GPT : Improving Language Understanding by Generative Pre-Training

Generative Pre-Training

- 생성적인 사전 훈련을 통한 언어 이해 향상
- 딥러닝 모델은 대부분 지도학습을 하므로 레이블링 필요
  - But, 구할 수 있는 대부분 데이터는 레이블링이 되어 있지 않음
- GPT는 레이블링이 되어있지 않은 데이터로 모델을 학습시켜

  ① 레이블링 데이터를 이용했을 때의 단점을 극복

  ② 사람이 알지 못하는 데이터의 특성까지 모델이 학습하게 함

  ③ 작은 수정만으로 효과적인 transfer를 하게 함으로써 높은 성능을 달성할 수 있음

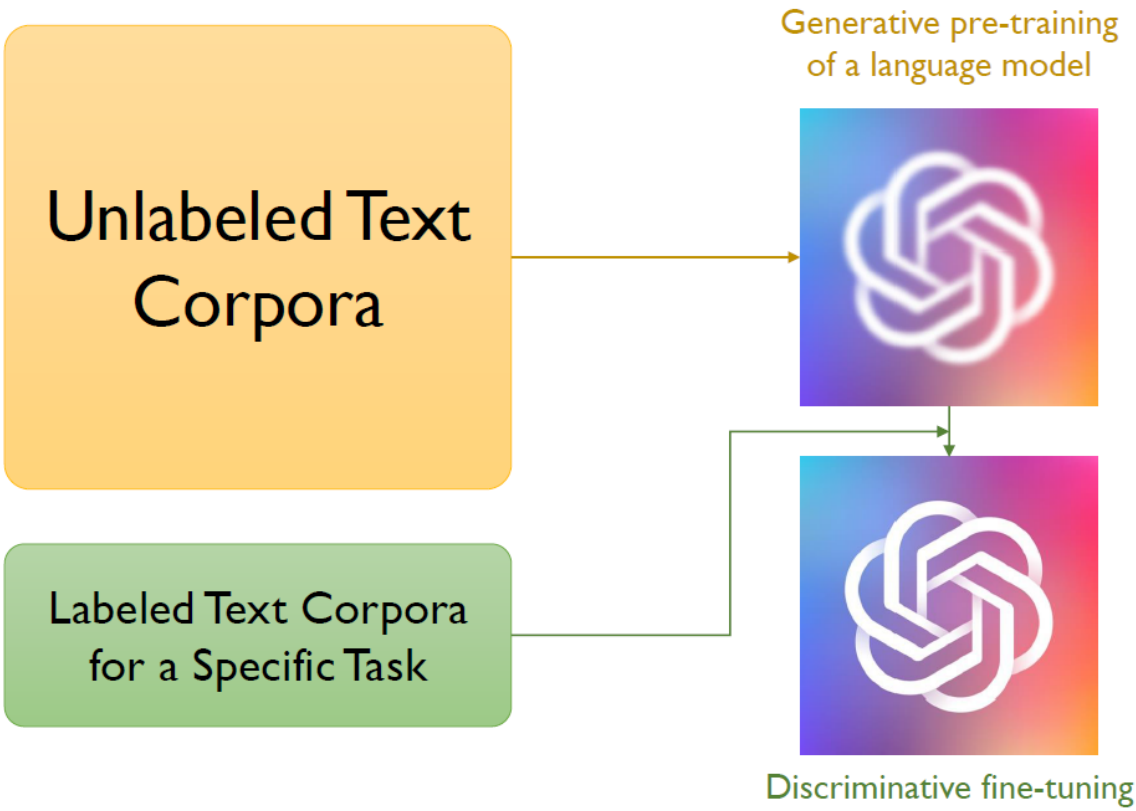
Unlabeled Data로 학습시키는 것의 문제점

1. 결과물을 transfer하기 위한 Text representation을 학습시키는 것에

   어떤 optimization objective가 효과적인지 불분명함
2. 학습된 특징을 target task에 어떤 방식으로 transfer할지 명확히 정의되어있지 않음

GPT

- 제안: GPT에서는 unsupervised pre-training과 supervised fine-tuning을 합친 **semi-supervised** 방식
- 목표: **다양한 task에 대해 작은 변화만으로 적용**시킬 수 있는 representation을 학습하는 것



- 대량의 unlabled data와 task에 알맞는 labeled data 가정
- **unlabled data로 모델의 초기 파라미터를 학습**하고, 최적화된 파라미터를 target task에 맞게 **labeled data로 추가 학습**시킴

- 구조: Transformer 사용
  - 기존 RNN과 비교했을 때, 멀리 떨어진 요소들 사이의 의존성 학습
  - 정확히는 Transformer의 decoder block만 사용
  - Transfer의 장점: Task에 특정한 입력을 생성해서 사용한다는 것

    → 다양한 Task에 해당 모델을 가지고 조금만의 변화로 Fine-Tuning

Framework(학습)

① Unsupervised Pre-Training

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

k = context window size / 파라미터는 SGD를 통해 훈련

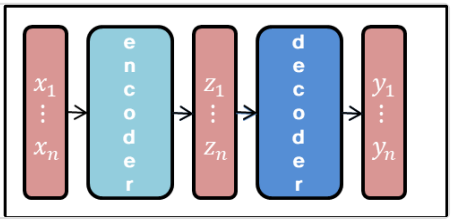
- 토큰으로 이루어진 코퍼스 v = {u1, u2, ..., un)가 주어지면 Unlabeled된 데이터에 대해서 **likelihood를 최대화하는 방향으로 학습을 진행**하는 standard language modeling objective 사용
- 학습을 위한 languge model로 transformer의 변형 중 하나인 multi-layer transformer decoder 사용
- 입력 문맥 token에 multi-headed self-attention을 적용 후, 목표 token에 대한 분포를 얻기 위해 position-wise feedforward layer 적용

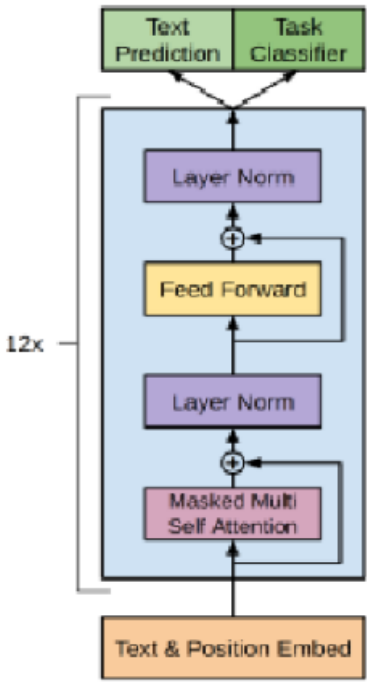
▼ multi-headed self-attention, position-wise feedforward

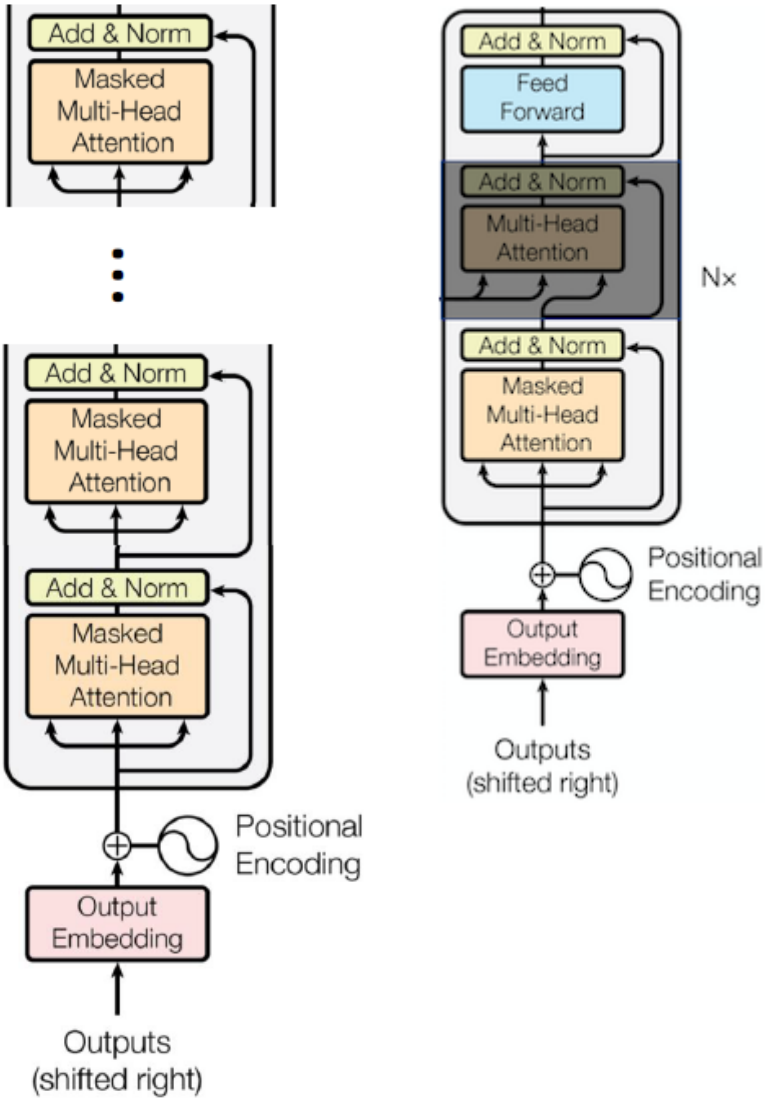
Attention is all you need paper 뽀개기

Transformer paper review

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







- 1: transformer의 decoder / 2, 3: GPT의 decoder
  - **multi-head attention이 GPT에서는 없어짐** → multi-head attention은 decoder의 self-attention과 encoder의 output으로 attention을 구하기 때문에, GPT에 없는 것이 당연함

$$h_0 = UW_e + W_p$$

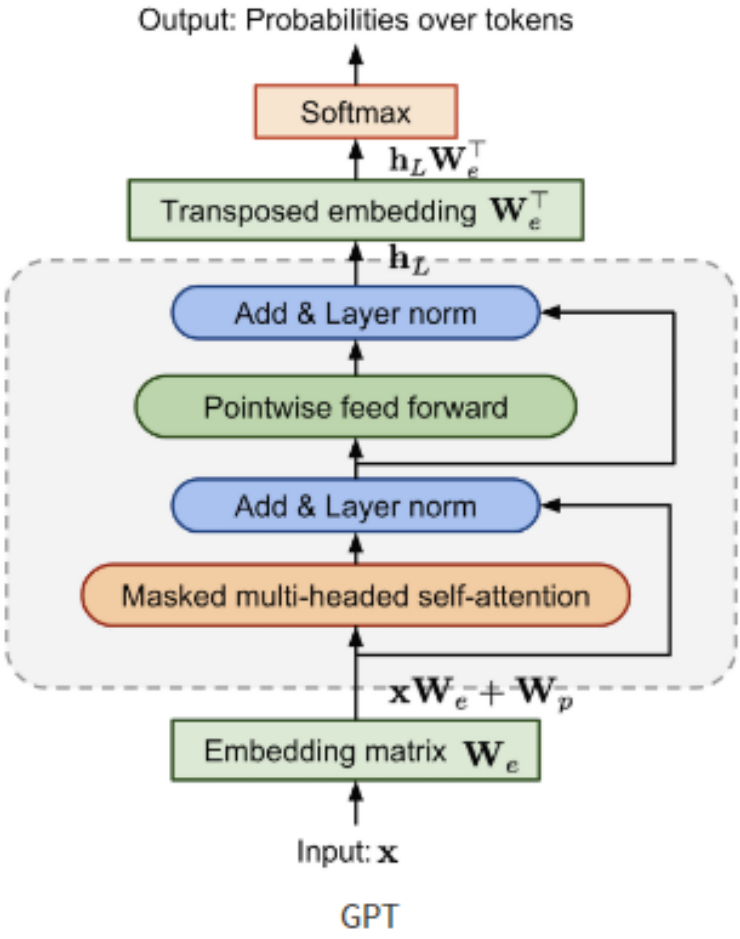
$$h_l = transformer\_block(h_l - 1), \quad \forall i \in [1, n]$$

$$P(u) = softmax(h_n W_e^T)$$

- U: 토큰의 context vector
- We: Token Embedding Matrix
- Wp: Position Embedding Matrix
- transformer: Decoder 부분만 사용 (Decoder만 12개로 구성)

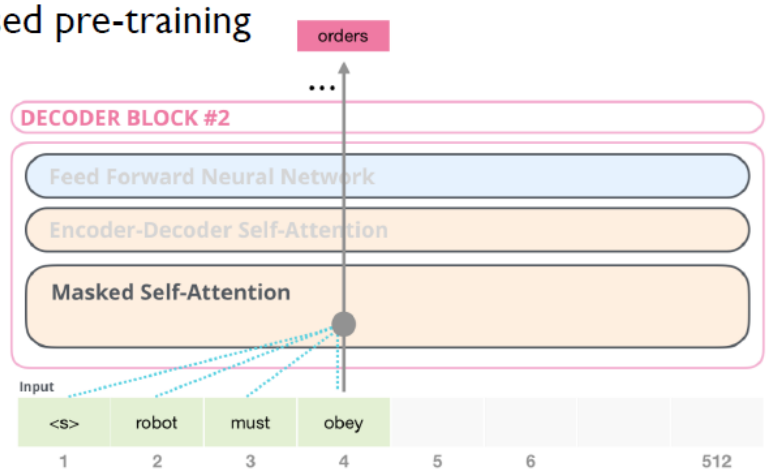
1) corpus U를 임베딩한다. We를 통해 워드 임베딩하고, 그 결과값에 Wp을 통해 구한 포지셔닝 임베딩 값을 더한다.
2) 그 후, n개의 transformer의 decoder block에 통과시킨다. (본 논문에서는 12개)
3) 최종 결과에 WeT를 내적하고, softmax를 적용한다.

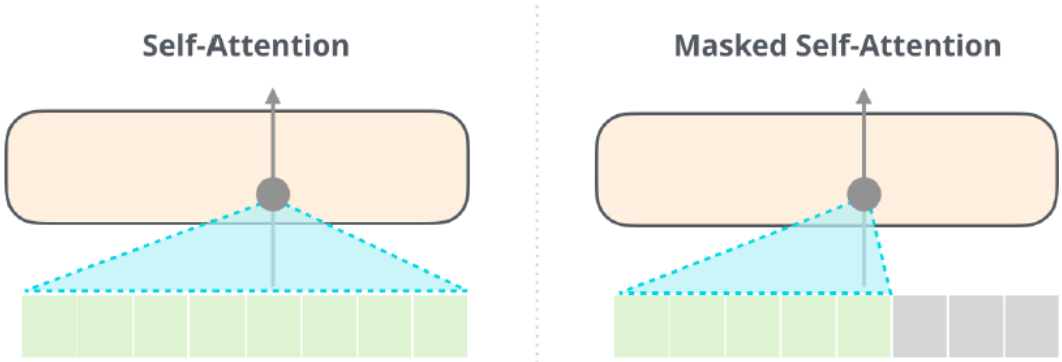
- 최종 모델 구조



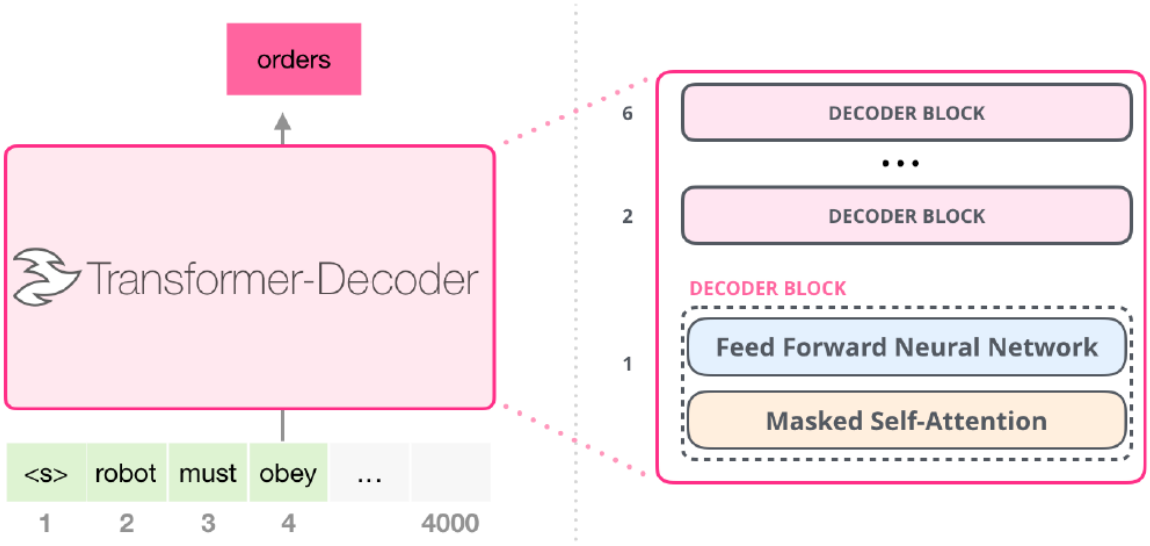
GPT

GPT: Unsupervised pre-training





✓ Decoder-only block



- Masked Self-Attention: 정보가 주어졌을 때, 해당하는 토큰의 바로 이전 것들까지만 사용
    - processing 하고자 하는 토큰 다음 시퀀스의 것들은 사용하지 않음

② Supervised fine-tuning

$$P(y|x^1,\ldots,x^m) = \text{softmax}(h_l^m W_y)$$

- Language model의 objective에 대해 모델을 pre-training 한 후, 각 Instance가 label y에 따른

  입력 토큰 x1,...,xm로 이루어져 있는 labeled dataset을 가지는 target task에 대해 파라미터를 조정

- 입력은 최종 transformer block의 활성값 hlm을 얻기 위해 Pre-trained 모델에 전달되고, label y를 예측하기 위해 parameter Wy와 함께 linear layer로 전달됨
  - 이 layer는 다음의 목적함수를 최대화하는 방향으로 학습됨
  - input token을 넣었을 때 라벨값을 반환할 likelihood를 의미함

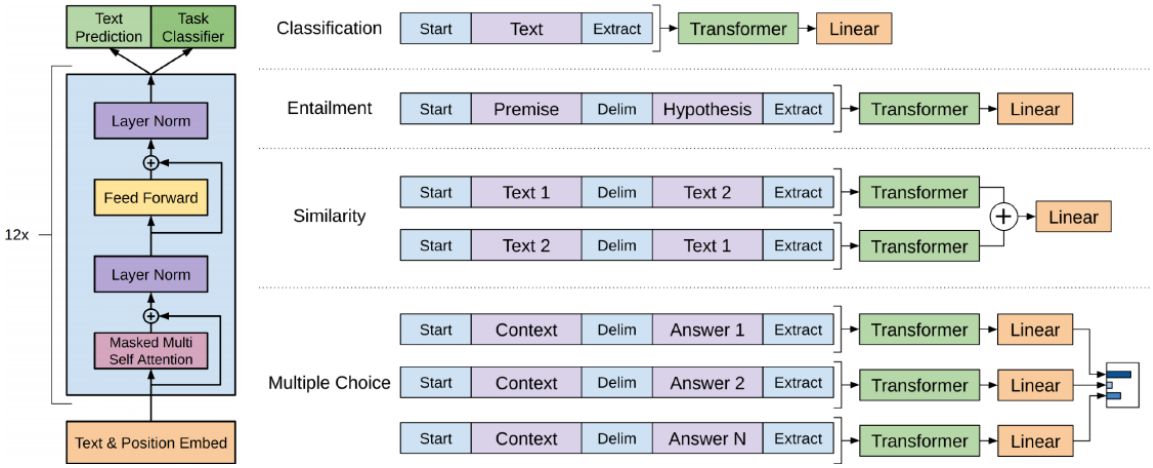
$$L_2(C) = \sum_{(x,y)} logP(y|x^1,\ldots,x^m)$$

- Fine-tuning에 보조 목적(Auxiliary objective)으로써 language model을 추가하는 것이 지도학습 모델의 일반화를 향상시키고(generalization) 모델이 빠르게 수렴하는 데(convergence) 도움을 주는 것을 확인하여 아래와 같은 Auxiliary objective를 사용

$$L_3 = L_2(C) + \lambda * L_1(C)$$

- 전체적으로 Fine-tuning 과정에서 추가로 학습해야하는 파라미터는 Wy와 Delimiter token들을 위한 임베딩뿐임

Task-specific input transformation



- Classification과 같은 task는 위에서 언급한 방법대로 Fine-tuning 진행가능
  - But, Question Answering이나 Entailment 같은 task는 여러 개의 문장이 필요하므로,

    delimiter로 각 sentence를 구분하여 하나로 연결하는 방식을 사용

- Textual Entailment: 전제(Premise)를 통해 가설(Hypothesis)의 참 거짓을 밝히는 Task
  - 전제 Premise와 가정 Hypothesis를 구분자 Delimeter로 연결
  - Delimeter로 나누어진 Premise, Hypothesis token을 concatenate하여 Fine-tuning 진행
  - ex) '가는 말이 고와야 오는 말이 곱다' (Premise)

    '말을 예쁘게 해야 한다' (Hypothesis)

    Premise와 Hypothesis 간의 관계의 Positive, Negative, Neutral 맞추는 것

- Similarity: 문장 간의 순서가 존재하지 않으므로, 가능한 2가지 순서

  [(문장 1,문장 2),(문장 2,문장 1)]를 모두 고려해야 함

  - 2가지 경우를 Input하여 독립적으로 얻은 결과 hlm을 최종적으로 element-wise addition한 결과로 Fine-tuning 진행

- Multiple choice (Question Answering):

- QA Task는 context document(문맥 문서) z에 대한 question(질문) q가 제시되고, 주어진 상황에서 가능한 다양한 Answer(가능한 답변) ak가 존재함
- 가능한 다양한 답변을 Delimeter인 $와 함께 [z; q; $; ak]로 concatenate하여 Input함
- 각각은 독립적으로 학습되며, 최종적으로 softmax를 통해 답변의 distribution을 계산

"GPT의 장점은 pre-train한 모델의 구조를 크게 바꾸지 않더라도 다양한 Task에 Fine-tuning이

가능하다는 것이다. 각 task에 적합하게 Input 데이터의 구조만 짜주면 된다!"

Results

Result 1: Natural language inference (자연어 추론)

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

- Natural language inference에서 거의 대부분 dataset에서 GPT가 큰 차이로 우수한 성능을 보임.
- 유일하게 저조한 성능을 보인 RTE dataset은 크기가 작은 데이터셋으로,
- 따라서 NLI task의 fine tuning은 상대적으로 데이터셋이 클수록 좋은 성능을 보임을 알 수 있음

Result 2: QA Task

Method	Story Cloze	RACE-m	RACE-h	RACE
val-LS-skip [55]	76.5	-	-	-
Hidden Coherence Model [7]	77.6	-	-	-
Dynamic Fusion Net [67] (9x)	-	55.6	49.4	51.2
BiAttention MRU [59] (9x)	-	60.2	50.3	53.3
Finetuned Transformer LM (ours)	**86.5**	**62.9**	**57.4**	**59.0**

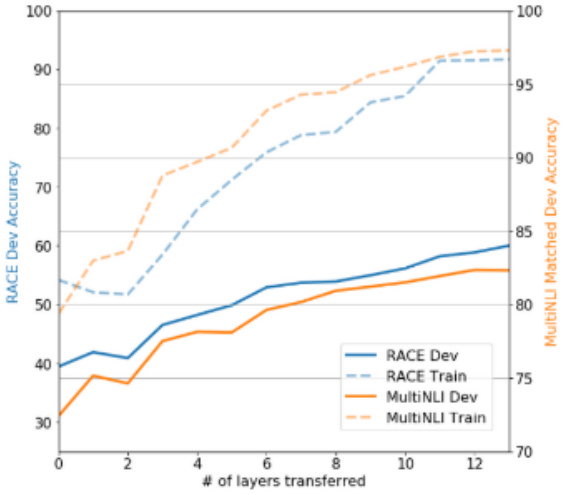
- 모든 데이터셋에서 QA Task에 대한 성능비교는 큰 차이로 GPT가 우수한 성능을 보임

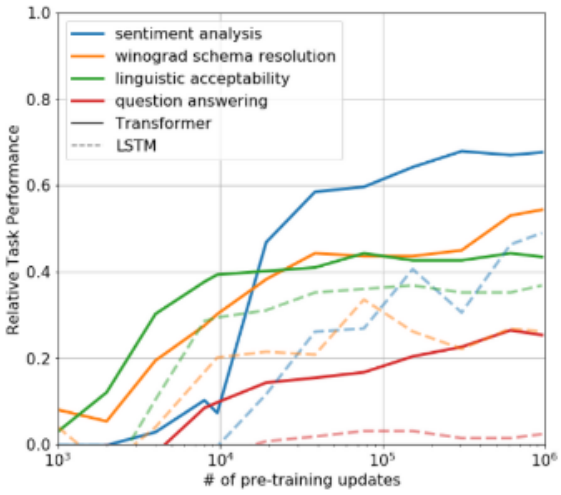
Result 3: Classification & Similarity Task

Method	**Classification**		**Semantic Similarity**			**GLUE**
	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	
Sparse byte mLSTM [16]	-	**93.2**	-	-	-	-
TF-KLD [23]	-	-	**86.0**	-	-	-
ECNU (mixed ensemble) [60]	-	-	-	81.0	-	-
Single-task BiLSTM + ELMo + Attn [64]	35.0	90.2	80.2	55.5	66.1	64.8
Multi-task BiLSTM + ELMo + Attn [64]	18.9	91.6	83.5	72.8	63.3	68.9
Finetuned Transformer LM (ours)	**45.4**	91.3	82.3	**82.0**	**70.3**	**72.8**

- 대부분의 실험에서 GPT가 우수한 성능을 보였으며,
- 특히, Attention을 더한 ELMO에 비해서도 좋은 성능을 보였다는 점이 주목됨

Result 4





- 왼쪽 그래프: Unsupervised pre-training에서 transformer layer 수에 따른 결과 비교
  - layer 수에 따른 유의미한 성능 향상

- 오른쪽 그래프: Transformer 유무와 pre-training에 따른 각각 Task의 Zero-shot 성능 비교
  - 실선: Transformer 사용 모델 / 점선: LSTM 사용 모델
  - 대부분의 Task가 Transformer를 사용했을 때 더 좋은 성능을 보이며, pretraining을 거듭할수록 차이가 더 커짐
  - 특히 Zero-shot에 대한 LSTM 성능의 분산이 더 컸다는 결과를 통해, pre-training이 전반적인 일반화 능력을 제공한다는 것을 알 수 있음

Result 5

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

- 세가지 조건변화(Ablation)에 따른 결과 분석

1) Transformer vs LSTM

LSTM에 비해서 Transformer를 사용한 모델은 평균 5.6의 성능 개선을 보여줌

2) with / without L1(C) auxiliary objective

L1(C) auxiliary objective는 NLI, QQP task에 도움을 주었으며,

특히 큰 데이터셋에서 성능개선을 가져옴

3) with / without pre-training

사전학습의 유,무에 대한 실험에서는 사전학습이 큰 성능개선(14.8%)을 가져온다는 것을 확인