BERT on SQUAD Dataset

- Most of the code is taken referance from google-research
- The bert model is **fine-tuned** only.
- Code has been modified as per necesscity
- Used the **bert uncased_L-12_H-768_A-12** model
- All the referance are mentioned in the referances section

In [6]:

```
# importing all necessary files

import zipfile
from matplotlib import pyplot as plt
%matplotlib inline
import sys
import datetime
import tensorflow as tf
import os
import json
import six

from collections import Counter
import string
import re
import argparse
import sys
```

Configuring TPU

This section, you perform the following tasks:

- Set up a Colab TPU running environment
- Verify that you are connected to a TPU device

Once done. Upload your credentials to TPU to access your GCS bucket.

In [0]:

```
import datetime
import json
import os
import pprint
import random
import string
import sys
import tensorflow as tf

assert 'COLAB_TPU_ADDR' in os.environ, 'ERROR: Not connected to a TPU runtime; please see the first cell in this notebook for instructions!'
TPU_ADDRESS = 'grpc://' + os.environ['COLAB_TPU_ADDR']
print('TPU address is', TPU_ADDRESS)

from google.colab import auth
auth.authenticate_user()
with tf.Session(TPU_ADDRESS) as session:
  print('TPU devices:')
  pprint.pprint(session.list_devices())

  # Upload credentials to TPU.
  with open('/content/adc.json', 'r') as f:
    auth_info = json.load(f)
  tf.contrib.cloud.configure_gcs(session, credentials=auth_info)
  # Now credentials are set for all future sessions on this TPU.
```

```
TPU address is grpc://10.12.236.42:8470

WARNING: Logging before flag parsing goes to stderr.
W0827 05:51:10.907844 140491252946816 lazy_loader.py:50]
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.


TPU devices:
[_DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:CPU:0, CPU, -1, 11
440417705514041917),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:XLA_CPU:0, XLA_CP
U, 17179869184, 11361402579519159698),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU:0, TPU, 171798
69184, 12357697597335777241),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU:1, TPU, 171798
69184, 7898279176896720777),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU:2, TPU, 171798
69184, 16769548510779700091),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU:3, TPU, 171798
69184, 9684062799403801468),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU:4, TPU, 171798
69184, 18088205161317314166),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU:5, TPU, 171798
69184, 2719725766142665814),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU:6, TPU, 171798
69184, 12910852574047901557),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU:7, TPU, 171798
69184, 9410472538681109359),
 _DeviceAttributes(/job:tpu_worker/replica:0/task:0/device:TPU_SYSTEM:0, TPU_
SYSTEM, 8589934592, 15068353228979054653)]
```

import necessary BERT modules

In [0]:

```
import sys

!test -d bert_repo || git clone https://github.com/google-research/bert bert_repo
if not 'bert_repo' in sys.path:
  sys.path += ['bert_repo']

# import python modules defined by BERT
import run_squad
import modeling
import optimization
import tokenization
import tensorflow as tf
import tokenization

# import tfhub
import tensorflow_hub as hub
```

```
Cloning into 'bert_repo'...
remote: Enumerating objects: 333, done.
remote: Total 333 (delta 0), reused 0 (delta 0), pack-reused 333
Receiving objects: 100% (333/333), 282.46 KiB | 3.98 MiB/s, done.
Resolving deltas: 100% (183/183), done.

W0827 06:05:06.749072 140491252946816 deprecation_wrapper.py:119] From bert_repo/optimization.py:87: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

This next section of code performs the following tasks:

- Specify BERT pretrained model [uncased_L-12_H-768_A-12]
- Create output directory for model checkpoints and eval results.

In [0]:

```
TASK = 'SQUAD' #@param {type:"string"}
assert TASK in ('MRPC', 'CoLA',"SQUAD"), 'Only (MRPC, CoLA) are demonstrated h
ere.'


BUCKET = 'bert-on-squad' #@param {type:"string"}
assert BUCKET, 'Must specify an existing GCS bucket name'
OUTPUT_DIR = 'gs://{}/bert-tfhub/models/{}'.format(BUCKET, TASK)
tf.gfile.MakeDirs(OUTPUT_DIR)
print('***** Model output directory: {} *****'.format(OUTPUT_DIR))

# Available pretrained model checkpoints:
#   uncased_L-12_H-768_A-12: uncased BERT base model
#   uncased_L-24_H-1024_A-16: uncased BERT large model
#   cased_L-12_H-768_A-12: cased BERT large model
BERT_MODEL = 'uncased_L-12_H-768_A-12' #@param {type:"string"}
BERT_MODEL_HUB = 'https://tfhub.dev/google/bert_' + BERT_MODEL + '/1'
```

```
***** Model output directory: gs://bert-on-squad/bert-tfhub/models/SQUAD ****
*
```

In [0]:

```
# Setup TPU related config
tpu_cluster_resolver = tf.contrib.cluster_resolver.TPUClusterResolver(TPU_ADDR
ESS)
NUM_TPU_CORES = 8
ITERATIONS_PER_LOOP = 1000
```

In [0]:

```
# Setup task specific model and TPU running config.
BERT_PRETRAINED_DIR = 'gs://cloud-tpu-checkpoints/bert/' + BERT_MODEL
print('***** BERT pretrained directory: {} *****'.format(BERT_PRETRAINED_DIR))
!gsutil ls $BERT_PRETRAINED_DIR
```

```
***** BERT pretrained directory: gs://cloud-tpu-checkpoints/bert/uncased_L-12
_H-768_A-12 *****
gs://cloud-tpu-checkpoints/bert/uncased_L-12_H-768_A-12/bert_config.json
gs://cloud-tpu-checkpoints/bert/uncased_L-12_H-768_A-12/bert_model.ckpt.data-
00000-of-00001
gs://cloud-tpu-checkpoints/bert/uncased_L-12_H-768_A-12/bert_model.ckpt.index
gs://cloud-tpu-checkpoints/bert/uncased_L-12_H-768_A-12/bert_model.ckpt.meta
gs://cloud-tpu-checkpoints/bert/uncased_L-12_H-768_A-12/checkpoint
gs://cloud-tpu-checkpoints/bert/uncased_L-12_H-768_A-12/vocab.txt
```

In [0]:

```
# seting up necessary path files for models
CONFIG_FILE = os.path.join(BERT_PRETRAINED_DIR, 'bert_config.json')
INIT_CHECKPOINT = os.path.join(BERT_PRETRAINED_DIR, 'bert_model.ckpt')
VOCAB_FILE = os.path.join(BERT_PRETRAINED_DIR, 'vocab.txt')
DO_LOWER_CASE = BERT_MODEL.startswith('uncased')
```

In [0]:

```
OUTPUT_DIR = OUTPUT_DIR.replace('bert-tfhub', 'bert-checkpoints')
tf.gfile.MakeDirs(OUTPUT_DIR)
```

Get SQUAD Dataset

In [0]:
```
with zipfile.ZipFile("/content/train-v1.1.json.zip","r") as zip_ref:
    zip_ref.extractall()

with zipfile.ZipFile("/content/dev-v1.1.json.zip","r") as zip_ref:
    zip_ref.extractall()
```

Important Functions:

read_squad_examples : This function takes input as **json** file and retunrs the object of SquadExample claas. Which is nothing but, container storinng our each data point.

In [0]:

```
def read_squad_examples(input_file, is_training):
  """Read a SQuAD json file into a list of SquadExample."""
  with tf.gfile.Open(input_file, "r") as reader:
    input_data = json.load(reader)["data"]

  def is_whitespace(c):
    if c == " " or c == "\t" or c == "\r" or c == "\n" or ord(c) == 0x202F:
      return True
    return False

  examples = []
  for entry in input_data:
    for paragraph in entry["paragraphs"]:
      paragraph_text = paragraph["context"]
      doc_tokens = []
      char_to_word_offset = []
      prev_is_whitespace = True
      for c in paragraph_text:
        if is_whitespace(c):
          prev_is_whitespace = True
        else:
          if prev_is_whitespace:
            doc_tokens.append(c)
          else:
            doc_tokens[-1] += c
          prev_is_whitespace = False
        char_to_word_offset.append(len(doc_tokens) - 1)

      for qa in paragraph["qas"]:
        qas_id = qa["id"]
        question_text = qa["question"]
        start_position = None
        end_position = None
        orig_answer_text = None
        is_impossible = False
        if is_training:

          if False:
            is_impossible = qa["is_impossible"]
          if (len(qa["answers"]) != 1) and (not is_impossible):
            raise ValueError(
                "For training, each question should have exactly 1 answer.")
          if not is_impossible:
            answer = qa["answers"][0]
            orig_answer_text = answer["text"]
            answer_offset = answer["answer_start"]
            answer_length = len(orig_answer_text)
            start_position = char_to_word_offset[answer_offset]
            end_position = char_to_word_offset[answer_offset + answer_length -
                                                1]
            # Only add answers where the text can be exactly recovered from th
e
            # document. If this CAN'T happen it's likely due to weird Unicode
            # stuff so we will just skip the example.
            #
            # Note that this means for training mode, every example is NOT
            # guaranteed to be preserved.
            actual_text = " ".join(
                doc_tokens[start_position:(end_position + 1)])
            cleaned_answer_text = " ".join(
                tokenization.whitespace_tokenize(orig_answer_text))
            if actual_text.find(cleaned_answer_text) == -1:
              tf.logging.warning("Could not find answer: '%s' vs. '%s'",
                                 actual_text, cleaned_answer_text)
              continue
          else:
            start_position = -1
            end_position = -1
            orig_answer_text = ""

        example = run_squad.SquadExample(
            qas_id=qas_id,
            question_text=question_text,
            doc_tokens=doc_tokens,
            orig_answer_text=orig_answer_text,
            start_position=start_position,
            end_position=end_position,
            is_impossible=is_impossible)
        examples.append(example)

  return examples
```

In [0]:

```
train_examples = read_squad_examples("/content/train-v1.1.json",True)
print("Total train examples are ",len(train_examples))
```

```
Total train examples are  87599
```

In [0]:
```
print("Type of return :\n",type(train_examples[0]))
print("*"*60)
print("Question ID : ",train_examples[0].qas_id)
print("*"*60)
print("question_text : ",train_examples[0].question_text)
print("*"*60)
print("doc_tokens : ",train_examples[0].doc_tokens)
print("*"*60)
print("start_position : ",train_examples[0].start_position)
print("end_position : ",train_examples[0].end_position)
print("*"*60)
print("is_impossible : ",train_examples[0].is_impossible)
```

```
Type of return :
 <class 'run_squad.SquadExample'>
************************************************************
Question ID :  5733be284776f41900661182
************************************************************
question_text :  To whom did the Virgin Mary allegedly appear in 1858 in Lour
des France?
************************************************************
doc_tokens :  ['Architecturally,', 'the', 'school', 'has', 'a', 'Catholic',
'character.', 'Atop', 'the', 'Main', "Building's", 'gold', 'dome', 'is', 'a',
'golden', 'statue', 'of', 'the', 'Virgin', 'Mary.', 'Immediately', 'in', 'fro
nt', 'of', 'the', 'Main', 'Building', 'and', 'facing', 'it,', 'is', 'a', 'cop
per', 'statue', 'of', 'Christ', 'with', 'arms', 'upraised', 'with', 'the', 'l
egend', '"Venite', 'Ad', 'Me', 'Omnes".', 'Next', 'to', 'the', 'Main', 'Build
ing', 'is', 'the', 'Basilica', 'of', 'the', 'Sacred', 'Heart.', 'Immediatel
y', 'behind', 'the', 'basilica', 'is', 'the', 'Grotto,', 'a', 'Marian', 'plac
e', 'of', 'prayer', 'and', 'reflection.', 'It', 'is', 'a', 'replica', 'of',
'the', 'grotto', 'at', 'Lourdes,', 'France', 'where', 'the', 'Virgin', 'Mar
y', 'reputedly', 'appeared', 'to', 'Saint', 'Bernadette', 'Soubirous', 'in',
'1858.', 'At', 'the', 'end', 'of', 'the', 'main', 'drive', '(and', 'in', 'a',
'direct', 'line', 'that', 'connects', 'through', '3', 'statues', 'and', 'th
e', 'Gold', 'Dome),', 'is', 'a', 'simple,', 'modern', 'stone', 'statue', 'o
f', 'Mary.']
************************************************************
start_position :  90
end_position :  92
************************************************************
is_impossible :  False
```

Write Predicitons :

This funciton is used to write predictions to output file provided. It internally uses get_final_text.

In [0]:

```
def get_final_text(pred_text, orig_text, do_lower_case):
  """Project the tokenized prediction back to the original text."""

  def _strip_spaces(text):
    ns_chars = []
    ns_to_s_map = collections.OrderedDict()
    for (i, c) in enumerate(text):
      if c == " ":
        continue
      ns_to_s_map[len(ns_chars)] = i
      ns_chars.append(c)
    ns_text = "".join(ns_chars)
    return (ns_text, ns_to_s_map)

  # We first tokenize `orig_text`, strip whitespace from the result
  # and `pred_text`, and check if they are the same length. If they are
  # NOT the same length, the heuristic has failed. If they are the same
  # length, we assume the characters are one-to-one aligned.
  tokenizer = tokenization.BasicTokenizer(do_lower_case=do_lower_case)

  tok_text = " ".join(tokenizer.tokenize(orig_text))

  start_position = tok_text.find(pred_text)
  if start_position == -1:
    if True:
      tf.logging.info(
          "Unable to find text: '%s' in '%s'" % (pred_text, orig_text))
    return orig_text
  end_position = start_position + len(pred_text) - 1

  (orig_ns_text, orig_ns_to_s_map) = _strip_spaces(orig_text)
  (tok_ns_text, tok_ns_to_s_map) = _strip_spaces(tok_text)

  if len(orig_ns_text) != len(tok_ns_text):
    if True:
      tf.logging.info("Length not equal after stripping spaces: '%s' vs '%s'",
                      orig_ns_text, tok_ns_text)
    return orig_text

  # We then project the characters in `pred_text` back to `orig_text` using
  # the character-to-character alignment.
  tok_s_to_ns_map = {}
  for (i, tok_index) in six.iteritems(tok_ns_to_s_map):
    tok_s_to_ns_map[tok_index] = i

  orig_start_position = None
  if start_position in tok_s_to_ns_map:
    ns_start_position = tok_s_to_ns_map[start_position]
    if ns_start_position in orig_ns_to_s_map:
      orig_start_position = orig_ns_to_s_map[ns_start_position]

  if orig_start_position is None:
    if True:
      tf.logging.info("Couldn't map start position")
    return orig_text
```

```
orig_end_position = None
if end_position in tok_s_to_ns_map:
  ns_end_position = tok_s_to_ns_map[end_position]
  if ns_end_position in orig_ns_to_s_map:
    orig_end_position = orig_ns_to_s_map[ns_end_position]

if orig_end_position is None:
  if True:
    tf.logging.info("Couldn't map end position")
  return orig_text

output_text = orig_text[orig_start_position:(orig_end_position + 1)]
return output_text
```

In [0]:

```
def write_predictions(all_examples, all_features, all_results, n_best_size,
                      max_answer_length, do_lower_case, output_prediction_file
,
                      output_nbest_file, output_null_log_odds_file):
  """Write final predictions to the json file and log-odds of null if neede
d."""

  tf.logging.info("Writing predictions to: %s" % (output_prediction_file))
  tf.logging.info("Writing nbest to: %s" % (output_nbest_file))

  example_index_to_features = collections.defaultdict(list)
  for feature in all_features:
    example_index_to_features[feature.example_index].append(feature)

  unique_id_to_result = {}
  for result in all_results:
    unique_id_to_result[result.unique_id] = result

  _PrelimPrediction = collections.namedtuple(  # pylint: disable=invalid-name
      "PrelimPrediction",
      ["feature_index", "start_index", "end_index", "start_logit", "end_logit"
])

  all_predictions = collections.OrderedDict()
  all_nbest_json = collections.OrderedDict()
  scores_diff_json = collections.OrderedDict()

  for (example_index, example) in enumerate(all_examples):
    features = example_index_to_features[example_index]

    prelim_predictions = []
    # keep track of the minimum score of null start+end of position 0
    score_null = 1000000  # large and positive
    min_null_feature_index = 0  # the paragraph slice with min mull score
    null_start_logit = 0  # the start logit at the slice with min null score
    null_end_logit = 0  # the end logit at the slice with min null score
    for (feature_index, feature) in enumerate(features):
      result = unique_id_to_result[feature.unique_id]
      start_indexes = run_squad._get_best_indexes(result.start_logits, n_best_
size)
      end_indexes = run_squad._get_best_indexes(result.end_logits, n_best_size
)
      # if we could have irrelevant answers, get the min score of irrelevant
      if False:
        feature_null_score = result.start_logits[0] + result.end_logits[0]
        if feature_null_score < score_null:
          score_null = feature_null_score
          min_null_feature_index = feature_index
          null_start_logit = result.start_logits[0]
          null_end_logit = result.end_logits[0]
      for start_index in start_indexes:
        for end_index in end_indexes:
          # We could hypothetically create invalid predictions, e.g., predict
          # that the start of the span is in the question. We throw out all
          # invalid predictions.
          if start_index >= len(feature.tokens):
```

```
          continue
        if end_index >= len(feature.tokens):
          continue
        if start_index not in feature.token_to_orig_map:
          continue
        if end_index not in feature.token_to_orig_map:
          continue
        if not feature.token_is_max_context.get(start_index, False):
          continue
        if end_index < start_index:
          continue
        length = end_index - start_index + 1
        if length > max_answer_length:
          continue
        prelim_predictions.append(
            _PrelimPrediction(
                feature_index=feature_index,
                start_index=start_index,
                end_index=end_index,
                start_logit=result.start_logits[start_index],
                end_logit=result.end_logits[end_index]))

  if False:
    prelim_predictions.append(
        _PrelimPrediction(
            feature_index=min_null_feature_index,
            start_index=0,
            end_index=0,
            start_logit=null_start_logit,
            end_logit=null_end_logit))
  prelim_predictions = sorted(
      prelim_predictions,
      key=lambda x: (x.start_logit + x.end_logit),
      reverse=True)

  _NbestPrediction = collections.namedtuple(  # pylint: disable=invalid-name
      "NbestPrediction", ["text", "start_logit", "end_logit"])

  seen_predictions = {}
  nbest = []
  for pred in prelim_predictions:
    if len(nbest) >= n_best_size:
      break
    feature = features[pred.feature_index]
    if pred.start_index > 0:  # this is a non-null prediction
      tok_tokens = feature.tokens[pred.start_index:(pred.end_index + 1)]
      orig_doc_start = feature.token_to_orig_map[pred.start_index]
      orig_doc_end = feature.token_to_orig_map[pred.end_index]
      orig_tokens = example.doc_tokens[orig_doc_start:(orig_doc_end + 1)]
      tok_text = " ".join(tok_tokens)

      # De-tokenize WordPieces that have been split off.
      tok_text = tok_text.replace(" ##", "")
      tok_text = tok_text.replace("##", "")

      # Clean whitespace
      tok_text = tok_text.strip()
      tok_text = " ".join(tok_text.split())
      orig_text = " ".join(orig_tokens)

      final_text = get_final_text(tok_text, orig_text, do_lower_case)
      if final_text in seen_predictions:
        continue

      seen_predictions[final_text] = True
    else:
      final_text = ""
      seen_predictions[final_text] = True

    nbest.append(
        _NbestPrediction(
            text=final_text,
            start_logit=pred.start_logit,
            end_logit=pred.end_logit))

  # if we didn't inlude the empty option in the n-best, inlcude it
  if False:
    if "" not in seen_predictions:
      nbest.append(
          _NbestPrediction(
              text="", start_logit=null_start_logit,
              end_logit=null_end_logit))
  # In very rare edge cases we could have no valid predictions. So we
  # just create a nonce prediction in this case to avoid failure.
  if not nbest:
    nbest.append(
        _NbestPrediction(text="empty", start_logit=0.0, end_logit=0.0))

  assert len(nbest) >= 1

  total_scores = []
  best_non_null_entry = None
  for entry in nbest:
    total_scores.append(entry.start_logit + entry.end_logit)
    if not best_non_null_entry:
      if entry.text:
        best_non_null_entry = entry

  probs = run_squad._compute_softmax(total_scores)

  nbest_json = []
  for (i, entry) in enumerate(nbest):
    output = collections.OrderedDict()
    output["text"] = entry.text
    output["probability"] = probs[i]
    output["start_logit"] = entry.start_logit
    output["end_logit"] = entry.end_logit
    nbest_json.append(output)

  assert len(nbest_json) >= 1

  if not False:
    all_predictions[example.qas_id] = nbest_json[0]["text"]
  else:
    # predict "" iff the null score - the score of best non-null > threshold
    score_diff = score_null - best_non_null_entry.start_logit - (
        best_non_null_entry.end_logit)
    scores_diff_json[example.qas_id] = score_diff
    if score_diff > 0.0:
      all_predictions[example.qas_id] = ""
    else:
      all_predictions[example.qas_id] = best_non_null_entry.text

  all_nbest_json[example.qas_id] = nbest_json

with tf.gfile.GFile(output_prediction_file, "w") as writer:
  writer.write(json.dumps(all_predictions, indent=4) + "\n")

with tf.gfile.GFile(output_nbest_file, "w") as writer:
  writer.write(json.dumps(all_nbest_json, indent=4) + "\n")

if False:
  with tf.gfile.GFile(output_null_log_odds_file, "w") as writer:
    writer.write(json.dumps(scores_diff_json, indent=4) + "\n")
```

Model Parameters

In [0]:

```
# Model Hyper Parameters
TRAIN_BATCH_SIZE = 16
LEARNING_RATE = 3e-5
NUM_TRAIN_EPOCHS = 2.0
WARMUP_PROPORTION = 0.1
MAX_SEQ_LENGTH = 256
EVAL_BATCH_SIZE = 8

tpu_cluster_resolver = None
SAVE_CHECKPOINTS_STEPS = 1000
ITERATIONS_PER_LOOP = 1000
NUM_TPU_CORES = 8
```

Tokenizer:

Takes vocab file as input.

In [10]:

```
tokenizer = tokenization.FullTokenizer(vocab_file=VOCAB_FILE, do_lower_case=Tr
ue)
```

Tokenization examples

In [9]:

```
tokenizer.tokenize("This is a different and longer example to check the 'Token
ization'.")
```

Out[9]:

```
['this',
 'is',
 'a',
 'different',
 'and',
 'longer',
 'example',
 'to',
 'check',
 'the',
 "'",
 'token',
 '##ization',
 "'",
 '.']
```

In [0]:

```
tokenizer.tokenize("this is demo example for tokenizer")
```

Out[0]:

```
['this', 'is', 'demo', 'example', 'for', 'token', '##izer']
```

In [0]:

```
num_train_steps = int(
    len(train_examples) / TRAIN_BATCH_SIZE * NUM_TRAIN_EPOCHS)
num_warmup_steps = int(num_train_steps * WARMUP_PROPORTION)
```

In [0]:

```
# Setup TPU related config
tpu_cluster_resolver = tf.contrib.cluster_resolver.TPUClusterResolver(TPU_ADDR
ESS)
NUM_TPU_CORES = 8
```

model function builds model

In [0]:

```
model_fn = run_squad.model_fn_builder(
    bert_config=modeling.BertConfig.from_json_file(CONFIG_FILE),
    init_checkpoint=INIT_CHECKPOINT,
    learning_rate=LEARNING_RATE,
    num_train_steps=num_train_steps,
    num_warmup_steps=num_warmup_steps,
    use_tpu=True, #If False training will fall on CPU or GPU, depending on wha
t is available
    use_one_hot_embeddings=True)
```

In [0]:

```
run_config = tf.contrib.tpu.RunConfig(
    cluster=tpu_cluster_resolver,
    model_dir=OUTPUT_DIR,
    save_checkpoints_steps=SAVE_CHECKPOINTS_STEPS,
    tpu_config=tf.contrib.tpu.TPUConfig(
        iterations_per_loop=ITERATIONS_PER_LOOP,
        num_shards=NUM_TPU_CORES,
        per_host_input_for_training=tf.contrib.tpu.InputPipelineConfig.PER_HOS
T_V2))
```

In [0]:

```
estimator = tf.contrib.tpu.TPUEstimator(
    use_tpu=True, #If False training will fall on CPU or GPU, depending on wha
t is available
    model_fn=model_fn,
    config=run_config,
    train_batch_size=TRAIN_BATCH_SIZE,
    predict_batch_size=EVAL_BATCH_SIZE,
    eval_batch_size=EVAL_BATCH_SIZE)
```

```
W0827 06:18:43.056533 140491252946816 estimator.py:1984] Estimator's model_fn
(<function model_fn_builder.<locals>.model_fn at 0x7fc65e7fe950>) includes pa
rams argument, but params are not passed to Estimator.
```

In [0]:

```
print('Please wait...')
train_writer = run_squad.FeatureWriter(
        filename=os.path.join(OUTPUT_DIR, "train.tf_record"),
        is_training=True)


def append_feature(feature):
  train_features.append(feature)
  train_writer.process_feature(feature)
```

```
W0827 06:18:46.242335 140491252946816 deprecation_wrapper.py:119] From bert_r
epo/run_squad.py:1065: The name tf.python_io.TFRecordWriter is deprecated. Pl
ease use tf.io.TFRecordWriter instead.


Please wait...
```

convert_examples_to_features

This function converts train examples got from read_squad_examples to features that can be fed to BERT model. It returns all features in InputFeatures class object.

Input Feature class schema:

```
unique_id,<br>
example_index,<br>
doc_span_index,<br>
tokens,<br>
token_to_orig_map,<br>
token_is_max_context,<br>
input_ids,<br>
input_mask,<br>
segment_ids,<br>
start_position=None,<br>
end_position=None,<br>
is_impossible=None<br>
```

In [0]:

```
train_features = []

run_squad.convert_examples_to_features(train_examples, tokenizer, MAX_SEQ_LENG
TH, 128, 64, True, output_fn=append_feature)

train_writer.close()
```

```
W0827 06:18:47.903628 140491252946816 deprecation_wrapper.py:119] From bert_r
epo/run_squad.py:431: The name tf.logging.info is deprecated. Please use tf.c
ompat.v1.logging.info instead.
```

In [67]:

```
print("example to feature :")
print(type(train_features[0]))
print("-"*70)
print("unique_id : ",train_features[0].unique_id)
print("-"*70)
print("example_index :",train_features[0].example_index)
print("-"*70)
print("doc_span_index :",train_features[0].doc_span_index)
print("-"*70)
print("tokens :")
print(train_features[0].tokens)
print("-"*70)
print("token_to_orig_map :")
print(train_features[0].token_to_orig_map)
print("-"*70)
print("token_is_max_context :")
print(train_features[0].token_is_max_context)
print("-"*70)
print("input_ids :")
print(train_features[0].input_ids)
print("-"*70)
print("segment_ids :")
print(train_features[0].segment_ids)
print("-"*70)
print("start_position : ",train_features[0].start_position)
print("-"*70)
print("end_position :",train_features[0].end_position)
print("-"*70)
print("is_impossible :",train_features[0].is_impossible)
```

```
example to feature :
<class 'run_squad.InputFeatures'>
----------------------------------------------------------------------
unique_id :  1000000000
----------------------------------------------------------------------
example_index : 0
----------------------------------------------------------------------
doc_span_index : 0
----------------------------------------------------------------------
tokens :
['[CLS]', 'to', 'whom', 'did', 'the', 'virgin', 'mary', 'allegedly', 'appea
r', 'in', '1858', 'in', 'lou', '##rdes', 'france', '?', '[SEP]', 'architectur
al', '##ly', ',', 'the', 'school', 'has', 'a', 'catholic', 'character', '.',
'atop', 'the', 'main', 'building', "'", 's', 'gold', 'dome', 'is', 'a', 'gold
en', 'statue', 'of', 'the', 'virgin', 'mary', '.', 'immediately', 'in', 'fron
t', 'of', 'the', 'main', 'building', 'and', 'facing', 'it', ',', 'is', 'a',
'copper', 'statue', 'of', 'christ', 'with', 'arms', 'up', '##rai', '##sed',
'with', 'the', 'legend', '"', 've', '##ni', '##te', 'ad', 'me', 'om', '##ne
s', '"', '.', 'next', 'to', 'the', 'main', 'building', 'is', 'the', 'basilic
a', 'of', 'the', 'sacred', 'heart', '.', 'immediately', 'behind', 'the', 'bas
ilica', 'is', 'the', 'gr', '##otto', ',', 'a', 'marian', 'place', 'of', 'pray
er', 'and', 'reflection', '.', 'it', 'is', 'a', 'replica', 'of', 'the', 'gr',
'##otto', 'at', 'lou', '##rdes', ',', 'france', 'where', 'the', 'virgin', 'ma
ry', 'reputed', '##ly', 'appeared', 'to', 'saint', 'bern', '##ade', '##tte',
'so', '##ub', '##iro', '##us', 'in', '1858', '.', 'at', 'the', 'end', 'of',
'the', 'main', 'drive', '(', 'and', 'in', 'a', 'direct', 'line', 'that', 'con
nects', 'through', '3', 'statues', 'and', 'the', 'gold', 'dome', ')', ',', 'i
s', 'a', 'simple', ',', 'modern', 'stone', 'statue', 'of', 'mary', '.', '[SE
P]']
----------------------------------------------------------------------
token_to_orig_map :
{17: 0, 18: 0, 19: 0, 20: 1, 21: 2, 22: 3, 23: 4, 24: 5, 25: 6, 26: 6, 27: 7,
28: 8, 29: 9, 30: 10, 31: 10, 32: 10, 33: 11, 34: 12, 35: 13, 36: 14, 37: 15,
38: 16, 39: 17, 40: 18, 41: 19, 42: 20, 43: 20, 44: 21, 45: 22, 46: 23, 47: 2
4, 48: 25, 49: 26, 50: 27, 51: 28, 52: 29, 53: 30, 54: 30, 55: 31, 56: 32, 5
7: 33, 58: 34, 59: 35, 60: 36, 61: 37, 62: 38, 63: 39, 64: 39, 65: 39, 66: 4
0, 67: 41, 68: 42, 69: 43, 70: 43, 71: 43, 72: 43, 73: 44, 74: 45, 75: 46, 7
6: 46, 77: 46, 78: 46, 79: 47, 80: 48, 81: 49, 82: 50, 83: 51, 84: 52, 85: 5
3, 86: 54, 87: 55, 88: 56, 89: 57, 90: 58, 91: 58, 92: 59, 93: 60, 94: 61, 9
5: 62, 96: 63, 97: 64, 98: 65, 99: 65, 100: 65, 101: 66, 102: 67, 103: 68, 10
4: 69, 105: 70, 106: 71, 107: 72, 108: 72, 109: 73, 110: 74, 111: 75, 112: 7
6, 113: 77, 114: 78, 115: 79, 116: 79, 117: 80, 118: 81, 119: 81, 120: 81, 12
1: 82, 122: 83, 123: 84, 124: 85, 125: 86, 126: 87, 127: 87, 128: 88, 129: 8
9, 130: 90, 131: 91, 132: 91, 133: 91, 134: 92, 135: 92, 136: 92, 137: 92, 13
8: 93, 139: 94, 140: 94, 141: 95, 142: 96, 143: 97, 144: 98, 145: 99, 146: 10
0, 147: 101, 148: 102, 149: 102, 150: 103, 151: 104, 152: 105, 153: 106, 154:
107, 155: 108, 156: 109, 157: 110, 158: 111, 159: 112, 160: 113, 161: 114, 16
2: 115, 163: 115, 164: 115, 165: 116, 166: 117, 167: 118, 168: 118, 169: 119,
170: 120, 171: 121, 172: 122, 173: 123, 174: 123}
----------------------------------------------------------------------
token_is_max_context :
{17: True, 18: True, 19: True, 20: True, 21: True, 22: True, 23: True, 24: Tr
ue, 25: True, 26: True, 27: True, 28: True, 29: True, 30: True, 31: True, 32:
True, 33: True, 34: True, 35: True, 36: True, 37: True, 38: True, 39: True, 4
0: True, 41: True, 42: True, 43: True, 44: True, 45: True, 46: True, 47: Tru
e, 48: True, 49: True, 50: True, 51: True, 52: True, 53: True, 54: True, 55:
True, 56: True, 57: True, 58: True, 59: True, 60: True, 61: True, 62: True, 6
3: True, 64: True, 65: True, 66: True, 67: True, 68: True, 69: True, 70: Tru
e, 71: True, 72: True, 73: True, 74: True, 75: True, 76: True, 77: True, 78:
True, 79: True, 80: True, 81: True, 82: True, 83: True, 84: True, 85: True, 8
6: True, 87: True, 88: True, 89: True, 90: True, 91: True, 92: True, 93: Tru
e, 94: True, 95: True, 96: True, 97: True, 98: True, 99: True, 100: True, 10
1: True, 102: True, 103: True, 104: True, 105: True, 106: True, 107: True, 10
8: True, 109: True, 110: True, 111: True, 112: True, 113: True, 114: True, 11
5: True, 116: True, 117: True, 118: True, 119: True, 120: True, 121: True, 12
2: True, 123: True, 124: True, 125: True, 126: True, 127: True, 128: True, 12
9: True, 130: True, 131: True, 132: True, 133: True, 134: True, 135: True, 13
6: True, 137: True, 138: True, 139: True, 140: True, 141: True, 142: True, 14
3: True, 144: True, 145: True, 146: True, 147: True, 148: True, 149: True, 15
0: True, 151: True, 152: True, 153: True, 154: True, 155: True, 156: True, 15
7: True, 158: True, 159: True, 160: True, 161: True, 162: True, 163: True, 16
4: True, 165: True, 166: True, 167: True, 168: True, 169: True, 170: True, 17
1: True, 172: True, 173: True, 174: True}
----------------------------------------------------------------------
input_ids :
[101, 2000, 3183, 2106, 1996, 6261, 2984, 9382, 3711, 1999, 8517, 1999, 1022
3, 26371, 2605, 1029, 102, 6549, 2135, 1010, 1996, 2082, 2038, 1037, 3234, 28
39, 1012, 10234, 1996, 2364, 2311, 1005, 1055, 2751, 8514, 2003, 1037, 3585,
6231, 1997, 1996, 6261, 2984, 1012, 3202, 1999, 2392, 1997, 1996, 2364, 2311,
1998, 5307, 2009, 1010, 2003, 1037, 6967, 6231, 1997, 4828, 2007, 2608, 2039,
14995, 6924, 2007, 1996, 5722, 1000, 2310, 3490, 2618, 4748, 2033, 18168, 526
7, 1000, 1012, 2279, 2000, 1996, 2364, 2311, 2003, 1996, 13546, 1997, 1996, 6
730, 2540, 1012, 3202, 2369, 1996, 13546, 2003, 1996, 24665, 23052, 1010, 103
7, 14042, 2173, 1997, 7083, 1998, 9185, 1012, 2009, 2003, 1037, 15059, 1997,
1996, 24665, 23052, 2012, 10223, 26371, 1010, 2605, 2073, 1996, 6261, 2984, 2
2353, 2135, 2596, 2000, 3002, 16595, 9648, 4674, 2061, 12083, 9711, 2271, 199
9, 8517, 1012, 2012, 1996, 2203, 1997, 1996, 2364, 3298, 1006, 1998, 1999, 10
37, 3622, 2240, 2008, 8539, 2083, 1017, 11342, 1998, 1996, 2751, 8514, 1007,
1010, 2003, 1037, 3722, 1010, 2715, 2962, 6231, 1997, 2984, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
----------------------------------------------------------------------
segment_ids :
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
----------------------------------------------------------------------
start_position :  130
----------------------------------------------------------------------
end_position : 137
----------------------------------------------------------------------
is_impossible : False
```

In [0]:

```
print(len(train_examples))
print(len(train_features))
```

```
87599
97077
```

In [0]:

```
tf.logging.set_verbosity(tf.logging.INFO)
```

Loss Function :

- It uses **categotical crossentropy** loss function internally.
- Calcaulates loss seperately for start_token and end_token
- total loss is average of start and end loss

In [0]:

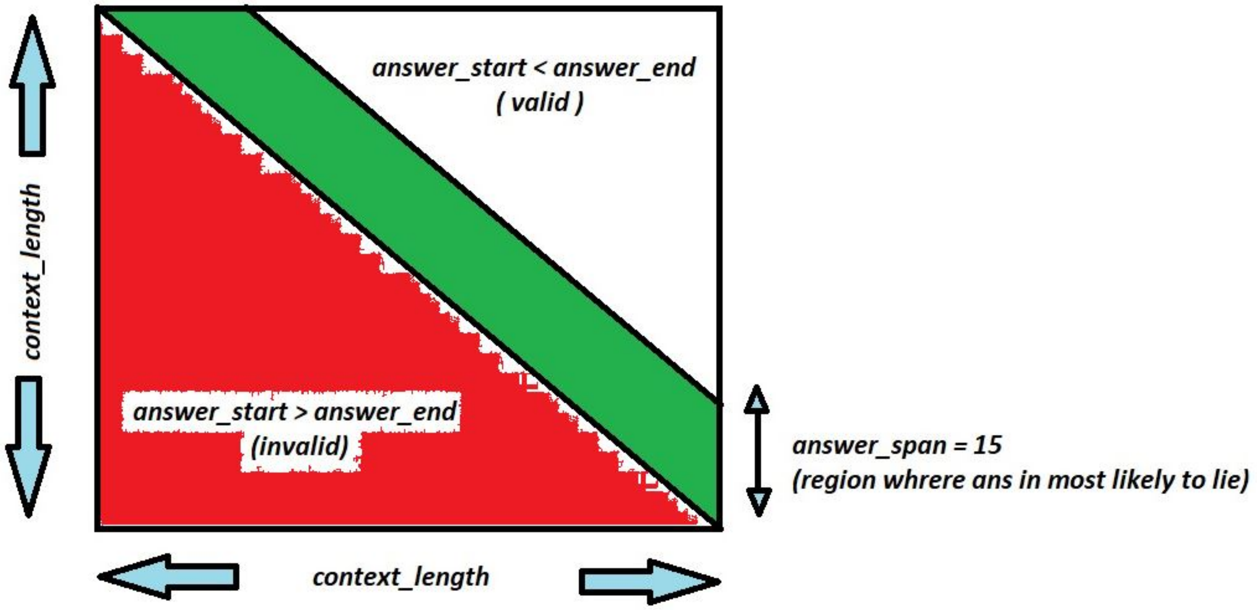
```
print('***** Started training at {} *****'.format(datetime.datetime.now()))
print('  Num examples = {}'.format(len(train_examples)))
print('  Batch size = {}'.format(TRAIN_BATCH_SIZE))


tf.logging.info("  Num steps = %d", num_train_steps)
train_input_fn = run_squad.input_fn_builder(
    input_file=train_writer.filename,
    seq_length=MAX_SEQ_LENGTH,
    is_training=True,
    drop_remainder=True)

estimator.train(input_fn=train_input_fn, max_steps=num_train_steps)
print('***** Finished training at {} *****'.format(datetime.datetime.now()))
```

```
I0827 06:24:27.257017 140491252946816 <ipython-input-41-9544e7b43547>:6]   Nu
m steps = 10949
W0827 06:24:27.258475 140491252946816 deprecation_wrapper.py:119] From bert_r
epo/run_squad.py:691: The name tf.FixedLenFeature is deprecated. Please use t
f.io.FixedLenFeature instead.


***** Started training at 2019-08-27 06:24:27.255979 *****
  Num examples = 87599
  Batch size = 16

I0827 06:24:28.046493 140491252946816 estimator.py:360] Skipping training sin
ce max_steps has already saved.
I0827 06:24:28.047625 140491252946816 error_handling.py:96] training_loop mar
ked as finished

***** Finished training at 2019-08-27 06:24:28.050430 *****
```

Prediction

The below steps are used while prediciton:

- A Dense layer followed by softmax layer is applied on the final output.
- The probability of word i being the start of the answer span is computed as a dot product between Ti and S followed by a softmax over all of the words in the paragraph.
- The end and start tokens are chosen which maximises probabilities.


answer_start < answer_end
(valid)
context_length
answer_start > answer_end
(invalid)
answer_span = 15
(region whrere ans in most likely to lie)
context_length

```
In [0]: eval_examples = read_squad_examples("/content/dev-v1.1.json",False)
```

In [57]:
```
print("type of eval ",type(eval_examples))
print("-"*60)
print(eval_examples[0].qas_id)
print("-"*60)
print(eval_examples[0].doc_tokens)
```

```
type of eval  <class 'list'>
------------------------------------------------------------
56be4db0acb8001400a502ec
------------------------------------------------------------
['Super', 'Bowl', '50', 'was', 'an', 'American', 'football', 'game', 'to', 'd
etermine', 'the', 'champion', 'of', 'the', 'National', 'Football', 'League',
'(NFL)', 'for', 'the', '2015', 'season.', 'The', 'American', 'Football', 'Con
ference', '(AFC)', 'champion', 'Denver', 'Broncos', 'defeated', 'the', 'Natio
nal', 'Football', 'Conference', '(NFC)', 'champion', 'Carolina', 'Panthers',
'24–10', 'to', 'earn', 'their', 'third', 'Super', 'Bowl', 'title.', 'The', 'g
ame', 'was', 'played', 'on', 'February', '7,', '2016,', 'at', "Levi's", 'Stad
ium', 'in', 'the', 'San', 'Francisco', 'Bay', 'Area', 'at', 'Santa', 'Clar
a,', 'California.', 'As', 'this', 'was', 'the', '50th', 'Super', 'Bowl,', 'th
e', 'league', 'emphasized', 'the', '"golden', 'anniversary"', 'with', 'variou
s', 'gold-themed', 'initiatives,', 'as', 'well', 'as', 'temporarily', 'suspen
ding', 'the', 'tradition', 'of', 'naming', 'each', 'Super', 'Bowl', 'game',
'with', 'Roman', 'numerals', '(under', 'which', 'the', 'game', 'would', 'hav
e', 'been', 'known', 'as', '"Super', 'Bowl', 'L"),', 'so', 'that', 'the', 'lo
go', 'could', 'prominently', 'feature', 'the', 'Arabic', 'numerals', '50.']
```

In [0]:
```
eval_writer = run_squad.FeatureWriter(
    filename=os.path.join(OUTPUT_DIR, "eval.tf_record"),
    is_training=False)
```

In [0]:
```
def append_feature(feature):
    eval_features.append(feature)
    eval_writer.process_feature(feature)
```

In [2]:
```
eval_features = []

run_squad.convert_examples_to_features(
        examples=eval_examples,
        tokenizer=tokenizer,
        max_seq_length=MAX_SEQ_LENGTH,
        doc_stride=128,
        max_query_length=64,
        is_training=False,
        output_fn=append_feature)

eval_writer.close()
```

In [0]:
```
print(len(eval_examples))
print(len(eval_features))
```

```
10570
12006
```

In [0]:

```
tf.logging.info("***** Running predictions *****")
tf.logging.info("  Num orig examples = %d", len(eval_examples))
tf.logging.info("  Num features = %d", len(eval_features))

predict_input_fn = run_squad.input_fn_builder(
    input_file=eval_writer.filename,
    seq_length=MAX_SEQ_LENGTH,
    is_training=False,
    drop_remainder=False)
```

```
I0827 06:25:10.172574 140491252946816 <ipython-input-47-84a684c96aaa>:1] ****
* Running predictions *****
I0827 06:25:10.174535 140491252946816 <ipython-input-47-84a684c96aaa>:2]   Nu
m orig examples = 10570
I0827 06:25:10.175938 140491252946816 <ipython-input-47-84a684c96aaa>:3]   Nu
m features = 12006
```

In [1]:

```
all_results = []
for result in estimator.predict(predict_input_fn, yield_single_examples=True):
    if len(all_results) % 1000 == 0:
        tf.logging.info("Processing example: %d" % (len(all_results)))
    unique_id = int(result["unique_ids"])
    start_logits = [float(x) for x in result["start_logits"].flat]
    end_logits = [float(x) for x in result["end_logits"].flat]
    all_results.append(
          run_squad.RawResult(
              unique_id=unique_id,
              start_logits=start_logits,
              end_logits=end_logits))
```

In [0]:

```
output_prediction_file = os.path.join(OUTPUT_DIR, "predictions.json")
output_nbest_file = os.path.join(OUTPUT_DIR, "nbest_predictions.json")
output_null_log_odds_file = os.path.join(OUTPUT_DIR, "null_odds.json")
```

In [0]:

```
import collections
import json

write_predictions(eval_examples, eval_features, all_results,
                      20, 30,
                      True, output_prediction_file,
                      output_nbest_file, output_null_log_odds_file)
```

F1 Score Calcualtion

In [0]:

```
# please refer https://github.com/allenai/bi-att-flow/blob/master/squad/evaluate-v1.1.py#L86
import re
from collections import Counter
def normalize_answer(s):
    """Lower text and remove punctuation, articles and extra whitespace."""
    def remove_articles(text):
        return re.sub(r'\b(a|an|the)\b', ' ', text)

    def white_space_fix(text):
        return ' '.join(text.split())

    def remove_punc(text):
        exclude = set(string.punctuation)
        return ''.join(ch for ch in text if ch not in exclude)

    def lower(text):
        return text.lower()

    return white_space_fix(remove_articles(remove_punc(lower(s))))

def f1_score(prediction, ground_truth):
    prediction_tokens = normalize_answer(prediction).split()
    ground_truth_tokens = normalize_answer(ground_truth).split()
    common = Counter(prediction_tokens) & Counter(ground_truth_tokens)
    num_same = sum(common.values())
    if num_same == 0:
        return 0
    precision = 1.0 * num_same / len(prediction_tokens)
    recall = 1.0 * num_same / len(ground_truth_tokens)
    f1 = (2 * precision * recall) / (precision + recall)
    return f1


def exact_match_score(prediction, ground_truth):
    return (normalize_answer(prediction) == normalize_answer(ground_truth))


def metric_max_over_ground_truths(metric_fn, prediction, ground_truths):
    scores_for_ground_truths = []
    for ground_truth in ground_truths:
        score = metric_fn(prediction, ground_truth)
        scores_for_ground_truths.append(score)
    return max(scores_for_ground_truths)


def evaluate(dataset, predictions):
    f1 = exact_match = total = 0
    for article in dataset:
        for paragraph in article['paragraphs']:
            for qa in paragraph['qas']:
                total += 1
                if qa['id'] not in predictions:
#                     message = 'Unanswered question ' + qa['id'] + \
#                               ' will receive score 0.'
#                     print(message, file=sys.stderr)
```

```
                    continue
                ground_truths = list(map(lambda x: x['text'], qa['answers']))
                prediction = predictions[qa['id']]
                exact_match += metric_max_over_ground_truths(
                    exact_match_score, prediction, ground_truths)
                f1 += metric_max_over_ground_truths(
                    f1_score, prediction, ground_truths)

    exact_match = 100.0 * exact_match / total
    f1 = 100.0 * f1 / total

    return {'exact_match': exact_match, 'f1': f1}


def evaluate_squad(data_file,pred_file):

  with open(data_file) as dataset_file:
    dataset_json = json.load(dataset_file)
    dataset = dataset_json['data']
  with open(pred_file) as prediction_file:
    predictions = json.load(prediction_file)
  print(evaluate(dataset, predictions))
```

In [0]:

```
evaluate_squad("/content/dev-v1.1.json","/content/bert-checkpoints_models_SQUAD_predictions.json")
```

```
{'exact_match': 80.85146641438033, 'f1': 88.0228956599229}
```

In [0]:

```
evaluate_squad("/content/train-v1.1.json","/content/bert-checkpoints_models_SQUAD_train_predictions.json")
```

```
{'exact_match': 84.38978240302744,'f1': 90.87081895814865}
```