高级 API 接口 -- keras

快速开始： 30s 上手 Keras

建立模型

```
from keras.models import Sequential
model = Sequential()
```

网络堆叠

```
from keras.layers import Dense
model.add(Dense(units=64, activation='relu',
                input_dim=100))
model.add(Dense(units=10, activation='softmax'))
```

编译模型

```
model.compile(
    loss='categorical_crossentropy',
    optimizer='sgd',
    metrics=['accuracy'])
```

训练模型

```
model.fit(x_train, y_train, epochs=5, batch_size=32)

model.train_on_batch(x_batch, y_batch)
```

验证模型

```
loss_and_metrics = model.evaluate(x_test, y_test,
                                  batch_size=128)
```

预测

```
classes = model.predict(x_test, batch_size=128)
```

```
Using TensorFlow backend.
Training started
Start generating data.
/media/song/_D/python/MobileNetV2/fer/data/FER2013Train
/media/song/_D/python/MobileNetV2/fer/data/FER2013Train/label.csv
Start generating data.
/media/song/_D/python/MobileNetV2/fer/data/FER2013Valid
/media/song/_D/python/MobileNetV2/fer/data/FER2013Valid/label.csv
Start generating data.
/media/song/_D/python/MobileNetV2/fer/data/FER2013Test
/media/song/_D/python/MobileNetV2/fer/data/FER2013Test/label.csv
Dataset load success!!
2018-06-07 11:15:03.337283: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
2018-06-07 11:15:03.436655: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:892] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2018-06-07 11:15:03.437061: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1030] Found device 0 with properties:
name: GeForce GTX 1080 major: 6 minor: 1 memoryClockRate(GHz): 1.7335
pciBusID: 0000:01:00.0
totalMemory: 7.92GiB freeMemory: 4.29GiB
2018-06-07 11:15:03.437074: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1120] Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: GeForce GTX 1080, pci bus id: 0000:01:00.0, compute capability: 6.1)
Train on 25045 samples, validate on 3191 samples
Epoch 1/500

   16/25045 [..............................] - ETA: 1:30:34 - loss: 2.5709 - acc: 0.1250
   48/25045 [..............................] - ETA: 30:48 - loss: 7.3772 - acc: 0.1875
   80/25045 [..............................] - ETA: 18:50 - loss: 7.0860 - acc: 0.2375
  112/25045 [..............................] - ETA: 13:43 - loss: 7.1052 - acc: 0.2232
  144/25045 [..............................] - ETA: 10:52 - loss: 6.2558 - acc: 0.2639
  176/25045 [..............................] - ETA: 9:03 - loss: 5.7273 - acc: 0.2557
  208/25045 [..............................] - ETA: 7:47 - loss: 5.4733 - acc: 0.2500
  240/25045 [..............................] - ETA: 6:52 - loss: 5.3174 - acc: 0.2667
  272/25045 [..............................] - ETA: 6:10 - loss: 4.9231 - acc: 0.2721
  304/25045 [..............................] - ETA: 5:36 - loss: 4.6516 - acc: 0.2763
  336/25045 [..............................] - ETA: 5:09 - loss: 4.4860 - acc: 0.2827
  368/25045 [..............................] - ETA: 4:47 - loss: 4.3030 - acc: 0.2853
  400/25045 [..............................] - ETA: 4:28 - loss: 4.2186 - acc: 0.2925
  432/25045 [..............................] - ETA: 4:12 - loss: 4.1229 - acc: 0.2917
  464/25045 [..............................] - ETA: 3:58 - loss: 4.0520 - acc: 0.2802
  496/25045 [..............................] - ETA: 3:46 - loss: 3.9573 - acc: 0.2863
  528/25045 [..............................] - ETA: 3:35 - loss: 3.8725 - acc: 0.2898
  560/25045 [..............................] - ETA: 3:26 - loss: 3.8550 - acc: 0.2911
```

```
24304/25045 [============================>.] - ETA: 1s - loss: 8.6100 - acc: 0.3410
24336/25045 [============================>.] - ETA: 1s - loss: 8.6073 - acc: 0.3413
24368/25045 [============================>.] - ETA: 1s - loss: 8.6066 - acc: 0.3415
24400/25045 [============================>.] - ETA: 1s - loss: 8.6112 - acc: 0.3414
24432/25045 [============================>.] - ETA: 1s - loss: 8.6131 - acc: 0.3414
24464/25045 [============================>.] - ETA: 1s - loss: 8.6137 - acc: 0.3416
24496/25045 [============================>.] - ETA: 1s - loss: 8.6169 - acc: 0.3415
24528/25045 [============================>.] - ETA: 1s - loss: 8.6214 - acc: 0.3414
24560/25045 [============================>.] - ETA: 1s - loss: 8.6220 - acc: 0.3415
24592/25045 [============================>.] - ETA: 1s - loss: 8.6239 - acc: 0.3416
24624/25045 [============================>.] - ETA: 1s - loss: 8.6251 - acc: 0.3417
24656/25045 [============================>.] - ETA: 0s - loss: 8.6290 - acc: 0.3416
24688/25045 [============================>.] - ETA: 0s - loss: 8.6276 - acc: 0.3418
24720/25045 [============================>.] - ETA: 0s - loss: 8.6308 - acc: 0.3418
24752/25045 [============================>.] - ETA: 0s - loss: 8.6346 - acc: 0.3417
24784/25045 [============================>.] - ETA: 0s - loss: 8.6403 - acc: 0.3415
24816/25045 [============================>.] - ETA: 0s - loss: 8.6448 - acc: 0.3414
24848/25045 [============================>.] - ETA: 0s - loss: 8.6473 - acc: 0.3414
24880/25045 [============================>.] - ETA: 0s - loss: 8.6485 - acc: 0.3415
24912/25045 [============================>.] - ETA: 0s - loss: 8.6490 - acc: 0.3416
24944/25045 [============================>.] - ETA: 0s - loss: 8.6515 - acc: 0.3416
24976/25045 [============================>.] - ETA: 0s - loss: 8.6552 - acc: 0.3415
25008/25045 [============================>.] - ETA: 0s - loss: 8.6564 - acc: 0.3416
25040/25045 [============================>.] - ETA: 0s - loss: 8.6634 - acc: 0.3413
25045/25045 [==============================] - 64s 3ms/step - loss: 8.6636 - acc: 0.3413 - val_loss: 10.1578 - val_acc: 0.3698
Epoch 2/500

   16/25045 [..............................] - ETA: 1:19 - loss: 12.0886 - acc: 0.2500
   48/25045 [..............................] - ETA: 1:03 - loss: 11.0812 - acc: 0.3125
   80/25045 [..............................] - ETA: 1:01 - loss: 9.8723 - acc: 0.3875
  112/25045 [..............................] - ETA: 1:00 - loss: 10.2177 - acc: 0.3661
  144/25045 [..............................] - ETA: 59s - loss: 9.9619 - acc: 0.3819
  176/25045 [..............................] - ETA: 58s - loss: 9.8906 - acc: 0.3864
  208/25045 [..............................] - ETA: 58s - loss: 10.2288 - acc: 0.3654
  240/25045 [..............................] - ETA: 58s - loss: 10.3424 - acc: 0.3583
  272/25045 [..............................] - ETA: 57s - loss: 10.2516 - acc: 0.3640
  304/25045 [..............................] - ETA: 57s - loss: 10.2859 - acc: 0.3618
  336/25045 [..............................] - ETA: 57s - loss: 10.2177 - acc: 0.3661
  368/25045 [..............................] - ETA: 57s - loss: 10.1614 - acc: 0.3696
  400/25045 [..............................] - ETA: 57s - loss: 10.1947 - acc: 0.3675
  432/25045 [..............................] - ETA: 57s - loss: 10.2977 - acc: 0.3611
  464/25045 [..............................] - ETA: 57s - loss: 10.3517 - acc: 0.3578
  496/25045 [..............................] - ETA: 57s - loss: 10.4963 - acc: 0.3488
```

Keras： 模型视角

TensorFlow/cntk/Theano： 计算图视角

LLVM/CUDA： 原语视角

Guiding principles

User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

Modularity. A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.

Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

```
pip install keras
```

```
git clone https://github.com/keras-team/keras.git
cd keras
sudo python setup.py install
```

Keras models

the sequential model

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Embedding
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_features, output_dim=256))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))


model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=16, epochs=10)
score = model.evaluate(x_test, y_test, batch_size=16)
```

the Model class used with functional API

```
from keras.layers import Input, Dense
from keras.models import Model

inputs = Input(shape=(784,))

x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)

model.compile(optimizer='rmsprop',loss='categorical_
            crossentropy', metrics=['accuracy'])
model.fit(data, labels)
```

Application

模型信息

模型	大小	Top1准确率	Top5准确率	参数数目	深度
Xception	88MB	0.790	0.945	22,910,480	126
VGG16	528MB	0.715	0.901	138,357,544	23
VGG19	549MB	0.727	0.910	143,667,240	26
ResNet50	99MB	0.759	0.929	25,636,712	168
InceptionV3	92MB	0.788	0.944	23,851,784	159
IncetionResNetV2	215MB	0.804	0.953	55,873,736	572
MobileNet	17MB	0.665	0.871	4,253,864	88

利用 ResNet50 网络进行 ImageNet 分类

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np
model = ResNet50(weights='imagenet')
img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

Dataset

- **CIFAR10** 小图片分类数据集
- **IMDB** 影评倾向分类
- 路透社新闻主题分类
- **MNIST** 手写数字识别
- **Boston** 房屋价格回归数据集

```
from keras.datasets import cifar100
(X_train, y_train), (X_test, y_test) = cifar100.load_data(label_mode='fine')
```

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

visualization

```
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```