# 2.建模方式

## 2.1.常用层

tf.keras.layers：tf.keras.layers是基于tf.nn的高度封装。

tf.keras.layers.LSTM(100)  
 tf.keras.layers.Dense(10)  
 tf.keras.layers.Dense(64, kernel\_regularizer=tf.keras.regularizers.l1(0.01))

tf.nn：底层的函数库，其他各种库可以说都是基于这个底层库来进行扩展的

tf.nn.softmax(b)  
 tf.keras.layers.Dense(64, activation=tf.nn.relu)

在大多数据情况下，可以使用TensorFlow封装的tf.keras.layers构建的一些层建模，Keras层是非常有用的。它包括Dense，Conv2D，LSTM，BatchNormalization，Dropout等。

## 2.2.三种建模方式

Sequential model (顺序模型)  
 Functional model (函数模型)  
 Subclassing model (子类化模型)

**Sequential model (顺序模型)**

**from** **tensorflow.keras** **import** layers

**import** **tensorflow** **as** **tf**

*# 方式一*

model = tf.keras.Sequential()

model.add(layers.Dense(64, activation='relu'))*#第一层*

model.add(layers.Dense(64, activation='relu'))*#第二层*

model.add(layers.Dense(10))*#第三层*

*# 。。。。。。。*

*# 方式二*

model = tf.keras.Sequential([

layers.Dense(64, activation='relu', input\_shape=(32,)),*#第一层*

layers.Dense(64, activation='relu'),*#第二层*

layers.Dense(10)*#第三层*

*#。。。。。*

])

*# 定义模型，分别定义优化器、损失函数、评估函数*

model.compile(optimizer=tf.keras.optimizers.Adam(0.01),

loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=**True**),

metrics=['accuracy'])

**import** **numpy** **as** **np**

*# 随机生成数据与labels*

data = np.random.random((1000, 32))

labels = np.random.random((1000, 10))

*# 训练模型*

model.fit(data, labels, epochs=10, batch\_size=32)

Train on 1000 samples

Epoch 1/10

1000/1000 [==============================] - 2s 2ms/sample - loss: 38.8803 - accuracy: 0.0810

...

Epoch 10/10

1000/1000 [==============================] - 1s 901us/sample - loss: 27653.8396 - accuracy: 0.1080

**Functional model (函数模型)**

函数式模型是一种创建模型的方法，该模型比tf.keras.Sequential更灵活。函数式模型可以处理具有非线性拓扑的模型，具有共享层的模型以及具有多个输入或输出的模型等等

深度学习模型通常是层的有向无环图（DAG）的主要思想。因此，函数式模型是一种构建层图的方法。

inputs = tf.keras.Input(shape=(32,))

x = layers.Dense(64, activation='relu')(inputs) *#第一层*

x = layers.Dense(64, activation='relu')(x) *#第二层*

predictions = layers.Dense(10)(x) *#第三层*

model = tf.keras.Model(inputs=inputs, outputs=predictions)

model.compile(optimizer=tf.keras.optimizers.RMSprop(0.001),

loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=**True**),

metrics=['accuracy'])

**import** **numpy** **as** **np**

data = np.random.random((1000, 32))

labels = np.random.random((1000, 10))

model.fit(data, labels, batch\_size=32, epochs=5)

Train on 1000 samples

Epoch 1/5

1000/1000 [==============================] - 1s 900us/sample - loss: 14.9791 - accuracy: 0.0840

...

Epoch 5/5

1000/1000 [==============================] - 0s 106us/sample - loss: 136.6048 - accuracy: 0.0890

*# 多个输入*

inputs1 = tf.keras.Input(shape=(32,)) *#输入1*

inputs2 = tf.keras.Input(shape=(32,)) *#输入2*

x1 = layers.Dense(64, activation='relu')(inputs1) *#第一层*

x2 = layers.Dense(64, activation='relu')(inputs2) *#第一层*

x = tf.concat([x1,x2],axis=-1) *#拼接*

x = layers.Dense(64, activation='relu')(x) *#第二层*

predictions = layers.Dense(10)(x) *#第三层*

model = tf.keras.Model(inputs=[inputs1,inputs2], outputs=predictions)

model.compile(optimizer=tf.keras.optimizers.RMSprop(0.001),

loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=**True**),

metrics=['accuracy'])

**import** **numpy** **as** **np**

data1 = np.random.random((1000, 32))

data2 = np.random.random((1000, 32))

labels = np.random.random((1000, 10))

*# 多个输入为元祖的形式*

model.fit((data1,data2), labels, batch\_size=32, epochs=5)

Train on 1000 samples

Epoch 1/5

1000/1000 [==============================] - 1s 1ms/sample - loss: 18.0165 - accuracy: 0.0920

...

Epoch 5/5

1000/1000 [==============================] - 0s 488us/sample - loss: 150.3576 - accuracy: 0.1060

**Subclassing model (子类化模型)**

通过子类化tf.keras.Model和定义自己的前向传播模型来构建完全可定制的模型。

**class** **MyModel**(tf.keras.Model):

**def** \_\_init\_\_(self, num\_classes=10):

super(MyModel, self).\_\_init\_\_(name='my\_model')

self.num\_classes = num\_classes

*# 定义自己需要的层*

self.dense\_1 = layers.Dense(32, activation='relu') *#*

self.dense\_2 = layers.Dense(num\_classes)

**def** call(self, inputs):

*#定义前向传播*

*# 使用在 (in `\_\_init\_\_`)定义的层*

x = self.dense\_1(inputs)

x = self.dense\_2(x)

**return** x

model = MyModel(num\_classes=10)

model.compile(optimizer=tf.keras.optimizers.RMSprop(0.001),

loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=**True**),

metrics=['accuracy'])

**import** **numpy** **as** **np**

data = np.random.random((1000, 32))

labels = np.random.random((1000, 10))

*# Trains for 5 epochs.*

model.fit(data, labels, batch\_size=32, epochs=5)

Train on 1000 samples

Epoch 1/5

1000/1000 [==============================] - 1s 853us/sample - loss: 12.9870 - accuracy: 0.1170

...

Epoch 5/5

1000/1000 [==============================] - 0s 274us/sample - loss: 21.4926 - accuracy: 0.1270

**三种构建模型的区别**

