National University of Computer and Emerging Sciences



Lab Manual

CL461-Artificial Intelligence Lab

Department of Computer Science

FAST-NU, Lahore, Pakistan

Introduction

There are many deep learning libraries out there, but the most popular ones are TensorFlow, Keras, and PyTorch. We will be focusing on Keras in this guide.

Keras is a high-level neural networks API, written in Python, and can run on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. The advantages of using Keras emanates from the fact that it focuses on being user-friendly, modular, and extensible.

In this guide, we will focus on how to use the Keras library to build classification models.

### Classification with Keras

Classification is a type of supervised machine learning algorithm used to predict a categorical label. A few useful examples of classification include predicting whether a customer will churn or not, classifying emails into spam or not, or whether a bank loan will default or not.

The basic architecture of the deep learning neural network, which we will be following, consists of three main components.

1. Input Layer: This is where the training observations are fed.
2. Hidden Layers: These are the intermediate layers between the input and output layers. The deep neural network learns about the relationships involved in data in this component.
3. Output Layer: This is the layer where the final output is extracted from what’s happening in the previous two layers.

## Problem Statement

Diabetes is a serious health issue which causes an increase in blood sugar. Many complications occur if diabetes remains untreated and unidentified.

The aim of this guide is to build a classification model to detect diabetes. We will be using the diabetes dataset which contains 768 observations and 9 variables, as described below:

* pregnancies - Number of times pregnant
* glucose - Plasma glucose concentration
* diastolic - diastolic blood pressure (mm Hg)
* triceps - Skinfold thickness (mm)
* insulin - Hour serum insulin (mu U/ml)
* bmi – Basal metabolic rate (weight in kg/height in m)
* dpf - Diabetes pedigree function
* age - Age in years
* diabetes - 1 represents the presence of diabetes while 0 represents the absence of it. This is the target variable.

## Steps

*Step 1 - Loading the required libraries and modules*

*Step 2 - Loading the data and performing basic data checks*

*Step 3 - Creating arrays for the features and the response variable*

*Step 4 - Creating the Training and Test datasets*

*Step 5 - Define, compile, and fit the Keras classification model*

*Step 6 - Predict on the test data and compute evaluation metrics*

The following sections will cover these steps.

### Step 1 - Loading the Required Libraries and Modules

# Import required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import sklearn

# Import necessary modules

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

from sklearn.metrics import mean\_squared\_error

from math import sqrt

# Keras specific

import keras

from keras.models import Sequential

from keras.layers import Dense

from keras.utils import to\_categorical

### Step 2 - Reading the Data and Performing Basic Data Checks

The *first line* of code reads in the data as pandas dataframe, while the *second line* of code prints the shape - 768 observations of 9 variables. The *third line* gives summary statistics of the numerical variables.

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

print(df.shape)

df.describe()

### Step 3 - Creating Arrays for the Features and the Response Variable.

The *first line* of code creates an object of the target variable, while the *second line* of code gives the list of all the features after excluding the target variable, 'Outcome'.

The *third line* does normalization of the predictors via scaling between 0 and 1. This is needed to eliminate the influence of the predictor's units and magnitude on the modelling process.

The *fourth line* displays the summary of the normalized data. The target variable remains unchanged.

target\_column = ['Outcome']

predictors = list(set(list(df.columns))-set(target\_column))

list(set(list(df.columns))-set(target\_column))

df[predictors] = df[predictors]/df[predictors].max()

df.describe()

### 

### 

### Step 4 - Creating the Training and Test Datasets

The *first couple of lines* creates arrays of independent (X) and dependent (y) variables, respectively. The *third line* splits the data into training and test datasets, with 30% of the observations in the test set. The *fourth line* of code prints the shape of the training set (537 observations of 8 variables) and test set (231 observations of 8 variables).

X = df[predictors].values

y = df[target\_column].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=40)

print(X\_train.shape); print(X\_test.shape)

Since our target variable represents a binary category which has been coded as numbers 0 and 1, we will have to encode it. We can easily achieve that using the "to\_categorical" function from the Keras utilities package. The *two lines* of code below accomplishes that in both training and test datasets.

# one hot encode outputs

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

count\_classes = y\_test.shape[1]

print(count\_classes)

### Step 5 - Define, Compile, and Fit the Keras Classification Model

We will start by setting up the model. The *first line* of code calls for the Sequential constructor. We are using the Sequential model because our network consists of a linear stack of layers.

The *second line* of code represents the input layer which specifies the activation function and the number of input dimensions, which in our case is 8 predictors. Then we repeat the same process in the *third and fourth line* of codes for the two hidden layers, but this time without the input\_dim parameter. The activation function used is a rectified linear unit, or ReLU. ReLU is the most widely used activation function because it is nonlinear, and has the ability to not activate all the neurons at the same time.

The *fifth line* of code creates the output layer with two nodes because there are two output classes, 0 and 1. We use 'softmax' as the activation function for the output layer, so that the sum of the predicted values from all the neurons in the output layer adds up to one.

In the above lines of codes, we have defined our deep learning model architecture. But before we can start training the model, we will configure the learning process. This is done in the last line of code using the model.compile() function.

In defining our compiler, we will use 'categorical cross-entropy' as our loss measure, 'adam' as the optimizer algorithm, and 'accuracy' as the evaluation metric. The main advantage of the "adam" optimizer is that we don't need to specify the learning rate, as is the case with gradient descent. Using “adam” will, thereby, save us the task of optimizing the learning rate for our model.

model = Sequential()

model.add(Dense(500, activation='relu', input\_dim=8))

model.add(Dense(100, activation='relu'))

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

model.add(Dense(2, activation='softmax'))

# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

Now we are ready to build the model which is done in the code below. We also provide the argument, epochs, which represents the number of training iterations. We have taken 20 epochs.

# build the model

model.fit(X\_train, y\_train, epochs=20)

### Step 6 - Predict on the Test Data and Compute Evaluation Metrics;

The *first line* of code predicts on the train data, while the *second line* evaluates the model, and the *third line* prints the accuracy and error on the training data.

The same is repeated in the *fourth, fifth and sixth lines* of code which is performed on the test data.

pred\_train= model.predict(X\_train)

scores = model.evaluate(X\_train, y\_train, verbose=0)

print('Accuracy on training data: {}% \n Error on training data: {}'.format(scores[1], 1 - scores[1]))

pred\_test= model.predict(X\_test)

scores2 = model.evaluate(X\_test, y\_test, verbose=0)

print('Accuracy on test data: {}% \n Error on test data: {}'.format(scores2[1], 1 - scores2[1]))