Transformer

PT

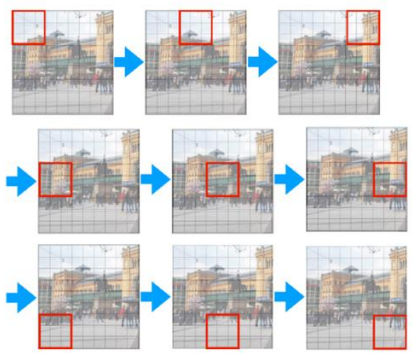
1. BAM: Bottleneck Attention module
2. Attention - Key, Query, Value
3. Attention Is All You Need
4. Residual Connection

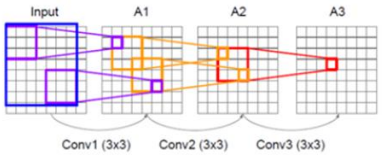
DELAB 2

트랜스포머를 처음 제안한 논문은 3번에 보이는 Attention is all you need이다. 그런데 필자는 비전에 관심이 많기 때문에 Attention 개념을 비전 관점에서 먼저 확인하고자 하였다. 따라서 Attention을 사용하는 BAM 논문을 간단하게 리뷰하고 Attention 개념을 설명한 뒤 트랜스포머를 리뷰하려고 한다. 또한 트랜스포머 뿐 만 아니라 다양한 아키텍쳐에서 쓰이는 Residual Connection에 대해 간단히 조사해보았다.

BAM: Bottleneck Attention module

- CNN의 단점
  - 이미지의 국소적인 부분만 확인할 수 있다
  - Convolution 층이 깊어지면 전체적인 부분도 간접적으로 볼 수는 있으나 초기 레이어에서 직접적으로 이미지의 다른 부분을 확인할 수는 없음





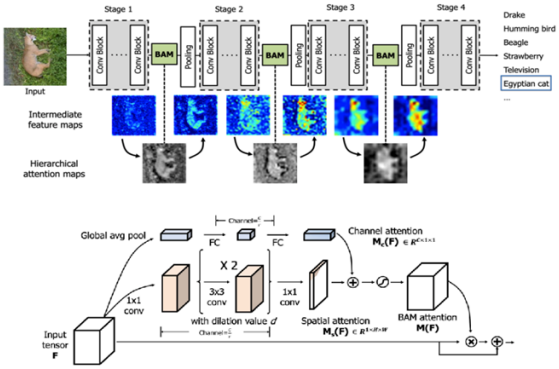
DELAB 3

먼저 Attention을 말하기 전에 CNN의 단점부터 알아보겠다.

CNN은 이미지의 국소적인 부분만 확인할 수 있다. 즉 전체를 고려해서 계산할 수 없다. 물론 오른쪽에 보이는 구조처럼 CNN 층이 깊어지면 전체적인 부분도 간접적으로 보지만, 초기 레이어부터 직접적으로 이미지의 다른 부분을 확인할 수 없다는 한계점은 항상 지닌다. 다시 말해, 처음부터 이미지 상 어느 부분이 중요한지는 확인할 수 없다. 그래서 3x3 필터로 Convolution을 진행할 때 이미지의 특정 부분만 확인할 수 있다는 것이다.

BAM: Bottleneck Attention module

- Attention
  - 이미지 전체에서 어느 부분을 주의 깊게 봐야 되는지를 나타내는 기술
  - Feature Map과 Attention Map을 연산한 결과를 Conv의 입력으로 사용
  - Convolution을 거치기 전에 이미지의 중요한 부분을 선택할 수 있음



$$F' = F + F * M(F)$$

DELAB 4

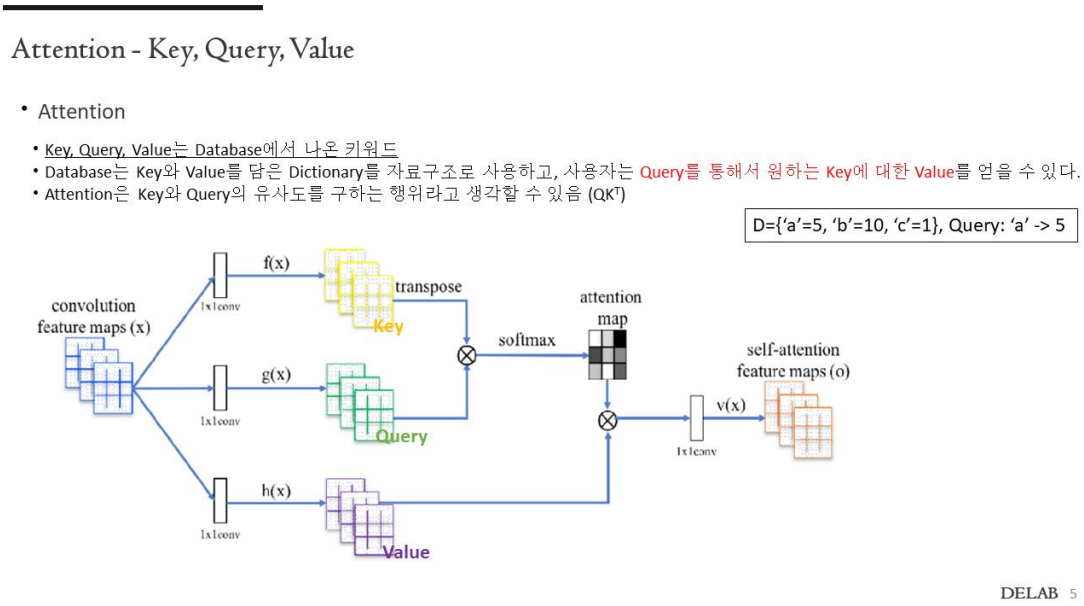
반면 Attention은 이미지를 전체적으로 확인할 수 있다. Attention을 비전 관점으로 정의하면 '이미지 전체에서 어느 부분을 주의 깊게 봐야 되는지를 나타내는 기술'이다. BAM이라는 논문에서 Attention한 결과를 시각화해서 잘 보여주고 있는데 ppt 그림을 보면 Conv Block의 결과가 다시 Conv Block의 입력으로 들어가기 전 attention map과 연산을 하는 것을 볼 수 있다. feature map을 시각화한 그림을 살펴보면 attention map과 연산을 하니 이미지에서 주의 깊게 살펴봐야 할 객체가 더 진하게 강조 처리된 것을 볼 수가 있다. 여기서 중요한 건 어떻게 attention map을 만들었는지와 어떻게 연산을 하는지이다.

attention map을 만들 때는 channel attention map과 spatial attention map을 먼저 만들어야 한다. channel attention map은 GAP와 FC를 통해 [채널 수가 같고 resolution이 1x1인 feature map]을 의미한다. spatial attention map은 여러 Conv Net을 거치면서 [채널 수가 1이고 resolution이 같은 feature map]을 의미한다.

이후 channel attention map(1x1Xc)와 spatial attention map(RxWx1)으로 element-wise 합을 구하고 activation function을 통과한 결과가 attention map(RxWxC)이다. 이 때 attention map의 모든 값은 0과 1사이의 값이 된다. 이 attention map이 원래의 input과 곱해지면서 중요하지 않은 부분은 0에 가까운 숫자와 곱해지고 중요한 부분은 1에 가까운 숫자와 곱해지면서 attention 기법이 적용되는 것이다.

연산은 F'=F+FxM(F) 식으로 이루어지는데 F를 더하는 것은 skip connection으로 원본의 정보도 취하는 것을 확인할 수 있다.

그래서 기존의 CNN이 Convolution 연산에서 이미지의 중요한 부분을 선택했다면,
위 방법은 Convolution을 거치기 전에도 이미지의 중요한 부분을 선택하는 것이다.



key, quey, value는 Database에서 나온 키워드이다. Database는 key와 value를 담는 Dictionary를 자료구조로 사용하고 사용자는 query를 통해서 원하는 key에 대한 value값을 얻을 수 있다. 예를 들어 d={'a'=5, 'b'=10, 'c'=1}인 dictionary가 있을 때, 'a'라는 query가 주어지면 5가 결과로 나오는 것이다. 즉 query가 주어지면 Database에서 그것에 맞는 key를 찾고 value를 반환하는 구조를 지닌다. 즉 Query를 통해서 원하는 Key에 대한 Value를 얻는 개념이 Attention이라고 볼 수 있고, 이러한 개념은 key와 query의 유사도를 구하는 행위라고 생각할 수도 있다. query(요청된 정보)가 key에 있는지 없는지를 확인하는 과정이기 때문에 key와 query의 유사도를 체크하는 것과 같은 행위이다.

그림을 보면, f(x), g(x), h(x)함수가 있는 데, 이 함수들은 각각 key(K), query(Q), value(V)를 산출하는 함수이다. 그리고 우리가 원하는 task는 주어진 input의 feature map에서 중요하다고 생각되는 값만 불러오는 것이다. 따라서 key와 query의 유사도를 구하기 위해 QxK^T를 계산(유사도는 내적으로 계산함)하고 softmax를 적용해서 모든 값이 0과 1로만 이루어지게 한다. 이 결과가 attention map이다.

(요청된 정보가 key에 있는지 없는지를 확인하고 난 결과로, 요청된 정보가 key에 있으면(유효하면) 1, 유효하지 않으면 0이 됨)

이후 attention map을 value에 곱함으로써 value에서 어느 부분이 중요한지 덜 중요한지를 나타낼

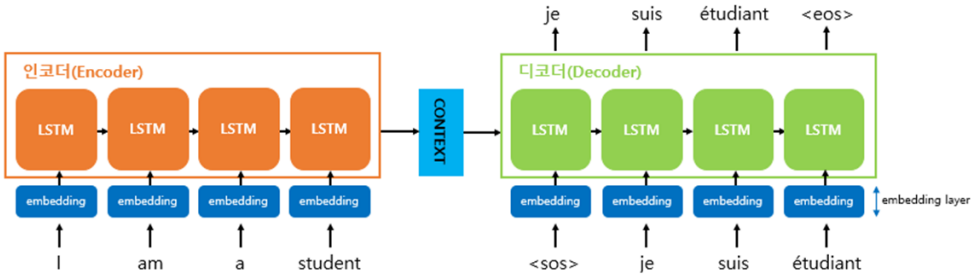
수 있겠고 마지막으로 1x1 conv를 한 결과가 Attention을 고려해서 만들어진 값이라고 할 수 있다.

(필자의 생각)

QK^T를 통해 [query가 요청하는 중요한 키 중 유효한 키]를 얻어낼 수 있고, V를 곱함으로써 [query가 요청하는 키에 대응되는 중요한 정보들 중 유효한 정보]를 얻어낼 수 있다.

Attention Is All You Need

- LSTM의 한계
  - LSTM: 장기 문맥을 기억할 수 있는 RNN (시계열 데이터를 다루는데 최적화된 인공신경망)
  - 문장의 의미와 단어 간의 관계가 항상 순차적이지는 않다 -> 문장의 길이가 길어지면 순서만을 가지고 제대로 된 번역이 불가
  - 어텐션 개념을 가지고 전체 문장을 고려할 필요가 있음



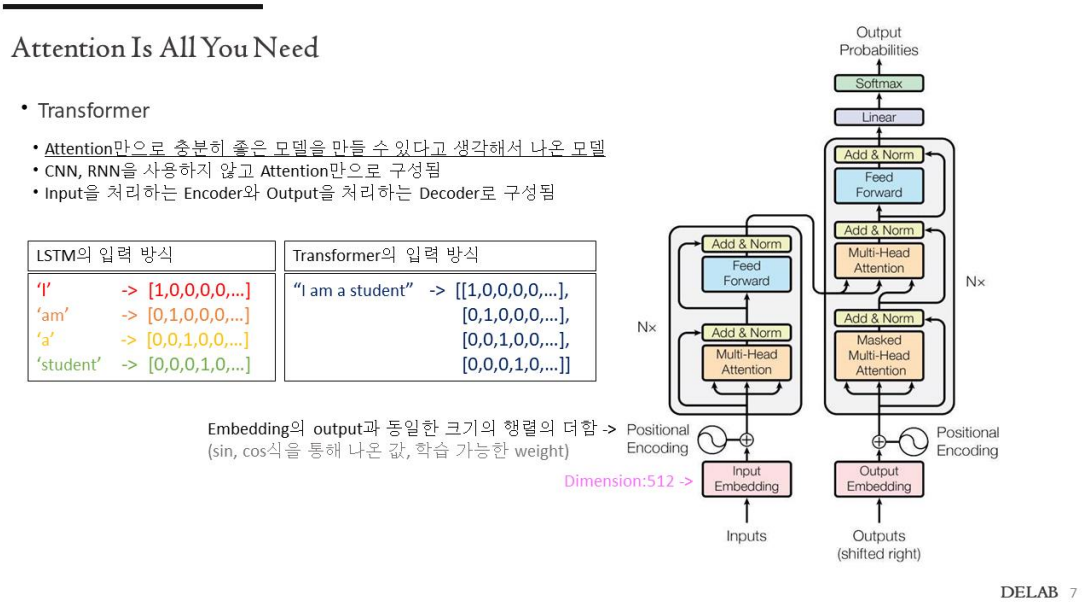
- Seq2Seq: LSTM을 이용한 번역 모델 (위 그림은 영어를 불어로 번역함)

DELAB 6

지금부터는 트랜스포머 구조를 제안한 Attention is all you need 논문을 리뷰하겠다. 이 논문은 자연어 처리 쪽 논문이기 때문에 NLP에 대한 이야기를 하겠다. NLP의 역사를 보면 지금까지 주를 이루었던 모델이 LSTM이 기반인 모델이다. LSTM은 장기 문맥을 기억할 수 있는 RNN이라고 생각하면 되고 RNN은 데이터를 순차적으로 처리하기 때문에 시계열 데이터를 다루는데 최적화된 인공 신경망이라고 생각하면 된다. 즉 LSTM은 순서를 고려하여 예측할 수 있다는 장점을 지니고 있고 아래 그림처럼 LSTM을 이용한 번역 모델이 많이 발달하였다. 이러한 번역 모델을 Sequence to Sequence라고 하고 본 예시에서는 영어를 불어로 번역하고 있다. 그래서 간단하게만 소개하면 I,am,a,student라는 각각의 시퀀스가 LSTM에 들어가기 전 embedding을 거쳐 사용자가 원하는 차원으로 벡터가 축소가 되고 이후 순차적으로 LSTM에 들어가서 feature를 파악하게 된다. 여기서 중요한 것은 이전 시간의 은닉층의 출력을 다음 시간의 은닉층으로 다시 집어 넣기 때문에 feature가 누적된다는 특징이 있고 encoder의 출력에 해당하는 context vector는 I am a student에 대한 모든 feature들이 누적되어 있다. 이 vector가 decoder의 입력으로 들어가서 현재 바라보고 있는 단어를 기준으로 다음에 나오는 단어를 순차적으로 예측하는 모델이라고 생각하면 된다.

하지만 문장의 의미나 단어 간의 관계가 항상 순차적이지는 않다. (관계가 항상 왼쪽에서 오른쪽은 아니다) 예를 들어 'I love you'라는 문장은 '나는 너를 사랑해'니까 love는 I하고도 연관이 있겠

지만 you하고도 연관이 있다. 이러한 문제는 문장이 길어질 때 더 안 좋은 결과가 나오는데 결론은 순서만을 가지고 제대로 된 번역을 할 수 없고 각 단어 간의 관계도 알 수 없다. 따라서 Attention 개념을 가지고 전체 문장(에서 중요한 정보)도 고려하려고 했던 것이다.



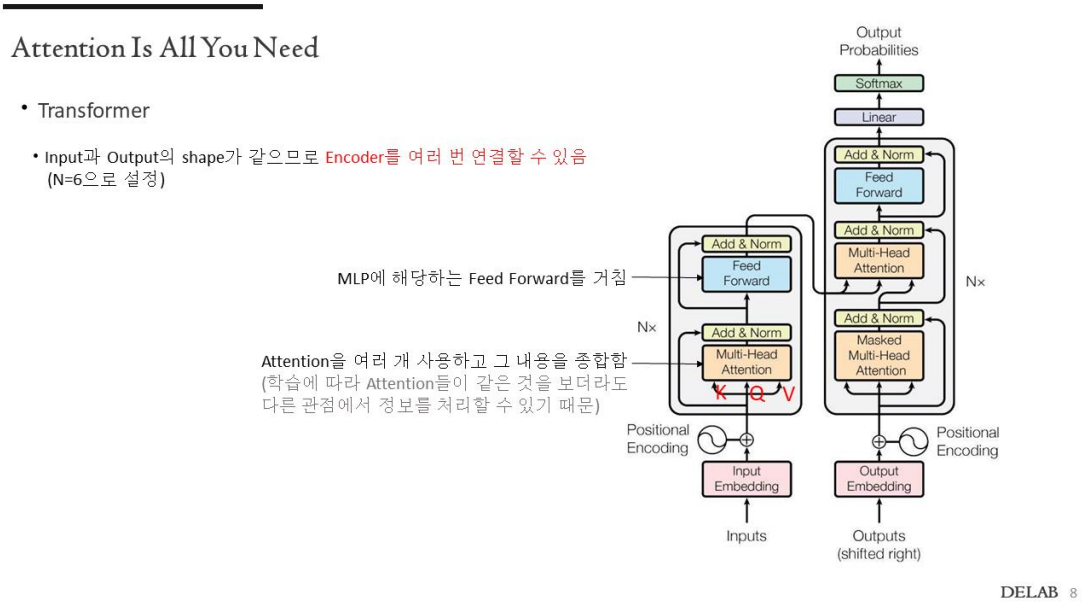
그렇게 Attention 개념이 보조적인 개념으로 사용되다가 Attention만으로도 충분히 좋은 모델을 만들 수 있을 것이라고 생각했다. 그래서 CNN, RNN을 사용하지 않고 Attention만으로 구성된 Transformer가 등장한 것이다.

오른쪽 그림은 Transformer의 구조이다.
Input을 처리하는 Encoder와 Output을 처리하는 Decoder로 이루어져 있다.

encoder의 가장 큰 특징은 문장이 통째로 들어간다는 것이다. 예를 들어 'I am a student' 라는 문장이 있으면 lstm은 각 시퀀스별로 모델에 들어가는데 transformer는 문장 전체가 들어간다. 각각의 단어가 원핫인코딩으로 표현되어 있다면 lstm은 원핫인코딩된 벡터가 한 개씩 들어갔다면 transformer는 전체 벡터, 즉 행렬이 입력으로 들어간다.

이 input은 input embedding을 통해 적절한 크기로 바꿔준다. 원핫인코딩 벡터의 크기가 단어 수였다면 embedding을 거친 벡터의 크기는 사용자가 지정한 크기가 된다. 논문에서는 512로 지정하였다.

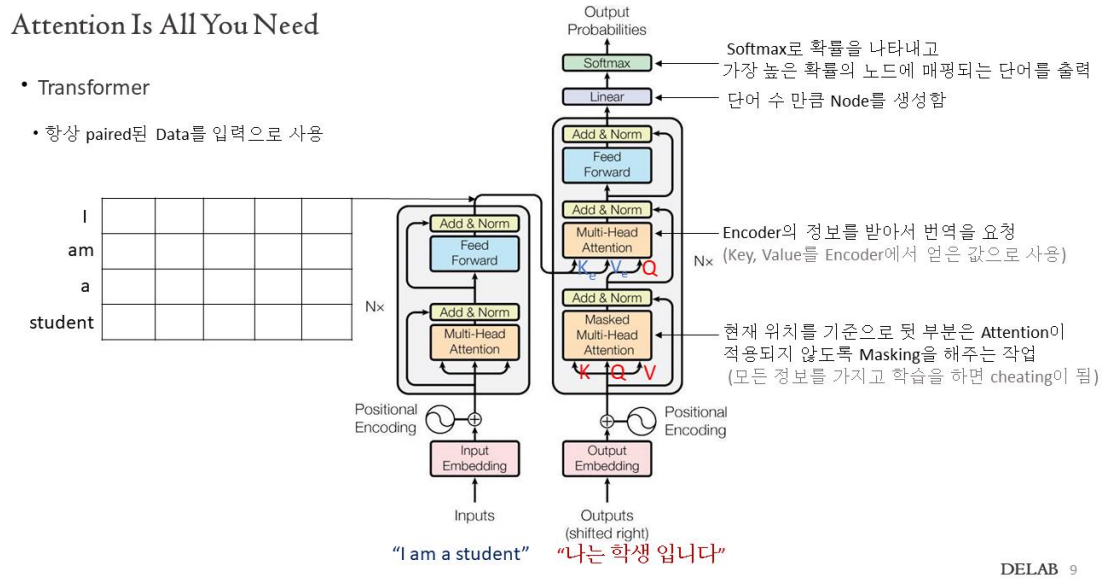
문장이 통째로 들어갈 때 중요한 점은 위치 정보이다. 따라서 positional encoding을 통해 input embedding의 output과 동일한 크기의 행렬을 더하는 방식이 추가되었다. 이 때의 행렬은 sin, cos식을 통해 나온 값이 될 수도 있고 학습한 가능한 weight가 될 수도 있다.



이후에는 본격적으로 attention을 사용하는 encoder에 들어가게 되는데 세 화살표는 각각 key, query, value를 나타낸다. key, query, value가 구해지면 multi-head attention이라는 것을 이용하는데 multi라는 것 자체가 attention을 여러 개 사용하겠다는 의미이다. 이유는 학습에 따라 attention들이 같은 것을 보더라도 다른 관점에서 정보를 처리할 수 있기 때문에 다수의 attention을 구해서 그 내용을 종합하겠다는 것이다. 그래서 여러 개의 attention을 계산한다고 보면 되고 그 내용을 합쳐서 나온 값을 이전 값이랑 더하고 정규화를 해준다.

마지막으로 MLP에 해당하는 feed forward를 거쳐서 다시 이전 값이랑 더하고 정규화를 한 번 더 해준다.

encoder의 중요한 점은 input과 output이 같다는 사실이다. 따라서 encoder를 여러 개 연결할 수가 있다. 그래서 논문은 encoder를 6번 반복해서 출력 값을 얻게 된다.



decoder도 input과 output이 같으므로 6번을 반복해서 출력 값을 얻는다고 한다. 다만 attention을 2개를 사용했다는 점에서 encoder와 차이점이 있다.

decoder에 대해 설명하기 전에 알아야 될 점은 transformer는 항상 paired된 data를 입력으로 사용한다는 것이다. 예를 들어 encoder에 'I am a student'가 들어갔다면 decoder에는 '나는 학생 입니다'가 들어간다는 것이다. decoder의 입력도 positional encoding까지는 encoder와 같다.

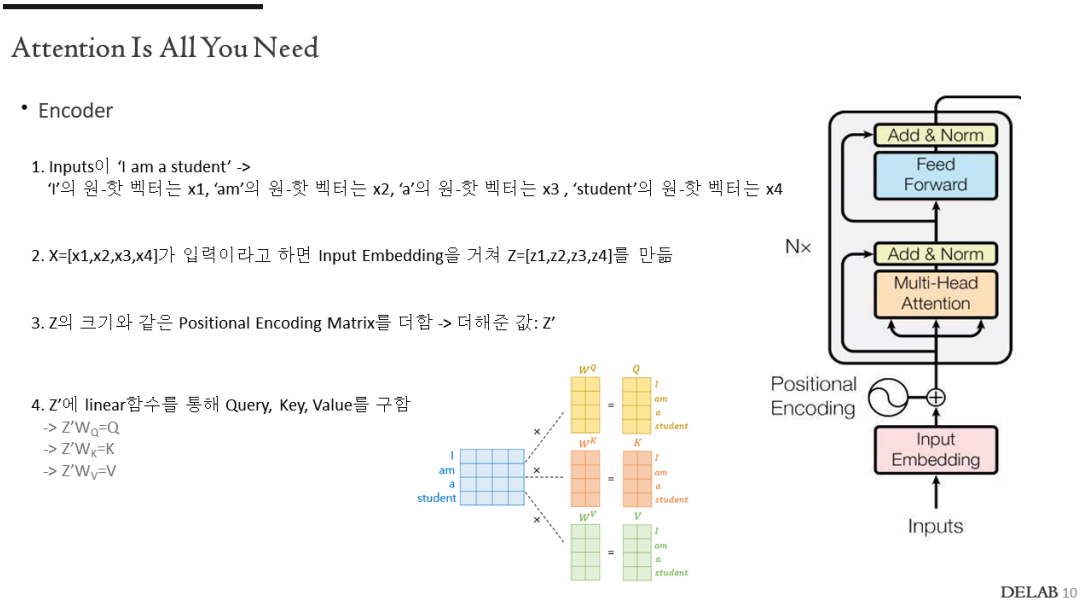
decoder의 attention은 masked multi head attention이라는 것을 사용하는데 이 내용은 transfomer의 task와 관련이 있다.
transformer의 encoder output 행렬이, 1행은 'I'에 대한 feature vector, 2행은 'am'에 대한 feature vector, 3행은 'a'에 대한 feature vector, 4행은 'student'에 대한 feature vector라고 하자. 이 행렬을 decoder의 입력으로 사용해서 '나는 학생 입니다'라는 번역을 하는 것이 task이다. 그래서 decoder는 처음에 '나는'를 만들어 내고 두 번째는 '나는' 뒤에 '학생'이 오는 것을 맞춘다. 이후에는 '나는 학생'뒤에 '입니다'가 오는 것을 맞춘다.

현재 '나는'이라는 시점에서 다음 단어를 예측한다면 뒤에 정보는 사용하면 안 된다. 만약 사용을 한다면 '나는 학생 입니다'라는 정보를 다 가지고 학습을 하는 것이기 때문에 cheating이 되는 것이다. 따라서 현재 위치를 기준으로 뒷 부분은 attention이 적용되지 않도록 masking을 해주는 작업이 포함돼야 한다. 이것이 바로 masked multi head attention이다. 이 attention의 결과 또한 residual connection을 따른다.

두 번째 attention은 본격적으로 영어에서 한글로 바꾸는 작업을 한다. 즉 encoder의 정보를 받아서 번역을 요청하는 것이기 때문에 key, value를 encoder에서 얻는 값으로 사용을 하고 decoder에서 얻은 값을 query로 이용하게 된다. (정보를 요청하는 것이 query이다. 주어진 데이터로 접근

을 해서 유사성을 계산하는 것이기 때문에 k,q,v가 위처럼 들어오는 것이 맞다.)

이 attention을 거친 결과는 feed forward block에 들어가는 데 두 block모두 residual connection을 사용한다. 최종 출력은 linear 연산을 통해 단어 개수만큼 node를 생성해낸다. ex) (0,1,0,0,0,0,...) 이 node들을 sotmax로 확률로 나타내면 가장 높은 확률에 해당하는 노드에 매핑되는 단어가 출력된다.



다음 Encoder를 좀 더 자세하게 살펴 보겠다.

1. inputs이 'I am a student'면 'I'의 원핫 벡터는 x1, 'am'의 원핫 벡터는 x2, 'a'의 원핫 벡터는 x3, 'student'의 원핫 벡터는 x4로 나타낼 수 있다.

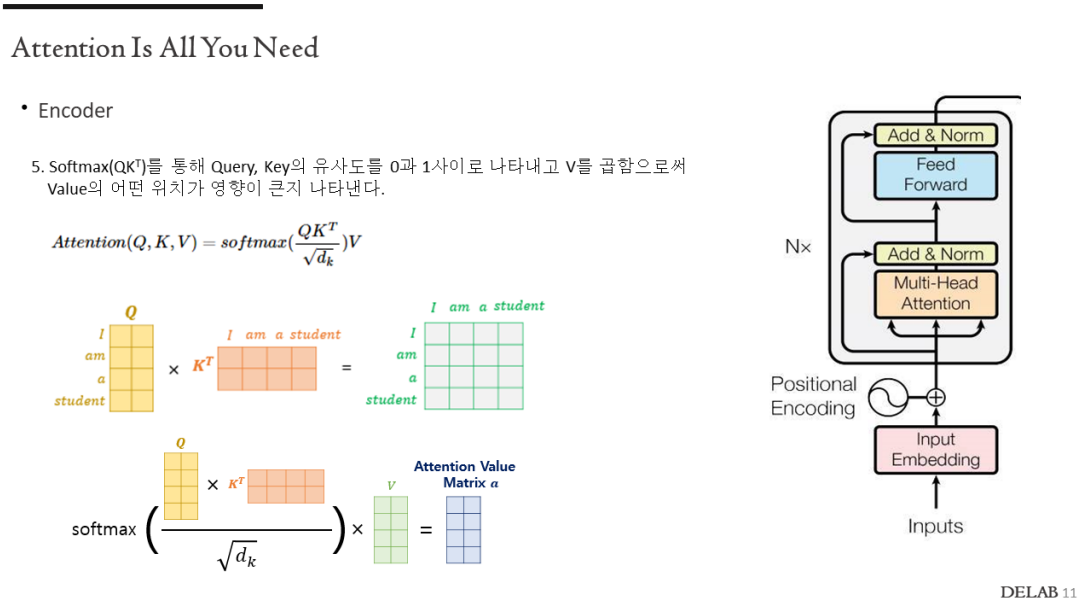
2. X=[x1,x2,x3,x4]가 입력이라고 하면 input embedding을 거쳐서 Z=[z1,z2,z3,z4]를 만들 수 있다.

3. Z의 크기와 같은 Positional encoding matrix를 더해준다. 더해준 값을 Z'이라고 하자.

4. Z'에 Query에 대한 linear함수를 통해서 Query를 구한다. ($Z'W_Q=Q$)

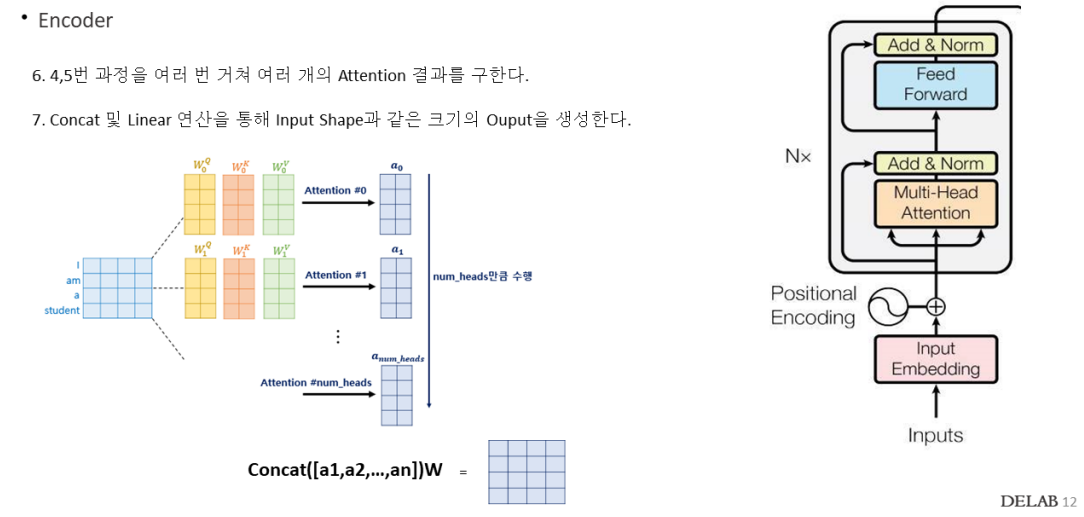
Z'에 Key에 대한 linear함수를 통해서 Key를 구한다. ($Z'W_K=K$)

Z'에 Value에 대한 linear함수를 통해서 Value를 구한다. ($Z'W_V=V$)



5. Softmax(QK^T)을 통해 Query와 Key의 유사도를 0과 1사이로 구하고 V를 곱함으로써 Value의 어떤 위치가 영향이 큰지 나타낸다. (attention: Softmax(QK^T)*V)
논문에서는 내적을 한 다음에 key, value의 dimension만큼을 루트씌워서 나눠준다. 왜냐하면 key의 dimension이 크면 내적 값도 커지면서 softmax에서 역전파를 진행할 때 Gradient Vanishing 문제가 발생할거라고 의심했기 때문이다.

Attention Map같은 경우, 비전에서는 중요한 픽셀들이 1, 덜 주요한 픽셀들이 0으로 채워진다면 nlp에서는 단어간의 관계가 높은 것이 1, 낮은 것이 0으로 채워지게 된다.



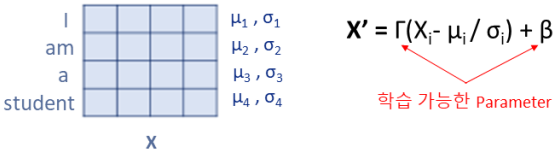
6. 4,5번 과정을 여러 번 거쳐 여러 개의 head를 구한다. (4번에서 W는 head마다 모두 다름)

7. concat([head1,head2,...,headn])을 하고 linear 연산을 하면 multi-head attention의 output이 된다. -> input크기와 같은 크기의 output이 됨 (concat([head1,head2,...,headn])W)

Attention Is All You Need

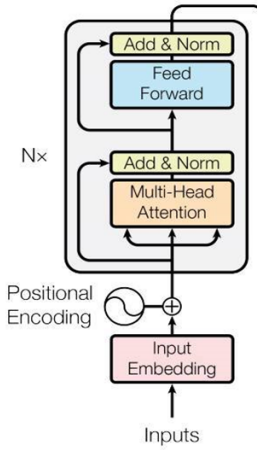
- Encoder

8. LayerNorm을 통해 각 벡터(단어)에 대한 정규화를 해준다.
(감마와 베타는 학습 가능한 Parameter)



9. Residual Connection이 적용된 Feed Forward를 진행한다.

$$FFNN(x) = MAX(0, xW_1 + b_1)W_2 + b_2$$

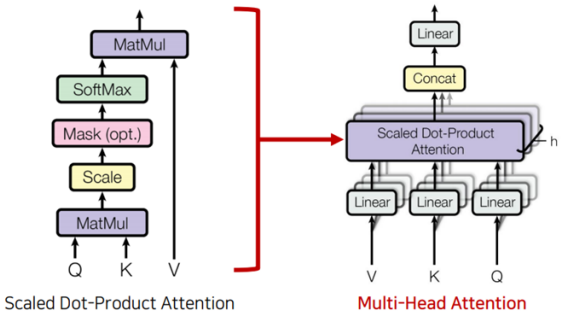


8. LayerNormalization을 통해 각 벡터(단어)에 대한 정규화를 해준다. 정규화할 때 감마(r)와 베타(b)는 학습 가능한 parameter이다. 이러한 정규화는 학습을 더 빠르게 하는 효과가 있다.

9. residual connection이 적용된 feed forward를 진행한다. Feed forward를 진행할 때는 linear 한 번 거치고 relu, 그리고 linear를 한 번 더 거치는 형태가 된다.

Attention Is All You Need

- Encoder
  - Matmul: 내적 계산 (QK^T)
  - Scale: √(key의 dimension)으로 나눠줌
  - Mask: Mask Attention으로 Decoder에서만 쓰임



지금까지의 설명을 그림으로 표현한 것이다.

다음은 Decoder를 자세히 살펴 보겠다.

Attention Is All You Need

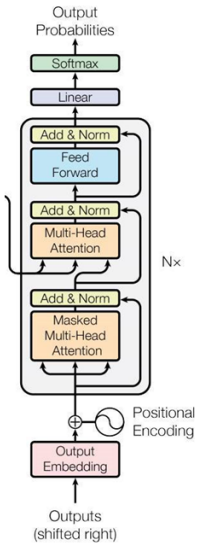
- Decoder

1. Encoder와 동일하게 output embedding과 positional encodin을 거쳐 Decoder에 들어오게 됨

2. Softmax(mask($QK^T/\sqrt{d_k}$)) 연산을 진행한다.

mask: 현재 시점 기준으로 다음 시점부터는 Attention을 고려하면 안 되니까 다음 시점에 해당하는 부분에 아주 작은 음수 값 (ex. -10^{-9})을 넣어준다. 이 작은 값을 넣어준 상태에서 Softmax를 취하면 음수 값에 해당하는 부분이 0에 가까워진다.

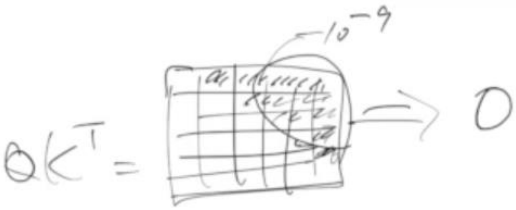
-> Attention Score를 0으로 만드는 테크닉



DELAB 15

1. encoder와 동일하게 output embedding과 positional encoding을 거쳐 decoder로 들어오게 된다.

2. Masked Multi Head Attention에서 softmax(mask(QK^T/sqrt(d_k)))을 진행한다. 현재 시점 기준으로 다음 시점부터는 어텐션을 고려하면 안 되니까 다음 시점에 해당하는 부분에 아주 작은 음수 값 (ex. -10^{-9})을 넣어준다. 이 작은 값을 넣어준 상태에서 softmax를 취하면 음수 값에 해당하는 부분이 0에 가까워진다. 즉 Attention Score를 0으로 만드는 테크닉이다.



Attention Is All You Need

- Decoder

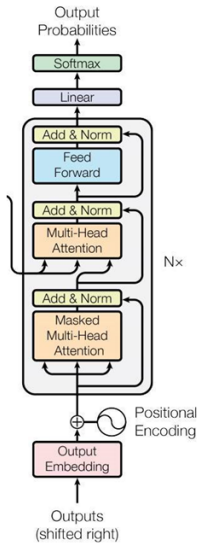
3. 본격적으로 번역을 위한 Attention을 할 때는 Query는 Decoder에 있는 것을 사용, Key와 Value는 Encoder에 나온 Output을 사용한다.

head = softmax($Q_D K_E^T / sqrt(d_k)) V_E$

4. 3번에서 구한 head를 여러 개 구하고 Concat한 것을 Linear해서 결과 값을 얻어낸다. 이후 이전 값을 더하고 LayerNorm처리한다.

5. Residual Connection이 적용된 Feed Forward를 진행한다.

6. Linear를 거쳐서 단어 수만큼의 노드를 만들고 Softmax를 거쳐서 다음 단어를 예측한다.



DELAB 16

3. 본격적으로 번역을 위한 attention을 할 때는 Query는 decoder에 있는 것을 사용하고 Key와 Value는 Encoder에 나온 output을 사용한다. 이것을 수식으로 나타내면 아래와 같다.

head = softmax($Q_d K_e^T / sqrt(d_k)$)*Ve

4. 3번에서 구한 head를 여러 개 구하고 concat한 것을 linear해서 결과값을 얻어낸다. 이후 이전 값 더하고 layer norm처리한다.

5. residual connection이 적용된 feed forward를 진행한다.

6. linear를 거쳐서 단어 수만큼의 노드를 만들고 softmax를 거쳐서 다음 단어를 예측한다.

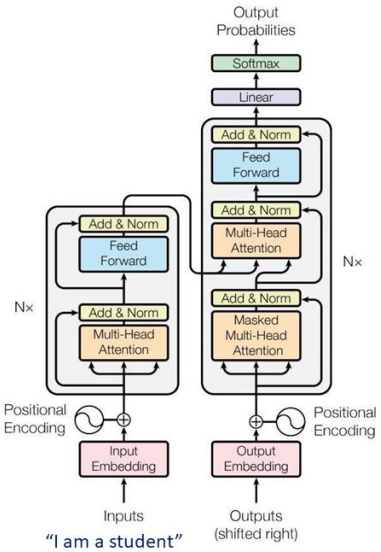
Attention Is All You Need

- Inference
  - Inference를 할 때는 Encoder의 Input만 사용된다.
  - Transformer는 Max_Length가 존재한다.

[Max_Length: 5]

- Encoder
"I am a student" -> [x1,x2,x3,x4,<\s>]

- Decoder
1. [<s>,<pad>,<pad>,<pad>,<pad>]
2. [<s>,<나는>,<pad>,<pad>,<pad>]
3. [<s>,<나는>,<학생>,<pad>,<pad>]
4. [<s>,<나는>,<학생>,<입니다>,<pad>]
5. [<s>,<나는>,<학생>,<입니다>,<\s>]



DELAB 17

다음은 예측(Inference)을 하는 방법에 대해 알아 보겠다.
학습을 할 때는 encoder의 input과 output의 input이 paired된 데이터라면,
예측을 할 때는 encoder의 input만 쓰인다고 보면 된다.
예를 들어 학습을 할 때는 한글과 영어가 input이었다면, 예측을 할 때는 한글만 input이다.

Transformer는 max_length라는 것이 있다. 예를 들어 max_len이 5이고 encoder의 input이 'I am a student'라면 시퀀스는 4개여서 x1~x4로 표현하고 단어가 끝남을 알리는 < \ s>토큰이 추가된다. 따라서 encoder의 input은 [x1,x2,x3,x4,< \ s>]가 된다.

decoder의 input도 사실 아예 없지는 않고 max_length를 따르는 입력이 들어가는 데,
문장의 의미있는 정보는 없어야 하므로 [<s>,<pad>,....,<pad>]가 된다. (pad는 아무 것도 없다는 뜻)

예측을 할 때마다 <pad>는 시퀀스로 바뀌게 되는데
처음 decoder를 통과하면 'I am a student'라는 attention결과를 이용해서 output은 '나는'이 된다.
이 output을 <pad> 위치에 넣으면 [<s>,<나는>,<pad>,<pad>,<pad>]와 같은 형태가 되는데 이것을 다음 input으로 사용한다. 이 과정을 반복하면 각 과정의 input은 아래와 같다. (빨간 색)

Attention Is All You Need

- Why Self-Attention
  - 각 Layer마다 계산 복잡도가 줄어든다.
  - Recurrence를 없앰으로써 병렬처리가 가능하다.
  - Long Range Depencencies에서도 잘 처리할 수 있다.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- n: sequence의 길이, 단어의 개수 (일반적으로 d보다 작음)
  -> n^2d가 nd^2보다 낮을 확률이 높으므로 유리한 복잡도를 가짐
- O(1): 시퀀스 데이터를 처리할 때 단 한번에 병렬적으로 구할 수 있음
  -> RNN과 비교했을 때 네트워크에 들어가는 입력의 횟수가 적음

DELAB 18

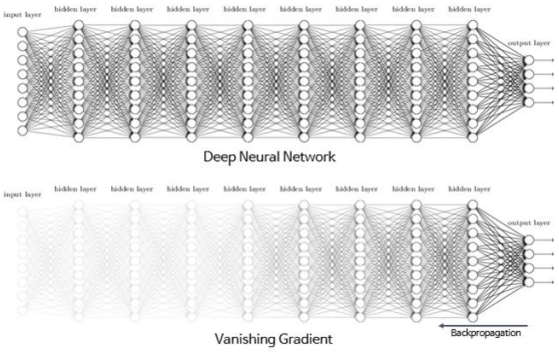
다음은 왜 Self Attention을 사용하는가에 대한 내용이다.

논문에서 세 가지 이점이 있다고 하는데, 첫 번째는 각 Layer마다 계산 복잡도가 줄어든다고 하고 두 번째 Recurrence를 없앰으로써 병렬 처리가 가능하다고 한다. 마지막으로 long range dependencies에서도 잘 처리할 수 있다고 한다. 표를 보면 더 자세하게 이해할 수 있는데, 표에서 n은 시퀀스의 길이 즉 단어의 개수에 해당한다. 근데 n은 d보다 작기 때문에 n^2d가 nd^2보다 낮을 확률이 높으므로 유리한 복잡도를 가진다.

그리고 Sequential Operations에서 O(1)에 대한 대한 내용은 시퀀스 데이터를 단 한 번에 병렬적으로 구할 수 있기 때문에 이렇고 RNN과 비교했을 때 네트워크에 들어가는 입력의 횟수가 적어서 유리한 복잡도를 가진다.

Residual Connection

- Vanishing Gradient
  - 일반적으로 딥러닝은 층을 깊게 할수록 좋다 -> (비선형성을 늘림, 모델 복잡도가 증가되면 정확도가 높아지는 경향, 매개변수 줄어듦)
  - 그러나 층을 깊게 하면 Vanishing Gradient라는 문제가 발생
  - 역전파가 Input Layer 방향으로 진행될 수록 Gradient 값이 미약해지는 문제



DELAB 19

이번에는 트랜스포머에서도 쓰이지만 다양한 아키텍쳐에서 자주 쓰이는 Residual Connection에 대해 알아보려고 한다.

일반적으로 딥러닝은 층을 깊게 할수록 성능이 좋다.

1. 중간에 Activation Function이 추가돼서 비선형성을 늘립니다.
2. 층이 깊어져서 모델 복잡도가 증가되면 정확도가 높아지는 경향이 있습니다.
3. 층을 깊게 하면 신경망의 매개변수(w)가 줄어들게 됩니다. (일부 모델 한정)

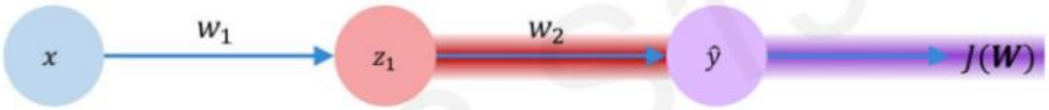
그러나 층을 깊게 하면 Vanishing Gradient라는 문제가 발생한다.
역전파가 Input Layer방향으로 진행될수록 Gradient가 미약해지는 문제이다.

(1) 출력층 전에 위치한 가중치 구하기 (w2로 부름)

w2를 구하려면 '출력층 노드의 오차 : dJ(W) / dy^'가 필요합니다.

[dJ(W) / dy^]가 존재하면 체인룰(연쇄법칙)에 의해 [dJ(W)/dw2]를 구할 수 있습니다.

[dJ(W)/dw2] 를 구했으면 w2를 구할 수 있습니다. (w2<- w2 – (learning rate) * dJ(W)/dw2)



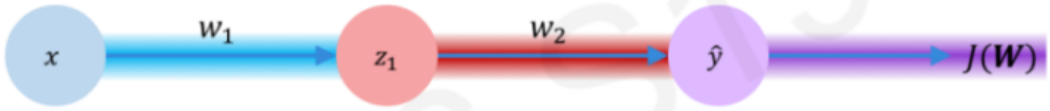
$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

(2) 은닉층 전에 위치한 가중치 구하기 (w1으로 부름)

w1을 구하려면 '다음 은닉층 노드의 오차 : dJ(W) / dz1'가 필요합니다.

[dJ(W) / dz1]이 존재하면 체인룰(연쇄법칙)에 의해 [dJ(W)/dw1]을 구할 수 있습니다.

[dJ(W)/dw1] 를 구했으면 w1을 구할 수 있습니다. (w1<- w1 – (learning rate) * dJ(W)/dw1)



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

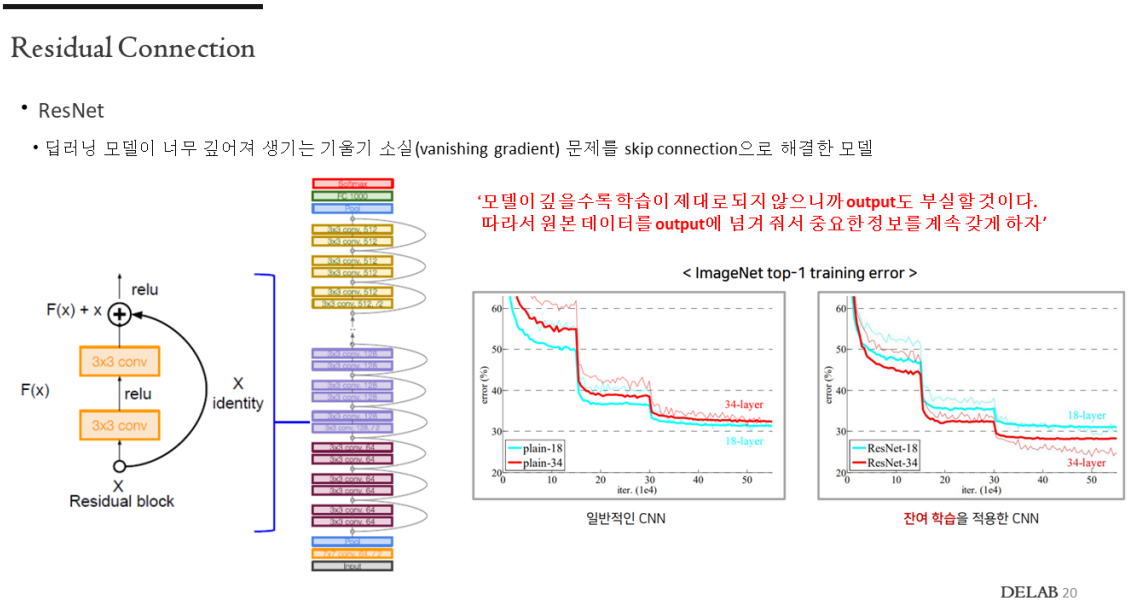
그런데 위 식을 자세히 보면 [dJ(W) / dz1] 을 구하기 위해 [dJ(W) / dy^] 를 이용합니다.

즉 현재 w의 그래디언트를 구하려면 다음 레이어들의 오차가 필요합니다.

따라서 출력층에서부터 오차를 역전파하는 것입니다.

위 설명에 따르면 결국 역전파는 이전 오차들을 불러와서 곱 연산을 반복하는 과정인데 input layer에 갈수록 값이 미약해진다. 예를 들어 rnn에 사용하는 y는 tanh에 대한 식인데 tanh는 -1에서 1사이의 값을 지니고 tanh의 도함수는 0에서 1사이의 값을 지닌다. 0에서 1사이의 값을 계속해서 곱하면 0에 가까워질 수밖에 없다.

이런 문제를 대비하기 위해 BatchNorm이나 ReLU같은 방법이 나왔고 ResNet에서는 Residual Learning을 적용해서 모델 자체를 업그레이드시켰다.



ResNet의 핵심 아이디어는 빨간 색 글씨와 같은데, '모델이 깊을수록 학습이 제대로 되지 않으니까 output도 부실할 것이다. 따라어 원본 데이터를 output에 넘겨줘서 중요한 정보를 계속 갖게 하자'라는 취지이다. 그래서 단순히 layer를 통과하는 것에서 끝나지 않고 residual connection을 추가하는 블록이 resnet의 블록이다.

이렇게 블록을 쌓았을 때 34 layer에서 18 layer보다 오차가 더 낮은 것을 확인할 수 있다.

Residual Connection

- ResNeXt
  - 같은 block을 반복적으로 구축함으로써 이미지 분류에서 더 높은 정확도를 보인 모델
  - width(채널 수)를 증가시키는 것보다 cardinality(블록 수)를 증가시키는 것이 더 좋은 성능을 보임 -> 결국은 채널 수를 높임
  - Grouped Convolution을 사용함

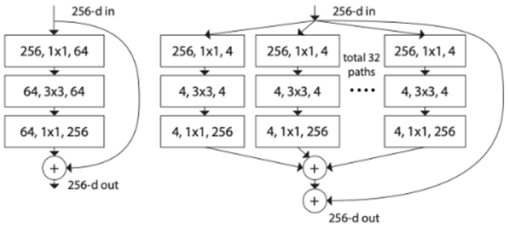


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Table 3. Ablation experiments on ImageNet-1K. (**Top**): ResNet-50 with preserved complexity (~4.1 billion FLOPs); (**Bottom**): ResNet-101 with preserved complexity (~7.8 billion FLOPs). The error rate is evaluated on the single crop of 224×224 pixels.

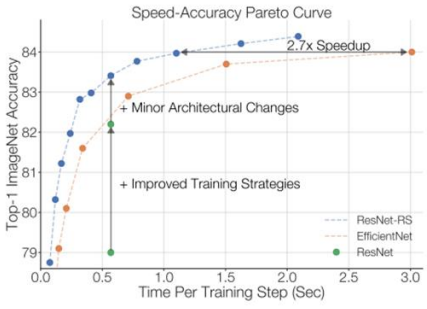
다음은 ResNet의 기본으로 발전된 모델에 대해 간단하게 말하겠다.

먼저 ResNext같은 경우 같은 block을 반복적으로 구축함으로써 이미지 분류에서 더 높은 정확도를 보인 모델이다. 여기서는 채널을 64로 변환시키는 bottleneck대신에 채널을 4로 변환시키고 feature를 구하는 블록을 여러 개 쌓아 grouped convolution을 하게 된다.

Residual Connection

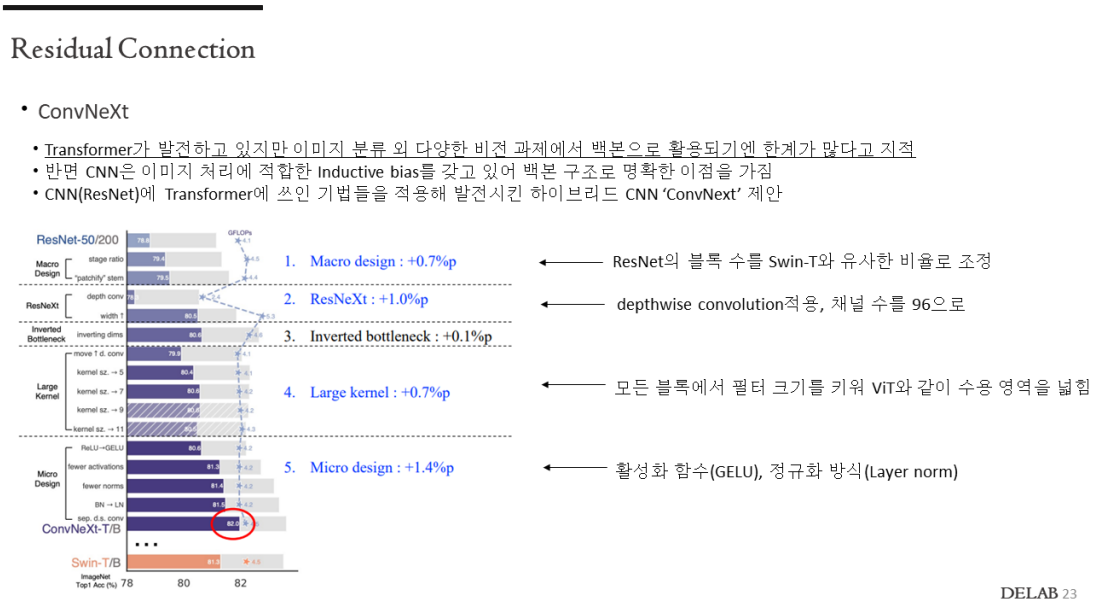
- ResNet-RS
  - CNN 기반 대표적인 모델인 ResNet의 이미지 분류 성능을 최대로 끌어올릴 수 있는 기법 탐색
  - 학습 방식, 규제 전략만으로 성능을 상당폭 개선할 수 있음을 지적함
  - 정확도는 EfficientNet이랑 비슷하지만 Training 시간은 더 빠름

Improvements	Top-1	Δ
ResNet-200	79.0	—
+ Cosine LR Decay	79.3	**+0.3**
+ Increase training epochs	78.8 †	-0.5
+ EMA of weights	79.1	**+0.3**
+ Label Smoothing	80.4	**+1.3**
+ Stochastic Depth	80.6	**+0.2**
+ RandAugment	81.0	**+0.4**
+ Dropout on FC	80.7 ‡	-0.3
+ Decrease weight decay	82.2	**+1.5**
+ Squeeze-and-Excitation	82.9	**+0.7**
+ ResNet-D	83.4	**+0.5**



다음은 ResNet-RS이다. 이 논문에서는 트랜스포머가 발달하고 있지만 CNN도 충분히 저력이 있다는 것을 밝힌 논문으로, 학습 방식, 규제 전략만으로 성능을 상당폭 개선시켰다. 파란색은 학습 방식, 초록색은 규제 전략, 노란색은 구조 변경이라고 생각하면 된다. LR을 서서히 줄이는 방법이

아니라 COSINE함수의 계형에 따라 줄이는 방법도 사용하고 LABEL SMOOTHING이라고 해서 원핫 인코딩을 좀 완화하게 표현하는 방법, Stochastic Depth라고 해서 ResNet의 Block중 일부를 사용하지 않는 방법 등이 쓰였다고 생각하면 된다. Squeeze-and-excitation같은 경우는 네트워크를 1x1xC로 변경하고 feature를 파악한 뒤 다시 네트워크 모형을 복구시키는 방법인데 성능이 0.7퍼센트가 올랐고 ResNet-D같은 경우 stride나 padding만 바꿨는데 성능이 올랐다고 한다.



다음으로 2022년 1월에 나온 ConvNeXt라는 논문이다. 이 논문은 트랜스포머의 문제점을 지적하고 있는데 트랜스포머가 다양한 비전 과제에서 백본으로 활용되기엔 아직 한계가 많다고 지적한다.

CNN은 이미지 처리에 Locality가 중요하다는 가정 아래에서 설계된 모델인데 트랜스포머는 자연어 처리에서 사용된 원리를 똑같이 가져와서 사용한거라서 Inductive Bias가 없다고 표현한다. 즉 이미지 처리에 특화된 가정이 없다는 뜻이라고 생각하면 된다.

그래서 ResNet을 백본으로 해서 트랜스포머에 쓰인 기법들을 적용해 발전시킨 하이브리드 CNN을 제안하였다. 결론만 말하면 ViT와 Swin Transformer보다 성능이 잘 나왔다고 한다. 이것에 대한 내용을 추후 리뷰하도록 하겠다.