TensorFlow

Курс “Практическое применение по TensorFlow”
Шигапова Фирюза Зинатуллаевна
1-й семестр, 2019 г.

https://github.com/Firyuza/TensorFlowPractice

High Level API

1. Keras

`tf.keras`

2. Estimators

`tf.estimator`

Keras. Create model

1. **Functional** API

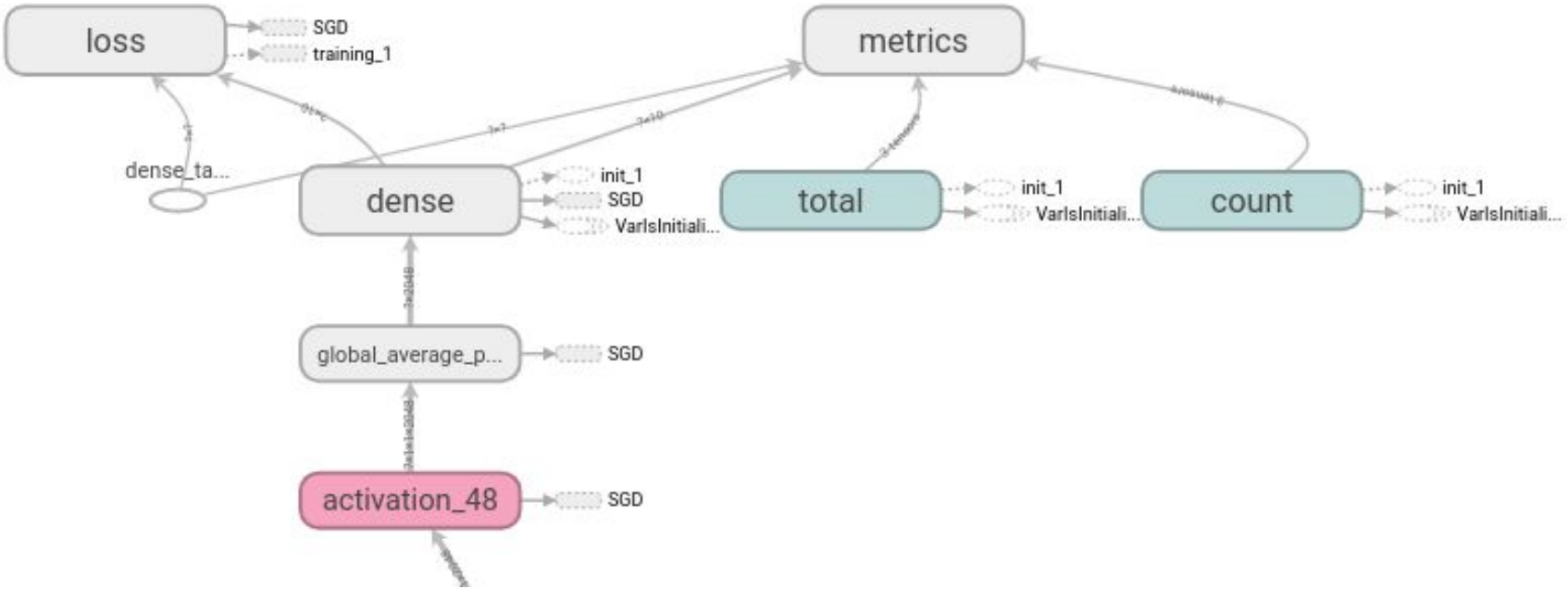
```
input = tf.keras.layers.Input(shape=(32, 32, 3))
output = tf.keras.layers.Dense(units=512)(input)
output = tf.keras.layers.Activation('relu')(output)
output = tf.keras.layers.Dropout(0.5)(output)
output = tf.keras.layers.Dense(units=10)(output)
output = tf.keras.layers.Activation('softmax')(output)

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

Keras. Create model

2. **Sequential** API

```
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')
])
```

Keras. It also has a Graph



Keras. Compile model

Define necessary components for model **training**:

- optimizer
- loss function
- metrics

```
# 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()])
```

Source: https://www.tensorflow.org/tutorials/keras/classification

Keras. Fit model

Define dataset for training and validation phases:

- training data (**tf.data** can be used)
- validation data (**tf.data** can be used)
- epochs
- batch size

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

Source: https://www.tensorflow.org/tutorials/keras/classification

Keras. Evaluate model

Returns loss and metrics values on valid/test set

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

Source: https://www.tensorflow.org/tutorials/keras/classification

Keras. Predict model

Returns predictions of trained model at inference

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

Source: https://www.tensorflow.org/tutorials/keras/classification

Keras. Callbacks

Customize the model behavior during training/evaluation/inference.

Pass callbacks to model.fit(...), model.evaluate(...) or model.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))
```

Custom Callback can be created

Source: https://www.tensorflow.org/tutorials/keras/classification

Keras. Save & restore model

```
model.save(path)
```

Save whole model

```
model.save_weights(path)
```

Save trainable variables only

```
model = tf.keras.models.load_model(path)
```

Estimator. Create estimator

1. Through **tf.estimator**

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

Estimator. Create estimator. model_fn

- **model_fn** contains code for training, evaluation, inference phases.
- Control each phase by current mode:

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

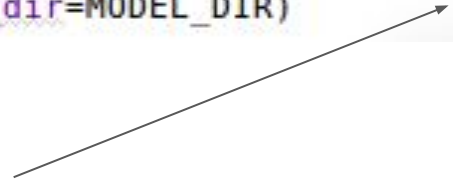
Estimator. Create estimator. model_fn

```
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)
```

This object should be returned by **model_fn**

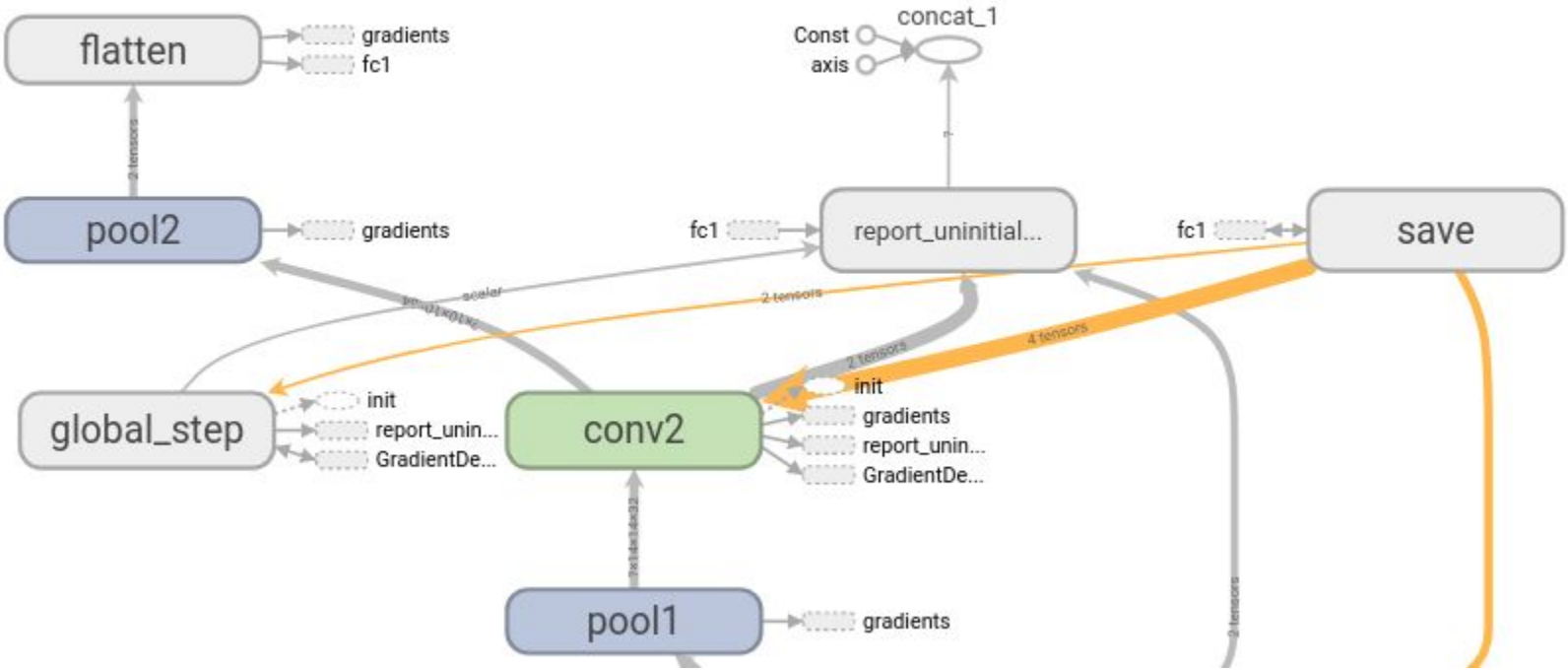
Estimator. Create estimator. Keras

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



Object of **tf.keras.models.Model**

Estimator. It also has a Graph



Estimator. Train & Evaluate & Predict by estimator

```
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))
```

Assignment

1. Create **Dataset** from *tf.data API*
2. Use *map(...), batch(...), shuffle(...), cache(...), repeat(...), prefetch(...)* in different orders and with different parameters.
3. Use different approaches:
   - *Transformation function takes single item from set and return multiple items.*
   - *In Transformation function try to use Database (fetch data from Database).*
   - *Put batch before map and vectorize Transformation function.*
   - *Cache transformed data*
   - *etc.*
4. Compare results