访问并控制模型内部过程，只要想在训练过程当中进行调整，就可以使用回调函数

使用 model.fit()在一个大型数据集上启动数十轮的训练，有点类似于扔一架纸飞机，一开始给它一点推力，之后你便再也无法控制其飞行轨迹或着陆点。如果想要避免不好的结果（并避免浪费纸飞机），更聪明的做法是不用纸飞机，而是用一架无人机，它可以感知其环境，将数据发回给操纵者，并且能够基于当前状态自主航行，可以自我反省并动态地采取行动

训练模型时，很多事情一开始都无法预测。尤其是你不知道需要多少轮才能得到最佳验证损失。前面所有例子都采用这样一种策略：训练足够多的轮次，这时模型已经开始过拟合，根据这第一次运行来确定训练所需要的正确轮数，然后使用这个最佳轮数从头开始再启动一次新的训练。当然，这种方法很浪费。

处理这个问题的更好方法是，当观测到验证损失不再改善时就停止训练。这可以使用 Keras 回调函数来实现。**回调函数**（callback）是在调用 `fit` 时传入模型的一个对象（即实现特定方法的类实例），它在训练过程中的不同时间点都会被模型调用。它可以访问关于模型状态与性能的所有可用数据，还可以采取行动：中断训练、保存模型、加载一组不同的权重或改变模型的状态。

回调函数的一些用法示例如下所示。

- **模型检查点**（model checkpointing）：在训练过程中的不同时间点保存模型的当前权重。
- **提前终止**（early stopping）：如果验证损失不再改善，则中断训练（当然，同时保存在训练过程中得到的最佳模型）。
- **在训练过程中动态调节某些参数值**：比如优化器的学习率。
- **在训练过程中记录训练指标和验证指标，或将模型学到的表示可视化（这些表示也在不断更新）**：你熟悉的 Keras 进度条就是一个回调函数！

回调函数举例

$$(tensorflow.keras.callbacks)$$

$$ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau, CSVLogger$$

In [1]:

```
import tensorflow as tf
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [5]:

```
from tensorflow.keras import Input, layers, Model, callbacks
```

In [4]:

```
import numpy as np
```

如果监控的目标指标在设定的轮数内不再改善，可以用 EarlyStopping 回调函数来中断，训练。比如，这个回调函数可以在刚开始过拟合的时候就中断训练，从而避免用更少的轮次重新训练模型。这个回调函数通常与 ModelCheckpoint 结合使用，后者可以在训练过程中持续不断地保存模型（你也可以选择只保存目前的最佳模型，即一轮结束后具有最佳性能的模型）。

In [6]:

```
callabacks_list = [
    callbacks.EarlyStopping(
        monitor='acc',
        patience=1,
    ),
    callbacks.ModelCheckpoint(
        filepath='my_model.h5',
        monitor='val_loss',
        save_best_only=True,
    )
]
```

In [7]:

```
# early stopping 监控模型的验证精度acc，如果精度在多于一轮的时间（即连续两轮）内不再改善，中断训练
# model checkpoint 在每轮过后保存当前权重，模型保存在my_model中，监控val_loss验证损失
# 如果 val_loss 没有改善，那么不需要覆盖模型文件。这就可以始终保存在训练过程中见到的最佳模型
```

In [ ]:

```
model.compile(
    optimizer='rmsprop',
    loss='binary_crossentropy',
    metrics=['acc']   # 监控精度，所以它应该是模型指标的一部分
)
```

In [ ]:

```
model.fit(x, y,
     epochs=10,
     batch_size=32,
     callbacks=callbacks_list,
     validation_data=(x_val, y_val)
          # 由于回调函数要监控验证损失和验证精度，所以在调用 fit 时需要传入 validation_data（验证
         )
```

如果验证损失不再改善，你可以使用这个回调函数来降低学习率。在训练过程中如果出现了损失平台（loss plateau），那么增大或减小学习率都是跳出局部最小值的有效策略。

In [ ]:

```
callabacks_list = [
    callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.1,
        patience=10
    )
]
```

In [8]:

```
# reduce lr on plateau 监控模型的验证损失
# 如果验证损失在 10 轮内都没有改善，那么就触发这个回调函数(在第11轮触发)
# 触发时将学习率除以 10
```

In [ ]:

```
model.fit(x, y,
     epochs=10,
     batch_size=32,
     callbacks=callbacks_list,
     validation_data=(x_val, y_val)
          # 由于回调函数要监控验证损失和验证精度，所以在调用 fit 时需要传入 validation_data (验证
         )
```

自定义损失函数

如果你需要在训练过程中采取特定行动，而这项行动又没有包含在内置回调函数中，那么可以编写你自己的回调函数。回调函数的实现方式是创建 keras.callbacks.Callback 类的子类。然后你可以实现下面这些方法（从名称中即可看出这些方法的作用），它们分别在训练过程中的不同时间点被调用。

```
on_epoch_begin     <—— 在每轮开始时被调用
on_epoch_end       <—— 在每轮结束时被调用

on_batch_begin     <—— 在处理每个批量之前被调用
on_batch_end       <—— 在处理每个批量之后被调用

on_train_begin     <—— 在训练开始时被调用
on_train_end       <—— 在训练结束时被调用
```

这些方法被调用时都有一个 logs 参数，这个参数是一个字典，里面包含前一个批量、前一个轮次或前一次训练的信息，即训练指标和验证指标等。此外，回调函数还可以访问下列属性。

- self.model：调用回调函数的模型实例。
- self.validation_data：传入 fit 作为验证数据的值。

，它可以在每轮结束后将模型每层的激活保存到硬盘（格式为 Numpy 数组），这个激活是对验证集的第一个样本计算得到的

In [9]:

```
class ActivationLogger(tf.keras.callbacks.Callback):
    def set_model(self, model):
        self.model = model
        layer_output = [layer.output for layer in model.layers]
        self.activations_model = tf.keras.models(model.input, layer_output)
    def on_epoch_end(self, epoch, logs=None):
        if self.validation_data is None:
            raise RuntimeError('Requires validation_data.')
        validation_sample = self.validation_data[0][0:1]
        activations = self.activations_model.predict(validation_sample)
        f = open('activations_at_epoch_' + str(epoch) + '.npz', 'w')
        np.savez(f, activations)
        f.close()
```

In [ ]: