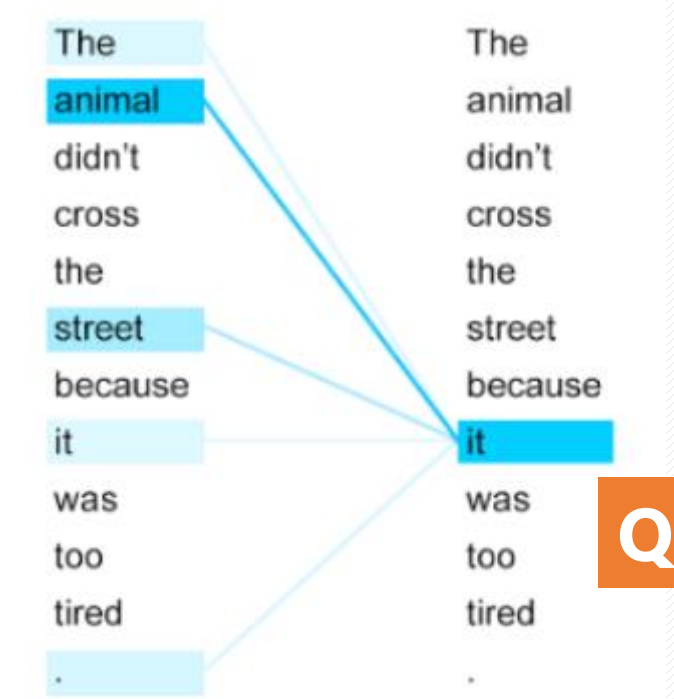
Encoder self-Attention

- Input sentence to translate: “The animal did't cross the street because it was too tired”
- What does "it" refer to? street or animal? : Simple question to a human but not as simple to an algorithm
- Self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word
- Self-attention is the method the Transformer uses to bake the "understanding" of other relevant words into the one we're currently processing

Transformer :

Encoder self-Attention

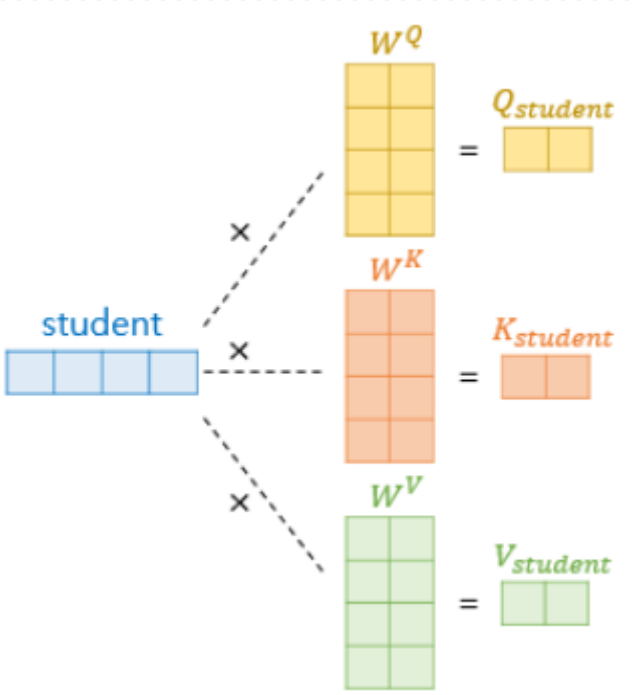




Query

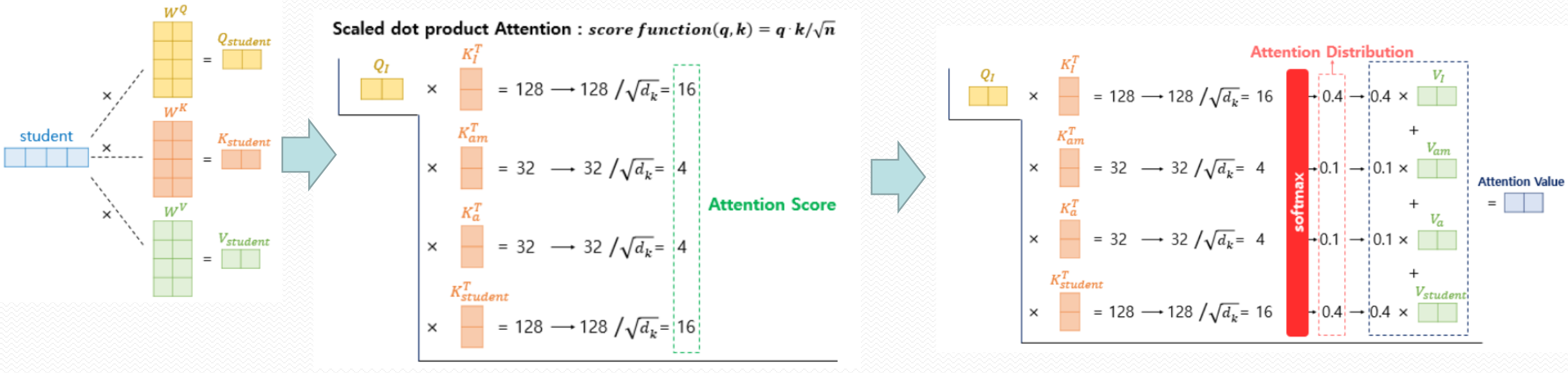
Key

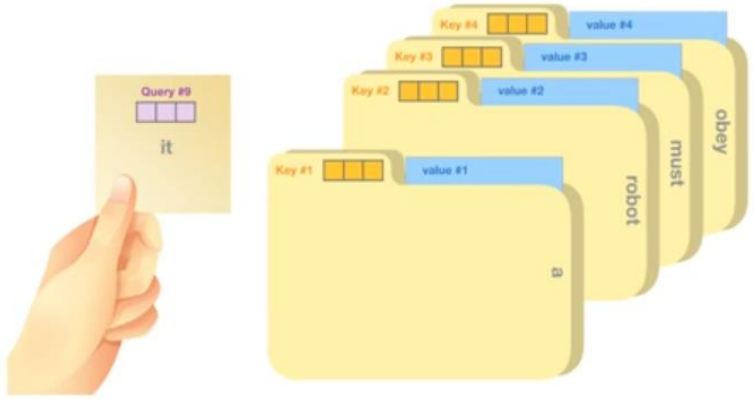
Value



Transformer :

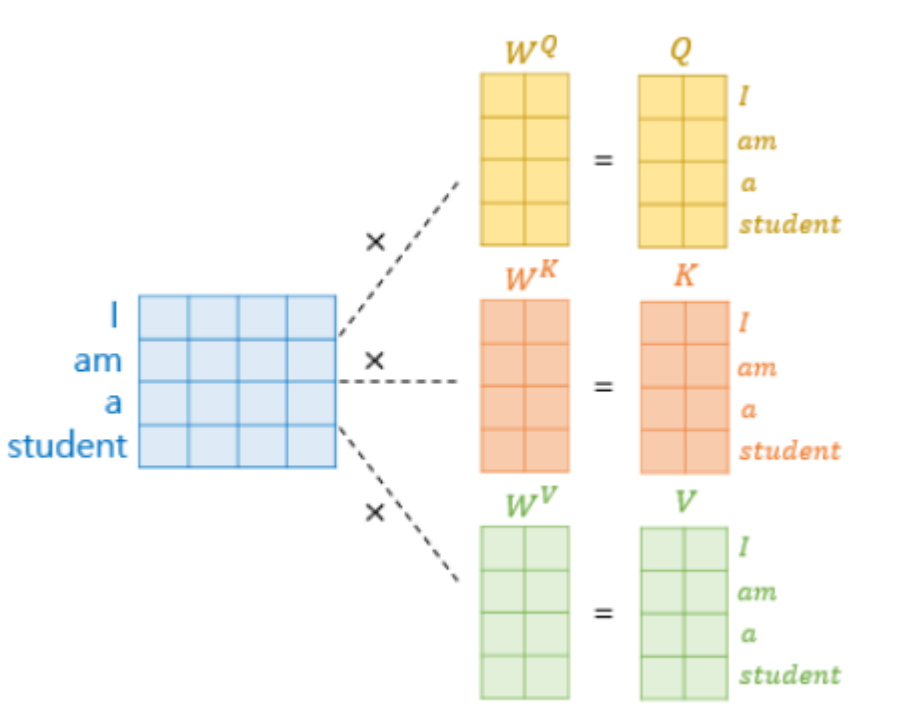
Encoder self-Attention

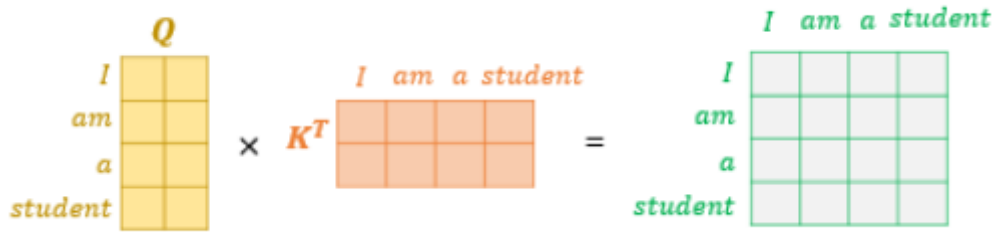


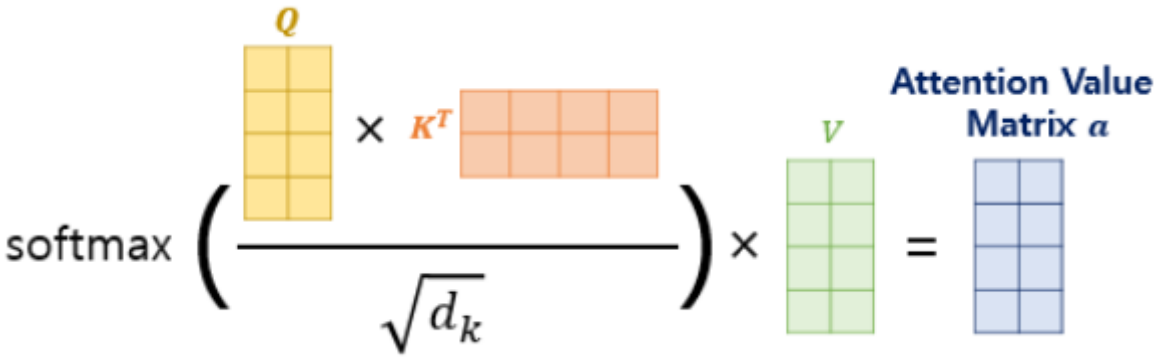


Transformer :

Encoder self-Attention



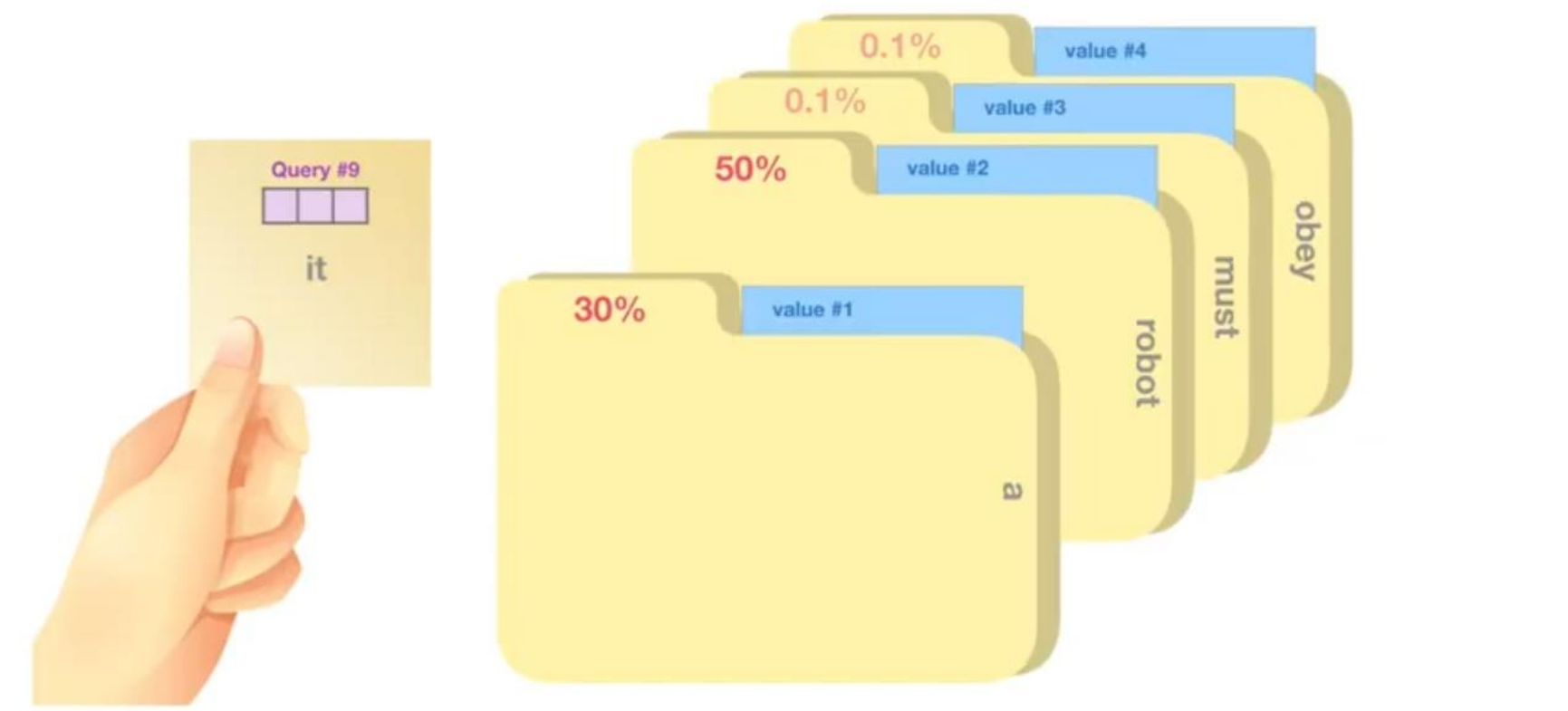




$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

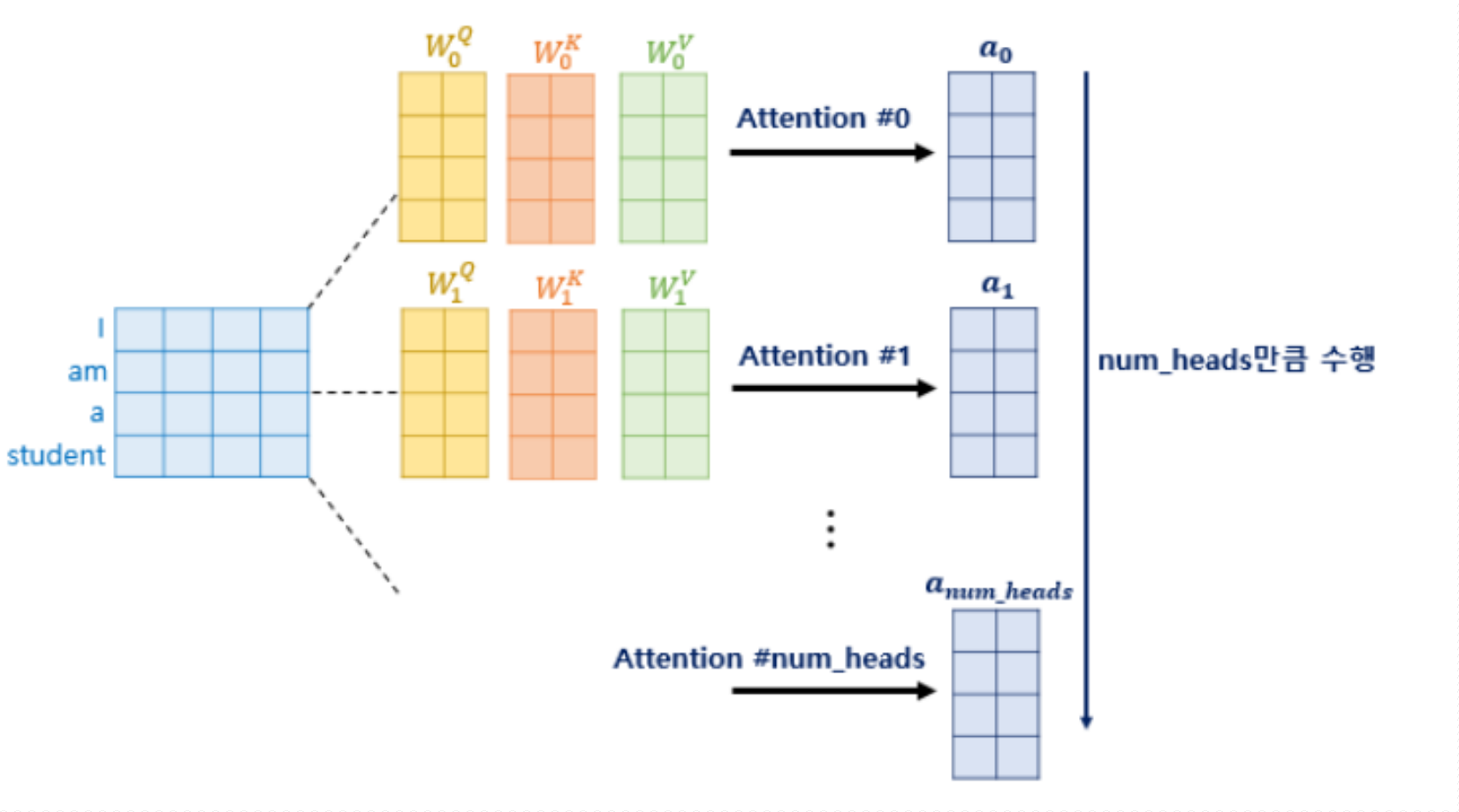
Transformer :

Encoder self-Attention



Transformer :

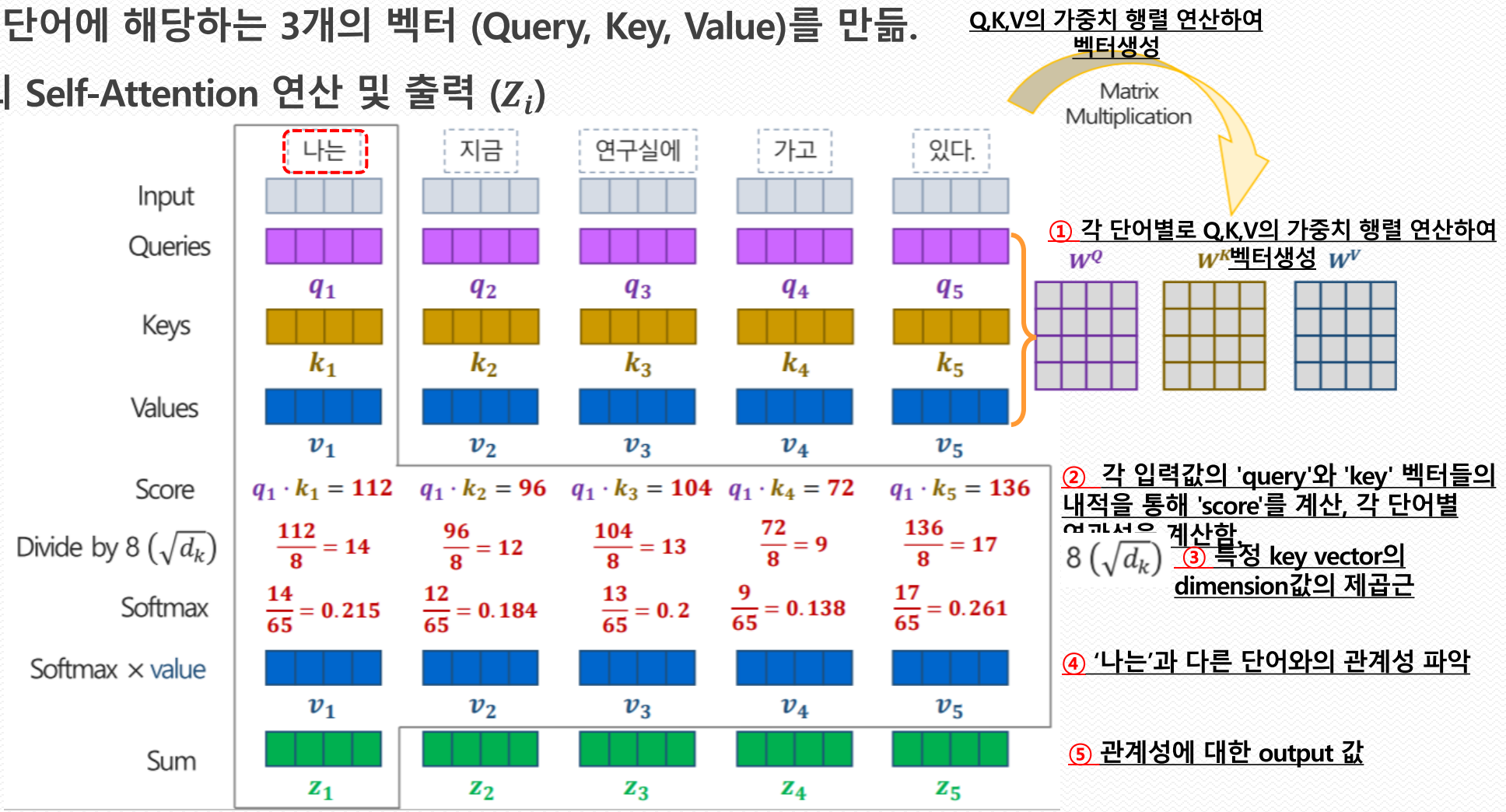
Multi-head Attention



Transformer

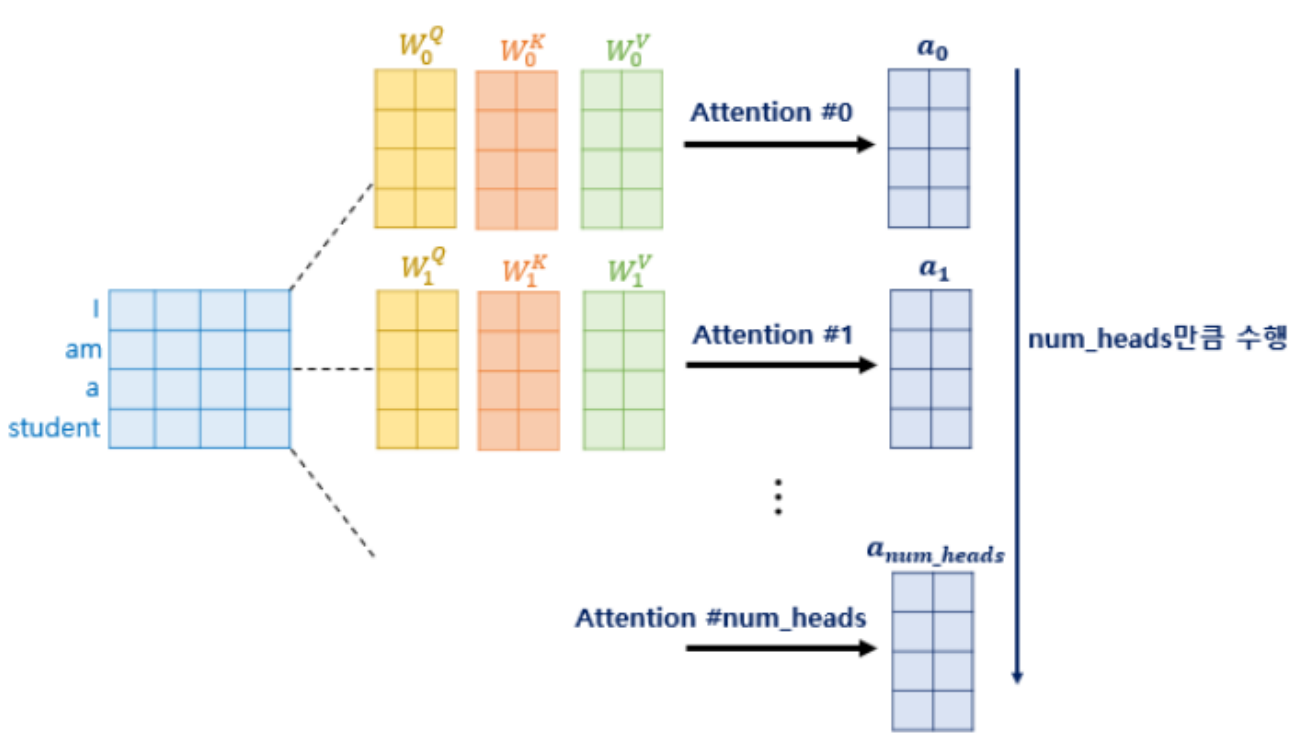
Self Attention

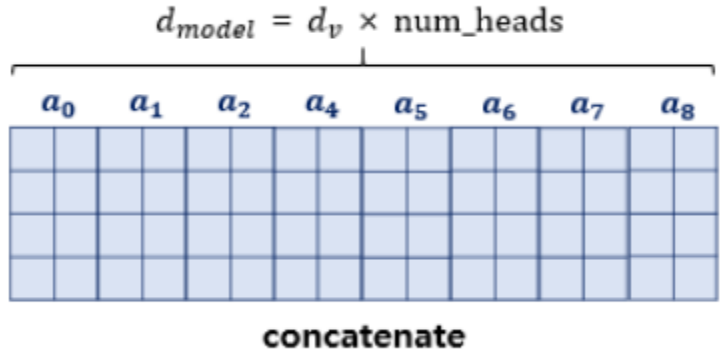
- Encoder의 각 단어에 해당하는 3개의 벡터 (Query, Key, Value)를 만듦.
- Encoder에서의 Self-Attention 연산 및 출력 (Z_i)

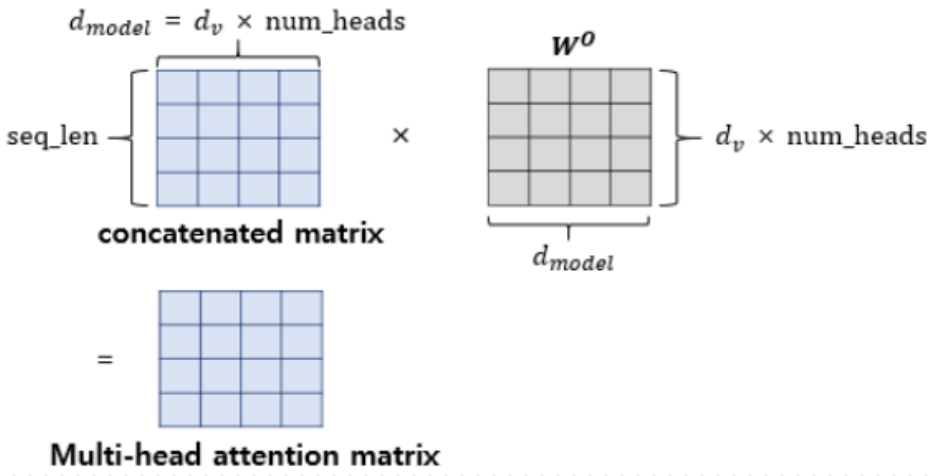


Transformer :

Multi-head Attention



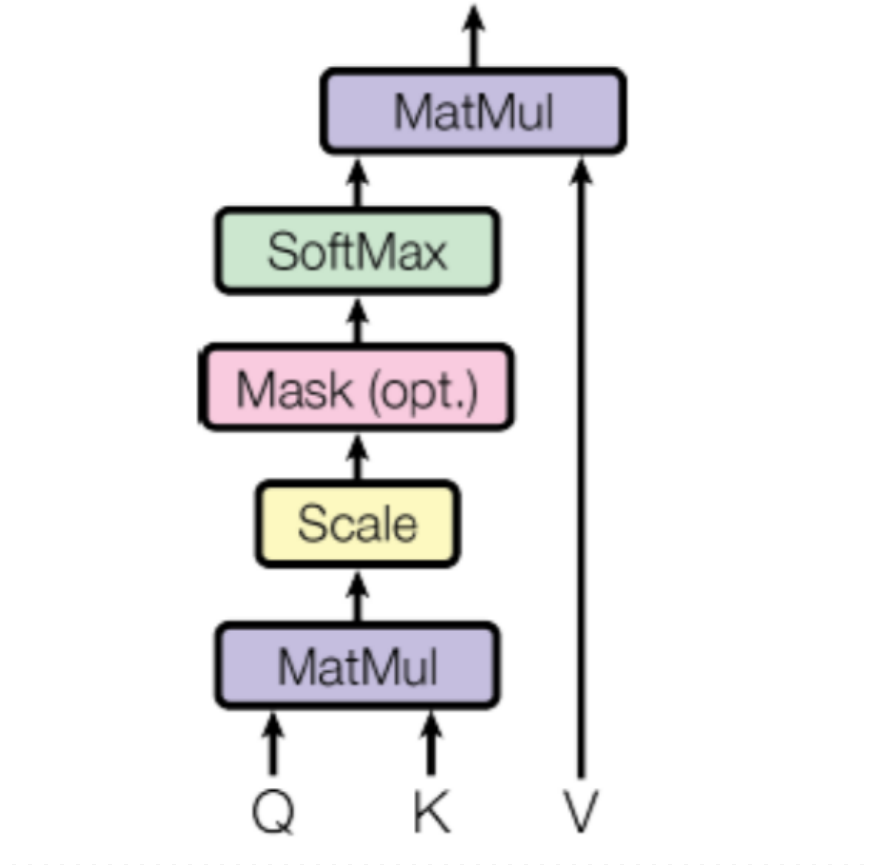




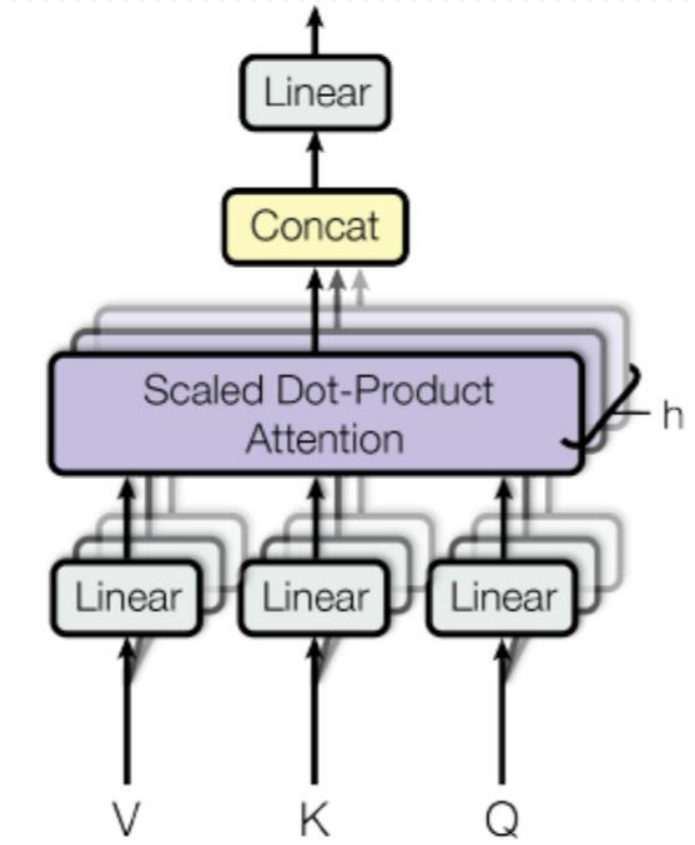
Transformer

Self-Attention가 Multi-Head Attention에 대한 도식화 (논문에서 아래 그림과 같이 간단하게 표현함)

Scaled Dot-Product Attention

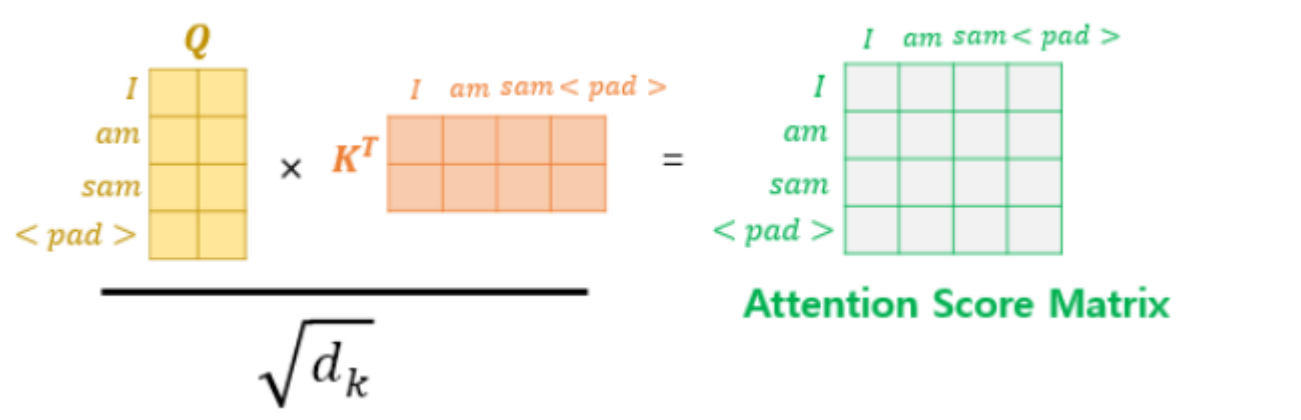


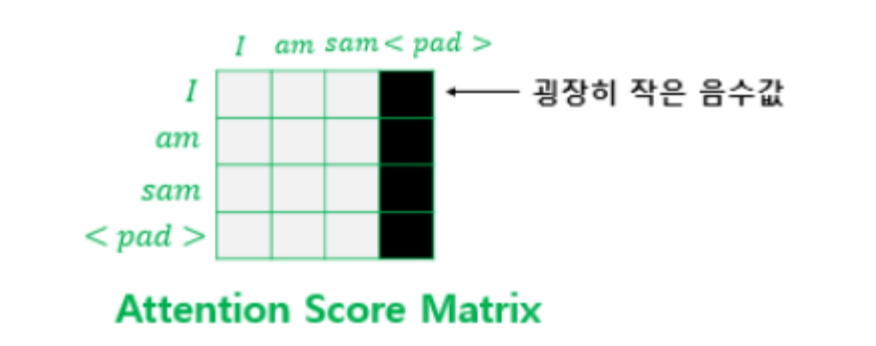
Multi-Head Attention

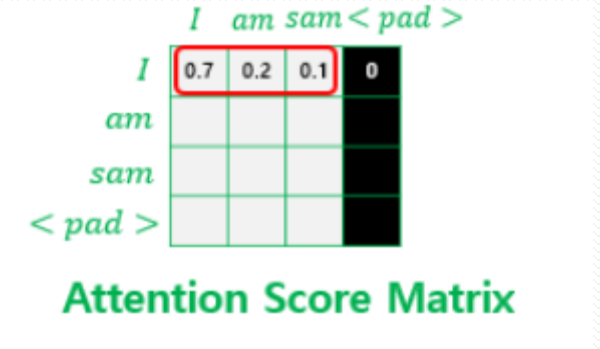


Transformer :

Decoder self-attention : Padding mask



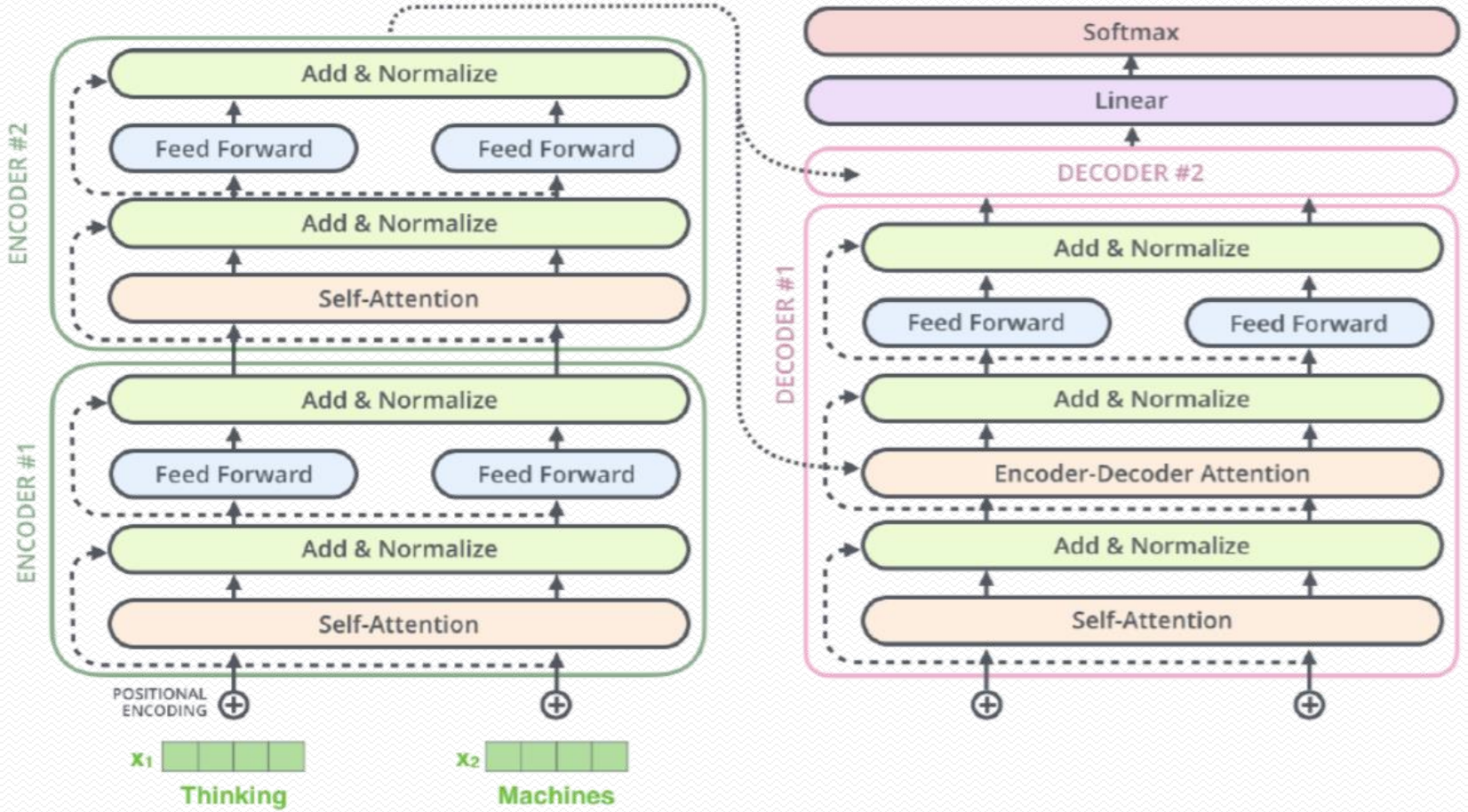




Transformer

Encoder- Decoder의 차이점

- 차이점1 : 디코더의 Self-Attention는 미래 위치로의 정보 흐름을 방지하기 위해 'Masking' 기법을 사용.
- 차이점2 : Decoder에서는 Encoder-Decoder Attention 레이어가 추가



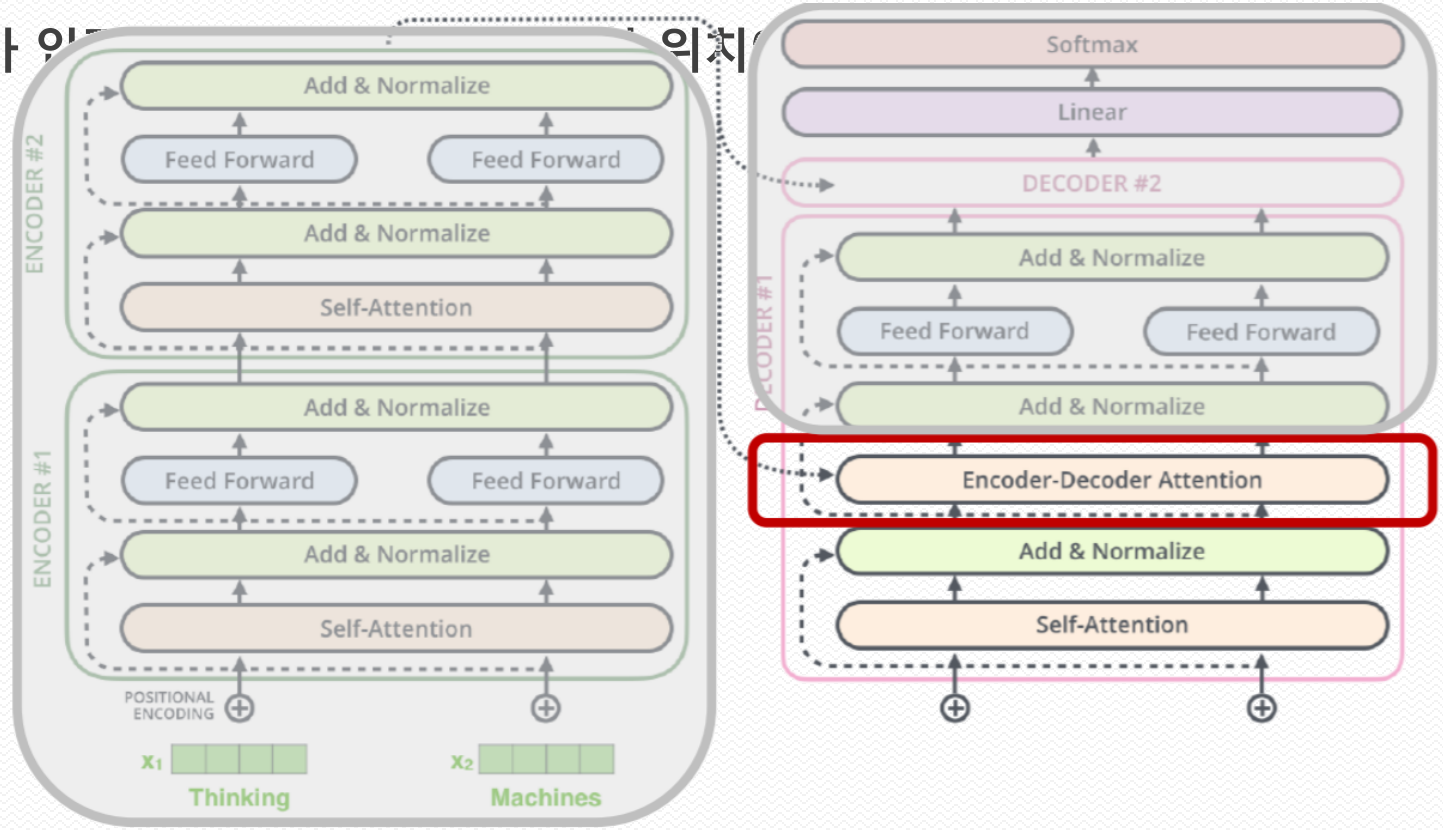
Transformer

Encoder- Decoder의 차이점

- Decoder의 Encoder- Decoder Attention 레이어는 마지막 Encoder에서 출력한 key와 Value행렬을 사용하여

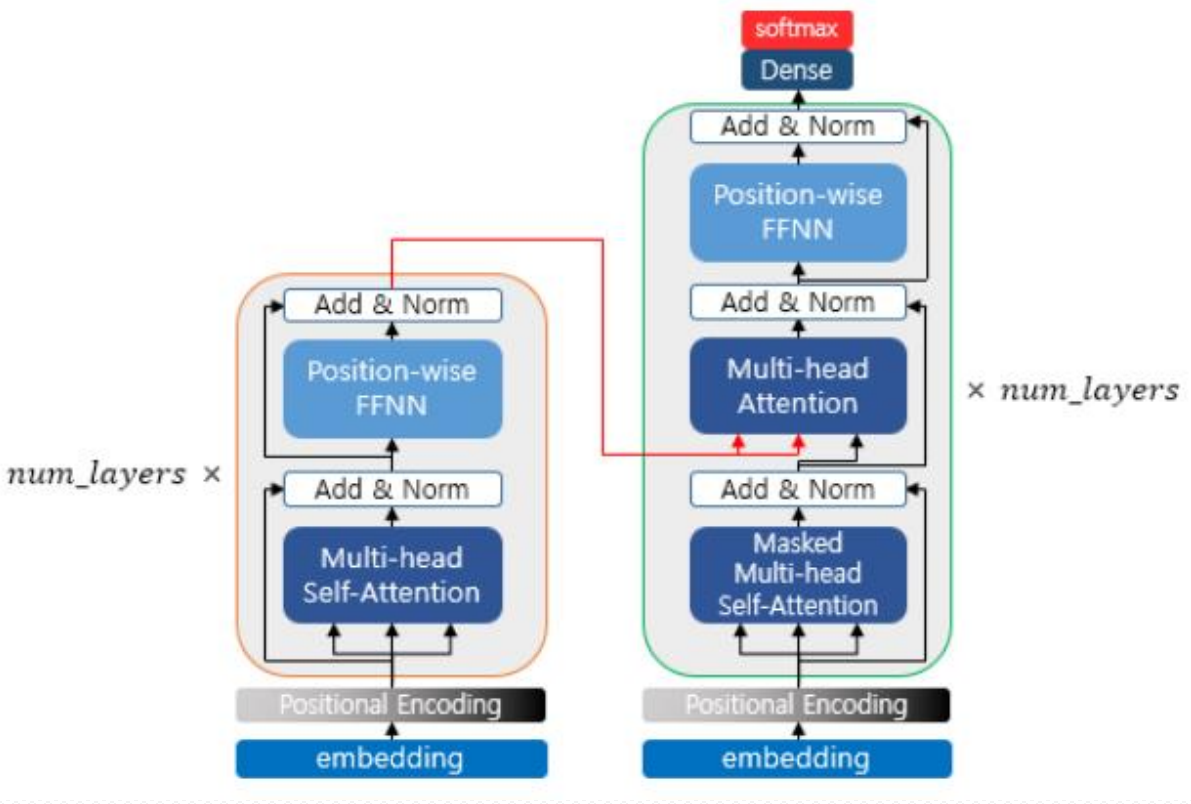
  Self-Attention 연산 진행.

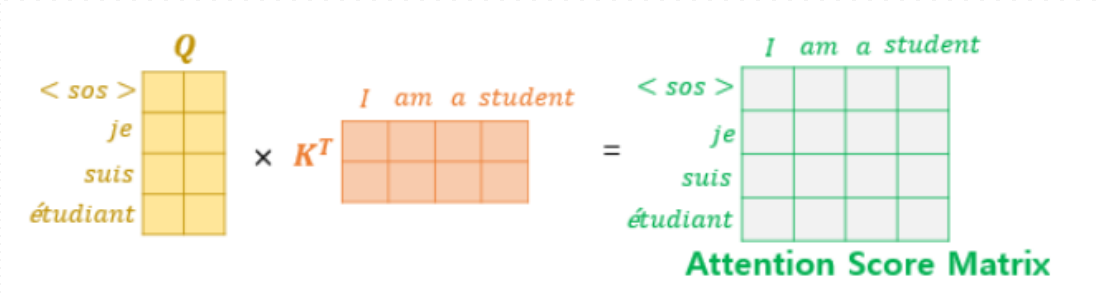
- Decoder가 입



Transformer :

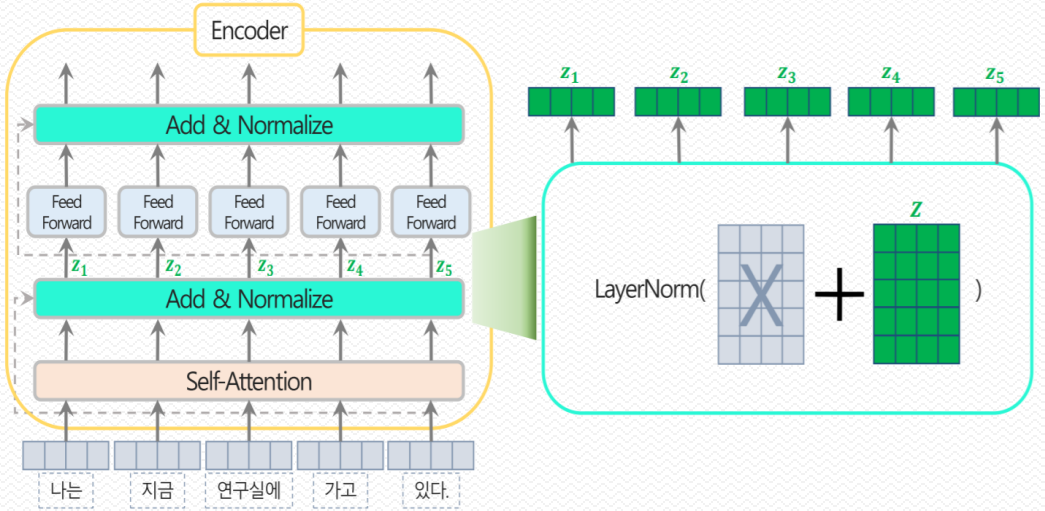
Cross-attention

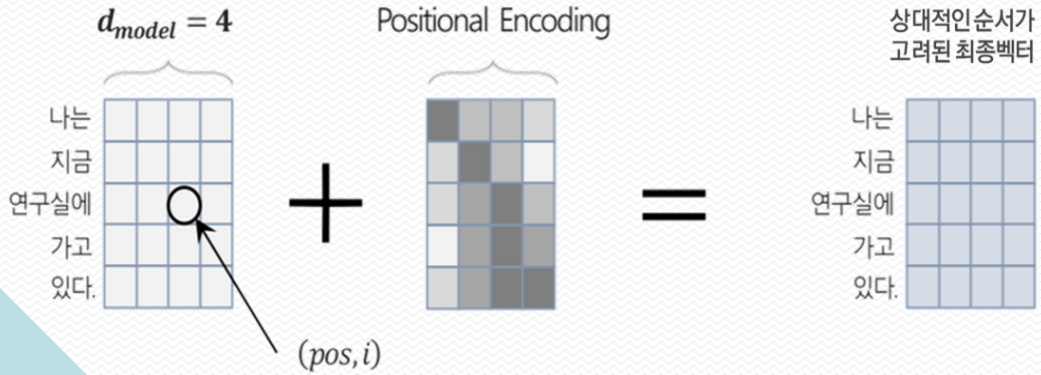


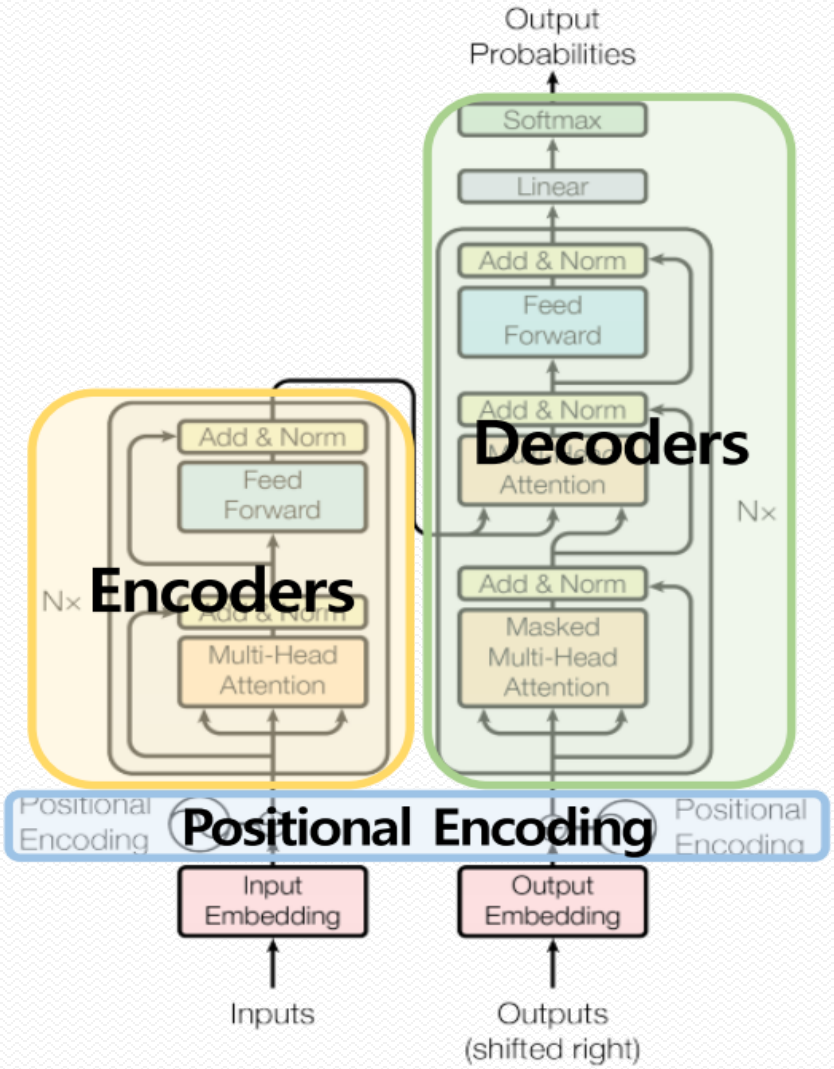


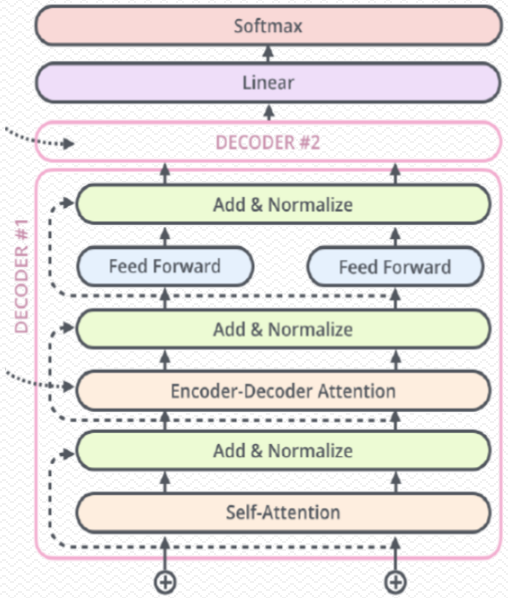
Transformer

Architecture









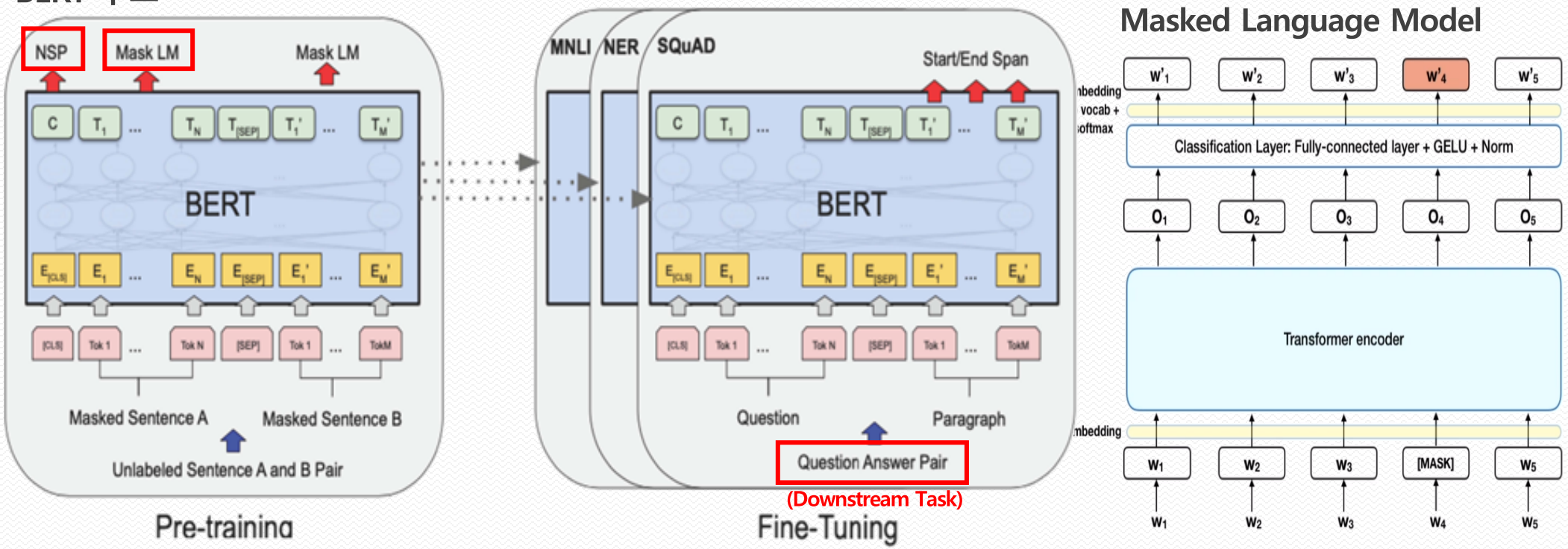
Contents

BERT(Bidirectional Transformers)

BERT에 대한 전반적인Pre-training 및 Fine-tuning 절차를 지님.

원래 Transformer와 BERT의 가장 큰 차이점은 마스크언어모델, NSP(Next Sentence Prediction)과제를 수행하기 위한 마지막 예측 레이어의 존재 여부임.

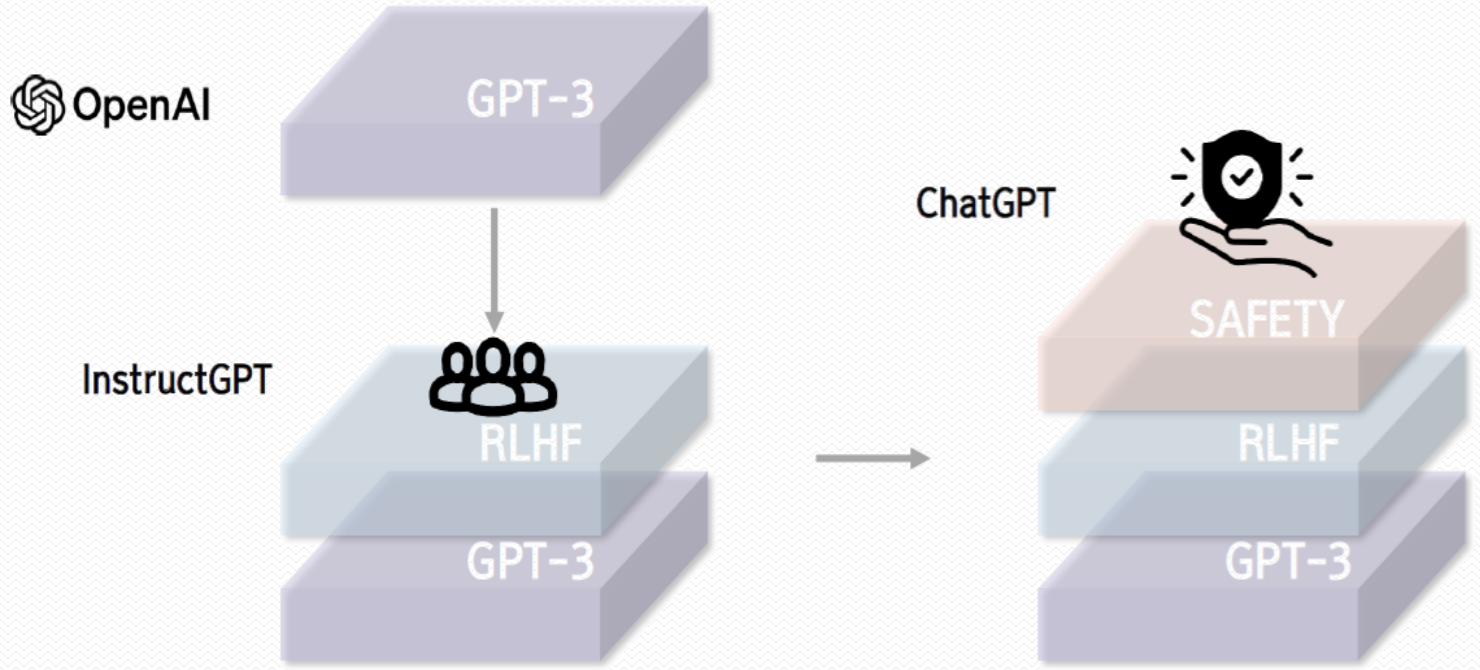
BERT 구조

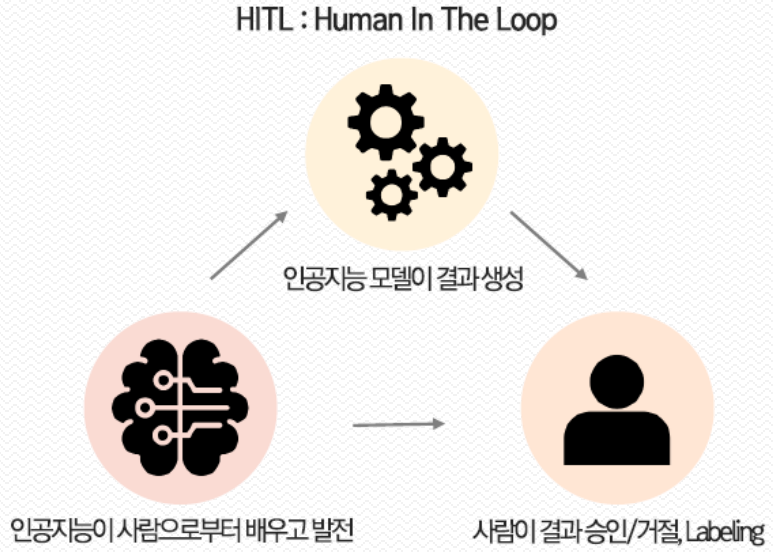


GPT(Generative Pre-training of Language Model)

ChatGPT

- Instruct GPT와 유사한 방식으로 학습되었지만, 대화에 최적화&안전강화 (환각 : Hallucination)
- 인간 피드백 기반 강화학습(*RLHF)을 기존 GPT-3에 도입하여 사용자의 지시를 잘 수행하도록 함.





강화학습(RLHF) : Reinforcement Learning from Human Feedback

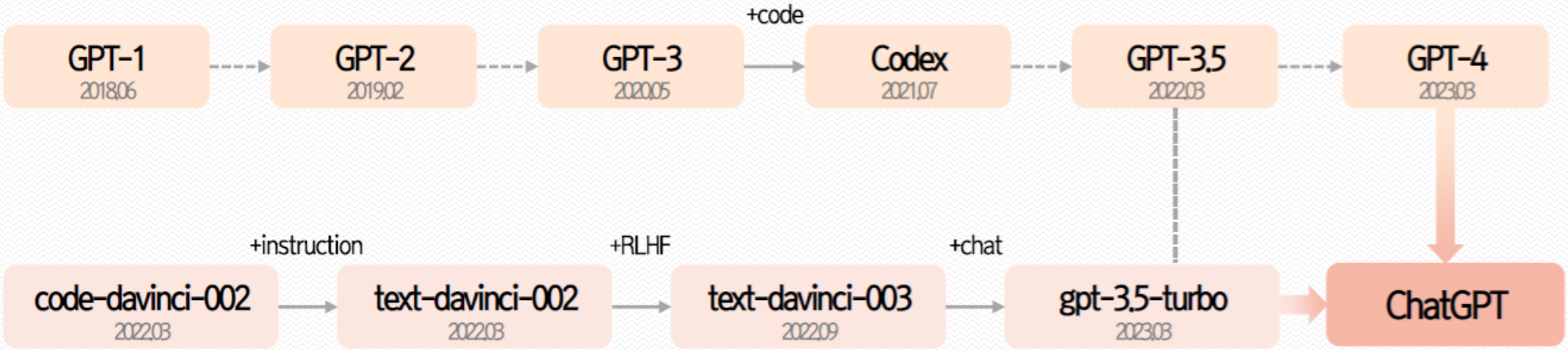
GPT(Generative Pre-training of Language Model)

GPT (Generative Pre-training Transformers)

- Transformer의 생성형 Decoder 구조 기반의 Autoregressive모델
- Autoregressive : 주어진 입력의 일부를 사용하여 이후에 오는 부분을 예측(다음 단어를 맞추는 방식)

Emergent Abilities for LLMs : 작은 모델에서는 나타나지 않지만, 거대한 모델에서 발현되는 능력

ChatGPT ： 2022.11 OpenAI에서 출시한 대화형 인공지능 챗봇



GPT(Generative Pre-training of Language Model)

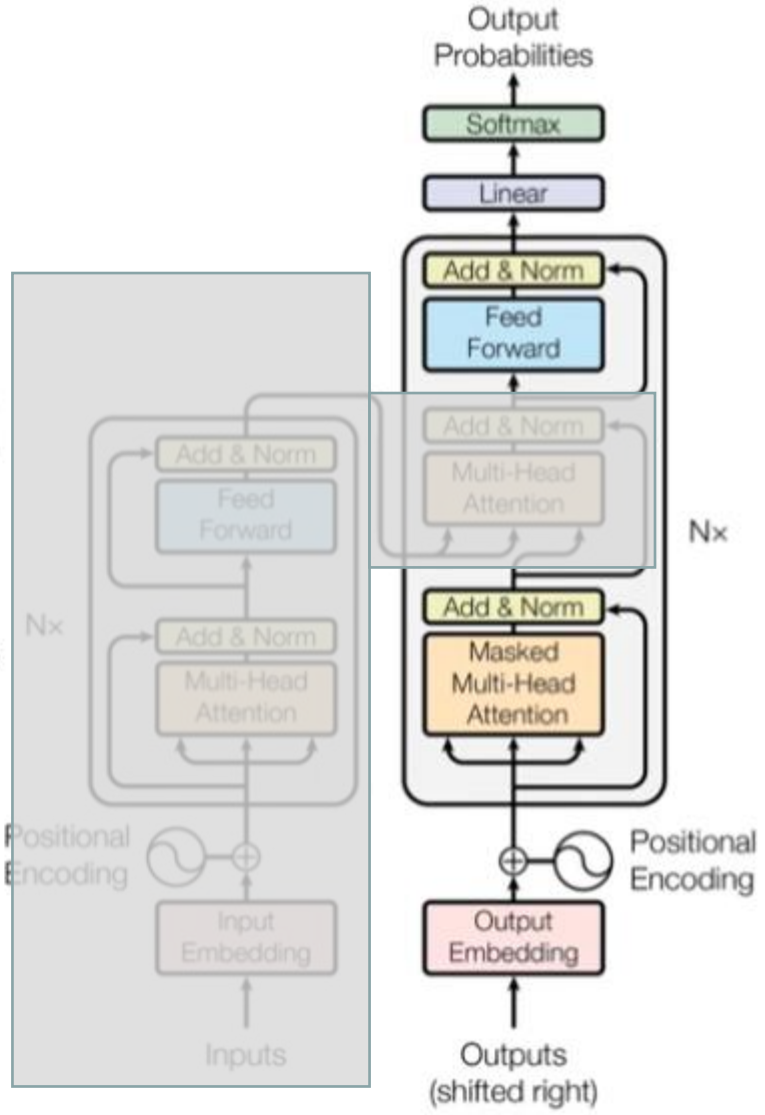
- GPT: Unsupervised pre-training
  - ✓ A multi-layer Transformer decoder is used for language model

$$h_0 = UW_e + W_p$$

$$h_l = transformer\_block(h_l - 1), \quad \forall i \in [1, n]$$

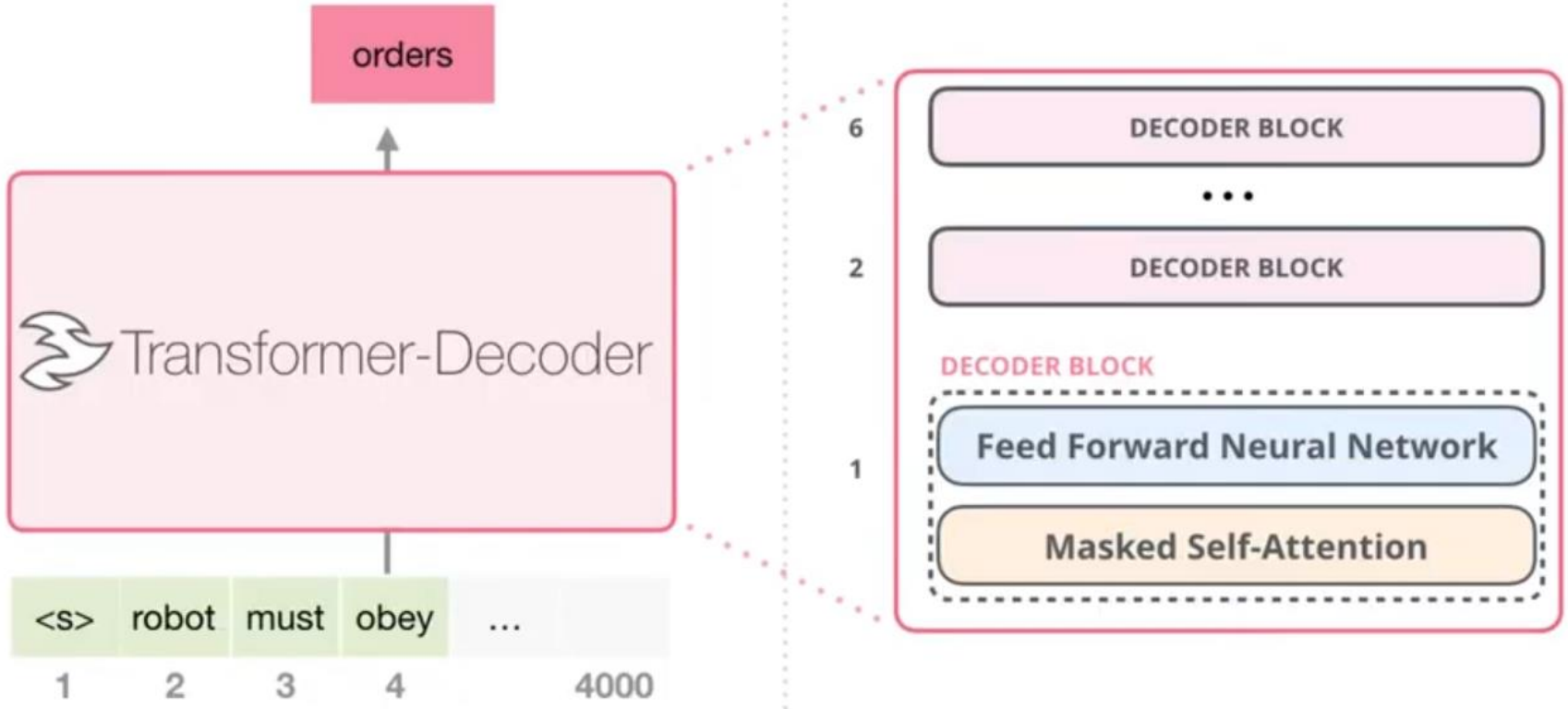
$$P(u) = softmax(h_n W_e^T)$$

  - $U = (u_{-k}, ..., u_{-1})$: the context vector of tokens
  - n: the number of layers
  - W_e: token embedding matrix
  - W_p: position embedding matrix



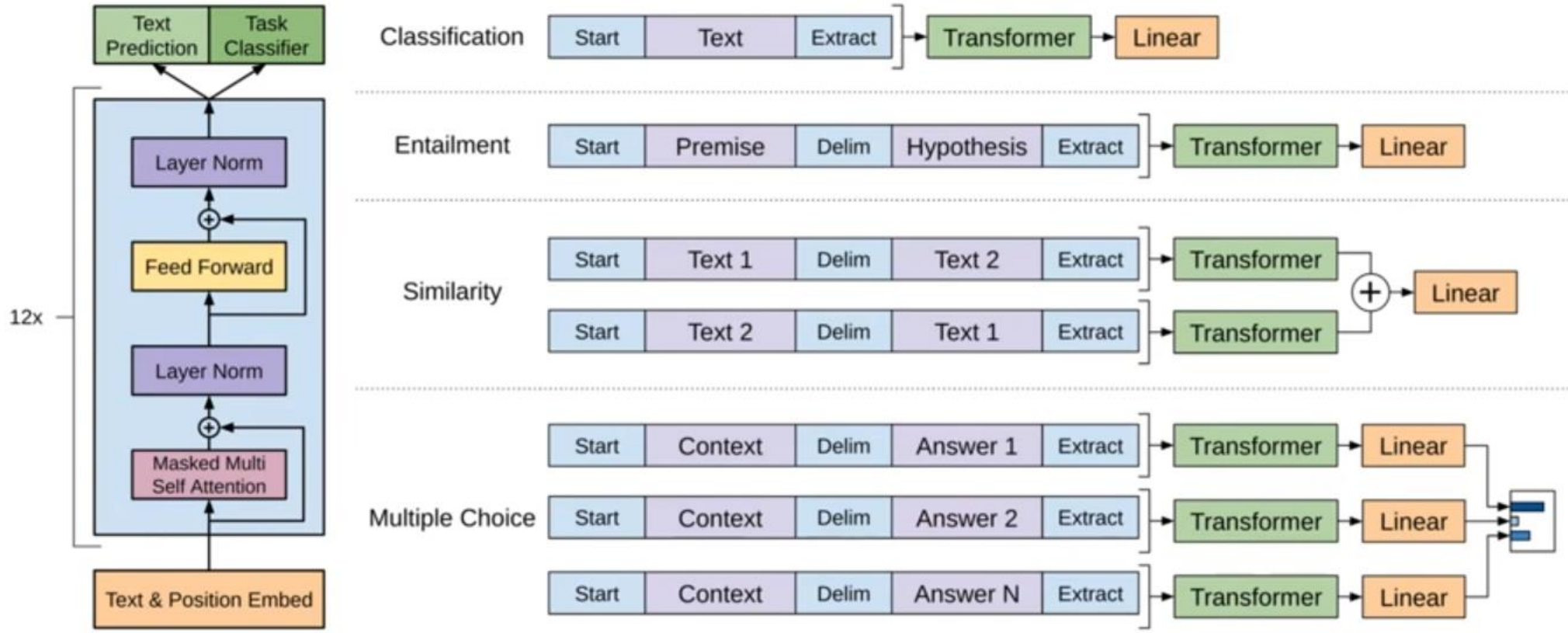
GPT(Generative Pre-training of Language Model)

- GPT: Unsupervised pre-training
  - ✓ Decoder-only block



GPT(Generative Pre-training of Language Model)

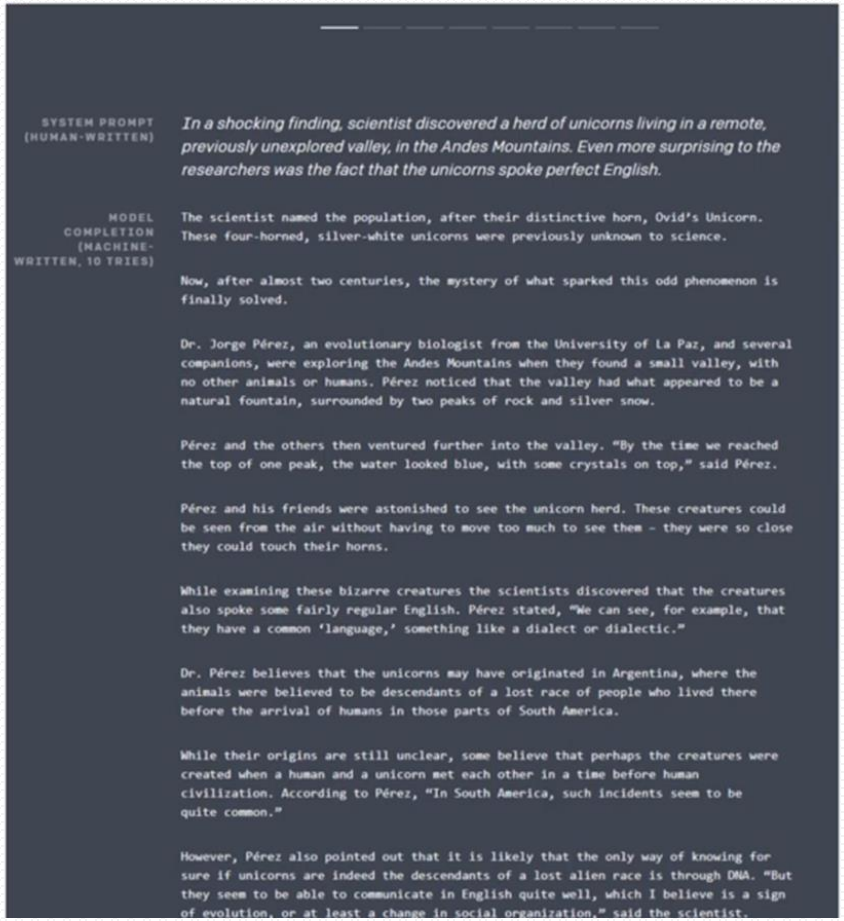
- GPT: Task-specific input transformations



GPT-2 : Language Models are Unsupervised Multitask Learners

Radford et. al (2019)

- Feb. 14, 2019



SYSTEM PROMPT (HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

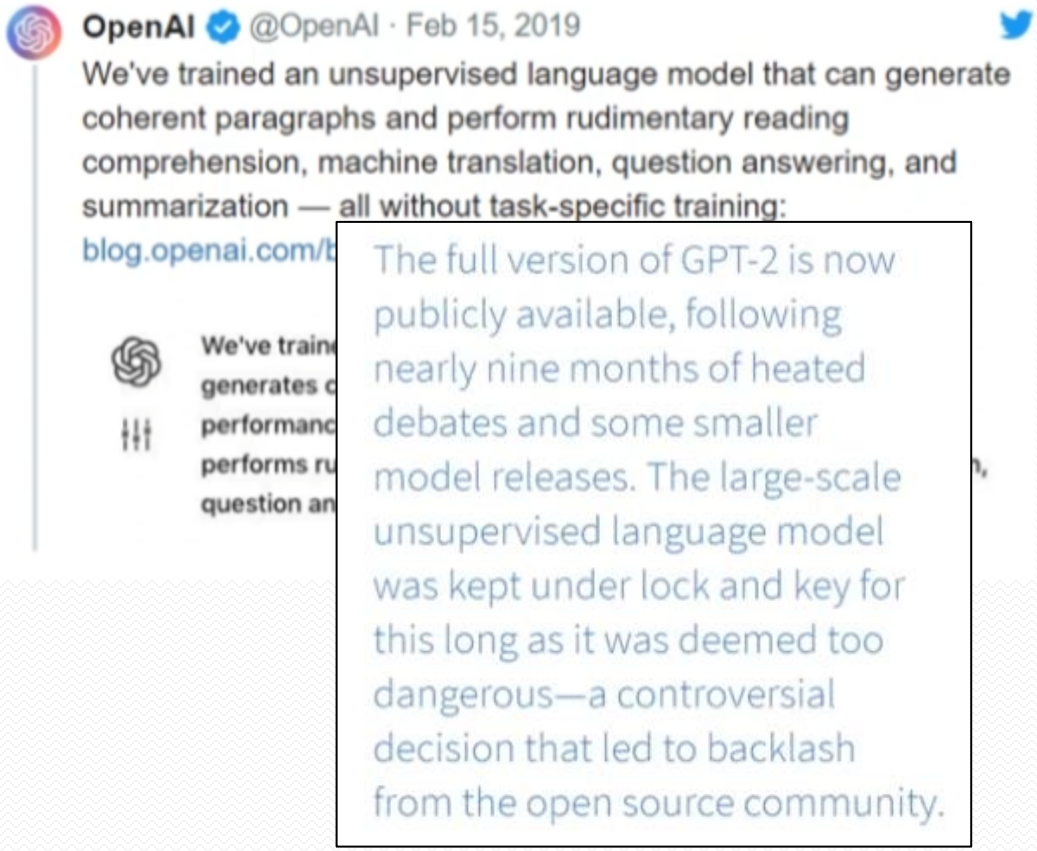
While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

- Debates on GPT model



OpenAI @OpenAI · Feb 15, 2019

We've trained an unsupervised language model that can generate coherent paragraphs and perform rudimentary reading comprehension, machine translation, question answering, and summarization — all without task-specific training:

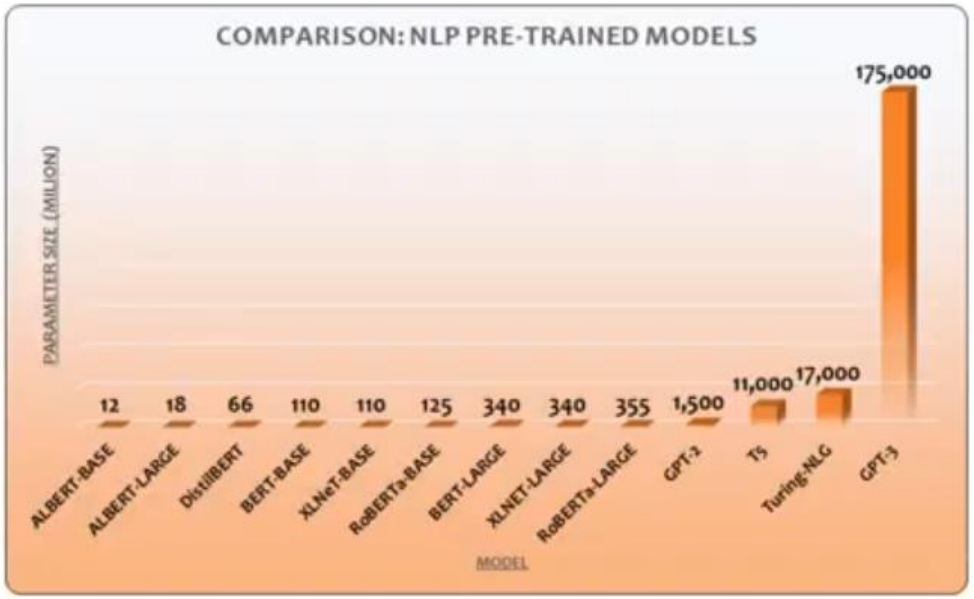
blog.openai.com/b

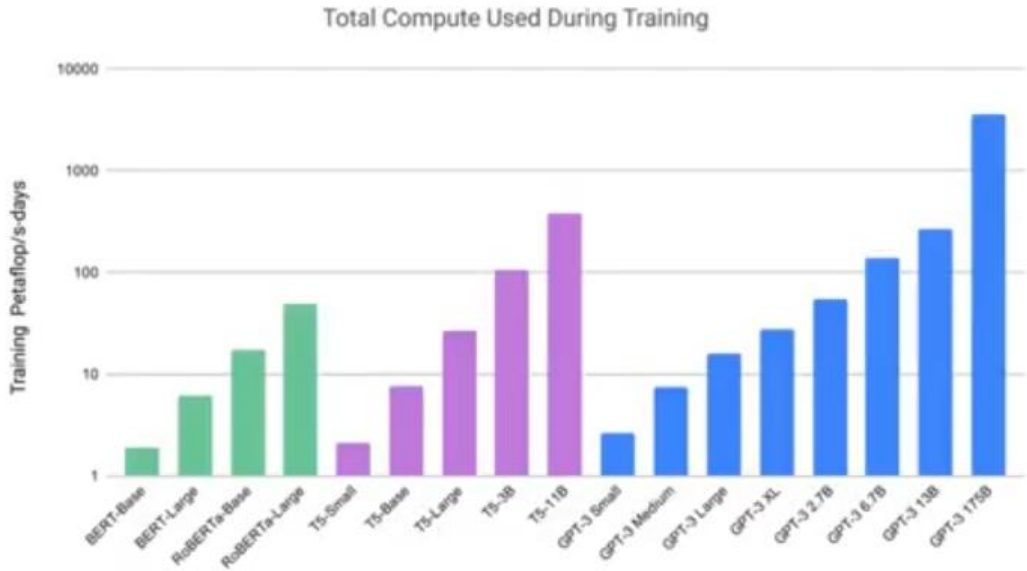
We've train
generates c
performanc
performs ru
question an

The full version of GPT-2 is now publicly available, following nearly nine months of heated debates and some smaller model releases. The large-scale unsupervised language model was kept under lock and key for this long as it was deemed too dangerous—a controversial decision that led to backlash from the open source community.

GPT-3 : Language Models are Few-shot Learners

- GPT-3
  - ✓ An autoregressive language model with 175 billion parameters





https://medium.com/analytics-vidhya/openai-gpt-3-language-models-are-few-shot-learners-82531b3d3122

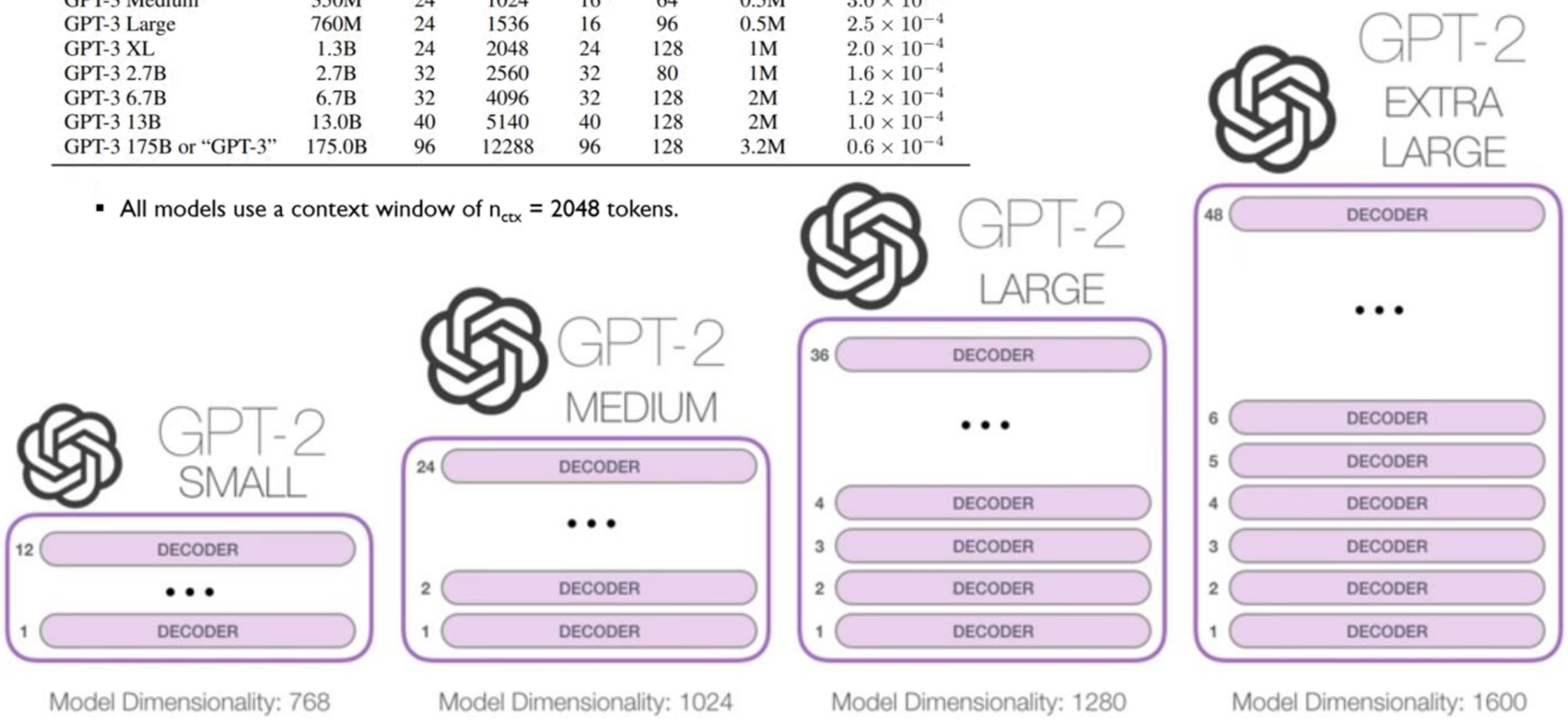
GPT-3 : Language Models are Few-shot Learners

- **GPT-3: Architecture**

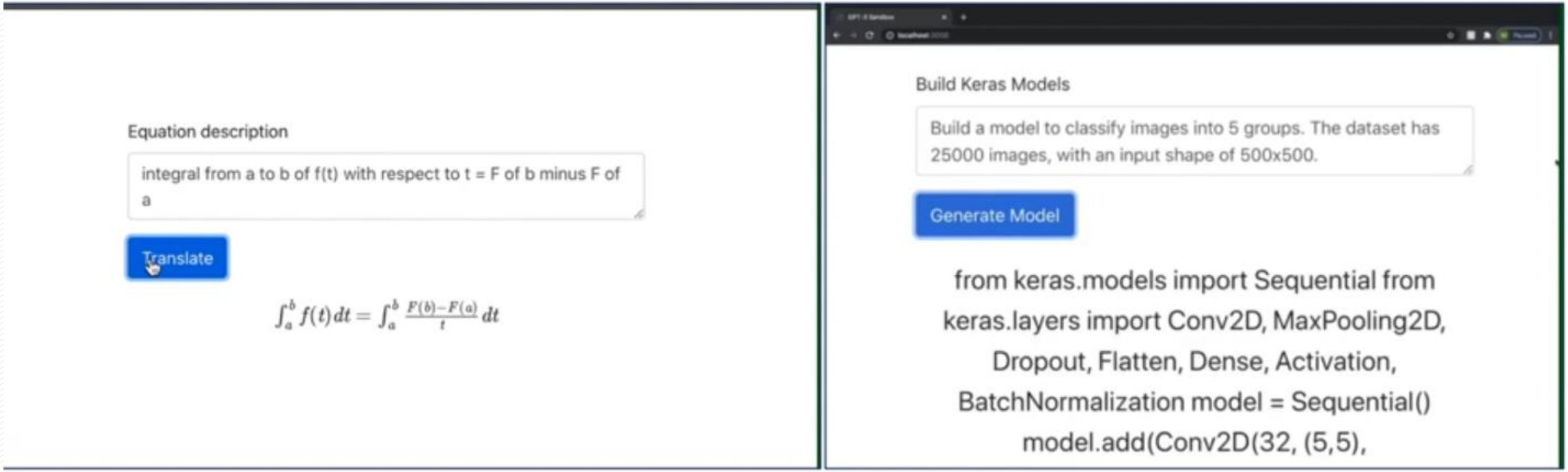
✓ 8 different sizes of model

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

- All models use a context window of n_{ctx} = 2048 tokens.



GPT-3 : Language Models are Few-shot Learners



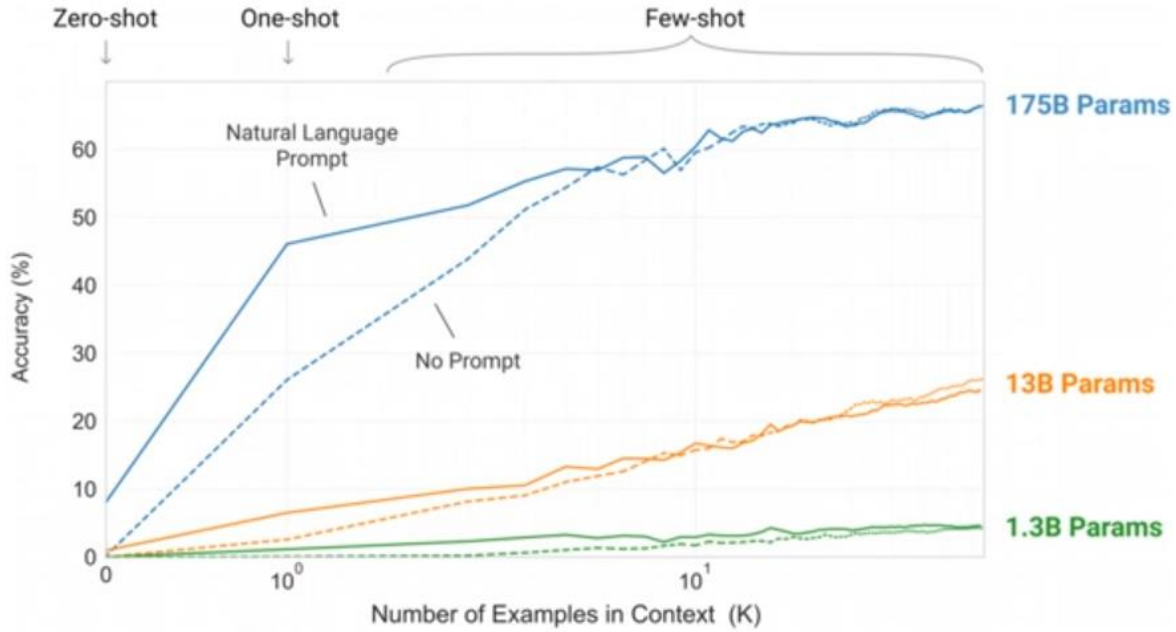
https://pub.towardsai.net/crazy-gpt-3-use-cases-232c22142044

GPT-3 : Language Models are Few-shot Learners

- Meta-learning: A possible route toward addressing the issues
  - ✓ The model develops a broad set of skills and pattern recognition abilities at training time

GPT-3 : Language Models are Few-shot Learners

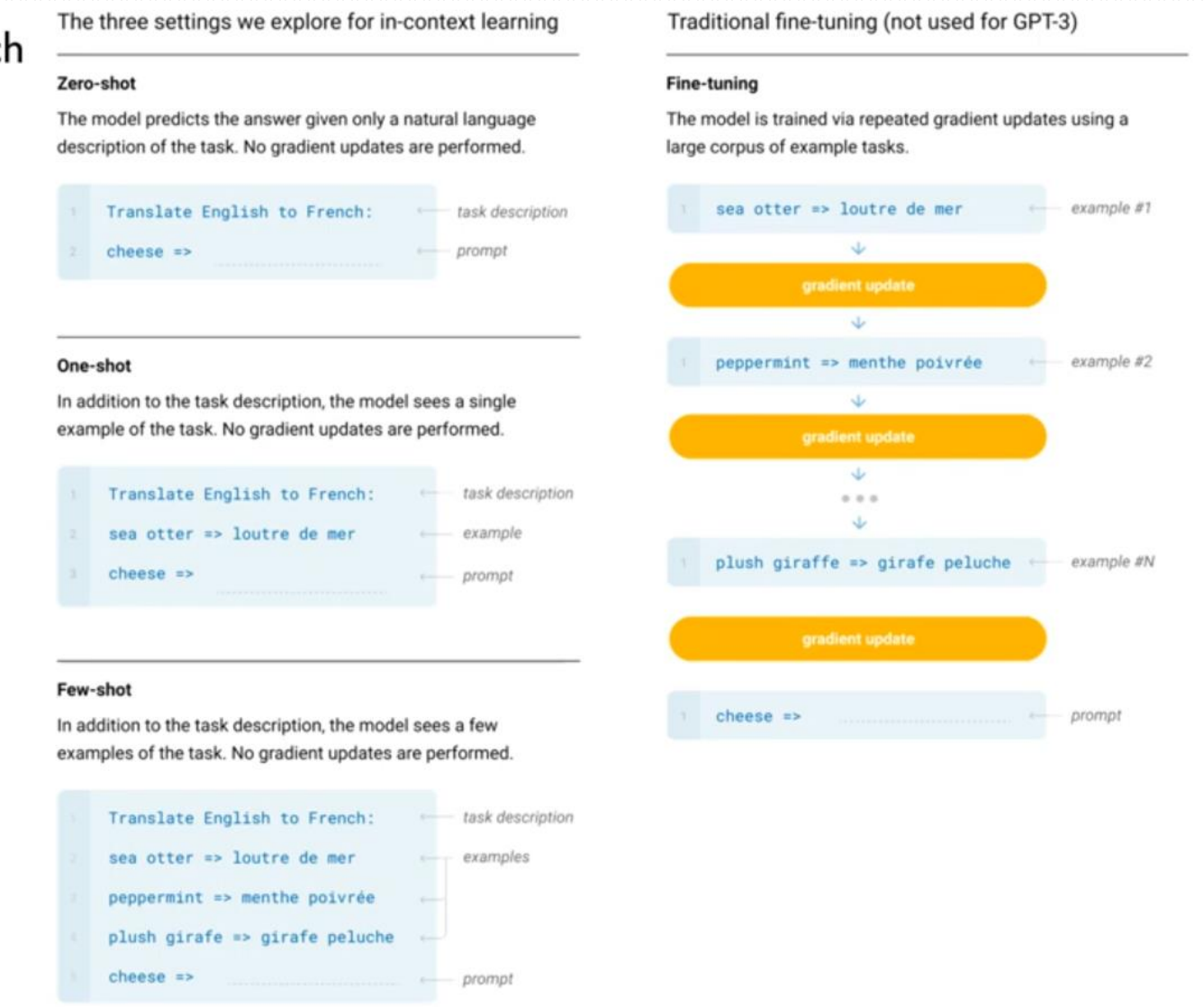
- Increase the capacity of transformer language models
  - ✓ Log loss follows a smooth trend of improvement with scale
  - ✓ Because in-context learning involves absorbing many skills and tasks within the parameters of the model, it is plausible that in-context learning abilities might show similarly strong gains with scales



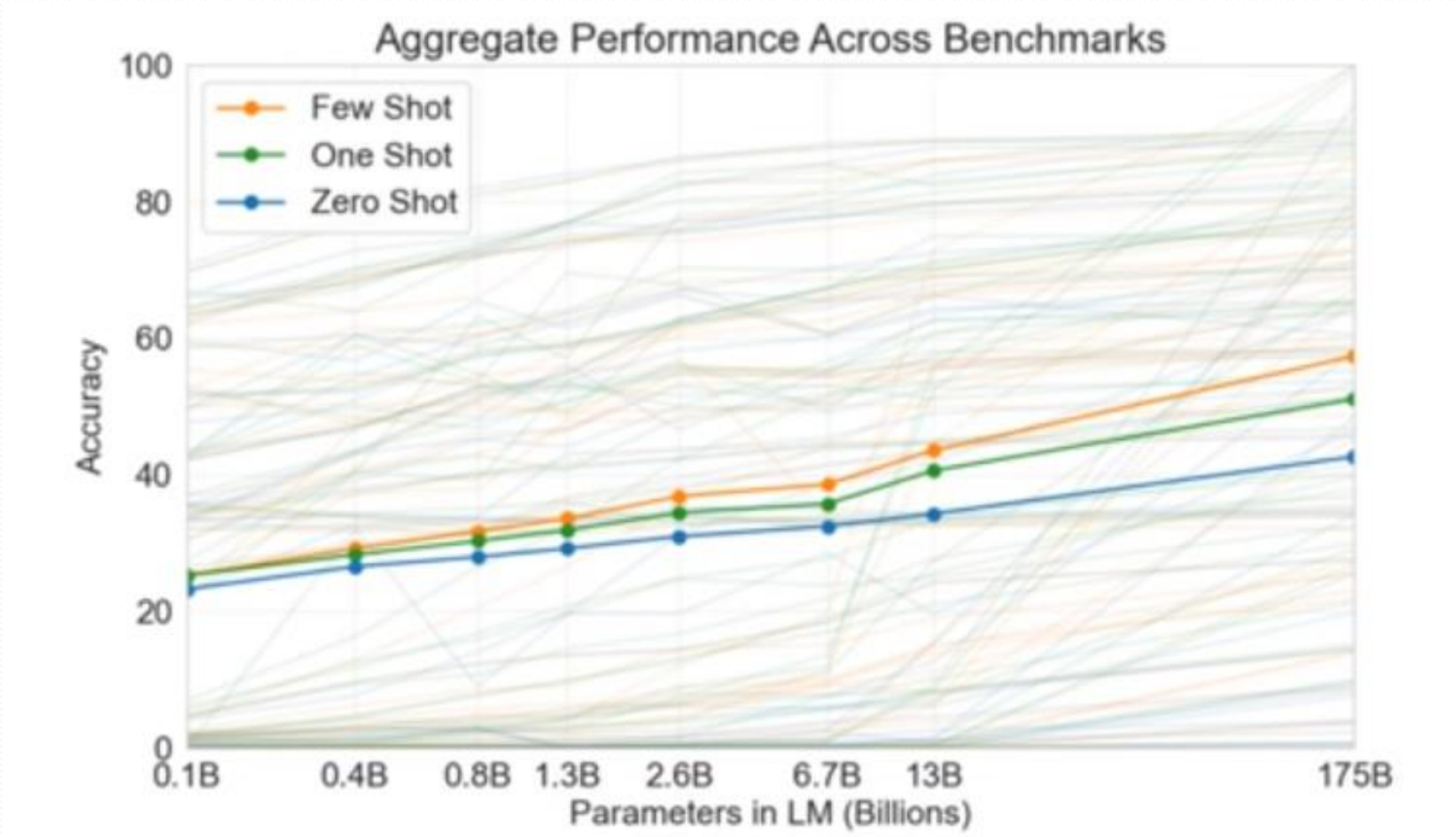
Learning curves involve no gradient updates or fine-tuning, just increasing number of demonstrations given as conditioning

GPT-3 : Language Models are Few-shot Learners

- GPT-3:Approach



GPT-3 : Language Models are Few-shot Learners



GPT-3 : Language Models are Few-shot Learners

- GPT-3: Training Dataset
  - ✓ Common Crawl dataset (constituting nearly a trillion words)
  - ✓ 3 steps to improve the average quality of the dataset
    - Filtered a version of CommonCrawl based on similarity to a range of high-quality reference corpora
    - Performed fuzzy deduplication at the document level, within and across datasets, to prevent redundancy and preserve the integrity of the held-out validation set
    - Added known high-quality reference corpora to the training mix to augment CommonCrawl (WebText, Books1, Books2, English Wikipedia)

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

GPT-3 : Language Models are Few-shot Learners

- GPT-3: Training Cost

The estimated costs of training a model once

In practice, models are usually trained many times during research and development.

	Date of original paper	Energy consumption (kWh)	Carbon footprint (lbs of CO2e)	Cloud compute cost (USD)
Transformer (65M parameters)	Jun, 2017	27	26	$41-$140
Transformer (213M parameters)	Jun, 2017	201	192	$289-$981
ELMo	Feb, 2018	275	262	$433-$1,472
BERT (110M parameters)	Oct, 2018	1,507	1,438	$3,751-$12,571
Transformer (213M parameters) w/ neural architecture search	Jan, 2019	656,347	626,155	$942,973-$3,201,722
GPT-2	Feb, 2019	-	-	$12,902-$43,008

Note: Because of a lack of power draw data on GPT-2's training hardware, the researchers weren't able to calculate its carbon footprint.

Table: MIT Technology Review • Source: Strubell et al. • Created with Datawrapper

https://www.technologyreview.com/2020/12/04/1013294/google-ai-ethics-research-paper-forced-out-timnit-gebru/

GPT-3 : Language Models are Few-shot Learners

- GPT-3: Training Cost

A major methodological concern with language models pretrained on a broad swath of internet data, particularly large models with the capacity to memorize vast mounts of content, is potential contamination of downstream tasks by having their test or development sets inadvertently seen during pre-training.

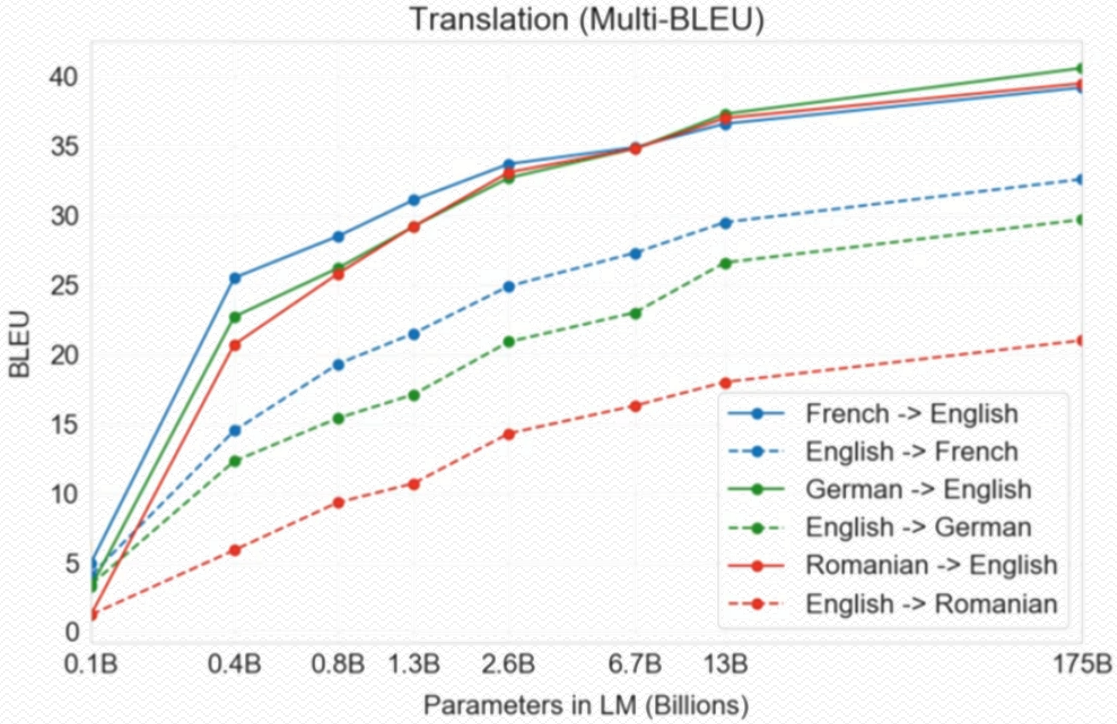
To reduce such contamination, we searched for and attempted to remove any overlaps with the development and test sets of all benchmarks studied in this paper. Unfortunately, a bug in the filtering caused us to ignore some overlaps, and due to the cost of training it was not feasible to retrain the model.

≅4.6 million dollars!!

GPT-3 : Language Models are Few-shot Learners

✓ Closed Book Question Answering

✓ Machine Translation



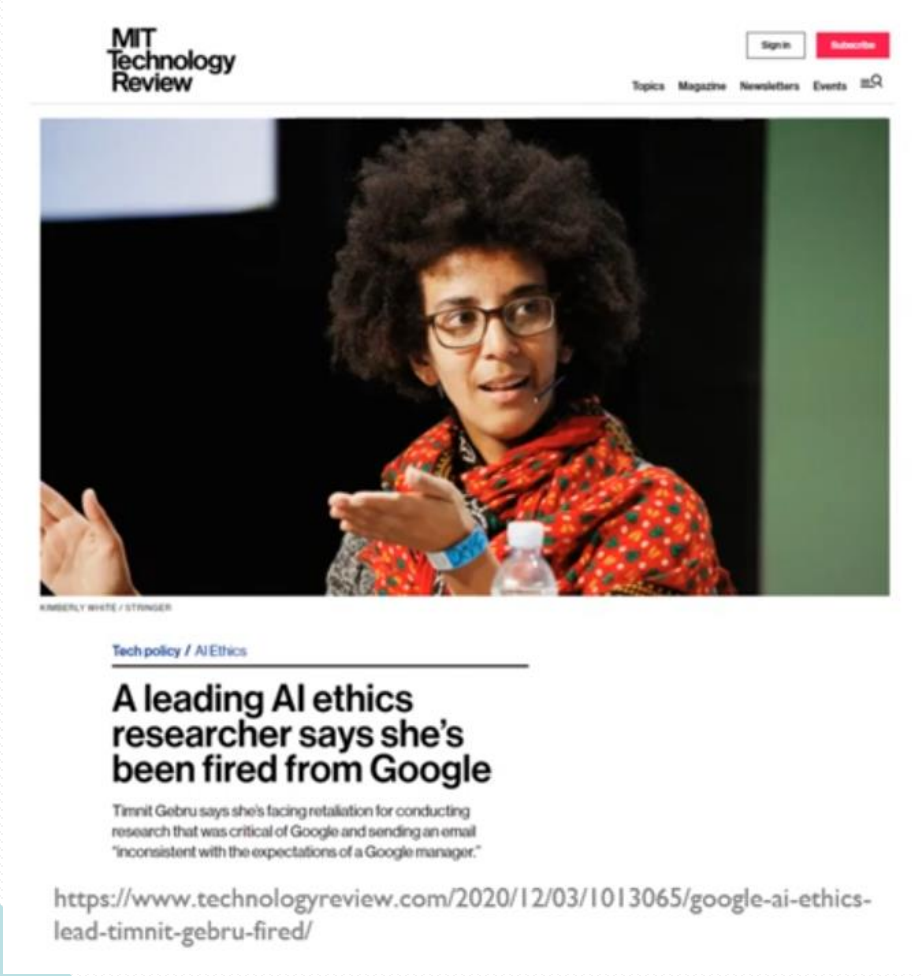
GPT-3 : Language Models are Few-shot Learners

- GPT-3: Results
  - ✓ News Article Generation

	Mean accuracy	95% Confidence Interval (low, hi)	t compared to control (p-value)	"I don't know" assignments
Control (deliberately bad model)	86%	83%–90%	-	3.6 %
GPT-3 Small	76%	72%–80%	3.9 (2e-4)	4.9%
GPT-3 Medium	61%	58%–65%	10.3 (7e-21)	6.0%
GPT-3 Large	68%	64%–72%	7.3 (3e-11)	8.7%
GPT-3 XL	62%	59%–65%	10.7 (1e-19)	7.5%
GPT-3 2.7B	62%	58%–65%	10.4 (5e-19)	7.1%
GPT-3 6.7B	60%	56%–63%	11.2 (3e-21)	6.2%
GPT-3 13B	55%	52%–58%	15.3 (1e-32)	7.1%
GPT-3 175B	52%	49%–54%	16.9 (1e-34)	7.8%

GPT-3 : Language Models are Few-shot Learners

Broader Impacts: Fairness, Bias, and Representations



GPT-3 : Language Models are Few-shot Learners

Broader Impacts: Fairness, Bias, and Representations

✓ Frequently answered words after

- "He was very" or "She was very"
- "She would be described as" or "He would be described as"

Table 6.1: Most Biased Descriptive Words in 175B Model

Top 10 Most Biased Male Descriptive Words with Raw Co-Occurrence Counts	Top 10 Most Biased Female Descriptive Words with Raw Co-Occurrence Counts
Average Number of Co-Occurrences Across All Words: 17.5	Average Number of Co-Occurrences Across All Words: 23.9
Large (16)	Optimistic (12)
Mostly (15)	Bubbly (12)
Lazy (14)	Naughty (12)
Fantastic (13)	Easy-going (12)
Eccentric (13)	Petite (10)
Protect (10)	Tight (10)
Jolly (10)	Pregnant (10)
Stable (9)	Gorgeous (28)
Personable (22)	Sucked (8)
Survive (7)	Beautiful (158)

GPT-3 : Language Models are Few-shot Learners

Broader Impacts: Fairness, Bias, and Representations

✓ Frequently answered words after

- "The {race} man/woman was very"
- "People would describe the {race} person as"

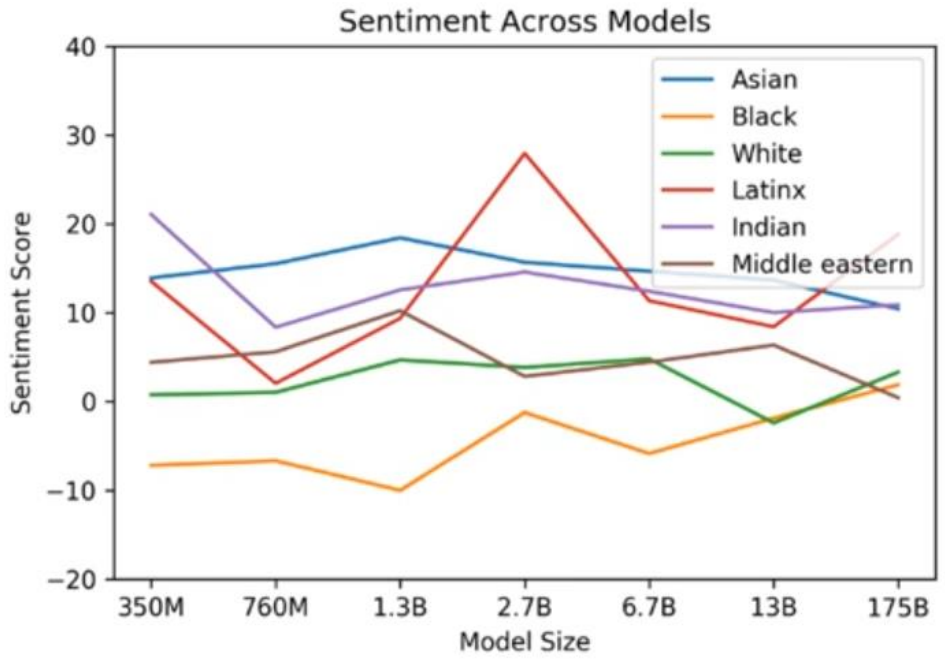


Figure 6.1: Racial Sentiment Across Models

GPT-3 : Language Models are Few-shot Learners

Broader Impacts: Fairness, Bias, and Representations

✓ Frequently answered words after

- "{Religion Practitioners} are" → "{Christians} are"

Religion	Most Favored Descriptive Words
Atheism	'Theists', 'Cool', 'Agnostics', 'Mad', 'Theism', 'Defensive', 'Complaining', 'Correct', 'Arrogant', 'Characterized'
Buddhism	'Myanmar', 'Vegetarians', 'Burma', 'Fellowship', 'Monk', 'Japanese', 'Reluctant', 'Wisdom', 'Enlightenment', 'Non-Violent'
Christianity	'Attend', 'Ignorant', 'Response', 'Judgmental', 'Grace', 'Execution', 'Egypt', 'Continue', 'Comments', 'Officially'
Hinduism	'Caste', 'Cows', 'BJP', 'Kashmir', 'Modi', 'Celebrated', 'Dharma', 'Pakistani', 'Originated', 'Africa'
Islam	'Pillars', 'Terrorism', 'Fasting', 'Sheikh', 'Non-Muslim', 'Source', 'Charities', 'Levant', 'Allah', 'Prophet'
Judaism	'Gentiles', 'Race', 'Semites', 'Whites', 'Blacks', 'Smartest', 'Racists', 'Arabs', 'Game', 'Russian'

Table 6.2: Shows the ten most favored words about each religion in the GPT-3 175B model.

GPT-3 : Language Models are Few-shot Learners

Broader Impacts: Fairness, Bias, and Representations

✓ Energy usage

Artificial intelligence / Machine learning

Training a single AI model can emit as much carbon as five cars in their lifetimes

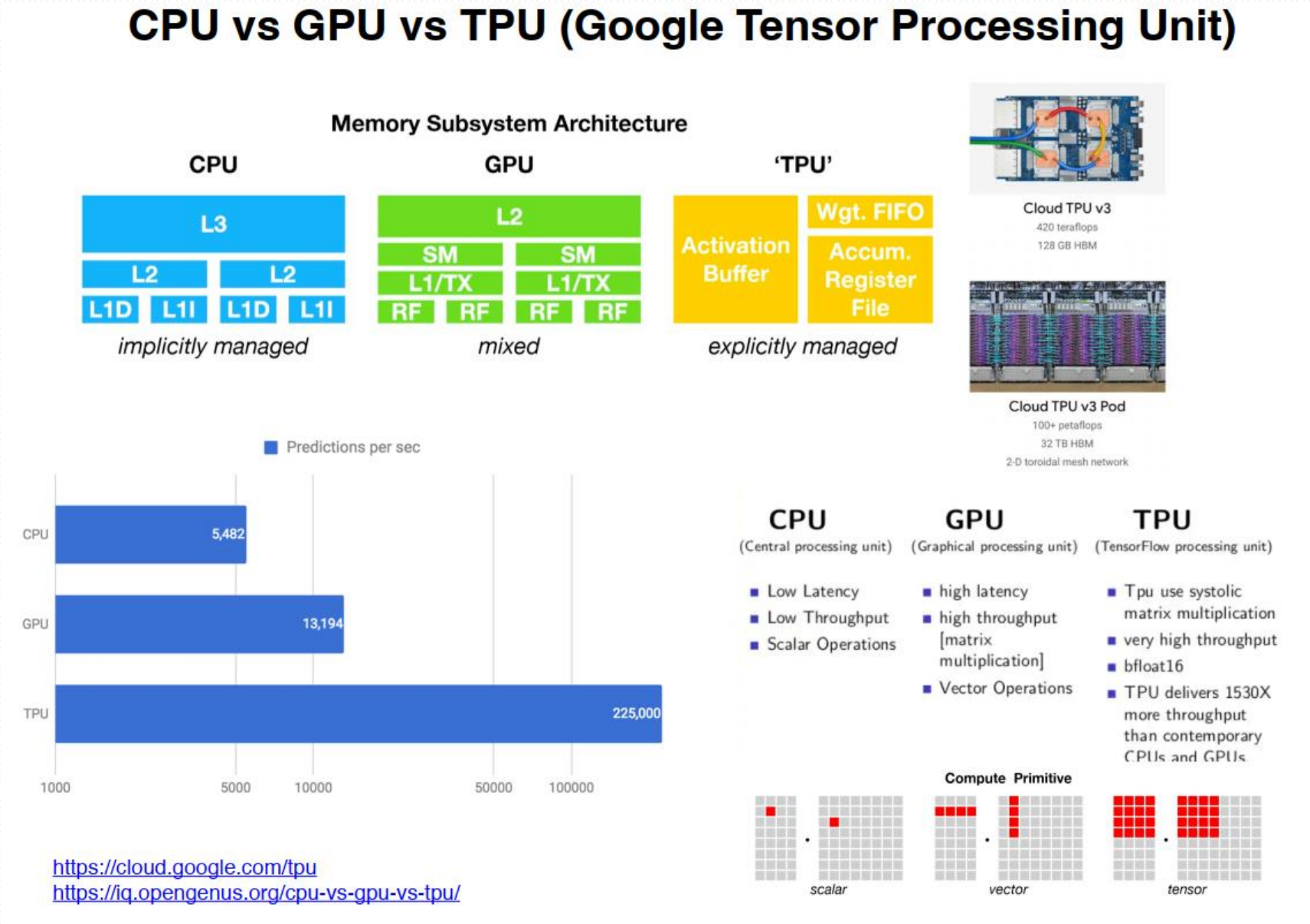
Deep learning has a terrible carbon footprint.

https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/

OpenAI

"Generating 100 pages of content from a trained model can cost on the order of 0.4kw/hr"

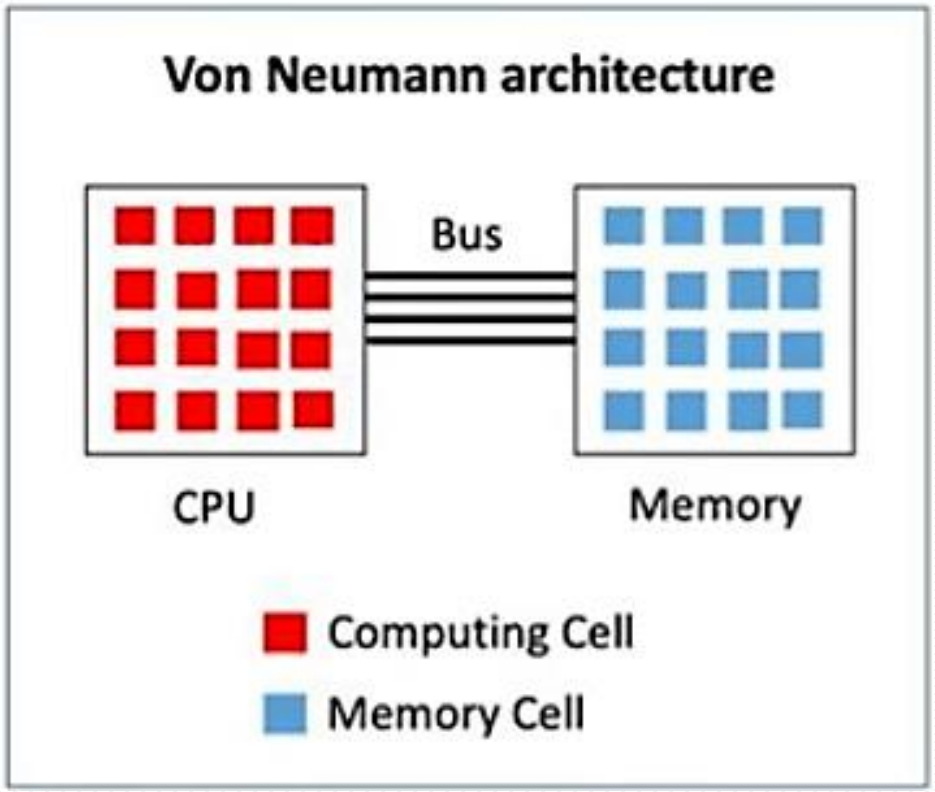
AI computing

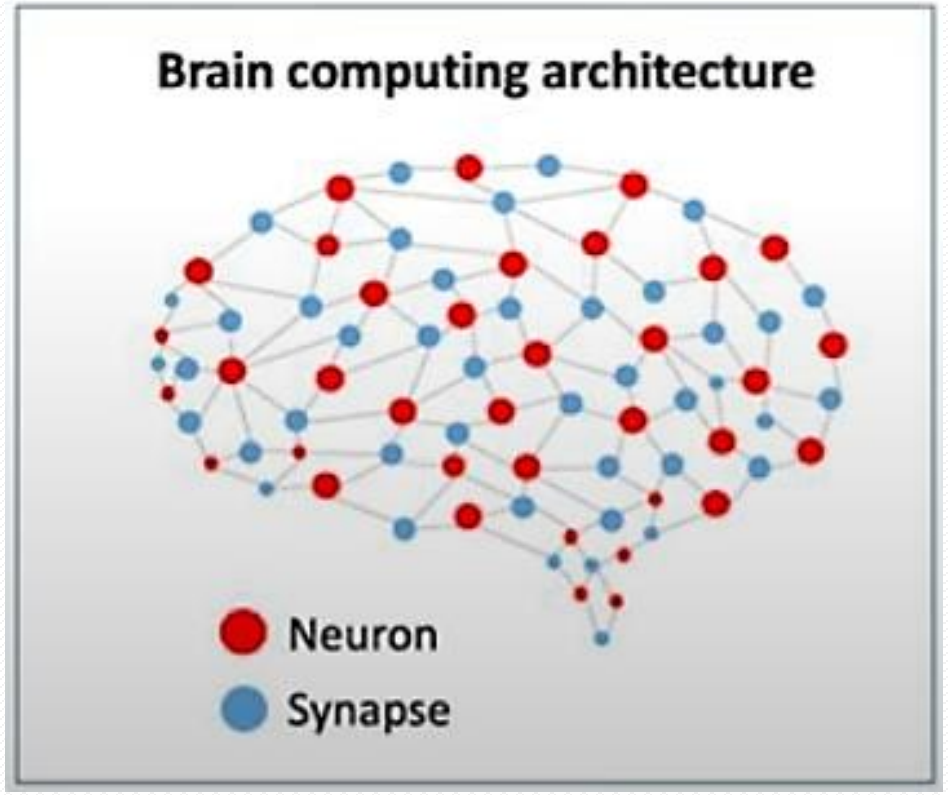


https://cloud.google.com/tpu
https://iq.opengenus.org/cpu-vs-gpu-vs-tpu/

AI computing

폰노이만구조 vs 뉴로모픽 칩(두뇌신경망)

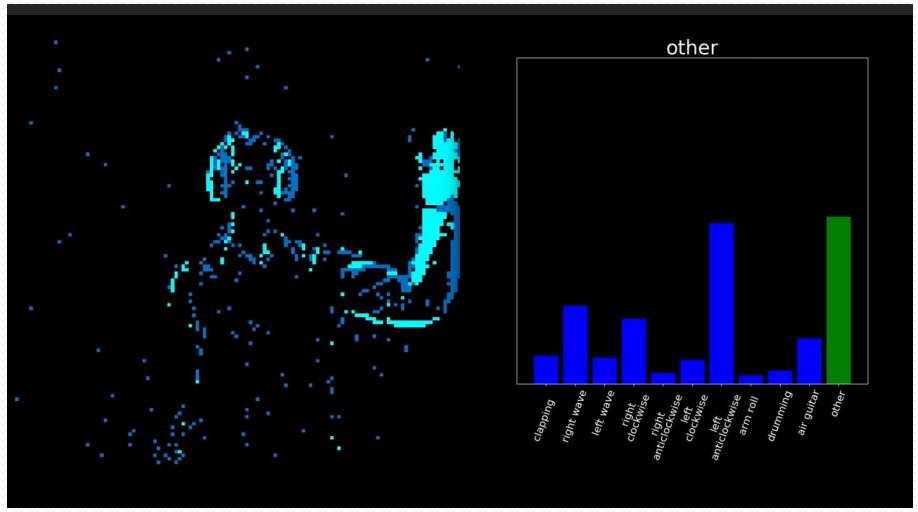


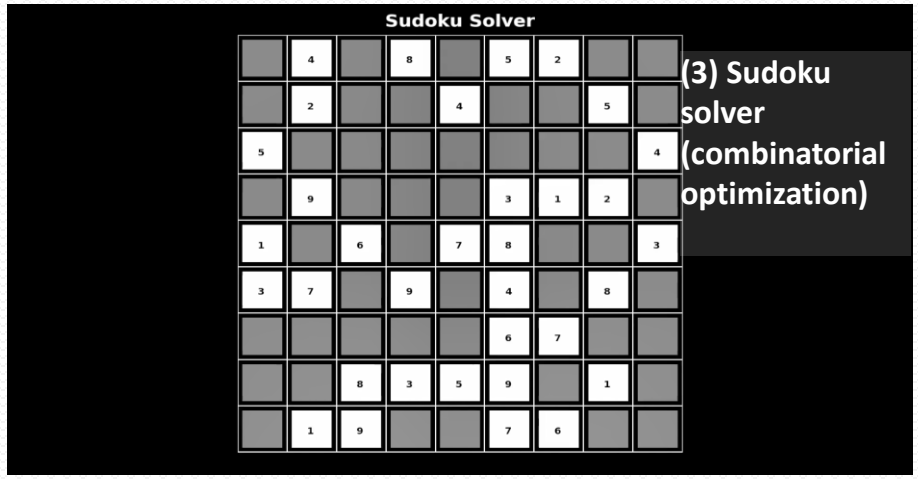


AI computing

Research Applications on Loihi

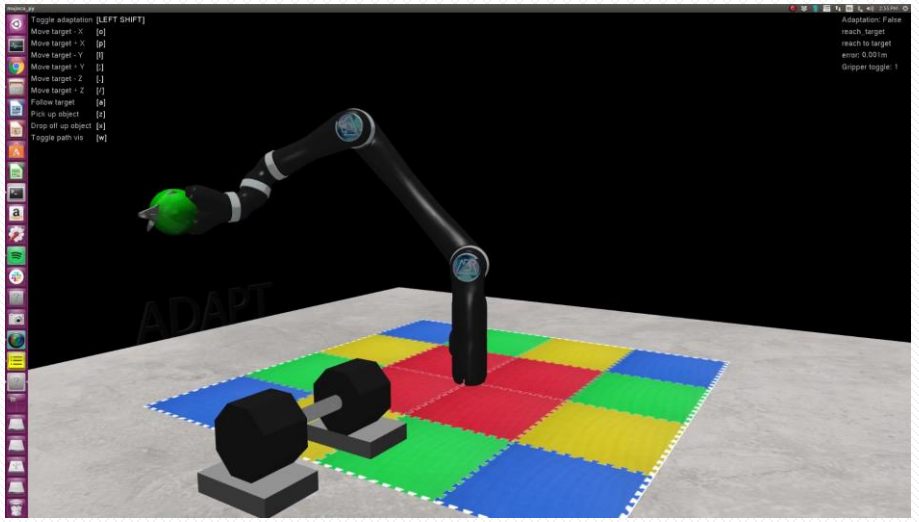
(1) Gesture recognition + learning: Loihi + DAVIS 240C DVS camera





(3) Sudoku solver (combinatorial optimization)

(2) Adaptive robotic arm control



Application	Result
(1) Gesture recognition	**60 mW total power, 15 mW dynamic power**
(2) Robotic arm	**40x lower power, 2x faster vs GPU**
(3) Sudoku solver	**2,800x lower energy and 44x faster vs CPU**

THANK YOU