

# BERTology のススメ

BERT fine-tuning のすすめ

※ 本資料の著作権は、引用元の論文および記事に準じます

# Goal

- BERT について, **人よりも知ってる気分**になる
- BERT の fine-tuning について, **やればできる気分**になる

# Agenda

- はじめに
- BERT 解説
  - BERT とは
  - Transformer
  - BERT fine-tuning
- BERT fine-tuning の実践
  - テキスト分類 Classifier
  - テキスト検索 Ranking
  - タグ付け NER
- BERTology 論文

# 2018年10月: BERT の衝撃

- タスクに特化した構造を持たずに,人間のスコアを大きく超えた

## SQuAD1.1 Leaderboard

Since the release of SQuAD1.0, the community has made rapid progress, with the best models now rivaling human performance on the task. Here are the ExactMatch (EM) and F1 scores evaluated on the test set of SQuAD v1.1.

Rank	Model	EM	F1
	Human Performance <i>Stanford University</i> (Rajpurkar et al. '16)	82.304	91.221
1	BERT (ensemble) <i>Google AI Language</i> <a href="https://arxiv.org/abs/1810.04805">https://arxiv.org/abs/1810.04805</a>	87.433	93.160
2	BERT (single model) <i>Google AI Language</i> <a href="https://arxiv.org/abs/1810.04805">https://arxiv.org/abs/1810.04805</a>	85.083	91.835
2	nlnet (ensemble) <i>Microsoft Research Asia</i>	85.356	91.202

<https://rajpurkar.github.io/SQuAD-explorer/>

- 機械読解タスク(左)で,完全一致と部分一致の両指標で最高精度(2018/10/5)
- 様々な自然言語理解タスクでSOTA (QA,含意,言い換え,NER等)
- タスク適応は,出力層をタスク毎に1層のみ追加してfine-tuning

# GLUE benchmark Results

- 9つの自然言語理解タスクでSOTA (2018/10 時点)

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	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>91.1</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>81.9</b>

Table 1: GLUE Test results, scored by the GLUE evaluation server (<https://gluebenchmark.com/leaderboard>).

MNLI: テキスト同士の関連性(含意・矛盾・中立)を判定

QQP: 2つの質問が同じ意味かを判定

QNLI: 文章内に質問の回答が含まれているかを判定

SST-2: 映画のレビューを基に感情分析し, 良し悪しを判定

CoLA: 文章が文法的に正しいかを判定

STS-B: 2つのニュース見出しの意味的類似性を判定

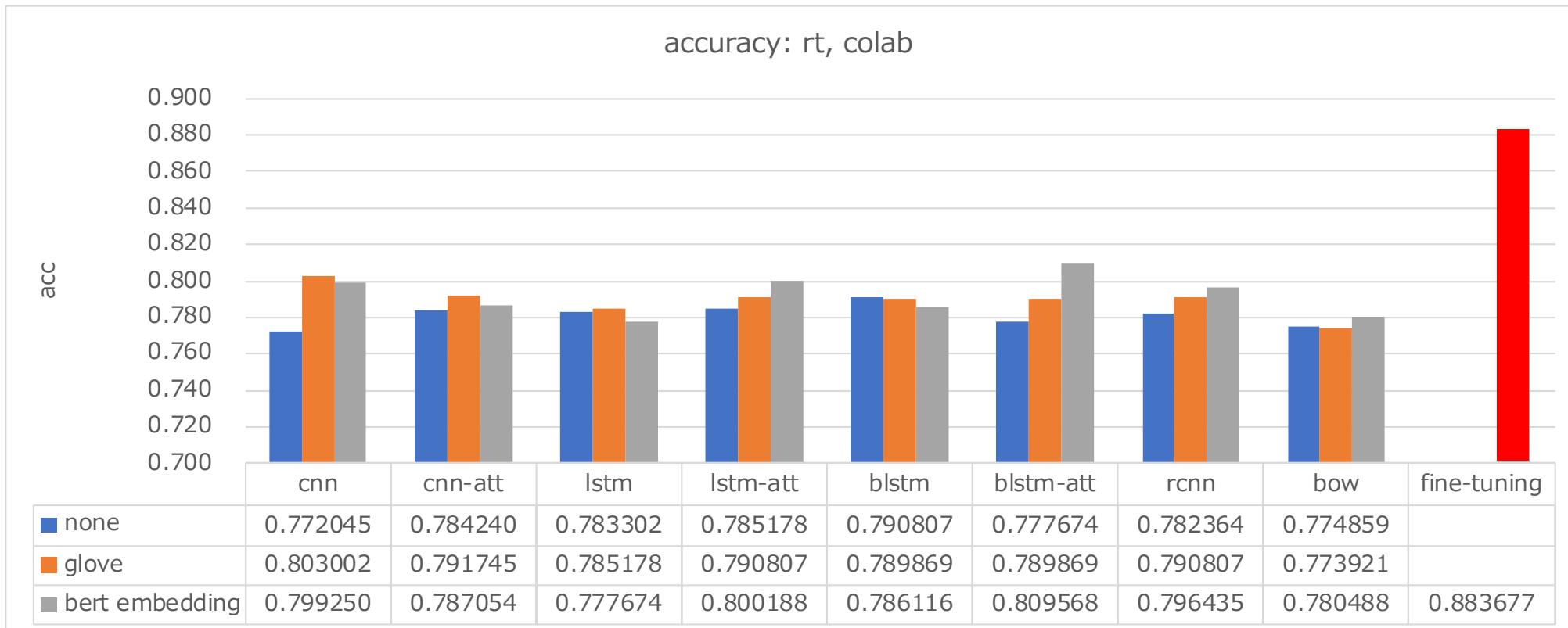
MRPC: 2つのニュース記事が意味的に等価かを判定

RTE: 2つの文章が含意関係かどうかを判定

# 手元のデータで簡易実験

<https://github.com/haradatm/nlp/tree/master/bert/classify>

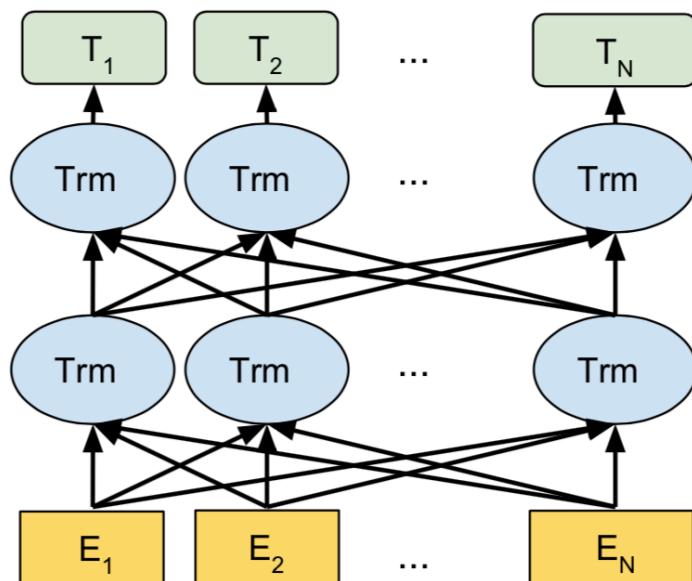
- Scale Movie Review Dataset (rt-polarity): pos/neg 2値



# BERTとは

- ・双方向 Transformer ブロックを24層重ねた言語モデル
- ・事前学習モデルが公開

BERT (Ours)



- 英語
  - 本家 Google の事前学習モデル \*1
  - Book Corpus 8億語 + 英語 Wikipedia 25億語 (語彙数 3万)
- 日本語
  - 黒橋研の事前学習モデル \*2
  - 日本語 Wikipedia 約1,800万文 (語彙数 3.2万)

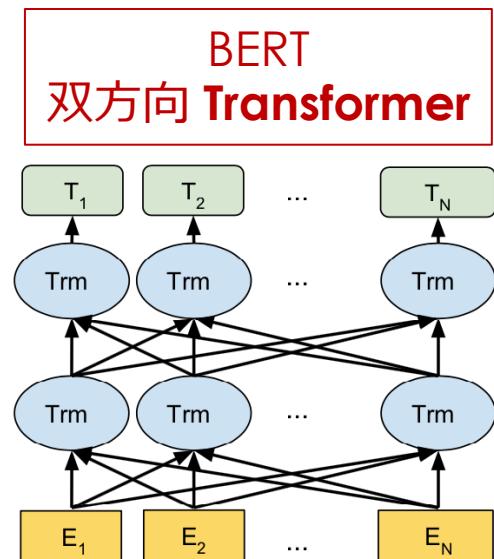
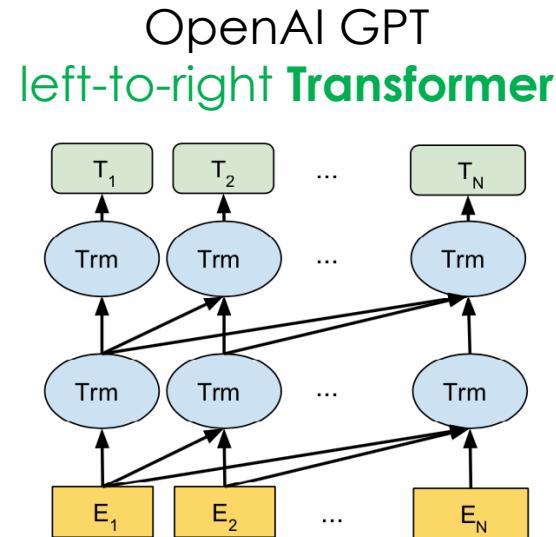
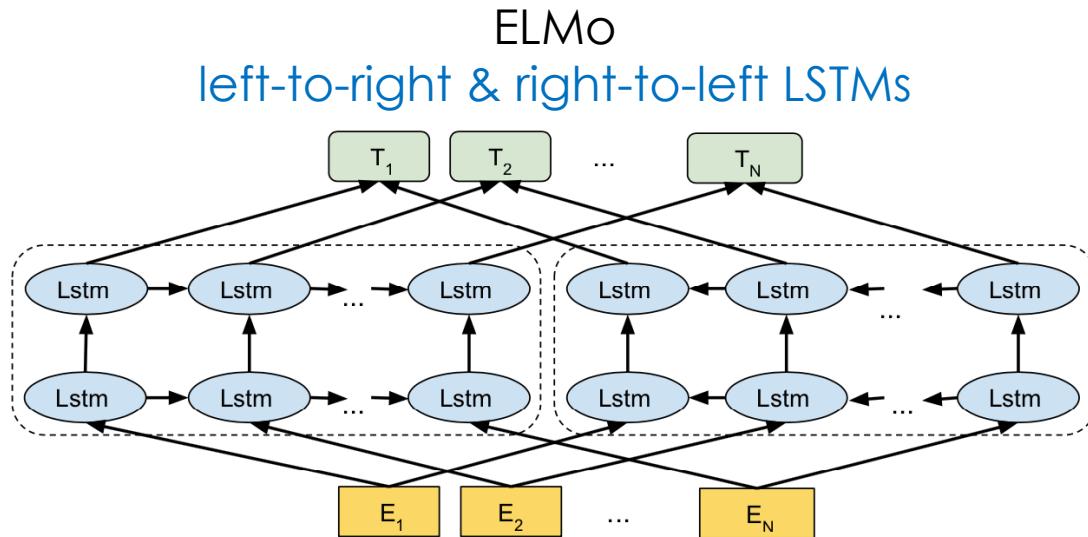
\*1 <https://github.com/google-research/bert>

\*2 <http://nlp.ist.i.kyoto-u.ac.jp/index.php?BERT日本語Pretrainedモデル>

# 言語モデルの Pre-train

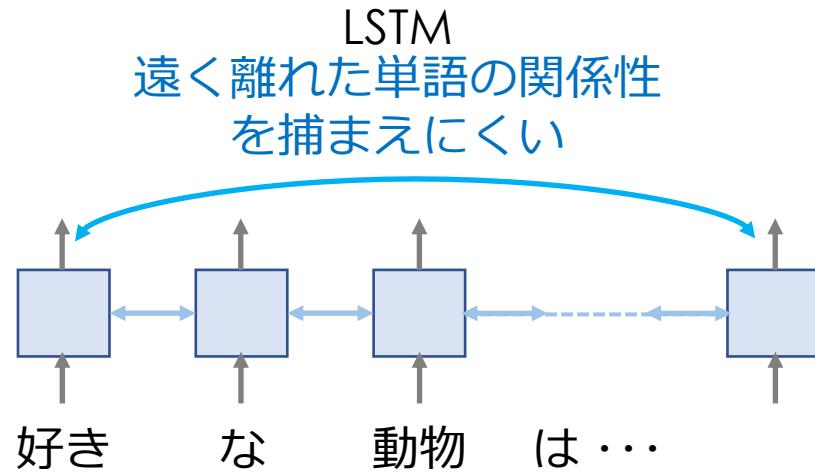
文献[1][2]

- ・自然言語処理の応用タスクは学習データが多くない(数十万件程度)ため、大規模コーパスから学習したモデルを転移することが流行
  - ELMo (2018/02に発表), OpenAI GPT (2018/06に発表)
- ・言語モデルは LTR あるいは RTL の次単語を予測するのが普通

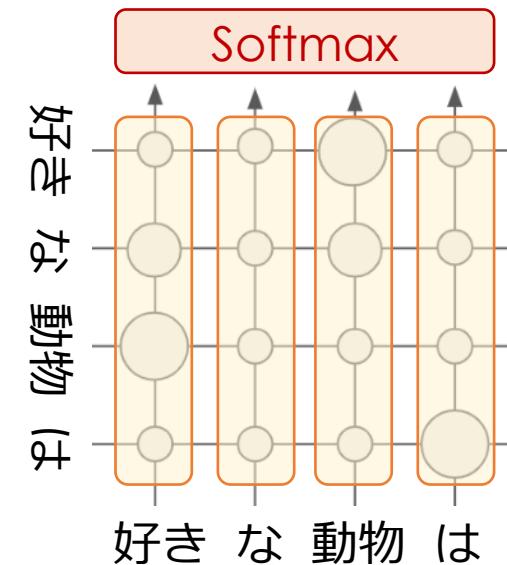


# Transformer

- RNNやCNNを使わずアテンションのみ使用したTransformerがニューラル機械翻訳で圧倒的な SOTA を達成
  - 従来, 単語系列の文脈理解は主にLSTM → 長期依存性の理解に限界
  - 離れた単語の関係性も直接考慮できる Self-Attention が性能向上に大きく寄与した (しかも省メモリで計算可)

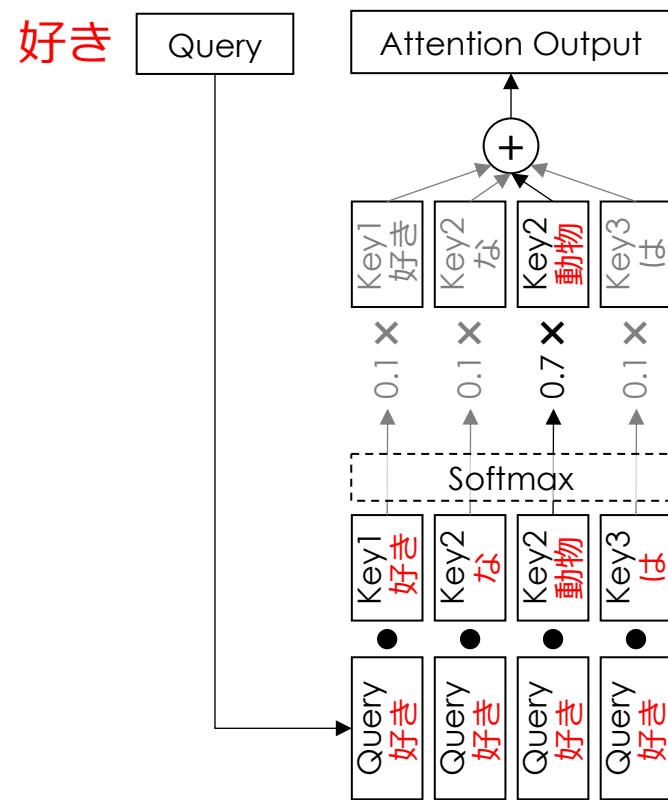


Self-Attention  
遠く離れた単語も  
直接関係性を考慮できる



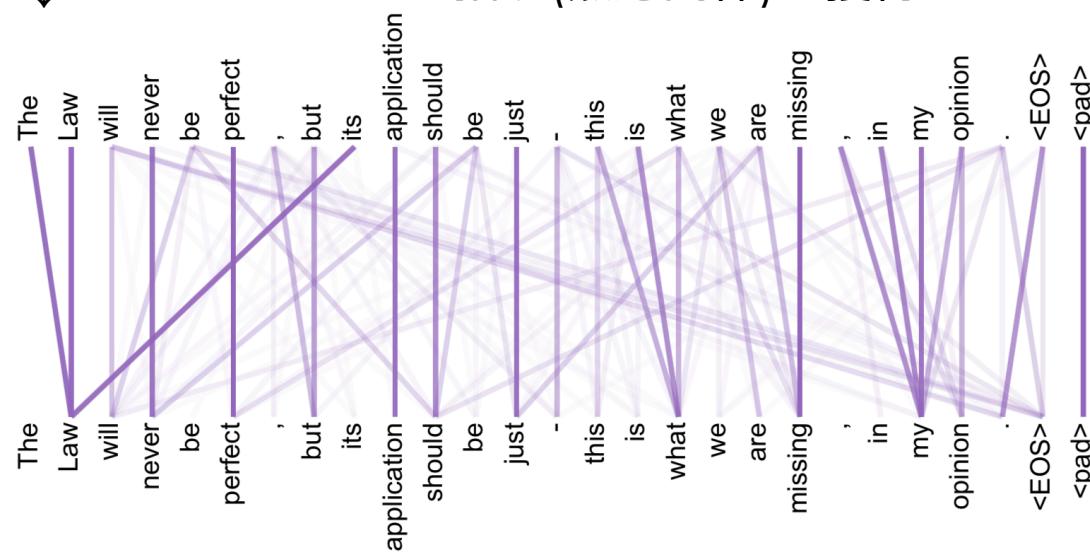
# Self-Attention

- Self-Attention は、文構造や照応関係を獲得するという見方



- ← アテンションの重みに従って value から情報を引き出す
- Self-Attentionでは Query, Key Value ともに同じコンテキストから生成されたベクトル

↓ “its” が “Law” を指す(照応関係)を獲得



# Transformer

- <http://nlp.seas.harvard.edu/2018/04/03/attention.html>

The screenshot shows the homepage of 'The Annotated Transformer'. At the top left is the HarvardNLP logo. To its right are links for 'Members', 'PI', 'Code', and 'Publications'. The main title 'The Annotated Transformer' is centered above a horizontal line. Below the line, the date 'Apr 3, 2018' is shown. A code block in a light gray box contains the following Python code:

```
from IPython.display import Image
Image(filename='images/aiayn.png')
```

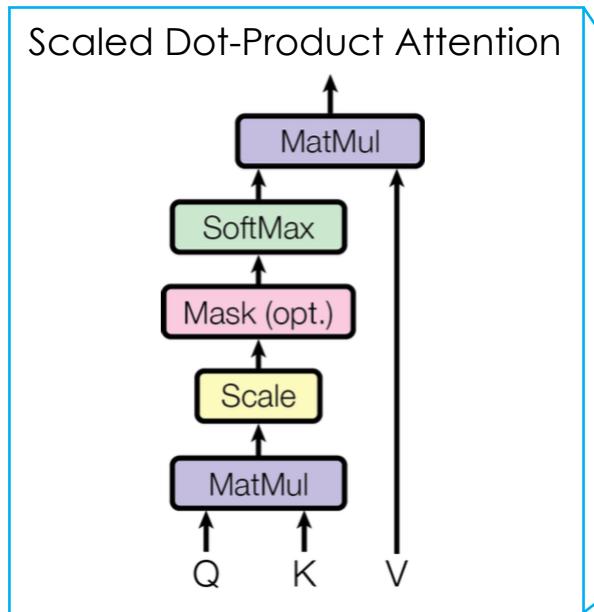
A horizontal line follows this. Below it, the section title 'Attention Is All You Need' is centered. At the bottom of the page, there is a row of author names and their affiliations and emails:

Ashish Vaswani*	Noam Shazeer*	Niki Parmar*	Jakob Uszkoreit*
Google Brain	Google Brain	Google Research	Google Research
avaswani@google.com	noam@google.com	nikip@google.com	usz@google.com
<hr/>			
Llion Jones*	Aidan N. Gomez* †	Lukasz Kaiser*	
Google Research	University of Toronto	Google Brain	
llion@google.com	aidan@cs.toronto.edu	lukasz.kaiser@google.com	
<hr/>			
Illia Polosukhin* ‡			
illia.polosukhin@gmail.com			

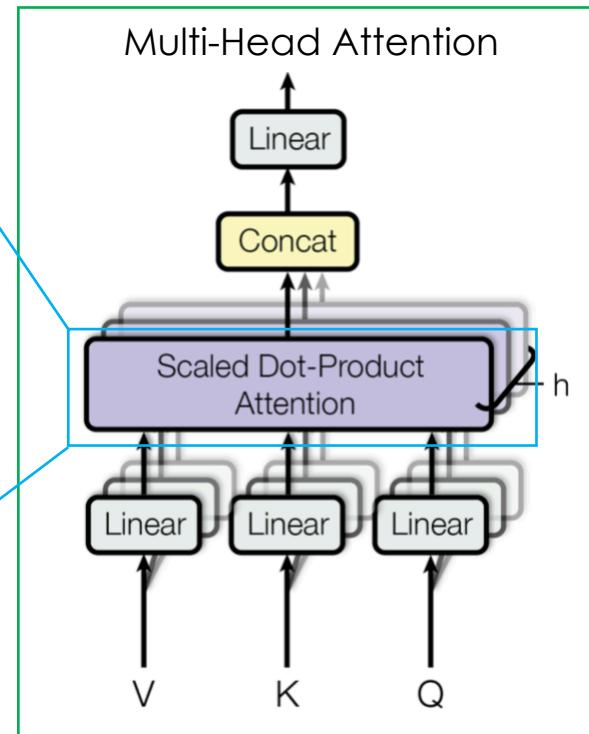
- 論文の注釈付きで、実装に沿って Transformer を解説したサイト
- Jupyter および Google Colab のノートブックも公開
- BERT や Transformer の初学者にオススメ

# Transformer

- 例: レイヤーN=6, ヘッドh=8, 長さ=512, 中間層=768



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

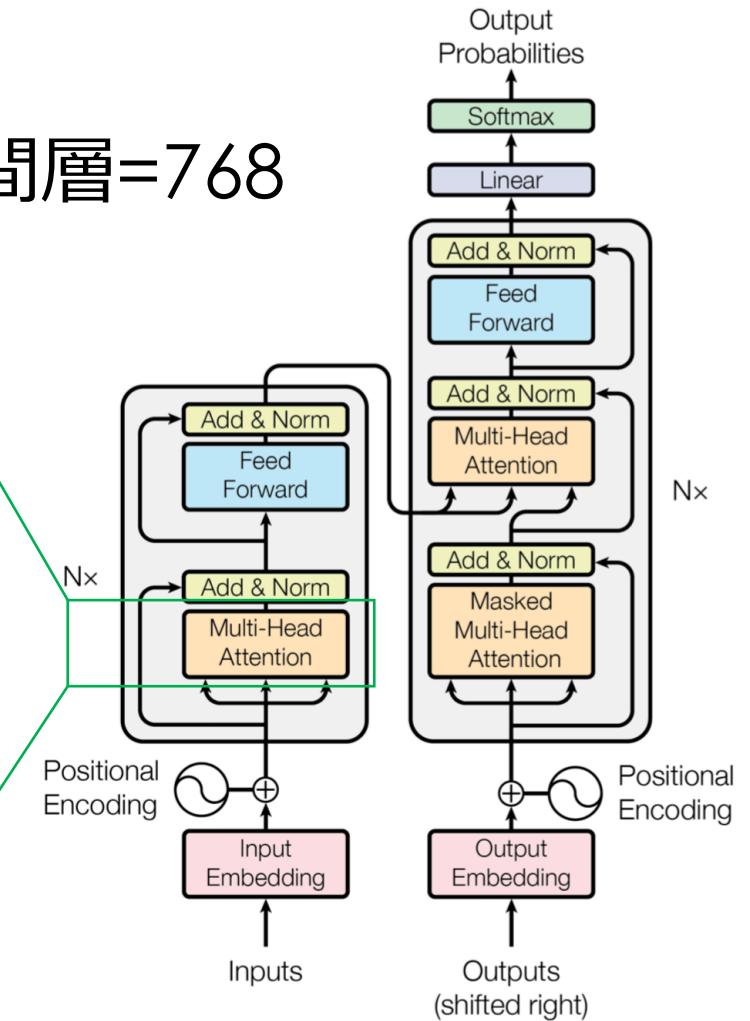
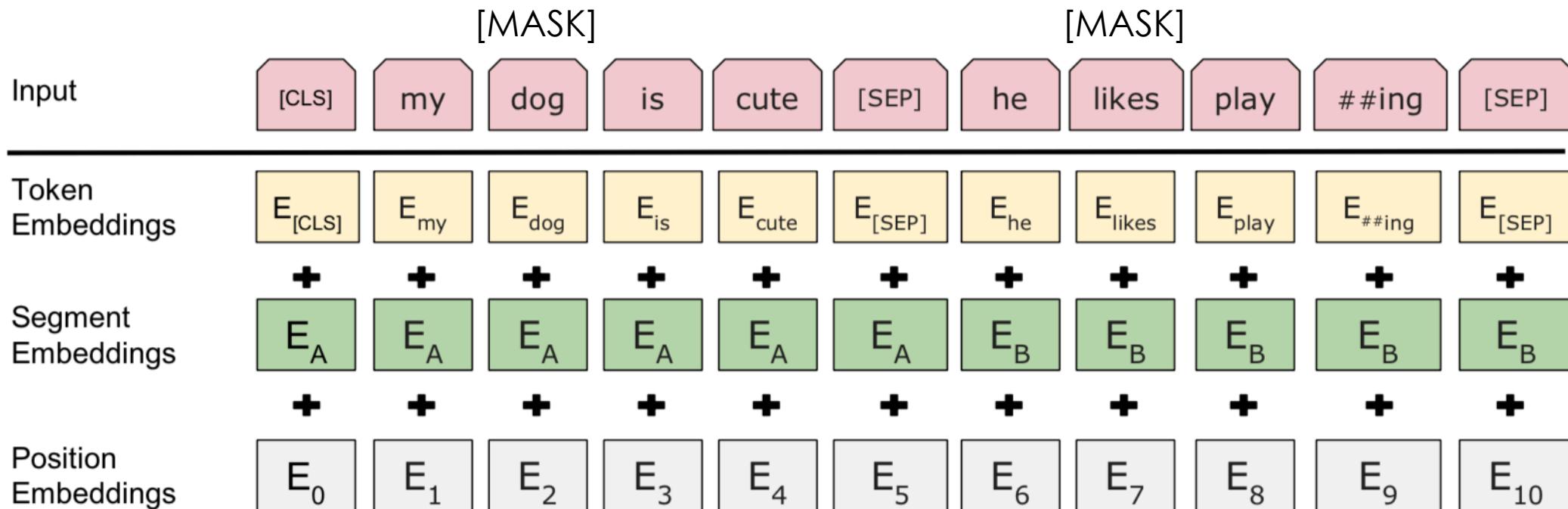


Figure 1: The Transformer - model architecture.

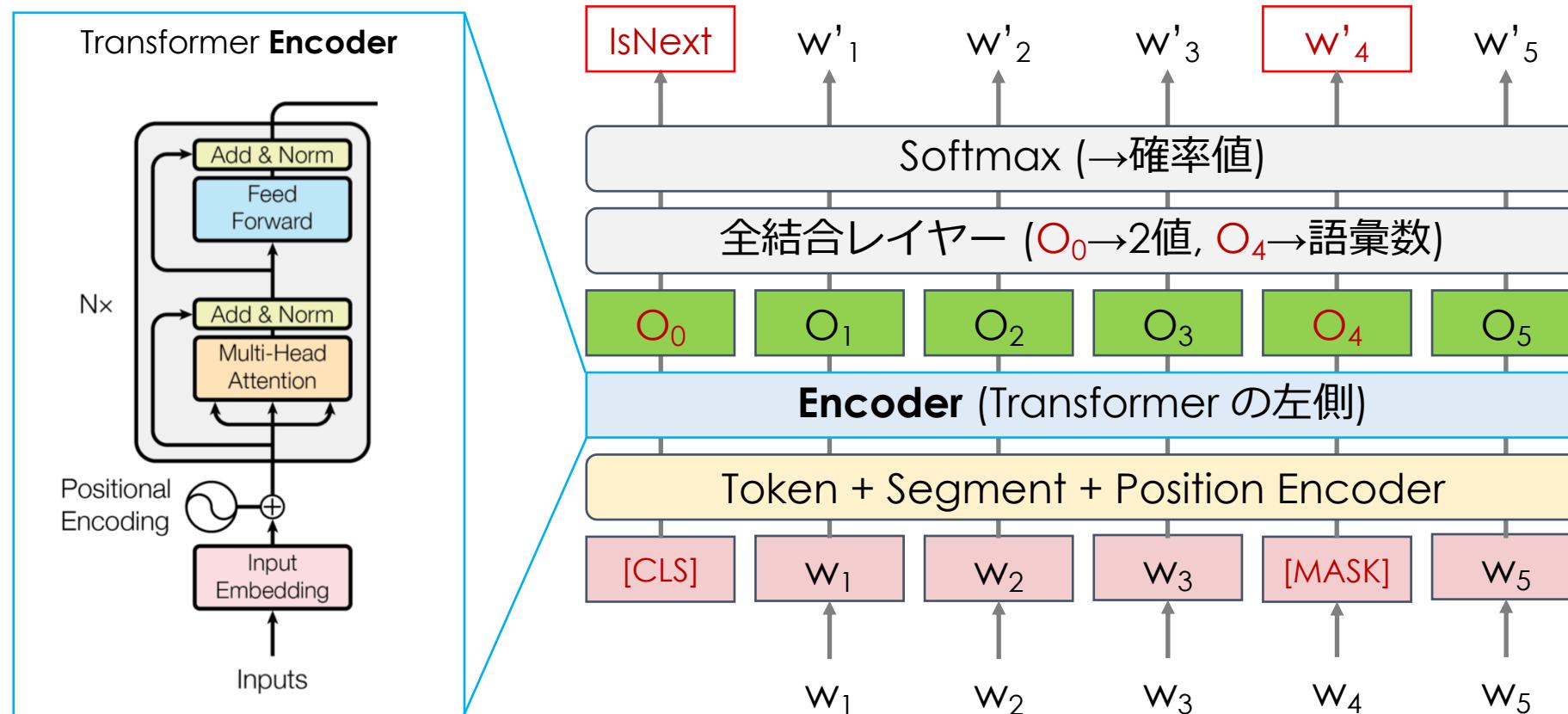
# BERT Task

- 隣接する文/ランダムな文をセパレータ([SEP])で繋げて、**隣接する文か否かの判定**(→CLS)を**穴埋め予測**(→[MASK])と同時に学習



# BERT Training

- Encoder 出力のすぐ上に分類レイヤーを追加→予測結果を学習



# BERT Pre-train モデル

文献[2][4]

モデル	英語 モデル		日本語 モデル
	Base	Large	
公開元	Google Research <sup>*1</sup>		京大 黒橋研 <sup>*2</sup>
コーパス	Book Corpus 8億ワード + 英語Wikipedia 25億 ワード		日本語Wikipedia 約1,800万文 (半角を全角に正規化)
語彙 (サブワード含む)	30K (WordPiece)		32K (Juman++ & BEP)
入力長	最大 512トークン <sup>*3</sup>		最大 128トークン <sup>*3</sup>
パラメタ	12層, 各層768次元	24層, 各層1024次元	12層, 各層768次元 (Base と同じ)
学習時間	4 Cloud TPUs で 4日間(≈100時間)	16 Cloud TPUs で 4日間(≈100時間)	1GPU (GTX 1080 Ti) で 約30日間(≈750時間)

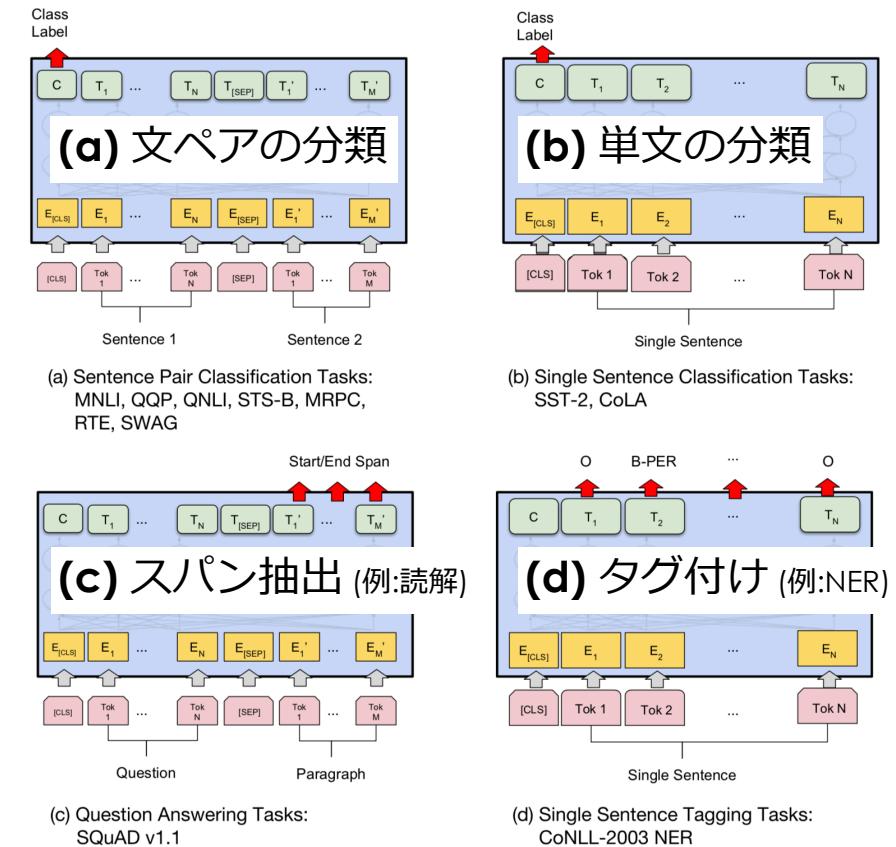
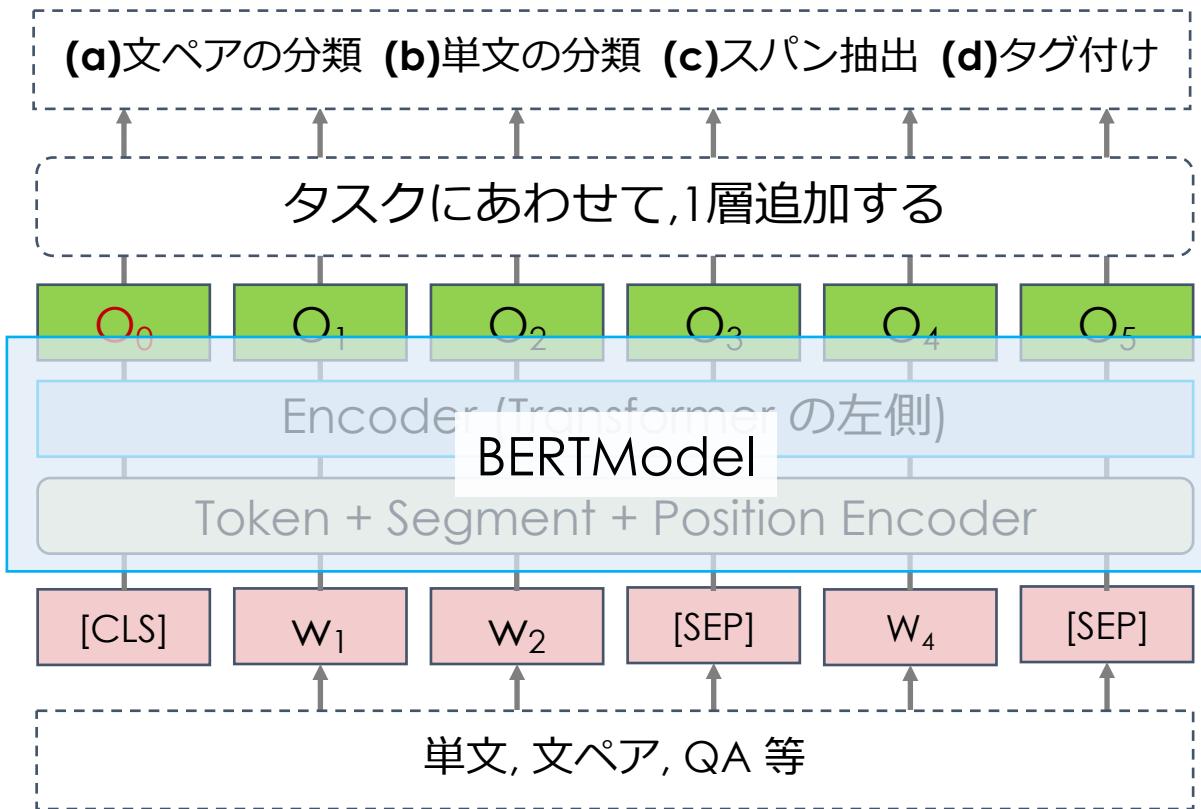
<sup>\*1</sup> <https://github.com/google-research/bert>

<sup>\*2</sup> <http://nlp.ist.i.kyoto-u.ac.jp/index.php?BERT日本語Pretrainedモデル>

<sup>\*3</sup> 入力できるシーケンスの長さに制限があることに注意

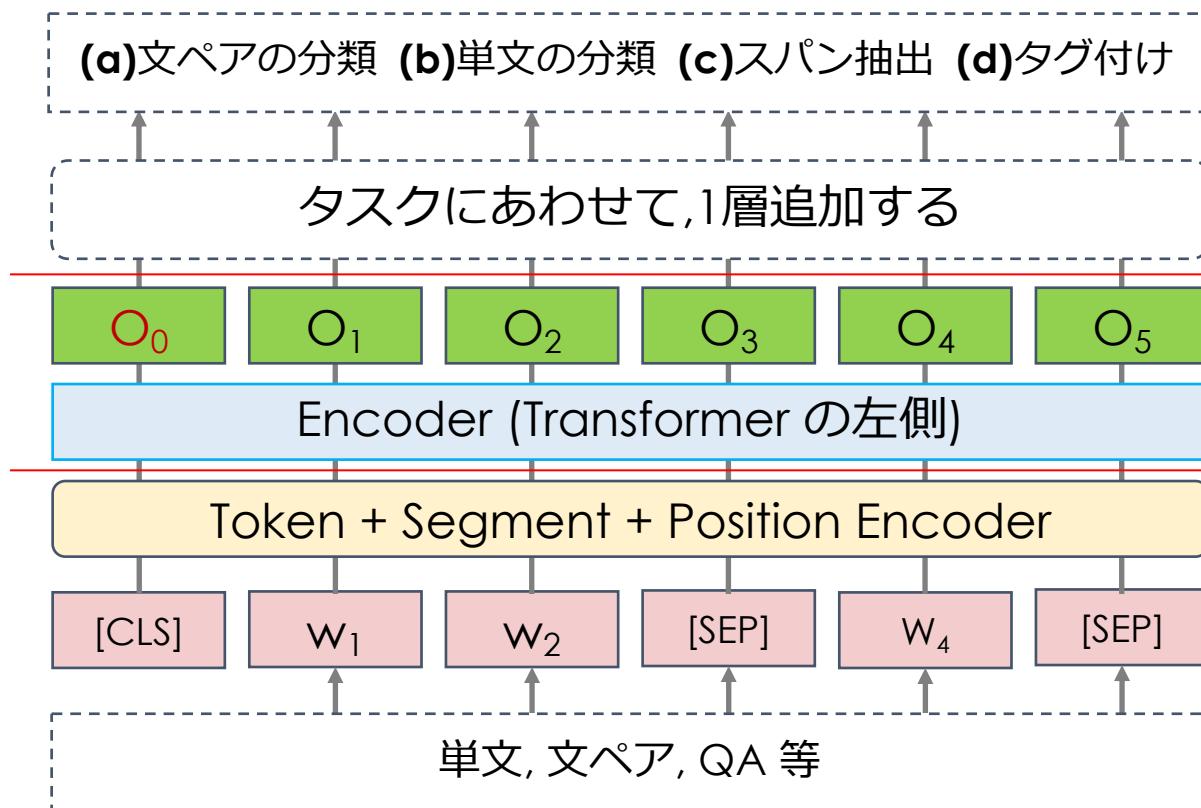
# BERT fine-tuning

- 出力層を1層追加してfine-tuningするのみで各タスクに適応



# BERT fine-tuning

- 出力層(ベクトル)は、BertModel のサービス関数で取り出せる



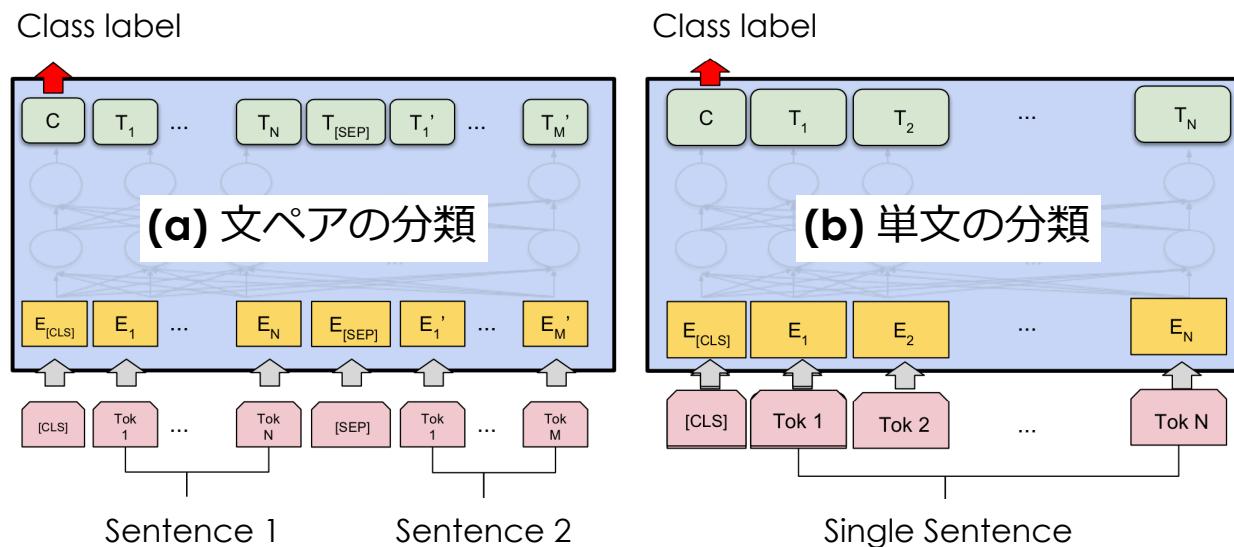
```
class BertModel(chainer.Chain):
    def __init__(self, config):
        :
    def __call__(self,
                :
                    def get_pooled_output(self, [CLS] のベクトルを取り出す
                        input_ids, input_mask=None, token_type_ids=None):
                            return self.__call__(input_ids, input_mask, token_type_ids)

                    def get_sequence_output(self, 各トークンのベクトルを取り出す
                        input_ids, input_mask=None, token_type_ids=None):
                            return self.__call__(input_ids, input_mask, token_type_ids,
                                                get_sequence_output=True)

                    def get_embedding_output(self, 学習済みの埋め込みを取り出す
                        input_ids, input_mask=None, token_type_ids=None):
                            return self.__call__(input_ids, input_mask, token_type_ids,
                                                get_embedding_output=True)
```

# BERT fine-tuning: テキスト分類 (a)(b)

- 文ペアまたは単文の分類: 全結合層を1層追加してfine-tuning
  - 入力: 先頭に [CLS], 文と文の境界には [SEP] トークンを追加



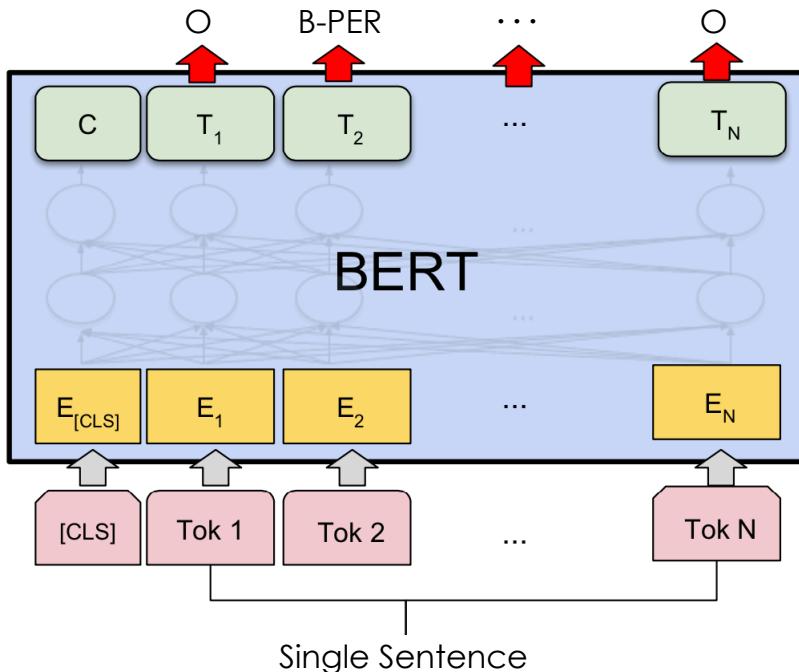
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, BRPC, RTE,  
SWAG

(b) Single Sentence Classification Task:  
MSST-2, CoLA

```
class BertClassifier(chainer.Chain):
    def __init__(self, bert, num_labels):
        :
    def __call__(self, input_ids, input_mask, token_type_ids, labels):
        output_layer = self.bert.get_pooled_output(input_ids, ...)
        output_layer = F.dropout(output_layer, 0.1)
        logits = self.output(output_layer)
        loss = F.softmax_cross_entropy(logits, labels)
    return loss
```

# BERT fine-tuning: タグ付け (d)

- NER: 全結合層を1層追加してfine-tuning
  - 入力: 先頭に [CLS], Qとパラグラフの境界には [SEP] トークンを追加

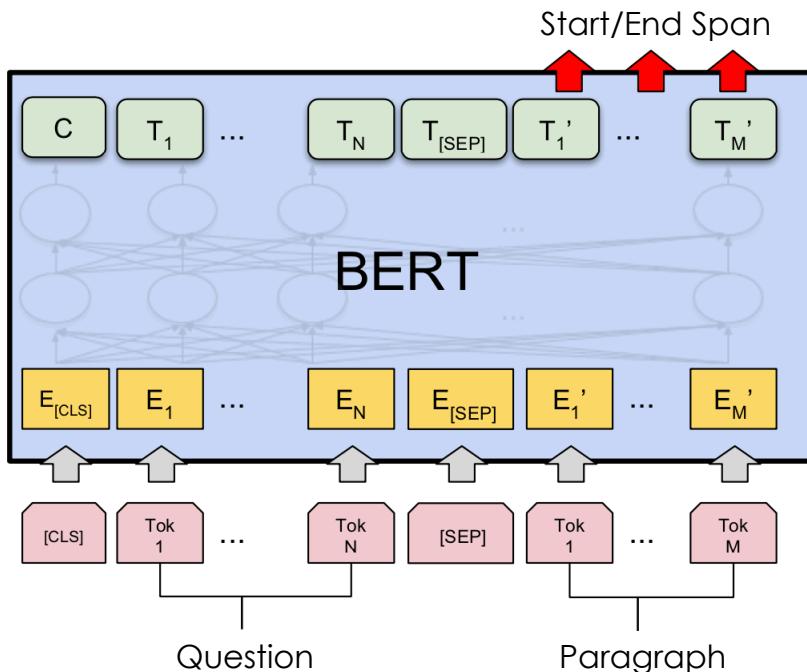


(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

```
class BertNER(chainer.Chain):
    def __init__(self, bert):
        :
    def __call__(self, input_ids, input_mask, token_type_ids, labels):
        final_hidden = self.bert.get_sequence_output(input_ids, ...)
        : (リサイズ等)
        logits = self.output(final_hidden_matrix)
        : (リサイズ等)
        loss = F.softmax_cross_entropy(logits, labels, ignore_label=0)
    return loss
```

# BERT fine-tuning: スパン抽出 (c)

- SQuAD: 全結合層を1層追加してfine-tuning
  - 入力: 先頭に [CLS], Qとパラグラフの境界には [SEP] トークンを追加



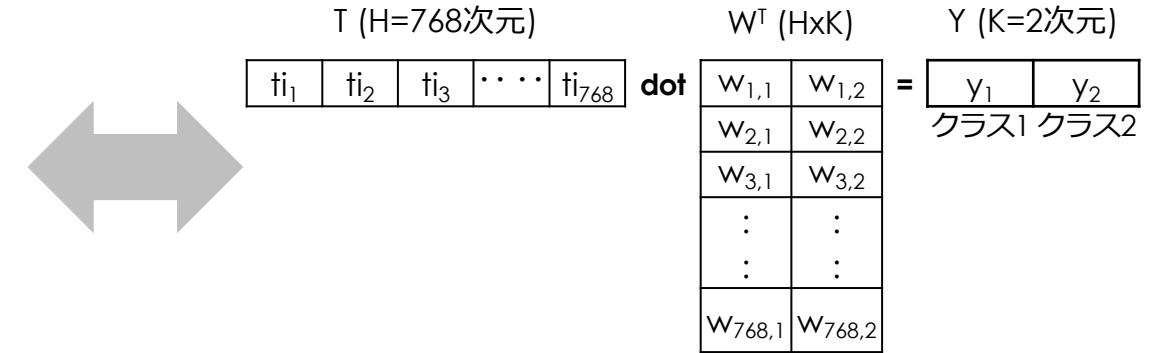
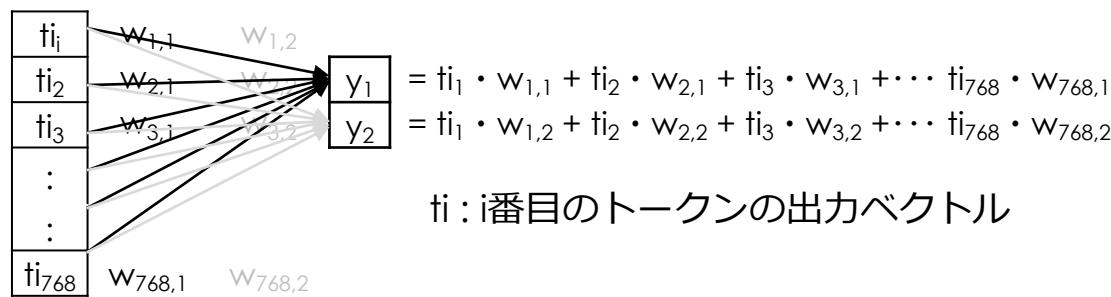
```

class BertSQuAD(chainer.Chain):
    def __init__(self, bert):
        :
    def __call__(self, input_ids, input_mask, token_type_ids):
        final_hidden = self.bert.get_sequence_output(input_ids, ...)
        : (リサイズ等)
        logits = self.output(final_hidden_matrix)
        : (リサイズ等)
        unstacked_logits = F.separate(logits, axis=0)
        (start_logits, end_logits) = (unstacked_logits[0], unstacked_logits[1])
        start_loss = F.softmax_cross_entropy(start_logits, start_positions)
        end_loss = F.softmax_cross_entropy(end_logits, end_positions)
        total_loss = (start_loss + end_loss) / 2.0
    return total_loss

```

# 解説 BERT fine-tuning: スパン抽出 (c)

- タグ付け(d):  $T \in \mathbb{R}^{K \times H}$ ,  $W \in \mathbb{R}^{K \times H}$ ,  $H=768$ ,  $K=2$  の場合

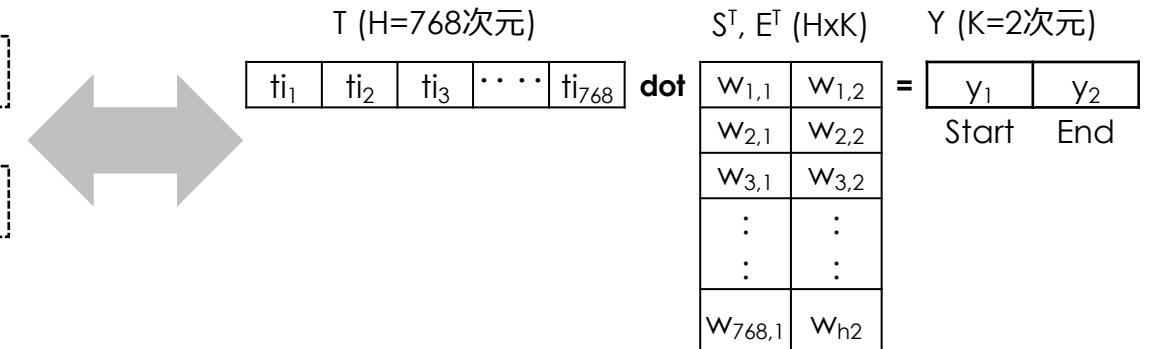


- スパン抽出(c):  $T \in \mathbb{R}^{K \times H}$ ,  $S_{\text{start}} \in \mathbb{R}^H$ ,  $E_{\text{end}} \in \mathbb{R}^H$ ,  $H=768$ ,  $K=2$  {Start,End}

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

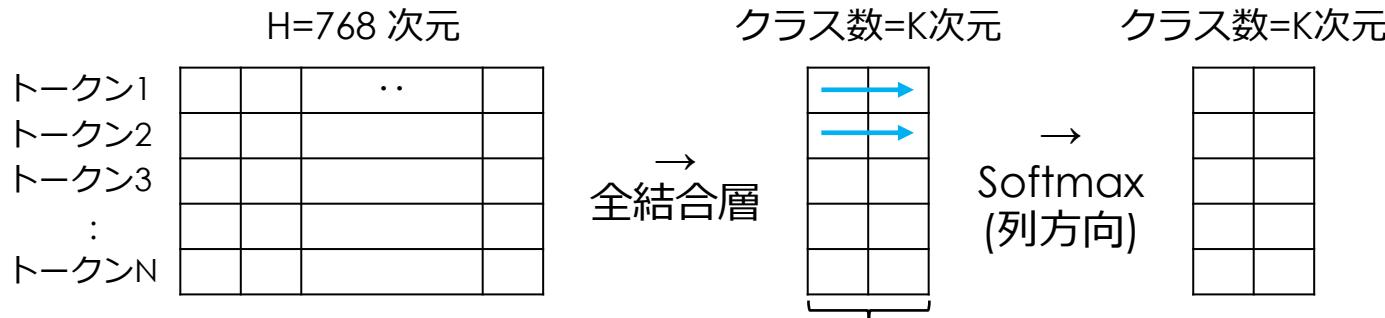
$softmax(S \cdot T_i) \Rightarrow [S \cdot T_i]$

同様に  
 $softmax(E \cdot T_i) \Rightarrow [E \cdot T_i]$

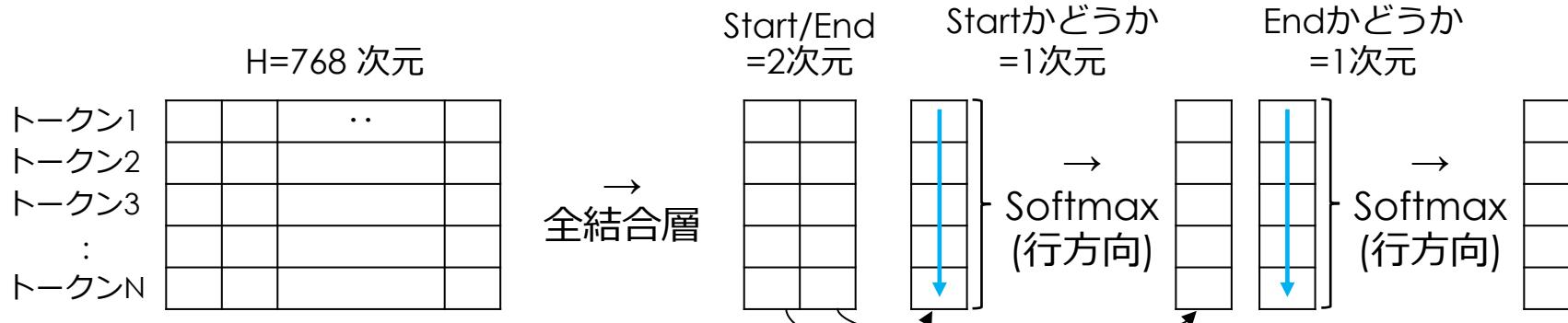


# 解説 BERT fine-tuning: スパン抽出 (c)

- タグ付け(d):  $T \in \mathbb{R}^{K \times H}$ ,  $W \in \mathbb{R}^{K \times H}$ ,  $H=768$ ,  $K=2$  の場合



- スパン抽出(c):  $T \in \mathbb{R}^{K \times H}$ ,  $S_{\text{start}} \in \mathbb{R}^H$ ,  $E_{\text{end}} \in \mathbb{R}^H$ ,  $H=768$ ,  $K=2$  {Start,End}



# BERT まとめ

- 2018年に Google AI が発表し,話題を席巻した **汎用言語モデル**  
→ **双方向 Transformer** ブロックを24層重ねたモデル
- 言語モデルを大規模コーパスで事前学習し,**出力層をタスクに応じた1層のみ追加**して fine-tuning
- fine-tuning モデルは,**タスクに特化した構造を持たずに,様々なタスクで汎用的かつ良好な性能**
- 発表以来,派生手法や応用タスクへの適用等の研究(**BERTology**)が盛んになり,引用数は1,276 に上る ※ 2019/09/08 時点

# BERT fine-tuning の実践

以降では chainer を想定しています

# fine-tuning 実装例と実験

- <https://github.com/haradatm/nlp/tree/master/bert>

haradatm / nlp

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

Branch: master nlp / bert / Create new file Upload files Find file History

Tomohiko HARADA and Tomohiko HARADA some updated Latest commit 4a617b3 12 days ago

..

classify → テキスト分類: rt-polarity(英), 国交省不具合DB(日)での実験 12 days ago

ner → テキスト検索: MedNLP 患者表現辞書(日)での実験 15 days ago

ranking → タグ付け: CoNLL-2013 (英)での追実験 15 days ago

# fine-tuning 実装例と実験

README.md

## BERT-Classification (Chainer example Classification using BERT fine-tuning)

### Description

This example code is a text entclassification using BERT fine-tuning.

### Dependencies

- python 3.7
- chainer 5.4

In addition, please add the project folder to PYTHONPATH and `conca install` the following packages:

- `matplotlib`

### Usage

### Preparation

#### BERT Pretrained model

- Download [Pretrained model \(English\)](#) and extract them in "BERT".
- Download [Pretrained model \(Japanese\)](#) and extract them in "BERT".

#### Data

- [Scale Movie Review Dataset \(rt-polarity\)](#): Predict its sentiment (positive, negative).
- [Road Transport Bureau of MLIT](#)
- Create train and test datasets and put them in the appropriate place.

```
wc -l datasets/rt-polarity/04-{train,test}.txt
9596 datasets/rt-polarity/04-train.txt
1066 datasets/rt-polarity/04-test.txt
10662 total
```

README.md

## BERT-Ranking (Chainer example Ad-Hoc Text Retrieval using BERT)

### Description

This example code is to convert a patient disease expression to a vector.

### Dependencies

- python 3.6
- chainer 5.4

In addition, please add the project folder to PYTHONPATH and `conca install` the following packages:

- `matplotlib`
- `pytrec_eval`

### Preparation

#### BERT Pretrained model

- Download [BERT Japanese Pretrained model](#) and extract them in "BERT".

#### Data

- Download [Patient Disease Expression](#), convert them to tsv format.

```
cd datasets/patient
cat d3_20190326.txt \
| cut -f 1-2 \
| sort -u \
| python ../tools/sampling.py /dev/stdin 8483 \
| grep -v "^$" > train-all.txt

cat train-all.txt | head -n 1000 > test.txt
cat train-all.txt | tail -n +1001 > train.txt
```

README.md

## BERT-NER (Chainer example code for Neural Architectures for Named Entity Recognition using BERT fine-tuning)

### Description

This example code is a named entity recognition using BERT fine-tuning.

- "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" by Jacob Devlin, et al.,

### Dependencies

- python 3.7
- chainer 5.4

In addition, please add the project folder to PYTHONPATH and `conca install` the following packages:

- `matplotlib`
- `seqeval`

### Usage

### Preparation

#### BERT Pretrained model

- Download [Pretrained model](#) and extract them in "BERT".

#### Data

- Download [CoNLL-2003 Datasets](#) and put them in the appropriate place.
- Convert from BIO to IOBES format

```
mkdir -p datasets && cd datasets
wget https://github.com/synalp/NER/raw/master/corpus/CoNLL-2003/eng.train
```

# bert-chainer <https://github.com/soskek/bert-chainer>

- BERT の chainer 実装が公開されている

README.md

## Chainer implementation of Google AI's BERT model with a script to load Google's pre-trained models

This repository contains a Chainer reimplementation of [Google's TensorFlow repository for the BERT model](#) for the paper [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#) by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova.

This implementation can load any pre-trained TensorFlow checkpoint for BERT (in particular [Google's pre-trained models](#)) and a conversion script is provided (see [below](#)).

# bert-chainer <https://github.com/soskek/bert-chainer>

- Pre-train モデルの TensorFlow→chainer (.npz) 変換が必要

To run this specific conversion script you will need to have TensorFlow and Chainer installed (`pip install tensorflow`).  
The rest of the repository only requires Chainer.

Here is an example of the conversion process for a pre-trained `BERT-Base Uncased` model:

```
export BERT_BASE_DIR=/path/to/bert/uncased_L-12_H-768_A-12  
  
python convert_tf_checkpoint_to_chainer.py \  
--tf_checkpoint_path $BERT_BASE_DIR/bert_model.ckpt \  
--npz_dump_path $BERT_BASE_DIR/arrays_bert_model.ckpt.npz
```

You can download Google's pre-trained models for the conversion [here](#).

# Python コードの配置

- BERT 実装の中から,fine-tuningに必要な 3ファイルを集める

左: <https://github.com/soskek/bert-chainer>

Branch: master New pull request

soskek Merge pull request #5 from soskek/fix-at

- bert
- to\_be\_modified
- README.md
- \_init\_.py
- convert\_tf\_checkpoint\_to\_chainer.py
- extract\_features.py
- modeling.py
- optimization.py
- run\_classifier.py
- run\_squad.py
- sample\_text.txt
- tokenization.py
- tokenization\_test.py

右: <https://github.com/haradatm/nlp/tree/master/bert/classify>

Branch: master nlp / bert / classify /

Tomohiko HARADA and Tomohiko HARADA

BERT ← Pre-train モデルはここ

- bertlib
- datasets
- results
- README.md
- test\_bert.py
- train\_bert.py

Branch: master nlp / bert / classify / bertlib /

Tomohiko HARADA and Tomohiko HARADA some up

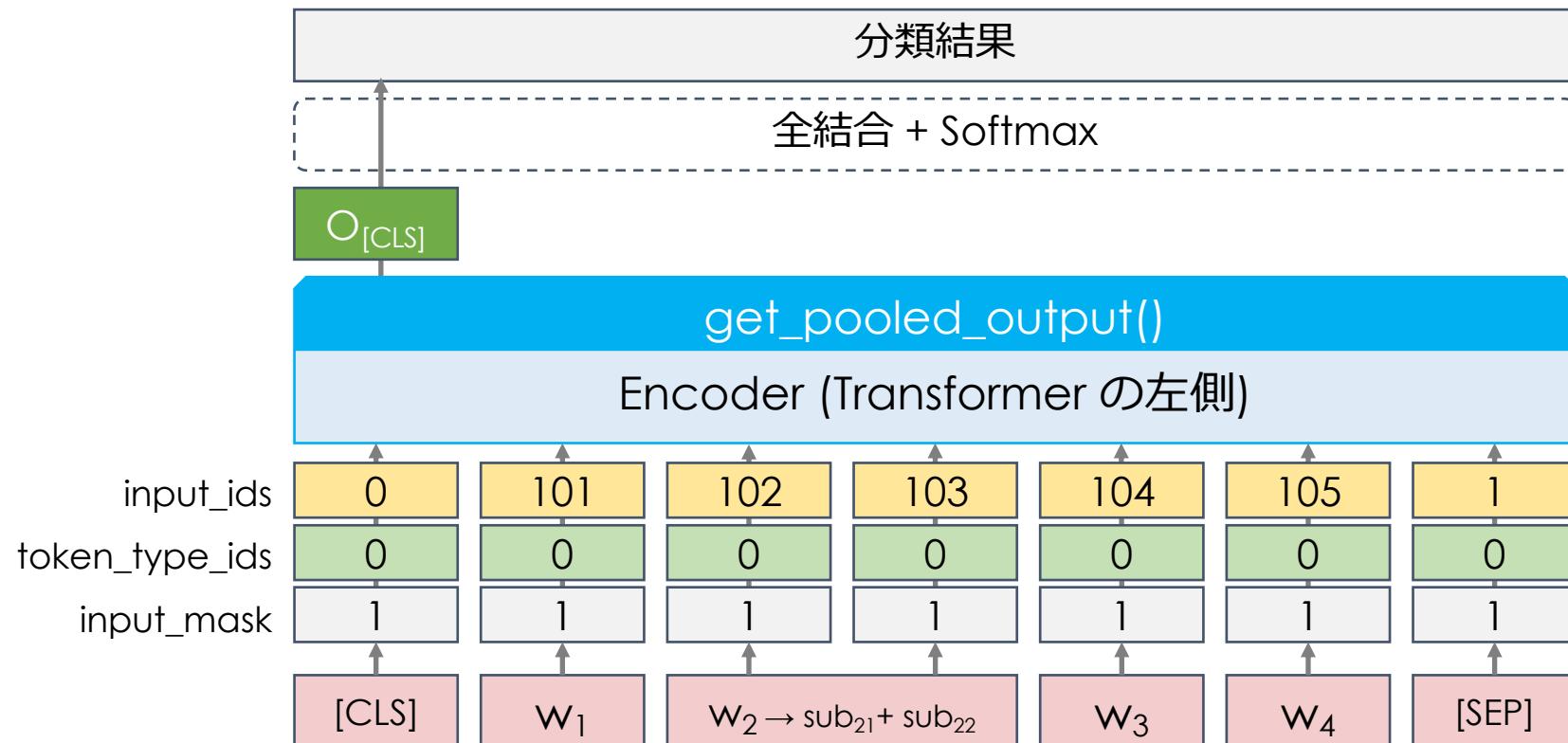
- modeling.py
- optimization.py
- tokenization.py

# テキスト分類: 実装

タイプ(b): 単文の分類  
→ [CLS] のベクトルを取り出す  
→ **get\_pooled\_output()**

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/classify>

- **train\_bert.py** : input\_ids, token\_type\_ids, input\_mask は以下の通り



- input\_ids
    1. FullTokenizer.tokenize() で入力文をトークン化
    2. FullTokenizer.convert\_tokens\_to\_ids() で ID に変換してセット
  - token\_type\_ids  
すべて同じ=0 をセット
  - input\_mask  
すべて同じ=1 をセット
- ※ FullTokenizer は、tokenization.py が提供

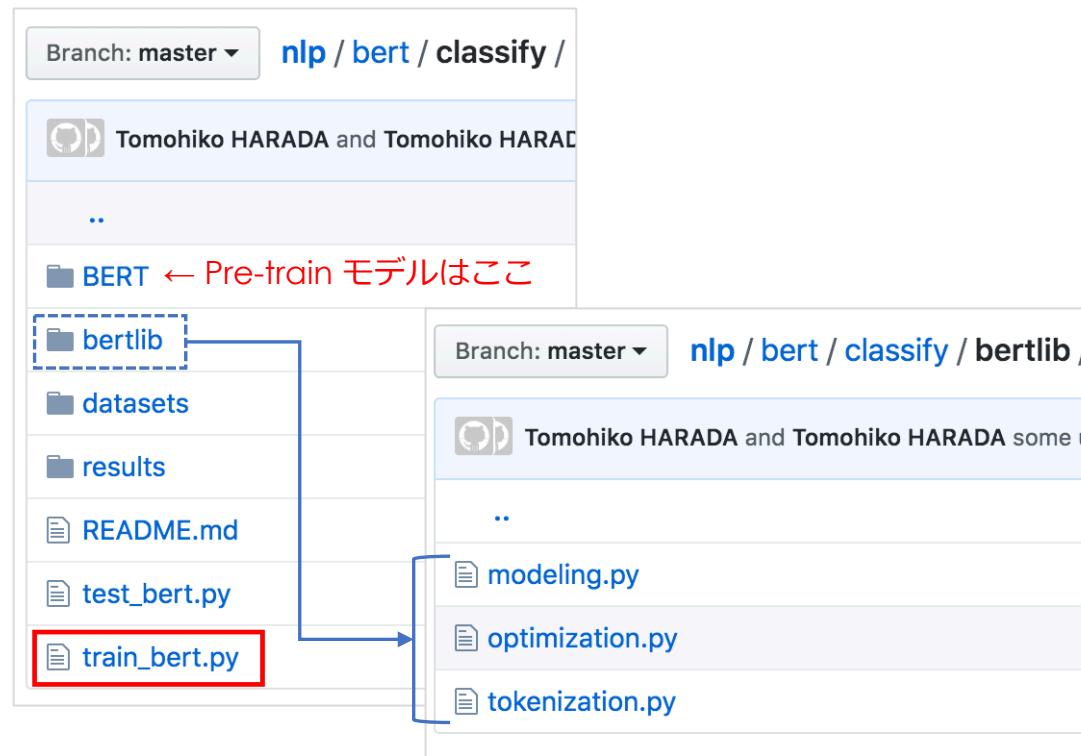
# テキスト分類: 実装

タイプ(b): 単文の分類  
→ [CLS] のベクトルを取り出す  
→ **get\_pooled\_output()**

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/classify>

- **train\_bert.py** : BERT 実装の run\_classifier.py とほぼ同じ

<https://github.com/haradatm/nlp/tree/master/bert/classify>



```
class BertClassifier(chainer.Chain):  
    def __init__(self, bert, num_labels):  
        :  
    def __call__(self, input_ids, input_mask, token_type_ids, labels):  
        output_layer = self.bert.get_pooled_output(input_ids, ...)  
        output_layer = F.dropout(output_layer, 0.1)  
        logits = self.output(output_layer)  
        loss = F.softmax_cross_entropy(logits, labels)  
        return loss
```

```
def main():  
    :  
    bert_config = BertConfig.from_json_file(bert_config_file)  
    tokenizer = FullTokenizer(vocab_file=vocab_file, do_lower_case=True)  
    :  
    bert = BertModel(config=bert_config)  
    model = BertClassifier(bert, num_labels=len(labels))  
    chainer.serializers.load_npz(  
        init_checkpoint, model, ignore_names=['output/W', 'output/b'])
```

# テキスト分類: 実験データ

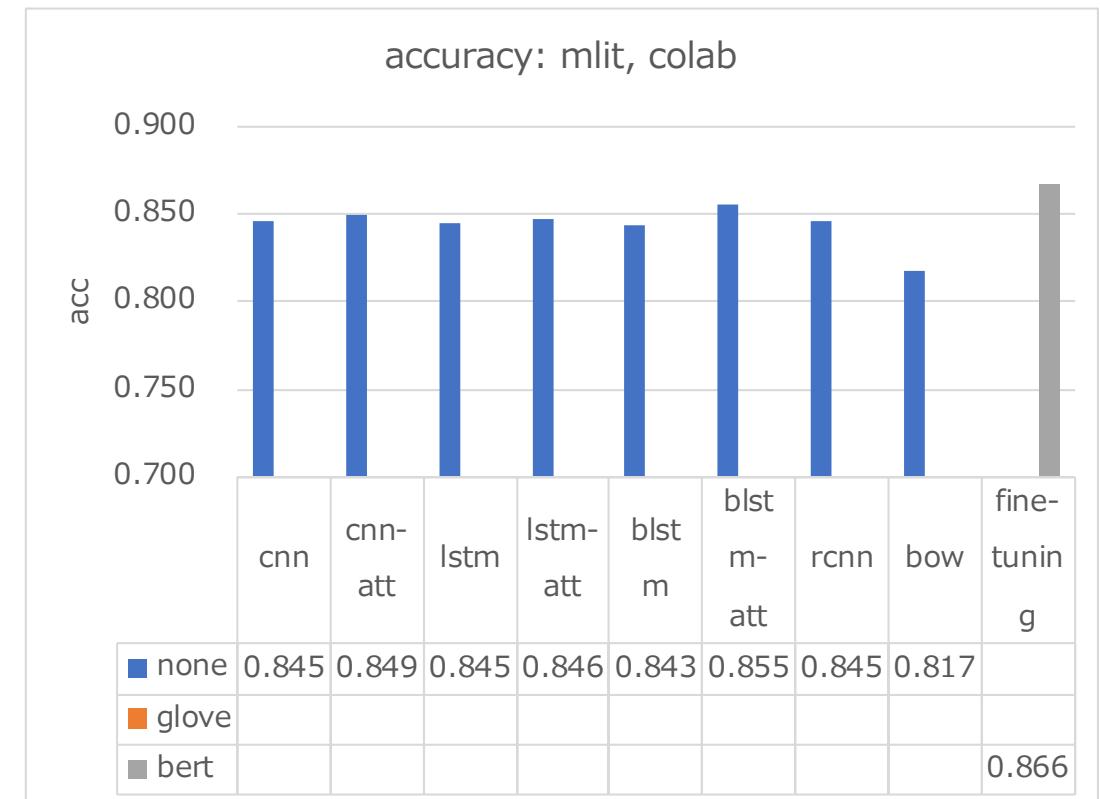
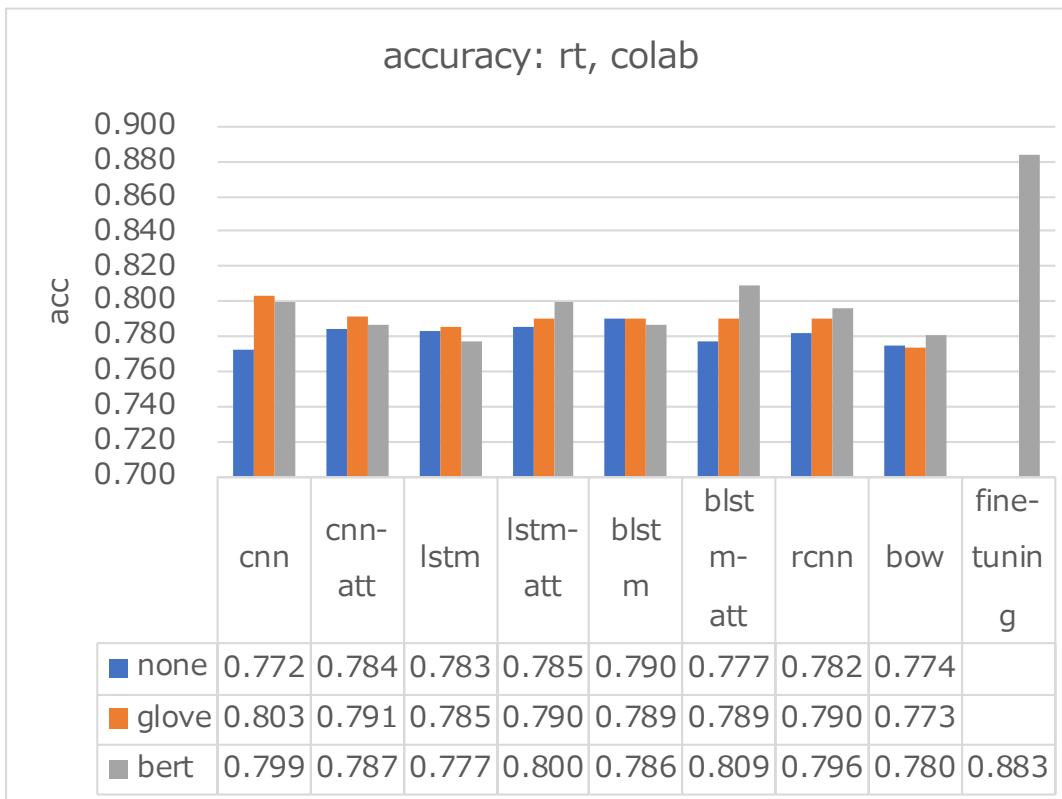
詳細→ <https://github.com/haradatm/nlp/tree/master/bert/classify>

- Datasets
  - 英語: [Scale Movie Review Dataset \(rt-polarity\)](#) → Pos/Neg **2クラス**
    - <https://www.cs.cornell.edu/people/pabo/movie-review-data/>
    - 全 51,940 件 → 学習 46,743, 評価 5,197 に分割
  - 日本語: 国交省「自動車の不具合情報」(mlit) → 不具合分類 **16クラス**
    - <http://carinf.mlit.go.jp/jidosha/carinf/opn/index.html>
    - クロール&スクレイピングした全10,662 件 → 学習 9,596, 評価 1,066 に分割
    - テキストは,あらかじめ Juman++ で分かち書き
- BERT モデル
  - 英語: Google AI, [uncased\\_L-12\\_H-768\\_A-12](#)
  - 日本語: 黒橋研, [Japanese\\_L-12\\_H-768\\_A-12\\_E-30\\_BPE](#)

# テキスト分類: 実験結果

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/classify>

- 英(rt-polarity)日(mlit)の両方で精度向上、特に英語が効果大

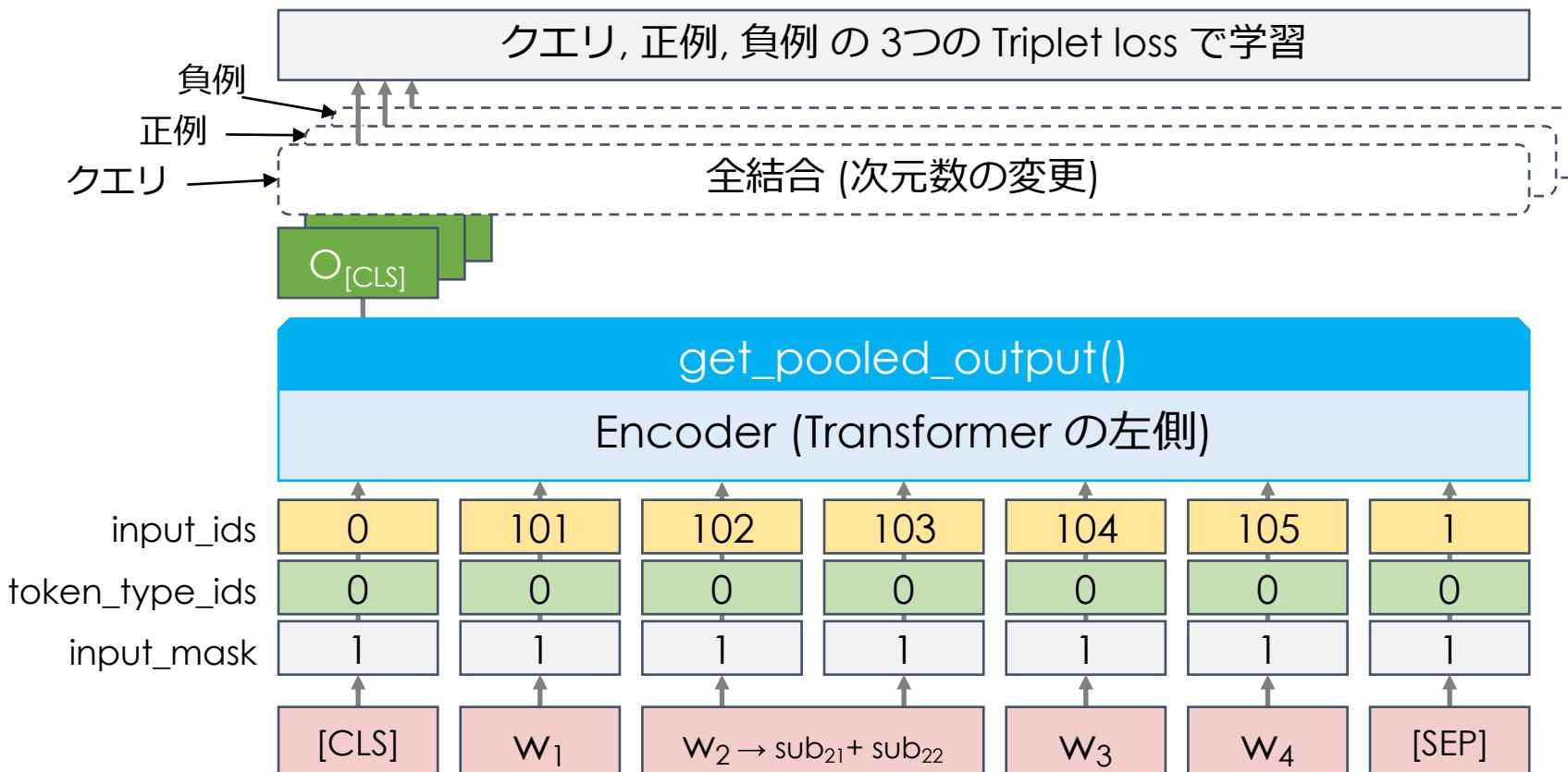


# テキスト検索: 実装

タイプ(b): 単文の分類  
→ [CLS] のベクトルを取り出す  
→ **get\_pooled\_output()**

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ranking>

- **train\_patient-triplet.py** : input\_ids, token\_type\_ids, input\_mask は以下の通り



- input\_ids
    1. FullTokenizer.tokenize() で入力文をトークン化
    2. FullTokenizer.convert\_tokens\_to\_ids() で ID に変換してセット
  - token\_type\_ids  
すべて同じ=0 をセット
  - input\_mask  
すべて同じ=1 をセット
- ※ FullTokenizer は、tokenization.py が提供

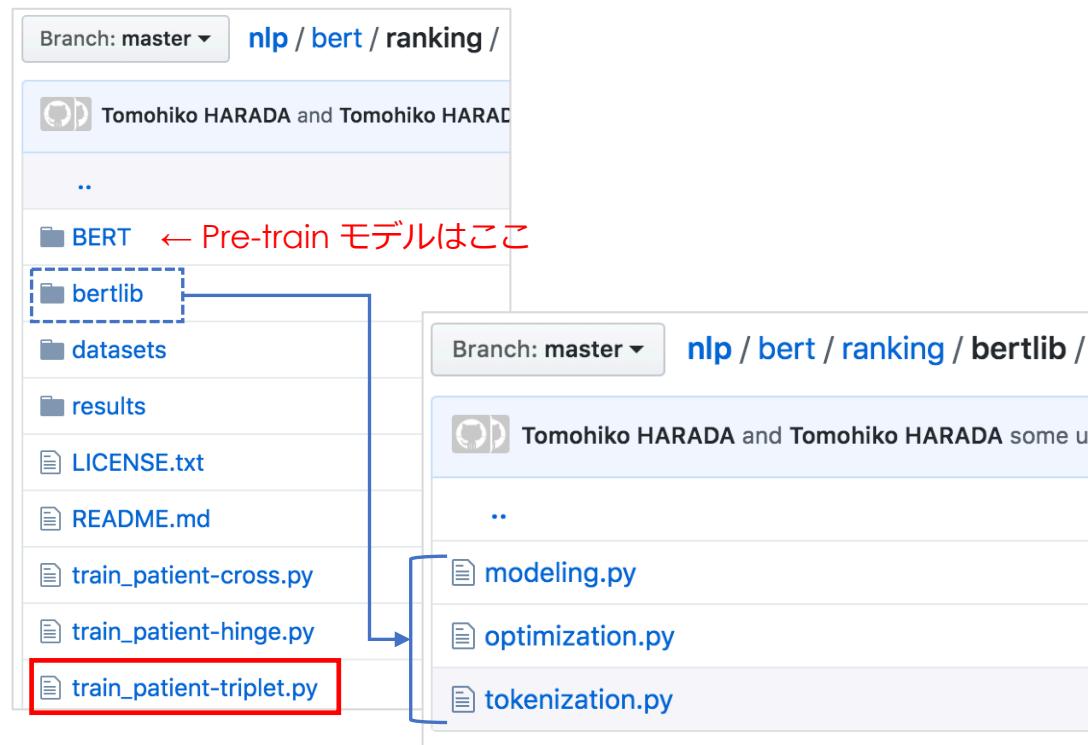
# テキスト検索: 実装

タイプ(b): 単文の分類  
→ [CLS] のベクトルを取り出す  
→ **get\_pooled\_output()**

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ranking>

- **train\_patient-triplet.py** : クエリ-正例-負例 の距離を学習

<https://github.com/haradatm/nlp/tree/master/bert/ranking>



```
class BertRanking(chainer.Chain):
    def __init__(self, bert, num_labels):
        :
    def forward(self, input_ids, input_mask, token_type_ids, labels):
        output_layer = self.bert.get_pooled_output(input_ids, ...)
        output_layer = F.dropout(output_layer, 0.1)
        output_layer = self.output(output_layer)
        return output_layer
```

```
def __call__(self, q_x1, q_x2, q_x3, d_x1, d_x2, d_x3, n_x1, n_x2, n_x3):
    y_0 = self.forward(q_x1, q_x2, q_x3)
    y_1 = self.forward(d_x1, d_x2, d_x3)
    y_2 = self.forward(n_x1, n_x2, n_x3)
    loss = F.triplet(y_0, y_1, y_2)
    return loss
```

```
def main():
    :
    bert_config = BertConfig.from_json_file(bert_config_file) ...
    (テキスト分類と同じ)
```

# テキスト検索: 実験データ

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ranking>

- Datasets

- 日本語: MedNLP 「患者表現辞書」 → 不具合分類 16クラス

患者が実際に用いる病名表現を網羅した辞書

- <http://sociocom.jp/~data/2019-pde/>
    - 患者表現-標準病名ペア 8,483件 → 学習 7,483, 評価 1,000 に分割 (重複なし)
    - ペアを正例, 負例はペアでない組み合わせをランダムに選択
    - テキストは, あらかじめ Juman++ で分かち書き

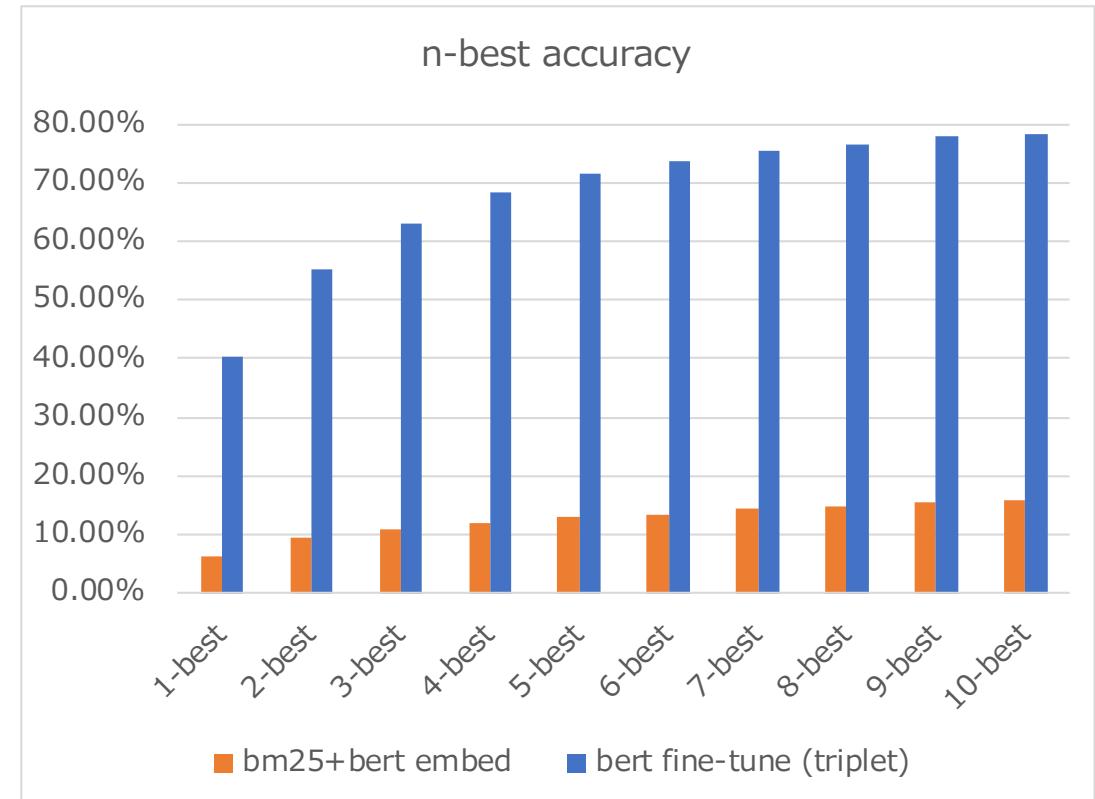
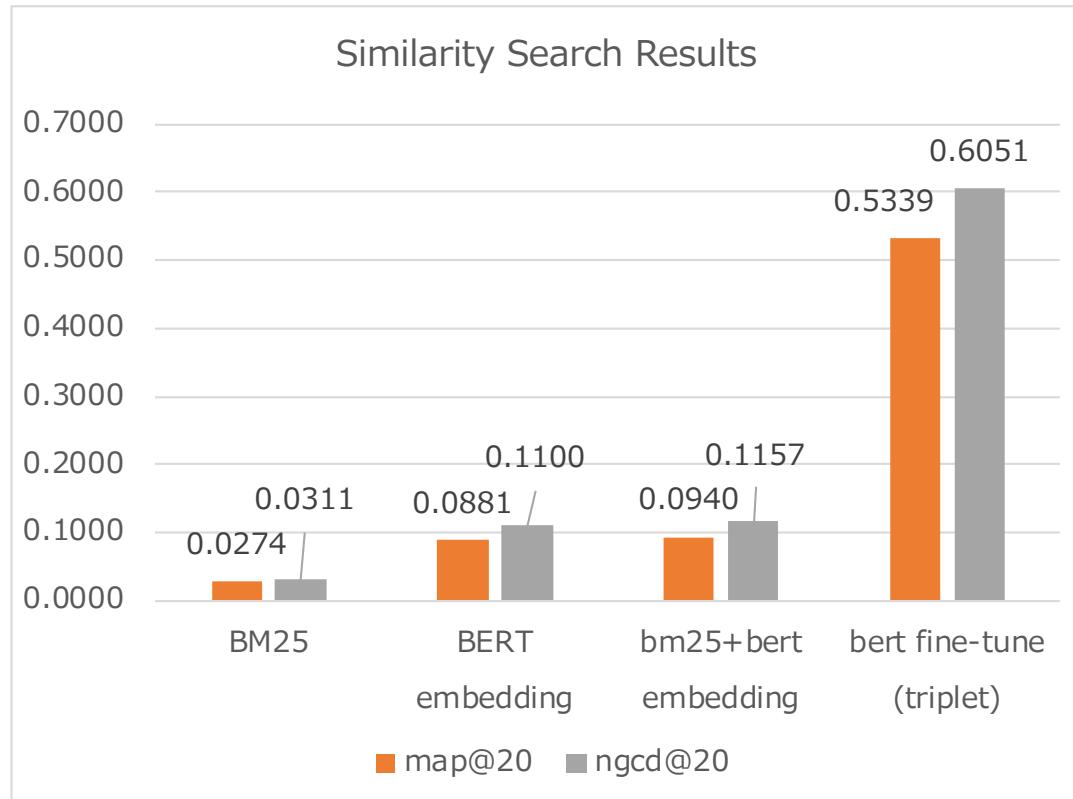
- BERT モデル

- 日本語: 黒橋研, Japanese\_L-12\_H-768\_A-12\_E-30\_BPE

# テキスト検索: 実験結果

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ranking>

- 表層が一致しない検索も, fine-tuning で「ある程度」検索可能



# テキスト検索: 実験結果

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ranking>

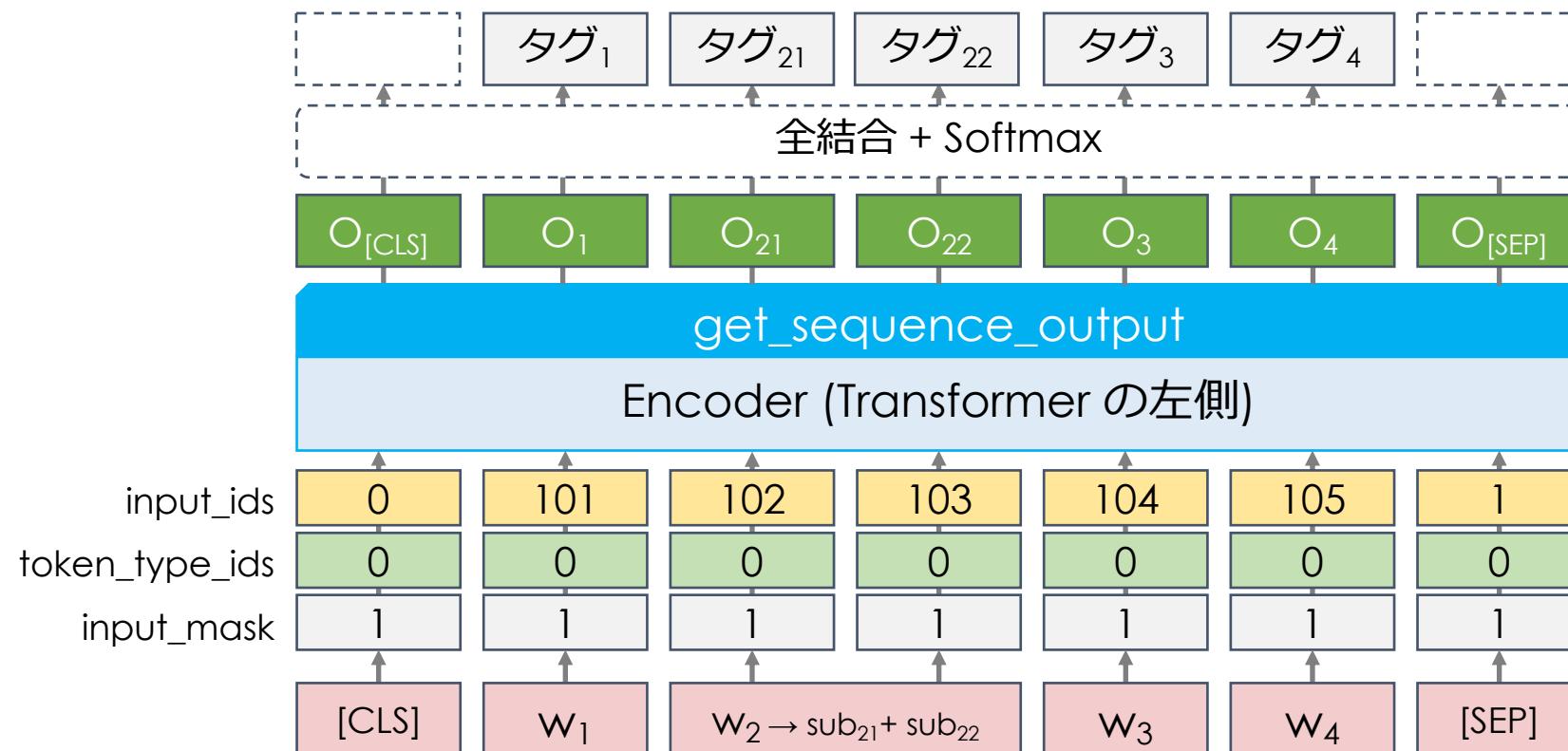
Q		Top-1		Similarity	Correct	Top-2		Similarity	Correct	Top-3		Similarity	Correct	Top-4		Similarity	Correct
Q-0000	喉が渴く	D-0183	乾性咳	0.885599	NG	D-0112	口渴症	0.881856	NG	D-0299	口内乾燥	0.858879	NG	D-0340	咽喉痛	0.855624	NG
Q-0001	見えないもの	D-0137	視力低下	0.911380	NG	D-0029	眼前暗黒	0.897037	NG	D-0208	視力障害	0.890591	NG	D-0001	幻覚	0.861191	OK
Q-0002	はり	D-0167	浮腫	0.869764	NG	D-0004	腹部膨満	0.842695	NG	D-0096	リンパ節腫大	0.839203	NG	D-0081	全身性浮腫	0.763212	NG
Q-0003	目で解る血の	D-0003	肉眼的血尿	0.942998	OK	D-0074	顕微鏡的血尿	0.932888	NG	D-0170	血尿	0.903606	NG	D-0107	蛋白尿	0.858269	NG
Q-0004	お腹が張って	D-0004	腹部膨満	0.929448	OK	D-0093	腹部膨満感	0.877450	NG	D-0016	腹部不快感	0.865900	NG	D-0172	肥満症	0.825045	NG
Q-0005	足の痛み	D-0204	四肢痛	0.913404	NG	D-0329	下肢関節痛	0.904597	NG	D-0005	疼痛	0.875342	OK	D-0063	膝関節痛	0.842915	NG
Q-0006	すっぽいのが	D-0006	過酸症	0.865487	OK	D-0154	高コレステロール	0.819257	NG	D-0182	高炭酸ガス血	0.814275	NG	D-0222	多汗症	0.804447	NG
Q-0007	声がでにくい	D-0007	発声異常,発声	0.934171	OK	D-0026	構音障害	0.897652	NG	D-0111	失語症	0.870566	NG	D-0288	構語障害	0.827894	NG
Q-0008	腹がふとる	D-0016	腹部不快感	0.901254	NG	D-0093	腹部膨満感	0.893847	NG	D-0004	腹部膨満	0.893364	OK	D-0315	腹部腫瘍	0.832161	NG
Q-0009	吐き気や吐く	D-0138	嘔気,嘔吐症	0.952630	NG	D-0008	恶心,嘔吐症	0.947225	OK	D-0142	嘔吐症	0.941800	NG	D-0294	恶心	0.818538	NG
Q-0010	呼吸がゼーゼ	D-0115	呼吸麻痺	0.923955	NG	D-0109	息切れ	0.901369	NG	D-0243	無呼吸発作	0.898481	NG	D-0000	多飲症	0.880517	NG
Q-0011	くされ	D-0018	腹水症	0.857270	NG	D-0142	嘔吐症	0.834113	NG	D-0060	脾腫	0.813213	NG	D-0127	動悸	0.813213	NG
Q-0012	ビリビリする	D-0011	しびれ感	0.866530	OK	D-0045	痙攣	0.825154	NG	D-0015	体感異常	0.788017	NG	D-0301	よろめき感	0.785956	NG
Q-0013	三しゃ神経痛	D-0026	構音障害	0.793456	NG	D-0090	急性低音障害	0.785502	NG	D-0288	構語障害	0.782463	NG	D-0012	三叉神経痛	0.780175	OK
Q-0014	ピリピリ感ジ	D-0011	しびれ感	0.919487	OK	D-0280	温痛覚過敏	0.846491	NG	D-0076	感覚異常症	0.845158	NG	D-0015	体感異常	0.844381	NG
Q-0015	意志に基づか	D-0013	不随意運動症	0.929022	OK	D-0181	活動低下状態	0.870676	NG	D-0084	異常行動	0.847516	NG	D-0206	行動緩慢	0.845760	NG
Q-0016	定期的な熱	D-0014	発熱	0.968770	OK	D-0121	微熱	0.962012	NG	D-0145	熱傷	0.935024	NG	D-0327	ほてり	0.926469	NG
Q-0017	まひ	D-0207	失神	0.895613	NG	D-0045	痙攣	0.877344	NG	D-0264	痙攣発作	0.876228	NG	D-0015	体感異常	0.866194	OK
Q-0018	いたさ	D-0005	疼痛	0.894811	OK	D-0184	鈍痛	0.879278	NG	D-0168	圧痛	0.802014	NG	D-0204	四肢痛	0.797205	NG
Q-0019	ふかいかん	D-0200	摂食機能障害	0.788983	NG	D-0034	食欲不振	0.783243	NG	D-0294	恶心	0.759134	NG	D-0301	よろめき感	0.747941	NG
Q-0020	べんつうがわ	D-0017	便秘症	0.863811	OK	D-0213	平衡異常	0.697408	NG	D-0198	頭痛	0.639967	NG	D-0016	腹部不快感	0.611184	NG
Q-0021	腎臓劣化	D-0053	肝機能検査異	0.851678	NG	D-0104	低血糖	0.846801	NG	D-0042	高血糖症	0.832775	NG	D-0023	腎機能検査異	0.807451	NG
Q-0022	酸っぱいのが	D-0182	高炭酸ガス血	0.910273	NG	D-0154	高コレステロール	0.887734	NG	D-0006	過酸症	0.873757	OK	D-0321	血中亜鉛異常	0.841391	NG
Q-0023	たくさん水分	D-0101	多尿	0.940549	NG	D-0277	多尿,頻尿症	0.930544	NG	D-0019	多飲症,多尿	0.915281	OK	D-0310	頻尿症,夜間頻	0.908842	NG
Q-0024	目が回るよう	D-0091	めまい感	0.872981	NG	D-0020	めまい	0.815685	OK	D-0301	よろめき感	0.797223	NG	D-0029	眼前暗黒	0.745551	NG
Q-0025	腸のあたりが	D-0016	腹部不快感	0.929498	OK	D-0155	心窓部不快	0.866360	NG	D-0093	腹部膨満感	0.862250	NG	D-0068	胸部不快感	0.818078	NG

# タグ付け-NER: 実装

タイプ(c): タグ付け  
→ 各トークンのベクトルを取り出す  
→ **get\_sequence\_output()**

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ner>

- **train\_ner.py** : input\_ids, token\_type\_ids, input\_mask は以下の通り



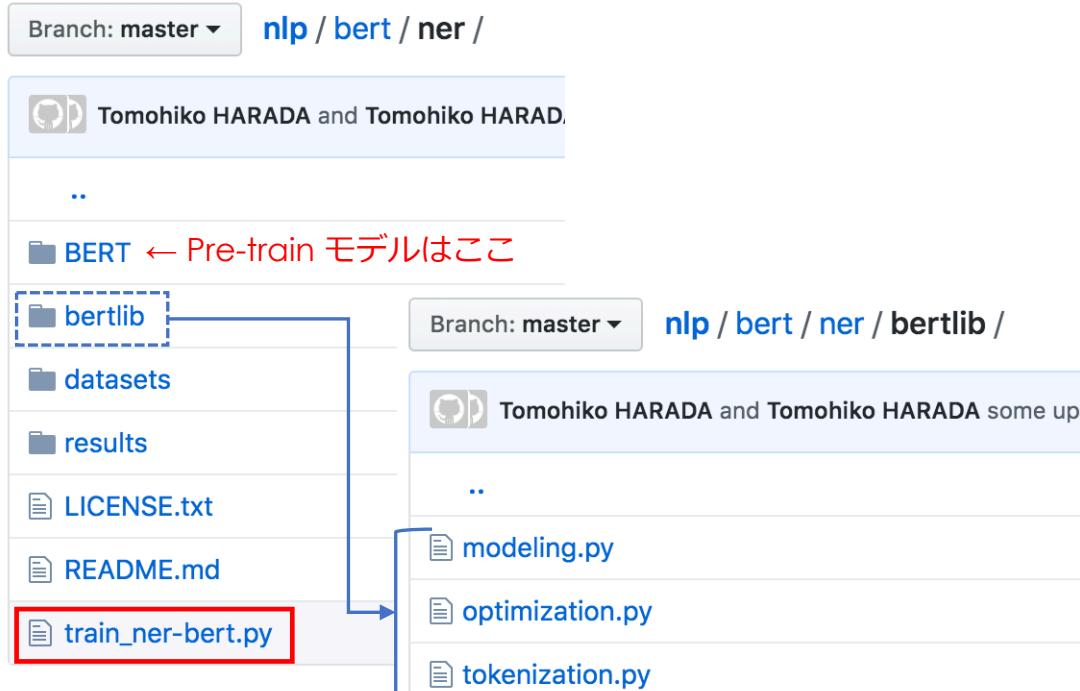
- input\_ids
    1. FullTokenizer.tokenize() で入力文をトークン化
    2. FullTokenizer.convert\_tokens\_to\_ids() で ID に変換してセット
  - token\_type\_ids  
すべて同じ=0 をセット
  - input\_mask  
すべて同じ=1 をセット
- ※ FullTokenizer は、tokenization.py が提供

# タグ付け-NER: 実装

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ner>

- **train\_ner.py** : BERT 実装にソースがないため, 実装した

<https://github.com/haradatm/nlp/tree/master/bert/ner>



```
class BertNER(chainer.Chain):
    def __init__(self, bert, num_labels):
        :
    def forward(self, input_ids, input_mask, token_type_ids, labels):
        final_hidden = self.bert.get_sequence_output(input_ids, ...)
        batch_size, seq_length, hidden_size = final_hidden.shape
        final_hidden_matrix = F.reshape(final_hidden, [batch_size * seq_length, hidden_size])
        output_layer = self.output(final_hidden_matrix)
        output_layer = F.reshape(output_layer, [batch_size, seq_length, self.n_label])
        return output_layer

    def __call__(self, input_ids, input_mask, token_type_ids, label_ids):
        logits = self.forward(input_ids, input_mask, token_type_ids)
        batch_size, seq_length, class_size = logits.shape
        y = F.reshape(logits, [batch_size * seq_length, class_size])
        t = F.reshape(label_ids, [batch_size * seq_length])
        loss = F.softmax_cross_entropy(y, t, ignore_label=0)
        return loss, logits

    def main(): (テキスト分類と同じ)
```

# タグ付け-NER: 実験データ

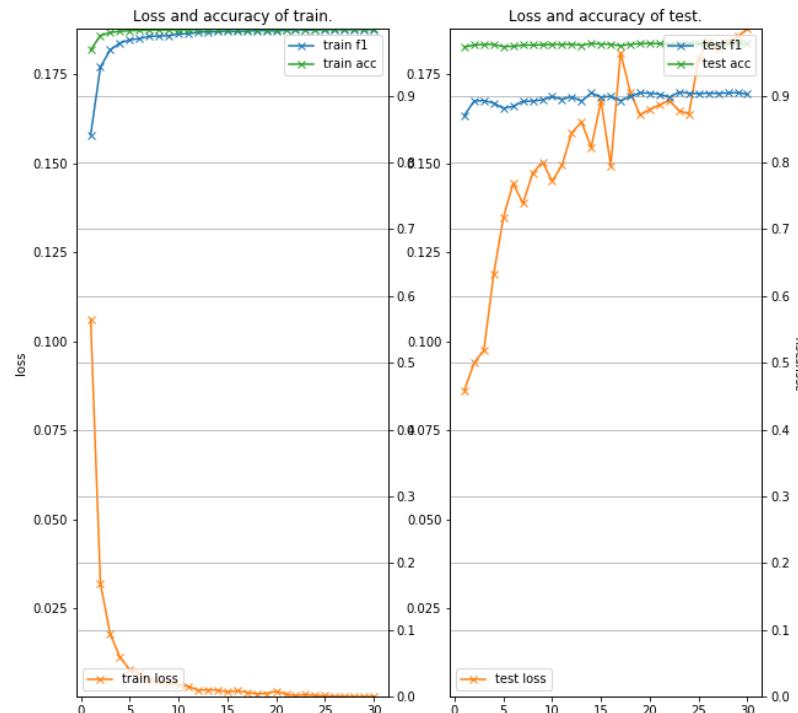
詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ner>

- Datasets
  - 英語: CoNLL-2013 NER datasets → 固有表現 4ラベル
    - <https://github.com/glample/tagger/tree/master/dataset>
    - PER(人名), LOC(場所), ORG(組織), MISC(その他) → IOBタグは11種
    - 全8,483件 → 学習14,041, 評価3,453に分割
    - Tokenizerで分割されたサブワードには,論文に従いラベル'X'を付与
  - BERT モデル
    - 英語: Google AI, [uncased\\_L-12\\_H-768\\_A-12](#)

# タグ付け-NER: 実験結果

詳細→ <https://github.com/haradatm/nlp/tree/master/bert/ner>

- 論文中の 92.4 には届かず



===== Classification report (early-stopped model) =====

	precision	recall	f1-score	support
PER	0.96	0.96	0.96	1617
LOC	0.92	0.93	0.92	1668
ORG	0.88	0.88	0.88	1661
MISC	0.80	0.79	0.79	702
micro avg	0.90	0.91	0.91	5648
macro avg	0.90	0.91	0.91	5648

# BERTology 論文

Knowledge Graph 界隈と BERT 分析

# Knowledge Graph (1)

Simple BERT Models for Relation Extraction and Semantic Role Labeling

文献[7]

- Relation Extraction のための Simple な拡張

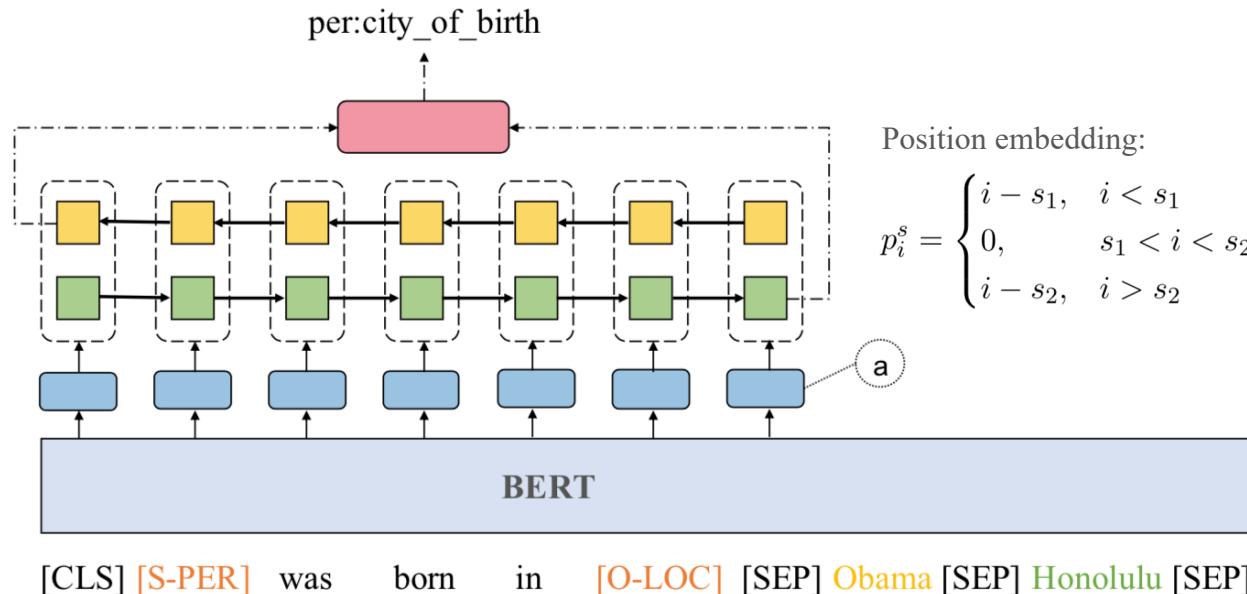


Figure 1: Architecture of our relation extraction model. (a) denotes the concatenation of BERT contextual embedding and position embedding. The final prediction is based on the concatenation of the final hidden state in each direction from the BiLSTM, fed through an MLP.

Model	P	R	F <sub>1</sub>
Zhang et al. (2017)	65.7	64.5	65.1
Zhang et al. (2018)	69.9	63.33	66.4
Wu et al. (2019)	-	-	67.0
Alt et al. (2019)	70.1	65.0	67.4
BERT-LSTM-base	<b>73.3</b>	63.10	67.8
Zhang et al. (2018) (ensemble)	71.3	<b>65.4</b>	<b>68.2</b>

Table 1: Results on the TACRED test set.

- 入力は [[CLS]文[SEP] subject [SEP] object [SEP]]
- 過適合を防ぐため, 文中の実体をマスクで置き換える ( $S_{\text{ubj}}\text{-PER}$ や $O_{\text{bj}}\text{-LOC}$ )
- BERTの出力のうち [SEP]以降のシーケンスを破棄
- 位置埋め込みと連結して, 一層の BiLSTM と MLPで予測

# Knowledge Graph (2)

ERNIE: Enhanced Language Representation with Informative Entities

- テキストとKGの両方を活用した拡張言語表現モデル

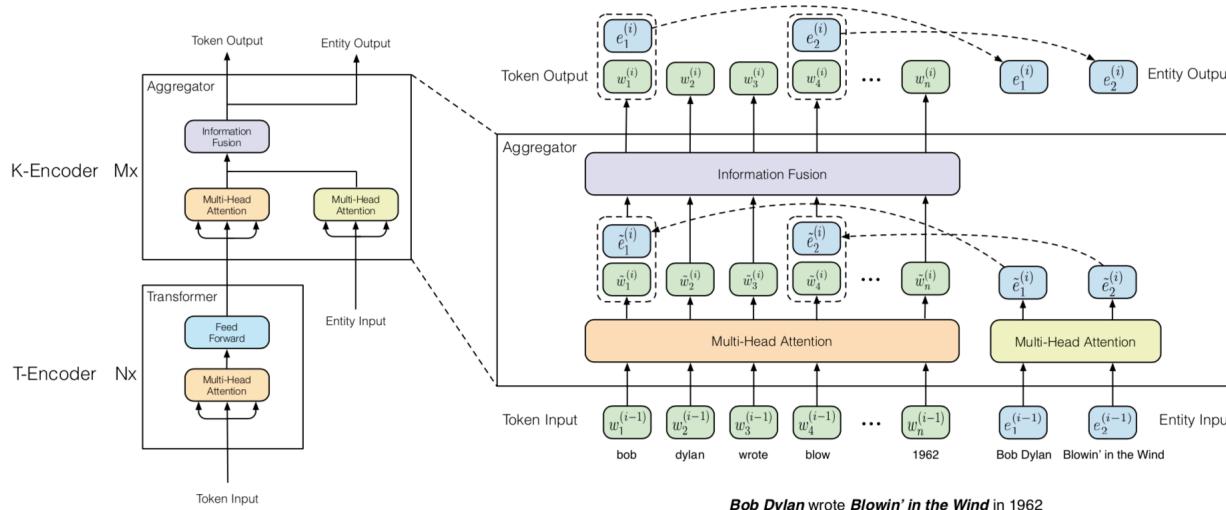


Figure 2: The left part is the architecture of ERNIE.

*Mark Twain wrote The Million Pound Bank Note in 1893.*

Input for Entity Typing:

[CLS] [ENT] mark twain [ENT] wrote [ ] the million pound bank note [ ] in 1893 [ ] [SEP]

Input for Relation Classification:

[CLS] [HD] mark twain [HD] wrote [TL] the million pound bank note [TL] in 1893 [ ] [SEP]

← Figure 3: Modifying the input sequence for the specific tasks.

- T-Encoderは、BERTの実装と一緒にで、トークンを埋め込む
- エンティティは、TransEなどの効果的なKG埋め込みで事前学習
- K-Encoderは、知識情報を言語表現に注入し、トークンとエンティティの異種情報を統一空間で表現
- 同一構造で Relation Extraction と Entity Typing の両タスクに対応

Model	FewRel		TACRED			
	P	R	F1	P	R	F1
CNN	69.51	69.64	69.35	70.30	54.20	61.20
PA-LSTM	-	-	-	65.70	64.50	65.10
C-GCN	-	-	-	69.90	63.30	66.40
BERT	85.05	85.11	84.89	67.23	64.81	66.00
ERNIE	88.49	88.44	<b>88.32</b>	69.97	66.08	<b>67.97</b>

Table 5: Results of various models on FewRel and TACRED (%).

# BERT自体の分析

Revealing the Dark Secrets of BERT [EMNLP'19]

文献[8]

- BERT の Self-Attention を分析した論文

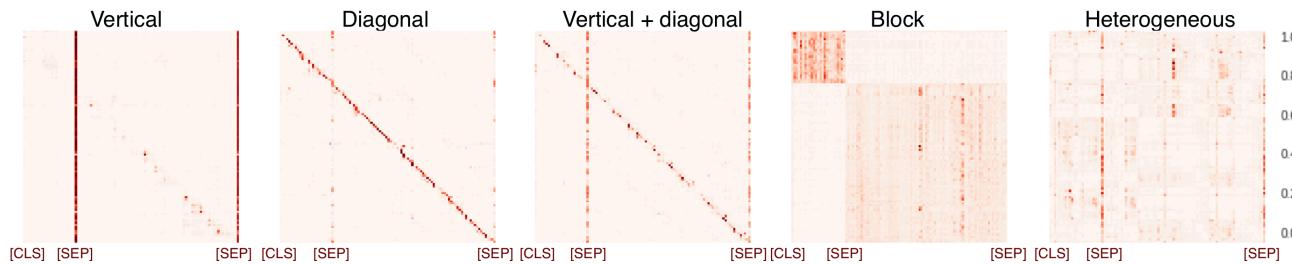


図1: アテンションを5パターンに分類。最初の3つは事前学習に関連、最後の2つは意味や構文情報を捉える。異なるヘッドで同じパターンの繰り返しがあった

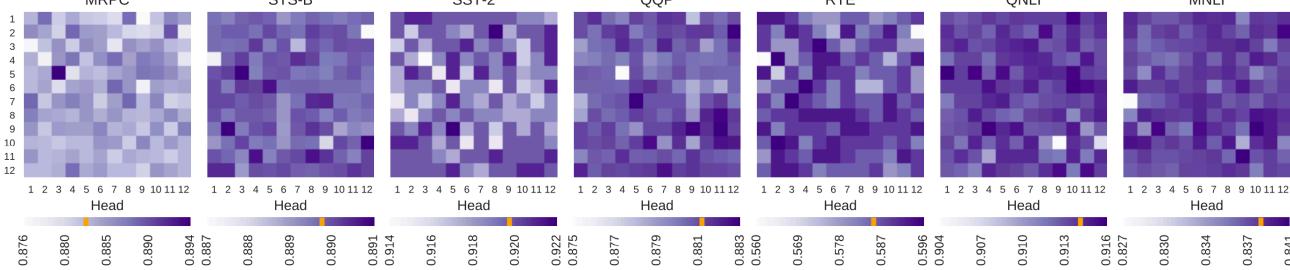


図8: ヘッドを1つ無効にしたときの精度 →すべて使う(オレンジ)より良い場合がある

- 異なるヘッドでタスクに関係なく同じパターンが一貫して繰り返された
- モデルの枝刈りとデータの繰り返しを減らす最適なアーキテクチャの発見を示唆

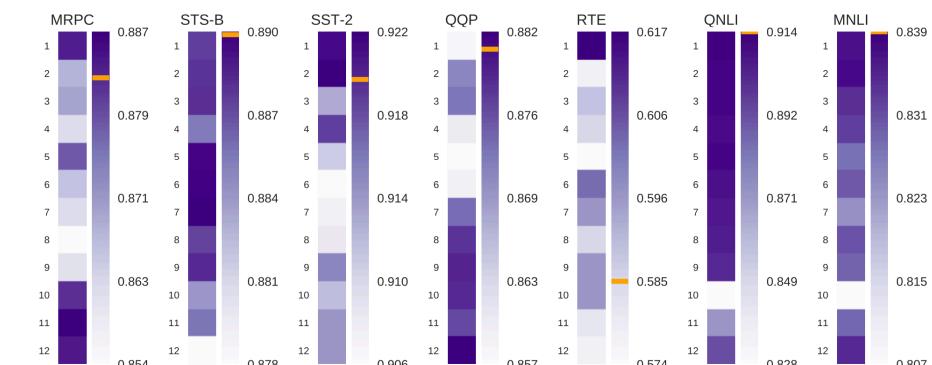


図9: レイヤー(全12ヘッド)を1つ無効にしたときの精度 →すべて使う(オレンジ)より良い場合がある

# BERT fine-tuning まとめ

- ・事前学習モデルとタスクに応じた **fine-tuning** で、容易に良好な性能が得られることを確認
- ・**BertModel クラス**にある 2つの Function の使い方をマスターするだけで、応用範囲は広い
  - ・[CLS] のベクトルを取り出す **get\_pooled\_output()**
  - ・各トークンのベクトルを取り出す **get\_sequence\_output()**
- ・様々な応用タスクへの適用等、楽しい **BERTology** 活動を!

# 文献

- [1] 西田 京介. "機械読解の現状と展望." 言語処理学会第25回年次大会チュートリアル資料, 2019 (2019).
- [2] Jacob Devlin, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." (2018)
- [3] Wang, Alex, et al. "Glue: A multi-task benchmark and analysis platform for natural language understanding." arXiv preprint arXiv:1804.07461 (2018).
- [4] 柴田 知秀, 河原 大輔, 黒橋 穎夫. "BERTによる日本語構文解析の精度向上." 言語処理学会 第25回年次大会, pp.205-208, 名古屋, (2019.3).
- [5] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.
- [6] Klein, Guillaume, et al. "Opennmt: Open-source toolkit for neural machine translation." arXiv preprint arXiv:1701.02810 (2017).

# 文献

- [7] Shi, Peng, and Jimmy Lin. "Simple BERT Models for Relation Extraction and Semantic Role Labeling." arXiv preprint arXiv:1904.05255 (2019).
- [8] Zhang, Zhengyan, et al. "ERNIE: Enhanced Language Representation with Informative Entities." arXiv preprint arXiv:1905.07129 (2019).
- [9] Kovaleva, Olga, et al. "Revealing the Dark Secrets of BERT." arXiv preprint arXiv:1908.08593 (2019).

# 付録: その他の汎用言語モデル

# 2019年5月: XLNet

文献[10]

- BERT の弱点を修正し, 20以上のタスクで BERT を超えた

## SQuAD1.1 Leaderboard

Here are the ExactMatch (EM) and F1 scores evaluated on the test set of SQuAD v1.1.

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar et al. '16)	82.304	91.221
1 May 21, 2019	XLNet (single model) XLNet Team	89.898 87.433	95.080 93.160
2 Oct 05, 2018	BERT (ensemble) Google AI Language <a href="https://arxiv.org/abs/1810.04805">https://arxiv.org/abs/1810.04805</a>		



<https://rajpurkar.github.io/SQuAD-explorer/>

- BERT のマスク単語予測の問題(通常発生しないためノイズにもなる)を克服
- 20タスクで BERT を超えた(2018/5/21)
- 機械読解タスク(左)では, single モデルで BERT の ensemble を超える

# 2019年2月: GPT-2

- 超大規模コーパス(800万文書)で学習した超巨大サイズのGPT

Language Models are Unsupervised Multitask Learners

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	<b>21.8</b>
117M	<b>35.13</b>	45.99	<b>87.65</b>	<b>83.4</b>	<b>29.41</b>	65.85	1.16	1.17	37.50	75.20
345M	<b>15.60</b>	55.48	<b>92.35</b>	<b>87.1</b>	<b>22.76</b>	47.33	1.01	<b>1.06</b>	26.37	55.72
762M	<b>10.87</b>	<b>60.12</b>	<b>93.45</b>	<b>88.0</b>	<b>19.93</b>	<b>40.31</b>	<b>0.97</b>	<b>1.02</b>	22.05	44.575
1542M	<b>8.63</b>	<b>63.24</b>	<b>93.30</b>	<b>89.05</b>	<b>18.34</b>	<b>35.76</b>	<b>0.93</b>	<b>0.98</b>	<b>17.48</b>	42.16

Table 3. Zero-shot results on many datasets.

- コーパスはCommon Crawlから人間がcurate/filterおよびRedditで絞り込み
- 1600次元、1024トークン(BPE)の修正版GPT(パラメタ数は元のGPTの10倍)
- zero-shotで7/8の言語モデルのタスクでTransformer-XLを上回りSOTA(上表)
- zero-shotで読解,要約,翻訳にも「ある程度」適応可(タスクに有用な表現が多い場合)

# 2019年7月: RoBERTa

- 言語理解タスクの事前学習のBERTを改善し、多くのタスクでSOTAを更新、SuperGLUEタスクでも人の成績に近く

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT <sub>LARGE</sub>	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet <sub>LARGE</sub>	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	<b>90.2/90.2</b>	<b>94.7</b>	<b>92.2</b>	<b>86.6</b>	<b>96.4</b>	<b>90.9</b>	<b>68.0</b>	<b>92.4</b>	<b>91.3</b>	-
<i>Ensembles on test (from leaderboard as of July 25, 2019)</i>										
ALICE	88.2/87.9	95.7	<b>90.7</b>	83.5	95.2	92.6	<b>68.6</b>	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	<b>96.8</b>	<b>93.0</b>	67.8	91.6	<b>90.4</b>	88.4
RoBERTa	<b>90.8/90.2</b>	<b>98.9</b>	90.2	<b>88.2</b>	96.7	92.3	67.8	<b>92.2</b>	89.0	<b>88.5</b>

Table 5: Results on GLUE. All results are based on a 24-layer architecture. BERT<sub>LARGE</sub> and XLNet<sub>LARGE</sub> results are from Devlin et al. (2019) and Yang et al. (2019), respectively. RoBERTa results on the development set are a median over five runs. RoBERTa results on the test set are ensembles of *single-task* models. For RTE, STS and MRPC we finetune starting from the MNLI model instead of the baseline pretrained model. Averages are obtained from the GLUE leaderboard.

BERTの改善:

- (1)動的なマスク
- (2)次文の予測を削除
- (3)大きなバッチサイズ
- (4)走査回数を増やす
- (5)文字ではなくバイト単位のBPEを使用

# SuperGLUE

<https://w4ngatang.github.io/static/papers/superglue.pdf>

文献[13]

- GLUEを新しいタスクで作り直したベンチマーク
  - 分類/文関係(NLI)が減り, 文書理解系(共参照/意味識別)が増えている

Corpus	Train	Dev	Test	Task	Metrics	Text Sources
BoolQ	9427	3270	3245	QA	acc.	Google queries, Wikipedia
CB	250	57	250	NLI	acc./F1	various
COPA	400	100	500	QA	acc.	blogs, photography encyclopedia
MultiRC	5100	953	1800	QA	F1 <sub>a</sub> /EM	various
ReCoRD	101k	10k	10k	QA	F1/EM	news (CNN, Daily Mail)
RTE	2500	278	300	NLI	acc.	news, Wikipedia
WiC	6000	638	1400	WSD	acc.	WordNet, VerbNet, Wiktionary
WSC	554	104	146	coref.	acc.	fiction books

# 文献

- [10] Yang, Zhilin, et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding." arXiv preprint arXiv:1906.08237 (2019).
- [11] Radford, Alec, et al. "Language models are unsupervised multitask learners." OpenAI Blog 1.8 (2019).
- [12] Liu, Yinhan, et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." arXiv preprint arXiv:1907.11692 (2019).
- [13] Wang, Alex, et al. "Superglue: A stickier benchmark for general-purpose language understanding systems." arXiv preprint arXiv:1905.00537 (2019).