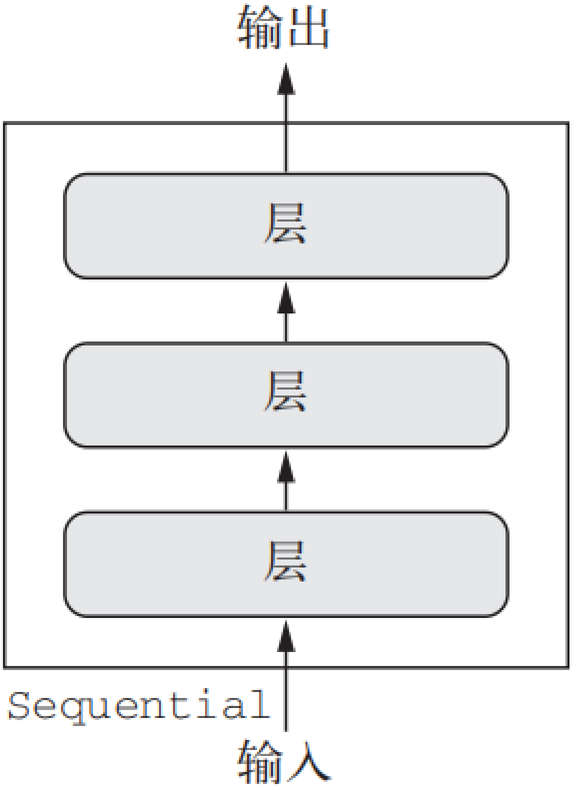
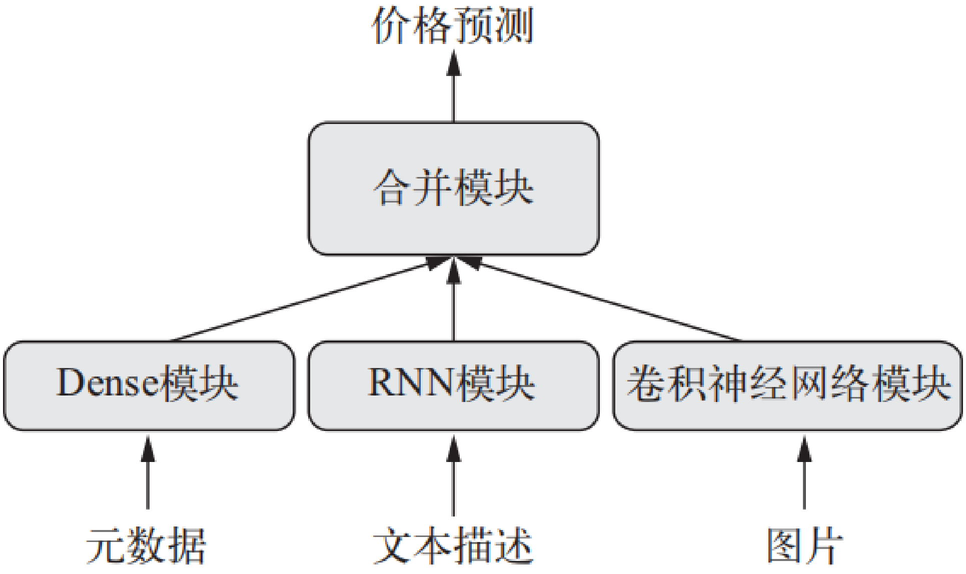
线性堆叠

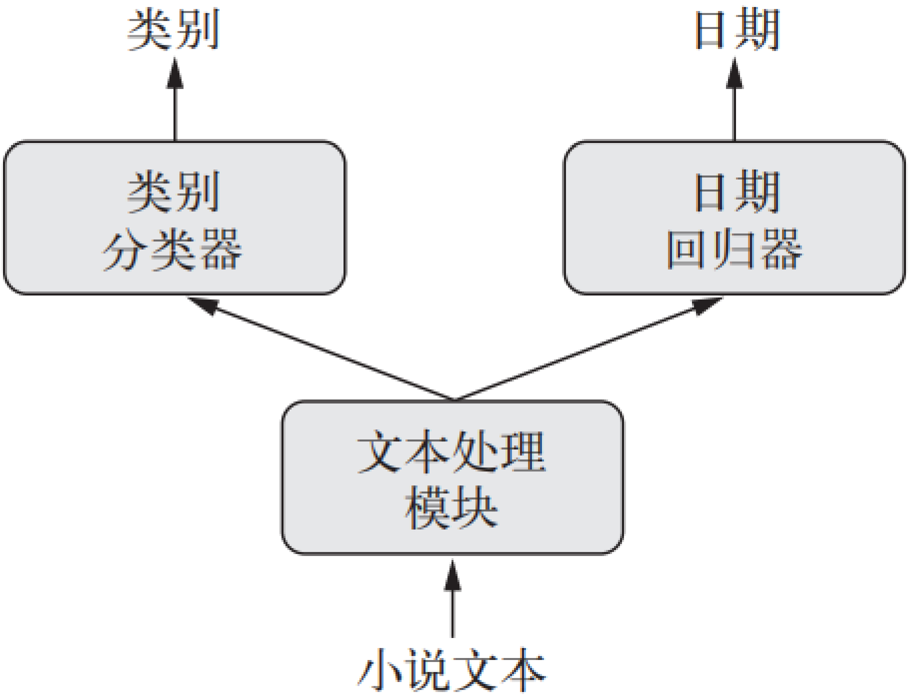


这种模式过于死板，有的网络需要多个独立的输入，有些网络则需要多个输出，而有些网络在层与层之间具有内部分支，这使得网络看起来像是层构成的图（graph），而不是层的线性堆叠

多输入模型



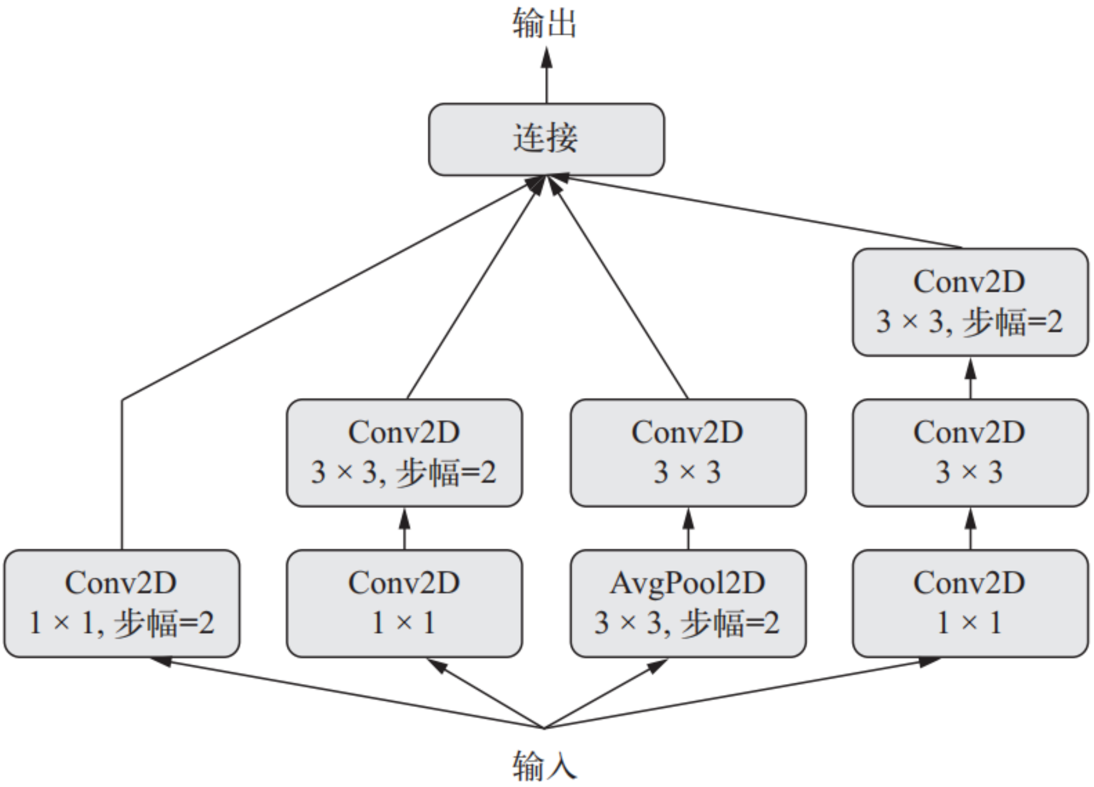
多输出模型



是可以训练多个独立的模型，然后对预测做加权平均。但这种方法可能不是最优的，因为模型提取的信息可能存在冗余或者信息之间存在着关联、一定的关系。更好的方法是使用一个可以同时查看所有可用的输入模态的模型，从而联合学习一个更加精确的数据模型

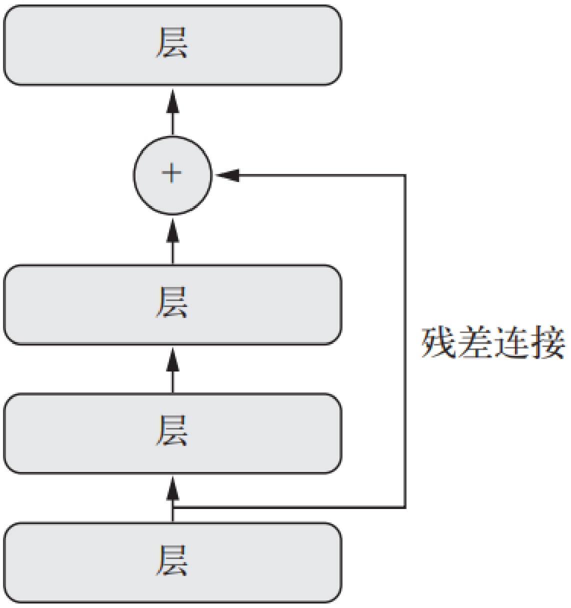
非线性，有向无环图网络结构

Inception，多个并行卷积分支合并，融合多个感受野视觉信息，使得特征更加丰富



Inception 模块：层组成的子图，具有多个并行卷积分支

残差连接，减少信息流失，合并前面输出，也可以提供短接，抑制梯度消失



残差连接：通过特征图相加将前面的信息重新注入下游数据

这些线性堆叠结构是无法实现的，必须使用函数式API(把层当作函数使用)

In [1]:

```
import tensorflow as tf
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [3]:

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

In [4]:

```
input_tensor = Input(shape=(32, ))
```

In [5]:

```
dense = layers.Dense(32, activation='relu')
```

In [6]:

```
output_tensor = dense(input_tensor)
```

In [7]:

```
output_tensor.shape
```

Out[7]:

```
TensorShape([None, 32])
```

In [8]:

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

In [10]:

```
seq_model = Sequential()
seq_model.add(layers.Dense(32, activation='relu', input_shape=(64, )))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))
```

In [11]:

```
seq_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056      
_________________________________________________________________
dense_3 (Dense)              (None, 10)                330       
=================================================================
Total params: 3,466
Trainable params: 3,466
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

```
input_tensor = Input(shape=(64, ))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)

model = Model(input_tensor, output_tensor)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_3 (InputLayer)         [(None, 64)]              0         
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_7 (Dense)              (None, 32)                1056      
_________________________________________________________________
dense_8 (Dense)              (None, 10)                330       
=================================================================
Total params: 3,466
Trainable params: 3,466
Non-trainable params: 0
_________________________________________________________________
```

后台检索从 input_tensor 到 output_tensor 所包含的每一层，并将这些层组合成一个类图的数据结构，即一个 Model。当然，这种方法有效的原因在于，output_tensor 是通过对input_tensor 进行多次变换得到的。

In [14]:

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

In [15]:

```
import numpy as np
```

In [16]:

```
x_train = np.random.random((1000,64))
y_train = np.random.random((1000,10))
```

In [17]:

```
model.fit(x_train, y_train, epochs=10, batch_size=128)
```

```
Train on 1000 samples
Epoch 1/10
1000/1000 [==============================] - 1s 676us/sample - loss: 12.2546
Epoch 2/10
1000/1000 [==============================] - 0s 28us/sample - loss: 13.9404
Epoch 3/10
1000/1000 [==============================] - 0s 31us/sample - loss: 15.9080
Epoch 4/10
1000/1000 [==============================] - 0s 31us/sample - loss: 17.5129
Epoch 5/10
1000/1000 [==============================] - 0s 30us/sample - loss: 18.2882
Epoch 6/10
1000/1000 [==============================] - 0s 21us/sample - loss: 19.2958
Epoch 7/10
1000/1000 [==============================] - 0s 19us/sample - loss: 20.8484
Epoch 8/10
1000/1000 [==============================] - 0s 23us/sample - loss: 22.6193
Epoch 9/10
1000/1000 [==============================] - 0s 27us/sample - loss: 24.7164
Epoch 10/10
1000/1000 [==============================] - 0s 21us/sample - loss: 26.8265
```

Out[17]:

```
<tensorflow.python.keras.callbacks.History at 0x1ee5d7688d0>
```

In [ ]: