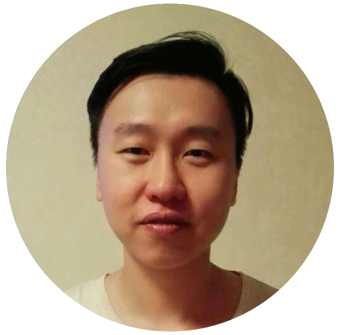
GDG Guangzhou

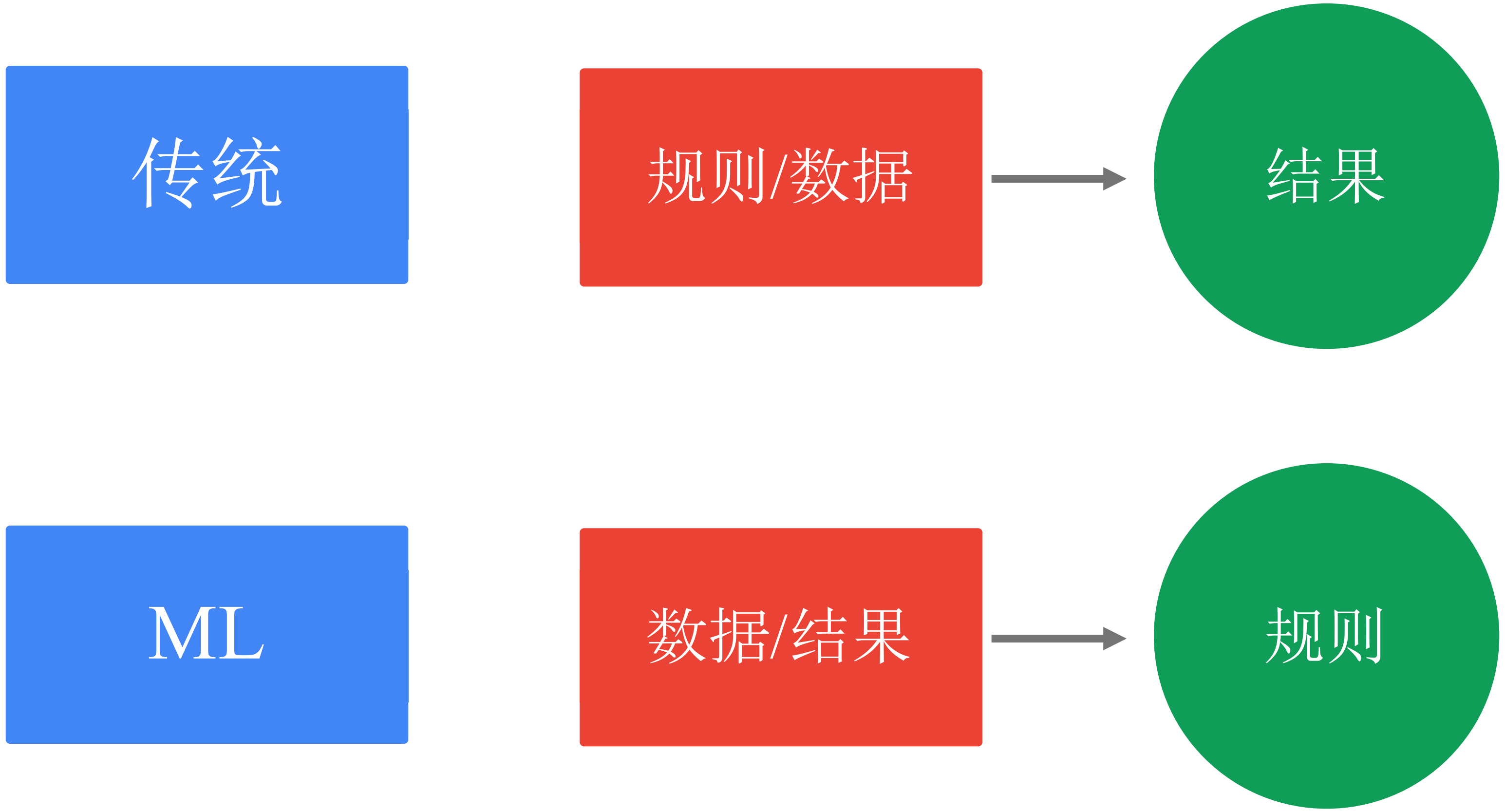
DevFest

TensorFlow 入门
- With high level API



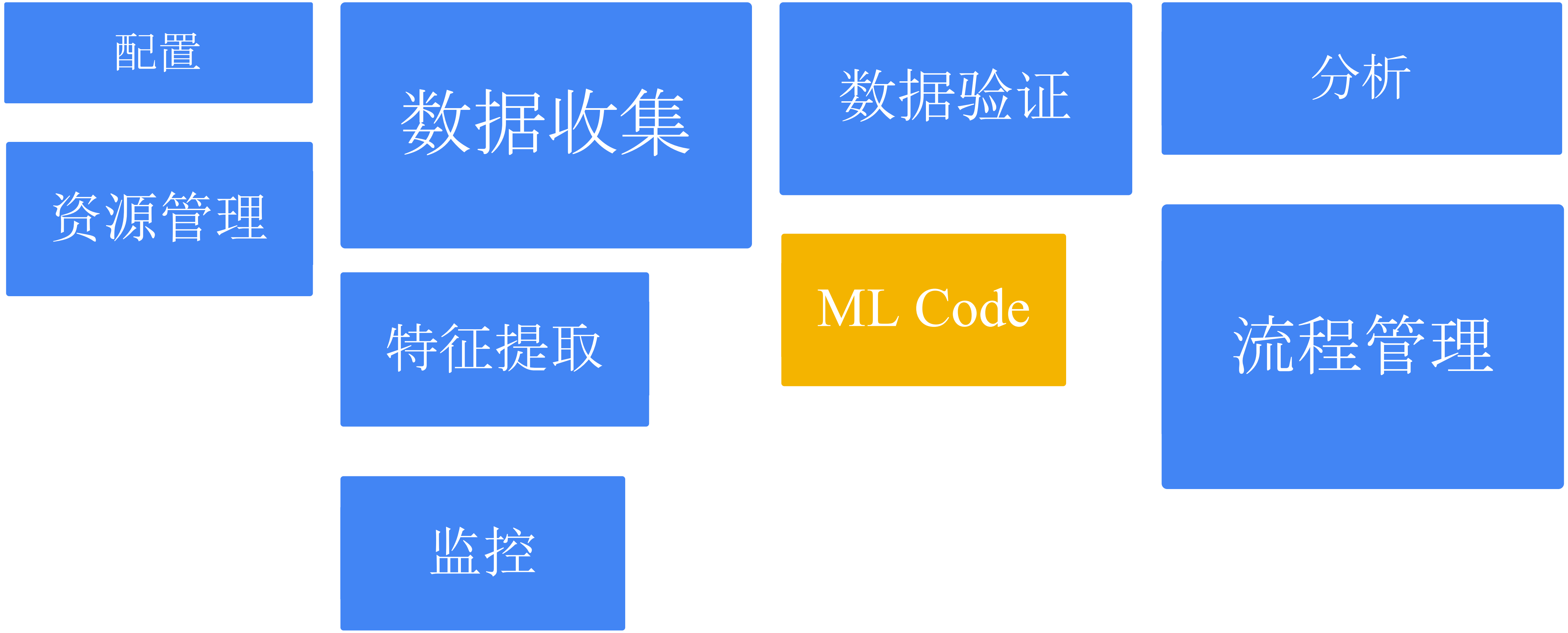
Beck, 亚当科技
beck@ilife.co

传统程序与 Machine Learning 区别



生产环境下 Machine Learning 的相关内容

ML Code 并非最重要的



配置

资源管理

数据收集

特征提取

监控

数据验证

ML Code

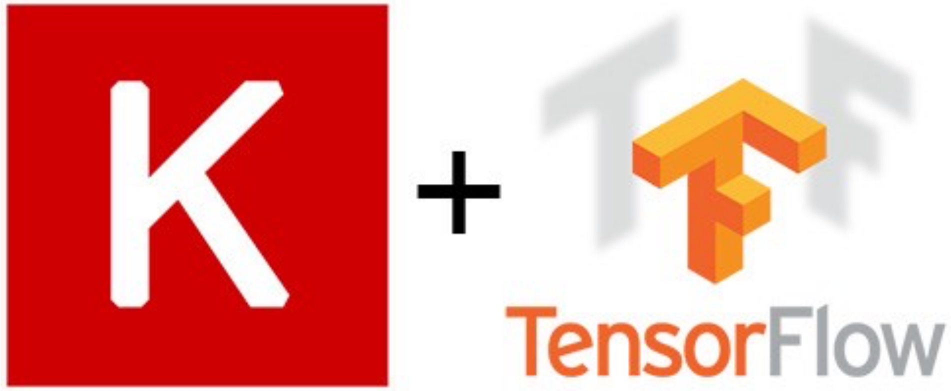
分析

流程管理

像玩 LEGO 一样
搭建 ML Model

Meet Keras with TensorFlow

Keras 是一款编写神经网络的 High-Level API.
Keras 更加侧重于实验. 用最短的时间实现你的想法.
默认情况, Keras 使用 TensorFlow 作为其 backend engine.



```
import tensorflow as tf

model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

```
FashionTest accuracy: 0.8715
```

查看 Model 结构

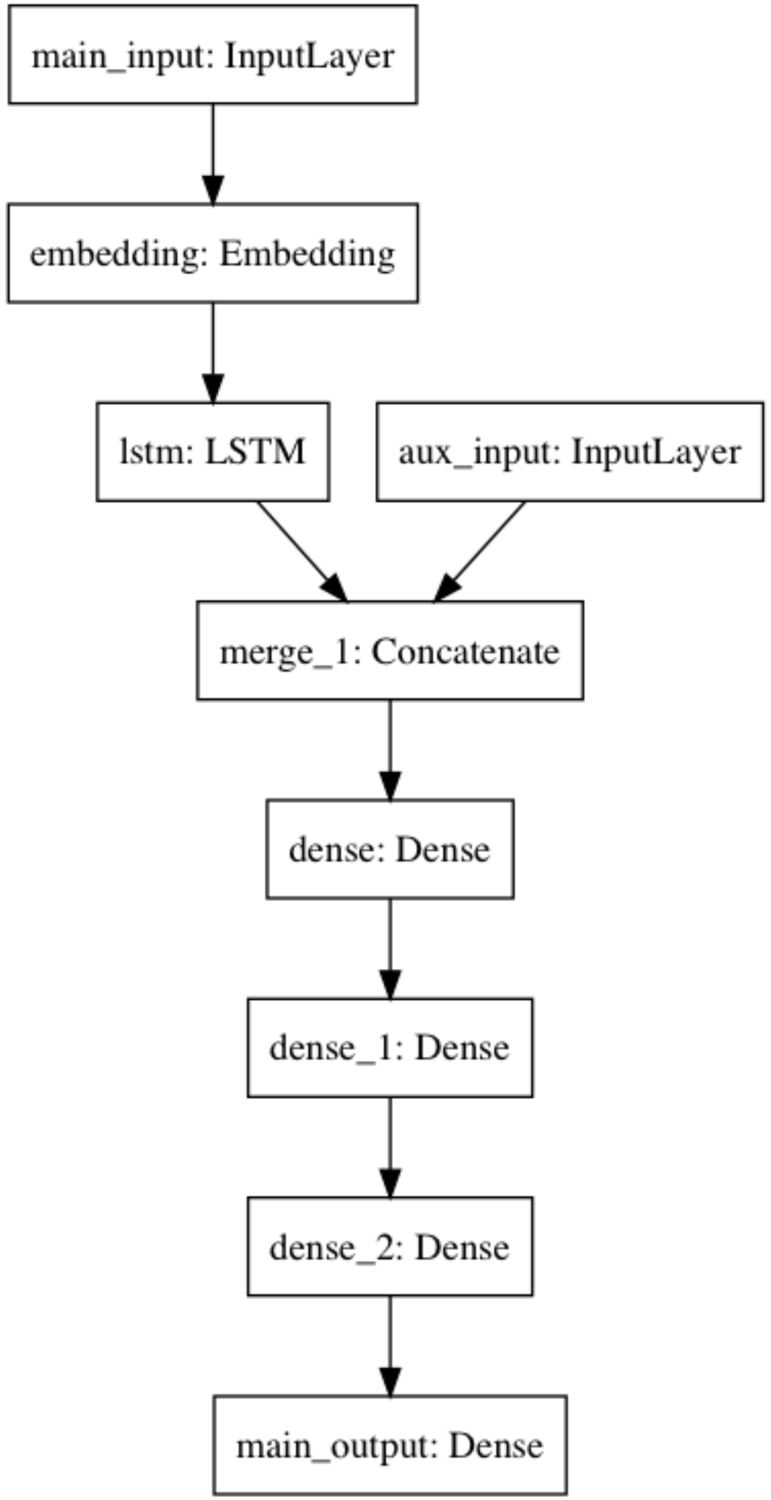
summary() 方法

```python
import tensorflow as tf

model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

能不能更LEGO 一点?

如何再现别人的 Model?


main_input: InputLayer

embedding: Embedding

lstm: LSTM

aux_input: InputLayer

merge_1: Concatenate

dense: Dense

dense_1: Dense

dense_2: Dense

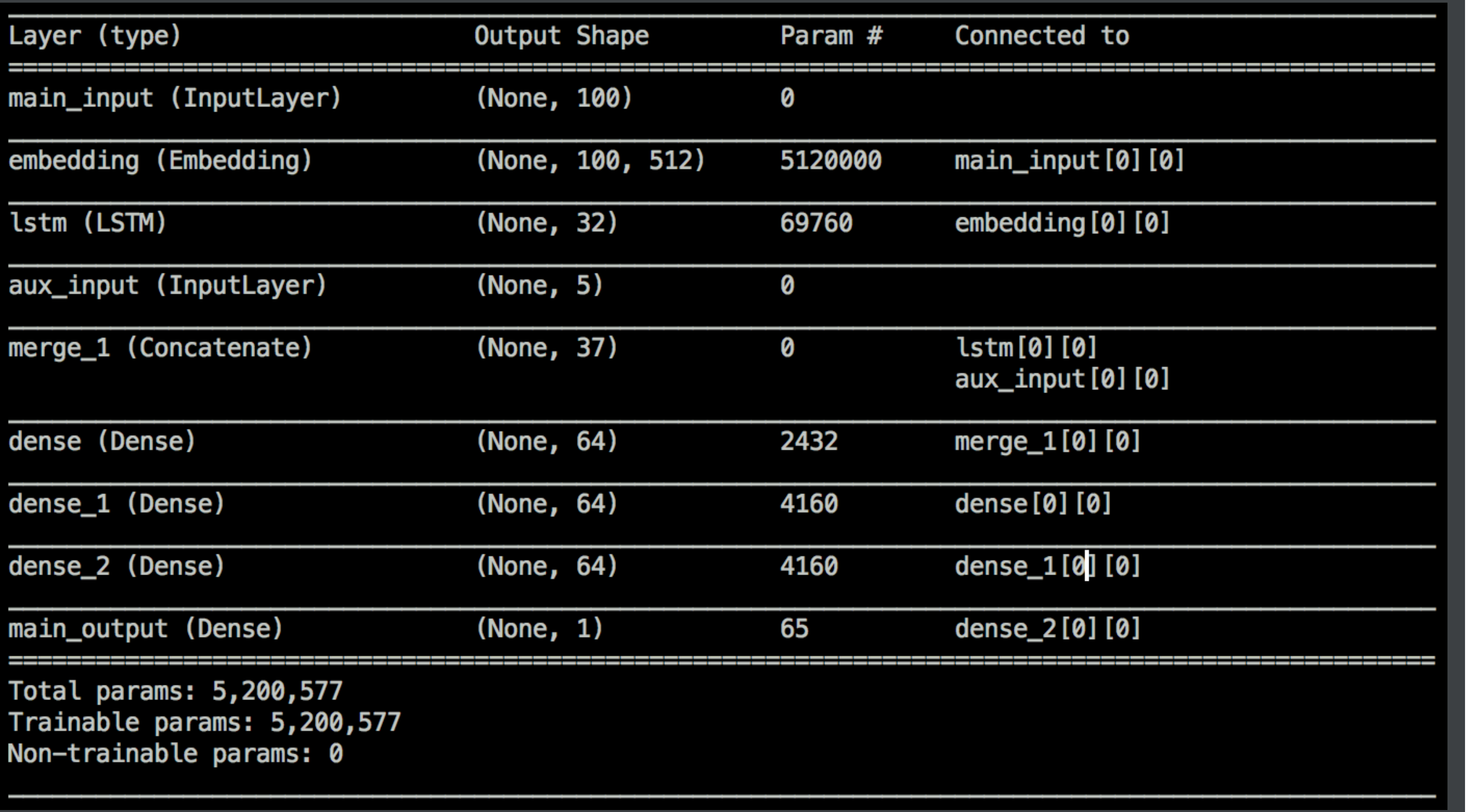
main_output: Dense

```
import tensorflow as tf


main_input= tf.keras.layers.Input(shape=(100,), dtype='int32’, name='main_input')

x = tf.keras.layers.Embedding(output_dim=512, input_dim=10000, input_length=100)(main_input)
lstm_out = tf.keras.layers.LSTM(32)(x)
anxiliary_output = tf.keras.layers.Dense(1, activation='sigmoid',
name='aux_output')(lstm_out)
auxiliary_input = tf.keras.layers.Input(shape=(5,), name='aux_input')
x = tf.keras.layers.concatenate([lstm_out, auxiliary_input], name='merge_1')

x = tf.keras.layers.Dense(64, activation='relu')(x)
x = tf.keras.layers.Dense(64, activation='relu')(x)
x = tf.keras.layers.Dense(64, activation='relu')(x)

main_output = tf.keras.layers.Dense(1, activation='sigmoid’, name='main_output')(x)

model = tf.keras.models.Model(inputs=[main_input, auxiliary_input], outputs=[main_output,
auxiliary_input])
```

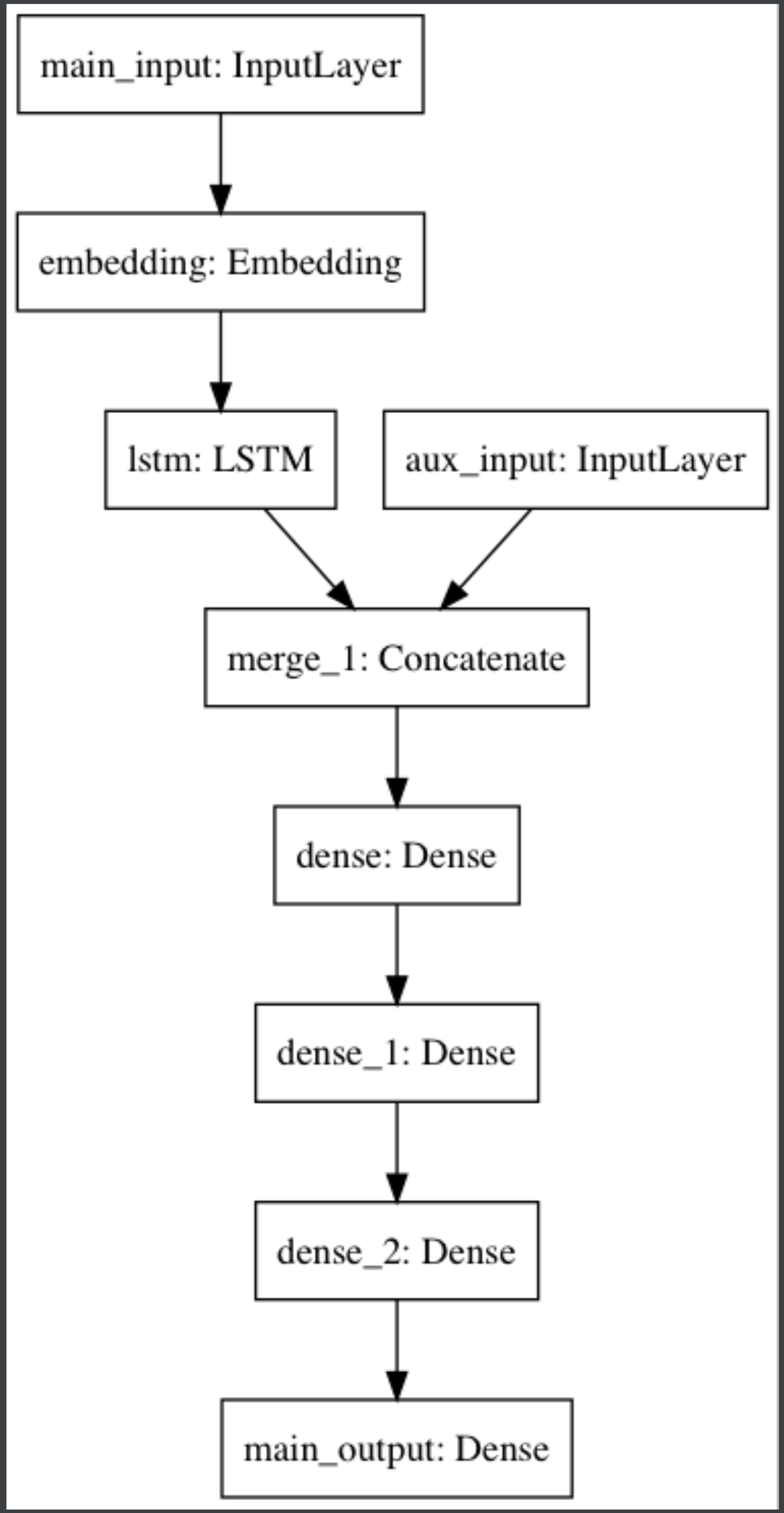
如何验证?

tf.keras.utils.plot_model

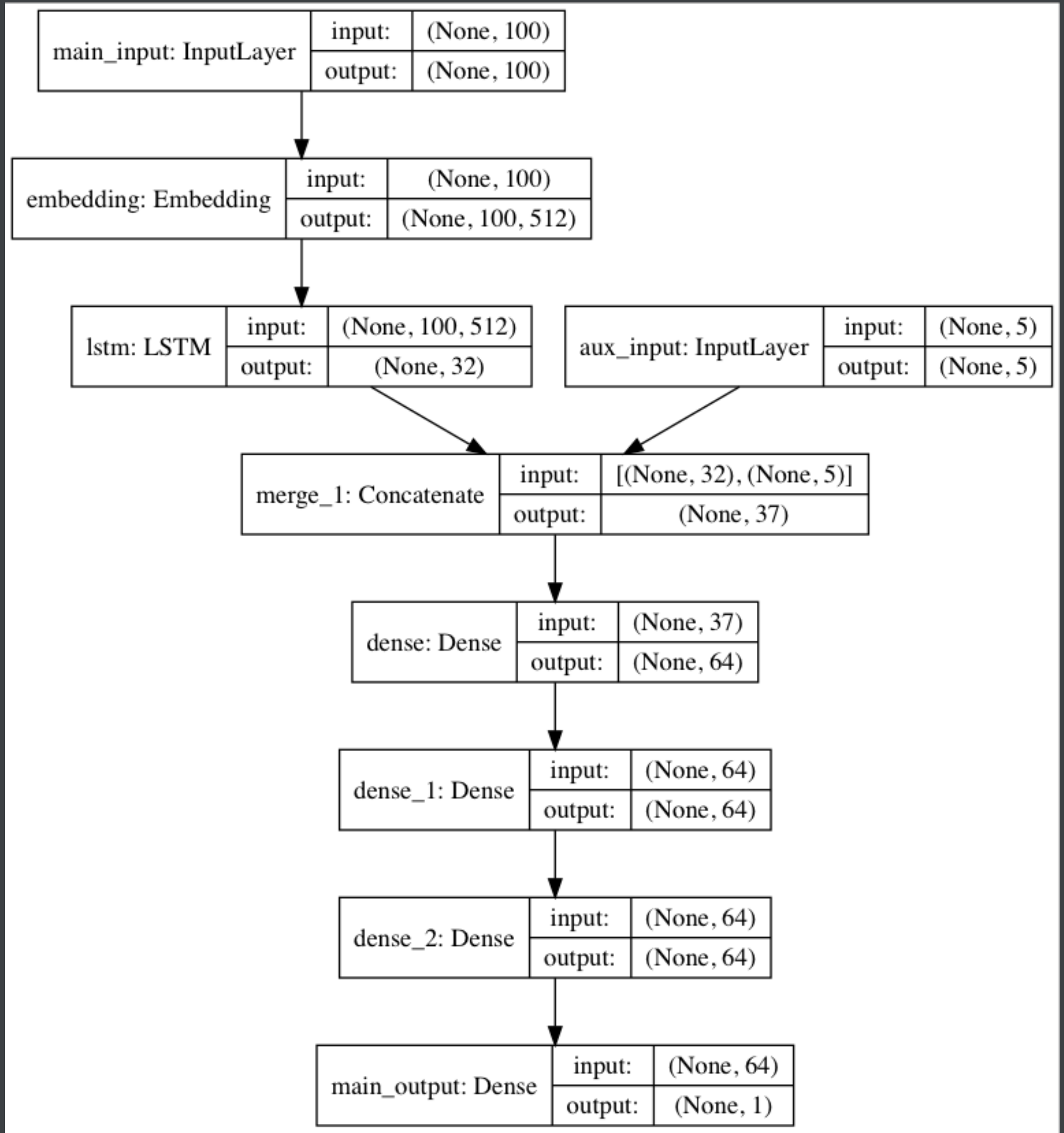
```
model.summary()
```

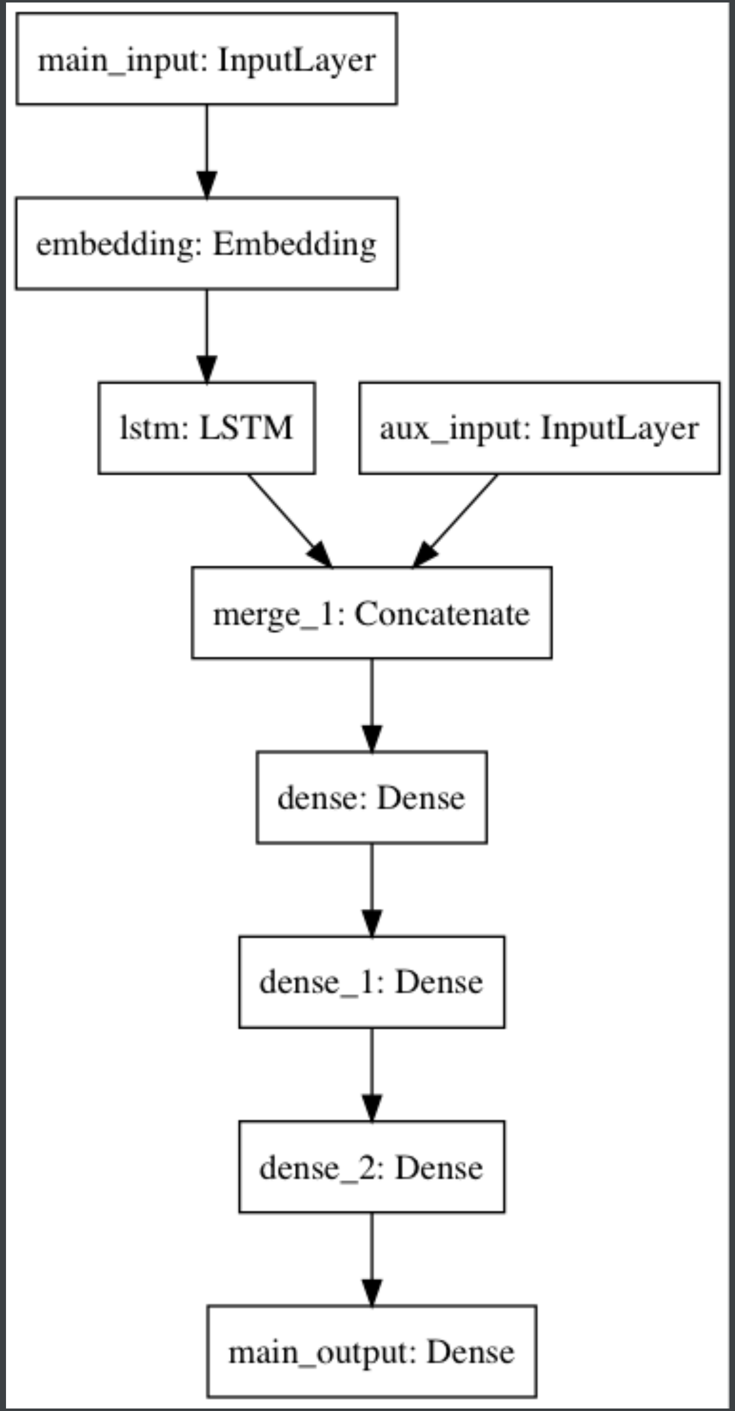


```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
main_input (InputLayer)         (None, 100)          0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 100, 512)     5120000     main_input[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 32)           69760       embedding[0][0]
__________________________________________________________________________________________________
aux_input (InputLayer)          (None, 5)            0
__________________________________________________________________________________________________
merge_1 (Concatenate)           (None, 37)           0           lstm[0][0]
                                                                 aux_input[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 64)           2432        merge_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 64)           4160        dense[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 64)           4160        dense_1[0][0]
__________________________________________________________________________________________________
main_output (Dense)             (None, 1)            65          dense_2[0][0]
==================================================================================================
Total params: 5,200,577
Trainable params: 5,200,577
Non-trainable params: 0
__________________________________________________________________________________________________
```



```
tf.keras.utils.plot_model(model, to_file='multi_input_output.png', show_shapes=True)
```





训练 & 验证

tf.keras.Model.compile
tf.keras.Model.fit

```
model.compile(optimizer=tf.train.RMSPropOptimizer(0.01),
             loss=tf.keras.losses.categorical_crossentropy,
             metrics=[tf.keras.metrics.categorical_accuracy])
```

```
model.fit(data, labels, epochs=10, batch_size=32,
           validation_data=(val_data, val_labels))
```

```
model.fit(data, labels, epochs=10, batch_size=32,
          validation_split=0.1, callbacks=[callback])
```

断点续传

Save & Restore

```
checkpoint_path = ”checkpoint/{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)
callback = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True,
    period=5)


history = model.fit(train_data, train_labels, epochs=EPOCHS,
                    validation_split=0.2, callbacks=[callback])



model.save('my_model.h5')
model.load_model(chekpoint_path)
history = model.fit(train_data, train_labels, epochs=EPOCHS,
                    validation_split=0.2, callbacks=[callback])



new_model.load_model('my_model.h5')
```

分享模型

tf.keras.Model. to_json()

tf.keras.models.model_from_json()

同样支持 yaml

```
json_string = model.to_json()

fresh_model = tf.keras.models.model_from_json(json_string)

yaml_string = model.to_yaml() # 需安装 `pyyaml`

fresh_model = tf.keras.models.model_from_yaml(yaml_string)
```

数据处理

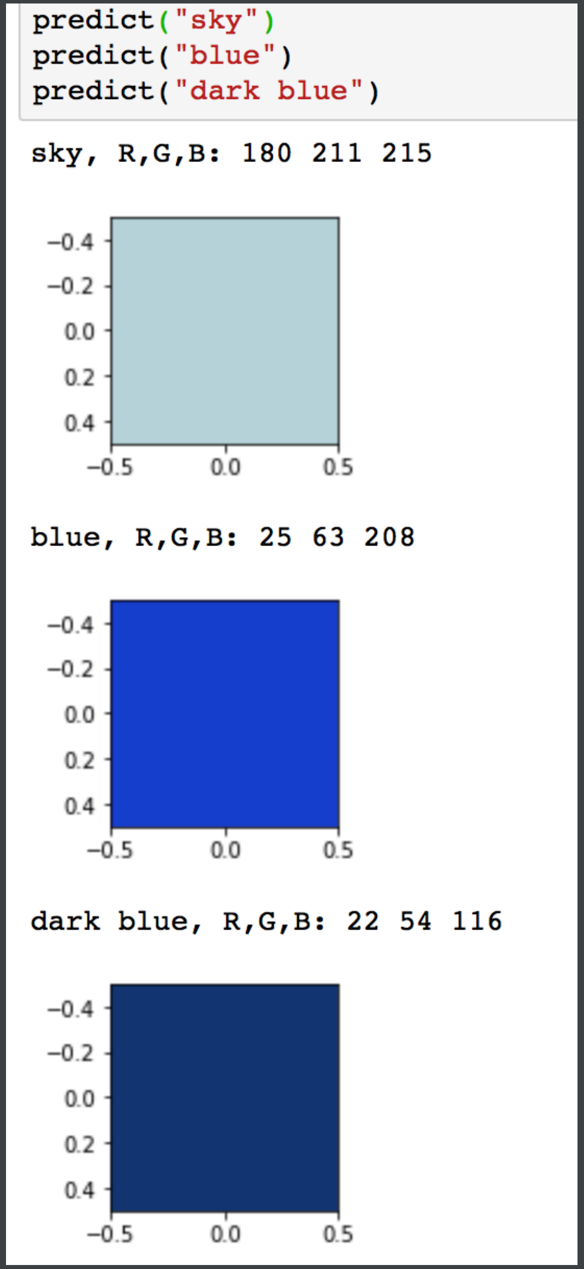
tf.keras,prerpocessing

万事开头难, 数据甚至比模型更重要
数据的获取, 数据的处理, 特征的提取往往比写模型本身更耗时.

text 预处理

tf.keras,prerpocessing.text

```
maxlen = 25
t = tf.keras.preprocessing.text.Tokenizer(char_level=True)
t.fit_on_texts(names)
tokenized = t.texts_to_sequences(names)
padded_names = tf.keras.preprocessing.sequence.pad_sequences(tokenized, maxlen=maxlen)
```

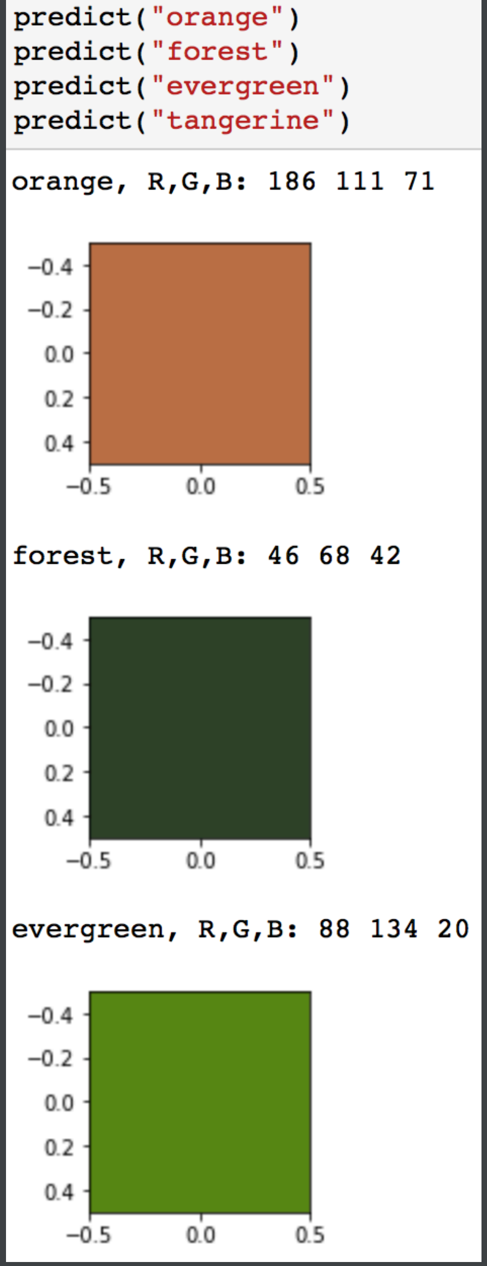


```
predict("sky")
predict("blue")
predict("dark blue")
```

```
sky, R,G,B: 180 211 215
```

```
blue, R,G,B: 25 63 208
```

```
dark blue, R,G,B: 22 54 116
```

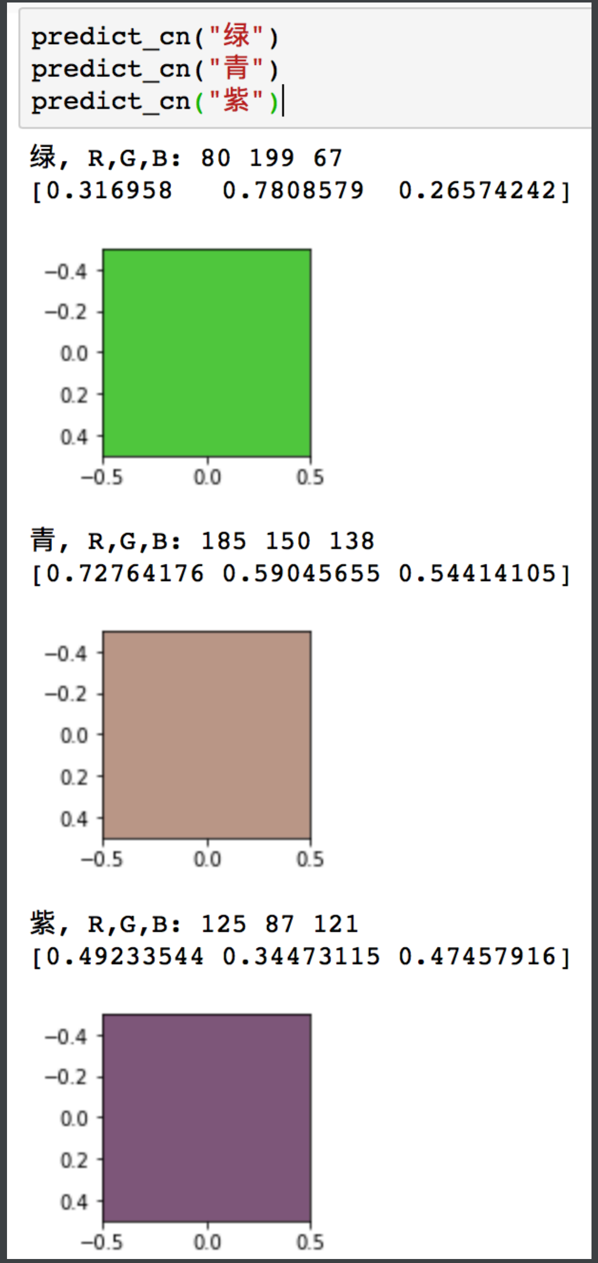


```
predict("orange")
predict("forest")
predict("evergreen")
predict("tangerine")
```

```
orange, R,G,B: 186 111 71
```

```
forest, R,G,B: 46 68 42
```

```
evergreen, R,G,B: 88 134 20
```

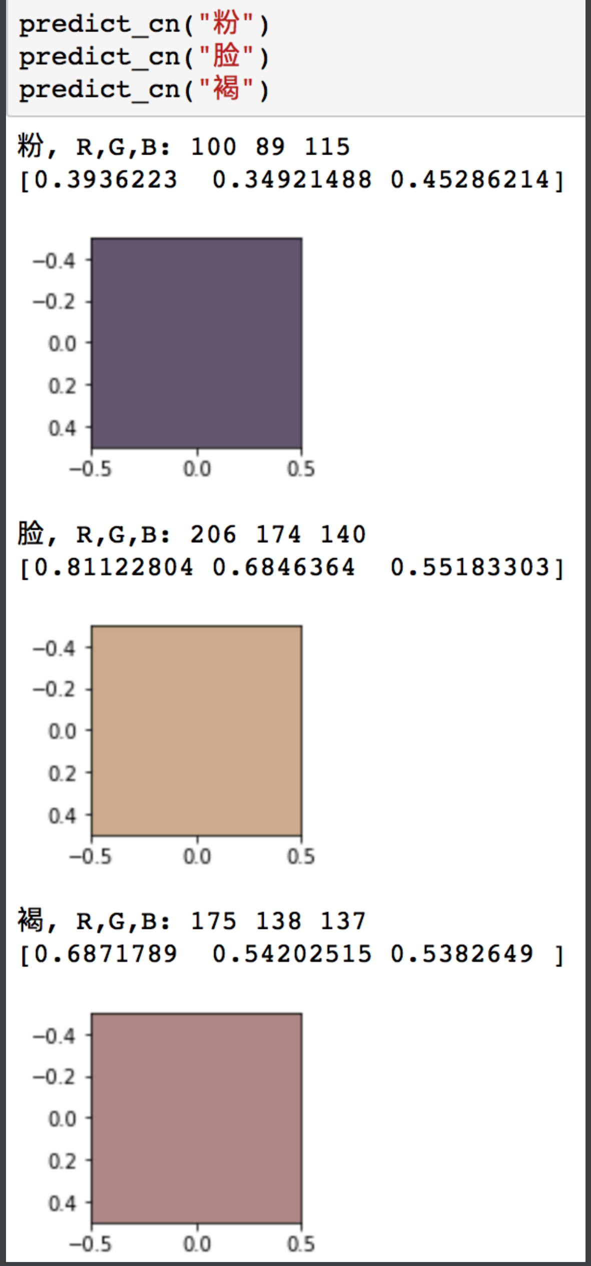


```
predict_cn("绿")
predict_cn("青")
predict_cn("紫")
```

```
绿, R,G,B: 80 199 67
[0.316958   0.7808579  0.26574242]
```

```
青, R,G,B: 185 150 138
[0.72764176 0.59045655 0.54414105]
```

```
紫, R,G,B: 125 87 121
[0.49233544 0.34473115 0.47457916]
```



```
predict_cn("粉")
predict_cn("脸")
predict_cn("褐")
```

```
粉, R,G,B: 100 89 115
[0.3936223  0.34921488 0.45286214]
```

```
脸, R,G,B: 206 174 140
[0.81122804 0.6846364  0.55183303]
```

```
褐, R,G,B: 175 138 137
[0.6871789  0.54202515 0.5382649 ]
```

Image 预处理

tf.keras,prerpocessing.image

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)

datagen= ImageDataGenerator(
  featurewise_center=True,
  featurewise_std_normalization=True,
  rotation_range=20,
  width_shift_range=0.2,
  height_shift_range=0.2,
  horizontal_flip=True)

datagen.fit(x_train)

model.fit_generator(datagen.flow(x_train, y_train, batch_size=32),
     steps_per_epoch=len(x_train) / 32, epochs=epochs)

for e in range(epochs):
  print('Epoch', e)
  batches = 0
  for x_batch, y_batch in datagen.flow(x_train, y_train, batch_size=32):
    model.fit(x_batch, y_batch)
    batches += 1
    if batches >= len(x_train) / 32:
      break
```

What’s next?

基于 keras Model 自定义 Model…

Tensor2Tensor

TensorFlow Extended

TensorHub

…

自定义 Model / Layer

tf.keras,Model
tf.keras.layers.Layer

```
inputs = tf.keras.Input(shape=(3,))
x = tf.keras.layers.Dense(4, activation=tf.nn.relu)(inputs)
outputs = tf.keras.layers.Dense(5, activation=tf.nn.softmax)(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

```
class MyModel(tf.keras.Model):
  def __init__(self, num_classes=10):
    super(MyModel, self).__init__(name='my_model‘)
    self.num_classes = num_classes
    # 在这里自定义 layer.
    self.dense_1 = layers.Dense(32, activation='relu‘)
    self.dense_2 = layers.Dense(num_classes, activation='sigmoid‘)

  def call(self, inputs):
    # 这里使用在 `__init__` 函数内定义的层来自定义前传
    x = self.dense_1(inputs)
    return self.dense_2(x)

  def compute_output_shape(self, input_shape):
    # 如果你想将自己定义的 model 子类作为 functional-style 模型的一部分，你需要重写这个方法
    # 如果不想，则不需要重写
    shape = tf.TensorShape(input_shape).as_list()
    shape[-1] = self.num_classes
    return tf.TensorShape(shape)
```

```
class MyLayer(layers.Layer):

  def __init__(self, output_dim, **kwargs):
    self.output_dim = output_dim
    super(MyLayer, self).__init__(**kwargs)

  def build(self, input_shape):
    shape = tf.TensorShape((input_shape[1], self.output_dim))
    # 为自定义 layer, 创建一个可以被训练的 weight 变量.
    self.kernel = self.add_weight(name=‘kernel‘, shape=shape, initializer='uniform‘,
                                  trainable=True)
    super(MyLayer, self).build(input_shape)

  def call(self, inputs):
    return tf.matmul(inputs, self.kernel)

  def compute_output_shape(self, input_shape):
    shape = tf.TensorShape(input_shape).as_list()
    shape[-1] = self.output_dim
    return tf.TensorShape(shape)

  def get_config(self):
    base_config = super(MyLayer, self).get_config()
    base_config['output_dim'] = self.output_dim
    return base_config

  @classmethod
  def from_config(cls, config):
  return cls(**config)
```

Tensor2Tensor

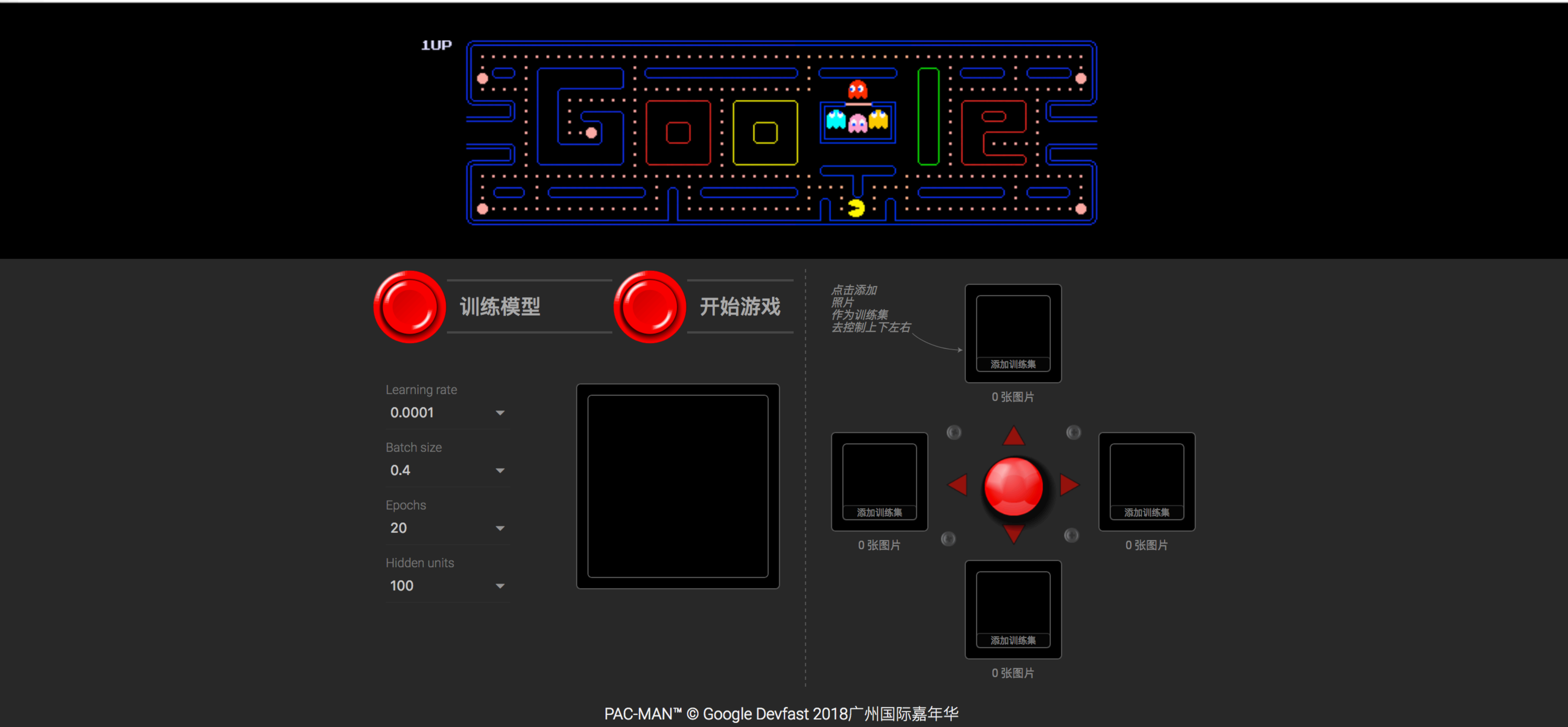
快速学习, 复现, 研究

```python
from tensor2tensor import models
from tensor2tensor import problems
from tensor2tensor.layers import common_layers
from tensor2tensor.utils import trainer_lib
from tensor2tensor.utils import t2t_model
from tensor2tensor.utils import registry
from tensor2tensor.utils import metrics

tensor2tensor.problems.available() # 628 个

# 获取 MNIST 课题
mnist_problem = problems.problem("image_mnist")
# generate_data 会将当前课题的数据下载，并处理为一个标准格式以供训练和评估.
mnist_problem.generate_data(data_dir, tmp_dir)
```

Google Developers

GDG Guangzhou

DevFest



Learning rate
0.0001

Batch size
0.4

Epochs
20

Hidden units
100

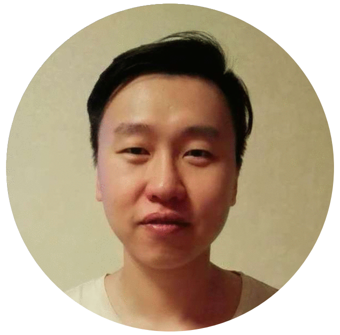
点击添加照片
作为训练集
去控制上下左右

PAC-MAN™ © Google Devfast 2018广州国际嘉年华

GDG Guangzhou

DevFest

Thank you!



Beck, 亚当科技
beck@ilife.co