众所周知，R语言是统计分析最好用的语言。但在Keras和TensorFlow的帮助下，R语言也可以进行深度学习了。

在机器学习的语言的选择上，R和Python之间选择一直是一个有争议的话题。但随着深度学习的爆炸性增长，越来越多的人选择了Python，因为它有一个很大的深度学习库和框架，而R却没有（直到现在）。
但是我就是想使用R语言进入深度学习空间，所以我就从Python领域转入到了R领域，继续我的深度学习的研究了。这可能看起来几乎不可能的。但是今天这变成了可能。
随着Keras在R上的推出，R与Python的斗争回到了中心。Python慢慢成为了最流行的深度学习模型。但是，随着Keras库在R后端的发布，并且在后台还可以使用张力流（TensorFlow）（CPU和GPU兼容性），所以在深度学习领域，R将再次与Python打成平手。
下面我们将看到如何使用Tensorflow在R中安装Keras，并在RStudio的经典MNIST数据集上构建我们的第一个神经网络模型。

目录：

1.在后端安装带有张量的Keras。
2.使用Keras可以在R中构建不同类型的模型。
3.在R中使用MLP对MNIST手写数字进行分类。
4.将MNIST结果与Python中的等效代码进行比较。
5.结束笔记。

1.在后端安装带有TensorFlow的Keras。

在RStudio中安装Keras的步骤非常简单。只需按照以下步骤，您将很顺利的在R中创建您的第一个神经网络模型。

```
install.packages("devtools")
devtools::install_github("rstudio/keras")
```

上述步骤将从GitHub仓库加载keras库。现在是将keras加载到R并安装TensorFlow的时候了。

```
library(keras)
```

默认情况下，RStudio加载TensorFlow的CPU版本。使用以下命令下载TensorFlow的CPU版本。

```
install_tensorflow()
```

要为单个用户/桌面系统安装具有GPU支持的TensorFlow版本，请使用以下命令。

```
install_tensorflow(gpu=TRUE)
```

有关更多的用户安装，请参阅本安装指南。

现在我们在RStudio中安装了keras和TensorFlow，让我们在R中启动和构建我们的第一个神经网络来解决MNIST数据集

2.使用keras可以在R中构建的不同类型的模型

以下是使用Keras可以在R中构建的模型列表。

1.多层感知器

2.卷积神经网络

3.循环神经网络

4.Skip-Gram模型

5.使用预先训练的模型，如VGG16，RESNET等

6.微调预先训练的模型。

让我们开始构建一个非常简单的MLP模型，只需一个隐藏的层来尝试分类手写数字。

3.使用R中的MLP对MNIST手写数字进行分类

```
#loading keras library
library(keras)
#loading the keras inbuilt mnist dataset
data<-dataset_mnist()
#separating train and test file
train_x<-data$train$x
train_y<-data$train$y
test_x<-data$test$x
test_y<-data$test$y
rm(data)
# converting a 2D array into a 1D array for feeding into the MLP and normalising the matri
train_x <- array(train_x, dim = c(dim(train_x)[1], prod(dim(train_x)[-1]))) / 255
test_x <- array(test_x, dim = c(dim(test_x)[1], prod(dim(test_x)[-1]))) / 255
#converting the target variable to once hot encoded vectors using keras inbuilt function
train_y<-to_categorical(train_y,10)
test_y<-to_categorical(test_y,10)
#defining a keras sequential model
model <- keras_model_sequential()
#defining the model with 1 input layer[784 neurons], 1 hidden layer[784 neurons] with drop
#i.e number of digits from 0 to 9
model %>%
layer_dense(units = 784, input_shape = 784) %>%
layer_dropout(rate=0.4)%>%
layer_activation(activation = 'relu') %>%
layer_dense(units = 10) %>%
layer_activation(activation = 'softmax')
#compiling the defined model with metric = accuracy and optimiser as adam.
model %>% compile(
loss = 'categorical_crossentropy',
```

```
optimizer = 'adam',
metrics = c('accuracy')
)
#fitting the model on the training dataset
model %>% fit(train_x, train_y, epochs = 100, batch_size = 128)
#Evaluating model on the cross validation dataset
loss_and_metrics <- model %>% evaluate(test_x, test_y, batch_size = 128)
```

上述代码的训练精度为99.14，验证准确率为96.89。代码在i5处理器上运行，运行时间为13.5秒，而在TITANx GPU上，验证精度为98.44，平均运行时间为2秒。

4.MLP使用keras–R VS Python

为了比较起见，我也在Python中实现了上述的MNIST问题。我觉得在keras-R和Python中应该没有任何区别，因为R中的keras创建了一个conda实例并在其中运行keras。你可以尝试运行一下下面等效的python代码。

```
#importing the required libraries for the MLP model
import  keras
from  keras.models  import Sequential
import  numpy  as  np
#loading  the  MNIST dataset  from  keras
from  keras.datasets  import  mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
#reshaping the x_train, y_train, x_test and y_test to conform to MLP input and output dime
x_train=np.reshape(x_train,(x_train.shape[0],-1))/255
x_test=np.reshape(x_test,(x_test.shape[0],-1))/255
import pandas as pd
y_train=pd.get_dummies(y_train)
y_test=pd.get_dummies(y_test)
#performing one-hot encoding on target variables for train and test
y_train=np.array(y_train)
y_test=np.array(y_test)
#defining model with one input layer[784 neurons], 1 hidden layer[784 neurons] with dropou
model=Sequential()
from keras.layers import Dense
model.add(Dense(784, input_dim=784, activation='relu'))
keras.layers.core.Dropout(rate=0.4)
model.add(Dense(10,input_dim=784,activation='softmax'))
# compiling model using adam optimiser and accuracy as metric
model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
# fitting model and performing validation
model.fit(x_train,y_train,epochs=50,batch_size=128,validation_data=(x_test,y_test))
```

上述模型在同一GPU上实现了98.42的验证精度。所以，我们最初猜到的结果是正确的。

5.结束笔记

如果这是你在R的第一个深度学习模型，我希望你喜欢它。通过一个非常简单的代码，您可以有98%位准确率对是否为手写数字进行分类。这应该是足够的动力让你开始深度学习。

如果您已经在Python中使用keras深度学习库，那么您将在R中找到keras库的语法和结构与Python中相似的地方。事实上，R中的keras包创建了一个conda环境，并安装了在该环境中运行keras所需的一切。但是，让我更为激动的是，现在看到数据科学家在R中建立现实生活中的深层次的学习模型。据说 - 竞争应该永远不会停止。我也想听听你对这一新发展观点的看法。你可以在下面留言分享你的看法。