Use of BERT for NLP tasks by HuggingFace's transformers

ROCLING 2020

Sep.25 2020

Keynote Speech B

Ibaraki University (JAPAN)

Hiroyuki Shinnou

Self Introduction

新納 浩幸（SHINNOU Hiroyuki）

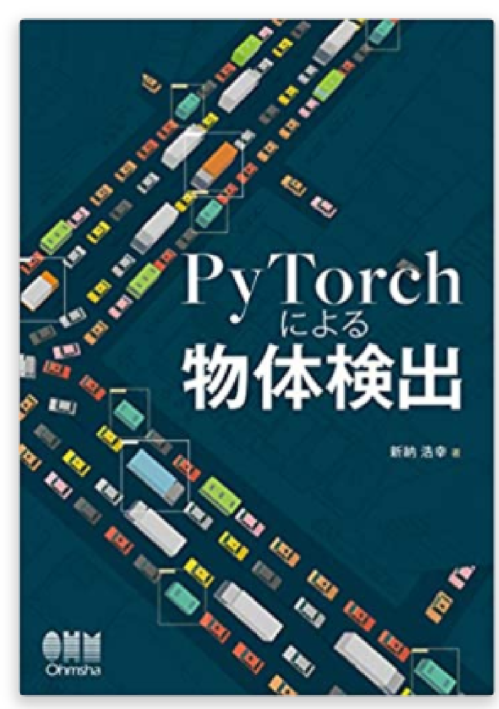
hiroyuki.shinnou.0828@vc.ibaraki.ac.jp

Ibaraki University（Professor）

I've been researching NLP ever since I graduated from the university. Recently, I study about transfer learning of pretraining model like BERT, and object detection in CV domain.

Last week, my book was published.
Please buy it if you can read Japanese.

本 › オーム社



PyTorchによる物体検出 (日本語) 単行本 –
2020/9/19
新納 浩幸 (著)

› その他 の形式およびエディションを表示する

単行本
￥3,300
獲得ポイント: 33pt

￥3,300 より 1 新品

「予約商品の価格保証」対象商品。 詳細

無料配送

PyTorchで物体検出アルゴリズムを実装しよう!

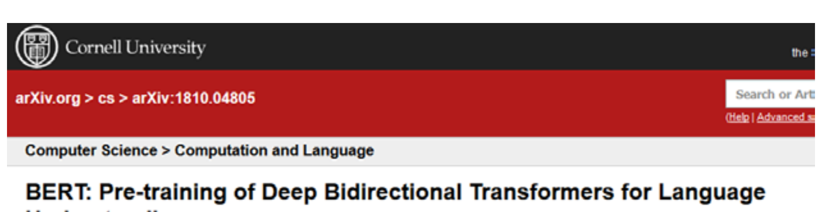
2点すべてのイメージを見る

Agenda

1. Introduction of BERT
2. Input/Output of BERT
3. Use of BERT through transformers
4. Downsizing of BERT model

- 1 -
Introduction of BERT

BERT


Cornell University

arXiv.org > cs > arXiv:1810.04805

Computer Science > Computation and Language

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

(Submitted on 11 Oct 2018)

We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT representations can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.
BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE benchmark to 80.4% (7.6% absolute improvement), MultiNLI accuracy to 86.7 (5.6% absolute improvement) and the SQuAD v1.1 question answering Test F1 to 93.2 (1.5% absolute improvement), outperforming human performance by 2.0%.

https://arxiv.org/abs/1810.04805

Oct. 2018.

Surpassed ELMo by far

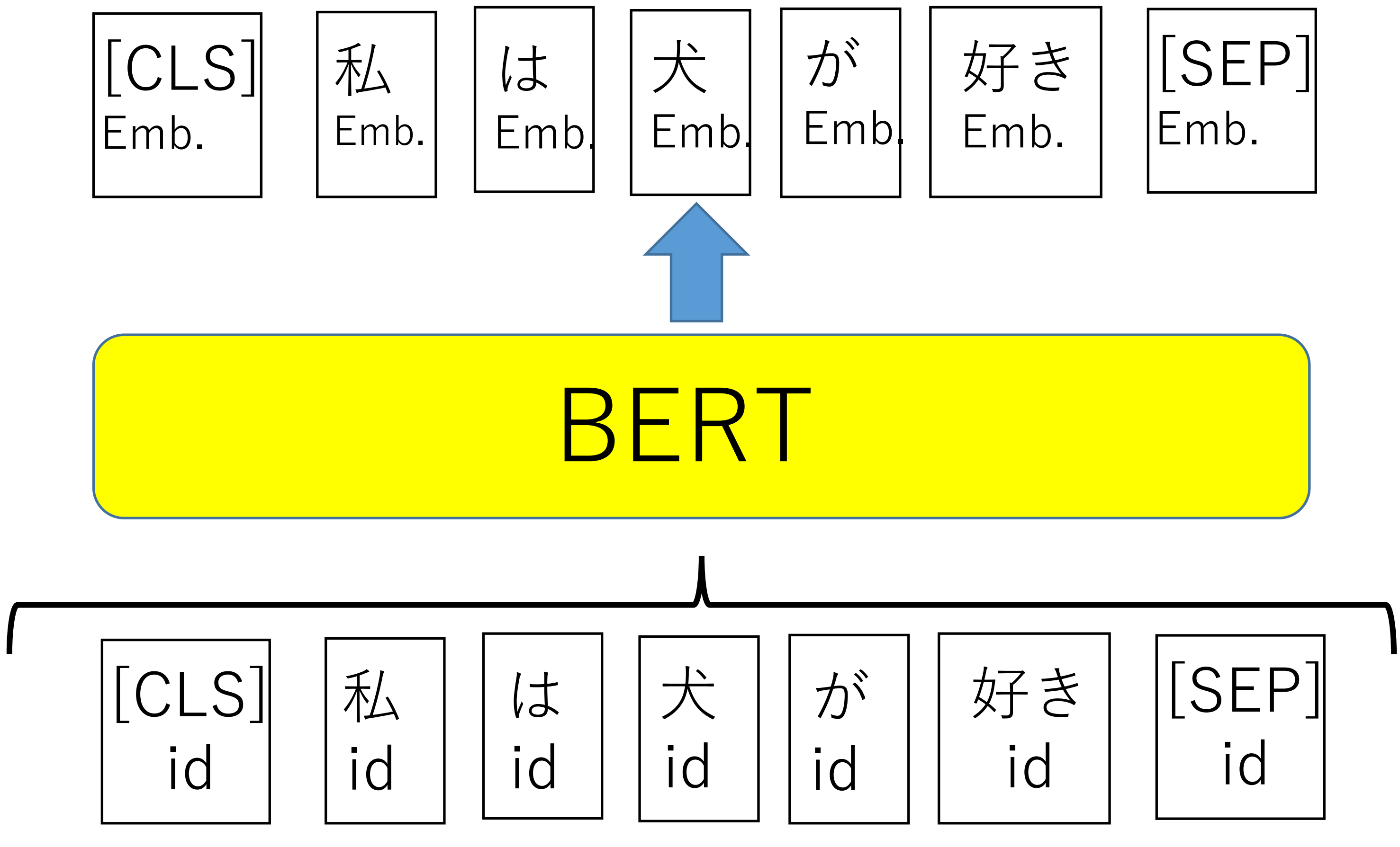
May. 2018.

Biggest impact on NLP world since word2vec !

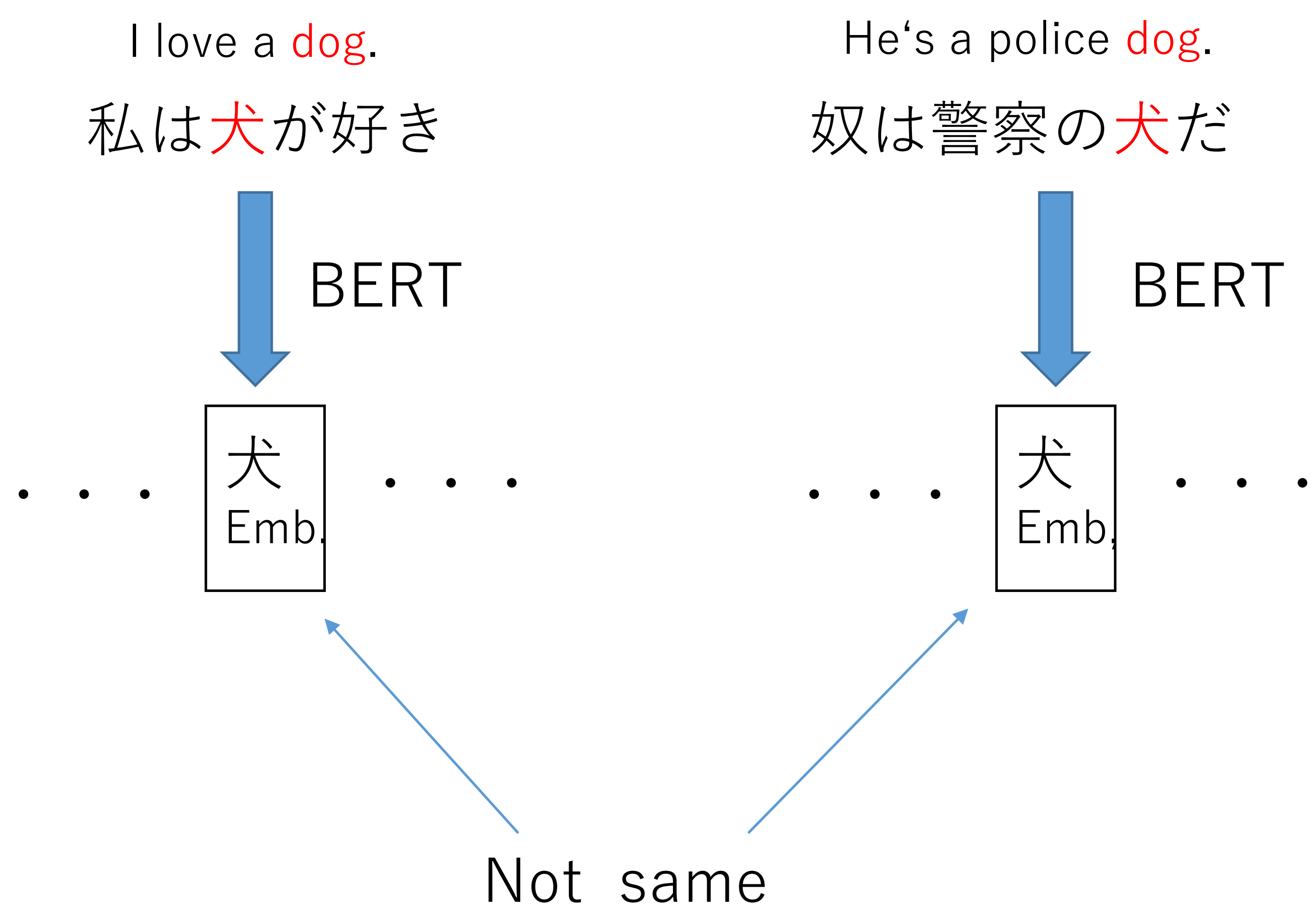
Main technique is multi-head attention used in the Transformer.

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Image of Input/Output of BERT

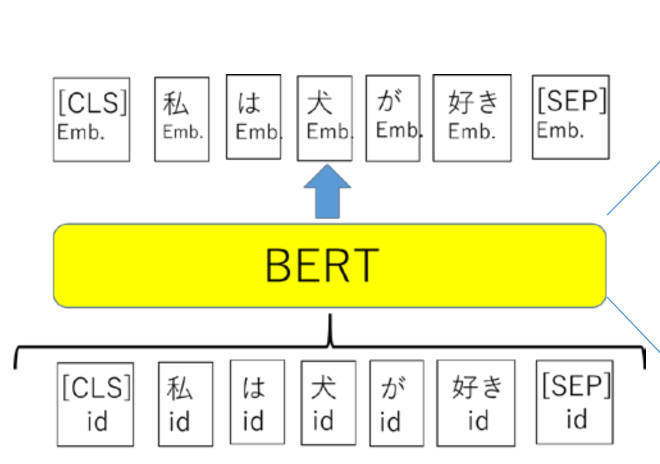


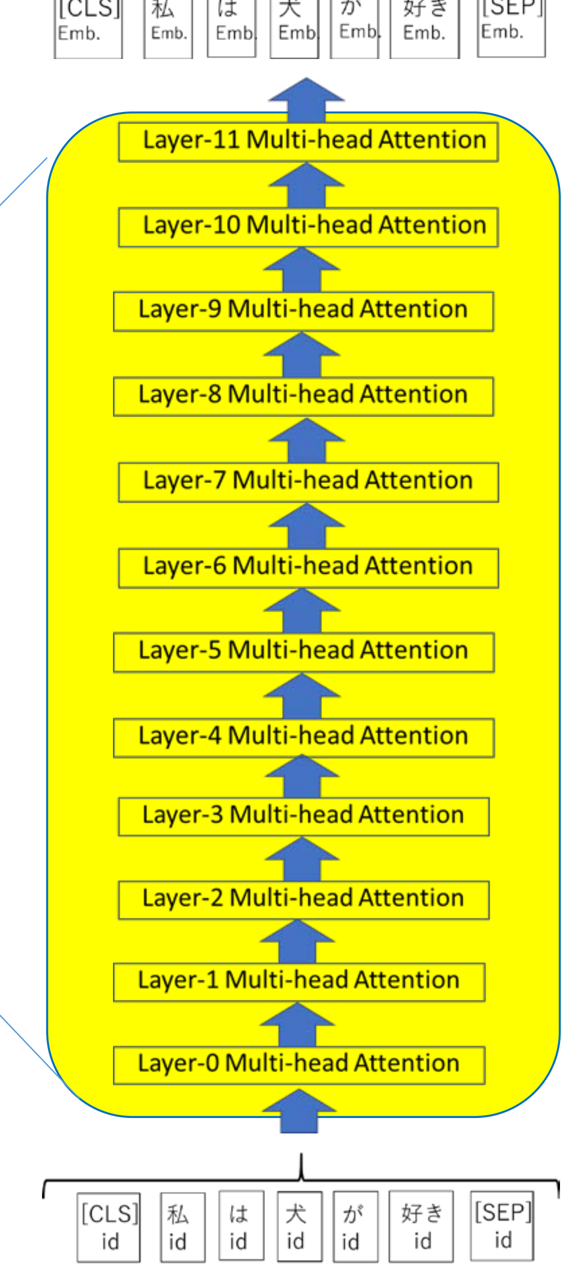
Contextual word embeddings



In word2vec, two embeddings are same.

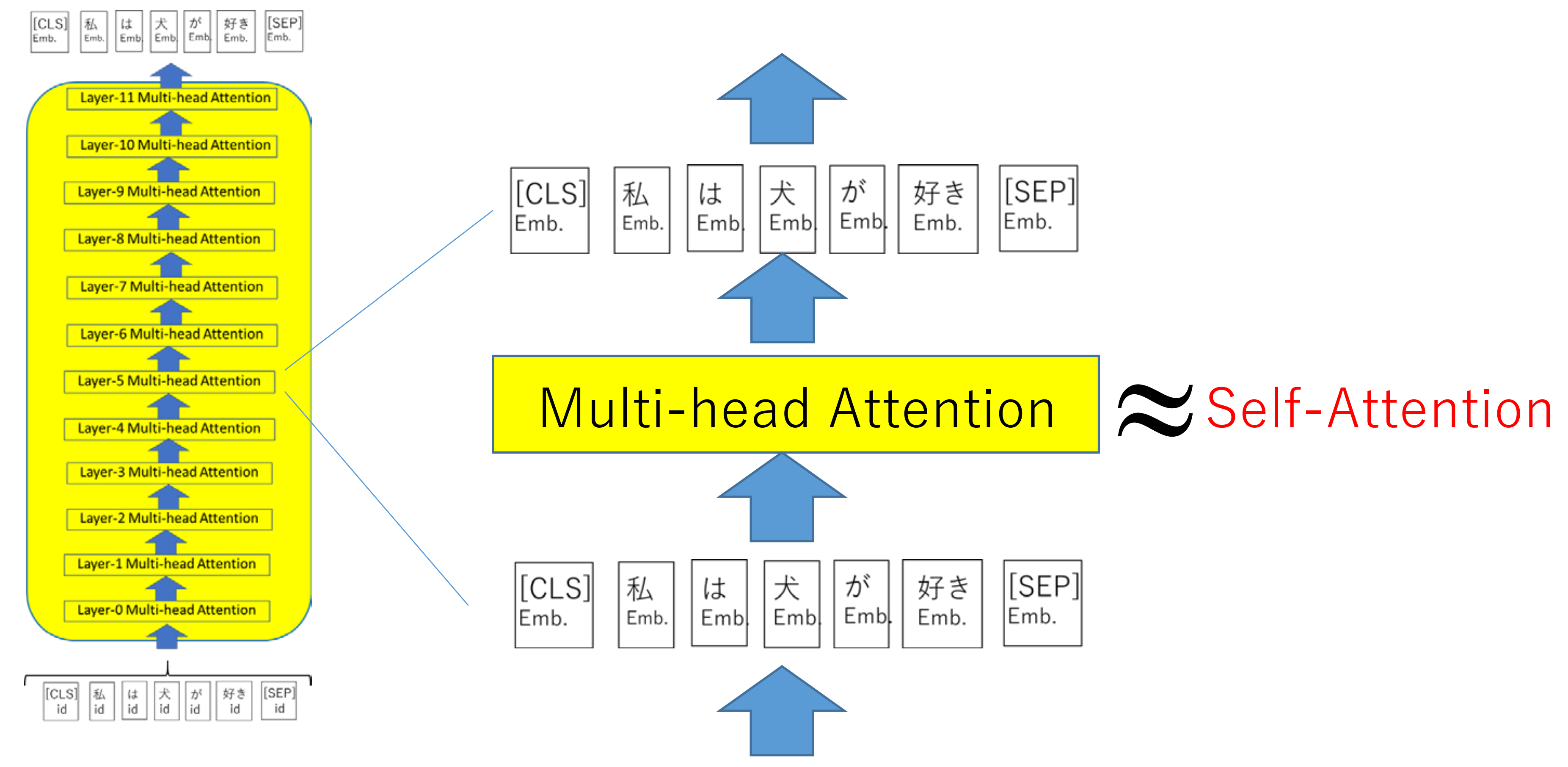
Multi-head Attention





Self-Attention

Multi-head Attention is not self-attention.
But they are similar, and same input/output form.



Query, Key, Value

$$X = [x_1, x_2, \cdots, x_n]$$
n × d

Input = seq. of word embeddings

x_k : k-th word embedding
d-dim. vector

W_q, W_k, W_V : d × d, parameters,
independent for size of X

XW_q n × d

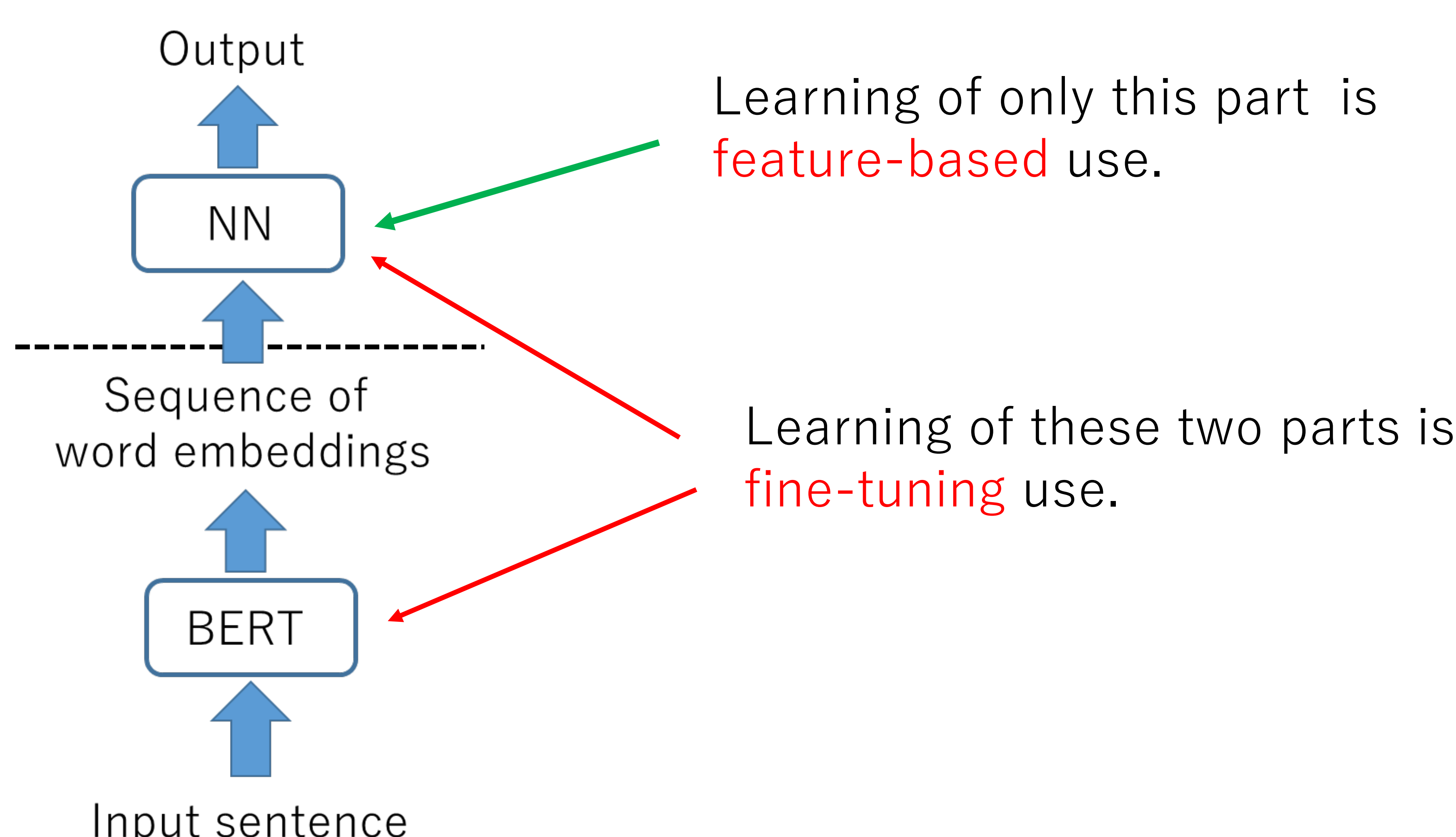
XW_k n × d

$\mathrm{softmax}(XW_q \bullet (XW_k)^T)$ n × n

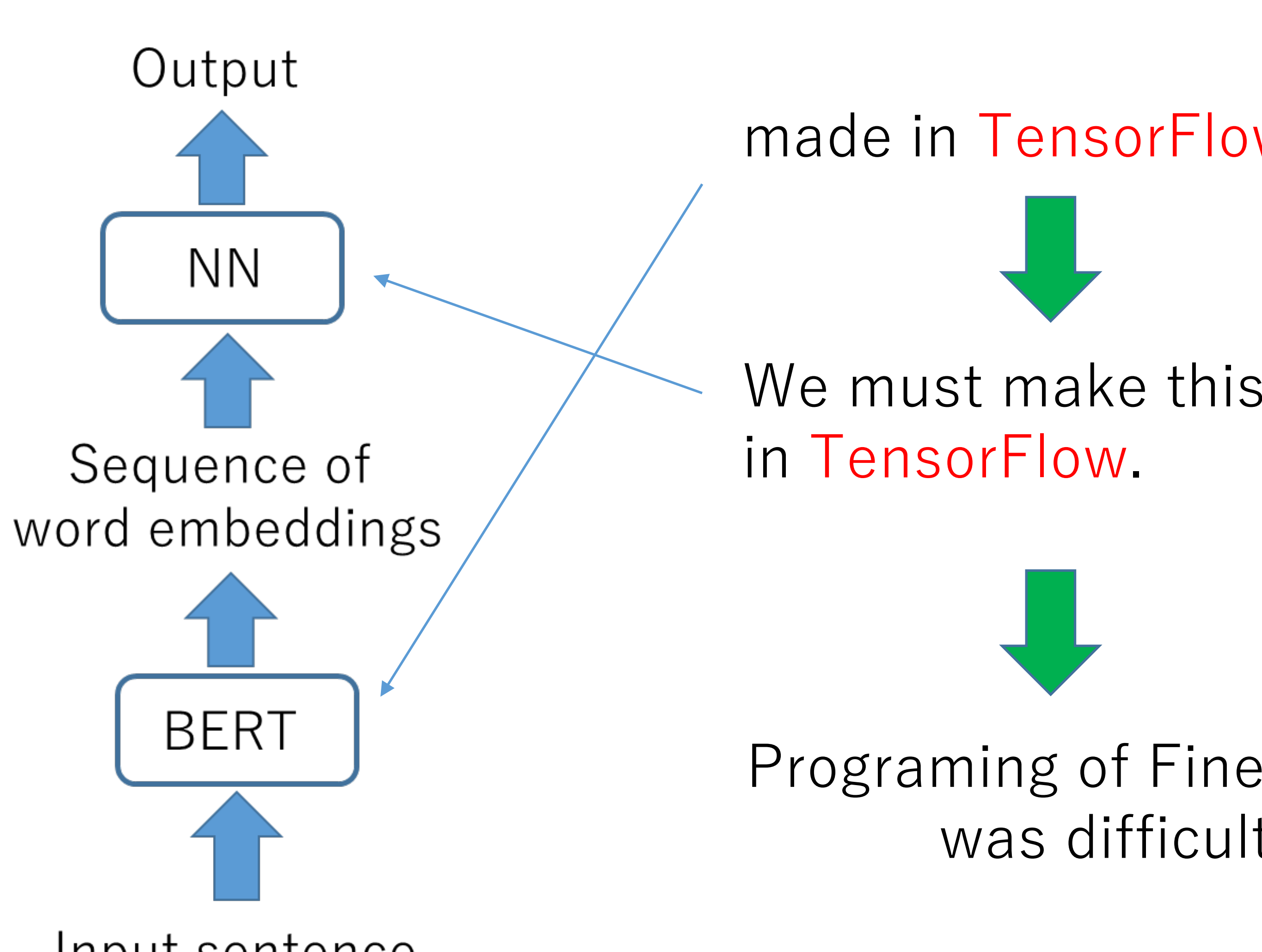
XW_V n × d

$\mathrm{softmax}(XW_q \bullet (XW_k)^T) XW_V$ n × d

feature-based and fine-tuning



Difficulty of Fine-tuning

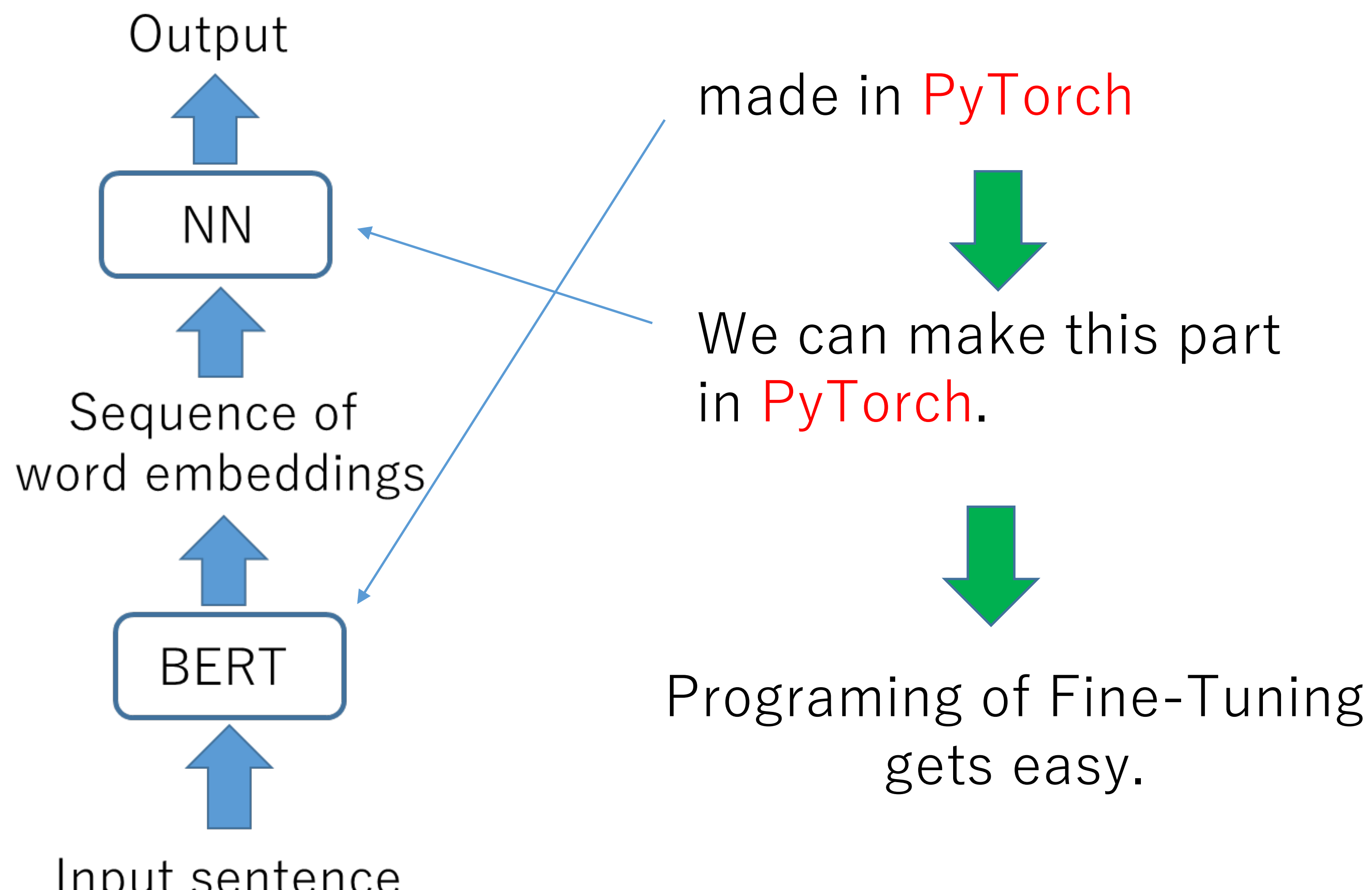


made in TensorFlow

We must make this part
in TensorFlow.

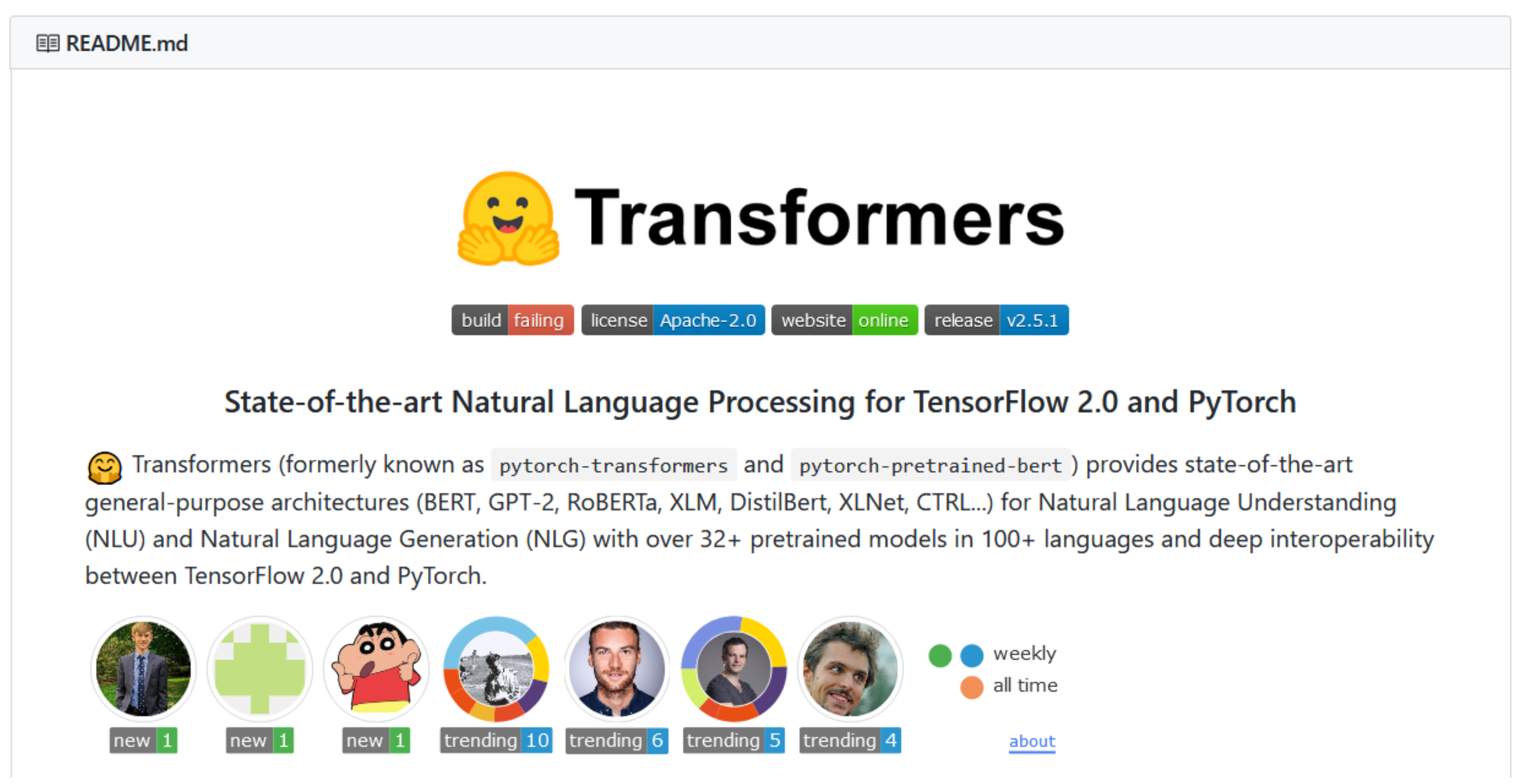
Programing of Fine-Tuning
was difficult.

Appearance of Hugging Face



- 2 -
Input/Output of BERT

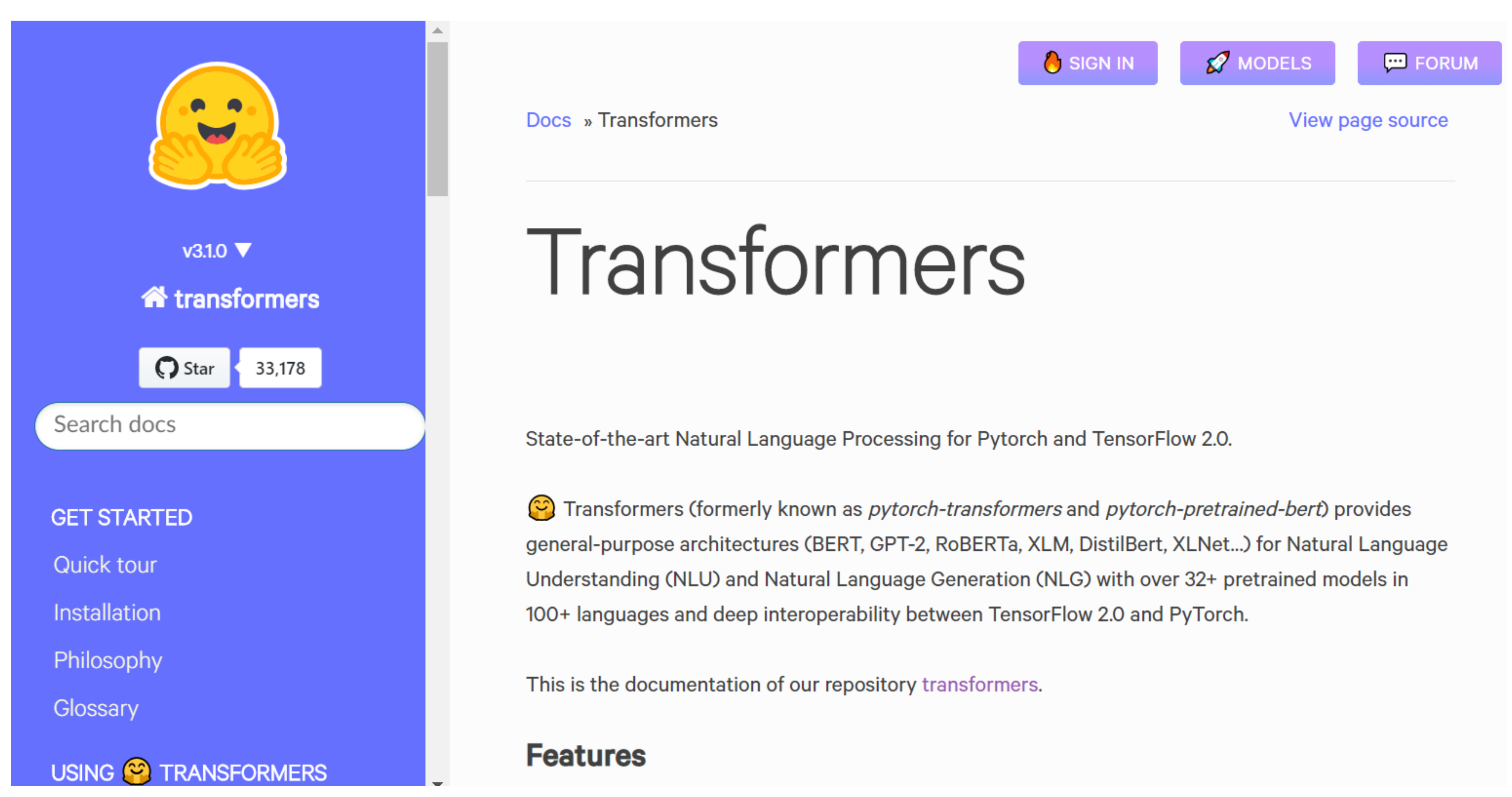
https://github.com/huggingface/transformers



We can download all codes of transformers from this site.

Documentation is in

https://huggingface.co/transformers/



Current version is 3.1.0

Four important classes on BERT

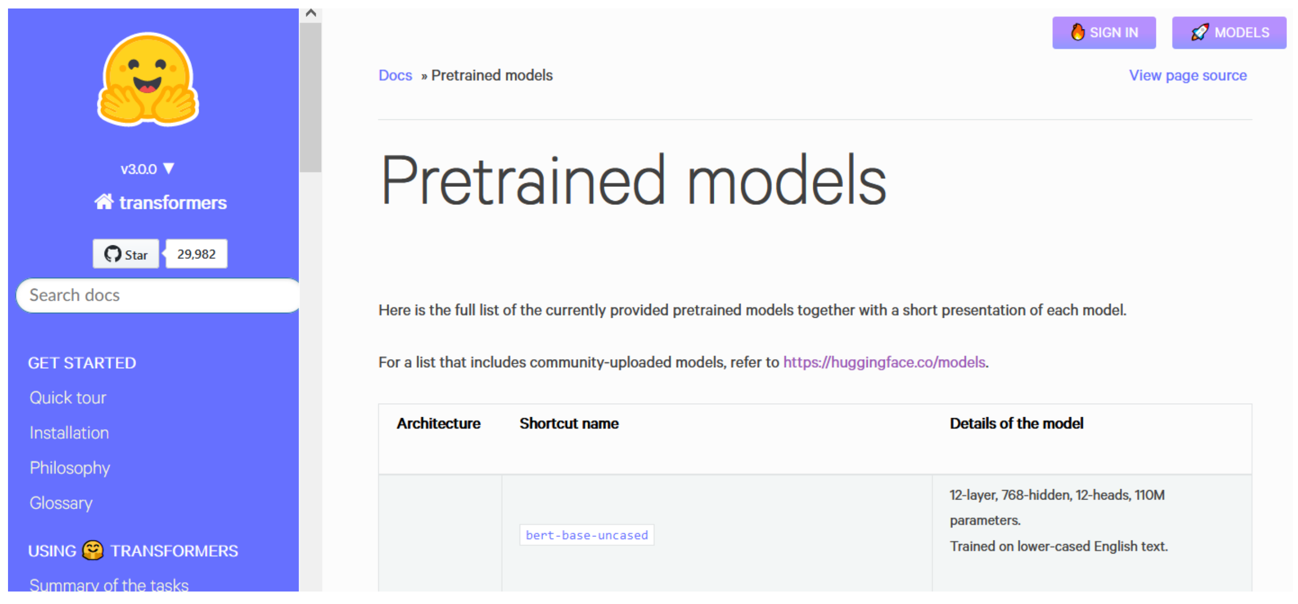
BertConfig
BertTokenizer
BertModel
BertForMaskedLM

If we only know these four classes,
we don't need the rest.

There are some classes for specific tasks.
However, it is more applicable to make it by yourself.
It is easy because we can use PyTorch.

Pretrained models

https://huggingface.co/transformers/pretrained_models.html



If the model is registered in above site,
we can use that model by its name

Required files to use BERT

(1) Model file

pytorch_model.bin

(2) Configuration file

config.json

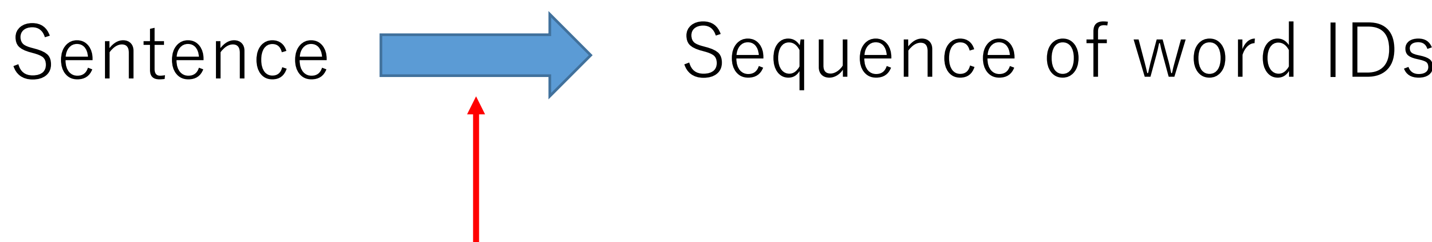
(3) Vocabulary file

vocab.txt

Standard file names

Tokenizer

Input of BERT is sequences of word IDs



Tokenizer does this transform

If you use Japanese BERT, it is better to choose the tokenizer used in learning BERT.

Not use of Tokenizer

Get word IDs from 'vocab.txt'

Attend to special tokens

[CLS] put the beginning of the sentence
[SEP] put the end of the sentence
[PAD] padding
[UNK] unknown word
[MASK] mask

dic.py

```
#!/usr/bin/python
# -*- coding: sjis -*-

text = ['[CLS]', '私', 'は', '犬', 'が', '好き', '。', '[SEP]']
ln = 0
dic = {}
with open('tohoku/vocab.txt','r',encoding="utf-8") as f:
    word = f.readline()
    while word:
        word = word.rstrip()
        dic[word] = ln
        ln += 1
        word = f.readline()

ids = [ dic[w] for w in text ]
print(ids)
```

```
$ python dic.py
[2, 1325, 9, 2928, 14, 3596, 8, 3]
```

Try BERT

```
from transformers import BertModel, BertConfig
import torch

config = BertConfig.from_json_file('config.json')
model = BertModel.from_pretrained('pytorch_model.bin',
                                                            config=config)

ids = [2, 1325, 9, 2928, 14, 3596, 8, 3]  # 私は犬が好き。

ids = torch.tensor(ids).unsqueeze(0)
a = model(ids)
```

Output of BERT

a = model(ids)

Output a is tuple. Size is various.

a[0] main output content

torchSize([batch size, # of words, dim. of word])

a[0][0] output of BERT for 0-th sentence

```
>>> a[0][0].shape
torch.Size([8, 768])    # 8 words, 768 dim.
```

Embedding of [CLS] in 0-th sentence

```
>>> a[0][0][0].shape
torch.Size([768])
```

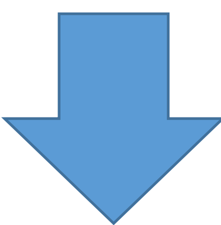
Use of BertJapaneseTokenizer

```
from transformers import BertJapaneseTokenizer

tknz = BertJapaneseTokenizer.from_pretrained(
                    'cl-tohoku/bert-base-japanese')

ids = tknz.encode('私は犬が好き。’)
print(ids)

#   [2, 1325, 9, 2928, 14, 3596, 8, 3]
```

We don’t need segment an input sentence
Into words.

Not use of model name

If the model is registered in Hugging Face site,
the tokenizer can be set by its model name.
If the model is not in Hugging Face site,
tokenizer of the model can be used as follows:



```
from transformers import BertJapaneseTokenizer
from transformers import tokenization_bert_japanese

tknz = BertJapaneseTokenizer('tohoku/vocab.txt',
                             do_lower_case=False,do_basic_tokenize=False)

tknz.word_tokenizer = tokenization_bert_japanese.MecabTokenizer()
```

Length of sentence

Limitation of length of sentence (number of words) is set in the 'config.json'.

"max_position_embeddings": 512

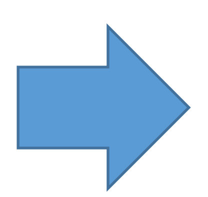
This number can be changed

If the input is over this limitation, errors come in putting it into the model.

Note on sentence length

When input is a single sentence,
[CLS] and [SEP] are added, that +2.
When input is a double sentences,
[CLS], [SEP] and [SEP] are added, that +3.

Sentence length is limited, but the tokenizer
is not suffered from this limitation.



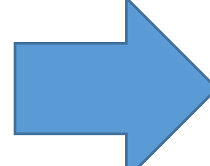
You must remove the overed part
in output of the tokenizer in yourself.
Auto remove may not be set in a system.

max_position_embeddings

This variable means the limitation
on sentence length.
This number can be changed.

No problem if we set a big number to this ?

No good!



・Speed gets slow.

・Need much memory.

・No sense if the number is bigger
than the number used in learning.

How do we get middle layer output ?

config.output_hidden_states=True

```
from transformers import BertModel, BertConfig
import torch

config = BertConfig.from_json_file('config.json')
model = BertModel.from_pretrained('pytorch_model.bin',
                                  config=config)

ids = [2, 1325, 9, 2928, 14, 3596, 8, 3] # 私は犬が好き。

ids = torch.tensor(ids).unsqueeze(0)
a = model(ids)
```

a = model(ids)

If output_hidden_states=True,
size of tuple a is 3.

a[2] is a tuple, the size is 13.
Each element of a[2] means each layer output.

a[2][-1] seq. of embeddings of 12th layer
a[2][-2] seq. of embeddings of 11th layer
・・・
a[2][-12] seq. of embeddings of first layer
a[2][-13] seq. of input embeddings of BERT

Note: a[0][0] == a[2][-1]

BertForMaskedLM

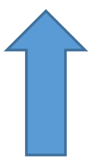
BERT can be used as Masked Language Model.

BertForMaskedLM can predict the [MASK] word.

```
>>> import torch
>>> from transformers import BertConfig, BertForMaskedLM
>>> from transformers import BertTokenizer
>>> config = BertConfig.from_json_file('config.json')
>>> model = BertForMaskedLM.from_pretrained('pytorch_model.bin',
                                              config=config)
>>> tknz = BertTokenizer('vocab.txt', do_lower_case=False,
                          do_basic_tokenize=False)
>>> ids = tknz.encode("私 は [MASK] が 好き 。")

>>>ids
[2, 1325, 9, 4, 14, 3596, 8, 3]
```

ID of [MASK] is 4, so ids[3] is [MASK]

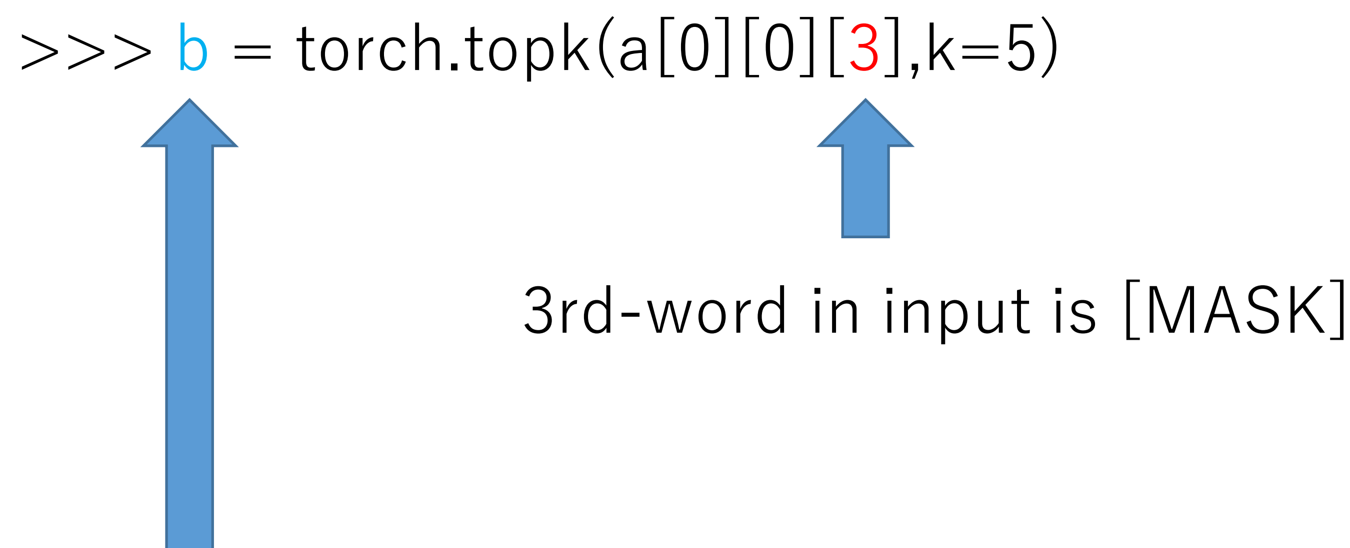
```
>>> ids = torch.tensor(ids).unsqueeze(0)
>>> a = model(ids)
```



a is a tuple, its size is 1.

```
>>> a[0].shape
torch.Size([1, 8, 32000])
```

a[0][0][k][m] means the probability that k-th word in input is m-th word in ‘vocab.txt’ whose size is 32000.



b is tuple, and its size is 2.
b[0] is a value, and b[1] is the index.

```
>>> ans = tknz.convert_ids_to_tokens(b[1])
>>> ans
['サッカー', '野球', '音楽', 'あなた', '映画']
```

（ soccer, baseball, music, you, movie ）

- 3 -
Use of BERT through transformers

Document classification

Data set: 「livedoor news corpus」

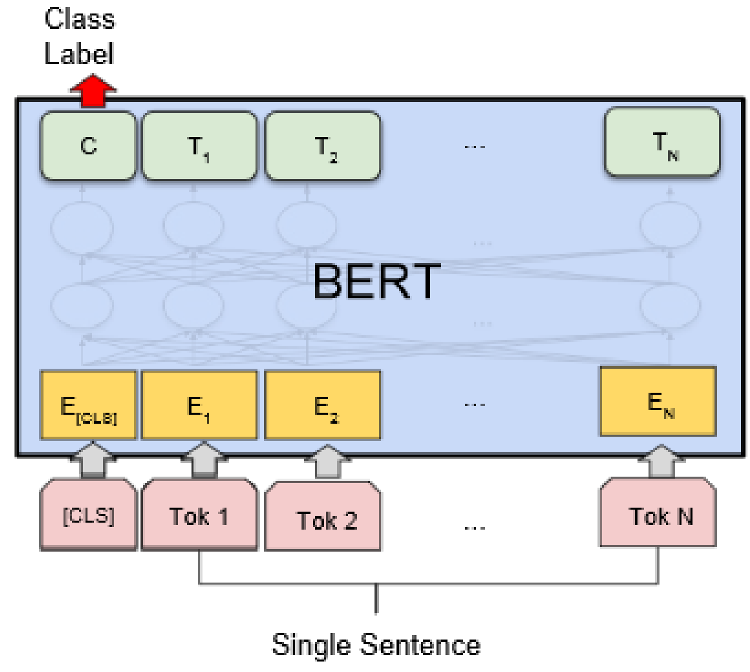
https://www.rondhuit.com/download.html#ldcc

9 category news articles

We use the headline (1 sentence) as data.

・pick up 100 training data and 100 test data from each category
・totally, 900 training data and 900 test
・learn the model by training data, and test the model by test data

Single sentence task



Learn the liner transfer W from the vector C of [CLS] to a class label. At the same time, BERT is fine-tuned.

$$\mathrm{softmax}(CW^T)$$

Model Definition

```
class DocCls(nn.Module):
    def __init__(self,bert):
        super(DocCls, self).__init__()
        self.bert = bert
        self.cls=nn.Linear(768,9)
    def forward(self,x):
        bout = self.bert(x)
        bs = len(bout[0])
        h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        h1 = self.cls(h0)
        return h1
```

Model setting

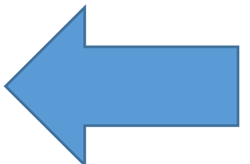
```
config = BertConfig.from_json_file('config.json')
bert = BertModel.from_pretrained('pytorch_model.bin',
                                          config=config)

model = DocCls(bert)
```

The rest is the same as always.

```
optimizer = optim.SGD(model.parameters(),lr=0.01)
criterion = nn.CrossEntropyLoss()
```

Fine-tuning

If pretrained BERT is included in the whole model,
fine-tuning is easy.
It is same as a regular learning program.



But, learning time is much.

If the task is simple like document classification,
feature-based or learning of only upper layers
of BERT is enough.

Switch to Feature-based

```
class DocCls(nn.Module):
    def __init__(self,bert):
        super(DocCls, self).__init__()
        self.bert = bert
        self.cls=nn.Linear(768,9)
    def forward(self,x):
        bout = self.bert(x)
        bs = len(bout[0])
        h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        h1 = self.cls(h0)
        return h1
```

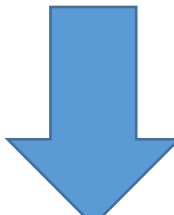
All you have to do is to freeze this part.

Tips of freeze

Freezing parameters are so many.

```
class DocCls(nn.Module):
    def __init__(self,bert):
        super(DocCls, self).__init__()
        self.bert = bert
        self.cls=nn.Linear(768,9)
    def forward(self,x):
        bout = self.bert(x)
        bs = len(bout[0])
        h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        h1 = self.cls(h0)
        return h1
```

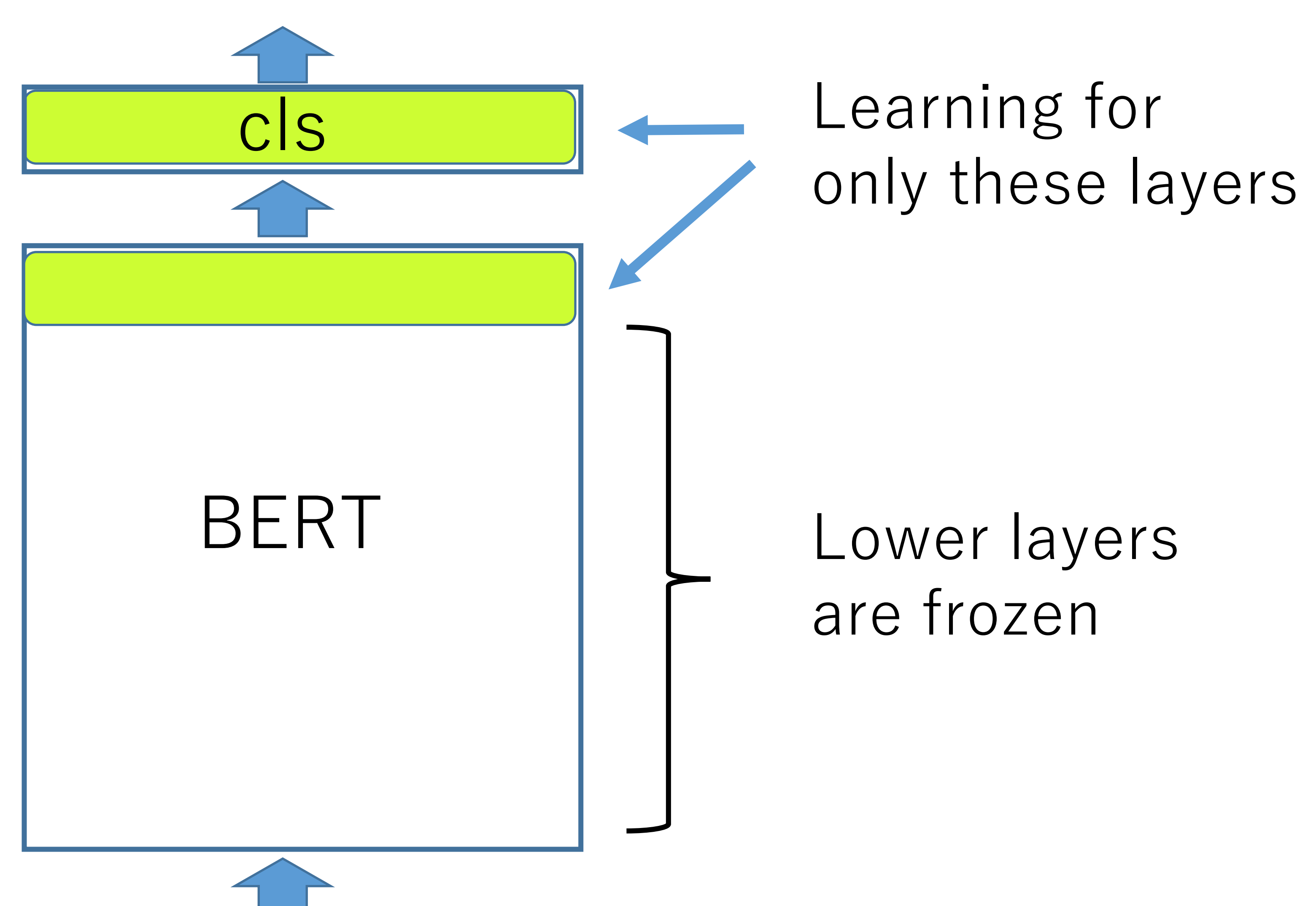
Learning parameters are a little.



First all parameters are frozen, and then only learning parameters are returned to active.

```python
#  all parameters are frozen

for name, param in model.named_parameters():
    param.requires_grad = False

#  only parameters in 'cls' are returned to active

for name, param in model.cls.named_parameters():
    param.requires_grad = True
```

Fine-tuning of only upper layers



Parameter names in BERT

We have to know parameter names of the model when only part of the model parameters is learned or frozen.
In PyTorch, we can confirm them by printing the model.

```
config = BertConfig.from_json_file('config.json')
bert = BertModel.from_pretrained('pytorch_model.bin’,
                                                  config=config)

model = DocCls(bert)
print(model)
```

```
DocCls(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(32000, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
   (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer( ・・・)
         ・・・
        (11): BertLayer( ・・・)
   (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (cls): Linear(in_features=768, out_features=9, bias=True)
)
```

This is 12 layers of multi-head attentions.

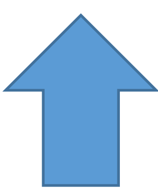
0-th multi-head Attention

```
(encoder): BertEncoder(
  (layer): ModuleList(
    (0): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
```

fine-tuning of only upper layers

Parameter names are important.
All parameters are frozen, and parameters of only upper layers are returned to active.

```
for na, pa in model.bert.encoder.layer[-1].named_parameters():
    pa.requires_grad = True
```



Multi-head attention in BERT
-1 means the top layer
-2 means the one layer below the top layer

Parameters given to optimized function

Easy if they are all parameters of the model.

```
optimizer = optim.SGD(model.parameters(),lr=0.01)
```

As follow if they are only part of parameters

```
optimizer = optim.SGD([
    {'params':model.bert.encoder.layer[-1].parameters(),
            'lr':0.001},
    {'params':model.cls.parameters(),
            'lr’:0.01}])
```

Following is a sample program which
top 2 layers of BERT and the classification layer
are learned.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import numpy as np

from transformers import BertModel, BertConfig

import pickle

config = BertConfig.from_json_file('../tohoku/config.json')
bert = BertModel.from_pretrained('../tohoku/pytorch_model.bin',config=config)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
# Data setting

data = pickle.load(open('train.pkl','rb'))

cls = []
sens = []
for i in range(len(data)):
    cls.append(data[i][0])
    dt = data[i][1]
    while len(dt) < 50:
        dt.append(0.0)
    dt = torch.Tensor(dt)
    sens.append(dt)

xs = torch.stack(sens,dim=0).type(torch.long).to(device)
ys = torch.LongTensor(cls).to(device)
```

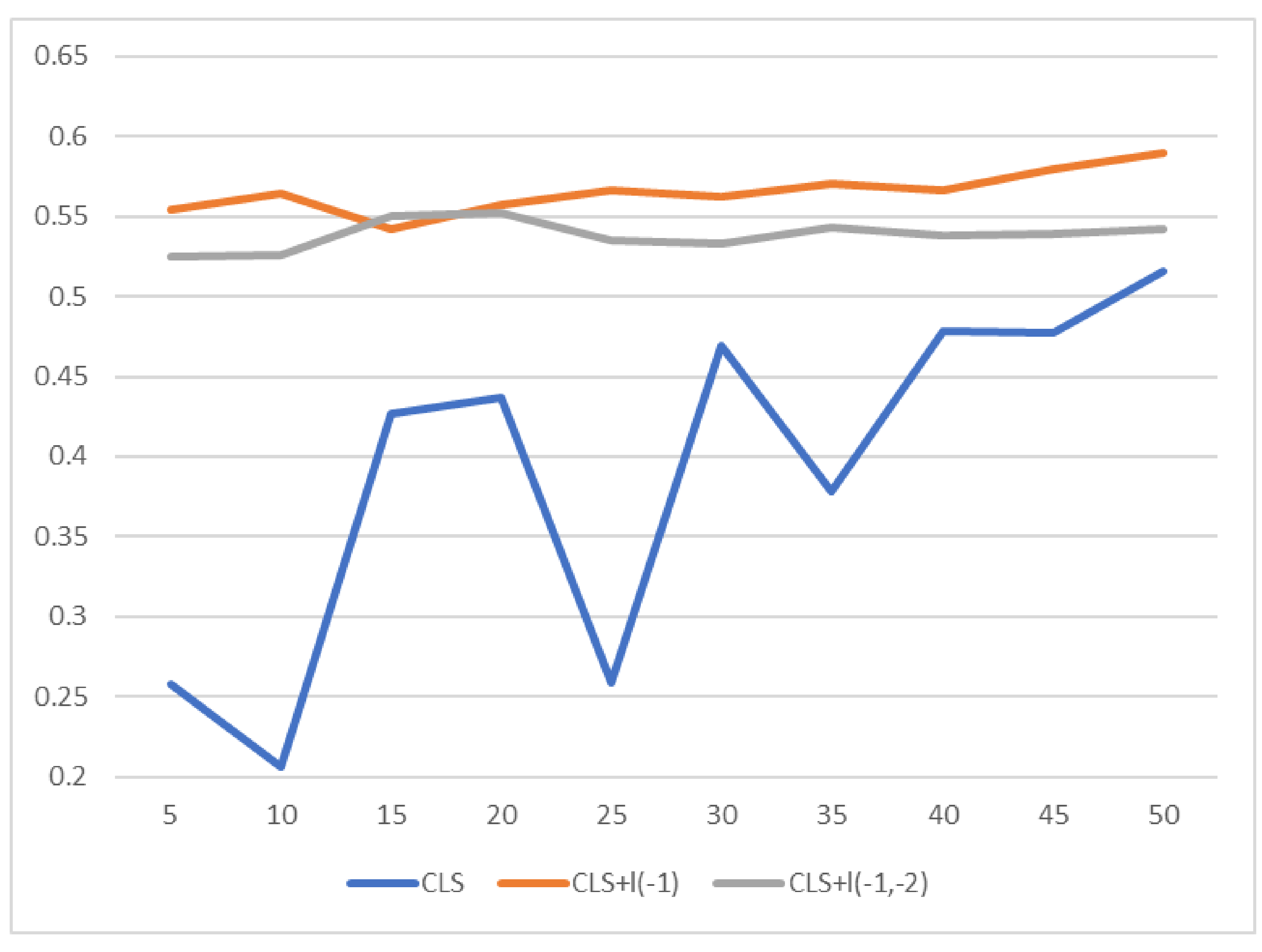
For batch processing, length of a sentence is fixed to 50 by padding.

```
# Define model

class DocCls(nn.Module):
    def __init__(self,bert):
        super(DocCls, self).__init__()
        self.bert = bert
        self.cls=nn.Linear(768,9)
    def forward(self,x):
        bout = self.bert(x)
        bs = len(bout[0])
        h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        h1 = self.cls(h0)
        return h1

model = DocCls(bert)
model.to(device)
```

```
for name, param in model.named_parameters():
    param.requires_grad = False

for name, param in model.cls.named_parameters():
    param.requires_grad = True

for name, param in model.bert.encoder.layer[-1].named_parameters():
    param.requires_grad = True

for name, param in model.bert.encoder.layer[-2].named_parameters():
    param.requires_grad = True

# optimizer = optim.SGD(model.parameters(),lr=0.1)

optimizer = optim.SGD([
    {'params':model.bert.encoder.layer[-2].parameters(), 'lr':0.0005},
    {'params':model.bert.encoder.layer[-1].parameters(), 'lr':0.001},
    {'params':model.cls.parameters(), 'lr':0.1}])

criterion = nn.CrossEntropyLoss()
```

```
# Learn

n = len(data)
bs = 10

for ep in range(1,51):
    idx = np.random.permutation(n)
    for j in range(0,n,bs):
        xtm = xs[idx[j:(j+bs) if (j+bs) < n else n]]
        ytm = ys[idx[j:(j+bs) if (j+bs) < n else n]]
        output = model(xtm)
        loss = criterion(output,ytm)
        print(ep, j, loss.item())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    if (ep % 5 == 0):
        outfile = "dcls2-" + str(ep) + ".model"
        torch.save(model.state_dict(),outfile)
        print(outfile," saved")
```

Result of the experiment



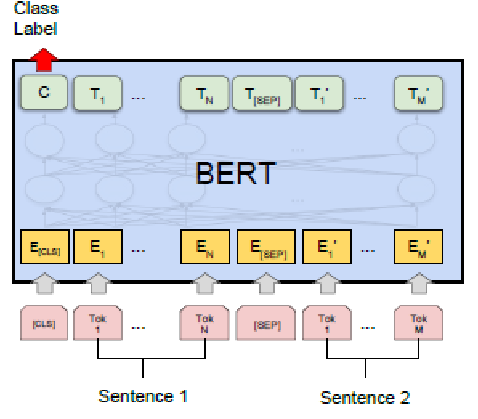
BertForSequenceClassification

The model generated from this class
is added one Linear layer connected to
CLS output of BERT.

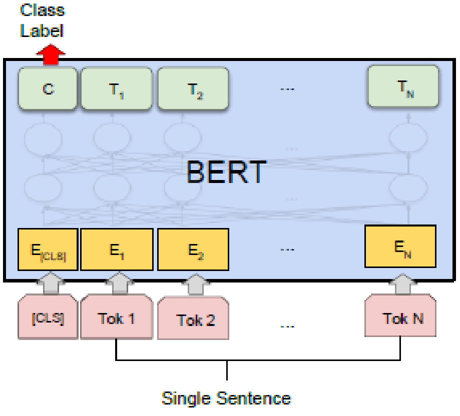
・class number is variable
・regression is also available
・there are some pretrained models
 for famous tasks.

This is useful for simple classification tasks.

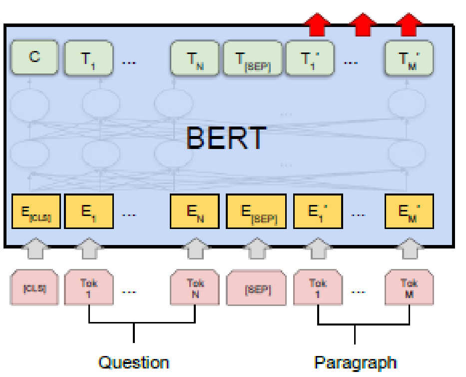
4 ways of use of BERT



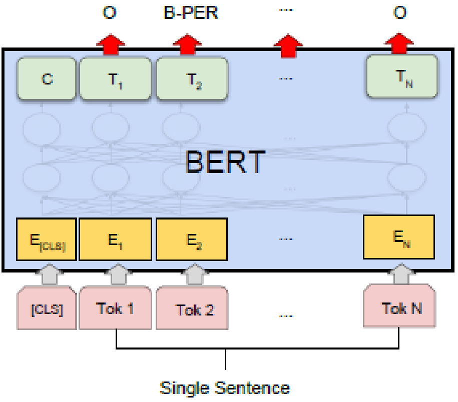
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



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



(c) Question Answering Tasks:
SQuAD v1.1



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

Examples in Hugging Face

https://huggingface.co/transformers/examples.html

The Big Table of Tasks

Task	Example datasets	Trainer support	TFTrainer support	pytorch-lightning	Colab
`language-modeling`	Raw text	✅	-	-	Open in Colab
`text-classification`	GLUE, XNLI	✅	✅	✅	Open in Colab
`token-classification`	CoNLL NER	✅	✅	✅	-
`multiple-choice`	SWAG, RACE, ARC	✅	✅	-	Open in Colab
`question-answering`	SQuAD	✅	✅	-	-
`text-generation`	-	n/a	n/a	n/a	Open in Colab
`distillation`	All	-	-	-	-

Learning for GLUE

We can use run_glue.py in examples of transformers.

(Note) It is not so difficult to make it yourself.

For some tasks, it takes much time.

```
#!/bin/bash

export GLUE_DIR=./glue_data
export TASK_NAME=MNLI

python3 run_glue.py ¥
  --model_type bert ¥
  --model_name_or_path model ¥
  --task_name $TASK_NAME ¥
  --do_train ¥
  --do_eval ¥
  --data_dir $GLUE_DIR/$TASK_NAME ¥
  --max_seq_length 128 ¥
  --learning_rate 3e-5 ¥
  --num_train_epochs 3.0 ¥
  --output_dir ./output/$TASK_NAME ¥
  --overwrite_output_dir ¥
  --logging_steps 50 ¥
  --save_steps 200
```

Name of the task

Use of pretrained models for tasks

There are some pretrained models for famous tasks like GLUE.

https://huggingface.co/models

↥ Back to home

All Models and checkpoints

Also check out our list of **Community contributors** 🏆 and **Organizations** 🌍.

Search models... | Tags: All ▾ | Sort: Most downloads ▾

bert-base-multilingual-cased ★

jplu/tf-xlm-roberta-base ★

All Models and checkpoints

Search by "MNLI"

Also check out our list of **Community contributors** 🏆 and **Organizations** 🌍.

mnli

Tags: All ▾

Sort: Default ▾

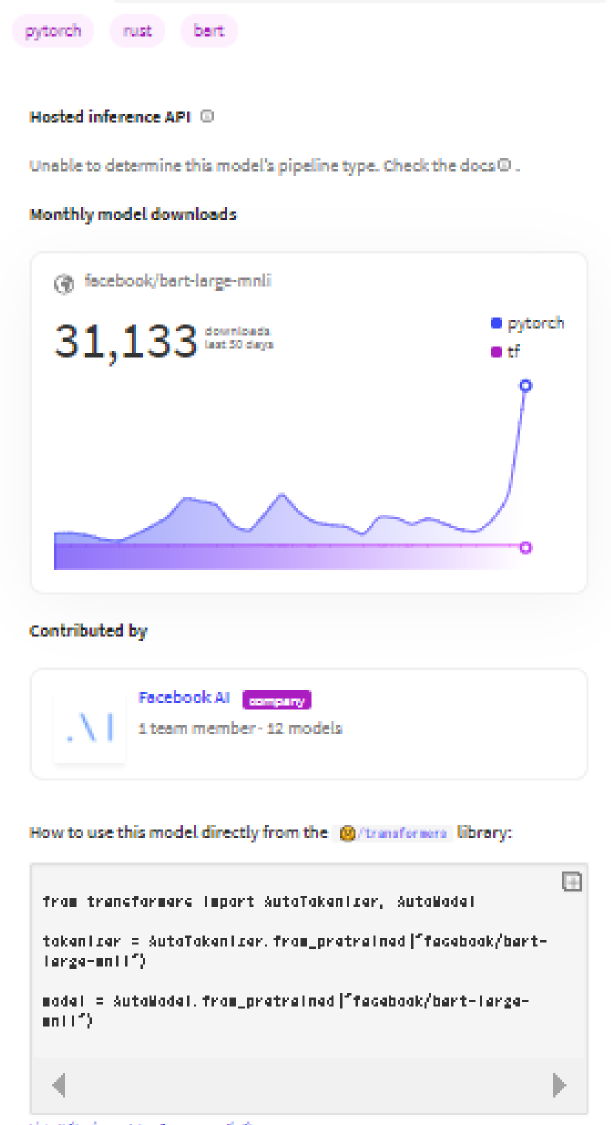
canwenxu/BERT-of-Theseus-MNLI ★

facebook/bart-large-mnli

prajjwal1/albert-base-v1-mnli

roberta-large-mnli

It looks useable.



Here, a code to use this model is shown.

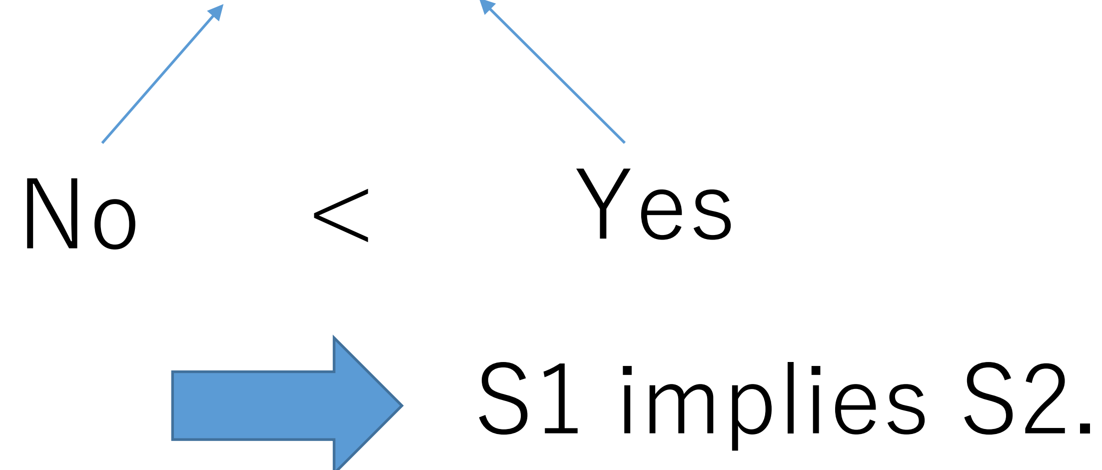
But, sometimes you need to change it.

My example code for MNLI.

```
from transformers import BertConfig, BertTokenizer, ¥
                                    BertForSequenceClassification

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained(
                   "facebook/bart-large-mnli",num_labels=2)

s1 = “The extent of ・・・ on accessing the funds."
s2 = "Many people would be ・・・ over their own money."

encoding = tokenizer.encode_plus(s1, s2)
input_ids, token_type_ids = encoding["input_ids"],¥
                                       encoding[“token_type_ids”]

out = model(torch.tensor([input_ids]),
                  token_type_ids=torch.tensor([token_type_ids]))
```

```
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
>>> s1 = "The extent of the behavioral effects would depend in part on the structure of the individual account program and any limits on accessing the funds."
>>> s2 = "Many people would be very unhappy to loose control over their own money."
>>> encoding = tokenizer.encode_plus(s1, s2)
>>> input_ids, token_type_ids = encoding["input_ids"], encoding["token_type_ids"]
>>> out = model(torch.tensor([input_ids]), token_type_ids=torch.tensor([token_type_ids]))
>>> out
(tensor([[0.0715, 0.1829]], grad_fn=<AddmmBackward>),)
>>> 
[--] S¥**-  *Python*      Bot L??    (Inferior Python:run Shell-Compile) 3:34午後 0.49
```

(tensor([[0.0715, 0.1829]], grad_fn=<AddmmBackward>),)



Learning of SQuAD

We can use run_squad.py in examples of transformers.

(Note) It may be difficult to make it yourself.

Where is it?

https://github.com/huggingface/transformers/tree/master/examples/question-answering

README.md

SQuAD

Based on the script `run_squad.py`.

Fine-tuning BERT on SQuAD1.0

This example code fine-tunes BERT on the SQuAD1.0 dataset. It runs in 24 min (with BERT-base) or 68 min (with BERT-large) on a single tesla V100 16GB. The data for SQuAD can be downloaded with the following links and should be saved in a $SQUAD_DIR directory.

- train-v1.1.json
- dev-v1.1.json
- evaluate-v1.1.py

And for SQuAD2.0, you need to download:

- train-v2.0.json
- dev-v2.0.json

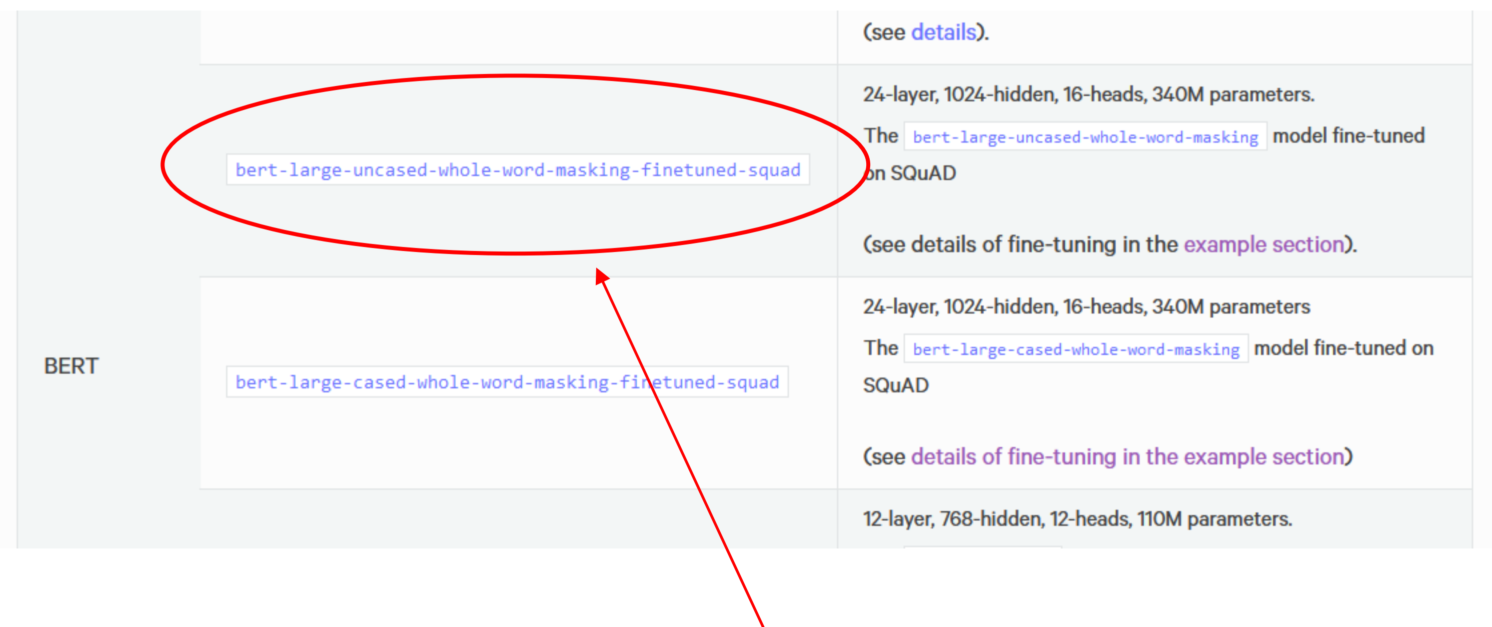
```
#!/bin/bash

export SQUAD_DIR=/path/to/SQUAD

python3 run_squad.py ¥
  --model_type bert ¥
  --model_name_or_path bert-base-uncased ¥
  --do_train ¥
  --do_eval ¥
  --train_file $SQUAD_DIR/train-v1.1.json ¥
  --predict_file $SQUAD_DIR/dev-v1.1.json ¥
  --per_gpu_train_batch_size 12 ¥
  --learning_rate 3e-5 ¥
  --num_train_epochs 2.0 ¥
  --max_seq_length 384 ¥
  --doc_stride 128 ¥
  --output_dir /tmp/debug_squad/
```

Change it to your path.

Auto download if you don't have.

Use of pretrained model for SQuAD

https://huggingface.co/transformers/pretrained_models.html



bert-large-uncased-whole-word-masking-finetuned-squad

We can use the model by this name.

https://www.dogonews.com/

Memorial Day Celebrations Get Innovative Amid COVID-19 Pandemic

Memorial Day celebrations usually involve parades, flag ceremonies, and other formal public recognitions to honor the brave men and women of the American Armed Forces who have sacrificed their lives in the line of duty. This includes those in the US Army, Navy, Marine Corps, National Guard, Air Force, and the Coast Guard. However, the COVID-19 pandemic social distancing requirement is causing American cities and towns to cancel the beloved traditions and find new ways to honor their fallen heroes. Here are a few innovative festivities planned for the holiday, which will be observed on May 25, 2020.

(Q) What caused celebrations to be cancelled?

```
from transformers import BertConfig, BertTokenizer, ¥
                                   BertForQuestionAnswering
import torch

##  following two models are automatically downloaded

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForQuestionAnswering.from_pretrained(
          'bert-large-uncased-whole-word-masking-finetuned-squad’)

##  give question and text

question =  "What caused celebrations to be cancelled?"
text = “Memorial Day celebrations ・・・on May 25, 2020"
```

```
encoding = tokenizer.encode_plus(question, text)

input_ids, token_type_ids = encoding["input_ids"], ¥
                            encoding["token_type_ids"]

# score of start position and end position of the span

start_scores, end_scores = model(torch.tensor([input_ids]),¥
                         token_type_ids=torch.tensor([token_type_ids]))


all_tokens = tokenizer.convert_ids_to_tokens(input_ids)

answer = ' '.join(all_tokens[torch.argmax(start_scores) : ¥
                            torch.argmax(end_scores)+1])

print(answer)
```

```
>>> text = "Memorial Day celebrations usually involve parades, flag ceremonies, and other formal public recognitions to honor the brave men and women of the American Armed Forces who have sacrificed their lives in the line of duty. This includes those in the US Army, Navy, Marine Corps, National Guard, Air Force, and the Coast Guard. However, the COVID-19 pandemic social distancing requirement is causing American cities and towns to cancel the beloved traditions and find new ways to honor their fallen heroes. Here are a few innovative festivities planned for the holiday, which will be observed on May 25, 2020"
>>> encoding = tokenizer.encode_plus(question, text)
>>> input_ids, token_type_ids = encoding["input_ids"], encoding["token_type_ids"]
>>> start_scores, end_scores = model(torch.tensor([input_ids]), token_type_ids=torch.tensor([token_type_ids]))
>>> all_tokens = tokenizer.convert_ids_to_tokens(input_ids)
>>> answer = ' '.join(all_tokens[torch.argmax(start_scores) : torch.argmax(end_scores)+1])
>>> answer
'co ##vid - 19 pan ##de ##mic social di ##stan ##cing requirement'
>>>
[--] S¥**-  *Python*      B    L??     (Inferior Python:run Shell-Compile) 3:36午後 0.39
```

answer

'co ##vid - 19 pan ##de ##mic social di ##stan ##cing requirement'

For (Q) What caused celebrations to be cancelled?

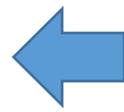
- 4 -
Downsizing of BERT model

Problems of BERT

Some problems of BERT are pointed out.

Following are my interest.

(1) Size of the model



I talk in this section.

(2) Limitation of input length

(3) Domain adaptation of BERT

・・・

I talk about the use of transformers for (1) problem

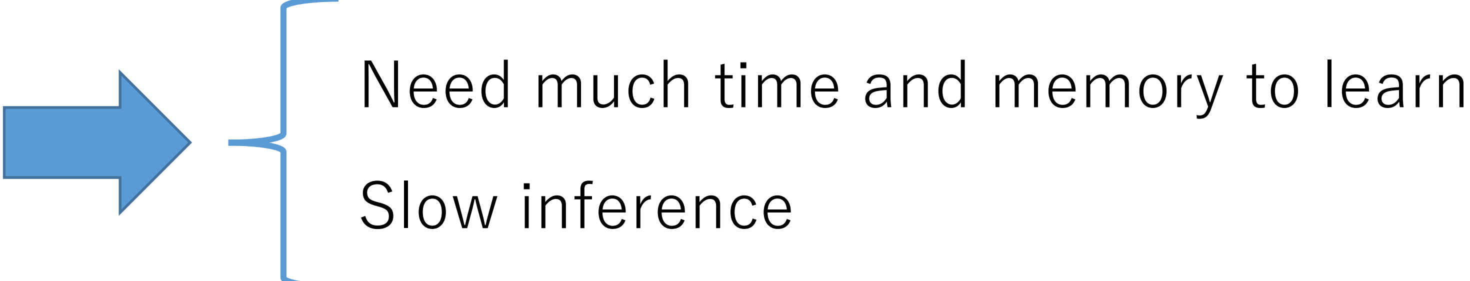
Problem of size of BERT

BERT is so powerful, but the size is too big.

BERT-base 110 M parameters

BERT-large 340 M parameters

cf) GPT-3 17.5 G parameters



Downsizing of BERT

（１） Quantization

Q8BERT

（２） Distillation

DistilBERT, MobileBERT, TinyBERT

（３） Pruning

Poor Man's BERT

（４） Others

ALBERT

Q8BERT

Computer Science > Computation and Language

[Submitted on 14 Oct 2019 (v1), last revised 17 Oct 2019 (this version, v2)]

Q8BERT: Quantized 8Bit BERT

Ofir Zafrir, Guy Boudoukh, Peter Izsak, Moshe Wasserblat

Recently, pre-trained Transformer based language models such as BERT and GPT, have shown great improvement in many Natural Language Processing (NLP) tasks. However, these models contain a large amount of parameters. The emergence of even larger and more accurate models such as GPT2 and Megatron, suggest a trend of large pre-trained Transformer models. However, using these large models in production environments is a complex task requiring a large amount of compute, memory and power resources. In this work we show how to perform quantization-aware training during the fine-tuning phase of BERT in order to compress BERT by $4\times$ with minimal accuracy loss. Furthermore, the produced quantized model can accelerate inference speed if it is optimized for 8bit Integer supporting hardware.

Comments: 5 Pages, Accepted at the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS 2019

Subjects: **Computation and Language (cs.CL)**; Machine Learning (cs.LG)

Cite as: arXiv:1910.06188 **[cs.CL]**

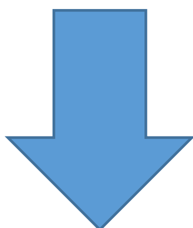
(or arXiv:1910.06188v2 **[cs.CL]** for this version)

https://arxiv.org/abs/1910.06188

Abstract of Q8BERT

All GEMM (General Matrix Multiply) operations in BERT Fully Connected (FC) and Embedding layers are quantized to 8bit.

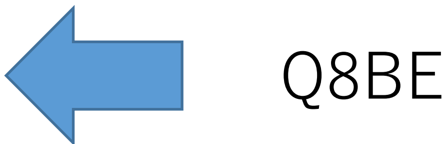
Operations required high accuracy (like Softmax or Normalization) are remained 32bit.



Keep 99% performance of 32bit BERT.
Reduce used memory to 25%.

QAT (Quantized-Aware Training)

When fine-tune BERT, QAT is used

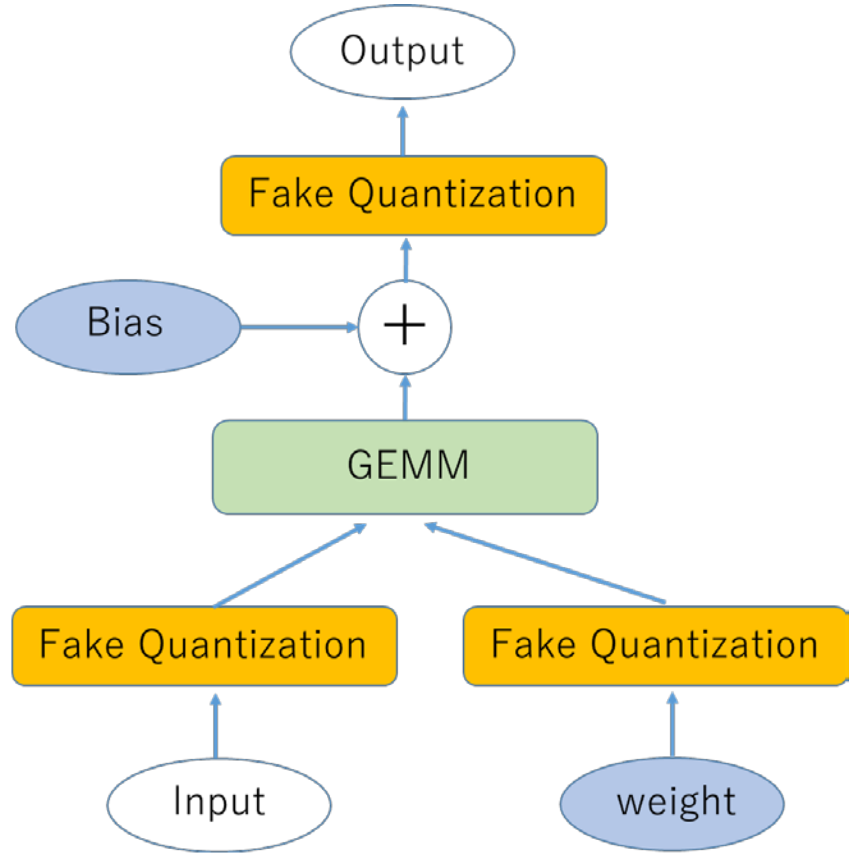


Q8BERT

QAT： learning method based on the use of quantization when inference

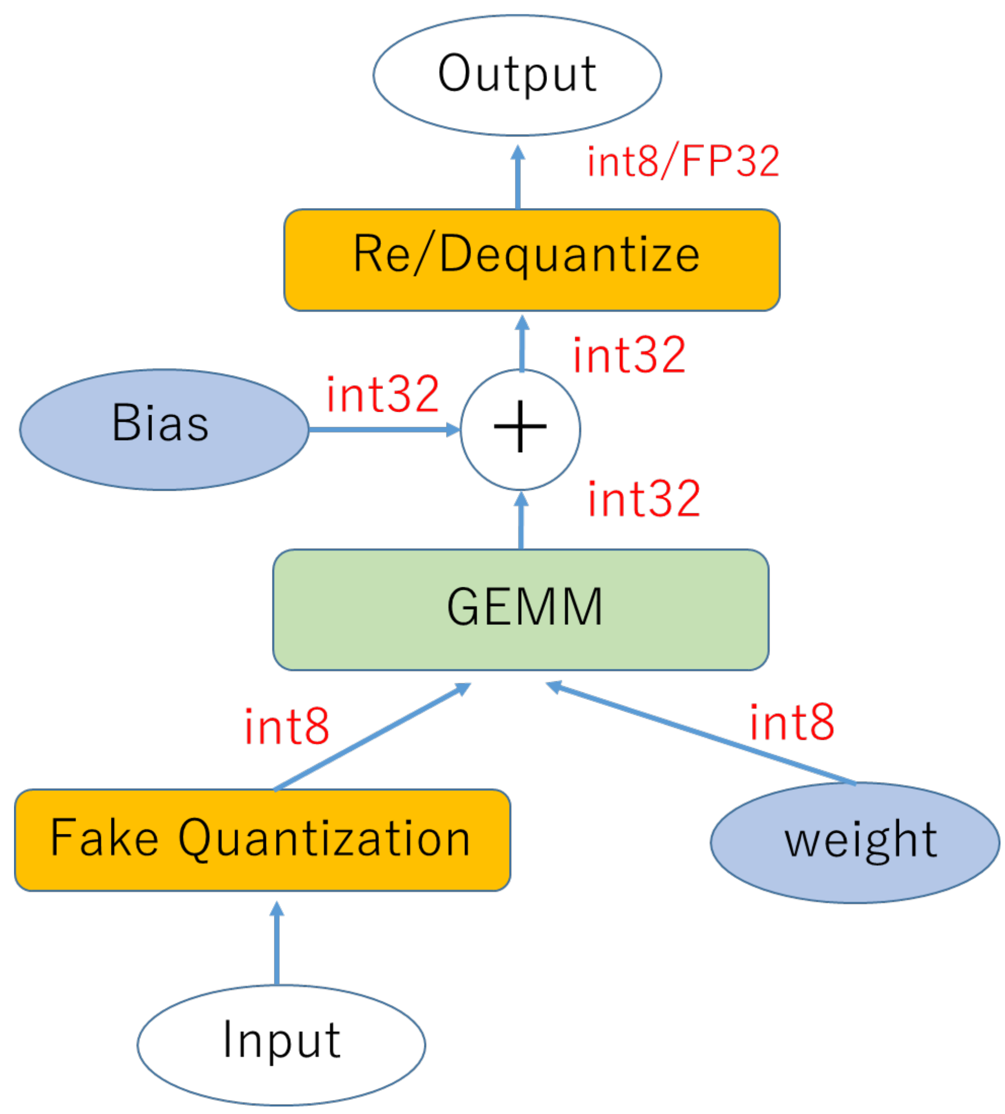
B. Jacob, et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference", In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2704–2713, 2018.

Fake Quantization (when fine-tuning)



When forward, quantized value are used.
When back propagation, original value (not quantized value) are used.

Fake Quantization (when inference)



Result of experiments

Table 1: GLUE tasks and SQuAD results. Each score is evaluated on the publicly available development set for the task using the metric specified for each task. For each task we present the score of a baseline (FP32) model, of a Quantization-Aware Training (QAT) model quantized to 8bit, and of a Dynamically Quantized (DQ) to 8bit model. Large means those tasks were trained with BERT-Large architecture.

Dataset	Metric	BERT baseline accuracy (STD)	QAT BERT 8bit (STD)	DQ BERT 8bit (STD)
CoLA	Matthew's corr.	**58.48** (1.54)	**58.48** (1.32)	56.74 (0.61)
MRPC	F1	**90** (0.23)	89.56 (0.18)	87.88 (2.03)
MRPC-Large	F1	90.86 (0.55)	**90.9** (0.29)	88.18 (2.19)
QNLI	Accuracy	90.3 (0.44)	**90.62** (0.29)	89.34 (0.61)
QNLI-Large	Accuracy	91.66 (0.15)	**91.74** (0.36)	88.38 (2.22)
QQP	F1	87.84 (0.19)	**87.96** (0.35)	84.98 (0.97)
RTE	Accuracy	**69.7** (1.5)	68.78 (3.52)	63.32 (4.58)
SST-2	Accuracy	**92.36** (0.59)	92.24 (0.27)	91.04 (0.43)
STS-B	Pearson corr.	**89.62** (0.31)	89.04 (0.17)	87.66 (0.41)
STS-B-Large	Pearson corr.	**90.34** (0.21)	90.12 (0.13)	83.04 (5.71)
SQuADv1.1	F1	**88.46** (0.15)	87.74 (0.15)	80.02 (2.38)

Tools

Additional Models

OPTIMIZED MODELS

⊟ Quantized BERT

Overview

Quantization Aware Training

Results

⊞ Running Modalities

References

Transformers Distillation

Sparse Neural Machine Translation

SOLUTIONS

Aspect Based Sentiment Analysis

Set Expansion

Docs » Quantize BERT with Quantization Aware Training

Quantize BERT with Quantization Aware Training

Overview

BERT - Bidirectional Encoder Representations from Transformers, is a language representation model introduced last year by Devlin et al [1] . It was shown that by fine-tuning a pre-trained BERT model it is possible to achieve state-of-the-art performance on a wide variety of Natural Language Processing (NLP) applications.

http://nlp_architect.nervanasys.com/quantized_bert.html

Training

To train Quantized BERT use the following code snippet:

```
nlp-train transformer_glue \
    --task_name mrpc \
    --model_name_or_path bert-base-uncased \
    --model_type quant_bert \
    --learning_rate 2e-5 \
    --output_dir /tmp/mrpc-8bit \
    --evaluate_during_training \
    --data_dir /path/to/MRPC \
    --do_lower_case
```

The model is saved at the end of training in 2 files:

1. A model saved in FP32 for further `pytorch_model.bin`
2. A quantized model for inference only `quant_pytorch_model.bin`

3 Implementation

Our goal is to quantize all the Embedding and FC layers in BERT to Int8 using the method described in Section 2. For this purpose we implemented quantized versions of Embedding and FC layers. During training, the Embedding layer returns fake quantized embedding vectors, and the quantized FC performs GEMM between the fake quantized input and the fake quantized weight, and then accumulates the products to the bias which is untouched since the bias will be later quantized to Int32. During inference, the quantized Embedding layer returns Int8 embedding vectors, and the quantized FC performs GEMM between Int8 inputs accumulated to the Int32 bias which is quantized using the weights' and activations' scaling-factors as described in [5]. Although the bias vectors are quantized to Int32 values, they only make up for a fraction of the amount of parameters in the model.

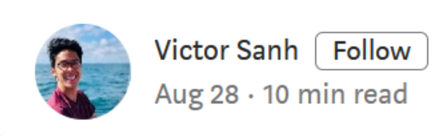
Our implementation of Quantized BERT is based on the BERT implementation provided by the `PyTorch-Transformers`[3] library. To implement quantized BERT we replaced all the Embedding and FC layers in BERT to the quantized Embedding and FC layers we had implemented. Operations that require higher precision, such as Softmax, Layer Normalization and GELU, are kept in FP32.

DistilBERT

https://medium.com/huggingface/distilbert-8cf3380435b5

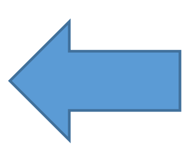
Author of the paper

Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT

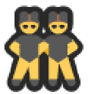


Distilation method

Standard Hinton's method



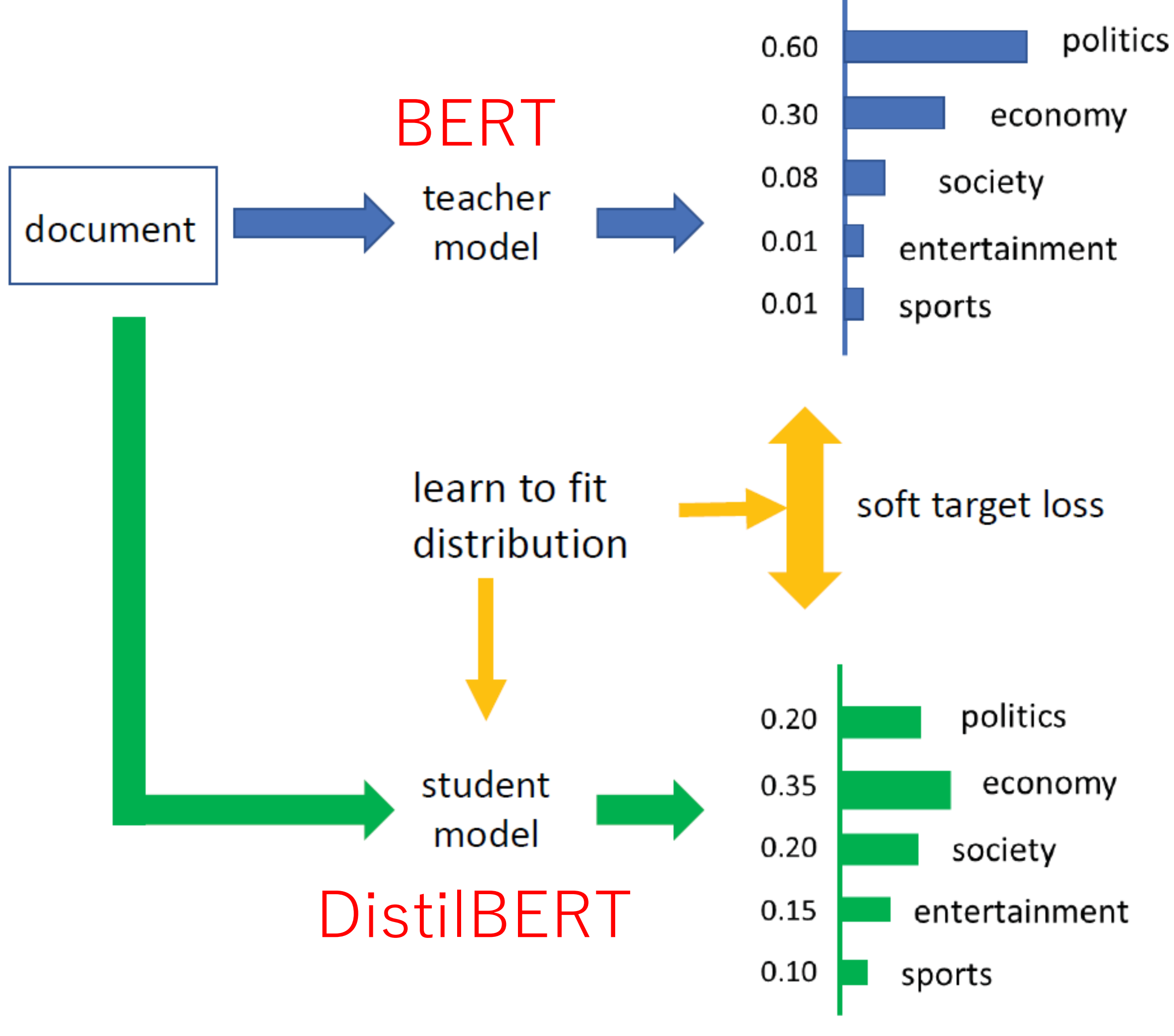
Use of Dark Knowledge.
Optimize the softmax distribution with temperature parameters by KL loss.

 How can we copy this dark knowledge?

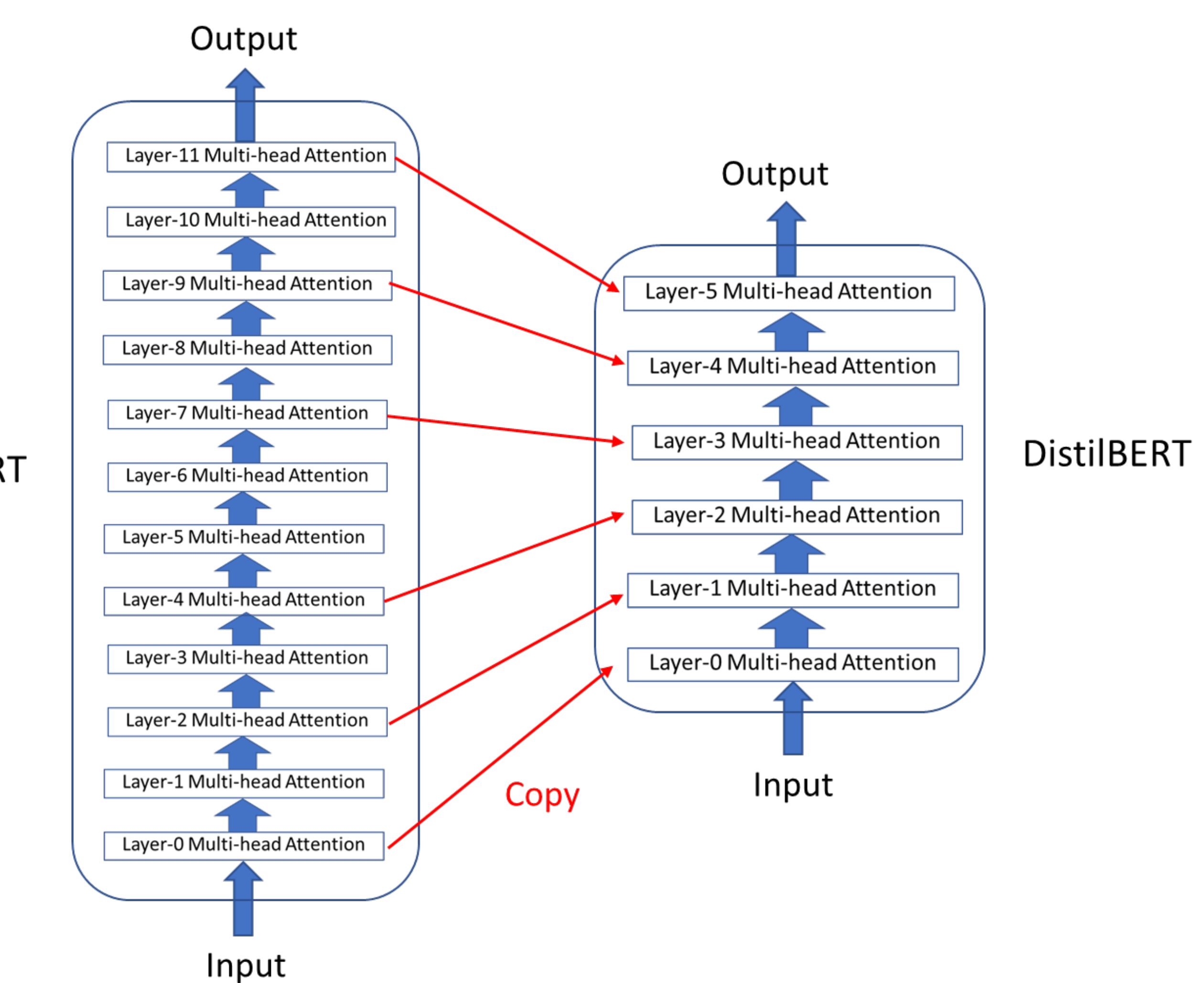
In the **teacher-student training,** we train a student network to mimic the **full output distribution** of the teacher network (its knowledge).

$$KL(p||q) = \mathbb{E}_p(log(\frac{p}{q})) = \sum_i p_i * log(p_i) - \sum_i p_i * log(q_i)$$

Hinton's method



Initial model



Result of experiments（1）

	Macro Score	CoLA	MNLI	MNLI-MM	MRPC		QNLI	QQP		RTE	SST-2	STS-B		WNLI
		mcc	acc	acc	acc	f1	acc	acc	f1	acc	acc	pearson	spearmanr	acc
GLUE BASELINE (ELMo + BiLSTMs)	**68.7**	44.1	68.6 (avg)		70.8	82.3	71.1	88.0	84.3	53.4	91.5	70.3	70.5	56.3
BERT base	**78.0**	55.8	83.7	84.1	86.3	90.5	91.1	90.9	87.7	68.6	92.1	89.0	88.6	43.7
DistilBERT	**75.2**	42.5	81.6	81.1	82.4	88.3	85.5	90.6	87.7	60.0	92.7	84.5	85.0	55.6

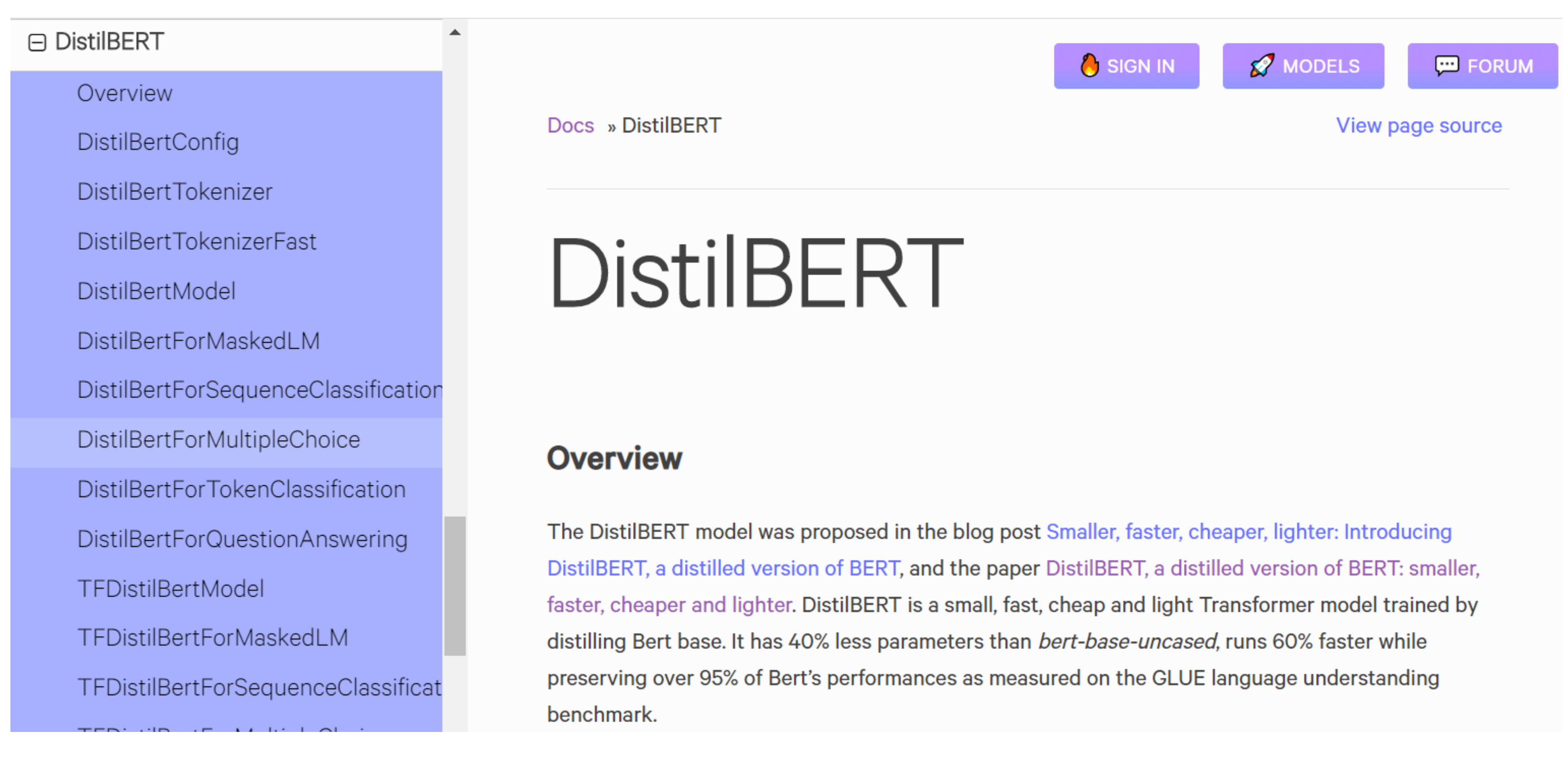
Keep about 95% performance of original BERT.
Much better than ELMo+BiLSTM.

Result of experiments（2）

	Nb of parameters (millions)	Inference Time (s)
GLUE BASELINE (ELMo + BiLSTMs)	180	895
BERT base	110	668
DistilBERT	66	410

Parameters are reduced to about 60% and
inference time to about 60% of original BERT

https://huggingface.co/transformers/model_doc/distilbert.html



Poor Man's BERT

BERT for people who don't have much PC resource

Computer Science > Computation and Language

[Submitted on 8 Apr 2020]

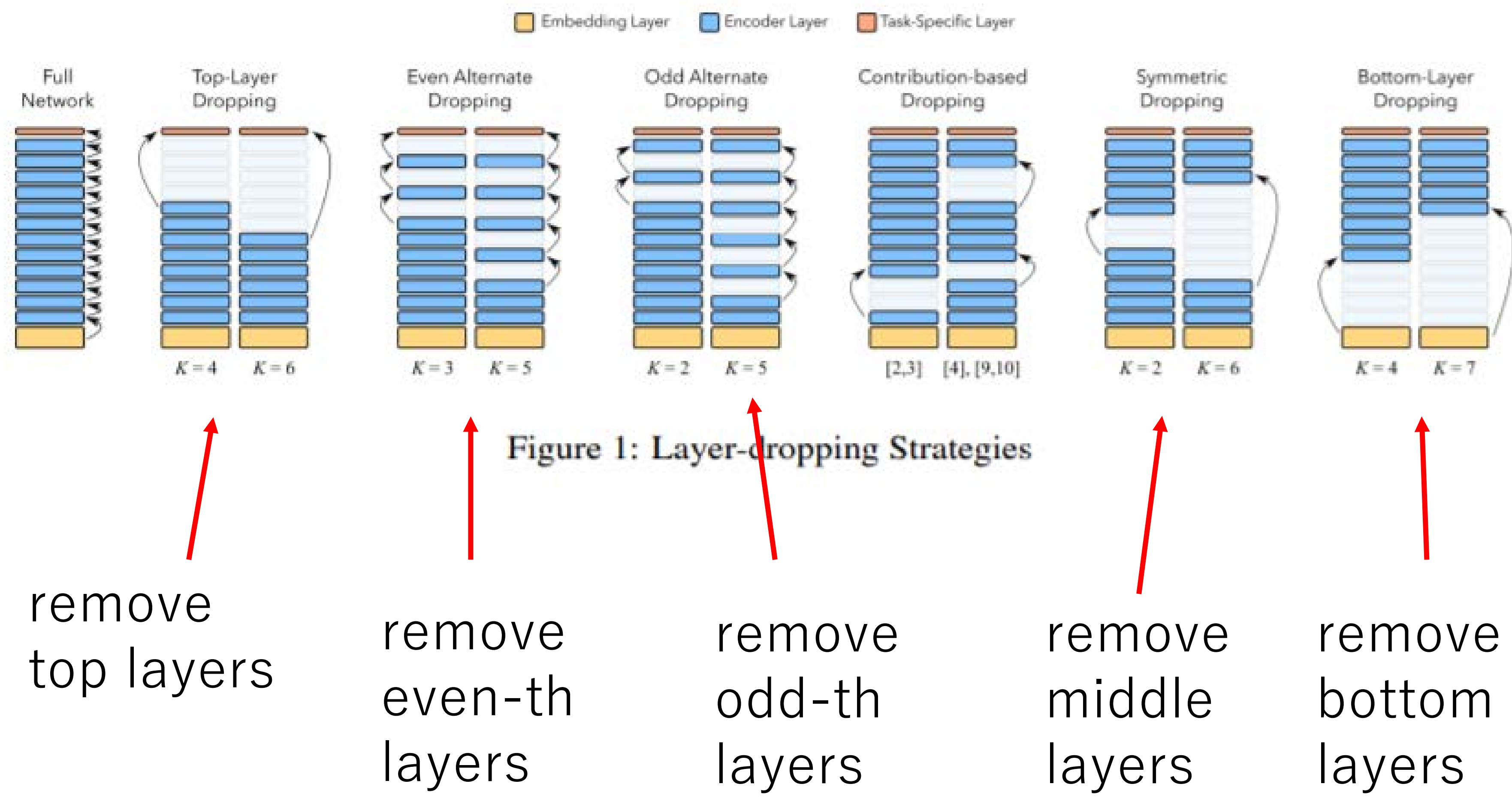
Poor Man's BERT: Smaller and Faster Transformer Models

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, Preslav Nakov

The ongoing neural revolution in Natural Language Processing has recently been dominated by large-scale pre-trained Transformer models, where size does matter: it has been shown that the number of parameters in such a model is typically positively correlated with its performance. Naturally, this situation has unleashed a race for ever larger models, many of which, including the large versions of popular models such as BERT, XLNet, and RoBERTa, are now out of reach for researchers and practitioners without large-memory GPUs/TPUs. To address this issue, we explore a number of memory-light model reduction strategies that do not require model pre-training from scratch. The experimental results show that we are able to prune BERT, RoBERTa and XLNet models by up to 40%, while maintaining up to 98% of their original performance. We also show that our pruned models are on par with DistilBERT in terms of both model size and performance. Finally, our pruning strategies enable interesting comparative analysis between BERT and XLNet.

https://arxiv.org/abs/2004.03844

Pruning layers



Experiment of removing 6 layers

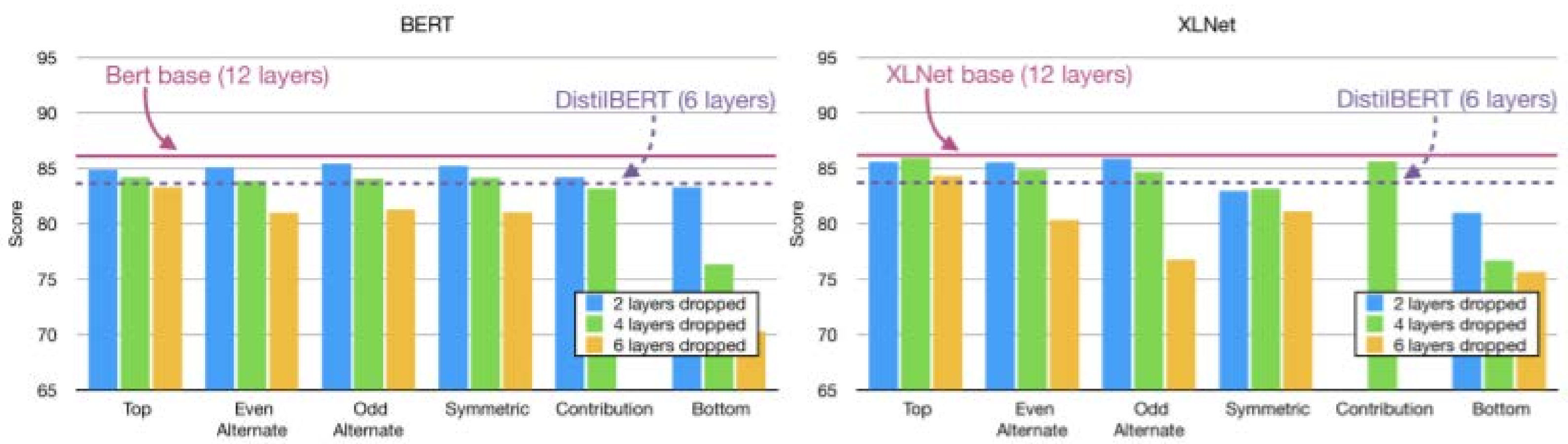


Figure 2: Average classification performance on GLUE tasks when using different layer-dropping strategies and when removing different numbers of layers for BERT and XLNet. Note that the contribution-based strategy selects layers based on the similarity threshold. In some cases it does not select (2,4 or 6) number of layers, which results in some missing bars in the figure.

Top Layer Dropping gets about same performance with DistilBERT of the same size.
Much smaller computational resource than DistilBERT

ALBERT

OpenReview.net Search ICLR 2020 Conference Login

← Go to **ICLR 2020 Conference** homepage

ALBERT: A Lite BERT for Self-supervised Learning of Language Representations

Anonymous

26 Sep 2019 (modified: 26 Sep 2019) ICLR 2020 Conference Blind Submission Readers: Everyone Show Bibtex

Keywords: Natural Language Processing, BERT, Representation Learning

TL;DR: A new pretraining method that establishes new state-of-the-art results on the GLUE, RACE, and SQuAD benchmarks while having fewer parameters compared to BERT-large.

Abstract: Increasing model size when pretraining natural language representations often results in improved performance on downstream tasks. However, at some point further model increases become harder due to GPU/TPU memory limitations, longer training times, and unexpected model degradation. To address these problems, we present two parameter-reduction techniques to lower memory consumption and increase the training speed of BERT. Comprehensive empirical evidence shows that our proposed methods lead to models that scale much better compared to the original BERT. We also use a self-supervised loss that focuses on modeling inter-sentence coherence, and show it consistently helps downstream tasks with multi-sentence inputs. As a result, our best model establishes new state-of-the-art results on the GLUE, RACE, and SQuAD benchmarks while having fewer parameters compared to BERT-large.

29 Replies Add **Public Comment**

https://openreview.net/forum?id=H1eA7AEtvS

Reduce of parameters

Cross-Layer Parameter Sharing

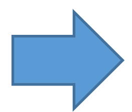
Each layer parameters are shared

Factorized Embedding Parameterization

Word embedding layer is approximated
by multiplying the matrices

Factorized Embedding Parameterization

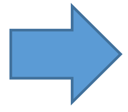
V: vocaburary, H: dimension of word embedding



VH parameters

Ex) BERT-large, V=30000、H=1024
→ 30,720,000 parameters

VH matrix → (VE)*(EH)



VE + EH parameters

Ex) E=128, V=30000、H=1024
→ 3,971,072 parameters (→13%)

Result of experiments

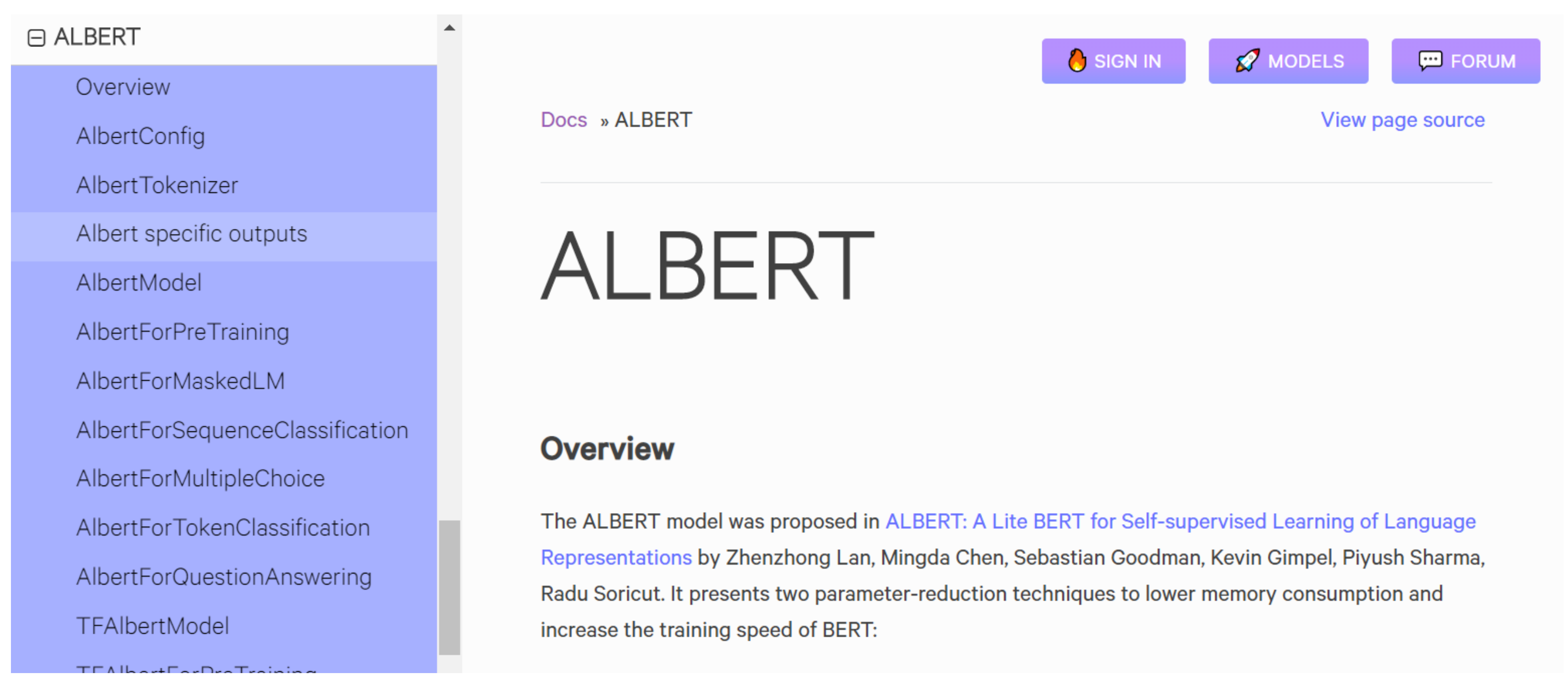
Performance gets a little down

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	17.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	3.8x
	xlarge	1270M	86.4/78.1	75.5/72.6	81.6	90.7	54.3	76.6	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	21.1x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	6.5x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	2.4x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	1.2x

Table 3: Dev set results for models pretrained over BOOKCORPUS and Wikipedia for 125k steps. Here and everywhere else, the Avg column is computed by averaging the scores of the downstream tasks to its left (the two numbers of F1 and EM for each SQuAD are first averaged).

We can build the big size ALBERT.
In this case, it gets SOTA.
Note that parameters of ALBERT-xxlarge
is less then parameters of BERT-large.

https://huggingface.co/transformers/model_doc/albert.html



ALBERT vs. DistilBERT

ALBERT

	Average	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE
V2						
ALBERT-base	82.3	90.2/83.2	82.1/79.3	84.6	92.9	66.8
ALBERT-large	85.7	91.8/85.2	84.9/81.8	86.5	94.9	75.2
ALBERT-xlarge	87.9	92.9/86.4	87.9/84.1	87.9	95.4	80.7
ALBERT-xxlarge	90.9	94.6/89.1	89.8/86.9	90.6	96.8	86.8

VS

DistilBERT

Model	Macro-score	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
BERT-base	**77.6**	48.9	84.3	88.6	89.3	89.5	71.3	91.7	91.2	43.7
DistilBERT	**76.8**	49.1	81.8	90.2	90.2	89.2	62.9	92.7	90.7	44.4

ALBERT is better than DistilBERT

Conclusion

- Introduced BERT which is powerful pretrained model.
- Hugging Face's transformers is very useful
  when we use BERT.
- showed examples to use BERT through the transformers
- BERT has some problems.
- One of them is the size of BERT.
- showed popular methods downsizing BERT.

Thank you very much!

Questions and comments are very welcome.
But I may not be able to answer them quickly because of my poor English. Thus, e-mail is welcome, too. It is OK even after this conference.