Improving Language Understanding by Generative Pre-Training – 2018, OpenAI

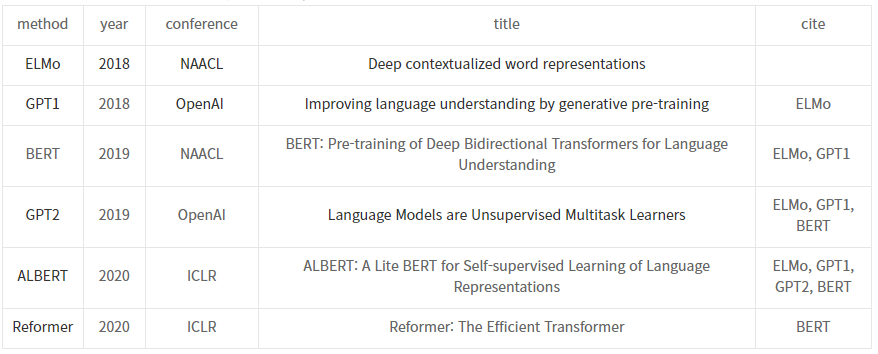
<https://www.quantumdl.com/entry/12%EC%A3%BC%EC%B0%A81-Improving-Language-Understanding-by-Generative-Pre-Training>

<https://davian-lab-junwoo.tistory.com/15>

<https://vanche.github.io/NLP_Pretrained_Model_GPT/>

<https://medium.com/@eyfydsyd97/%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-improving-language-understanding-by-generative-pre-training-gpt1-c65bed865990>

그전에 NLP에서 최근까지 제안된 pretrained model



**Improving Language Understanding by Generative Pre-Training**

**Abstract**

Labeling된 데이터는 극히 일부분

많은 데이터가 있더라도(label or unlabel), 그 데이터가 질이 좋아야 하며, 어떤 방식으로 학습 시키냐가 중요한 문제이다.

지금까지 많은 pretrain embedding을 사용했다.

여기서 확장해서 transfer learning을 한다면 더 좋지 않을까? 하지만 전이 학습도 신중히 잘 해야한다. CNN에서는 이미 전이 학습이 많이 사용되었다. CNN에서는 이미 어느정도 방법이 정해져 있었다. 피처 추출 방법 자체가 확실하게 정해져있고 여기서 확장돼 여러 학습 모델이 제시된다.

NLP에서는 이와 같은 방법이 정립되지 않았다.

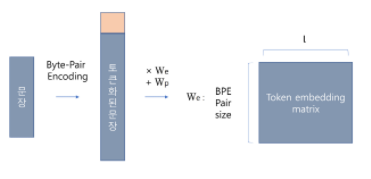
임베딩에 대한 문제도 있는데 문장(tokens)를 어떻게 임베딩할까도 논점이다. 이전에 Word-Level을 넘어 Context-Level로 임베딩하는 ELMo 등이 있었고, Unsupervised Learning을 어떻게 pretrain 할지도 현재 많은 논점이다.

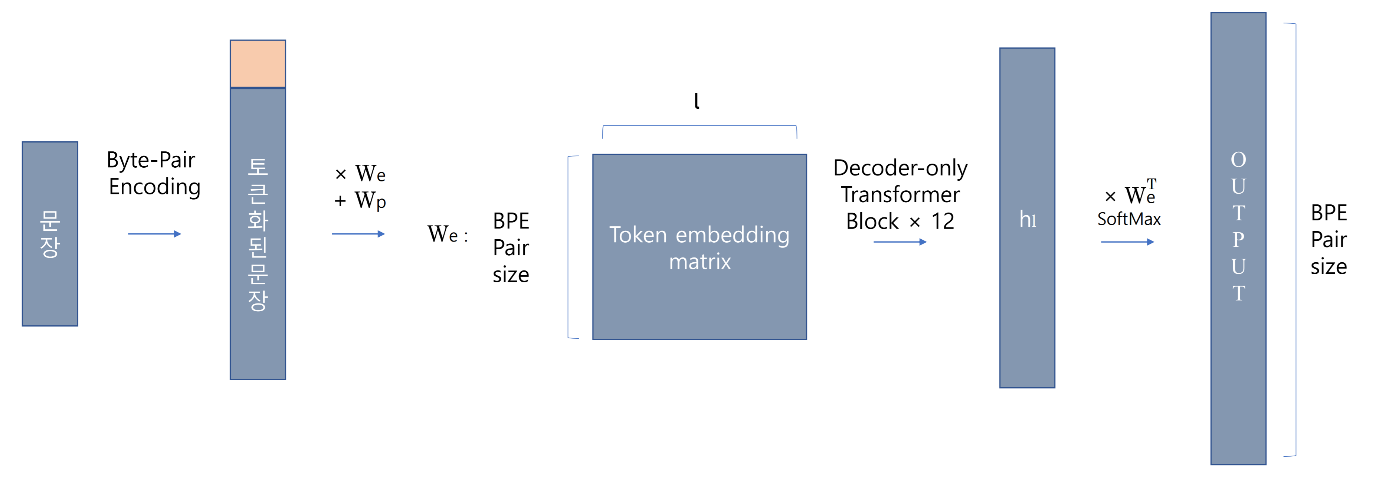
또한 supervised learning으로 fine-tune하여 특정 task에 맞게 모델을 어떻게 구성할 것인가도 논점이다.

**Contribution**

**Unsupervised**

여기도 ELMo 처럼 Unsupervised Learning + LM 이용 하지만 BiLSTM이 아닌 Transformer Block을 이용

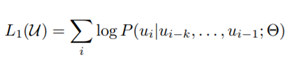




토큰화된 문장 -> 토큰 임베딩 matrix 과정에서 트랜스포머의 디코더 부분만 이용함.

대신 기존 인코더/디코더 각 6쌍 대신에 디코더 12쌍을 이용하는 것(12Layers)

요약: 기본 문장 -> BPE -> 토큰화된 문장 -> 트랜스포머 디코더들 -> 임베딩 matrix

(auxiliary objective 라고도함.)

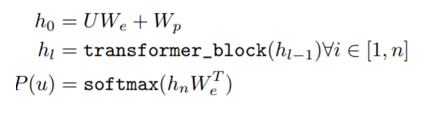
이식은 LM objective function -> 이전 단어보고 현재 단어 뭐일까 맞추는 방법, k는 context window



아래 식을 보면 U는 BPE 후의 토큰화된 문장, W-e는 토큰 임베딩 matrix, W-p는 포지셔널 임베딩 이다. 이렇게 h0은 첫번째 디코더의 입력으로 준비가 된다. 쭉쭉 디코딩을 하다보면 결론적으로 hn(n은 레이어수)이 나오고 마지막 디코더의 아웃풋이다.

그리고 마지막 WeT를 곱하고 softmax하면 backprop을 하기 위한 확률이 나오게 된다.

하지만 우리가 원하는 최종 결과는 We,Wp와 트랜스포머 블록이다. 임을 알 수 있다.(softmax+WeT는 학습을 하기 위한 장치일뿐, 앞에서 이미 We 한번 나와서 가능한듯)



**Supervised**

아래 그림이 모델 구조에 대한 내용이다.

Unsupervised 과정에서는 Global한 NLP 피처를 학습하게 된다.

임베딩할 때 BPE를 사용했었는데 이는 context에 대한 소실이 거의 없이 학습하는 representation 방법이다. 이후 디코더를 통해 task(specific task fine-tune)에 맞는 정답 피처를 추출하며 답을 도출한다.

기존 부분은 동결되는지 아닌지가 궁금하다. (절대 아니겠지)

즉, 트랜스포머를 치면은 각 timestep(위치)에서 을 구할 수 있다.(각 단어의 최종 임베딩이라 보면 될 것 같다.) 여기서 task에 맞게 예측을 해야 되는데 그때에 따른 specific-task weight가 Wy이다.(즉, Wy는 task마다 다르다.)

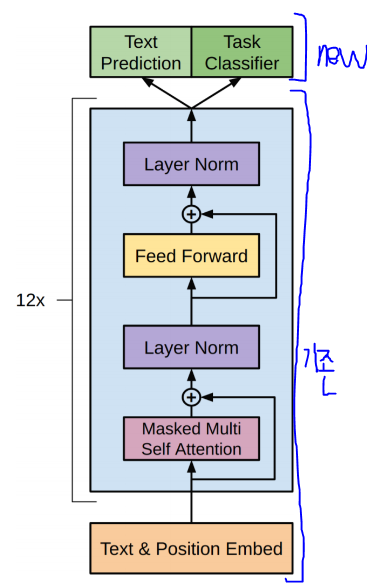
 따라서 이 확률을 최대로 하는 y가 예측 값이고 이걸로 objective를 구성하면 된다.



또한 다음과 같이 regularization 역할을 auxiliary objective가 할 수 있다.

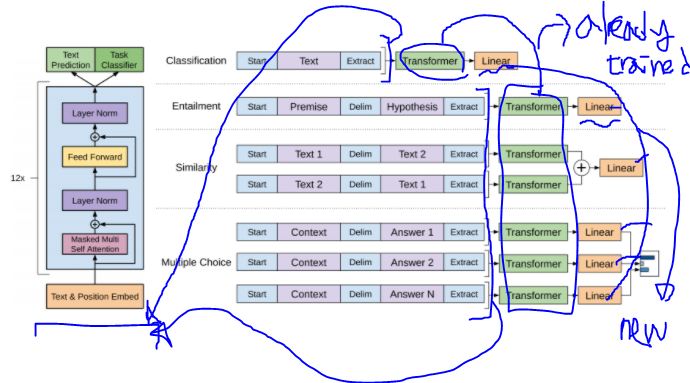


아마 근데 데이터셋이 labeled, unlabeled로 다르니까 end-to-end는 아니고 2번 진행하는 듯 싶다.



위 그림처럼

아래 그림처럼 각 task마다 불편하지만, fine-tune 모델 구조를 조금씩 다르게 해야한다.



**Experiments**

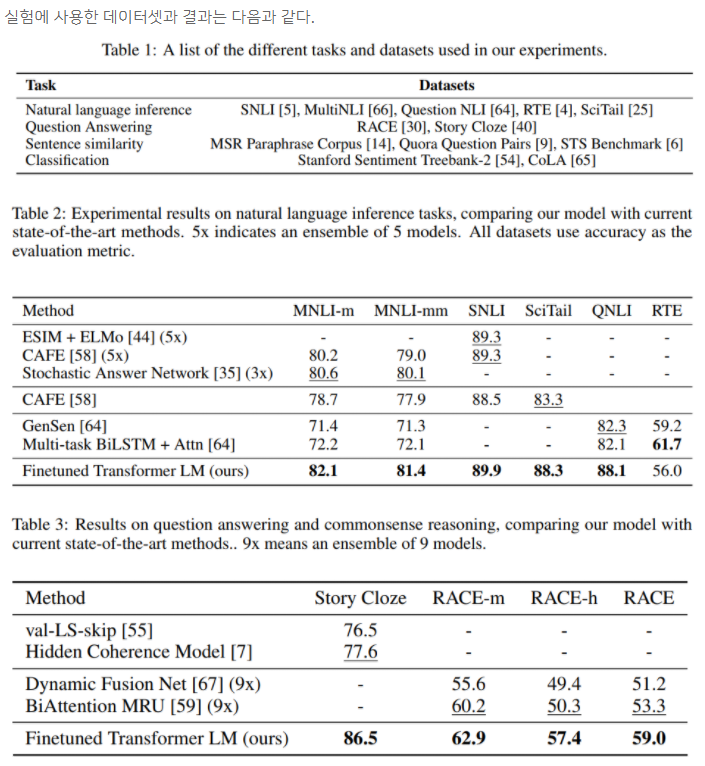
Pretraining 데이터로 BookCorpus Dataset

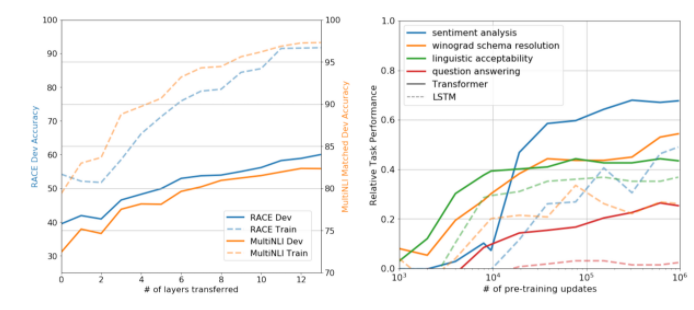
BPE로 임베딩을 했다는점

Activation function? 으로 GELU(Gaussian Error Linear Unit)을 사용한 점

Raw text를 클리닝할 때 fifty library 사용

일부 구두점, 스페이스를 표준화하고 spacy tokenizer를 사용했다.

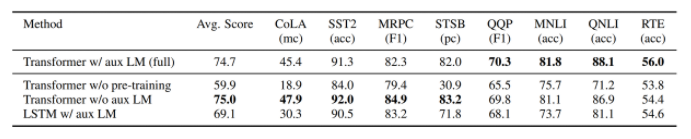




왼쪽은 디코더의 layer를 늘려가며 측정한 RACE, MultiNLI에 대한 정확도(각각 QA Task, textual entailment or NLI Task) => 12개 정도에서 Converge!

오른쪽은 각 Task에 대한 Transformer vs LSTM

아래는 Ablation Studies



이것은 Auxiliary Objective(objective regularization)가 없을 때, pretrain 자체가 없을 때 vs 있을 때

왼쪽 4개와 오른쪽 4개의 결과가 다른데 오른쪽 4개는 데이터셋 크기가 작기 때문이다. 만약 컸다면 더 높을 것으로 예상된다.

또한 pretrain이 없다면 성능 엄청 저하.

**Conclusion**

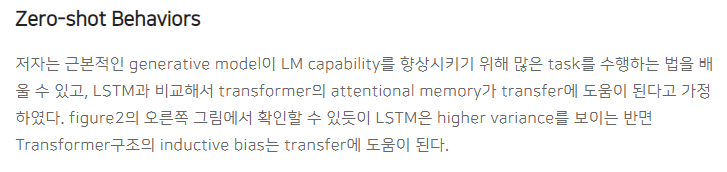
사실 GPT-1은 BERT에 비해 주목 그리 x, BERT가 좀 더 범용적이며 용이, 성능도 더 떨어진다는 소문(SQuAD 1.1, 2.0 결과를 숨긴 이유가 아닐까?)

하지만 그래도 디코더로 트랜스포머 사용한 것이 최초? -> BERT가 이 부분은 캐치한 듯

즉, generative pre-training + discriminative fine-tunning -> task-agnostic model(task를 가리지 않는?), 길고 연속된 텍스트로 이루어진 다양한 종류의 코퍼스로 pre-train -> 상당한 world knowledge와 long-range dependencies!

모델이 개개의 task를 풀 수 있도록함.

그 당시 12개 부문에서 9개 부문 SOTA!(BERT 나오기전)



이건 무슨말..?