

A²PI² version2

第一版的文档的格式有些混乱，尤其是keras引用的地方，现在做修正，

文档主要是Python描述，涉及小部分JavaScript语言

1.imageio

1.1.imread()

读取图片并返回一个numpy数组

```
import imageio
image = imageio.imread(uri) # 文件对象或http地址，图片的路径
```

2.keras

keras在tensorflow r1.x和r2.1以及plaidml中版本均是2.2.4；除了模型初始化时的初始化器不同，不能混合续训，但是代码完全兼容，且可以混用执行预测和部署

2.1.applications

2.1.1.inception_resnet_v2

2.1.1.1.InceptionResNetV2()

InceptionResNetV2的预训练模型

```
from keras.applications.inception_resnet_v2 import InceptionResNetV2
model = InceptionResNetV2(include_top, # 是否包含全连接的输出层
                           weights, # 权重，可以是随机初始化，也可以加载'imagenet'的权重，或者自定义权重的路径
                           input_tensor) # 输入层，需要使用keras.layers.Input()
```

2.1.2.inception_v3

2.1.2.1.InceptionV3()

InceptionV3的预训练模型

```
from keras.applications.inception_v3 import InceptionV3
model = InceptionV3(include_top,# 是否包含全连接的输出层
                    weights,# 权重, 可以是随机初始化, 也可以加载'imagenet'的权重, 或者
                    自定权重的路径
                    input_tensor)# 输入层, 需要使用keras.layers.Input()
```

2.1.3.resnet_v2

2.1.3.1.ResNet152V2()

ResNet152V2的预训练模型

```
from keras.applications.resnet_v2 import ResNet152V2
model = ResNet152V2(include_top,# 是否包含全连接的输出层
                    weights,# 权重, 可以是随机初始化, 也可以加载'imagenet'的权重, 或者
                    自定权重的路径
                    input_tensor)# 输入层, 需要使用keras.layers.Input()
```

2.1.4.resnet50

2.1.4.1.ResNet50()

ResNet50的预训练模型

```
from keras.applications.resnet50 import ResNet50
model = ResNet50(include_top,# 是否包含全连接的输出层
                 weights,# 权重, 可以是随机初始化, 也可以加载'imagenet'的权重, 或者自定
                 权重的路径
                 input_tensor)# 输入层, 需要使用keras.layers.Input()
```

2.1.5.vgg19

2.1.5.1.preprocess_input()

对数据进行预处理

```
from tensorflow.keras.applications.vgg19 import preprocess_input
preprocessed_input = preprocess_input(x)# 要预处理的数据
```

2.1.5.2.VGG19()

VGG19的预训练模型

```
from keras.applications.vgg19 import VGG19
model = VGG19(include_top,# 是否包含全连接的输出层
               weights,# 权重, 可以是随机初始化, 也可以加载'imagenet'的权重, 或者自定义权重
               input_tensor)# 输入层, 需要使用keras.layers.Input()
```

2.1.6.xception

2.1.6.1.Xception()

Xception的预训练模型

```
from keras.applications.xception import Xception
model = Xception(include_top,# 是否包含全连接的输出层
                  weights,# 权重, 可以是随机初始化, 也可以加载'imagenet'的权重, 或者自定义权重
                  input_tensor)# 输入层, 需要使用keras.layers.Input()
```

2.2.backend

2.2.1.cast()

转换张量的类型

```
from keras.backend import cast
tensor = cast(x,# 张量
              dtype)# 转换后的类型
```

2.2.2.clip()

将变量的值裁切到某个区间（类似标准化）

```
from keras.backend import clip
tensor = clip(x,# 张量
              min_value,# 最小值
              max_value)# 最大值
```

2.2.3.ones_like()

创建一个全1的张量

```
from keras.backend import ones_like
tensor = ones_like(x)# 张量
```

2.2.4.shape()

返回张量的形状

```
from keras.backend import shape
shape = shape(x)# 张量
```

2.2.5.zeros_like()

创建一个全0的张量

```
from keras.backend import zeros_like
tensor = zeros_like(x)# 张量
```

2.3.callbacks

2.3.1.ModelCheckpoint()

保存模型（主要是断点）

```
from keras.callbacks import ModelCheckpoint
callbacks = [ModelCheckpoint(filepath,# 模型保存的路径
                             period)]# 保存间隔
```

2.4.datasets

2.4.1.mnist

keras自带的数据集之一

2.4.1.1.load_data()

加载mnist数据集

```
from keras import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

2.5.layers

keras提供Model（函数式API）和Sequential（普通API），如果函数名相同则不赘述

2.5.1.Add()

将多个层通过矩阵加法合并

```
from keras.layers import Add
layer = Add()(_Merge)# 相同形状的张量（层）列表
```

2.5.2.BatchNormalization()

批量标准化层

```
from keras.layers import BatchNormalization
layer = BatchNormalization()
```

2.5.3.Concatenate()

连接输入的层

```
from keras.layers import Concatenate
layer = Concatenate(axis)(_Merge)
"""
axis 维度
_Merge 张量（层）列表
"""
```

2.5.4.Conv1D()

1D卷积层（例如时序卷积）

```
from keras.layers import Conv1D
layer = Conv1D(filters,# 整数, 卷积核的数量
               kernel_size,# 整数, 卷积核的大小
               strides,# 整数、或者列表或者元组, 滑动步长
               padding,# 'same','valid','causal', 是否使用全零填充
               data_format,# 'channels_first','channels_last' 数据格式CBHW或BHWC
               activation,# 'relu','sigmoid' 激活函数
               use_bias,# 布尔值 是否使用偏置
               kernel_initializer,# 权值初始化
               bias_initializer)# 偏置项初始化
```

2.5.5.Conv2D()

2D卷积层（例如时序卷积）

```
from keras.layers import Conv2D
layer = Conv2D(filters,# 整数, 卷积核的数量
               kernel_size,# 整数, 卷积核的大小
               strides,# 整数、或者列表或者元组, 滑动步长
               padding,# 'same','valid', 是否使用全零填充
               input_shape)# 元组, 第一层需要指定输入
```

2.5.6.Conv2DTranspose()

转置卷积层 (有时被成为反卷积), 将具有卷积输出尺寸的东西 转换为具有卷积输入尺寸的东西

```
from keras.layers import Conv2DTranspose
layer = Conv2DTranspose(filters,# 整数, 输出空间的维度
                        kernel_size,# 整数, 卷积核的大小
                        strides,# 整数、或者列表或者元组, 滑动步长
                        padding,# 'same' 'valid', 是否使用全零填充
                        use_bias)# 布尔值 是否使用偏置
```

2.5.7.Dense()

全连接层

```
from keras.layers import Dense
layer = Dense(units,# 整数, 神经元数量
              use_bias,# 布尔值 是否使用偏置
              input_shape)# 元组, 第一层需要指定输入
```

2.5.8.Dropout()

在训练阶段按照比例随机丢弃神经元

```
from keras.layers import Dropout
layer = Dropout(rate)# 丢弃率
```

2.5.9.Flatten()

将输入展平, 不影响批次大小

```
from keras.layers import Flatten
layer = Flatten()
```

2.5.10.Input()

输入层

```
from keras.layers import Input
input_tensor = Input(shape)# 整数, 形状元组
```

2.5.11.LeakyReLU()

带泄漏的 ReLU 层

```
from keras.layers import LeakyReLU
layer = LeakyReLU(alpha)# 负斜率系数, 默认为0.3
```

2.5.12.MaxPooling1D()

对时序数据进行最大池化

```
from keras.layers import MaxPooling1D
layer = MaxPooling1D(pool_size,# 整数, 池化核数量
                      strides,# 整数、或者列表或者元组, 滑动步长
                      padding)# 'same' 'valid' 'causal', 是否使用全零填充
```

2.5.13.Reshape()

将输入重新调整为特定的尺寸

```
from keras.layers import Reshape
layer = Reshape(target_shape)# 整数元组, 目标尺寸
```

2.6.losses

2.6.1.BinaryCrossentropy()

计算真实标签和预测值标签的交叉熵损失（二分类）

```
from keras.losses import BinaryCrossentropy
cross_entropy = BinaryCrossentropy(from_logits)# 是否将y_pred解释为张量
```

2.6.2.SparseCategoricalCrossentropy()

计算真实标签和预测值标签的交叉熵损失（多分类）

```
from keras.losses import SparseCategoricalCrossentropy
cross_entropy = SparseCategoricalCrossentropy(from_logits)# 是否将y_pred解释为张量
```

2.7.models

2.7.1.Model()

keras自定义模型对象

```
from keras.models import Model
model = Model(inputs,# 输入层
               outputs)# 输出层
```

2.7.1.1.fit_generator()

生成批次训练数据，按批训练数据

```
model.fit_generator(generator,# 数据生成器, 比如ImageDataGenerator()  
                    steps_per_epoch,# 整数, 每批次步数  
                    epochs,# 整数, 轮数  
                    verbose)# 日志显示模式 0=安静模型 1=进度条 2每轮显示
```

2.7.1.2.load_model()

加载模型

```
from keras.models import load_model  
model = load_model(filepath)# 文件路径, 可以是saved model或者h5py
```

2.7.1.3.save()

将模型保存为SavedModel或者HDF5文件

```
model.save(filepath,# 保存路径  
            save_format)# 保存格式, 默认是h5, 可选tf
```

2.7.2.Sequential()

构建一个线性堆叠的网络模型

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

2.7.2.1.add()

将一个具体的单层神经网络加入模型

```
model.add(layer)
```

2.7.2.2.compile()

用于配置训练模型

```
model.compile(optimizer,# 优化器  
              loss,# 损失函数  
              metrics)# 评估标准['accuracy']
```

2.7.2.3.evaluate()

在测试模式下返回损失值和准确率


```
model.evaluate(x,# 训练数据
               y,# 标签
               batch_size,# 整数, 批次大小, 默认32
               verbose)# 日志显示模式 0=安静模型 1=进度条 2每轮显示
```

2.7.2.4.fit()

以给定批次训练模型

```
history = model.fit(x,# 训练数据, 包括数据生成器, 比如ImageDataGenerator(), 使用生成器, 不需要参数y
                    y,# 标签
                    batch_size,# 整数, 批次大小, 默认32
                    epochs,# 整数, 轮数
                    verbose,# 日志显示模式 0=安静模型 1=进度条 2每轮显示
                    callbacks,# 回调函数
                    validation_split,# 浮点数, 验证集可以从训练集中划分
                    validation_data,# 元组(x,y) 验证集数据可以直接指定, 会直接覆盖 validation_split
                    shuffle)# 布尔值, 打乱数据
```

2.7.2.5.load_weights()

加载所有的神经网络层的参数

```
model.load_weights(filepath)# 检查点文件路径
```

2.7.2.6.predict()

生成预测结果

```
model.predict(x,# 测试数据
              batch_size,# 整数, 批次大小, 默认32
              verbose)# 日志显示模式 0=安静模型 1=进度条
```

2.7.2.7.output_shape()

返回模型的输出层形状

```
print(model.output_shape())
```

2.7.2.8.summary()

查看模型各层参数

```
model.summary()
```

2.8.optimizers

2.8.1.Adam()

Adam优化器

```
from keras.optimizers import Adam
optimizer = Adam(lr)# 学习率
```

2.8.2.apply_gradients()

将梯度带计算出的梯度赋给优化器

```
optimizer.apply_gradients(grads_and_vars)# 对应梯度和变量组成的列表
```

2.9.preprocessing

2.9.1.image

2.9.1.1.ImageDataGenerator()

对图片数据进行实时的数据增强类

```
from keras.preprocessing.image import ImageDataGenerator
data_generator = ImageDataGenerator(rotation_range,# 整数, 随机旋转度数
                                    width_shift_range,# 浮点数, 水平偏移范围
                                    height_shift_range,# 浮点数, 垂直偏移范围
                                    shear_range,# 浮点数(角度), 裁切角范围
                                    zoom_range,# 浮点数, 随机缩放倍数
                                    channel_shift_range,# 浮点数, 随机色彩通道移位
                                    fill_mode, # 填充模式, 'constant','nearest','reflect','wrap'
                                    horizontal_flip)# 布尔值, 水平随机翻转
```

2.9.1.1.1.flow_from_dataframe()

从给定的dataframe标签读入数据并增强

```
data_generator.flow_from_dataframe(dataframe,# dataframe标签
                                directory,# 图片文件路径
                                x_col,# dataframe中文件路径列
                                y_col,# dataframe中文件标签列
                                target_size,# 元组, 调整后大小
                                interpolation,# 插值, 调整尺寸

'nearest','bilinear','bicubic'

                                class_mode,# 返回标签数组类型, 默
认'categorical'

                                classes,# 标签名列表, 如果为None, 则自动根据
y_col生成

                                shuffle,# 布尔值, 打乱顺序
                                batch_size,# 整数, 批次大小, 默认32
                                validate_filenames)# 验证文件是否有效, 默认True
```

2.9.1.1.2.flow_from_directory()

从给定路径读入数据并增强（要求每个类别必须单独一个文件夹）

```
data_generator.flow_from_directory(directory,# 路径
                                target_size,# 元组, 调整后大小
                                classes,# 标签名列表, 如果为None, 则自动生成
                                class_mode,# 返回标签数组类型, 默

认'categorical'

                                batch_size,# 整数, 批次大小, 默认32
                                shuffle,# 布尔值, 打乱顺序
                                interpolation)# 插值, 调整尺寸

'nearest','bilinear','bicubic'
```

2.9.1.1.2.1.class_indices

返回训练数据的索引

```
from keras.preprocessing.image import ImageDataGenerator
batches = ImageDataGenerator().flow_from_directory()
print(batches.class_indices)
```

2.9.1.2.load_image()

按照PIL的格式加载图像

```
from keras.preprocessing import image
img = image.load_img(path,# 路径
                    target_size)# 元组, 调整后大小
```

2.10.utils

2.10.1.multi_gpu_model()

多GPU并行训练模型

```
from keras.utils import multi_gpu_model
parallel_model = multi_gpu_model(model, # 模型
                                gpus) # 整数（大于等于2），并行GPU数量
```

2.10.2.plot_model()

保存keras模型成图片

```
from keras.utils import plot_model
plot_model(model, # 模型
           to_file, # 保存路径
           show_shapes, # 是否显示每层的shape
           show_layer_names, # 是否显示每层的名称
           rankdir, # 绘图方向，垂直'TB'水平'LR'
           dpi) # 每英寸点数
```

3.PIL

3.1.Image

3.1.1.fromarray()

从输入的图片返回一个数组

```
from PIL import Image
array = Image.fromarray(obj) # 图片对象
```

3.1.2.resize()

返回调整大小后的图像的副本

```
from PIL import Image
new_image = image.resize(size) # 有宽度和高度的二元组
```

4.sklearn

4.1.linear_model

4.1.1.LogisticRegression()

构建一个对数几率回归模型

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

4.1.1.1.fit()

以给定数据训练模型

```
model.fit(X,# 训练数据
          y,# 标签
          sample_weight)# 数组, 类别权重, 默认为None
```

4.1.1.2.predict()

生成预测结果

```
model.predict(X)# 测试数据
```

4.2.metrics

4.2.1.accuracy_score()

计算准确率

```
import sklearn
sklearn.metrics.accuracy_score(y_true,# 真实标签
                               y_pred,# 预测结果
                               sample_weight)# 数组, 类别权重, 默认为None
```

4.3.model_selection

4.3.1.cross_val_predict()

使用交叉验证法验证

```
import sklearn
sklearn.model_selection.cross_val_predict(estimator,# 训练的模型对象
                                          X,# 训练数据
                                          y,# 标签
                                          cv)# 整数, 划分数, 默认为3
```

4.3.2.LeaveOneOut()

使用留一法验证

```
import sklearn
LOO = sklearn.model_selection.LeaveOneOut() # 返回一个BaseCrossValidator对象
```

4.3.2.1.split()

按照具体BaseCrossValidator对象将数据划分为训练和测试集

```
LOO.split(X) # 训练数据
```

4.4.tree

4.4.1.DecisionTreeClassifier()

生成一个分类决策树实例

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(criterion, # 划分条件 'gini' 'entropy'
                             random_state)
# 随机状态 默认是None (通过np.random自动生成), 也可以
# 赋一个整数
```

4.4.1.1.fit()

以给定数据训练模型

```
tree.fit(X, # 训练数据
         y) # 标签
```

4.4.2.export_graphviz()

以dot文件导出决策树

```
from sklearn.tree import export_graphviz
dot_data = export_graphviz(decision_tree, # 决策树
                           out_file, # 生成dot文件 默认为None (返回str)
                           feature_names, # 属性名称
                           class_names) # 分类名称
```

5.tensorflow r1.x

tensorflow r1.x将逐步停止支持, 可能只有为了支持tensorflow.js@0.x会用到

5.1.concats()

按某个维度连接多个张量

```
import tensorflow as tf
tensor_a = [[1, 2], [3, 4]]
tensor_b = [[5, 6], [7, 8]]
tensor_c = tf.concat(value=[tensor_a, tensor_b], axis=1, name="concat")
"""
value待合并的张量, axis按某个维度合并, name张量的名字
"""
```

5.2.ConfigProto()

用于配置会话的选项

```
import tensorflow as tf
tf.ConfigProto(gpu_option)# 配置显存
```

5.3.gfile

5.3.1.FastGFile()

没有线程锁的文件I/O封装器

```
import tensorflow as tf
fp = tf.gfile.FastGfile(name,# 保存名称
                        mode)# 模式, 默认'r'
```

5.4.global_variables()

返回默认会话中所有的全局变量

```
import tensorflow as tf
var = tf.global_variables()
```

5.5.global_variables_initializer()

初始化全局的变量

```
import tensorflow as tf
init = tf.global_variables_initializer()
```

5.6.GPUOptions()

使用GPU时对显存使用的控制选项

```
import tensorflow as tf
gpu_options = tf.GPUOptions(allow_growth)# 布尔值, 设置为True时, 显存按需分配
(tensorflow默认是占用全部显存)
```

5.7.GraphDef()

tensorflow的信息协议

```
import tensorflow as tf
graph_def = tf.GraphDef()
```

5.7.1.ParseFromString()

将序列化的数据转换成GraphDef

```
graph_def.ParseFromString(serialized)# 序列化数据, 数据流
```

5.8.import_graph_def()

将GraphDef实例导入默认的Graph计算图

```
import tensorflow as tf
tensor = tf.import_graph_def(graph_def,# graph_def实例
                             input_map,# 字典, 输入张量的名称和对应的张量
                             return_elements)# 字符串列表, 输出张量的名称
```

5.9.nn

5.9.1.avg_pool()

均值池化层

```
import tensorflow as tf
tf.nn.avg_pool(value,# 输入张量
               ksize,# 整数, 池化核数量
               strides,# 整数、或者列表或者元组, 滑动步长
               padding,# 'SAME' 'VALID', 是否使用全零填充
               data_format="NHWC",# 数据格式, 默认"NHWC"
               name)# 名称
```

5.9.2.dropout()

在训练阶段按照比例随机丢弃神经元


```
import tensorflow as tf
tf.nn.dropout(x, # 输入张量
              keep_prob, # 保留概率
              name) # 整数
```

5.9.3.lrn()

局部响应归一化层(Local Response Normalization)

$$b_{x,y}^i = \frac{a_{x,y}^i}{\left(k + \alpha \sum_{j=\max(0, i-\frac{n}{2})}^{\min(N-1, i+\frac{n}{2})} (a_{x,y}^j)^2\right)^\beta}$$

其中 $a_{x,y}^i$ 是input, $\frac{n}{2}$ 是depth_radius, k 是bias, α 是alpha, β 是beta

AlexNet使用的一种类似Dropout的减少过拟合方法, 不改变size; 个人认为是一种类似于池化但不改变size大小的方法, 在采样半径下, 输入对输出的和的标准化

```
import tensorflow as tf
tf.nn.lrn(input, # 输入张量
          depth_radius, # 采样半径, 默认5
          bias, # 超参数, 默认1
          alpha, # 超参数, 默认1
          beta, # 超参数, 默认0.5
          name) # 名称
```

5.9.4.max_pool()

最大池化层

```
import tensorflow as tf
tf.nn.max_pool(value, # 输入张量
               ksize, # 整数, 池化核数量
               strides, # 整数、或者列表或者元组, 滑动步长
               padding, # 'SAME' 'VALID', 是否使用全零填充
               data_format="NHWC", # 数据格式, 默认"NHWC"
               name) # 名称
```

5.9.5.softmax()

softmax激活函数

```
import tensorflow as tf
tf.nn.softmax(logits) # 输入张量 (非空)
```

5.10.placeholder()

添加一个占位符

```
import tensorflow as tf
x = tf.placeholder(dtype, # 自变量数据类型
                  shape, # 张量形状
                  name) # 名称
```

5.11.python

5.11.1.framework

5.11.1.1.graph_util

5.11.1.1.1.convert_variables_to_constants

将计算图中的变量转换为常量

```
from tensorflow.python.framework.graph_util import
convert_variables_to_constants
output_graph = convert_variables_to_constants(sess, # 需要转换变量的会话
                                             input_graph_def, # 会话的graph_def
                                             对象
                                             output_node_names) # 字符串列表，输出
                                             层的名称
```

5.11.1.1.1.1.SerializeToString()

将protobuf数据转换为二进制字符串

```
# GraphDef就是一种protobuf
output_graph.SerializeToString()
```

5.12.saved_model

5.12.1.builder

5.12.1.1.SavedModelBuilder()

构建一个生成SavedModel的实例

```
from tensorflow.saved_model import builder
builder = builder.SavedModelBuilder(export_dir) # SavedModel的保存路径
```

5.12.1.1.1.add_meta_graph_and_variables()

添加图结构和变量信息

```
builder.add_meta_graph_and_variables(sess,# 会话
                                     tags,# 标签, 自定义
                                     signature_def_map)# 预测签名字典
```

5.12.1.1.2.save()

将SaveModel写入磁盘

```
builder.save()
```

5.12.2.loader

5.12.2.1.load()

从标签指定的SavedModel加载模型

```
import tensorflow as tf
tf.saved_model.loader.load(sess,# 模型还原到的会话
                           tags,# 字符串, 标签, 参见tf.saved_model.tag_constants
                           export_dir)# 待还原的SavedModel目录
```

5.12.3.signature_def_utils

5.12.3.1.predict_signature_def()

构建预测签名

```
from tensorflow.saved_model import signature_def_utils
signature = signature_def_utils.predict_signature_def(inputs,# 字典, 输入变量
                                                      outputs)# 字典, 输出变量
```

5.12.4.simple_save()

使用简单方法构建SavedModel用于服务器

```
from tensorflow.saved_model import simple_save
simple_save(session,# 会话
            export_dir,# SavedModel的保存目录
            inputs,# 字典, 输入变量
            outputs)# 字典, 输出变量
```

5.12.5.tag_constants

SaveModel的标签

```
from tensorflow.saved_model import tag_constants
tags = tag_constants.SERVING
"""
标签有TPU SERVING GPU TRAINING
"""
```

5.13.Session()

生成一个tensorflow的会话

```
import tensorflow as tf
sess = tf.Session(config)# 使用ConfigProto配置会话
```

5.13.1.close()

关闭当前会话

```
sess.close()
```

5.13.2.graph

5.13.2.1.get_tensor_by_name()

根据名称返回张量，可以使用多个线程同时调用

```
sess.graph.get_tensor_by_name(name)# 张量的名称
```

5.13.3.run()

运行传入会话的操作，返回结果张量

```
sess.run(fetches,# 待计算的操作 'Operation' 'Tensor'
          feed_dict# 输入的值，默认为None)
```

5.14.split()

将张量按某个维度拆分成多个张量

```
import tensorflow as tf
tensor = [[1, 2, 5, 6], [3, 4, 7, 8]]
tensor_list = tf.split(value=tensor, num_or_size_splits=2, axis=1,
name="split")
"""
value需要拆分的张量, num_or_size_splits要拆分的数量, axis按某个维度拆分, name张量的名字
"""
```

5.15.train

5.15.1.AdamOptimizer()

Adam优化器

```
import tensorflow as tf
optimizer = tf.train.AdamOptimizer(learning_rate)# 学习率
```

5.15.2.GradientDescentOptimizer()

梯度下降优化器

```
import tensorflow as tf
optimizer = tf.train.GradientDescentOptimizer(learning_rate)# 学习率
```

5.15.3.latest_checkpoint()

查找最近的保存点文件

```
import tensorflow as tf
checkpoint = tf.train.latest_checkpoint(checkpoint_dir)# 保存点路径
```

5.15.4.Saver()

生成用于保存和还原计算图的实例

```
import tensorflow as tf
saver = tf.train.Saver(var_list)
# 将被保存和恢复的变量列表或者变量字典，默认为None（保存全部）
```

5.15.4.1.restore()

恢复保存的变量

```
saver.restore(sess,# 会话, eager模式为None
               save_path)# 检查点文件的路径
```

5.16.variable_scope()

用于定义变量操作的上下文管理器

```
import tensorflow as tf
with tf.variable_scope(name_or_scope):# 字符串，作用域
```

6.tensorflow r2.x

6.1.config

6.1.1.experimental

6.1.1.1.list_physical_devices()

返回主机运行时的可见的物理设备列表

```
import tensorflow as tf
devices_list = tf.config.experimental.list_physical_devices(device_type)# 设备类型, 可选CPU或GPU
```

6.1.1.2.set_memory_growth()

设置物理设备的内存的按需增长

```
import tensorflow as tf
tf.config.experimental.set_memory_growth(device,# 物理设备
                                         enable)# 是否运行内存按需增长
```

6.2.data

6.2.1.Datasets

6.2.1.1.batch()

给数据集划分批次

```
import tensorflow as tf
dataset = tf.data.Dataset.range(6)
dataset_shuffle = dataset.batch(batch_size=3)# 批次的大小
```

6.2.1.2.from_tensor_slices()

返回一个数据集对象

```
import tensorflow as tf
dataset = tf.data.Dataset.from_tensor_slices(tensor)# 张量, 必须有相同的第一维
```

6.2.1.3.shuffle()

随机打乱数据集对象的元素

```
import tensorflow as tf
dataset = tf.data.Dataset.range(3)
dataset_shuffle = dataset.shuffle(buffer_size=3)# 数据集元素的数量
```

6.3.einsum()

爱因斯坦求和约定，返回根据等式定义的张量

```
from tensorflow import einsum
result = einsum(equation,# 规则等式 乘号用',' 等号用'-'>'
               *inputs)# 待求和的张量
```

6.4.GradientTape()

创建一个梯度带，可用于自动求导

```
import tensorflow as tf
with tf.GradientTape() as tape:
    # 在tf.GradientTape()的上下文管理器内的计算都将被用于求导
```

6.4.1.gradient()

在梯度带内计算导数

```
grad = tape.gradient(target, sources)# 计算target关于sources的导数
```

6.5.image

6.5.1.convert_image_dtype()

转换图片的数据类型，并根据实际缩放

```
import tensorflow as tf
img_tensor = tf.image.convert_image_dtype(image,# 图片的张量
                                          dtype)# 转换后的张量类型
```

6.5.2.decode_image()

将BMP、GIF、JPEG或PNG的编码字节流转换为张量

```
import tensorflow as tf
img_tensor = tf.image.decode_image(contents,# 图片的编码字节流
                                   channels,# 色彩通道数
                                   dtype)# 转换后的张量类型
```

6.5.3.resize()

按照指定方法调整图片的大小

```
import tensorflow as tf
img_tensor = tf.image.resize(images,# 输入图片
                             size)# 调整后的尺寸
```

6.6.io

6.6.1.read_file()

读入文件

```
import tensorflow as tf
img = tf.io.read_file(filename)# 文件的路径, 返回图片的编码字符流
```

6.7.ones_like()

创建一个全1的张量

```
import tensorflow as tf
tensor = tf.ones_like(input)# 张量
```

6.8.random

6.8.1.normal()

生成一个正态分布的张量

```
import tensorflow as tf
tensor = tf.random.normal(shape)# 张量的形状
```

6.9.zeros_like()

创建一个全0的张量

```
import tensorflow as tf
tensor = tf.zeros_like(input)# 张量
```

7.tensorflow_hub 0.8.0

7.1.KerasLayer()

将一个SavedModel模型修饰成keras的一个层

```
import tensorflow_hub as hub
efficient_layer = hub.KerasLayer(handle, # tensorflow_hub模型的路径
                                trainable)# 是否可训练
```

8.tensorflow.js@0.x

tensorflow.js@0.x版本必须和tensorflow r1.x生成的SavedModel模型配套，不兼容tensorflow r2.x

8.1.dispose()

手动释放显存，推荐使用tf.tidy()

```
import * as tf from '@tensorflow/tfjs';
const t = tf.tensor([1, 2]);
t.dispose();
```

8.2.fromPixels()

从一张图像创建一个三维张量

```
import * as tf from '@tensorflow/tfjs';
const image = tf.fromPixels(pixels, numChannels);
/*
pixels 输入图像
numChannels 输入图像的通道数（可选）
*/
```

8.3.image

8.3.1.resizeBilinear()

使用双线性法改变图片的尺寸

```
import * as tf from '@tensorflow/tfjs';
const resized = tf.image.resizeBilinear(images, size, alignCorners);
/*
images 输入图像的张量 tf.Tensor3D|tf.Tensor4D|TypedArray|Array
size 改变后尺寸 [number, number]
alignCorners 布尔值，对齐角落（可选）
*/
```

8.4.layers

8.4.1.dense()

全连接层

```
tf.layers.dense({units, activation, inputShape});  
/*  
units 整数, 神经元数量  
activation 激活函数 relu|'sigmoid'|'softmax'|'tanh'  
inputShape 此参数只在模型第一层使用  
*/
```

8.5.loadFrozenModel()

通过url加载固化的模型（异步执行）

```
import * as tf from '@tensorflow/tfjs';  
var cmodel;  
tf.loadFrozenModel(modelUrl, weightsManifestUrl).then((model) => {cmodel =  
model;});  
/*  
modelUrl pb模型的url  
weightsManifestUrl json权重的url（可选）  
*/
```

8.6.scalar()

创建一个标量（tf.tensor()可替代）

```
import * as tf from '@tensorflow/tfjs';  
const s = tf.scalar(value, dtype);  
/*  
value 标量的值 number|boolean|string|Uint8Array  
dtype 数据类型（可选） 'float32'|'int32'|'bool'|'complex64'|'string'  
*/
```

8.7.sequential()

构建一个线性堆叠的网络模型，模型拓扑是简单的层“堆栈”，没有分支或跳过。

```
import * as tf from '@tensorflow/tfjs';  
const model = tf.sequential(tf.layers.dense({}));
```

8.7.1.add()

将一个具体的单层神经网络加入模型

```
// 全连接层
model.add(tf.layers.dense({units, activation, inputShape}));
```

8.7.2.compile()

用于配置训练模型

```
model.compile(args); // args 配置参数包括optimizer、loss、metrics
```

8.7.3.fit()

以给定批次训练模型

```
model.fit(x, y, args);
/*
x 训练数据
y 标签
args(可选) batchSize 批次大小, 默认32
           epochs 训练轮数
           verbose 日志显示模式 0=安静模型 1=进度条 2=每轮显示, 默认1
           callbacks 回调
           validationSplit 浮点数, 验证集可以从训练集中划分
           validationData 元组[x,y] 验证集数据可以直接指定, 会直接覆盖validationSplit
*/
```

8.7.4.predict()

生成预测结果

```
model.predict(x); // 测试数据, 需要是张量或者张量数组
```

8.7.5.summary()

查看模型的全局参数

```
model.summary();
```

8.8.ones()

创建一个元素值全为一的张量

```
import * as tf from '@tensorflow/tfjs';
const t = tf.ones(shape, dtype);
/*
shape 张量的形状
dtype 数据类型 (可选) 'float32' | 'int32' | 'bool' | 'complex64' | 'string'
*/
```

8.9.tensor()

创建一个张量，注意张量的值一经创建不可改变

```
import * as tf from '@tensorflow/tfjs';
const t = tf.tensor(values, shape, dtype);
/*
values 张量的值
shape 张量的形状 (可选)
dtype 数据类型 (可选) 'float32' | 'int32' | 'bool' | 'complex64' | 'string'
*/
```

8.9.1.dataSync()

同步数据，此时阻塞线程直到同步完成

```
const new_t = t.dataSync();
```

8.9.2.expandDims()

增加张量的维度

```
t.expandDims(axis); // 维度 (可选)
```

8.9.3.toFloat()

将张量的数据类型转换为float32

```
t.toFloat();
```

8.10.tensor1d()

创建一个一维张量 (tf.tensor()可替代)

```
import * as tf from '@tensorflow/tfjs';
const t = tf.tensor1d(values, dtype);
/*
values 张量的值
dtype 数据类型 (可选) 'float32' | 'int32' | 'bool' | 'complex64' | 'string'
*/
```

8.11.tensor2d()

创建一个二维张量 (tf.tensor()可替代)

```
import * as tf from '@tensorflow/tfjs';
const t = tf.tensor2d(values, dtype);
/*
values 张量的值
dtype 数据类型 (可选) 'float32' | 'int32' | 'bool' | 'complex64' | 'string'
*/
```

8.12.tidy()

执行传入的函数后，自动清除除返回值以外的系统分配的所有的中间张量，防止内存泄露

```
import * as tf from '@tensorflow/tfjs';
const result = tf.tidy(fn); // 传入一个箭头函数
```

8.13.train

8.13.1adam()

Adam优化器

```
import * as tf from '@tensorflow/tfjs';
optimizer = tf.train.adam(learningRate); // 学习率
```

8.14.variable()

创建一个变量

```
import * as tf from '@tensorflow/tfjs';
const v = tf.variable(initialValue, trainable, name, dtype);
/*
initialValue 初始值, 必须是一个tf.Tensor
trainable 可训练的 (可选) 'bool'
name 名称 (可选)
dtype 数据类型 (可选) 'float32' | 'int32' | 'bool' | 'complex64' | 'string'
*/
```

8.14.1.assign()

给变量赋予新值

```
v.assign(newValue); // newValue 新值, 必须是一个tf.Tensor
```

8.14.2.print()

输出变量的值在控制台

```
v.print();
```

8.15.zeros()

创建一个元素值全为零的张量

```
import * as tf from '@tensorflow/tfjs';
const t = tf.zeros(shape, dtype);
/*
shape 张量的形状
dtype 数据类型 (可选) 'float32' | 'int32' | 'bool' | 'complex64' | 'string'
*/
```

9.matplotlib

9.1.pyplot

9.1.1.plot()

绘制函数

```
import matplotlib
matplotlib.pyplot.plot(x, # 自变量的值
                       y) # 因变量的值
```

9.1.2.scatter()

绘制散点图

```
import matplotlib
matplotlib.pyplot.scatter(x,# x轴数据
                           y)# y轴数据
```

9.1.3.show()

显示图像

10.numpy

10.1.argmax()

返回最大值的索引

```
import numpy as np
a = [1, 2, 3]
max = np.argmax(a)# 输入可以是lists, tuples, ndarrays
```

10.2.asarray()

将输入转化为ndarray

```
import numpy as np
a = [[1, 2, 3]]
a = np.asarray(a,# 输入可以是lists, tuples, ndarrays
                dtype)# 数据类型, 可选
```

10.3.astype()

强制转换成新的数据类型

```
import numpy as np
a = [1.0, 2.0]
new_a = a.astype(dtype)# 数据类型
```

10.4.expand_dims()

增加ndarray的维度

```
import numpy as np
a = [[1, 2], [3, 4]]
a = np.expand_dims(a, # 输入可以是lists, tuples, ndarrays
                    axis)# 维度
```

10.5.linalg

10.5.1.norm()

计算矩阵或者向量范数

```
import numpy as np
x = [[1, 2], [3, 4]]
norm = np.linalg.norm(x, # 输入矩阵或向量, 维数必须是1-D或2-D
                      ord)# 范数选项, 默认None, 计算2范数
```

10.6.linspace()

生成一个等差数列

```
import numpy as np
a = np.linspace(start, # 序列的起始值
                stop, # 序列的结束值
                num) # 生成样本数, 默认50
```

10.7.load()

从npy或者npz文件中加载数组

```
import numpy as np
np.load(file, # 文件路径
        allow_pickle, # 使用pickle, 默认False
        encoding) # 编码格式, 默认ASCII
```

10.8.mat()

从列表或者数组生成一个矩阵对象

```
import numpy as np
a = [[1, 2, 3]]
a = np.mat(a)
```

10.9.matmul()

矩阵乘法

```
import numpy as np
a1 = [[1, 2, 3]]
a2 = [[1], [2], [3]]
a = np.matmul(a1, a2)
```

10.10.mean()

按照指定的维度计算算术平均值

```
import numpy as np
np.mean(a, # 待计算均值的列表、矩阵
        axis)# 维度
```

10.11.transpose()

对矩阵进行转置

```
import numpy as np
a = [[1, 2], [3, 4]]
a_t = np.transpose(a)
```

10.12.random

10.12.1.normal()

生成一个正态分布的数组

```
import numpy as np
a = np.random.normal(size=[2,3])# 形状
```

10.13.reshape()

在不改变数据内容的情况下，改变数据形状

```
import numpy as np
a = [1, 2, 3, 4]
a = np.asarray(a)
a.reshape((2, 2))# 将a转换成2行2列的二维数组
b = [[1, 2], [3, 4]]
b = np.asarray(b)
b = b.reshape((-1, 2, 1))# 第一个为-1，将按照后面的输入增加一个维度
```

10.14.split()

将张量按某个维度拆分成多个张量

```
import numpy as np
tensor = np.asarray([[1, 2, 5, 6], [3, 4, 7, 8]])
tensor_list = np.split(ary=tensor, # 需要拆分的张量
                       indices_or_sections=2, # 要拆分的数量
                       axis=1) # axis按某个维度拆分
```

11.pandas

11.1.DataFrame()

将其他数据格式转换为DataFrame

```
import pandas as pd
df = {'index': [0, 1, 2], 'value': [1, 2, 3]}
df = pd.DataFrame(df)
```

11.1.1.astype()

转换DataFrame到指定类型

```
df['value'] = df['value'].astype(dtype) # 转换后的数据类型
```

11.1.2.replace()

新值替换旧值

```
df.replace(to_replace, # 旧值
           value, # 新值
           inplace) # 布尔值，默认False，修改源文件
df.replace(dict) # 可以是字典的键为旧值，值为新值，进行替换
```

11.2.read_csv()

读取csv文件，返回一个DataFrame对象

```
import pandas as pd
df = pd.read_csv(filepath_or_buffer, # 文件或者缓冲区路径
                 header, # 列名，默认是0，否则是None
                 index_col) # 指定索引列，默认是None
```

11.3.Series()

11.3.1.values()

将DataFrame的值转换为ndarray

```
import pandas as pd
df = {'index': [0, 1, 2], 'value': [1, 2, 3]}
df = pd.DataFrame(df)
array = df['index'].values
```

11.4.to_csv()

将DataFrame生成csv文件

```
import pandas as pd
df.to_csv(path_or_buf, # 保存的文件和路径
          header, # 列名, 默认是True
          index, # 索引, 默认是True
          encoding) # 编码方式, 默认是'utf-8'
```

12.pydot

12.1.graph_from_dot_data()

从dot字符串中加载图像

```
import pydot
graph = pydot.graph_from_dot_data(s) # dot字符串, 返回一个list
```

12.2.graph_from_dot_file()

从dot文件中加载图像

```
import pydot
graph = pydot.graph_from_dot_data(path) # dot文件路径, 返回一个list
```

12.3.Dot

12.3.1.write_png()

生成png图像

```
graph.write_png(path)# png图像的路径
```