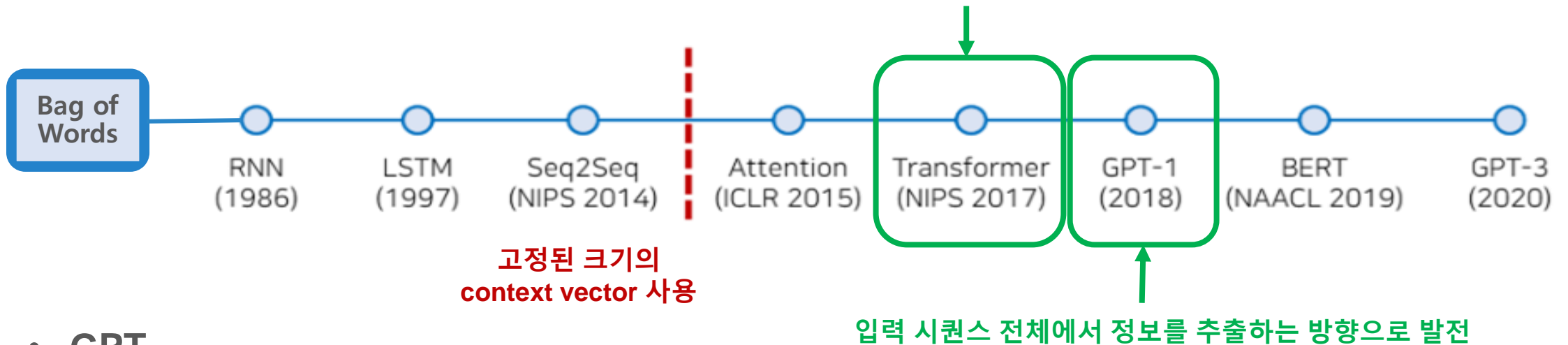


# Part 1. 자연어 처리의 발전 과정

# The Evolution of Language Models

- 2017년, Google의 논문 ‘Attention Is All You Need’ 가 Transformer의 시작점



- GPT**

- ✓ 2018년, OpenAI의 논문 ‘Improving Language Understanding by Generative Pre-Training’에서 소개된 **GPT**는 논문 제목 중 **Generative Pre-Training**의 축약어
- ✓ **GPT-1**이라고 불리며 GPT-2, GPT-3, Chat GPT(GPT-3.5), GPT-4의 시초가 됨

# Bag of Words (BoW)

- BoW
  - ✓ 단어들의 순서는 전혀 고려하지 않고, 단어들의 frequency를 기준으로 수치화
- BoW를 만드는 과정
  - ✓ 각 단어에 고유한 정수 인덱스를 부여 (단어 집합 생성)
  - ✓ 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터 생성
- BoW 예시

“정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다.”

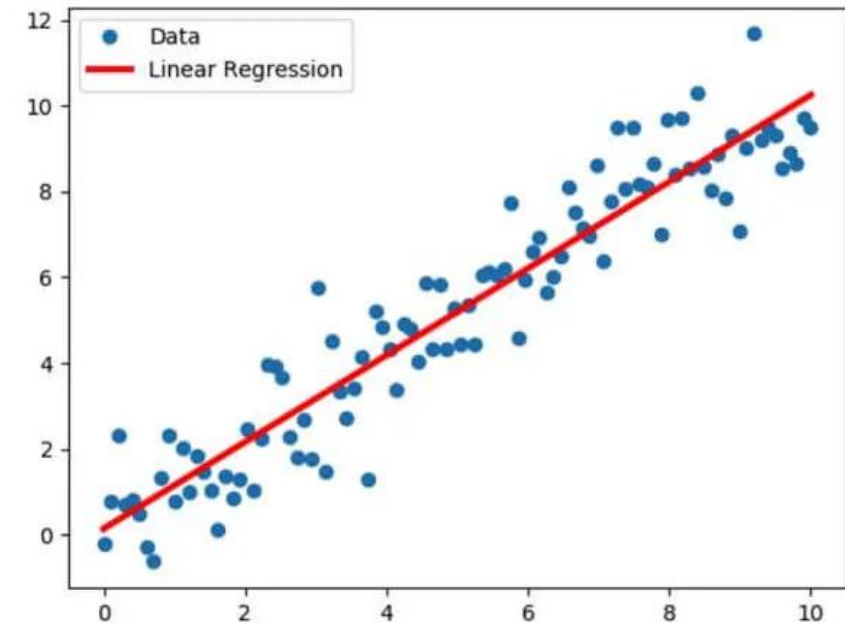
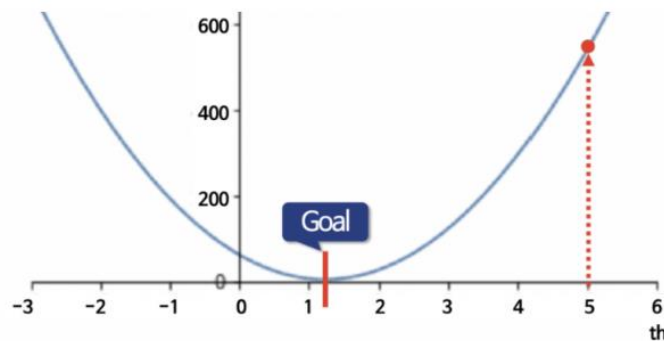
  - ✓ vocabulary : {'정부': 0, '가': 1, '발표': 2, '하는': 3, '물가상승률': 4, '과': 5, '소비자': 6, '느끼는': 7, '은': 8, '다르다': 9}
  - ✓ bag of words vector : [1, 2, 1, 1, 2, 1, 1, 1, 1, 1]

# [Essential AI concepts] Linear Regression

## Hypothesis and cost function

$$H(x) = Wx + b$$

$$\text{cost}(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$



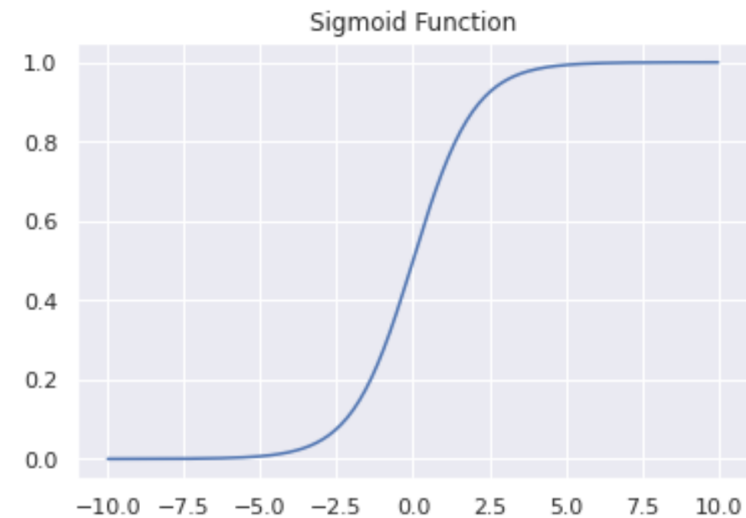
## Gradient Descent and Cost Function

# [Essential AI concepts] Sigmoid Function

## Limitations of the Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}$$

where,  $\sigma(z) \rightarrow 0$  as  $z \rightarrow -\infty$   
and  $\sigma(z) \rightarrow 1$  as  $z \rightarrow \infty$



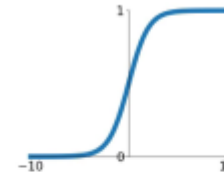
# [Essential AI concepts] Activation Functions

- **tanh** is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1).
- **tanh** is also sigmoidal (s shaped).

## Activation Functions

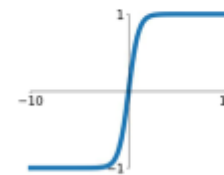
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



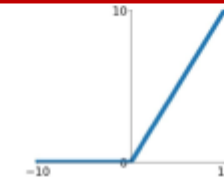
**tanh**

$$\tanh(x)$$



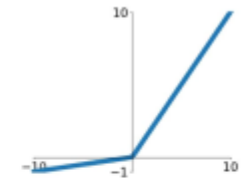
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

$$\max(0.1x, x)$$

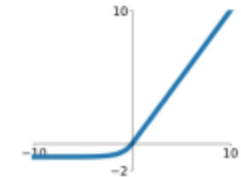


**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# [Essential AI concepts] Multiclass Classification

## Softmax Activation Function

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^N e^{z_j}}$$

$$\begin{aligned} \Pr[y_i = \text{seal}] &= \text{softmax}(\mathbf{z})_0 \\ &= \frac{e^{0.25}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.249 \end{aligned}$$






$$\begin{aligned} \Pr[y_i = \text{panda}] &= \text{softmax}(\mathbf{z})_1 \\ &= \frac{e^{1.23}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.664 \end{aligned}$$

$$\begin{aligned} \Pr[y_i = \text{duck}] &= \text{softmax}(\mathbf{z})_2 \\ &= \frac{e^{-0.8}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.087 \end{aligned}$$

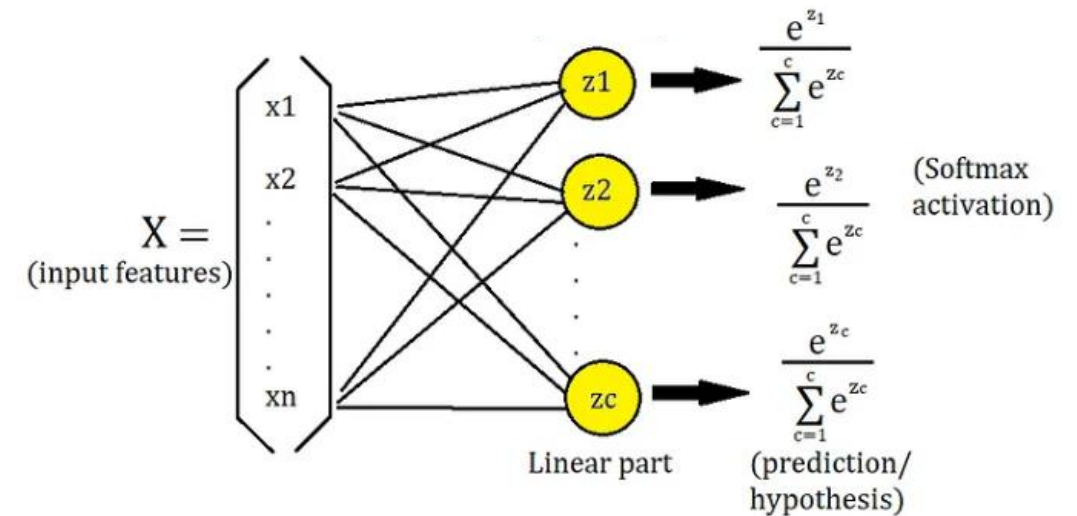
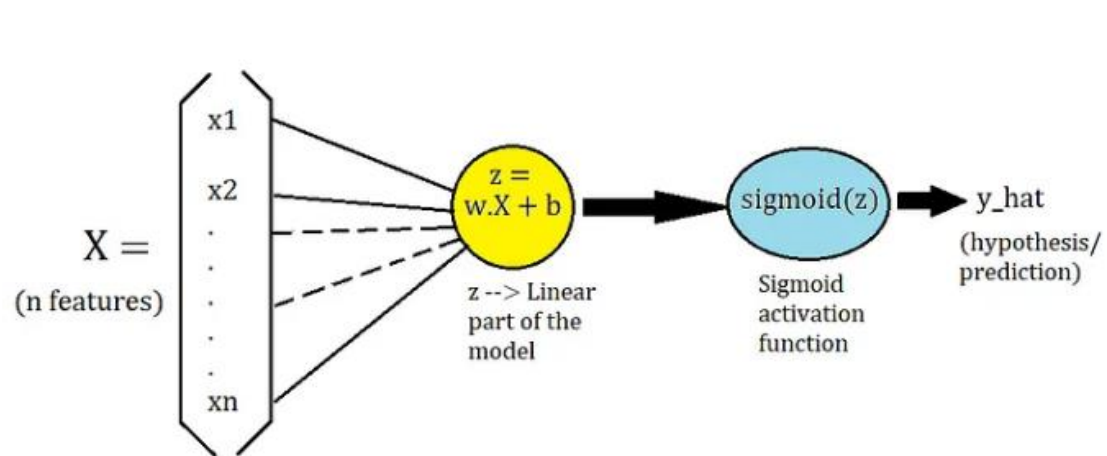


## Multiclass Classification Revisited

Class	Value	One-Hot Encoding
0		[1, 0, 0]
1		[0, 1, 0]
2		[0, 0, 1]

# [Essential AI concepts] Multiclass Classification

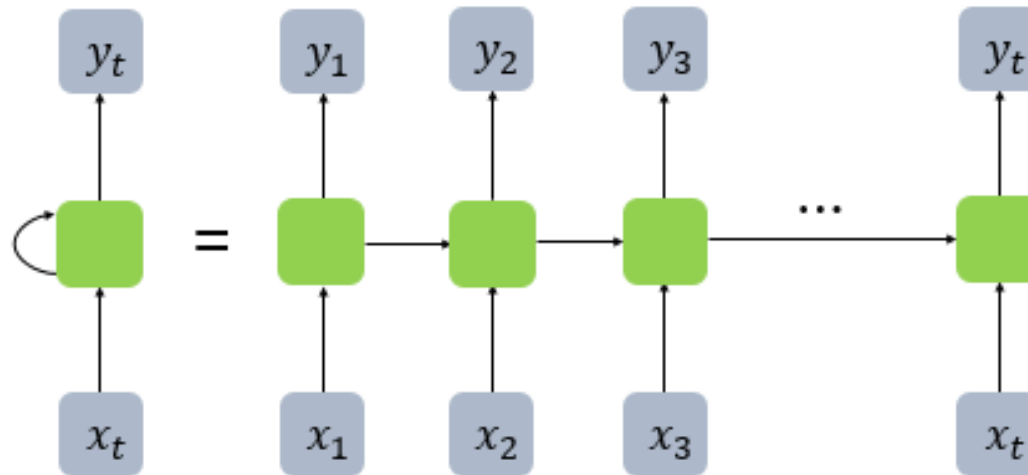
## Logistic Regression Recap





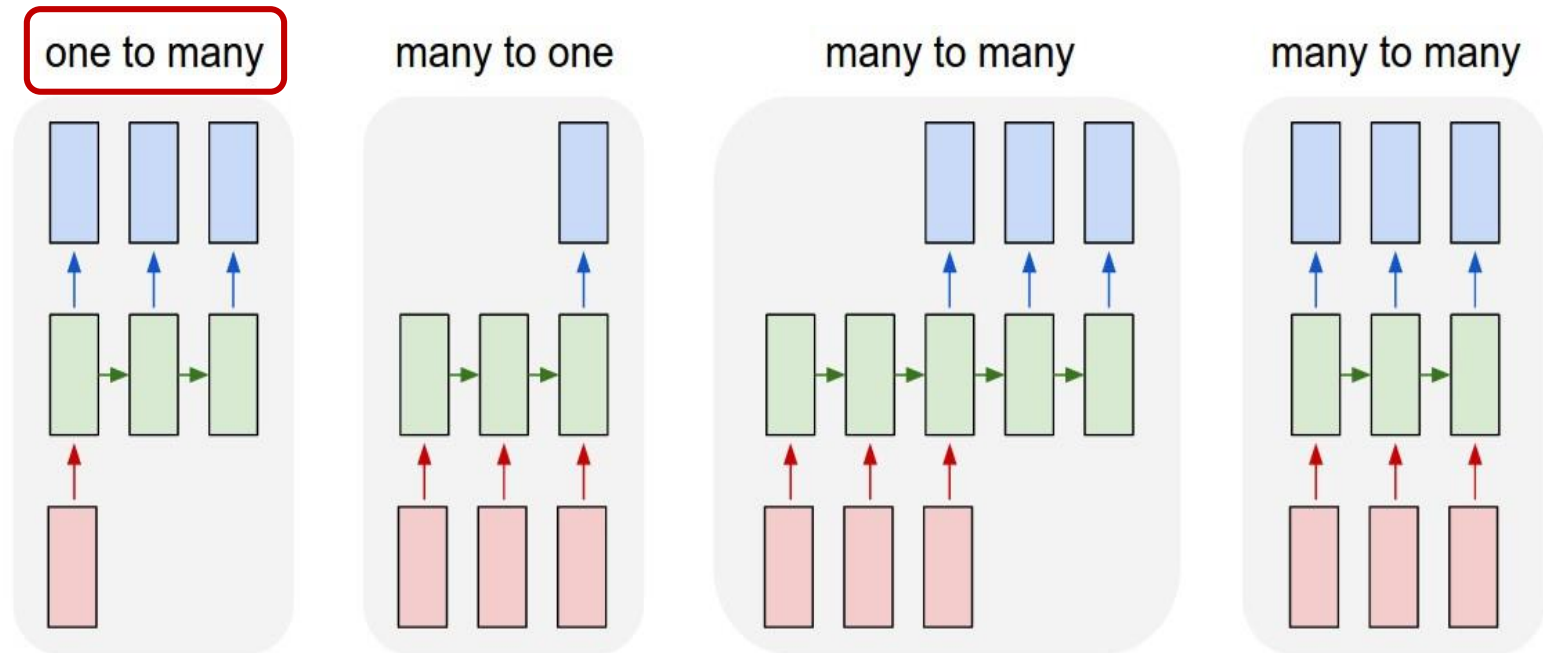
# Recurrent Neural Networks (RNN)

- RNN
  - ✓ Hidden layer에서 activation function을 통해 나온 값을 출력층 방향으로 보내면서, 다시 hidden layer 노드의 다음 계산 입력으로 보내는 순환 구조



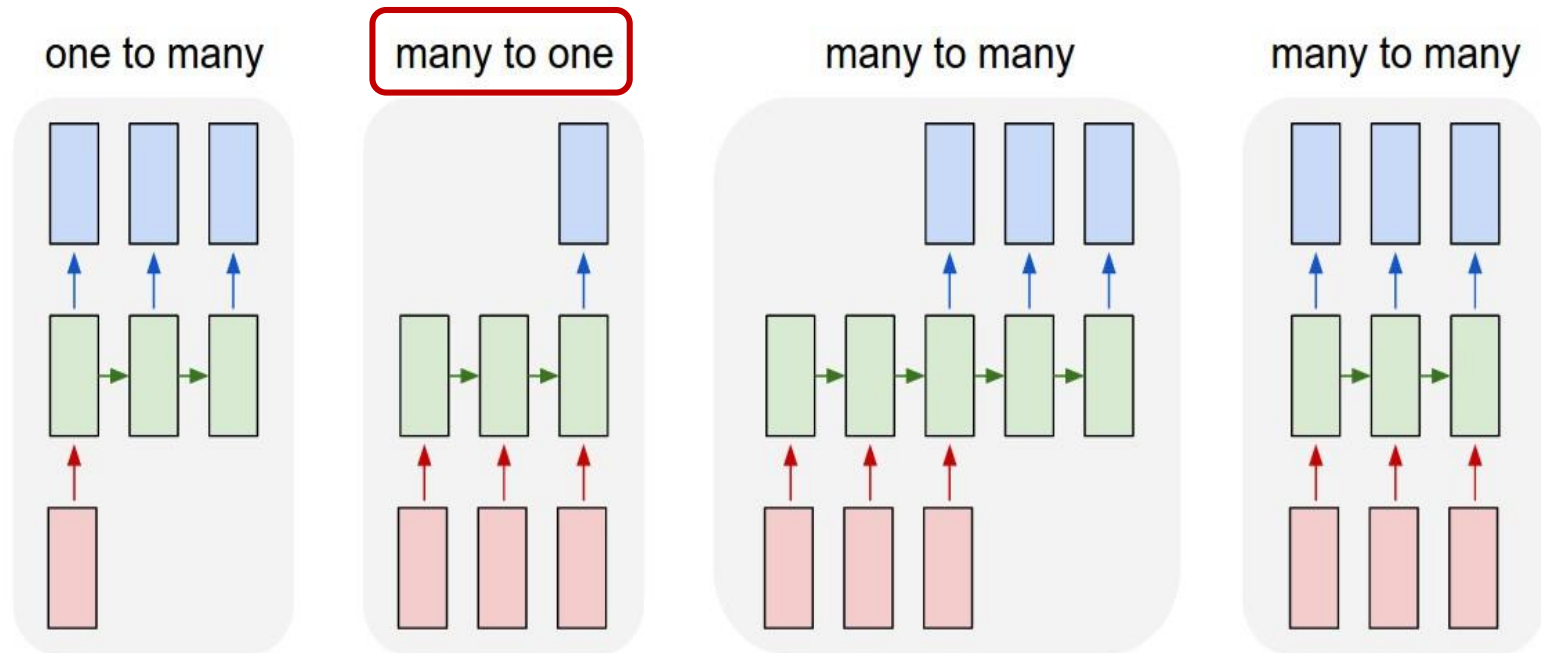
# Recurrent Neural Networks (RNN)

하나의 input 이미지에 대해 설명하는 문장 출력  
ex) image captioning



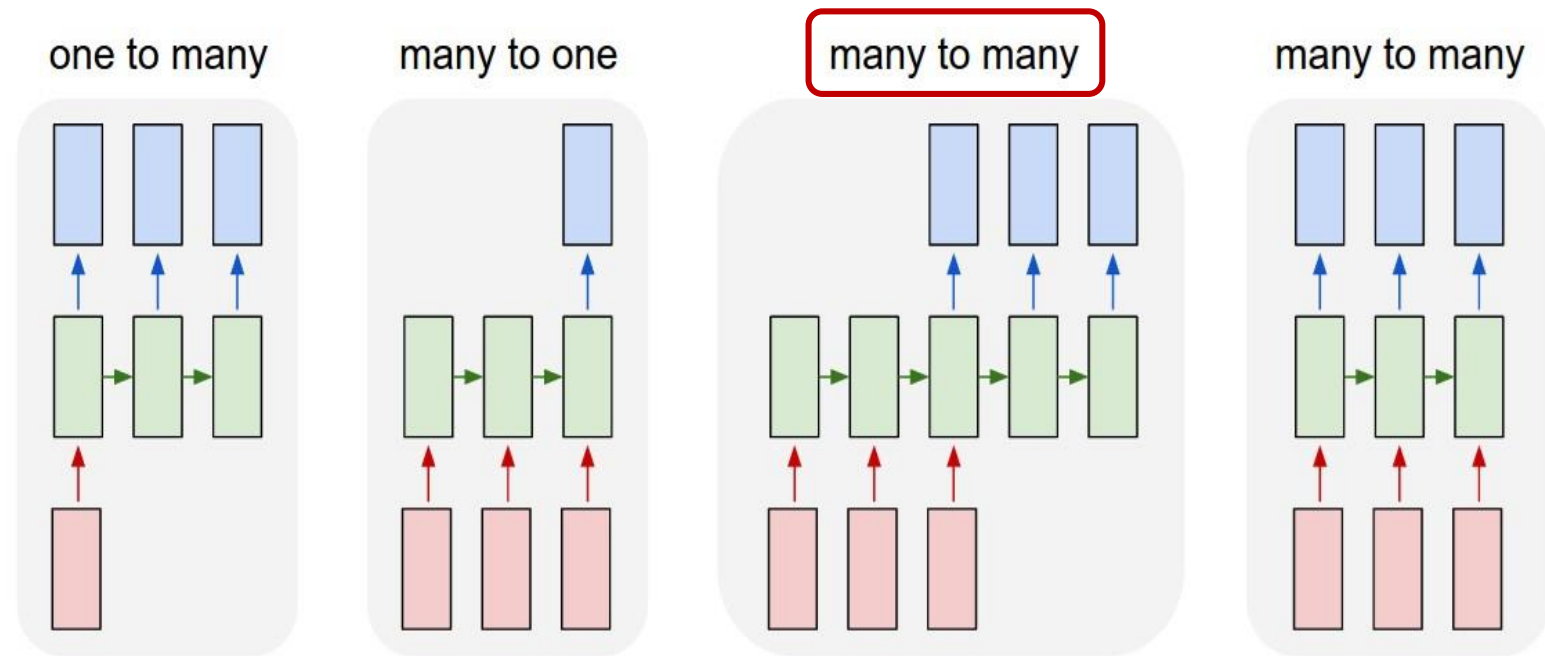
# Recurrent Neural Networks (RNN)

문장을 보고 긍정, 부정 등의 감정 분석 결과 출력  
ex) sentiment classification



# Recurrent Neural Networks (RNN)

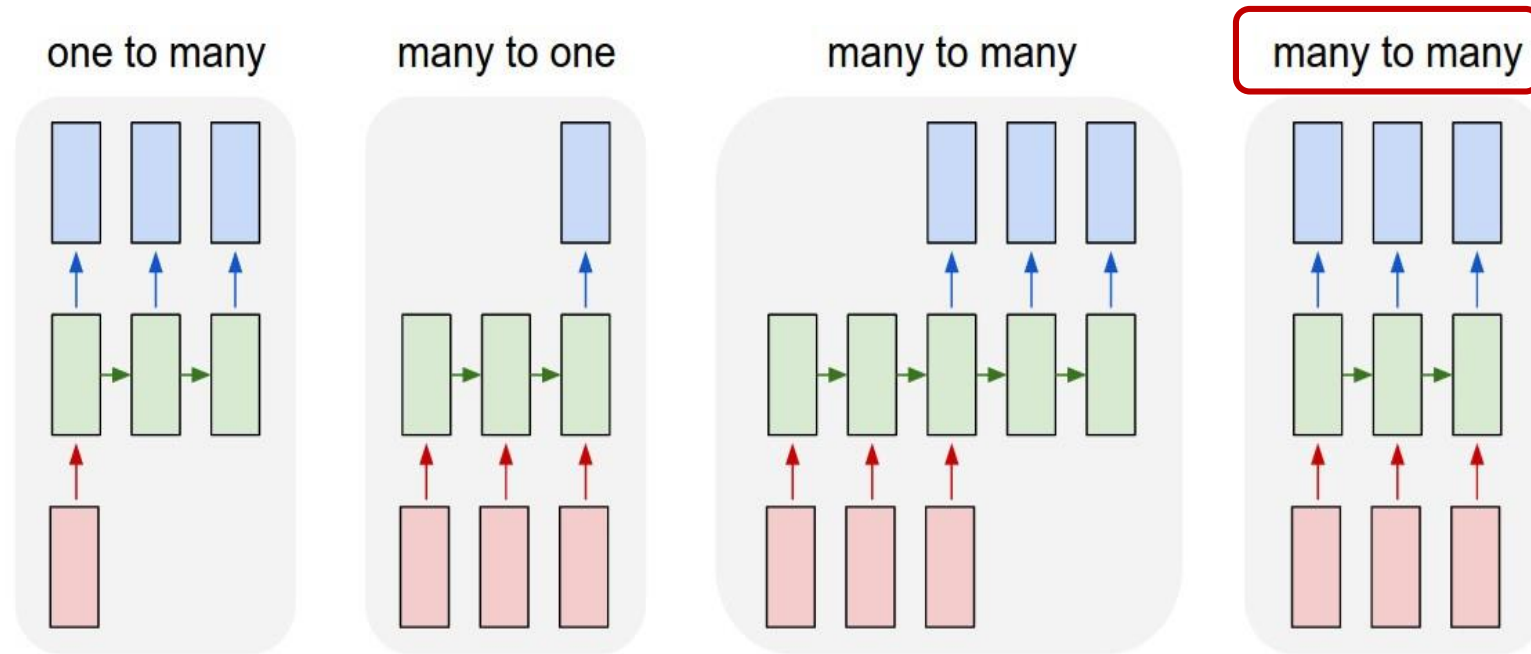
문장에서 문장을 생성  
ex) machine translation



번역의 경우, 문장을 모두 입력했을 때 해당 문장을 모두 학습한 후에 그 문장을 번역하는 것이 보다 정확한 번역이 될 수 있다.  
그래서 문장의 입력이 끝난 시점에서 출력이 생성되는 구조로 학습을 수행한다.

# Recurrent Neural Networks (RNN)

동영상 frame 분석  
ex) video classification on frame level



영상의 경우, 프레임 단위로 해당 프레임에서의 영상 자료가 무엇을 의미하는지에 대해서 인식한다.  
그래서 입력과 동시에 출력이 생성되는 구조로 학습을 수행한다.

# Recurrent Neural Networks (RNN)

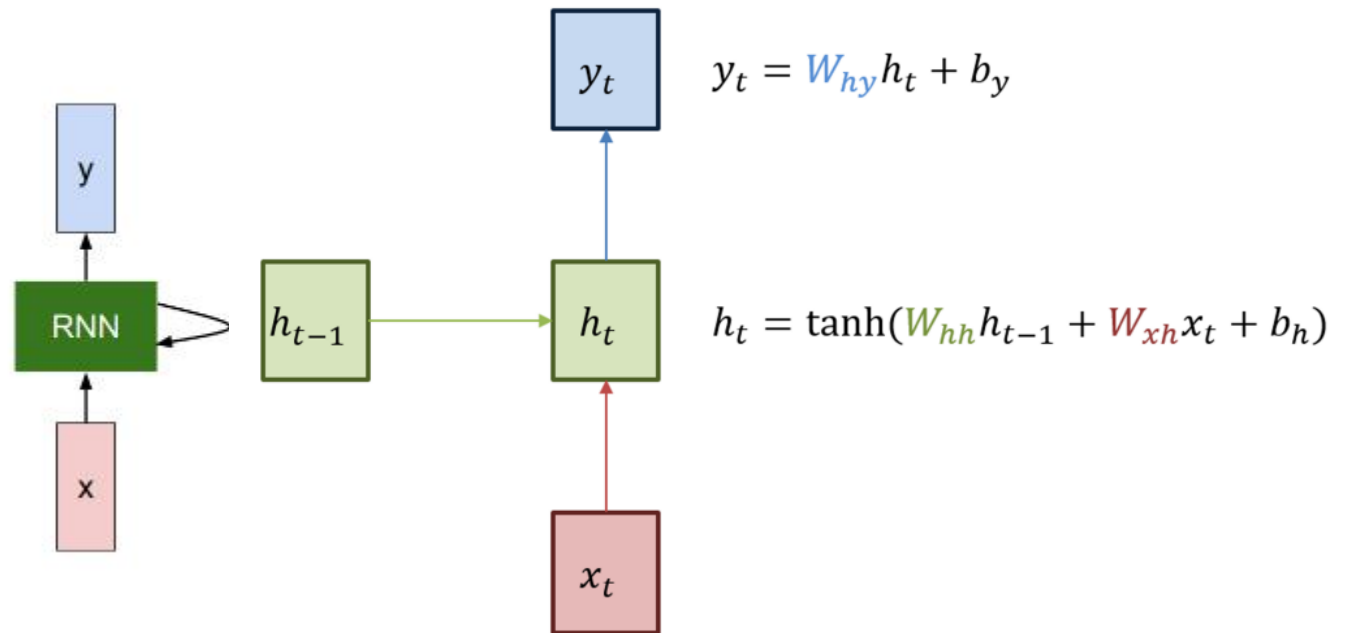
RNN에는 input vector 뿐만 아니라,  
hidden state가 추가되어  
input vector와 hidden state를 기반으로 output 계산

## Recurrent Neural Network

We can process a sequence of vectors  $\mathbf{x}$  by  
applying a **recurrence formula** at every time step:

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

new state      some function with parameters  $W$       old state      input vector at some time step



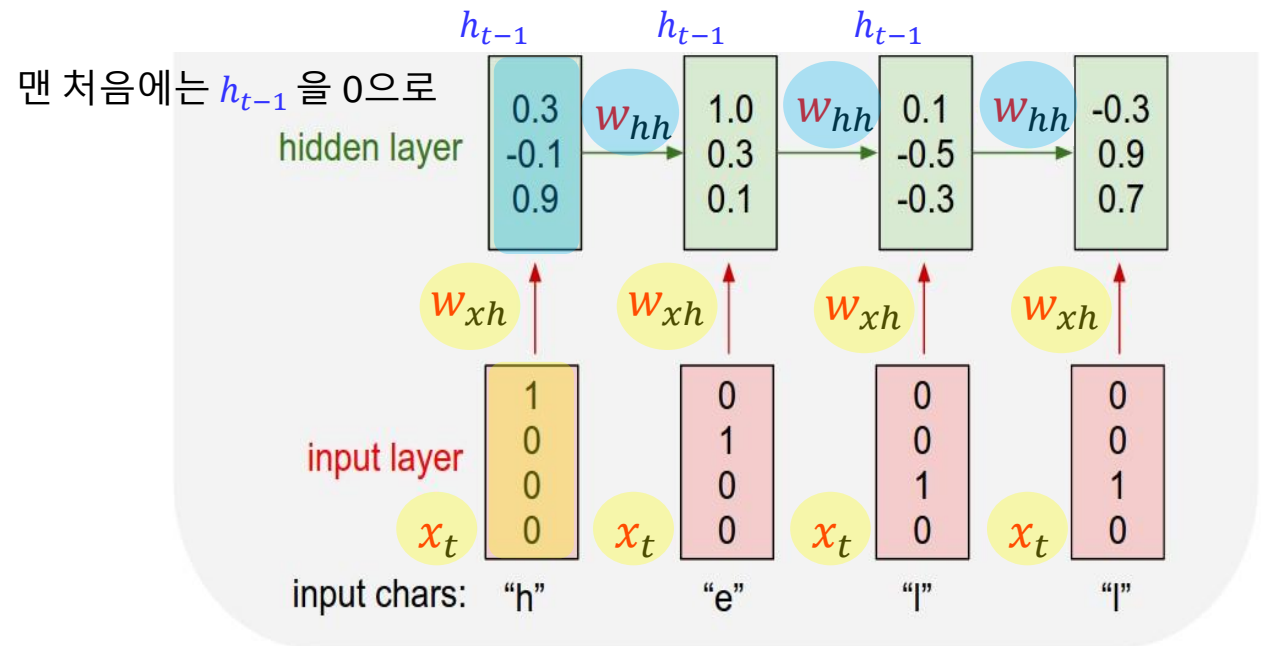
# Recurrent Neural Networks (RNN)

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$



Vocabulary:  
[h,e,l,o]

Example training sequence:  
"hello"



# Recurrent Neural Networks (RNN)

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$



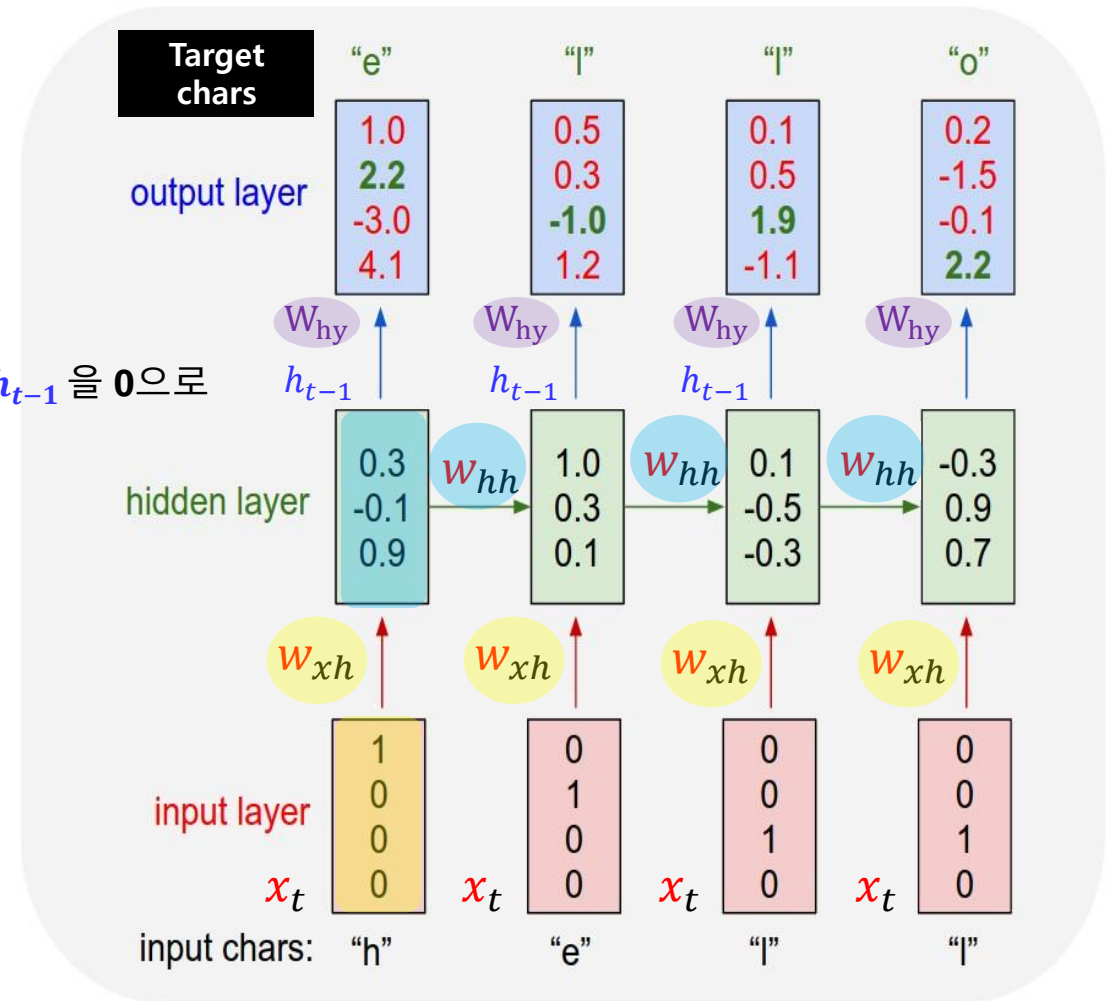
3가지 weight가 서로 다름



맨 처음에는  $h_{t-1}$  을 **0**으로

Vocabulary:  
[h,e,l,o]

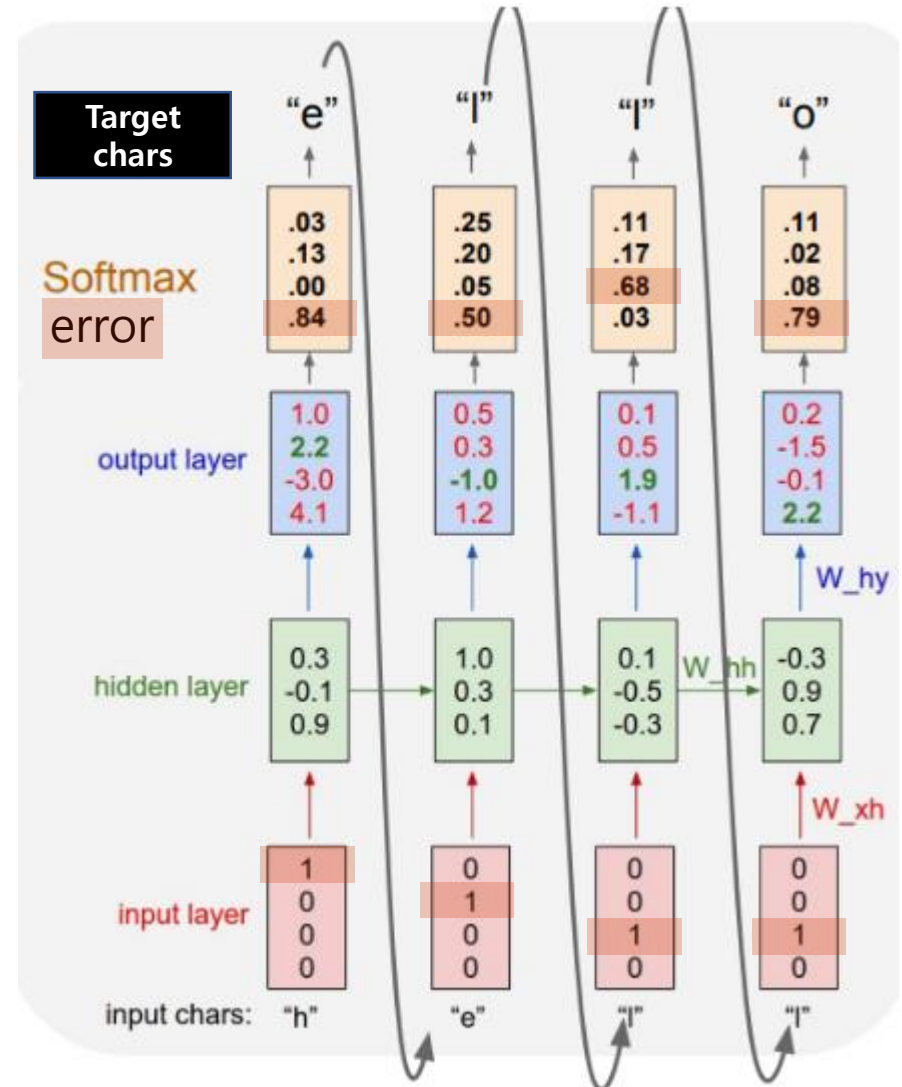
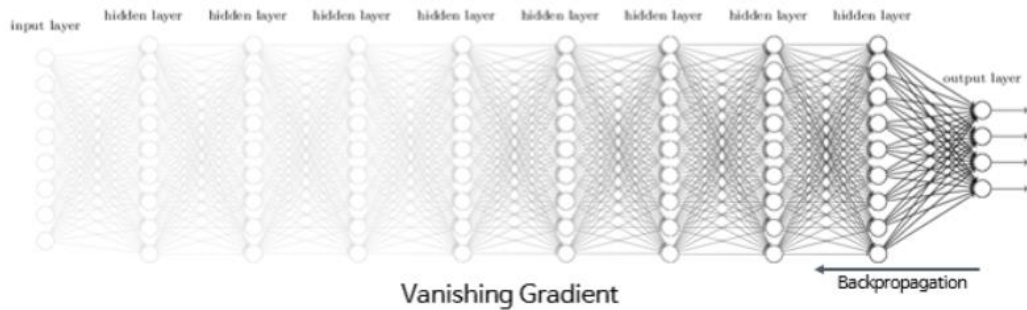
Example training sequence:  
"hello"





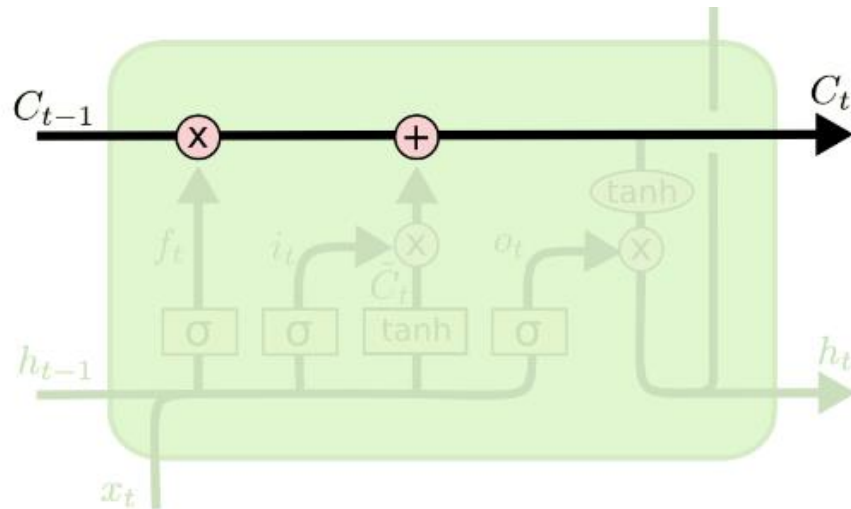
# Recurrent Neural Networks (RNN)

- Output layer 예측에 **error** 발생할 경우,  
backpropagation 수행으로 parameter값들을 갱신
- Vanishing Gradient Problem에 의해 긴 시퀀스  
데이터의 경우 장기적인 의존성 유지에 문제 발생



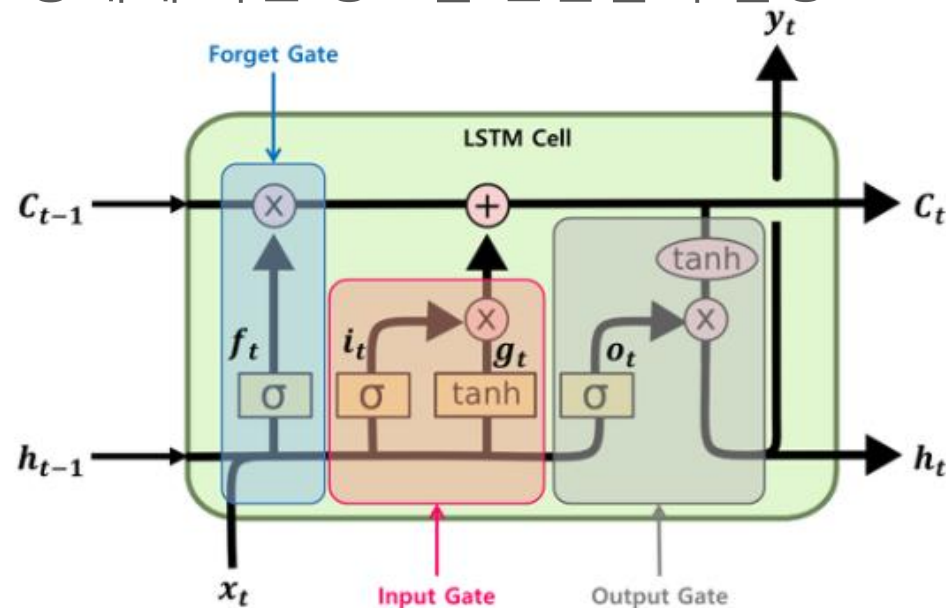
# Long Short Term Memory (LSTM)

- RNN이 출력과 먼 위치에 있는 정보를 기억할 수 없다는 단점을 보완하여 장·단기 기억을 가능하게 설계한 신경망의 구조
- LSTM 구성 요소
  - ✓ Cell state : LSTM의 메모리 역할을 수행하여 정보를 유지하고 전달



# Long Short Term Memory (LSTM)

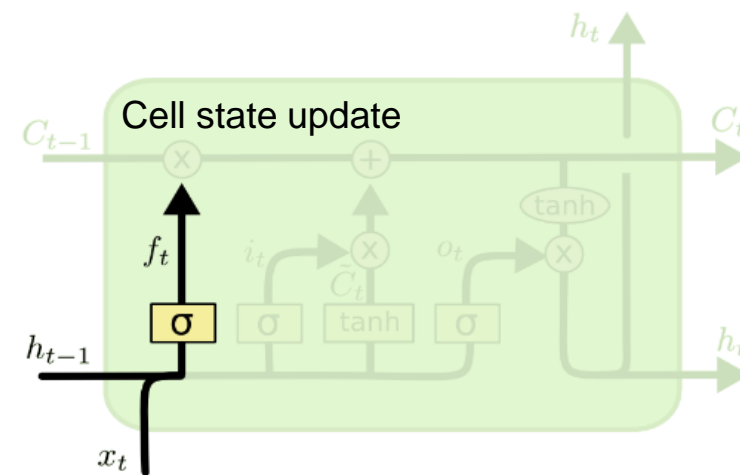
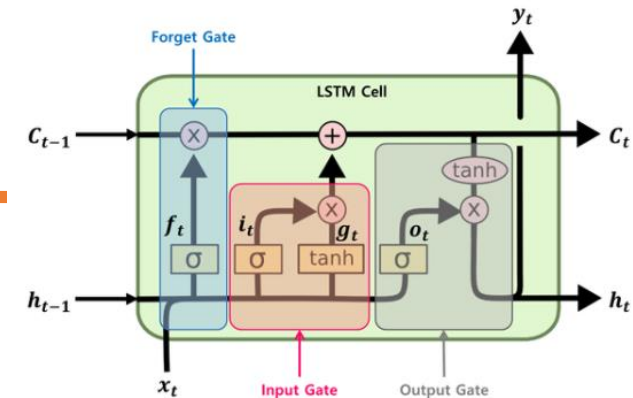
- LSTM 구성 요소
  - ✓ **Forget gate** : 이전 상태에서 어떤 정보를 잊을지 결정
  - ✓ **Input gate** : 현재 입력과 이전 상태에서 어떤 정보를 저장할지 결정
  - ✓ **Output gate** : 다음 상태에 어떤 정보를 전달할지 결정



# Long Short Term Memory (LSTM)

- Forget gate

- ✓ 이전 상태와 현재 입력을 받아 **Sigmoid 함수를 통과시킨 후, 이전 상태의 각 요소에 대한 가중치를 계산하여 가중치가 0에 가까울수록 해당 요소는 잊혀지게 함**
- ✓ 이전 상태에 저장된 정보 중에서 현재 상태와 관련이 없거나 중요하지 않은 정보를 필터링하는 역할
- ✓ 과거의 장기적인 의존성을 기억하면서도 현재 입력과 관련 있는 정보에 초점을 맞춤

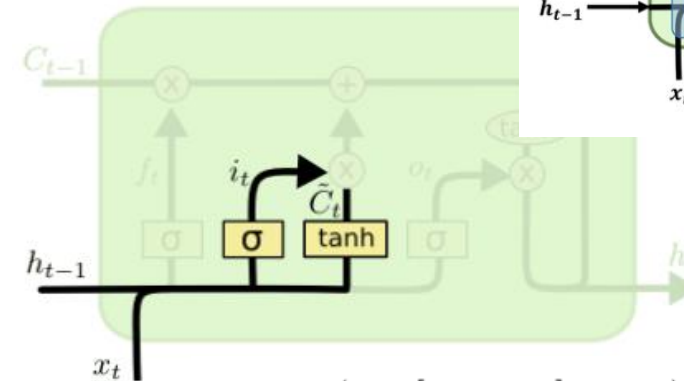


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

# Long Short Term Memory (LSTM)

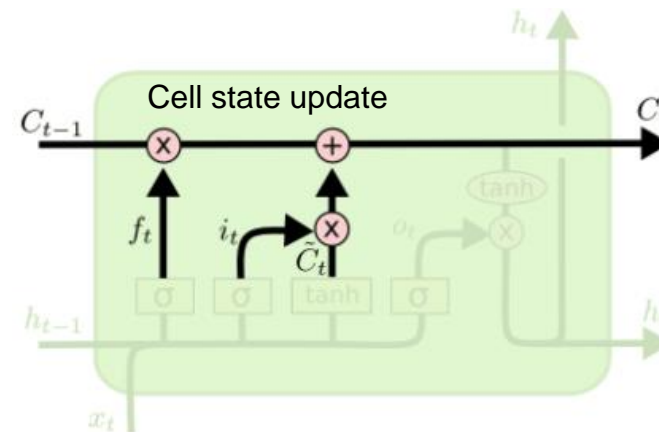
- Input gate

- ✓ 현재 입력과 이전 상태에서 어떤 정보를 저장해야 할지 결정
- ✓ 현재 입력과 이전 상태를 입력으로 받아 Sigmoid 함수와 Tanh 함수를 통과시킨 후, 새로운 후보 상태를 계산
- ✓ **Sigmoid 함수는 어떤 정보를 저장할지 결정**하고, tanh 함수는 후보 상태의 값 범위를 조정함



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

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

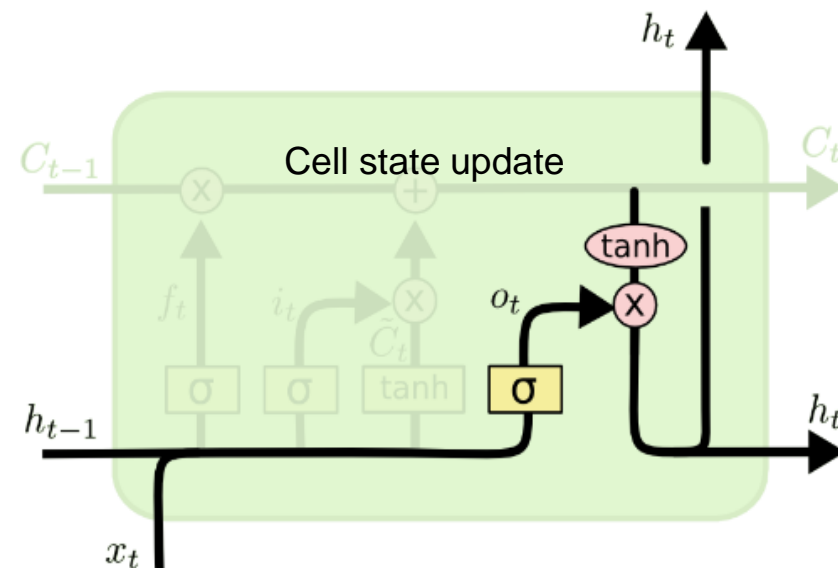
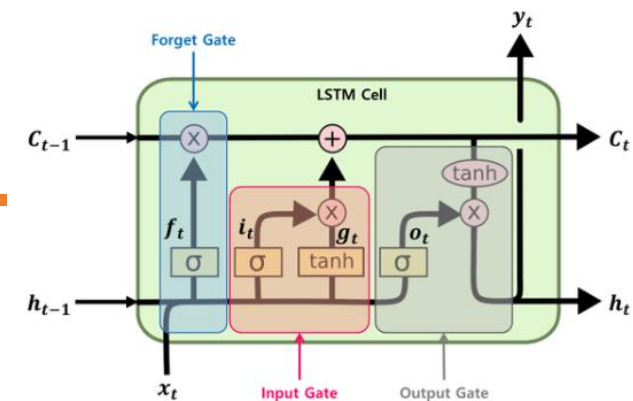


앞으로 들어오는 새로운 정보 중에서 어떤 것을 cell state에 저장할 것인지를 결정

# Long Short Term Memory (LSTM)

- **Output gate**

- ✓ 다음 상태에 어떤 정보를 전달할지 결정
- ✓ 현재 입력과 이전 상태를 입력으로 받아 Sigmoid 함수와 tanh 함수를 통과시킨 후, 출력 상태를 계산
- ✓ **Sigmoid 함수는 어떤 정보를 출력할지 결정**하고, tanh 함수는 출력 상태의 값 범위를 조정함
- ✓ Sigmoid 함수의 가중치가 0에 가까울수록 해당 요소는 출력에 반영되지 않고, 가중치가 1에 가까울수록 해당 요소는 출력에 중요하게 반영

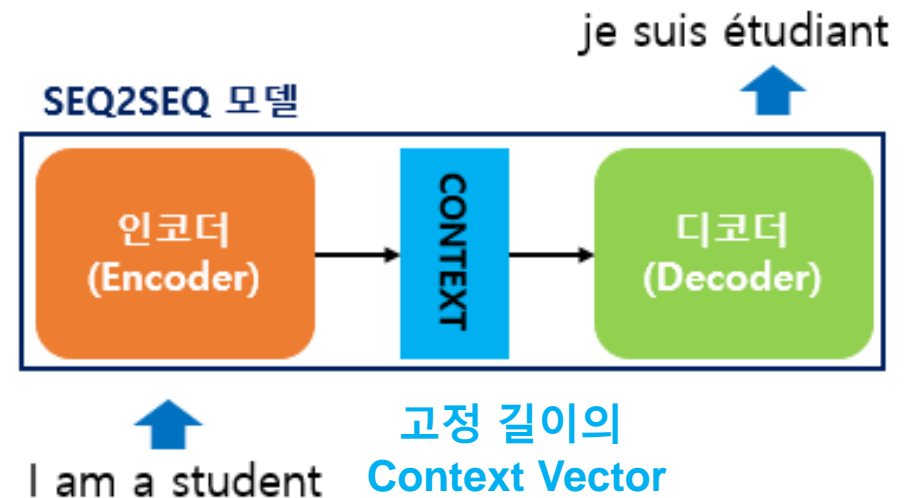
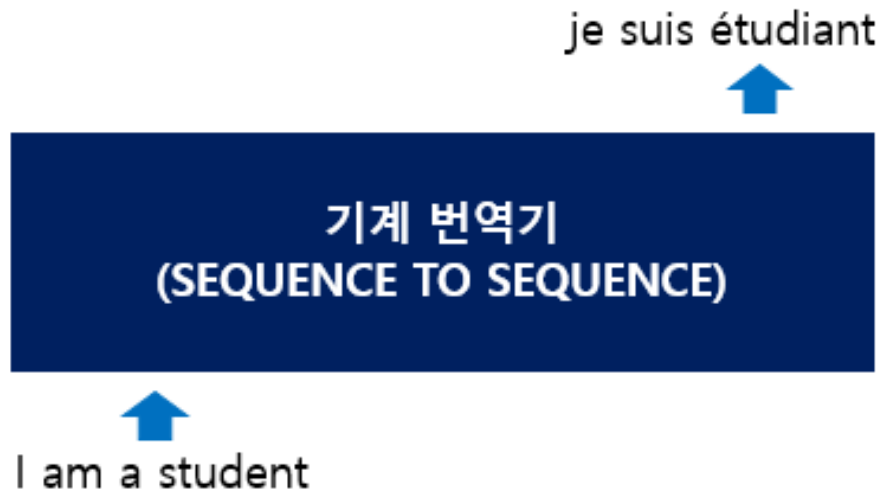


$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

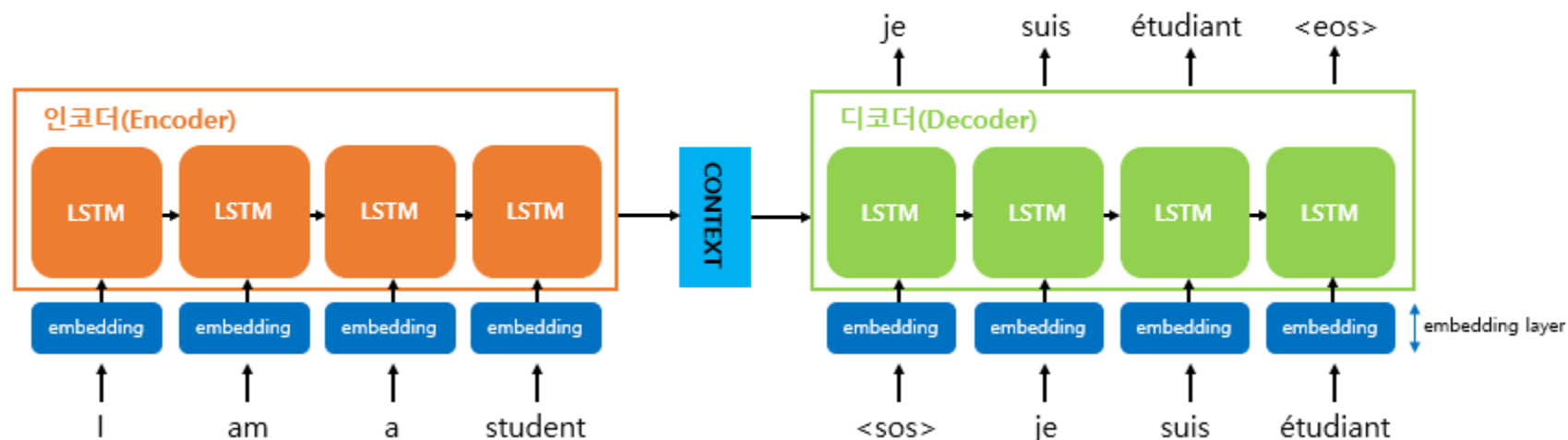
$$h_t = o_t * \tanh(C_t)$$

# Seq2Seq

- 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 모델
  - ✓ 기계번역, 챗봇



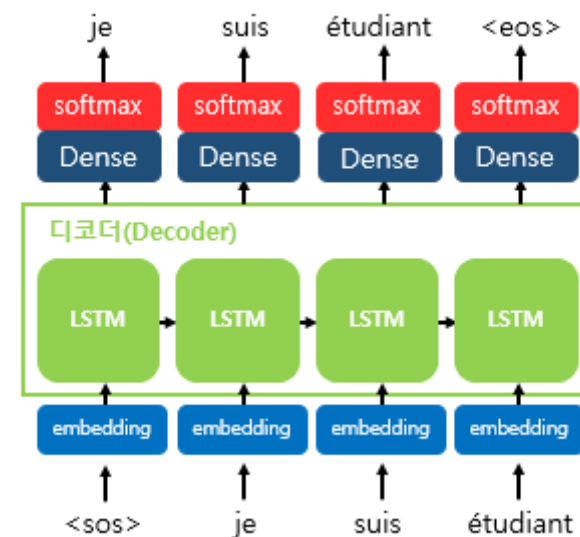
# Seq2Seq



embedding layer

I	0.157	am	0.78	a	0.75	student	0.88
	-0.25		0.29		-0.81		-0.17
	0.478		-0.96		0.96		0.29
	-0.78		0.52		0.12		0.48

보통 embedding vector는 수백 개의 차원으로 구성

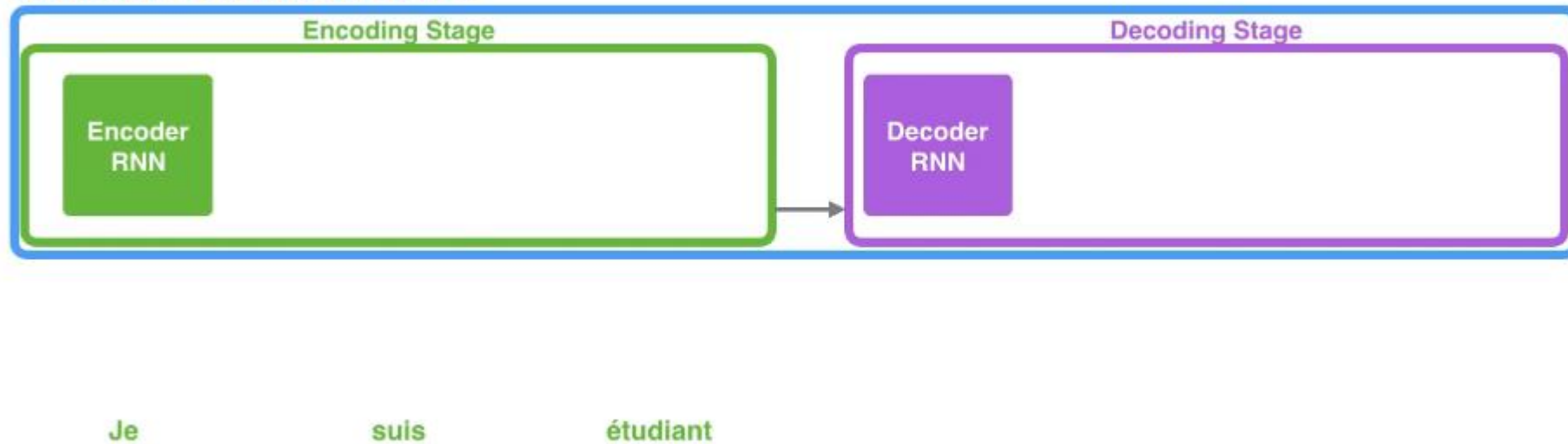




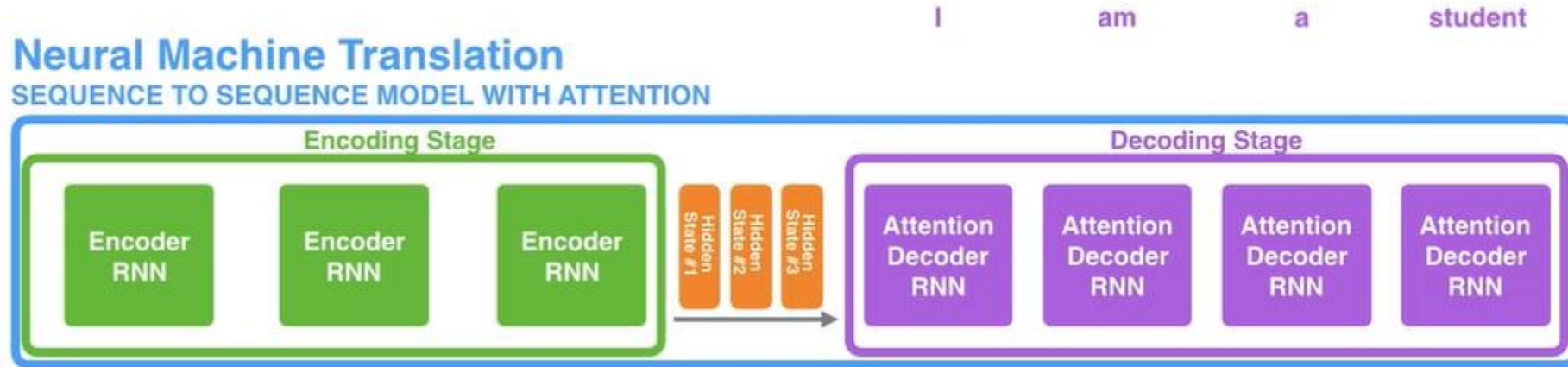
# Seq2Seq

The **context vector** turned out to be a **bottleneck** for these types of models.

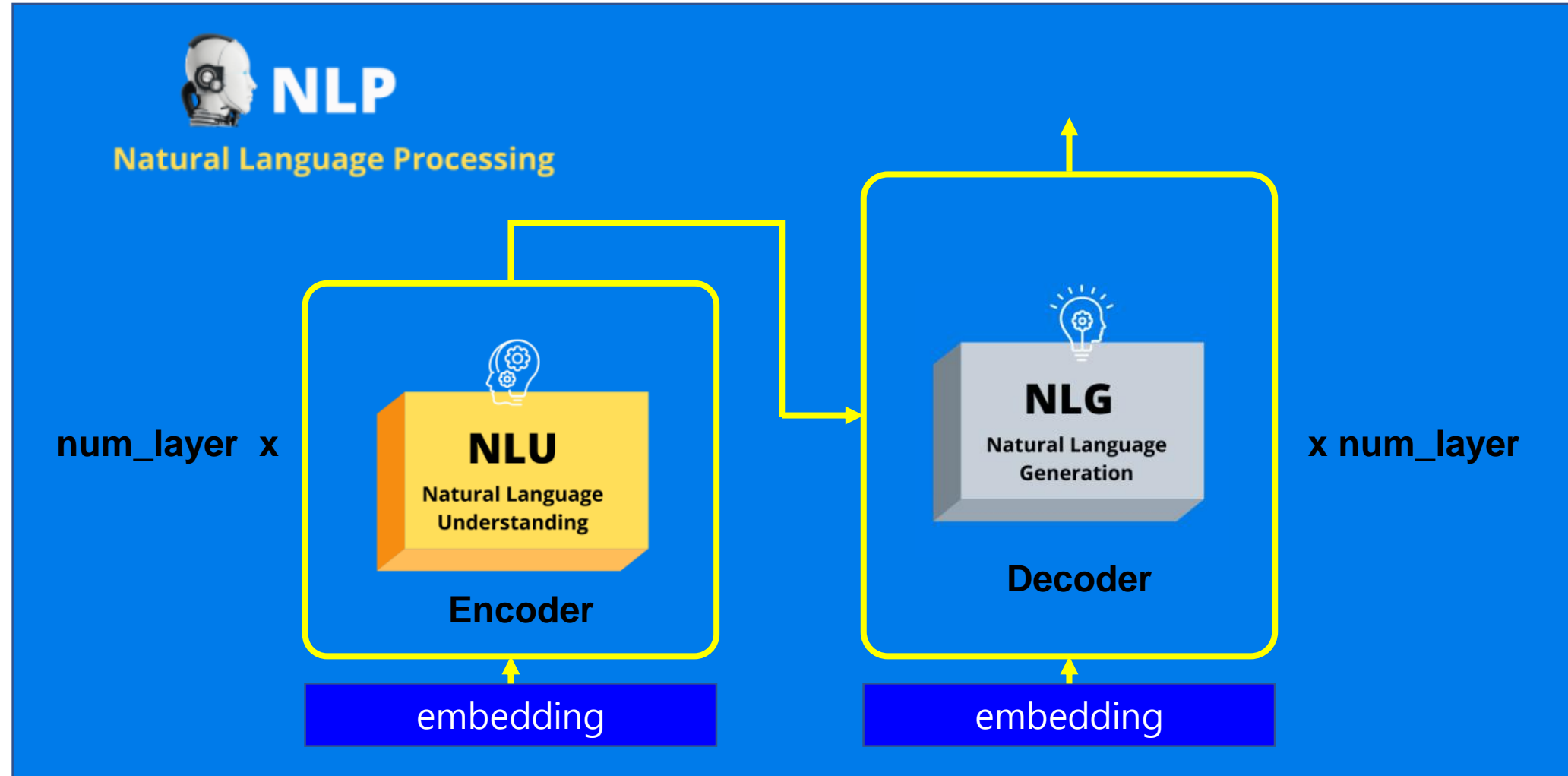
## Neural Machine Translation SEQUENCE TO SEQUENCE MODEL



# Attention



# Encoder-Decoder



# Attention Encoder

---

- 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 Decoder

---

- The encoder hidden state is most associated with **a certain word** in the input sentence
- Give each hidden state **a score**
- Multiply **each hidden state by its softmaxed score**, thus **amplifying** hidden states **with high scores**, and **drowning** out hidden states **with low scores**

# Attention Decoder

## Attention at time step 4

1. Prepare inputs



$h_1$



$h_2$



$h_3$

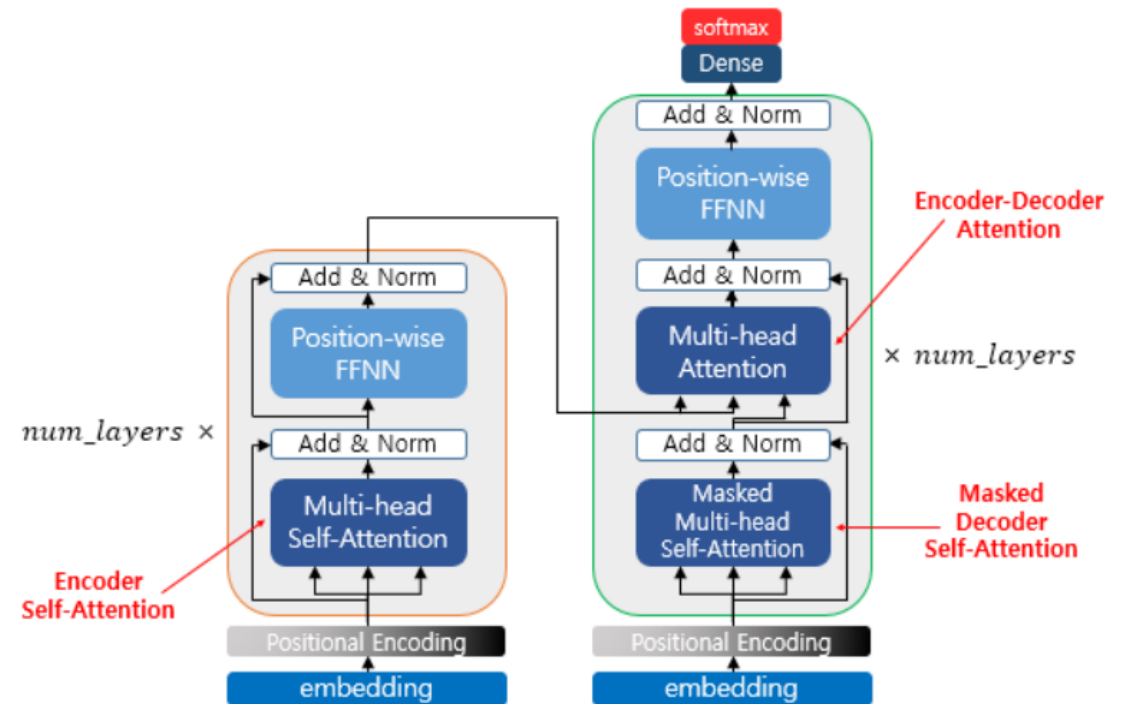
Encoder  
hidden  
states



Decoder hidden  
state at time step 4

# Attention Mechanism

- 기본 아이디어는 Decoder에서 출력 단어를 예측하는 Time Step마다 Encoder에서의 전체 입력 문장을 다시 한 번 참고한다는 점
- 전체 입력 문장을 전부 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야 할 단어와 연관이 있는 입력 단어 부분을 좀 더 Attention해서 본다는 점



# Transformer

- 2017년, Google 공개한 신경망 Transformer가 시작점!!

- Attention 기반의  
Encoder-Decoder 알고리즘 사용

arXiv:1706.03762v5 [cs.CL] 6 Dec 2017

## 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\*<sup>†</sup>  
University of Toronto  
aidan@cs.toronto.edu

Lukasz Kaiser\*  
Google Brain  
lukaszkaizer@google.com

Illia Polosukhin\*<sup>‡</sup>  
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.

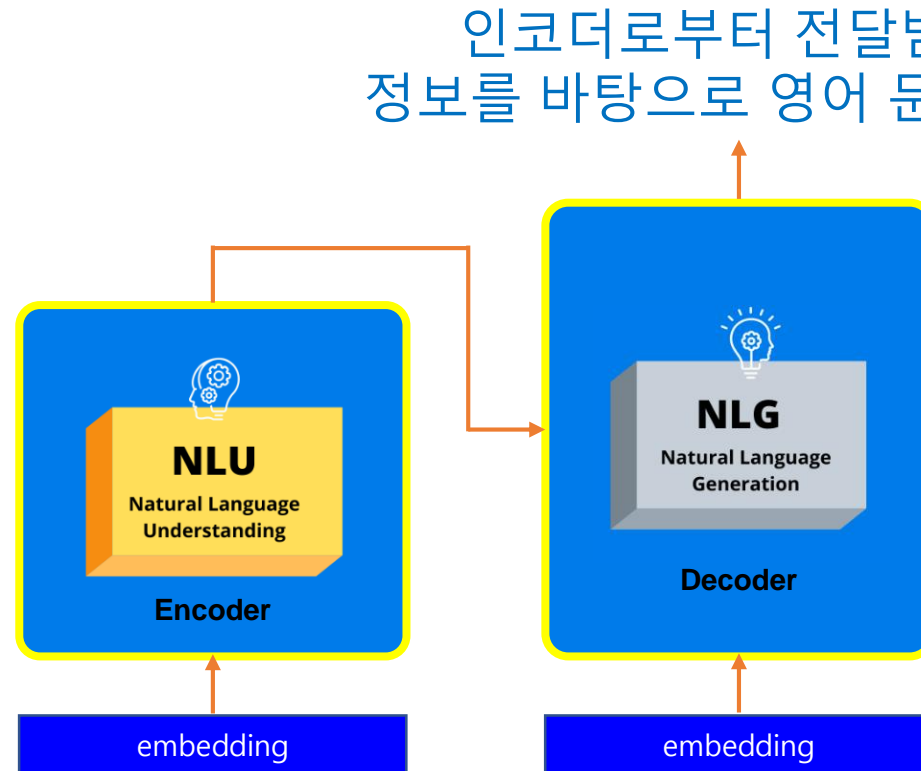
### 1 Introduction



# Transformer's Encoder-Decoder

**2019년, Google**

**BERT(Bidirectional Encoder Representations from Transformers):** Transformer의 Encoder 블록만 쌓아서 구현



입력된 한글 문장에 대한 문장 구조나 의미 등을 내부의 인공 신경망을 통해 파악

**2018년, OpenAI**

**GPT(Generative Pre-trained Transformer):** Transformer의 Decoder 블록만 쌓아서 구현

[Decoder Large Language Models]

- **MicroSoft:** 1,750억개 파라미터를 가진 GPT-3
- **Google:** 1,370억개 파라미터를 가진 LaMDA
- **Naver:** 2,040억개 파라미터를 가진 하이퍼클로바