Branch: master ▾ Multi-Label-Text-classification-Using-BERT / multi-label-classification-bert.ipynb Find file Copy path

javaidnabi31 Update multi-label-classification-bert.ipynb 3146021 on 8 Jun 2019

1 contributor

1370 lines (1370 sloc) 48 KB Raw Blame History

BERT for Multi-Label Classification

Refer post : https://medium.com/@javaid.nabi/building-a-multi-label-text-classifier-using-bert-and-tensorflow-f188e0ecdc5d (https://medium.com/@javaid.nabi/building-a-multi-label-text-classifier-using-bert-and-tensorflow-f188e0ecdc5d)

In [1]:
```
import os
import collections
import pandas as pd
import tensorflow as tf
import tensorflow_hub as hub
from datetime import datetime
```

```
WARNING: Logging before flag parsing goes to stderr.
W0511 13:58:07.529313  4008 __init__.py:56] Some hub symbols are no
t available because TensorFlow version is less than 1.14
```

In [3]:
```
##install bert if not already done
##!pip install bert-tensorflow
```

In [4]:
```
import bert
from bert import run_classifier
from bert import optimization
from bert import tokenization
from bert import modeling
```

In [5]:
```
##use downloaded model, change path accordingly
BERT_VOCAB= './uncased_L-12_H-768_A-12/vocab.txt'
BERT_INIT_CHKPNT = './uncased_L-12_H-768_A-12/bert_model.ckpt'
BERT_CONFIG = './uncased_L-12_H-768_A-12/bert_config.json'
```

In [6]:
```
tokenization.validate_case_matches_checkpoint(True,BERT_INIT_CHKPN
T)
tokenizer = tokenization.FullTokenizer(
      vocab_file=BERT_VOCAB, do_lower_case=True)
```

In [7]:
```
tokenizer.tokenize("This here's an example of using the BERT token
```

```
izer")
```

Out[7]:
```
['this',
 'here',
 "'",
 's',
 'an',
 'example',
 'of',
 'using',
 'the',
 'bert',
 'token',
 '##izer']
```

In [8]:
```
##change path accordingly
train_data_path='./Downloads/train.csv'
train = pd.read_csv(train_data_path)
test = pd.read_csv('./Downloads/test.csv')
```

In [9]:
```
train.head()
```

Out[9]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0
		You, sir, are my					

4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0

In [10]:
```
ID = 'id'
DATA_COLUMN = 'comment_text'
LABEL_COLUMNS = ['toxic','severe_toxic','obscene','threat','insul
t','identity_hate']
```

In [11]:
```
class InputExample(object):
    """A single training/test example for simple sequence classifi
cation."""

    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 sequ
ence. 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_i
ds, is_real_example=True):
        self.input_ids = input_ids
        self.input_mask = input_mask
```

```
        self.input_mask = input_mask
        self.segment_ids = segment_ids
        self.label_ids = label_ids,
        self.is_real_example=is_real_example
```

In [12]:
```
def create_examples(df, 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 = [0,0,0,0,0,0]
        examples.append(
            InputExample(guid=guid, text_a=text_a, labels=labels))
    return examples
```

In [13]:
```
TRAIN_VAL_RATIO = 0.9
LEN = train.shape[0]
SIZE_TRAIN = int(TRAIN_VAL_RATIO*LEN)

x_train = train[:SIZE_TRAIN]
x_val = train[SIZE_TRAIN:]

train_examples = create_examples(x_train)
```

In [15]:
```
def convert_examples_to_features(examples,  max_seq_length, tokenizer):
    """Loads a data file into a list of `InputBatch`s."""

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

        tokens_b = None
        if example.text_b:
            tokens_b = tokenizer.tokenize(example.text_b)
            # Modifies `tokens_a` and `tokens_b` in place so that
 the total
```

```
 the total
            # length is less than the specified length.
            # Account for [CLS], [SEP], [SEP] with "- 3"
            _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[:(max_seq_length - 2)]

        # The convention in BERT is:
        # (a) For sequence pairs:
        #  tokens:   [CLS] is this jack ##son ##ville ? [SEP] no i
t 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 th
e first
        # sequence or the second sequence. The embedding vectors f
or `type=0` and
        # `type=1` were learned during pre-training and are added
 to the wordpiece
        # embedding vector (and position vector). This is not *str
ictly* necessary
        # since the [SEP] token unambigiously separates the sequen
ces, but it makes
        # it easier for the model to learn the concept of sequence
s.
        #
        # For classification tasks, the first vector (correspondin
g to [CLS]) is
        # used as as the "sentence vector". Note that this only ma
kes sense because
        # the entire model is fine-tuned.
        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)

        # 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.
        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(int(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 i
n 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, l
abels_ids))

        features.append(
                InputFeatures(input_ids=input_ids,
                              input_mask=input_mask,
                              segment_ids=segment_ids,
```

```
                                    label_ids=labels_ids))
    return features
```

In [16]:
```
# We'll set sequences to be at most 128 tokens long.
MAX_SEQ_LENGTH = 128
```

In [17]:
```
# Compute train and warmup steps from batch size
# These hyperparameters are copied from this colab notebook (http
s://colab.sandbox.google.com/github/tensorflow/tpu/blob/master/too
ls/colab/bert_finetuning_with_cloud_tpus.ipynb)
BATCH_SIZE = 32
LEARNING_RATE = 2e-5
NUM_TRAIN_EPOCHS = 1.0
# Warmup is a period of time where hte learning rate
# is small and gradually increases--usually helps training.
WARMUP_PROPORTION = 0.1
# Model configs
SAVE_CHECKPOINTS_STEPS = 1000
SAVE_SUMMARY_STEPS = 500
```

In [18]:
```
class PaddingInputExample(object):
    """Fake example so the num input examples is a multiple of the
batch size.
    When running eval/predict on the TPU, we need to pad the numbe
r of examples
    to be a multiple of the batch size, because the TPU requires a
fixed batch
    size. The alternative is to drop the last batch, which is bad
 because it means
    the entire output data won't be generated.
    We use this class instead of `None` because treating `None` as
padding
    battches could cause silent errors.
    """


def convert_single_example(ex_index, example, max_seq_length,
                           tokenizer):
    """Converts a single `InputExample` into a single `InputFeatur
es`."""
```

```
    if isinstance(example, PaddingInputExample):
        return InputFeatures(
            input_ids=[0] * max_seq_length,
            input_mask=[0] * max_seq_length,
            segment_ids=[0] * max_seq_length,
            label_ids=0,
            is_real_example=False)

    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"
        _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 fi
rst
    # sequence or the second sequence. The embedding vectors for `
type=0` and
    # `type=1` were learned during pre-training and are added to t
he wordpiece
    # embedding vector (and position vector). This is not *strictl
y* necessary
    # since the [SEP] token unambiguously separates the sequences
```

```
    # 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 the "sentence vector". Note that this only makes sen
se 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. Onl
y 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
```

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


    feature = InputFeatures(
        input_ids=input_ids,
        input_mask=input_mask,
        segment_ids=segment_ids,
        label_ids=labels_ids,
        is_real_example=True)
    return feature


def file_based_convert_examples_to_features(
        examples, 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_inde
x, len(examples)))

        feature = convert_single_example(ex_index, example,
                                         max_seq_length, tokenizer
)

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

        features = collections.OrderedDict()
        features["input_ids"] = create_int_feature(feature.input_i
ds)
        features["input_mask"] = create_int_feature(feature.input_
mask)
        features["segment_ids"] = create_int_feature(feature.segme
nt_ids)
        features["is_real_example"] = create_int_feature(
```

```
            [int(feature.is_real_example)])
        if isinstance(feature.label_ids, list):
            label_ids = feature.label_ids
        else:
            label_ids = feature.label_ids[0]
        features["label_ids"] = create_int_feature(label_ids)

        tf_example = tf.train.Example(features=tf.train.Features(f
eature=features))
        writer.write(tf_example.SerializeToString())
    writer.close()


def file_based_input_fn_builder(input_file, seq_length, is_trainin
g,
                                drop_remainder):
    """Creates an `input_fn` closure to be passed to TPUEstimato
r."""

    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([6], tf.int64),
        "is_real_example": 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 supp
orts 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 shuf
fling.
        # For eval, we want no shuffling and parallel reading does
n'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_feat
ures),
                batch_size=batch_size,
                drop_remainder=drop_remainder))

        return d

    return input_fn


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

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

In [19]:
```
# Compute # train and warmup steps from batch size
num_train_steps = int(len(train_examples) / BATCH_SIZE * NUM_TRAIN
_EPOCHS)
num_warmup_steps = int(num_train_steps * WARMUP_PROPORTION)
```

In [20]:
```
train_file = os.path.join('./working', "train.tf_record")
#filename = Path(train_file)
if not os.path.exists(train_file):
    open(train_file, 'w').close()
```

In [21]:
```
file_based_convert_examples_to_features(
            train_examples, MAX_SEQ_LENGTH, tokenizer, train_file)
tf.logging.info("***** Running training *****")
tf.logging.info("  Num examples = %d", len(train_examples))
tf.logging.info("  Batch size = %d", BATCH_SIZE)
tf.logging.info("  Num steps = %d", num_train_steps)
```

```
INFO:tensorflow:***** Running training *****
I0511 14:08:54.477026  4008 tf_logging.py:115] ***** Running traini
ng *****
INFO:tensorflow:  Num examples = 10000
I0511 14:08:54.480028  4008 tf_logging.py:115]   Num examples = 100
00
INFO:tensorflow:  Batch size = 32
I0511 14:08:54.481029  4008 tf_logging.py:115]   Batch size = 32
INFO:tensorflow:  Num steps = 312
I0511 14:08:54.483028  4008 tf_logging.py:115]   Num steps = 312
```

In [22]:
```
train_input_fn = file_based_input_fn_builder(
    input_file=train_file,
    seq_length=MAX_SEQ_LENGTH,
    is_training=True,
    drop_remainder=True)
```

In [23]:
```
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) ### multi class case
```

```
class case
        probabilities = tf.nn.sigmoid(logits)#### multi-label case

        labels = tf.cast(labels, tf.float32)
        tf.logging.info("num_labels:{};logits:{};labels:{}".format
(num_labels, logits, labels))
        per_example_loss = tf.nn.sigmoid_cross_entropy_with_logits
(labels=labels, logits=logits)
        loss = tf.reduce_mean(per_example_loss)

        # 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, dt
ype=tf.float32)
        #
        # per_example_loss = -tf.reduce_sum(one_hot_labels * log_p
robs, axis=-1)
        # loss = tf.reduce_mean(per_example_loss)

        return (loss, per_example_loss, logits, probabilities)


def model_fn_builder(bert_config, num_labels, init_checkpoint, lea
rning_rate,
                     num_train_steps, num_warmup_steps, use_tpu,
                     use_one_hot_embeddings):
    """Returns `model_fn` closure for TPUEstimator."""

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

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

        input_ids = features["input_ids"]
        input_mask = features["input_mask"]
        segment_ids = features["segment_ids"]
        label_ids = features["label_ids"]
        is_real_example = None
```

```
        if "is_real_example" in features:
             is_real_example = tf.cast(features["is_real_example"
], dtype=tf.float32)
        else:
             is_real_example = tf.ones(tf.shape(label_ids), dtype=
tf.float32)

        is_training = (mode == tf.estimator.ModeKeys.TRAIN)

        (total_loss, per_example_loss, logits, probabilities) = cr
eate_model(
            bert_config, is_training, input_ids, input_mask, segme
nt_ids, label_ids,
            num_labels, use_one_hot_embeddings)

        tvars = tf.trainable_variables()
        initialized_variable_names = {}
        scaffold_fn = None
        if init_checkpoint:
            (assignment_map, initialized_variable_names
             ) = modeling.get_assignment_map_from_checkpoint(tvars
, init_checkpoint)
            if use_tpu:

                def tpu_scaffold():
                    tf.train.init_from_checkpoint(init_checkpoint,
assignment_map)
                    return tf.train.Scaffold()

                scaffold_fn = tpu_scaffold
            else:
                tf.train.init_from_checkpoint(init_checkpoint, ass
ignment_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.nam
e, var.shape,init_string)
```

```
        output_spec = None
        if mode == tf.estimator.ModeKeys.TRAIN:

            train_op = optimization.create_optimizer(
                total_loss, learning_rate, num_train_steps, num_wa
rmup_steps, use_tpu)

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

            def metric_fn(per_example_loss, label_ids, probabiliti
es, is_real_example):

                logits_split = tf.split(probabilities, num_labels,
axis=-1)
                label_ids_split = tf.split(label_ids, num_labels,
axis=-1)
                # metrics change to auc of every class
                eval_dict = {}
                for j, logits in enumerate(logits_split):
                    label_id_ = tf.cast(label_ids_split[j], dtype=
tf.int32)
                    current_auc, update_op_auc = tf.metrics.auc(la
bel_id_, logits)
                    eval_dict[str(j)] = (current_auc, update_op_au
c)
                eval_dict['eval_loss'] = tf.metrics.mean(values=pe
r_example_loss)
                return eval_dict

                ## original eval metrics
                # predictions = tf.argmax(logits, axis=-1, output_
type=tf.int32)
                # accuracy = tf.metrics.accuracy(
                #     labels=label_ids, predictions=predictions, w
eights=is_real_example)
                # loss = tf.metrics.mean(values=per_example_loss,
 weights=is_real_example)
                # return {
```

```
                    # return {
                    #     "eval_accuracy": accuracy,
                    #     "eval_loss": loss,
                    # }

                eval_metrics = metric_fn(per_example_loss, label_ids,
probabilities, is_real_example)
                output_spec = tf.estimator.EstimatorSpec(
                    mode=mode,
                    loss=total_loss,
                    eval_metric_ops=eval_metrics,
                    scaffold=scaffold_fn)
            else:
                print("mode:", mode,"probabilities:", probabilities)
                output_spec = tf.estimator.EstimatorSpec(
                    mode=mode,
                    predictions={"probabilities": probabilities},
                    scaffold=scaffold_fn)
            return output_spec

        return model_fn
```

In [ ]:
```
OUTPUT_DIR = "./working/output"
# Specify outpit directory and number of checkpoint steps to save
run_config = tf.estimator.RunConfig(
    model_dir=OUTPUT_DIR,
    save_summary_steps=SAVE_SUMMARY_STEPS,
    keep_checkpoint_max=1,
    save_checkpoints_steps=SAVE_CHECKPOINTS_STEPS)
```

In [ ]:
```
bert_config = modeling.BertConfig.from_json_file(BERT_CONFIG)
model_fn = model_fn_builder(
  bert_config=bert_config,
  num_labels= len(LABEL_COLUMNS),
  init_checkpoint=BERT_INIT_CHKPNT,
  learning_rate=LEARNING_RATE,
  num_train_steps=num_train_steps,
  num_warmup_steps=num_warmup_steps,
  use_tpu=False,
  use_one_hot_embeddings=False)

estimator = tf.estimator.Estimator(
```

```
  model_fn=model_fn,
  config=run_config,
  params={"batch_size": BATCH_SIZE})
```

```
INFO:tensorflow:Using config: {'_model_dir': './working/output', '_
tf_random_seed': None, '_save_summary_steps': 500, '_save_checkpoin
ts_steps': 1000, '_save_checkpoints_secs': None, '_session_config':
None, '_keep_checkpoint_max': 1, '_keep_checkpoint_every_n_hours':
10000, '_log_step_count_steps': 100, '_train_distribute': None, '_d
evice_fn': None, '_service': None, '_cluster_spec': <tensorflow.pyt
hon.training.server_lib.ClusterSpec object at 0x0000026C1A74AEF0>,
'_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster': 0,
'_master': '', '_evaluation_master': '', '_is_chief': True, '_num_p
s_replicas': 0, '_num_worker_replicas': 1}
```

```
I0511 14:08:54.546028  4008 tf_logging.py:115] Using config: {'_mod
el_dir': './working/output', '_tf_random_seed': None, '_save_summar
y_steps': 500, '_save_checkpoints_steps': 1000, '_save_checkpoints_
secs': None, '_session_config': None, '_keep_checkpoint_max': 1, '_
keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 10
0, '_train_distribute': None, '_device_fn': None, '_service': None,
'_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec
object at 0x0000026C1A74AEF0>, '_task_type': 'worker', '_task_id':
0, '_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replica
s': 1}
```

In [ ]:
```
print(f'Beginning Training!')
current_time = datetime.now()
estimator.train(input_fn=train_input_fn, max_steps=num_train_steps
)
print("Training took time ", datetime.now() - current_time)
```

```
Beginning Training!
INFO:tensorflow:Calling model_fn.
```

```
I0511 14:09:03.337898  4008 tf_logging.py:115] Calling model_fn.
```

```
INFO:tensorflow:num_labels:6;logits:Tensor("loss/BiasAdd:0", shape=
(32, 6), dtype=float32);labels:Tensor("loss/Cast:0", shape=(32, 6),
dtype=float32)
```

```
I0511 14:09:06.024065  4008 tf_logging.py:115] num_labels:6;logits:
Tensor("loss/BiasAdd:0", shape=(32, 6), dtype=float32);labels:Tenso
r("loss/Cast:0", shape=(32, 6), dtype=float32)
```

```
INFO:tensorflow:**** Trainable Variables ****

I0511 14:09:06.839105  4008 tf_logging.py:115] **** Trainable Varia
bles ****

INFO:tensorflow:Done calling model_fn.

I0511 14:09:16.661103  4008 tf_logging.py:115] Done calling model_f
n.

INFO:tensorflow:Create CheckpointSaverHook.

I0511 14:09:16.666073  4008 tf_logging.py:115] Create CheckpointSav
erHook.

INFO:tensorflow:Graph was finalized.

I0511 14:09:35.468607  4008 tf_logging.py:115] Graph was finalized.

INFO:tensorflow:Restoring parameters from ./working/output\model.ck
pt-0
```