Attention is All you need 리뷰

Seq2Seq with attention module의 확장판

Sequence를 RNN 기반으로 encode/decode/self -attention(attention의 일반화된 형태)

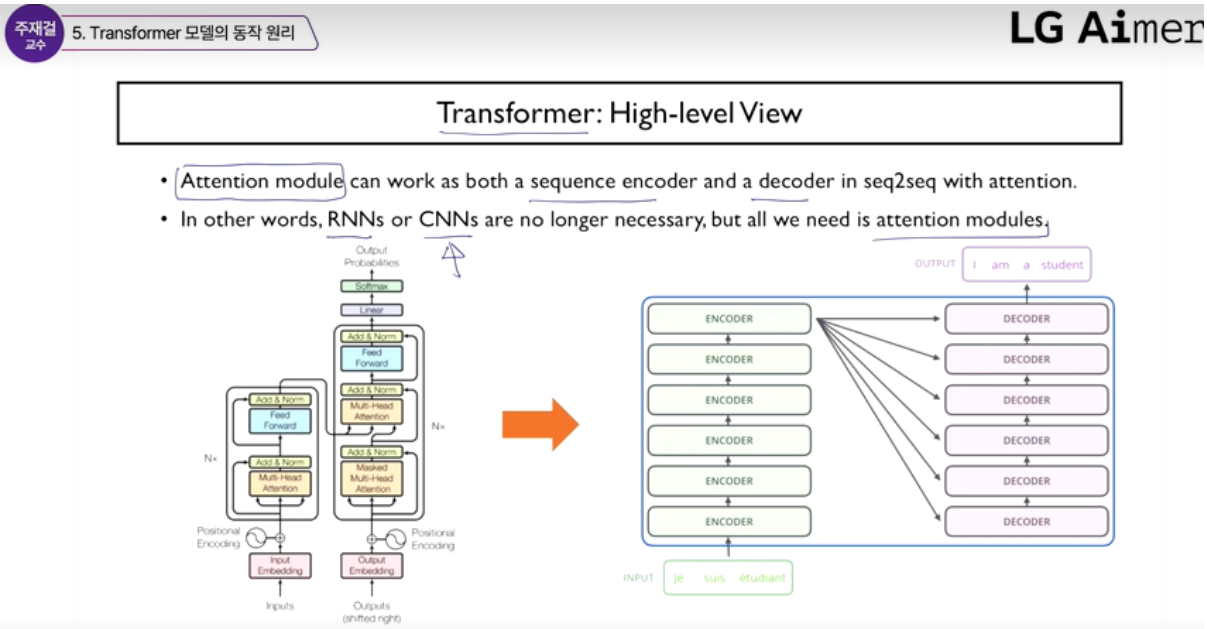
Layer Normalization, masked Self-attention…

Transformer: CV, NLP, 추천시스템 등 여러 분야에서 다 쓰임..

Transformer 모델의 동작 원리 - Seq2Seq Model의 개선판

Attention Module은, seq2seq with attention에서 sequence encoder/decoder 모두로서 작동할 수 있다.

RNN과 CNN를 기존에는 사용했었는데, Transformer에서는 attention module 만을 사용한다.

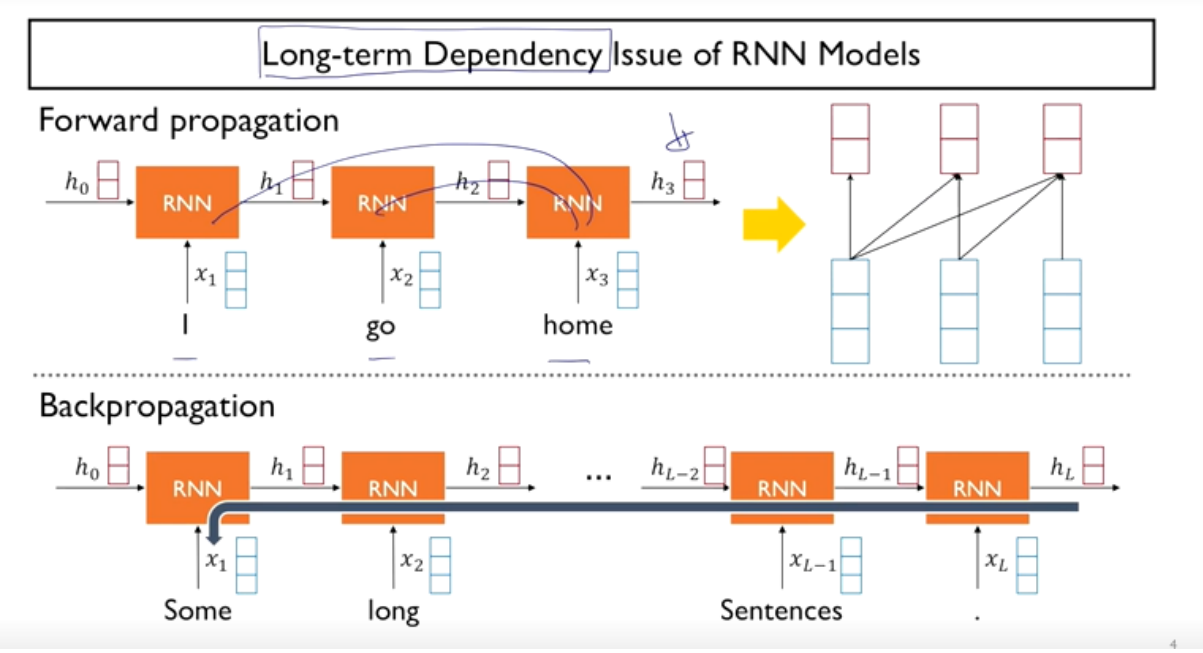


다양한 task에서 좋은 성능을 내고 있음.

RNN을 사용하여 sequence를 처리할 때 어떠한 Issue가 있는가?

→ **Long-term Dependency**

주어진 sequence를 잘 encoding 하여 정보를 축적할 때,



Hidden state vector h3에서 이전에 나타났었던 정보를 잘 담기 위해서는, 두 번의 rnn에 걸쳐 정보가 변질되거나 소실되지 말아야 한다.

또, 모델 학습 과정에서 가장 마지막 time step에서 계산된 hidden state vector가 예측 task에서 필요로 했던 정보가 이전의 문제를 소실시킴으로서,여러 time step에 걸쳐 정보가 전달되면서 정보가 변질이 됨

⇒ Attention model로서, **Long-term dependency의 issue를 근본적으로 해결**하고, 어떠한 **time step 정보에 대해 접근하여 attention을 sequence model을 활용**한다.

기존의 Seq2Seq with attention model:

정보를 찾고자 하는 쿼리: 특정 time step의 **decoder**의 hidden state vector 였음.

그것이, 어떠한 query로 동작하여, encoder hidden state vector 과의 내적을 통해 유사도를 구하고 softmax를 통과시켜서 상대적 합이 1인 가중치를 통해 encoder의 hidden state vector의 가중합을 attention model의 결과값으로 준다.

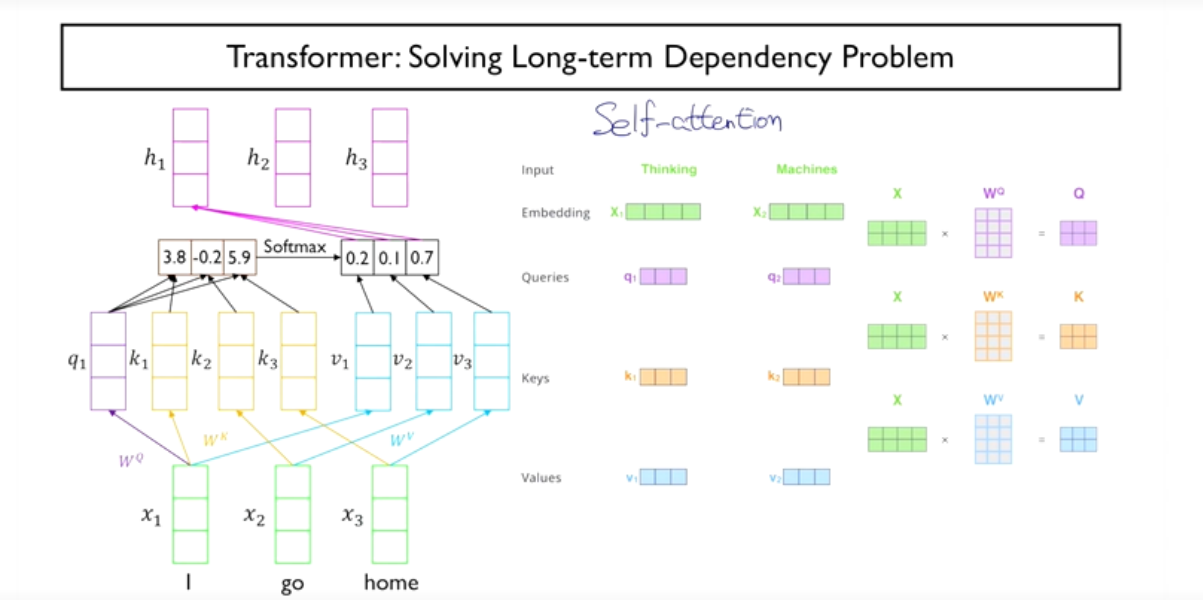
정보를 찾고자 하는 query(decoder hidden state vector)는, 정보를 꺼내가려고 하는 대상인 재료 정보들에 해당하는 encoder hidden state와는 분리된 상태임.

정보를 찾고자 하는 주체(쿼리)를 재료들과 같은 주체로서 생각을 하면, 주어진 sequence를 attention model로서 encoding이 가능함

⇒ 동일시가 된다

이러한 sequence data를 encoding 할 때 쓰이는 attention module

→ **Self-attention**



ex. I go home이라는 sequence를 encoding 할 때,

I라는 단어를 vector을 decoder의 hidden state vector(query vector)로 생각

*왜 해당 단어 임베딩 벡터가 쿼리 벡터인가? → Decoder입장에서,,, 얘를 decoding 해주세요라는 query를 받기 때문에 해당 단어 임베딩 벡터는 query 벡터임.

1. 정보를 끌어오려는 source vector 간의 (자신을 포함하여 내적. 그래서 self-임) 내적을 구하여 softmax를 통과시켜 유사도 가중치 벡터를 생성함.
2. 가중치 벡터를, 각 vector(초록색)의 가중치로 적용해서, source vector의 가중합을 구할 수 있음.

이 가중합은, 당시의 query로 사용되었던 단어에 대응하는 sequence 전체적인 정보를 잘 버무려서 만든 임베딩벡터라고 할 수 있음.

sequence 상의 encoding vector로서 사용되는 것임.

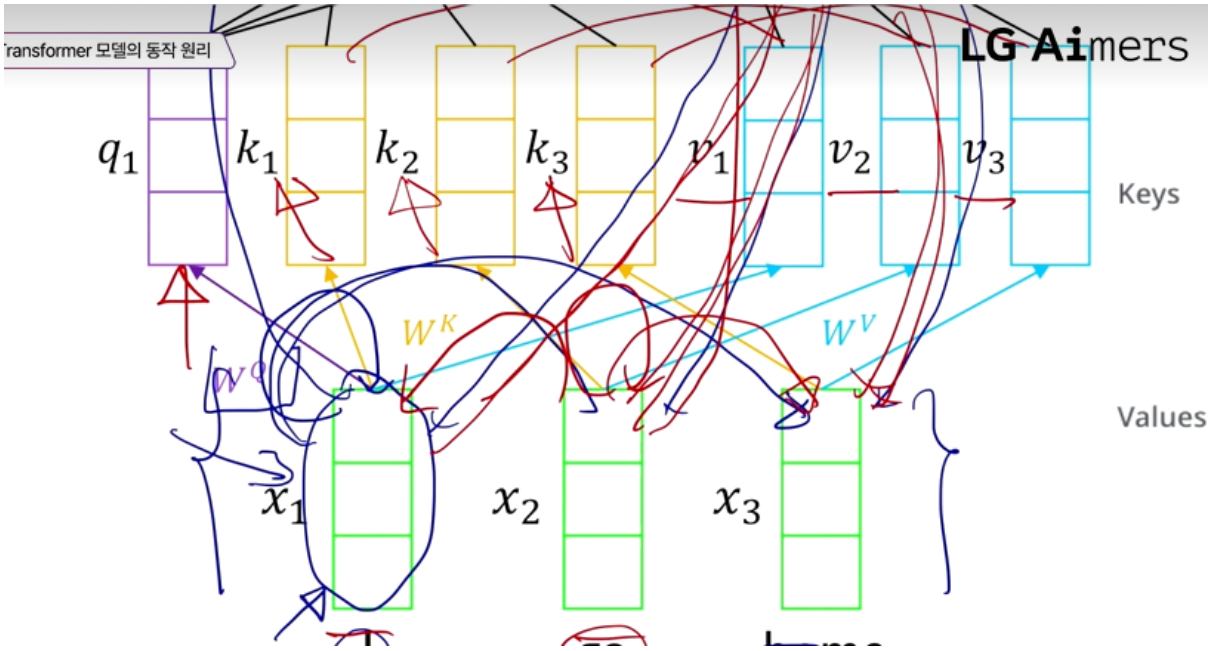
초록색 vector set의 각 vector들은,

1. query vector (혹은 decoder hidden state vector)처럼 사용이 되기도 하고
2. 재료 vector로서, query vector와 유사도를 구하게 되는 대상으로 사용되기도 하며
3. Softmax로 나온 가중치를 실질적으로 적용해서 가중합을 구하게 되는 재료 vector들 자체로도 사용된다.

Query, Key, key vector과 짝을 이뤄, 실제 유사도를 적용하여 가중치가 적용된 가중합으로서의 재료 vector 들인 value vector로서 세가지 다른 용도로서 사용됨

Self attention model에선,

vector들이 각각의 용도로서 사용될 떄, 좀더 layer들의 확장성과 유연성을 위해, 각 목적에 따라 3가지 다른 linear transformation을 따로 설계를 함



Query vector로서 사용이 되는 경우 → W_q로서의 linear transformation

Key vector(Source Vector)로서 사용이 되는 경우(Query vector의 내적의 대상이 되는 경우)→ W_k

Query vector와의 내적을 통해 유사도가 계산되어, softmax를 통해 나온 가중치를 실제 적용하게 되는 재료 vector들을 value vector들이고,

value vector들을 만들때에는, 주어진 입력 vector들은 W_v의 선형변환을 통해 value vector들로 변환하게 된다.

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

소프트맥스 함수를 통해 나온 결과값은 단어 각각이 출력 단어를 예측할 때 얼마나 도움이 되는지의 정도를 수치화한 값임.

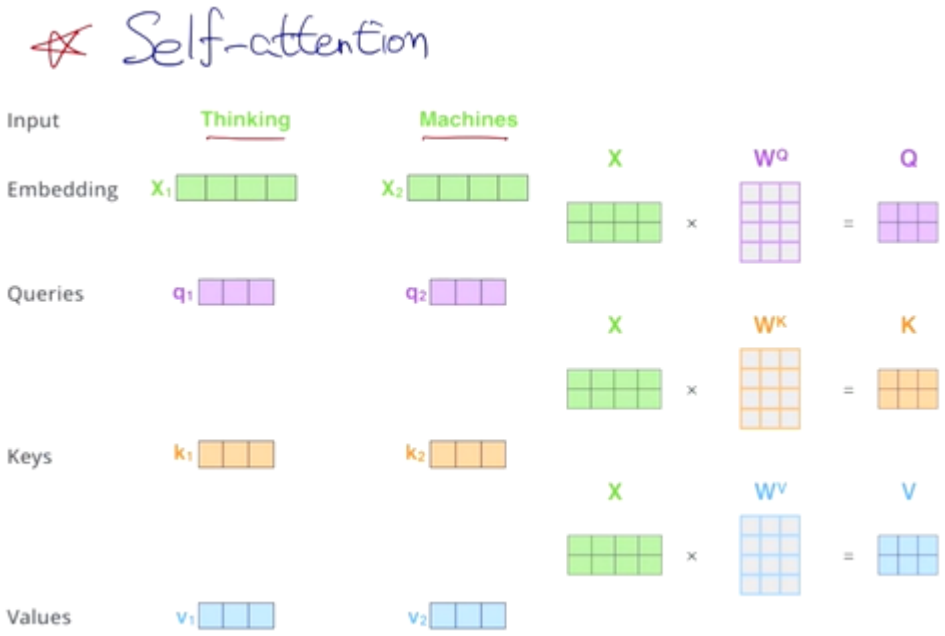
어텐션 함수는 주어진 '쿼리(Query)'에 대해서 모든 '키(Key)'와의 유사도를 각각 구한다. 그리고 구해낸 이 유사도를 키와 맵핑되어있는 각각의 '값(Value)'에 반영해준다. 그리고 유사도가 반영된 '값(Value)'을 모두 더해서 리턴한다. 여기서는 이를 어텐션 값(Attention Value)이라고 한다.

Attention은, 해당 t 시점에서 나올 다음 단어를 예측하기 위해, decoder는 예전에 받았던 문장들을 처음부터 다시 읽게 된다.

따라서, 문장의 각 단어들이 input 되었을 때, encoder의 key와 value들의 hidden state vector들을 참고하게 되는 것이다.

각 단어를 임베딩한 word vector들은, key로서도 쓰이고 value로서도 쓰일 수 있으며 최종적으로 가중치벡터을 구할떄도 쓰이는데, 이 때 각각의 3가지 용도로서 사용될 때마다 각각의 선형변환을 거쳐서 이용된다.

행렬연산의 관점)



주어진 sequence가 두 개의 단어만으로 이루어져 있다고 가정

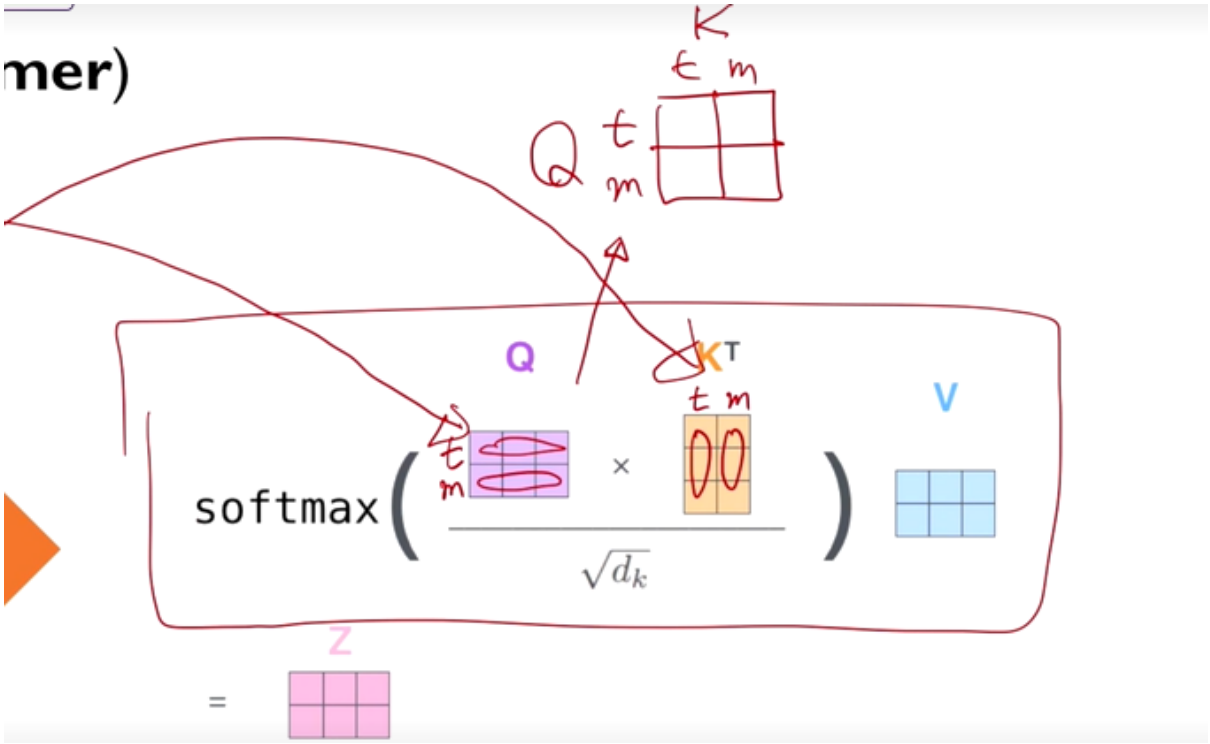
각각의 단어가 4차원 vector로서 나타난다고 하면, vector을 concat하여 입력 vector X로서 행렬을 만들어 줄 수 있음.

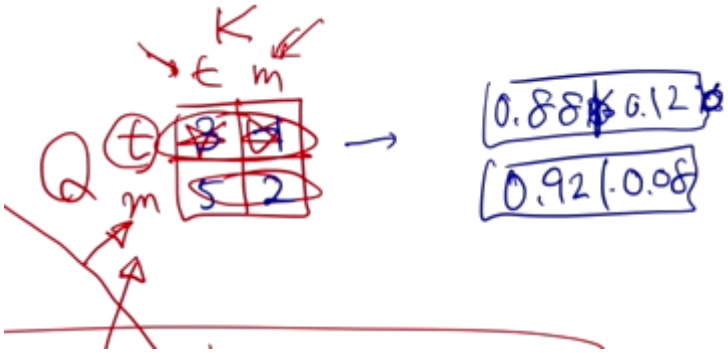
선형 변환을 통해, Query, Key, value vector들로 각각 변환이 가능

어텐션 스코어란,

현재 디코더의 시점 t에서 단어를 예측하기 위해, 인코더의 모든 은닉 상태 각각이 디코더의 현 시점의 은닉 상태 st
와 얼마나 유사한지를 판단하는 스코어값임!

각 단어별로 attention module / Self attention module을 통해, 각 단어를 sequence 상에서 encoding 하는 수식: (결과는, 각 단어에 대한 임베딩 벡터이다)

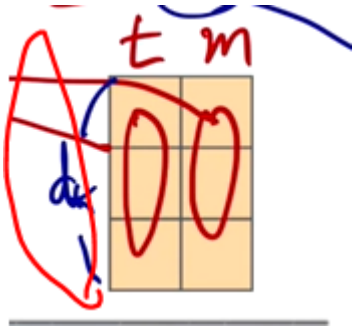




Query로서 thinking 이라는 단어가 key로서 thinking 과 machine과 내적했을 때 나오는 값들이 첫 row에 계산됨.

machine도 마찬가지. 이 때 내적했을 때 나오는 값은 유사도값들임.

Root of d_k: Dimension of K.



⇒ d_k는, W_k의 선형변환matrix에 의해 결정되는 dimension.

⇒ W_q의 해당 outputvector의 dimension 또한 같아야 한다.

<u>이 때 Q의 d_k(열)와 K_T의 row(기존 K의 column 수)의 dimension과 같아야 내적이 정의가 된다.********</u>**

이 때 d_k가 서로 다른 값을 가질 때를 생각을 해보면,

dimension이 크면 클수록 각 dimension 별로 원소 값들이 곱해지고 dimension의 개수만큼 곱한 값이 다 더해지는 형태가 되는데,

d_k가 커지면 커질수록 더 많은 개수들의 숫자들의 합이 될 것.

일반적으로 더 고차원 vector의 내적값들이 확률분포에서의 분산값들이 커지게 된다.

이러한 값들이 softmax layer들의 input으로 들어가게 될텐데,

값의 분산이 커질때는 softmax의 결과가,

지수함수가 어떤 값이 크면 클수록 기하급수적으로 커지기 때문에,

하나의 값에 대해서 거의 100%을 mapping 해줄 것임.(굉장히 하나의 값으로 단일한 픽으로서의 attention을 가지게 될 것임)

⇒ 이 경우, 학습의 측면에서 gradient가 잘 흐르지 않게 되는 문제점을 가지게 됨.

⇒ d_k라는 요소가, 원치않게 softmax의 입력으로 사용되는 logit vector의 분포의 분산값에 의도치않게 영향을 줌

⇒ 이러한 영향을 줄여주기 위해서,

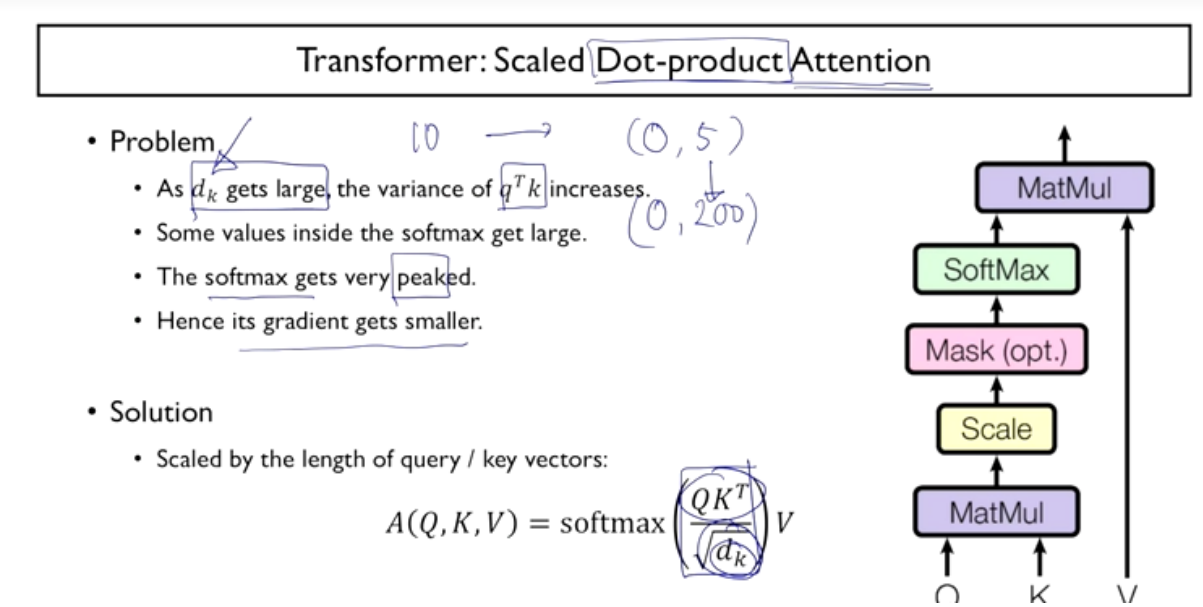
Query vector와 Key_transpose의 간의 내적에서

key vector 혹은 query vector에 dimension의 root 값을 나눠주어, 일정한 분산값을 만들어주게 됨(Normalization). 이 때 평균을 0으로 두어, 표준정규화 할 때

X-E(X)/sqrt(D_k)로서 해준다고 이해.

지금 Q*K_T에서 보면 Q의 열값과 K_T의 행수가 같게 되는데… 이를 보완해주고자 표준정규화를 진행.

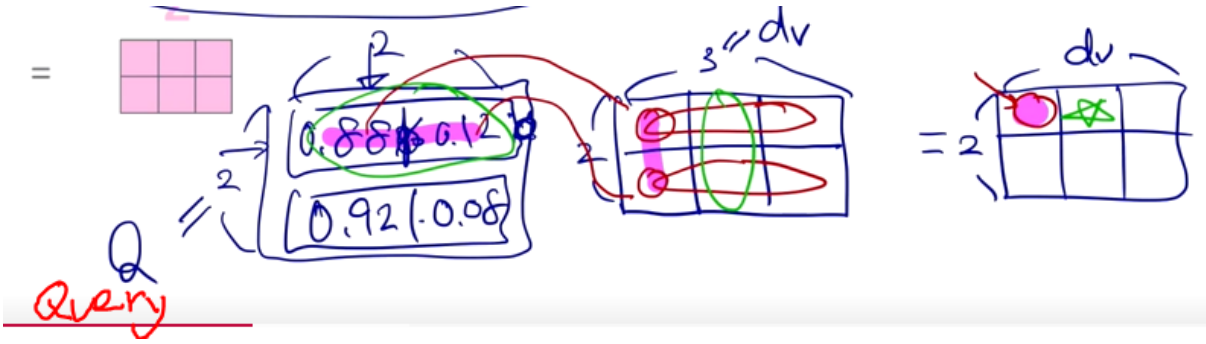
⇒ Dot-Product Attention



Dot Product / sqrt of d_k ⇒ scaled dot product attention

Softmax를 취하게 되면, 이 때 row 별로 softmax를 취하게 되는데

그렇게 되면 첫 query에 대해서 주어진 단어를 key로 봤을 때 가중치 벡터가 나오게 됨.

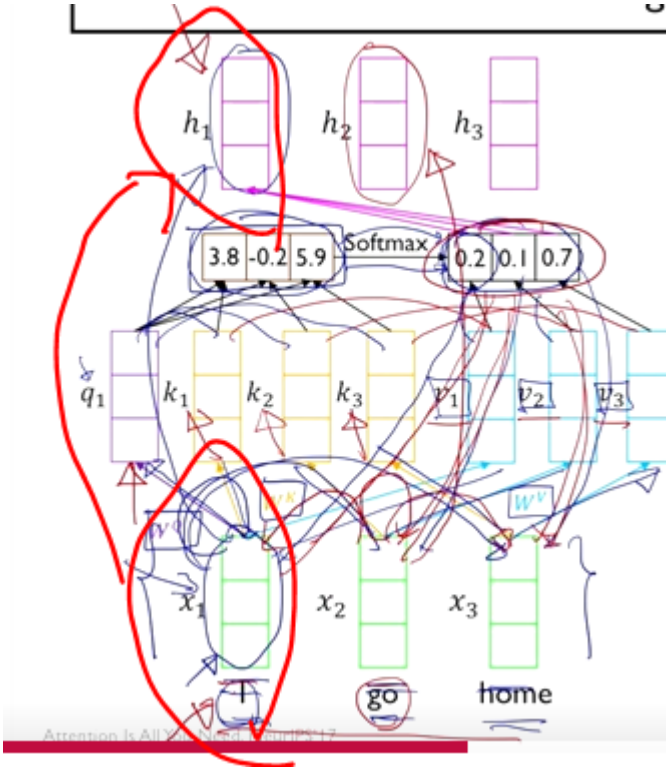


2*dv ⇒ 2 = Number of Query

첫번째 행 벡터: 첫번째 query vector을 사용했을 때의 가중치를 이용해 만들어진 가중평균의 최종 vector가 됨.

⇒ 해당 query 단어를 sequence를 반영한 여러 정보를 embedding 한 vector가 되는 것임.

⇒ 첫 단어가 query로 쓰였을 때 나타나는 hidden state vector 가 되는 것임.



즉, 각 선형변환(w_q, w_k, w_v)를 통해 각각의 단어들의 행 vector들을 하나의 행들로 가지는 행렬들이 있을 때, 해당하는 벡터들에 대해 softmax(q*k_T/sqrt(d_k))*v

이로서 전체 sequence를 encoding 했을 때 나타나는 hidden state vector을 나타낼 수 있게 된다.

이러한 self-attention module을 보다 다양한 형태로 확장한 것이 Multi-head Attention

주어진 sequence의 각각의 입력 vector들을 encoding 할 때, 어떤 특정한 W_q, W_k, W_v를 사용하면, 어떠한 선형 변환이 수행될 것일텐데,

특정 단어(e.g. 사람 이름)를 encoding할 때

그 단어를 encoding 할 때 이 사람이 어떤 행동을 했는가에 대한 정보를 위주로 encoding 할 경우가 있을 수도 있고,

문장상에서 그 사람이 그 행동을 했던 시간 정보를 위주로 할 수도 있고.

결국 다양한 형태로서 정보를 추출해볼 수 있을 것이다.

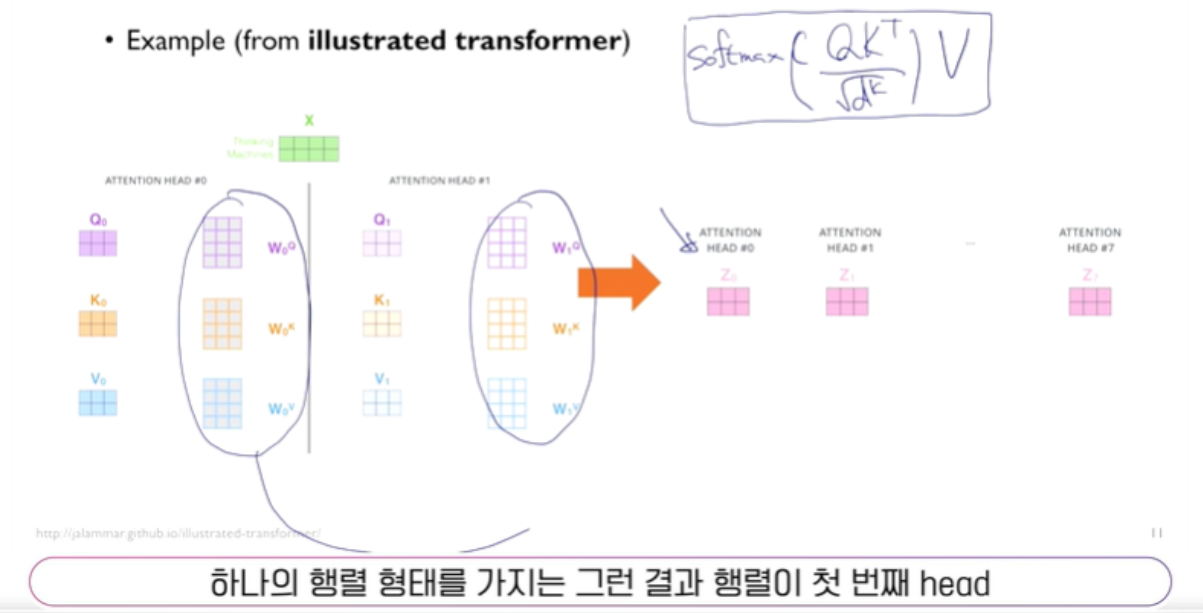
한 layer 내에서 self attention model을 sequence encoding을 위해 사용할 때, 어떤 기준으로 유사도 및 attention weight를 구하고 value vector도 어떤 기준으로 만들것인지에 대한 후보군을 여러개 두어,

각각의 set을 통해 변환된 q,k,v를 통해 만들어진 attention을 통해 sequence를 encoding하고,

그 다음에 각각의 set의 선형변환을 사용해 나온 attention model의 output을 나중에 concat하는 형태로 정보를 합쳐, 다양한 기준고 ㅏ다양한 정보를 바탕으로 sequence를 encoding하는 방식을 취할 수 있음.

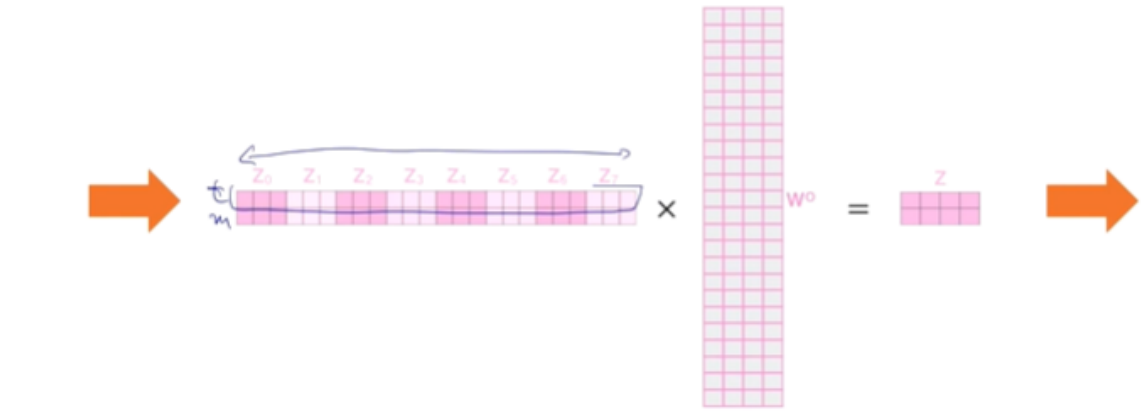
여러개의 transformation set을 활용 → 여러개의 'HEAD'를 사용

→ Multi-head Attention!



이러한 head의 개수(선형변환 set) 만큼 encoded hidden state vector가 나올 것.

• Example (from **illustrated transformer**)



다양한 기준(head)를 활용해, encoded vector가 나올 것임.

이 때, head수를 크게 하면 dimension은 크게 됨!!!!

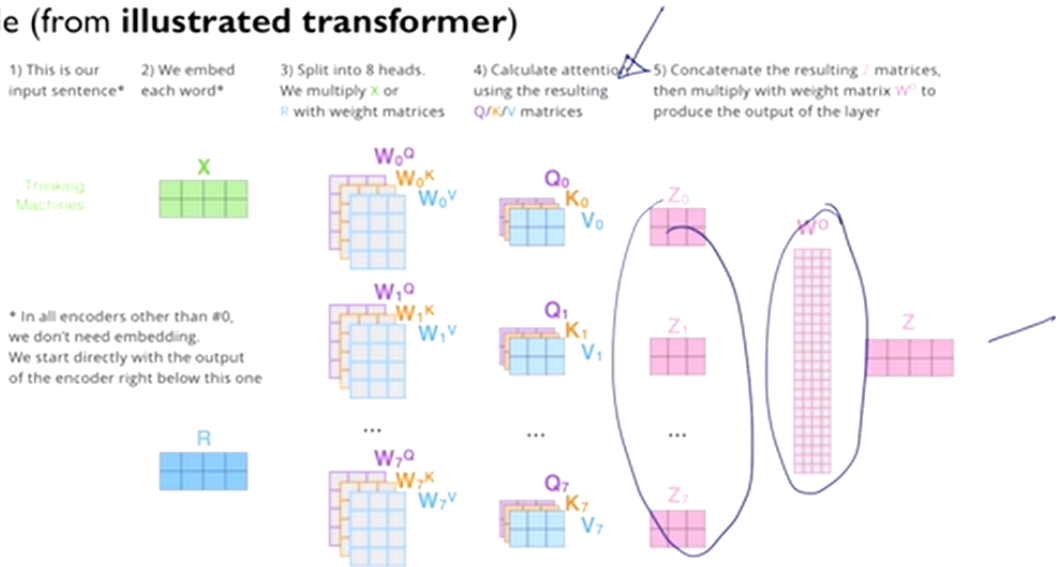
그때, encoding 하고 나서의 hidden state vector을 우리가 원하는 dimension으로 encoding 해주기 위해,

최종적으로 각각의 head별로 나온 attention model의 output vector을

추가적인 linear transformation을 통해 원하는 dimension으로 바꿔주게 된다.

이때 최종적인 output vector Z은, 원래 임베딩 vector의 크기와 같게 맞춰주어야 한다.

• Example (from **illustrated transformer**)



Self attention으로 인코딩 할 때에는, long term dependency 문제를 해결

→ 각 단어들이 query로 사용되어, 전체 sequence에 존재하는 각 단어들의 직접적인 유사도를 구하고

→ 유사도가 높은 단어를 단어의 time step과 관계 없이, 정보를 직접적으로 꺼내갈 수 있는 장점이있음.

하지만,

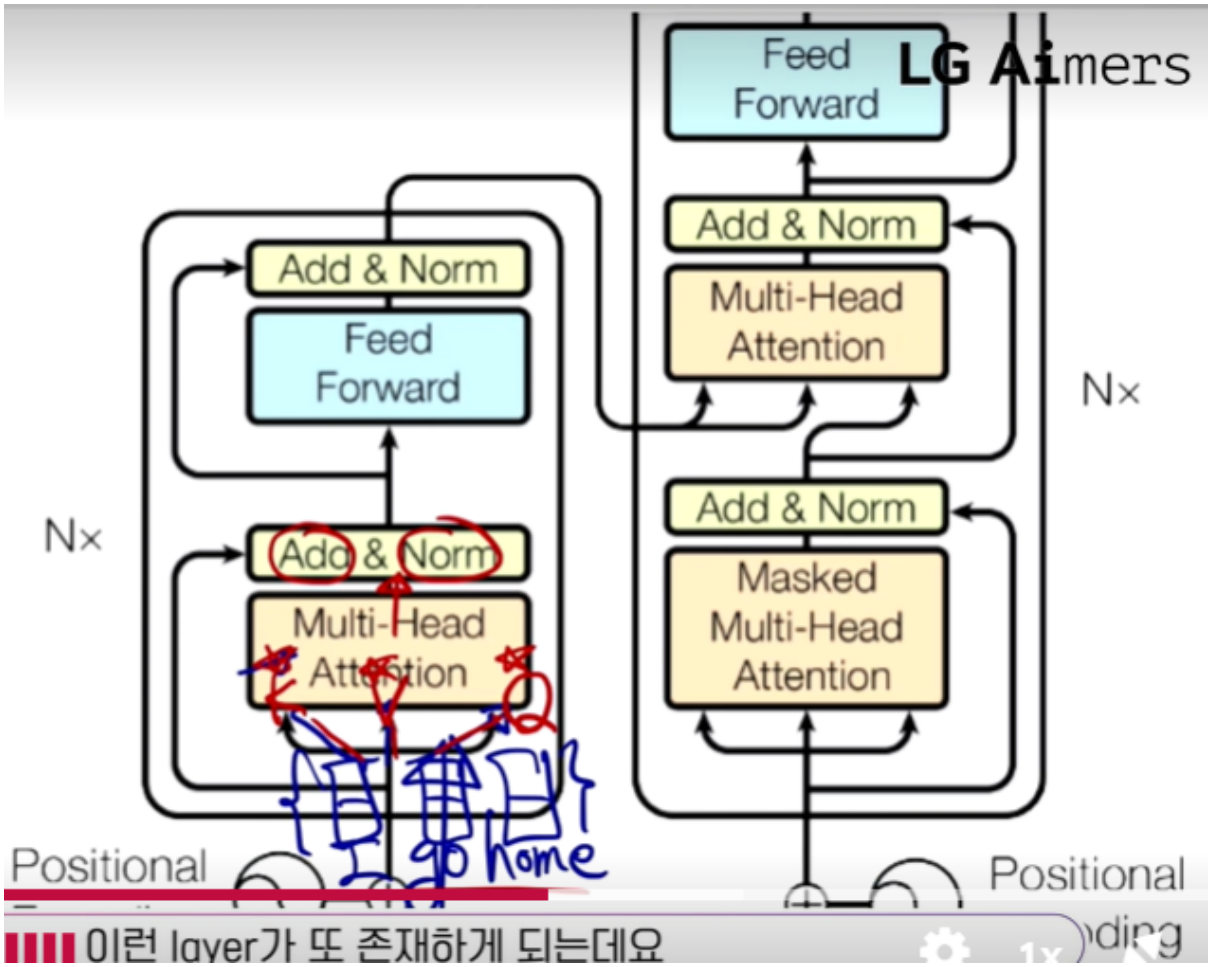
Memory 요구량이 굉장히 많이 들음.

Q*K_T 내적 연산 → 입력 sequence 길이가 늘어나면 늘어날 수록,

n^2만큼의 메모리가 필요함.

이러한 self-attention의 결과는, 임베딩 벡터임!!!!!

Tranformer 모델)



I go home에 대한 vector들이 2차원 vector라 하면,

multi-head attention을 통해 나오는 output vector는

각각의 단어에 대한 인코딩 된 벡터가 존재할 것이고,

각각의 output vector의 dimension은,

2차원 vector로서 주어지게 된다.

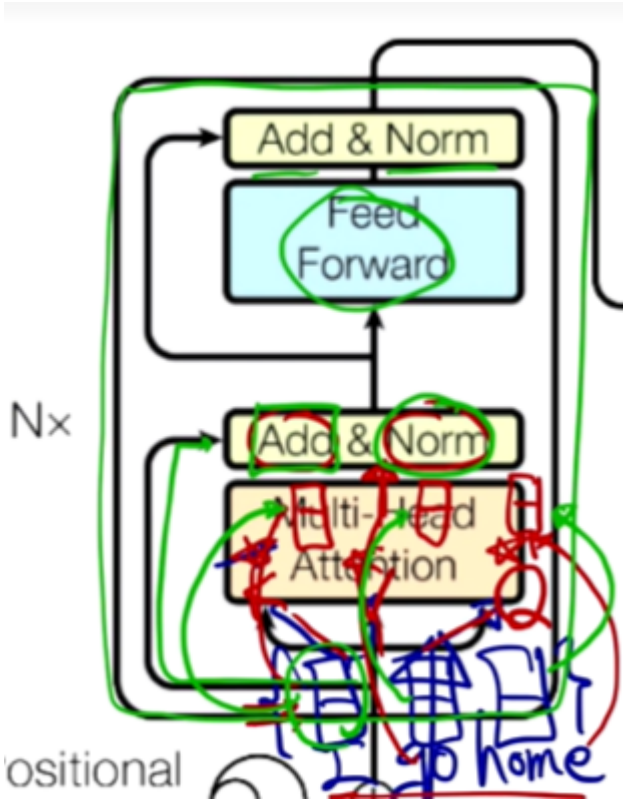
따라서, output vector은, input vector와 그 차원이 동일하다.

이 차원이 동일한 이유는,

residual network에서의 skip connection을 사용하기 위함임.(ADD)

인풋을 선형변환 거치기 전에 뺴와서, 나중에 output vector 나오면 더해준다.(Skip connection)

Norm → Layer Normalization Layer



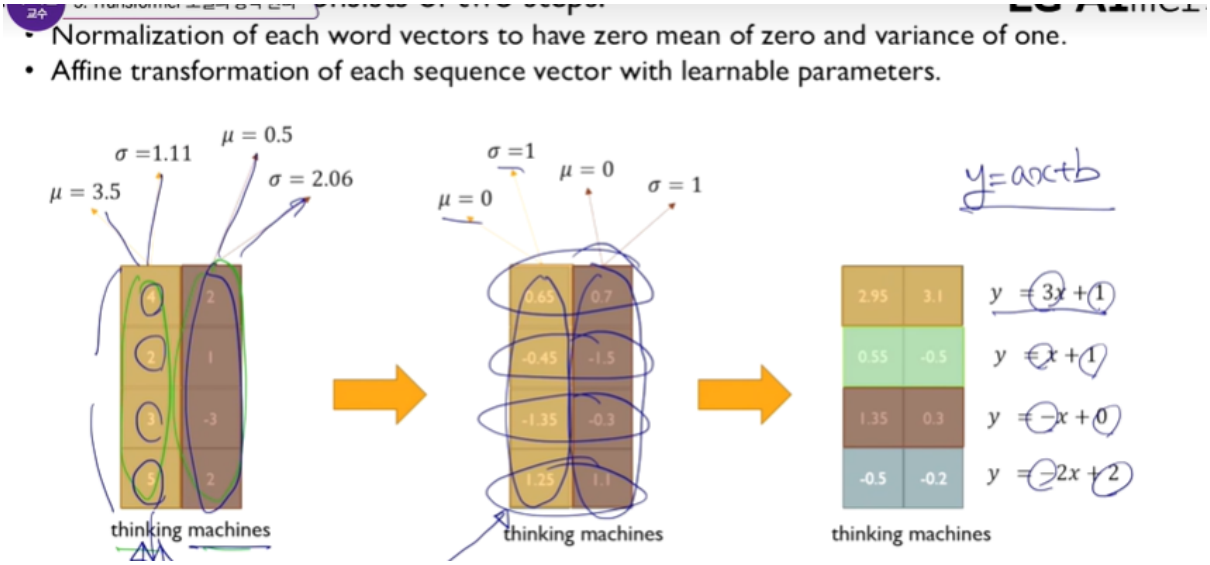
입력 vector를 출력 vector에 더해줌으로서,

각 단어들에 대한 최종적인 encoding vector가 얻어졌다면,

batch norm과 흡사한 layer normalization을 수행하게 됨.

1. 각 단어에서 발견된 특정한 dimension으로 이루어진 vector의 각 원소들을 모아서,
2. 한 단어 내에서 평균과 분산을 구하게 된다.
3. 각 단어 내에서 발생된 노드로 이루어진 vector의 각각의 원소값의 평균과 분산이 0과 1이 되도록 정규화를 진행하게 된다.(Batch Normalization)
4. 이후, affine transformation을 진행한다. 이 때 일차함수 y=ax+b에서의 파라미터(a,b)는 학습 가능한 존재이다

.



이러한 layer normalization을 통해, 학습을 안정시켜 주고, 성능을 향상시킴.

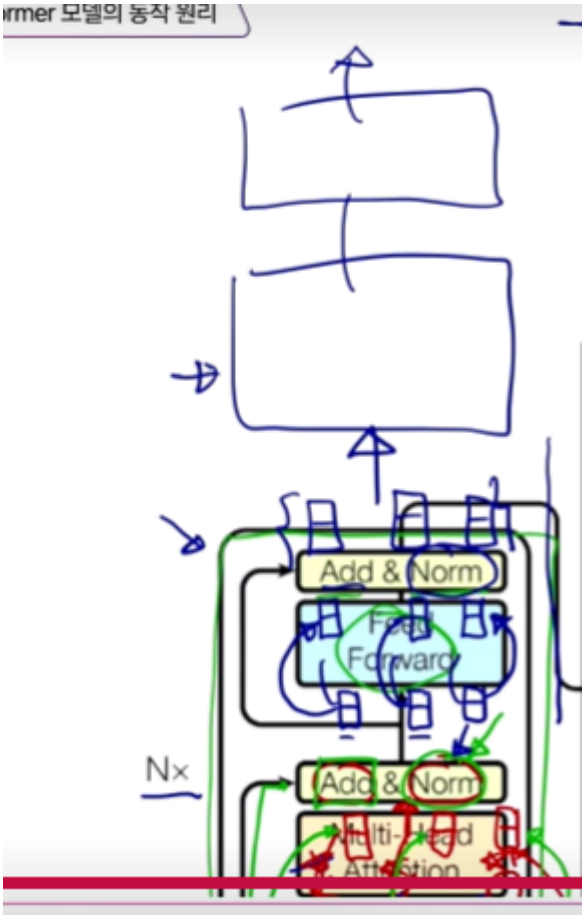
이후, Fully Connected layer을 거쳐, 또다른 2차원 vector로 선형변환에 의해 만들어지게 되며,

skip connection을 통해 입력 vector와 출력 vector을 각각의 time step에 대해 더해주게 된다.

추가적으로 layer normalization을 한번 더 거쳐서

최종적으로 각각의 단어별로 인코딩 된 hidden state vector을 얻게 된다.

N: 이러한 block 자체가 N번 구성되게 된다.



이렇게 하나의 block이 쌓이고 쌓이게 된다.

최종 output을 받아서, 그 다음 block의 input으로 들어가게 된다.

N번의 self attention block을 통해 encoding 된 각 단어별 embedding vector가 얻어지게 된다.

Positional Encoding

⇒ 가령, I go home이라는 특정한 순서로 세개의 단어를 input을 주었을 때, self attention을 통해 각 단어를 encoding하는 과정을 생각해 보면,

이 순서를 바꾸어서 동일한 self attention block을 통과하여 나오는 hidden state vector을 생각해 볼 때,

home이라는 동일 단어에 대한 query,key,value 벡터는 동일 할 것임.

Self attention module을 적용할 때,

벡터들의 순서는 상관없이, 가중평균을 구할 때, 덧셈은 교환법칙이 성립하기 때문에,

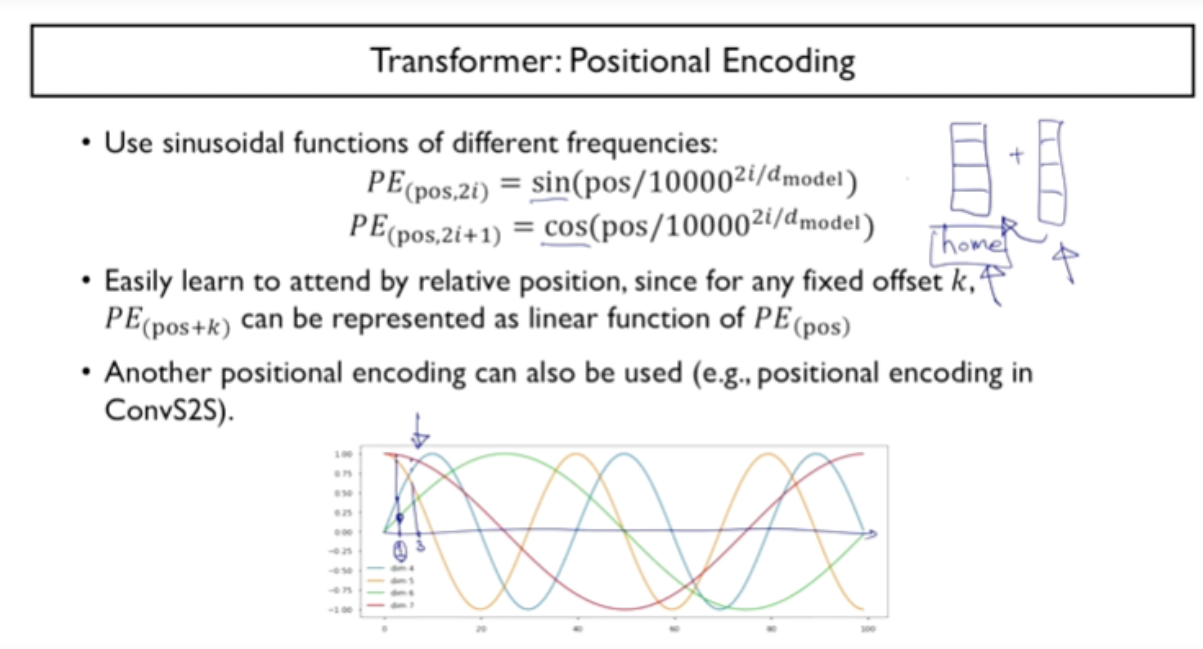
입력 vector의 순서는 home이라는 동일 단어를 encoding 할 때, 각 단어의 순서는 영향을 주지 못하게 된다.

따라서, 이러한 ‘sequence’를 구분하지 못하는 점을 보완하고자,

주어진 각 단어의 입력 vector에 이 단어가 지금 몇번째 순서로 나타났다라는 정보를 주입하게 되는데, 이것을 ‘position encoding’이라 한다.

ex) 입력 단어 vector의 dimension이 4개의 dimension이라고 생각해 보면,

sin t, cos t 에서 sequence t를 넣었을 때 분명, 같은 단어라고 하더라도 sin t와 cos t는 다른 값을 보일 것임. ⇒ 서로 다른 vector로 구분이 될 수 있음



Self-attention 기반의 sequence encoding이, 각 단어의 순서와 위치를 구별할 수 있게 되는 것임.

서로 다른 주파수 성분을 가진 sin, cos 등의 함수를 사용할 수 있겠지만, positioning encoding 자체도, 특정 task나 loss function에 기반하여 자체적으로 학습 및 최적화를 수행할 수도 있음.