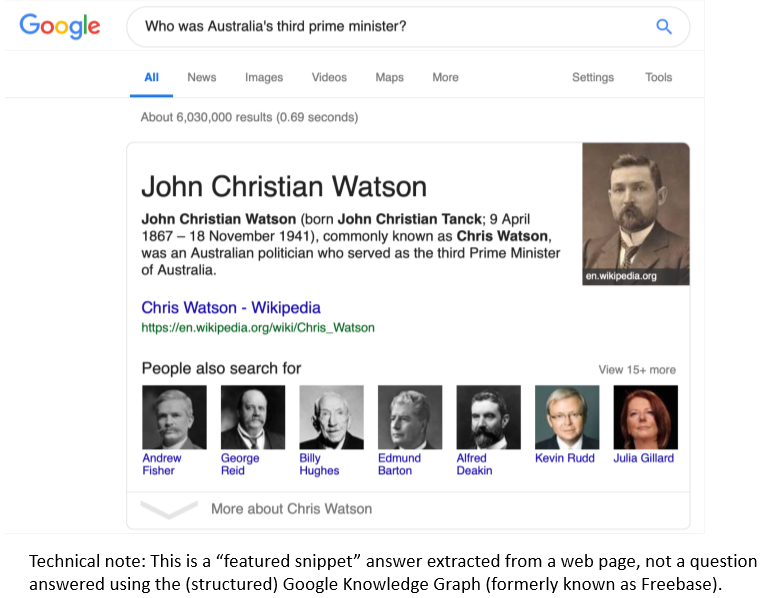
**Question Answering**



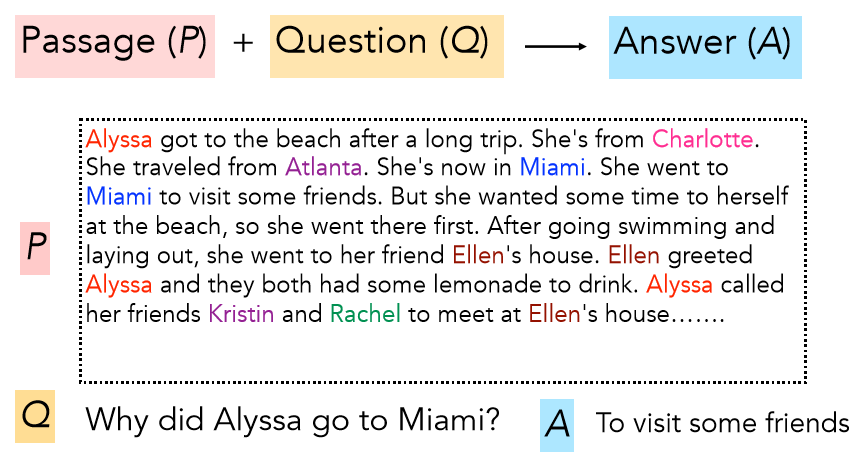
QA with NLP vs Google KG Answering 등이 있다.

방대한 양의 full-text documents 에서 관련있는 문서 찾기 힘듬.  
또한 question에 대한 answer로 그 관련있는 문서를 받는 과정을 원함.

두 가지의 방법

1. Finding document that might contain an answer(traditional)
2. Finding answer in a paragraph or a document ?? 무슨 뜻인가

**Reading Comprehension/Machine Comprehension**



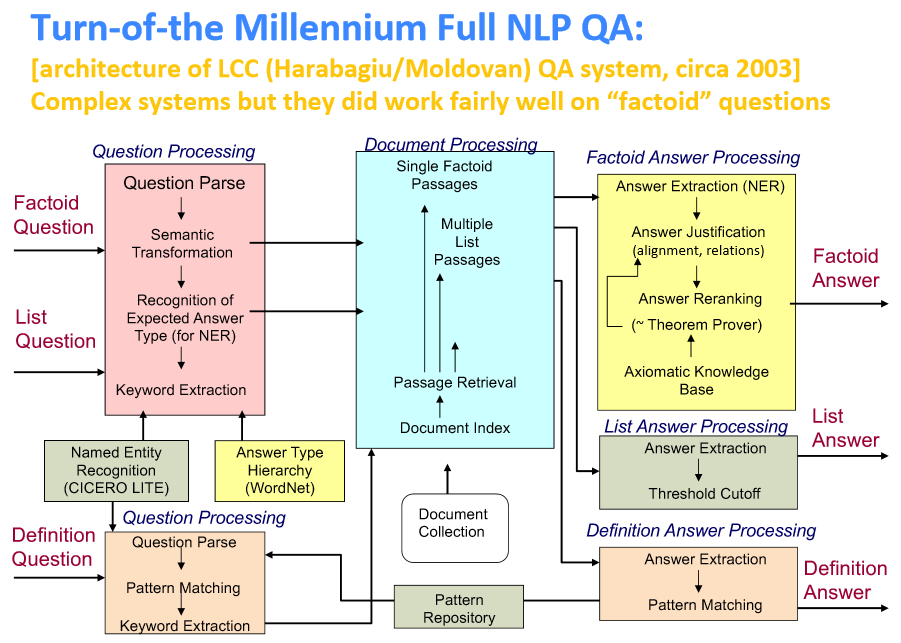
와 같이 MCTest 란 대회가 있는데 P, Q가 주어졌을 때 A를 하는 것이 목표.  
현재 비슷한 데이터셋이 많이 있다.

아래 그림은 LCC의 QA 아키텍쳐이다.

매우 복잡하고 인간이 직접 설계한 많은 components가 존재하며 그것이 각각 따로 기능하여 합쳐지는 형태이다.

Neural Net이 나오기 전까지는 이런 형태로 많이 처리하였으며 매우 복잡하다.

IBM의 DeepQA(2011)도 여러 methods를 ensemble 했으며 이거보다 더 복잡한 형태를 보이고 있다.

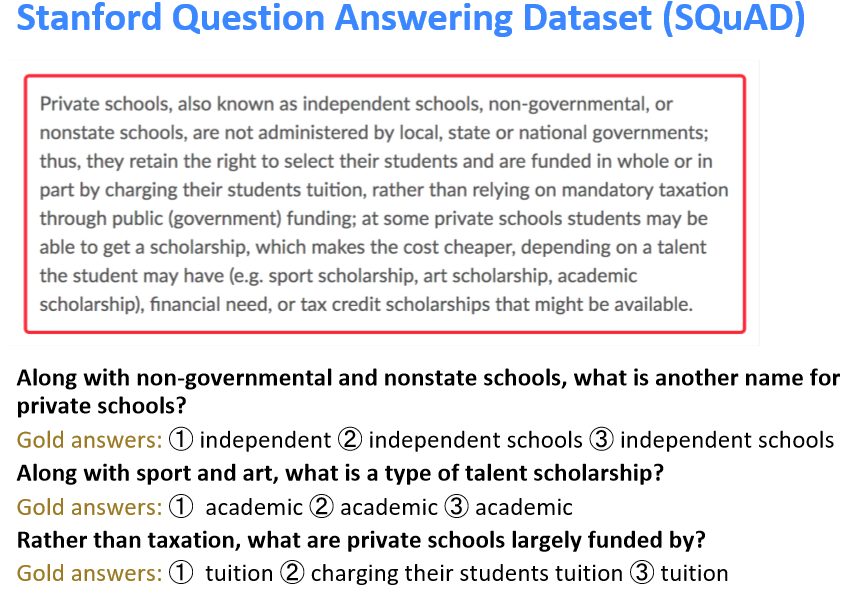


**Stanford Question Answering Dataset(SQuAD, 2016)**

Questions는 각각 passage 한 개씩 가진다.(Wikipedia에 있는) 그리고 Question이 존재한다.  
그리고 이 시스템의 목표는 question에 답을 하는 것이다.(answering)

1.0 (first version)에는 100k examples (한 paragraph 당 5개의 질문) / answer이 paragraph의 span으로 존재(따라서 yes or no, count question에 답할 수 없다.) / 따라서 a.k.a extractive question answering 이라고도 한다.

하지만 아래 그림처럼 span의 길이가 다를 수 있다.(사람마다, 기계마다)  
따라서, 그 점을 보완하기 위해서 answer의 variation을 감안하여 정답을 3개로 할 수 있다.  
또한 이것은 evaluation을 할 때 쓰인다.



**Evaluation (v1.1)**

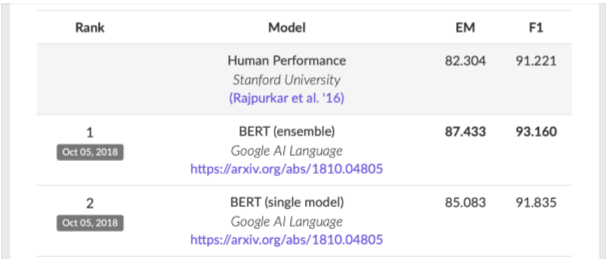
Exact match: 정답이 3개의 보기 중에 있다 = 1점 / 없다 = 0점

<https://sumniya.tistory.com/26> (F1 score 참고)

F1 score: each gold answer 와 system의 예측 span을 BoW로 바꾼다. 그 후 precision을 계산한다.(each system 예측 단어가 실제로 gold span 안에 있을 확률) 또한 recall을 계산한다. (gold answer의 단어가 system 예측 단어에 있을 확률) => harmonic mean F1 = 2PR/(P+R)로 마무리하면 F1 score 나옴. 3 gold answer에 대해 f1을 계산했고 그 중 가장 큰 값을 선택하도록 한다. 그리고 모든 question에 대해 평균을 내면 final f1 score가 나오게 된다.

이 방법은 완전히 정확할 때를 Exact match 처럼 따지는게 아니라 다양한 변수(line breaks 등등)를 따지면서 계산하기 때문에 좀 더 정확한 metric이 될 수 있다.

두 metric 모두 punctuation and articles(구둣점과 정관사)를 무시한다.

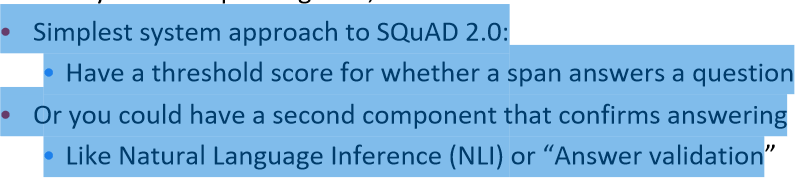


앙상블은 일단 간단히 여러 모델의 결과를 종합한 것이라 생각하면 좋다. 위 자료를 보면 인간보다 잘하고 있는 BERT의 모습을 볼 수 있다.

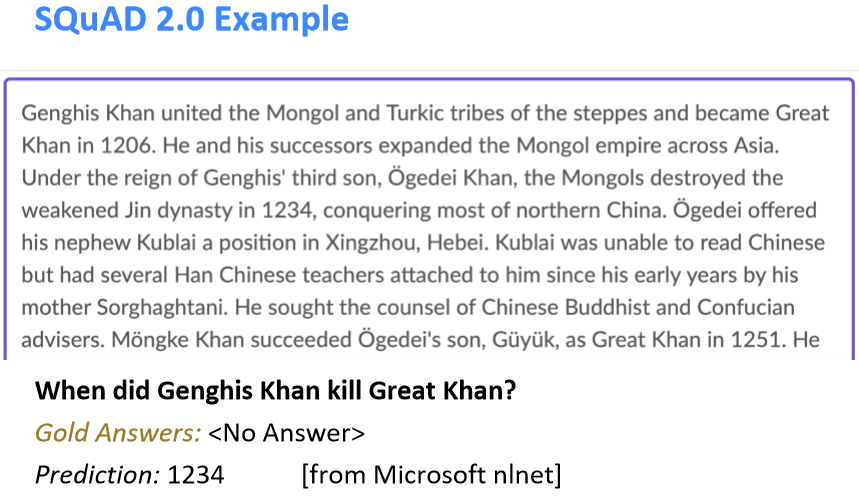
**SQuAD 2.0**

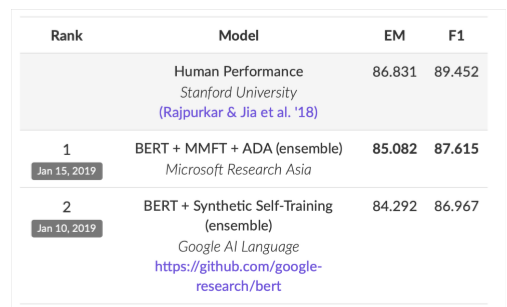
이전 버전의 SQuAD는 답을 찾으려고 하는 경향이 있었다. 하지만 더 일반적인 경우, 어려운 상황은 현실 세계에서 존재하기 때문에 Questions의 반은 Answer가 있고, 반은 Paragraph에 Answer가 없도록 설정하였다. (training questions는 1/3이 No answer)

따라서 metric 계산 시 Exact Match든 F1 score든 No answer의 데이터 테스트 경우 No answer로 예측할 경우 1, 아닐 경우 0으로 매겨진다.



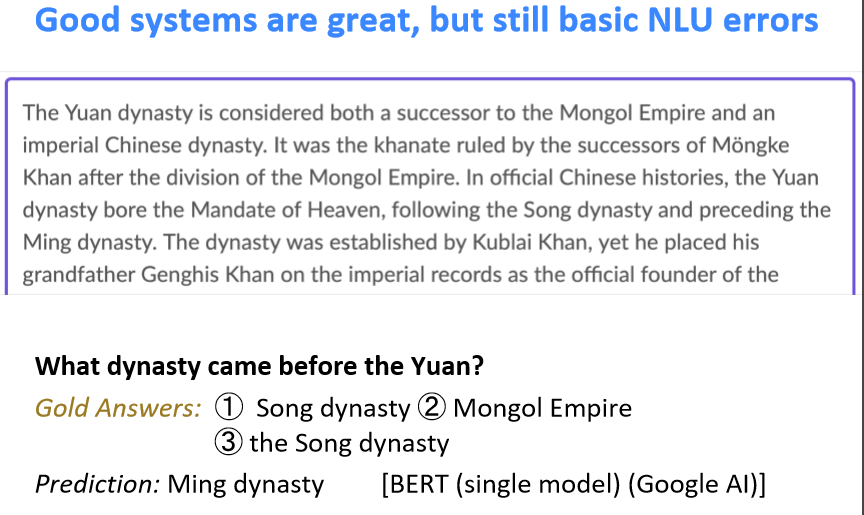
이 말은 무엇인지 모르겠다.





아직 인간을 넘진 못했다.

하지만 아직 좋은 System 이라도 Natural Language Understanding Errors를 보이고 있다.



SQuAD의 한계점

1. 오직 span-based answers(no yes/no, counting, implicit why)
2. Questions이 오직 Passage에 관련된다. (현실 세계에서는 필요 없는 질문일 수도 있다.)  
   또한 IRL(in real life)에서 얻는 것 보다 더 뭔가 형식적, 언어적, 문법적이다.(Wikipedia..)
3. 정답을 향한 다양한 정보의 결합과 활용을 할 수가 없는 구조의 데이터셋??(barely any multi-fact/sentence inference가 없다..?(단순 공동 참조인 coreference를 넘어선)??이거 헷갈

그럼에도 불구하고 많이 쓰이고 뭔가 비슷한 도메인에서든 아니든 처음에 초기화? 등에 사용할 때 좋은 데이터셋으로 작동할 수 있는 가능성이 있다.

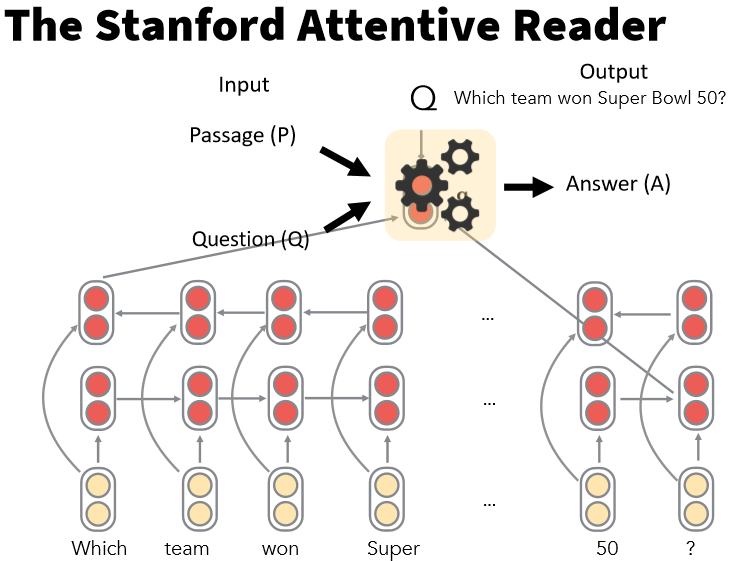
**이제 QA 모델**

**Stanford Attentive Reader = DrQA(Chen 2016, 2017)**

**Stanford Attentive Reader++ (2018..?)**

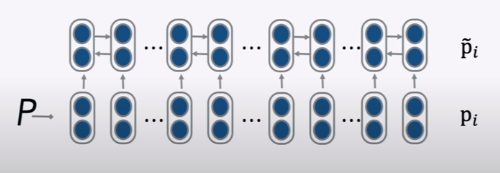
**Bidirectional Encoding + Bidirectional attention 이 핵심적인 기술!!!**

<https://ratsgo.github.io/natural%20language%20processing/2017/10/22/manning/>



GloVe로 word embedding(300 dimension)

그 다음 BiLSTM으로 Question Encoding을 한다. => representation of question!이 나온다.



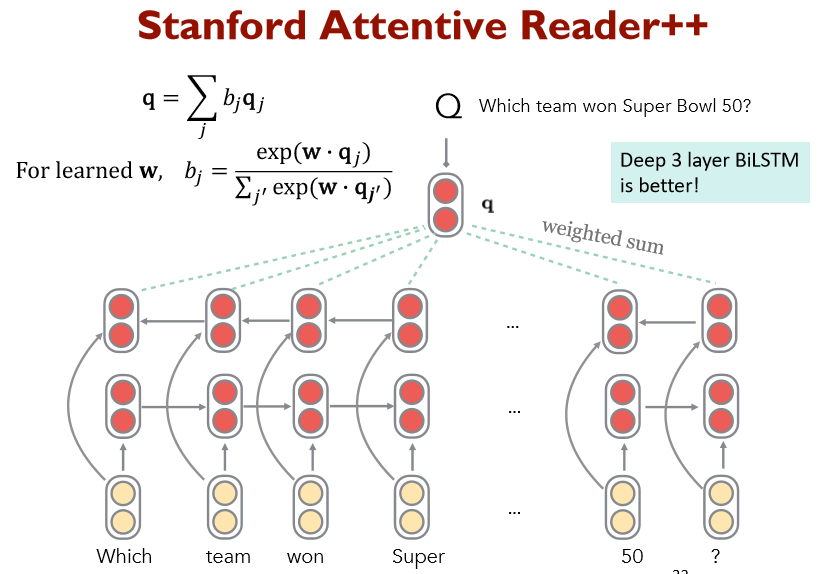
또한 passage(paragraph)에 대해서도 같은 작업을 해준다. 하지만 이 경우 양 끝단에서의 vector representation을 가져오는 것이 아닌 각 token의 output을 concatenate한 결과를 가지고온다. = pi~

그리고 나서 각 ai = softmax(q \* Ws \* pi~) 를 실행하고 passage의 모든 단어에 대한 attention weight를 구할 수 있다.

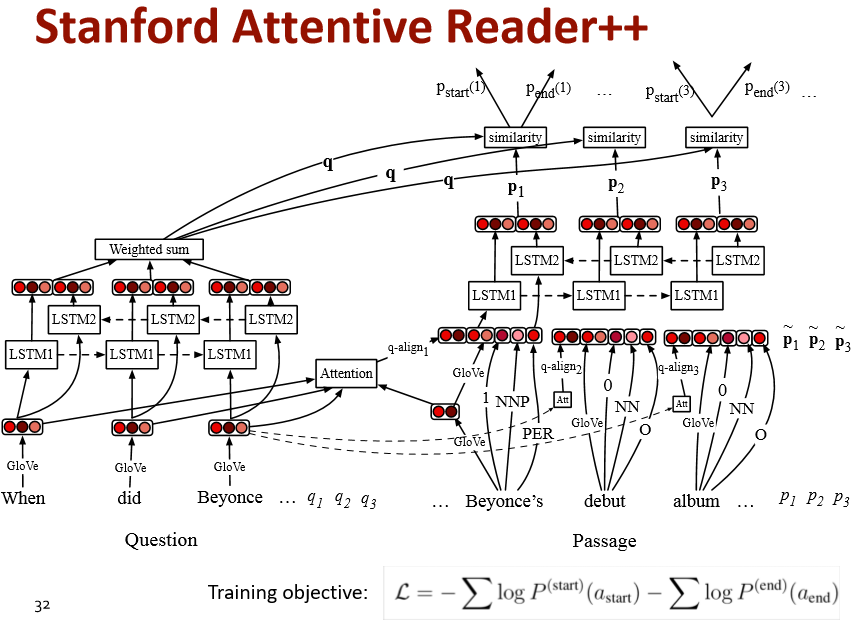
또한 최종 output vector는 ai \* pi 들의 합이 된다. 이 vector와 answer vector를 비교해 loss를 구하고 학습이 진행된다.

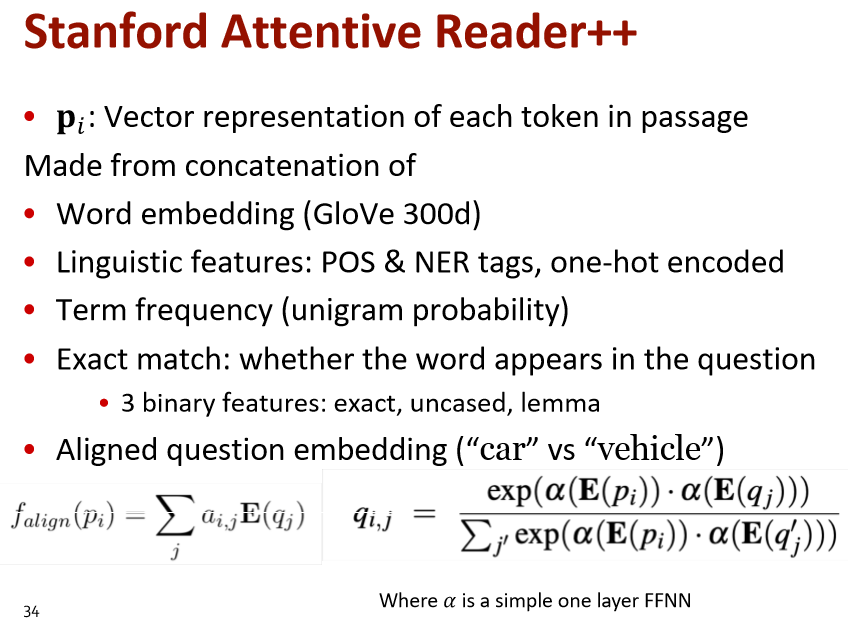
우리는 정답을 고를 때 start token과 end token을 구해서 그 구간을 찾는 것이 목표이다. 따라서 다른 weight지만 위 과정과 똑같이 해서 ai` = softmax(q \* Ws` \* pi~)를 해준다.

* 각 단어가 start가 될 확률 / 각 단어가 end가 될 확률을 가지고 종합하여 학습



위와 달리 ++에서는 q를 구하는데 있어서 방법이 바뀜.(BiLSTM Encoding의 방법 새롭게 변경)





너무 복잡하니 생략

**그 다음은 BiDAF(Bi-Directional Attention Flow for Machine Comprehension, ICLR 2017)**

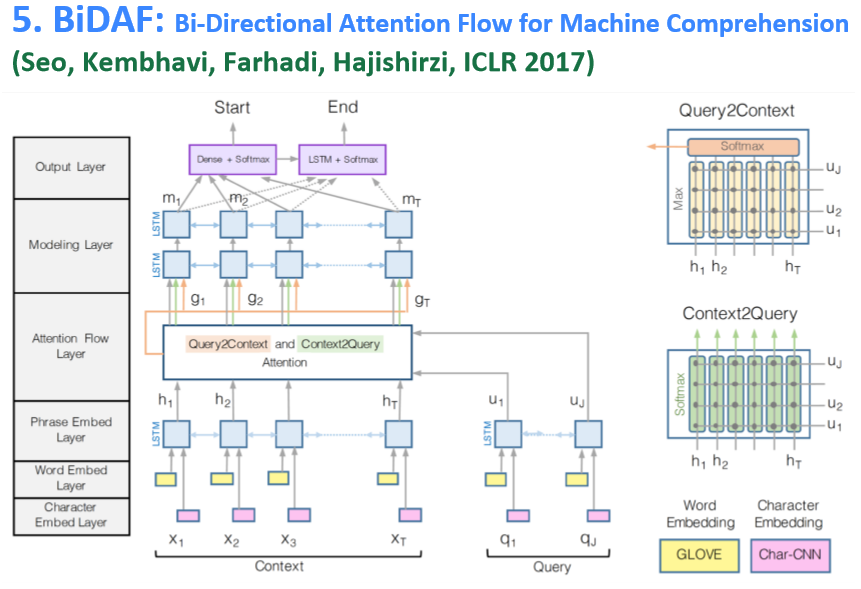
이 당시 앞서 말했듯이 Attention이 다양하게 쓰이기 시작했음(+MC task에 대해서도 적용하는 시도 많았음.)

대개 크게 3가지 컨셉이 있음

1. Document -> fixed sized vector로 압축 -> attention 계산(seq2seq with attention 같은 건가??)
2. 현재의 attention은 이전 timestep의 attention vector에 기반하여 만들어진 함수와 같다.
3. Query를 보고 주목해야할 Context의 부분을 찾는다..

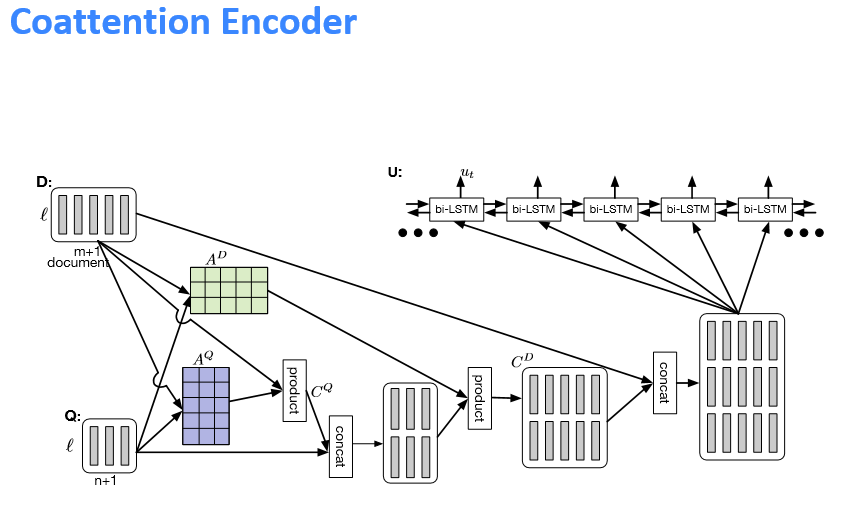
아직 무슨 말인지 이해하기 힘들지만 좀 더 보자.

여기서는 Query to Context 만이 아닌 Context to Query의 attention도 고려??



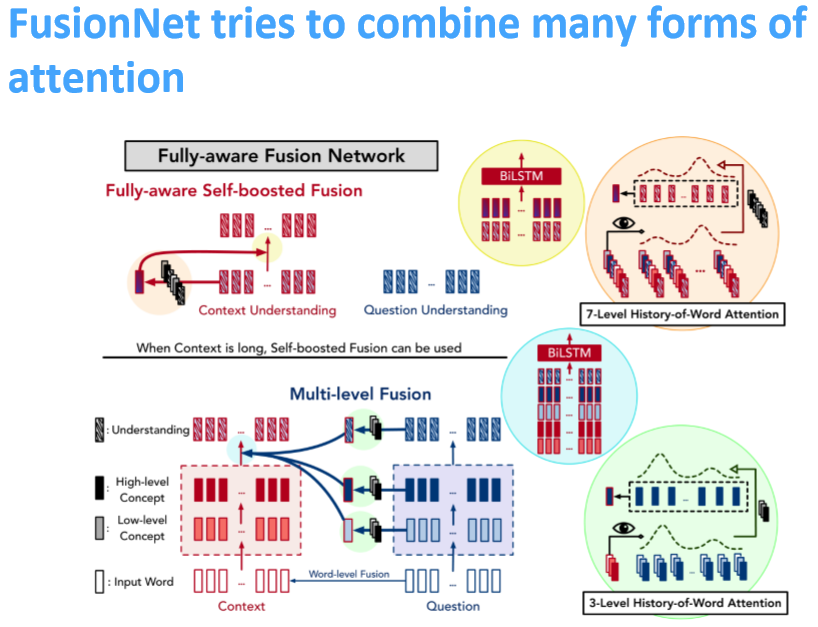
1. Character Embed Layer: Char-CNN으로 d차원 임베딩(각 단어에 ID vector 부여)
2. Word Embed Layer: Pre-trained Word 임베딩으로 d차원 임베딩(GloVe)
3. Contextual Embed Layer: BiLSTM으로 문맥 파악함. (DrQA의 Passage BiLSTM이랑 마찬가지로,, 방향 당 1개의 LSTM -> 각 step에서 d차원 output -> 2d 차원 output으로 concatenate 가능 -> GRU로 계산량 감소)-> h1~ht / u1~uj 가 다 2d차원이다.
4. 어텐션 양방향이라서 BiDAF  
   - Context의 어느 정보가 Query와 관련이 있는가를 학습 = Query2Context  
   - Query의 어느 정보가 Context와 관련이 있는가를 학습 = Context2Query  
   - 유사도 파악하기 위해 Similarity Matrix 사용, Context는 전체 Document로 t-th context word와 j-th Query Word 간의 Similarity 학습함.  
   - attention이 single vector로 요약되는 느낌(fixed-sized vector)이 아니라 정보 손실의 문제가 발생하지 않음
5. Modeling Layer: 정보를 정리하는 단계, Query와 Context간의 Interaction을 학습 -> BiLSTM(GRU)
6. Output Layer: Start, End 토큰을 찾게됨.

**Coattention Encoder(ICLR, 2017)**



얘도 파랑, 초록 보면 two-way attention! 자세한건 논문..

**FusionNet**



다양한 언텐션을 시도하고 결합하려 했다..!

48p) FusionNet이 Multi-level inter-attention 비스무리 이용 -> 나중에 더 발전 -> 빠르게 스킵했다.

50p) 이러이러한 것이 있다. Transformer 등 여러 방법 이용해서 contextual word representations 를 더 발전시켜 QA 문제 해결에 도움을 주었다.

52p ~ ) DrQA 데모와 더불어 Document Retriever 같은 것이 있는데 Question이 Query로 오면은 그에 따른 관련 문서를 어떤 방법으로든 Wikipedia에서 찾는 건가??(KG 같은 거 이용??) 그 후에 위에처럼 Reader Model(DrQA 같은)것을 이용