


TRANSFORMER



딥러닝프레임워크

202184032 안신영, 2021 최정환



목차

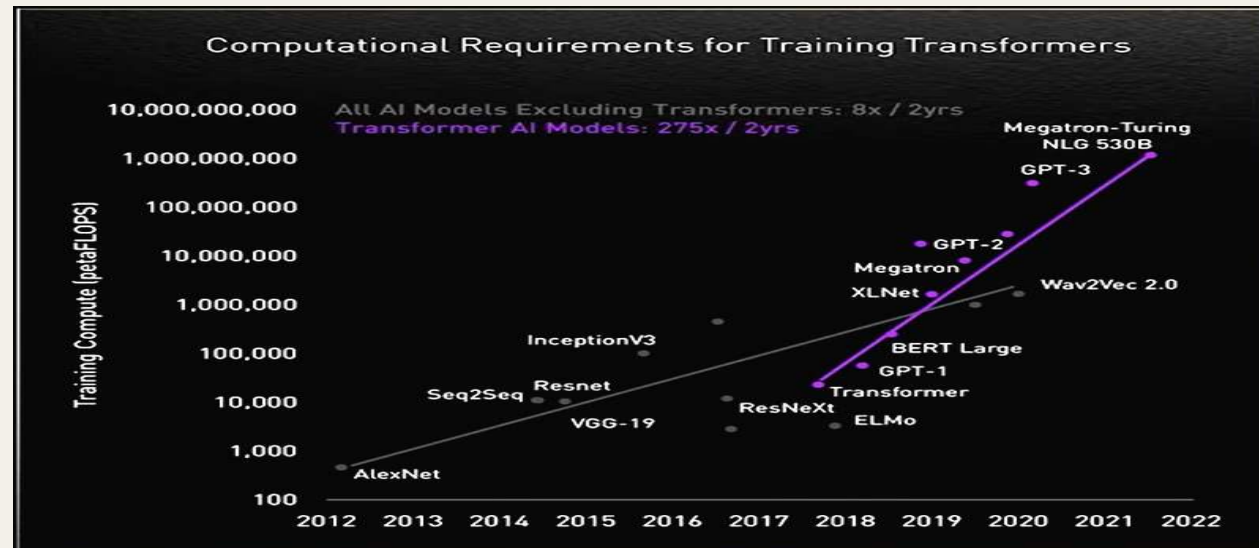
1. 서론
2. 이론적 배경
3. Transformer 구조 분석

1.서론

- Transformer란?

Transformer는 문장 속 단어와 같은 순차 데이터 내의 관계를 추적해 맥락과 의미를 학습하는 신경망 모델이다.

Transformer 모델은 RNN 계열의 모델 없이 오직 Attention만으로 이루어진 모델로, 연산 효율과 성능이 더욱 좋아졌고, 현재 지구상에서 가장 강력하다고 손꼽히는 모델 중 하나이다.



Transformer의 중요성

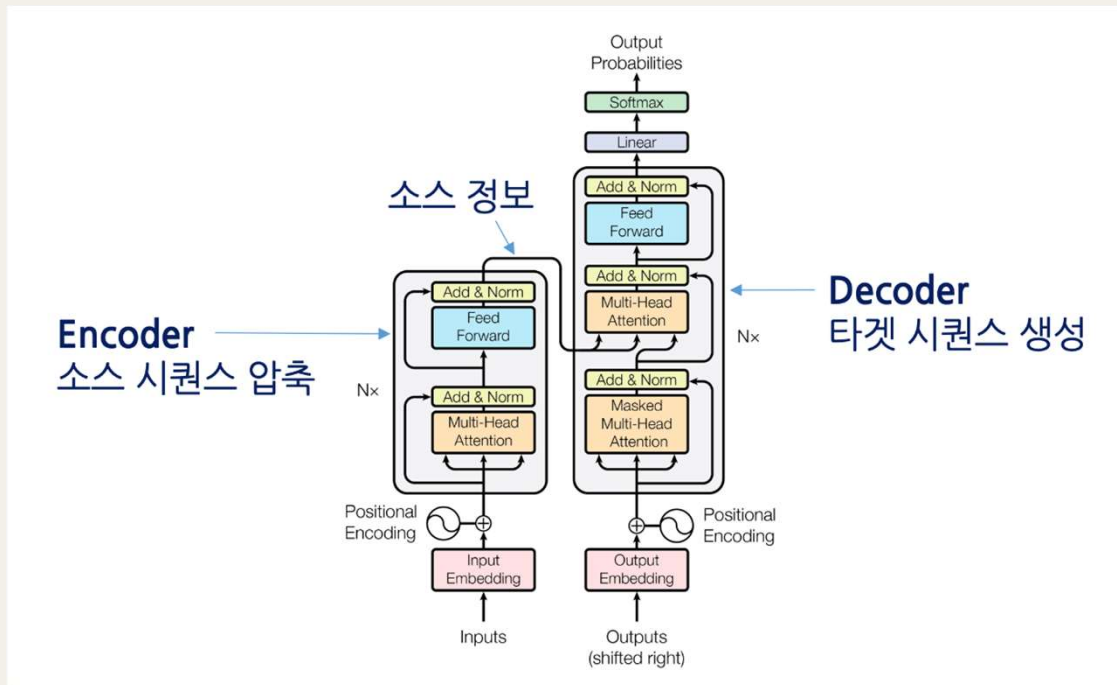
Transformer모델은 ChatGPT나 구글, 페이스북등이 내놓는 언어 기반 모델들이 기반으로 하고 있는 모델이다.

Transformer는 자연어처리 뿐만 아니라 컴퓨터 비전이나 음성 인식 등 다른 분야에도 적용되며 최고 수준의 성능을 기록할 것으로 기대되고 있다.

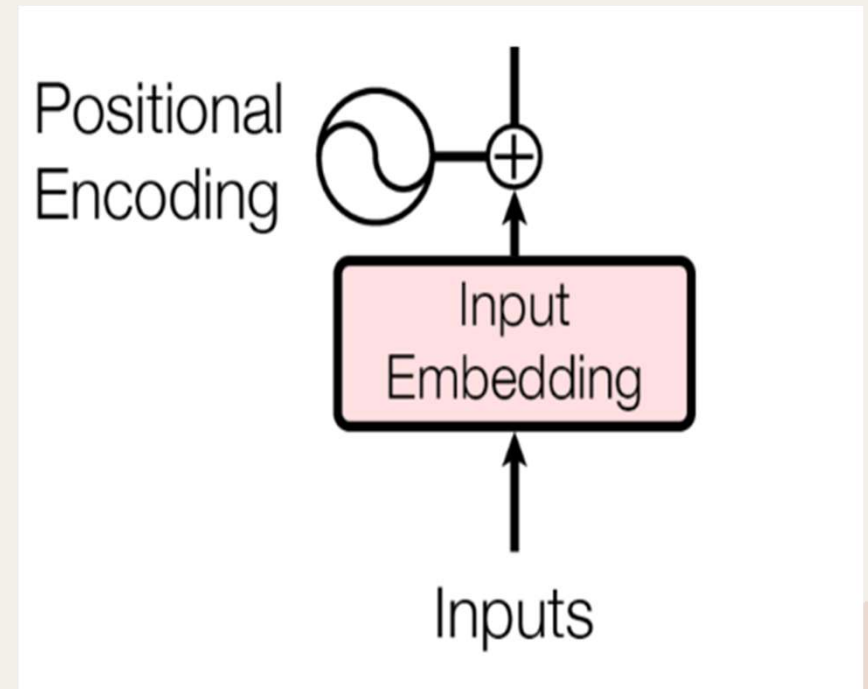
그 중에서도 이미지 처리 분야를 예를 들어보면 텍스트를 넣으면 이미지를 생성하는 모델인 Dalle2나 Stable Diffusion에도 트랜스포머가 활용되며, 2020년 발표된 Vision Transformer모델 또한 트랜스포머 모델을 기반으로 하여 현재 좋은 성적을 보이고 있다.

2. 이론적 배경

- Self-Attention 작동원리



Transformer 전체 구조



인코더 입력

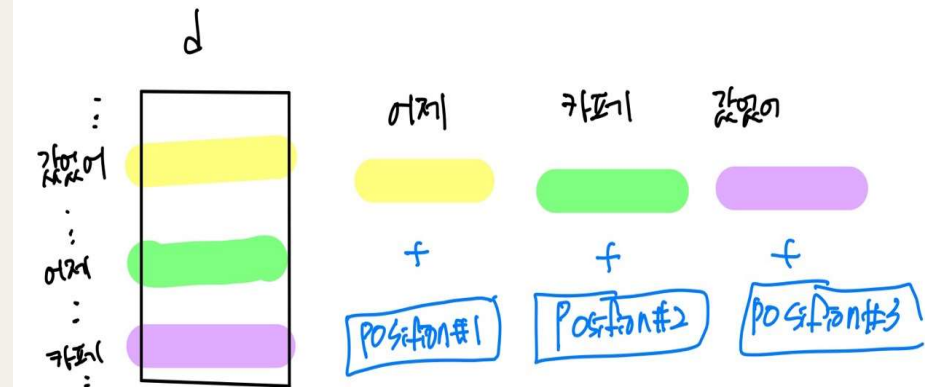
Self-Attention

작동원리

인코더 입력과 같이 모델 입력을 만드는 계층(layer)을 **입력층(input layer)**이라고 합니다.

인코더 입력에서 확인할 수 있듯이 인코더 입력은 소스 시퀀스의 입력 임베딩(input embedding)에 위치 정보(positional encoding)를 더해서 만듭니다. 한국어에서 영어로 기계 번역을 수행하는 트랜스포머 모델을 구축하다고 가정해 봅시다. 이때 인코더 입력은 소스 언어 문장의 토큰 인덱스(index) 시퀀스가 됩니다.

소스 언어의 토큰 시퀀스가 어제, 카페, 갔었어라면 인코더 입력층의 직접적인 입력값은 이들 토큰들에 대응하는 인덱스 시퀀스가 되며 인코더 입력은 다음 그림과 같은 방식으로 만들어집니다. 다음 그림은 이해를 돕고자 토큰 인덱스(어제의 고유 ID) 대신 토큰(어제)으로 표기했습니다.



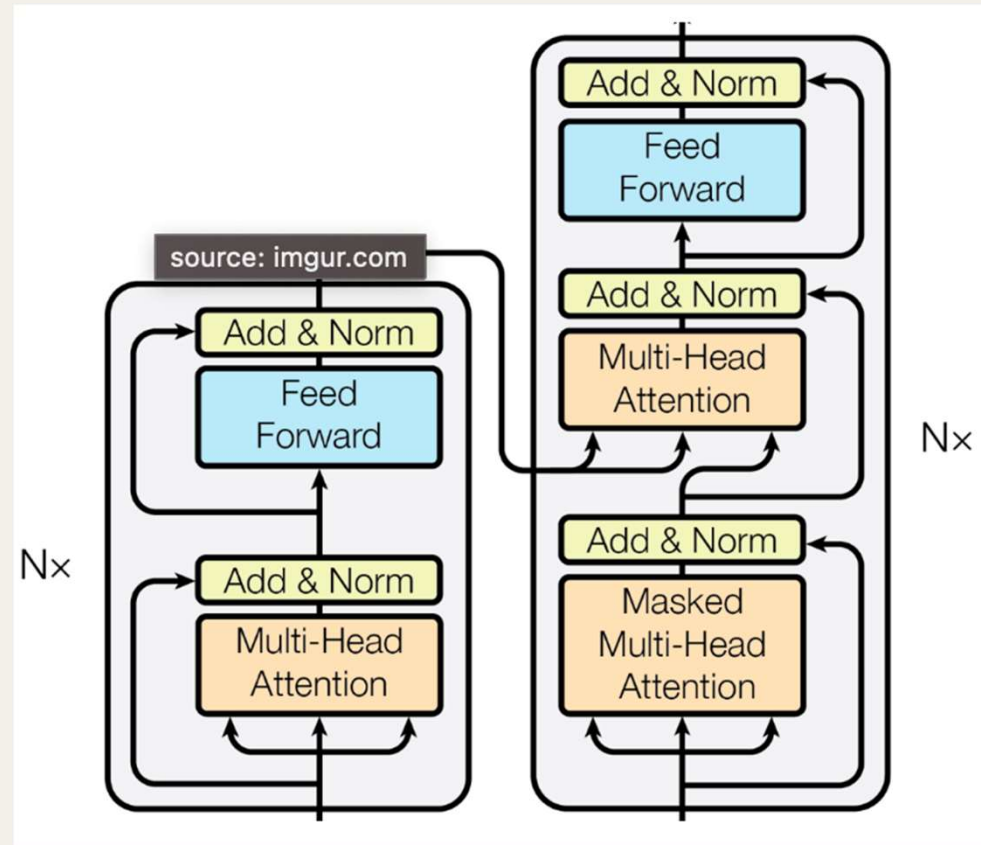
Self-Attention 작동원리

원편 행렬(matrix)은 소스 언어의 각 어휘에 대응하는 단어 수준 임베딩인데요. 단어 수준 임베딩 행렬에서 현재 입력의 각 토큰 인덱스에 대응하는 벡터를 참조(lookup)해 가져온 것이 인코더 입력의 입력 임베딩(input embedding)입니다. 단어 수준 임베딩은 트랜스포머의 다른 요소들처럼 소스 언어를 타겟 언어로 번역하는 태스크를 수행하는 과정에서 같이 업데이트(학습)됩니다.

입력 임베딩에 더하는 위치 정보는 해당 토큰이 문장 내에서 몇 번째 위치인지 정보를 나타냅니다. 작동원리 그림 예시에서는 어제가 첫번째, 카페가 두번째, 갔었어가 세번째입니다.

트랜스포머 모델은 이같은 방식으로 소스 언어의 토큰 시퀀스를 이에 대응하는 벡터 시퀀스로 변환해 인코더 입력을 만듭니다. 디코더 입력 역시 만드는 방식이 거의 같습니다.

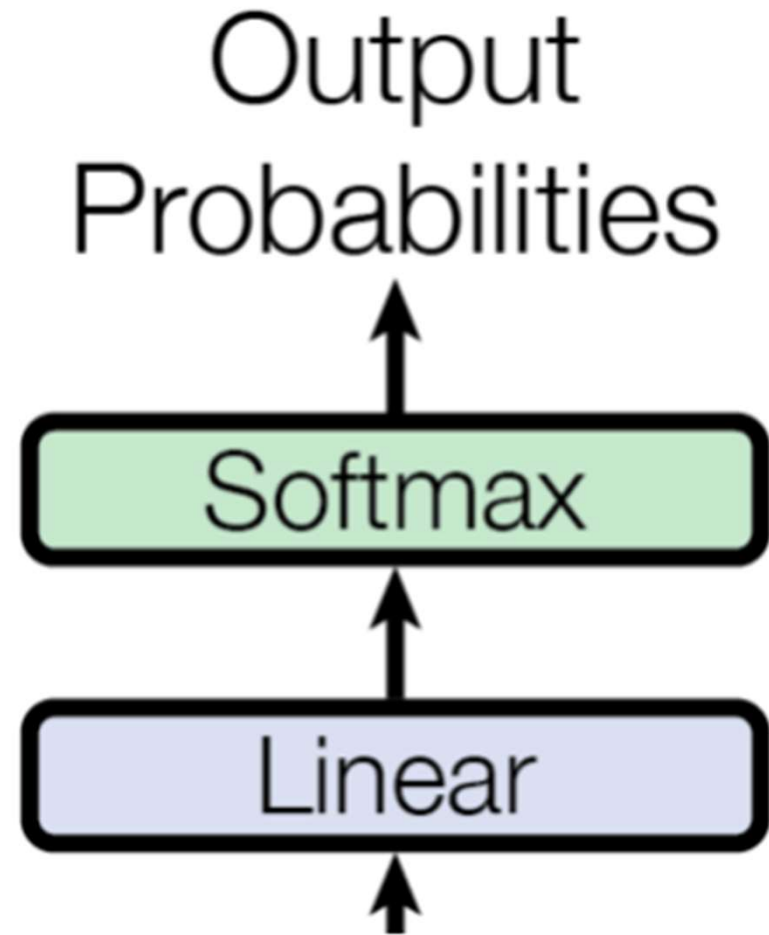
인코더 - 디코더는 Transformer 전체구조에서 인코더와 디코더 블록만을 떼어 그린 그림인데요. 인코더 입력층에서 만들어진 벡터 시퀀스가 최초 인코더 블록의 입력이 되며, 그 출력 벡터 시퀀스가 두 번째 인코더 블록의 입력이 됩니다. 다음 인코더 블록의 입력은 이전 블록의 출력입니다. 이를 NN 번 반복합니다.



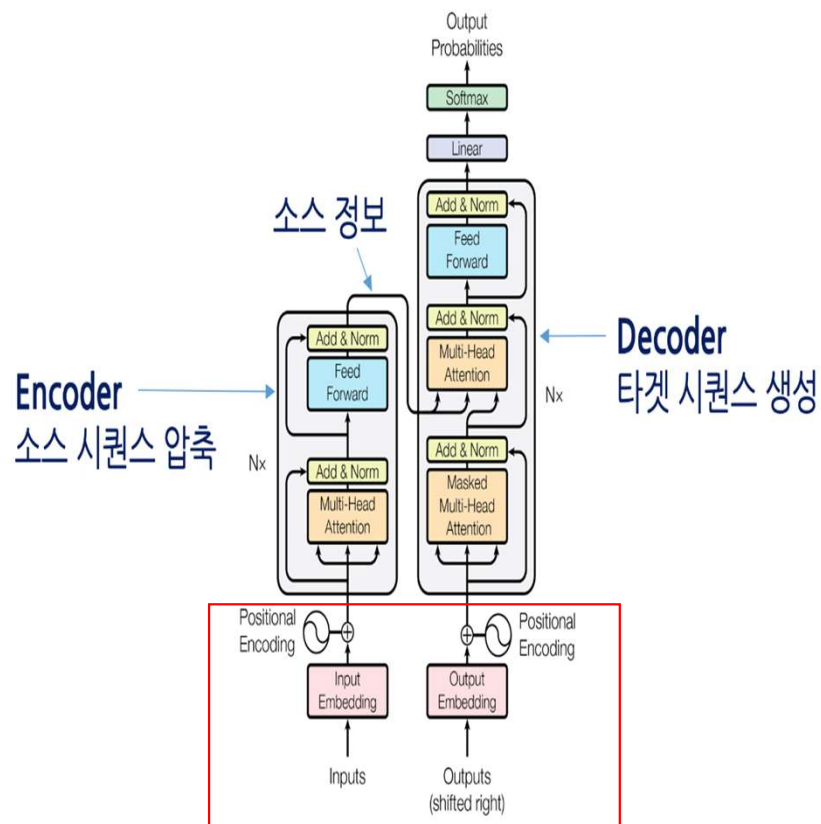
Self-Attention 작동원리

트랜스포머의 전체 구조에서 모델의 **출력층(output layer)**만을 떼어낸 것입니다. 이 출력층의 입력은 디코더 마지막 블록의 출력 벡터 시퀀스입니다. 출력층의 출력은 타겟 언어의 어휘 수만큼의 차원을 갖는 확률 벡터가 됩니다. 소스 언어의 어휘가 총 3만개라고 가정하면 이 벡터의 차원수는 3만이 되며 3만 개 요소값을 모두 더하면 그 합은 1이 됩니다. 이 벡터는 디코더에 입력된 타겟 시퀀스의 다음 토큰 확률 분포를 가리킵니다.

트랜스포머의 학습(train)은 인코더와 디코더 입력이 주어졌을 때 모델 최종 출력에서 정답에 해당하는 단어의 확률 값을 높이는 방식으로 수행됩니다.



Positional Encoding

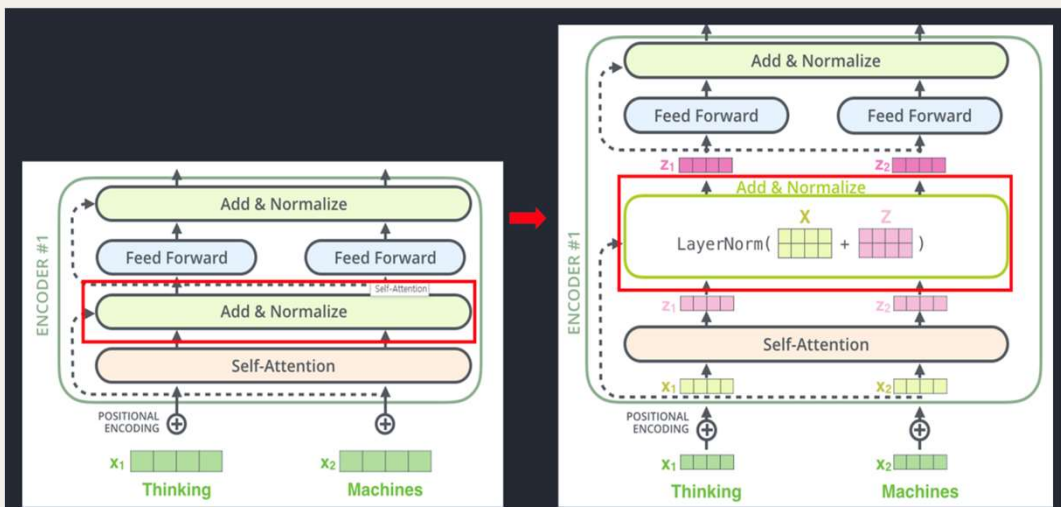


트랜스포머는 입력된 데이터를 한번에 병렬로 처리해서 속도가 빠르다는 장점이 있다. 하지만 RNN과 LSTM과 다르게 트랜스포머는 입력 순서가 단어 순서에 대한 정보를 보장하지 않는다. 다시 말하면, 트랜스포머의 경우 시퀀스가 한번에 병렬로 입력되기에 단어 순서에 대한 정보가 사라진다. 따라서 단어 위치, 정보를 별도로 넣어줘야한다.

각각의 단어 벡터에 Positional Encoding을 통해 얻은 위치정보를 더해줘야된다. 이때 반드시 지켜야 될 규칙 두 가지가 있다.

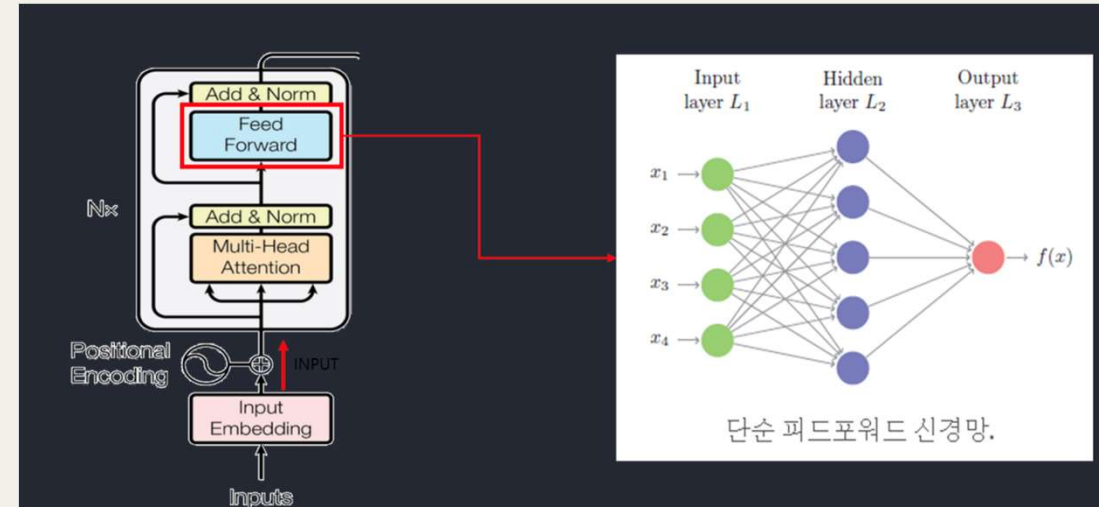
1. 모든 위치값은 시퀀스의 길이나 input에 관계없이 동일한 식별자를 가져야한다. 따라서 시퀀스가 변경되더라도 위치 임베딩은 동일하게 유지될 수 있다.
2. 모든 위치값은 너무 크면 안된다. 위치값이 너무 커져버리면, 단어 간의 상관관계 및 의미를 유추할 수 있는 의미 정보 값이 상대적으로 작아지게 되고, Attention layer에서 제대로 학습 및 훈련이 되지 않을 수 있다.

3. Transformer 구조 분석(Encoder)



Encoder - Add & Normalize(계층정규화)

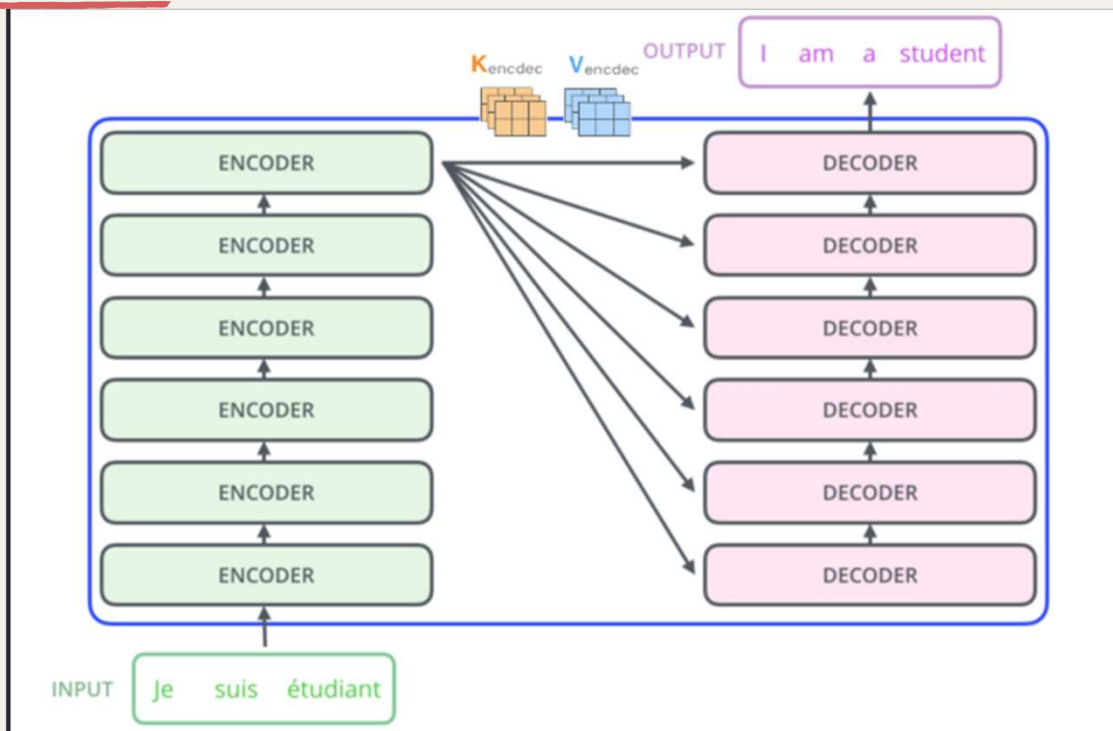
이 과정은 Residual Connection이라고도 불리고 모델 학습을 돕기 위해 Vision분야에서 자주 사용하는 기법입니다. Encoder에서 Attention에서 나온 출력값은 Normalize과정을 거치게 됩니다. 계층 정규화 과정은 Self-Attention과정 전, 후 값인 입력값(X)과 출력값(Z)을 합(sum)함으로써 한 계층에서 과도하게 데이터가 변경되는 부분을 방지할 수 있습니다.



Encoder - Feed_Forward

Add & Norm을 거친 결과는 Feed-Forward 네트워크를 통과하게 됩니다. Feed-Forward 네트워크는 2개의 MLP 레이어와 ReLU 함수로 구성되어 있는 신경망 레이어로써 가중치를 학습하는 공간이라고 생각하면 됩니다.

3. Transformer 구조 분석(Decoder)



Encoder과정은 입력과 출력을 N번 반복해서 실행하게 됩니다.

마지막으로 나온 출력값 K,V는 Decoder의 입력값으로 각각 들어가게 됩니다.

Decoder도 Encoder의 입력값처럼 Embedding, Positional Encoding과정을 거치게 됩니다. 하지만 차이점이 있는데, Encoder의 입력값으로는 문장의 토큰이 한번에 들어왔지만 Decoder의 입력값은 RNN입력처럼 오른쪽으로 이동하면서 단어가 하나씩 입력되는 형식입니다. 입력값의 마지막은 eos(end of Sentence)로 처리해줍니다.

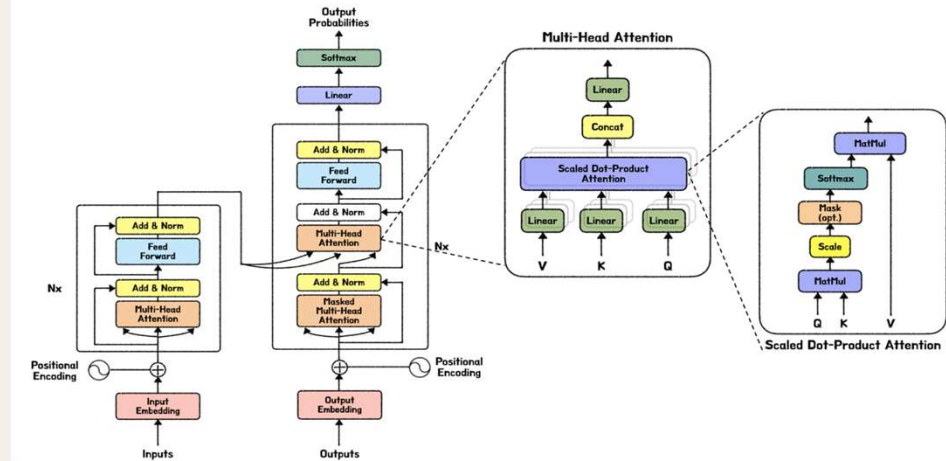
Multi-Head Attention

멀티-헤드 어텐션(Multi-Head Attention)은 **셀프 어텐션(self attention)**을 여러 번 수행한 걸 가리킵니다. 여러 헤드가 독자적으로 셀프 어텐션을 계산한다는 이야기입니다. 비유하자면 같은 문서(입력)를 두고 독자(헤드) 여러 명이 함께 읽는 구조라 할 수 있겠습니다.

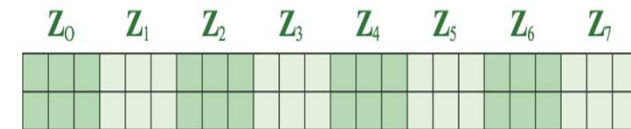
멀티헤드어텐션은 입력 단어 수는 2개, 밸류의 차원수는 3, 헤드는 8개인 멀티-헤드 어텐션을 나타낸 그림입니다. 개별 헤드의 셀프 어텐션 수행 결과는 '입력 단어 수 \times 밸류 차원수', 즉 $2 \times 32 \times 3$ 크기를 갖는 행렬입니다. 8개 헤드의 셀프 어텐션 수행 결과를 다음 그림의 ①처럼 이어 붙이면 $2 \times 242 \times 24$ 의 행렬이 됩니다.

멀티-헤드 어텐션은 개별 헤드의 셀프 어텐션 수행 결과를 이어붙인 행렬(①)에 $\mathbf{W}^O\mathbf{W}^O$ 를 행렬곱해서 마무리됩니다. $\mathbf{W}^O\mathbf{W}^O$ 의 크기는 '셀프 어텐션 수행 결과 행렬의 열(column)의 수 \times 목표 차원수'가 됩니다. 만일 멀티-헤드 어텐션 수행 결과를 그림9와 같이 4차원으로 설정해 두고 싶다면 $\mathbf{W}^O\mathbf{W}^O$ 는 $24 \times 424 \times 4$ 크기의 행렬이 되어야 합니다.

멀티-헤드 어텐션의 최종 수행 결과는 '입력 단어 수 \times 목표 차원수'입니다. 그림9에서는 입력 단어 두 개 각각에 대해 3차원짜리 벡터가 멀티-헤드 어텐션의 최종 결과물로 도출되었습니다. 멀티 헤드 어텐션은 인코더, 디코더 블록 모두에 적용됩니다.



① 모든 헤드의 셀프 어텐션 출력 결과를 이어 붙인다.



② ①의 결과로 도출된 행렬에 \mathbf{W}^O 를 곱한다. 이 행렬은 개별 헤드의 셀프 어텐션 관련 다른 행렬(\mathbf{W}^Q , \mathbf{W}^K , \mathbf{W}^V)과 마찬가지로 태스크(기계 번역)를 가장 잘 수행하는 방향으로 업데이트된다.



③ 새롭게 도출된 \mathbf{Z} 행렬은 동일한 입력(문서)에 대해 각각의 헤드가 분석한 결과의 총합이다.

$$\mathbf{Z} = \begin{bmatrix} \text{ } & \text{ } & \text{ } \\ \text{ } & \text{ } & \text{ } \end{bmatrix}$$

Residual Connection(잔차연결)

Residual Connection은 ResNet에서 등장한 개념입니다. 과거 GoogleNet, VGG 등으로 Deep CNN에 대한 연구가 진행되었으나 깊이가 깊어질 수록 학습이 원활하게 잘 되지 않고 있음을 확인했습니다. 즉, 오히려 **shallow한 network** 구조가 **deep depth**를 가지는 **모델보다 좋은 성능을 발휘**하게 되는 것입니다.

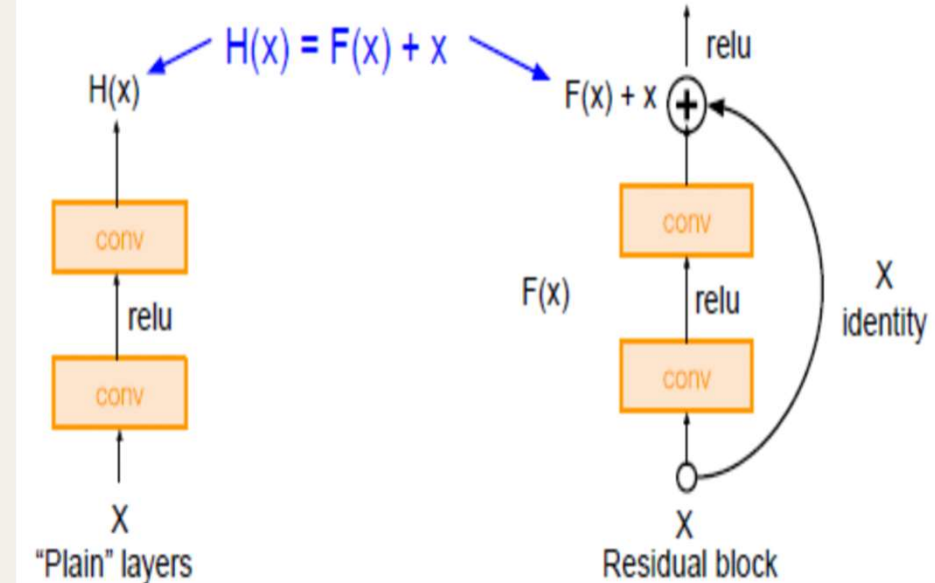
기본적으로 parameter 숫자가 많게 되면 feature space가 복잡해지기 때문에 overfitting이 잘 일어나게 됩니다. 이러한 특성 때문에 Deep CNN 역시 네트워크가 커짐에 따라서 점점 학습이 안 되는 것이었습니다. 이를 해결하고자 한 것이 ResNet입니다.

왼쪽은 기존에 사용하던 구조이며 오른쪽은 ResNet에서 사용되는 것입니다. 연산은 동일하게 진행하지만 **차이점은 Input X를 Activation Function을 거치기 이전에 더해주느냐 그렇지 않느냐**입니다.

기존의 layer는 input X를 넣어주고 function을 통해 나온 \hat{y} 는 이후의 layer에서 새롭게 학습해야 할 정보입니다. 즉, input X에 대한 내용이 존재하지 않고 오직 \hat{y} 를 통해서 반복적으로 학습이 일어나게 되는 것입니다. 즉, $H(x) = \hat{y}$ 가 우리의 target 값인 Y에 가까워지도록 학습을 해야 하는데 이렇게 되면 학습해야 하는 내용이 많아지게 됩니다.

특정 위치의 출력은 특정 위치에서의 **입력과 residual 함수의 합으로 표현이 가능해지기 때문에 학습구조가 매우 단순화**됩니다.

즉, $F(x)$ 가 0이 되도록 학습을 진행해주게 되는데(진짜 0이 되지는 못함) x와 함께 갈 약간의 작은 변화만을 갖고 가게 됩니다. 결과적으로 **작은 잔차만을 학습하기 때문에 Residual connection**이라 말합니다.



Layer Normaliztion(층 정규화)

Layer Normalization은 data sample 단위로 평균(mean)과 표준편차(std)를 계산해서 정규화를 실행한다.

특성의 개수와 상관없이 batch 내부의 데이터 개수가 3개이기 때문에, 3개의 데이터 샘플에 3개의 평균, 3개의 표준편차 값을 계산하여 Layer Normalization을 실행한다.

장점

1. Batch Normalization은 작은 배치 크기에서 극단적 결과를 내는데 반해, 작은 batch size에서도 효과적인 이용이 가능
2. sequence에 따른 고정길이 정규화로 batch normalization에 비해 RNN 모델에 더 효과적인 방법이다.
3. 일반화 성능 향상이 가능하다.

단점

1. 추가 계산 및 메모리 오버헤드가 발생할 수 있다.
2. 반대로 피드포워드 네트워크 모델(input - hidden - output 순서로 한방향으로 흐르는 인공신경망 모델, mnist 0~9 손글씨 분류 등)에서는 Batch Normalization 만큼 잘 동작하지 않을 수 있다.
3. learning_rate, 가중치 초기화 같은 하이퍼파라미터 조정에 민감할 수 있다.

Layer Normalization

