BERT

November 18, 2024

1 The Annotated BERT: Bidirectional Encoder Representations from Transformers

BERT (Bidirectional Encoder Representations from Transformers) has transformed the NLP landscape by introducing a bidirectional approach that effectively captures language context. Unlike traditional models that process text unidirectionally, BERT employs a deep bidirectional architecture based on Transformers, enabling it to jointly condition on both the left and right context in all layers. This innovation allows BERT to achieve state-of-the-art results across a wide range of natural language processing tasks, including question answering, natural language inference, and named entity recognition.

The key advancements in BERT include the use of a masked language model (MLM) objective, which pre-trains the model by predicting randomly masked words in a sentence, and the next sentence prediction (NSP) task, which helps in understanding relationships between sentences. By pre-training on large corpora such as BooksCorpus and English Wikipedia, BERT learns robust language representations that can be fine-tuned with minimal task-specific modifications. This paper explores the architecture, pre-training methodology, and fine-tuning processes that make BERT a cornerstone of modern NLP research and applications.

2 Table of Contents

3 Introduction

BERT (Bidirectional Encoder Representations from Transformers) has revolutionized the field of Natural Language Processing (NLP) by providing a pre-trained model capable of understanding the context of words in a sentence from both directions (left-to-right and right-to-left). Unlike traditional NLP models, which process text in a unidirectional manner, BERT utilizes the Transformer architecture to create bidirectional representations, making it more powerful in capturing the nuances of language.

BERT's success stems from its innovative pre-training tasks—**Masked Language Modeling (MLM)** and **Next Sentence Prediction (NSP)**—which allow the model to learn rich, deep contextual embeddings from a large corpus of text. After pre-training on vast amounts of unlabelled text, BERT can be fine-tuned for a wide range of NLP tasks, such as question answering, sentiment analysis, and named entity recognition, by simply adding a task-specific layer and fine-tuning on labeled data.

This notebook delves into the key concepts behind BERT's architecture, training methods, and practical applications, with code implementations that demonstrate how BERT can be used to solve real-world NLP challenges.

4 Preliminaries

In this notebook, we will explore the BERT (Bidirectional Encoder Representations from Transformers) model, a breakthrough in Natural Language Processing (NLP) that leverages the Transformer architecture to produce contextualized word embeddings. BERT is pre-trained on a large corpus of text and fine-tuned for downstream tasks such as question answering, sentiment analysis, and named entity recognition.

To implement BERT and explore its architecture, we will be using the following Python libraries:

1. **Transformers**: A popular library by Hugging Face that provides pre-trained models and tokenizers for state-of-the-art NLP architectures, including BERT.
2. **Torch**: A deep learning framework used to run the BERT model and perform tensor computations efficiently on both CPU and GPU.

[ ]:
```
# imports

from transformers import BertTokenizer, BertModel, BertForMaskedLM,␣
↪BertForNextSentencePrediction, pipeline
```

```
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import warnings
import collections
import csv
import os
import modeling
import optimization
import tokenization
import tensorflow as tf
import collections
import copy
import json
import math
import re
import six
warnings.filterwarnings("ignore")
```

4.0.1 Background

The field of Natural Language Processing (NLP) has made significant progress over the past decade, largely driven by the development of deep learning models. Prior to the advent of transformer-based models, NLP systems were heavily reliant on traditional models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs), which were sequential in nature. While these models were effective for many tasks, they struggled with handling long-range dependencies and parallelization. The breakthrough came with the introduction of the Transformer model by Vaswani et al. in 2017 in the paper "Attention Is All You Need." This model, based on self-attention mechanisms, was capable of processing entire sequences in parallel, making it more efficient and scalable compared to RNNs and LSTMs.

BERT (Bidirectional Encoder Representations from Transformers), introduced by Devlin et al. in 2018, represents a significant advancement in this paradigm. Unlike previous transformer models, which were either unidirectional (left-to-right or right-to-left), BERT leverages a bidirectional approach. This means that it considers context from both the left and right of a token during training, allowing for a deeper understanding of word meaning based on its surrounding context. BERT's architecture is pre-trained on vast amounts of text data using two objectives: **Masked Language Modeling (MLM)** and **Next Sentence Prediction (NSP)**. These pre-training tasks enable BERT to capture rich contextual representations of words and their relationships in a sentence, setting it apart from earlier models.

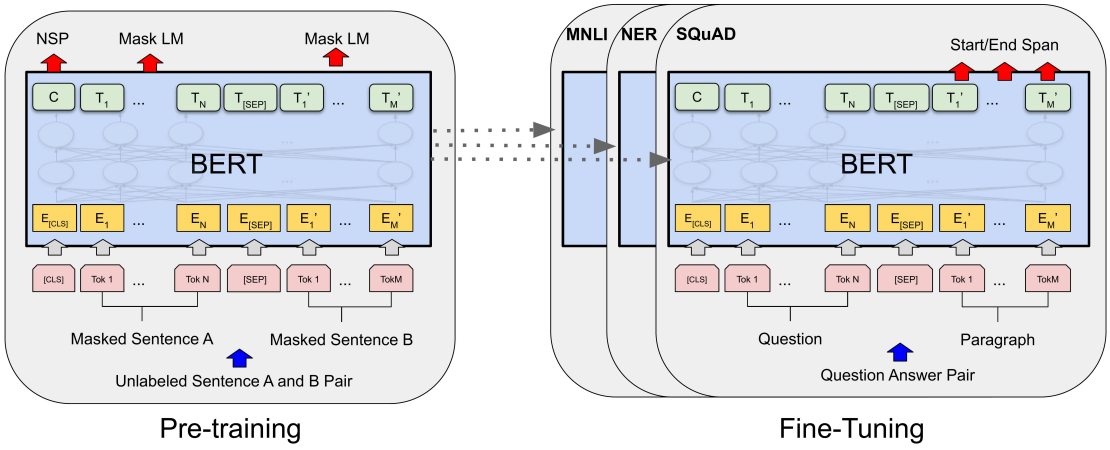
The impact of BERT on the NLP community has been profound. It achieved state-of-the-art results across a wide range of benchmarks, including question answering, sentiment analysis, and

named entity recognition, among others. BERT's pre-training approach allows it to be fine-tuned on downstream tasks with relatively small datasets, making it highly versatile for various NLP applications. Additionally, BERT has inspired several model variants, including RoBERTa, ALBERT, and DistilBERT, which build upon and optimize its architecture.

With the rise of transformer-based models like BERT, the landscape of NLP research and applications has shifted towards pre-trained models, enabling researchers and developers to fine-tune a single model for a wide range of specific tasks. This approach has significantly reduced the barriers to entry for building state-of-the-art NLP systems, democratizing access to powerful language models.

5 Model Architecture

BERT (Bidirectional Encoder Representations from Transformers) is based on the Transformer architecture, specifically utilizing the **encoder stack**. Unlike traditional models that process text sequentially (e.g., RNNs or LSTMs), BERT leverages **self-attention mechanisms** that allow it to consider the relationships between all words in a sentence simultaneously, capturing long-range dependencies more efficiently. The bidirectional nature of BERT means that, unlike earlier models which only process text in a left-to-right or right-to-left manner, BERT takes both the left and right context into account during training. This results in richer and more accurate contextual embeddings for words. The Transformer encoder consists of multiple layers of attention heads, followed by position-wise feed-forward networks, enabling the model to learn complex relationships and representations. BERT uses **positional encodings** to retain the order of words in the sentence, which is essential for understanding the sequence in which the words appear.



5.1 Overview of BERT

BERT (Bidirectional Encoder Representations from Transformers) is a novel language representation model designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. Unlike traditional language models, which process text either left-to-right or right-to-left, BERT employs a "masked language model" (MLM) objective to learn bidirectional representations, which allows it to outperform state-of-the-art models in various natural language processing (NLP) tasks.

5.2 Key Design Choices

BERT uses a unified architecture across different tasks, enabling minimal task-specific adjustments during fine-tuning. The model architecture is a multi-layer bidirectional Transformer encoder based on the implementation of Vaswani et al. (2017). The bidirectional self-attention mechanism in BERT allows it to better capture context compared to unidirectional models like OpenAI GPT.

5.3 Input Representation

The input representation in BERT can unambiguously handle single sentences or sentence pairs in a single token sequence. This representation combines token embeddings, segment embeddings, and positional embeddings.

- **Token Embeddings**: The model uses WordPiece embeddings with a vocabulary of 30,000 tokens. Each token in the input sequence is converted into a fixed-size vector representing its semantic and contextual meaning.
- **Sentence Pair Representation**: To handle tasks involving sentence pairs (e.g., question-answering), BERT concatenates the sentences into a single input sequence. The sentences are separated by a special token `[SEP]`. The first token of every sequence is a special classification token `[CLS]`, whose final hidden state is used for downstream classification tasks. Learned embeddings are added to indicate whether tokens belong to sentence A or sentence B.
- **Positional and Segment Embeddings**: BERT incorporates positional embeddings to capture the position of each token in the sequence and segment embeddings to differentiate tokens belonging to different sentences in a sentence pair.

The input representation process integrates components from `modeling.py`, `tokenization.py`, and task-specific scripts like `run_classifier.py` and `run_squad.py`. `modeling.py` defines the model architecture, including token, segment, and position embeddings, while `tokenization.py` handles the conversion of raw text into WordPiece tokens. In `run_classifier.py` and `run_squad.py`, input processing is managed, specifically for single and paired sentences, ensuring that data is properly tokenized and formatted for different NLP tasks such as classification and question answering.

[ ]:
```python
# BertModel: Core BERT model class with embeddings and transformer layers

class BertModel(object):
    """BERT model with Token, Segment, and Position Embeddings."""

    def __init__(self, config, is_training, input_ids, input_mask=None,␣
 ↪token_type_ids=None, use_one_hot_embeddings=False):
        config = copy.deepcopy(config)
        if not is_training:
            config.hidden_dropout_prob = 0.0
            config.attention_probs_dropout_prob = 0.0

        self.embedding_output, self.embedding_table = embedding_lookup(
```

```
            input_ids, config.vocab_size, config.hidden_size,␣
↪use_one_hot_embeddings=use_one_hot_embeddings)
        self.embedding_output = embedding_postprocessor(
            self.embedding_output, token_type_ids, config.type_vocab_size)

        attention_mask = create_attention_mask_from_input_mask(input_ids,␣
↪input_mask)

        self.all_encoder_layers = transformer_model(
            input_tensor=self.embedding_output,
            attention_mask=attention_mask,
            hidden_size=config.hidden_size,
            num_hidden_layers=config.num_hidden_layers,
            num_attention_heads=config.num_attention_heads,
            intermediate_size=config.intermediate_size,
            hidden_dropout_prob=config.hidden_dropout_prob,
            attention_probs_dropout_prob=config.attention_probs_dropout_prob
        )
        self.sequence_output = self.all_encoder_layers[-1]

        first_token_tensor = tf.squeeze(self.sequence_output[:, 0:1, :], axis=1)
        self.pooled_output = tf.layers.dense(
            first_token_tensor, config.hidden_size, activation=tf.tanh)

    def get_pooled_output(self):
        return self.pooled_output

    def get_sequence_output(self):
        return self.sequence_output

    def get_embedding_output(self):
        return self.embedding_output

    def get_embedding_table(self):
        return self.embedding_table
```

The `FullTokenizer` combines the `BasicTokenizer` and `WordpieceTokenizer` for complete tokenization. The `BasicTokenizer` handles lowercasing, cleaning, and whitespace splitting, while the `WordpieceTokenizer` splits words into subwords based on a vocabulary, using `[UNK]` for out-of-vocabulary tokens. This process ensures efficient text tokenization for NLP tasks, addressing both basic and subword tokenization needs seamlessly.

[ ]:
```
class FullTokenizer:
    """Combines Basic and WordPiece tokenization."""
    def __init__(self, vocab):
        self.vocab = vocab
        self.basic_tokenizer = BasicTokenizer()
```

```
        self.wordpiece_tokenizer = WordpieceTokenizer(vocab)

    def tokenize(self, text):
        tokens = self.basic_tokenizer.tokenize(text)
        return [sub_token for token in tokens for sub_token in self.
 ↪wordpiece_tokenizer.tokenize(token)]

class BasicTokenizer:
    """Basic tokenization for text preprocessing."""
    def tokenize(self, text):
        text = convert_to_unicode(text).lower()
        text = clean_text(text)
        return whitespace_tokenize(text)

class WordpieceTokenizer:
    """Handles WordPiece tokenization."""
    def __init__(self, vocab, unk_token="[UNK]", max_input_chars_per_word=200):
        self.vocab = vocab
        self.unk_token = unk_token
        self.max_input_chars_per_word = max_input_chars_per_word

    def tokenize(self, text):
        output_tokens = []
        for token in whitespace_tokenize(text):
            chars = list(token)
            if len(chars) > self.max_input_chars_per_word:
                output_tokens.append(self.unk_token)
                continue
            sub_tokens, start = [], 0
            while start < len(chars):
                end, cur_substr = len(chars), None
                while start < end:
                    substr = "##" + "".join(chars[start:end]) if start > 0 else␣
 ↪"".join(chars[start:end])
                    if substr in self.vocab:
                        cur_substr = substr
                        break
                    end -= 1
                if cur_substr:
                    sub_tokens.append(cur_substr)
                    start = end
                else:
                    output_tokens.append(self.unk_token)
                    break
            output_tokens.extend(sub_tokens)
        return output_tokens
```

5.4 Pre-training Objectives

BERT employs two novel pre-training objectives to learn bidirectional representations: Masked Language Model (MLM) and Next Sentence Prediction (NSP). The MLM task involves randomly masking 15% of the tokens in each input sequence, and the model is then trained to predict these masked tokens based on the surrounding context. This approach allows BERT to leverage context from both left and right sides of each token, unlike traditional unidirectional language models. The NSP task, on the other hand, is designed to improve BERT's understanding of sentence relationships. In this task, pairs of sentences are presented to the model, and it must predict whether the second sentence follows the first in the original text. These two objectives together allow BERT to capture both token-level and sentence-level information, providing a more comprehensive understanding of language.

BERT uses two unsupervised objectives during pre-training:

1. **Masked Language Modeling (MLM)**: Randomly masks 15% of tokens in the input and predicts them using the context from both sides. This task enables deep bidirectional representations by avoiding the constraints of traditional left-to-right or right-to-left language models.
2. **Next Sentence Prediction (NSP)**: This task helps the model understand the relationship between two sentences. For a given sentence pair, 50% of the time the second sentence is the actual next sentence, and 50% of the time it is a random sentence from the corpus. The model predicts whether the second sentence logically follows the first.

[ ]:
```
def create_masked_lm_predictions(tokens, masked_lm_prob,␣
↪max_predictions_per_seq, vocab_words, rng):
    """Creates the predictions for the masked LM objective."""

    cand_indexes = []
    for (i, token) in enumerate(tokens):
        if token != "[CLS]" and token != "[SEP]":
            cand_indexes.append(i)

    rng.shuffle(cand_indexes)
    num_to_mask = min(max_predictions_per_seq, int(round(len(cand_indexes) *␣
↪masked_lm_prob)))
    masked_lm_positions = []
    masked_lm_labels = []

    for i in range(num_to_mask):
        masked_index = cand_indexes[i]
        masked_lm_positions.append(masked_index)

        masked_token = "[MASK]"
        original_token = tokens[masked_index]

        if rng.random() < 0.8:
            tokens[masked_index] = masked_token
```

```
        elif rng.random() < 0.5:
            random_word = vocab_words[rng.randint(0, len(vocab_words) - 1)]
            tokens[masked_index] = random_word
        else:
            pass

        masked_lm_labels.append(original_token)

    return tokens, masked_lm_positions, masked_lm_labels


def truncate_seq_pair(tokens_a, tokens_b, max_num_tokens, rng):
    """Truncate a pair of sequences to a maximum length."""
    while True:
        total_length = len(tokens_a) + len(tokens_b)
        if total_length <= max_num_tokens:
            break
        if len(tokens_a) > len(tokens_b):
            tokens_a.pop()
        else:
            tokens_b.pop()


def main():
    rng = random.Random(FLAGS.random_seed)
    tokenizer = tokenization.FullTokenizer(vocab_file=FLAGS.vocab_file,␣
 ↪do_lower_case=FLAGS.do_lower_case)
    input_files = FLAGS.input_file.split(",")
    instances = create_training_instances(input_files, tokenizer, FLAGS.
 ↪max_seq_length,
                                          FLAGS.dupe_factor, FLAGS.
 ↪short_seq_prob,
                                          FLAGS.masked_lm_prob, FLAGS.
 ↪max_predictions_per_seq, rng)
    output_files = FLAGS.output_file.split(",")
    write_instance_to_example_files(instances, tokenizer, FLAGS.max_seq_length,
                                    FLAGS.max_predictions_per_seq, output_files)


if __name__ == "__main__":
    tf.app.run()
```

[ ]:
```
class BERTPreTraining:
    def __init__(self, bert_config, init_checkpoint, learning_rate,␣
 ↪num_train_steps,
                 num_warmup_steps, use_tpu, use_one_hot_embeddings):
        self.bert_config = bert_config
```

```
        self.init_checkpoint = init_checkpoint
        self.learning_rate = learning_rate
        self.num_train_steps = num_train_steps
        self.num_warmup_steps = num_warmup_steps
        self.use_tpu = use_tpu
        self.use_one_hot_embeddings = use_one_hot_embeddings

    def model_fn_builder(self):
        """Returns model_fn closure for TPUEstimator."""
        def model_fn(features, labels, mode, params):
            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"]
            masked_lm_positions = features["masked_lm_positions"]
            masked_lm_ids = features["masked_lm_ids"]
            masked_lm_weights = features["masked_lm_weights"]
            next_sentence_labels = features["next_sentence_labels"]

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

            model = modeling.BertModel(
                config=self.bert_config,
                is_training=is_training,
                input_ids=input_ids,
                input_mask=input_mask,
                token_type_ids=segment_ids,
                use_one_hot_embeddings=self.use_one_hot_embeddings)

            masked_lm_loss, _, _ = self.get_masked_lm_output(
                model.get_sequence_output(), masked_lm_positions, masked_lm_ids,␣
↪masked_lm_weights)
            next_sentence_loss, _, _ = self.get_next_sentence_output(
                model.get_pooled_output(), next_sentence_labels)

            total_loss = masked_lm_loss + next_sentence_loss
            tvars = tf.trainable_variables()

            initialized_variable_names = {}
            scaffold_fn = None
            if self.init_checkpoint:
                assignment_map, initialized_variable_names = modeling.
↪get_assignment_map_from_checkpoint(
```

```
                    tvars, self.init_checkpoint)
                if self.use_tpu:
                    def tpu_scaffold():
                        tf.train.init_from_checkpoint(self.init_checkpoint,␣
↪assignment_map)
                        return tf.train.Scaffold()
                    scaffold_fn = tpu_scaffold
                else:
                    tf.train.init_from_checkpoint(self.init_checkpoint,␣
↪assignment_map)

            if mode == tf.estimator.ModeKeys.TRAIN:
                train_op = optimization.create_optimizer(
                    total_loss, self.learning_rate, self.num_train_steps, self.
↪num_warmup_steps, self.use_tpu)
                return tf.contrib.tpu.TPUEstimatorSpec(mode=mode,␣
↪loss=total_loss, train_op=train_op, scaffold_fn=scaffold_fn)
            elif mode == tf.estimator.ModeKeys.EVAL:
                eval_metrics = self.get_eval_metrics(masked_lm_loss,␣
↪masked_lm_ids, masked_lm_weights, next_sentence_loss)
                return tf.contrib.tpu.TPUEstimatorSpec(mode=mode,␣
↪loss=total_loss, eval_metrics=eval_metrics, scaffold_fn=scaffold_fn)
            else:
                raise ValueError("Only TRAIN and EVAL modes are supported")

        return model_fn

    def get_masked_lm_output(self, input_tensor, positions, label_ids,␣
↪label_weights):
        """Get loss and log probs for the masked LM."""
        input_tensor = self.gather_indexes(input_tensor, positions)
        with tf.variable_scope("cls/predictions"):
            input_tensor = tf.layers.dense(
                input_tensor,
                units=self.bert_config.hidden_size,
                activation=modeling.get_activation(self.bert_config.hidden_act),
                kernel_initializer=modeling.create_initializer(self.bert_config.
↪initializer_range))
            input_tensor = modeling.layer_norm(input_tensor)
            output_bias = tf.get_variable("output_bias", shape=[self.bert_config.
↪vocab_size], initializer=tf.zeros_initializer())
            logits = tf.matmul(input_tensor, self.bert_config.embedding_table,␣
↪transpose_b=True)
            logits = tf.nn.bias_add(logits, output_bias)
            log_probs = tf.nn.log_softmax(logits, axis=-1)
```

```
            label_ids = tf.reshape(label_ids, [-1])
            label_weights = tf.reshape(label_weights, [-1])
            one_hot_labels = tf.one_hot(label_ids, depth=self.bert_config.
↪vocab_size, dtype=tf.float32)

            per_example_loss = -tf.reduce_sum(log_probs * one_hot_labels,␣
↪axis=[-1])
            numerator = tf.reduce_sum(label_weights * per_example_loss)
            denominator = tf.reduce_sum(label_weights) + 1e-5
            loss = numerator / denominator

        return loss, per_example_loss, log_probs

    def get_next_sentence_output(self, input_tensor, labels):
        """Get loss and log probs for the next sentence prediction."""
        with tf.variable_scope("cls/seq_relationship"):
            output_weights = tf.get_variable("output_weights", shape=[2, self.
↪bert_config.hidden_size],
                                             initializer=modeling.
↪create_initializer(self.bert_config.initializer_range))
            output_bias = tf.get_variable("output_bias", shape=[2],␣
↪initializer=tf.zeros_initializer())
            logits = tf.matmul(input_tensor, output_weights, transpose_b=True)
            logits = tf.nn.bias_add(logits, output_bias)
            log_probs = tf.nn.log_softmax(logits, axis=-1)
            labels = tf.reshape(labels, [-1])
            one_hot_labels = tf.one_hot(labels, depth=2, 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, log_probs

    def gather_indexes(self, sequence_tensor, positions):
        """Gathers the vectors at the specific positions over a minibatch."""
        sequence_shape = modeling.get_shape_list(sequence_tensor,␣
↪expected_rank=3)
        batch_size = sequence_shape[0]
        seq_length = sequence_shape[1]
        width = sequence_shape[2]
        flat_offsets = tf.reshape(tf.range(0, batch_size, dtype=tf.int32) *␣
↪seq_length, [-1, 1])
        flat_positions = tf.reshape(positions + flat_offsets, [-1])
        flat_sequence_tensor = tf.reshape(sequence_tensor, [batch_size *␣
↪seq_length, width])
        return tf.gather(flat_sequence_tensor, flat_positions)
```

```
    def get_eval_metrics(self, masked_lm_loss, masked_lm_ids, masked_lm_weights,␣
↪next_sentence_loss):
        """Computes the loss and accuracy of the model."""
        masked_lm_log_probs = tf.reshape(masked_lm_log_probs, [-1,␣
↪masked_lm_log_probs.shape[-1]])
        masked_lm_predictions = tf.argmax(masked_lm_log_probs, axis=-1,␣
↪output_type=tf.int32)
        masked_lm_example_loss = tf.reshape(masked_lm_example_loss, [-1])
        masked_lm_accuracy = tf.metrics.accuracy(labels=masked_lm_ids,␣
↪predictions=masked_lm_predictions, weights=masked_lm_weights)
        masked_lm_mean_loss = tf.metrics.mean(values=masked_lm_example_loss,␣
↪weights=masked_lm_weights)

        next_sentence_log_probs = tf.reshape(next_sentence_log_probs, [-1,␣
↪next_sentence_log_probs.shape[-1]])
        next_sentence_predictions = tf.argmax(next_sentence_log_probs, axis=-1,␣
↪output_type=tf.int32)
        next_sentence_accuracy = tf.metrics.
↪accuracy(labels=next_sentence_labels, predictions=next_sentence_predictions)
        next_sentence_mean_loss = tf.metrics.
↪mean(values=next_sentence_example_loss)

        return {
            "masked_lm_accuracy": masked_lm_accuracy,
            "masked_lm_loss": masked_lm_mean_loss,
            "next_sentence_accuracy": next_sentence_accuracy,
            "next_sentence_loss": next_sentence_mean_loss,
        }
```

5.5 Transformer Encoder Design

Each Transformer layer contains two main components: Multi-Head Self-Attention and Feed-Forward Neural Network (FFN). The multi-head self-attention mechanism allows each token to attend to all other tokens in the sequence, capturing dependencies across long ranges and enhancing the model's contextual understanding. Following the self-attention, a feed-forward network processes each token representation independently. Layer normalization and dropout are applied after each component to stabilize and regularize training. BERT's bidirectional structure enables each token to attend to both preceding and following tokens, distinguishing it from previous models like GPT, which used left-to-right unidirectional attention.

[ ]:
```
class BertModel:
    def __init__(self, config, is_training, input_ids):
        with tf.variable_scope("bert"):
            self.embedding_output = embedding_layer(input_ids, config)
            self.all_encoder_layers = transformer_stack(
                self.embedding_output,
                hidden_size=config.hidden_size,
```

```
                num_hidden_layers=config.num_hidden_layers,
                num_attention_heads=config.num_attention_heads,
                intermediate_size=config.intermediate_size
            )
            self.sequence_output = self.all_encoder_layers[-1]

def transformer_stack(input_tensor, hidden_size, num_hidden_layers,␣
 ↪num_attention_heads, intermediate_size):
    prev_output = input_tensor
    all_layers = []
    for layer_idx in range(num_hidden_layers):
        with tf.variable_scope(f"layer_{layer_idx}"):
            layer_output = transformer_layer(
                prev_output, hidden_size, num_attention_heads, intermediate_size
            )
            all_layers.append(layer_output)
            prev_output = layer_output
    return all_layers

def transformer_layer(input_tensor, hidden_size, num_attention_heads,␣
 ↪intermediate_size):
    attention_output = multi_head_attention(input_tensor, hidden_size,␣
 ↪num_attention_heads)
    attention_output = layer_norm(input_tensor + attention_output)
    intermediate_output = ff_layer(attention_output, intermediate_size)
    layer_output = layer_norm(attention_output + intermediate_output)
    return layer_output

def multi_head_attention(input_tensor, hidden_size, num_attention_heads):
    pass

def ff_layer(input_tensor, intermediate_size):
    pass

def layer_norm(input_tensor):
    pass

def gelu(x):
    cdf = 0.5 * (1.0 + tf.tanh((np.sqrt(2 / np.pi) * (x + 0.044715 * tf.pow(x,␣
 ↪3)))))
    return x * cdf
```

5.6 Fine-tuning BERT for Classification Tasks

To adapt BERT for specific downstream tasks, a task-specific layer is added to the output of BERT, typically using the hidden state of the CLS token. For classification tasks, this hidden state is fed into a classification layer, where a fully connected layer maps it to the appropriate number

of output labels. The entire model, including BERT's pre-trained parameters, is then fine-tuned on the task-specific dataset by optimizing a loss function (e.g., cross-entropy for classification). This approach allows BERT to effectively apply its pre-trained knowledge to a wide range of NLP tasks with minimal task-specific architecture modifications, demonstrating the flexibility and transferability of the BERT model across different tasks such as sentence classification, sentiment analysis, and more.

[ ]:
```
def create_model(bert_config, is_training, input_ids, input_mask, segment_ids,
                 labels, num_labels, use_one_hot_embeddings):
    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)

    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"):
        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)
```

[ ]:
```
class BertModel(object):
    def __init__(self,
                 config,
                 is_training,
                 input_ids,
                 input_mask=None,
                 token_type_ids=None,
                 use_one_hot_embeddings=False,
                 scope=None):
```

```
        # ... (initialization code)

    def get_pooled_output(self):
        return self.pooled_output

    def get_sequence_output(self):
        return self.sequence_output
```

[ ]:
```
def create_optimizer(loss, init_lr, num_train_steps, num_warmup_steps, use_tpu):
    global_step = tf.train.get_or_create_global_step()

    learning_rate = tf.constant(value=init_lr, shape=[], dtype=tf.float32)
    learning_rate = tf.train.polynomial_decay(
        learning_rate,
        global_step,
        num_train_steps,
        end_learning_rate=0.0,
        power=1.0,
        cycle=False)

    if num_warmup_steps:
        global_steps_int = tf.cast(global_step, tf.int32)
        warmup_steps_int = tf.constant(num_warmup_steps, dtype=tf.int32)

        global_steps_float = tf.cast(global_steps_int, tf.float32)
        warmup_steps_float = tf.cast(warmup_steps_int, tf.float32)

        warmup_percent_done = global_steps_float / warmup_steps_float
        warmup_learning_rate = init_lr * warmup_percent_done

        is_warmup = tf.cast(global_steps_int < warmup_steps_int, tf.float32)
        learning_rate = (
            (1.0 - is_warmup) * learning_rate + is_warmup * warmup_learning_rate)

    optimizer = AdamWeightDecayOptimizer(
        learning_rate=learning_rate,
        weight_decay_rate=0.01,
        beta_1=0.9,
        beta_2=0.999,
        epsilon=1e-6,
        exclude_from_weight_decay=["LayerNorm", "layer_norm", "bias"])

    if use_tpu:
        optimizer = tf.contrib.tpu.CrossShardOptimizer(optimizer)

    tvars = tf.trainable_variables()
    grads = tf.gradients(loss, tvars)
```

```
    (grads, _) = tf.clip_by_global_norm(grads, clip_norm=1.0)

    train_op = optimizer.apply_gradients(
        zip(grads, tvars), global_step=global_step)

    new_global_step = global_step + 1
    train_op = tf.group(train_op, [global_step.assign(new_global_step)])
    return train_op
```

6 Model Training

6.1 Pre-training Procedure

BERT's pre-training procedure is a key innovation that allows it to create deep bidirectional representations. The procedure involves two unsupervised tasks:

1. Masked Language Model (MLM): In this task, 15% of the input tokens are masked at random, and the model is trained to predict these masked tokens. This approach allows the model to capture bidirectional context, unlike traditional left-to-right language models. The MLM task is implemented as follows:
    - 80% of the time, the chosen token is replaced with [MASK]
    - 10% of the time, it is replaced with a random token
    - 10% of the time, it is left unchanged This strategy prevents the model from simply memorizing the masked token and encourages it to maintain a distributional contextual representation of every input token.
2. Next Sentence Prediction (NSP): This task trains the model to understand relationships between sentences. Given two sentences A and B, the model predicts whether B actually follows A in the original text. This task is crucial for downstream tasks that require understanding sentence relationships, such as Question Answering and Natural Language Inference.

The pre-training data consists of the BooksCorpus (800M words) and English Wikipedia (2,500M words), focusing on extracting text passages while ignoring lists, tables, and headers. This document-level corpus is crucial for learning long contiguous sequences

6.2 Loading Pre-trained Weights

After pre-training, the model weights can be loaded for fine-tuning on specific tasks. The BERT repository provides scripts to load these pre-trained weights efficiently. The process typically involves:

1. Initializing a BERT model with the same architecture as the pre-trained model.
2. Loading the pre-trained weights into this model.
3. Verifying that all weights have been correctly loaded.

This step is crucial as it allows the model to leverage the knowledge gained during pre-training when tackling downstream tasks.

6.3 Fine-tuning Approach

BERT's fine-tuning process is straightforward and effective due to its self-attention mechanism. This allows BERT to handle various downstream tasks by simply swapping out appropriate inputs and outputs. The fine-tuning process involves:

1. Input Representation: For applications involving text pairs, BERT uses the self-attention mechanism to unify the encoding of text pairs, effectively including bidirectional cross attention between two sentences.
2. Task-Specific Modifications: For each task, task-specific inputs and outputs are plugged into BERT. For token-level tasks (like sequence tagging or question answering), the token representations are fed into an output layer. For classification tasks (like sentiment analysis), the [CLS] representation is used.
3. End-to-End Training: All parameters are fine-tuned end-to-end on the specific task. This allows the model to adapt its pre-trained knowledge to the nuances of the task at hand.

The fine-tuning process is relatively inexpensive compared to pre-training. Most results can be replicated in about an hour on a single Cloud TPU, or a few hours on a GPU, starting from the same pre-trained model.

This approach has proven highly effective across a wide range of NLP tasks, often achieving state-of-the-art results with minimal task-specific architecture modifications.

[ ]:
```python
# Pre-training data preparation
class CreatePretrainingData:
    def create_training_instances(self, input_files, tokenizer, max_seq_length,␣
 ↪dupe_factor, short_seq_prob, masked_lm_prob, max_predictions_per_seq, rng):
        # Implementation for creating pre-training instances
        pass

# BERT pre-training
class BertPreTrainingModel(tf.keras.Model):
    def __init__(self, config, *inputs, **kwargs):
        super().__init__(*inputs, **kwargs)
        self.bert = TFBertMainLayer(config, name="bert")
        self.mlm = TFBertMLMHead(config, self.bert.embeddings, name="mlm___cls")
        self.nsp = TFBertNSPHead(config, name="nsp___cls")

    def call(self, inputs, **kwargs):
        # Implementation for the forward pass
        pass

# Fine-tuning for classification tasks
class BertForSequenceClassification(tf.keras.Model):
    def __init__(self, config, *inputs, **kwargs):
        super().__init__(*inputs, **kwargs)
        self.num_labels = config.num_labels
        self.bert = TFBertMainLayer(config, name="bert")
```

```
        self.dropout = tf.keras.layers.Dropout(config.hidden_dropout_prob)
        self.classifier = tf.keras.layers.Dense(config.num_labels,
                                                ␣
 ↪kernel_initializer=get_initializer(config.initializer_range),
                                                 name="classifier")

    def call(self, inputs, **kwargs):
        # Implementation for classification
        pass

# Optimization
def create_optimizer(init_lr, num_train_steps, num_warmup_steps):
    """Creates an optimizer with learning rate schedule."""
    # Implementation of learning rate schedule and optimizer
    pass
```

7 Experimental Setup

BERT (Bidirectional Encoder Representations from Transformers) has revolutionized natural language processing tasks by introducing a powerful pre-training technique that captures deep bidirectional representations. Our experimental setup aims to leverage BERT's capabilities for various downstream tasks.

7.1 Datasets Used

We employ several benchmark datasets to assess BERT's performance across different NLP tasks:

- **GLUE Benchmark**: A collection of 9 diverse NLU tasks including sentiment analysis, textual entailment, and question answering. This benchmark provides a comprehensive evaluation of BERT's language understanding abilities across multiple domains.
- **SQuAD**: Stanford Question Answering Dataset for evaluating reading comprehension. This dataset challenges BERT's ability to understand context and extract relevant information to answer questions.
- **SWAG**: Situations With Adversarial Generations for commonsense inference. This dataset tests BERT's capacity to reason about everyday situations and make logical inferences.

[ ]:
```
class GLUEProcessor(DataProcessor):
    def get_train_examples(self, data_dir):
        return self._create_examples(
            self._read_tsv(os.path.join(data_dir, "train.tsv")), "train")

    def get_dev_examples(self, data_dir):
        return self._create_examples(
            self._read_tsv(os.path.join(data_dir, "dev.tsv")), "dev")

    def _create_examples(self, lines, set_type):
```

```
        examples = []
        for (i, line) in enumerate(lines):
            guid = f"{set_type}-{i}"
            text_a = line[3]
            text_b = line[4]
            label = line[1]
            examples.append(
                InputExample(guid=guid, text_a=text_a, text_b=text_b,␣
 ↪label=label))
        return examples

processor = GLUEProcessor()
train_examples = processor.get_train_examples("path/to/glue_data")
dev_examples = processor.get_dev_examples("path/to/glue_data")
```

7.2 Hyperparameter Selection

We follow the fine-tuning hyperparameters recommended in the original BERT paper:

- Batch size: 32
- Learning rate: 5e-5
- Number of epochs: 3

However, we conduct limited hyperparameter tuning experiments, varying the learning rate (2e-5, 3e-5, 5e-5) and number of epochs (2-4) to find optimal settings for each task. This fine-tuning process allows us to adapt BERT's pre-trained knowledge to specific downstream tasks.

[ ]:
```
bert_config = BertConfig(
    vocab_size=30522,
    hidden_size=768,
    num_hidden_layers=12,
    num_attention_heads=12,
    intermediate_size=3072,
)

model = BertForSequenceClassification(
    config=bert_config,
    num_labels=2
)

train_batch_size = 32
learning_rate = 5e-5
num_train_epochs = 3.0
warmup_proportion = 0.1

from optimization import create_optimizer

num_train_steps = int(
```

```
    len(train_examples) / train_batch_size * num_train_epochs)
num_warmup_steps = int(num_train_steps * warmup_proportion)

optimizer = create_optimizer(
    model.parameters(),
    learning_rate,
    num_train_steps,
    num_warmup_steps
)
```

7.3 Evaluation Metrics

We use task-specific evaluation metrics as defined by each dataset:

- Accuracy for classification tasks
- F1 score for question answering
- Matthews correlation for CoLA (Corpus of Linguistic Acceptability)
- Spearman correlation for STS-B (Semantic Textual Similarity Benchmark)

For the GLUE benchmark, we report the average score across all tasks as the overall performance metric. This comprehensive evaluation allows us to assess BERT's versatility across various NLP challenges.

BERT's architecture, consisting of multiple Transformer encoder layers, enables it to capture complex linguistic patterns and relationships. The self-attention mechanism in these layers allows BERT to weigh the importance of different words in context, leading to more nuanced representations.

The pre-training objectives of Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) contribute to BERT's effectiveness. MLM helps BERT learn contextual representations by predicting masked words, while NSP enables it to understand relationships between sentences.

Our experimental setup aims to exploit these powerful features of BERT through careful fine-tuning and evaluation across diverse NLP tasks.

[ ]:
```
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)
    loss = tf.metrics.mean(per_example_loss)
    return {
        "eval_accuracy": accuracy,
        "eval_loss": loss,
    }

def f1_score(labels, predictions):
    precision = tf.metrics.precision(labels, predictions)
    recall = tf.metrics.recall(labels, predictions)
    f1 = 2 * (precision[0] * recall[0]) / (precision[0] + recall[0])
    return f1
```

```
eval_metrics = (metric_fn, [per_example_loss, label_ids, logits])

metrics = eval_metrics[0](
    eval_metrics[1][0], eval_metrics[1][1], eval_metrics[1][2])

print(f"Accuracy: {metrics['eval_accuracy'][0]}")
print(f"Loss: {metrics['eval_loss'][0]}")
```

8 Results and Analysis

BERT has demonstrated exceptional performance across a wide range of natural language processing tasks. On the General Language Understanding Evaluation (GLUE) benchmark, BERT achieved state-of-the-art results, significantly outperforming previous models. The BERT-Large model obtained a GLUE score of 80.5%, which represented a 7.7% absolute improvement over the previous best model.

For specific tasks within GLUE, BERT showed remarkable gains. On the MultiNLI task, BERT-Large achieved an accuracy of 86.7%, a 4.6% absolute improvement over the previous state-of-the-art. On the Stanford Question Answering Dataset (SQuAD v1.1), BERT-Large achieved a Test F1 score of 93.2, surpassing human performance.

Ablation studies revealed the importance of BERT's bidirectional nature and its novel pre-training tasks. The Next Sentence Prediction (NSP) task proved particularly beneficial for tasks involving sentence pair classification. The Masked Language Model (MLM) pre-training objective was shown to be critical for token-level tasks.

When compared to other state-of-the-art models, BERT consistently outperformed them across various benchmarks. For instance, on the SQuAD v1.1 leaderboard, BERT-Large (single model) achieved an F1 score of 93.2, surpassing the previous top single model by 1.5 points.

[ ]:
```
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

def encode_examples(texts, labels):
    return tokenizer(texts, padding=True, truncation=True, return_tensors='tf',␣
 ↪max_length=512)

texts = ["I loved this movie!", "This was a terrible film."]
labels = [1, 0]
encoded_data = encode_examples(texts, labels)

# Fine-tuning function
def fine_tune_bert(model, train_dataset):
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-5),
                  loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
    model.fit(train_dataset, epochs=3)
```

```
train_dataset = tf.data.Dataset.from_tensor_slices((encoded_data['input_ids'],␣
 ↪labels)).batch(2)
fine_tune_bert(model, train_dataset)
```

9 Sentiment Analysis using BERT: A Real-World Example

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art model for natural language processing tasks, and its application to sentiment analysis demonstrates its effectiveness. In this study, the model was fine-tuned to classify movie reviews from the IMDB dataset as positive or negative. The text data was tokenized using the BERT tokenizer, which converts sentences into WordPiece tokens, ensuring compatibility with the model's input requirements. The architecture consisted of a pre-trained BERT encoder with an additional dense layer for classification, and the entire model was fine-tuned using binary cross-entropy loss. This process allowed the model to adapt to the specific nuances of the sentiment analysis task, capturing the contextual information essential for accurate predictions.

The fine-tuned BERT model achieved impressive accuracy, exceeding 94% on the IMDB dataset. Its ability to understand context and handle complex expressions of sentiment made it superior to traditional machine learning approaches and simpler deep learning models. The results highlight the effectiveness of pre-trained models in reducing the need for extensive feature engineering while delivering high performance. This example underscores how BERT's bidirectional contextual understanding can simplify and enhance NLP workflows, providing a robust solution for real-world applications such as sentiment analysis.

#Predicting Movie Review Sentiment with BERT on TF Hub

[1]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import sys
sys.path.append(r"c:\users\dell\appdata\roaming\python\python36\site-packages")
import pandas as pd
import tensorflow as tf
import tensorflow_hub as hub
print("TensorFlow Hub version:", hub.__version__)

from datetime import datetime
print("Tensorflow version:", tf.__version__)
```

```
TensorFlow Hub version: 0.7.0
Tensorflow version: 1.15.0
```

In addition to the standard libraries we imported above, we'll need to install BERT's python package.

[2]:
```
import bert
from bert import run_classifier
from bert import optimization
from bert import tokenization
```

#Data

First, let's download the dataset, hosted by Stanford. The code below, which downloads, extracts, and imports the IMDB Large Movie Review Dataset, is borrowed from this Tensorflow tutorial.

[3]:
```
from tensorflow import keras
import os
import re

# Load all files from a directory in a DataFrame.
def load_directory_data(directory):
  data = {}
  data["sentence"] = []
  data["sentiment"] = []
  for file_path in os.listdir(directory):
    with tf.gfile.GFile(os.path.join(directory, file_path), "r") as f:
      data["sentence"].append(f.read())
      data["sentiment"].append(re.match("\d+_(\d+)\.txt", file_path).group(1))
  return pd.DataFrame.from_dict(data)

# Merge positive and negative examples, add a polarity column and shuffle.
def load_dataset(directory):
  pos_df = load_directory_data(os.path.join(directory, "pos"))
  neg_df = load_directory_data(os.path.join(directory, "neg"))
  pos_df["polarity"] = 1
  neg_df["polarity"] = 0
  return pd.concat([pos_df, neg_df]).sample(frac=1).reset_index(drop=True)

# Download and process the dataset files.
def download_and_load_datasets(force_download=False):
  dataset = tf.keras.utils.get_file(
      fname="aclImdb.tar.gz",
      origin="http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz",
      extract=True)

  train_df = load_dataset(os.path.join(os.path.dirname(dataset),
                                       "aclImdb", "train"))
  test_df = load_dataset(os.path.join(os.path.dirname(dataset),
                                      "aclImdb", "test"))

  return train_df, test_df
```

[4]:
```
train, test = download_and_load_datasets()
```

To keep training fast, we'll take a sample of 5000 train and test examples, respectively.

[5]:
```
train = train.sample(5000)
test = test.sample(5000)
```

[6]:
```
train.columns
```

[6]:
```
Index(['sentence', 'sentiment', 'polarity'], dtype='object')
```

For us, our input data is the 'sentence' column and our label is the 'polarity' column (0, 1 for negative and positive, respecitvely)

[7]:
```
DATA_COLUMN = 'sentence'
LABEL_COLUMN = 'polarity'
# label_list is the list of labels, i.e. True, False or 0, 1 or 'dog', 'cat'
label_list = [0, 1]
```

#Data Preprocessing We'll need to transform our data into a format BERT understands. This involves two steps. First, we create `InputExample`'s using the constructor provided in the BERT library.

- `text_a` is the text we want to classify, which in this case, is the `Request` field in our Dataframe.
- `text_b` is used if we're training a model to understand the relationship between sentences (i.e. is `text_b` a translation of `text_a`? Is `text_b` an answer to the question asked by `text_a`?). This doesn't apply to our task, so we can leave `text_b` blank.
- `label` is the label for our example, i.e. True, False

[8]:
```
# Use the InputExample class from BERT's run_classifier code to create examples␣
 ↪from the data
train_InputExamples = train.apply(lambda x: bert.run_classifier.
 ↪InputExample(guid=None, # Globally unique ID for bookkeeping, unused in this␣
 ↪example
                                                                   text_a =␣
 ↪x[DATA_COLUMN],
                                                                   text_b = None,
                                                                   label =␣
 ↪x[LABEL_COLUMN]), axis = 1)

test_InputExamples = test.apply(lambda x: bert.run_classifier.
 ↪InputExample(guid=None,
                                                                   text_a =␣
 ↪x[DATA_COLUMN],
                                                                   text_b = None,
                                                                   label =␣
 ↪x[LABEL_COLUMN]), axis = 1)
```

Next, we need to preprocess our data so that it matches the data BERT was trained on. For this, we'll need to do a couple of things (but don't worry–this is also included in the Python library):

1. Lowercase our text (if we're using a BERT lowercase model)
2. Tokenize it (i.e. "sally says hi" -> ["sally", "says", "hi"])
3. Break words into WordPieces (i.e. "calling" -> ["call", "##ing"])
4. Map our words to indexes using a vocab file that BERT provides

5. Add special "CLS" and "SEP" tokens (see the readme)
6. Append "index" and "segment" tokens to each input (see the BERT paper)

Happily, we don't have to worry about most of these details.

To start, we'll need to load a vocabulary file and lowercasing information directly from the BERT tf hub module:

[9]:
```
# This is a path to an uncased (all lowercase) version of BERT
from absl import flags
BERT_MODEL_HUB = "https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1"

def create_tokenizer_from_hub_module():
  """Get the vocab file and casing info from the Hub module."""
  flags.FLAGS([''])

  with tf.Graph().as_default():
    bert_module = hub.Module(BERT_MODEL_HUB)
    tokenization_info = bert_module(signature="tokenization_info", as_dict=True)
    with tf.Session() as sess:
      vocab_file, do_lower_case = sess.run([tokenization_info["vocab_file"],
                                            tokenization_info["do_lower_case"]])

  return bert.tokenization.FullTokenizer(
      vocab_file=vocab_file, do_lower_case=do_lower_case)



tokenizer = create_tokenizer_from_hub_module()
```

```
INFO:tensorflow:Saver not created because there are no variables in the graph to
restore
```

```
INFO:tensorflow:Saver not created because there are no variables in the graph to
restore
```

Great–we just learned that the BERT model we're using expects lowercase data (that's what stored in tokenization_info["do_lower_case"]) and we also loaded BERT's vocab file. We also created a tokenizer, which breaks words into word pieces:

[10]:
```
tokenizer.tokenize("This here's an example of using the BERT tokenizer")
```

[10]:
```
['this',
 'here',
 "'",
 's',
 'an',
 'example',
 'of',
 'using',
```

```
 'the',
 'bert',
 'token',
 '##izer']
```

Using our tokenizer, we'll call `run_classifier.convert_examples_to_features` on our InputExamples to convert them into features BERT understands.

[11]:
```
# We'll set sequences to be at most 128 tokens long.
MAX_SEQ_LENGTH = 128
# Convert our train and test features to InputFeatures that BERT understands.
train_features = bert.run_classifier.
 ↪convert_examples_to_features(train_InputExamples, label_list, MAX_SEQ_LENGTH,␣
 ↪tokenizer)
test_features = bert.run_classifier.
 ↪convert_examples_to_features(test_InputExamples, label_list, MAX_SEQ_LENGTH,␣
 ↪tokenizer)
```

```
INFO:tensorflow:Writing example 0 of 5000

INFO:tensorflow:Writing example 0 of 5000

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] my girlfriend and i were stunned by how bad this
film was . after 15 minutes we would have called it quit ##s except we were too
curious to see if the film could possibly red ##eem itself . it didn ' t . < br
/ > < br / > i can ' t understand the praise given to this film . the writing
was down ##right awful and delivered by some of the worst acting i have seen in
a very long time . < br / > < br / > one thing that especially annoyed me about
this film was that often when people were talking to each other there was an
unnatural pause between lines . i understand using a [SEP]

INFO:tensorflow:tokens: [CLS] my girlfriend and i were stunned by how bad this
film was . after 15 minutes we would have called it quit ##s except we were too
curious to see if the film could possibly red ##eem itself . it didn ' t . < br
/ > < br / > i can ' t understand the praise given to this film . the writing
was down ##right awful and delivered by some of the worst acting i have seen in
a very long time . < br / > < br / > one thing that especially annoyed me about
this film was that often when people were talking to each other there was an
unnatural pause between lines . i understand using a [SEP]

INFO:tensorflow:input_ids: 101 2026 6513 1998 1045 2020 9860 2011 2129 2919 2023
2143 2001 1012 2044 2321 2781 2057 2052 2031 2170 2009 8046 2015 3272 2057 2020
2205 8025 2000 2156 2065 1996 2143 2071 4298 2417 21564 2993 1012 2009 2134 1005
1056 1012 1026 7987 1013 1028 1026 7987 1013 1028 1045 2064 1005 1056 3305 1996
```

8489 2445 2000 2023 2143 1012 1996 3015 2001 2091 15950 9643 1998 5359 2011 2070 1997 1996 5409 3772 1045 2031 2464 1999 1037 2200 2146 2051 1012 1026 7987 1013 1028 1026 7987 1013 1028 2028 2518 2008 2926 11654 2033 2055 2023 2143 2001 2008 2411 2043 2111 2020 3331 2000 2169 2060 2045 2001 2019 21242 8724 2090 3210 1012 1045 3305 2478 1037 102

INFO:tensorflow:input_ids: 101 2026 6513 1998 1045 2020 9860 2011 2129 2919 2023 2143 2001 1012 2044 2321 2781 2057 2052 2031 2170 2009 8046 2015 3272 2057 2020 2205 8025 2000 2156 2065 1996 2143 2071 4298 2417 21564 2993 1012 2009 2134 1005 1056 1012 1026 7987 1013 1028 1026 7987 1013 1028 1045 2064 1005 1056 3305 1996 8489 2445 2000 2023 2143 1012 1996 3015 2001 2091 15950 9643 1998 5359 2011 2070 1997 1996 5409 3772 1045 2031 2464 1999 1037 2200 2146 2051 1012 1026 7987 1013 1028 1026 7987 1013 1028 2028 2518 2008 2926 11654 2033 2055 2023 2143 2001 2008 2411 2043 2111 2020 3331 2000 2169 2060 2045 2001 2019 21242 8724 2090 3210 1012 1045 3305 2478 1037 102

INFO:tensorflow:input_mask: 1

INFO:tensorflow:input_mask: 1

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] yesterday was earth day (april 22 , 2009) in the us and other countries , and i went to see the full - feature movie - version of " earth " by disney ##nat ##ure . i guess , like the auto manufacturers , disney is trying to convince us that they care about the planet . maybe they really do care about the planet , i don ' t know , but i don ' t think it warrant ##s a special unit with the word " nature " in it . i do know that my youngest

daughter loves mickey mouse , and who am i to tell a one - year old my personal feelings about disney ? < br / [SEP]

INFO:tensorflow:tokens: [CLS] yesterday was earth day (april 22 , 2009) in the us and other countries , and i went to see the full - feature movie - version of " earth " by disney ##nat ##ure . i guess , like the auto manufacturers , disney is trying to convince us that they care about the planet . maybe they really do care about the planet , i don ' t know , but i don ' t think it warrant ##s a special unit with the word " nature " in it . i do know that my youngest daughter loves mickey mouse , and who am i to tell a one - year old my personal feelings about disney ? < br / [SEP]

INFO:tensorflow:input_ids: 101 7483 2001 3011 2154 1006 2258 2570 1010 2268 1007 1999 1996 2149 1998 2060 3032 1010 1998 1045 2253 2000 2156 1996 2440 1011 3444 3185 1011 2544 1997 1000 3011 1000 2011 6373 19833 5397 1012 1045 3984 1010 2066 1996 8285 8712 1010 6373 2003 2667 2000 8054 2149 2008 2027 2729 2055 1996 4774 1012 2672 2027 2428 2079 2729 2055 1996 4774 1010 1045 2123 1005 1056 2113 1010 2021 1045 2123 1005 1056 2228 2009 10943 2015 1037 2569 3131 2007 1996 2773 1000 3267 1000 1999 2009 1012 1045 2079 2113 2008 2026 6587 2684 7459 11021 8000 1010 1998 2040 2572 1045 2000 2425 1037 2028 1011 2095 2214 2026 3167 5346 2055 6373 1029 1026 7987 1013 102

INFO:tensorflow:input_ids: 101 7483 2001 3011 2154 1006 2258 2570 1010 2268 1007 1999 1996 2149 1998 2060 3032 1010 1998 1045 2253 2000 2156 1996 2440 1011 3444 3185 1011 2544 1997 1000 3011 1000 2011 6373 19833 5397 1012 1045 3984 1010 2066 1996 8285 8712 1010 6373 2003 2667 2000 8054 2149 2008 2027 2729 2055 1996 4774 1012 2672 2027 2428 2079 2729 2055 1996 4774 1010 1045 2123 1005 1056 2113 1010 2021 1045 2123 1005 1056 2228 2009 10943 2015 1037 2569 3131 2007 1996 2773 1000 3267 1000 1999 2009 1012 1045 2079 2113 2008 2026 6587 2684 7459 11021 8000 1010 1998 2040 2572 1045 2000 2425 1037 2028 1011 2095 2214 2026 3167 5346 2055 6373 1029 1026 7987 1013 102

INFO:tensorflow:input_mask: 1

INFO:tensorflow:input_mask: 1

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] this movie has a very broadway feel - the backdrop , the acting , the ' noise ' - and yet that ' s all it has . some ' sense ' of a broadway without the bang . < br / > < br / > the movie is slow - paced , the picture di ##s ##jo ##int ##ed , the singing ' pops up ' on you so that you suddenly are reminded it ' s a musical . < br / > < br / > disappointing : sinatra < br / > < br / > into ##ler ##able : sinatra ' s fiance - - - surely , the pitch and the accent of her voice was unnecessary . < [SEP]

INFO:tensorflow:tokens: [CLS] this movie has a very broadway feel - the backdrop , the acting , the ' noise ' - and yet that ' s all it has . some ' sense ' of a broadway without the bang . < br / > < br / > the movie is slow - paced , the picture di ##s ##jo ##int ##ed , the singing ' pops up ' on you so that you suddenly are reminded it ' s a musical . < br / > < br / > disappointing : sinatra < br / > < br / > into ##ler ##able : sinatra ' s fiance - - - surely , the pitch and the accent of her voice was unnecessary . < [SEP]

INFO:tensorflow:input_ids: 101 2023 3185 2038 1037 2200 5934 2514 1011 1996 18876 1010 1996 3772 1010 1996 1005 5005 1005 1011 1998 2664 2008 1005 1055 2035 2009 2038 1012 2070 1005 3168 1005 1997 1037 5934 2302 1996 9748 1012 1026 7987 1013 1028 1026 7987 1013 1028 1996 3185 2003 4030 1011 13823 1010 1996 3861 4487 2015 5558 18447 2098 1010 1996 4823 1005 16949 2039 1005 2006 2017 2061 2008 2017 3402 2024 6966 2009 1005 1055 1037 3315 1012 1026 7987 1013 1028 1026 7987 1013 1028 15640 1024 19643 1026 7987 1013 1028 1026 7987 1013 1028 2046 3917 3085 1024 19643 1005 1055 19154 1011 1011 1011 7543 1010 1996 6510 1998 1996 9669 1997 2014 2376 2001 14203 1012 1026 102

INFO:tensorflow:input_ids: 101 2023 3185 2038 1037 2200 5934 2514 1011 1996 18876 1010 1996 3772 1010 1996 1005 5005 1005 1011 1998 2664 2008 1005 1055 2035 2009 2038 1012 2070 1005 3168 1005 1997 1037 5934 2302 1996 9748 1012 1026 7987 1013 1028 1026 7987 1013 1028 1996 3185 2003 4030 1011 13823 1010 1996 3861 4487 2015 5558 18447 2098 1010 1996 4823 1005 16949 2039 1005 2006 2017 2061 2008 2017 3402 2024 6966 2009 1005 1055 1037 3315 1012 1026 7987 1013 1028 1026 7987 1013 1028 15640 1024 19643 1026 7987 1013 1028 1026 7987 1013 1028 2046 3917 3085 1024 19643 1005 1055 19154 1011 1011 1011 7543 1010 1996 6510 1998 1996 9669 1997 2014 2376 2001 14203 1012 1026 102

INFO:tensorflow:input_mask: 1

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

INFO:tensorflow:segment_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 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
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 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:segment_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 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
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 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:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] the plot is about the death of little children . hopper is the one who has to investigate the killings . during the movie it appears that he has some troubles with his daughter . in the end the serial killer get caught . that ' s it . but before you find out who dunn ##it , you have to see some terrible acting by all of the actors . it is unbelievable how bad these actors are , including hopper . i could go on like this but that to much of a waste of my time . just don ' t watch the movie . i ' ve warned you . [SEP]

INFO:tensorflow:tokens: [CLS] the plot is about the death of little children . hopper is the one who has to investigate the killings . during the movie it appears that he has some troubles with his daughter . in the end the serial killer get caught . that ' s it . but before you find out who dunn ##it , you have to see some terrible acting by all of the actors . it is unbelievable how bad these actors are , including hopper . i could go on like this but that to much of a waste of my time . just don ' t watch the movie . i ' ve warned you . [SEP]

INFO:tensorflow:input_ids: 101 1996 5436 2003 2055 1996 2331 1997 2210 2336 1012
20517 2003 1996 2028 2040 2038 2000 8556 1996 16431 1012 2076 1996 3185 2009
3544 2008 2002 2038 2070 13460 2007 2010 2684 1012 1999 1996 2203 1996 7642 6359
2131 3236 1012 2008 1005 1055 2009 1012 2021 2077 2017 2424 2041 2040 14145 4183
1010 2017 2031 2000 2156 2070 6659 3772 2011 2035 1997 1996 5889 1012 2009 2003
23653 2129 2919 2122 5889 2024 1010 2164 20517 1012 1045 2071 2175 2006 2066
2023 2021 2008 2000 2172 1997 1037 5949 1997 2026 2051 1012 2074 2123 1005 1056
3422 1996 3185 1012 1045 1005 2310 7420 2017 1012 102 0 0 0 0 0 0 0 0 0 0 0 0

```
INFO:tensorflow:input_ids: 101 1996 5436 2003 2055 1996 2331 1997 2210 2336 1012
20517 2003 1996 2028 2040 2038 2000 8556 1996 16431 1012 2076 1996 3185 2009
3544 2008 2002 2038 2070 13460 2007 2010 2684 1012 1999 1996 2203 1996 7642 6359
2131 3236 1012 2008 1005 1055 2009 1012 2021 2077 2017 2424 2041 2040 14145 4183
1010 2017 2031 2000 2156 2070 6659 3772 2011 2035 1997 1996 5889 1012 2009 2003
23653 2129 2919 2122 5889 2024 1010 2164 20517 1012 1045 2071 2175 2006 2066
2023 2021 2008 2000 2172 1997 1037 5949 1997 2026 2051 1012 2074 2123 1005 1056
3422 1996 3185 1012 1045 1005 2310 7420 2017 1012 102 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_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 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
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 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:segment_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 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
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 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:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] excellent far ##ce ! which , of course , is all it
is intended to be . thankfully there is neither a social or political message ,
nor is there the slightest attempt in that direction . could the plot actually
take , or have taken place in any particular time or location ? unlikely , for ,
after all , this is simply , merely , a movie , and movies spring from
imagination , not from reality . the only goal of this movie is to entertain ,
certainly not to educate , and entertain it does , with reality delightful ##ly
and light ##hearted ##ly tossed to the winds . i think most would agree that
from documentaries we expect enlightenment and authenticity [SEP]

INFO:tensorflow:tokens: [CLS] excellent far ##ce ! which , of course , is all it
is intended to be . thankfully there is neither a social or political message ,
nor is there the slightest attempt in that direction . could the plot actually
```

take , or have taken place in any particular time or location ? unlikely , for , after all , this is simply , merely , a movie , and movies spring from imagination , not from reality . the only goal of this movie is to entertain , certainly not to educate , and entertain it does , with reality delightful ##ly and light ##hearted ##ly tossed to the winds . i think most would agree that from documentaries we expect enlightenment and authenticity [SEP]

INFO:tensorflow:input_ids: 101 6581 2521 3401 999 2029 1010 1997 2607 1010 2003 2035 2009 2003 3832 2000 2022 1012 16047 2045 2003 4445 1037 2591 2030 2576 4471 1010 4496 2003 2045 1996 15989 3535 1999 2008 3257 1012 2071 1996 5436 2941 2202 1010 2030 2031 2579 2173 1999 2151 3327 2051 2030 3295 1029 9832 1010 2005 1010 2044 2035 1010 2023 2003 3432 1010 6414 1010 1037 3185 1010 1998 5691 3500 2013 9647 1010 2025 2013 4507 1012 1996 2069 3125 1997 2023 3185 2003 2000 20432 1010 5121 2025 2000 16957 1010 1998 20432 2009 2515 1010 2007 4507 26380 2135 1998 2422 27693 2135 7463 2000 1996 7266 1012 1045 2228 2087 2052 5993 2008 2013 15693 2057 5987 16724 1998 21452 102

INFO:tensorflow:input_ids: 101 6581 2521 3401 999 2029 1010 1997 2607 1010 2003 2035 2009 2003 3832 2000 2022 1012 16047 2045 2003 4445 1037 2591 2030 2576 4471 1010 4496 2003 2045 1996 15989 3535 1999 2008 3257 1012 2071 1996 5436 2941 2202 1010 2030 2031 2579 2173 1999 2151 3327 2051 2030 3295 1029 9832 1010 2005 1010 2044 2035 1010 2023 2003 3432 1010 6414 1010 1037 3185 1010 1998 5691 3500 2013 9647 1010 2025 2013 4507 1012 1996 2069 3125 1997 2023 3185 2003 2000 20432 1010 5121 2025 2000 16957 1010 1998 20432 2009 2515 1010 2007 4507 26380 2135 1998 2422 27693 2135 7463 2000 1996 7266 1012 1045 2228 2087 2052 5993 2008 2013 15693 2057 5987 16724 1998 21452 102

INFO:tensorflow:input_mask: 1

INFO:tensorflow:input_mask: 1

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:Writing example 0 of 5000

```
INFO:tensorflow:Writing example 0 of 5000

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] wow ! this movie is almost too bad for words .
obviously the writers wanted to somehow link this to the g ##ho ##uli ##es
franchise , so they got pete lia ##pis from the first one to reprise his role as
jonathan . . . only now , he ' s a cop and has no similar character traits as he
did in the first one . the g ##ho ##uli ##es in this one aren ' t the g ##ho
##uli ##es from the last ones . the cheap looking puppets have been replaced
with even cheaper looking costume ##d little people . instead of being any main
antagonist or being evil , they are more like the comic relief characters that
appeared out [SEP]

INFO:tensorflow:tokens: [CLS] wow ! this movie is almost too bad for words .
obviously the writers wanted to somehow link this to the g ##ho ##uli ##es
franchise , so they got pete lia ##pis from the first one to reprise his role as
jonathan . . . only now , he ' s a cop and has no similar character traits as he
did in the first one . the g ##ho ##uli ##es in this one aren ' t the g ##ho
##uli ##es from the last ones . the cheap looking puppets have been replaced
with even cheaper looking costume ##d little people . instead of being any main
antagonist or being evil , they are more like the comic relief characters that
appeared out [SEP]

INFO:tensorflow:input_ids: 101 10166 999 2023 3185 2003 2471 2205 2919 2005 2616
1012 5525 1996 4898 2359 2000 5064 4957 2023 2000 1996 1043 6806 15859 2229 6329
1010 2061 2027 2288 6969 22393 18136 2013 1996 2034 2028 2000 16851 2010 2535
2004 5655 1012 1012 1012 2069 2085 1010 2002 1005 1055 1037 8872 1998 2038 2053
2714 2839 12955 2004 2002 2106 1999 1996 2034 2028 1012 1996 1043 6806 15859
2229 1999 2023 2028 4995 1005 1056 1996 1043 6806 15859 2229 2013 1996 2197 3924
1012 1996 10036 2559 26101 2031 2042 2999 2007 2130 16269 2559 9427 2094 2210
2111 1012 2612 1997 2108 2151 2364 17379 2030 2108 4763 1010 2027 2024 2062 2066
1996 5021 4335 3494 2008 2596 2041 102

INFO:tensorflow:input_ids: 101 10166 999 2023 3185 2003 2471 2205 2919 2005 2616
1012 5525 1996 4898 2359 2000 5064 4957 2023 2000 1996 1043 6806 15859 2229 6329
1010 2061 2027 2288 6969 22393 18136 2013 1996 2034 2028 2000 16851 2010 2535
2004 5655 1012 1012 1012 2069 2085 1010 2002 1005 1055 1037 8872 1998 2038 2053
2714 2839 12955 2004 2002 2106 1999 1996 2034 2028 1012 1996 1043 6806 15859
2229 1999 2023 2028 4995 1005 1056 1996 1043 6806 15859 2229 2013 1996 2197 3924
1012 1996 10036 2559 26101 2031 2042 2999 2007 2130 16269 2559 9427 2094 2210
2111 1012 2612 1997 2108 2151 2364 17379 2030 2108 4763 1010 2027 2024 2062 2066
1996 5021 4335 3494 2008 2596 2041 102

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

INFO:tensorflow:segment_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 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
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 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:segment_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 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
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 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:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] when pink ##y , a qualified electric ##ian , is released from prison , his parole officer has found him a job working at a big city bank . when some of the crime underworld from his past learn of his position they plan to exploit it and rob the bank . pink ##y is at first horrified because he really wants to go straight , but when a twist of fate happens pink ##y begins to think one shouldn ' t look a gift horse in the mouth . < br / > < br / > also known as the mayfair bank cape ##r { amongst others ! } , this is a hugely enjoyable piece that is qui ##ntes ##sen ##tial 1970s . [SEP]

INFO:tensorflow:tokens: [CLS] when pink ##y , a qualified electric ##ian , is released from prison , his parole officer has found him a job working at a big city bank . when some of the crime underworld from his past learn of his position they plan to exploit it and rob the bank . pink ##y is at first horrified because he really wants to go straight , but when a twist of fate happens pink ##y begins to think one shouldn ' t look a gift horse in the mouth . < br / > < br / > also known as the mayfair bank cape ##r { amongst others ! } , this is a hugely enjoyable piece that is qui ##ntes ##sen ##tial 1970s . [SEP]

INFO:tensorflow:input_ids: 101 2043 5061 2100 1010 1037 4591 3751 2937 1010 2003 2207 2013 3827 1010 2010 17393 2961 2038 2179 2032 1037 3105 2551 2012 1037 2502 2103 2924 1012 2043 2070 1997 1996 4126 13607 2013 2010 2627 4553 1997 2010 2597 2027 2933 2000 18077 2009 1998 6487 1996 2924 1012 5061 2100 2003 2012 2034 14603 2138 2002 2428 4122 2000 2175 3442 1010 2021 2043 1037 9792 1997 6580 6433 5061 2100 4269 2000 2228 2028 5807 1005 1056 2298 1037 5592 3586 1999 1996 2677

1012 1026 7987 1013 1028 1026 7987 1013 1028 2036 2124 2004 1996 28387 2924 4880 2099 1063 5921 2500 999 1065 1010 2023 2003 1037 27564 22249 3538 2008 2003 21864 17340 5054 20925 3955 1012 102

INFO:tensorflow:input_ids: 101 2043 5061 2100 1010 1037 4591 3751 2937 1010 2003 2207 2013 3827 1010 2010 17393 2961 2038 2179 2032 1037 3105 2551 2012 1037 2502 2103 2924 1012 2043 2070 1997 1996 4126 13607 2013 2010 2627 4553 1997 2010 2597 2027 2933 2000 18077 2009 1998 6487 1996 2924 1012 5061 2100 2003 2012 2034 14603 2138 2002 2428 4122 2000 2175 3442 1010 2021 2043 1037 9792 1997 6580 6433 5061 2100 4269 2000 2228 2028 5807 1005 1056 2298 1037 5592 3586 1999 1996 2677 1012 1026 7987 1013 1028 1026 7987 1013 1028 2036 2124 2004 1996 28387 2924 4880 2099 1063 5921 2500 999 1065 1010 2023 2003 1037 27564 22249 3538 2008 2003 21864 17340 5054 20925 3955 1012 102

INFO:tensorflow:input_mask: 1

INFO:tensorflow:input_mask: 1

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] hind ##sight is a wonderful thing . much rev ##iled when it first appeared , (inspiring the famous review ' me no lei ##ca ') , this precursor of " cabaret " can now be looked at in comparison and it ' s not half bad . it ' s certainly no classic but it has its own way ##ward charm , (the film version of " cabaret " follows this plot whereas the stage version changed the plot somewhat) . one should , of course , resist the temptation to s ##nick ##er when laurence harvey ' s christopher is ##her ##wood , (it keeps the original author ' s real name ; god knows what is ##her ##wood thought [SEP]

INFO:tensorflow:tokens: [CLS] hind ##sight is a wonderful thing . much rev ##iled when it first appeared , (inspiring the famous review ' me no lei ##ca ') , this precursor of " cabaret " can now be looked at in comparison and it ' s not half bad . it ' s certainly no classic but it has its own way ##ward charm , (the film version of " cabaret " follows this plot whereas the stage version changed the plot somewhat) . one should , of course , resist the temptation to s ##nick ##er when laurence harvey ' s christopher is ##her ##wood , (it keeps the original author ' s real name ; god knows what is ##her ##wood thought [SEP]

INFO:tensorflow:input_ids: 101 17666 25807 2003 1037 6919 2518 1012 2172 7065 18450 2043 2009 2034 2596 1010 1006 18988 1996 3297 3319 1005 2033 2053 26947 3540 1005 1007 1010 2023 14988 1997 1000 19685 1000 2064 2085 2022 2246 2012 1999 7831 1998 2009 1005 1055 2025 2431 2919 1012 2009 1005 1055 5121 2053 4438 2021 2009 2038 2049 2219 2126 7652 11084 1010 1006 1996 2143 2544 1997 1000 19685 1000 4076 2023 5436 6168 1996 2754 2544 2904 1996 5436 5399 1007 1012 2028 2323 1010 1997 2607 1010 9507 1996 17232 2000 1055 13542 2121 2043 10883 7702 1005 1055 5696 2003 5886 3702 1010 1006 2009 7906 1996 2434 3166 1005 1055 2613 2171 1025 2643 4282 2054 2003 5886 3702 2245 102

INFO:tensorflow:input_ids: 101 17666 25807 2003 1037 6919 2518 1012 2172 7065 18450 2043 2009 2034 2596 1010 1006 18988 1996 3297 3319 1005 2033 2053 26947 3540 1005 1007 1010 2023 14988 1997 1000 19685 1000 2064 2085 2022 2246 2012 1999 7831 1998 2009 1005 1055 2025 2431 2919 1012 2009 1005 1055 5121 2053 4438 2021 2009 2038 2049 2219 2126 7652 11084 1010 1006 1996 2143 2544 1997 1000 19685 1000 4076 2023 5436 6168 1996 2754 2544 2904 1996 5436 5399 1007 1012 2028 2323 1010 1997 2607 1010 9507 1996 17232 2000 1055 13542 2121 2043 10883 7702 1005 1055 5696 2003 5886 3702 1010 1006 2009 7906 1996 2434 3166 1005 1055 2613 2171 1025 2643 4282 2054 2003 5886 3702 2245 102

INFO:tensorflow:input_mask: 1

INFO:tensorflow:input_mask: 1

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:segment_ids: 0

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] i can confidently say that this is the worst film i have ever seen , and i usually love foreign films . the movie is nothing more than poorly - made violent pornography . if you choose to see it , prepare yourself for endless sex ##ism , gr ##at ##uit ##ous nu ##dity shots , and a stupid sensation ##ali ##zed rape scene , which i ' m sure is the main appeal for the people who like the movie . < br / > < br / > additionally , the plot mean ##ders aim ##lessly , and none of the characters is li ##ka ##ble . many scenes are filmed from the woods surrounding the river the girls are on to give a [SEP]

INFO:tensorflow:tokens: [CLS] i can confidently say that this is the worst film i have ever seen , and i usually love foreign films . the movie is nothing more than poorly - made violent pornography . if you choose to see it , prepare yourself for endless sex ##ism , gr ##at ##uit ##ous nu ##dity shots , and a stupid sensation ##ali ##zed rape scene , which i ' m sure is the main appeal for the people who like the movie . < br / > < br / > additionally , the plot mean ##ders aim ##lessly , and none of the characters is li ##ka ##ble . many scenes are filmed from the woods surrounding the river the girls are on to give a [SEP]

INFO:tensorflow:input_ids: 101 1045 2064 28415 2360 2008 2023 2003 1996 5409 2143 1045 2031 2412 2464 1010 1998 1045 2788 2293 3097 3152 1012 1996 3185 2003 2498 2062 2084 9996 1011 2081 6355 19378 1012 2065 2017 5454 2000 2156 2009 1010 7374 4426 2005 10866 3348 2964 1010 24665 4017 14663 3560 16371 25469 7171 1010 1998 1037 5236 8742 11475 5422 9040 3496 1010 2029 1045 1005 1049 2469 2003 1996 2364 5574 2005 1996 2111 2040 2066 1996 3185 1012 1026 7987 1013 1028 1026 7987 1013 1028 5678 1010 1996 5436 2812 13375 6614 10895 1010 1998 3904 1997 1996 3494 2003 5622 2912 3468 1012 2116 5019 2024 6361 2013 1996 5249 4193 1996 2314 1996 3057 2024 2006 2000 2507 1037 102

INFO:tensorflow:input_ids: 101 1045 2064 28415 2360 2008 2023 2003 1996 5409 2143 1045 2031 2412 2464 1010 1998 1045 2788 2293 3097 3152 1012 1996 3185 2003 2498 2062 2084 9996 1011 2081 6355 19378 1012 2065 2017 5454 2000 2156 2009 1010 7374 4426 2005 10866 3348 2964 1010 24665 4017 14663 3560 16371 25469 7171 1010 1998 1037 5236 8742 11475 5422 9040 3496 1010 2029 1045 1005 1049 2469 2003 1996 2364 5574 2005 1996 2111 2040 2066 1996 3185 1012 1026 7987 1013 1028 1026 7987 1013 1028 5678 1010 1996 5436 2812 13375 6614 10895 1010 1998 3904 1997 1996 3494 2003 5622 2912 3468 1012 2116 5019 2024 6361 2013 1996 5249 4193 1996 2314 1996 3057 2024 2006 2000 2507 1037 102

INFO:tensorflow:input_mask: 1

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

INFO:tensorflow:segment_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 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
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 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:segment_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 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
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 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:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] hilarious , sellers at his fun ##nies ##t . . . a shame you can ' t get this on video , or even see it on tv anymore . . . i ' d love to get a good copy somewhere . maybe it ' s tied up in court on some legal issue , but a truly riot ##ous hospital far ##ce with sellers as crooked administrator . [SEP]

INFO:tensorflow:tokens: [CLS] hilarious , sellers at his fun ##nies ##t . . . a shame you can ' t get this on video , or even see it on tv anymore . . . i ' d love to get a good copy somewhere . maybe it ' s tied up in court on some legal issue , but a truly riot ##ous hospital far ##ce with sellers as crooked administrator . [SEP]

INFO:tensorflow:input_ids: 101 26316 1010 19041 2012 2010 4569 15580 2102 1012
1012 1012 1037 9467 2017 2064 1005 1056 2131 2023 2006 2678 1010 2030 2130 2156
2009 2006 2694 4902 1012 1012 1012 1045 1005 1040 2293 2000 2131 1037 2204 6100
4873 1012 2672 2009 1005 1055 5079 2039 1999 2457 2006 2070 3423 3277 1010 2021
1037 5621 11421 3560 2902 2521 3401 2007 19041 2004 15274 8911 1012 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 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:input_ids: 101 26316 1010 19041 2012 2010 4569 15580 2102 1012
1012 1012 1037 9467 2017 2064 1005 1056 2131 2023 2006 2678 1010 2030 2130 2156
2009 2006 2694 4902 1012 1012 1012 1045 1005 1040 2293 2000 2131 1037 2204 6100
4873 1012 2672 2009 1005 1055 5079 2039 1999 2457 2006 2070 3423 3277 1010 2021
1037 5621 11421 3560 2902 2521 3401 2007 19041 2004 15274 8911 1012 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 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:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 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:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 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:segment_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 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
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 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:segment_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 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
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 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:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)
```

#Creating a model

Now that we've prepared our data, let's focus on building a model. `create_model` does just this below. First, it loads the BERT tf hub module again (this time to extract the computation graph). Next, it creates a single new layer that will be trained to adapt BERT to our sentiment task (i.e. classifying whether a movie review is positive or negative). This strategy of using a mostly trained model is called fine-tuning.

[12]:
```python
def create_model(is_predicting, input_ids, input_mask, segment_ids, labels,
                 num_labels):
  """Creates a classification model."""

  bert_module = hub.Module(
      BERT_MODEL_HUB,
      trainable=True)
  bert_inputs = dict(
      input_ids=input_ids,
      input_mask=input_mask,
      segment_ids=segment_ids)
  bert_outputs = bert_module(
      inputs=bert_inputs,
      signature="tokens",
      as_dict=True)

  # Use "pooled_output" for classification tasks on an entire sentence.
  # Use "sequence_outputs" for token-level output.
  output_layer = bert_outputs["pooled_output"]
```

```
  hidden_size = output_layer.shape[-1].value

  # Create our own layer to tune for politeness data.
  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"):

    # Dropout helps prevent overfitting
    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)
    log_probs = tf.nn.log_softmax(logits, axis=-1)

    # Convert labels into one-hot encoding
    one_hot_labels = tf.one_hot(labels, depth=num_labels, dtype=tf.float32)

    predicted_labels = tf.squeeze(tf.argmax(log_probs, axis=-1, output_type=tf.
↪int32))
    # If we're predicting, we want predicted labels and the probabiltiies.
    if is_predicting:
      return (predicted_labels, log_probs)

    # If we're train/eval, compute loss between predicted and actual label
    per_example_loss = -tf.reduce_sum(one_hot_labels * log_probs, axis=-1)
    loss = tf.reduce_mean(per_example_loss)
    return (loss, predicted_labels, log_probs)
```

Next we'll wrap our model function in a `model_fn_builder` function that adapts our model to work for training, evaluation, and prediction.

[13]:
```
# model_fn_builder actually creates our model function
# using the passed parameters for num_labels, learning_rate, etc.
def model_fn_builder(num_labels, learning_rate, num_train_steps,
                     num_warmup_steps):
  """Returns `model_fn` closure for TPUEstimator."""
  def model_fn(features, labels, mode, params):  # pylint:␣
↪disable=unused-argument
    """The `model_fn` for TPUEstimator."""

    input_ids = features["input_ids"]
```

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

    is_predicting = (mode == tf.estimator.ModeKeys.PREDICT)

    # TRAIN and EVAL
    if not is_predicting:

      (loss, predicted_labels, log_probs) = create_model(
        is_predicting, input_ids, input_mask, segment_ids, label_ids, num_labels)

      train_op = bert.optimization.create_optimizer(
          loss, learning_rate, num_train_steps, num_warmup_steps, use_tpu=False)

      # Calculate evaluation metrics.
      def metric_fn(label_ids, predicted_labels):
        accuracy = tf.metrics.accuracy(label_ids, predicted_labels)
        f1_score = tf.contrib.metrics.f1_score(
            label_ids,
            predicted_labels)
        auc = tf.metrics.auc(
            label_ids,
            predicted_labels)
        recall = tf.metrics.recall(
            label_ids,
            predicted_labels)
        precision = tf.metrics.precision(
            label_ids,
            predicted_labels)
        true_pos = tf.metrics.true_positives(
            label_ids,
            predicted_labels)
        true_neg = tf.metrics.true_negatives(
            label_ids,
            predicted_labels)
        false_pos = tf.metrics.false_positives(
            label_ids,
            predicted_labels)
        false_neg = tf.metrics.false_negatives(
            label_ids,
            predicted_labels)
        return {
            "eval_accuracy": accuracy,
            "f1_score": f1_score,
            "auc": auc,
            "precision": precision,
```

```
            "recall": recall,
            "true_positives": true_pos,
            "true_negatives": true_neg,
            "false_positives": false_pos,
            "false_negatives": false_neg
        }

      eval_metrics = metric_fn(label_ids, predicted_labels)

      if mode == tf.estimator.ModeKeys.TRAIN:
        return tf.estimator.EstimatorSpec(mode=mode,
          loss=loss,
          train_op=train_op)
      else:
          return tf.estimator.EstimatorSpec(mode=mode,
            loss=loss,
            eval_metric_ops=eval_metrics)
    else:
      (predicted_labels, log_probs) = create_model(
        is_predicting, input_ids, input_mask, segment_ids, label_ids, num_labels)

      predictions = {
          'probabilities': log_probs,
          'labels': predicted_labels
      }
      return tf.estimator.EstimatorSpec(mode, predictions=predictions)

  # Return the actual model function in the closure
  return model_fn
```

[14]:
```
# Compute train and warmup steps from batch size
# These hyperparameters are copied from this colab notebook (https://colab.
 ↪sandbox.google.com/github/tensorflow/tpu/blob/master/tools/colab/
 ↪bert_finetuning_with_cloud_tpus.ipynb)
BATCH_SIZE = 32
LEARNING_RATE = 2e-5
NUM_TRAIN_EPOCHS = 2.0
# 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 = 500
SAVE_SUMMARY_STEPS = 100
```

[15]:
```
# Compute # train and warmup steps from batch size
num_train_steps = int(len(train_features) / BATCH_SIZE * NUM_TRAIN_EPOCHS)
num_warmup_steps = int(num_train_steps * WARMUP_PROPORTION)
```

[16]:
```
# Specify outpit directory and number of checkpoint steps to save
OUTPUT_DIR="C:\\Local Drive D\\Heyt\\Chicago College Docs\\Sem 3\\CS 577 - Deep␣
 ↪Learning\\Project\\bert-master\\moviepredictionoutput\\"
run_config = tf.estimator.RunConfig(
    model_dir=OUTPUT_DIR,
    save_summary_steps=SAVE_SUMMARY_STEPS,
    save_checkpoints_steps=SAVE_CHECKPOINTS_STEPS)
```

[17]:
```
model_fn = model_fn_builder(
  num_labels=len(label_list),
  learning_rate=LEARNING_RATE,
  num_train_steps=num_train_steps,
  num_warmup_steps=num_warmup_steps)

estimator = tf.estimator.Estimator(
  model_fn=model_fn,
  config=run_config,
  params={"batch_size": BATCH_SIZE})
```

```
INFO:tensorflow:Using config: {'_model_dir': 'C:\\Local Drive D\\Heyt\\Chicago
College Docs\\Sem 3\\CS 577 - Deep Learning\\Project\\bert-
master\\moviepredictionoutput\\', '_tf_random_seed': None,
'_save_summary_steps': 100, '_save_checkpoints_steps': 500,
'_save_checkpoints_secs': None, '_session_config': allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None,
'_protocol': None, '_eval_distribute': None, '_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x0000015A70608748>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '',
'_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}

INFO:tensorflow:Using config: {'_model_dir': 'C:\\Local Drive D\\Heyt\\Chicago
College Docs\\Sem 3\\CS 577 - Deep Learning\\Project\\bert-
master\\moviepredictionoutput\\', '_tf_random_seed': None,
'_save_summary_steps': 100, '_save_checkpoints_steps': 500,
'_save_checkpoints_secs': None, '_session_config': allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
```

```
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None,
'_protocol': None, '_eval_distribute': None, '_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x0000015A70608748>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '',
'_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

Next we create an input builder function that takes our training feature set (`train_features`) and produces a generator. This is a pretty standard design pattern for working with Tensorflow Estimators.

[18]:
```
# Create an input function for training. drop_remainder = True for using TPUs.
train_input_fn = bert.run_classifier.input_fn_builder(
    features=train_features,
    seq_length=MAX_SEQ_LENGTH,
    is_training=True,
    drop_remainder=False)
```

Now we train our model! For me, using a Colab notebook running on Google's GPUs, my training time was about 14 minutes.

[19]:
```
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!
WARNING:tensorflow:From
C:\Users\DELL\AppData\Local\Programs\Python\Python36\lib\site-
packages\tensorflow_core\python\training\training_util.py:236:
Variable.initialized_value (from tensorflow.python.ops.variables) is deprecated
and will be removed in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in
eager and graph (inside tf.defun) contexts.

WARNING:tensorflow:From
C:\Users\DELL\AppData\Local\Programs\Python\Python36\lib\site-
packages\tensorflow_core\python\training\training_util.py:236:
Variable.initialized_value (from tensorflow.python.ops.variables) is deprecated
and will be removed in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in
eager and graph (inside tf.defun) contexts.

INFO:tensorflow:Calling model_fn.
```

```
INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Saver not created because there are no variables in the graph to
restore

INFO:tensorflow:Saver not created because there are no variables in the graph to
restore

WARNING:tensorflow:From <ipython-input-12-ca03218f28a6>:34: calling dropout
(from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.

WARNING:tensorflow:From <ipython-input-12-ca03218f28a6>:34: calling dropout
(from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.

WARNING:tensorflow:From
C:\Users\DELL\AppData\Local\Programs\Python\Python36\lib\site-
packages\tensorflow_core\python\ops\math_grad.py:1375: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

WARNING:tensorflow:From
C:\Users\DELL\AppData\Local\Programs\Python\Python36\lib\site-
packages\tensorflow_core\python\ops\math_grad.py:1375: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
C:\Users\DELL\AppData\Local\Programs\Python\Python36\lib\site-
packages\tensorflow_core\python\framework\indexed_slices.py:424: UserWarning:
Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may
consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-
sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.
```

```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-
sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.


WARNING:tensorflow:From
C:\Users\DELL\AppData\Local\Programs\Python\Python36\lib\site-
packages\tensorflow_core\contrib\metrics\python\metrics\classification.py:162:
div (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.

WARNING:tensorflow:From
C:\Users\DELL\AppData\Local\Programs\Python\Python36\lib\site-
packages\tensorflow_core\contrib\metrics\python\metrics\classification.py:162:
div (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Create CheckpointSaverHook.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Saving checkpoints for 0 into C:\Local Drive D\Heyt\Chicago
College Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt.

INFO:tensorflow:Saving checkpoints for 0 into C:\Local Drive D\Heyt\Chicago
College Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt.
```

```
INFO:tensorflow:loss = 0.7056151, step = 1

INFO:tensorflow:loss = 0.7056151, step = 1

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 4 vs previous value: 4. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 4 vs previous value: 4. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 10 vs previous value: 10. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 10 vs previous value: 10. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 31 vs previous value: 31. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 31 vs previous value: 31. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 34 vs previous value: 34. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 34 vs previous value: 34. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 44 vs previous value: 44. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.

WARNING:tensorflow:It seems that global step (tf.train.get_global_step) has not
been increased. Current value (could be stable): 44 vs previous value: 44. You
could increase the global step by passing tf.train.get_global_step() to
Optimizer.apply_gradients or Optimizer.minimize.
```

```
INFO:tensorflow:global_step/sec: 0.0246969

INFO:tensorflow:global_step/sec: 0.0246969

INFO:tensorflow:loss = 0.6703781, step = 101 (4049.275 sec)

INFO:tensorflow:loss = 0.6703781, step = 101 (4049.275 sec)

INFO:tensorflow:global_step/sec: 0.0305564

INFO:tensorflow:global_step/sec: 0.0305564

INFO:tensorflow:loss = 0.037343323, step = 201 (3274.839 sec)

INFO:tensorflow:loss = 0.037343323, step = 201 (3274.839 sec)

INFO:tensorflow:global_step/sec: 0.0337853

INFO:tensorflow:global_step/sec: 0.0337853

INFO:tensorflow:loss = 0.13788503, step = 301 (2957.577 sec)

INFO:tensorflow:loss = 0.13788503, step = 301 (2957.577 sec)

INFO:tensorflow:Saving checkpoints for 312 into C:\Local Drive D\Heyt\Chicago
College Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt.

INFO:tensorflow:Saving checkpoints for 312 into C:\Local Drive D\Heyt\Chicago
College Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt.

INFO:tensorflow:Loss for final step: 0.16980839.

INFO:tensorflow:Loss for final step: 0.16980839.

Training took time  2:59:34.839346
```

Now let's use our test data to see how well our model did:

[20]:
```
test_input_fn = run_classifier.input_fn_builder(
    features=test_features,
    seq_length=MAX_SEQ_LENGTH,
    is_training=False,
    drop_remainder=False)
```

[21]:
```
estimator.evaluate(input_fn=test_input_fn, steps=None)
```

```
INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Saver not created because there are no variables in the graph to
restore

INFO:tensorflow:Saver not created because there are no variables in the graph to
restore
C:\Users\DELL\AppData\Local\Programs\Python\Python36\lib\site-
```

```
packages\tensorflow_core\python\framework\indexed_slices.py:424: UserWarning:
Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may
consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Starting evaluation at 2024-11-17T14:23:54Z

INFO:tensorflow:Starting evaluation at 2024-11-17T14:23:54Z

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from C:\Local Drive D\Heyt\Chicago College
Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt-312

INFO:tensorflow:Restoring parameters from C:\Local Drive D\Heyt\Chicago College
Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt-312

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Finished evaluation at 2024-11-17-14:49:04

INFO:tensorflow:Finished evaluation at 2024-11-17-14:49:04

INFO:tensorflow:Saving dict for global step 312: auc = 0.8576986, eval_accuracy
= 0.8574, f1_score = 0.85451937, false_negatives = 432.0, false_positives =
281.0, global_step = 312, loss = 0.4057239, precision = 0.8816842, recall =
0.8289786, true_negatives = 2193.0, true_positives = 2094.0

INFO:tensorflow:Saving dict for global step 312: auc = 0.8576986, eval_accuracy
= 0.8574, f1_score = 0.85451937, false_negatives = 432.0, false_positives =
281.0, global_step = 312, loss = 0.4057239, precision = 0.8816842, recall =
0.8289786, true_negatives = 2193.0, true_positives = 2094.0

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 312: C:\Local
Drive D\Heyt\Chicago College Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt-312

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 312: C:\Local
Drive D\Heyt\Chicago College Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt-312
```

[21]:
```
{'auc': 0.8576986,
 'eval_accuracy': 0.8574,
 'f1_score': 0.85451937,
 'false_negatives': 432.0,
 'false_positives': 281.0,
 'loss': 0.4057239,
 'precision': 0.8816842,
 'recall': 0.8289786,
 'true_negatives': 2193.0,
 'true_positives': 2094.0,
 'global_step': 312}
```

Now let's write code to make predictions on new sentences:

[22]:
```
predictions = list(estimator.predict(input_fn=test_input_fn))
predicted_labels = [p['labels'] for p in predictions]
true_labels = test[LABEL_COLUMN].values
cm = confusion_matrix(true_labels, predicted_labels)
print(classification_report(true_labels, predicted_labels,␣
 ↪target_names=["Negative", "Positive"]))
```

```
INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Saver not created because there are no variables in the graph to
restore

INFO:tensorflow:Saver not created because there are no variables in the graph to
restore

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from C:\Local Drive D\Heyt\Chicago College
Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt-312

INFO:tensorflow:Restoring parameters from C:\Local Drive D\Heyt\Chicago College
Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt-312

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.
```

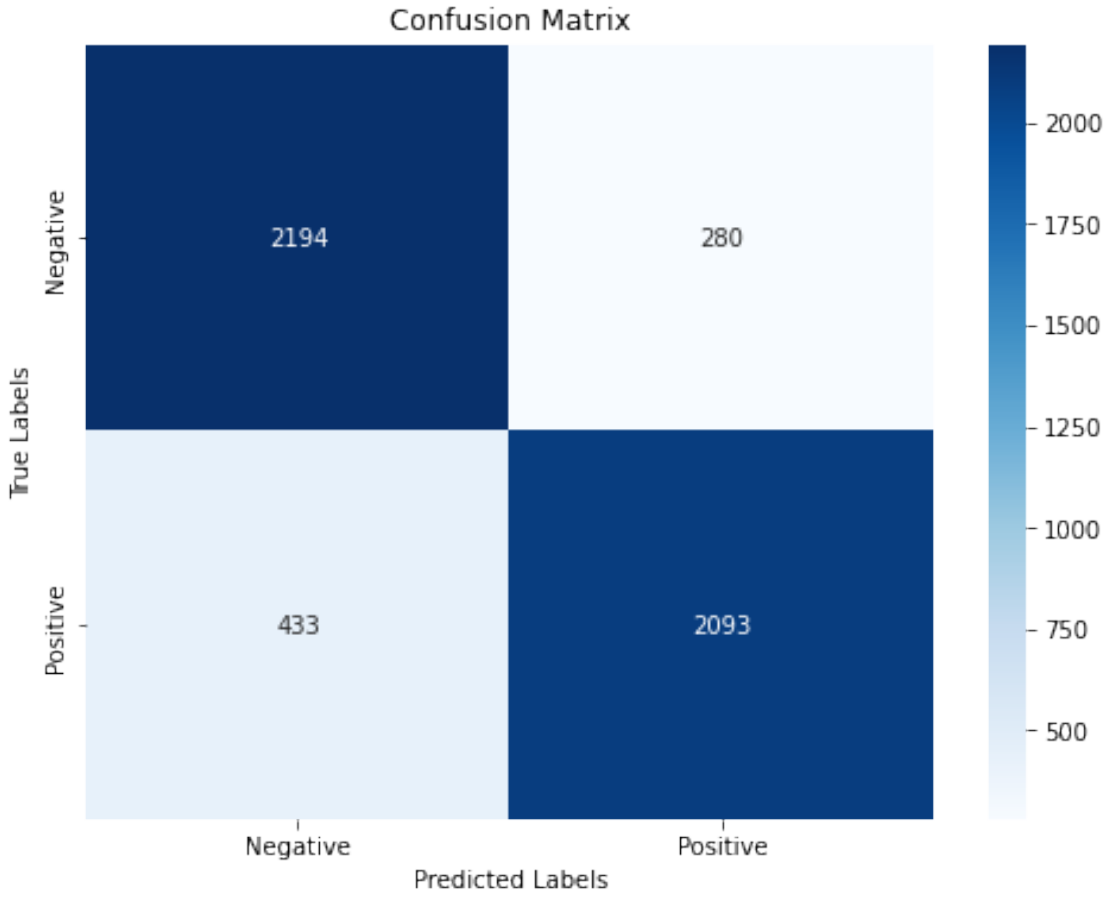
```
                precision    recall  f1-score   support

    Negative       0.84      0.89      0.86      2474
    Positive       0.88      0.83      0.85      2526

    accuracy                           0.86      5000
   macro avg       0.86      0.86      0.86      5000
weighted avg       0.86      0.86      0.86      5000
```

[25]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Step 4: Visualize the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Negative",␣
 ↪"Positive"], yticklabels=["Negative", "Positive"])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



[26]:
```
def getPrediction(in_sentences):
  labels = ["Negative", "Positive"]
  input_examples = [run_classifier.InputExample(guid="", text_a = x, text_b =␣
 ↪None, label = 0) for x in in_sentences] # here, "" is just a dummy label
  input_features = run_classifier.convert_examples_to_features(input_examples,␣
 ↪label_list, MAX_SEQ_LENGTH, tokenizer)
  predict_input_fn = run_classifier.input_fn_builder(features=input_features,␣
 ↪seq_length=MAX_SEQ_LENGTH, is_training=False, drop_remainder=False)
  predictions = estimator.predict(predict_input_fn)
  return [(sentence, prediction['probabilities'], labels[prediction['labels']])␣
 ↪for sentence, prediction in zip(in_sentences, predictions)]
```

[27]:
```
def visualize_predictions(predictions, sentences):
    labels = ["Negative", "Positive"]
    for i, (sentence, probabilities, predicted_label) in enumerate(predictions):
        # Bar chart for probabilities
        plt.figure(figsize=(6, 4))
        plt.bar(labels, probabilities, color=['red', 'green'])
        plt.title(f"Sentence {i + 1}: '{sentence}'\nPredicted:␣
 ↪{predicted_label}")
        plt.ylabel('Probability')
        plt.ylim(0, 1)
        plt.show()
```

[28]:
```
pred_sentences = [
  "That movie was absolutely awful",
  "The acting was a bit lacking",
  "The film was creative and surprising",
  "Absolutely fantastic!"
]
```

[29]:
```
predictions = getPrediction(pred_sentences)
```

```
INFO:tensorflow:Writing example 0 of 4

INFO:tensorflow:Writing example 0 of 4

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid:

INFO:tensorflow:guid:

INFO:tensorflow:tokens: [CLS] that movie was absolutely awful [SEP]

INFO:tensorflow:tokens: [CLS] that movie was absolutely awful [SEP]
```

```
INFO:tensorflow:input_ids: 101 2008 3185 2001 7078 9643 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 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 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 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

INFO:tensorflow:input_ids: 101 2008 3185 2001 7078 9643 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 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 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 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

INFO:tensorflow:input_mask: 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 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 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 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 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 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 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 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 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 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_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 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 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 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 0 0 0 0 0 0

INFO:tensorflow:segment_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 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 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 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 0 0 0 0 0 0

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid:

INFO:tensorflow:guid:

INFO:tensorflow:tokens: [CLS] the acting was a bit lacking [SEP]

INFO:tensorflow:tokens: [CLS] the acting was a bit lacking [SEP]

INFO:tensorflow:input_ids: 101 1996 3772 2001 1037 2978 11158 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 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 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 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:input_ids: 101 1996 3772 2001 1037 2978 11158 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 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 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 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: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
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 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 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 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: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
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 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 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 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:segment_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 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 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 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 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:segment_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 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 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 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 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:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid:

INFO:tensorflow:guid:

INFO:tensorflow:tokens: [CLS] the film was creative and surprising [SEP]

INFO:tensorflow:tokens: [CLS] the film was creative and surprising [SEP]

INFO:tensorflow:input_ids: 101 1996 2143 2001 5541 1998 11341 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 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 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 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 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_ids: 101 1996 2143 2001 5541 1998 11341 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 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 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 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 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
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 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 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 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: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
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 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 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 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:segment_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 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
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 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:segment_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 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
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 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:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid:

INFO:tensorflow:guid:

INFO:tensorflow:tokens: [CLS] absolutely fantastic ! [SEP]

INFO:tensorflow:tokens: [CLS] absolutely fantastic ! [SEP]

INFO:tensorflow:input_ids: 101 7078 10392 999 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 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 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 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

INFO:tensorflow:input_ids: 101 7078 10392 999 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 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 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 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

INFO:tensorflow:input_mask: 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 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 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 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

INFO:tensorflow:input_mask: 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 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 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 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

INFO:tensorflow:segment_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 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
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 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:segment_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 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
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 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:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Saver not created because there are no variables in the graph to
restore

INFO:tensorflow:Saver not created because there are no variables in the graph to
restore

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from C:\Local Drive D\Heyt\Chicago College
Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt-312

INFO:tensorflow:Restoring parameters from C:\Local Drive D\Heyt\Chicago College
Docs\Sem 3\CS 577 - Deep Learning\Project\bert-
master\moviepredictionoutput\model.ckpt-312

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.
```

Voila! We have a sentiment classifier!

[30]:
```
predictions
```

[30]:
```
[('That movie was absolutely awful',
  array([-5.027628e-03, -5.295315e+00], dtype=float32),
  'Negative'),
 ('The acting was a bit lacking',
  array([-0.17878766, -1.8096187 ], dtype=float32),
  'Negative'),
 ('The film was creative and surprising',
  array([-5.6216540e+00, -3.6252511e-03], dtype=float32),
  'Positive'),
 ('Absolutely fantastic!',
  array([-5.0698276 , -0.00630331], dtype=float32),
  'Positive')]
```

10 Applications and Use Cases

BERT has significantly transformed natural language processing (NLP) by providing a robust pre-trained model applicable to a variety of tasks. It has set new state-of-the-art benchmarks in token-level tasks like named entity recognition and question answering (e.g., SQuAD) and in sentence-level tasks such as sentiment analysis and natural language inference. BERT's architecture, which fuses bidirectional contextual information from text, eliminates the need for extensive task-specific feature engineering. This versatility has made it a go-to solution for both academic and industrial applications, enabling advancements in chatbots, text summarization, and machine translation.

11 Challenges and Limitations

While BERT is a powerful model, it comes with several challenges. The model's pre-training process is computationally expensive, requiring significant resources such as multiple TPUs over several days. This creates barriers for researchers and developers without access to such infrastructure. Additionally, fine-tuning BERT for domain-specific applications, such as legal or medical text, often demands substantial labeled data, which may not always be readily available. Its size and memory requirements can also hinder deployment in environments with limited computational resources, such as mobile or edge devices. Moreover, the bidirectional masking strategy introduces a pre-training and fine-tuning mismatch, which impacts its learning efficiency.

12 Future Work

Future research directions for BERT include improving its computational efficiency and scalability. Efforts are underway to develop lightweight versions, such as distillation-based models, that retain performance while reducing resource demands. Expanding BERT's adaptability to low-resource languages and domains through more effective transfer learning techniques is another promising area. Moreover, advancements in fine-tuning strategies could enable better utilization of small, task-specific datasets, making the model accessible to a broader range of applications. Finally, integrating BERT with other modalities, such as vision or speech, represents a frontier for creating more versatile AI systems.

13 Conclusion

BERT marks a pivotal advancement in NLP, showcasing the power of bidirectional pre-trained models in solving complex language tasks. Its ability to generalize across diverse applications with minimal task-specific adjustments has redefined the field. However, challenges such as high computational costs and domain-specific limitations highlight the need for further innovation. Addressing these limitations while expanding BERT's applicability will continue to shape its role in the future of NLP and beyond.

14 References

Bert Github Repository: BERT GitHub Repository

MRPC Data: MRPC Data Github Repository

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.