https://blog.csdn.net/lk7688535/article/details/52862377

keras是一个开源是的python深度学习库，可以基于theano或者tenserflow，下面大体介绍下keras的几个重要模块

重要的模块

1、优化器（optimizers）

优化器是调整每个节点权重的方法，看一个代码示例：

model = Sequential()

model.add(Dense(64, init='uniform', input\_dim=10)) model.add(Activation('tanh'))

model.add(Activation('softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True) model.compile(loss='mean\_squared\_error', optimizer=sgd)

可以看到优化器在模型编译前定义，作为编译时的两个参数之一

代码中的sgd是随机梯度下降算法

lr表示学习速率

momentum表示动量项

decay是学习速率的衰减系数(每个epoch衰减一次)

Nesterov的值是False或者True，表示使不使用Nesterov momentum

以上4个参数以后具体学习了再解析

除了sgd，还可以选择的优化器有RMSprop（适合递归神经网络）、Adagrad、Adadelta、Adam、Adamax、Nadam

2、目标函数（objectives）

目标函数又称损失函数（loss），目的是计算神经网络的输出与样本标记的差的一种方法，代码示例：

model = Sequential()

model.add(Dense(64, init='uniform', input\_dim=10)) model.add(Activation('tanh'))

model.add(Activation('softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True) model.compile(loss='mean\_squared\_error', optimizer=sgd)

mean\_squared\_error就是损失函数的名称

可以选择的损失函数有：

mean\_squared\_error，mean\_absolute\_error，squared\_hinge，hinge，binary\_crossentropy，categorical\_crossentropy

这里binary\_crossentropy 和 categorical\_crossentropy也就是logloss

3、激活函数（activations）

每一个神经网络层都需要一个激活函数，代码示例：

from keras.layers.core import Activation, Dense

model.add(Dense(64))

model.add(Activation('tanh'))

或把上面两行合并为：

model.add(Dense(64, activation='tanh'))

可以选择的激活函数有：

linear、sigmoid、hard\_sigmoid、tanh、softplus、relu、 softplus，softmax、softsign

还有一些高级激活函数，比如如PReLU，LeakyReLU等

4、参数初始化（Initializations）

这个模块的作用是在添加layer时调用init进行这一层的权重初始化，有两种初始化方法

4.1 通过制定初始化方法的名称：

示例代码：

model.add(Dense(64, init='uniform'))

1

可以选择的初始化方法有：

uniform、lecun\_uniform、normal、orthogonal、zero、glorot\_normal、he\_normal等

4.2 通过调用对象：

该对象必须包含两个参数:shape(待初始化的变量的shape)和name(该变量的名字),该可调用对象必须返回一个(Keras)变量,例如K.variable()返回的就是这种变量，示例代码：

from keras import backend as K

import numpy as np

def my\_init(shape, name=None):

value = np.random.random(shape)

return K.variable(value, name=name)

model.add(Dense(64, init=my\_init))

或者

from keras import initializations

def my\_init(shape, name=None):

return initializations.normal(shape, scale=0.01, name=name)

model.add(Dense(64, init=my\_init))

所以说可以通过库中的方法设定每一层的初始化权重，

也可以自己初始化权重，自己设定的话可以精确到每个节点的权重，

那么是否可以在这儿做文章优化特征呢？我觉得可以针对不同的task深入试验看看

5、层（layer）

keras的层主要包括：

常用层（Core）、卷积层（Convolutional）、池化层（Pooling）、局部连接层、递归层（Recurrent）、嵌入层（ Embedding）、高级激活层、规范层、噪声层、包装层，当然也可以编写自己的层

5.1对于层的操作

layer.get\_weights() #返回该层的权重

layer.set\_weights(weights)#将权重加载到该层

config = layer.get\_config()#保存该层的配置

layer = layer\_from\_config(config)#加载一个配置到该层

#该层有一个节点时，获得输入张量、输出张量、及各自的形状：

layer.input

layer.output

layer.input\_shape

layer.output\_shape

#该层有多个节点时（node\_index为节点序号）：

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)

5.2 Dense层(全连接层）

keras.layers.core.Dense(output\_dim, init='glorot\_uniform', activation='linear', weights=None, W\_regularizer=None, b\_regularizer=None, activity\_regularizer=None, W\_constraint=None, b\_constraint=None, bias=True, input\_dim=None)

output\_dim：输出数据的维度

init：初始化该层权重的方法

activation：该层的激活函数

weights：numpy array的list。该list应含有一个形如（input\_dim,output\_dim）的权重矩阵和一个形如(output\_dim,)的偏置向量

regularizer：正则项，w为权重的、b为偏执的，activity为输出的

constraints：约束项

bias：是否包含偏执向量，是布尔值

input\_dim：输入数据的维度

5.3 dropout层

keras.layers.core.Dropout(p)

为输入数据施加Dropout。Dropout将在训练过程中每次更新参数时随机断开一定百分比（p）的输入神经元连接，Dropout层用于防止过拟合。

参考文章：http://blog.csdn.net/stdcoutzyx/article/details/49022443理解dropout

5.4 递归层（Recurrent）

递归层包含三种模型：LSTM、GRU和SimpleRNN

5.4.1抽象层，不能直接使用

keras.layers.recurrent.Recurrent(weights=None, return\_sequences=False, go\_backwards=False, stateful=False, unroll=False, consume\_less='cpu', input\_dim=None, input\_length=None)

1

return\_sequences：True返回整个序列，false返回输出序列的最后一个输出

go\_backwards：True，逆向处理输入序列，默认为False

stateful：布尔值，默认为False，若为True，则一个batch中下标为i的样本的最终状态将会用作下一个batch同样下标的样本的初始状态

5.4.2全连接RNN网络

keras.layers.recurrent.SimpleRNN(output\_dim, init='glorot\_uniform', inner\_init='orthogonal', activation='tanh', W\_regularizer=None, U\_regularizer=None, b\_regularizer=None, dropout\_W=0.0, dropout\_U=0.0)

1

inner\_init：内部单元的初始化方法

dropout\_W：0~1之间的浮点数，控制输入单元到输入门的连接断开比例

dropout\_U：0~1之间的浮点数，控制输入单元到递归连接的断开比例

5.4.3 LSTM层

keras.layers.recurrent.LSTM(output\_dim, init='glorot\_uniform', inner\_init='orthogonal', forget\_bias\_init='one', activation='tanh', inner\_activation='hard\_sigmoid', W\_regularizer=None, U\_regularizer=None, b\_regularizer=None, dropout\_W=0.0, dropout\_U=0.0)

1

forget\_bias\_init：遗忘门偏置的初始化函数，Jozefowicz et al.建议初始化为全1元素

inner\_activation：内部单元激活函数

5.5 Embedding层

keras.layers.embeddings.Embedding(input\_dim, output\_dim, init='uniform', input\_length=None, W\_regularizer=None, activity\_regularizer=None, W\_constraint=None, mask\_zero=False, weights=None, dropout=0.0)

1

只能作为模型第一层

mask\_zero：布尔值，确定是否将输入中的‘0’看作是应该被忽略的‘填充’（padding）值，该参数在使用递归层处理变长输入时有用。设置为True的话，模型中后续的层必须都支持masking，否则会抛出异常

5.6 model层（最重要）

model层是最主要的模块，model层可以将上面定义了各种基本组件组合起来

model的方法：

model.summary() ： 打印出模型概况

model.get\_config() ：返回包含模型配置信息的Python字典

model.get\_weights()：返回模型权重张量的列表，类型为numpy array

model.set\_weights()：从numpy array里将权重载入给模型

model.to\_json：返回代表模型的JSON字符串，仅包含网络结构，不包含权值。可以从JSON字符串中重构原模型：

from models import model\_from\_json

json\_string = model.to\_json()

model = model\_from\_json(json\_string)

model.to\_yaml：与model.to\_json类似，同样可以从产生的YAML字符串中重构模型

from models import model\_from\_yaml

yaml\_string = model.to\_yaml()

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, by\_name=False)：从HDF5文件中加载权重到当前模型中, 默认情况下模型的结构将保持不变。如果想将权重载入不同的模型（有些层相同）中，则设置by\_name=True，只有名字匹配的层才会载入权重

keras有两种model，分别是Sequential模型和泛型模型

5.6.1 Sequential模型

Sequential是多个网络层的线性堆叠

可以通过向Sequential模型传递一个layer的list来构造该模型：

from keras.models import Sequential

from keras.layers import Dense, Activation

model = Sequential([

Dense(32, input\_dim=784),

Activation('relu'),

Dense(10),

Activation('softmax'),

])

也可以通过.add()方法一个个的将layer加入模型中：

model = Sequential()

model.add(Dense(32, input\_dim=784))

model.add(Activation('relu'))

还可以通过merge将两个Sequential模型通过某种方式合并

Sequential模型的方法：

compile(self, optimizer, loss, metrics=[], sample\_weight\_mode=None)

fit(self, x, y, batch\_size=32, nb\_epoch=10, verbose=1, callbacks=[], validation\_split=0.0, validation\_data=None, shuffle=True, class\_weight=None, sample\_weight=None)

evaluate(self, x, y, batch\_size=32, verbose=1, sample\_weight=None)

#按batch获得输入数据对应的输出，函数的返回值是预测值的numpy array

predict(self, x, batch\_size=32, verbose=0)

#按batch产生输入数据的类别预测结果,函数的返回值是类别预测结果的numpy array或numpy

predict\_classes(self, x, batch\_size=32, verbose=1)

#本函数按batch产生输入数据属于各个类别的概率，函数的返回值是类别概率的numpy array

predict\_proba(self, x, batch\_size=32, verbose=1)

train\_on\_batch(self, x, y, class\_weight=None, sample\_weight=None)

test\_on\_batch(self, x, y, sample\_weight=None)

predict\_on\_batch(self, x)

fit\_generator(self, generator, samples\_per\_epoch, nb\_epoch, verbose=1, callbacks=[], validation\_data=None, nb\_val\_samples=None, class\_weight=None, max\_q\_size=10)

evaluate\_generator(self, generator, val\_samples, max\_q\_size=10)

5.6.2 泛型模型

Keras泛型模型接口是：

用户定义多输出模型、非循环有向模型或具有共享层的模型等复杂模型的途径

适用于实现：全连接网络和多输入多输出模型

多输入多输出，官方例子给出：预测一条新闻的点赞转发数，主要输入是新闻本身，还可以加入额外输入，比如新闻发布日期，新闻作者等，具体的实现还是看官网文档吧：

http://keras-cn.readthedocs.io/en/latest/getting\_started/functional\_API/

所以感觉这个模型可以针对特定task搞一些创新哦

泛型模型model的属性：

model.layers：组成模型图的各个层

model.inputs：模型的输入张量列表

model.outputs：模型的输出张量列表

方法：类似序列模型的方法

补充get\_layer

get\_layer(self, name=None, index=None)

本函数依据模型中层的下标或名字获得层对象，泛型模型中层的下标依据自底向上，水平遍历的顺序。

name：字符串，层的名字

index： 整数，层的下标

函数的返回值是层对象