**Source Code**

**1.**from tensorboardcolab import \*

from \_\_future\_\_ import print\_function

import os

from datetime import time

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import keras

from keras import metrics

from keras import regularizers

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Activation

from keras.layers import Conv2D, MaxPooling2D

from keras.optimizers import Adam, RMSprop

from keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint

from keras.utils import plot\_model

from keras.models import load\_model

from sklearn.model\_selection import train\_test\_split

tbc=TensorBoardColab()

df = pd.read\_csv('Boston.csv')

kc\_data = pd.DataFrame(df, columns=["crim","zn","indus","chas","nox","rm","age","dis","rad","tax","ptratio","lstat","medv"

])

label\_col = 'medv'

#print(kc\_data.describe())

kc\_x\_train, kc\_x\_valid, kc\_y\_train, kc\_y\_valid = train\_test\_split(kc\_data.iloc[:,0:6], kc\_data.iloc[:,6],

test\_size=0.3, random\_state=87)

np.random.seed(155)

def norm\_statistics(df1, df2):

dfs = df1.append(df2)

minimum = np.min(dfs)

maximum = np.max(dfs)

mu = np.mean(dfs)

sigma = np.std(dfs)

return (minimum, maximum, mu, sigma)

def z\_score(col, stats):

m, M, mu, s = stats

df2 = pd.DataFrame()

for c in col.columns:

df2[c] = (col[c]-mu[c])/s[c]

return df2

stats = norm\_statistics(kc\_x\_train, kc\_x\_valid)

arr\_x\_train = np.array(z\_score(kc\_x\_train, stats))

arr\_y\_train = np.array(kc\_y\_train)

arr\_x\_valid = np.array(z\_score(kc\_x\_valid, stats))

arr\_y\_valid = np.array(kc\_y\_valid)

print('Training shape:', arr\_x\_train.shape)

print('ddd',arr\_y\_train.shape)

print('Training samples: ', arr\_x\_train.shape[0])

print('Validation samples: ', arr\_x\_valid.shape[0])

def basicmodel1(x\_size, y\_size):

t\_model = Sequential()

t\_model.add(Dense(100, activation="sigmoid", input\_shape=(x\_size,)))

t\_model.add(Dense(50, activation="tanh"))

t\_model.add(Dense(y\_size))

t\_model.compile(loss='mse',

optimizer=Adam(),

metrics=[metrics.mae])

return(t\_model)

def basicmodel2(x\_size, y\_size):

t\_model = Sequential()

t\_model.add(Dense(100, activation="sigmoid", input\_shape=(x\_size,)))

t\_model.add(Dropout(0.2))

t\_model.add(Dense(50, activation="tanh"))

t\_model.add(Dense(20, activation="relu"))

t\_model.add(Dense(y\_size))

keras.optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

t\_model.compile(loss='mse',

optimizer=Adam(),

metrics=[metrics.mae])

tensorboard = TensorBoard(log\_dir="logs/final1",histogram\_freq=0, write\_graph=True, write\_images=True)

return(t\_model)

model = basicmodel2(arr\_x\_train.shape[1], 1)

model.summary()

epochs = 10

batch\_size =32

history = model.fit(arr\_x\_train, arr\_y\_train,

batch\_size=batch\_size,

epochs=epochs,

shuffle=True,

verbose=2, # Change it to 2, if wished to observe execution

validation\_data=(arr\_x\_valid, arr\_y\_valid),callbacks=[TensorBoardColabCallback(tbc)])

train\_score = model.evaluate(arr\_x\_train, arr\_y\_train, verbose=0)

valid\_score = model.evaluate(arr\_x\_valid, arr\_y\_valid, verbose=0)

print('Train MAE: ', round(train\_score[1], 4), ', Train Loss: ', round(train\_score[0], 4))

print('Val MAE: ', round(valid\_score[1], 4), ', Val Loss: ', round(valid\_score[0], 4))

keras\_callbacks = [

ModelCheckpoint('/tmp/keras\_checkpoints/model.{epoch:02d}-{val\_loss:.2f}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=2),

ModelCheckpoint('/tmp/keras\_checkpoints/model.{epoch:02d}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=0),

TensorBoard(log\_dir='./model\_3', histogram\_freq=0, write\_graph=True, write\_images=True, embeddings\_freq=0, embeddings\_layer\_names=None, embeddings\_metadata=None),

EarlyStopping(monitor='val\_mean\_absolute\_error', patience=20, verbose=0)

]

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

# summarize history for MAE

plt.subplot(211)

plt.plot(h['mean\_absolute\_error'])

plt.plot(h['val\_mean\_absolute\_error'])

plt.title('Training vs Validation MAE')

plt.ylabel('MAE')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

# summarize history for loss

plt.subplot(212)

plt.plot(h['loss'])

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

plt.title('Training vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

# Plot it all in IPython (non-interactive)

plt.draw()

plt.show()

return

**2.** from tensorboardcolab import \*

from \_\_future\_\_ import print\_function

import os

from datetime import time

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import keras

from keras import metrics

# from keras.regularizers import l1l2

# from keras import regularizers

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Activation

from keras.layers import Conv2D, MaxPooling2D

from keras.optimizers import Adam, RMSprop

from keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint

from keras.utils import plot\_model

from keras.models import load\_model

from sklearn.model\_selection import train\_test\_split

tbc = TensorBoardColab()

df = pd.read\_csv('heart.csv')

kc\_data = pd.DataFrame(df,

columns=["age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach", "exang", "oldpeak",

"slope", "ca", "thal", "target"])

label\_col = 'ca'

print(kc\_data.describe())

kc\_X\_train, kc\_X\_valid, kc\_Y\_train, kc\_Y\_valid = train\_test\_split(kc\_data.iloc[:, 0:13], kc\_data.iloc[:, 13],test\_size=0.3, random\_state=87)

np.random.seed(155)

def norm\_stats(df1, df2):

dfs = df1.append(df2)

minimum = np.min(dfs)

maximum = np.max(dfs)

mu = np.mean(dfs)

sigma = np.std(dfs)

return (minimum, maximum, mu, sigma)

def z\_score(col, stats):

m, M, mu, s = stats

df2 = pd.DataFrame()

for c in col.columns:

df2[c] = (col[c] - mu[c]) / s[c]

return df2

stats = norm\_stats(kc\_X\_train, kc\_X\_valid)

arr\_X\_train = np.array(z\_score(kc\_X\_train, stats))

arr\_Y\_train = np.array(kc\_Y\_train)

arr\_X\_valid = np.array(z\_score(kc\_X\_valid, stats))

arr\_Y\_valid = np.array(kc\_Y\_valid)

print('Training shape:', arr\_X\_train.shape)

print('ddd', arr\_Y\_train.shape)

print('Training samples: ', arr\_X\_train.shape[0])

print('Validation samples: ', arr\_X\_valid.shape[0])

def model1(X\_size, Y\_size):

model = Sequential()

model.add(Dense(100, activation='tanh', input\_shape=(X\_size,)))

# model.compile(optimizer='rmsprop', loss='binary\_crossentropy')

# model.add(Dense(100, activation="tanh", input\_shape=(X\_size,)))

model.add(Dense(50, activation="tanh"))

model.add(Dense(Y\_size))

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

optimizer='rmsprop',

metrics=[metrics.mae])

return (model)

# model2 is different from basic\_model\_1 but doing the same task with different structure

def model2(X\_size, Y\_size):

# reg = l1l2(l1=0.01, l2=0.01)

model = Sequential()

model.add(Dense(100, activation='tanh', input\_shape=(X\_size,)))

model.add(Dropout(0.2))

model.add(Dense(50, activation='tanh'))

model.add(Dense(20, activation='tanh'))

model.add(Dense(Y\_size))

keras.optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

optimizers = ['rmsprop', 'adam']

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

optimizer=Adam(),

metrics=[metrics.mae])

return (model)

model = model2(arr\_X\_train.shape[1], 1)

model.summary()

epochs = 15

batch\_size = 32

history = model.fit(arr\_X\_train, arr\_Y\_train,

batch\_size=batch\_size,

epochs=epochs,

shuffle=True,

verbose=2, # Change it to 2, if wished to observe execution

validation\_data=(arr\_X\_valid, arr\_Y\_valid), callbacks=[TensorBoardColabCallback(tbc)])

train\_score = model.evaluate(arr\_X\_train, arr\_Y\_train, verbose=0)

valid\_score = model.evaluate(arr\_X\_valid, arr\_Y\_valid, verbose=0)

print('Train MAE: ', round(train\_score[1], 4), ', Train Loss: ', round(train\_score[0], 4))

print('Val MAE: ', round(valid\_score[1], 4), ', Val Loss: ', round(valid\_score[0], 4))

keras\_callbacks = [

ModelCheckpoint('/tmp/keras\_checkpoints/model.{epoch:02d}-{val\_loss:.2f}.hdf5', monitor='val\_loss',

save\_best\_only=True, verbose=2),

ModelCheckpoint('/tmp/keras\_checkpoints/model.{epoch:02d}.hdf5', monitor='val\_loss', save\_best\_only=True,

verbose=0),

TensorBoard(log\_dir='./model\_3', histogram\_freq=0, write\_graph=True, write\_images=True, embeddings\_freq=0,embeddings\_layer\_names=None, embeddings\_metadata=None),

EarlyStopping(monitor='val\_mean\_absolute\_error', patience=20, verbose=0)

]

def plot\_histogram(h, Xsize=6, Ysize=10):

# Preparing plot

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [Xsize, Ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

# summarizing history for MAE

plt.subplot(211)

plt.plot(h['mean\_absolute\_error'])

plt.plot(h['val\_mean\_absolute\_error'])

plt.title('Training vs Validation MAE')

plt.ylabel('MAE')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

# summarizing history for loss

plt.subplot(212)

plt.plot(h['loss'])

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

plt.title('Training vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

# Plot it in IPython (non-interactive)

plt.draw()

plt.show()

return

plot\_histogram(history.history, Xsize=8, Ysize=12)

score = model.evaluate(arr\_X\_valid, arr\_Y\_valid)

print('test accuracy', score[1])

# Creating hyperparameter space

epochs = [5, 10]

batches = [5, 10, 100]

optimizers = ['rmsprop', 'adam']

# Creating hyperparameter options

hyperparameters = dict(optimizer=optimizers, epochs=epochs, batch\_size=batches)

**3.** import numpy as np # linear algebra

import pandas as pd # data processing

from subprocess import check\_output

# Any results we write to the current directory are saved as output.

DATA\_FILE = 'spam.csv'

df = pd.read\_csv(DATA\_FILE, encoding='latin-1')

print(df.head())

tags = df.v1

texts = df.v2

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation

from keras.layers import Embedding

from keras.layers import Conv1D, GlobalMaxPooling1D

from keras.preprocessing.text import Tokenizer

from keras.preprocessing import sequence

from sklearn.preprocessing import LabelEncoder

import time

from keras import metrics

print('import done')

num\_max = 1000

# preprocessing

le = LabelEncoder()

tags = le.fit\_transform(tags)

tok = Tokenizer(num\_words=num\_max)

tok.fit\_on\_texts(texts)

mat\_texts = tok.texts\_to\_matrix(texts, mode='count')

print(tags[:5])

print(mat\_texts[:5])

print(tags.shape, mat\_texts.shape)

# trying a simple model first

def get\_simple\_model():

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(num\_max,)))

model.add(Dropout(0.2))

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

model.summary()

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

optimizer='adam',

metrics=['acc', metrics.binary\_accuracy])

print('compile done')

return model

def check\_model(model, x, y):

model.fit(x, y, batch\_size=32, epochs=10, verbose=1, validation\_split=0.2)

m = get\_simple\_model()

check\_model(m, mat\_texts, tags)

# for cnn preprocessing

max\_len = 100

cnn\_texts\_seq = tok.texts\_to\_sequences(texts)

print(cnn\_texts\_seq[0])

cnn\_texts\_mat = sequence.pad\_sequences(cnn\_texts\_seq, maxlen=max\_len)

print(cnn\_texts\_mat[0])

print(cnn\_texts\_mat.shape)

def get\_cnn\_modelv1():

model = Sequential()

# start off with an efficient embedding layer which maps

# our vocab indices into embedding\_dims dimensions

# 1000 is num\_max

model.add(Embedding(1000,

20,

input\_length=max\_len))

model.add(Dropout(0.1))

model.add(Conv1D(64,

3,

padding='valid',

activation='relu',

strides=1))

model.add(GlobalMaxPooling1D())

model.add(Dense(256))

model.add(Dropout(0.2))

model.add(Activation('tanh'))

model.add(Dense(1))

model.add(Activation('relu'))

model.summary()

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

optimizer='adam',

metrics=['acc', metrics.binary\_accuracy])

return model

m = get\_cnn\_modelv1()

check\_model(m, cnn\_texts\_mat, tags)

def get\_cnn\_modelv2(): # added embed

model = Sequential()

model.add(Embedding(1000,

50,

input\_length=max\_len))

model.add(Dropout(0.1))

model.add(Conv1D(64,

3,

padding='valid',

activation='tanh',

strides=1))

model.add(GlobalMaxPooling1D())

model.add(Dense(256))

model.add(Dropout(0.2))

model.add(Activation('tanh'))

model.add(Dense(1))

model.add(Activation('relu'))

model.summary()

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

optimizer='adam',

metrics=['acc', metrics.binary\_accuracy])

return model

m = get\_cnn\_modelv2()

check\_model(m, cnn\_texts\_mat, tags)

def get\_cnn\_modelv3(): # added filter

model = Sequential()

model.add(Embedding(1000,

20,

input\_length=max\_len))

model.add(Dropout(0.1))

model.add(Conv1D(256,

3,

padding='valid',

activation='tanh',

strides=1))

model.add(GlobalMaxPooling1D())

model.add(Dense(256))

model.add(Dropout(0.2))

model.add(Activation('relu'))

model.add(Dense(1))

model.add(Activation('tanh'))

model.summary()

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

optimizer='adam',

metrics=['acc', metrics.binary\_accuracy])

return model

m = get\_cnn\_modelv3()

check\_model(m, cnn\_texts\_mat, tags)

**4.** import numpy as np # linear algebra

import pandas as pd # processing data

import matplotlib.pyplot as plt

# we need to fit model with sequence of tokens with specific length

from keras.preprocessing.sequence import pad\_sequences

from keras.preprocessing.text import Tokenizer

from keras.models import Sequential

# normal LSTM/GRU

from keras.layers import Dense, Embedding, GRU, LSTM, Dropout, Bidirectional

from keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint

from keras.optimizers import Adam, rmsprop

# keras wrapper for k-fold cross-validation

from keras.wrappers.scikit\_learn import KerasClassifier

# normsl cross validation

from sklearn.model\_selection import cross\_val\_score, train\_test\_split

# cross validation for tuning hyperparameter

from sklearn.model\_selection import GridSearchCV

import os

x\_raw = []

y\_raw = []

with open("spam.csv", encoding = "ISO-8859-1") as f:

for line in f:

y\_raw.append(line.split()[0])

x\_raw.append(' '.join(i for i in line.split()[1:]))

y = [1 if i=='ham' else 0 for i in y\_raw]

print(max(len(s) for s in x\_raw))

print(min(len(s)for s in x\_raw))

sorted\_X = sorted(len(s) for s in x\_raw)

print(sorted\_X[len(sorted\_X) // 2])

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(x\_raw)

sequences = tokenizer.texts\_to\_sequences(x\_raw)

vocab\_size = len(tokenizer.word\_index)+1

print(vocab\_size)

# divide sum of length of all sequences by number of all sequences to find averge length of each sequence

sum([len(x) for x in sequences]) // len(sequences)

pad = 'post'

max\_len = 25

embedding\_size = 100

batch\_size = 32

sequences = pad\_sequences(sequences, maxlen=max\_len, padding=pad, truncating=pad)

sequences.shape

X\_train, X\_test, y\_train, y\_test = train\_test\_split(sequences, y, test\_size = 0.3, random\_state= 0)

model = Sequential()

model.add(Embedding(input\_dim=vocab\_size, output\_dim=embedding\_size, input\_length=max\_len))

model.add(Dropout(0.1))

model.add(LSTM(140, return\_sequences=False))

model.add(Dropout(0.1))

model.add(Dense(1, activation='tanh', name='Classification'))

model.summary()

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

# model.fit(X\_train, y\_train, epochs=n\_epochs, batch\_size=batch\_size, validation\_split=0.1, callbacks=[callback\_early\_stopping])

n\_epochs = 10

results = model.fit(X\_train, y\_train, epochs=n\_epochs, batch\_size=batch\_size, validation\_split=0.2, verbose=1)

# model evaluation

eval\_ = model.evaluate(X\_test, y\_test)

print(eval\_[0], eval\_[1]) # loss / accuracy

**6.** import numpy as np

import keras as K

import tensorflow as tf

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'

import matplotlib.pyplot as plt

def main():

# starting

print("\MNIST image recognition using Keras/TensorFlow ")

np.random.seed(1)

tf.set\_random\_seed(1)

if \_\_name\_\_ == "\_\_main\_\_":

main()

# loading data

print("Loading Keras version MNIST data into memory \n")

(train\_x, train\_y), (test\_x, test\_y) = K.datasets.mnist.load\_data()

train\_x = train\_x.reshape(60000, 28, 28, 1).astype(np.float32)

test\_x = test\_x.reshape(10000, 28, 28, 1).astype(np.float32)

train\_x /= 255;

test\_x /= 255

train\_y = K.utils.to\_categorical(train\_y, 10).astype(np.float32)

test\_y = K.utils.to\_categorical(test\_y, 10).astype(np.float32)

# defining model

init = K.initializers.glorot\_uniform(seed=1)

model = K.models.Sequential()

model.add(K.layers.Conv2D(filters=32, kernel\_size=(3, 3),

strides=(1, 1), padding='valid', kernel\_initializer=init,

activation='relu', input\_shape=(28, 28, 1)))

model.add(K.layers.Conv2D(filters=64, kernel\_size=(3, 3),

strides=(1, 1), padding='valid', kernel\_initializer=init,

activation='tanh'))

model.add(K.layers.MaxPooling2D(pool\_size=(2, 2)))

model.add(K.layers.Dropout(0.2))

model.add(K.layers.Flatten())

model.add(K.layers.Dense(units=100, kernel\_initializer=init,

activation='sigmoid'))

model.add(K.layers.Dropout(0.2))

model.add(K.layers.Dense(units=10, kernel\_initializer=init,

activation='softmax'))

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

optimizer='adadelta', metrics=['acc'])

# training model

bat\_size = 64

max\_epochs = 3

print("Starting training ")

model.fit(train\_x, train\_y, batch\_size=bat\_size,

epochs=max\_epochs, verbose=1)

print("Training complete")

# evaluating model

loss\_acc = model.evaluate(test\_x, test\_y, verbose=0)

print("\nTest data loss = %0.4f accuracy = %0.2f%%" % \

(loss\_acc[0], loss\_acc[1] \* 100))