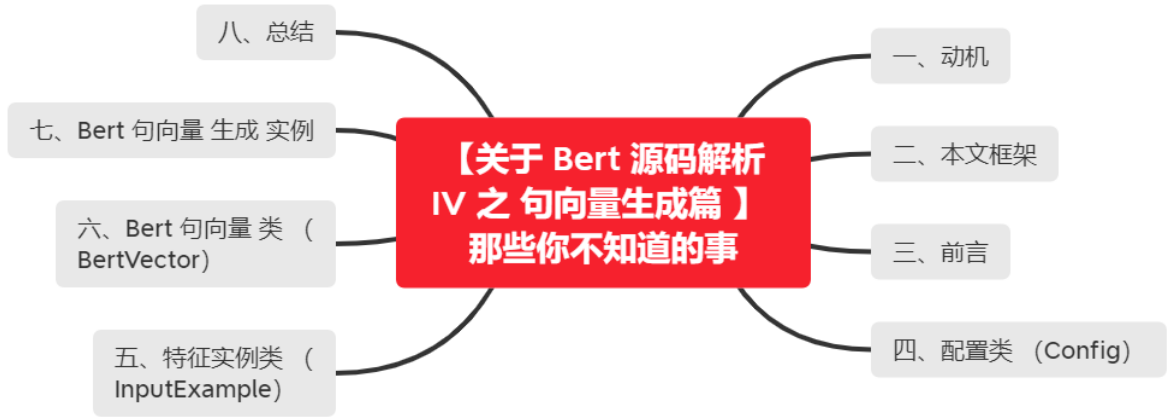
【关于 Bert 源码解析IV 之 句向量生成篇 】那些你不知道的事



一、前言

本文 主要 解读 Bert 模型的 微调 模块代码：

- extract_feature.py：主要用于 生成 Bert 句向量

二、配置类 （Config）

该类主要包含 一些 Bert 模型地址，和一些采用配置信息

```
import os
import tensorflow as tf
class Config():
    def __init__(self):
        tf.logging.set_verbosity(tf.logging.INFO)
        self.file_path = os.path.dirname(__file__)
        # Bert 模型 的 路径
        self.model_dir = os.path.join(self.file_path,
'F:/document/datasets/nlpData/bert/chinese_L-12_H-768_A-12/')
        # Bert 模型 配置
        self.config_name = os.path.join(self.model_dir, 'bert_config.json')
        # Bert 模型 文件
        self.ckpt_name = os.path.join(self.model_dir, 'bert_model.ckpt')
        # Bert 输出
        self.output_dir = os.path.join("", 'output/')
        # Bert 词库
        self.vocab_file = os.path.join(self.model_dir, 'vocab.txt')
        # 训练数据地址
        self.data_dir = os.path.join("", 'data/')
        # 训练 epochs
        self.num_train_epochs = 10
        # 训练 batch_size
        self.batch_size = 128
        self.learning_rate = 0.00005
```

```
        # gpu使用率
        self.gpu_memory_fraction = 0.8
        # 默认取倒数第二层的输出值作为句向量
        self.layer_indexes = [-2]
        # 序列的最大程度，单文本建议把该值调小
        self.max_seq_len = 32
```

三、特征实例类 （InputExample）

```
class InputExample(object):
    def __init__(self, unique_id, text_a, text_b):
        self.unique_id = unique_id
        self.text_a = text_a
        self.text_b = text_b

class InputFeatures(object):
    """A single set of features of data."""
    def __init__(self, unique_id, tokens, input_ids, input_mask,
input_type_ids):
        self.unique_id = unique_id
        self.tokens = tokens
        self.input_ids = input_ids
        self.input_mask = input_mask
        self.input_type_ids = input_type_ids
```

四、Bert 句向量 类 （BertVector）

这一个 是 生成 Bert 句向量的 类，流程：

1. 模型加载 （get_estimator）；
2. predict input 预处理 （queue_predict_input_fn）；
    1. 将 实例（examples） 转化为 特征（features） （convert_examples_to_features）；
3. encode sentence （encode）；

```
class BertVector:
    def __init__(self, batch_size=32):
        """
        init BertVector
        :param batch_size:     Depending on your memory default is 32
        """
        self.max_seq_length = args.max_seq_len
        self.layer_indexes = args.layer_indexes
        self.gpu_memory_fraction = 1
        self.graph_path = optimize_graph()
        self.tokenizer = tokenization.FullTokenizer(vocab_file=args.vocab_file,
do_lower_case=True)
        self.batch_size = batch_size
        #  获取 estimator
        self.estimator = self.get_estimator()
        # 输入 队列
        self.input_queue = Queue(maxsize=1)
        # 输出 队列
        self.output_queue = Queue(maxsize=1)
```

```
        # 预测 线程
        self.predict_thread = Thread(target=self.predict_from_queue,
daemon=True)
        self.predict_thread.start()
        self.sentence_len = 0
    #  获取 estimator
    def get_estimator(self):
        from tensorflow.python.estimator.estimator import Estimator
        from tensorflow.python.estimator.run_config import RunConfig
        from tensorflow.python.estimator.model_fn import EstimatorSpec

        def model_fn(features, labels, mode, params):
            with tf.gfile.GFile(self.graph_path, 'rb') as f:
                graph_def = tf.GraphDef()
                graph_def.ParseFromString(f.read())

            input_names = ['input_ids', 'input_mask', 'input_type_ids']

            output = tf.import_graph_def(
                graph_def,
                input_map={k + ':0': features[k] for k in
input_names},return_elements=['final_encodes:0']
            )
            return EstimatorSpec(mode=mode, predictions={
                'encodes': output[0]
            })
        # GPU 配置信息
        config = tf.ConfigProto()
        config.gpu_options.allow_growth = True
        config.gpu_options.per_process_gpu_memory_fraction =
self.gpu_memory_fraction
        config.log_device_placement = False
        config.graph_options.optimizer_options.global_jit_level =
tf.OptimizerOptions.ON_1
        return Estimator(
            model_fn=model_fn,
            config=RunConfig(session_config=config),
            params={'batch_size': self.batch_size}
        )
    # 预测
    def predict_from_queue(self):
        prediction = self.estimator.predict(input_fn=self.queue_predict_input_fn,
yield_single_examples=False)
        for i in prediction:
            self.output_queue.put(i)
    # encode sentence
    def encode(self, sentence):
        self.sentence_len = len(sentence)
        self.input_queue.put(sentence)
        prediction = self.output_queue.get()['encodes']
        return prediction
    # 预测 input 生成
    def queue_predict_input_fn(self):
        return (
            tf.data.Dataset.from_generator(
```

```
                self.generate_from_queue,
                output_types={
                    'unique_ids': tf.int32,
                    'input_ids': tf.int32,
                    'input_mask': tf.int32,
                    'input_type_ids': tf.int32
                },
                output_shapes={
                    'unique_ids': (self.sentence_len,),
                    'input_ids': (None, self.max_seq_length),
                    'input_mask': (None, self.max_seq_length),
                    'input_type_ids': (None, self.max_seq_length)
                }
            ).prefetch(10))

    def generate_from_queue(self):
        while True:
            features =
list(self.convert_examples_to_features(seq_length=self.max_seq_length,
tokenizer=self.tokenizer))
            yield {
                'unique_ids': [f.unique_id for f in features],
                'input_ids': [f.input_ids for f in features],
                'input_mask': [f.input_mask for f in features],
                'input_type_ids': [f.input_type_ids for f in features]
            }

    def input_fn_builder(self, features, seq_length):
        """Creates an `input_fn` closure to be passed to Estimator."""

        all_unique_ids = []
        all_input_ids = []
        all_input_mask = []
        all_input_type_ids = []

        for feature in features:
            all_unique_ids.append(feature.unique_id)
            all_input_ids.append(feature.input_ids)
            all_input_mask.append(feature.input_mask)
            all_input_type_ids.append(feature.input_type_ids)

        def input_fn(params):
            """The actual input function."""
            batch_size = params["batch_size"]

            num_examples = len(features)

            # This is for demo purposes and does NOT scale to large data sets. We
do
            # not use Dataset.from_generator() because that uses tf.py_func which
is
            # not TPU compatible. The right way to load data is with
TFRecordReader.
            d = tf.data.Dataset.from_tensor_slices({
                "unique_ids":
```

```
                    tf.constant(all_unique_ids, shape=[num_examples],
dtype=tf.int32),
                "input_ids":
                    tf.constant(
                        all_input_ids, shape=[num_examples, seq_length],
                        dtype=tf.int32),
                "input_mask":
                    tf.constant(
                        all_input_mask,
                        shape=[num_examples, seq_length],
                        dtype=tf.int32),
                "input_type_ids":
                    tf.constant(
                        all_input_type_ids,
                        shape=[num_examples, seq_length],
                        dtype=tf.int32),
            })

            d = d.batch(batch_size=batch_size, drop_remainder=False)
            return d

        return input_fn

    def model_fn_builder(self, bert_config, init_checkpoint, layer_indexes):
        """Returns `model_fn` closure for TPUEstimator."""

        def model_fn(features, labels, mode, params):  # pylint: disable=unused-
argument
            """The `model_fn` for TPUEstimator."""

            unique_ids = features["unique_ids"]
            input_ids = features["input_ids"]
            input_mask = features["input_mask"]
            input_type_ids = features["input_type_ids"]

            jit_scope = tf.contrib.compiler.jit.experimental_jit_scope

            with jit_scope():
                model = modeling.BertModel(
                    config=bert_config,
                    is_training=False,
                    input_ids=input_ids,
                    input_mask=input_mask,
                    token_type_ids=input_type_ids)

                if mode != tf.estimator.ModeKeys.PREDICT:
                    raise ValueError("Only PREDICT modes are supported: %s" %
(mode))

                tvars = tf.trainable_variables()

                (assignment_map, initialized_variable_names) =
modeling.get_assignment_map_from_checkpoint(tvars, init_checkpoint)

                tf.logging.info("**** Trainable Variables ****")
```

```
                for var in tvars:
                    init_string = ""
                    if var.name in initialized_variable_names:
                        init_string = ", *INIT_FROM_CKPT*"
                    tf.logging.info("  name = %s, shape = %s%s", var.name,
var.shape, init_string)

                all_layers = model.get_all_encoder_layers()

                predictions = {
                    "unique_id": unique_ids,
                }

                for (i, layer_index) in enumerate(layer_indexes):
                    predictions["layer_output_%d" % i] = all_layers[layer_index]

                from tensorflow.python.estimator.model_fn import EstimatorSpec

                output_spec = EstimatorSpec(mode=mode, predictions=predictions)
                return output_spec

        return model_fn

    def convert_examples_to_features(self, seq_length, tokenizer):
        """将数据文件加载到 “InputBatch” 队列中 【这一部分 之前介绍过】"""

        features = []
        input_masks = []
        examples = self._to_example(self.input_queue.get())
        for (ex_index, example) in enumerate(examples):
            tokens_a = tokenizer.tokenize(example.text_a)

            # 如果 句子 长度 大于 seq_len，只取 左边句子
            if len(tokens_a) > seq_length - 2:
                tokens_a = tokens_a[0:(seq_length - 2)]

            # The convention in BERT is:
            # (a) For sequence pairs:
            #  tokens:   [CLS] is this jack ##son ##ville ? [SEP] no it is not .
[SEP]
            #  type_ids: 0     0  0    0    0     0       0 0     1  1  1  1   1
1
            # (b) For single sequences:
            #  tokens:   [CLS] the dog is hairy . [SEP]
            #  type_ids: 0     0   0   0  0     0 0
            #
            # Where "type_ids" are used to indicate whether this is the first
sequence or the second sequence. The embedding vectors for `type=0` and `type=1`
were learned during pre-training and are added to the wordpiece embedding vector
(and position vector). This is not *strictly* necessary since the [SEP] token
unambiguously separates the sequences, but it makes it easier for the model to
learn the concept of sequences.
            #
            # For classification tasks, the first vector (corresponding to [CLS])
is used as as the "sentence vector". Note that this only makes sense because
```

```
            # the entire model is fine-tuned.
            tokens = []
            input_type_ids = []
            tokens.append("[CLS]")
            input_type_ids.append(0)
            for token in tokens_a:
                tokens.append(token)
                input_type_ids.append(0)
            tokens.append("[SEP]")
            input_type_ids.append(0)

            # Where "input_ids" are tokens's index in vocabulary
            input_ids = tokenizer.convert_tokens_to_ids(tokens)

            # The mask has 1 for real tokens and 0 for padding tokens. Only real
            # tokens are attended to.
            input_mask = [1] * len(input_ids)
            input_masks.append(input_mask)
            # Zero-pad up to the sequence length.
            while len(input_ids) < seq_length:
                input_ids.append(0)
                input_mask.append(0)
                input_type_ids.append(0)

            assert len(input_ids) == seq_length
            assert len(input_mask) == seq_length
            assert len(input_type_ids) == seq_length

            if ex_index < 5:
                tf.logging.info("*** Example ***")
                tf.logging.info("unique_id: %s" % (example.unique_id))
                tf.logging.info("tokens: %s" % " ".join(
                    [tokenization.printable_text(x) for x in tokens]))
                tf.logging.info("input_ids: %s" % " ".join([str(x) for x in
input_ids]))
                tf.logging.info("input_mask: %s" % " ".join([str(x) for x in
input_mask]))
                tf.logging.info(
                    "input_type_ids: %s" % " ".join([str(x) for x in
input_type_ids]))

            yield InputFeatures(
                unique_id=example.unique_id,
                tokens=tokens,
                input_ids=input_ids,
                input_mask=input_mask,
                input_type_ids=input_type_ids)

    def _truncate_seq_pair(self, tokens_a, tokens_b, max_length):
        """将序列对截断到最大长度"""

        # 这是一个简单的启发式方法，它总是一次一个 token 地截断较长的序列。这比从每个 token
中截取相等百分比的 token 更有意义，因为如果一个序列非常短，那么每个被截断的 token 可能包含比较
长序列更多的信息。
```

```
        # This is a simple heuristic which will always truncate the longer
sequence one token at a time. This makes more sense than truncating an equal
percent of tokens from each, since if one sequence is very short then each token
that's truncated likely contains more information than a longer sequence.
        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()

    # sentences 转 InputExamples
    @staticmethod
    def _to_example(sentences):
        import re
        """
        sentences to InputExample
        :param sentences: list of strings
        :return: list of InputExample
        """
        unique_id = 0
        for ss in sentences:
            line = tokenization.convert_to_unicode(ss)
            if not line:
                continue
            line = line.strip()
            text_a = None
            text_b = None
            m = re.match(r"^(.*) \|\|\| (.*)$", line)
            if m is None:
                text_a = line
            else:
                text_a = m.group(1)
                text_b = m.group(2)
            yield InputExample(unique_id=unique_id, text_a=text_a,
text_b=text_b)
            unique_id += 1
```

五、Bert 句向量 生成 实例

```
if __name__ == "__main__":
    bert = BertVector()
    # while True:
    #     question = input('question: ')
    vectors = bert.encode(['你好', '哈哈'])
    print(str(vectors))
>>>
INFO:tensorflow:*** Example ***
INFO:tensorflow:unique_id: 0
INFO:tensorflow:tokens: [CLS] 你 好 ， be ##rt ！ [SEP]
INFO:tensorflow:input_ids: 101 872 1962 8024 8815 8716 8013 102 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INFO:tensorflow:input_type_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INFO:tensorflow:*** Example ***
INFO:tensorflow:unique_id: 1
INFO:tensorflow:tokens: [CLS] 这 是 一 个 例 子 [SEP]
INFO:tensorflow:input_ids: 101 6821 3221 671 702 891 2094 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INFO:tensorflow:input_type_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[[ 0.77314675  0.00324035  0.35886183 ... -0.00801571  0.6570408
   0.3012028 ]
 [-0.26962122  0.49693802  0.33362615 ...  0.42230296  0.5397997
  -0.47371814]]
```

六、总结

本章 主要介绍了 利用 Bert 生成 句向量，代码比较简单。

参考资料

1. 【【关于Bert】 那些的你不知道的事】
2. 【Bert】论文
3. 【Bert】源码