Keras 是一个高层神经网络 API，Keras 由纯 Python 编写而成并基于 Theano/Tensorflow/CNTK 后端。Keras 为支持快速实验而生，能够把你的 idea 迅速转换为结果。

Keras 后端：
Keras 是一个模型级的库，提供了快速构建深度学习网络的模块。Keras 并不处理如张量乘法、卷积等底层操作。这些操作依赖于某种特定的、优化良好的张量操作库。Keras 依赖于处理张量的库就称为“后端引擎”。Keras 提供了三种后端引擎 Theano/Tensorflow/CNTK，并将其函数统一封装，使得用户可以以同一个接口调用不同后端引擎的函数

序贯模型 Sequential：由多个网络层线性堆叠，是函数式模型 Model 的一种特殊情况。
两类模型的一些方法：
```
from keras.models import model_from_json
from keras.models import model_from_yaml
```
已定义模型对象 model
`model.summary()`：打印模型情况
`model.get_config()`：返回包含模型信息的 Python 字典
`model.get_layer()`：依据层名或下标获取层对象
`weights = model.get_weights()`：返回模型权重的列表 ↔
`model = model.set_weights(weights)`
`json_string = model.to_json()`：返回模型的 Json 字符 ↔
`model = model_from_json(json_string)`
`yaml_string = model.to_yaml()`：返回模型的 yaml 字符 ↔
`model = model_from_yaml(yaml_string)`
`model.save_weights(filepath)`：将模型权重保存到指定路径，文件类型是 HDF5（后缀是.h5）↔
`model.load_weights(filepath)`

Keras 模型 API
1. 序贯模型常用属性：
`model.layers` 是添加到模型上的层的 list
2. 函数式模型常用属性：
`model.layers`：组成模型图的各个层
`model.input(s)`：模型的输入张量列表
`model.output(s)`：模型的输出张量列表
3. 两种模型常用方法：
`add(self, layer)` 向模型中添加一个层，layer 是 Layer 对象
`pop(self)` 弹出模型最后的一层，无返回值
`compile(self, optimizer, loss, metrics=None, sample_weight_mode=None)` 编译用来配置模型的学习过程
`fit(self, x, y, batch_size=32, epochs=10, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0)` 模型训练 nb_epoch 轮
`evaluate(self, x, y, batch_size=32, verbose=1, sample_weight=None)` 按 batch 计算在某些输入数据上模型的误差
`predict(self, x, batch_size=32, verbose=0)` 按 batch 获得输入数据对应的输出
`train_on_batch(self, x, y, class_weight=None, sample_weight=None)` 在一个 batch 的数据上进行一次参数更新
`test_on_batch(self, x, y, sample_weight=None)` 在一个 batch 的样本上对模型进行评估
`predict_on_batch(self, x)` 在一个 batch 的样本上对模型进行测试，函数返回模型在一个 batch 上的预测结果

Keras 层 API

所有的 Keras 层对象都有如下方法：

layer.get_weights()：返回层的权重（numpy array）

layer.set_weights(weights)：从 numpy array 中将权重加载到该层中，要求 numpy array 的形状与 *layer.get_weights()*的形状相同

layer.get_config()：返回当前层配置信息的字典，层也可以借由配置信息重构

如果层仅有一个计算节点（即该层不是共享层），则可以通过下列方法获得输入张量、输出张量、输入数据的形状和输出数据的形状：

```
layer.input
layer.output
layer.input_shape
layer.output_shape
```

如果该层有多个计算节点。可以使用下面的方法(node_index 由 0 开始往上计)：

```
layer.get_input_at(node_index)
layer.get_output_at(node_index)
layer.get_input_shape_at(node_index)
layer.get_output_shape_at(node_index)
```

看下边的例子：

```
a = Input(shape=(140, 256))
b = Input(shape=(140, 256))
lstm = LSTM(32)
encoded_a = lstm(a)
encoded_b = lstm(b)
# lstm.output 会因歧义而报错，应该使用下边形式：
assert lstm.get_output_at(0) == encoded_a
assert lstm.get_output_at(1) == encoded_b
```

再来一个 input_shape 的例子：

```
a = Input(shape=(32, 32, 3))
b = Input(shape=(64, 64, 3))
conv = Conv2D(16, (3, 3), padding='same')
conved_a = conv(a)
# Only one input so far, the following will work:
assert conv.input_shape == (None, 32, 32, 3)
conved_b = conv(b)
# now the `.input_shape` property wouldn't work, but this does:
assert conv.get_input_shape_at(0) == (None, 32, 32, 3)
assert conv.get_input_shape_at(1) == (None, 64, 64, 3)
```

讲完概念和常用 API，接下来构建实际的模型，如下是一个完整的十分类模型：

```
# For a single-input model with 10 classes (categorical classification):
import keras
from keras.models import Sequential
from keras.layers import Dense,Activation
model = Sequential()
# Dense(32) is a fully-connected layer with 32 hidden units.
# in the first layer, you must specify the expected input data shape:
# here, 100-dimensional vectors.
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accu
racy'])
# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(10, size=(1000, 1))
# Convert labels to categorical one-hot encoding. util 是该层的输出维度
one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)
```

```
# Train the model, iterating on the data in batches of 32 samples
model.fit(data, one_hot_labels, epochs=10, batch_size=32)
```

[关于 batch_size：梯度下降的更新方式有两种，一种是每看一个数据就算一下损失函数，然后求梯度更新参数，称为随机梯度下降（Stochastic gradient descent），该方法速度快，但收敛性能不太好，两次参数的更新也有可能抵消掉;另一种是遍历全部数据集后算一次损失函数，然后对函数各个参数求梯度更新参数，称为批梯度下降（Batch gradient descent），该方法每更新一次参数需要遍历一遍数据集，计算速度慢。现在多采用折中两种方法的小批量梯度下降（mini-batch gradient descent），即把数据集分为若干批，按批来更新参数，即减少了随机性，计算量也不是很大。Keras 训练过程中的 batch_size 即指的对应每次训练的小批量的数据量大小。关于 epochs：轮次，每个 batch 对应的是网络的一次更新，而一个 epoch 对应网络的一轮更新。每一轮更新中网络更新的次数可以随意，通常设置为遍历一遍数据集。设置 epoch 将模型的训练分为若干段，可以更好的观察和调整模型的训练。]

类似(data,one_hot_labels)，定义验证集(x_test,y_test),然后测试：

```
score = model.evaluate(x_test,y_test,batch_size=32)
```

卷积神经网络

```
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.layers import Conv2D,MaxPooling2D
model = Sequential()
# input: 100x100 images with 3 channels -> (100, 100, 3) tensors.
# this applies 32 convolution filters of size 3x3 each -> Conv2D(32,(3,3).
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(100,100,3)))
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
# Flatten 层用来将输入压平，即将多为输入一维化，常用于从卷积层到全连接层的过渡。
# 如，上一步 model.output_shape == (None,49,49,32)
model.add(Flatten())
# 此时 model.output_shape == (None,76832)
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

在 CNN 中，卷积层的常用广义定义为：

```
keras.layers.convolutional.Conv2D(filters, kernel_size, strides=(1, 1),
padding='valid', data_format=None)
```

池化层的广义定义为：

```
keras.layers.pooling.MaxPooling2D(pool_size=(2, 2),strides=None,padding='valid',
data_format=None)
```

其中 padding 默认都为 valid（不填充），还可取 same(全 0 填充)

循环神经网络

实现 stateful LSTM，该模型在处理过一个 batch 的训练数据后，其内部状态（记忆）会被作为下一个 batch 的训练数据的初始状态。状态 LSTM 使得我们可以在合理的计算复杂度内处理较长序列。

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np

data_dim = 16
timesteps = 8
num_classes = 10
batch_size = 32
# Expected input batch shape: (batch_size, timesteps, data_dim)
# Note that we have to provide the full batch_input_shape since the network is
stateful.
# the sample of index i in batch k is the follow-up for the sample i in batch k-
# 1.
model = Sequential()
# LSTM 中 32 为输出维度，stateful 为 True,则下一个 batch 中下标为 i 的样本的最终状态将会作为下一
# 个 batch 同样下标的样本的初始状态
model.add(LSTM(32, return_sequences=True, stateful=True,
               batch_input_shape=(batch_size, timesteps, data_dim)))
model.add(LSTM(32, return_sequences=True, stateful=True))
model.add(LSTM(32, stateful=True))
```

```
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
# Generate dummy training data
x_train = np.random.random((batch_size * 10, timesteps, data_dim))
y_train = np.random.random((batch_size * 10, num_classes))
# Generate dummy validation data
x_val = np.random.random((batch_size * 3, timesteps, data_dim))
y_val = np.random.random((batch_size * 3, num_classes))
# Shuffle 为 False 表示在训练每个 epoch 前不随机打乱输入样本的顺序，默认为 True
model.fit(x_train, y_train,
          batch_size=batch_size, epochs=5, shuffle=False,
          validation_data=(x_val, y_val))
```

除了 LSTM，Keras 还提供了 GRU,SimpleRNN 等经典 RNN 网络。

函数式模型接口

```
from keras.models import Model
from keras.layers import Input,Dense
a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a,outputs=b)
```

构建了以 a 为输入，b 为输出的模型;也可以构造多输入多输出模型：

```
model = Model(inputs=[a1,a2],outputs=[b1,b2,b3])
```

函数式模型

现在来学习更加广义的模型编程，有几个概念需要澄清：

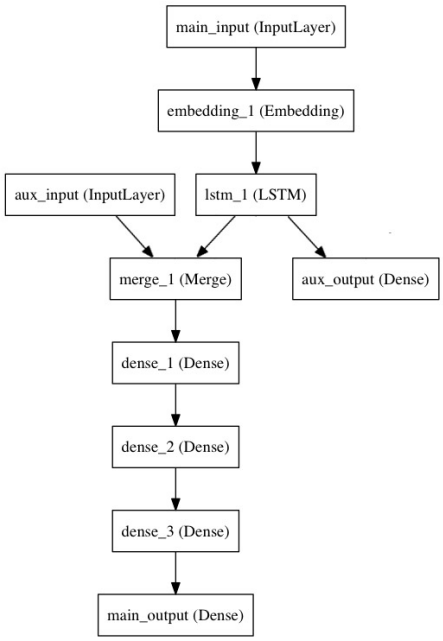
- 层对象接受张量为参数，返回一个张量。
- 输入是张量，输出也是张量的一个框架就是一个模型，通过 Model 定义。
- 这样的模型可以像 Keras 的 Sequential 一样被训练

首先使用 Model 来实现三层全连接网络：

```
from keras.models import Model
from keras.layers import Input,Dense
input = Input(shape=(784,))
x = Dense(64,activation='relu')(inputs)
x = Dense(64,activation='relu')(x)
predictions = Dense(10,activation='softmax')(x)
model = Model(inputs=input, outputs=predictions)
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accu
racy'])
model.fit(data,labels) #starts training
```

训练好的模型是可调用的，就像层一样。

使用函数式模型的一个典型场景是搭建多输入、多输出模型。考虑这样一个模型。我们希望预测 Twitter 上一条新闻会被转发和点赞多少次。模型的主要输入是新闻本身，也就是一个词语的序列。但我们还可以拥有额外的输入，如新闻发布的日期等。这个模型的损失函数将由两部分组成，辅助的损失函数评估仅仅基于新闻本身做出预测的情况，主损失函数评估基于新闻和额外信息的预测的情况，即使来自主损失函数的梯度发生弥散，来自辅助损失函数的信息也能够训练 Embeddding 和 LSTM 层。在模型中早点使用主要的损失函数是对于深度网络的一个良好的正则方法。总而言之，该模型框图如下：




```
from keras.layers import Input, Embedding, LSTM, Dense
from keras.models import Model
# Headline input: meant to receive sequences of 100 integers, between 1 and
10000.
# Note that we can name any layer by passing it a "name" argument.
main_input = Input(shape=(100,), dtype='int32', name='main_input')
# This embedding layer will encode the input sequence
# into a sequence of dense 512-dimensional vectors.
x = Embedding(output_dim=512, input_dim=10000, input_length=100)(main_input)
# A LSTM will transform the vector sequence into a single vector,
# containing information about the entire sequence
lstm_out = LSTM(32)(x)
# 插入一个额外的损失，使得即使在主损失很高的情况下，LSTM 和 Embedding 层也可以平滑的训练
auxiliary_output = Dense(1, activation='sigmoid', name='aux_output')(lstm_out)
# 将 LSTM 与额外的输入数据串联起来组成输入，送入模型中：
auxiliary_input = Input(shape=(5,), name='aux_input')
x = keras.layers.concatenate([lstm_out, auxiliary_input])
# We stack a deep densely-connected network on top
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
# And finally we add the main logistic regression layer
main_output = Dense(1, activation='sigmoid', name='main_output')(x)
# 定义 2 输入 2 输出模型：
model = Model(inputs=[main_input, auxiliary_input], outputs=[main_output,
auxiliary_output])
# 赋权重
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
              loss_weights=[1., 0.2])
# 编译好了后，通过传递训练数据和目标值训练模型：
model.fit([headline_data, additional_data], [labels, labels],
          epochs=50, batch_size=32)
```

另一个常用的场景是使用**共享层**。考虑 Twitter 的微博数据，我们希望建立模型来判别两条微博是否是来自同一个用户，这个需求同样可以用来判断一个用户的两条微博的相似性。

一种实现方式是，我们建立一个模型，它分别将两条微博的数据映射到两个特征向量上，然后将特征向量串联并加一个 logistic 回归层，输出它们来自同一个用户的概率。这种模型的训练数据是一对对的微博。因为这个问题是对称的，所以处理第一条微博的模型当然也能重用于处理第二条微博。所以这里我们使用一个共享的 LSTM 层来进行映射。因为一条微博最多有 140 个字符，而扩展的 ASCII 码表编码了常见

的 256 个字符，所以我们将微博的数据转为（140，256）的矩阵，即每条微博有 140 个字符，每个单词的特征由一个 256 维的词向量表示，向量的每个元素为 1 表示某个字符出现，为 0 表示不出现，这是一个 one-hot 编码。

```
import keras
from keras.layers import Input, LSTM, Dense
from keras.models import Model
tweet_a = Input(shape=(140, 256))
tweet_b = Input(shape=(140, 256))
# This layer can take as input a matrix and will return a vector of size 64
shared_lstm = LSTM(64)
# When we reuse the same layer instance multiple times, the weights of the layer
# are also being reused (it is effectively *the same* layer)
encoded_a = shared_lstm(tweet_a)
encoded_b = shared_lstm(tweet_b)
# We can then concatenate the two vectors:
merged_vector = keras.layers.concatenate([encoded_a, encoded_b], axis=-1)
# And add a logistic regression on top
predictions = Dense(1, activation='sigmoid')(merged_vector)
# We define a trainable model linking the
# tweet inputs to the predictions
model = Model(inputs=[tweet_a, tweet_b], outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit([data_a, data_b], labels, epochs=10)
```

经典模型实现

Inception 模型

```
from keras.layers import Conv2D, MaxPooling2D, Input
input_img = Input(shape=(256, 256, 3))
tower_1 = Conv2D(64, (1, 1), padding='same', activation='relu')(input_img)
tower_1 = Conv2D(64, (3, 3), padding='same', activation='relu')(tower_1)
tower_2 = Conv2D(64, (1, 1), padding='same', activation='relu')(input_img)
tower_2 = Conv2D(64, (5, 5), padding='same', activation='relu')(tower_2)
tower_3 = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(input_img)
tower_3 = Conv2D(64, (1, 1), padding='same', activation='relu')(tower_3)
output = keras.layers.concatenate([tower_1, tower_2, tower_3], axis=1)
```

共享视觉模型

该模型在两个输入上重用了图像处理的模型，用来判别两个 MNIST 数字是否是相同的数字，与上面判断微博是否来自同一用户的例子不同（共享层），这个模型是共享模型。

```
from keras.layers import Conv2D, MaxPooling2D, Input, Dense, Flatten
from keras.models import Model
# First, define the vision modules
digit_input = Input(shape=(27, 27, 1))
x = Conv2D(64, (3, 3))(digit_input)
x = Conv2D(64, (3, 3))(x)
x = MaxPooling2D((2, 2))(x)
out = Flatten()(x)
vision_model = Model(digit_input, out)
# Then define the tell-digits-apart model
digit_a = Input(shape=(27, 27, 1))
digit_b = Input(shape=(27, 27, 1))
# The vision model will be shared, weights and all
out_a = vision_model(digit_a)
out_b = vision_model(digit_b)
concatenated = keras.layers.concatenate([out_a, out_b])
out = Dense(1, activation='sigmoid')(concatenated)
classification_model = Model([digit_a, digit_b], out)
```

视觉问答模型

在针对一幅图片使用自然语言进行提问时，该模型能够提供关于该图片的一个单词的答案

这个模型将自然语言的问题和图片分别映射为特征向量，将二者合并后训练一个 logistic 回归层，从一系列可能的回答中挑选一个。

```
from keras.layers import Conv2D, MaxPooling2D, Flatten
from keras.layers import Input, LSTM, Embedding, Dense
from keras.models import Model, Sequential
# First, let's define a vision model using a Sequential model.
# This model will encode an image into a vector.
vision_model = Sequential()
vision_model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
input_shape=(224, 224, 3)))
vision_model.add(Conv2D(64, (3, 3), activation='relu'))
vision_model.add(MaxPooling2D((2, 2)))
vision_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
vision_model.add(Conv2D(128, (3, 3), activation='relu'))
vision_model.add(MaxPooling2D((2, 2)))
vision_model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
vision_model.add(Conv2D(256, (3, 3), activation='relu'))
vision_model.add(Conv2D(256, (3, 3), activation='relu'))
vision_model.add(MaxPooling2D((2, 2)))
vision_model.add(Flatten())
# Now let's get a tensor with the output of our vision model:
image_input = Input(shape=(224, 224, 3))
encoded_image = vision_model(image_input)
# Next, let's define a language model to encode the question into a vector.
# Each question will be at most 100 word long,
# and we will index words as integers from 1 to 9999.
question_input = Input(shape=(100,), dtype='int32')
embedded_question = Embedding(input_dim=10000, output_dim=256, input_length=100)
(question_input)
encoded_question = LSTM(256)(embedded_question)
# Let's concatenate the question vector and the image vector:
merged = keras.layers.concatenate([encoded_question, encoded_image])
# And let's train a logistic regression over 1000 words on top:
output = Dense(1000, activation='softmax')(merged)
# This is our final model:
vqa_model = Model(inputs=[image_input, question_input], outputs=output)
# The next stage would be training this model on actual data.
```

以上笔记全部整理自 Keras 中文文档：
https://keras-cn.readthedocs.io