TensorFlow and Keras Lab Exercise

Palacode Narayana Iyer Anantharaman

14 Aug 2018

Goal of this lab session

- Get started with TF programming with the new concepts in TF 1.9 and 1.10 (See next slide for some high level concepts that we will experiment in a few lab sessions)

- Today: Work with Data API, Estimators

- Optional: TensorBoard

Key Topics

- Low level APIs and TF metaphor
- High Level APIs
- Estimators
- TF Serving
- TF Debugger
- TensorBoard

High Level APIs

- Keras layer integrated in to TF
- Eager Execution
- Estimators
- Data Pipelines

Datasets API

```
dataset = tf.data.Dataset.from_tensor_slices((data, labels))
dataset = dataset.batch(32).repeat()

val_dataset = tf.data.Dataset.from_tensor_slices((val_data, val_labels))
val_dataset = val_dataset.batch(32).repeat()

model.fit(dataset, epochs=10, steps_per_epoch=30,
        validation_data=val_dataset,
        validation_steps=3)
```

Evaluate and Predict

```
model.evaluate(x, y, batch_size=32)
model.evaluate(dataset, steps=30)
```

```
model.predict(x, batch_size=32)
model.predict(dataset, steps=30)
```

Saving/Restoring the model

- Save and Restore model weights

- Save and restore model configuration (JSON, YAML serialization)

- Save and Restore the entire model
  - Saves weight values, the model's configuration, optimizer's configuration
  - Allows checkpointing the model to resume later from exactly the same state even without the original source code

```
# Save entire model to a HDF5 file
model.save('my_model.h5')

# Recreate the exact same model, including weights and optimizer
model = keras.models.load_model('my_model.h5')
```

Eager Execution

- The TF compute graph metaphor where the model is specified symbolically requires compilation and run time binding to actual inputs.
- Often it is a bit confusing for those who expect the statements to be “interpreted” instantly as in Python
- Eager Execution allows instant execution without the need for starting a session, binding variables with feed_dict and so on
- This helps debugging as it provides instant feedback and also is more intuitive interface for a python developer as this is more “pythonic”

Sample Code

```
from __future__ import absolute_import, division, print_function
import tensorflow as tf

tf.enable_eager_execution()

tf.executing_eagerly()        # => True

x = [[2.]]

m = tf.matmul(x, x)

print("hello, {}".format(m))  # => "hello, [[4.]]"
```

Estimators

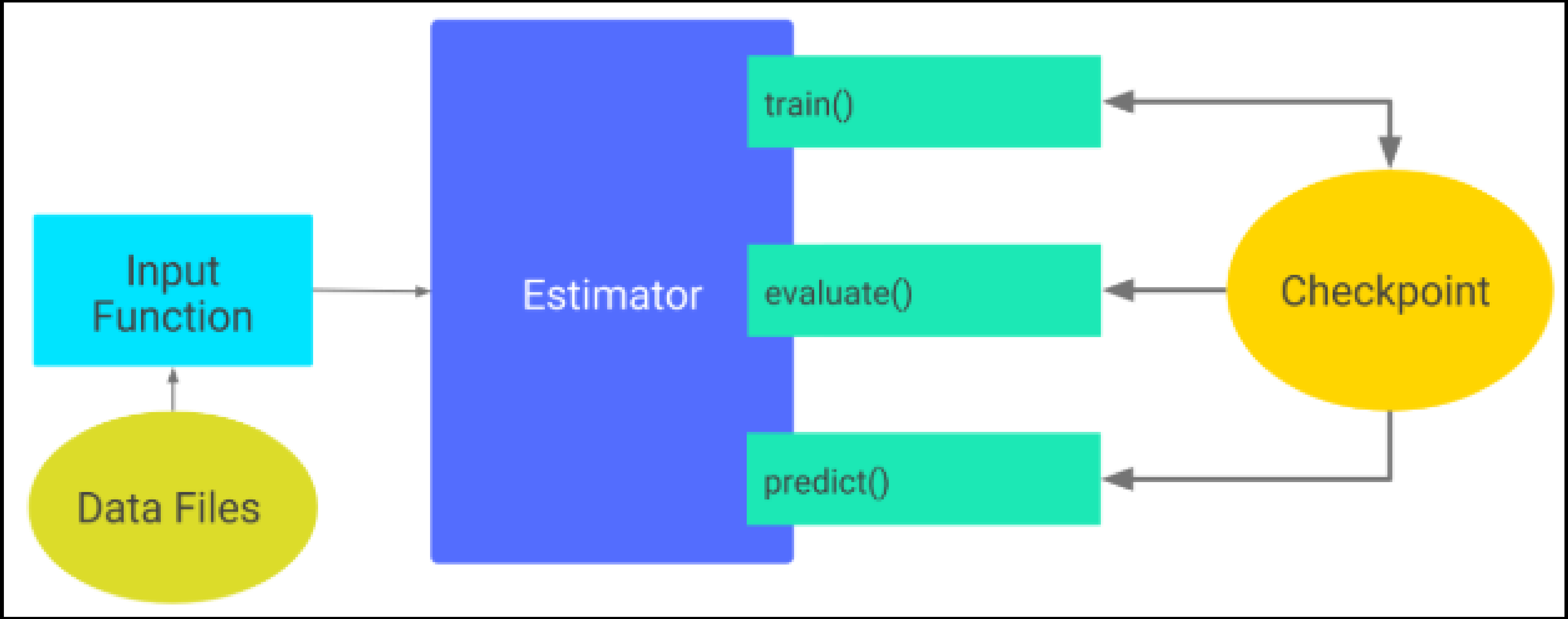
Estimators are high level APIs that have the following advantages:

- Estimators are easier to program as they are high level API
- They support distributed environment, on CPU/GPU/TPU without recoding
- Estimators make graph building transparent
- A number of pre-made estimators are available

Four Steps to a pre-made estimator application

1. Write one or more dataset importing functions
2. Define the feature columns
3. Instantiate the required pre-made estimator
4. Call a training, evaluation, predict function

Estimators and Checkpointing



Sample Code

```
def input_fn(dataset):
  ...  # manipulate dataset, extracting the feature dict and the label
  return feature_dict, label
# Define three numeric feature columns.
population = tf.feature_column.numeric_column('population')
crime_rate = tf.feature_column.numeric_column('crime_rate')
median_education = tf.feature_column.numeric_column('median_education',
                    normalizer_fn=lambda x: x - global_education_mean)
# Instantiate an estimator, passing the feature columns.
estimator = tf.estimator.LinearClassifier(feature_columns=[population, crime_rate,
median_education],)
# my_training_set is the function created in Step 1
estimator.train(input_fn=my_training_set, steps=2000)
```

Lab Assignment # Task1

- You are provided with the dataset for classification and regression: ds1.csv, ds2.csv, ds3.csv (10k samples in each file)
  - Link: https://drive.google.com/open?id=1iDmaOsQAv0U9_4jpoRHMf0QZ9J0f-BFs

- You are required to use the linear model estimators to perform the classification and regression tasks: Determine which datasets represent a linear target function

- For the dataset that has a linear behaviour, you are required to identify the formula f(x) and guess what the input and output quantities represent, report theta and bias values after de-normalizing ☺

Task#2

- There is 1 dataset for regression and 1 for classification. You are required to build a model for each of these tasks
- For the non linear target functions, build a Neural Network (Shallow and deep)
- Measure the MAE for regression, accuracy for classification and report the final accuracy along with the configuration (Architecture, Hyperparams) that you used to get the accuracy

Rules

- You are required to use TF 1.9 or 1.10 and use the Data API, Estimators
- You are required to determine the nature of target function only using the linear estimator. You shouldn’t visualize the input dataset
- You should demonstrate “eager execution”
- (Optional) Visualize the model with TensorBoard
- Report the metrics (MAE, Cross Entropy losses, Accuracies) as well as the configuration used to get them
- Participate in the brainstorm discussions (either after this lab or in the next class)