BERT
Fine Tuning & Conc’

자여너학땅

BaekTree

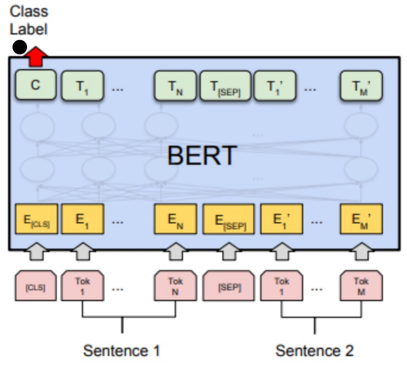
Purpose: 논문에 익숙해지자!

- from 부캠 강의
- to 논문 자체

4. Experiment(Fine Tuning)

- 4.1 GLUE
- 4.2 SQuAD 1.1
- 4.3 SQuAD 2
- 4.4 SWAG

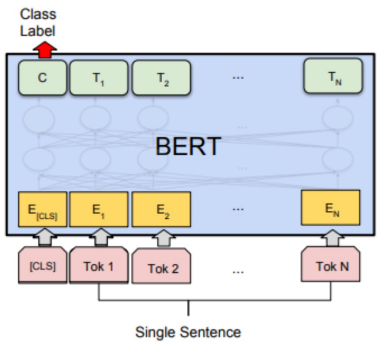
4.1 GLUE



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

sentenece pairs

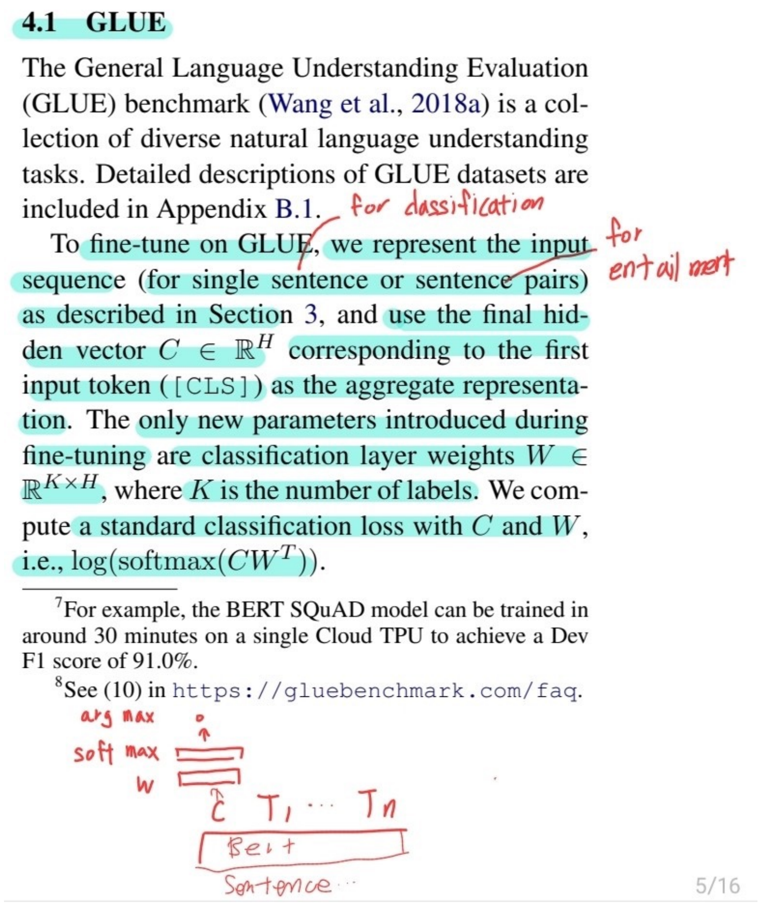
Task #2
(Next Sentence Prediction에 대응)



(b) Single Sentence Classification Tasks: SST-2, CoLA

single sentenece

Task #1
(Masked Language Model에 대응)



4.1 GLUE

The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018a) is a collection of diverse natural language understanding tasks. Detailed descriptions of GLUE datasets are included in Appendix B.1. for classification

To fine-tune on GLUE, we represent the input sequence (for single sentence or sentence pairs) as described in Section 3, and use the final hidden vector $C \in \mathbb{R}^H$ corresponding to the first input token (`[CLS]`) as the aggregate representation. The only new parameters introduced during fine-tuning are classification layer weights $W \in \mathbb{R}^{K \times H}$, where K is the number of labels. We compute a standard classification loss with C and W, i.e., $\log(\text{softmax}(CW^T))$. for entailment

[7]For example, the BERT SQuAD model can be trained in around 30 minutes on a single Cloud TPU to achieve a Dev F1 score of 91.0%.

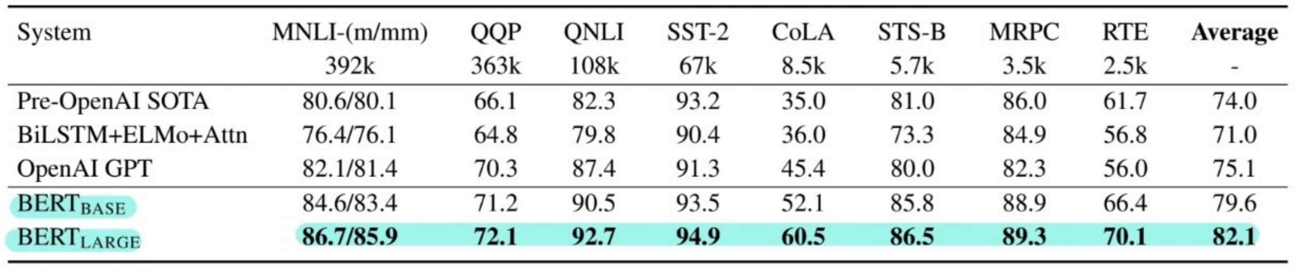
[8]See (10) in `https://gluebenchmark.com/faq`.

- GLUE tasks에 대한 fine tuning.
- 크게 단일 문장으로 수행할 수 있는 task와 두 문장의 관계 사이에서 수행하는 tasks으로 나눌 수 있다.
- 단일 문장 task: 이 문장 혹은 문단이 어느 주제 속하는가?
- 두 문장 task: 두 문장 사이의 논리 구조가 타당한지 검사하기

- 사실 상 classification 문제로 치환(reduce) 가능하다.

- 두 문장 task의 경우에 SEP 토큰을 사이에 넣어서 1개의 sequence으로 만든다. 문장 혹은 문단 전체가 어떤 주제에 속하는지 등의 classification 문제로 해결하기 때문에 문장 전체에 대한 이해가 필요하다.
- 그 결과가 output의 CLS 토큰에 있도록 학습함.
- 그래서 CLS 토큰을 class의 수 만큼 softmax으로 보내기 위해서 FC layer 1개짜리 넣으면서 차원까지 맞춤!
- 확률 값 가장 큰 값으로 Loss 계산해서 학습!

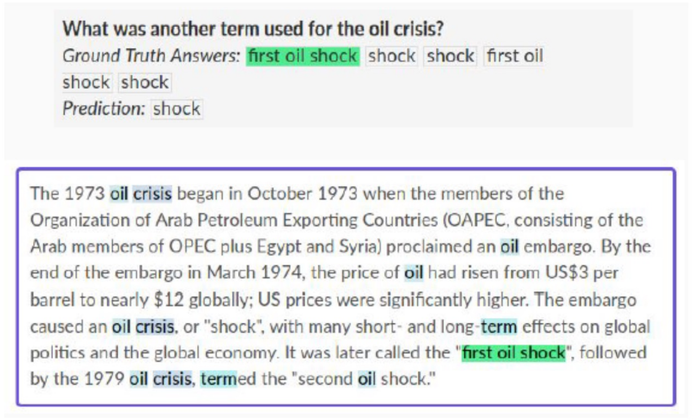
4.1 GLUE: Result



System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (`https://gluebenchmark.com/leaderboard`). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

4.2 SQuAD 1.1

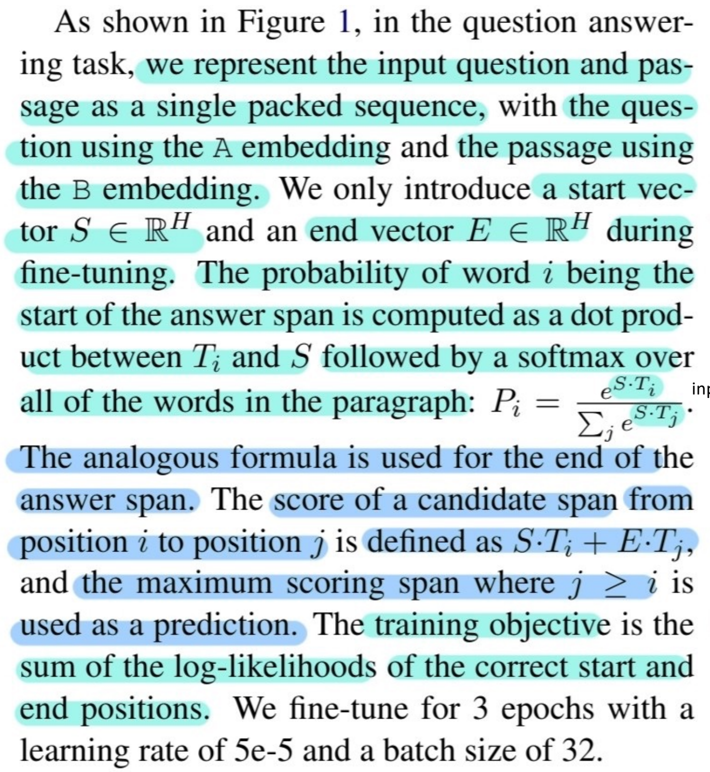


What was another term used for the oil crisis?
Ground Truth Answers: first oil shock | shock | shock | first oil shock | shock
Prediction: shock

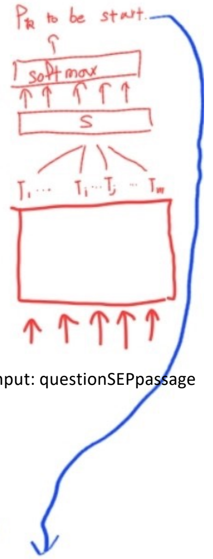
The 1973 oil crisis began in October 1973 when the members of the Organization of Arab Petroleum Exporting Countries (OAPEC, consisting of the Arab members of OPEC plus Egypt and Syria) proclaimed an oil embargo. By the end of the embargo in March 1974, the price of oil had risen from US$3 per barrel to nearly $12 globally; US prices were significantly higher. The embargo caused an oil crisis, or "shock", with many short- and long-term effects on global politics and the global economy. It was later called the "first oil shock", followed by the 1979 oil crisis, termed the "second oil shock."

Only new parameters: Start vector and end vector

$$P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$$



As shown in Figure 1, in the question answering task, we represent the input question and passage as a single packed sequence, with the question using the A embedding and the passage using the B embedding. We only introduce a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$ during fine-tuning. The probability of word i being the start of the answer span is computed as a dot product between T_i and S followed by a softmax over all of the words in the paragraph: $P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$. The analogous formula is used for the end of the answer span. The score of a candidate span from position i to position j is defined as $S{\cdot}T_i + E{\cdot}T_j$, and the maximum scoring span where $j \geq i$ is used as a prediction. The training objective is the sum of the log-likelihoods of the correct start and end positions. We fine-tune for 3 epochs with a learning rate of 5e-5 and a batch size of 32.


P_k to be start.
softmax
S
$T_1 \cdots T_i \cdots T_j \cdots T_m$
input: questionSEPpassage

$\log \hat{S}^* + \log \hat{E}^*$

- 문단 주고, 질문 주면, 거기에 해당하는 답변을 문단에서 찾는 task.
- input: question + sep + passage으로 넣는다.

- 정답을 문장 자체에서 찾는다. 따라서 위치만 가지고 잘 학습하는지 판단해서 loss을 만들면 된다.
- start 위치와 end 위치만 파악하면 된다.

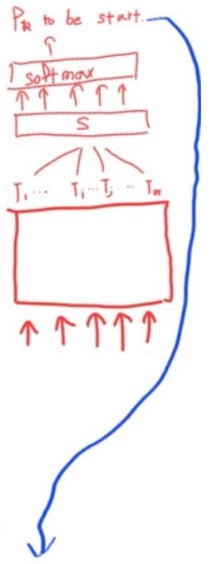
- 각 output token에 S토큰와 내적. S의 차원이 output token과 동일하게(사실 버트 전체에 대해 동일한 그 차원 H).
- 그래서 각 차원과 내적한 결과는 스칼라...이지만 그냥 단순하게 내적한 결과들을 softmax에 넣는다. 그러면 어차피 차원이 토큰의 수와 똑같음.

end 위치도 마찬가지.

- 확률 값 가장 큰 위치의 log liklihood Loss 만들고 start와 end 합쳐서 학습.

4.2 SQuAD 1.1

As shown in Figure 1, in the question answering task, we represent the input question and passage as a single packed sequence, with the question using the A embedding and the passage using the B embedding. We only introduce a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$ during fine-tuning. The probability of word i being the start of the answer span is computed as a dot product between T_i and S followed by a softmax over all of the words in the paragraph: $P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$. The analogous formula is used for the end of the answer span. The score of a candidate span from position i to position j is defined as $S \cdot T_i + E \cdot T_j$, and the maximum scoring span where $j \geq i$ is used as a prediction. The training objective is the sum of the log-likelihoods of the correct start and end positions. We fine-tune for 3 epochs with a learning rate of 5e-5 and a batch size of 32.



$\log \hat{S}^* + \log \hat{E}^*$

- loss

```
def compute_loss(logits, positions):
  one_hot_positions = tf.one_hot(
      positions, depth=seq_length, dtype=tf.float32)
  log_probs = tf.nn.log_softmax(logits, axis=-1)
  loss = -tf.reduce_mean(
      tf.reduce_sum(one_hot_positions * log_probs, axis=-1))
  return loss
```

```
start_loss = compute_loss(start_logits, start_positions)
end_loss = compute_loss(end_logits, end_positions)

total_loss = (start_loss + end_loss) / 2.0
```

https://github.com/google-research/bert/blob/master/run_squad.py

- logits: S 혹은 E와 내적한 벡터. 토큰 수와 동일한 차원.
- position: 실제 어느 위치가 Start 혹은 End인지 값 들어있음.
- position을 ont hot vector으로 만들고

- 내적 값을 softmax으로 확률화.

- 각 확률(예측 확률)과 실제 one hot vector으로 loss 계산.

- start와 end에 대한 각각의 loss 합쳐서 평균 낸다. 왜? 다음 슬라이드에 나옴.

tf: squad 데이터 만들기

```python
examples = []
for entry in input_data:
  for paragraph in entry["paragraphs"]:
    paragraph_text = paragraph["context"]
    doc_tokens = []
    char_to_word_offset = []
    prev_is_whitespace = True
    for c in paragraph_text:
      if is_whitespace(c):
        prev_is_whitespace = True
      else:
        if prev_is_whitespace:
          doc_tokens.append(c)
        else:
          doc_tokens[-1] += c
        prev_is_whitespace = False
      char_to_word_offset.append(len(doc_tokens) - 1)
```

```python
for qa in paragraph["qas"]:
  qas_id = qa["id"]
  question_text = qa["question"]
  start_position = None
  end_position = None
  orig_answer_text = None
  is_impossible = False
  if is_training:

    if FLAGS.version_2_with_negative:
      is_impossible = qa["is_impossible"]
    if (len(qa["answers"]) != 1) and (not is_impossible):
      raise ValueError(
          "For training, each question should have exactly 1 answer.")
    if not is_impossible:
      answer = qa["answers"][0]
      orig_answer_text = answer["text"]
      answer_offset = answer["answer_start"]
      answer_length = len(orig_answer_text)
      start_position = char_to_word_offset[answer_offset]
      end_position = char_to_word_offset[answer_offset + answer_length -
                                         1]
```

- char_to_word_offset: 0,1,1,1,2,2,2,2,2
- 각 char 마다 토큰 단위. 0은 1글자 토큰. 1~3까지가 하나의 단어 토큰. 4~8까지가 하나의 단어 토큰.
- 그래서 cahr_to_word_offset[answer_offset]하면... answer가 시작하는 char 위치에서 answer의 처음 단어의 index가 나온다. 이 예시에서는 2번째 단어부터 시작.

tf: 데이터 불러옴

```
tok_to_orig_index = []
orig_to_tok_index = []
all_doc_tokens = []
for (i, token) in enumerate(example.doc_tokens):
  orig_to_tok_index.append(len(all_doc_tokens))
  sub_tokens = tokenizer.tokenize(token)
  for sub_token in sub_tokens:
    tok_to_orig_index.append(i)
    all_doc_tokens.append(sub_token)
```

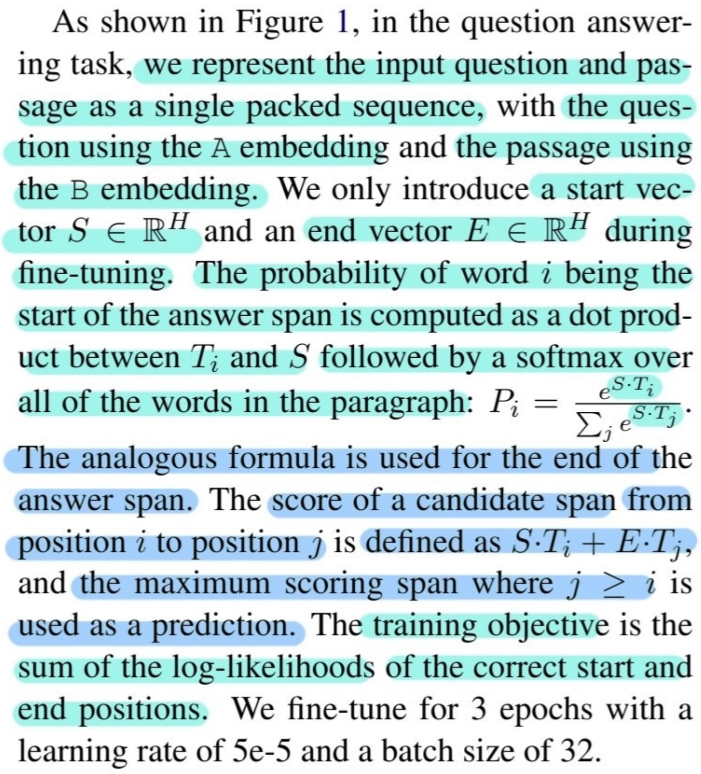
```
tok_start_position = None
tok_end_position = None
if is_training and example.is_impossible:
  tok_start_position = -1
  tok_end_position = -1
if is_training and not example.is_impossible:
  tok_start_position = orig_to_tok_index[example.start_position]
  if example.end_position < len(example.doc_tokens) - 1:
    tok_end_position = orig_to_tok_index[example.end_position + 1] - 1
  else:
    tok_end_position = len(all_doc_tokens) - 1
  (tok_start_position, tok_end_position) = _improve_answer_span(
      all_doc_tokens, tok_start_position, tok_end_position, tokenizer,
      example.orig_answer_text)
```

- 각 단어를 wordpiece 단위로 쪼갠다. 그리고 그 길이를 누적해서 저장.

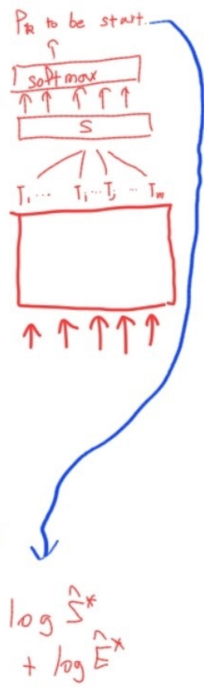
orig_to_tok_index = 0, 2, 5의 경우, 1번째 단어가 2개로 쪼개지고 2번째 단어가 3개로 쪼개진다.

orig_to_tok_index[example.start_position]
example.start_position은 answer가 시작되는 단어 토큰의 index을 반환한다.
orig_to_tok_index의 index 역시 각 단어 토큰의 index이므로...
반환하는 것은 answer 직전까지의 subword 수이다.

4.2 SQuAD 1.1: maximum scoring span



As shown in Figure 1, in the question answering task, we represent the input question and passage as a single packed sequence, with the question using the A embedding and the passage using the B embedding. We only introduce a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$ during fine-tuning. The probability of word i being the start of the answer span is computed as a dot product between T_i and S followed by a softmax over all of the words in the paragraph: $P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$. The analogous formula is used for the end of the answer span. The score of a candidate span from position i to position j is defined as $S \cdot T_i + E \cdot T_j$, and the maximum scoring span where $j \geq i$ is used as a prediction. The training objective is the sum of the log-likelihoods of the correct start and end positions. We fine-tune for 3 epochs with a learning rate of 5e-5 and a batch size of 32.



- https://ai.stackexchange.com/questions/11900/understanding-how-the-loss-was-calculated-for-the-squad-task-in-bert-paper/11947#11947

passage가 너무 길어서 sequence가 두개로 쪼개졌을 때.

- 하지만 결과는 하나여야 한다!
- passage > max_len 일때, input을 쪼갠다. sen = sen1 + sen2.
- CLS + Q + SEP + SEN1 + SEP
CLS + Q + SEP + SEN12+ SEP
- 여기에 경우의 수가 있음
- 1.첫번째 input에 정답이 있다.
- 2.두번째 input에 정답이 있다.
- 3.중간에 걸쳐서 정답이 있다.
- S 벡터와 각 토큰의 내적을 구함. 어디에 정답이 있을지 모름. sen1과 sen2에 대해서 모두 다 함.
- 가장 큰 것 고른다.
- End도 마찬가지. 가장 큰 토큰 고른다.
- no answer이면? 현재 sen1에서 없으면?
- label = 0이다. 그래서… no answer에 해당하는 0,0에 해당하는 logit이 클 수록 정답. 0에 해당하는 확률이 작을 수록 Loss차이로 gradient 계산 된다. Cross Entropy 때문에!!!

만약... 겹쳐져 있다면...? sen1과 sen2으로 갈라진 사이에 answer span이 있다면?

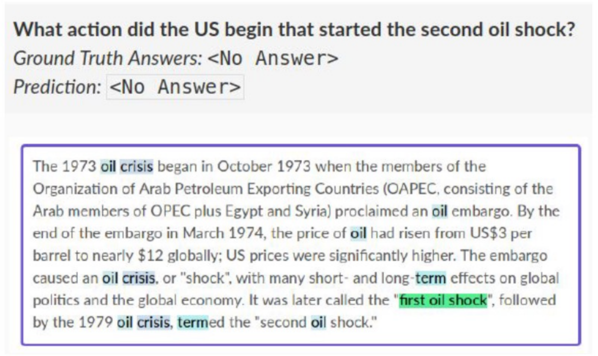
- tensorflow 버전에서는 겹쳐있으면 해결을 못하는 것처럼 보임.
- tok_start_position: answer 시작 위치
- tok_end_position: answer 끝 위치
- doc_start: 현재 잘린 문서 시작 위치
- doct_end: 현재 잘린 문서 끝 위치
- if not(tok_start_position >= doc_start and tok_end_position <= doc_end): answer 위치가 현재 doc 사이에 있지 않으면 answer 위치를 0으로 만든다. 겹쳐 있는 경우는 tok_start_position <= doc_start 와 tok_end_position <= doc_end이다. 그래서 아마 tf에서는 겹친 answer span은 처리 못하지 않을까...?

```
start_position = None
end_position = None
if is_training and not example.is_impossible:
  # For training, if our document chunk does not contain an annotation
  # we throw it out, since there is nothing to predict.
  doc_start = doc_span.start
  doc_end = doc_span.start + doc_span.length - 1
  out_of_span = False
  if not (tok_start_position >= doc_start and
          tok_end_position <= doc_end):
    out_of_span = True
  if out_of_span:
    start_position = 0
    end_position = 0
  else:
    doc_offset = len(query_tokens) + 2
    start_position = tok_start_position - doc_start + doc_offset
    end_position = tok_end_position - doc_start + doc_offset

if is_training and example.is_impossible:
  start_position = 0
  end_position = 0
```

4.3 SQuAD 2

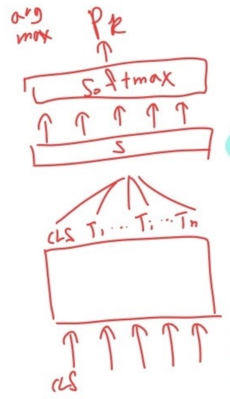
BERT: SQuAD 2.0

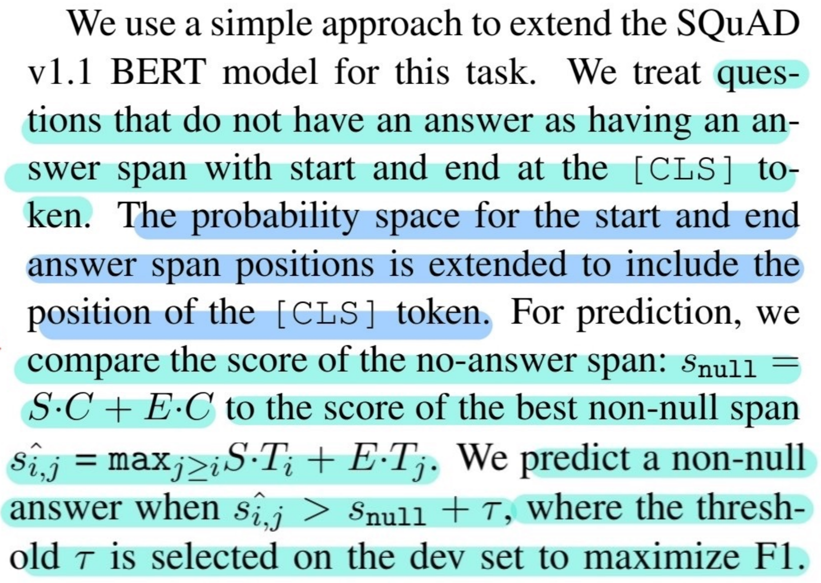
- Use token 0 ([CLS]) to emit logit for "no answer"
- "No answer" directly competes with answer span
- Threshold is optimized on dev set



What action did the US begin that started the second oil shock?
Ground Truth Answers: `<No Answer>`
Prediction: `<No Answer>`

The 1973 oil crisis began in October 1973 when the members of the Organization of Arab Petroleum Exporting Countries (OAPEC, consisting of the Arab members of OPEC plus Egypt and Syria) proclaimed an oil embargo. By the end of the embargo in March 1974, the price of oil had risen from US$3 per barrel to nearly $12 globally; US prices were significantly higher. The embargo caused an oil crisis, or "shock", with many short- and long-term effects on global politics and the global economy. It was later called the "first oil shock", followed by the 1979 oil crisis, termed the "second oil shock."





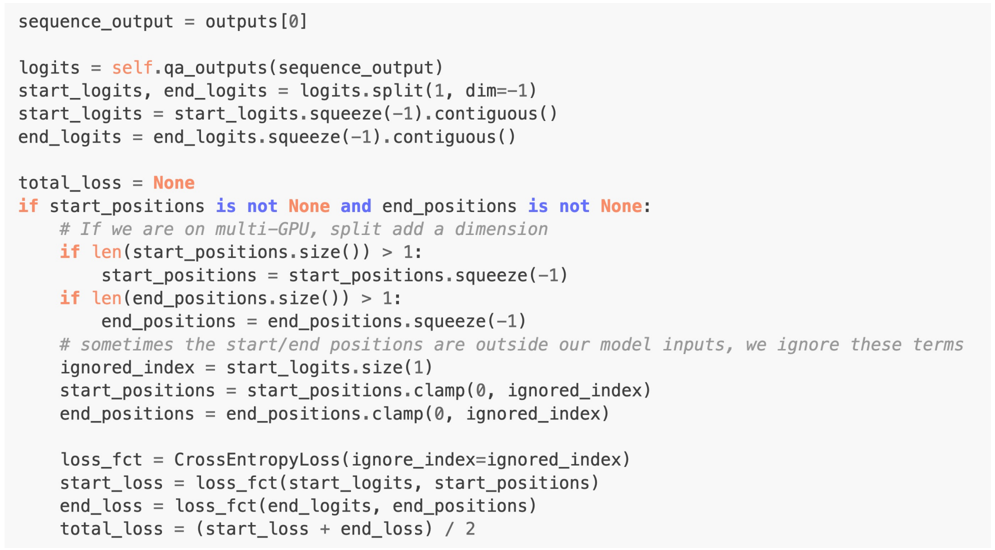
We use a simple approach to extend the SQuAD v1.1 BERT model for this task. We treat questions that do not have an answer as having an answer span with start and end at the `[CLS]` token. The probability space for the start and end answer span positions is extended to include the position of the `[CLS]` token. For prediction, we compare the score of the no-answer span: $s_{\mathtt{null}} = S{\cdot}C + E{\cdot}C$ to the score of the best non-null span $s_{\hat{i,j}} = \mathtt{max}_{j \geq i} S{\cdot}T_i + E{\cdot}T_j$. We predict a non-null answer when $s_{\hat{i,j}} > s_{\mathtt{null}} + \tau$, where the threshold τ is selected on the dev set to maximize F1.

- CLS까지 softmax에 넣는다.
- one hot vector으로 만들어서 값을 만들수 있는지 없는지까지도 확률화.
- 그 값이 가장 크면? 없는 것으로 학습.

- predictoin에서는 SC+EC값으로 비교.

Huggingface: QA

- huggingface에서는... max length을 넘으면 그냥 자름.
- 그리고 여기에서 열띈 토론이... 만약 QA에서 passage가 512을 넘으면 어케 해요????
    - https://github.com/huggingface/transformers/issues/1791
    - 근데 그냥 512에서 더 높은 수로 더 키우면 되는거 아닌가???



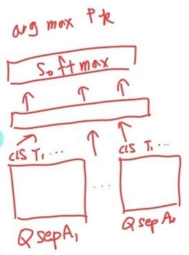
```python
sequence_output = outputs[0]

logits = self.qa_outputs(sequence_output)
start_logits, end_logits = logits.split(1, dim=-1)
start_logits = start_logits.squeeze(-1).contiguous()
end_logits = end_logits.squeeze(-1).contiguous()

total_loss = None
if start_positions is not None and end_positions is not None:
    # If we are on multi-GPU, split add a dimension
    if len(start_positions.size()) > 1:
        start_positions = start_positions.squeeze(-1)
    if len(end_positions.size()) > 1:
        end_positions = end_positions.squeeze(-1)
    # sometimes the start/end positions are outside our model inputs, we ignore these terms
    ignored_index = start_logits.size(1)
    start_positions = start_positions.clamp(0, ignored_index)
    end_positions = end_positions.clamp(0, ignored_index)

    loss_fct = CrossEntropyLoss(ignore_index=ignored_index)
    start_loss = loss_fct(start_logits, start_positions)
    end_loss = loss_fct(end_logits, end_positions)
    total_loss = (start_loss + end_loss) / 2
```

4.4 SWAG

- Run each Premise + Ending through BERT
- Produce logit for each pair on token 0 ([CLS])

```
A girl is going across a set of monkey bars.  She
(i)  jumps up across the monkey bars.
(ii) struggles onto the bars to grab her head.
(iii) gets to the end and stands on a wooden plank.
(iv) jumps up and does a back flip.
```

When fine-tuning on the SWAG dataset, we construct four input sequences, each containing the concatenation of the given sentence (sentence `A`) and a possible continuation (sentence `B`). The only task-specific parameters introduced is a vector whose dot product with the `[CLS]` token representation C denotes a score for each choice which is normalized with a softmax layer.



$$P_i = \frac{e^{V \cdot C_i}}{\sum_{j=1}^{4} e^{V \cdot C_j}}$$

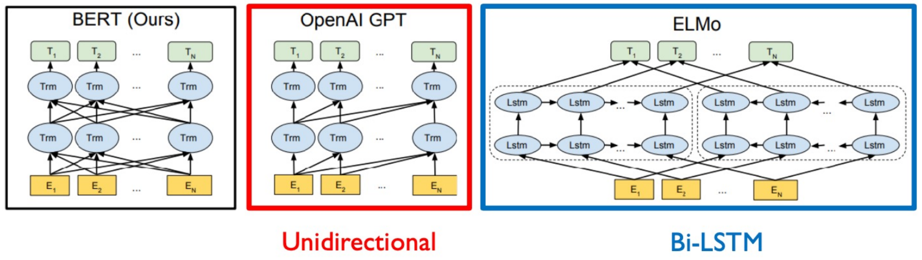
- question Sep Answer 1
- 이렇게 여러개 각각 encoding.

- 동일한 벡터에 넣어서 softmax.
- 확률 값이 가장 큰 것 선택해서 log liklihood loss으로 학습

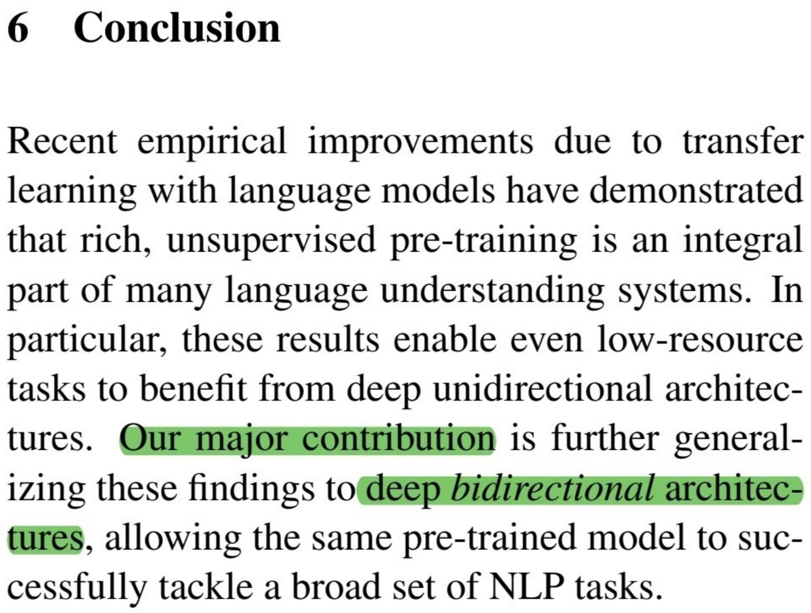
5. Conclusion

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- Learn through masked language modeling task
- Use large-scale data and large-scale model



BERT: Pre-training of deep bidirectional transformers for language understanding, NAACL'19



6 Conclusion

Recent empirical improvements due to transfer learning with language models have demonstrated that rich, unsupervised pre-training is an integral part of many language understanding systems. In particular, these results enable even low-resource tasks to benefit from deep unidirectional architectures. Our major contribution is further generalizing these findings to deep *bidirectional* architectures, allowing the same pre-trained model to successfully tackle a broad set of NLP tasks.