EDA Assignment

Group A

2/19/2025

Christian Reza

Goal:

- General schema description about an existing DB schema dataset (table/column count, relationships, etc.)
- Created & populated DB schema
- Learning points taken as we begin applying & fine-tuning T5 model to query DB schema (we do not expect to finish this, rather do technical exploration of a T5 model and how it can be fine-tuned)

Using spider data from Kaggle, specifically the soccer_1 data: https://www.kaggle.com/datasets/jeromeblanchet/yale-universitys-spider-10-nlp-dataset/data

- I will be focusing on the `Player` table and create the training data off the following rows:
  - "player_name"
  - "birthday"
  - "height"
  - "weight"

Install packages and pull TAPAS QA model to fine-tune

In [1]:

```
!pip install tf-keras
!pip install -q transformers
!pip install -q setuptools
!pip install -q tensorflow
!pip install -q pandas
import pandas as pd
from transformers import TapasConfig, TapasTokenizer, TFTapasForQuestionAnswering
import tensorflow as tf
```

```
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\tqdm\auto.py:21: TqdmWarning:
IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.re
adthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
WARNING:tensorflow:From C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundati
on.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\tf_ke
ras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecate
d. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [2]:

```python
# The base sized model with WTQ configuration
model_name = "google/tapas-base-finetuned-wtq"
config = TapasConfig.from_pretrained(model_name)
model = TFTapasForQuestionAnswering.from_pretrained("google/tapas-base", config=con
```

```
WARNING:tensorflow:From C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundati
on.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\tf_ke
ras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.c
ompat.v1.get_default_graph instead.
```

```
All model checkpoint layers were used when initializing TFTapasForQuestionAnswering.

Some layers of TFTapasForQuestionAnswering were not initialized from the model check
point at google/tapas-base and are newly initialized: ['dropout_37', 'compute_column
_logits', 'compute_token_logits', 'aggregation_classifier']
You should probably TRAIN this model on a down-stream task to be able to use it for
predictions and inference.
Some layers of TFTapasForQuestionAnswering were not initialized from the model check
point at google/tapas-base and are newly initialized: ['dropout_37', 'compute_column
_logits', 'compute_token_logits', 'aggregation_classifier']
You should probably TRAIN this model on a down-stream task to be able to use it for
predictions and inference.
```

Create training data for TAPAS from Spider football data

Util script used supplied from tapas_utils:

https://github.com/NielsRogge/tapas_utils/blob/master/parse_answer_texts.py

In [3]:

```
!pip install -q frozendict
!pip install -q scipy
!pip install -q numpy
```

In [4]:

```python
# coding=utf-8
# Copyright 2019 The Google AI Language Team Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# Lint as: python3
"""This module implements a simple parser that can be used for TAPAS.
```

```
Given a table, a question and one or more answer_texts, it will parse the texts
to populate other fields (e.g. answer_coordinates, float_value) that are required
by TAPAS.

Please note that exceptions in this module are concise and not parameterized,
since they are used as counter names in a BEAM pipeline.
"""

import enum
from typing import Callable, List, Text, Optional

import six
import struct
import unicodedata
import re

import frozendict
import numpy as np
import scipy.optimize


class SupervisionMode(enum.Enum):
  # Don't filter out any supervised information.
  NONE = 0
  # Remove all the supervised signals and recompute them by parsing answer
  # texts.
  REMOVE_ALL = 2
  # Same as above but discard ambiguous examples
  # (where an answer matches multiple cells).
  REMOVE_ALL_STRICT = 3


def _find_matching_coordinates(table, answer_text,
                               normalize):
  normalized_text = normalize(answer_text)
  for row_index, row in table.iterrows():
    for column_index, cell in enumerate(row):
      if normalized_text == normalize(str(cell)):
        yield (row_index, column_index)


def _compute_cost_matrix_inner(
    table,
    answer_texts,
    normalize,
    discard_ambiguous_examples,
):
  """Returns a cost matrix M where the value M[i,j] contains a matching cost from a

  The matrix is a binary matrix and -1 is used to indicate a possible match from
  a given answer_texts to a specific cell table. The cost matrix can then be
  usedto compute the optimal assignments that minimizes the cost using the
  hungarian algorithm (see scipy.optimize.linear_sum_assignment).

  Args:
    table: a Pandas dataframe.
    answer_texts: a list of strings.
    normalize: a function that normalizes a string.
    discard_ambiguous_examples: If true discard if answer has multiple matches.

  Raises:
    ValueError if:
      - we cannot correctly construct the cost matrix or the text-cell
      assignment is ambiguous.
      - we cannot find a matching cell for a given answer_text.

  Returns:
    A numpy matrix with shape (num_answer_texts, num_rows * num_columns).
  """
  max_candidates = 0
  n_rows, n_columns = table.shape[0], table.shape[1]
  num_cells = n_rows * n_columns
  num_candidates = np.zeros((n_rows, n_columns))
  cost_matrix = np.zeros((len(answer_texts), num_cells))

  for index, answer_text in enumerate(answer_texts):
    found = 0
    for row, column in _find_matching_coordinates(table, answer_text,
                                                  normalize):
      found += 1
      cost_matrix[index, (row * len(table.columns)) + column] = -1
      num_candidates[row, column] += 1
      max_candidates = max(max_candidates, num_candidates[row, column])
    if found == 0:
      return None
    if discard_ambiguous_examples and found > 1:
      raise ValueError("Found multiple cells for answers")

  # TODO(piccinno): Shall we allow ambiguous assignments?
  if max_candidates > 1:
    raise ValueError("Assignment is ambiguous")

  return cost_matrix


def _compute_cost_matrix(
    table,
    answer_texts,
    discard_ambiguous_examples,
):
  """Computes cost matrix."""
  for index, normalize_fn in enumerate(STRING_NORMALIZATIONS):
    try:
      result = _compute_cost_matrix_inner(
          table,
          answer_texts,
          normalize_fn,
          discard_ambiguous_examples,
      )
      if result is None:
        continue
      return result
    except ValueError:
      if index == len(STRING_NORMALIZATIONS) - 1:
        raise
  return None


def _parse_answer_coordinates(table,
                              answer_texts,
                              discard_ambiguous_examples):
  """Populates answer_coordinates using answer_texts.

  Args:
    table: a Table message, needed to compute the answer coordinates.
    answer_texts: a list of strings
    discard_ambiguous_examples: If true discard if answer has multiple matches.

  Raises:
    ValueError if the conversion fails.
  """

  cost_matrix = _compute_cost_matrix(
      table,
      answer_texts,
      discard_ambiguous_examples,
  )
  if cost_matrix is None:
    return
  row_indices, column_indices = scipy.optimize.linear_sum_assignment(
      cost_matrix)

  # create answer coordinates as list of tuples
  answer_coordinates = []
  for row_index in row_indices:
    flatten_position = column_indices[row_index]
    row_coordinate = flatten_position // len(table.columns)
    column_coordinate = flatten_position % len(table.columns)
    answer_coordinates.append((row_coordinate, column_coordinate))

  return answer_coordinates


### START OF UTILITIES FROM TEXT_UTILS.PY ###

def wtq_normalize(x):
  """Returns the normalized version of x.
  This normalization function is taken from WikiTableQuestions github, hence the
  wtq prefix. For more information, see
  https://github.com/ppasupat/WikiTableQuestions/blob/master/evaluator.py
  Args:
    x: the object (integer type or string) to normalize.
  Returns:
    A normalized string.
  """
  x = x if isinstance(x, six.text_type) else six.text_type(x)
  # Remove diacritics.
  x = "".join(
      c for c in unicodedata.normalize("NFKD", x)
      if unicodedata.category(c) != "Mn")
  # Normalize quotes and dashes.
  x = re.sub(u"[‘’´`]", "'", x)
  x = re.sub(u"[“”]", '"', x)
  x = re.sub(u"[‐‑‒–—−]", "-", x)
  x = re.sub(u"[-]", "", x)
  while True:
    old_x = x
    # Remove citations.
    x = re.sub(u"((?", "", x)
  x = x.replace("\n", " ")
  return x


_TOKENIZER = re.compile(r"\w+|[^\w\s]+", re.UNICODE)


def tokenize_string(x):
  return list(_TOKENIZER.findall(x.lower()))


# List of string normalization functions to be applied in order. We go from
# simplest to more complex normalization procedures.
STRING_NORMALIZATIONS = (
    lambda x: x,
    lambda x: x.lower(),
    tokenize_string,
    wtq_normalize,
)


def to_float32(v):
  """If v is a float reduce precision to that of a 32 bit float."""
  if not isinstance(v, float):
    return v
  return struct.unpack("!f", struct.pack("!f", v))[0]


def convert_to_float(value):
  """Converts value to a float using a series of increasingly complex heuristics.
  Args:
    value: object that needs to be converted. Allowed types include
      float/int/strings.
  Returns:
    A float interpretation of value.
  Raises:
    ValueError if the float conversion of value fails.
  """
  if isinstance(value, float):
    return value
  if isinstance(value, int):
    return float(value)
  if not isinstance(value, six.string_types):
    raise ValueError("Argument value is not a string. Can't parse it as float")
  sanitized = value

  try:
    # Example: 1,000.7
    if "." in sanitized and "," in sanitized:
      return float(sanitized.replace(",", ""))
    # 1,000
    if "," in sanitized and _split_thousands(",", sanitized):
      return float(sanitized.replace(",", ""))
    # 5,5556
    if "," in sanitized and sanitized.count(",") == 1 and not _split_thousands(
        ",", sanitized):
      return float(sanitized.replace(",", "."))
    # 0.0.0.1
    if sanitized.count(".") > 1:
      return float(sanitized.replace(".", ""))
    # 0,0,0,1
    if sanitized.count(",") > 1:
      return float(sanitized.replace(",", ""))
    return float(sanitized)
  except ValueError:
    # Avoid adding the sanitized value in the error message.
    raise ValueError("Unable to convert value to float")

### END OF UTILITIES FROM TEXT_UTILS.PY ###

def _parse_answer_float(answer_texts, float_value):
  if len(answer_texts) > 1:
    raise ValueError("Cannot convert to multiple answers to single float")
  float_value = convert_to_float(answer_texts[0])
  float_value = float_value

  return answer_texts, float_value


def _has_single_float_answer_equal_to(question, answer_texts, target):
  """Returns true if the question has a single answer whose value equals to target.
  if len(answer_texts) != 1:
    return False
  try:
    float_value = convert_to_float(answer_texts[0])
    # In general answer_float is derived by applying the same conver_to_float
    # function at interaction creation time, hence here we use exact match to
    # avoid any false positive.
    return to_float32(float_value) == to_float32(target)
  except ValueError:
    return False


def _parse_question(
    table,
    original_question,
    answer_texts,
    answer_coordinates,
    float_value,
    aggregation_function,
    clear_fields,
    discard_ambiguous_examples,
):
  """Parses question's answer_texts fields to possibly populate additional fields.

  Args:
    table: a Pandas dataframe, needed to compute the answer coordinates.
    original_question: a string.
    answer_texts: a list of strings, serving as the answer to the question.
    anser_coordinates:
    float_value: a float, serves as float value signal.
    aggregation_function:
    clear_fields: A list of strings indicating which fields need to be cleared
      and possibly repopulated.
    discard_ambiguous_examples: If true, discard ambiguous examples.

  Returns:
    A Question message with answer_coordinates or float_value field populated.

  Raises:
    ValueError if we cannot parse correctly the question message.
  """
  question = original_question

  # If we have a float value signal we just copy its string representation to
  # the answer text (if multiple answers texts are present OR the answer text
  # cannot be parsed to float OR the float value is different), after clearing
  # this field.
  if "float_value" in clear_fields and float_value is not None:
    if not _has_single_float_answer_equal_to(question, answer_texts, float_value):
      del answer_texts[:]
      float_value = float(float_value)
      if float_value.is_integer():
        number_str = str(int(float_value))
      else:
        number_str = str(float_value)
      answer_texts = []
      answer_texts.append(number_str)

  if not answer_texts:
    raise ValueError("No answer_texts provided")

  for field_name in clear_fields:
    if field_name == "answer_coordinates":
        answer_coordinates = None
    if field_name == "float_value":
        float_value = None
    if field_name == "aggregation_function":
        aggregation_function = None

  error_message = ""
  if not answer_coordinates:
    try:
      answer_coordinates = _parse_answer_coordinates(
          table,
          answer_texts,
          discard_ambiguous_examples,
      )
    except ValueError as exc:
      error_message += "[answer_coordinates: {}]".format(str(exc))
      if discard_ambiguous_examples:
        raise ValueError(f"Cannot parse answer: {error_message}")

  if not float_value:
    try:
      answer_texts, float_value = _parse_answer_float(answer_texts, float_value)
    except ValueError as exc:
      error_message += "[float_value: {}]".format(str(exc))

  # Raises an exception if we cannot set any of the two fields.
  if not answer_coordinates and not float_value:
    raise ValueError("Cannot parse answer: {}".format(error_message))

  return question, answer_texts, answer_coordinates, float_value, aggregation_funct


# TODO(piccinno): Use some sort of introspection here to get the field names of
# the proto.
_CLEAR_FIELDS = frozendict.frozendict({
    SupervisionMode.REMOVE_ALL: [
        "answer_coordinates", "float_value", "aggregation_function"
    ],
    SupervisionMode.REMOVE_ALL_STRICT: [
        "answer_coordinates", "float_value", "aggregation_function"
    ]
})


def parse_question(table, question, answer_texts, answer_coordinates=None, float_va
                    mode=SupervisionMode.REMOVE_ALL):
    """Parses answer_text field of a question to populate additional fields require

    Args:
        table: a Pandas dataframe, needed to compute the answer coordinates. Note t
        before supplying the table to this function.
        question: a string.
        answer_texts: a list of strings, containing one or more answer texts that s
        answer_coordinates: optional answer coordinates supervision signal, if you
        float_value: optional float supervision signal, if you already have this.
        aggregation_function: optional aggregation function supervised signal, if y
        mode: see SupervisionMode enum for more information.

    Returns:
        A list with the question, populated answer_coordinates or float_value.

    Raises:
        ValueError if we cannot parse correctly the question string.
    """
    if mode == SupervisionMode.NONE:
        return question, answer_texts

    clear_fields = _CLEAR_FIELDS.get(mode, None)
    if clear_fields is None:
        raise ValueError(f"Mode {mode.name} is not supported")

    return _parse_question(
        table,
        question,
        answer_texts,
        answer_coordinates,
        float_value,
        aggregation_function,
        clear_fields,
        discard_ambiguous_examples=mode == SupervisionMode.REMOVE_ALL_STRICT,
    )
```

In [70]:

```
data = {
    "Player Name": [
        "Aaron Appindangoye",
        "Aaron Cresswell",
        "Aaron Doran",
        "Aaron Galindo",
        "Aaron Hughes",
        "Aaron Hunt",
        "Aaron Kuhl",
    ],
    "Birthday": [
        "1992-02-29 00:00:00",
        "1989-12-15 00:00:00",
        "1991-05-13 00:00:00",
        "1982-05-08 00:00:00",
        "1979-11-08 00:00:00",
        "1986-09-04 00:00:00",
        "1996-01-30 00:00:00",
    ],
    "Weight": [
        "187",
        "146",
        "163",
        "198",
        "154",
        "161",
        "146",
    ],
    "Height": [
        "182.88",
        "170.18",
        "170.18",
        "182.88",
        "182.88",
        "182.88",
        "172.72",
    ],
}


table = pd.DataFrame.from_dict(data)
table = table.astype(str)
```

```
table.head(8)
```

Out[70]:

	Player Name	Birthday	Weight	Height
0	Aaron Appindangoye	1992-02-29 00:00:00	187	182.88
1	Aaron Cresswell	1989-12-15 00:00:00	146	170.18
2	Aaron Doran	1991-05-13 00:00:00	163	170.18
3	Aaron Galindo	1982-05-08 00:00:00	198	182.88
4	Aaron Hughes	1979-11-08 00:00:00	154	182.88
5	Aaron Hunt	1986-09-04 00:00:00	161	182.88
6	Aaron Kuhl	1996-01-30 00:00:00	146	172.72

In [71]:

```
question = "How much does Aaron Hughes weigh?"

answer_texts = ["154"]

question, answer_texts, answer_coordinates, float_value, aggregation_function = par
print(question)
print(answer_texts)
print("Found coordinates:", answer_coordinates)
print("Found float value:", float_value)
```

```
How much does Aaron Hughes weigh?
['154']
Found coordinates: [(np.int64(4), np.int64(2))]
Found float value: 154.0
```

In [72]:

```
question = "What is Aaron Appindangoye's birthday?"

answer_texts = ["1992-02-29 00:00:00"]

question, answer_texts, answer_coordinates, float_value, aggregation_function = par
print(question)
print(answer_texts)
print("Found coordinates:", answer_coordinates)
print("Found float value:", float_value)
```

```
What is Aaron Appindangoye's birthday?
['1992-02-29 00:00:00']
Found coordinates: [(np.int64(0), np.int64(1))]
Found float value: None
```

In [73]:

```
question = "How many players weigh more than 150?"

answer_texts = ["187", "163", "198", "154", "161"]
float_value = 4

question, answer_texts, answer_coordinates, float_value, aggregation_function = par
print(question)
print(answer_texts)
print("Found coordinates:", answer_coordinates)
print("Found float value:", float_value)
```

```
How many players weigh more than 150?
['187', '163', '198', '154', '161']
Found coordinates: [(np.int64(0), np.int64(2)), (np.int64(2), np.int64(2)), (np.int6
4(3), np.int64(2)), (np.int64(4), np.int64(2)), (np.int64(5), np.int64(2))]
Found float value: None
```

Take output from SQA Util script to create tensor data to fine-tune model

In [9]:

```
tokenizer = TapasTokenizer.from_pretrained(model_name)

data = {
    "Player Name": [
        "Aaron Appindangoye",
        "Aaron Cresswell",
        "Aaron Doran",
        "Aaron Galindo",
        "Aaron Hughes",
        "Aaron Hunt",
        "Aaron Kuhl",
    ],
    "Birthday": [
        "1992-02-29 00:00:00",
        "1989-12-15 00:00:00",
        "1991-05-13 00:00:00",
        "1982-05-08 00:00:00",
        "1979-11-08 00:00:00",
        "1986-09-04 00:00:00",
        "1996-01-30 00:00:00",
    ],
    "Weight": [
        "187",
        "146",
        "163",
        "198",
        "154",
        "161",
        "146",
    ],
    "Height": [
        "182.88",
        "170.18",
        "170.18",
        "182.88",
        "182.88",
        "182.88",
        "172.72",
    ],
}
queries = [
    "How much does Aaron Hughes weigh?",
    "What is Aaron Appindangoye's birthday?",
    "How many players weigh more than 150?",
]
answer_coordinates = [[(4, 2)], [(0, 1)], [(0, 2), (2, 2), (3, 2), (4, 2), (5, 2)]]
answer_text = [["154"], ["1992-02-29 00:00:00"], ["4"]]
table = pd.DataFrame.from_dict(data)
inputs = tokenizer(
    table=table,
    queries=queries,
    answer_coordinates=answer_coordinates,
    answer_text=answer_text,
    padding="max_length",
    return_tensors="tf",
)
inputs
```

```
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:2699: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  text = normalize_for_match(row[col_index].text)
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:1493: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  cell = row[col_index]
```

Out[9]:

```
{'input_ids': <tf.Tensor: shape=(3, 512), dtype=int32, numpy=
array([[ 101, 2129, 2172, ...,    0,    0,    0],
       [ 101, 2054, 2003, ...,    0,    0,    0],
       [ 101, 2129, 2116, ...,    0,    0,    0]], dtype=int32)>, 'labels': <tf.Te
nsor: shape=(3, 512), dtype=int32, numpy=
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int32)>, 'numeric_values': <tf.Tensor: shap
e=(3, 512), dtype=float32, numpy=
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]], dtype=float32)>, 'numeric_values_scal
e': <tf.Tensor: shape=(3, 512), dtype=float32, numpy=
array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.]], dtype=float32)>, 'token_type_ids': <tf.Tens
or: shape=(3, 512, 7), dtype=int32, numpy=
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]]], dtype=int32)>, 'attention_mask': <tf.Tensor: sh
ape=(3, 512), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0]], dtype=int32)>}
```

In [67]:

```python
import os
from pathlib import Path

sqa_path = Path("C:/School/Spring 2025/DSCI-2025-TEAM-A/reza/data/sqa_train_set.csv
table_csv_path = Path("C:/School/Spring 2025/DSCI-2025-TEAM-A/reza/data/")



class TableDataset:
    """
    A dataset class for handling table data and tokenizing it for model input.
    Attributes:
        data (pd.DataFrame): The input data containing table file paths, questions,
        tokenizer (Tokenizer): The tokenizer used to process the table data and que
    Methods:
        __iter__():
            Iterates over the dataset, tokenizes the table data and questions, and
        __len__():
            Returns the length of the dataset.
    """

    def __init__(self, data, tokenizer):
        self.data = data
        self.tokenizer = tokenizer

    def __iter__(self):
        for idx in range(self.__len__()):
            item = self.data.iloc[idx]
            table = pd.read_csv(str(table_csv_path) + os.sep + item.table_file).ast
                str
            )  # be sure to make your table data text only
            encoding = self.tokenizer(
                table=table,
                queries=item.question,
                answer_coordinates=eval(item.answer_coordinates),
                # answer_coordinates=item.answer_coordinates,
                answer_text=item.answer_text,
                truncation=True,
                padding="max_length",
                return_tensors="tf",
            )
            # remove the batch dimension which the tokenizer adds by default
            encoding = {key: tf.squeeze(val, 0) for key, val in encoding.items()}
            # add the float_answer which is also required (weak supervision for agg
            float_answer = item.float_answer if not pd.isna(item.float_answer) else
            encoding["float_answer"] = tf.convert_to_tensor(
                item.float_answer, dtype=tf.float32
            )
            yield encoding["input_ids"], encoding["attention_mask"], encoding[
                "numeric_values"
            ], encoding["numeric_values_scale"], encoding["token_type_ids"], encodi
                "labels"
            ], encoding[
                "float_answer"
            ]

    def __len__(self):
        return len(self.data)


data = pd.read_csv(sqa_path, sep="\t")
train_dataset = TableDataset(data, tokenizer)
output_signature = (
    tf.TensorSpec(shape=(512,), dtype=tf.int32),
    tf.TensorSpec(shape=(512,), dtype=tf.int32),
    tf.TensorSpec(shape=(512,), dtype=tf.float32),
    tf.TensorSpec(shape=(512,), dtype=tf.float32),
    tf.TensorSpec(shape=(512, 7), dtype=tf.int32),
    tf.TensorSpec(shape=(512,), dtype=tf.int32),
    tf.TensorSpec(shape=(), dtype=tf.float32),
)
train_dataloader = tf.data.Dataset.from_generator(
    lambda: iter(train_dataset), output_signature=output_signature
).batch(32)
```

Fine-tuning the TAPAS model with new data!

- This I am confused on, what was the purpose of creating the `inputs` from the dataframe, do we still need the tsv to train the data?
- What is wrong with my `train_dataloader` that is causing an error in the batches - is there something I missed in the data loader, do I need more training data?

In [68]:

```
# this is the default WTQ configuration
config = TapasConfig(
    num_aggregation_labels=4,
    use_answer_as_supervision=True,
    answer_loss_cutoff=0.664694,
    cell_selection_preference=0.207951,
    huber_loss_delta=0.121194,
    init_cell_selection_weights_to_zero=True,
    select_one_column=True,
    allow_empty_column_selection=False,
    temperature=0.0352513,
)
model = TFTapasForQuestionAnswering.from_pretrained(model_name, config=config)

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)

for epoch in range(2):  # loop over the dataset multiple times
    for batch in train_dataloader:
        # get the inputs;
        input_ids = batch[0]
        attention_mask = batch[1]
        token_type_ids = batch[4]
        labels = batch[-1]
        numeric_values = batch[2]
        numeric_values_scale = batch[3]
        float_answer = batch[6]

        # forward + backward + optimize
        with tf.GradientTape() as tape:
            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask,
                token_type_ids=token_type_ids,
                labels=labels,
                numeric_values=numeric_values,
                numeric_values_scale=numeric_values_scale,
                float_answer=float_answer,
            )
        grads = tape.gradient(outputs.loss, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
```

```
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:2699: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  text = normalize_for_match(row[col_index].text)
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:1493: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  cell = row[col_index]
Some layers from the model checkpoint at google/tapas-base-finetuned-wtq were not us
ed when initializing TFTapasForQuestionAnswering: ['dropout_570']
- This IS expected if you are initializing TFTapasForQuestionAnswering from the chec
kpoint of a model trained on another task or with another architecture (e.g. initial
izing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFTapasForQuestionAnswering from the
checkpoint of a model that you expect to be exactly identical (initializing a BertFo
rSequenceClassification model from a BertForSequenceClassification model).
Some layers of TFTapasForQuestionAnswering were not initialized from the model check
point at google/tapas-base-finetuned-wtq and are newly initialized: ['dropout_417']
You should probably TRAIN this model on a down-stream task to be able to use it for
predictions and inference.
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:2699: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  text = normalize_for_match(row[col_index].text)
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:1493: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  cell = row[col_index]
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:2699: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  text = normalize_for_match(row[col_index].text)
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:1493: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  cell = row[col_index]
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
```

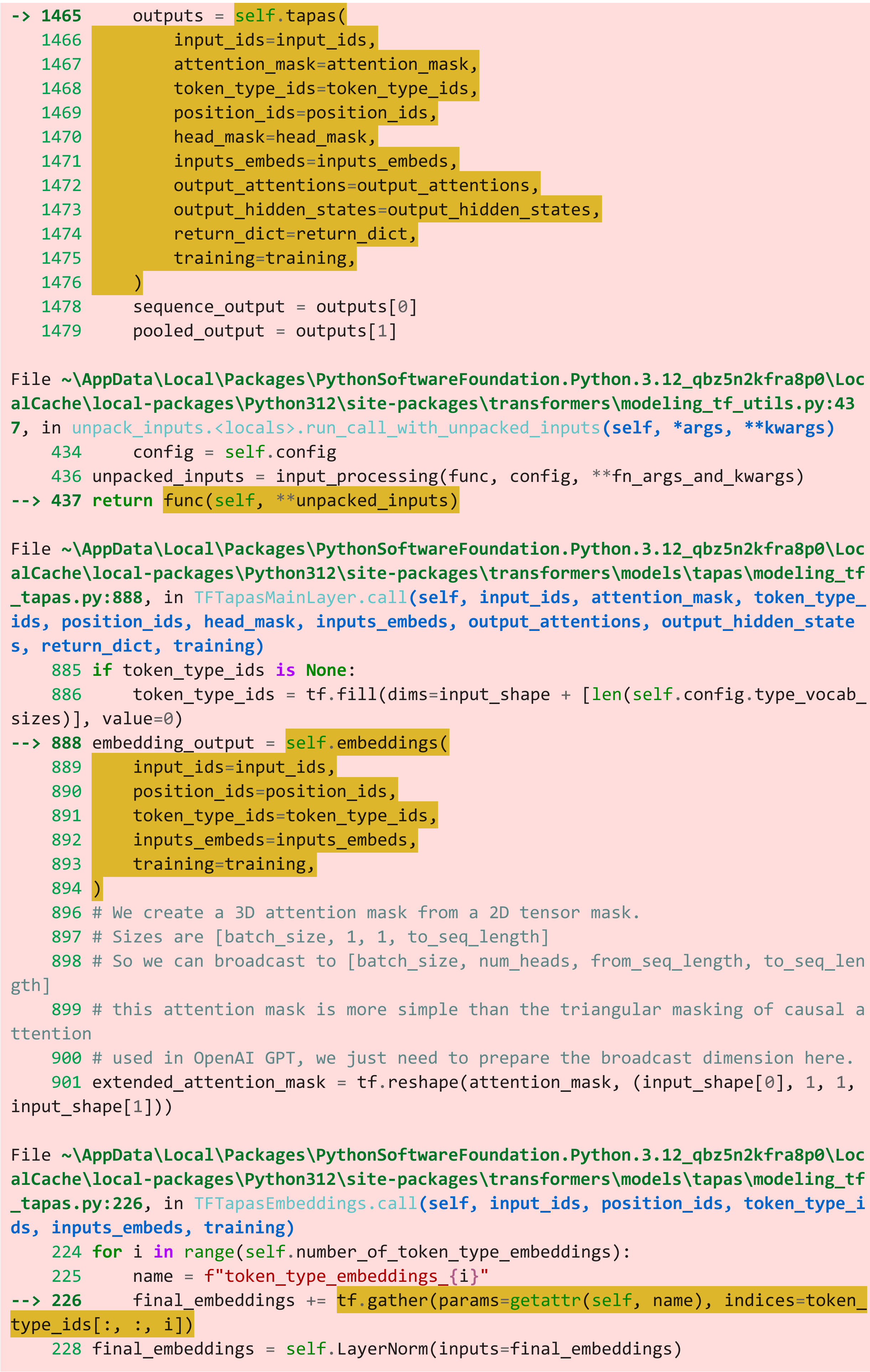
```
enization_tapas.py:2699: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  text = normalize_for_match(row[col_index].text)
C:\Users\rza_t\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfr
a8p0\LocalCache\local-packages\Python312\site-packages\transformers\models\tapas\tok
enization_tapas.py:1493: FutureWarning: Series.__getitem__ treating keys as position
s is deprecated. In a future version, integer keys will always be treated as labels
(consistent with DataFrame behavior). To access a value by position, use `ser.iloc[p
os]`
  cell = row[col_index]
```

```
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
Cell In[68], line 30
     28 # forward + backward + optimize
     29 with tf.GradientTape() as tape:
---> 30     outputs = model(
     31         input_ids=input_ids,
     32         attention_mask=attention_mask,
     33         token_type_ids=token_type_ids,
     34         labels=labels,
     35         numeric_values=numeric_values,
     36         numeric_values_scale=numeric_values_scale,
     37         float_answer=float_answer,
     38     )
     39 grads = tape.gradient(outputs.loss, model.trainable_weights)
     40 optimizer.apply_gradients(zip(grads, model.trainable_weights))

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\Loc
alCache\local-packages\Python312\site-packages\tf_keras\src\utils\traceback_utils.p
y:70, in filter_traceback.<locals>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\Loc
alCache\local-packages\Python312\site-packages\transformers\modeling_tf_utils.py:43
7, in unpack_inputs.<locals>.run_call_with_unpacked_inputs(self, *args, **kwargs)
    434     config = self.config
    436 unpacked_inputs = input_processing(func, config, **fn_args_and_kwargs)
--> 437 return func(self, **unpacked_inputs)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\Loc
alCache\local-packages\Python312\site-packages\transformers\models\tapas\modeling_tf
_tapas.py:1465, in TFTapasForQuestionAnswering.call(self, input_ids, attention_mask,
token_type_ids, position_ids, head_mask, inputs_embeds, table_mask, aggregation_labe
ls, float_answer, numeric_values, numeric_values_scale, output_attentions, output_hi
dden_states, return_dict, labels, training)
   1391 @unpack_inputs
   1392 @add_start_docstrings_to_model_forward(TAPAS_INPUTS_DOCSTRING.format("batch_
size, sequence_length"))
   1393 @replace_return_docstrings(output_type=TFTableQuestionAnsweringOutput, confi
g_class=_CONFIG_FOR_DOC)
   (...)
   1411     training: Optional[bool] = False,
   1412 ) -> Union[TFTableQuestionAnsweringOutput, Tuple[tf.Tensor]]:
   1413     r"""
   1414     table_mask (`tf.Tensor` of shape `(batch_size, seq_length)`, *optional
*):
   1415         Mask for the table. Indicates which tokens belong to the table (1).
Question tokens, table headers and
   (...)
   1462     >>> logits_aggregation = outputs.logits_aggregation
   1463     ```"""
```



```
-> 1465     outputs = self.tapas(
   1466         input_ids=input_ids,
   1467         attention_mask=attention_mask,
   1468         token_type_ids=token_type_ids,
   1469         position_ids=position_ids,
   1470         head_mask=head_mask,
   1471         inputs_embeds=inputs_embeds,
   1472         output_attentions=output_attentions,
   1473         output_hidden_states=output_hidden_states,
   1474         return_dict=return_dict,
   1475         training=training,
   1476     )
   1478     sequence_output = outputs[0]
   1479     pooled_output = outputs[1]

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\Loc
alCache\local-packages\Python312\site-packages\transformers\modeling_tf_utils.py:43
7, in unpack_inputs.<locals>.run_call_with_unpacked_inputs(self, *args, **kwargs)
    434     config = self.config
    436 unpacked_inputs = input_processing(func, config, **fn_args_and_kwargs)
--> 437 return func(self, **unpacked_inputs)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\Loc
alCache\local-packages\Python312\site-packages\transformers\models\tapas\modeling_tf
_tapas.py:888, in TFTapasMainLayer.call(self, input_ids, attention_mask, token_type_
ids, position_ids, head_mask, inputs_embeds, output_attentions, output_hidden_state
s, return_dict, training)
    885 if token_type_ids is None:
    886     token_type_ids = tf.fill(dims=input_shape + [len(self.config.type_vocab_
sizes)], value=0)
--> 888 embedding_output = self.embeddings(
    889     input_ids=input_ids,
    890     position_ids=position_ids,
    891     token_type_ids=token_type_ids,
    892     inputs_embeds=inputs_embeds,
    893     training=training,
    894 )
    896 # We create a 3D attention mask from a 2D tensor mask.
    897 # Sizes are [batch_size, 1, 1, to_seq_length]
    898 # So we can broadcast to [batch_size, num_heads, from_seq_length, to_seq_len
gth]
    899 # this attention mask is more simple than the triangular masking of causal a
ttention
    900 # used in OpenAI GPT, we just need to prepare the broadcast dimension here.
    901 extended_attention_mask = tf.reshape(attention_mask, (input_shape[0], 1, 1,
input_shape[1]))

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\Loc
alCache\local-packages\Python312\site-packages\transformers\models\tapas\modeling_tf
_tapas.py:226, in TFTapasEmbeddings.call(self, input_ids, position_ids, token_type_i
ds, inputs_embeds, training)
    224 for i in range(self.number_of_token_type_embeddings):
    225     name = f"token_type_embeddings_{i}"
--> 226     final_embeddings += tf.gather(params=getattr(self, name), indices=token_
type_ids[:, :, i])
    228 final_embeddings = self.LayerNorm(inputs=final_embeddings)
    229 final_embeddings = self.dropout(inputs=final_embeddings, training=training)

InvalidArgumentError: Exception encountered when calling layer 'embeddings' (type TF
TapasEmbeddings).

{{function_node __wrapped__ResourceGather_device_/job:localhost/replica:0/task:0/dev
ice:CPU:0}} indices[1,160] = 5341 is not in [0, 256) [Op:ResourceGather] name:

Call arguments received by layer 'embeddings' (type TFTapasEmbeddings):
  • input_ids=tf.Tensor(shape=(3, 512), dtype=int32)
  • position_ids=None
  • token_type_ids=tf.Tensor(shape=(3, 512, 7), dtype=int32)
  • inputs_embeds=None
  • training=False
```

Use the newly fine-tune model against soccer dataset!

TBD