13.4 Sequential模型

tf.keras

- TensorFlow的**高阶API**
- 快速**搭建**和**训练**神经网络模型
- 主要数据结构是**模型**(model)

Sequential模型

- 神经网络框架
- 只有**一组输入**和**一组输出**
- 各个**层**按照先后顺序**堆叠**

建立Sequential模型

```
model=tf.keras.Sequential()
```

```
model
<tensorflow.python.keras.engine.sequential.Sequential at 0x2cfb865f4e0>
```

添加层

全连接层，卷积层，池化层.....

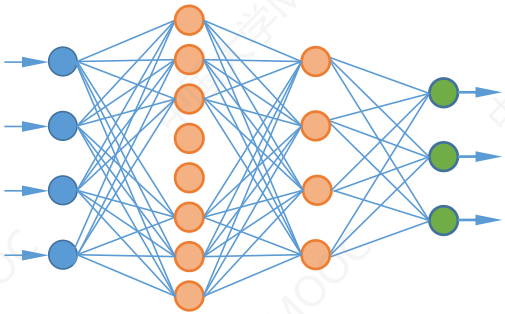
```
model.add (tf.keras.layers.... )
```

```
tf.keras.layers.Dense(
    inputs # 输入该网络层的数据
    activation # 激活函数  'relu', 'softmax','sigmoid','tanh'
    input_shape #输入数据的形状 )
```

查看摘要

```
model.summary()
```

多分类任务——三层神经网络




```
model=tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8,activation="relu",input_shape=(4,)))
model.add(tf.keras.layers.Dense(4,activation="relu"))
model.add(tf.keras.layers.Dense(3,activation="softmax"))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 8)                 40
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 15
=================================================================
Total params: 91
Trainable params: 91
Non-trainable params: 0
```

隐含层1

隐含层2

输出层

配置训练方法

model.compile(loss, optimizer, metrics)

损失函数

均方差损失函数	'mse'	tf.keras.losses.mean_squared_error() tf.keras.losses.MeanSquaredError()
多分类交叉熵损失函数	'categorical_crossentropy' 'sparse_categorical_crossentropy'	tf.keras.losses.categorical_crossentropy(from_logits=False) tf.keras.losses.sparse_categorical_crossentropy(from_logits=False) tf.keras.losses.CategoricalCrossentropy() tf.keras.losses.SparseCategoricalCrossentropy()
二分类交叉熵损失函数	'binary_crossentropy'	tf.keras.losses.binary_crossentropy() tf.keras.losses.BinaryCrossentropy()

配置训练方法

model.compile(loss, optimizer, metrics)

优化器

'sgd'	tf.keras.optimizers.SGD (lr, momentum)
'adagrad'	tf.keras.optimizers.Adagrad (lr)
'adadelta'	tf.keras.optimizers.Adadelta (lr)
'adam'	tf.keras.optimizers.Adam (lr, beta_1=0.9, beta_2=0.999)

tensorflow1.x tf.train.optimizer

tensorflow2.0+ tf.keras.optimizer

配置训练方法

- keras模型性能评估函数
- 自定义性能评估函数

```
model.compile(loss, optimizer, metrics)
```

标签值：数值 预测值：数值	'accuracy'	tf.keras.metrics.Accuracy()
二分类 标签值：数值 预测值：概率	'binary_accuracy '	tf.keras.metrics.binary_accuracy(threshold=0.5) tf.keras.metrics.BinaryAccuracy(threshold=0.5)
多分类 标签值：独热编码 预测值：独热编码	'categorical_accuracy'	tf.keras.metrics.categorical_accuracy() tf.keras.metrics.CategoricalAccuracy()
多分类 标签值：数值 预测值：独热编码	'sparse_categorical_accuracy'	tf.keras.metrics.sparse_categorical_accuracy() tf.keras.metrics.SparseCategoricalAccuracy()
多分类 前k种标签 标签值：数值 预测值：独热编码	'top_k_categorical_accuracy'	tf.keras.metrics.top_k_categorical_accuracy() tf.keras.metrics.TopKCategoricalAccuracy()
多分类 前k种标签 标签值：数值 预测值：独热编码	'sparse_top_k_categorical_accuracy'	tf.keras.metrics.sparse_top_k_categorical_accuracy() tf.keras.metrics.SparseTopKCategoricalAccuracy()

配置训练方法

model.compile(loss, optimizer, metrics)

AUC SUM Mean Precision	BinaryCrossentropy CategoricalCrossentropy SparseCategoricalCrossentropy Hinge CategoricalHinge	MeanAbsoluteError MeanAbsolutePercentageError MeanRelativeError MeanSquareError .....

https://tensorflow.google.cn/versions/r2.0/api_docs/python/tf/keras/metrics/

Iris

```
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])
```

IRIS

标签值： 1 ⟶ **[0, 1, 0]**

预测值：[0.1, 0.7, 0.1]

loss=tf.keras.losses.CategoricalCrossentropy(from_logits=False),
metrics=[tf.keras.metrics.CategoricalAccuracy()])

Mnist

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])
```

训练模型

```
model.fit (训练集的输入特征, 训练集的标签,
          batch_size=批量大小 ,
          epochs= 迭代次数,
        shuffle=是否每轮训练之前打乱数据,
          validation_data=(测试集的输入特征，测试集的标签),
          validation_split=从训练集划分多少比例给测试集,
          validation_freq = 测试频率
        verbose=日志显示形式
        )
```

```
verbose = 0   不在标准输出流输出
verbose = 1   输出进度条记录
verbose = 2   每个epoch输出一行记录
```

```
model.fit (x=None, y=None, batch_size=32 , epochs= 1,  shuffle=True,
          validation_data=None,  validation_split=0.0,  validation_freq = 1,
        verbose=1)
```

手写数字识别

```
model.fit(train_x, train_y, batch_size=32,epochs=5,validation_split=0.2) validation_freq = 1
Train on 48000 samples, validate on 12000 samples
Epoch 1/5
48000/48000 [==============================] - 4s 75us/sample - loss: 0.2813 - accuracy: 0.9198 - val_loss: 0.1512 - val_accuracy: 0.9562
Epoch 2/5
48000/48000 [==============================] - 3s 61us/sample - loss: 0.1239 - accuracy: 0.9632 - val_loss: 0.1179 - val_accuracy: 0.9639
Epoch 3/5
48000/48000 [==============================] - 3s 67us/sample - loss: 0.0880 - accuracy: 0.9740 - val_loss: 0.1008 - val_accuracy: 0.9688
Epoch 4/5
48000/48000 [==============================] - 3s 66us/sample - loss: 0.0648 - accuracy: 0.9806 - val_loss: 0.0950 - val_accuracy: 0.9713
Epoch 5/5
48000/48000 [==============================] - 3s 62us/sample - loss: 0.0510 - accuracy: 0.9849 - val_loss: 0.0915 - val_accuracy: 0.9726
<tensorflow.python.keras.callbacks.History at 0x13de91f7748>
```

History.history 损失函数 + **性能指标**
compile.metrics

```
In [20]: print(model.metrics_names)
['loss', 'accuracy']
```

■ 手写数字识别

```
model.fit(X_train, y_train, batch_size=32, epochs=5, validation_split=0.2)
```

validation_freq = 1

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/5
48000/48000 [==============================] - 5s 103us/sample - loss: 0.2846 - sparse_categorical_accuracy: 0.9183 - val_loss: 0.1582 - val
_sparse_categorical_accuracy: 0.9533
Epoch 2/5
48000/48000 [==============================] - 3s 72us/sample - loss: 0.1272 - sparse_categorical_accuracy: 0.9624 - val_loss: 0.1199 - val_
sparse_categorical_accuracy: 0.9646
Epoch 3/5
48000/48000 [==============================] - 3s 67us/sample - loss: 0.0860 - sparse_categorical_accuracy: 0.9743 - val_loss: 0.1020 - val_
sparse_categorical_accuracy: 0.9712
Epoch 4/5
48000/48000 [==============================] - 3s 66us/sample - loss: 0.0632 - sparse_categorical_accuracy: 0.9804 - val_loss: 0.1007 - val_
sparse_categorical_accuracy: 0.9701
Epoch 5/5
48000/48000 [==============================] - 3s 67us/sample - loss: 0.0486 - sparse_categorical_accuracy: 0.9851 - val_loss: 0.0901 - val_
sparse_categorical_accuracy: 0.9748

<tensorflow.python.keras.callbacks.History at 0x1cc5e7f5080>
```

```
model.metrics_names
```

model.compile(metrics=[ ])

```
['loss', 'sparse_categorical_accuracy']
```

■ 手写数字识别

```
history.history
```

```
{'loss': [0.2865329485485951,
  0.12679306019532183,
  0.0847434730535994,
  0.06226444375965123,
  0.046863987305822474],
 'sparse_categorical_accuracy': [0.91714585,
  0.9636667,
  0.9751875,
  0.9815625,
  0.9855625],
 'val_loss': [0.15371777984748283,
  0.1143894852194935,
  0.09567862997855991,
  0.08741304606727014,
  0.0886051772281838],
 'val_sparse_categorical_accuracy': [0.959,
  0.96675,
  0.97291666,
  0.9745,
  0.9755]}
```

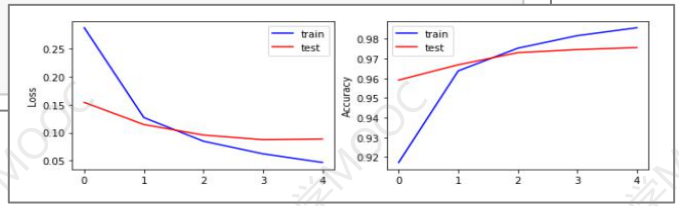
```
loss = history.history['loss']
val_loss = history.history['val_loss']
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
```

```
plt.figure(figsize=(10,3))

plt.subplot(121)
plt.plot(loss,color="blue",label="train")
plt.plot(val_loss,color="red",label="test")
plt.ylabel("Loss")
plt.legend()

plt.subplot(122)
plt.plot(acc,color="blue",label="train")
plt.plot(val_acc,color="red",label="test")
plt.ylabel("Accuracy")

plt.legend()
plt.show()
```



手写数字识别

```
history=model.fit(X_train, y_train, batch_size=32,epochs=5,validation_split=0.2)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/5
48000/48000 [==============================] - 5s 98us/sample - loss: 0.2865 - sparse_categorical_accuracy: 0.9171 - val_loss: 0.1537 - val_
sparse_categorical_accuracy: 0.9590
Epoch 2/5
48000/48000 [==============================] - 4s 84us/sample - loss: 0.1268 - sparse_categorical_accuracy: 0.9637 - val_loss: 0.1144 - val_
sparse_categorical_accuracy: 0.9668
Epoch 3/5
48000/48000 [==============================] - 4s 86us/sample - loss: 0.0847 - sparse_categorical_accuracy: 0.9752 - val_loss: 0.0957 - val_
sparse_categorical_accuracy: 0.9729
Epoch 4/5
48000/48000 [==============================] - 4s 91us/sample - loss: 0.0623 - sparse_categorical_accuracy: 0.9816 - val_loss: 0.0874 - val_
sparse_categorical_accuracy: 0.9745
Epoch 5/5
48000/48000 [==============================] - 4s 84us/sample - loss: 0.0469 - sparse_categorical_accuracy: 0.9856 - val_loss: 0.0886 - val_
sparse_categorical_accuracy: 0.9755
```

```
type(history)
```

```
tensorflow.python.keras.callbacks.History
```

手写数字识别

```
model.fit(train_x, train_y, batch_size=64,epochs=5) validation_split=0.0

Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 4s 60us/sample - loss: 0.3001 - accuracy: 0.9157
Epoch 2/5
60000/60000 [==============================] - 2s 34us/sample - loss: 0.1333 - accuracy: 0.9614
Epoch 3/5
60000/60000 [==============================] - 2s 31us/sample - loss: 0.0921 - accuracy: 0.9730
Epoch 4/5
60000/60000 [==============================] - 2s 34us/sample - loss: 0.0693 - accuracy: 0.9798
Epoch 5/5
60000/60000 [==============================] - 2s 32us/sample - loss: 0.0548 - accuracy: 0.9836

<tensorflow.python.keras.callbacks.History at 0x22d06a91080>
```

评估模型

model.evaluate（test_set_x，test_set_y ,batch_size, verbose）

- **Mnist**
  - 训练集：60000
  - 测试集：10000

```
In  [17]: model.evaluate(test_x, test_y, batch_size=32, verbose=2)
          10000/1 - 0s - loss: 0.0427 - accuracy: 0.9722
Out[17]: [0.08461135778911412, 0.9722]
```

使用模型

model.predict（x, batch_size, verbose）

https://tensorflow.google.cn/versions