17기 정규세션

ToBig's 16기 장준원

NLP Advanced (Transformer)

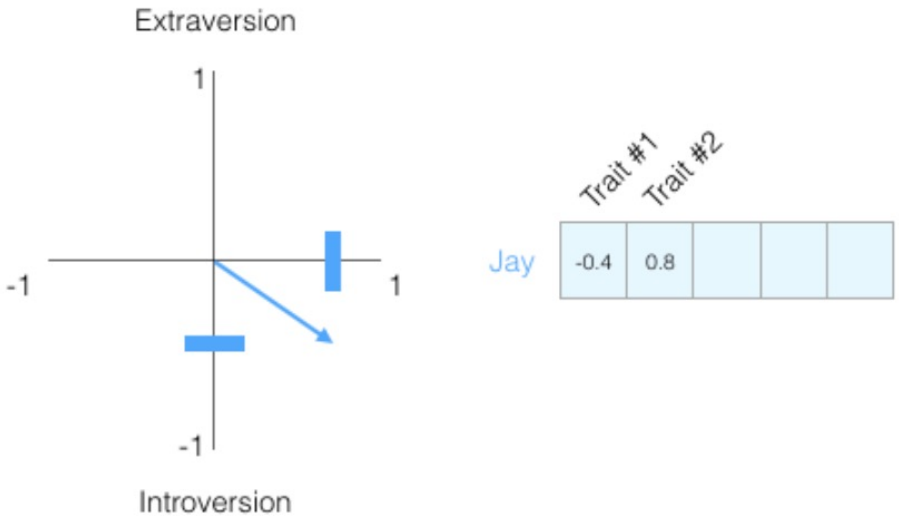
Contents

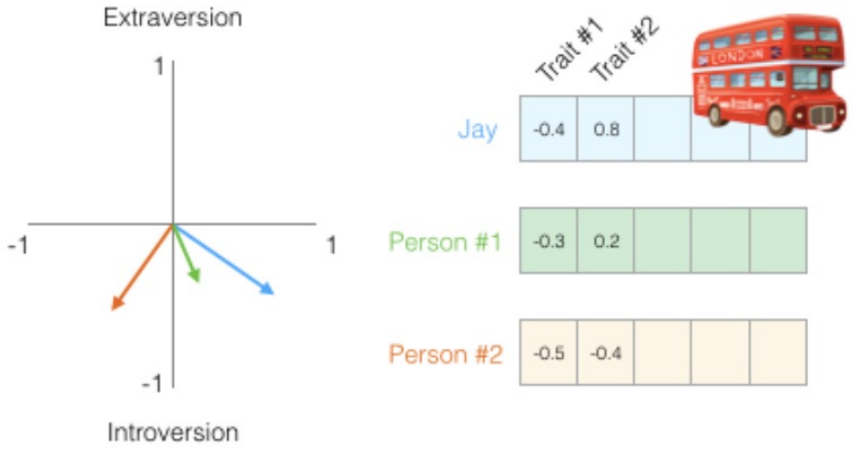
Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Recap : Mapping word (token) to vector

We can represent word as vectors of numbers!

단어를 Vector Space로 투영하는 순간,
'특정 Axis(or basis)가 단어가 내포하는 의미론적인 특징을 담을 수 있다'고 기대할 수 있음!

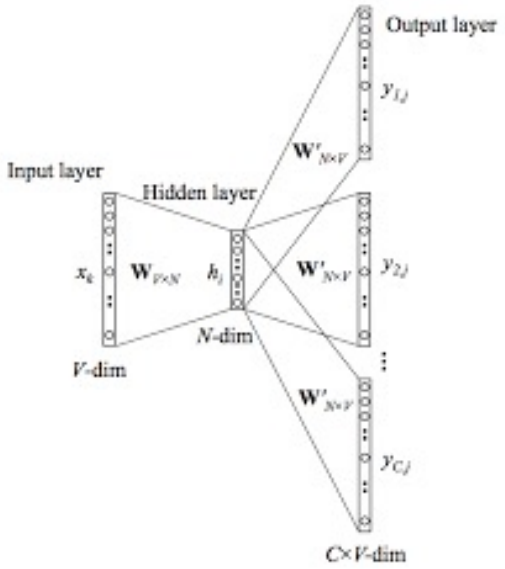




Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Recap : Word Embedding

Word Embedding은 결국 자연어의 특성을 가장 잘 표현할 수 있는 Vector Space를 학습한 것이라고 볼 수 있다!



Word2Vec

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

- $k = solid$일 경우, steam 보다는 ice과 관련이 높을 것임으로, $\frac{P(solid|ice)}{P(solid|steam)}$는 클 것이다
- $k = gas$일 경우, ice 보다는 steam과 관련이 높을 것임으로, $\frac{P(gas|ice)}{P(gas|steam)}$는 작을 것이다
- $k = water$일 경우, steam과 ice 둘다 관련이 높을 것임으로, $\frac{P(water|ice)}{P(water|steam)}$는 1에 가까울 것이다
- $k = fashion$일 경우, steam과 ice과 둘다 관련이 낮을 것임으로, $\frac{P(fashion|ice)}{P(fashion|steam)}$는 1에 가까울 것이다

▶ 확률(P_{ik})보다는 "확률의 비(P_{ik}/P_{jk})" 를 활용하면,
k와 i,j의 관계(단어 상호 간의 비율 정보)를 좀 더 정확하게 임베딩 할 수 있지 않을까?

Glove

Fasttext

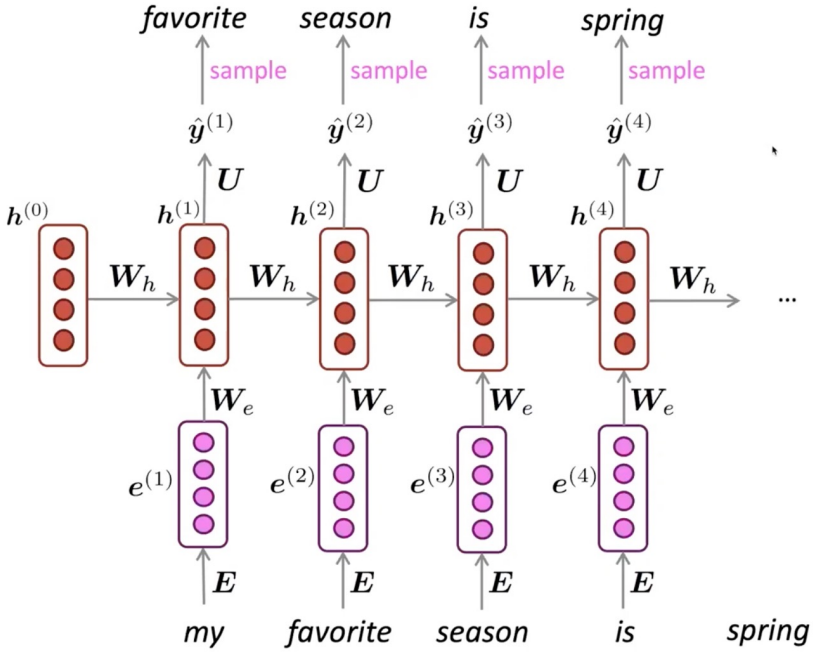
Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Recap : RNN Structure (Language Modeling with RNN)

인간이 글을 쓰는 과정을 생각해보자!
RNN 구조는 모델 본질적으로 인간이 글을 쓰는 구조를 반영할 수 있다!
Recurrent networks carry the **inductive biases of temporal invariance and locality via their Markovian structure**.

Inductive Bias

학습 시에는 만나보지 않았던 상황에 대하여 정확한 예측을 하기 위해 사용하는 추가적인 가정



Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

RNN Forward Pass

. The Simple RNN Language Model

output distribution

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

hidden states

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

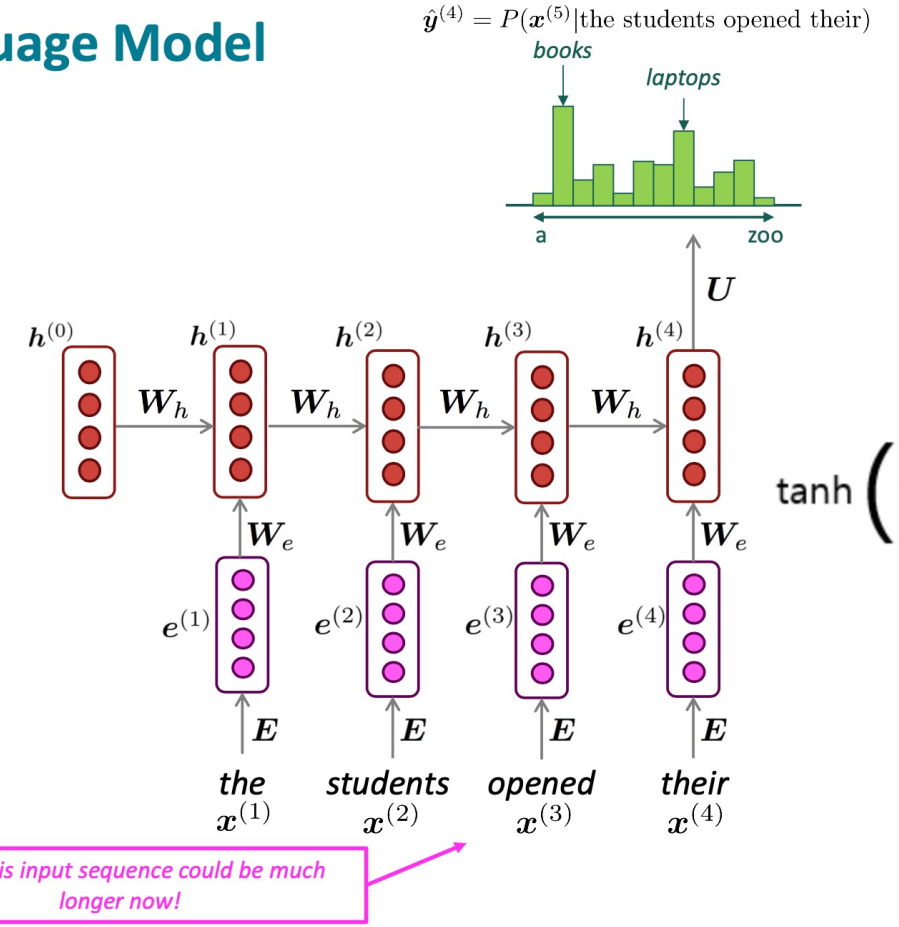
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

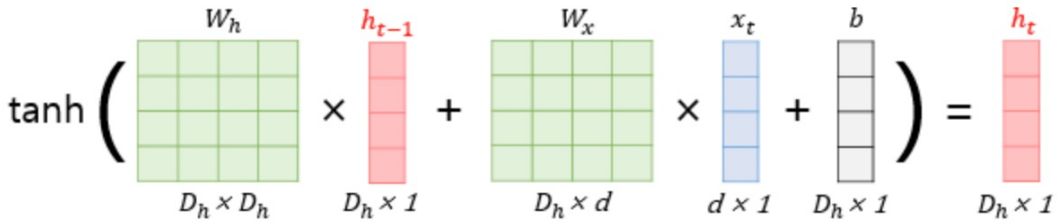
$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

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



***Note**: this input sequence could be much longer now!*



Hidden state at time *t*

Back Propagation of RNN : Vanishing Gradient Problem

Multivariate Chain Rule

Example about how to use chain rule to calculate total derivative of function *f* with respect to variable *t* through intermediate variables *x*, *y* and *z* as follows.

$$
\begin{aligned}
f &= \sin(x)e^{yz^2} \\
x &= t - 1 \\
y &= t^2 \\
z &= \frac{1}{t} \\
\frac{df}{dt} &= \frac{\partial f}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial f}{\partial y} \cdot \frac{dy}{dt} + \frac{\partial f}{\partial z} \cdot \frac{dz}{dt} \\
&= \cos(x)e^{yz^2} \cdot (1) + z^2 \sin(x)e^{yz^2} \cdot (2t) + 2y \sin(x)e^{yz^2} \cdot (-\frac{1}{t^2}) \\
&= e^{yz^2}\left[\cos(x) + 2tz^2 \sin(x) - \frac{2yz}{t^2}\sin(x)\right]
\end{aligned}
$$

Back Propagation of RNN : Vanishing Gradient Problem

Multivariate Chain Rule

Now let's formalize the definition of chain rule in multivariate calculus.

We start with a function *f* with *n* input variables $f(x_1, x_2, \cdots, x_n) = f(\mathbf{x})$

If each of the input variable in *X* is also a function of another variable t, our function *f* can re-expressed by follows.

$$f(\mathbf{x}(t)) = f(x_1(t), x_2(t), \cdots, x_n(t))$$

To compute the derivative of function *f* with respect to *t*, we can write a dot product of two n-dimensional vectors.

$$\frac{df}{dt} = \frac{\partial f}{\partial x_1} \cdot \frac{dx_1}{dt} + \frac{\partial f}{\partial x_2} \cdot \frac{dx_2}{dt} + \cdots + \frac{\partial f}{\partial x_n} \cdot \frac{dx_n}{dt}$$

$$= \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{pmatrix} \cdot \begin{pmatrix} \frac{dx_1}{dt} \\ \frac{dx_2}{dt} \\ \vdots \\ \frac{dx_n}{dt} \end{pmatrix} \longrightarrow \frac{df}{dt} = (J_f) \cdot \frac{d\mathbf{x}}{dt}$$

First vector is just the partial derivatives of function f with respect to all its input variables
= Jacobian vector

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Back Propagation of RNN : Vanishing Gradient Problem

Vanishing Gradient Intuition

Recap : $h_t = f(Wh_{t-1} + W_{xh}x_t + b_h),\ where\ f\ is\ nonlinear\ function$

Recurrent neural networks propagate weight matrices from one time step to the next. Recall the goal of a RNN implementation is to enable propagating context information through faraway time-steps.

Sentence 1

"Jane walked into the room. John walked in too. Jane said hi to ___"

Sentence 2

"Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to ___"

During the back-propagation phase, the contribution of gradient values gradually vanishes as they propagate to earlier timesteps.

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Back Propagation of RNN : Vanishing Gradient Problem

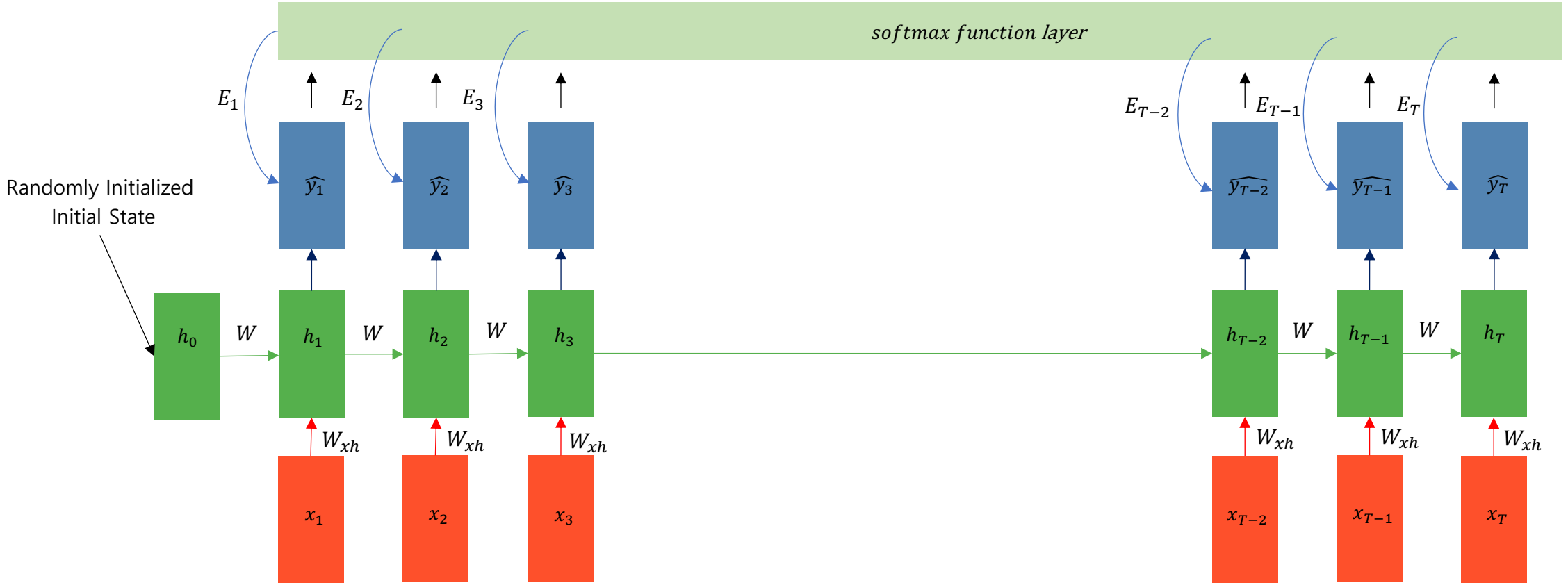
Gradient of W respect to Error

$$\frac{\partial E}{\partial W} = \sum_{i=1}^{T} \frac{\partial E_i}{\partial W}$$

$$\frac{\partial E}{\partial W} = \frac{\partial E_1}{\partial W} + \frac{\partial E_2}{\partial W} + \frac{\partial E_3}{\partial W} + \ \ldots + \frac{\partial E_{T-2}}{\partial W} + \frac{\partial E_{T-1}}{\partial W} + \frac{\partial E_T}{\partial W}$$

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Back Propagation of RNN : Vanishing Gradient Problem : Structure and Backpropagation of RNN



$$h_t = f(Wh_{t-1} + W_{xh}x_t + b_h), \qquad where\ f\ is\ nonlinear\ function$$

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Back Propagation of RNN : Vanishing Gradient Problem

Gradient of W respect to Error at each time-step

The Error for each time step is computed through applying the chain rule. $\frac{\partial h_t}{\partial h_k}$ is derivative of h_t respect to all previous k time steps

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \widehat{y_t}} \frac{\partial \widehat{y_t}}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$\frac{\partial h_t}{\partial h_k}$ is simply a chain rule differentiation over all hidden layers within the [k, t] time interval

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T diag[f'(j_{j-1})]$$

$$h_j = f(W h_{j-1} + W_{xh} x_j + b_h)$$

$$j_{j-1} = W h_{j-1} + W_{xh} x_j + b_h$$

$diag[f'(j_{j-1})]$: nonlinear function applied element-wise

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Back Propagation of RNN : Vanishing Gradient Problem

Each $\frac{\partial h_j}{\partial h_{j-1}}$ is the Jacobian Matrix for h, where all Hidden State have dimension of D_n

$$\frac{\partial h_j}{\partial h_{j-1}} = \left[\frac{\partial h_j}{\partial h_{j-1,1}} \frac{\partial h_j}{\partial h_{j-1,2}} \dots \frac{\partial h_j}{\partial h_{j-1,D_n}}\right] = \begin{bmatrix} \frac{\partial h_{j,1}}{\partial h_{j-1,1}} & \cdots & \frac{\partial h_{j,1}}{\partial h_{j-1,D_n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_{j,D_n}}{\partial h_{j-1,1}} & \cdots & \frac{\partial h_{j,D_n}}{\partial h_{j-1,D_n}} \end{bmatrix}, \textit{same dimension as weight matrix}$$

$$h_j = f\left(W h_{j-1} + W_{xh} x_j + b_h\right)$$

Recap : $\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T diag[f'(j_{j-1})]$

$$\begin{bmatrix} h_{j,1} \\ \vdots \\ h_{j,D_n} \end{bmatrix} = \begin{bmatrix} f_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & f_{D_n} \end{bmatrix} \left(\begin{bmatrix} W_{1,1} & \cdots & W_{1,D_n} \\ \vdots & \ddots & \vdots \\ W_{D_n,1} & \cdots & W_{D_n,D_n} \end{bmatrix} \begin{bmatrix} h_{j-1,1} \\ \vdots \\ h_{j-1,D_n} \end{bmatrix} + \begin{bmatrix} W_{1,1} & \cdots & W_{1,D_x} \\ \vdots & \ddots & \vdots \\ W_{D_n,1} & \cdots & W_{D_n,D_x} \end{bmatrix} \begin{bmatrix} x_{j,1} \\ \vdots \\ x_{j,D_x} \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_{D_n} \end{bmatrix} \right)$$

$, where\ hidden\ state\ has\ dimension\ of\ D_n,\ \ input\ vector\ has\ dimension\ of\ D_x$

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Back Propagation of RNN : Vanishing Gradient Problem

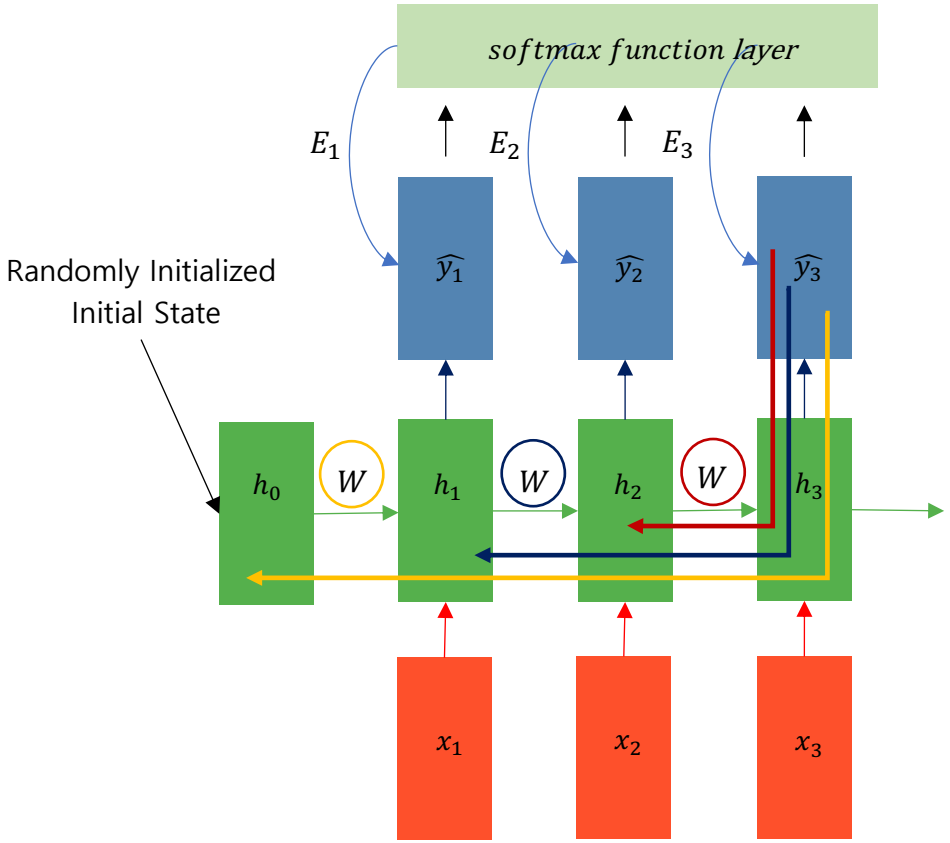
Gradient of W respect to Error

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T}\sum_{k=1}^{t}\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\left(\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial h_k}{\partial W}$$

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Back Propagation of RNN : Vanishing Gradient Problem

Gradient of W respect to Error at time-step 03. (t=3)



$E_3, \widehat{y_3}$를 구하는데 W는 3번 활용된다.

$$\frac{\partial E_3}{\partial W} = \sum_{k=1}^{3} \frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W} \quad \left(\text{Recap} : \frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \widehat{y_t}} \frac{\partial \widehat{y_t}}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}\right)$$

$$\text{k= } 2, \prod_{j=k+1}^{3} \frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial h_3}{\partial h_2} \quad k = 1, \prod_{j=k+1}^{3} \frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

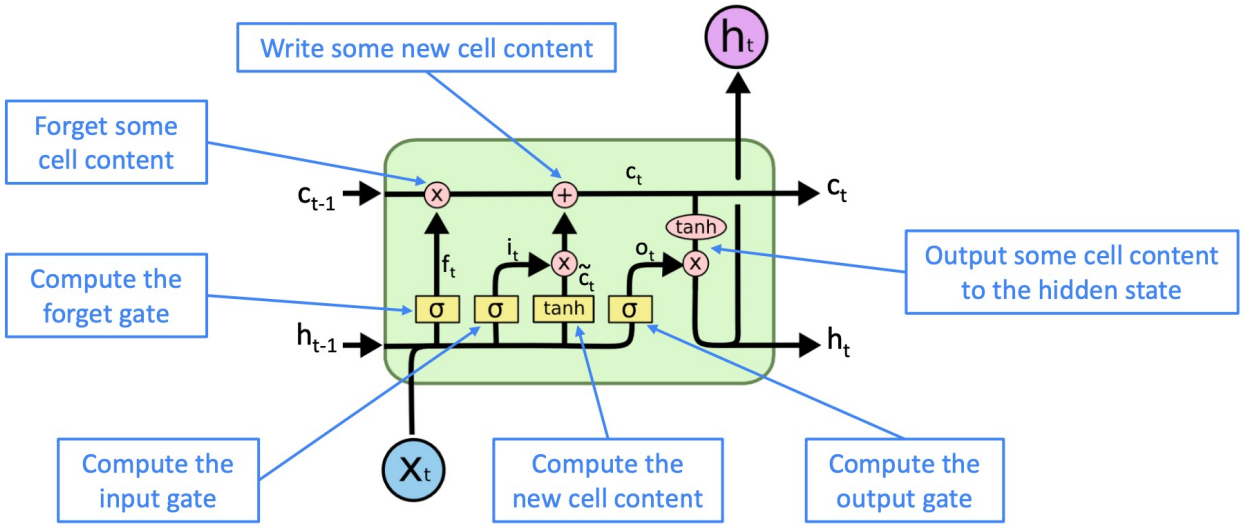
$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3} \frac{\partial h_3}{\partial h_3} \frac{\partial h_3}{\partial W} + \frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W}$$

$$\frac{\partial h_j}{\partial h_{j-1}} = \begin{bmatrix} \frac{\partial h_{j,1}}{\partial h_{j-1,1}} & \cdots & \frac{\partial h_{j,1}}{\partial h_{j-1,D_n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_{j,D_n}}{\partial h_{j-1,1}} & \cdots & \frac{\partial h_{j,D_n}}{\partial h_{j-1,D_n}} \end{bmatrix}$$

앞에 있는 hidden state일수록 더 많이 곱해진다.
= Vanishing Gradient / Exploding Gradient 확률 ↑
= 앞에 있는 정보를 덜 반영한다.

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Solution to Vanishing Gradient Problem : Fancy RNN



LSTM

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

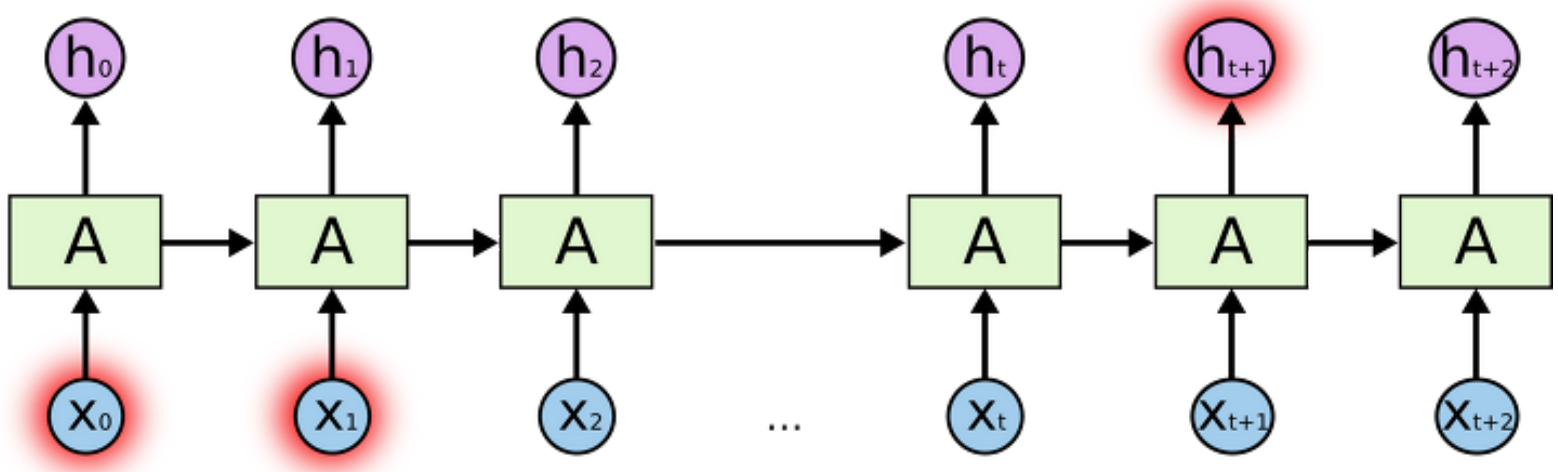
GRU

Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Disadvantages of Recurrent Structure

1. **Computation is slow - because it is sequential, it cannot be parallelized**

2. **In practice, it is difficult to access information from many steps back due to problems like vanishing and exploding gradients, which we discuss in the following subsection**

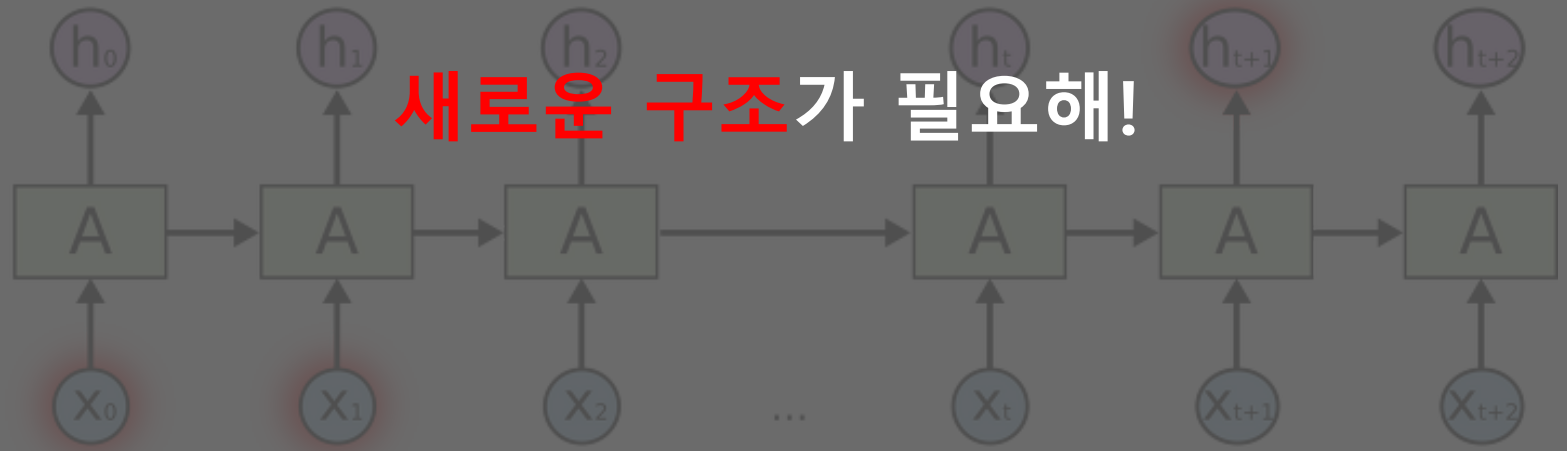


Unit 01 | Vanishing Gradient Problem in RNN (Long-Term Dependency)

Disadvantages of Recurrent Structure

1. Computation is slow - because it is sequential, it cannot be parallelized
2. In practice, it is difficult to access information from many steps back due to problems like vanishing and exploding gradients, which we discuss in the following subsection

Gate를 추가해도 결국 RNN의 근본적인 한계를 극복하지 못한다.

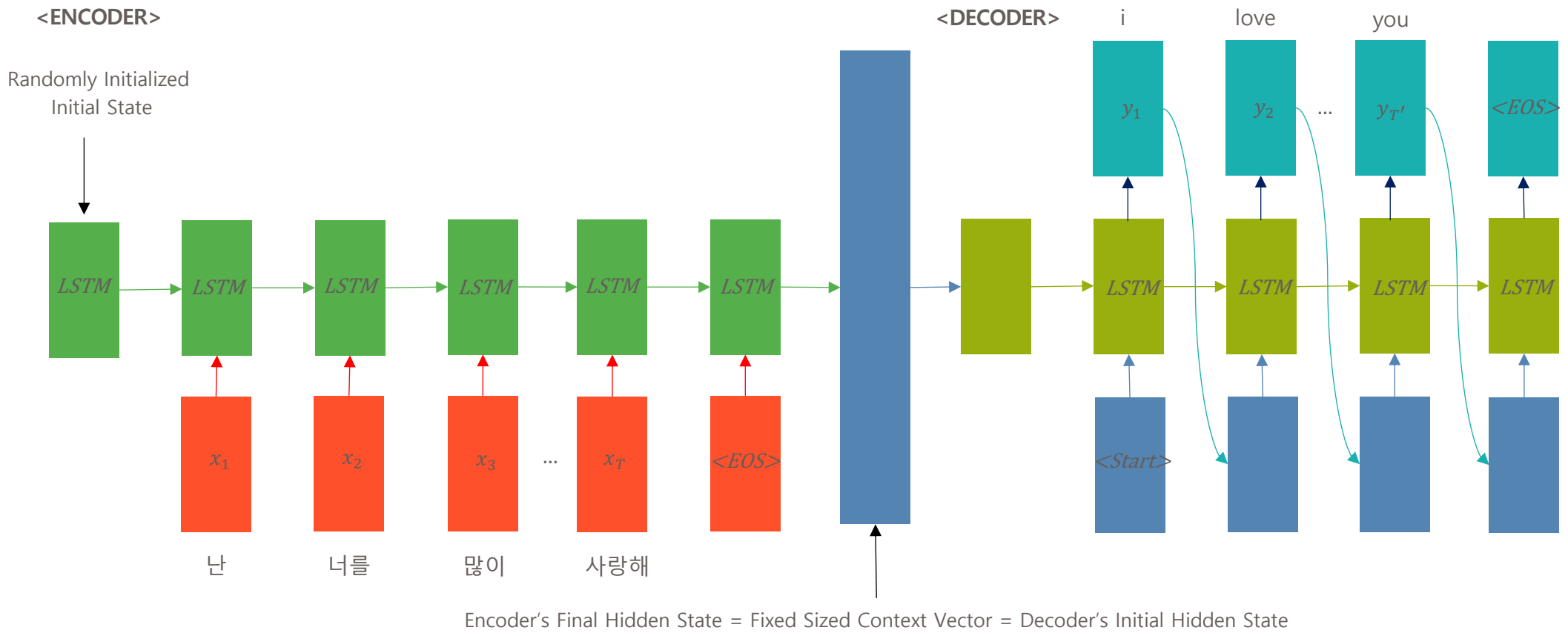


새로운 구조가 필요해!

Unit 02 | Seq2Seq & Alignment & Attention

Seq2Seq & Alignment

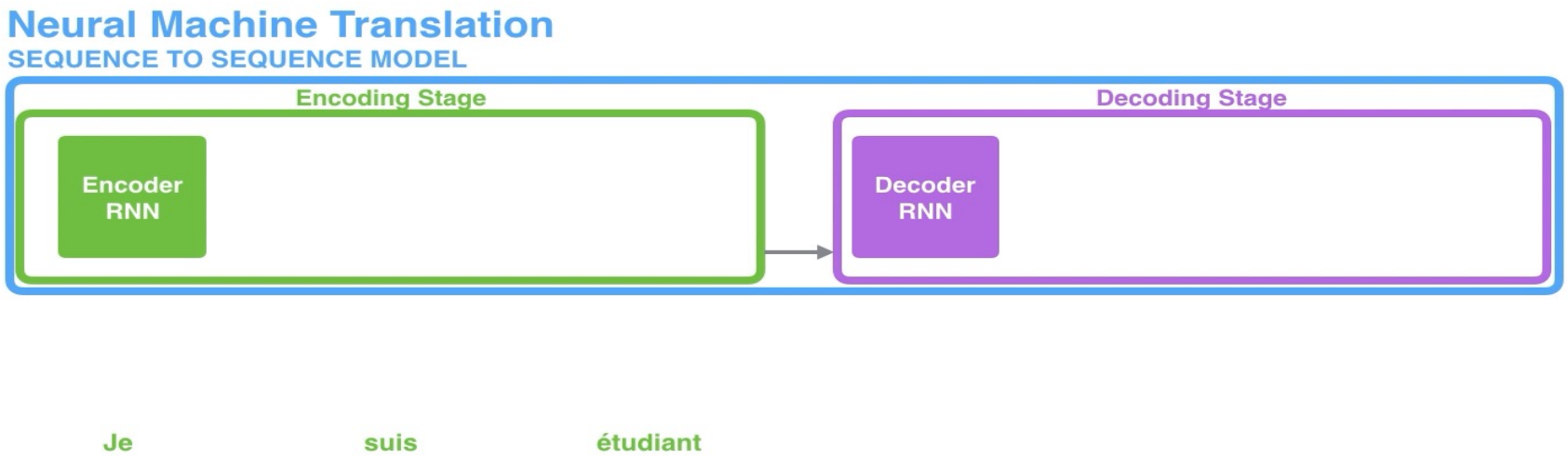
Structure of Seq2Seq (하나의 연속된 순서의 단어를 다른 연속된 순서의 단어로 변환하는 모델)



Unit 02 | Seq2Seq & Alignment & Attention

Seq2Seq & Alignment

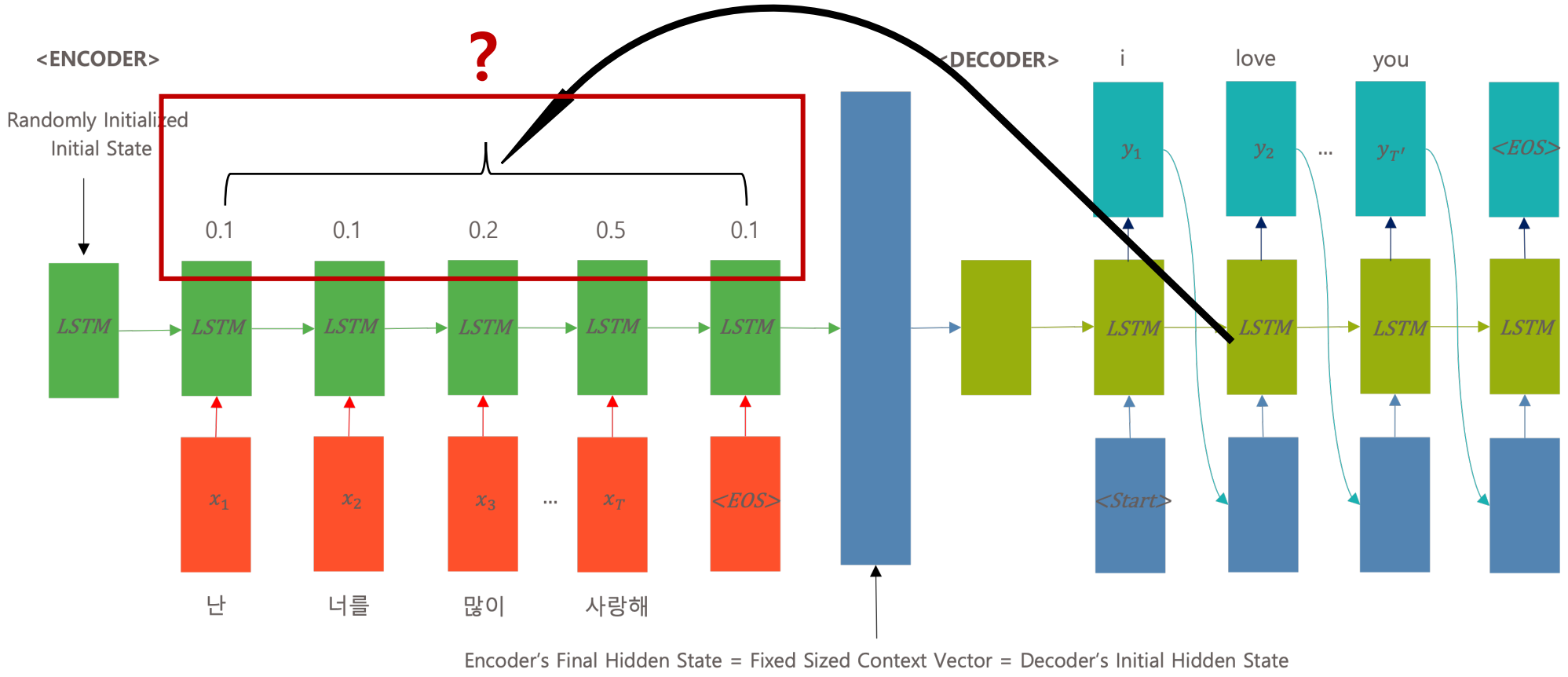
Seq2Seq Intuition



Unit 02 | Seq2Seq & Alignment & Attention

Seq2Seq & Alignment

Love라는 단어를 만들때, '사랑해'라는 단어에 더 많은 영향을 받게 하면 안될까?

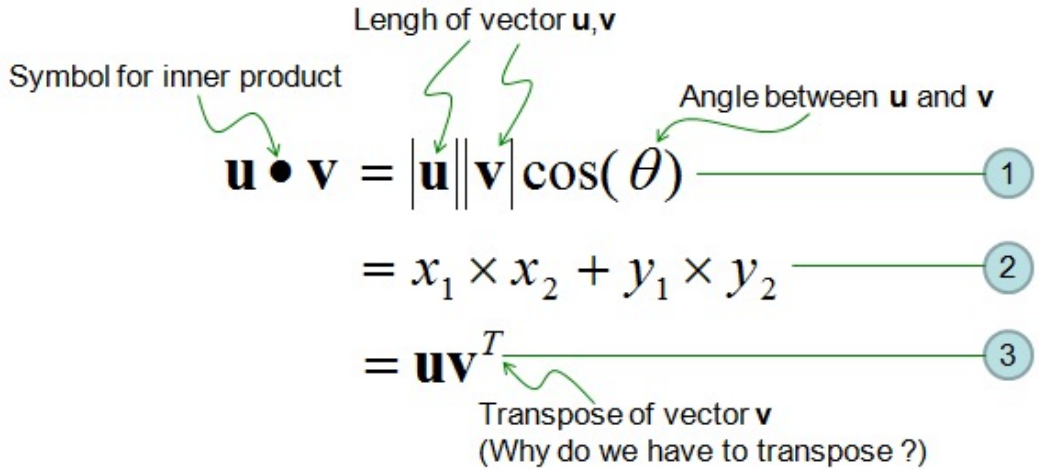


Attention (Dot-Product Attention)

Word(=Token) > Embedding = Vector > Hidden State = Vector

Similarity Measure in Vector Space?

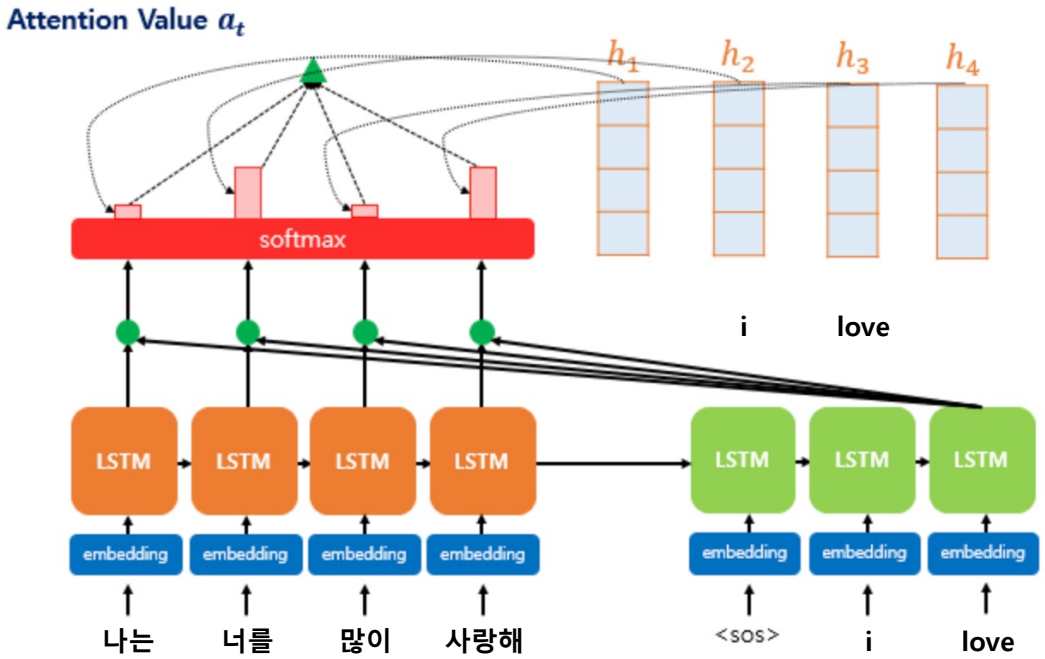
Dot Product (내적)
Cosine Similarity



Unit 02 | Seq2Seq & Alignment & Attention

Attention

How attention work? : Weighted Sum

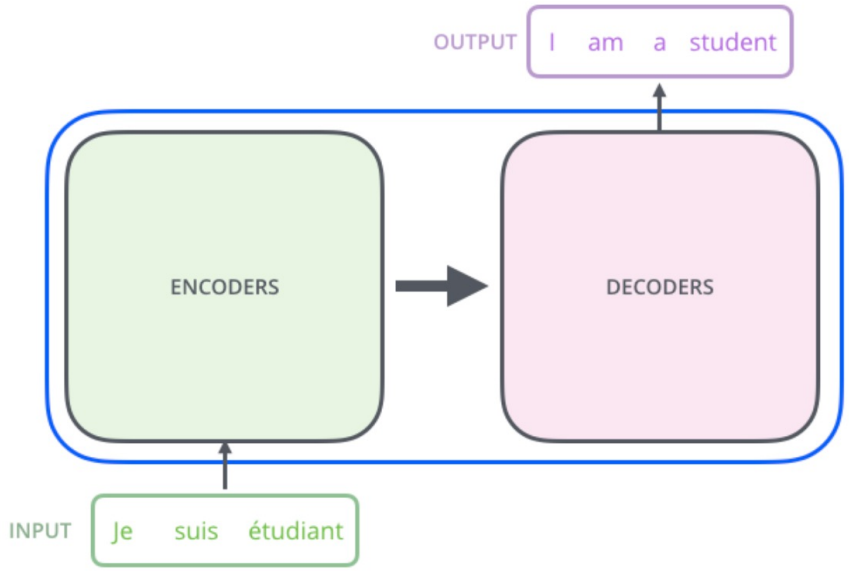


$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i$$

Unit 03 | Transformer - Overview

Transformer is Machine Translation Model (Seq2Seq Model)

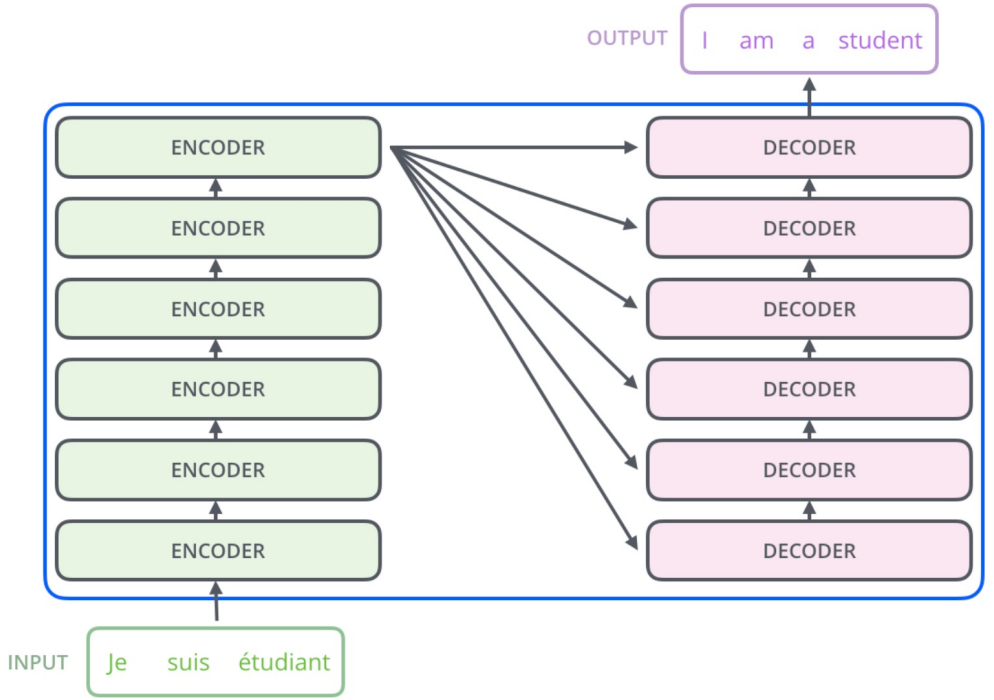
Transformer는
A언어로 이루어진 문장에 대한 표현을 학습하는 Encoder와
Encoder가 학습한 A언어를 바탕으로 B언어로 된 번역문을 생성하는 Decoder로 이루어져있다.



Unit 03 | Transformer - Overview

Transformer Structure

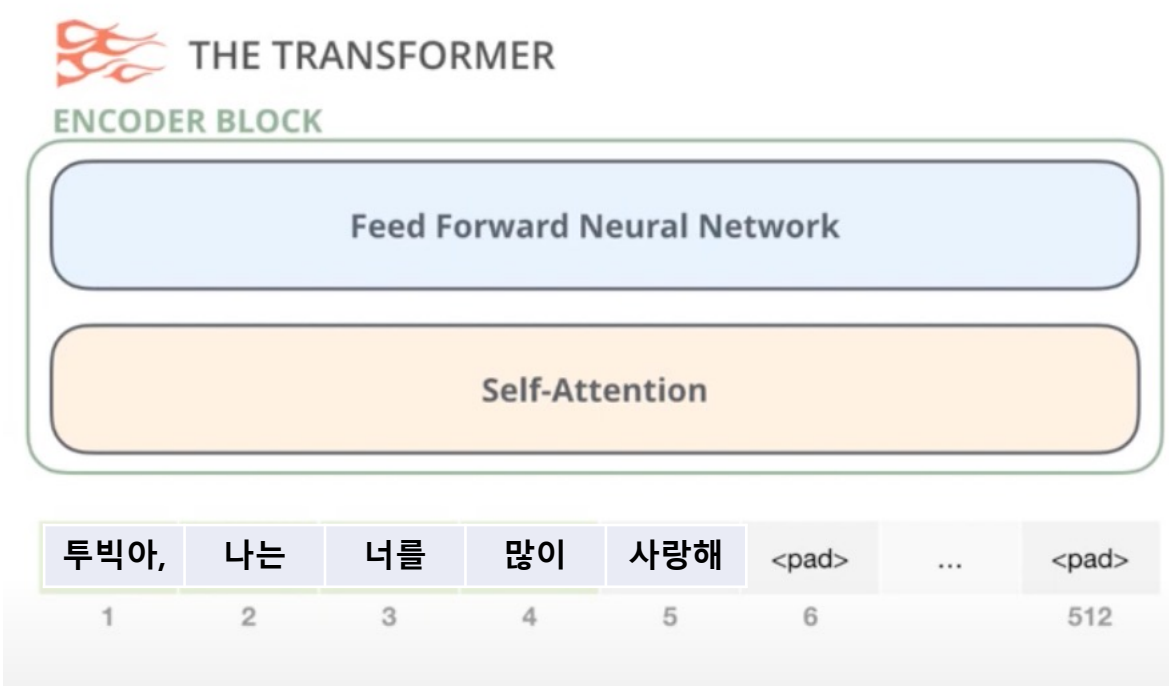
Encoder is composed of a stack of N=6 identical layers (each layer has two sub-layers)
Decoder is composed of a stack of N=6 identical layers (each layer has three sub-layers)

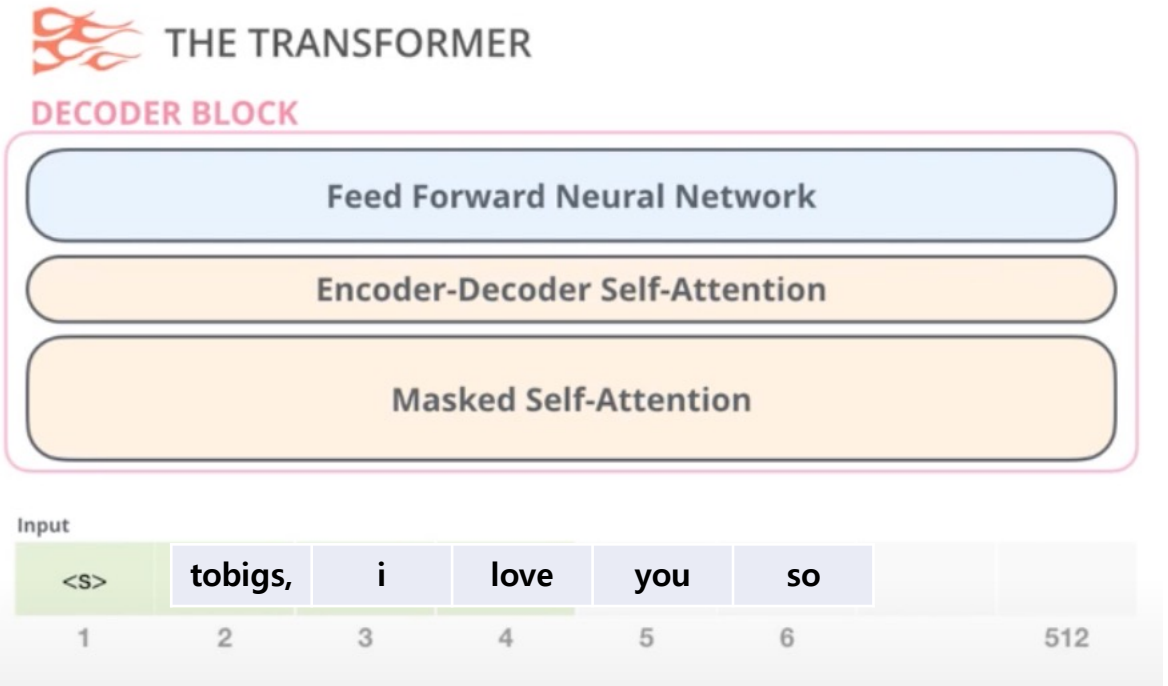


Unit 03 | Transformer - Overview

Transformer Structure

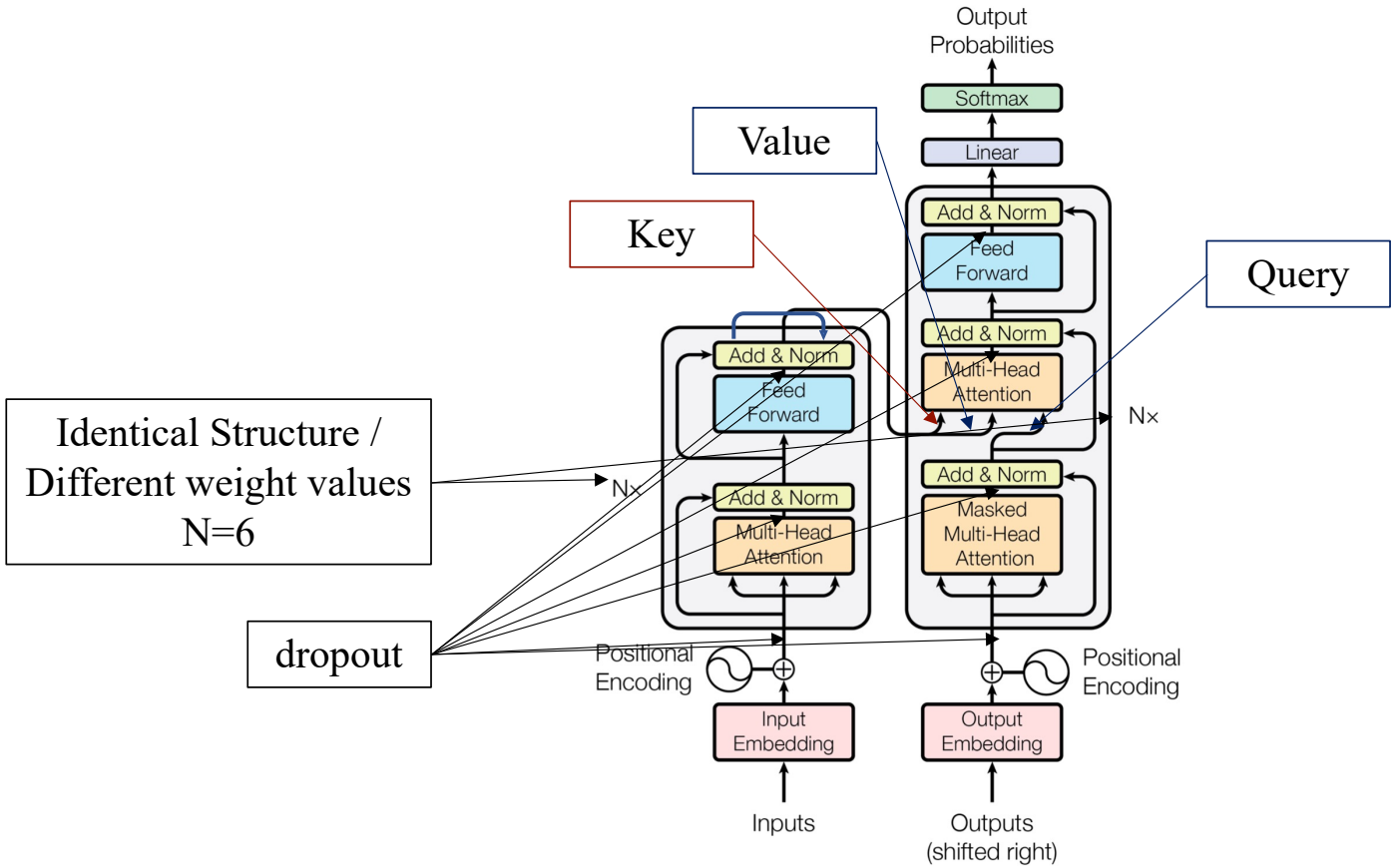
Encoder is based on Unmasked Multi-head Self-Attention (원문은 양방향 정보를 다 반영하자!)
Decoder is based on Masked Multi-head Self-Attention (번역문을 생성할 때는 이전 정보만 반영하자!)





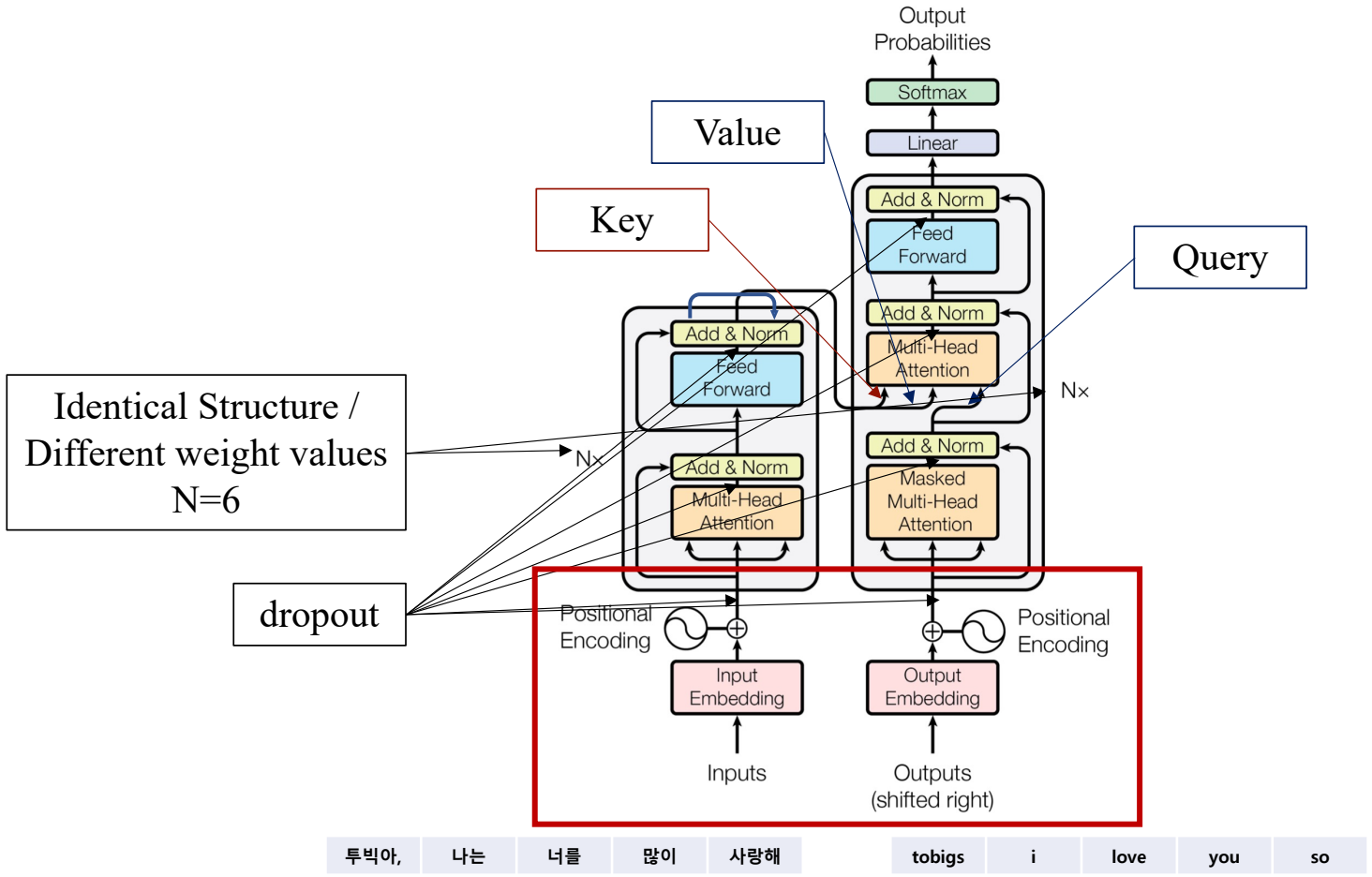
Unit 03 | Transformer - Overview

Transformer Structure



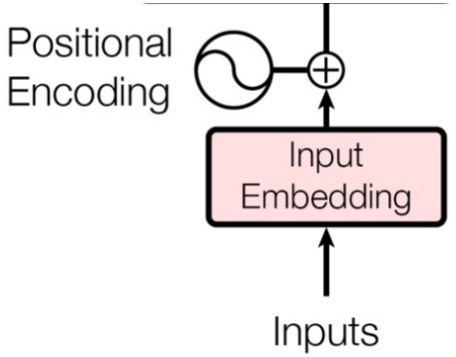
Unit 03 | Transformer - Overview

Word Embedding Layer & Positional Embedding



Unit 03 | Transformer - Overview

Word Embedding Layer & Positional Embedding

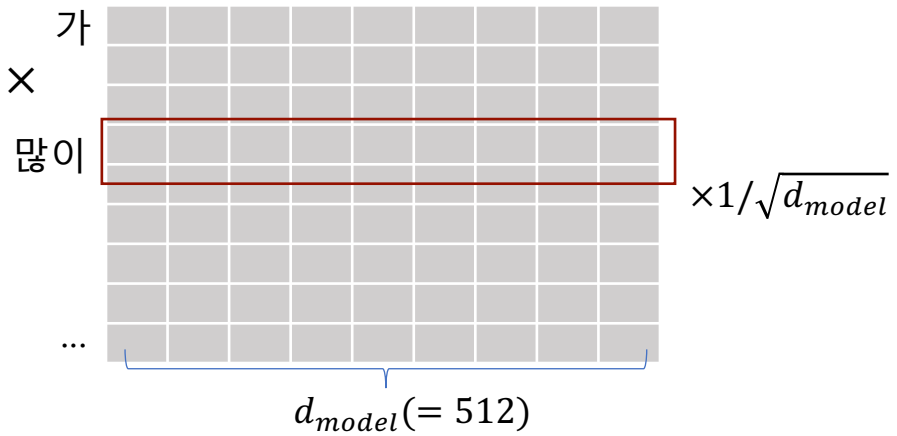


Input Embedding에서 원문/번역문의 각 단어의 Word Embedding을 가져온다.

- 기존에 훈련시킨 Word2Vec같은 Embedding
- Random Initialize해서 훈련

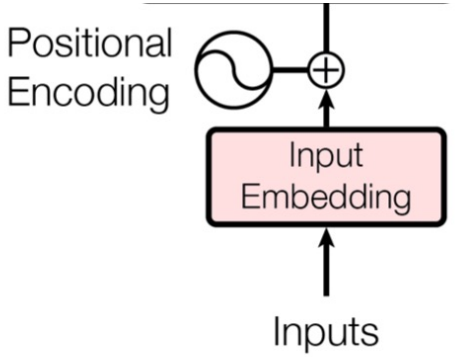
In embedding Layer, we multiply those weights by $\sqrt{d_{model}}$.

투빅아,	나는	너를	많이	사랑해

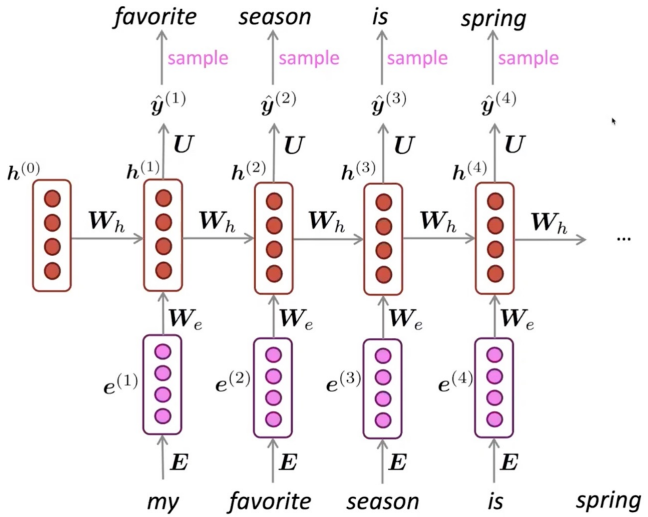


Unit 03 | Transformer - Overview

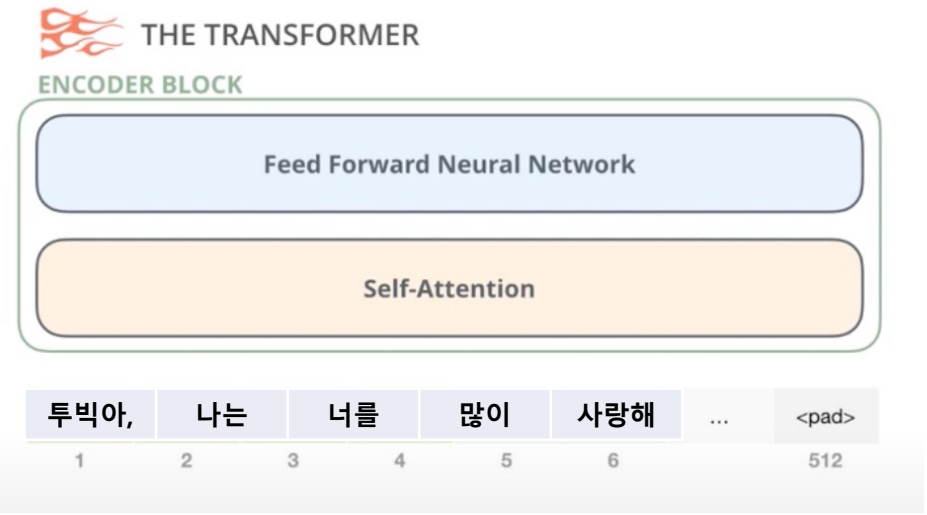
Why Positional Embedding?



Since transformer contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject information about the relative or absolute position of the tokens in the sequence



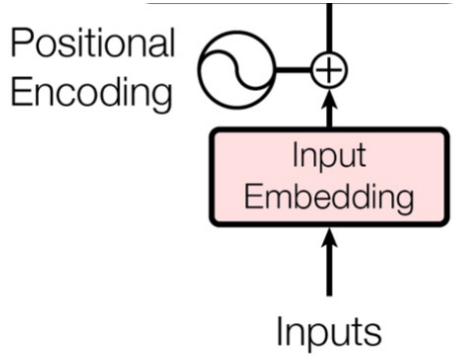
RNN



Transformer Encoder

Unit 03 | Transformer - Overview

Word Embedding Layer & Positional Embedding



The **norm of encoding vector is the same for all positions**
The **further the two positions, the larger the distance**

1. Sinusoidal Function

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

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

pos : position
i : dimension

- **Distance between two vector (seq_len=10, dim=10)**

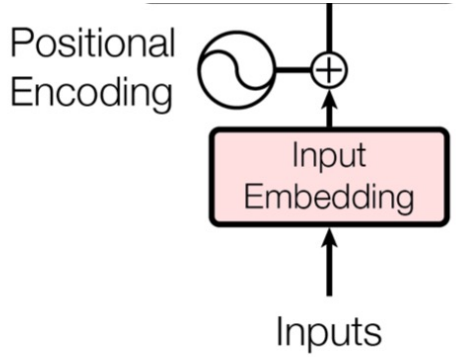
	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
X1	0.000	1.275	2.167	2.823	3.361	3.508	3.392	3.440	3.417	3.266
X2	1.275	0.000	1.104	2.195	3.135	3.511	3.452	3.442	3.387	3.308
X3	2.167	1.104	0.000	1.296	2.468	3.067	3.256	3.464	3.498	3.371
X4	2.823	2.195	1.296	0.000	1.275	2.110	2.746	3.399	3.624	3.399
X5	3.361	3.135	2.468	1.275	0.000	1.057	2.176	3.242	3.659	3.434
X6	3.508	3.511	3.067	2.110	1.057	0.000	1.333	2.601	3.169	3.118
X7	3.392	3.452	3.256	2.746	2.176	1.333	0.000	1.338	2.063	2.429
X8	3.440	3.442	3.464	3.399	3.242	2.601	1.338	0.000	0.912	1.891
X9	3.417	3.387	3.498	3.624	3.659	3.169	2.063	0.912	0.000	1.277
X10	3.266	3.308	3.371	3.399	3.434	3.118	2.429	1.891	1.277	0.000

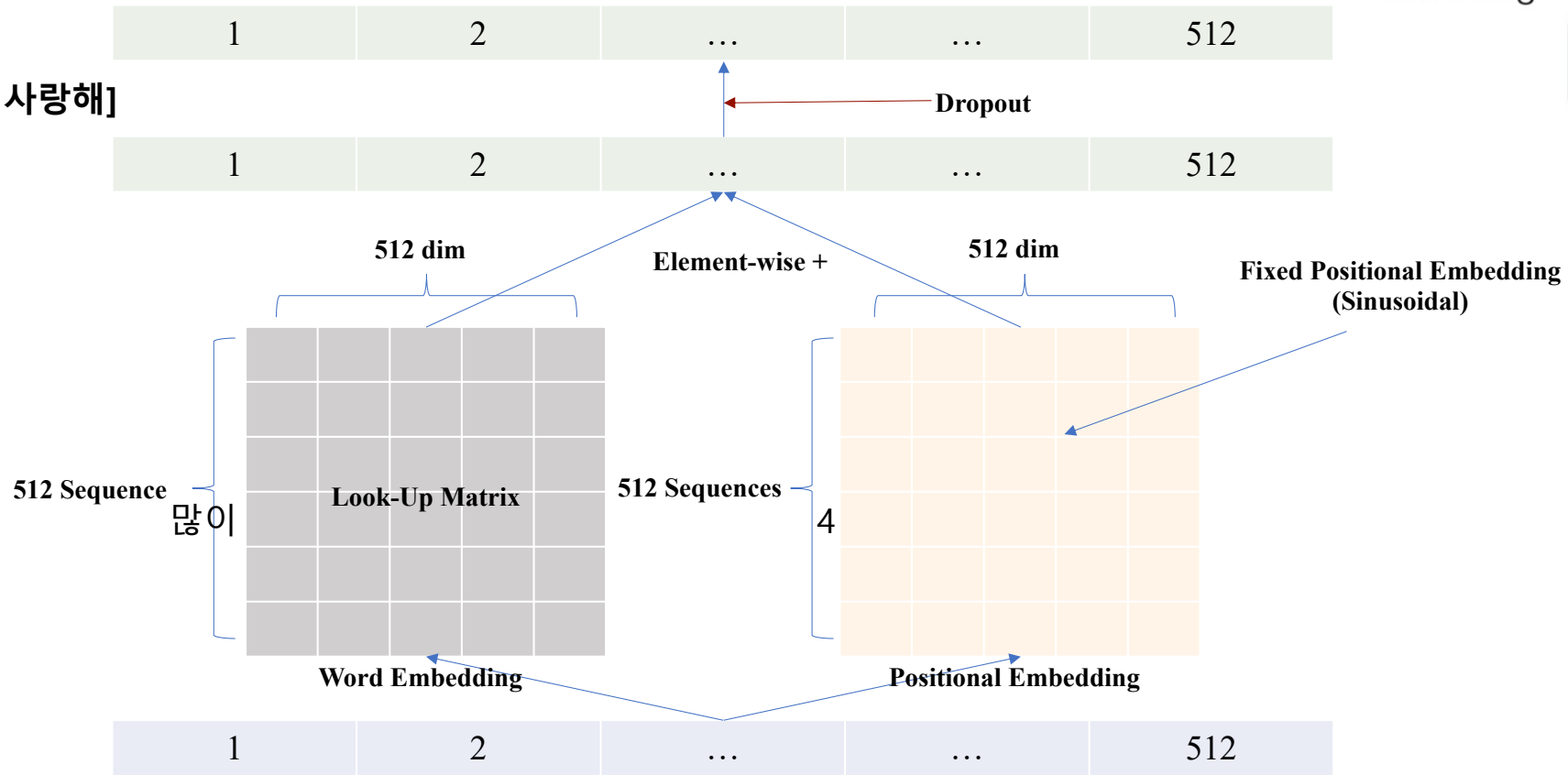
2. Learned Embedding (update with gradients)

Unit 03 | Transformer - Overview

Word Embedding Layer & Positional Embedding

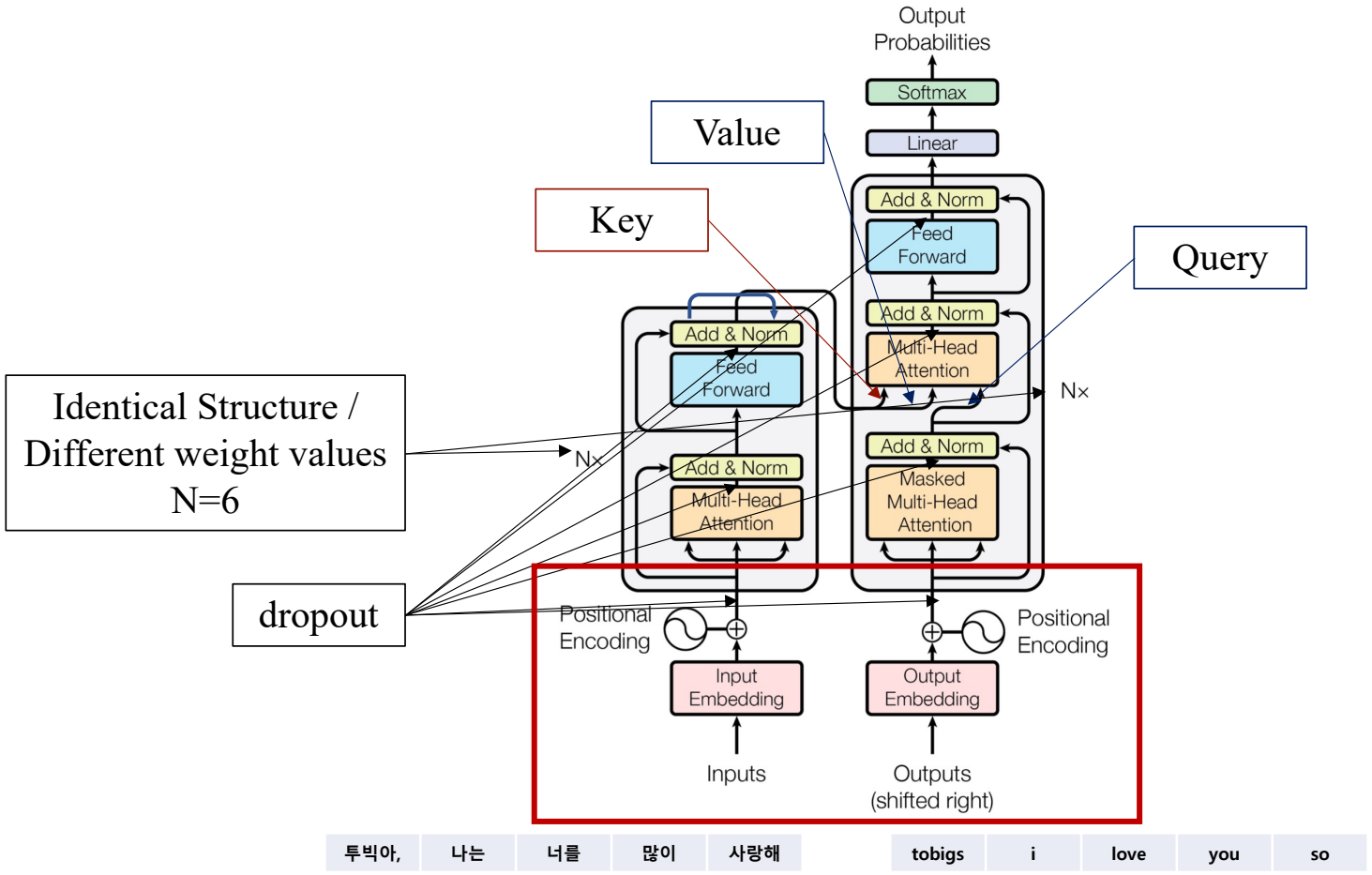
Input Example
[투빅아, 나는, 너를, 많이, 사랑해]





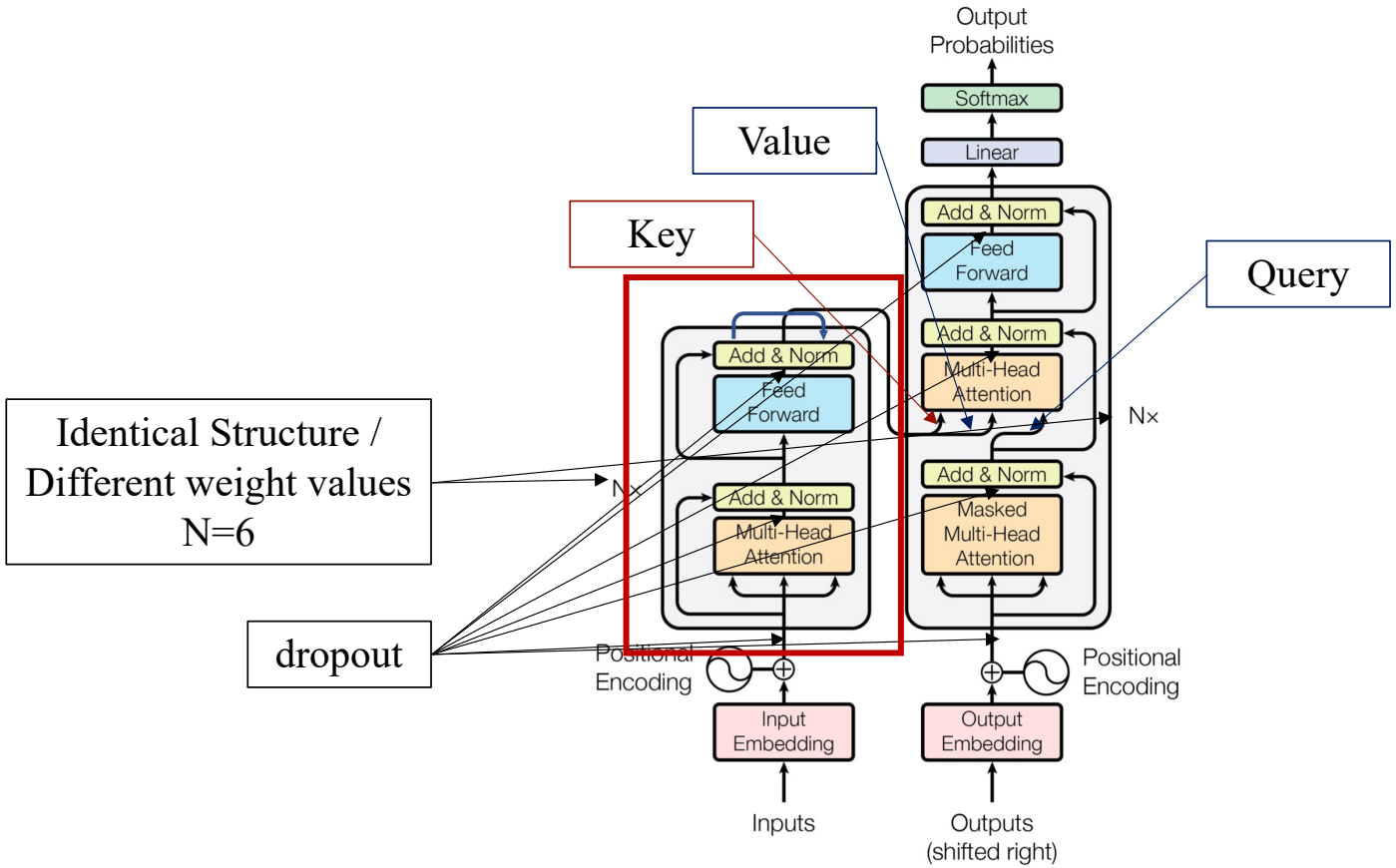
Unit 03 | Transformer - Overview

Recap : Input Embedding + Positional Encoding



Unit 04 | Transformer - Encoder

Transformer Encoder



Multi-head Self Attention Intuition

우리가 번역할 문장 : “투빅아, 나는 너를 많이 사랑해.”

‘투빅아,’는 ‘나’를 지칭할까, ‘너’를 지칭할까?

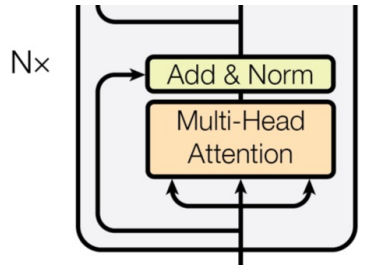
Unit 04 | Transformer - Encoder

Multi-head Self Attention Intuition

‘투빅아,’라는 단어에 대한 표현을 만들때,
우리가 ‘너를’이라는 의미를 투영하면 더 좋지 않을까?

단어 간의 관계? : Alignment
단어가 어떻게 표현되어있지? : Vector
그럼 관계는 어떻게? : Dot Product Attention

Unit 04 | Transformer - Encoder



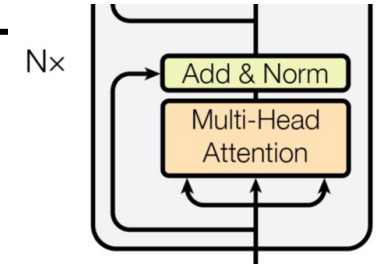
Multi-head Self Attention Intuition

우리가 번역할 문장 : "투빅아, 나는 너를 많이 사랑해.'
'투빅아,'라는 단어벡터에 '너를'의 의미를 투영시키자!

Query : '투빅아,'가 '투빅아,', '나는', '너를', '많이', '사랑해.'와 **관계를 질문**하기 위해 만들어진 벡터

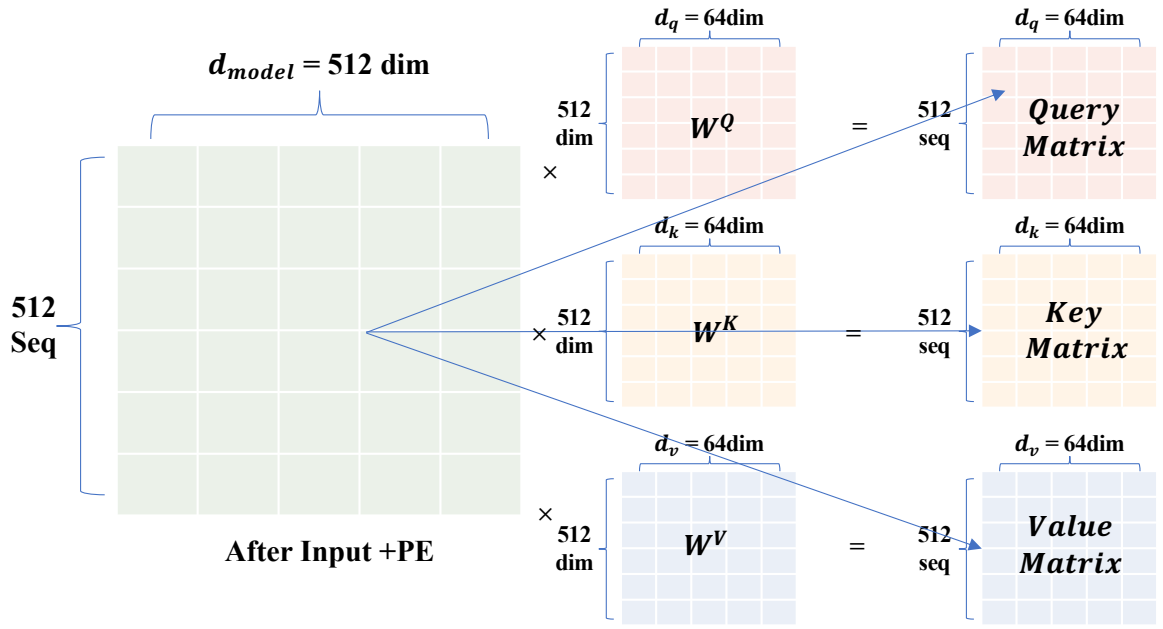
Key : '투빅아,', '나는', '너를', '많이', '사랑해.'가 '투빅아,'의 **Query에 답변**을 하기 위해 만들어진 벡터

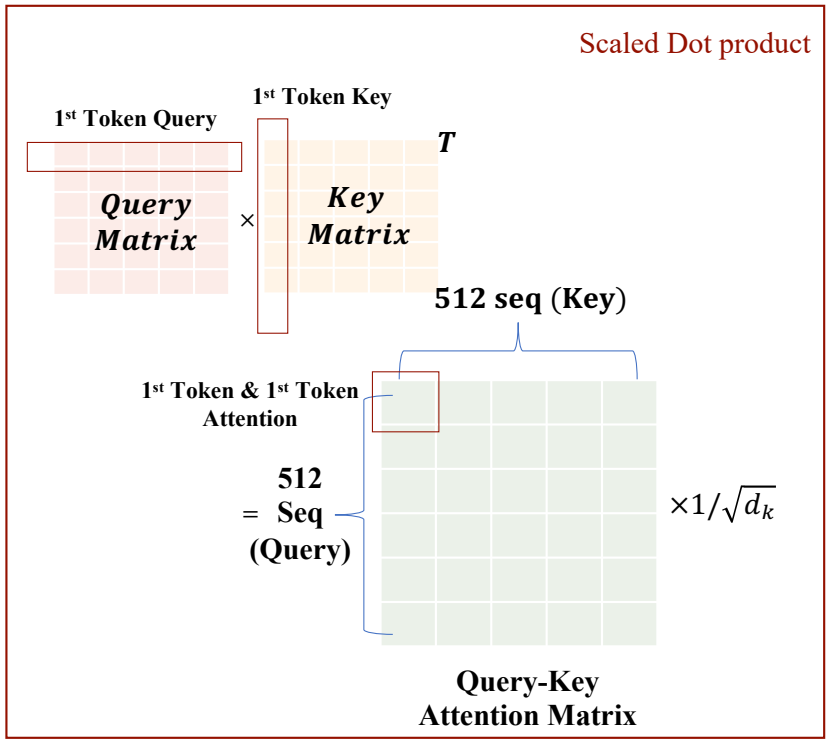
Value : '투빅아,'를 다시 만들때, **Query와 Key가 만든 관계를 반영**하기 위해 새로 만들어진 '투빅아,', '나는', '너를', '많이', '사랑해.' 벡터



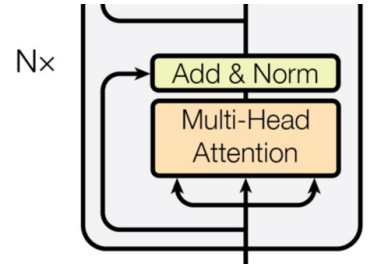
Multi-head Self Attention : Query, Key, Value / Scaled Dot product

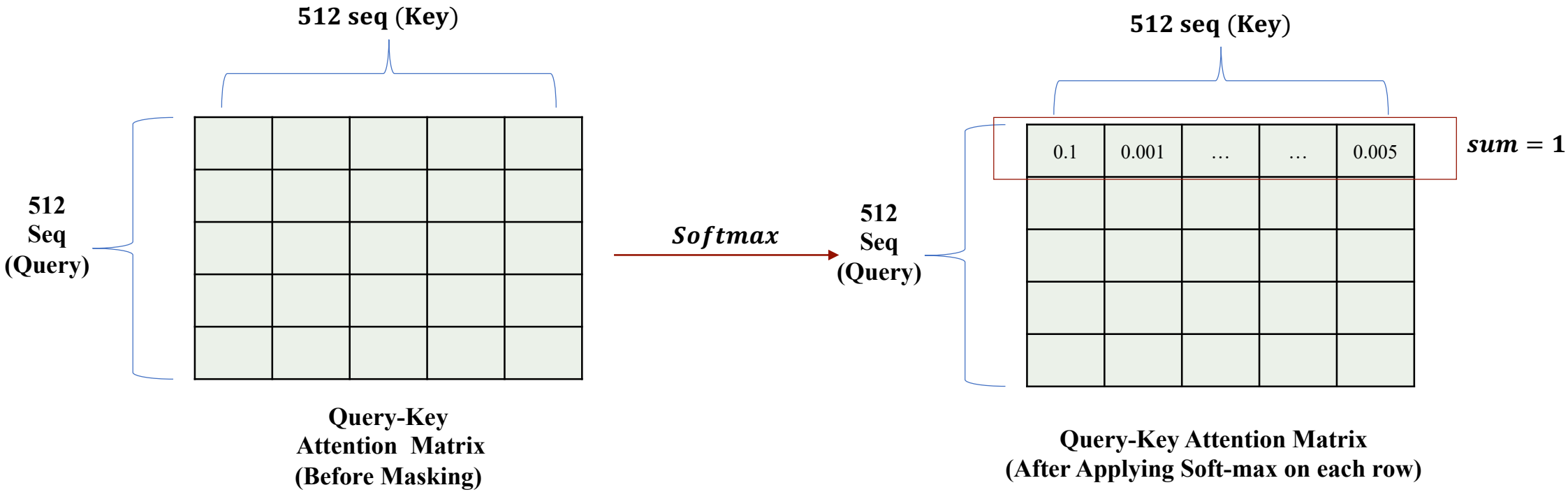
- Input Embedding과 PE를 통과한 512차원의 벡터들을 64차원의 작은 Query, Key, Value 벡터로 투영시킨 후 Query와 Key Matrix를 내적!
- 512차원/64차원 = 8개 (8개의 다른 Query, Key, Value 벡터로 투영이 가능! = Multi-head)

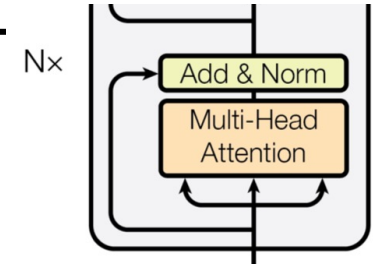




Multi-head Self Attention : Query, Key, Value / Softmax







Multi-head Self Attention : Weighted Sum

Weighted Sum = Linear combination

512 seq (Value)

d_v **= 64 dim**

d_v **= 64 dim**

0.1	0.001	…	…	0.005

512 Seq (Query)

×

512 Seq (Value)

=

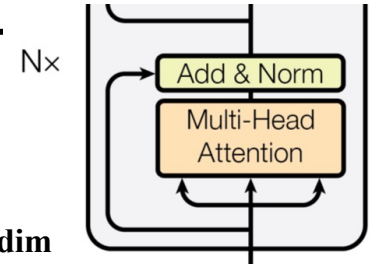
512 Seq

Query-Key Attention Matrix (After Applying Soft-max on each row)

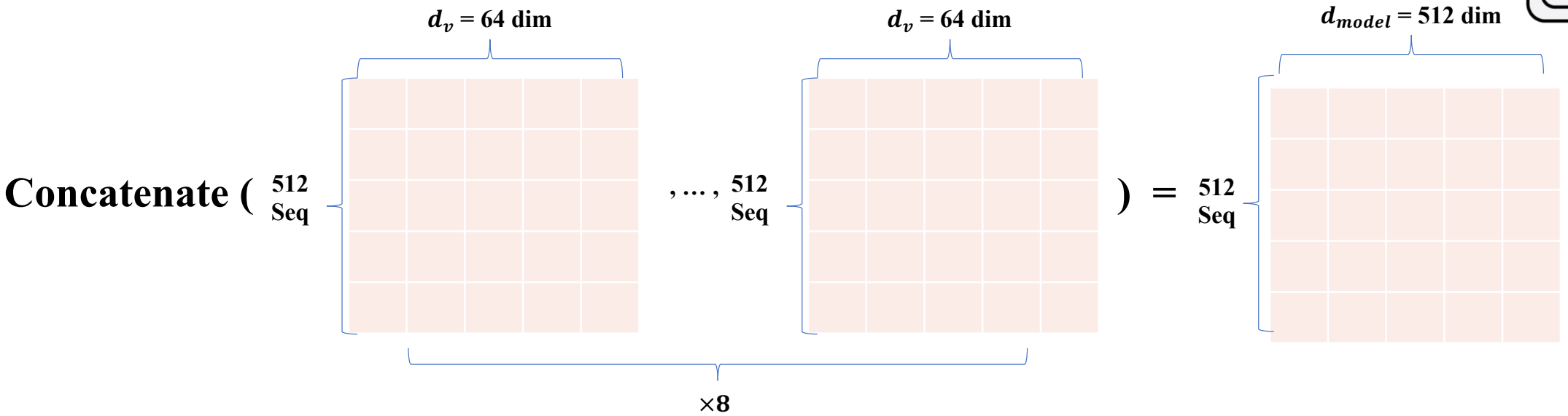
Value Matrix

After Attention

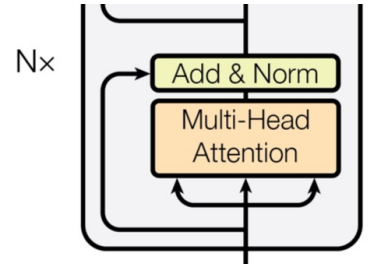
Unit 04 | Transformer - Encoder



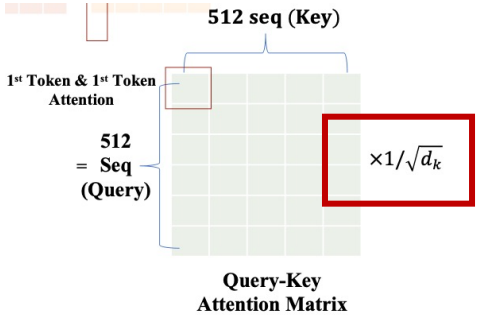
Multi-head Self Attention : Mixed-up Multi-head Information



Unit 04 | Transformer - Encoder



Multi-head Self Attention : Why Scaling?



- Scaled-dot Product : $\sqrt{d_k}$ 가 커질수록 Dot-product값은 커지게 된다.
- 이럴 경우, Softmax가 극단적인 분포를 따르게 되어 gradient가 잘 안 흐를 수가 있다.

a=

vector	a
a_1	100
a_2	50
a_3	50

Execute | Clear | Store/Read | Print | 14digit

Softmax function

b	value
b_1	1
b_2	1.9287498479639E-22
b_3	1.9287498479639E-22

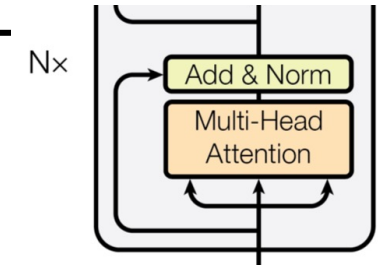
a=

vector	a
a_1	10
a_2	5
a_3	5

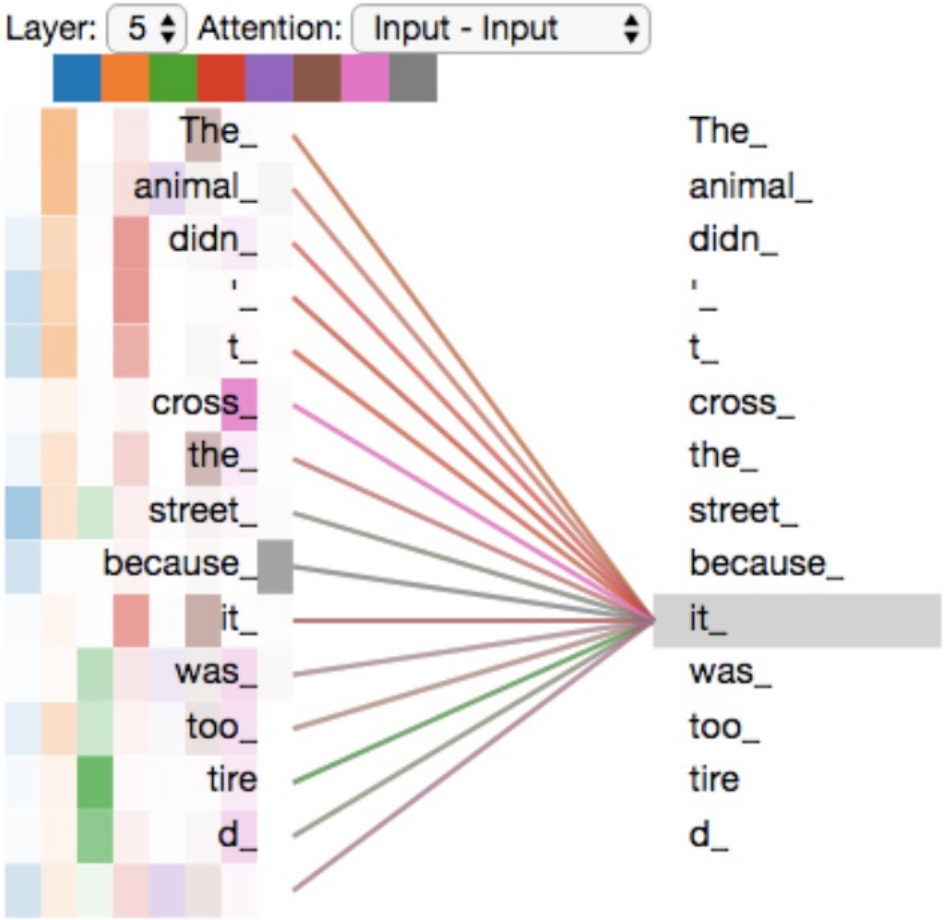
Execute | Clear | Store/Read | Print | 14digit

Softmax function

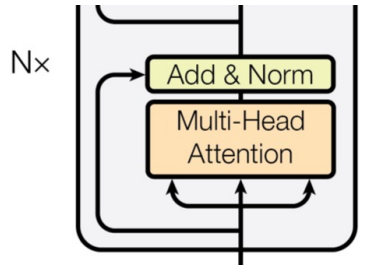
b	value
b_1	0.98670329104227
b_2	0.006648354478866
b_3	0.006648354478866



Multi-head Self Attention Intuition



Unit 04 | Transformer - Encoder



Skip-Connection (Residual Connection)

- Multi-head Attention 이후에 dropout을 해준 후, Multi-head Attention에 들어가기 이전의 Input 과 한번 더해주는 Skip-Connection을 진행

- Skip-Connection을 진행해주면 상위 레벨의 Gradient가 그대로 한번 더 흘러서 학습이 원활해짐

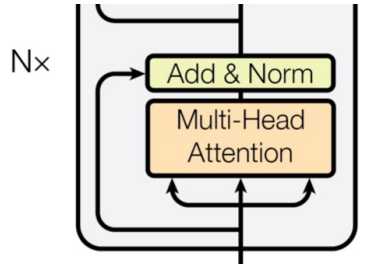
$$x_{i+1} = x_i + \sum_{n=1}^{i} f(x_i, w_i)$$

$$x_{i+2} = x_{i+1} + \sum_{n=1}^{i} f(x_i, w_i)$$

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial x_{i+1}} \times \frac{\partial x_{i+1}}{\partial x_i}$$

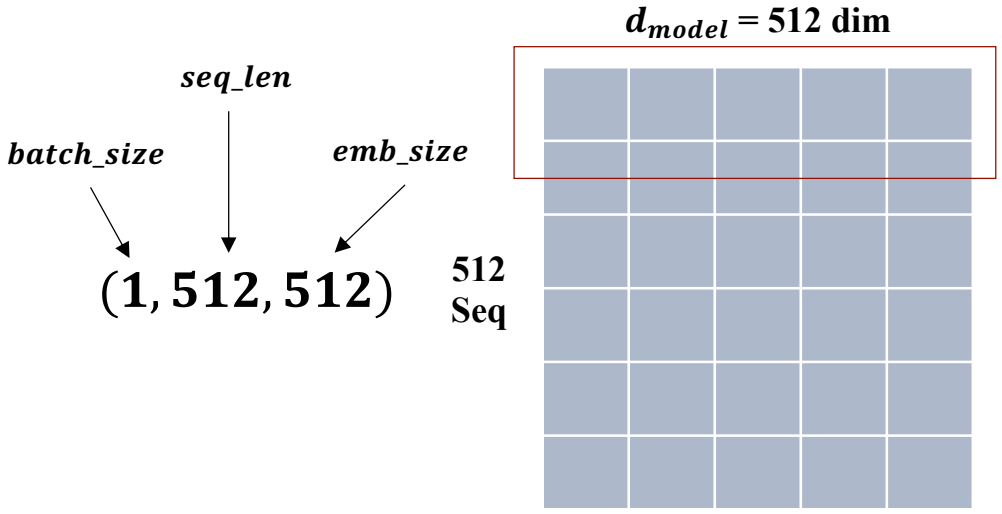
$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial x_{i+1}} \times (1 + \frac{\partial}{\partial x_i} \sum_{n=1}^{i} f(x_i, w_i))$$

Unit 04 | Transformer - Encoder



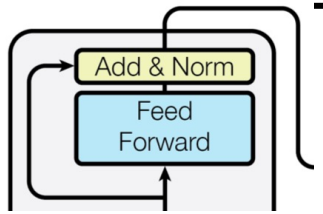
Layer Normalization

- **Skip-Connection 이후에 Layer Normalization을 수행**
- **Layer Normalization : Layer 단위로 Normalization을 수행해 Gradient를 안정적으로 흘림**



- ✓ 현재 예시에서는 각 token의 embedding에 대해서 Normalization을 수행해주면 Layer Normalization이 된다.
- ✓ $matrix_{[0,:]} = \frac{\alpha(matrix_{[0,:]} - \mu)}{\sigma} + \beta$
- ✓ α, β에 따라 각 token마다 표준정규분포에서 서로 다른 mean과 variance를 가진 분포로 변환된다.
- ✓ α, β 역시 학습되는 파라미터.

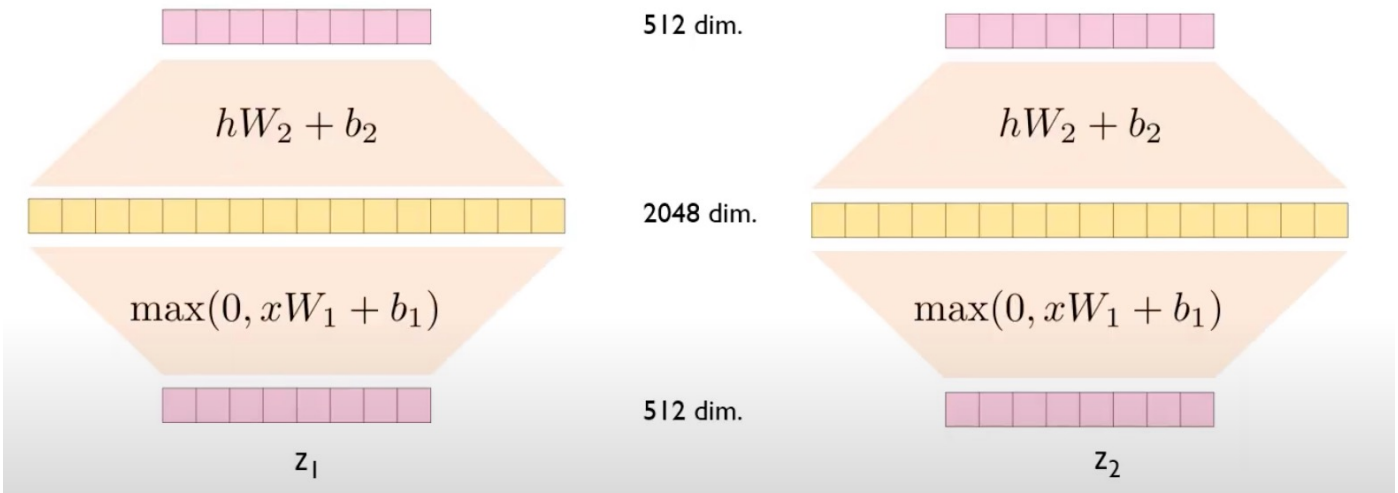
Unit 04 | Transformer - Encoder



Position wide Feed Forward Network

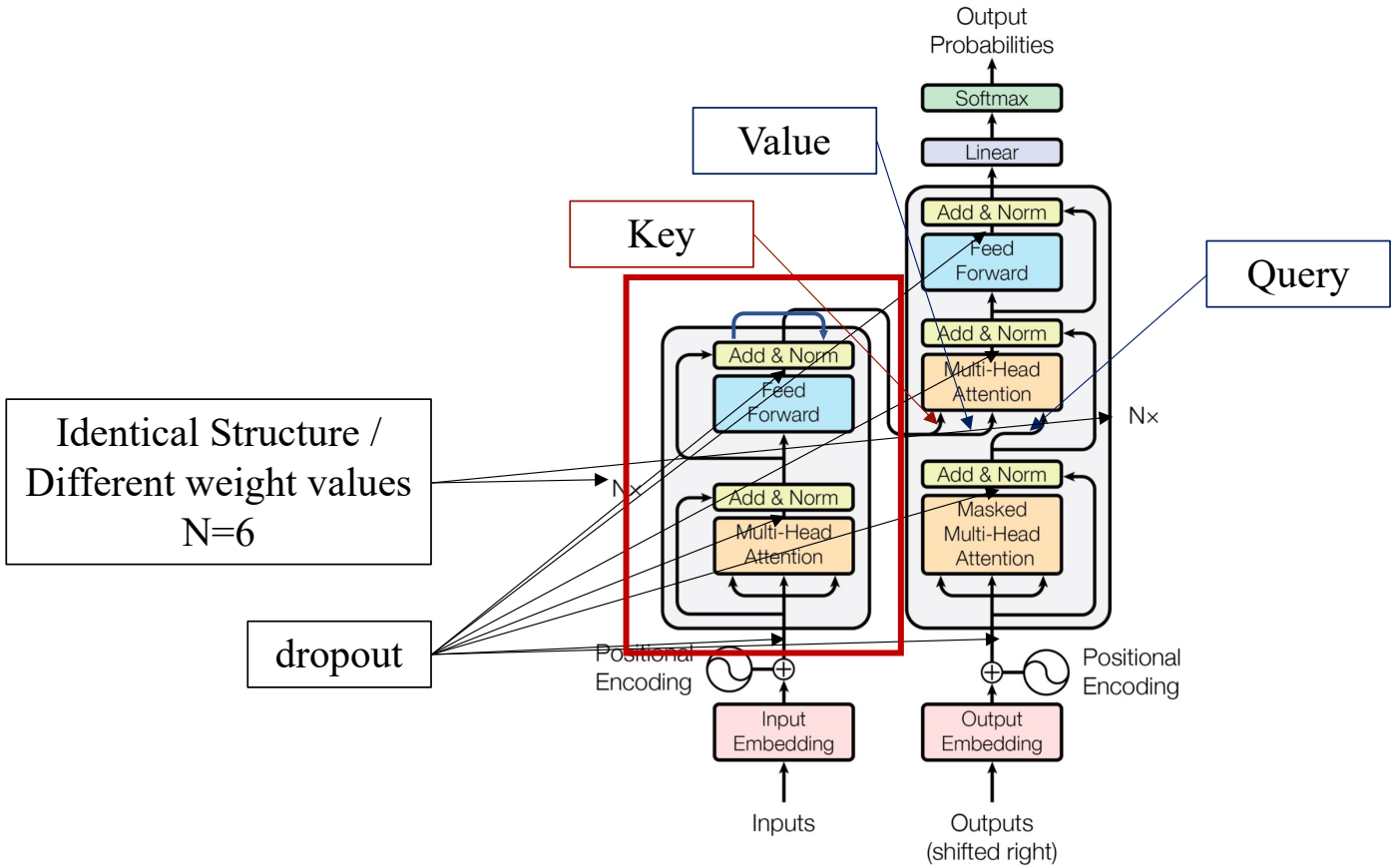
- **Applied to each position separately and identically**
- **Linear Transformations are the same across the different positions**
- **Different parameters from layer to layer**

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



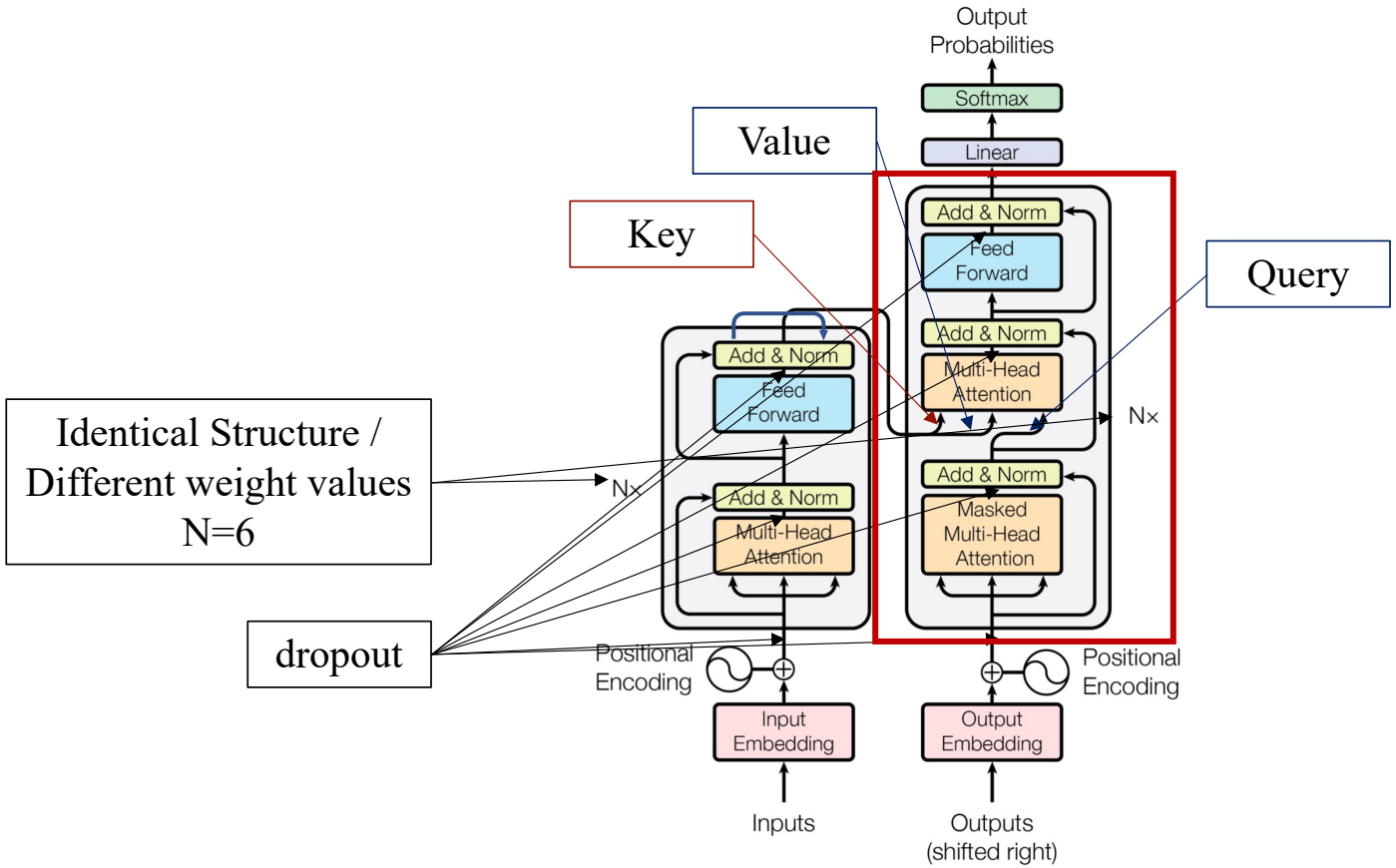
Unit 04 | Transformer - Encoder

Recap Encoder



Unit 05 | Transformer - Decoder

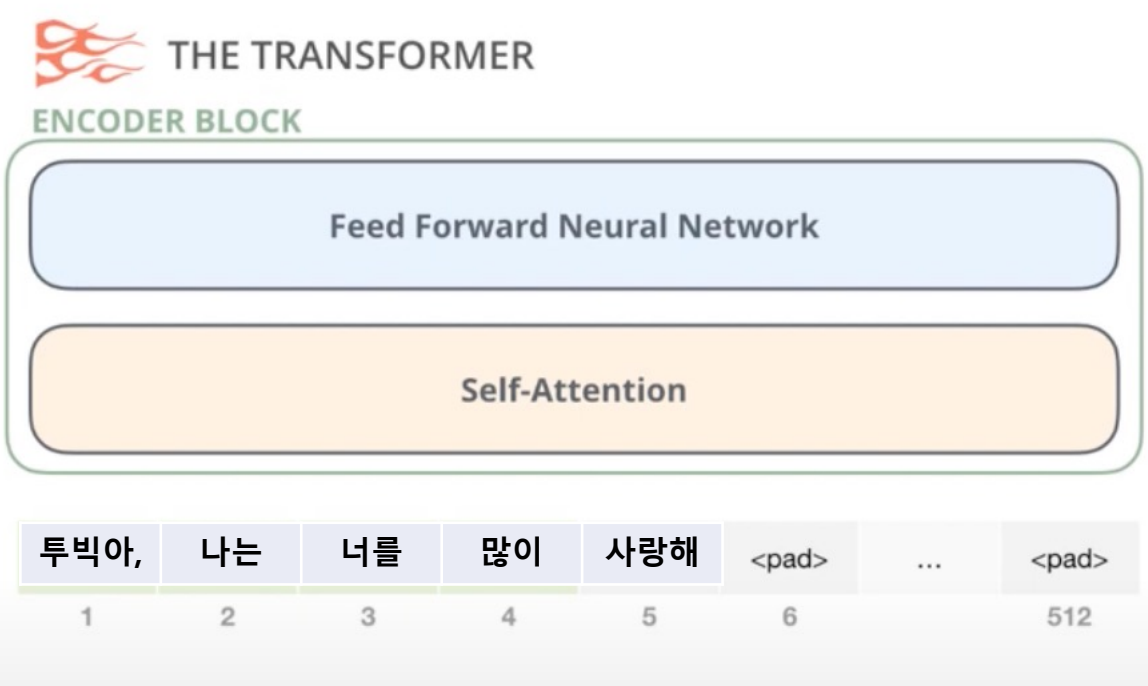
Transformer Decoder

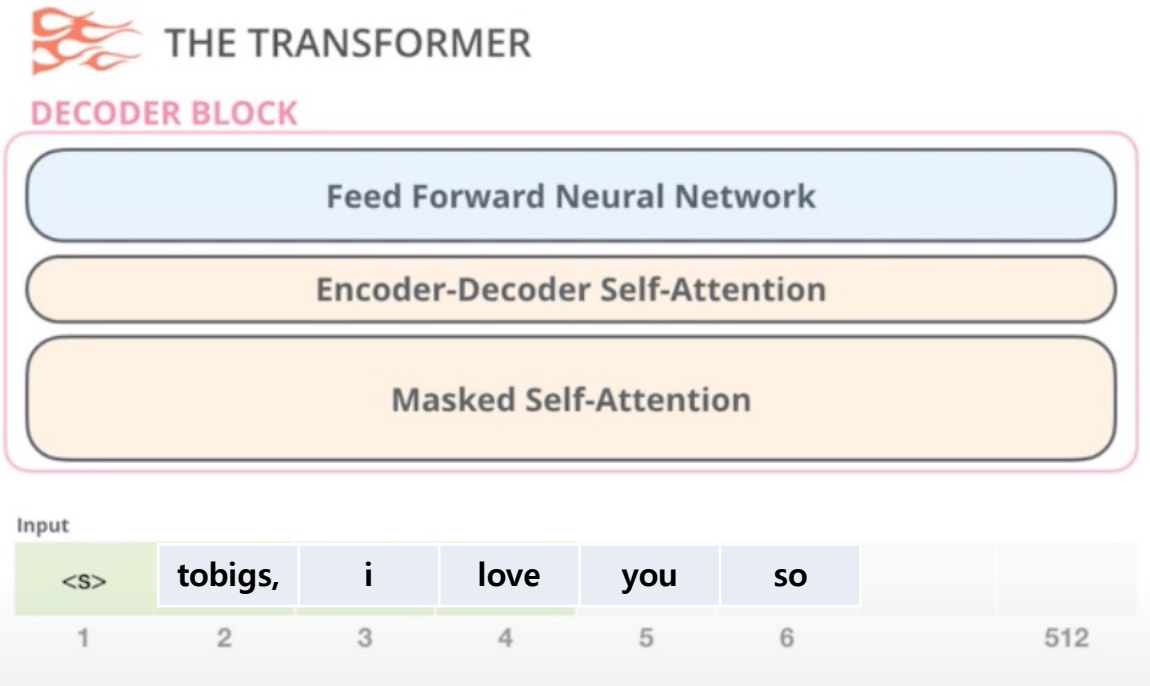


Unit 05 | Transformer - Decoder

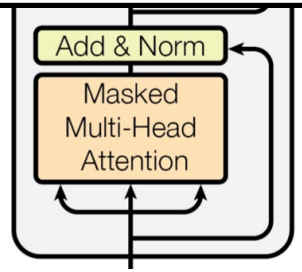
Transformer Decoder

Encoder is based on Unmasked Multi-head Self-Attention (원문은 양방향 정보를 다 반영하자!)
Decoder is based on Masked Multi-head Self-Attention (번역문을 생성할 때는 이전 정보만 반영하자!)



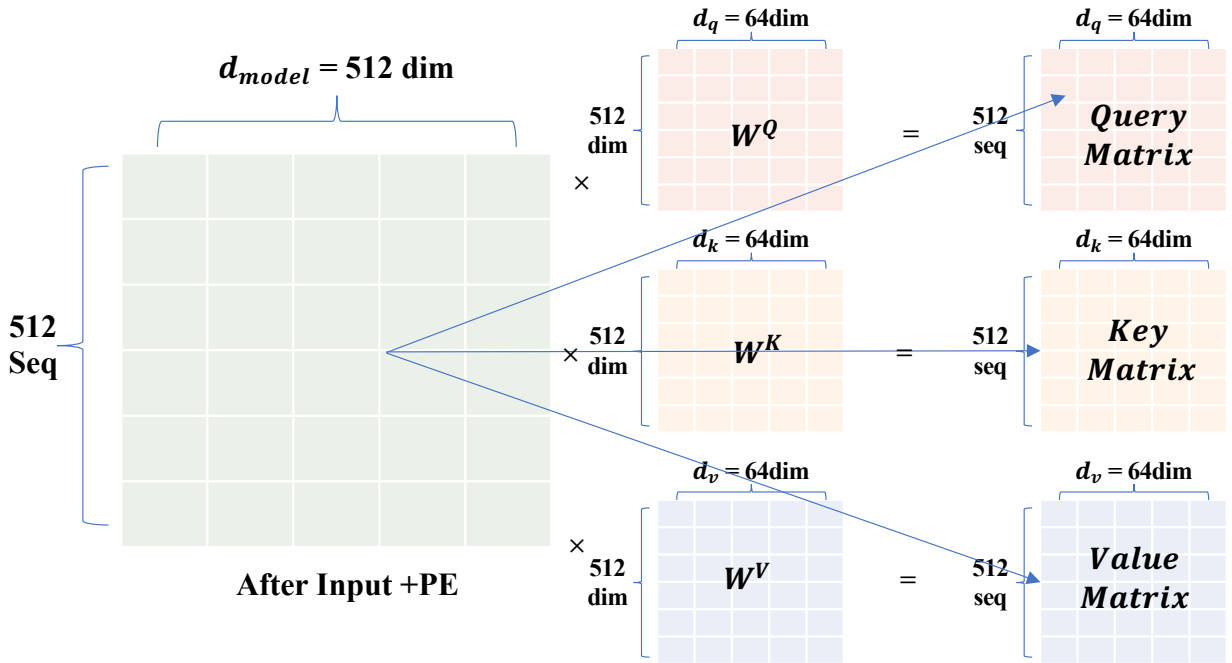


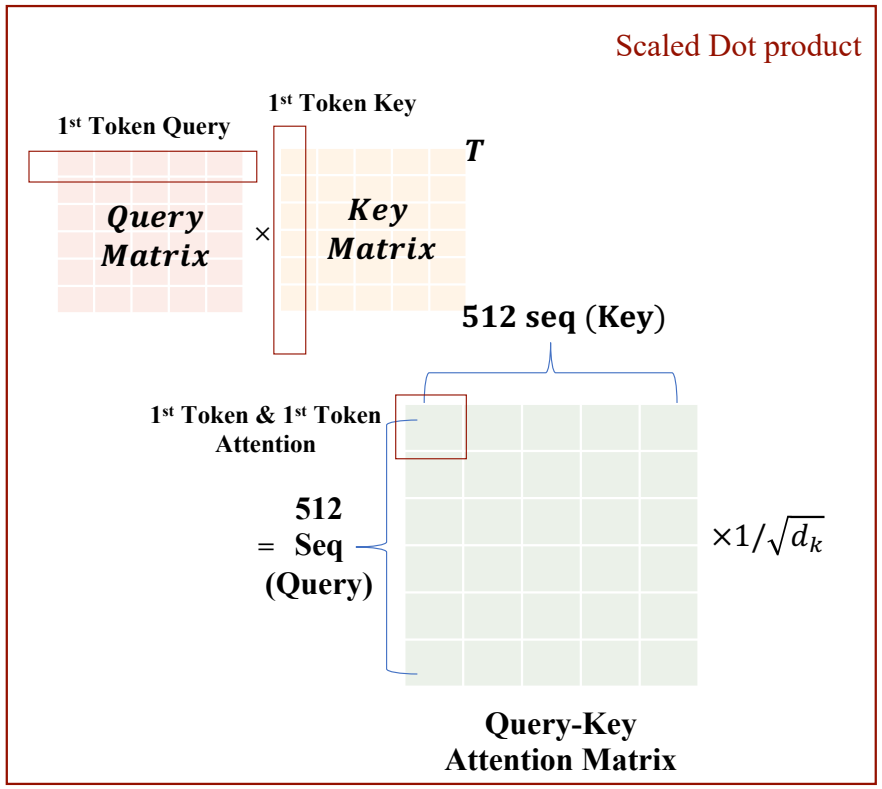
Unit 05 | Transformer - Decoder

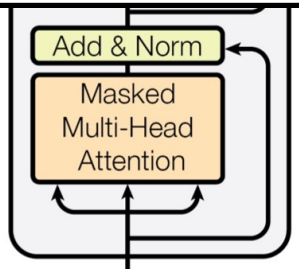


Masked Multi-head Self Attention : Query, Key, Value / Scaled Dot product

- Scaled Dot Product 과정은 Encoder와 동일

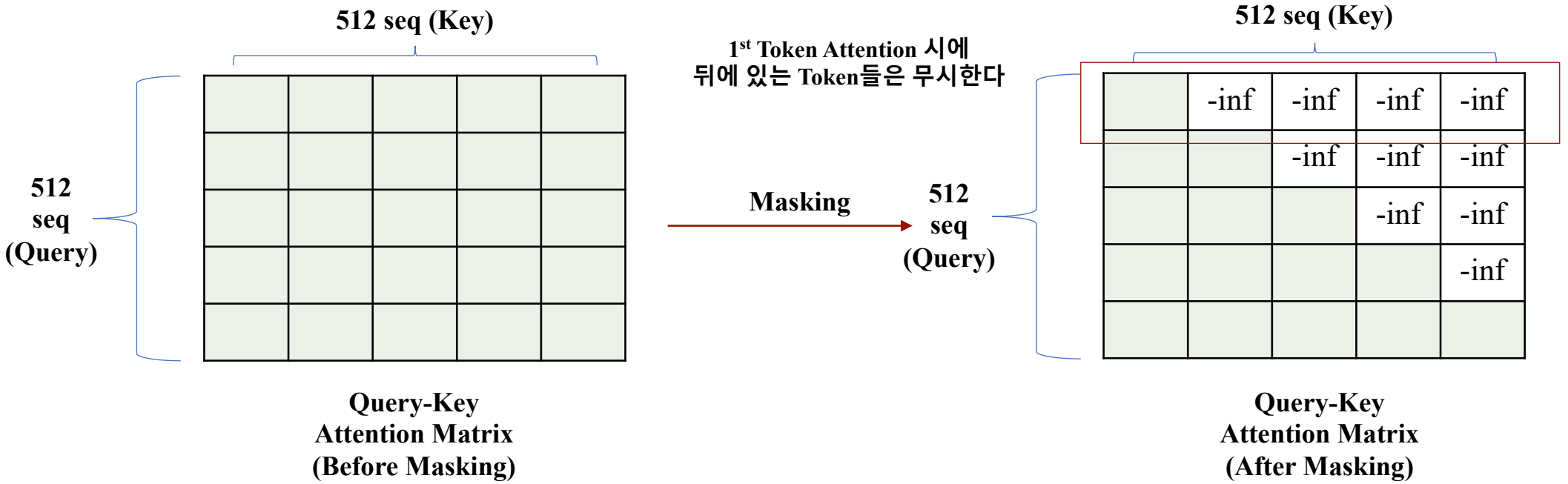






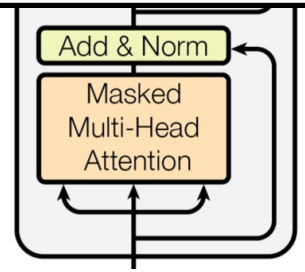
Masked Multi-head Self Attention : Masking

- 현재 position의 token을 예측하는데 있어서 현재 position의 token 정보까지만을 활용
- 이를 위해 Attention Matrix의 주대각성분 오른쪽 값에 매우 낮은 값을 부여해 Masking



Unit 05 | Transformer - Decoder

Masked Multi-head Self Attention : Weighted Sum



512 seq (Key)

512 seq (Query)

1	0	0	0	0
0.46	0.54	0	0	0
0.34	0.16	0.5	0	0
				0

Query-Key Attention Matrix
(After Applying Soft-max on each row)

× 512 seq (Value)

64 dim

Value Matrix

=

512 seq

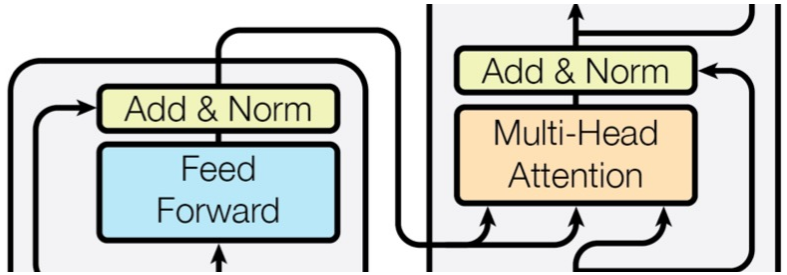
64 dim

After Masked Attention

이후 과정(Multi-head Mixing, Skip-Connection, Layer Normalization)은 Encoder와 동일함

Unit 05 | Transformer - Decoder

Cross Attention (Encoder-Decoder Attention Intuition



우리가 번역할 문장 : "투빅아, 나는 너를 많이 사랑해."

번역된 문장 : "Tobigs, I love you so much."

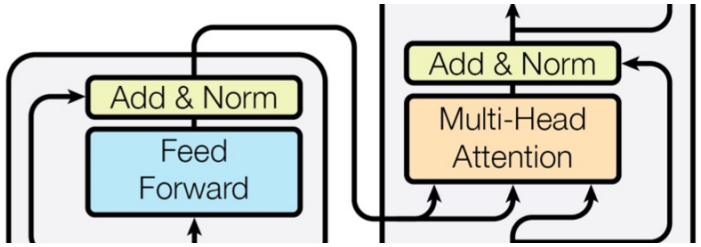
'Tobigs'를 생성할 때 '투빅아',와 '너'를 반영하면 좋지 않을까?

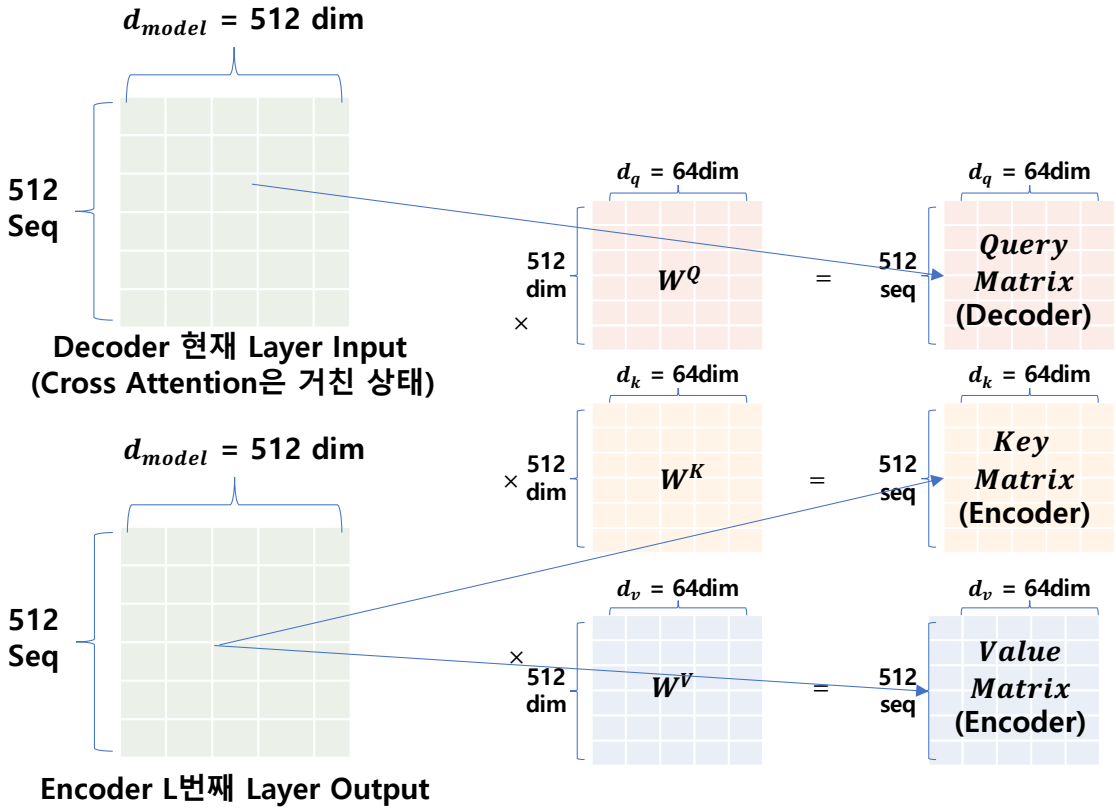
Query **Key & Value**

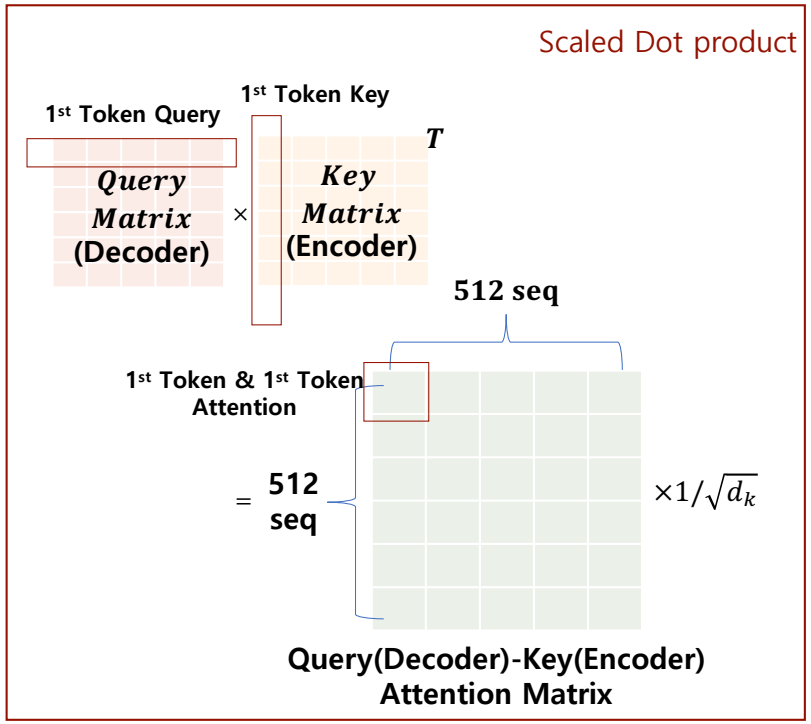
Unit 05 | Transformer - Decoder

Cross Attention (Encoder-Decoder Attention)

- Encoder의 마지막 Layer Matrix는 Decoder의 각 Layer, Head마다 다른 벡터로 투영됨 (이후 Weighted Sum 과정은 동일)



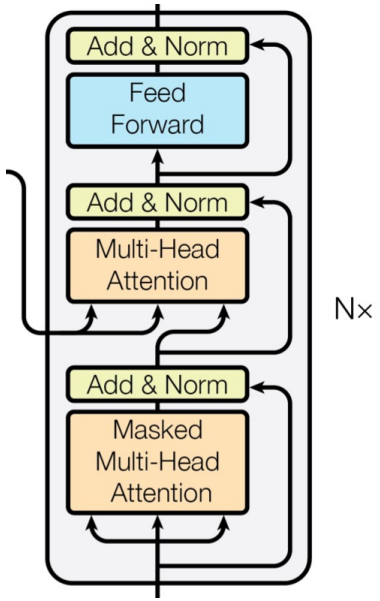




Unit 05 | Transformer - Decoder

Skip Connection & Layer Normalization & Position-wide FFN

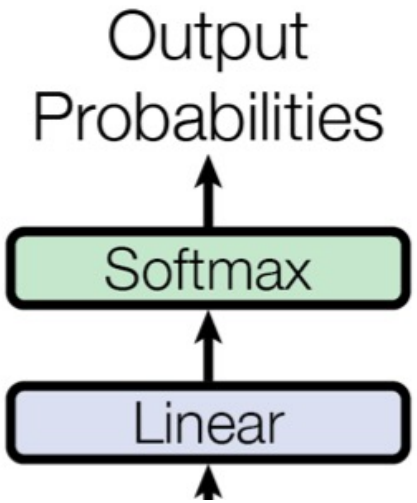
- **Decoder 역시 Attention 작업 이후에 FFN을 거쳐서 비선형성을 부여**
- **Masked Multi-head Attention, Cross Attention, FFN를 통과한 후에 Skip Connection & Layer Normalization을 거치도록 설계함 (Gradient 안정성)**

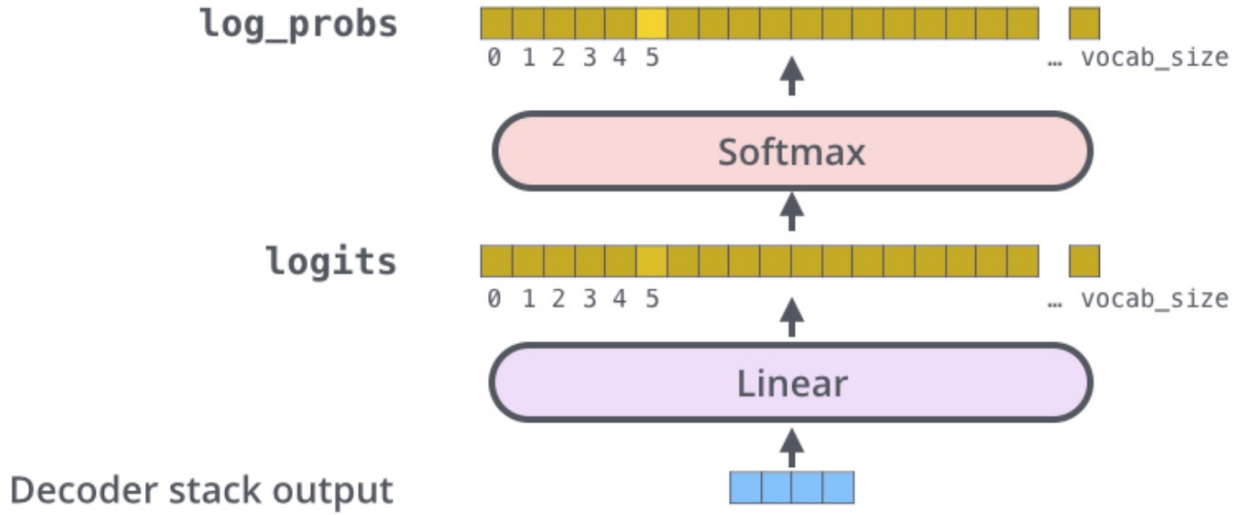


Unit 05 | Transformer - Decoder

Linear Layer & Softmax Layer

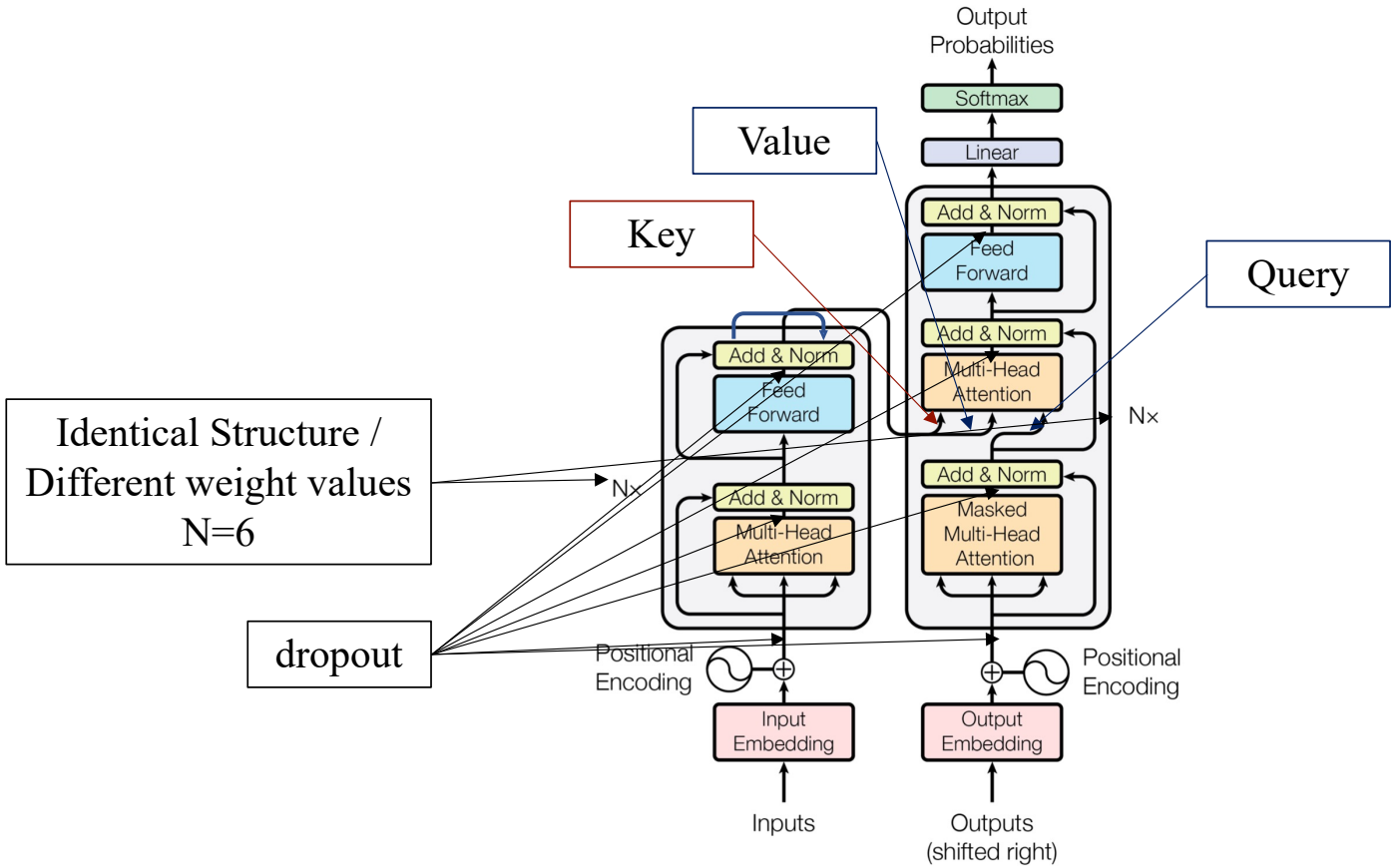
- **Linear Layer를 통과해 단어개수만큼 차원을 맞춰줌 (Logit Matrix)**
- **Softmax Layer를 통과해 현재 position의 단어를 예측**





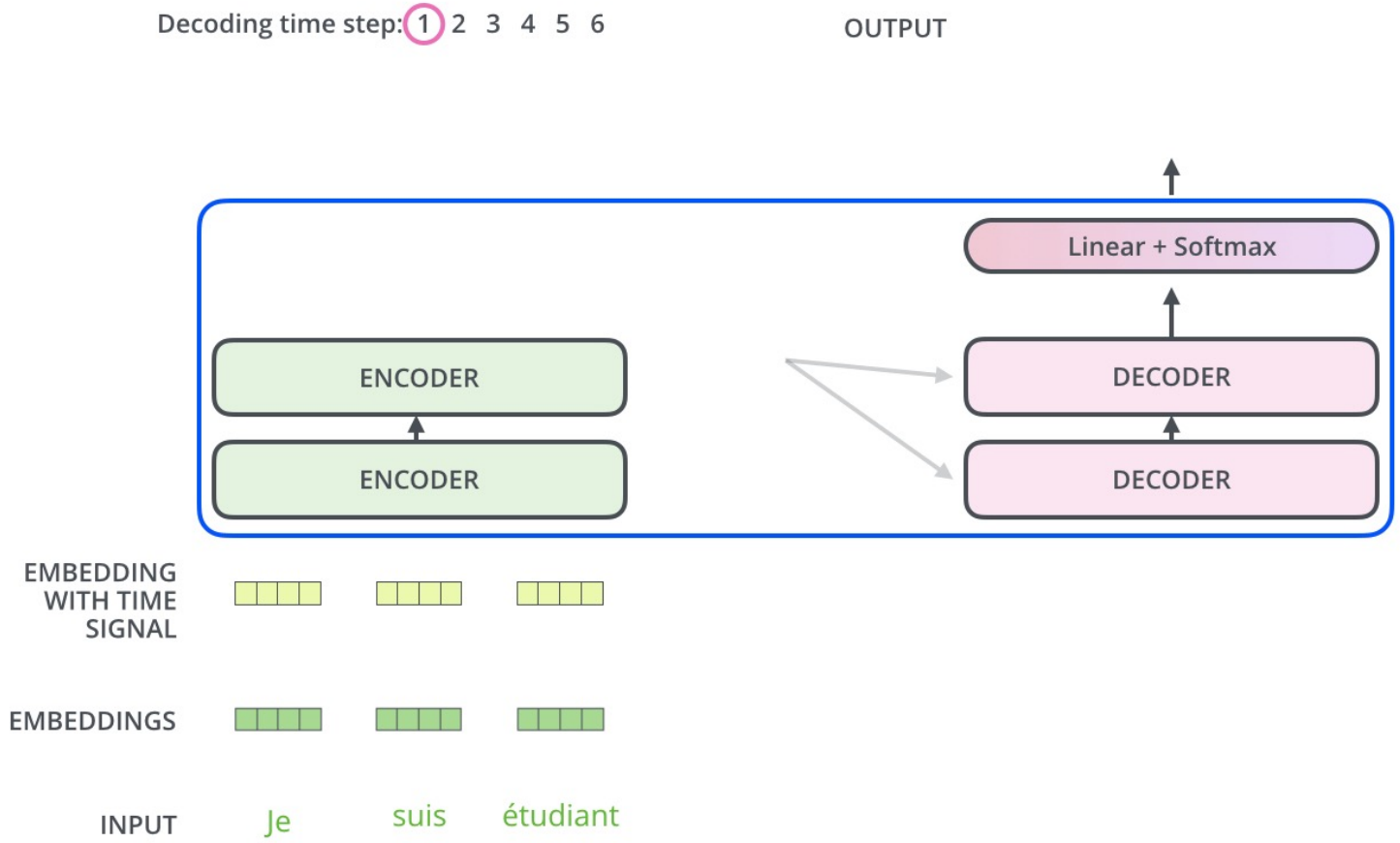
Unit 05 | Transformer - Decoder

Recap Transformer



Unit 05 | Transformer - Decoder

Recap Transformer



Transformer Decoder-based (Auto Regressive Model)

Autoregressive models

As mentioned before, these models rely on the decoder part of the original transformer and use an attention mask so that at each position, the model can only look at the tokens before the attention heads.

Original GPT

All model pages openai-gpt Model documentation openai-gpt

Improving Language Understanding by Generative Pre-Training, Alec Radford et al.

The first autoregressive model based on the transformer architecture, pretrained on the Book Corpus dataset.

The library provides versions of the model for language modeling and multitask language modeling/multiple choice classification.

GPT-2

All model pages gpt2 Model documentation gpt2

Language Models are Unsupervised Multitask Learners, Alec Radford et al.

A bigger and better version of GPT, pretrained on WebText (web pages from outgoing links in Reddit with 3 karmas or more).

The library provides versions of the model for language modeling and multitask language modeling/multiple choice classification.

CTRL

All model pages ctrl Model documentation ctrl

CTRL: A Conditional Transformer Language Model for Controllable Generation, Nitish Shirish Keskar et al.

Same as the GPT model but adds the idea of control codes. Text is generated from a prompt (can be empty) and one (or several) of those control codes which are then used to influence the text generation: generate with the style of wikipedia article, a book or a movie review.

The library provides a version of the model for language modeling only.

Transformer-XL

Transformer Encoder-based ((Denoising) Auto Encoder Model)

BERT

All model pages bert Model documentation bert

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob Devlin et al.

Corrupts the inputs by using random masking, more precisely, during pretraining, a given percentage of tokens (usually 15%) is masked by:

- a special mask token with probability 0.8
- a random token different from the one masked with probability 0.1
- the same token with probability 0.1

The model must predict the original sentence, but has a second objective: inputs are two sentences A and B (with a separation token in between). With probability 50%, the sentences are consecutive in the corpus, in the remaining 50% they are not related. The model has to predict if the sentences are consecutive or not.

The library provides a version of the model for language modeling (traditional or masked), next sentence prediction, token classification, sentence classification, multiple choice classification and question answering.

ALBERT

All model pages albert Model documentation albert

ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, Zhenzhong Lan et al.

Same as BERT but with a few tweaks:

- Embedding size E is different from hidden size H justified because the embeddings are context independent (one embedding vector represents one token), whereas hidden states are context dependent (one hidden state represents a sequence of tokens) so it's more logical to have H >> E. Also, the embedding matrix is large since it's V x E (V being the vocab size). If E < H, it has less parameters.
- Layers are split in groups that share parameters (to save memory).
- Next sentence prediction is replaced by a sentence ordering prediction: in the inputs, we have two sentences A and B (that are consecutive) and we either feed A followed by B or B followed by A. The model must predict if they have been swapped or not.

The library provides a version of the model for masked language modeling, token classification, sentence classification, multiple choice classification and question answering.

RoBERTa

Transformer based Model (Seq2Seq Model)

BART

All model pages bart Model documentation bart

BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension, Mike Lewis et al.

Sequence-to-sequence model with an encoder and a decoder. Encoder is fed a corrupted version of the tokens, decoder is fed the original tokens (but has a mask to hide the future words like a regular transformers decoder). A composition of the following transformations are applied on the pretraining tasks for the encoder:

- mask random tokens (like in BERT)
- delete random tokens
- mask a span of k tokens with a single mask token (a span of 0 tokens is an insertion of a mask token)
- permute sentences
- rotate the document to make it start at a specific token

The library provides a version of this model for conditional generation and sequence classification.

Pegasus

All model pages pegasus Model documentation pegasus

PEGASUS: Pre-training with Extracted Gap-sentences forAbstractive Summarization, Jingqing Zhang, Yao Zhao, Mohammad Saleh and Peter J. Liu on Dec 18, 2019.

Sequence-to-sequence model with the same encoder-decoder model architecture as BART. Pegasus is pre-trained jointly on two self-supervised objective functions: Masked Language Modeling (MLM) and a novel summarization specific pretraining objective, called Gap Sentence Generation (GSG).

- MLM: encoder input tokens are randomly replaced by a mask tokens and have to be predicted by the encoder (like in BERT)
- GSG: whole encoder input sentences are replaced by a second mask token and fed to the decoder, but which has a causal mask to hide the future words like a regular auto-regressive transformer decoder.

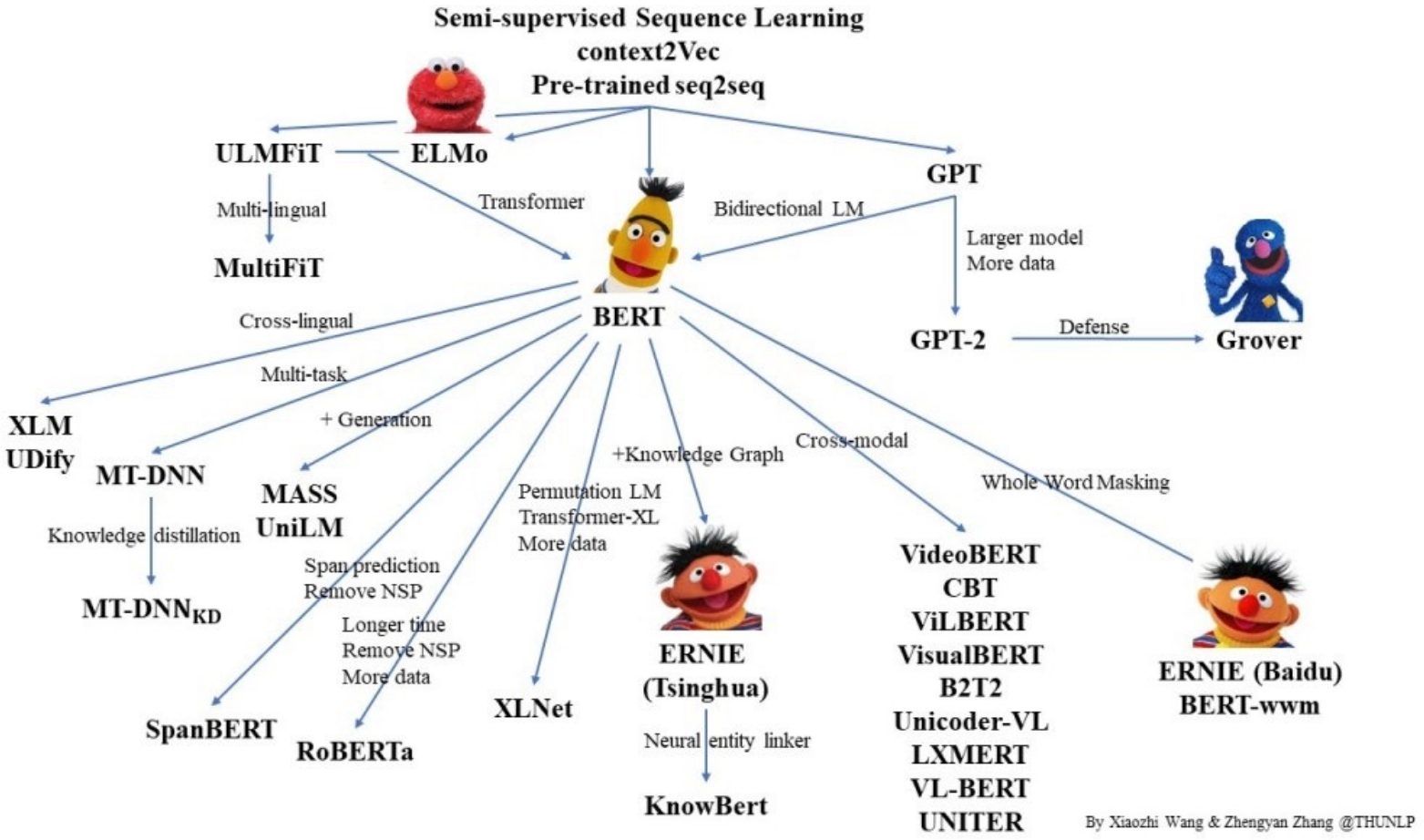
In contrast to BART, Pegasus' pretraining task is intentionally similar to summarization: important sentences are masked and are generated together as one output sequence from the remaining sentences, similar to an extractive summary.

The library provides a version of this model for conditional generation, which should be used for summarization.

MarianMT

Unit 06 | Transformer Variants

Sesame street



Unit 07 | Assignments

논문 리뷰 : 관심있는 분야의 트랜스포머 관련 논문을 리뷰해주세요! (아래 논문 중 택1)
제출 양식은 PPT or 블로그 or 아이패드 위에 직접 필사 중에 편하신 걸로 하셔서 .pdf 형식으로 제출해주세요!
필사를 선택해주신 분은 과제 제출 시에 contribution 3문장씩 적어주세요 (마감 : 03/29 23:59)

NLP
- **ELECTRA- Pre-training Text Encoders as Discriminators Rather Than Generators**
- **Big Bird: Transformers for Longer Sequences**

Vision
- **Image Transformer**
- **AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE (ViT)**

Audio
- **SPEECH-TRANSFORMER: A NO-RECURRENCE SEQUENCE-TO-SEQUENCE MODEL FOR SPEECH RECOGNITION**
- **Neural Speech Synthesis with Transformer Network**

Recommender system
- **Contrastive Learning for Sequential Recommendation**
- **Self-Attentive Sequential Recommendation**

예시는 과제에 같이 첨부되어있습니다!

References

Attention is All you Need (Transformer Paper)
https://arxiv.org/pdf/1706.03762.pdf

A Survey of Transformers
https://arxiv.org/pdf/2106.04554.pdf

CS224n Lecture Notes
http://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes05-LM_RNN.pdf
http://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture06-fancy-rnn.pdf

Jay Alammar 'The Illustrated Transformer'
https://jalammar.github.io/illustrated-transformer/

Multivariate Calculus Examples
https://zhuanlan.zhihu.com/p/101429370

Hugging Face Transformers (4.3.3)
https://huggingface.co/transformers/v4.3.3/index.html

고려대학교 강필성 교수님 Transformer 강의
https://www.youtube.com/watch?v=Yk1tV_cXMMU

이기창님 블로그
https://ratsgo.github.io/

들어주셔서 감사합니다.