8 Keras 高层接口

Keras != tf.keras

- ✓ datasets
- ✓ layers
- ✓ losses
- ✓ metrics
- ✓ optimizers

8.1 Keras 高层 API

metrics

- ✓ update_state
- ✓ result().numpy()
- ✓ reset_states

Step 1. Build a meter

```
acc_meter = tf.keras.metrics.Accuracy()
loss_meter = tf.keras.metrics.Mean()
```

Step 2. Update data

```
loss_meter.update_state(loss)
acc_meter.update_state(y, pred)
```

Step 3. Get Average data

```
print(step, 'loss:', loss_meter.result().numpy())
print(step, 'Evaluate Acc', total_correct/total, acc_meter.result().n
umpy())
```

Clear buffer

```
if step % 100 == 0:
    print(step,'loss:',loss_meter.result().numpy())
    loss_meter.reset_states()
```

Compile & Fit

- ✓ compile
- ✓ fit
- ✓ evaluate
- ✓ predict

compile

Individual loss and optimize

```
    with tf.GradientTape() as tape:
        # [b, 28, 28] => [b, 784]
        x = tf.reshape(x, (-1, 28*28))
        # [b, 784] => [b, 10]
        out = network(x)
        # [b] => [b, 10]
        y_onehot = tf.one_hot(y, depth=10)
        # [b]
        loss = tf.reduce_mean(tf.losses.categorical_crossentropy(y_onehot, out,
from_logits=True))

    grads = tape.gradient(loss, network.trainable_variables)
    optimizer.apply_gradients(zip(grads, network.trainable_variables))
```

```
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )
```

Individual epoch and step

```
for epoch in range(epochs):

  for step, (x, y) in enumerate(db):
    ....
```

```
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )

network.fit(db, epochs=10)
```

Individual evaluation

```
    # evaluate
    if step % 500 == 0:
        total, total_correct = 0., 0

        for step, (x, y) in enumerate(ds_val):
            # [b, 28, 28] => [b, 784]
            x = tf.reshape(x, (-1, 28*28))
            # [b, 784] => [b, 10]
            out = network(x)
            # [b, 10] => [b]
            pred = tf.argmax(out, axis=1)
            pred = tf.cast(pred, dtype=tf.int32)
            # bool type
            correct = tf.equal(pred, y)
            # bool tensor => int tensor => numpy
            total_correct += tf.reduce_sum(tf.cast(correct,
dtype=tf.int32)).numpy()
            total += x.shape[0]

        print(step, 'Evaluate Acc:', total_correct/total)
```

```
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )

network.fit(db, epochs=10, validation_data=ds_val,
             validation_steps=2)
```

validation_freq

```
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )

network.fit(db, epochs=10, validation_data=ds_val,
             validation_steps=2)

network.evaluate(ds_val)
```

两者差不多，只不过前者是在训练的时候就做了一个测试，因为并不知道训练什么时候到头

Predict

```
sample = next(iter(ds_val))
x = sample[0]
y = sample[1] # one-hot
pred = network.predict(x) # [b, 10]
# convert back to number
y = tf.argmax(y, axis=1)
pred = tf.argmax(pred, axis=1)

print(pred)
print(y)
```

自定义网络层

- ✓ keras.Sequential
- ✓ keras.layers.Layer
- ✓ keras.Model

keras.Sequential

```
network = Sequential([layers.Dense(256, activation='relu'),
                      layers.Dense(128, activation='relu'),
                      layers.Dense(64, activation='relu'),
                      layers.Dense(32, activation='relu'),
                      layers.Dense(10)])
network.build(input_shape=(None, 28*28))
network.summary()
```

将参数打印出来

model.trainable_variables 和 model.call()

keras.layers.Layers & keras.Model

需要继承 Model 类，实现__init__、call 方法，可以使用 Model 类中的 compile、fit、evaluate 以及 predict 方法。

```
class MyDense(layers.Layer):

    def __init__(self, inp_dim, outp_dim):
        super(MyDense, self).__init__()

        self.kernel = self.add_variable('w', [inp_dim, outp_dim])
        self.bias = self.add_variable('b', [outp_dim])

    def call(self, inputs, training=None):

        out = inputs @ self.kernel + self.bias

        return out
```

```
class MyModel(keras.Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = MyDense(28*28, 256)
        self.fc2 = MyDense(256, 128)
        self.fc3 = MyDense(128, 64)
        self.fc4 = MyDense(64, 32)
        self.fc5 = MyDense(32, 10)

    def call(self, inputs, training=None):
        x = self.fc1(inputs)
        x = tf.nn.relu(x)
        x = self.fc2(x)
        x = tf.nn.relu(x)
        x = self.fc3(x)
        x = tf.nn.relu(x)
        x = self.fc4(x)
        x = tf.nn.relu(x)
        x = self.fc5(x)
        return x
```

8.2 模型的保存与加载

- ✓ save/load weights
- ✓ save/load entire model
- ✓ saved_model

save & load weights

```
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')

# Restore the weights
model = create_model()
model.load_weights('./checkpoints/my_checkpoint')

loss,acc = model.evaluate(test_images, test_labels)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

```
network.save_weights('weights.ckpt')
print('saved weights.')
del network

network = Sequential([layers.Dense(256, activation='relu'),
                      layers.Dense(128, activation='relu'),
                      layers.Dense(64, activation='relu'),
                      layers.Dense(32, activation='relu'),
                      layers.Dense(10)])
network.compile(optimizer=optimizers.Adam(lr=0.01),
        loss=tf.losses.CategoricalCrossentropy(from_logits=True),
        metrics=['accuracy']
    )
network.load_weights('weights.ckpt')
network.evaluate(ds_val)
```

save/load entire model

```
network.save('model.h5')
print('saved total model.')
del network

print('load model from file')
network = tf.keras.models.load_model('model.h5')

network.evaluate(x_val, y_val)
```

saved_model

```
tf.saved_model.save(m, '/tmp/saved_model/')

imported = tf.saved_model.load(path)
f = imported.signatures["serving_default"]
print(f(x=tf.ones([1, 28, 28, 3])))
```

8.3 keras 实战

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets,layers,optimizers,Sequential,me
trics

def preprocess(x,y):
    # x is a single image, not a batch
    x = tf.cast(x, dtype=tf.float32) / 255.
    y = tf.cast(y, dtype=tf.int32)
    return x,y

batchsz = 128
(x,y),(x_val,y_val) = datasets.cifar10.load_data()
print("original datasets:", y.shape, y_val.shape)   # (50000, 1) (100
00, 1)
y = tf.squeeze(y)
y_val = tf.squeeze(y_val)
y = tf.one_hot(y, depth=10)     # (50000, 10)
y_val = tf.one_hot(y_val, depth=10)
print("datasets:", x.shape, y.shape, x.max(), x.min())

train_db = tf.data.Dataset.from_tensor_slices((x,y))
train_db = train_db.map(preprocess).shuffle(10000).batch(batchsz)
test_db = tf.data.Dataset.from_tensor_slices((x_val,y_val))
test_db = test_db.map(preprocess).batch(batchsz)

class MyDense(layers.Layer):
    def __init__(self, inp_dim, outp_dim):
        super(MyDense, self).__init__()
        self.kernel = self.add_variable('w', [inp_dim, outp_dim])
        # self.bias = self.add_variable('b', [outp_dim])
```

```
    def call(self, inputs, training=None):
        x = inputs  @ self.kernel
        return x

class MyNetwork(keras.Model):
    def __init__(self):
        super(MyNetwork, self).__init__()

        self.fc1 = MyDense(32*32*3, 256)
        self.fc2 = MyDense(256, 128)
        self.fc3 = MyDense(128, 64)
        self.fc4 = MyDense(64, 32)
        self.fc5 = MyDense(32, 10)

    def call(self, inputs, training=None):
        # inputs: [b, 32, 32, 3]
        x = tf.reshape(inputs, [-1,32*32*3])
        # [b, 32*32*3] => [b, 256]
        x = self.fc1(x)
        x = tf.nn.relu(x)
        # [b, 256] => [b, 128]
        x = self.fc2(x)
        x = tf.nn.relu(x)
        # [b, 128] => [b, 64]
        x = self.fc3(x)
        x = tf.nn.relu(x)
        # [b, 64] => [b, 32]
        x = self.fc4(x)
        x = tf.nn.relu(x)
        # [b, 32] => [b, 10]
        x = self.fc5(x)

        return x

network = MyNetwork()
network.compile(optimizer=optimizers.Adam(lr=1e-3),
                loss=tf.losses.CategoricalCrossentropy(from_logits=Tr
ue),
                metrics=['accuracy'])
network.fit(train_db, epochs=15, validation_data=test_db, validation_
freq=1)

network.evaluate(test_db)
```

```
network.save_weights('ckpt/weights.ckpt')
del network
print('save to ckpt/weights.ckpt')
network = MyNetwork()
network.compile(optimizer=optimizers.Adam(lr=1e-3),
                loss=tf.losses.CategoricalCrossentropy(from_logits=Tr
ue),
                metrics=['accuracy'])
network.load_weights('ckpt/weights.ckpt')
print("loaded weights from file.")
network.evaluate(test_db)
```