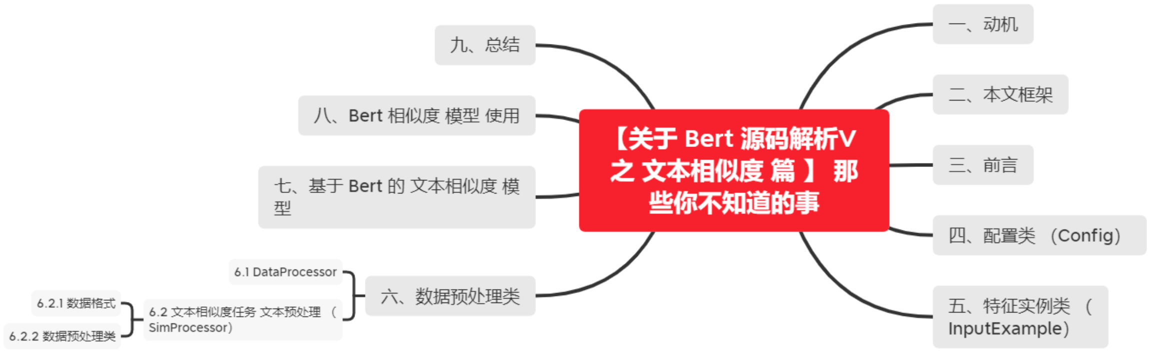
【关于 Bert 源码解析V 之 文本相似度 篇 】 那些你不知道的事


【关于 Bert 源码解析V 之 文本相似度 篇 】 那些你不知道的事
一、动机
二、本文框架
三、前言
四、配置类 （Config）
五、特征实例类 （InputExample）
六、数据预处理类
6.1 DataProcessor
6.2 文本相似度任务 文本预处理 （SimProcessor）
6.2.1 数据格式
6.2.2 数据预处理类
七、基于 Bert 的 文本相似度 模型
八、Bert 相似度 模型 使用
九、总结

一、前言

本文 主要 解读 Bert 模型的 微调 模块代码：

- similarity.py：主要用于 计算文本相似度

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）

这部分代码在《BERT源码解析-微调篇》有做过详细介绍，此处不展开重新介绍

```
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、数据预处理类

4.1 DataProcessor

```
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):
        """Gets a collection of `InputExample`s for prediction."""
        raise NotImplementedError()

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

4.2 文本相似度任务 文本预处理 （SimProcessor）

4.2.1 数据格式

```
query,reply,label
可以组合贷吗？,可以的,1
...
```

从上面实例中可以看出，每一行有query,reply,label组成，该任务主要是 判别 query 与 reply 是否存在关系，当 label 为 1 时，表示存在关系，为 0 时，表示无关系；

4.2.2 数据预处理类

```
class SimProcessor(DataProcessor):
    # 加载 训练数据
    def get_train_examples(self, data_dir):
        file_path = os.path.join(data_dir, 'train.csv')
        train_df = pd.read_csv(file_path, encoding='utf-8')
        train_data = []
        for index, train in enumerate(train_df.values):
            guid = 'train-%d' % index
            text_a = tokenization.convert_to_unicode(str(train[0]))
            text_b = tokenization.convert_to_unicode(str(train[1]))
            label = str(train[2])
            train_data.append(InputExample(guid=guid, text_a=text_a,
text_b=text_b, label=label))
        random.shuffle(train_data)
        return train_data
    # 加载 验证数据
    def get_dev_examples(self, data_dir):
        file_path = os.path.join(data_dir, 'dev.csv')
        dev_df = pd.read_csv(file_path, encoding='utf-8')
        dev_data = []
        for index, dev in enumerate(dev_df.values):
            guid = 'test-%d' % index
            text_a = tokenization.convert_to_unicode(str(dev[0]))
            text_b = tokenization.convert_to_unicode(str(dev[1]))
            label = str(dev[2])
            dev_data.append(InputExample(guid=guid, text_a=text_a, text_b=text_b,
label=label))
        random.shuffle(dev_data)
        return dev_data

    def get_test_examples(self, data_dir):
        file_path = os.path.join(data_dir, 'test.csv')
        test_df = pd.read_csv(file_path, encoding='utf-8')
        test_data = []
        for index, test in enumerate(test_df.values):
            test_data.append([str(test[0]),str(test[1])])
        return test_data

    def get_sentence_examples(self, questions):
        for index, data in enumerate(questions):
            guid = 'test-%d' % index
            text_a = tokenization.convert_to_unicode(str(data[0]))
            text_b = tokenization.convert_to_unicode(str(data[1]))
            label = str(0)
            yield InputExample(guid=guid, text_a=text_a, text_b=text_b,
label=label)
    # 获取标签
    def get_labels(self):
        return ['0', '1']
```

五、基于 Bert 的 文本相似度 模型

这部分代码，很多方法都未作修改， 与《BERT源码解析-微调篇》一样，

```
class BertSim:
    def __init__(self, batch_size=args.batch_size):
        self.mode = None
        self.max_seq_length = args.max_seq_len
        self.tokenizer = tokenization.FullTokenizer(vocab_file=args.vocab_file,
do_lower_case=True)
        self.batch_size = batch_size
        self.estimator = None
        self.processor = SimProcessor()
        tf.logging.set_verbosity(tf.logging.INFO)
    # 选择模式
    def set_mode(self, mode):
        self.mode = mode
        self.estimator = self.get_estimator()
        if mode == tf.estimator.ModeKeys.PREDICT:
            self.input_queue = Queue(maxsize=1)
            self.output_queue = Queue(maxsize=1)
            self.predict_thread = Thread(target=self.predict_from_queue,
daemon=True)
            self.predict_thread.start()
        if mode == "test":
            self.input_queue = Queue(maxsize=1)
            self.output_queue = Queue(maxsize=1)
            self.predict_thread = Thread(target=self.predict_from_queue,
daemon=True)
            self.predict_thread.start()
    # 构建模型
    @staticmethod
    def create_model(bert_config, is_training, input_ids, input_mask,
segment_ids,
                     labels, num_labels, use_one_hot_embeddings):
        """Creates a classification model."""
        model = modeling.BertModel(
            config=bert_config,
            is_training=is_training,
            input_ids=input_ids,
            input_mask=input_mask,
            token_type_ids=segment_ids,
            use_one_hot_embeddings=use_one_hot_embeddings)

        # In the demo, we are doing a simple classification task on the entire
        # segment.
        #
        # If you want to use the token-level output, use
model.get_sequence_output()
        # instead.
        output_layer = model.get_pooled_output()

        hidden_size = output_layer.shape[-1].value

        output_weights = tf.get_variable(
            "output_weights", [num_labels, hidden_size],
            initializer=tf.truncated_normal_initializer(stddev=0.02))

        output_bias = tf.get_variable(
            "output_bias", [num_labels], initializer=tf.zeros_initializer())

        with tf.variable_scope("loss"):
            if is_training:
                # I.e., 0.1 dropout
                output_layer = tf.nn.dropout(output_layer, keep_prob=0.9)

            logits = tf.matmul(output_layer, output_weights, transpose_b=True)
            logits = tf.nn.bias_add(logits, output_bias)
            probabilities = tf.nn.softmax(logits, axis=-1)
            log_probs = tf.nn.log_softmax(logits, axis=-1)

            one_hot_labels = tf.one_hot(labels, depth=num_labels,
dtype=tf.float32)

            per_example_loss = -tf.reduce_sum(one_hot_labels * log_probs,
axis=-1)
            loss = tf.reduce_mean(per_example_loss)

            return (loss, per_example_loss, logits, probabilities)

    def model_fn_builder(self, bert_config, num_labels, init_checkpoint,
learning_rate,
                         num_train_steps, num_warmup_steps,
                         use_one_hot_embeddings):
        """Returns `model_fn` closurimport_tfe for TPUEstimator."""

        def model_fn(features, labels, mode, params):  # pylint: disable=unused-
argument
            from tensorflow.python.estimator.model_fn import EstimatorSpec

            tf.logging.info("*** Features ***")
            for name in sorted(features.keys()):
                tf.logging.info("  name = %s, shape = %s" % (name,
features[name].shape))

            input_ids = features["input_ids"]
            input_mask = features["input_mask"]
            segment_ids = features["segment_ids"]
            label_ids = features["label_ids"]

            is_training = (mode == tf.estimator.ModeKeys.TRAIN)
            (total_loss, per_example_loss, logits, probabilities) =
BertSim.create_model(
                bert_config, is_training, input_ids, input_mask, segment_ids,
label_ids,
                num_labels, use_one_hot_embeddings)

            tvars = tf.trainable_variables()
            initialized_variable_names = {}

            if init_checkpoint:
                (assignment_map, initialized_variable_names) \
                    = modeling.get_assignment_map_from_checkpoint(tvars,
init_checkpoint)
                tf.train.init_from_checkpoint(init_checkpoint, assignment_map)

            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)

            if mode == tf.estimator.ModeKeys.TRAIN:

                train_op = optimization.create_optimizer(
                    total_loss, learning_rate, num_train_steps, num_warmup_steps,
False)

                output_spec = EstimatorSpec(
                    mode=mode,
                    loss=total_loss,
                    train_op=train_op)
            elif mode == tf.estimator.ModeKeys.EVAL:

                def metric_fn(per_example_loss, label_ids, logits):
                    predictions = tf.argmax(logits, axis=-1,
output_type=tf.int32)
                    accuracy = tf.metrics.accuracy(label_ids, predictions)
                    auc = tf.metrics.auc(label_ids, predictions)
                    loss = tf.metrics.mean(per_example_loss)
                    return {
                        "eval_accuracy": accuracy,
                        "eval_auc": auc,
                        "eval_loss": loss,
                    }

                eval_metrics = metric_fn(per_example_loss, label_ids, logits)
                output_spec = EstimatorSpec(
                    mode=mode,
                    loss=total_loss,
                    eval_metric_ops=eval_metrics)
            else:
                output_spec = EstimatorSpec(mode=mode,
predictions=probabilities)
            return output_spec
        return model_fn

    def get_estimator(self):

        from tensorflow.python.estimator.estimator import Estimator
        from tensorflow.python.estimator.run_config import RunConfig

        bert_config = modeling.BertConfig.from_json_file(args.config_name)
        label_list = self.processor.get_labels()
        train_examples = self.processor.get_train_examples(args.data_dir)
        num_train_steps = int(
            len(train_examples) / self.batch_size * args.num_train_epochs)
        num_warmup_steps = int(num_train_steps * 0.1)

        if self.mode == tf.estimator.ModeKeys.TRAIN:
            init_checkpoint = args.ckpt_name
        else:
            init_checkpoint = args.output_dir

        model_fn = self.model_fn_builder(
            bert_config=bert_config,
            num_labels=len(label_list),
            init_checkpoint=init_checkpoint,
            learning_rate=args.learning_rate,
            num_train_steps=num_train_steps,
            num_warmup_steps=num_warmup_steps,
            use_one_hot_embeddings=False)

        config = tf.ConfigProto()
        config.gpu_options.allow_growth = True
        config.gpu_options.per_process_gpu_memory_fraction =
args.gpu_memory_fraction
        config.log_device_placement = False

        return Estimator(model_fn=model_fn,
config=RunConfig(session_config=config), model_dir=args.output_dir,
                         params={'batch_size': self.batch_size})

    def predict_from_queue(self):
        for i in self.estimator.predict(input_fn=self.queue_predict_input_fn,
yield_single_examples=False):
            self.output_queue.put(i)

    def queue_predict_input_fn(self):
        return (tf.data.Dataset.from_generator(
            self.generate_from_queue,
            output_types={
                'input_ids': tf.int32,
                'input_mask': tf.int32,
                'segment_ids': tf.int32,
                'label_ids': tf.int32},
            output_shapes={
                'input_ids': (None, self.max_seq_length),
                'input_mask': (None, self.max_seq_length),
                'segment_ids': (None, self.max_seq_length),
                'label_ids': (1,)}).prefetch(10))

    def convert_examples_to_features(self, examples, label_list, max_seq_length,
tokenizer):
        """Convert a set of `InputExample`s to a list of `InputFeatures`."""

        for (ex_index, example) in enumerate(examples):
            label_map = {}
            for (i, label) in enumerate(label_list):
                label_map[label] = i

            tokens_a = tokenizer.tokenize(example.text_a)
            tokens_b = None
            if example.text_b:
                tokens_b = tokenizer.tokenize(example.text_b)

            if tokens_b:
                # Modifies `tokens_a` and `tokens_b` in place so that the total
                # length is less than the specified length.
                # Account for [CLS], [SEP], [SEP] with "- 3"
                self._truncate_seq_pair(tokens_a, tokens_b, max_seq_length - 3)
            else:
                # Account for [CLS] and [SEP] with "- 2"
                if len(tokens_a) > max_seq_length - 2:
                    tokens_a = tokens_a[0:(max_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 = []
            segment_ids = []
            tokens.append("[CLS]")
            segment_ids.append(0)
            for token in tokens_a:
                tokens.append(token)
                segment_ids.append(0)
            tokens.append("[SEP]")
            segment_ids.append(0)

            if tokens_b:
                for token in tokens_b:
                    tokens.append(token)
                    segment_ids.append(1)
                tokens.append("[SEP]")
                segment_ids.append(1)

            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)

            # Zero-pad up to the sequence length.
            while len(input_ids) < max_seq_length:
                input_ids.append(0)
                input_mask.append(0)
                segment_ids.append(0)

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

            label_id = label_map[example.label]
            if ex_index < 5 and self.mode!="test":
                tf.logging.info("*** Example ***")
                tf.logging.info("guid: %s" % (example.guid))
                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("segment_ids: %s" % " ".join([str(x) for x in
segment_ids]))
                tf.logging.info("label: %s (id = %d)" % (example.label,
label_id))

            feature = InputFeatures(
                input_ids=input_ids,
                input_mask=input_mask,
                segment_ids=segment_ids,
                label_id=label_id)

            yield feature
```

```
    def generate_from_queue(self):
        while True:
            predict_examples =
self.processor.get_sentence_examples(self.input_queue.get())
            features = list(self.convert_examples_to_features(predict_examples,
self.processor.get_labels(),
                                                              args.max_seq_len,
self.tokenizer))
            yield {
                'input_ids': [f.input_ids for f in features],
                'input_mask': [f.input_mask for f in features],
                'segment_ids': [f.segment_ids for f in features],
                'label_ids': [f.label_id for f in features]
            }

    def _truncate_seq_pair(self, tokens_a, tokens_b, max_length):
        """Truncates a sequence pair in place to the maximum length."""

        # 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()

    def convert_single_example(self, ex_index, example, label_list,
max_seq_length, tokenizer):
        """Converts a single `InputExample` into a single `InputFeatures`."""
        label_map = {}
        for (i, label) in enumerate(label_list):
            label_map[label] = i

        tokens_a = tokenizer.tokenize(example.text_a)
        tokens_b = None
        if example.text_b:
            tokens_b = tokenizer.tokenize(example.text_b)

        if tokens_b:
            # Modifies `tokens_a` and `tokens_b` in place so that the total
            # length is less than the specified length.
            # Account for [CLS], [SEP], [SEP] with "- 3"
            self._truncate_seq_pair(tokens_a, tokens_b, max_seq_length - 3)
        else:
            # Account for [CLS] and [SEP] with "- 2"
            if len(tokens_a) > max_seq_length - 2:
                tokens_a = tokens_a[0:(max_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 = []
        segment_ids = []
        tokens.append("[CLS]")
        segment_ids.append(0)
        for token in tokens_a:
            tokens.append(token)
            segment_ids.append(0)
        tokens.append("[SEP]")
        segment_ids.append(0)

        if tokens_b:
            for token in tokens_b:
                tokens.append(token)
                segment_ids.append(1)
            tokens.append("[SEP]")
            segment_ids.append(1)

        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)

        # Zero-pad up to the sequence length.
        while len(input_ids) < max_seq_length:
            input_ids.append(0)
            input_mask.append(0)
            segment_ids.append(0)
        assert len(input_ids) == max_seq_length
        assert len(input_mask) == max_seq_length
        assert len(segment_ids) == max_seq_length

        label_id = label_map[example.label]
        if ex_index < 5:
            tf.logging.info("*** Example ***")
            tf.logging.info("guid: %s" % (example.guid))
            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("segment_ids: %s" % " ".join([str(x) for x in
segment_ids]))
            tf.logging.info("label: %s (id = %d)" % (example.label, label_id))

        feature = InputFeatures(
            input_ids=input_ids,
            input_mask=input_mask,
            segment_ids=segment_ids,
            label_id=label_id)
        return feature

    def file_based_convert_examples_to_features(self, examples, label_list,
max_seq_length, tokenizer, output_file):
        """Convert a set of `InputExample`s to a TFRecord file."""

        writer = tf.python_io.TFRecordWriter(output_file)

        for (ex_index, example) in enumerate(examples):
            if ex_index % 10000 == 0:
                tf.logging.info("Writing example %d of %d" % (ex_index,
len(examples)))

            feature = self.convert_single_example(ex_index, example, label_list,
                                                  max_seq_length, tokenizer)

            def create_int_feature(values):
                f =
tf.train.Feature(int64_list=tf.train.Int64List(value=list(values)))
                return f

            features = collections.OrderedDict()
            features["input_ids"] = create_int_feature(feature.input_ids)
            features["input_mask"] = create_int_feature(feature.input_mask)
            features["segment_ids"] = create_int_feature(feature.segment_ids)
            features["label_ids"] = create_int_feature([feature.label_id])

            tf_example =
tf.train.Example(features=tf.train.Features(feature=features))
            writer.write(tf_example.SerializeToString())
    def file_based_input_fn_builder(self, input_file, seq_length, is_training,
drop_remainder):
        """Creates an `input_fn` closure to be passed to TPUEstimator."""

        name_to_features = {
            "input_ids": tf.FixedLenFeature([seq_length], tf.int64),
            "input_mask": tf.FixedLenFeature([seq_length], tf.int64),
            "segment_ids": tf.FixedLenFeature([seq_length], tf.int64),
            "label_ids": tf.FixedLenFeature([], tf.int64),
        }

        def _decode_record(record, name_to_features):
            """Decodes a record to a TensorFlow example."""
            example = tf.parse_single_example(record, name_to_features)

            # tf.Example only supports tf.int64, but the TPU only supports
tf.int32.
            # So cast all int64 to int32.
            for name in list(example.keys()):
                t = example[name]
                if t.dtype == tf.int64:
                    t = tf.to_int32(t)
                example[name] = t

            return example

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

            # For training, we want a lot of parallel reading and shuffling.
            # For eval, we want no shuffling and parallel reading doesn't
matter.
            d = tf.data.TFRecordDataset(input_file)
            if is_training:
                d = d.repeat()
                d = d.shuffle(buffer_size=100)

            d = d.apply(
                tf.contrib.data.map_and_batch(
                    lambda record: _decode_record(record, name_to_features),
                    batch_size=batch_size,
                    drop_remainder=drop_remainder))

            return d

        return input_fn

    def train(self):
        if self.mode is None:
            raise ValueError("Please set the 'mode' parameter")

        bert_config = modeling.BertConfig.from_json_file(args.config_name)

        if args.max_seq_len > bert_config.max_position_embeddings:
            raise ValueError(
                "Cannot use sequence length %d because the BERT model "
                "was only trained up to sequence length %d" %
                (args.max_seq_len, bert_config.max_position_embeddings))

        tf.gfile.MakeDirs(args.output_dir)

        label_list = self.processor.get_labels()

        train_examples = self.processor.get_train_examples(args.data_dir)
        num_train_steps = int(len(train_examples) / args.batch_size *
args.num_train_epochs)

        estimator = self.get_estimator()

        train_file = os.path.join(args.output_dir, "train.tf_record")
        self.file_based_convert_examples_to_features(train_examples, label_list,
args.max_seq_len, self.tokenizer,
                                                     train_file)
        tf.logging.info("***** Running training *****")
        tf.logging.info("  Num examples = %d", len(train_examples))
        tf.logging.info("  Batch size = %d", args.batch_size)
        tf.logging.info("  Num steps = %d", num_train_steps)
        train_input_fn = self.file_based_input_fn_builder(input_file=train_file,
seq_length=args.max_seq_len,
                                                          is_training=True,
                                                          drop_remainder=True)

        # early_stopping = tf.contrib.estimator.stop_if_no_decrease_hook(
        #     estimator,
        #     metric_name='loss',
        #     max_steps_without_decrease=10,
        #     min_steps=num_train_steps)

        # estimator.train(input_fn=train_input_fn, hooks=[early_stopping])
        estimator.train(input_fn=train_input_fn, max_steps=num_train_steps)

    def eval(self):
        if self.mode is None:
            raise ValueError("Please set the 'mode' parameter")
        eval_examples = self.processor.get_dev_examples(args.data_dir)
        eval_file = os.path.join(args.output_dir, "eval.tf_record")
        label_list = self.processor.get_labels()
        self.file_based_convert_examples_to_features(
            eval_examples, label_list, args.max_seq_len, self.tokenizer,
eval_file)

        tf.logging.info("***** Running evaluation *****")
        tf.logging.info("  Num examples = %d", len(eval_examples))
        tf.logging.info("  Batch size = %d", self.batch_size)

        eval_input_fn = self.file_based_input_fn_builder(
            input_file=eval_file,
            seq_length=args.max_seq_len,
            is_training=False,
            drop_remainder=False)

        estimator = self.get_estimator()
        result = estimator.evaluate(input_fn=eval_input_fn, steps=None)

        output_eval_file = os.path.join(args.output_dir, "eval_results.txt")
        with tf.gfile.GFile(output_eval_file, "w") as writer:
            tf.logging.info("***** Eval results *****")
            for key in sorted(result.keys()):
                tf.logging.info("  %s = %s", key, str(result[key]))
                writer.write("%s = %s\n" % (key, str(result[key])))

    def predict(self, sentence1, sentence2):
        if self.mode is None:
            raise ValueError("Please set the 'mode' parameter")
        self.input_queue.put([(sentence1, sentence2)])
        prediction = self.output_queue.get()
        return prediction

    def test(self):
        if self.mode is None:
            raise ValueError("Please set the 'mode' parameter")
        test_examples = self.processor.get_test_examples(args.data_dir)
        output_eval_file = os.path.join(args.output_dir, "test_results.txt")
        with tf.gfile.GFile(output_eval_file, "w") as writer:
            for sentenceItem in test_examples:
                self.input_queue.put([(sentenceItem[0], sentenceItem[1])])
                prediction = self.output_queue.get()
                writer.write("%s,%s,%s\n" % (sentenceItem[0],
sentenceItem[1],prediction[0][1]))
```

六、Bert 相似度 模型 使用

- 训练

```
    sim = BertSim()
    sim.set_mode(tf.estimator.ModeKeys.TRAIN)
    sim.train()
```

- 验证

```
    sim = BertSim()
    sim.set_mode(tf.estimator.ModeKeys.EVAL)
    sim.eval()
```

- 测试

```
    sim = BertSim()
    sim.set_mode("test")
    sim.test()
```

- 预测

```
sim = BertSim()
sim.set_mode(tf.estimator.ModeKeys.PREDICT)
while True:
    sentence1 = input('sentence1: ')
    sentence2 = input('sentence2: ')
    import time
    s = time.time()
    predict = sim.predict(sentence1, sentence2)
    print(time.time() - s)
    print(f'similarity：{predict[0][1]}')
```

七、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. 【Bert】论文
2. 【Bert】源码