High Level API

Keras

tf.keras -- это **TensorFlow** реализация **Keras API** спецификации.
Представляет собой библиотеку для глубокого обучения. Существующие методы позволяют реализовывать код обучения, обработки данных, валидацию результатов и многое другое. В отличие от классического подхода Tensorflow 1.x, Keras предоставляет интерфейс, который внутри уже содержит выполнение необходимых операций, задавая значения для параметров, позволяет регулировать алгоритм метода.

- *User friendly*
- *Modular and composable*
- *Easy to extend*

На сегодняшний момент Keras практически полностью интегрирован в Tensorflow, что позволяет писать гибридный код и более удобный, уходя от создания экземпляров статического графа tf.Graph() и tf.Session().

Построение модели с помощью Keras.
Keras позволяет реализовать модель несколькими способами:

1. **Functional API**

Этот подход похож на то, как это можно сделать в Tensorflow: есть функции, которые создают необходимые компоненты модели, и это выстраивается в целую рабочую систему:

```
input = tf.keras.layers.Input(shape=(32, 32, 3))
output = tf.keras.layers.Dense(units=512)(input)
output = tf.keras.layers.Activation('relu')(output)
output = tf.keras.layers.Dropout(0.5)(output)
output = tf.keras.layers.Dense(units=10)(output)
output = tf.keras.layers.Activation('softmax')(output)

model = tf.keras.models.Model(inputs=input, outputs=output)
```

2. **Sequential API**

Модель представляет собой последовательную цепочку операция, где каждая операция строго идет согласно своей очереди. Такую модель можно реализовать с помощью класса tf.keras.models.Sequential.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(32, 32, 3)),
    tf.keras.layers.Dense(units=512),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=10),
    tf.keras.layers.Activation('softmax')
])
```

lo

!!!Важным моментом является в Keras то, что реализуемая модель в итоге является экземпляром одного из класса модуля ***tf.keras.models***. В дальнейшем *только* с

объектом данного класса реализуется и запускается весь процесс обучения и тестирования. Интуитивно очень понятное решение.

Основные методы Keras:
Для того чтобы подготовить данные и запустить модель обучаться, то у Keras есть основные методы, которые отвечают за эти процессы.

1. **compile**

Принимает три важных аргумента:

- optimizer: то, каким методом (градиентный спуск), будет происходить обновление обучаемых параметров.
- loss: целевая функция для поставленной задачи, которая будет минимизироваться с помощью градиентного спуска.
- metrics: метрики для подсчета точности, отслеживания качества обучения и валидации.

```
# Configure a model for categorical classification.
model.compile(optimizer=tf.keras.optimizers.RMSprop(0.01),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=[tf.keras.metrics.CategoricalAccuracy()])
```

Метод существует у созданной модели (метод класса tf.keras.models). Необходимо его вызвать, что указать, как будет организован процесс обучения.

В качестве optimizer, loss и metrics можно подать и свои кастомные, но только то, что предлагает Keras.

2. **fit**

Указывает модели, с какими данными работать в процессе обучения и валидации.
Принимает на вход четыре важных аргумента:

- x, y: данные и соответсвующие метки обучающей выборки
- epochs: количество эпох для обучения
- batch_size: размер батча для каждого шага
- validation_data: данные для валидации модели, принимает также сами данные и их метки.

```
model.fit(data, labels, epochs=10, batch_size=32,
          validation_data=(val_data, val_labels))
```

Метод существует у созданной модели (метод класса tf.keras.models). Необходимо его вызвать, что указать данные для модели.
В качестве данных для обучения и валидации можно подавать объект tf.data ю

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

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

model.fit(dataset, epochs=10,
          validation_data=val_dataset)
```

3. **evaluate**

Возвращает значение целевой функции и значение метрик на данных в режиме inference. Данные также могут быть, как NumPy массивами, так и объектом tf.data.

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

4. **predict**

Делает предсказание по данным, которые были поданы в данный метод.

```
predictions = model.predict(test_images)
```

Callbacks

Keras предоставляет возможность выполнять необходимые операция в определенный момент обучения, например остановить обучения после заданного количества эпох, или логировать в Tensorboard.

Для этого необходимо создать объект одного из класса модуля **tf.keras.callbacks**.

Keras предоставляет готовые callbacks, но можно создать и свой.

Созданный *callback* должен быть подан в метод **fit/evaluate/predict:**

```
callbacks = [
  # Interrupt training if `val_loss` stops improving for over 2 epochs
  tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_loss'),
  # Write TensorBoard logs to `./logs` directory
  tf.keras.callbacks.TensorBoard(log_dir='./logs')
]
model.fit(data, labels, batch_size=32, epochs=5, callbacks=callbacks,
          validation_data=(val_data, val_labels))
```

Callbacks

https://habr.com/ru/company/ods/blog/325432/

https://keras.io/callbacks/

**Источник: https://www.tensorflow.org/tutorials/keras/classification*

Сохранение модели

Keras предлагает сохранить модель целиком ***model.save('my_model.h5')***,

либо только обучаемые параметры ***model.save_weights('my_model.h5')***.

Модель можно загрузить с помощью метода

tf.keras.models.load_model('my_model.h5')

Что касается ***EagerExecution***, то он поддерживает *tf.keras*, но *tf.keras* может работать и без *EagerExecution* с *TensorFlow 1.x*
TensorFlow 2.0 вплотную работает с Keras.

Estimators

https://arxiv.org/pdf/1708.02637.pdf

Представляет собой высокоуровневый интерфейс для построения и обучения моделей. Основная цель данной библиотеки - построение кода, предназначенного для распределенного обучения. Estimators используется в TensorFlow Extended (TFX).

Модель можно создать, как с помощью TensorFlow операций, так и с помощью Keras.

Объект класса Estimator можно создать двумя способами:

1. С помощью средств TensorFlow и **tf.estimator:**

```
image_classifier = tf.estimator.Estimator(model_fn=self.model_fn,
                                          model_dir=MODEL_DIR)
```

Функция ***model_fn*** содержит в себе создание модели и поэтапного процесса обучения, а также валидации через ***tf.estimator.EstimatorSpec.***
При создании объекта *estimator* функция *model_fn* вызывается для построения графа, поэтому в данной функции необходимо указать, что будет происходить в процессе обучения, валидации или инференса. Для этого Estimator предлагает три варианта режима, которые помогают в дальнейшем идентифицировать, какой код должен исполняться:

1) tf.estimator.ModeKeys.TRAIN
2) tf.estimator.ModeKeys.EVAL
3) tf.estimator.ModeKeys.PREDICT

Для каждого режима создаются свой объект tf.estimator.EstimatorSpec. Примеры того, как это может выглядеть

```
if mode == tf.estimator.ModeKeys.EVAL:
    return tf.estimator.EstimatorSpec(
        mode=mode,
        loss=cross_entropy,
        eval_metric_ops={'accuracy/accuracy': accuracy},
        evaluation_hooks=valid_hook_list)

if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode,
                                      predictions=predictions)

if mode == tf.estimator.ModeKeys.TRAIN:
    return tf.estimator.EstimatorSpec(
        mode=mode,
        loss=cross_entropy,
        train_op=train_op,
        training_hooks=train_hook_list)
```

То есть результатом функции ***model_fn*** является объект **tf.estimator.EstimatorSpec**. И в зависимости от того, какой процесс происходит, будет выполняться соответствующий код.

2. С помощью **Keras:**

```
image_classifier = tf.keras.estimator.model_to_estimator(keras_model=model,
                                                          model_dir=MODEL_DIR)
```

keras_model -- это объект класса ***tf.keras.models.Model***

Существуют *Estimators*, предназначенные для конкретных задач: *LinearClassifier*, *DNNClassifier* и др. TensorFlow считает хорошей практикой использовать уже существующие Estimators, так как их код организован оптимально с точки зрения вычислений и построения вычислительного графа, а также лучшим образом реализовано логирование.

После того как объект estimator создан, то можно вызвать 2 важные функции:

1) **train**: запустить обучение, указав функцию, которая вернет обучающую выборку;
2) **evaluate**: проведет валидацию на соответствующей выборке;
3) **predict**: вернет предсказания на обученной модели для поданных данных.

```
image_classifier.train(input_fn=
                       lambda : self.train_input_fn(self.x_train, self.y_train))

metrics = image_classifier.evaluate(input_fn=
                                    lambda : self.valid_input_fn(self.x_test, self.y_test))

predictions = image_classifier.predict(input_fn=
                                       lambda : self.valid_input_fn(self.x_test, self.y_test))
```