自定义层详解

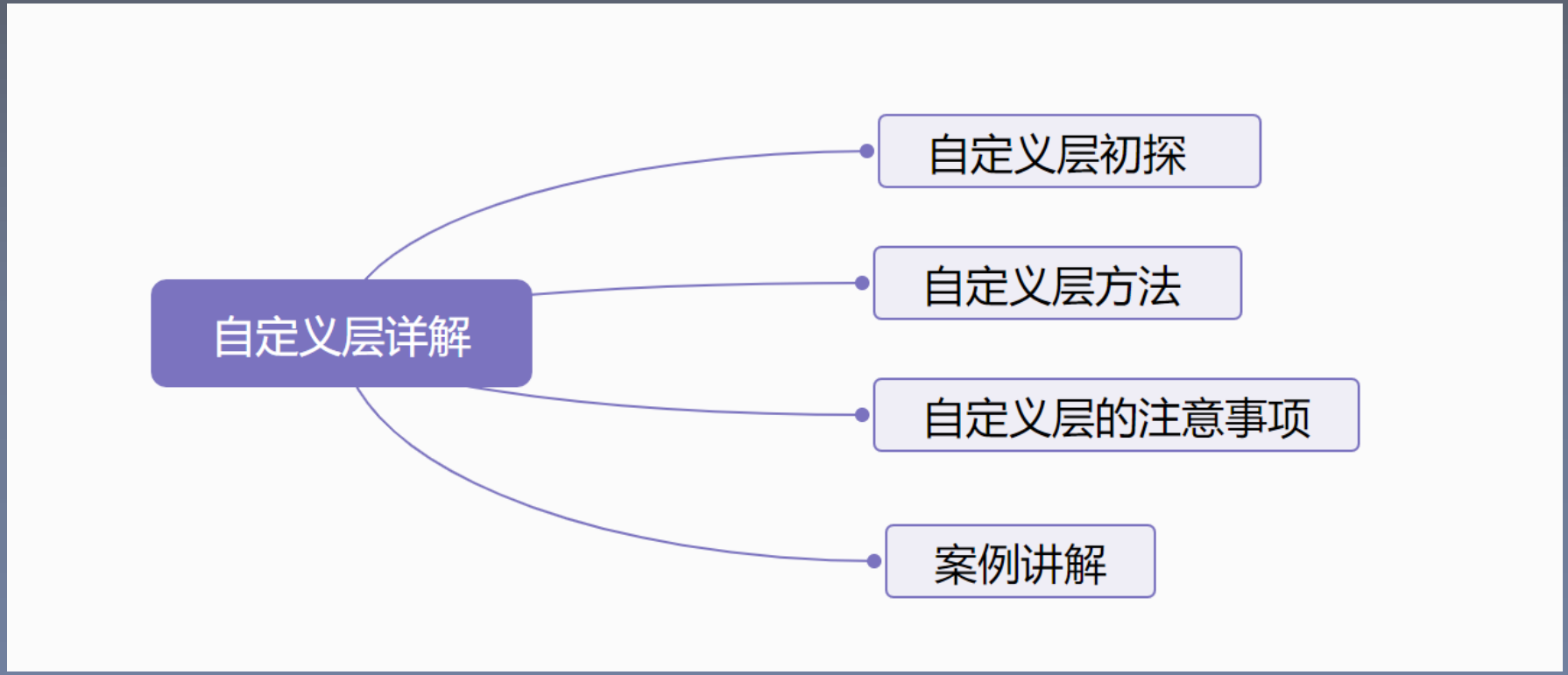
导师：GAUSS

深度之眼
deepshare.net

目录

知识树

Knowledge tree



深度之眼
deepshare.net

自定义层初探

自定义层

Custom layer

使用的主要数据结构是**Layer**

实现自定义层的最佳方法是扩展**tf.keras.layers.Layer类**并实现：

- **__init__** ：可以在其中进行所有与输入无关的初始化，定义相关的层
- **build**：知道输入张量的形状并可以进行其余的初始化
- **call**：在这里进行前向传播

注意：不一定需要在build中创建变量时，也可以在__init__中创建它们。

自定义层

tf.keras.Model和tf.keras.layers.Layer区别和联系

tf.keras.Model和tf.keras.layers.Layer有什么区别和联系？

- 通过继承 tf.keras.Model 编写自己的**模型类**
- 通过继承 tf.keras.layers.Layer 编写自己的**层**
- tf.keras中的模型和层都是继承tf.Module实现的
- tf.keras.Model继承tf.keras.layers.Layer实现的

解释：

tf.Module： 定位为一个轻量级的状态容器，因为可以收集变量，所以这个类型可以用来建模，配合tf.GradientTape使用。

```
class Linear(tf.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
```

深度之眼
deepshare.net

自定义层方法

三种方法自定义层

详见：notebook**案例1**

方法1：最基础的方法

方法2：使用self.add_weight创建变量

方法3：build函数中创建变量

深度之眼
deepshare.net

自定义层的注意事项

自定义层的注意事项

深度之眼
deepshare.net

注意一

注意一：这个问题是因为我们没有在自定义网络层时重写**get_config**导致的！

那我们该怎么去实现该方法呢？

```
   1940           filtered_inbound_nodes.append(node_data)
   1941
-> 1942     layer_config = serialize_layer_fn(layer)
   1943     layer_config['name'] = layer.name
   1944     layer_config['inbound_nodes'] = filtered_inbound_nodes

d:\miniconda3\envs\python3\lib\site-packages\tensorflow_core\python\keras\utils\generic_utils.py in serialize_keras_object(ins
tance)
    138   if hasattr(instance, 'get_config'):
    139     return serialize_keras_class_and_config(instance.__class__.__name__,
--> 140                                             instance.get_config())
    141   if hasattr(instance, '__name__'):
    142     return instance.__name__

d:\miniconda3\envs\python3\lib\site-packages\tensorflow_core\python\keras\engine\base_layer.py in get_config(self)
    571     # or that `get_config` has been overridden:
    572     if len(extra_args) > 1 and hasattr(self.get_config, '_is_default'):
--> 573       raise NotImplementedError('Layers with arguments in `__init__` must '
    574                                 'override `get_config`.')
    575     # TODO(reedwm): Handle serializing self._dtype_policy.

NotImplementedError: Layers with arguments in `__init__` must override `get_config`.
```

自定义层的注意事项

注意一

解决方案：我们主要看传入__init__接口时有哪些配置参数，然后在get_config内一一的将它们转为字典键值并且返回使用，以Mylayer为例：

```
def get_config(self):
    config = super(Linear, self).get_config()
    config.update({'units': self.units})
    return config
```

get_config的作用：获取该层的参数配置，以便模型保存时使用

自定义层的注意事项

深度之眼
deepshare.net

注意二

注意二：若模型保存（model.save）报错如下图时，则可能是自定义层的biuld 中创建初始矩阵时，name属性没写，会导致model.save报错

```python
import tensorflow as tf
class MyDense(tf.keras.layers.Layer):
    def __init__(self, units=32, **kwargs):
        self.units = units
        super(MyDense, self).__init__(**kwargs)

    #build方法一般定义Layer需要被训练的参数。
    def build(self, input_shape):
        self.w = self.add_weight(shape=(input_shape[-1], self.units),
                                 initializer='random_normal',
                                 trainable=True,
                                 name='w')
        self.b = self.add_weight(shape=(self.units,),
                                 initializer='random_normal',
                                 trainable=True,
                                 name='b')
        super(MyDense,self).build(input_shape) # 相当于设置self.built = True

    #call方法一般定义正向传播运算逻辑，__call__方法调用了它。
    def call(self, inputs):
        return tf.matmul(inputs, self.w) + self.b

    #如果要让自定义的Layer通过Functional API 组合成模型时可以序列化，需要自定义get_config方法。
    def get_config(self):
        config = super(MyDense, self).get_config()
        config.update({'units': self.units})
        return config
```

```
d:\miniconda3\envs\python3\lib\site-packages\h5py\_hl\group.py in create_d
wds)
    137             dset = dataset.Dataset(dsid)
    138             if name is not None:
--> 139                 self[name] = dset
    140             return dset
    141

d:\miniconda3\envs\python3\lib\site-packages\h5py\_hl\group.py in __setite
    369
    370             if isinstance(obj, HLObject):
--> 371                 h5o.link(obj.id, self.id, name, lcpl=lcpl, lapl=se
    372
    373             elif isinstance(obj, SoftLink):

h5py\_objects.pyx in h5py._objects.with_phil.wrapper()

h5py\_objects.pyx in h5py._objects.with_phil.wrapper()

h5py\h5o.pyx in h5py.h5o.link()

RuntimeError: Unable to create link (name already exists)
```

自定义层的注意事项

注意三

注意三：当我们自定义网络层并且有效保存模型后，希望使用tf.keras.models.load_model进行模型加载时，可能会报如下错误：

```
    101       custom_objects=custom_objects,
--> 102       printable_module_name='layer')

d:\miniconda3\envs\python3\lib\site-packages\tensorflow_core\python\keras\utils\generic_utils.py in deserialize_keras_object(identifier, module_objects, custom_objects, printable_module_name)
    178     config = identifier
    179     (cls, cls_config) = class_and_config_for_serialized_keras_object(
--> 180         config, module_objects, custom_objects, printable_module_name)
    181
    182     if hasattr(cls, 'from_config'):

d:\miniconda3\envs\python3\lib\site-packages\tensorflow_core\python\keras\utils\generic_utils.py in class_and_config_for_serialized_keras_object(config, module_objects, custom_objects, printable_module_name)
    163     cls = module_objects.get(class_name)
    164     if cls is None:
--> 165       raise ValueError('Unknown ' + printable_module_name + ': ' + class_name)
    166   return (cls, config['config'])
    167

ValueError: Unknown layer: MyDense
```

自定义层的注意事项

注意三

解决方案：首先，建立一个字典，该字典的键是自定义网络层时设定该层的名字，其值为该自定义网络层的类名，该字典将用于加载模型时使用！

然后，在tf.keras.models.load_model内传入custom_objects告知如何解析重建自定义网络层，其方法如下：

```
_custom_objects = {
    "MyDense" : MyDense,
}
tf.keras.models.load_model("keras_model_tf_version.h5",custom_objects=
_custom_objects)
```

自定义层的注意事项

注意四：当我们自定义一个网络层其名字与默认的tf.keras网络层一样时，可能会报出一些奇怪的问题，其实是因为重名了。

自定义层的注意事项

注意五：我们在实现自定义网络层时，最好统一在初始化时传入可变参数**kwargs，这是因为在model推理时，有时我们需要对所有构成该模型的网络层进行统一的传参。

```python
import tensorflow as tf
class MyDense(tf.keras.layers.Layer):
    def __init__(self, units=32, **kwargs):
        self.units = units
        super(MyDense, self).__init__(**kwargs)

    #build方法一般定义Layer需要被训练的参数。
    def build(self, input_shape):
        self.w = self.add_weight(shape=(input_shape[-1], self.units),
                                 initializer='random_normal',
                                 trainable=True,
                                 name='w')
        self.b = self.add_weight(shape=(self.units,),
                                 initializer='random_normal',
                                 trainable=True,
                                 name='b')
        super(MyDense,self).build(input_shape) # 相当于设置self.built = True

    #call方法一般定义正向传播运算逻辑，__call__方法调用了它。
    def call(self, inputs):
        return tf.matmul(inputs, self.w) + self.b

    #如果要让自定义的Layer通过Functional API 组合成模型时可以序列化，需要自定义get_config方法。
    def get_config(self):
        config = super(MyDense, self).get_config()
        config.update({'units': self.units})
        return config
```

深度之眼
deepshare.net

案例讲解

本节小结

Summary

自定义层详解	自定义层初探	自定义层组成部分
	自定义层方法	三种方法自定义层
	自定义层的注意事项	**五大注意事项（遇到问题的时候继续增加）**
	案例讲解	

结语

我 说

GAUSS老师个人公众号，主要分享NLP、推荐、比赛实战相关知识！

深度之眼
deepshare.net

联系我们：

电话：18001992849

邮箱：service@deepshare.net

Q Q： 2677693114

公众号

客服微信