# CS224n Review - 2021

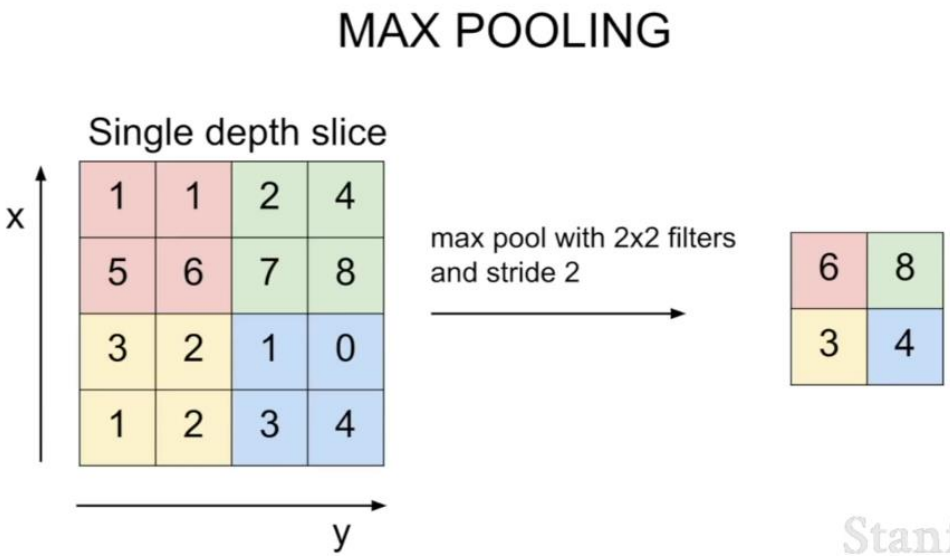
## Lecture 1 ~ 8

2024.08.02

유하영

# CNN

## Pooling




### Max pooling

필터 내에서 가장 큰 값을 선택하여 중요한 특징을 강조

- 이미지의 엣지, 텍스처와 같은 중요한 디테일을 유지하는 데 유리
- 위치 변동에 강하다.
- 모델이 불필요한 세부사항에 과적합되는 것을 방지하는 데 도움

### Average pooling

필터 내의 모든 값을 평균내어 특징을 생성

- 모든 값을 고르게 반영하므로 특징이 뭉개질 수 있음
- 이미지의 디테일이 희석되는 경향이 있다.
- 불변성이 상대적으로 덜하여 이미지의 위치 변동에 민감할 수 있음.

### 자연어 처리에서의 Max Pooling

- **중요 특징 강조**

  : 문장 내에서 가장 두드러진 단어들이 문장 임베딩에 반영될 수 있음
- **잡음 제거**

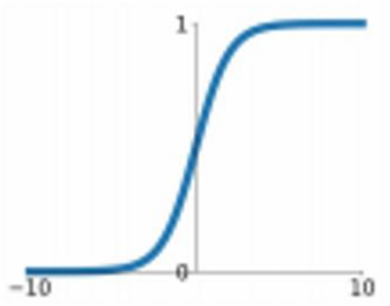
  : 상대적으로 작은 값을 가지는 노이즈나 불필요한 세부사항을 제거하여 중요한 정보만 보존
- **간단한 계산**: 차원 축소

# Activation Function

신경망이 비선형성을 학습하고, 복잡한 패턴을 인식할 수 있도록 하기 위함

**Sigmoid**

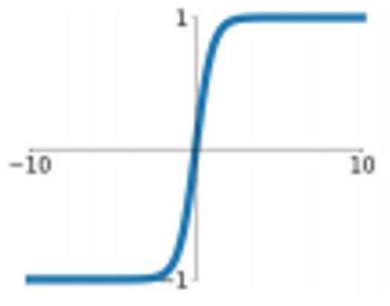
$$\sigma(x) = \frac{1}{1+e^{-x}}$$




1. Saturated neurons ‘kill’ the gradients
2. not zero-centered (gradient 균형 X)

**tanh**

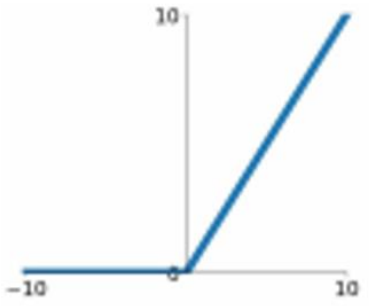
$$\tanh(x)$$




1. Saturated neurons ‘kill’ the gradients
2. not zero-centered (gradient 균형 X)

**ReLU**

$$\max(0, x)$$




1. Saturated neurons ‘kill’ the gradients (양수이면 good)
2. not zero-centered (gradient 균형 X)

계산효율이 좋음

음의 영역에서는 saturation -> dead ReLU(gradient 절반을 죽임)

**Leaky ReLU - 작은 음의 기울기

# CS224n

- 1강 - word embedding
- 2강 - Neural Classifier

word embedding
Word2Vec

- ~~3강 - Neural Networks~~
- 4강 - Syntactic Structure & Dependency Parsing

Dependency Parsing
(문장 구조 파악)

- 5강 - RNN
- 6강 - LSTM
- 7강 - seq2seq, Attention

RNN, LSTM …
Beam search
perplexity, BLUE
seq2seq

- 8강 - self Attention, Transformer

Transformer

# Lecture 1 - 2

## Word Embedding

## Word2Vec

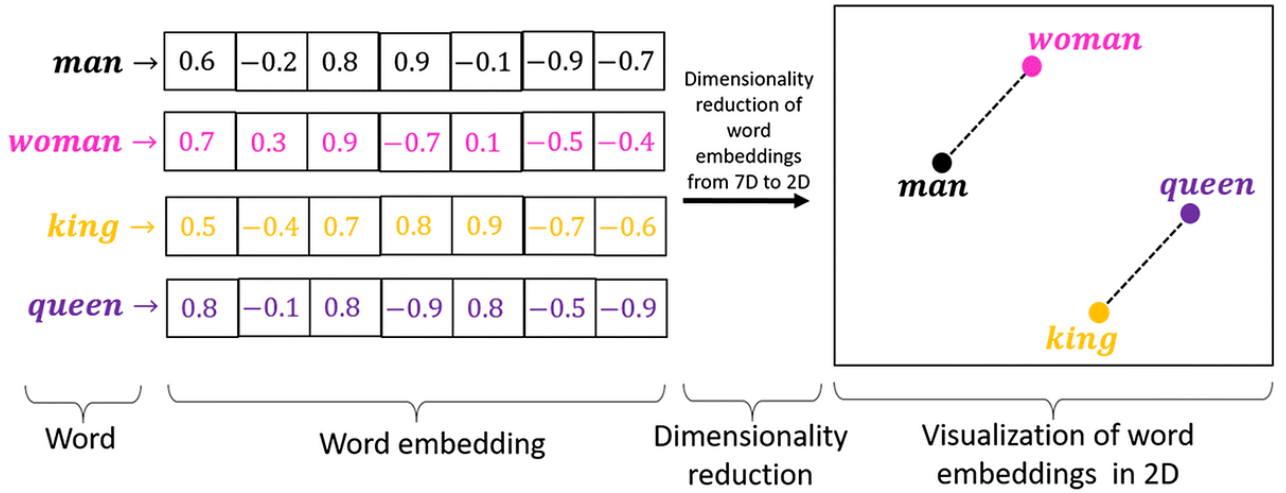
# Word Vectors

## One-hot vector


Sparse Vector

문제
- 단어가 많아지면, 벡터의 크기가 매우 커진다.
- 단어들 사이의 관계를 나타내지 못한다.

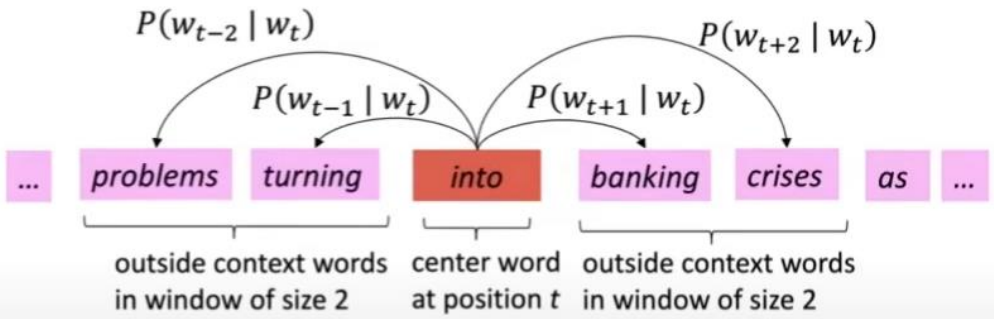
## Word Vector




- context를 보고 단어의 뜻을 예측한다.
- distributed representation 사용
- dense-vector로 표현
  (모든 숫자가 0이 아닌 숫자로 구성)
- 2차원 공간에 벡터들을 투영해, 시각화

# Word2Vec

Example windows and process for computing $P(w_{t+j} \mid w_t)$




확률을 알아내는 것이 목표
→ 단어 벡터로 정의(초반에는 모두 random하게 초기화)
→ 중심단어의 문맥단어를 예측할 확률
→ 계속해서 단어 벡터를 조정
→ 중앙 단어 맥락에서 실제로 발생하는 단어에 할당되는 확률 극대화

전체단어수 T로 나누어 평균을 구함 → 각 샘플의 기여도를 동일하게 맞춘다.

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$ ?
- **Answer:** We will *use two* vectors per word *w*:
  - $v_w$ when *w* is a center word
  - $u_w$ when *w* is a context word

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

① Dot product compares similarity of *o* and *c*.
$u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

# Word2Vec

## Two model variants

1. Skip-gram(SG) 더 자연스럽다!

`center word`을 가지고 `outside(context) word` 예측

2. Continuous Bag of Words(CBOW)

`outside(context) word` 을 가지고 `center word` 예측

$$\max \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log P(w_{t+j}|w_t)$$

softmax로 모델을 훈련시킬 수 있다. → 하지만 비싼 훈련방법! → negative sampling 방법 제안!

# Skip-gram Negative Sampling

softmax를 사용하는 대신 → **logistic function(sigmoid)** 사용

## Skip-gram negative sampling(SGNS)

- From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)
- Overall objective function (they maximize):

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$$

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j\sim P(w)}\left[\log \sigma\left(-u_j^T v_c\right)\right]$$

전체 단어 쌍을 고려하는 대신,
일부 부정적인 샘플(실제로 관찰된 주변 단어들과 무작위로 선택된 negative sampling)
을 포함하여 손실 함수를 최적화

# Lecture 5 - 7

**RNN**
**LSTM**
**seq2seq**

# RNN

output distribution

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

hidden states

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

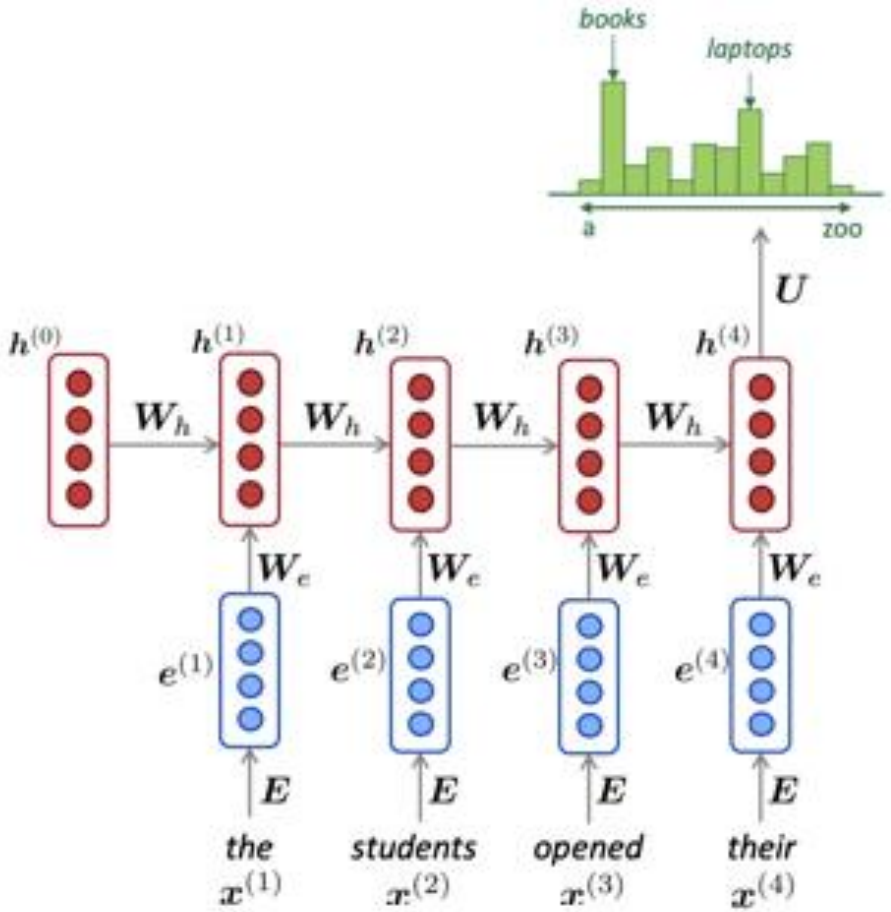
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

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

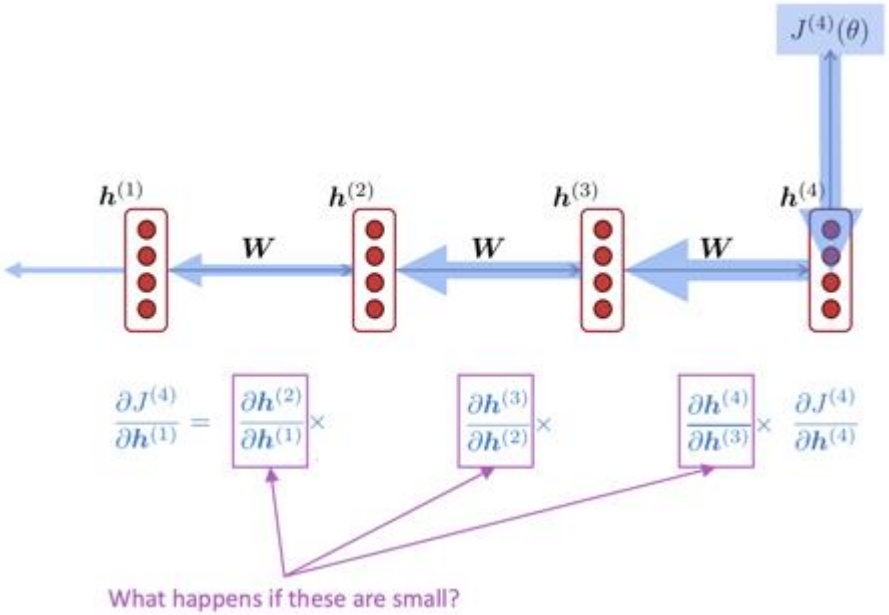
words / one-hot vectors

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




Problem

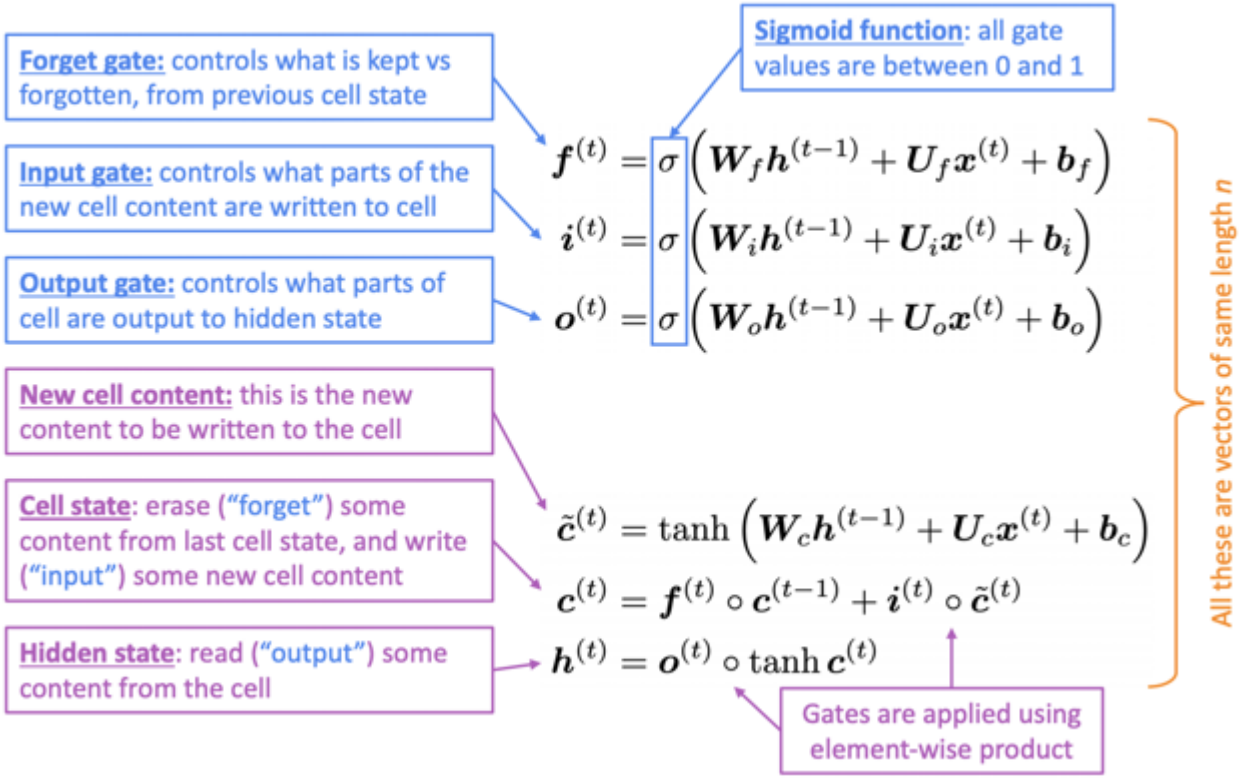
Vanishing gradient!




→ 멀리 떨어진 단어들 사이의 dependency를 학습하지 못한다는 문제발생!

# LSTM

## LSTM




- 기존의 hidden state와 함께 cell state를 사용
- Cell state는 필요한 정보를 저장하기 위한 메모리
  → 별도의 gate를 유지해서 cell state에 값을 읽을지, 쓸지, 지울지 판단

## Bi-LSTM

Forward RNN $\overrightarrow{h}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{h}^{(t-1)}, x^{(t)})$

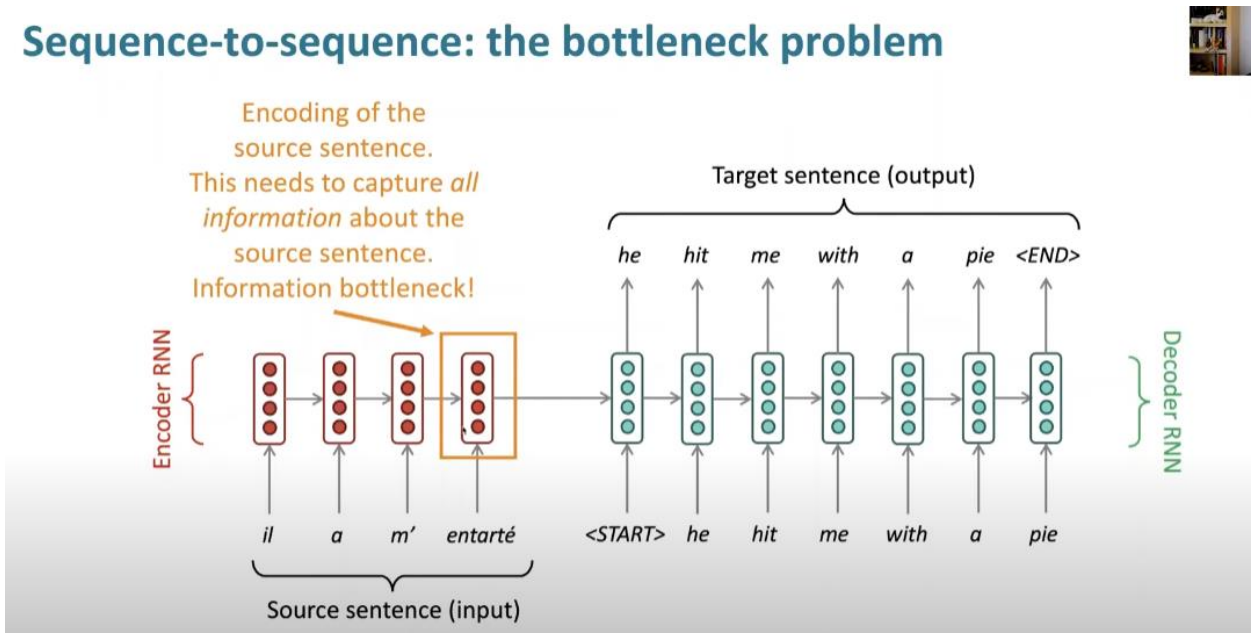
Backward RNN $\overleftarrow{h}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{h}^{(t+1)}, x^{(t)})$

Concatenated hidden states $h^{(t)} = [\overrightarrow{h}^{(t)}; \overleftarrow{h}^{(t)}]$

두 RNN이 생성한 hidden state를 연결해서
전체 모델의 hidden state로 사용

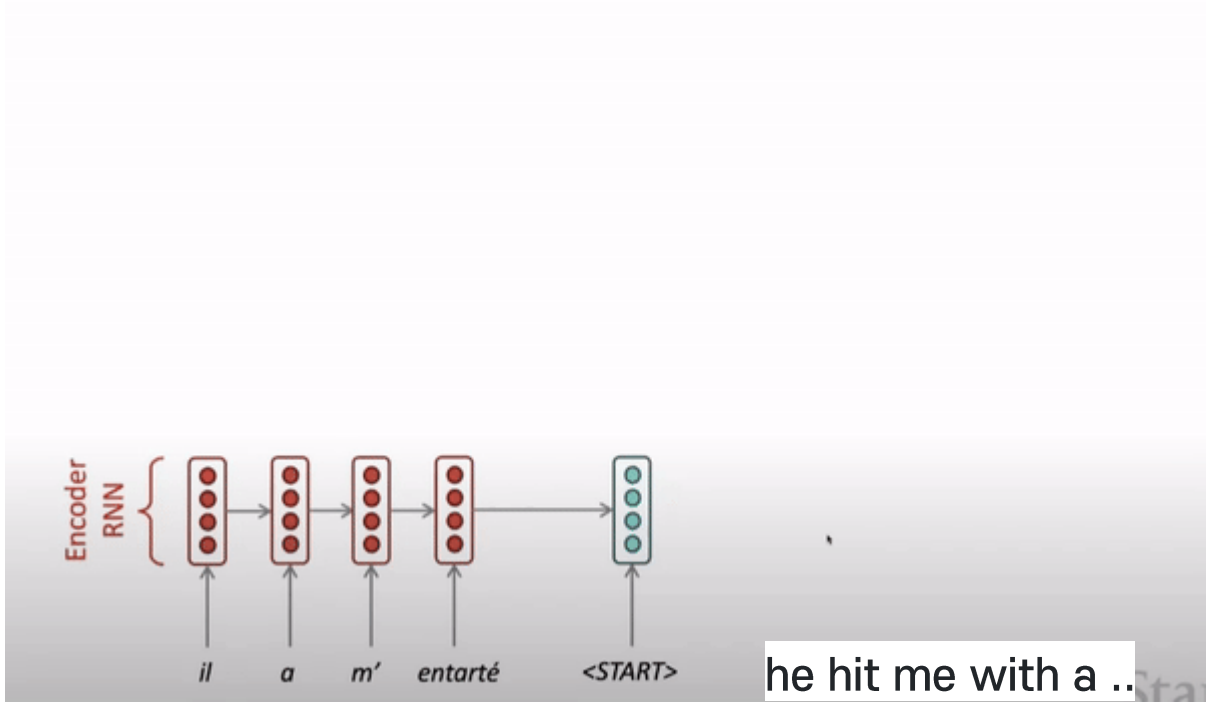
# seq2seq / Attention

## Bottleneck Problem




## Attention

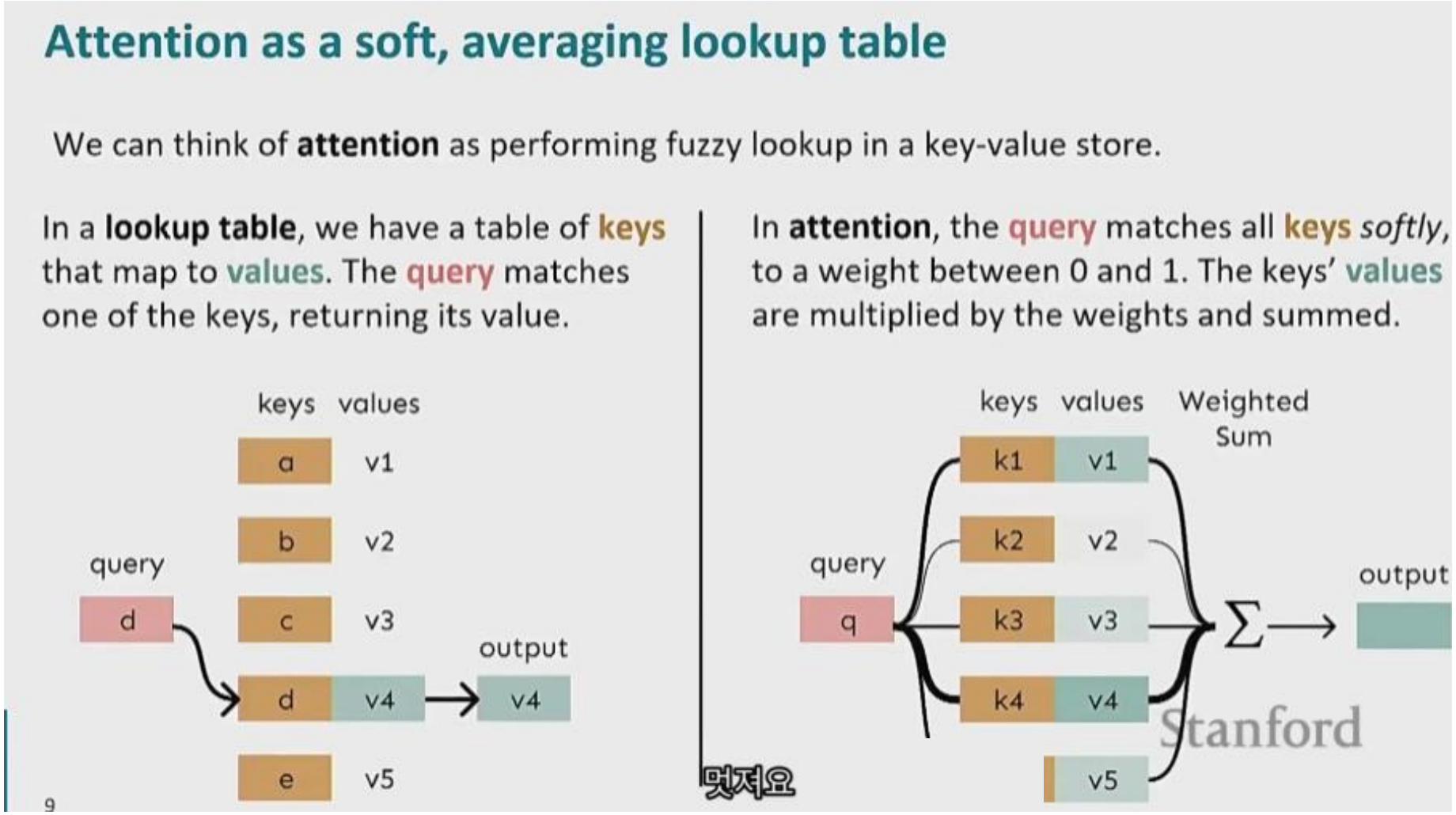
Encoder과 Decoder간의 직접적인 연결 사용



# Lecture 8

## Transformer

# How about Attention?




layer 수를 줄여
연산량을 줄이고

gradient vanishing
문제를 해결하자

# self-attention

## Self-Attention: keys, queries, values from the same sequence

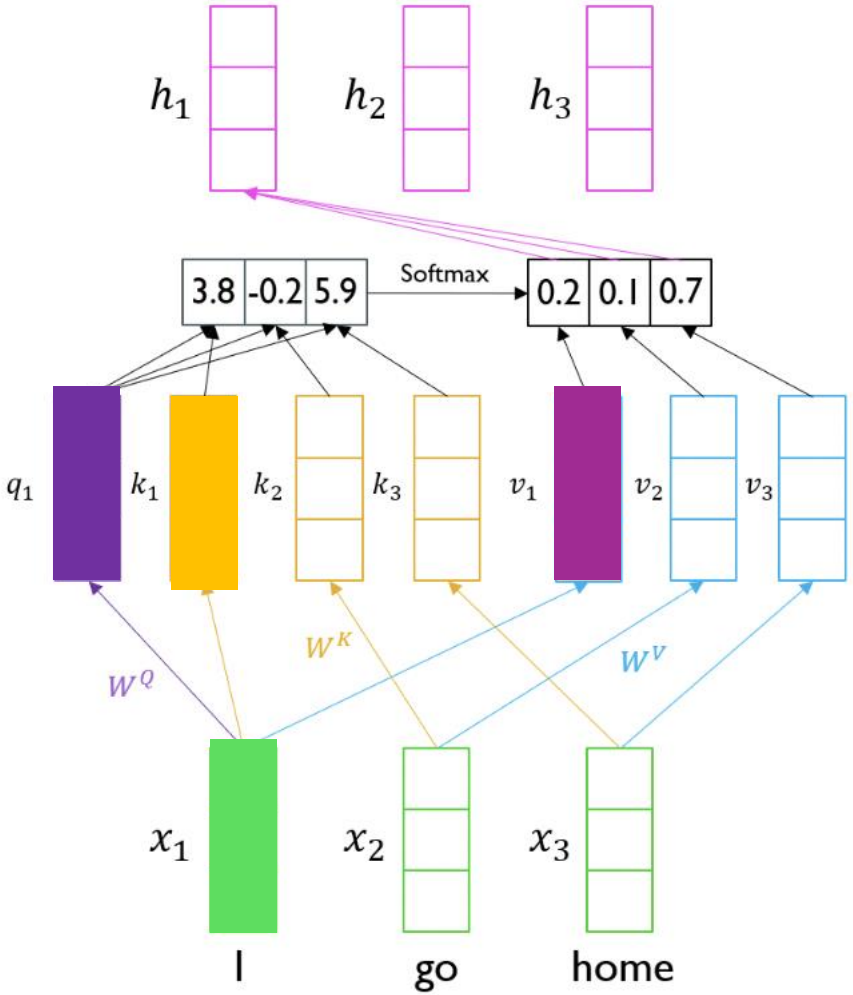
Let $\boldsymbol{w}_{1:n}$ be a sequence of words in vocabulary $V$, like *Zuko made his uncle tea*.

For each $\boldsymbol{w}_i$, let $\boldsymbol{x}_i = E\boldsymbol{w}_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

1. Transform each word embedding with weight matrices Q, K, V, each in $\mathbb{R}^{d \times d}$

$\boldsymbol{q}_i = Q\boldsymbol{x}_i$ (queries) $\quad \boldsymbol{k}_i = K\boldsymbol{x}_i$ (keys) $\quad \boldsymbol{v}_i = V\boldsymbol{x}_i$ (values)

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




Query 단어가 **다른 단어와 어떻게 상호작용**하는지?
Key **정보의 특징**(어떤 query와 연관이 있는지?)
Value **실제 정보(내용)**

# self-Attention vs Attention

s - 디코더 hidden state h - 인코더 hidden state

| | Self-Attention | Attention |
|---|---|---|
| | $v_i = k_i = q_i = x_i$ | |
| Attention score : | $e_{ij} = q_i^T k_j$ | $\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$ |
| Attention weight : | $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$ | $\alpha^t = \mathrm{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$ |
| Attention output : | $\mathrm{output}_i = \sum_j \alpha_{ij} v_j$ | $\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$ |

# Barrier

1. **내재적인 순서 개념이 없음**
   **(Doesn't have an inherent notion of order)**

   모델은 자체적으로 순서를 인식하지 못함
   → `Positional Encoding`

2. **선형성이 없음**
   **(No nonlinearities for deep learning magic! It's all just weighted averages)**

   깊은 학습의 마법을 위한 비선형성이 없고, 단순히 가중 평균만 사용
   → `피드 포워드 신경망`

3. **예측할 때 '미래를 보지 않도록' 해야 함**
   **(Need to ensure we don’t "look at the future" when predicting a sequence)**

   예를 들어 기계 번역이나 언어 모델링에서 미래 정보를 사용하지 않도록 해야
   → `masked multihead attention`

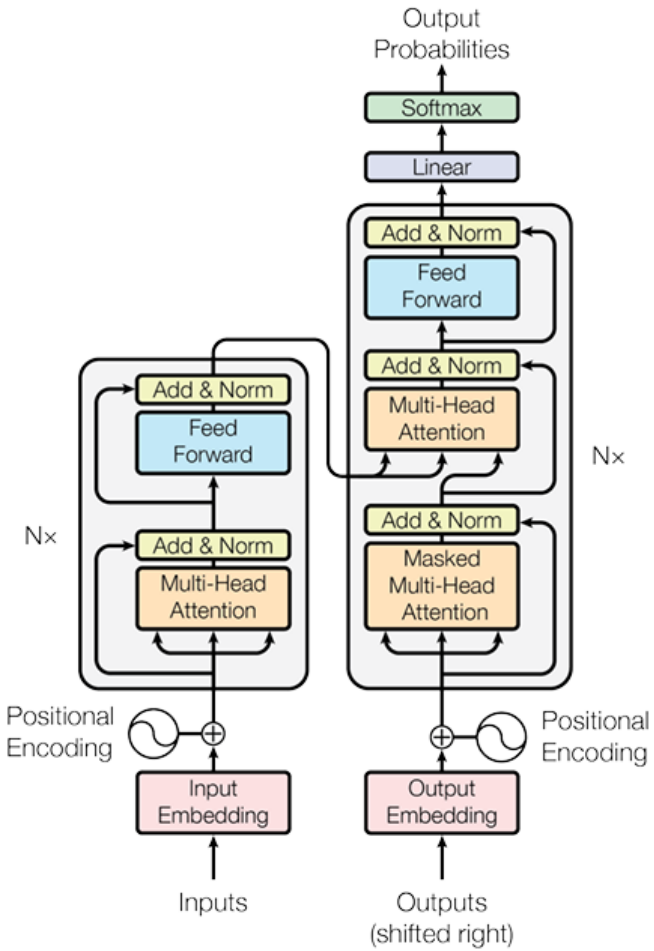



Figure 1: The Transformer - model architecture.

# Positional Encoding

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$\mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

장점)
- 절대적 위치가 아닌, 상대적인 비교가 더 중요
- 함수주기보다 sequence가 길어도 정보를 잃지 않음

단점)
- 학습이 안된다. -〉 concat
- n보다 긴 문장은 사용할 수 없다. -〉 적절히 설정

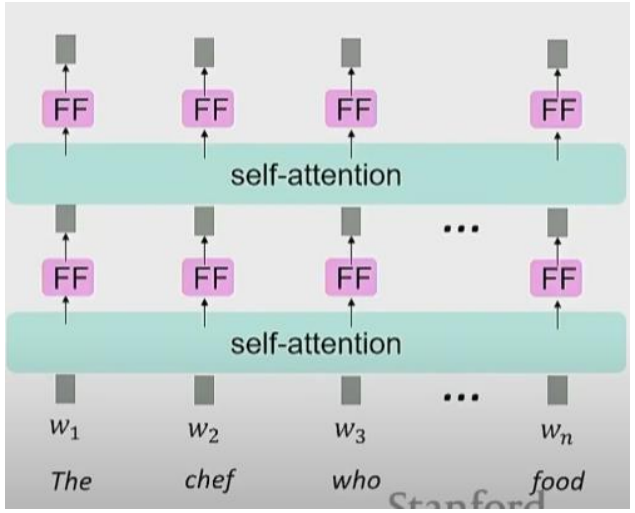
n : 최대 시퀀스 길이

gpt-3같은 경우는 입력 최대 토큰이 2048 이고, 주기함수는 10000 으로 설정되어있다.

# Barrier

1. 내재적인 순서 개념이 없음
   **(Doesn't have an inherent notion of order)**

   모델은 자체적으로 순서를 인식하지 못함
   → `Positional Encoding`

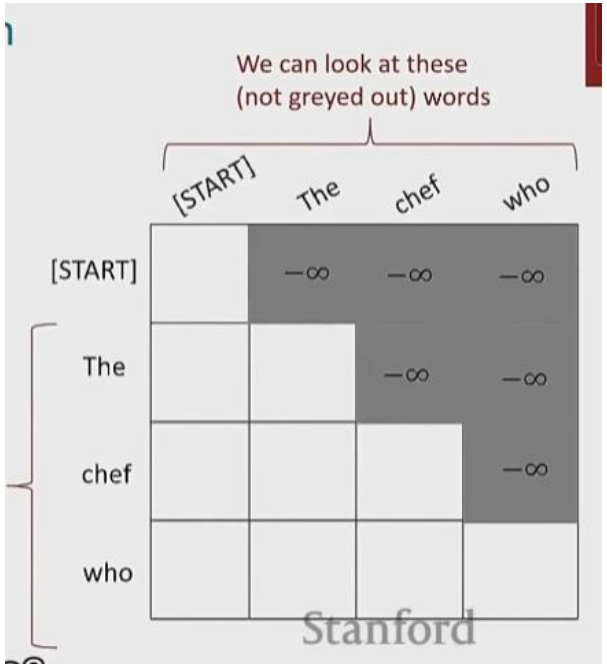



2. 선형성이 없음
   **(No nonlinearities for deep learning magic! It's all just weighted averages)**

   깊은 학습의 마법을 위한 비선형성이 없고, 단순히 가중 평균만 사용
   → `피드 포워드 신경망`

3. 예측할 때 '미래를 보지 않도록' 해야 함
   **(Need to ensure we don't "look at the future" when predicting a sequence)**

   기계 번역이나 언어 모델링에서 미래 정보를 사용하지 않도록 해야
   → `masked multihead attention`

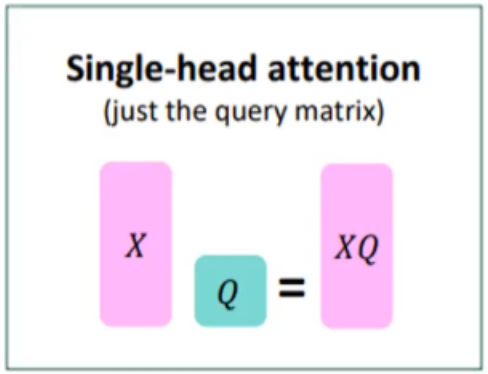


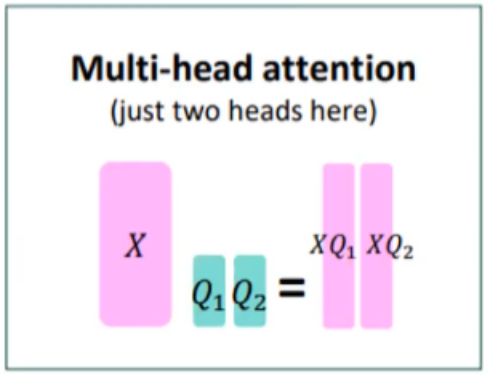
# Multi-Head Attention

- We compute $XQ \in \mathbb{R}^{n \times d}$, and then reshape to $\mathbb{R}^{n \times d/h}$. (Likewise for $XK, XV$.)

d

- Multi-head attention을 수행하여도 결국 input과 같은 크기의 output을 산출한다.
- Multi-head attention을 수행하여도 총 계산 량은 동일하다.




n: sequence 길이
h : head수
d/h: 축소된 dimension

모든 쿼리-키 쌍에 대해 유사도를 계산해야 하므로,
n^2의 시간복잡도가 발생

모든 단어가 한 번의 연산으로 서로 상호작용

| Layer Type | Complexity per Layer | Sequential Operations | Maximum Path Length |
|---|---|---|---|
| Self-Attention | $O(n^2 \cdot d)$ | $O(1)$ | $O(1)$ |
| Recurrent | $O(n \cdot d^2)$ | $O(n)$ | $O(n)$ |
| Convolutional | $O(k \cdot n \cdot d^2)$ | $O(1)$ | $O(log_k(n))$ |
| Self-Attention (restricted) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

병렬 연산 수행가능하므로

# 3-4주차: 기본 구현 (2) + 분야별 기초 이론

- 코딩 목표: 1주차때의 내용을 발전시켜보기
  이론 목표: Vision 인턴이라면 cs231n 뒷 내용 숙지, NLP인턴이라면 cs224n 숙지
- 내용:
  - 각 코드들 모듈화 진행 (예; model, dataset, train, valid(test), main, util)
  - argparser 추가
  - 가중치 값 저장 및 로드
  - loss, accuracy 등에 대한 log 출력 및 시각화
  - data augmentation 진행 (선택)
  - Gradient clipping, scheduler 등 다양한 기법들 활용하여 성능 향상 (목표 정확도: 85%)
  - 분야별 기초 이론 공부를 위해 vision 인턴은 cs231n 뒷부분 까지 전부 수강,
    NLP인턴은 cs224n 수강