기존의 word representation은 좋은 성능을 가져왔었지만(유사도, 위치 기반으로) <UNK> 발생 시 (모르는 단어들 -> one-hot -> word2vec -> 다 같은 벡터의 과정으로 <UNK> 생성된다고 생각함.) 모르는 단어들은 다 <UNK>로 발생돼 구분할 수가 없게 된다. <UNK>의 발생은 QA Task에 큰 문제가 된다.(+ 맥락을 파악하려 하지는 못함)

따라서 이전에 OOV를 해결하는 character-level modeling을 배웠다.(last lecture)

아니면 1. <UNK>가 unsupervised word embedding에 존재한다면 그걸 사용하고 2. 아예 모르는 것은 random vector로 만든 다음에 vocab에 추가할 수도 있다...?? (Dhingra 2017)

우리는 Word2Vec, GloVe, fastText와 같은 word representations를 사용하였다.

하지만 두 가지 문제점이 있다.

1. 항상 same representation(문맥에서 어디에 word token이 발생하는 지 고려하지 않고 word type을 무시하며)
2. 다른 중의적 의미를 가졌을지도 모르지만 어떤 한 단어를 오로지 한 벡터로 표현한다.

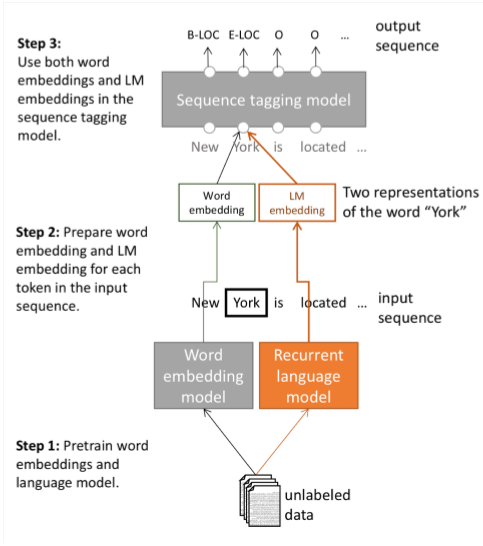
자 그러면 이제 word representation을 다시 한번 강화시켜보자..  
그 해결책으로 NLM의 아이디어를 도입하였다. Next word를 맞추는 task인 LM을 이용하다 보면 LM은 each position에서 context-specific word representation을 도출하는데 이것은 word representation으로 사용할 수 있다.

**TagLM – “Pre-ELMo”(Peters 2017)**

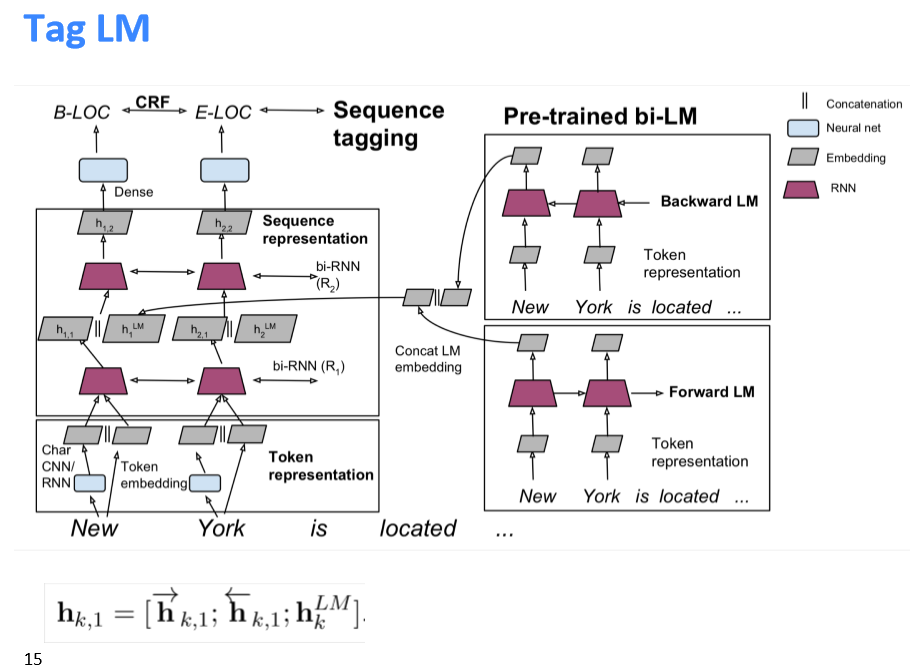
Context sensitive word embedding, NER 같은 small task-labeled data의 소규모가 한계 같은 느낌? -> semi supervised 방법을 채택하면 어떨까?

Context independent 한(word2vec 같은) Word Embedding Model 과 맥락을 파악할 수 있는 Recurrent Language Model(두 모델은 unlabeled data로 미리 학습, RLM은 biLSTM을 사용한다.)을 동시에 사용

그 후 아래와 같이 sequence tagging model에 들어가게 된다.

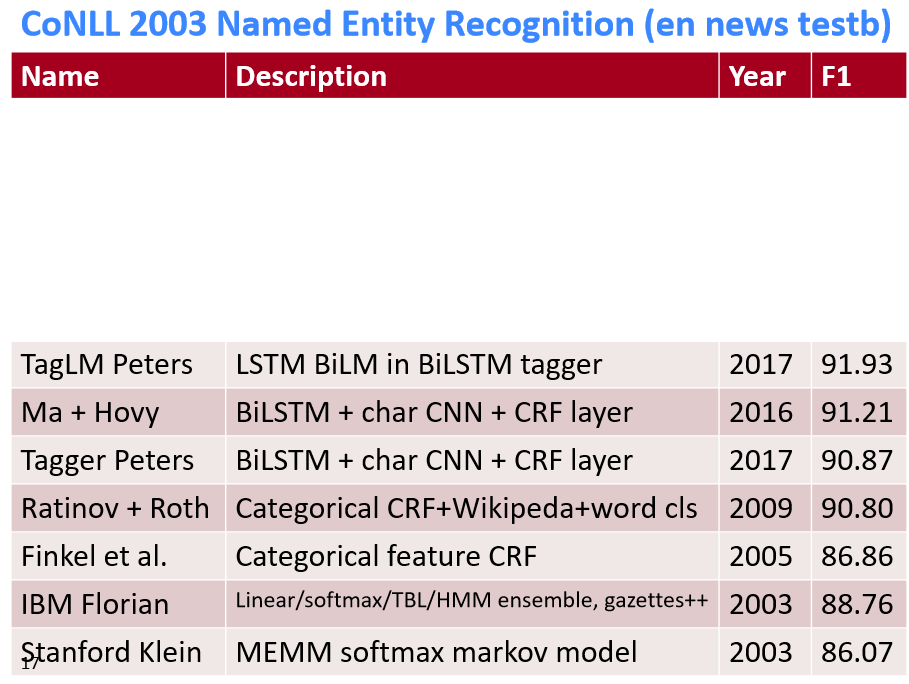


좀 더 자세하게 살펴보면



1. Bi LM으로 임베딩
2. Char CNN or RNN + Token Embedding(word2vec) -> bi-RNN -> 임베딩
3. 1, 2를 합쳐서 다시 bi-RNN 후 최종 sequence representation -> sequence tagging에 들어간다.

즉 word representation의 새로운 방법을 제안한다.(word2vec + character level + LM embedding)



의 성능을 보인다.

TagLM 특성 정리 - LM을 supervised로 훈련하는건 도움이 안된다. 그냥 LM이 아니라 bi-LM이다. Billion Word Benchmark로 학습 시켰다.

**CoVe(McCann 2017)** – 교수에 의해서 skip 당했다.

**ELMo(Peters 2018)**

<https://wikidocs.net/33930>

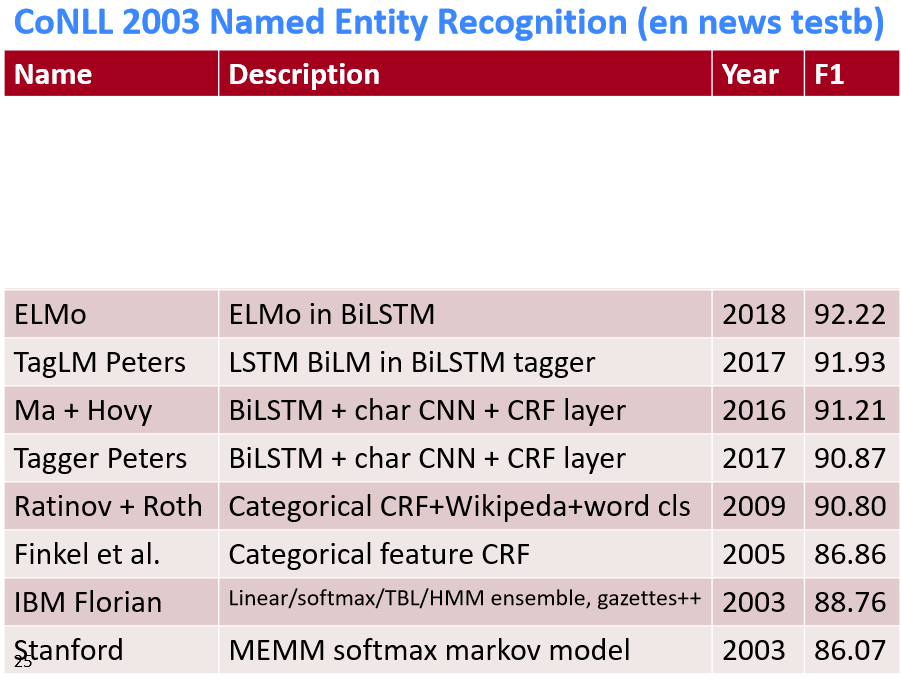
이전 TagLM 과 크게 다르지 않다. => 역시 Contextualized Word Embedding 이다. (bi-LM 부분의 구조가 약간 바뀌게 된다, 그 외에 char CNN/RNN or Token embedding은 변하지 않는다.)

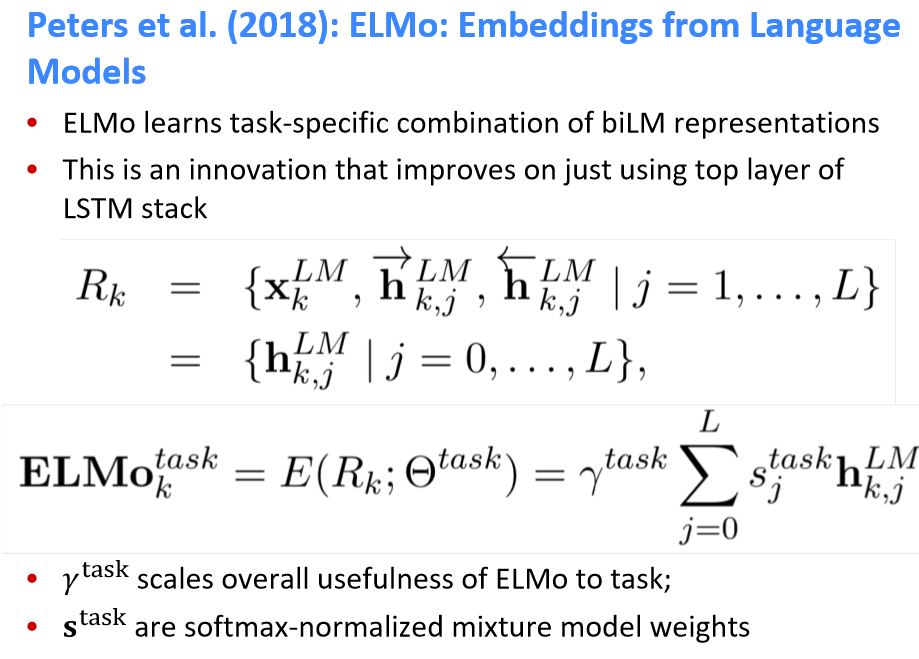
2 biLSTM, bi-LM 입력은 character CNN으로 임베딩 후 적용(기존에는 Token embedding), 각 hidden/cell states dim = 4096 이고 projections to next input dim = 512, residual connection

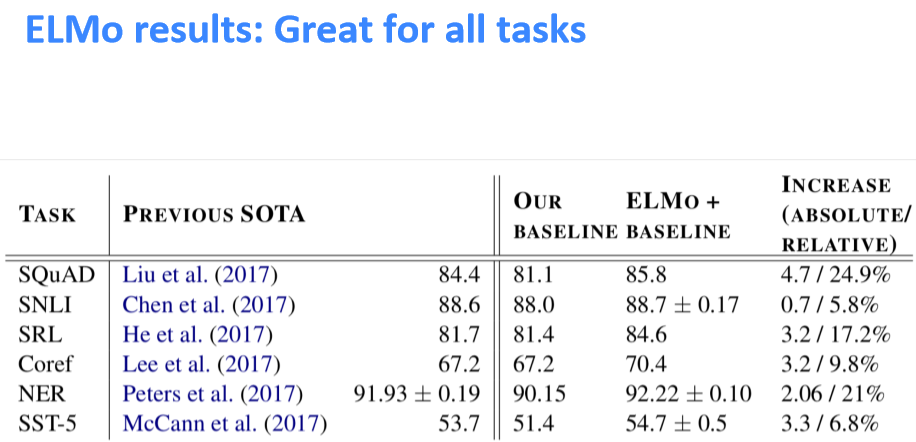
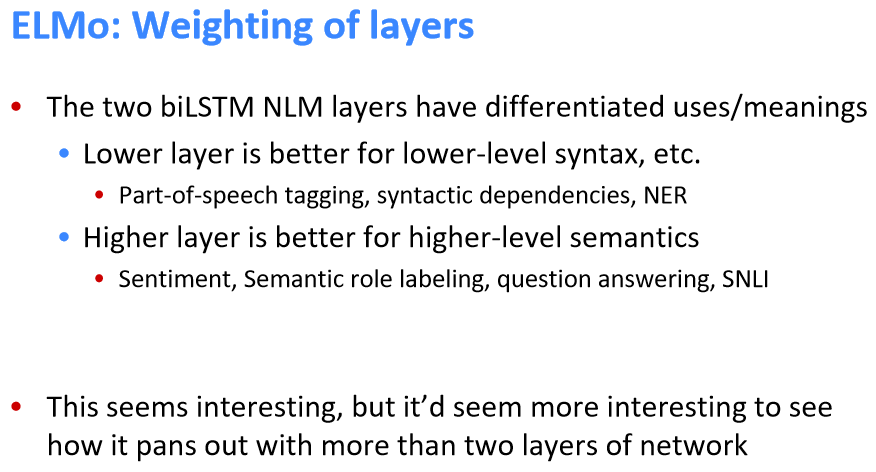
아래와 같이 진행되는 방식으로 TagLM에서 업그레이드 된다.

이때 s-j-task 들과 r-task는 훈련과정(pre-training) 중에 학습된다.

r-task: global scaling factor gamma ( 특정 task에 대해 ELMo Representation이 얼마나 기여할지 정도를 정함.)



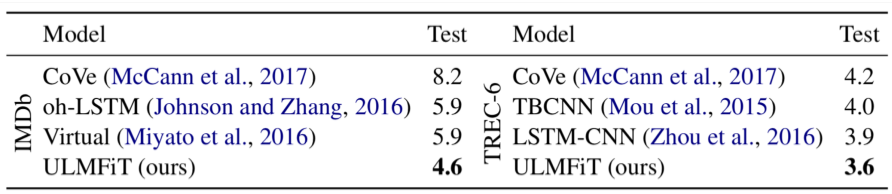


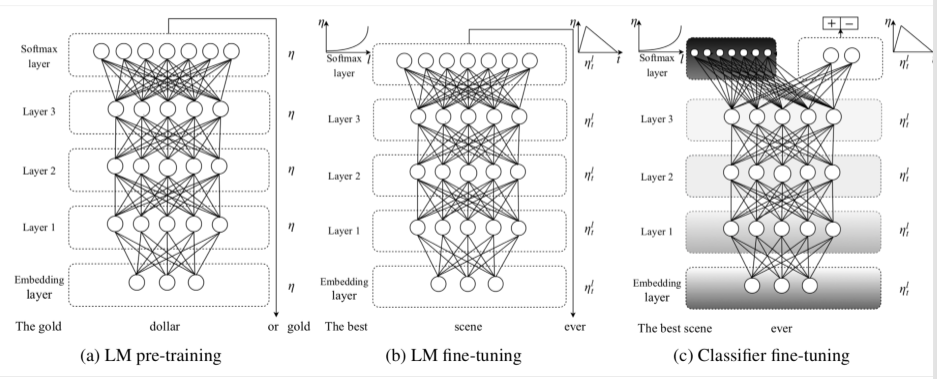
  
라는 특성을 가진다 한다.

**ULMfit(Howard 2018)**

언어 모델을 이용해 곧바로 문서 분류기를 생성하려 했다. Using transfer

1. 일반 언어 모델 학습: AWD-LSTM(ICLR 2018, 다양한 정규화 방법을 적용) 이용, 위키피디아 영어 버전 전체에 대해서 학습되었다. (아마 unsupervised 일 것임)
2. Task 맞춤형 언어 모델 튜닝: 언어 모델을 새로운 task 데이터에 대해 학습(input text만 이용하여 unsupervised로 진행)  
   - discriminative fine-tuning: 각 레이어 별 learning rate를 서로 다르게(깊을수록 작은 lr)  
   - slanted triangular learning rates: epoch가 커질수록 lr 크게
3. 이번에는 마지막 softmax layer를 제거하고 task에 맞게 재구성한 후 supervised로 학습한다.  
   - concat pooling: 마지막 분류 레이어 입력으로 사용될 벡터 구성할 때 마지막 잠재 상태 + 입력의 max or mean pooling vector를 concat하여 이용  
   - gradual unfreezing: 분류 레이어에서 가까운 레이어부터 차례대로 unfreeze





**GPT 1,2 (짧게, OpenAI)**: 그냥 근황만 소개하심.

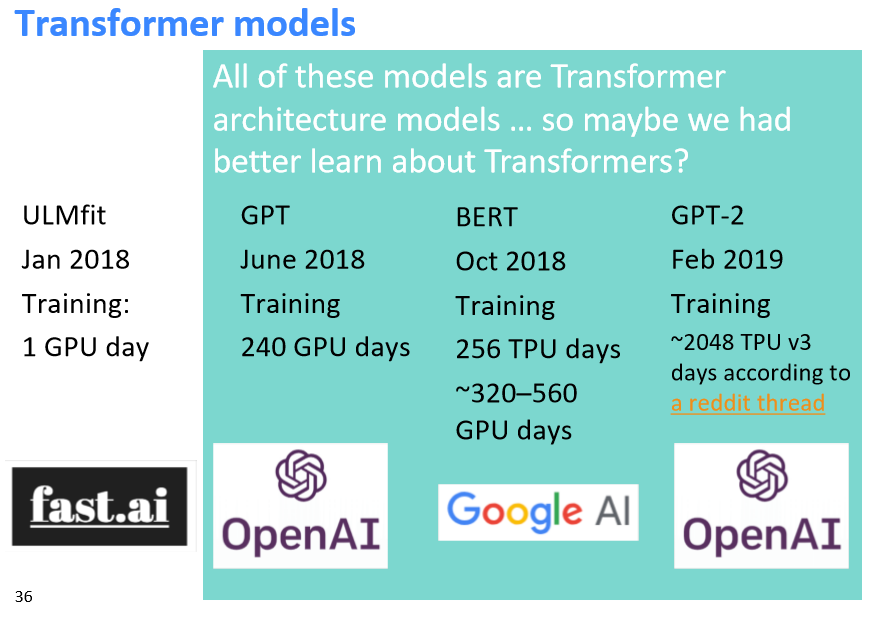
**Transformer(Vaswani 2017)**

<https://wikidocs.net/31379>

<https://pozalabs.github.io/transformer/>

<https://namhandong.tistory.com/48>

<https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transfomer-self-attention-842498fd3225>



SLOW Computation -> parallelization Computation 필요하다. 또한 long range dependency 해결 위한 attention에서 우리가 만약 only attention 만 필요로 한다면?? => RNN이 과연 꼭 필요한가?? (참고로 target task는 NMT)

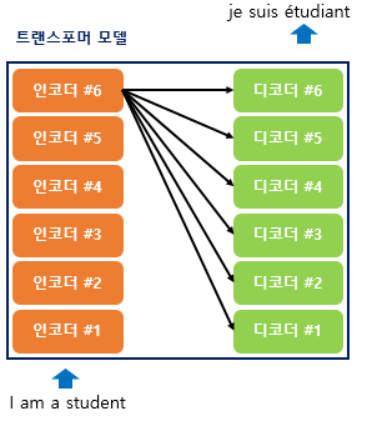
Transformer는 인코더-디코더 구조를 설계했지만 RNN을 쓰지 않으면서 우수한 성능을 보인다.

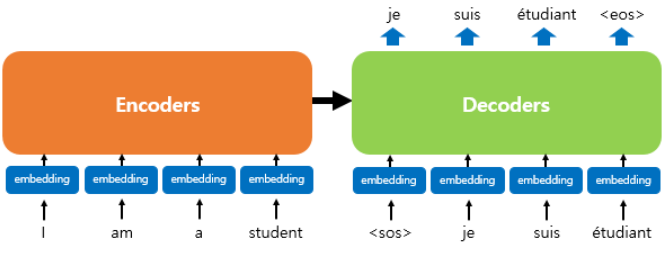
하이퍼파라미터 – d-model = 512(임베딩 크기, 다음 층의 인코더와 디코더에 값 보낼 때의 크기)

한 인코더 or 디코더를 layer로 생각했을 때 layer 개수 = 6 layers

어텐션 병렬 개수 = 8

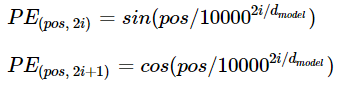
d-ff = 내부 feed forward 신경망의 은닉층 크기 = 2048 (입력층과 출력층은 d-model 크기와 같다.)

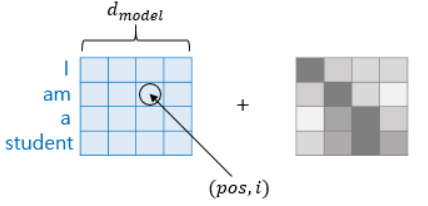


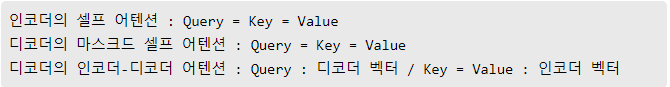


RNN을 이용하지 않지만 <sos>, <eos> 의 시작, 끝 심볼을 통해 연산을 하는 등 여전히 기존 seq2seq의 인코더-디코더 구조를 보이고 있다.

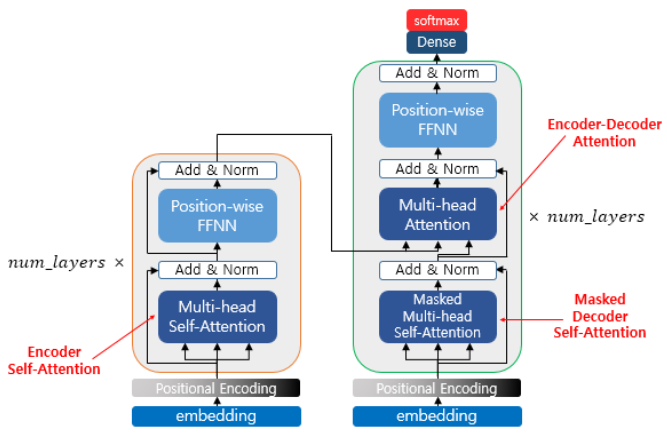
**트랜스포머 입력 – 포지셔널 인코딩**  
기존 RNN에서는 timestep 단위로 정보 처리 -> 위치 정보를 자연스럽게 획득  
하지만 여기서는 인위적으로 조절 필요 -> 포지셔널 인코딩 필요



(i가 홀수, 짝수냐에 따라 sin, cos 어떤 것을 쓸지 달라짐) 이렇게 인코딩 입력 행렬이 위 그림처럼 만들어진다.



쿼리, 키, 밸류는 이전 seq2seq에서 설명했었다.

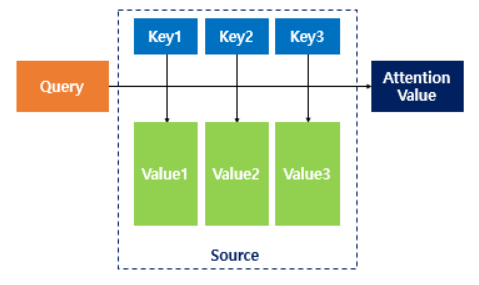
이런식으로 모델링을 하게 된다. (멀티 헤드 라는 것은 어텐션을 병력적으로 수행한다는 것을 의미)

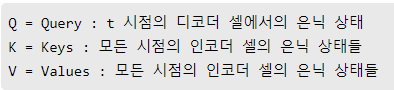
**인코더 with multi-head self-attention**

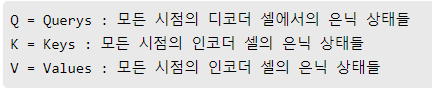
6개의 인코더를 사용하고, 1개의 인코더에는 크게 2개의 sublayer로 나뉘어짐.

멀티 헤드 셀프 어텐션 = 셀프 어텐션을 병렬적으로 사용했다는 의미  
포지션 와이즈 피드 포워드 = 일반적인 피드 포워드

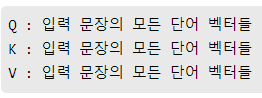
앞서 seq2seq에서 다음과 같이 정의하였다.



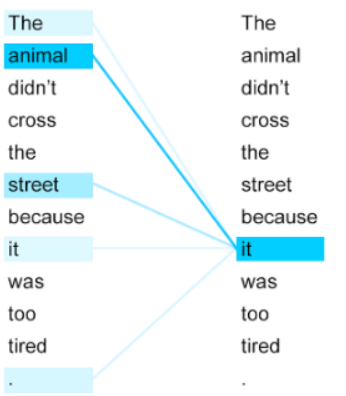


하지만 이것을 t 시점이 아니라 반복적인 쿼리를 수행한 최종의 전체 시점에 대해 일반화 할 수 있다. 

그렇지만 셀프 어텐션은 이거를 다시 한번 꼬게 된다.



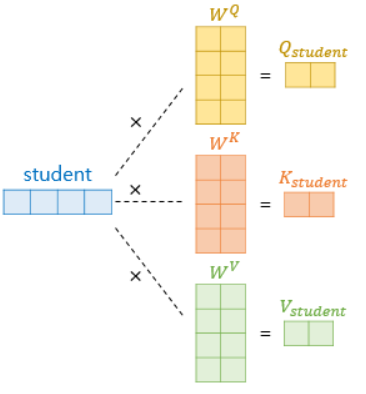
셀프 어텐션의 효과란?

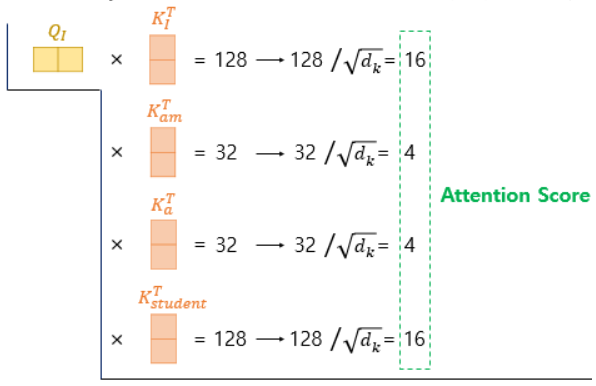
에서 it이 animal인지 street 인지 기계는 잘 몰랐지만 입력 문장 내의 단어들끼리 유사도를 구함으로써 it이 animal과 연관되었을 확률이 높다는 것을 스스로 찾아낸다.

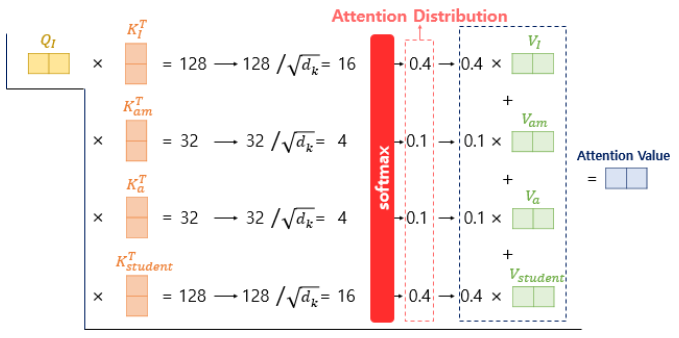
Seq2seq의 구조로는 찾을 수 없던(다른 문장끼리 연관성 찾아냄) 방식이다.

+ 레이어당 전체 연산량 줄고 병렬화가 가능한 연산이 늘어나며 long-range term들의 dependency도 잘 학습한다..?

이제 그러면 Q, K, V 벡터를 얻으면 되는데 그냥 단어 벡터를(d-model dim) 이용하는 것이 아니라 d-model을 num\_heads로 나눈 값(512/8= 64) 를 사용하게 된다.

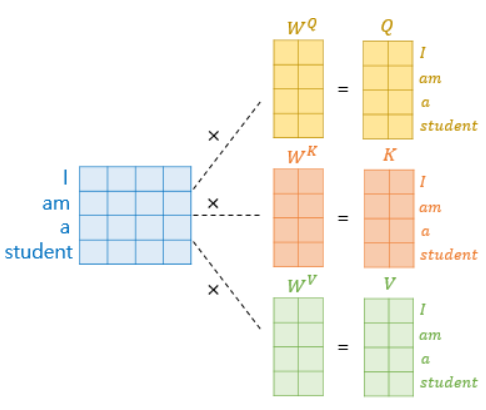
의 그림처럼 가중치 행렬(학습 가능한)을 곱해 Q, K, V – student 벡터를 얻게 된다. 그 후 기존 어텐션 메커니즘처럼 어텐션 스코어를 구하되 여기서는 Scaled Dot-Product Attention을 이용하게 된다.(기존 dot product에 대해서 스케일링이 추가됨.)

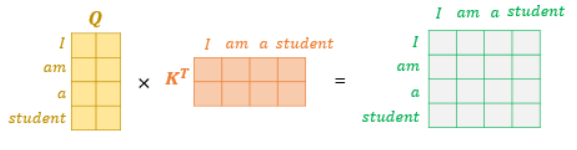


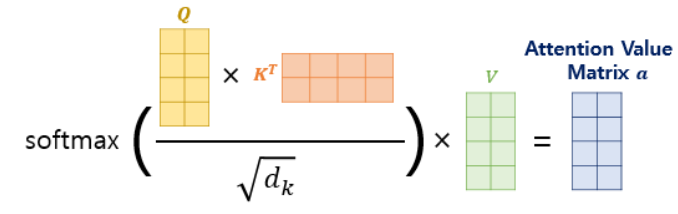


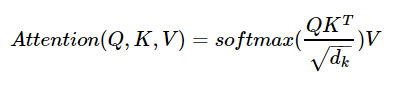
보통 dk = d-model / num\_heads = 64 이다. 어텐션 스코어 결과는 쿼리 단어가 키 단어와 얼마나 연관되어 있는지를 나타낸다.

어텐션 분포를 구하고 value 단어 벡터에 곱하여 weighted sum을 하여 최종적으로 어텐션 밸류를 얻게 된다. 이제 이 과정을 총 여기서는 4개의 쿼리 단어 벡터에 대해서 실행하면 되는데 굳이 각 Q 벡터마다 따로 할 필요가 있을까? => 행렬 연산으로 처리하자



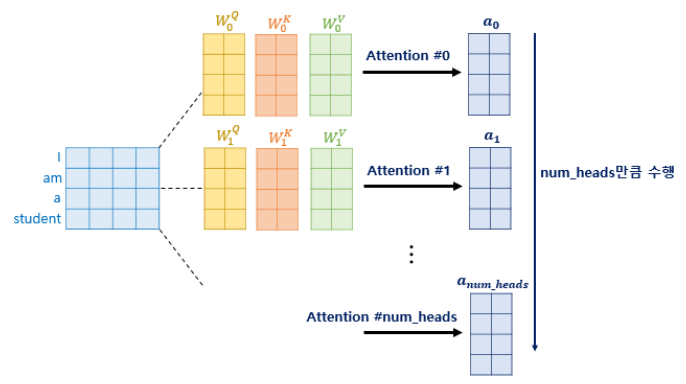


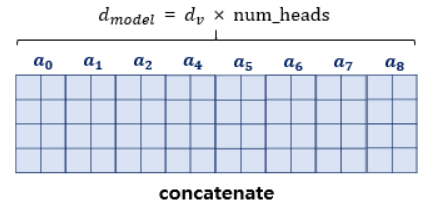


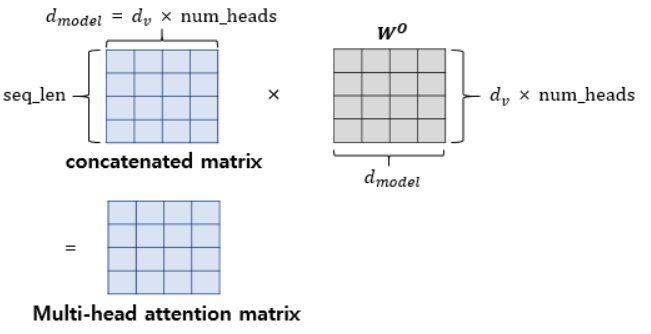
= 가 된다.

왜 앞서 단어 벡터를 d-model에서 d-model/num\_heads = dk = dv 값으로 차원을 조절하여 사용하였을까? 일단 어텐션을 여러 번 병렬로 사용하는 것이 효과적이라고 판단했었고 그 이유가 각 헤드가(어텐션이) 여러 시각으로 놓치는 데이터를 적게 한다고 생각했기 때문이다.

Ex) 아까 예문에서 첫 어텐션이 it과 animal을 신경 써서 본다면 두번째 어텐션은 it과 tired를 신경써서 볼려한다.



 가 되면 concat 결과는 인코더의 맨 처음 입력이였던 (seq\_len, d-model) dim이 된다. = head 한 개 결과가 (seq\_len, dv) dim 이다. 그리고 진짜 마지막으로 가중치 행렬 한 개 더 곱해준다.



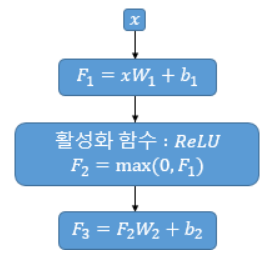
예외적으로 padding mask라는 처리를 하는 경우가 있다. 입력 문장에 <PAD> 단어 벡터가 있을 경우 <PAD>는 실질적 의미가 없기 때문에 연산에서 제외시키는 것이다.

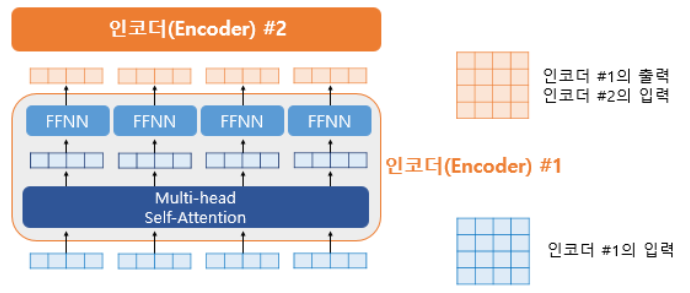
따라서 Key의 <PAD>의 경우 아주 작은 음수 값을 넣어주어 마스킹한다. = dot product 값이 매우 낮게 되고 softmax 까지 지난다면 어텐션 분포 값은 0에 가까울 것이며 유사도 구하는 일에 <PAD> 토큰이 반영되지 않게 된다.

이제 진짜 멀티헤드 어텐션 in 인코더 끝

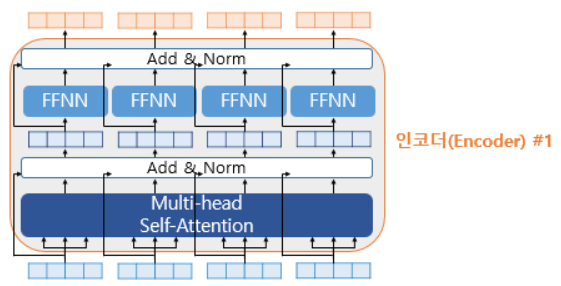
**포지션-와이즈 피드 포워드 신경망**

이 부분은 인코더, 디코더 공통 부분인 서브층이다.(fully connected FFNN)

여기서 x가 멀티헤드 어텐션의 아웃풋(seq\_len,dv) dim 이자 FFNN의 입력으로 작동한다. W1: d-model x d-ff, W2: d-ff x d-model. FFNN의 최종 output도 (seq\_len, d-model) 이며 다음 인코더의 인풋으로 들어가게 된다.



**Residual Connection & Layer Normalization**

동일한 차원이라 가능, 우리가 아는 그 skip connection(함수 거치는 과정과 backprop을 생략한다..?)



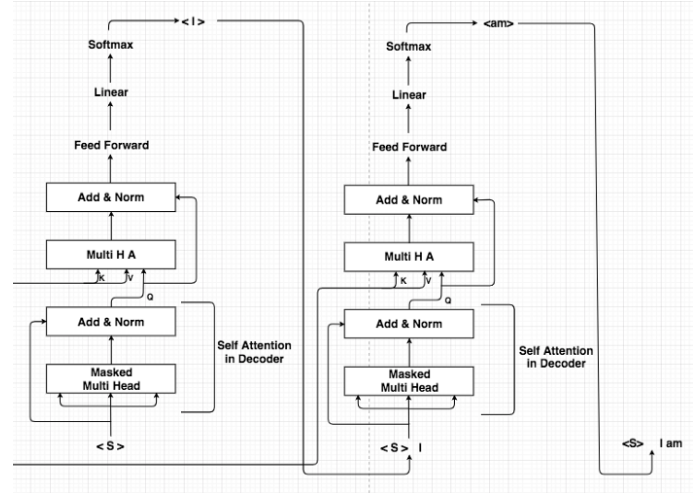
Layer Norm 은 이 레이어의 입력을 mean 0 & variance 1의 분포로 바꾸게 한다. (배치 정규화 느낌 물씬)

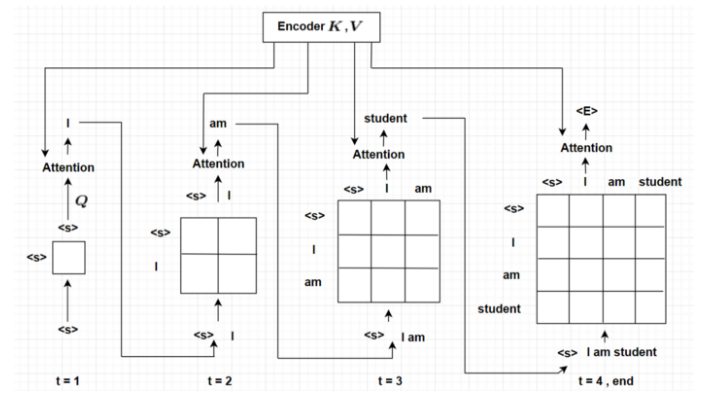
**Masked Multi-Head Attention & Encoder-Decoder Attention**

Masking을 하는 이유는 timestep 때문인데 position i에서 그 이전의 정보만 decoder의 인풋으로 넣기 위함이다. 뒷 부분은 -inf로 해서 유사도 측정(내적값)에 영향을 주지 않아 어텐션 작동을 하지 않게 한다. 구조는 아래 그림을 보면 바로 이해가 간다.

Inference 시에 greedy or beam search를 이용하도록 한다. 또한 인코더의 결과는 Encoder-Decoder Attention의 Key & Value 로써 작동한다.

**질문? 근데 그러면 항상 디코더에서도 (seq\_len, d-model)을 유지해야 하는 것 아닌가?? 근데 디코더의 output은 seq\_len가 정해지지 않자나 이거 어떻게 해결하지??**





내일(8/27) – 버트 & ppt & 다음 강의 조금 & GUI --

**BERT(Devlin 2018)**

<https://ebbnflow.tistory.com/151>

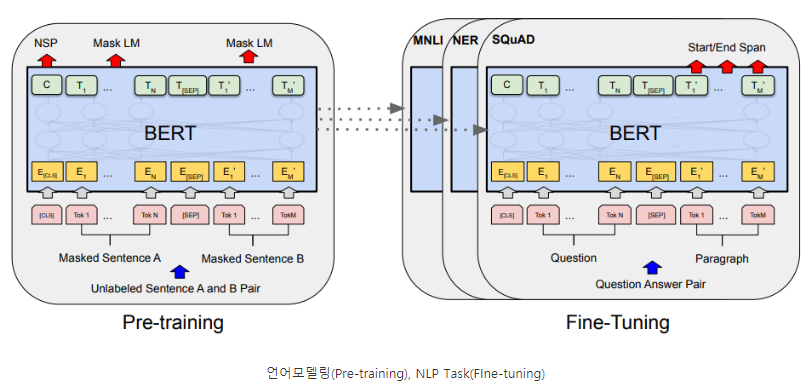
Pre-training of Deep Bidirectional Transformers for Language Understanding

모든 자연어 처리 tasks에서 좋은 성능을 내는 범용 LM

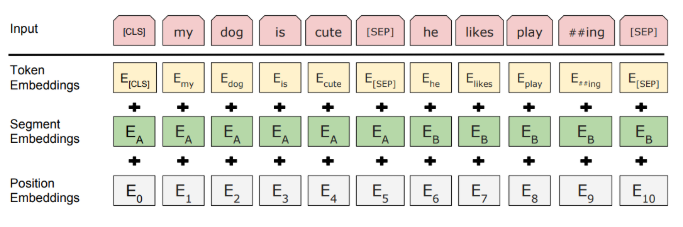
LM은 왜 오직 Left or right context를(unidirectional, ELMo도 그저 단방향을 두 번 이용한 것일 뿐이다.) 사용? Bidirectional도 필요하다.

Directionality는 좋은 확률 분포 생성에 있어서 꼭 필요하며 bidirectional 인코더에서 “see themselves” 해야 한다.(self-attention 인가??)

이것 역시 ELMo 처럼 word embedding을 위한 LM인데 그 모델로 Transformer를 이용한 것



그럼 저 그림에서 BERT라고 쓰여진 부분이 모델의 부분이다.

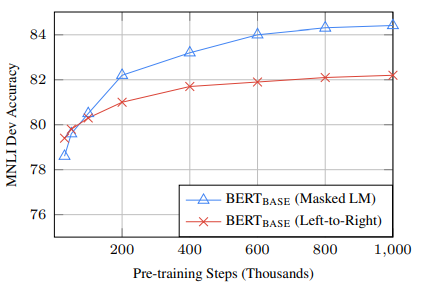


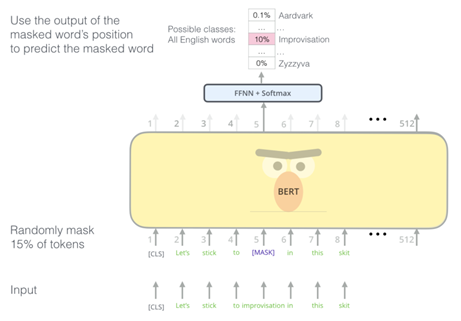
Token Embedding: Word Piece (OOV, sub-word, char)

Segment Embedding: sub-word를 하나의 문장으로 다시 묶는 정체성을 주는 과정(하나의 세그먼트에서 보통 sub-word 최대 60개 안 넘어감, 제한을 512개로 설정했고 문장 구분을 위해 입력 시 [SEP]를 이용한다.

Position Embeddings: BERT는 Transformer의 인코더 부분만 이용하는데 따라서 Position 정보가 필요하다.

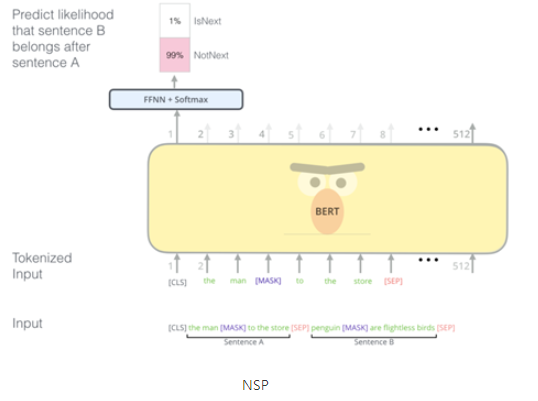
이제 LM을 실행하면 된다.(왼쪽에서 오른쪽으로 다음 단어 예측 기계 or 예측할 단어 좌우 문맥 고려하여 예측(word2vec처럼)) 그런데 BERT는 언어의 특성을 잘 학습하도록 MLM, NSP 이용



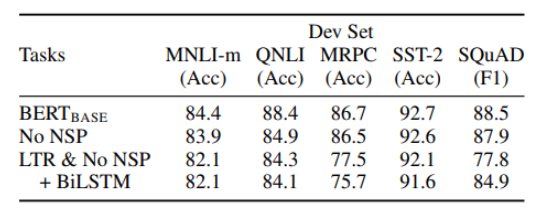


MLM(Masked Language Model): 입력 문장에서 임의로 토큰을 버리고(Mask), 그 토큰을 맞추는 방식으로 학습을 진행

Too little masking: Too expensive to train / Too much masking: Not enough context



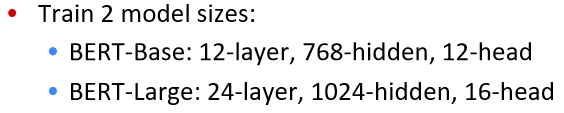
NSP(Next Sentence Prediction): 두 문장의 순서를 예측(두 문장이 주어졌을 때), NLI(Natural Language Inference)나 QA 같은 Task를 위해 필요하며

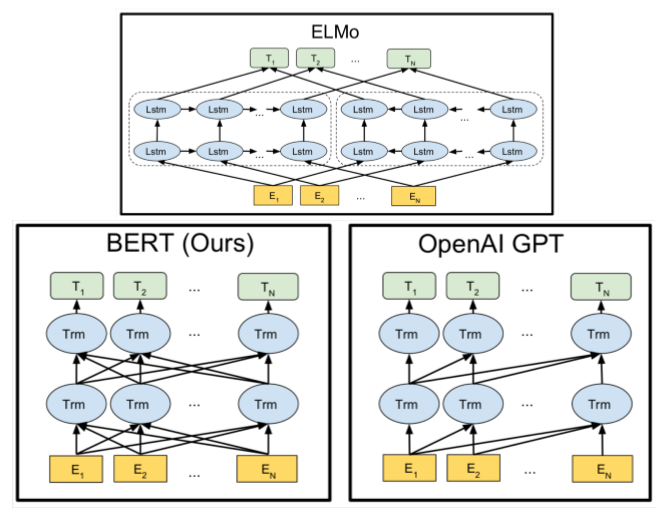


No NSP: MLM만 적용

LTR & No NSP: MLM과 NSP 둘 다 적용하지 않은 기존 좌우 모델

BERT base(MLM + NSP)





학습된 언어모델을 다양한 Task에 수행한다.

BERT의 한계점: 일반 NLP에서는 잘 작동하지만 아직 Bio, Science, Finance 등의 특정 도메인에서 잘 적용이 안된다. 사용 단어들이 다르고 언어의 특성이 다르기 때문이다.

따라서 특정 분야의 특성을 포함한 언어 데이터로 finetuning이 필요하다.

