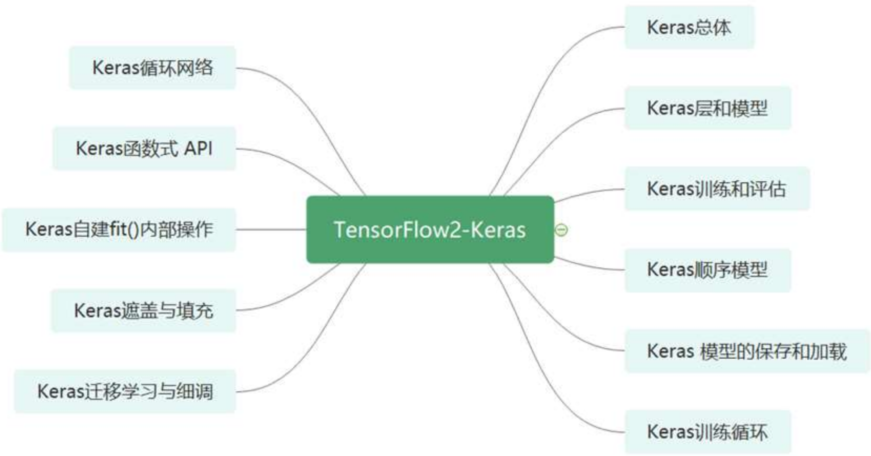
TensorFlow2-Keras

Keras

训练和评估

导学



Keras总体-1

- Keras层和模型
  - https://tensorflow.google.cn/guide/keras/custom_layers_and_models
- Keras训练和评估
  - https://tensorflow.google.cn/guide/keras/train_and_evaluate
- Keras顺序模型
  - https://tensorflow.google.cn/guide/keras/sequential_model
- Keras 模型的保存和加载
  - https://tensorflow.google.cn/guide/keras/save_and_serialize
- Keras函数式 API
  - https://tensorflow.google.cn/guide/keras/functional
- Keras训练循环
  - https://tensorflow.google.cn/guide/keras/writing_a_training_loop_from_scratch

Keras总体-2

- Keras自建fit()内部操作
  - https://tensorflow.google.cn/guide/keras/customizing_what_happens_in_fit
- Keras循环网络
  - https://tensorflow.google.cn/guide/keras/rnn
- Keras遮盖与填充
  - https://tensorflow.google.cn/guide/keras/masking_and_padding
- Keras迁移学习与细调
  - https://tensorflow.google.cn/guide/keras/transfer_learning

训练和评估初步

keras

典型的端到端工作流

- 训练流程
- 根据从原始训练数据生成的预留集进行验证
- 根据测试数据进行评估

定义网络模型

• 3层全连接网络

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
inputs = keras.Input(shape=(784,), name="digits")
x = layers.Dense(64, activation="relu", name="dense_1")(inputs)
x = layers.Dense(64, activation="relu", name="dense_2")(x)
outputs = layers.Dense(10, activation="softmax", name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

数据准备：mnist数据集

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Preprocess the data (these are NumPy arrays)
x_train = x_train.reshape(60000, 784).astype("float32") / 255
x_test = x_test.reshape(10000, 784).astype("float32") / 255

y_train = y_train.astype("float32")
y_test = y_test.astype("float32")

# Reserve 10,000 samples for validation
x_val = x_train[-10000:]
y_val = y_train[-10000:]
x_train = x_train[:-10000]
y_train = y_train[:-10000]
```

每一行是一张图

验证集

取出验证集

去掉验证集的训练集

0-49999, 50000-59999, 60000-69999

compile() 方法：优化器和指定损失、指标

```
model.compile(
    optimizer=keras.optimizers.RMSprop(),  # Optimizer
    # Loss function to minimize
    loss=keras.losses.SparseCategoricalCrossentropy(),
    # List of metrics to monitor
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
)
```

→ 优化器

→ 损失

→ 指标

↕ 等价

```
model.compile(
    optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["sparse_categorical_accuracy"],
)
```

fit() 方法: 训练模型

每批多少个

- fit()通过将数据切成大小为"batch_size"的"批次"， 然后将整个数据集重复迭代给定数量的"周期"来训练模型。
- 返回的"历史"对象保留训练期间的损失值和指标值记录

```
print("Fit model on training data")
history = model.fit(
    x_train,
    y_train,
    batch_size=64,
    epochs=2,
    # We pass some validation for
    # monitoring validation loss and metrics
    # at the end of each epoch
    validation_data=(x_val, y_val),
)
```

```
history.history
```

evaluate()： 测试数据评估模型

```
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(x_test, y_test, batch_size=128)
print("test loss, test acc:", results)

# Generate predictions (probabilities -- the output of the last layer)
# on new data using `predict`
print("Generate predictions for 3 samples")
predictions = model.predict(x_test[:3])
print("predictions shape:", predictions.shape)
```

拿出测试集前3个进行预测

训练和评估进阶

keras

定义神经网络

```
def get_uncompiled_model():
    inputs = keras.Input(shape=(784,), name="digits")
    x = layers.Dense(64, activation="relu", name="dense_1")(inputs)
    x = layers.Dense(64, activation="relu", name="dense_2")(x)
    outputs = layers.Dense(10, activation="softmax", name="predictions")(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model


def get_compiled_model():
    model = get_uncompiled_model()
    model.compile(
        optimizer="rmsprop",
        loss="sparse_categorical_crossentropy",
        metrics=["sparse_categorical_accuracy"],
    )
    return model
```

compile() 方法：优化器和指定损失、指标

- Keras API 内置损失、指标、优化器

优化器：
- SGD()（有或没有动量）
- RMSprop()
- Adam()
- 等等

损失：
- MeanSquaredError()
- CategoricalCrossentropy()
- KLDivergence()
- CosineSimilarity()
- 等等

指标：
- Precision()
- Recall()
- AUC()
- 等等

自定义损失函数-方法1

- 方式一：创建一个接受输入 y_true 和 y_pred 的函数。
- 下面的示例显示了一个计算实际数据与预测值之间的均方误差的损失函数：

```
def custom_mean_squared_error(y_true, y_pred):
    return tf.math.reduce_mean(tf.square(y_true - y_pred))


model = get_uncompiled_model()
model.compile(optimizer=keras.optimizers.Adam(),
loss=custom_mean_squared_error())

# We need to one-hot encode the labels to use MSE
y_train_one_hot = tf.one_hot(y_train, depth=10)
model.fit(x_train, y_train_one_hot, batch_size=64, epochs=1)
```

独热编码

自定义损失-方法2

- 方式二：使用除 y_true 和 y_pred 之外的其他参数的损失函数，则可以将 **tf.keras.losses.Loss 类子类化**
  - **__init__(self)**：接受要在调用损失函数期间传递的参数
  - **call(self, y_true, y_pred)**：使用目标 (y_true) 和模型预测 (y_pred) 来计算模型的损失

方式二举例:

- 假设使用均方误差时，但在远离 0.5时会抑制预测值时计算损失。

```
class CustomMSE(keras.losses.Loss):
    def __init__(self, regularization_factor=0.1, name="custom_mse"):
        super().__init__(name=name)
        self.regularization_factor = regularization_factor

    def call(self, y_true, y_pred):
        mse = tf.math.reduce_mean(tf.square(y_true - y_pred))
        reg = tf.math.reduce_mean(tf.square(0.5 - y_pred))
        return mse + reg * self.regularization_factor


model = get_uncompiled_model()
model.compile(optimizer=keras.optimizers.Adam(), loss=CustomMSE())

y_train_one_hot = tf.one_hot(y_train, depth=10)
model.fit(x_train, y_train_one_hot, batch_size=64, epochs=1)
```

or

通过添加层计算损失

```
class ActivityRegularizationLayer(layers.Layer):
    def call(self, inputs):
        self.add_loss(tf.reduce_sum(inputs) * 0.1)
        return inputs  # Pass-through layer.
```

通过添加层计算损失

```
inputs = keras.Input(shape=(784,), name="digits")
x = layers.Dense(64, activation="relu", name="dense_1")(inputs)
# Insert activity regularization as a layer
x = ActivityRegularizationLayer()(x)

x = layers.Dense(64, activation="relu", name="dense_2")(x)
outputs = layers.Dense(10, name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer=keras.optimizers.RMSprop(learning_rate=1e-3),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
)

# The displayed loss will be much higher than before
# due to the regularization component.
model.fit(x_train, y_train, batch_size=64, epochs=1)
```

通过添加层计算指标

```
class MetricLoggingLayer(layers.Layer):
  def call(self, inputs):
    # The `aggregation` argument defines
    # how to aggregate the per-batch values
    # over each epoch:
    # in this case we simply average them.
    self.add_metric(
        keras.backend.std(inputs), name="std_of_activation", aggregation="mean"
    )
    return inputs  # Pass-through layer.
```

通过添加层计算指标

```
inputs = keras.Input(shape=(784,), name="digits")
x = layers.Dense(64, activation="relu", name="dense_1")(inputs)
# Insert std logging as a layer.
x = MetricLoggingLayer()(x)
x = layers.Dense(64, activation="relu", name="dense_2")(x)
outputs = layers.Dense(10, name="predictions")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer=keras.optimizers.RMSprop(learning_rate=1e-3),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
)
model.fit(x_train, y_train, batch_size=64, epochs=1)
```

数据集处理

tf.data

自动分离验证预留集

- 参数 **validation_split** 允许自动保留部分训练数据以供验证
- 参数值表示要保留用于验证的数据比例，因此应将其设置为大于 0 且小于 1 的数字。
- 例如，validation_split=0.2 表示“使用 20% 的数据进行验证”，而 validation_split=0.6 表示“使用 60% 的数据进行验证”。

```
model = get_compiled_model()
model.fit(x_train, y_train, batch_size=64, validation_split=0.2, epochs=1)
```

通过 tf.data 数据集进行训练和评估

- tf.data API是 TensorFlow 2.0 中的一组实用工具，用于以快速且可扩展的方式加载和预处理数据
- 可以将 Dataset 实例直接传递给 fit()、evaluate() 和 predict()
- 有关创建 Datasets 的完整指南，请参阅 tf.data 文档

通过 tf.data 数据集进行训练和评估

```
model = get_compiled_model()
# First, let's create a training Dataset instance.
# For the sake of our example, we'll use the same MNIST data as before.
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
# Shuffle and slice the dataset.
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(64)
# Now we get a test dataset.
test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test))
test_dataset = test_dataset.batch(64)
# Since the dataset already takes care of batching,
# we don't pass a `batch_size` argument.
model.fit(train_dataset, epochs=3)
# You can also evaluate or predict on a dataset.
print("Evaluate")
result = model.evaluate(test_dataset)
dict(zip(model.metrics_names, result))
```

steps_per_epoch 参数

- 利用steps_per_epoch 参数在数据集的特定数量批次上进行训练

```
model = get_compiled_model()

# Prepare the training dataset
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(64)

# Only use the 100 batches per epoch (that's 64 * 100 samples)
model.fit(train_dataset, epochs=3, steps_per_epoch=100)
```

使用验证数据集

- 在 fit() 中将 Dataset 实例作为 validation_data 参数传递：

```
model = get_compiled_model()

# Prepare the training dataset
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(64)

# Prepare the validation dataset
val_dataset = tf.data.Dataset.from_tensor_slices((x_val, y_val))
val_dataset = val_dataset.batch(64)

model.fit(train_dataset, epochs=1, validation_data=val_dataset)
```

支持的其他输入格式

- 建议使用：
  - NumPy， 前提是数据很小且适合装入内存
  - Dataset 对象，前提是大型数据集，且需要执行分布式训练
  - Sequence 对象，前提具有大型数据集，且需要执行很多无法在 TensorFlow 中完成的自定义 Python 端处理（例如，依赖外部库进行数据加载或预处理）
- 支持其他的数据格式：
  - Pandas，利用数据帧或通过产生批量数据和标签的 Python 生成器训练 Keras 模型
  - keras.utils.Sequence 对象

使用样本加权

- 类权重:
  - 通过将字典传递给 Model.fit() 的 class_weight 参数来进行设置。此字典会将类索引映射到应当用于属于此类的样本的权重
  - 例如，在您的数据中，如果类“0”表示类“1”的一半，则可以使用 Model.fit(..., class_weight={0: 1., 1: 0.5})

使用样本加权

```
import numpy as np
class_weight = {
    0: 1.0,
    1: 1.0,
    2: 1.0,
    3: 1.0,
    4: 1.0,
    # Set weight "2" for class "5",
    # making this class 2x more important
    5: 2.0,
    6: 1.0,
    7: 1.0,
    8: 1.0,
    9: 1.0,
}
print("Fit with class weight")
model = get_compiled_model()
model.fit(x_train, y_train, class_weight=class_weight,
batch_size=64, epochs=1)
```

使用类加权

- 如果不构建分类器，则可以使用“样本权重”
  - 通过 NumPy 数据进行训练时：将 sample_weight 参数传递给 Model.fit()
  - 通过 tf.data 或任何其他类型的迭代器进行训练时：产生 (input_batch, label_batch, sample_weight_batch) 元组

使用类加权

使用numpy示例：

```
sample_weight = np.ones(shape=(len(y_train),))
sample_weight[y_train == 5] = 2.0

print("Fit with sample weight")
model = get_compiled_model()
model.fit(x_train, y_train, sample_weight=sample_weight,
batch_size=64, epochs=1)
```

使用类加权

- Dataset 示例：

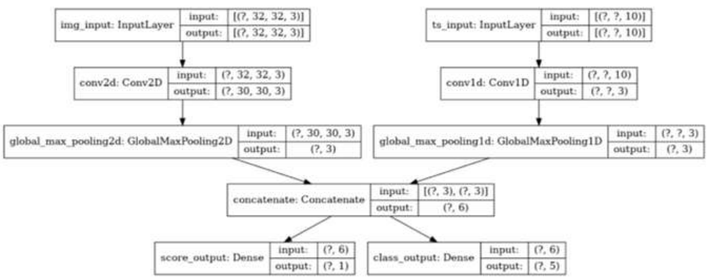
```
sample_weight = np.ones(shape=(len(y_train),))
sample_weight[y_train == 5] = 2.0

# Create a Dataset that includes sample weights
# (3rd element in the return tuple).
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train,
sample_weight))

# Shuffle and slice the dataset.
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(64)

model = get_compiled_model()
model.fit(train_dataset, epochs=1)
```

将数据传递到多输入、多输出模型

- 考虑以下模型：
  - 该模型具有形状为 (32, 32, 3) 的图像输入（即 (height, width, channels)）和形状为 (None, 10) 的时间序列输入（即 (timesteps, features)）。
  - 模型将具有根据这些输入的组合计算出的两个输出："得分"（形状为 (1,)）和在五个类上的概率分布（形状为 (5,)）。

将数据传递到多输入、多输出模型

```
image_input = keras.Input(shape=(32, 32, 3), name="img_input")
timeseries_input = keras.Input(shape=(None, 10), name="ts_input")

x1 = layers.Conv2D(3, 3)(image_input)
x1 = layers.GlobalMaxPooling2D()(x1)

x2 = layers.Conv1D(3, 3)(timeseries_input)
x2 = layers.GlobalMaxPooling1D()(x2)

x = layers.concatenate([x1, x2])

score_output = layers.Dense(1, name="score_output")(x)
class_output = layers.Dense(5, activation="softmax", name="class_output")(x)

model = keras.Model(
    inputs=[image_input, timeseries_input], outputs=[score_output, class_output]
)
```



将数据传递到多输入、多输出模型

- 多个损失、指标和权重

```
model.compile(
    optimizer=keras.optimizers.RMSprop(1e-3),
    loss={
        "score_output": keras.losses.MeanSquaredError(),
        "class_output": keras.losses.CategoricalCrossentropy(),
    },
    metrics={
        "score_output": [
            keras.metrics.MeanAbsolutePercentageError(),
            keras.metrics.MeanAbsoluteError(),
        ],
        "class_output": [keras.metrics.CategoricalAccuracy()],
    },
    loss_weights={"score_output": 2.0, "class_output": 1.0},
)
```

回调

- Keras 中的回调是训练期间（某个周期开始时、某个批次结束时、某个周期结束时等）在不同时间点调用的对象，这些对象可用于实现以下行为：
  - 在训练期间的不同时间点进行验证（除了内置的按周期验证外）
  - 定期或在超过一定准确率阈值时为模型设置检查点
  - 当训练似乎停滞不前时，更改模型的学习率
  - 当训练似乎停滞不前时，对顶层进行微调
  - 在训练结束或超出特定性能阈值时发送电子邮件或即时消息通知

回调

```
model = get_compiled_model()
callbacks = [
    keras.callbacks.EarlyStopping(
        # Stop training when `val_loss` is no longer improving
        monitor="val_loss",
        # "no longer improving" being defined as "no better than 1e-2 less"
        min_delta=1e-2,
        # "no longer improving" being further defined as "for at least 2 epochs"
        patience=2,
        verbose=1,
    )
]
model.fit(
    x_train,
    y_train,
    epochs=20,
    batch_size=64,
    callbacks=callbacks,
    validation_split=0.2,
)
```

回调

- 提供多个内置回调
  - ModelCheckpoint：定期保存模型
  - EarlyStopping：当训练不再改善验证指标时，停止训练
  - TensorBoard：定期编写可在 TensorBoard 中可视化的模型日志（更多详细信息，请参阅“可视化”部分）
  - CSVLogger：将损失和指标数据流式传输到 CSV 文件
- 编写您自己的回调
  - 可以通过扩展基类 keras.callbacks.Callback 来创建自定义回调

检查点回调

```
model = get_compiled_model()

callbacks = [
    keras.callbacks.ModelCheckpoint(
        # Path where to save the model
        # The two parameters below mean that we will overwrite
        # the current checkpoint if and only if
        # the `val_loss` score has improved.
        # The saved model name will include the current epoch.
        filepath="mymodel_{epoch}",
        save_best_only=True,  # Only save a model if `val_loss` has improved.
        monitor="val_loss",
        verbose=1,
    )
]
model.fit(
    x_train, y_train, epochs=2, batch_size=64, callbacks=callbacks, validation_split=0.2
)
```

检查点回调

```python
import os
# Prepare a directory to store all the checkpoints.
checkpoint_dir = "./ckpt"
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)
def make_or_restore_model():
    # Either restore the latest model, or create a fresh one
    # if there is no checkpoint available.
    checkpoints = [checkpoint_dir + "/" + name for name in
os.listdir(checkpoint_dir)]
    if checkpoints:
        latest_checkpoint = max(checkpoints, key=os.path.getctime)
        print("Restoring from", latest_checkpoint)
        return keras.models.load_model(latest_checkpoint)
    print("Creating a new model")
    return get_compiled_model()
model = make_or_restore_model()
callbacks = [
    # This callback saves a SavedModel every 100 batches.
    # We include the training loss in the saved model name.
    keras.callbacks.ModelCheckpoint(
        filepath=checkpoint_dir + "/ckpt-loss={loss:.2f}", save_freq=100
    )
]
model.fit(x_train, y_train, epochs=1, callbacks=callbacks)
```

使用 TensorBoard 回调

```python
keras.callbacks.TensorBoard(
    log_dir="/full_path_to_your_logs",
    histogram_freq=0,  # How often to log histogram visualizations
    embeddings_freq=0,  # How often to log embedding visualizations
    update_freq="epoch",
)  # How often to write logs (default: once per epoch)
```

使用学习率时间表

- 训练深度学习模型的常见模式是随着训练的进行逐渐减少学习。这通常称为“学习率衰减”

- 学习衰减时间表可以是静态的（根据当前周期或当前批次索引提前确定），也可以是动态的（响应模型的当前行为，尤其是验证损失）

使用学习率时间表

- 将时间表对象作为优化器中的 learning_rate 参数传递使用静态学习率衰减时间表

```
initial_learning_rate = 0.1
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=100000, decay_rate=0.96,
staircase=True
)

optimizer = keras.optimizers.RMSprop(learning_rate=lr_schedule)
```

参考书

- 英文版:
  - Francois Chollet, Deep Learning with Python, Manning press, November 2017.
- 中文版:
  - [美] 弗朗索瓦·肖莱（Francois Chollet） 著，Python深度学习，人民邮电出版社，2018年8月.

- 智能硬件TensorFlow实践，清华大学出版社，2017.

谢谢指正！