AI Platform: Qwik Start

This lab gives you an introductory, end-to-end experience of training and prediction on AI Platform. The lab will use a census dataset to:

- Create a TensorFlow 2.x training application and validate it locally.
- Run your training job on a single worker instance in the cloud.
- Deploy a model to support prediction.
- Request an online prediction and see the response.

In [1]:
```
import os
```

Step 1: Get your training data

The relevant data files, adult.data and adult.test, are hosted in a public Cloud Storage bucket.

You can read the files directly from Cloud Storage or copy them to your local environment. For this lab you will download the samples for local training, and later upload them to your own Cloud Storage bucket for cloud training.

Run the following command to download the data to a local file directory and set variables that point to the downloaded data files:

In [2]:
```
%%bash

mkdir data
gsutil -m cp gs://cloud-samples-data/ml-engine/census/data/* data/
```

```
Copying gs://cloud-samples-data/ml-engine/census/data/adult.test.csv...
Copying gs://cloud-samples-data/ml-engine/census/data/census.test.csv...
Copying gs://cloud-samples-data/ml-engine/census/data/adult.data.csv...
Copying gs://cloud-samples-data/ml-engine/census/data/census.train.csv...
Copying gs://cloud-samples-data/ml-engine/census/data/test.csv...
Copying gs://cloud-samples-data/ml-engine/census/data/test.json...
/ [6/6 files][ 10.7 MiB/ 10.7 MiB] 100% Done
Operation completed over 6 objects/10.7 MiB.
```

In [4]:
```
%%bash

export TRAIN_DATA=$(pwd)/data/adult.data.csv
export EVAL_DATA=$(pwd)/data/adult.test.csv
```

Inspect what the data looks like by looking at the first couple of rows:

In [5]:
```
%%bash

head data/adult.data.csv
```

```
39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K
50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 13, United-States, <=50K
38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners, Not-in-family, White, Male, 0, 0, 40, United-States, <=50K
53, Private, 234721, 11th, 7, Married-civ-spouse, Handlers-cleaners, Husband, Black, Male, 0, 0, 40, United-States, <=50K
28, Private, 338409, Bachelors, 13, Married-civ-spouse, Prof-specialty, Wife, Black, Female, 0, 0, 40, Cuba, <=50K
37, Private, 284582, Masters, 14, Married-civ-spouse, Exec-managerial, Wife, White, Female, 0, 0, 40, United-States, <=50K
49, Private, 160187, 9th, 5, Married-spouse-absent, Other-service, Not-in-family, Black, Female, 0, 0, 16, Jamaica, <=50K
52, Self-emp-not-inc, 209642, HS-grad, 9, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 45, United-States, >50K
31, Private, 45781, Masters, 14, Never-married, Prof-specialty, Not-in-family, White, Female, 14084, 0, 50, United-States, >50K
42, Private, 159449, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 5178, 0, 40, United-States, >50K
```

Step 2: Run a local training job

A local training job loads your Python training program and starts a training process in an environment that's similar to that of a live Cloud AI Platform cloud training job.

Step 2.1: Create files to hold the Python program

To do that, let's create three files. The first, called util.py, will contain utility methods for cleaning and preprocessing the data, as well as performing any feature engineering needed by transforming and normalizing the data.

In [6]:
```
%%bash
mkdir -p trainer
touch trainer/__init__.py
```

In [7]:
```
%%writefile trainer/util.py
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
from six.moves import urllib
import tempfile

import numpy as np
import pandas as pd
import tensorflow as tf

# Storage directory
DATA_DIR = os.path.join(tempfile.gettempdir(), 'census_data')
```

```
# Download options.
DATA_URL = (
    'https://storage.googleapis.com/cloud-samples-data/ai-platform/census'
    '/data')
TRAINING_FILE = 'adult.data.csv'
EVAL_FILE = 'adult.test.csv'
TRAINING_URL = '%s/%s' % (DATA_URL, TRAINING_FILE)
EVAL_URL = '%s/%s' % (DATA_URL, EVAL_FILE)

# These are the features in the dataset.
# Dataset information: https://archive.ics.uci.edu/ml/datasets/census+income
_CSV_COLUMNS = [
    'age', 'workclass', 'fnlwgt', 'education', 'education_num',
    'marital_status', 'occupation', 'relationship', 'race', 'gender',
    'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
    'income_bracket'
]

# This is the label (target) we want to predict.
_LABEL_COLUMN = 'income_bracket'

# These are columns we will not use as features for training. There are many
# reasons not to use certain attributes of data for training. Perhaps their
# values are noisy or inconsistent, or perhaps they encode bias that we do not
# want our model to learn. For a deep dive into the features of this Census
# dataset and the challenges they pose, see the Introduction to ML Fairness
# Notebook: https://colab.research.google.com/github/google/eng-edu/blob
# /master/ml/cc/exercises/intro_to_fairness.ipynb
UNUSED_COLUMNS = ['fnlwgt', 'education', 'gender']

_CATEGORICAL_TYPES = {
    'workclass': pd.api.types.CategoricalDtype(categories=[
        'Federal-gov', 'Local-gov', 'Never-worked', 'Private', 'Self-emp-inc',
        'Self-emp-not-inc', 'State-gov', 'Without-pay'
    ]),
    'marital_status': pd.api.types.CategoricalDtype(categories=[
        'Divorced', 'Married-AF-spouse', 'Married-civ-spouse',
        'Married-spouse-absent', 'Never-married', 'Separated', 'Widowed'
    ]),
    'occupation': pd.api.types.CategoricalDtype([
        'Adm-clerical', 'Armed-Forces', 'Craft-repair', 'Exec-managerial',
        'Farming-fishing', 'Handlers-cleaners', 'Machine-op-inspct',
        'Other-service', 'Priv-house-serv', 'Prof-specialty', 'Protective-serv',
        'Sales', 'Tech-support', 'Transport-moving'
    ]),
    'relationship': pd.api.types.CategoricalDtype(categories=[
        'Husband', 'Not-in-family', 'Other-relative', 'Own-child', 'Unmarried',
        'Wife'
    ]),
    'race': pd.api.types.CategoricalDtype(categories=[
        'Amer-Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other', 'White'
    ]),
    'native_country': pd.api.types.CategoricalDtype(categories=[
        'Cambodia', 'Canada', 'China', 'Columbia', 'Cuba', 'Dominican-Republic',
        'Ecuador', 'El-Salvador', 'England', 'France', 'Germany', 'Greece',
        'Guatemala', 'Haiti', 'Holand-Netherlands', 'Honduras', 'Hong',
        'Hungary',
        'India', 'Iran', 'Ireland', 'Italy', 'Jamaica', 'Japan', 'Laos',
        'Mexico',
        'Nicaragua', 'Outlying-US(Guam-USVI-etc)', 'Peru', 'Philippines',
        'Poland',
        'Portugal', 'Puerto-Rico', 'Scotland', 'South', 'Taiwan', 'Thailand',
        'Trinadad&Tobago', 'United-States', 'Vietnam', 'Yugoslavia'
    ]),
    'income_bracket': pd.api.types.CategoricalDtype(categories=[
        '<=50K', '>50K'
    ])
}


def _download_and_clean_file(filename, url):
    """Downloads data from url, and makes changes to match the CSV format.

    The CSVs may use spaces after the comma delimters (non-standard) or include
    rows which do not represent well-formed examples. This function strips out
    some of these problems.

    Args:
      filename: filename to save url to
      url: URL of resource to download
    """
    temp_file, _ = urllib.request.urlretrieve(url)
    with tf.io.gfile.GFile(temp_file, 'r') as temp_file_object:
        with tf.io.gfile.GFile(filename, 'w') as file_object:
            for line in temp_file_object:
                line = line.strip()
                line = line.replace(', ', ',')
                if not line or ',' not in line:
                    continue
                if line[-1] == '.':
                    line = line[:-1]
                line += '\n'
                file_object.write(line)
    tf.io.gfile.remove(temp_file)


def download(data_dir):
    """Downloads census data if it is not already present.
```

```
    Args:
      data_dir: directory where we will access/save the census data
    """
    tf.io.gfile.makedirs(data_dir)

    training_file_path = os.path.join(data_dir, TRAINING_FILE)
    if not tf.io.gfile.exists(training_file_path):
        _download_and_clean_file(training_file_path, TRAINING_URL)

    eval_file_path = os.path.join(data_dir, EVAL_FILE)
    if not tf.io.gfile.exists(eval_file_path):
        _download_and_clean_file(eval_file_path, EVAL_URL)

    return training_file_path, eval_file_path


def preprocess(dataframe):
    """Converts categorical features to numeric. Removes unused columns.

    Args:
      dataframe: Pandas dataframe with raw data

    Returns:
      Dataframe with preprocessed data
    """
    dataframe = dataframe.drop(columns=UNUSED_COLUMNS)

    # Convert integer valued (numeric) columns to floating point
    numeric_columns = dataframe.select_dtypes(['int64']).columns
    dataframe[numeric_columns] = dataframe[numeric_columns].astype('float32')

    # Convert categorical columns to numeric
    cat_columns = dataframe.select_dtypes(['object']).columns
    dataframe[cat_columns] = dataframe[cat_columns].apply(lambda x: x.astype(
        _CATEGORICAL_TYPES[x.name]))
    dataframe[cat_columns] = dataframe[cat_columns].apply(lambda x: x.cat.codes)
    return dataframe


def standardize(dataframe):
    """Scales numerical columns using their means and standard deviation to get
    z-scores: the mean of each numerical column becomes 0, and the standard
    deviation becomes 1. This can help the model converge during training.

    Args:
      dataframe: Pandas dataframe

    Returns:
      Input dataframe with the numerical columns scaled to z-scores
    """
    dtypes = list(zip(dataframe.dtypes.index, map(str, dataframe.dtypes)))
    # Normalize numeric columns.
    for column, dtype in dtypes:
        if dtype == 'float32':
            dataframe[column] -= dataframe[column].mean()
            dataframe[column] /= dataframe[column].std()
    return dataframe


def load_data():
    """Loads data into preprocessed (train_x, train_y, eval_y, eval_y)
    dataframes.

    Returns:
      A tuple (train_x, train_y, eval_x, eval_y), where train_x and eval_x are
      Pandas dataframes with features for training and train_y and eval_y are
      numpy arrays with the corresponding labels.
    """
    # Download Census dataset: Training and eval csv files.
    training_file_path, eval_file_path = download(DATA_DIR)

    # This census data uses the value '?' for missing entries. We use
    # na_values to
    # find ? and set it to NaN.
    # https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv
    # .html
    train_df = pd.read_csv(training_file_path, names=_CSV_COLUMNS,
                           na_values='?')
    eval_df = pd.read_csv(eval_file_path, names=_CSV_COLUMNS, na_values='?')

    train_df = preprocess(train_df)
    eval_df = preprocess(eval_df)

    # Split train and eval data with labels. The pop method copies and removes
    # the label column from the dataframe.
    train_x, train_y = train_df, train_df.pop(_LABEL_COLUMN)
    eval_x, eval_y = eval_df, eval_df.pop(_LABEL_COLUMN)

    # Join train_x and eval_x to normalize on overall means and standard
    # deviations. Then separate them again.
    all_x = pd.concat([train_x, eval_x], keys=['train', 'eval'])
    all_x = standardize(all_x)
    train_x, eval_x = all_x.xs('train'), all_x.xs('eval')

    # Reshape label columns for use with tf.data.Dataset
    train_y = np.asarray(train_y).astype('float32').reshape((-1, 1))
    eval_y = np.asarray(eval_y).astype('float32').reshape((-1, 1))
```

```
    return train_x, train_y, eval_x, eval_y
```

```
Writing trainer/util.py
```

The second file, called model.py, defines the input function and the model architecture. In this example, we use tf.data API for the data pipeline and create the model using the Keras Sequential API. We define a DNN with an input layer and 3 additonal layers using the Relu activation function. Since the task is a binary classification, the output layer uses the sigmoid activation.

In [8]:

```
%%writefile trainer/model.py
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf


def input_fn(features, labels, shuffle, num_epochs, batch_size):
    """Generates an input function to be used for model training.

    Args:
      features: numpy array of features used for training or inference
      labels: numpy array of labels for each example
      shuffle: boolean for whether to shuffle the data or not (set True for
        training, False for evaluation)
      num_epochs: number of epochs to provide the data for
      batch_size: batch size for training

    Returns:
      A tf.data.Dataset that can provide data to the Keras model for training or
        evaluation
    """
    if labels is None:
        inputs = features
    else:
        inputs = (features, labels)
    dataset = tf.data.Dataset.from_tensor_slices(inputs)

    if shuffle:
        dataset = dataset.shuffle(buffer_size=len(features))

    # We call repeat after shuffling, rather than before, to prevent separate
    # epochs from blending together.
    dataset = dataset.repeat(num_epochs)
    dataset = dataset.batch(batch_size)
    return dataset


def create_keras_model(input_dim, learning_rate):
    """Creates Keras Model for Binary Classification.

    The single output node + Sigmoid activation makes this a Logistic
    Regression.

    Args:
      input_dim: How many features the input has
      learning_rate: Learning rate for training

    Returns:
      The compiled Keras model (still needs to be trained)
    """
    Dense = tf.keras.layers.Dense
    model = tf.keras.Sequential(
        [
            Dense(100, activation=tf.nn.relu, kernel_initializer='uniform',
                  input_shape=(input_dim,)),
            Dense(75, activation=tf.nn.relu),
            Dense(50, activation=tf.nn.relu),
            Dense(25, activation=tf.nn.relu),
            Dense(1, activation=tf.nn.sigmoid)
        ])

    # Custom Optimizer:
    # https://www.tensorflow.org/api_docs/python/tf/train/RMSPropOptimizer
    optimizer = tf.keras.optimizers.RMSprop(lr=learning_rate)

    # Compile Keras model
    model.compile(
        loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
```

```
Writing trainer/model.py
```

The last file, called task.py, trains on data loaded and preprocessed in util.py. Using the tf.distribute.MirroredStrategy() scope, it is possible to train on a distributed fashion. The trained model is then saved in a TensorFlow SavedModel format.

In [9]:

```
%%writefile trainer/task.py
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import os

from . import model
from . import util
```

```
import tensorflow as tf


def get_args():
    """Argument parser.

    Returns:
      Dictionary of arguments.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--job-dir',
        type=str,
        required=True,
        help='local or GCS location for writing checkpoints and exporting '
             'models')
    parser.add_argument(
        '--num-epochs',
        type=int,
        default=20,
        help='number of times to go through the data, default=20')
    parser.add_argument(
        '--batch-size',
        default=128,
        type=int,
        help='number of records to read during each training step, default=128')
    parser.add_argument(
        '--learning-rate',
        default=.01,
        type=float,
        help='learning rate for gradient descent, default=.01')
    parser.add_argument(
        '--verbosity',
        choices=['DEBUG', 'ERROR', 'FATAL', 'INFO', 'WARN'],
        default='INFO')
    args, _ = parser.parse_known_args()
    return args


def train_and_evaluate(args):
    """Trains and evaluates the Keras model.

    Uses the Keras model defined in model.py and trains on data loaded and
    preprocessed in util.py. Saves the trained model in TensorFlow SavedModel
    format to the path defined in part by the --job-dir argument.

    Args:
      args: dictionary of arguments - see get_args() for details
    """

    train_x, train_y, eval_x, eval_y = util.load_data()

    # dimensions
    num_train_examples, input_dim = train_x.shape
    num_eval_examples = eval_x.shape[0]

    # Create the Keras Model
    keras_model = model.create_keras_model(
        input_dim=input_dim, learning_rate=args.learning_rate)

    # Pass a numpy array by passing DataFrame.values
    training_dataset = model.input_fn(
        features=train_x.values,
        labels=train_y,
        shuffle=True,
        num_epochs=args.num_epochs,
        batch_size=args.batch_size)

    # Pass a numpy array by passing DataFrame.values
    validation_dataset = model.input_fn(
        features=eval_x.values,
        labels=eval_y,
        shuffle=False,
        num_epochs=args.num_epochs,
        batch_size=num_eval_examples)

    # Setup Learning Rate decay.
    lr_decay_cb = tf.keras.callbacks.LearningRateScheduler(
        lambda epoch: args.learning_rate + 0.02 * (0.5 ** (1 + epoch)),
        verbose=True)

    # Setup TensorBoard callback.
    tensorboard_cb = tf.keras.callbacks.TensorBoard(
        os.path.join(args.job_dir, 'keras_tensorboard'),
        histogram_freq=1)

    # Train model
    keras_model.fit(
        training_dataset,
        steps_per_epoch=int(num_train_examples / args.batch_size),
        epochs=args.num_epochs,
        validation_data=validation_dataset,
        validation_steps=1,
        verbose=1,
        callbacks=[lr_decay_cb, tensorboard_cb])

    export_path = os.path.join(args.job_dir, 'keras_export')
    tf.keras.models.save_model(keras_model, export_path)
```

```
    print('Model exported to: {}'.format(export_path))


if __name__ == '__main__':
    strategy = tf.distribute.MirroredStrategy()
    with strategy.scope():
        args = get_args()
        tf.compat.v1.logging.set_verbosity(args.verbosity)
        train_and_evaluate(args)
```

```
Writing trainer/task.py
```

Step 2.2: Run a training job locally using the Python training program

NOTE When you run the same training job on AI Platform later in the lab, you'll see that the command is not much different from the above.

Specify an output directory and set a MODEL_DIR variable to hold the trained model, then run the training job locally by running the following command (by default, verbose logging is turned off. You can enable it by setting the --verbosity tag to DEBUG):

In [10]:

```
%%bash

MODEL_DIR=output
gcloud ai-platform local train \
    --module-name trainer.task \
    --package-path trainer/ \
    --job-dir $MODEL_DIR \
    -- \
    --train-files $TRAIN_DATA \
    --eval-files $EVAL_DATA \
    --train-steps 1000 \
    --eval-steps 100
```

```
Epoch 1/20

Epoch 00001: LearningRateScheduler setting learning rate to 0.02.
254/254 [==============================] - 6s 11ms/step - loss: 0.6414 - accuracy: 0.7865 - val_loss: 0.3761 - val_accuracy: 0.8270
Epoch 2/20

Epoch 00002: LearningRateScheduler setting learning rate to 0.015.
254/254 [==============================] - 2s 7ms/step - loss: 0.3653 - accuracy: 0.8313 - val_loss: 0.3330 - val_accuracy: 0.8459
Epoch 3/20

Epoch 00003: LearningRateScheduler setting learning rate to 0.0125.
254/254 [==============================] - 1s 6ms/step - loss: 0.3431 - accuracy: 0.8403 - val_loss: 0.3266 - val_accuracy: 0.8481
Epoch 4/20

Epoch 00004: LearningRateScheduler setting learning rate to 0.01125.
254/254 [==============================] - 2s 8ms/step - loss: 0.3401 - accuracy: 0.8432 - val_loss: 0.3248 - val_accuracy: 0.8505
Epoch 5/20

Epoch 00005: LearningRateScheduler setting learning rate to 0.010625.
254/254 [==============================] - 2s 7ms/step - loss: 0.3340 - accuracy: 0.8438 - val_loss: 0.3370 - val_accuracy: 0.8470
Epoch 6/20

Epoch 00006: LearningRateScheduler setting learning rate to 0.0103125.
254/254 [==============================] - 1s 5ms/step - loss: 0.3324 - accuracy: 0.8468 - val_loss: 0.3357 - val_accuracy: 0.8493
Epoch 7/20

Epoch 00007: LearningRateScheduler setting learning rate to 0.01015625.
254/254 [==============================] - 1s 5ms/step - loss: 0.3309 - accuracy: 0.8476 - val_loss: 0.3497 - val_accuracy: 0.8395
Epoch 8/20

Epoch 00008: LearningRateScheduler setting learning rate to 0.010078125.
254/254 [==============================] - 2s 7ms/step - loss: 0.3293 - accuracy: 0.8473 - val_loss: 0.3315 - val_accuracy: 0.8497
Epoch 9/20

Epoch 00009: LearningRateScheduler setting learning rate to 0.0100390625.
254/254 [==============================] - 2s 7ms/step - loss: 0.3289 - accuracy: 0.8487 - val_loss: 0.3316 - val_accuracy: 0.8507
Epoch 10/20

Epoch 00010: LearningRateScheduler setting learning rate to 0.01001953125.
254/254 [==============================] - 2s 6ms/step - loss: 0.3279 - accuracy: 0.8486 - val_loss: 0.3299 - val_accuracy: 0.8499
Epoch 11/20

Epoch 00011: LearningRateScheduler setting learning rate to 0.010009765625.
254/254 [==============================] - 2s 6ms/step - loss: 0.3290 - accuracy: 0.8490 - val_loss: 0.3337 - val_accuracy: 0.8521
Epoch 12/20

Epoch 00012: LearningRateScheduler setting learning rate to 0.0100048828125000001.
254/254 [==============================] - 1s 6ms/step - loss: 0.3257 - accuracy: 0.8493 - val_loss: 0.3268 - val_accuracy: 0.8503
Epoch 13/20

Epoch 00013: LearningRateScheduler setting learning rate to 0.01000244140625.
254/254 [==============================] - 2s 6ms/step - loss: 0.3273 - accuracy: 0.8501 - val_loss: 0.3214 - val_accuracy: 0.8535
Epoch 14/20

Epoch 00014: LearningRateScheduler setting learning rate to 0.010001220703125.
254/254 [==============================] - 2s 6ms/step - loss: 0.3246 - accuracy: 0.8500 - val_loss: 0.3256 - val_accuracy: 0.8507
Epoch 15/20

Epoch 00015: LearningRateScheduler setting learning rate to 0.0100006103515625.
254/254 [==============================] - 2s 7ms/step - loss: 0.3257 - accuracy: 0.8513 - val_loss: 0.3236 - val_accuracy: 0.8508
Epoch 16/20

Epoch 00016: LearningRateScheduler setting learning rate to 0.01000030517578125.
254/254 [==============================] - 2s 7ms/step - loss: 0.3262 - accuracy: 0.8495 - val_loss: 0.3297 - val_accuracy: 0.8507
```

```
Epoch 17/20

Epoch 00017: LearningRateScheduler setting learning rate to 0.010000152587890625.
254/254 [==============================] - 1s 5ms/step - loss: 0.3240 - accuracy: 0.8504 - val_loss: 0.3438 - val_accuracy: 0.8500
Epoch 18/20

Epoch 00018: LearningRateScheduler setting learning rate to 0.010000076293945313.
254/254 [==============================] - 1s 6ms/step - loss: 0.3222 - accuracy: 0.8511 - val_loss: 0.3364 - val_accuracy: 0.8506
Epoch 19/20

Epoch 00019: LearningRateScheduler setting learning rate to 0.010000038146972657.
254/254 [==============================] - 2s 7ms/step - loss: 0.3229 - accuracy: 0.8495 - val_loss: 0.3281 - val_accuracy: 0.8467
Epoch 20/20

Epoch 00020: LearningRateScheduler setting learning rate to 0.010000019073486329.
254/254 [==============================] - 1s 5ms/step - loss: 0.3230 - accuracy: 0.8514 - val_loss: 0.3229 - val_accuracy: 0.8475
Model exported to: output/keras_export
2021-10-22 03:55:47.520283: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 2.
Tune using inter_op_parallelism_threads for best performance.
WARNING:tensorflow:There are non-GPU devices in `tf.distribute.Strategy`, not using nccl allreduce.
2021-10-22 03:55:49.254323: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2021-10-22 03:55:49.254377: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2021-10-22 03:55:49.254931: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
2021-10-22 03:55:49.308915: W tensorflow/core/grappler/optimizers/data/auto_shard.cc:695] AUTO sharding policy will apply DATA sharding polic
y as it failed to apply FILE sharding policy because of the following reason: Found an unshardable source dataset: name: "TensorSliceDataset/
_2"
op: "TensorSliceDataset"
input: "Placeholder/_0"
input: "Placeholder/_1"
attr {
  key: "Toutput_types"
  value {
    list {
      type: DT_FLOAT
      type: DT_FLOAT
    }
  }
}
attr {
  key: "output_shapes"
  value {
    list {
      shape {
        dim {
          size: 11
        }
      }
      shape {
        dim {
          size: 1
        }
      }
    }
  }
}

2021-10-22 03:55:49.365493: W tensorflow/core/framework/dataset.cc:679] Input of GeneratorDatasetOp::Dataset will not be optimized because th
e dataset does not implement the AsGraphDefInternal() method needed to apply optimizations.
2021-10-22 03:55:49.368355: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled
(registered 2)
2021-10-22 03:55:52.631958: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2021-10-22 03:55:52.632902: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2021-10-22 03:55:52.646962: I tensorflow/core/profiler/lib/profiler_session.cc:66] Profiler session collecting data.
2021-10-22 03:55:52.657881: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
2021-10-22 03:55:52.684832: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: output/keras_tensorboard/train/plu
gins/profile/2021_10_22_03_55_52

2021-10-22 03:55:52.693737: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for trace.json.gz to output/k
eras_tensorboard/train/plugins/profile/2021_10_22_03_55_52/tensorflow-notebook.trace.json.gz
2021-10-22 03:55:52.721019: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: output/keras_tensorboard/train/plu
gins/profile/2021_10_22_03_55_52

2021-10-22 03:55:52.722521: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for memory_profile.json.gz to
output/keras_tensorboard/train/plugins/profile/2021_10_22_03_55_52/tensorflow-notebook.memory_profile.json.gz
2021-10-22 03:55:52.724629: I tensorflow/core/profiler/rpc/client/capture_profile.cc:251] Creating directory: output/keras_tensorboard/train/
plugins/profile/2021_10_22_03_55_52
Dumped tool data for xplane.pb to output/keras_tensorboard/train/plugins/profile/2021_10_22_03_55_52/tensorflow-notebook.xplane.pb
Dumped tool data for overview_page.pb to output/keras_tensorboard/train/plugins/profile/2021_10_22_03_55_52/tensorflow-notebook.overview_pag
e.pb
Dumped tool data for input_pipeline.pb to output/keras_tensorboard/train/plugins/profile/2021_10_22_03_55_52/tensorflow-notebook.input_pipeli
ne.pb
Dumped tool data for tensorflow_stats.pb to output/keras_tensorboard/train/plugins/profile/2021_10_22_03_55_52/tensorflow-notebook.tensorflow
_stats.pb
Dumped tool data for kernel_stats.pb to output/keras_tensorboard/train/plugins/profile/2021_10_22_03_55_52/tensorflow-notebook.kernel_stats.p
b

2021-10-22 03:55:53.808038: W tensorflow/core/grappler/optimizers/data/auto_shard.cc:695] AUTO sharding policy will apply DATA sharding polic
y as it failed to apply FILE sharding policy because of the following reason: Found an unshardable source dataset: name: "TensorSliceDataset/
_2"
op: "TensorSliceDataset"
input: "Placeholder/_0"
input: "Placeholder/_1"
attr {
  key: "Toutput_types"
  value {
    list {
      type: DT_FLOAT
      type: DT_FLOAT
```

```
    }
  }
}
attr {
  key: "output_shapes"
  value {
    list {
      shape {
        dim {
          size: 11
        }
      }
      shape {
        dim {
          size: 1
        }
      }
    }
  }
}
```

```
2021-10-22 03:56:20.474330: W tensorflow/core/framework/dataset.cc:679] Input of GeneratorDatasetOp::Dataset will not be optimized because th
e dataset does not implement the AsGraphDefInternal() method needed to apply optimizations.
2021-10-22 03:56:33.630675: W tensorflow/python/util/util.cc:348] Sets are not currently considered sequences, but this may change in the fut
ure, so consider avoiding using them.
/opt/conda/lib/python3.7/site-packages/keras/optimizer_v2/optimizer_v2.py:356: UserWarning: The `lr` argument is deprecated, use `learning_ra
te` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
INFO:tensorflow:Assets written to: output/keras_export/assets
```

Check if the output has been written to the output folder:

In [11]:
```
%%bash

ls output/keras_export/
```

```
assets
keras_metadata.pb
saved_model.pb
variables
```

Step 2.3: Prepare input for prediction

To receive valid and useful predictions, you must preprocess input for prediction in the same way that training data was preprocessed. In a production system, you may want to create a preprocessing pipeline that can be used identically at training time and prediction time.

For this exercise, use the training package's data-loading code to select a random sample from the evaluation data. This data is in the form that was used to evaluate accuracy after each epoch of training, so it can be used to send test predictions without further preprocessing.

Run the following snippet of code to preprocess the raw data from the adult.test.csv file. Here, we are grabbing 5 examples to run predictions on:

In [12]:
```
from trainer import util
_, _, eval_x, eval_y = util.load_data()

prediction_input = eval_x.sample(5)
prediction_targets = eval_y[prediction_input.index]
```

Check the numerical representation of the features by printing the preprocessed data:

In [13]:
```
print(prediction_input)
```

```
            age  workclass  education_num  marital_status  occupation  \
5197   0.171864          6      -0.419365               0           0
2515   0.463600          5      -0.030311               2           5
13472 -1.505617         -1      -1.197472               4          -1
9847  -0.265739          3       1.525903               2           3
4948   1.192940          6       1.136850               0           0

       relationship  race  capital_gain  capital_loss  hours_per_week  \
5197              4     4     -0.144807     -0.217119        0.369506
2515              0     4     -0.144807     -0.217119       -0.034043
13472             3     4     -0.144807     -0.217119       -0.034043
9847              0     4     -0.144807     -0.217119        0.369506
4948              1     4      1.266999     -0.217119       -0.034043

       native_country
5197               38
2515               38
13472              38
9847               38
4948               38
```

Notice that categorical fields, like occupation, have already been converted to integers (with the same mapping that was used for training). Numerical fields, like age, have been scaled to a z-score. Some fields have been dropped from the original data.

Export the prediction input to a newline-delimited JSON file:

In [14]:
```
import json

with open('test.json', 'w') as json_file:
  for row in prediction_input.values.tolist():
    json.dump(row, json_file)
    json_file.write('\n')
```

Inspect the .json file:

In [15]:
```
%%bash

cat test.json
```

```
[0.1718643307685852, 6.0, -0.4193645119667053, 0.0, 0.0, 4.0, 4.0, -0.14480669796466827, -0.2171185314655304, 0.36950573325157166, 38.0]
[0.46360018849372864, 5.0, -0.03031095676124096, 2.0, 5.0, 0.0, 4.0, -0.14480669796466827, -0.2171185314655304, -0.034042954444885254, 38.0]
[-1.5056167840957642, -1.0, -1.1974716186523438, 4.0, -1.0, 3.0, 4.0, -0.14480669796466827, -0.2171185314655304, -0.034042954444885254, 38.0]
[-0.26573944091796875, 3.0, 1.525903344154358, 2.0, 3.0, 0.0, 4.0, -0.14480669796466827, -0.2171185314655304, 0.36950573325157166, 38.0]
[1.1929397583007812, 6.0, 1.1368497610092163, 0.0, 0.0, 1.0, 4.0, 1.266999363899231, -0.2171185314655304, -0.034042954444885254, 38.0]
```

Step 2.4: Use your trained model for prediction

Once you've trained your TensorFlow model, you can use it for prediction on new data. In this case, you've trained a census model to predict income category given some information about a person.

Run the following command to run prediction on the test.json file we created above:

Note: If you get a "Bad magic number in .pyc file" error, go to the terminal and run:

> cd ../../usr/lib/google-cloud-sdk/lib/googlecloudsdk/command_lib/ml_engine/
>
> sudo rm *.pyc

In [16]:
```
%%bash

gcloud ai-platform local predict \
    --model-dir output/keras_export/ \
    --json-instances ./test.json
```

```
DENSE_4
[0.010347604751586914]
[0.3736071288585663]
[0.000253826379776001]
[0.7333061695098877]
[0.8493887186050415]
If the signature defined in the model is not serving_default then you must specify it via --signature-name flag, otherwise the command may fa
il.
WARNING: WARNING:tensorflow:From /opt/conda/lib/python3.7/site-packages/tensorflow/python/compat/v2_compat.py:101: disable_resource_variables
(from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
2021-10-22 03:56:39.343408: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 2.
Tune using inter_op_parallelism_threads for best performance.
WARNING:tensorflow:From /usr/lib/google-cloud-sdk/lib/third_party/ml_sdk/cloud/ml/prediction/frameworks/tf_prediction_lib.py:236: load (from
tensorflow.python.saved_model.loader_impl) is deprecated and will be removed in a future version.
Instructions for updating:
This function will only be available through the v1 compatibility library as tf.compat.v1.saved_model.loader.load or tf.compat.v1.saved_mode
l.load. There will be a new function for importing SavedModels in Tensorflow 2.0.
WARNING:tensorflow:From /usr/lib/google-cloud-sdk/lib/third_party/ml_sdk/cloud/ml/prediction/frameworks/tf_prediction_lib.py:236: load (from
tensorflow.python.saved_model.loader_impl) is deprecated and will be removed in a future version.
Instructions for updating:
This function will only be available through the v1 compatibility library as tf.compat.v1.saved_model.loader.load or tf.compat.v1.saved_mode
l.load. There will be a new function for importing SavedModels in Tensorflow 2.0.
WARNING:root:Error updating signature __saved_model_init_op: The name 'NoOp' refers to an Operation, not a Tensor. Tensor names must be of th
e form "<op_name>:<output_index>".
```

Since the model's last layer uses a sigmoid function for its activation, outputs between 0 and 0.5 represent negative predictions **("<=50K")** and outputs between 0.5 and 1 represent positive ones **(">50K")**.

Step 3: Run your training job in the cloud

Now that you've validated your model by running it locally, you will now get practice training using Cloud AI Platform.

Note: The initial job request will take several minutes to start, but subsequent jobs run more quickly. This enables quick iteration as you develop and validate your training job.

First, set the following variables:

In [22]:
```
%%bash
export PROJECT=$(gcloud config list project --format "value(core.project)")
echo "Your current GCP Project Name is: "${PROJECT}
```

```
Your current GCP Project Name is: qwiklabs-gcp-04-dfcf1e5e1598
```

In [24]:
```
PROJECT = "tensorflow-bucket"  # Replace with your project name
BUCKET_NAME=PROJECT+"-aiplatform"
REGION="us-central1"
```

In [25]:
```
os.environ["PROJECT"] = PROJECT
os.environ["BUCKET_NAME"] = BUCKET_NAME
os.environ["REGION"] = REGION
os.environ["TFVERSION"] = "2.1"
os.environ["PYTHONVERSION"] = "3.7"
```

Step 3.1: Set up a Cloud Storage bucket

The AI Platform services need to access Cloud Storage (GCS) to read and write data during model training and batch prediction.

Create a bucket using BUCKET_NAME as the name for the bucket and copy the data into it.

In [26]:
```
%%bash

if ! gsutil ls | grep -q gs://${BUCKET_NAME}; then
    gsutil mb -l ${REGION} gs://${BUCKET_NAME}
fi
gsutil cp -r data gs://$BUCKET_NAME/data
```

```
Creating gs://tensorflow-bucket-aiplatform/...
Copying file://data/census.test.csv [Content-Type=text/csv]...
Copying file://data/test.csv [Content-Type=text/csv]...
Copying file://data/test.json [Content-Type=application/json]...
Copying file://data/adult.test.csv [Content-Type=text/csv]...
- [4 files][  3.6 MiB/  3.6 MiB]
==> NOTE: You are performing a sequence of gsutil operations that may
run significantly faster if you instead use gsutil -m cp ... Please
see the -m section under "gsutil help options" for further information
about when gsutil -m can be advantageous.

Copying file://data/adult.data.csv [Content-Type=text/csv]...
Copying file://data/census.train.csv [Content-Type=text/csv]...
\ [6 files][ 10.7 MiB/ 10.7 MiB]
Operation completed over 6 objects/10.7 MiB.
```

Set the TRAIN_DATA and EVAL_DATA variables to point to the files:

In [27]:
```
%%bash
export TRAIN_DATA=gs://$BUCKET_NAME/data/adult.data.csv
export EVAL_DATA=gs://$BUCKET_NAME/data/adult.test.csv
```

Use gsutil again to copy the JSON test file test.json to your Cloud Storage bucket:

In [28]:
```
%%bash

gsutil cp test.json gs://$BUCKET_NAME/data/test.json
```

```
Copying file://test.json [Content-Type=application/json]...
/ [1 files][  692.0 B/  692.0 B]
Operation completed over 1 objects/692.0 B.
```

Set the TEST_JSON variable to point to that file:

In [29]:
```
%%bash

export TEST_JSON=gs://$BUCKET_NAME/data/test.json
```

Go back to the lab instructions and check your progress by testing the completed tasks:

- "Set up a Google Cloud Storage".

- "Upload the data files to your Cloud Storage bucket".

Step 3.2: Run a single-instance trainer in the cloud

With a validated training job that runs in both single-instance and distributed mode, you're now ready to run a training job in the cloud. For this example, we will be requesting a single-instance training job.

Use the default BASIC scale tier to run a single-instance training job. The initial job request can take a few minutes to start, but subsequent jobs run more quickly. This enables quick iteration as you develop and validate your training job.

Select a name for the initial training run that distinguishes it from any subsequent training runs. For example, we can use date and time to compose the job id.

Specify a directory for output generated by AI Platform by setting an OUTPUT_PATH variable to include when requesting training and prediction jobs. The OUTPUT_PATH represents the fully qualified Cloud Storage location for model checkpoints, summaries, and exports. You can use the BUCKET_NAME variable you defined in a previous step. It's a good practice to use the job name as the output directory.

Run the following command to submit a training job in the cloud that uses a single process. This time, set the --verbosity tag to DEBUG so that you can inspect the full logging output and retrieve accuracy, loss, and other metrics. The output also contains a number of other warning messages that you can ignore for the purposes of this sample:

In [30]:
```
%%bash

JOB_ID=census_$(date -u +%y%m%d_%H%M%S)
OUTPUT_PATH=gs://$BUCKET_NAME/$JOB_ID
gcloud ai-platform jobs submit training $JOB_ID \
    --job-dir $OUTPUT_PATH \
    --runtime-version $TFVERSION \
    --python-version $PYTHONVERSION \
    --module-name trainer.task \
    --package-path trainer/ \
    --region $REGION \
    -- \
    --train-files $TRAIN_DATA \
    --eval-files $EVAL_DATA \
    --train-steps 1000 \
```

```
    --eval-steps 100 \
    --verbosity DEBUG
```

```
jobId: census_211022_040729
state: QUEUED
Job [census_211022_040729] submitted successfully.
Your job is still active. You may view the status of your job with the command

  $ gcloud ai-platform jobs describe census_211022_040729

or continue streaming the logs with the command

  $ gcloud ai-platform jobs stream-logs census_211022_040729
```

Set an environment variable with the jobId generated above:

In [42]:
```
os.environ["JOB_ID"] = "census_211022_040729" # Replace with your job id
```

You can monitor the progress of your training job by watching the logs on the command line by running:

```
gcloud ai-platform jobs stream-logs $JOB_ID
```

Or monitor it in the Console at `AI Platform > Jobs` . Wait until your AI Platform training job is done. It is finished when you see a green check mark by the jobname in the Cloud Console, or when you see the message Job completed successfully from the Cloud Shell command line.

Go back to the lab instructions and check your progress by testing the completed task:

- "Run a single-instance trainer in the cloud".

Step 3.3: Deploy your model to support prediction

By deploying your trained model to AI Platform to serve online prediction requests, you get the benefit of scalable serving. This is useful if you expect your trained model to be hit with many prediction requests in a short period of time.

Create an AI Platform model:

In [43]:
```
os.environ["MODEL_NAME"] = "census"
```

In [44]:
```
%%bash

gcloud ai-platform models create $MODEL_NAME --regions=$REGION
```

```
Using endpoint [https://ml.googleapis.com/]
ERROR: (gcloud.ai-platform.models.create) Resource in projects [qwiklabs-gcp-04-dfcf1e5e1598] is the subject of a conflict: Field: model.name
Error: A model with the same name already exists.
- '@type': type.googleapis.com/google.rpc.BadRequest
  fieldViolations:
  - description: A model with the same name already exists.
    field: model.name
---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
/tmp/ipykernel_25947/1174739956.py in <module>
----> 1 get_ipython().run_cell_magic('bash', '', '\ngcloud ai-platform models create $MODEL_NAME --regions=$REGION\n')

/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py in run_cell_magic(self, magic_name, line, cell)
   2404             with self.builtin_trap:
   2405                 args = (magic_arg_s, cell)
-> 2406                 result = fn(*args, **kwargs)
   2407             return result
   2408

/opt/conda/lib/python3.7/site-packages/IPython/core/magics/script.py in named_script_magic(line, cell)
    140             else:
    141                 line = script
--> 142             return self.shebang(line, cell)
    143
    144         # write a basic docstring:

/opt/conda/lib/python3.7/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

/opt/conda/lib/python3.7/site-packages/IPython/core/magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/conda/lib/python3.7/site-packages/IPython/core/magics/script.py in shebang(self, line, cell)
    243             sys.stderr.flush()
    244         if args.raise_error and p.returncode!=0:
--> 245             raise CalledProcessError(p.returncode, cell, output=out, stderr=err)
    246
    247     def _run_script(self, p, cell, to_close):

CalledProcessError: Command 'b'\ngcloud ai-platform models create $MODEL_NAME --regions=$REGION\n'' returned non-zero exit status 1.
```

Set the environment variable MODEL_BINARIES to the full path of your exported trained model binaries `$OUTPUT_PATH/keras_export/` .

You'll deploy this trained model.

Run the following command to create a version v1 of your model:

In [45]:
```
%%bash

OUTPUT_PATH=gs://$BUCKET_NAME/$JOB_ID
MODEL_BINARIES=$OUTPUT_PATH/keras_export/
gcloud ai-platform versions create v1 \
--model $MODEL_NAME \
--origin $MODEL_BINARIES \
--runtime-version $TFVERSION \
--python-version $PYTHONVERSION \
--region=global
```

```
Using endpoint [https://ml.googleapis.com/]
Creating version (this might take a few minutes)......
.....................................................................................................................................
done.
```

It may take several minutes to deploy your trained model. When done, you can see a list of your models using the models list command:

In [46]:
```
%%bash

gcloud ai-platform models list
```

```
Using endpoint [https://us-central1-ml.googleapis.com/]
Listed 0 items.
```

Go back to the lab instructions and check your progress by testing the completed tasks:

- "Create an AI Platform model".

- "Create a version v1 of your model".

Step 3.4: Send an online prediction request to your deployed model

You can now send prediction requests to your deployed model. The following command sends a prediction request using the test.json.

The response includes the probabilities of each label **(>50K and <=50K)** based on the data entry in test.json, thus indicating whether the predicted income is greater than or less than 50,000 dollars.

In [47]:
```
%%bash

gcloud ai-platform predict \
--model $MODEL_NAME \
--version v1 \
--json-instances ./test.json \
--region global
```

```
DENSE_4
[0.0003304928250145167]
[0.44405555725097656]
[2.8857943107141182e-05]
[0.7626214623451233]
[0.9999933242797852]
Using endpoint [https://ml.googleapis.com/]
```

Note: AI Platform supports batch prediction, too, but it's not included in this lab. See the documentation for more info.

Go back to the lab instructions to answer some multiple choice questions to reinforce your uncerstanding of some of these lab's concepts.

Congratulations!

In this lab you've learned how to train a TensorFlow model both locally and on AI Platform, how to prepare data for prediction and to perform predictions both locally and in the Cloud AI Platform.

In [ ]:
```

```

In [ ]:
```

```