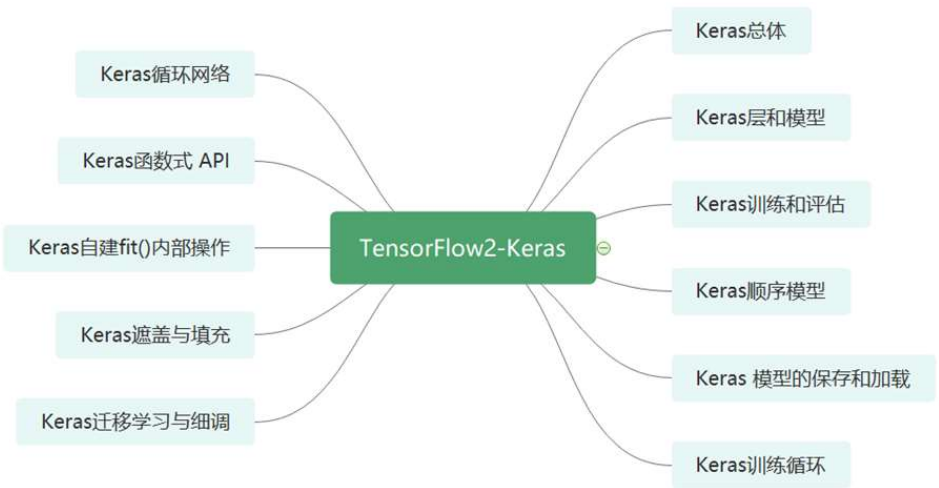
TensorFlow2-Keras

层和模型

Layer\Model

导学整体



Keras层和模型

- Keras层的实现

- Keras模型的实现

Keras层(Layer)的实现

Keras

Layer 类：权重和部分计算的组合

```
import tensorflow as tf
from tensorflow import keras
```

- Keras 的一个中心抽象是 Layer 类。
  - layer层封装了状态（层的“权重”）和从输入到输出的转换（“调用”，即层的前向传递）。
- 下面是一个密集连接的层。
  - 全连接层具有一个状态：变量 w 和 b。

全连接层Layer

- 一个密集连接的层，全连接层具有一个状态：变量 w 和 b。

```
class Linear(keras.layers.Layer):
    def __init__(self, units=32, input_dim=32):
        super(Linear, self).__init__()
        w_init = tf.random_normal_initializer()
        self.w = tf.Variable(
            initial_value=w_init(shape=(input_dim, units), dtype="float32"),
            trainable=True,
        )
        b_init = tf.zeros_initializer()
        self.b = tf.Variable(
            initial_value=b_init(shape=(units,), dtype="float32"), trainable=True
        )

    def call(self, inputs):
        return tf.matmul(inputs, self.w) + self.b
```

2×2 2×4 1×4

Layer 类：权重和部分计算的组合

units input_dim

```
x = tf.ones((2, 2))
linear_layer = Linear(4, 2)
y = linear_layer(x)
print(y)
```

输入2维

输出4维

```
tf.Tensor( [[ 0.02562864 -0.09071901 -0.13720123 0.0189665 ]
[ 0.02562864 -0.09071901 -0.13720123 0.0189665 ]], shape=(2, 4),
dtype=float32)
```

使用add_weight()方法，添加权重

效果同上

```
class Linear(keras.layers.Layer):
    def __init__(self, units=32, input_dim=32):
        super(Linear, self).__init__()
        self.w = self.add_weight(
            shape=(input_dim, units), initializer="random_normal", trainable=True
        )
        self.b = self.add_weight(shape=(units,), initializer="zeros", trainable=True)

    def call(self, inputs):
        return tf.matmul(inputs, self.w) + self.b


x = tf.ones((2, 2))
linear_layer = Linear(4, 2)
y = linear_layer(x)
print(y)
```

add_weight()添加权重

```
tf.Tensor( [[-0.05742075 -0.05801918 -0.06137164 -0.06648195]
[-0.05742075 -0.05801918 -0.06137164 -0.06648195]], shape=(2, 4), dtype=float32)
```

可训练与不可训练权重

除了可训练权重外，您还可以向层添加不可训练权重。对层进行训练时，不必在反向传播期间考虑此类权重。

```
class ComputeSum(keras.layers.Layer):
  def __init__(self, input_dim):
    super(ComputeSum, self).__init__()
    self.total = tf.Variable(initial_value=tf.zeros((input_dim,)), trainable=False)

  def call(self, inputs):
    self.total.assign_add(tf.reduce_sum(inputs, axis=0))
    return self.total

x = tf.ones((2, 2))
my_sum = ComputeSum(2)
y = my_sum(x)
print(y.numpy())
y = my_sum(x)
print(y.numpy())
```

trainable=False指定不可训练

```
x=[[1. 1.],
   [1. 1.]]

total=[2,2]

total=[4,4]
```

可训练与不可训练权重

除了可训练权重外，您还可以向层添加不可训练权重。
对层进行训练时，不必在反向传播期间考虑此类权重。

```
print("weights:", len(my_sum.weights))
print("non-trainable weights:", len(my_sum.non_trainable_weights))

# It's not included in the trainable weights:
print("trainable_weights:", my_sum.trainable_weights)
```

打印权重的个数
打印不可训练权重的个数

打印可训练权重

```
weights: 1
non-trainable weights: 1
trainable_weights: []
```

将权重创建推迟到得知输入的形状之后

- 在许多情况下，可能事先不知道输入的大小，并希望在得知该值时（对层进行实例化后的某个时间）再延迟创建权重。
- 在 Keras API 中，可以在层的 build(self, inputs_shape) 方法中创建层的权重。如下所示：

```
class Linear(keras.layers.Layer):
    def __init__(self, units=32):
        super(Linear, self).__init__()
        self.units = units

    def build(self, input_shape):
        self.w = self.add_weight(shape=(input_shape[-1], self.units), initializer="random_normal", trainable=True)
        self.b = self.add_weight(shape=(self.units,), initializer="random_normal", trainable=True)

    def call(self, inputs):
        return tf.matmul(inputs, self.w) + self.b
```

层可递归组合

- 如果将一个层实例分配为另一个层的特性，则外部层将开始跟踪内部层的权重。
- 可以在 __init__() 方法中创建此类子层（由于子层通常具有构建方法，它们将与外部层同时构建）。

```
class MLPBlock(keras.layers.Layer):
    def __init__(self):
        super(MLPBlock, self).__init__()
        self.linear_1 = Linear(32)
        self.linear_2 = Linear(32)
        self.linear_3 = Linear(1)

    def call(self, inputs):
        x = self.linear_1(inputs)
        x = tf.nn.relu(x)
        x = self.linear_2(x)
        x = tf.nn.relu(x)
        return self.linear_3(x)
```

```
mlp = MLPBlock()
y = mlp(tf.ones(shape=(3, 64)))
print("weights:", len(mlp.weights))
print("trainable weights:", len(mlp.trainable_weights))
```

```
weights: 6
trainable weights: 6
```

三个线性层，每层有一个w和一个b

Keras模型的实现

Keras

Model 类

- 使用 Layer 类来定义内部计算块，使用 Model 类来定义外部模型
- Model 类具有与 Layer 相同的 API，但有如下区别：
- 它会公开内置训练、评估和预测循环（model.fit()、model.evaluate()、model.predict()）。
- 它会通过 model.layers 属性公开其内部层的列表。
- 它会公开保存和序列化 API（save()、save_weights()…）

Keras模型与层的概念讨论

- Layer 类对应于在文献中所称的“层”（如“卷积层”或“循环层”）或“块”（如“ResNet 块”或“Inception 块”）。
- Model 类对应于文献中所称的“模型”（如“深度学习模型”）或“网络”（如“深度神经网络”）。
- 因此，如果要确定“应该用 Layer 类还是 Model 类？”
  - 请问自己：是否需要在它上面调用 fit()？是否需要在它上面调用 save()？如果是，则使用 Model。
  - 如果不是（要么因为这个类只是更大系统中的一个块，要么因为需要自己编写训练和保存代码），则使用 Layer。

Model 类-定义resnet

```
class ResNet(tf.keras.Model):
    def __init__(self):
        super(ResNet, self).__init__()
        self.block_1 = ResNetBlock()
        self.block_2 = ResNetBlock()
        self.global_pool =layers.GlobalAveragePooling2D()
        self.classifier = Dense(num_classes)

    def call(self, inputs):
        x = self.block_1(inputs)
        x = self.block_2(x)
        x = self.global_pool(x)
        return self.classifier(x)

resnet = ResNet()
dataset = ...
resnet.fit(dataset, epochs=10)
resnet.save(filepath)
```

模型

层

模型有fit, save等接口

谢谢指正！