supervised_topic_model_training

March 24, 2022

[ ]:
```
!pip install pytorch_pretrained_bert
!pip install pathlib
```

[ ]:
```
%%writefile setup.sh

git clone https://github.com/NVIDIA/apex
pip install -v --no-cache-dir --global-option="--cpp_ext"␣
 ↪--global-option="--cuda_ext" ./apex
```

[ ]:
```
!sh setup.sh
```

[ ]:
```
from pytorch_pretrained_bert.tokenization import BertTokenizer,␣
 ↪WordpieceTokenizer
from pytorch_pretrained_bert.modeling import BertForPreTraining,␣
 ↪BertPreTrainedModel, BertModel, BertConfig, BertForMaskedLM,␣
 ↪BertForSequenceClassification
from pathlib import Path
import torch
import re
from torch import Tensor
from torch.nn import BCEWithLogitsLoss
from fastai.text import Tokenizer, Vocab
import pandas as pd
import collections
import os
import pdb
from tqdm import tqdm, trange
import sys
import random
import numpy as np
import apex
from sklearn.model_selection import train_test_split
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)

from sklearn.metrics import roc_curve, auc
```

```
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,␣
↪SequentialSampler
from torch.utils.data.distributed import DistributedSampler
from pytorch_pretrained_bert.optimization import BertAdam
```

```
`fused_weight_gradient_mlp_cuda` module not found. gradient accumulation fusion
with weight gradient computation disabled.
```

[ ]:
```
import logging
logging.basicConfig(format='%(asctime)s - %(levelname)s - %(name)s -  ␣
↪%(message)s',
                    datefmt='%m/%d/%Y %H:%M:%S',
                    level=logging.INFO)
logger = logging.getLogger(__name__)
```

[ ]:
```
import csv
csv.field_size_limit(sys.maxsize)
```

[ ]:
```
131072
```

[ ]:
```
import os
from google.colab import drive
drive.mount('drive')
os.chdir('/content/drive/MyDrive/Colab Notebooks/')
```

```
Mounted at drive
```

[ ]:
```
DATA_PATH=Path('./data')
DATA_PATH.mkdir(exist_ok=True)

PATH=Path('./tmp')
PATH.mkdir(exist_ok=True)

CLAS_DATA_PATH=PATH/'class'
CLAS_DATA_PATH.mkdir(exist_ok=True)

model_state_dict = None

BERT_PRETRAINED_PATH = Path('./uncased_L-12_H-768_A-12/')


PYTORCH_PRETRAINED_BERT_CACHE = BERT_PRETRAINED_PATH/'cache/'
PYTORCH_PRETRAINED_BERT_CACHE.mkdir(exist_ok=True)
```

0.1 Model Parameters

[ ]:
```
args = {
    "train_size": 86519,
    "val_size": 5000,
    "full_data_dir": DATA_PATH,
    "data_dir": PATH,
    "task_name": "topic_multilabel",
    "no_cuda": False,
    "bert_model": BERT_PRETRAINED_PATH,
    "output_dir": CLAS_DATA_PATH/'output',
    "max_seq_length": 128,
    "do_train": True,
    "do_eval": True,
    "do_lower_case": True,
    "train_batch_size": 8,
    "eval_batch_size": 1,
    "learning_rate": 3e-5,
    "num_train_epochs": 4.0,
    "warmup_proportion": 0.1,
    "no_cuda": False,
    "local_rank": -1,
    "seed": 42,
    "gradient_accumulation_steps": 1,
    "optimize_on_cpu": False,
    "fp16": False,
    "loss_scale": 128
}
```

0.1.1 Model Class

[ ]:
```
class BertForMultiLabelSequenceClassification(BertPreTrainedModel):
    def __init__(self, config, num_labels=2):
        super(BertForMultiLabelSequenceClassification, self).__init__(config)
        self.num_labels = num_labels
        self.bert = BertModel(config)
        self.dropout = torch.nn.Dropout(config.hidden_dropout_prob)
        self.classifier = torch.nn.Linear(config.hidden_size, num_labels)
        self.apply(self.init_bert_weights)

    def forward(self, input_ids, token_type_ids=None, attention_mask=None,␣
↪labels=None):
        _, pooled_output = self.bert(input_ids, token_type_ids, attention_mask,␣
↪output_all_encoded_layers=False)
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
```

```
        if labels is not None:
            loss_fct = BCEWithLogitsLoss()
            loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1,␣
↪self.num_labels))
            return loss
        else:
            return logits

    def freeze_bert_encoder(self):
        for param in self.bert.parameters():
            param.requires_grad = False

    def unfreeze_bert_encoder(self):
        for param in self.bert.parameters():
            param.requires_grad = True
```

0.2 Data representation class

[ ]:
```
class InputExample(object):
    """A single training/test example for simple sequence classification."""

    def __init__(self, guid, text_a, text_b=None, labels=None):
        """Constructs a InputExample.

        Args:
            guid: Unique id for the example.
            text_a: string. The untokenized text of the first sequence. For␣
↪single
            sequence tasks, only this sequence must be specified.
            text_b: (Optional) string. The untokenized text of the second␣
↪sequence.
            Only must be specified for sequence pair tasks.
            labels: (Optional) [string]. The label of the example. This should␣
↪be
            specified for train and dev examples, but not for test examples.
        """
        self.guid = guid
        self.text_a = text_a
        self.text_b = text_b
        self.labels = labels


class InputFeatures(object):
    """A single set of features of data."""

    def __init__(self, input_ids, input_mask, segment_ids, label_ids):
```

```
        self.input_ids = input_ids
        self.input_mask = input_mask
        self.segment_ids = segment_ids
        self.label_ids = label_ids
```

[ ]:
```
class DataProcessor(object):
    """Base class for data converters for sequence classification data sets."""

    def get_train_examples(self, data_dir):
        """Gets a collection of `InputExample`s for the train set."""
        raise NotImplementedError()

    def get_dev_examples(self, data_dir):
        """Gets a collection of `InputExample`s for the dev set."""
        raise NotImplementedError()

    def get_test_examples(self, data_dir, data_file_name, size=-1):
        """Gets a collection of `InputExample`s for the dev set."""
        raise NotImplementedError()

    def get_labels(self):
        """Gets the list of labels for this data set."""
        raise NotImplementedError()
```

[ ]:
```
class MultiLabelTextProcessor(DataProcessor):

    def __init__(self, data_dir):
        self.data_dir = data_dir
        self.labels = None


    def get_train_examples(self, data_dir, size=-1):
        filename = 'fulltext_cleaned_train.csv'

        logger.info("LOOKING AT {}".format(os.path.join(data_dir, filename)))
        if size == -1:
            data_df = pd.read_csv(os.path.join(data_dir, filename), engine =␣
 ↪'python')
#             data_df['comment_text'] = data_df['comment_text'].apply(cleanHtml)
            return self._create_examples(data_df, "train")
        else:
            data_df = pd.read_csv(os.path.join(data_dir, filename), engine =␣
 ↪'python')
#             data_df['comment_text'] = data_df['comment_text'].apply(cleanHtml)
            return self._create_examples(data_df.sample(size), "train")

    def get_dev_examples(self, data_dir, size=-1):
```

```
        """See base class."""
        filename = 'fulltext_cleaned_test.csv'

        if size == -1:
            data_df = pd.read_csv(os.path.join(data_dir, filename),␣
 ↪error_bad_lines = False, engine = "python")
#             data_df['comment_text'] = data_df['comment_text'].apply(cleanHtml)
            return self._create_examples(data_df, "dev")
        else:
            data_df = pd.read_csv(os.path.join(data_dir, filename),␣
 ↪error_bad_lines = False, engine = "python")
#             data_df['comment_text'] = data_df['comment_text'].apply(cleanHtml)
            return self._create_examples(data_df.sample(size), "dev")

    def get_test_examples(self, data_dir, data_file_name, size=-1):
        data_df = pd.read_csv(os.path.join(data_dir, data_file_name))
#         data_df['comment_text'] = data_df['comment_text'].apply(cleanHtml)
        if size == -1:
            return self._create_examples(data_df, "test")
        else:
            return self._create_examples(data_df.sample(size), "test")

    def get_labels(self):
        """See base class."""
        if self.labels == None:
            self.labels = list(pd.read_csv(os.path.join(self.data_dir, "classes.
 ↪txt"),header=None)[0].values)
        return self.labels

    def _create_examples(self, df, set_type, labels_available=True):
        """Creates examples for the training and dev sets."""
        examples = []
        for (i, row) in enumerate(df.values):
            guid = row[0]
            text_a = row[1]
            if labels_available:
                labels = row[2:]
            else:
                labels = []
            examples.append(
                InputExample(guid=guid, text_a=text_a, labels=labels))
        return examples
```

[ ]:
```
def convert_examples_to_features(examples, label_list, max_seq_length,␣
 ↪tokenizer):
    """Loads a data file into a list of `InputBatch`s."""
```

```
    label_map = {label : i for i, label in enumerate(label_list)}

    features = []
    for (ex_index, example) in enumerate(examples):
        tokens_a = tokenizer.tokenize(example.text_a)

        tokens_b = None
        if example.text_b:
            tokens_b = tokenizer.tokenize(example.text_b)
            _truncate_seq_pair(tokens_a, tokens_b, max_seq_length - 3)
        else:
            if len(tokens_a) > max_seq_length - 2:
                tokens_a = tokens_a[:(max_seq_length - 2)]
        tokens = ["[CLS]"] + tokens_a + ["[SEP]"]
        segment_ids = [0] * len(tokens)

        if tokens_b:
            tokens += tokens_b + ["[SEP]"]
            segment_ids += [1] * (len(tokens_b) + 1)

        input_ids = tokenizer.convert_tokens_to_ids(tokens)

        input_mask = [1] * len(input_ids)

        padding = [0] * (max_seq_length - len(input_ids))
        input_ids += padding
        input_mask += padding
        segment_ids += padding

        assert len(input_ids) == max_seq_length
        assert len(input_mask) == max_seq_length
        assert len(segment_ids) == max_seq_length

        labels_ids = []
        for label in example.labels:
            labels_ids.append(float(label))


        if ex_index < 0:
            logger.info("*** Example ***")
            logger.info("guid: %s" % (example.guid))
            logger.info("tokens: %s" % " ".join(
                    [str(x) for x in tokens]))
            logger.info("input_ids: %s" % " ".join([str(x) for x in input_ids]))
            logger.info("input_mask: %s" % " ".join([str(x) for x in␣
↪input_mask]))
```

```
            logger.info(
                    "segment_ids: %s" % " ".join([str(x) for x in segment_ids]))
            logger.info("label: %s (id = %s)" % (example.labels, label_id))

        features.append(
                InputFeatures(input_ids=input_ids,
                              input_mask=input_mask,
                              segment_ids=segment_ids,
                              label_ids=labels_ids))
    return features
```

[ ]:
```
def _truncate_seq_pair(tokens_a, tokens_b, max_length):
    """Truncates a sequence pair in place to the maximum length."""
    while True:
        total_length = len(tokens_a) + len(tokens_b)
        if total_length <= max_length:
            break
        if len(tokens_a) > len(tokens_b):
            tokens_a.pop()
        else:
            tokens_b.pop()
```

0.3 Metric functions

[ ]:
```
def accuracy(out, labels):
    outputs = np.argmax(out, axis=1)
    return np.sum(outputs == labels)

def accuracy_thresh(y_pred:Tensor, y_true:Tensor, thresh:float=0.5, sigmoid:
 ↪bool=True):
    "Compute accuracy when `y_pred` and `y_true` are the same size."
    if sigmoid: y_pred = y_pred.sigmoid()
#     return ((y_pred>thresh)==y_true.byte()).float().mean().item()
    return np.mean(((y_pred>thresh)==y_true.byte()).float().cpu().numpy(),␣
 ↪axis=1).sum()


def fbeta(y_pred:Tensor, y_true:Tensor, thresh:float=0.2, beta:float=2, eps:
 ↪float=1e-9, sigmoid:bool=True):
    "Computes the f_beta between `preds` and `targets`"
    beta2 = beta ** 2
    if sigmoid: y_pred = y_pred.sigmoid()
    y_pred = (y_pred>thresh).float()
    y_true = y_true.float()
    TP = (y_pred*y_true).sum(dim=1)
    prec = TP/(y_pred.sum(dim=1)+eps)
    rec = TP/(y_true.sum(dim=1)+eps)
```

```
    res = (prec*rec)/(prec*beta2+rec+eps)*(1+beta2)
    return res.mean().item()
```

0.4 Training warmup

[ ]:
```
def warmup_linear(x, warmup=0.002):
    if x < warmup:
        return x/warmup
    return 1.0 - x
```

[ ]:
```
processors = {
    "topic_multilabel": MultiLabelTextProcessor
}

# Setup GPU parameters

if args["local_rank"] == -1 or args["no_cuda"]:
    device = torch.device("cuda" if torch.cuda.is_available() and not␣
 ↪args["no_cuda"] else "cpu")
    n_gpu = torch.cuda.device_count()
#     n_gpu = 1
else:
    torch.cuda.set_device(args['local_rank'])
    device = torch.device("cuda", args['local_rank'])
    n_gpu = 1
    # Initializes the distributed backend which will take care of sychronizing␣
 ↪nodes/GPUs
    torch.distributed.init_process_group(backend='nccl')
logger.info("device: {} n_gpu: {}, distributed training: {}, 16-bits training:␣
 ↪{}".format(
        device, n_gpu, bool(args['local_rank'] != -1), args['fp16']))
```

```
03/04/2022 23:55:26 - INFO - __main__ -   device: cuda n_gpu: 1, distributed
training: False, 16-bits training: False
```

[ ]:
```
args['train_batch_size'] = int(args['train_batch_size'] /␣
 ↪args['gradient_accumulation_steps'])
```

[ ]:
```
random.seed(args['seed'])
np.random.seed(args['seed'])
torch.manual_seed(args['seed'])
if n_gpu > 0:
    torch.cuda.manual_seed_all(args['seed'])
```

[ ]:
```
task_name = args['task_name'].lower()
```

```
if task_name not in processors:
    raise ValueError("Task not found: %s" % (task_name))
```

[ ]:
```
processor = processors[task_name](args['data_dir'])
label_list = processor.get_labels()
num_labels = len(label_list)
```

[ ]:
```
label_list
```

[ ]:
```
['abortion',
 'antiasian',
 'antiblack',
 'antiimmigrant',
 'antilatinx',
 'antilgbt',
 'antimuslim',
 'antisemitic',
 'antivaxx',
 'biden',
 'bigtech',
 'coronavirus',
 'misogyny',
 'pseudoscience',
 'voterfraud',
 'whitesupremacy']
```

[ ]:
```
tokenizer = BertTokenizer.from_pretrained(args['bert_model'],␣
↪do_lower_case=args['do_lower_case'])
```

```
03/04/2022 23:55:38 - INFO - pytorch_pretrained_bert.tokenization -   loading
vocabulary file uncased_L-12_H-768_A-12/vocab.txt
```

[ ]:
```
train_examples = None
num_train_steps = None
if args['do_train']:
    train_examples = processor.get_train_examples(args['full_data_dir'],␣
↪size=args['train_size'])
#     train_examples = processor.get_train_examples(args['data_dir'],␣
↪size=args['train_size'])
    num_train_steps = int(
        len(train_examples) / args['train_batch_size'] /␣
↪args['gradient_accumulation_steps'] * args['num_train_epochs'])
```

```
03/04/2022 23:55:39 - INFO - __main__ -   LOOKING AT
data/fulltext_cleaned_train.csv
```

[ ]:
```
train_examples[0].labels
```

[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=object)
```

[ ]:
```
# Prepare model
def get_model():
    if model_state_dict:
        model = BertForMultiLabelSequenceClassification.
 ↪from_pretrained(args['bert_model'], num_labels = num_labels,␣
 ↪state_dict=model_state_dict)
    else:
        model = BertForMultiLabelSequenceClassification.
 ↪from_pretrained(args['bert_model'], num_labels = num_labels)
    return model

model = get_model()

if args['fp16']:
    model.half()
model.to(device)
if args['local_rank'] != -1:
    try:
        from apex.parallel import DistributedDataParallel as DDP
    except ImportError:
        raise ImportError("Please install apex from https://www.github.com/
 ↪nvidia/apex to use distributed and fp16 training.")

    model = DDP(model)
elif n_gpu > 1:
    model = torch.nn.DataParallel(model)
```

```
03/04/2022 23:56:00 - INFO - pytorch_pretrained_bert.modeling -   loading
archive file uncased_L-12_H-768_A-12 from cache at uncased_L-12_H-768_A-12
03/04/2022 23:56:00 - INFO - pytorch_pretrained_bert.modeling -   Model config {
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "type_vocab_size": 2,
  "vocab_size": 30522
}

03/04/2022 23:56:09 - INFO - pytorch_pretrained_bert.modeling -   Weights of
BertForMultiLabelSequenceClassification not initialized from pretrained model:
```

```
['classifier.weight', 'classifier.bias']
03/04/2022 23:56:09 - INFO - pytorch_pretrained_bert.modeling -   Weights from
pretrained model not used in BertForMultiLabelSequenceClassification:
['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
```

[ ]:
```python
from torch.optim.lr_scheduler import _LRScheduler, Optimizer

class CyclicLR(object):

    def __init__(self, optimizer, base_lr=1e-3, max_lr=6e-3,
                 step_size=2000, mode='triangular', gamma=1.,
                 scale_fn=None, scale_mode='cycle', last_batch_iteration=-1):

        self.optimizer = optimizer

        if isinstance(base_lr, list) or isinstance(base_lr, tuple):
            if len(base_lr) != len(optimizer.param_groups):
                raise ValueError("expected {} base_lr, got {}".format(
                    len(optimizer.param_groups), len(base_lr)))
            self.base_lrs = list(base_lr)
        else:
            self.base_lrs = [base_lr] * len(optimizer.param_groups)

        if isinstance(max_lr, list) or isinstance(max_lr, tuple):
            if len(max_lr) != len(optimizer.param_groups):
                raise ValueError("expected {} max_lr, got {}".format(
                    len(optimizer.param_groups), len(max_lr)))
            self.max_lrs = list(max_lr)
        else:
            self.max_lrs = [max_lr] * len(optimizer.param_groups)

        self.step_size = step_size

        if mode not in ['triangular', 'triangular2', 'exp_range'] \
                and scale_fn is None:
            raise ValueError('mode is invalid and scale_fn is None')

        self.mode = mode
        self.gamma = gamma

        if scale_fn is None:
            if self.mode == 'triangular':
                self.scale_fn = self._triangular_scale_fn
```

```
                self.scale_mode = 'cycle'
            elif self.mode == 'triangular2':
                self.scale_fn = self._triangular2_scale_fn
                self.scale_mode = 'cycle'
            elif self.mode == 'exp_range':
                self.scale_fn = self._exp_range_scale_fn
                self.scale_mode = 'iterations'
        else:
            self.scale_fn = scale_fn
            self.scale_mode = scale_mode

        self.batch_step(last_batch_iteration + 1)
        self.last_batch_iteration = last_batch_iteration

    def batch_step(self, batch_iteration=None):
        if batch_iteration is None:
            batch_iteration = self.last_batch_iteration + 1
        self.last_batch_iteration = batch_iteration
        for param_group, lr in zip(self.optimizer.param_groups, self.get_lr()):
            param_group['lr'] = lr

    def _triangular_scale_fn(self, x):
        return 1.

    def _triangular2_scale_fn(self, x):
        return 1 / (2. ** (x - 1))

    def _exp_range_scale_fn(self, x):
        return self.gamma**(x)

    def get_lr(self):
        step_size = float(self.step_size)
        cycle = np.floor(1 + self.last_batch_iteration / (2 * step_size))
        x = np.abs(self.last_batch_iteration / step_size - 2 * cycle + 1)

        lrs = []
        param_lrs = zip(self.optimizer.param_groups, self.base_lrs, self.
↪max_lrs)
        for param_group, base_lr, max_lr in param_lrs:
            base_height = (max_lr - base_lr) * np.maximum(0, (1 - x))
            if self.scale_mode == 'cycle':
                lr = base_lr + base_height * self.scale_fn(cycle)
            else:
                lr = base_lr + base_height * self.scale_fn(self.
↪last_batch_iteration)
            lrs.append(lr)
        return lrs
```

[ ]:
```
# Prepare optimizer
param_optimizer = list(model.named_parameters())
no_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
    {'params': [p for n, p in param_optimizer if not any(nd in n for nd in␣
 ↪no_decay)], 'weight_decay': 0.01},
    {'params': [p for n, p in param_optimizer if any(nd in n for nd in␣
 ↪no_decay)], 'weight_decay': 0.0}
    ]
t_total = num_train_steps
if args['local_rank'] != -1:
    t_total = t_total // torch.distributed.get_world_size()
if args['fp16']:
    try:
        from apex.optimizers import FP16_Optimizer
        from apex.optimizers import FusedAdam
    except ImportError:
        raise ImportError("Please install apex from https://www.github.com/
 ↪nvidia/apex to use distributed and fp16 training.")

    optimizer = FusedAdam(optimizer_grouped_parameters,
                          lr=args['learning_rate'],
                          bias_correction=False,
                          max_grad_norm=1.0)
    if args['loss_scale'] == 0:
        optimizer = FP16_Optimizer(optimizer, dynamic_loss_scale=True)
    else:
        optimizer = FP16_Optimizer(optimizer,␣
 ↪static_loss_scale=args['loss_scale'])

else:
    optimizer = BertAdam(optimizer_grouped_parameters,
                         lr=args['learning_rate'],
                         warmup=args['warmup_proportion'],
                         t_total=t_total)

scheduler = CyclicLR(optimizer, base_lr=2e-5, max_lr=5e-5, step_size=2500,␣
 ↪last_batch_iteration=0)
```

[ ]:
```
# Eval Fn
eval_examples = processor.get_dev_examples(args['data_dir'],␣
 ↪size=args['val_size'])
def eval():
    args['output_dir'].mkdir(exist_ok=True)


    eval_features = convert_examples_to_features(
```

```
        eval_examples, label_list, args['max_seq_length'], tokenizer)
    logger.info("***** Running evaluation *****")
    logger.info("  Num examples = %d", len(eval_examples))
    logger.info("  Batch size = %d", args['eval_batch_size'])
    all_input_ids = torch.tensor([f.input_ids for f in eval_features],␣
↪dtype=torch.long)
    all_input_mask = torch.tensor([f.input_mask for f in eval_features],␣
↪dtype=torch.long)
    all_segment_ids = torch.tensor([f.segment_ids for f in eval_features],␣
↪dtype=torch.long)
    all_label_ids = torch.tensor([f.label_ids for f in eval_features],␣
↪dtype=torch.float)
    eval_data = TensorDataset(all_input_ids, all_input_mask, all_segment_ids,␣
↪all_label_ids)
    # Run prediction for full data
    eval_sampler = SequentialSampler(eval_data)
    eval_dataloader = DataLoader(eval_data, sampler=eval_sampler,␣
↪batch_size=args['eval_batch_size'])

    all_logits = None
    all_labels = None

    model.eval()
    eval_loss, eval_accuracy = 0, 0
    nb_eval_steps, nb_eval_examples = 0, 0
    for input_ids, input_mask, segment_ids, label_ids in eval_dataloader:
        input_ids = input_ids.to(device)
        input_mask = input_mask.to(device)
        segment_ids = segment_ids.to(device)
        label_ids = label_ids.to(device)

        with torch.no_grad():
            tmp_eval_loss = model(input_ids, segment_ids, input_mask, label_ids)
            logits = model(input_ids, segment_ids, input_mask)

        tmp_eval_accuracy = accuracy_thresh(logits, label_ids)
        if all_logits is None:
            all_logits = logits.detach().cpu().numpy()
        else:
            all_logits = np.concatenate((all_logits, logits.detach().cpu().
↪numpy()), axis=0)

        if all_labels is None:
            all_labels = label_ids.detach().cpu().numpy()
        else:
```

```
            all_labels = np.concatenate((all_labels, label_ids.detach().cpu().
 ↪numpy()), axis=0)


        eval_loss += tmp_eval_loss.mean().item()
        eval_accuracy += tmp_eval_accuracy

        nb_eval_examples += input_ids.size(0)
        nb_eval_steps += 1

    eval_loss = eval_loss / nb_eval_steps
    eval_accuracy = eval_accuracy / nb_eval_examples

#     ROC-AUC calcualation
    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    for i in range(num_labels):
        fpr[i], tpr[i], _ = roc_curve(all_labels[:, i], all_logits[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(all_labels.ravel(), all_logits.
 ↪ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    result = {'eval_loss': eval_loss,
              'eval_accuracy': eval_accuracy,
              'roc_auc': roc_auc  }

    output_eval_file = os.path.join(args['output_dir'], "eval_results.txt")
    with open(output_eval_file, "w") as writer:
        logger.info("***** Eval results *****")
        for key in sorted(result.keys()):
            logger.info("  %s = %s", key, str(result[key]))
    return result
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning:
The error_bad_lines argument has been deprecated and will be removed in a future
version.
```

0.4.1 Load training data

[ ]:
```
train_features = convert_examples_to_features(
    train_examples, label_list, args['max_seq_length'], tokenizer)
```

[ ]:
```
logger.info("***** Running training *****")
logger.info("  Num examples = %d", len(train_examples))
logger.info("  Batch size = %d", args['train_batch_size'])
logger.info("  Num steps = %d", num_train_steps)
all_input_ids = torch.tensor([f.input_ids for f in train_features], dtype=torch.
 ↪long)
all_input_mask = torch.tensor([f.input_mask for f in train_features],␣
 ↪dtype=torch.long)
all_segment_ids = torch.tensor([f.segment_ids for f in train_features],␣
 ↪dtype=torch.long)
all_label_ids = torch.tensor([f.label_ids for f in train_features], dtype=torch.
 ↪float)
train_data = TensorDataset(all_input_ids, all_input_mask, all_segment_ids,␣
 ↪all_label_ids)
if args['local_rank'] == -1:
    train_sampler = RandomSampler(train_data)
else:
    train_sampler = DistributedSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler,␣
 ↪batch_size=args['train_batch_size'])
```

```
03/05/2022 00:19:34 - INFO - __main__ -   ***** Running training *****
03/05/2022 00:19:34 - INFO - __main__ -     Num examples = 86519
03/05/2022 00:19:34 - INFO - __main__ -     Batch size = 8
03/05/2022 00:19:34 - INFO - __main__ -     Num steps = 43259
```

[ ]:
```
from tqdm import tqdm_notebook as tqdm
```

0.4.2 Train Model

[ ]:
```
def fit():
    num_epocs=args['num_train_epochs']
    global_step = 0
    model.train()
    for i_ in tqdm(range(int(num_epocs)), desc="Epoch"):

        tr_loss = 0
        nb_tr_examples, nb_tr_steps = 0, 0
        for step, batch in enumerate(tqdm(train_dataloader, desc="Iteration")):

            batch = tuple(t.to(device) for t in batch)
            input_ids, input_mask, segment_ids, label_ids = batch
```

```
            loss = model(input_ids, segment_ids, input_mask, label_ids)
            if n_gpu > 1:
                loss = loss.mean() # mean() to average on multi-gpu.
            if args['gradient_accumulation_steps'] > 1:
                loss = loss / args['gradient_accumulation_steps']

            if args['fp16']:
                optimizer.backward(loss)
            else:
                loss.backward()

            tr_loss += loss.item()
            nb_tr_examples += input_ids.size(0)
            nb_tr_steps += 1
            if (step + 1) % args['gradient_accumulation_steps'] == 0:
    #             scheduler.batch_step()
                # modify learning rate with special warm up BERT uses
                lr_this_step = args['learning_rate'] *␣
↪warmup_linear(global_step/t_total, args['warmup_proportion'])
                for param_group in optimizer.param_groups:
                    param_group['lr'] = lr_this_step
                optimizer.step()
                optimizer.zero_grad()
                global_step += 1

        logger.info('Loss after epoc {}'.format(tr_loss / nb_tr_steps))
        logger.info('Eval after epoc {}'.format(i_+1))
        eval()
```

[ ]:
```
# Freeze BERT layers for 1 epoch
# model.module.freeze_bert_encoder()
```

[ ]:
```
# fit(1)
```

[ ]:
```
model.unfreeze_bert_encoder()
```

[ ]:
```
fit()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """

Epoch:   0%|          | 0/4 [00:00<?, ?it/s]

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9:
TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
```

```
  if __name__ == '__main__':

Iteration:   0%|          | 0/10815 [00:00<?, ?it/s]

/usr/local/lib/python3.7/dist-
packages/pytorch_pretrained_bert/optimization.py:275: UserWarning: This overload
of add_ is deprecated:
        add_(Number alpha, Tensor other)
Consider using one of the following signatures instead:
        add_(Tensor other, *, Number alpha) (Triggered internally at
../torch/csrc/utils/python_arg_parser.cpp:1050.)
  next_m.mul_(beta1).add_(1 - beta1, grad)
03/05/2022 00:56:06 - INFO - __main__ -   Loss after epoc 0.17034753629015487
03/05/2022 00:56:06 - INFO - __main__ -   Eval after epoc 1
03/05/2022 00:57:28 - INFO - __main__ -   ***** Running evaluation *****
03/05/2022 00:57:28 - INFO - __main__ -     Num examples = 5000
03/05/2022 00:57:28 - INFO - __main__ -     Batch size = 1
03/05/2022 00:59:49 - INFO - __main__ -   ***** Eval results *****
03/05/2022 00:59:49 - INFO - __main__ -     eval_accuracy = 0.95615
03/05/2022 00:59:49 - INFO - __main__ -     eval_loss = 0.10693050102265551
03/05/2022 00:59:49 - INFO - __main__ -     roc_auc = {0: 0.9514481707317074, 1:
0.8997448294386248, 2: 0.9663539269945898, 3: 0.9619150993714203, 4:
0.9519209123739287, 5: 0.9242663311084363, 6: 0.9280042911004798, 7:
0.9120572352994227, 8: 0.9653050011094791, 9: 0.8660744112858099, 10:
0.8369735801662636, 11: 0.9131374436358335, 12: 0.9447970643525552, 13:
0.977452849117175, 14: 0.9244709062994128, 15: 0.9335964673913042, 'micro':
0.9586707836738443}
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9:
TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  if __name__ == '__main__':

Iteration:   0%|          | 0/10815 [00:00<?, ?it/s]

03/05/2022 01:36:03 - INFO - __main__ -   Loss after epoc 0.09441004225547618
03/05/2022 01:36:03 - INFO - __main__ -   Eval after epoc 2
03/05/2022 01:37:20 - INFO - __main__ -   ***** Running evaluation *****
03/05/2022 01:37:20 - INFO - __main__ -     Num examples = 5000
03/05/2022 01:37:20 - INFO - __main__ -     Batch size = 1
03/05/2022 01:39:33 - INFO - __main__ -   ***** Eval results *****
03/05/2022 01:39:33 - INFO - __main__ -     eval_accuracy = 0.9600625
03/05/2022 01:39:33 - INFO - __main__ -     eval_loss = 0.09152252937280574
03/05/2022 01:39:33 - INFO - __main__ -     roc_auc = {0: 0.9847103658536585, 1:
0.9445115945921747, 2: 0.9777299297307381, 3: 0.967866916569879, 4:
0.9540209112113727, 5: 0.9561297734981946, 6: 0.9628538595601969, 7:
0.9414302060786436, 8: 0.9924396918817824, 9: 0.8947722768944668, 10:
0.9016402006424871, 11: 0.9404456287829597, 12: 0.9661901095318364, 13:
0.9978054775280899, 14: 0.9376930932865062, 15: 0.9608214673913044, 'micro':
0.9713698685190051}
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9:
```

```
TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  if __name__ == '__main__':

Iteration:   0%|          | 0/10815 [00:00<?, ?it/s]

03/05/2022 02:15:15 - INFO - __main__ -   Loss after epoc 0.07472782800513587
03/05/2022 02:15:15 - INFO - __main__ -   Eval after epoc 3
03/05/2022 02:16:33 - INFO - __main__ -   ***** Running evaluation *****
03/05/2022 02:16:33 - INFO - __main__ -     Num examples = 5000
03/05/2022 02:16:33 - INFO - __main__ -     Batch size = 1
03/05/2022 02:18:46 - INFO - __main__ -   ***** Eval results *****
03/05/2022 02:18:46 - INFO - __main__ -     eval_accuracy = 0.9596375
03/05/2022 02:18:46 - INFO - __main__ -     eval_loss = 0.08686907133776695
03/05/2022 02:18:46 - INFO - __main__ -     roc_auc = {0: 0.9813541666666666, 1:
0.9736827379353568, 2: 0.981185094210559, 3: 0.9672899292660921, 4:
0.9560384742615391, 5: 0.9637651821862347, 6: 0.9631524312764288, 7:
0.9375577764249639, 8: 0.9878644110778855, 9: 0.8976660958687978, 10:
0.9145801233724742, 11: 0.9451145835466628, 12: 0.9586914448629638, 13:
0.9935543739967897, 14: 0.9408265614966353, 15: 0.9617051630434783, 'micro':
0.9735338752653296}
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9:
TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  if __name__ == '__main__':

Iteration:   0%|          | 0/10815 [00:00<?, ?it/s]

03/05/2022 02:54:29 - INFO - __main__ -   Loss after epoc 0.06512773808365721
03/05/2022 02:54:29 - INFO - __main__ -   Eval after epoc 4
03/05/2022 02:55:46 - INFO - __main__ -   ***** Running evaluation *****
03/05/2022 02:55:46 - INFO - __main__ -     Num examples = 5000
03/05/2022 02:55:46 - INFO - __main__ -     Batch size = 1
03/05/2022 02:57:58 - INFO - __main__ -   ***** Eval results *****
03/05/2022 02:57:58 - INFO - __main__ -     eval_accuracy = 0.958575
03/05/2022 02:57:58 - INFO - __main__ -     eval_loss = 0.08830282707642764
03/05/2022 02:57:58 - INFO - __main__ -     roc_auc = {0: 0.9812118902439025, 1:
0.9710079237174322, 2: 0.9822305826752067, 3: 0.9655290587255932, 4:
0.9515214158663525, 5: 0.9648561111719006, 6: 0.964341510495143, 7:
0.9381947262806638, 8: 0.9838174536924523, 9: 0.8968249613975152, 10:
0.9129084692377147, 11: 0.941818567317156, 12: 0.957761915662567, 13:
0.9903566412520064, 14: 0.9375543378950534, 15: 0.9621839673913044, 'micro':
0.9725236272287239}
```

[ ]:
```
# Save a trained model
model_to_save = model.module if hasattr(model, 'module') else model  # Only␣
 ↪save the model it-self
output_model_file = os.path.join(PYTORCH_PRETRAINED_BERT_CACHE,␣
 ↪"finetuned_pytorch_model.bin")
```

```
torch.save(model_to_save.state_dict(), output_model_file)

# Load a trained model that you have fine-tuned
model_state_dict = torch.load(output_model_file)
model = BertForMultiLabelSequenceClassification.
 ↪from_pretrained(args['bert_model'], num_labels = num_labels,␣
 ↪state_dict=model_state_dict)
model.to(device)
```

```
03/05/2022 02:58:06 - INFO - pytorch_pretrained_bert.modeling -   loading
archive file uncased_L-12_H-768_A-12 from cache at uncased_L-12_H-768_A-12
03/05/2022 02:58:06 - INFO - pytorch_pretrained_bert.modeling -   Model config {
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "type_vocab_size": 2,
  "vocab_size": 30522
}
```

[ ]:
```
BertForMultiLabelSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (1): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (2): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (3): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (4): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (5): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
```

```
        (6): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (7): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
```

```
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (8): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (9): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (10): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (11): 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): FusedLayerNorm(torch.Size([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): FusedLayerNorm(torch.Size([768]), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=16, bias=True)
)
```