Keras是深度学习引擎的高级封装API，支持Tensorflow, Theano, CNTK三种后端，

Tensorflow在CPU上以张量运算库Eigen运行，在GPU上以运算库NVIDIA CUDA运行

import tensorflow as tf

import keras

环境配置

## 检查GPU

import torch

if torch.cuda.is\_available():

device = torch.device("cuda")

print('There are %d GPU(s) available.' % torch.cuda.device\_count())

print('We will use the GPU')

else:

print('No GPU available, using the CPU instead.')

device = torch.device("cpu")

## 检查内存

from psutil import virtual\_memory

ram\_gb = virtual\_memory().total / 1e9

print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram\_gb))

if ram\_gb < 20:print('Not using a high-RAM runtime')

else:print('You are using a high-RAM runtime!')

张量

Tensorflow张量类型是基于numpy ndarray实现的类，但不能直接调用np的函数处理，需要先转换为ndarray类型，不过tf库自身有处理该张量类型的函数，且使用方法与np类似

**创建常规张量**

t=tf.constant(ndarray,\*dtype=tf.float32)

#ndarray的dtype应为float32或int，否则会造成信息损失，因为tf张量默认用32位精度

#常规张量不能被用于给需要被不断更新的张量，如权重，进行赋值

**创建变量张量**

t=tf.Variable(ndarray,\*dtype=tf.float32)

#变量张量的值可以被更新

**查看张量形状** t.shape

**查看张量数据类型** t.dtype

**查看张量轴数** t.ndim

**张量转ndarray** t.numpy()

**张量复制** t.copy()

**张量广播：**两个不同ndim的张量相加减时，与numpy类似，较小的张量会被重复扩张至大张量的ndim

**张量点乘结果的形状：**两个张量点乘的轴相消，其余轴的形状都保留到结果的形状中。

**张量四则运算**

t=t±n

t=t\*/n

**张量元素开根** t=tf.square(t)

**张量转置** t=tf.transpose(t)

**张量均值** tf.reduce\_mean(t)

**张量加和** tf.reduce\_sum(t)

**张量取对数** tf.math.log(t)

**张量最大/最小值**

tf.reduce\_max(t)

tf.reduce\_min(t)

**张量变形** tf.reshape(newshape)

数据预处理

**标签array独热化**

from keras.utils import to\_categorical

labels= to\_categorical(labels)

**Keras.preprocessing.text.Tokenizer**

from keras.preprocessing.text import Tokenizer

tokenizer=Tokenizer(num\_words=n) #输出的向量只考虑频数最高的n个词

tokenizer.fit\_on\_texts(texts)

sequences=tokenizer.texts\_to\_sequences(texts) #输出每个词整数索引化的序列

word\_index=tokenizer.word\_index #每个词的索引

print('Found %s unique tokens.'% len(word\_index)) #查看共有多少个不同的词

#Tokenizer实现了简单的文本向量化

**pad.sequence()统一序列长度**

from keras import preprocessing

preprocessing.sequence.pad\_sequences(原始序列数据,maxlen,padding.truncating,value)

#该函数的作用是统一序列数据中各序列的长度

#maxlen是统一的长度

#padding=’pre’，是长度不足时填充的方向，默认为尾

#truncating=’pre’，是长度溢出时截断的方向，默认为尾

#value是填充值，默认为0

建模基本

**顺序模型架构**

keras.models.Sequential()

**设置权重**

model.set\_weights(W)

**访问权重**

model.get\_weights()

**加载模型**

model=keras.models.load\_model(‘model.h5’)

**保存模型**

model.save(‘model.h5’)

**克隆模型架构(不会克隆权重，需用set\_weights设置)**

model\_B=keras.models.clone\_model(model\_A)

**模型总结**

model.summary()

**模型编译**

model.compile(loss,optimizer,metrics)

**模型评估**

loss,metrics=model.evaluate(X\_val,y\_val)

**模型预测**

y\_pred=model.predict(X\_pred)

层

**定义层/添加层/删除层**

keras.layers.层类型(…)

model.add(layers.层类型(…))

model.pop(layers.层类型(…))

**输入层**

keras.layers.Input(shape=input\_shape)

#input\_shape为batch中一个样本的shape，如序列为(n,)，矩阵为(m,n)

**平化层（将批量重整为向量）**

keras.layers.Flatten()

常用于批量降维

**批量归一化层**

keras.layers.BatchNormalization()

**激活层**

keras.layers.Activation(‘激活函数名’)

**Reshape层**

keras.layers.Reshape(shape)

**Dropout层**

keras.layers.Dropout(rate=n)

#循环网络取0.2-0.3，卷积网络取0.4-0.5

#常规Dropout机制会破坏SELU激活函数的自归一化机制

**Alpha Dropout**

keras.layers.AlphaDropout(rate=n)

#Alpha Dropout不会破坏SELU激活函数的自归一化机制

**全连接层**

keras.layers.Dense(n,activation)

**RNN类层**

keras.layers.SimpleRNN(n,return\_sequences=bool, dropout,recurrent\_dropout)

keras.layers.LSTM(n,return\_sequences=bool, dropout,recurrent\_dropout)

keras.layers.GRU(n,return\_sequences=bool, dropout,recurrent\_dropout)

keras.layers.Bidirectional(layers.LSTM(…)) #双向RNN类层

#n是层的输出序列长度

**卷积层**

keras.layers.Conv1D(input\_shape,filters, kernel\_size=n, strides, padding,activation)

keras.layers.Conv2D(input\_shape,filters, kernel\_size=(m,n), strides, padding,activation)

#input\_shape是(channels, rows, cols)或(rows, cols, channels)

#filters为输出深度，也是输出特征图的通道数

#kernel\_size=(卷积核宽,卷积核高)，kernel\_size和strides,padding决定了输出特征图的形状

#strides是卷积窗口之间的中心距，strides越大窗口之间间隔越远，卷积采样越稀疏

#padding=’valid’/’same’，valid表示不使用padding, same表示输出特征图形状和原图相同

**池化层**

keras.layers.MaxPooling1D(pool\_size=2, strides=None, padding='valid')

keras.layers.MaxPooling2D(pool\_size=(2, 2), strides=None, padding='valid')

keras.layers.MaxPooling3D(pool\_size=(2, 2, 2), strides=None, padding='valid')

keras.layers.AveragePooling1D(pool\_size=2, strides=None, padding='valid')

keras.layers.AveragePooling2D(pool\_size=(2, 2), strides=None, padding='valid')

keras.layers.AveragePooling3D(pool\_size=(2, 2, 2), strides=None, padding='valid')

**全局池化层(channel池化)**

keras.layers.GlobalAveragePooling1D()

keras.layers.GlobalAveragePooling2D()

keras.layers.GlobalAveragePooling3D()

keras.layers.GlobalMaxPooling1D()

keras.layers.GlobalMaxPooling2D()

keras.layers.GlobalMaxPooling3D()

**Embedding层**

#embedding层实现上是一种计算，将把单词整数序号化的文本数据，转化为把单词one-hot化的文本数据，然后进行词向量的训练，生成的词向量即是该层的weights矩阵

keras.layers.Embedding(文本样本数,单词嵌入维度,input\_length=每条文本的限制词数)

#输入同维度的词向量编码，可以是one-hot编码。

#input\_length限制并统一了每条文本的词数

#利用Embedding层，可以实现现场词向量训练

#利用layers[embedding\_index].set\_weights([预训练词向量矩阵])，可以加载预训练词向量

#利用layers[embedding\_index].trainable=False，可以阻止迭代中更新词向量，从而保留预训练结果

**层参数初始化方法**

Keras默认使用均匀分布Glorot初始化，可通过kernel\_initializer超参数来设置每层的初始化方法。如kernel\_initializer=’he\_normal’/’he\_uniform’/‘lecun\_normal’

**针对不同激活函数的初始化方法选择**

None/tanh/sigmoid/softmax -glorot\_uniform初始化

Relu/Leaky Relu/其它Relu变体 -’he\_normal’/’he\_uniform’初始化

SELU -‘lecun\_normal’初始化

**访问模型中的层**

Layers=model.layers

Layer=model.layers[i]

Layer=model.get\_layer(name=None,index=None)

**访问层的权重**

Layer.get\_weights()

**设置层的权重**

Layer.set\_weights(W)

**查看层中的变量**

Layers.variables

**查看变量名**

Layers.variables[i].name

**冻结层Freeze**

Layer.trainable=False

**设置正则化**

**L1**

kernel\_regularizer=keras.regularizers.l1(正则系数)

**L2**

kernel\_regularizer=keras.regularizers.l2(正则系数)

**L1+L2**

kernel\_regularizer=keras.regularizers.l1\_l2(正则系数1, 正则系数2)

**最大范数正则化**

kernel\_constraint=keras.constraints.max\_norm(最大范数,\*axis=0)

#最大范数会限制权重W，使其L2范数不会大于该值，通常为float(1)

激活函数

**使用方法**

1. 定义层时，通过超参数activation设置
2. 使用keras.layers来将激活函数定义为一个‘激活层’，同时上一层不要设激活函数

**LeakyReLU**

keras.layers.LeakyReLU(alpha=泄漏系数)

**SELU**

activation=’selu’,kernel\_initializer=’lecun\_normal’

**Tanh**

优化器

**设置梯度裁剪**

阈值裁剪：clipvalue=裁剪阈值

等比例裁剪：clipnorm=裁剪阈值

**随机梯度下降**

keras.optimizers.SGD(lr=学习率)

**设置SGD加速**

动量优化：momentum=动量系数(0-1)

Nesterov加速梯度：nesterov=True

**AdaGrad自适应梯度下降**

keras.optimizers.Adagrad(…)

#不建议用于训练深度神经网络

**RMSProp**

keras.optimizers.RMSprop(lr,\*rho=0.9)

**Adam矩估计**

keras.optimizers.Adam(lr,\*beta1=0.9,\*beta2=0.999)

**设置学习率幂调度**

decay=1e-4

损失函数

mse,mae #回归问题

binary\_crossentropy #二分类问题

categorical\_crossentropy #标签为one-hot输入的多分类问题

sparse\_categorical\_crossentropy #标签为序列输入的多分类问题

建模

训练前，先将数据标准化

**模型拟合**

history=model.fit(X,y,\*batch\_size,epochs,\*validation\_data=(X\_val,y\_val),\*class\_weight=标签权重,\*sample\_weight=样本权重,verbose=是否显示训练过程)

**训练参数**

history.params

**轮次列表**

history.epoch

**查看训练指标在每个epoch训练中的历史值并访问**

history.history.keys()

history.history[‘key’]

**绘制epoch-loss图**

history=model.fit(X,y)

plt.figure()

timestep=range(1,总epoch数+1)

plt.plot(timestep,history.history['val\_loss'],'b',label=’Validation’)

plt.plot(timestep,history.history['loss'],'bo',label=’Train’)

plt.legend()

plt.show()

**模型评估**

loss,accuracy=model.evaluate(X\_test,y\_test)

#也可以通过在model.fit中设置validation\_data参数来直接评估

**模型预测**

y\_pred=model.predict(X)

#如果是分类模型，将输出各标签概率值

**分类模型预测**

y\_pred=model.predict\_classes(X)

回调

**设置回调点-保存训练过程最佳模型**

checkpoint=keras.callbacks.ModelCheckpoint(‘model.h5’,save\_best\_only=True)

model.fit(X,y,epochs,validation\_data=(X\_val,y\_val),callbacks=[checkpoint])

model=keras.models.load\_model(‘model.h5’)

**设置回调点-EarlyStopping回滚**

earlystop=keras.callbacks.EarlyStopping(patience=耐心轮数,restore\_best\_weights=True)

model.fit(X,y,validation\_data,callbacks=[earlystop])

迁移学习

冻结所有可重复使用的层并训练几轮以较好地初始化新定义的层，拿初始化后的模型训练查看效果→向上解冻一些层，开启对其的反向传播，训练查看效果→调参再训练→从深至浅去除或改变隐藏层，使用同样的过程初始化新层并训练查看效果。

迁移学习时，对于开启反向传播的层可以采用较小的学习率，来避免已调节好的原始权重过大改变。

**无监督预训练**

通过训练当前资源能满足的任务来获得具有良好底层特征的模型，在目标模型的训练中使用这些模型的底层权重，从而在目标模型训练的开始就获得较好的底层特征。

网格搜索



如果是分类任务，就用KerasClassifier

数据集

**minst 手写数字**

from keras.datasets import mnist

(train\_image, train\_labels), (test\_image, test\_labels)=minst.load\_data()

#每个数据集是以numpy形式存储的数字化灰度图片

**IMDB情感二分类文本数据**

from keras.datasets import imdb

(train\_data, train\_labels), (test\_data, test\_labels)=imdb.load\_data(num\_words=n)

#n指示每条数据中仅保留训练数据中前n个频率最高的词

#每条数据是数字编码化的文本，标签是0-1的整数对应2个类别

word\_dict=imdb.get\_word\_index()

reversed\_dict=dict([(value,key) for (key,value) in word\_dict.items()])

decode=’ ’.join([reverse\_dict.get(i-3,’?’) for in train\_data[0]]) #i-3是因为0,1,2是保留索引

#解码文本数据

**reuters主题多分类文本数据**

from keras.datasets import reuters

(train\_data, train\_labels), (test\_data, test\_labels)= reuters.load\_data(num\_words=n)

#n指示每条数据中仅保留训练数据中前n个频率最高的词

#每条数据是数字编码化的文本，标签是0-45的整数对应46个类别

word\_dict= reuters.get\_word\_index()

reversed\_dict=dict([(value,key) for (key,value) in word\_dict.items()])

decode=’ ’.join([reverse\_dict.get(i-3,’?’) for in train\_data[0]]) #i-3是因为0,1,2是保留索引

#解码文本数据

**Boston房价数据回归**

from keras.datasets import boston\_housing

(train\_data, train\_targets), (test\_data, test\_targets)=

boston\_housing.load\_data(num\_words=n)

**fashion minist图片分类**

from keras.datasets import fashion\_mnist

(X\_train, y\_train), (X\_test, y\_test) = fashion\_mnist.load\_data()

常用架构

**0-1分类**

Dense(relu)+ Dense(relu)+ Dense(1,sigmoid)

**多分类**

Dense(relu)+ Dense(relu)+ Dense(1,softmax)

**回归**

Dense(relu)+ Dense(relu)+ Dense(n,None) #向量回归n>1，值回归n=1

**卷积**

Conv2D(relu)+MaxPooling()+Flatten()+Dense()

**多标签输出**

**序列生成**

可视化

**模型结构可视化**

from keras.utils.vis\_utils import plot\_model

**训练过程可视化**

def training\_plots(history):

history\_dict=history.history

loss\_values=history\_dict['loss']

val\_loss\_values=history\_dict['val\_loss']

train\_acc=history\_dict['mae']

valid\_acc=history\_dict['val\_mae']

epochs=range(1,len(loss\_values)+1)

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)

plt.plot(epochs,loss\_values,'b',label='Training Loss')

plt.plot(epochs,val\_loss\_values,'b',label='Validation loss',color='coral')

plt.title('Training & Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1,2,2)

plt.plot(epochs,train\_acc,'b',label='Training Mae')

plt.plot(epochs,valid\_acc,'b',label='Validation Mae',color='coral')

plt.title('Training & Validation Acc')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

自定义组件

## 自定义损失函数

输入：单一y\_true,y\_pred

输出：单一loss值

**普通损失函数**

def My\_loss(y\_true,y\_pred):

损失计算方法

return loss

#此处的y\_true,y\_pred都是单对的，计算的loss值也仅是一对y\_true,y\_pred的loss

**有参数的损失函数**

def My\_loss\_compiler(n1,n2,…)

def My\_loss(y\_true,y\_pred):

损失计算方法

return loss

return My\_loss

#n1,n2,…以局部变量的方式在My\_loss种被直接引用

**调用**

model.compile(loss=My\_loss\_compiler(n1,n2,…))

## 自定义激活函数

输入：W\*X+B的结果张量

输出：非线性变换值张量

def My\_activation(layer\_output):

layer\_output\_copy= layer\_output.copy() #copy避免修改上一层的原数据

对layer\_output\_copy的变换

return layer\_output\_copy

#由于此处layer\_output是tf库的张量类型，变换过程需要tf库对张量的操作函数来实现

## 自定义正则化

输入：层权重张量

输出：针对每个权重的正则化值张量

def My\_regularizer(weights):

基于weights元素的正则化值计算方法，也可以是一常数构成的张量

return penalty

#由于此处weights是tf库的张量类型，正则化值计算需要tf库对张量的操作函数来实现

## 加载包含自定义组件的模型

keras.models.load\_model(modelpath,custom\_objects={自定义组件名1:自定义组件1,…}

#当自定义组件有输入参数时，所保存的模型不会保存到这些参数，在加载时需要在对应组件后指定参数值。

预训练模型

调用模板

pretrained= PretrainedModel(include\_top=False)

for layer in pretrained.layers: layer.trainable = False #锁预训练层

model = Sequential()

model.add(pretrained)

…

model.compile()

## CV

**VGG-16**

from keras.applications.vgg16 import VGG16

pretrained=VGG16(weights = 'imagenet', include\_top = False, input\_shape=(h,w,channel))

**ResNet50**

from keras.applications.resnet50 import ResNet50

pretrained=ResNet50(weights= 'imagenet', include\_top=False, input\_shape=(h,w,channel))

**Inception-v3**

from keras.applications.inception\_v3 import InceptionV3

pretrained=InceptionV3(weights= 'imagenet', include\_top=False, input\_shape=(h,w,channel))

**Xception**

from tensorflow.keras.applications import Xception

pretrained=Xception(weights= 'imagenet', include\_top=False, input\_shape=(h,w,channel))