**The Pale Blue Dot**

*“From this distant vantage point, the Earth might not seem of any particular interest. But for us, it’s different. Consider again that dot. That’s here, that’s home, that’s us. On it everyone you love, everyone you know, everyone you ever heard of, every human being who ever was, lived out their lives. The aggregate of our joy and suffering, thousands of confident religions, ideologies, and economic doctrines, every hunter and forager, every hero and coward, every creator and destroyer of civilization, every king and peasant, every young couple in love, every mother and father, hopeful child, inventor and explorer, every teacher of morals, every corrupt politician, every “superstar,” every “supreme leader,” every saint and sinner in the history of our species lived there—on the mote of dust suspended in a sunbeam.”*

Carl Sagan

Tensorflow and Keras are Deep Learning frameworks that really simplify a lot of things to the user. If you are familiar with Machine Learning and Deep Learning concepts then Tensorflow and Keras are really a playground to realize your ideas.  In this post I show how you can get started with Tensorflow in both Python and R

**Tensorflow in Python**

For tensorflow in Python, I found [Google’s Colab](https://colab.research.google.com/) an ideal environment for running your Deep Learning code. This is an Google’s research project  where you can execute your code  on GPUs, TPUs etc

**Tensorflow in R (RStudio)**

To execute tensorflow in R (RStudio) you need to install tensorflow and keras as shown below  
In this post I show how to get started with Tensorflow and Keras in R.

# Install Tensorflow in RStudio

#install\_tensorflow()

# Install Keras

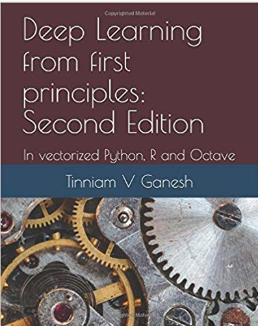
#install\_packages("keras")

library(tensorflow)

libary(keras)

This post takes 3 different Machine Learning problems and uses the  
Tensorflow/Keras framework to solve it

**Note**:  
You can view the Google Colab notebook at [Tensorflow in Python](https://colab.research.google.com/drive/1hJb5DCsCuWxSXnJF7noUr8ZTqwLRTWIe)  
The RMarkdown file has been published at RPubs and can be accessed  
at [Getting started with Tensorflow in R](http://rpubs.com/tvganesh/516627)

Checkout my book ‘Deep Learning from first principles: Second Edition – In vectorized Python, R and Octave’. My book starts with the implementation of a simple 2-layer Neural Network and works its way to a generic L-Layer Deep Learning Network, with all the bells and whistles. The derivations have been discussed in detail. The code has been extensively commented and included in its entirety in the Appendix sections. My book is available on Amazon as [paperback](https://www.amazon.com/dp/1791596177)($14.99) and in [kindle version](https://www.amazon.com/dp/B07LBG542L)($9.99/Rs449).

**1. Multivariate regression with Tensorflow – Python**

This code performs multivariate regression using Tensorflow and keras on the advent of Parkinson disease through sound recordings see [Parkinson Speech Dataset with Multiple Types of Sound Recordings Data Set](https://archive.ics.uci.edu/ml/datasets/Parkinson+Speech+Dataset+with++Multiple+Types+of+Sound+Recordings). The clinician’s motorUPDRS score has to be predicted from the set of features

In [0]:

# Import tensorflow

import tensorflow as tf

from tensorflow import keras

In [2]:

#Get the data rom the UCI Machine Learning repository

dataset = keras.utils.get\_file("parkinsons\_updrs.data", "https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/telemonitoring/parkinsons\_updrs.data")

Downloading data from https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/telemonitoring/parkinsons\_updrs.data

917504/911261 [==============================] - 0s 0us/step

In [3]:

# Read the CSV file

import pandas as pd

parkinsons = pd.read\_csv(dataset, na\_values = "?", comment='\t',

sep=",", skipinitialspace=True)

print(parkinsons.shape)

print(parkinsons.columns)

#Check if there are any NAs in the rows

parkinsons.isna().sum()

(5875, 22)

Index(['subject#', 'age', 'sex', 'test\_time', 'motor\_UPDRS', 'total\_UPDRS',

'Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'Jitter:PPQ5', 'Jitter:DDP',

'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',

'Shimmer:APQ11', 'Shimmer:DDA', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE'],

dtype='object')

Out[3]:

subject# 0

age 0

sex 0

test\_time 0

motor\_UPDRS 0

total\_UPDRS 0

Jitter(%) 0

Jitter(Abs) 0

Jitter:RAP 0

Jitter:PPQ5 0

Jitter:DDP 0

Shimmer 0

Shimmer(dB) 0

Shimmer:APQ3 0

Shimmer:APQ5 0

Shimmer:APQ11 0

Shimmer:DDA 0

NHR 0

HNR 0

RPDE 0

DFA 0

PPE 0

dtype: int64

**Note**: To see how to create dummy variables see my post [Practical Machine Learning with R and Python – Part 2](https://gigadom.in/2017/10/13/practical-machine-learning-with-r-and-python-part-2/)

In [4]:

# Drop the columns subject number as it is not relevant

parkinsons1=parkinsons.drop(['subject#'],axis=1)

# Create dummy variables for sex (M/F)

parkinsons2=pd.get\_dummies(parkinsons1,columns=['sex'])

parkinsons2.head()

Out[4]

age test\_time motor\_UPDRS total\_UPDRS Jitter(%) Jitter(Abs) Jitter:RAP Jitter:PPQ5 Jitter:DDP Shimmer Shimmer(dB) Shimmer:APQ3 Shimmer:APQ5 Shimmer:APQ11 Shimmer:DDA NHR HNR RPDE DFA PPE sex\_0 sex\_1

0 72 5.6431 28.199 34.398 0.00662 0.000034 0.00401 0.00317 0.01204 0.02565 0.230 0.01438 0.01309 0.01662 0.04314 0.014290 21.640 0.41888 0.54842 0.16006 1 0

1 72 12.6660 28.447 34.894 0.00300 0.000017 0.00132 0.00150 0.00395 0.02024 0.179 0.00994 0.01072 0.01689 0.02982 0.011112 27.183 0.43493 0.56477 0.10810 1 0

2 72 19.6810 28.695 35.389 0.00481 0.000025 0.00205 0.00208 0.00616 0.01675 0.181 0.00734 0.00844 0.01458 0.02202 0.020220 23.047 0.46222 0.54405 0.21014 1 0

3 72 25.6470 28.905 35.810 0.00528 0.000027 0.00191 0.00264 0.00573 0.02309 0.327 0.01106 0.01265 0.01963 0.03317 0.027837 24.445 0.48730 0.57794 0.33277 1 0

4 72 33.6420 29.187 36.375 0.00335 0.000020 0.00093 0.00130 0.00278 0.01703 0.176 0.00679 0.00929 0.01819 0.02036 0.011625 26.126 0.47188 0.56122 0.19361 1 0

# Create a training and test data set with 80%/20%

train\_dataset = parkinsons2.sample(frac=0.8,random\_state=0)

test\_dataset = parkinsons2.drop(train\_dataset.index)

# Select columns

train\_dataset1= train\_dataset[['age', 'test\_time', 'Jitter(%)', 'Jitter(Abs)',

'Jitter:RAP', 'Jitter:PPQ5', 'Jitter:DDP', 'Shimmer', 'Shimmer(dB)',

'Shimmer:APQ3', 'Shimmer:APQ5', 'Shimmer:APQ11', 'Shimmer:DDA', 'NHR',

'HNR', 'RPDE', 'DFA', 'PPE', 'sex\_0', 'sex\_1']]

test\_dataset1= test\_dataset[['age','test\_time', 'Jitter(%)', 'Jitter(Abs)',

'Jitter:RAP', 'Jitter:PPQ5', 'Jitter:DDP', 'Shimmer', 'Shimmer(dB)',

'Shimmer:APQ3', 'Shimmer:APQ5', 'Shimmer:APQ11', 'Shimmer:DDA', 'NHR',

'HNR', 'RPDE', 'DFA', 'PPE', 'sex\_0', 'sex\_1']]

In [7]:

# Generate the statistics of the columns for use in normalization of the data

train\_stats = train\_dataset1.describe()

train\_stats = train\_stats.transpose()

train\_stats

Out[7]:

count mean std min 25% 50% 75% max  
age 4700.0 64.792766 8.870401 36.000000 58.000000 65.000000 72.000000 85.000000  
test\_time 4700.0 93.399490 53.630411 -4.262500 46.852250 93.405000 139.367500 215.490000  
Jitter(%) 4700.0 0.006136 0.005612 0.000830 0.003560 0.004900 0.006770 0.099990  
Jitter(Abs) 4700.0 0.000044 0.000036 0.000002 0.000022 0.000034 0.000053 0.000396  
Jitter:RAP 4700.0 0.002969 0.003089 0.000330 0.001570 0.002235 0.003260 0.057540  
Jitter:PPQ5 4700.0 0.003271 0.003760 0.000430 0.001810 0.002480 0.003460 0.069560  
Jitter:DDP 4700.0 0.008908 0.009267 0.000980 0.004710 0.006705 0.009790 0.172630  
Shimmer 4700.0 0.033992 0.025922 0.003060 0.019020 0.027385 0.039810 0.268630  
Shimmer(dB) 4700.0 0.310487 0.231016 0.026000 0.175000 0.251000 0.363250 2.107000  
Shimmer:APQ3 4700.0 0.017125 0.013275 0.001610 0.009190 0.013615 0.020562 0.162670  
Shimmer:APQ5 4700.0 0.020151 0.016848 0.001940 0.010750 0.015785 0.023733 0.167020  
Shimmer:APQ11 4700.0 0.027508 0.020270 0.002490 0.015630 0.022685 0.032713 0.275460  
Shimmer:DDA 4700.0 0.051375 0.039826 0.004840 0.027567 0.040845 0.061683 0.488020  
NHR 4700.0 0.032116 0.060206 0.000304 0.010827 0.018403 0.031452 0.748260  
HNR 4700.0 21.704631 4.288853 1.659000 19.447750 21.973000 24.445250 37.187000  
RPDE 4700.0 0.542549 0.100212 0.151020 0.471235 0.543490 0.614335 0.966080  
DFA 4700.0 0.653015 0.070446 0.514040 0.596470 0.643285 0.710618 0.865600  
PPE 4700.0 0.219559 0.091506 0.021983 0.156470 0.205340 0.264017 0.731730  
sex\_0 4700.0 0.681489 0.465948 0.000000 0.000000 1.000000 1.000000 1.000000  
sex\_1 4700.0 0.318511 0.465948 0.000000 0.000000 0.000000 1.000000 1.000000

In [0]:

# Create the target variable

train\_labels = train\_dataset.pop('motor\_UPDRS')

test\_labels = test\_dataset.pop('motor\_UPDRS')

In [0]:

# Normalize the data by subtracting the mean and dividing by the standard deviation

def normalize(x):

return (x - train\_stats['mean']) / train\_stats['std']

# Create normalized training and test data

normalized\_train\_data = normalize(train\_dataset1)

normalized\_test\_data = normalize(test\_dataset1)

In [0]:

# Create a Deep Learning model with keras

model = tf.keras.Sequential([

keras.layers.Dense(6, activation=tf.nn.relu, input\_shape=[len(train\_dataset1.keys())]),

keras.layers.Dense(9, activation=tf.nn.relu),

keras.layers.Dense(6,activation=tf.nn.relu),

keras.layers.Dense(1)

])

# Use the Adam optimizer with a learning rate of 0.01

optimizer=keras.optimizers.Adam(lr=.01, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

# Set the metrics required to be Mean Absolute Error and Mean Squared Error.For regression, the loss is mean\_squared\_error

model.compile(loss='mean\_squared\_error',

optimizer=optimizer,

metrics=['mean\_absolute\_error', 'mean\_squared\_error'])

In [0]:

# Create a model

history=model.fit(

normalized\_train\_data, train\_labels,

epochs=1000, validation\_data = (normalized\_test\_data,test\_labels), verbose=0)

In [26]:

hist = pd.DataFrame(history.history)

hist['epoch'] = history.epoch

hist.tail()

Out[26]:

|  | **loss** | **mean\_absolute\_error** | **mean\_squared\_error** | **val\_loss** | **val\_mean\_absolute\_error** | **val\_mean\_squared\_error** | **epoch** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **995** | 15.773989 | 2.936990 | 15.773988 | 16.980803 | 3.028168 | 16.980803 | 995 |
| **996** | 15.238623 | 2.873420 | 15.238622 | 17.458752 | 3.101033 | 17.458752 | 996 |
| **997** | 15.437594 | 2.895500 | 15.437593 | 16.926016 | 2.971508 | 16.926018 | 997 |
| **998** | 15.867891 | 2.943521 | 15.867892 | 16.950249 | 2.985036 | 16.950249 | 998 |
| **999** | 15.846878 | 2.938914 | 15.846880 | 17.095623 | 3.014504 | 17.095625 | 999 |

In [30]:

def plot\_history(history):

hist = pd.DataFrame(history.history)

hist['epoch'] = history.epoch

plt.figure()

plt.xlabel('Epoch')

plt.ylabel('Mean Abs Error')

plt.plot(hist['epoch'], hist['mean\_absolute\_error'],

label='Train Error')

plt.plot(hist['epoch'], hist['val\_mean\_absolute\_error'],

label = 'Val Error')

plt.ylim([2,5])

plt.legend()

plt.figure()

plt.xlabel('Epoch')

plt.ylabel('Mean Square Error ')

plt.plot(hist['epoch'], hist['mean\_squared\_error'],

label='Train Error')

plt.plot(hist['epoch'], hist['val\_mean\_squared\_error'],

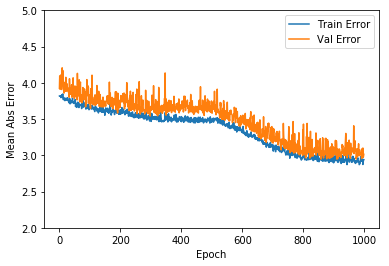
label = 'Val Error')

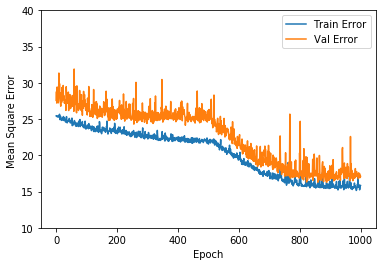
plt.ylim([10,40])

plt.legend()

plt.show()

plot\_history(history)





**Observation**

It can be seen that the mean absolute error is on an average about +/- 4.0. The validation error also is about the same. This can be reduced by playing around with the hyperparamaters and increasing the number of iterations

**1a. Multivariate Regression in Tensorflow – R**

# Install Tensorflow in RStudio

#install\_tensorflow()

# Install Keras

#install\_packages("keras")

library(tensorflow)

library(keras)

library(dplyr)

library(dummies)

## dummies-1.5.6 provided by Decision Patterns

library(tensorflow)

library(keras)

**Multivariate regression**

This code performs multivariate regression using Tensorflow and keras on the advent of Parkinson disease through sound recordings see [Parkinson Speech Dataset with Multiple Types of Sound Recordings Data Set](https://archive.ics.uci.edu/ml/datasets/Parkinson+Speech+Dataset+with++Multiple+Types+of+Sound+Recordings). The clinician’s motorUPDRS score has to be predicted from the set of features.

**Read the data**

# Download the Parkinson's data from UCI Machine Learning repository

dataset <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/telemonitoring/parkinsons\_updrs.data")

# Set the column names

names(dataset) <- c("subject","age", "sex", "test\_time","motor\_UPDRS","total\_UPDRS","Jitter","Jitter.Abs",

"Jitter.RAP","Jitter.PPQ5","Jitter.DDP","Shimmer", "Shimmer.dB", "Shimmer.APQ3",

"Shimmer.APQ5","Shimmer.APQ11","Shimmer.DDA", "NHR","HNR", "RPDE", "DFA","PPE")

# Remove the column 'subject' as it is not relevant to analysis

dataset1 <- subset(dataset, select = -c(subject))

# Make the column 'sex' as a factor for using dummies

dataset1$sex=as.factor(dataset1$sex)

# Add dummy variables for categorical cariable 'sex'

dataset2 <- dummy.data.frame(dataset1, sep = ".")

## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =

## FALSE): non-list contrasts argument ignored

dataset3 <- na.omit(dataset2)

**Split the data as training and test in 80/20**

## Split data 80% training and 20% test

sample\_size <- floor(0.8 \* nrow(dataset3))

## set the seed to make your partition reproducible

set.seed(12)

train\_index <- sample(seq\_len(nrow(dataset3)), size = sample\_size)

train\_dataset <- dataset3[train\_index, ]

test\_dataset <- dataset3[-train\_index, ]

train\_data <- train\_dataset %>% select(sex.0,sex.1,age, test\_time,Jitter,Jitter.Abs,Jitter.PPQ5,Jitter.DDP,

Shimmer, Shimmer.dB,Shimmer.APQ3,Shimmer.APQ11,

Shimmer.DDA,NHR,HNR,RPDE,DFA,PPE)

train\_labels <- select(train\_dataset,motor\_UPDRS)

test\_data <- test\_dataset %>% select(sex.0,sex.1,age, test\_time,Jitter,Jitter.Abs,Jitter.PPQ5,Jitter.DDP,

Shimmer, Shimmer.dB,Shimmer.APQ3,Shimmer.APQ11,

Shimmer.DDA,NHR,HNR,RPDE,DFA,PPE)

test\_labels <- select(test\_dataset,motor\_UPDRS)

**Normalize the data**

# Normalize the data by subtracting the mean and dividing by the standard deviation

normalize<-function(x) {

y<-(x - mean(x)) / sd(x)

return(y)

}

normalized\_train\_data <-apply(train\_data,2,normalize)

# Convert to matrix

train\_labels <- as.matrix(train\_labels)

normalized\_test\_data <- apply(test\_data,2,normalize)

test\_labels <- as.matrix(test\_labels)

**Create the Deep Learning Model**

model <- keras\_model\_sequential()

model %>%

layer\_dense(units = 6, activation = 'relu', input\_shape = dim(normalized\_train\_data)[2]) %>%

layer\_dense(units = 9, activation = 'relu') %>%

layer\_dense(units = 6, activation = 'relu') %>%

layer\_dense(units = 1)

# Set the metrics required to be Mean Absolute Error and Mean Squared Error.For regression, the loss is

# mean\_squared\_error

model %>% compile(

loss = 'mean\_squared\_error',

optimizer = optimizer\_rmsprop(),

metrics = c('mean\_absolute\_error','mean\_squared\_error')

)

# Fit the model

# Use the test data for validation

history <- model %>% fit(

normalized\_train\_data, train\_labels,

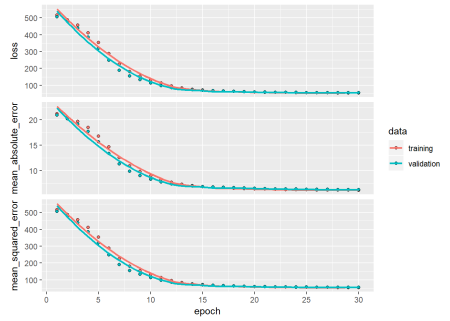
epochs = 30, batch\_size = 128,

validation\_data = list(normalized\_test\_data,test\_labels)

)

**Plot mean squared error, mean absolute error and loss for training data and test data**

plot(history)

Fig1  


**2. Binary classification in Tensorflow – Python**

This is a simple binary classification problem from UCI Machine Learning repository and deals with data on Breast cancer from the Univ. of Wisconsin [Breast Cancer Wisconsin (Diagnostic) Data Set](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29) **bold text**

In [31]:

import tensorflow as tf

from tensorflow import keras

import pandas as pd

# Read the data set from UCI ML site

dataset\_path = keras.utils.get\_file("breast-cancer-wisconsin.data", "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data")

raw\_dataset = pd.read\_csv(dataset\_path, sep=",", na\_values = "?", skipinitialspace=True,)

dataset = raw\_dataset.copy()

#Check for Null and drop

dataset.isna().sum()

dataset = dataset.dropna()

dataset.isna().sum()

# Set the column names

dataset.columns = ["id","thickness", "cellsize", "cellshape","adhesion","epicellsize",

"barenuclei","chromatin","normalnucleoli","mitoses","class"]

dataset.head()

Downloading data from https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data

24576/19889 [=====================================] - 0s 1us/step

id thickness cellsize cellshape adhesion epicellsize barenuclei chromatin normalnucleoli mitoses class

0 1002945 5 4 4 5 7 10.0 3 2 1 2

1 1015425 3 1 1 1 2 2.0 3 1 1 2

2 1016277 6 8 8 1 3 4.0 3 7 1 2

3 1017023 4 1 1 3 2 1.0 3 1 1 2

4 1017122 8 10 10 8 7 10.0 9 7 1 4

# Create a training/test set in the ratio 80/20

train\_dataset = dataset.sample(frac=0.8,random\_state=0)

test\_dataset = dataset.drop(train\_dataset.index)

# Set the training and test set

train\_dataset1= train\_dataset[['thickness','cellsize','cellshape','adhesion',

'epicellsize', 'barenuclei', 'chromatin', 'normalnucleoli','mitoses']]

test\_dataset1=test\_dataset[['thickness','cellsize','cellshape','adhesion',

'epicellsize', 'barenuclei', 'chromatin', 'normalnucleoli','mitoses']]

In [34]:

# Generate the stats for each column to be used for normalization

train\_stats = train\_dataset1.describe()

train\_stats = train\_stats.transpose()

train\_stats

Out[34]:

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **thickness** | 546.0 | 4.430403 | 2.812768 | 1.0 | 2.0 | 4.0 | 6.0 | 10.0 |
| **cellsize** | 546.0 | 3.179487 | 3.083668 | 1.0 | 1.0 | 1.0 | 5.0 | 10.0 |
| **cellshape** | 546.0 | 3.225275 | 3.005588 | 1.0 | 1.0 | 1.0 | 5.0 | 10.0 |
| **adhesion** | 546.0 | 2.921245 | 2.937144 | 1.0 | 1.0 | 1.0 | 4.0 | 10.0 |
| **epicellsize** | 546.0 | 3.261905 | 2.252643 | 1.0 | 2.0 | 2.0 | 4.0 | 10.0 |
| **barenuclei** | 546.0 | 3.560440 | 3.651946 | 1.0 | 1.0 | 1.0 | 7.0 | 10.0 |
| **chromatin** | 546.0 | 3.483516 | 2.492687 | 1.0 | 2.0 | 3.0 | 5.0 | 10.0 |
| **normalnucleoli** | 546.0 | 2.875458 | 3.064305 | 1.0 | 1.0 | 1.0 | 4.0 | 10.0 |
| **mitoses** | 546.0 | 1.609890 | 1.736762 | 1.0 | 1.0 | 1.0 | 1.0 | 10.0 |

In [0]:

# Create target variables

train\_labels = train\_dataset.pop('class')

test\_labels = test\_dataset.pop('class')

In [0]:

# Set the target variables as 0 or 1

train\_labels[train\_labels==2] =0 # benign

train\_labels[train\_labels==4] =1 # malignant

test\_labels[test\_labels==2] =0 # benign

test\_labels[test\_labels==4] =1 # malignant

In [0]:

# Normalize by subtracting mean and dividing by standard deviation

def normalize(x):

return (x - train\_stats['mean']) / train\_stats['std']

# Convert columns to numeric

train\_dataset1 = train\_dataset1.apply(pd.to\_numeric)

test\_dataset1 = test\_dataset1.apply(pd.to\_numeric)

# Normalize

normalized\_train\_data = normalize(train\_dataset1)

normalized\_test\_data = normalize(test\_dataset1)

In [0]:

# Create a model

model = tf.keras.Sequential([

keras.layers.Dense(6, activation=tf.nn.relu, input\_shape=[len(train\_dataset1.keys())]),

keras.layers.Dense(9, activation=tf.nn.relu),

keras.layers.Dense(6,activation=tf.nn.relu),

keras.layers.Dense(1)

])

# Use the RMSProp optimizer

optimizer = tf.keras.optimizers.RMSprop(0.01)

# Since this is binary classification use binary\_crossentropy

model.compile(loss='binary\_crossentropy',

optimizer=optimizer,

metrics=['acc'])

# Fit a model

history=model.fit(

normalized\_train\_data, train\_labels,

epochs=1000, validation\_data=(normalized\_test\_data,test\_labels), verbose=0)

In [55]:

hist = pd.DataFrame(history.history)

hist['epoch'] = history.epoch

hist.tail()

|  | **loss** | **acc** | **val\_loss** | **val\_acc** | **epoch** |
| --- | --- | --- | --- | --- | --- |
| **995** | 0.112499 | 0.992674 | 0.454739 | 0.970588 | 995 |
| **996** | 0.112499 | 0.992674 | 0.454739 | 0.970588 | 996 |
| **997** | 0.112499 | 0.992674 | 0.454739 | 0.970588 | 997 |
| **998** | 0.112499 | 0.992674 | 0.454739 | 0.970588 | 998 |
| **999** | 0.112499 | 0.992674 | 0.454739 | 0.970588 | 999 |

In [58]:

# Plot training and test accuracy

plt.plot(history.history['acc'])

plt.plot(history.history['val\_acc'])

plt.title('model accuracy')

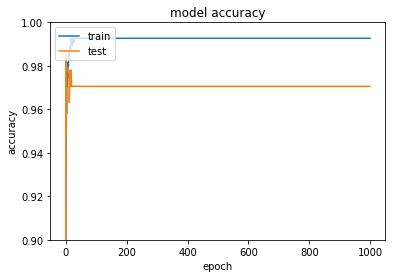
plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.ylim([0.9,1])

plt.show()



# Plot training and test loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

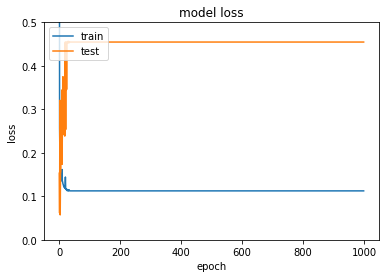
plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.ylim([0,0.5])

plt.show()



**2a. Binary classification in Tensorflow -R**

This is a simple binary classification problem from UCI Machine Learning repository and deals with data on Breast cancer from the Univ. of Wisconsin [Breast Cancer Wisconsin (Diagnostic) Data Set](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29)

# Read the data for Breast cancer (Wisconsin)

dataset <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data")

# Rename the columns

names(dataset) <- c("id","thickness", "cellsize", "cellshape","adhesion","epicellsize",

"barenuclei","chromatin","normalnucleoli","mitoses","class")

# Remove the columns id and class

dataset1 <- subset(dataset, select = -c(id, class))

dataset2 <- na.omit(dataset1)

# Convert the column to numeric

dataset2$barenuclei <- as.numeric(dataset2$barenuclei)

**Normalize the data**

train\_data <-apply(dataset2,2,normalize)

train\_labels <- as.matrix(select(dataset,class))

# Set the target variables as 0 or 1 as it binary classification

train\_labels[train\_labels==2,]=0

train\_labels[train\_labels==4,]=1

**Create the Deep Learning model**

model <- keras\_model\_sequential()

model %>%

layer\_dense(units = 6, activation = 'relu', input\_shape = dim(train\_data)[2]) %>%

layer\_dense(units = 9, activation = 'relu') %>%

layer\_dense(units = 6, activation = 'relu') %>%

layer\_dense(units = 1)

# Since this is a binary classification we use binary cross entropy

model %>% compile(

loss = 'binary\_crossentropy',

optimizer = optimizer\_rmsprop(),

metrics = c('accuracy') # Metrics is accuracy

)

**Fit the model. Use 20% of data for validation**

history <- model %>% fit(

train\_data, train\_labels,

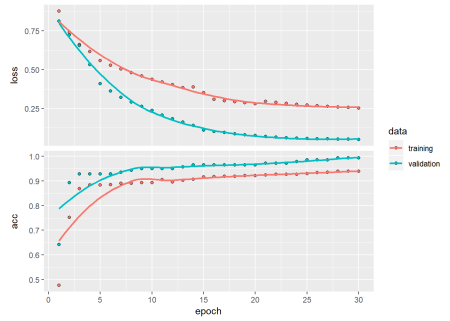
epochs = 30, batch\_size = 128,

validation\_split = 0.2

)

**Plot the accuracy and loss for training and validation data**

plot(history)



**3. MNIST in Tensorflow – Python**

This takes the famous MNIST handwritten digits . It ca be seen that Tensorflow and Keras make short work of this famous problem of the late 1980s

# Download MNIST data

mnist=tf.keras.datasets.mnist

# Set training and test data and labels

(training\_images,training\_labels),(test\_images,test\_labels)=mnist.load\_data()

print(training\_images.shape)

print(test\_images.shape)

(60000, 28, 28)

(10000, 28, 28)

In [61]:

# Plot a sample image from MNIST and show contents

import matplotlib.pyplot as plt

plt.imshow(training\_images[1])

print(training\_images[1])

[[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 51 159 253

159 50 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 48 238 252 252

252 237 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 54 227 253 252 239

233 252 57 6 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 10 60 224 252 253 252 202

84 252 253 122 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 163 252 252 252 253 252 252

96 189 253 167 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 51 238 253 253 190 114 253 228

47 79 255 168 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 48 238 252 252 179 12 75 121 21

0 0 253 243 50 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 38 165 253 233 208 84 0 0 0 0

0 0 253 252 165 0 0 0 0 0]

[ 0 0 0 0 0 0 0 7 178 252 240 71 19 28 0 0 0 0

0 0 253 252 195 0 0 0 0 0]

[ 0 0 0 0 0 0 0 57 252 252 63 0 0 0 0 0 0 0

0 0 253 252 195 0 0 0 0 0]

[ 0 0 0 0 0 0 0 198 253 190 0 0 0 0 0 0 0 0

0 0 255 253 196 0 0 0 0 0]

[ 0 0 0 0 0 0 76 246 252 112 0 0 0 0 0 0 0 0

0 0 253 252 148 0 0 0 0 0]

[ 0 0 0 0 0 0 85 252 230 25 0 0 0 0 0 0 0 0

7 135 253 186 12 0 0 0 0 0]

[ 0 0 0 0 0 0 85 252 223 0 0 0 0 0 0 0 0 7

131 252 225 71 0 0 0 0 0 0]

[ 0 0 0 0 0 0 85 252 145 0 0 0 0 0 0 0 48 165

252 173 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 86 253 225 0 0 0 0 0 0 114 238 253

162 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 85 252 249 146 48 29 85 178 225 253 223 167

56 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 85 252 252 252 229 215 252 252 252 196 130 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 28 199 252 252 253 252 252 233 145 0 0 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 25 128 252 253 252 141 37 0 0 0 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

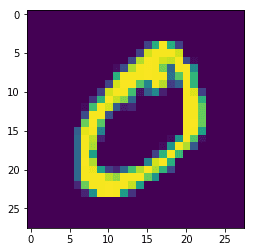
0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0]]



# Normalize the images by dividing by 255.0

training\_images = training\_images/255.0

test\_images = test\_images/255.0

# Create a Sequential Keras model

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1024,activation=tf.nn.relu),

tf.keras.layers.Dense(10,activation=tf.nn.softmax)])

model.compile(optimizer='adam',loss='sparse\_categorical\_crossentropy',metrics=['accuracy'])

In [68]:

history=model.fit(training\_images,training\_labels,validation\_data=(test\_images, test\_labels), epochs=5, verbose=1)

Train on 60000 samples, validate on 10000 samples

Epoch 1/5

60000/60000 [==============================] - 17s 291us/sample - loss: 0.0020 - acc: 0.9999 - val\_loss: 0.0719 - val\_acc: 0.9810

Epoch 2/5

60000/60000 [==============================] - 17s 284us/sample - loss: 0.0021 - acc: 0.9998 - val\_loss: 0.0705 - val\_acc: 0.9821

Epoch 3/5

60000/60000 [==============================] - 17s 286us/sample - loss: 0.0017 - acc: 0.9999 - val\_loss: 0.0729 - val\_acc: 0.9805

Epoch 4/5

60000/60000 [==============================] - 17s 284us/sample - loss: 0.0014 - acc: 0.9999 - val\_loss: 0.0762 - val\_acc: 0.9804

Epoch 5/5

60000/60000 [==============================] - 17s 280us/sample - loss: 0.0015 - acc: 0.9999 - val\_loss: 0.0735 - val\_acc: 0.9812

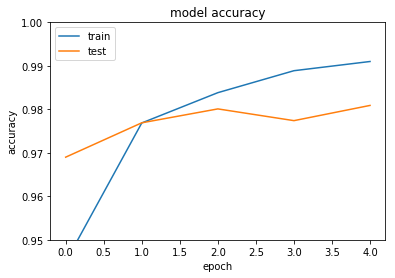
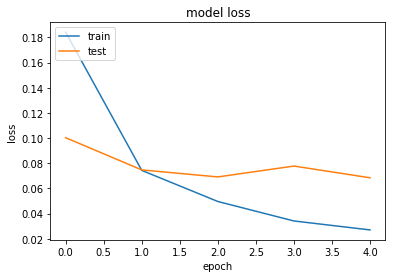
Fig 1  


Fig 2



**MNIST in Tensorflow – R**

The following code uses Tensorflow to learn MNIST’s handwritten digits ### Load MNIST data

mnist <- dataset\_mnist()

x\_train <- mnist$train$x

y\_train <- mnist$train$y

x\_test <- mnist$test$x

y\_test <- mnist$test$y

**Reshape and rescale**

# Reshape the array

x\_train <- array\_reshape(x\_train, c(nrow(x\_train), 784))

x\_test <- array\_reshape(x\_test, c(nrow(x\_test), 784))

# Rescale

x\_train <- x\_train / 255

x\_test <- x\_test / 255

**Convert out put to One Hot encoded format**

y\_train <- to\_categorical(y\_train, 10)

y\_test <- to\_categorical(y\_test, 10)

**Fit the model**

Use the softmax activation for recognizing 10 digits and categorical cross entropy for loss

model <- keras\_model\_sequential()

model %>%

layer\_dense(units = 256, activation = 'relu', input\_shape = c(784)) %>%

layer\_dense(units = 128, activation = 'relu') %>%

layer\_dense(units = 10, activation = 'softmax') # Use softmax

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_rmsprop(),

metrics = c('accuracy')

)

**Fit the model**

*Note*: A smaller number of epochs has been used. For better performance increase number of epochs

history <- model %>% fit(

x\_train, y\_train,

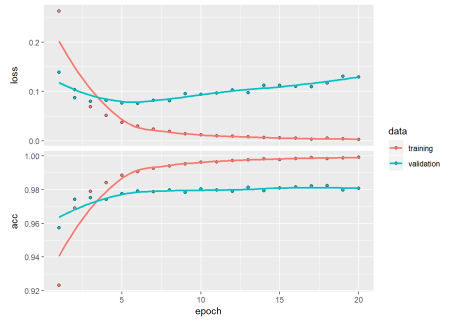
epochs = 5, batch\_size = 128,

validation\_data = list(x\_test,y\_test)

)

**Plot the accuracy and loss for training and test data**

plot(history)



Conclusion  
This post shows how to use Tensorflow and Keras in both Python & R  
Hope you have fun with Tensorflow!!