Diabetes dataset

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [3]:

```
from tensorflow import keras
```

In [4]:

```
from tensorflow.keras import layers
```

In [5]:

```
# see keras version
keras.__version__
```

Out[5]:

'2.12.0'

In [6]:

```
#pd.set_option('display.max_columns', None)
```

Get data

In [7]:

```
df = pd.read_csv('diabetes.csv')
df
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PedigreeFunc	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

```
The diabetes.csv comes from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict whether or not a
patient has diabetes, based on certain diagnostic measurements. All patients here are females at least 21 years old of Pima India heritage. The
variables are:

Pregnancies: the number of times pregnant
Glucose Plasma: glucose concentration at 2 hours in an oral glucose tolerlance test.
BloodPressure: Diastolic blood pressure (mm Hg)
SkinThickness: Triceps skin fold thickness (mm)
Insulin: 2-Hour serum insulin (mu U/ml)
BMI: Body mass index(weight in kg/(height in m)^2)
DiabetesPedigreeFunction
Age
```

```
Outcome(0 or 1)
```

In [8]:

```
df.isnull().sum() # It has no missing values, because all missing values are replaced by 0.
```

Out[8]:

```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
PedigreeFunc     0
Age              0
Outcome          0
dtype: int64
```

In [8]:

```
# How many diabetes patients?
```

In [10]:

```
df.Outcome.value_counts()
```

Out[10]:

```
0    500
1    268
Name: Outcome, dtype: int64
```

In [11]:

```
df.Outcome.value_counts()/768
```

Out[11]:

```
0    0.651042
1    0.348958
Name: Outcome, dtype: float64
```

In [11]:

```
# There are 35% diabetes patients
```

Data Preparation

In [12]:

```
# looking for missing values and outliers)
# Some columns have entries equal to zero
# For some columns that is not possible (i.e.,BMI, Insulin)
```

In [12]:

```
for col in df.columns:
    zeros = df.loc[df[col]==0].shape[0]
    print(col+": "+str(zeros))
```

```
Pregnancies: 111
Glucose: 5
BloodPressure: 35
SkinThickness: 227
Insulin: 374
BMI: 11
PedigreeFunc: 0
Age: 0
Outcome: 500
```

In [14]:

```
# Imputation
```

In [15]:

```
# replace the zeros with nan
```

In [13]:

```
df['Glucose'] = df['Glucose'].replace(0, np.nan)
df['BloodPressure'] = df['BloodPressure'].replace(0, np.nan)
df['SkinThickness'] = df['SkinThickness'].replace(0, np.nan)
df['Insulin'] = df['Insulin'].replace(0, np.nan)
df['BMI'] = df['BMI'].replace(0, np.nan)
```

In [14]:

```
df.isnull().sum()
```

Out[14]:

```
Pregnancies        0
Glucose            5
BloodPressure     35
SkinThickness    227
Insulin          374
BMI               11
PedigreeFunc       0
Age                0
Outcome            0
dtype: int64
```

In [17]:

```
# replace the nan with the average of that column
```

In [15]:

```
df['Glucose'] = df['Glucose'].fillna(df['Glucose'].mean())
df['BloodPressure'] = df['BloodPressure'].fillna(df['BloodPressure'].mean())
df['SkinThickness'] = df['SkinThickness'].fillna(df['SkinThickness'].mean())
df['Insulin'] = df['Insulin'].fillna(df['Insulin'].mean())
df['BMI'] = df['BMI'].fillna(df['BMI'].mean())
```

Split data

In [16]:

```
X = df.drop(['Outcome'],axis = 1)
Y = df.Outcome
```

In [17]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, stratify = Y,
                                                    test_size=0.2,
                                                    random_state=1)
```

In [18]:

```
y_train.shape
```

Out[18]:

```
(614,)
```

In [19]:

```
y_test.shape
```

Out[19]:

```
(154,)
```

```
Scale the data
```

In [20]:

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:

```

```

Build Model

Build the Dense Neural Network (DNN) with 2 hidden layers

In [21]:

```
network1 = keras.Sequential([
    layers.Dense(32,activation='relu'), # first hidden layer
    layers.Dense(16,activation='relu'), # second hidden layer
    layers.Dense(1,activation='sigmoid') # output layer
])
```

Compile Model

In [22]:

```
# use loss function binary_crossentropy
network1.compile(optimizer='rmsprop',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])
```

train(fit) model

In [23]:

```
n_epochs = 55
```

In [24]:

```
history = network1.fit(X_train_scaled, y_train,
                       epochs=n_epochs, batch_size=20,
                       validation_split = 0.20,
                       verbose = 0,
                      )
```

In [69]:

```
# values for train loss and train accuracy are shown at each step
```

In [25]:

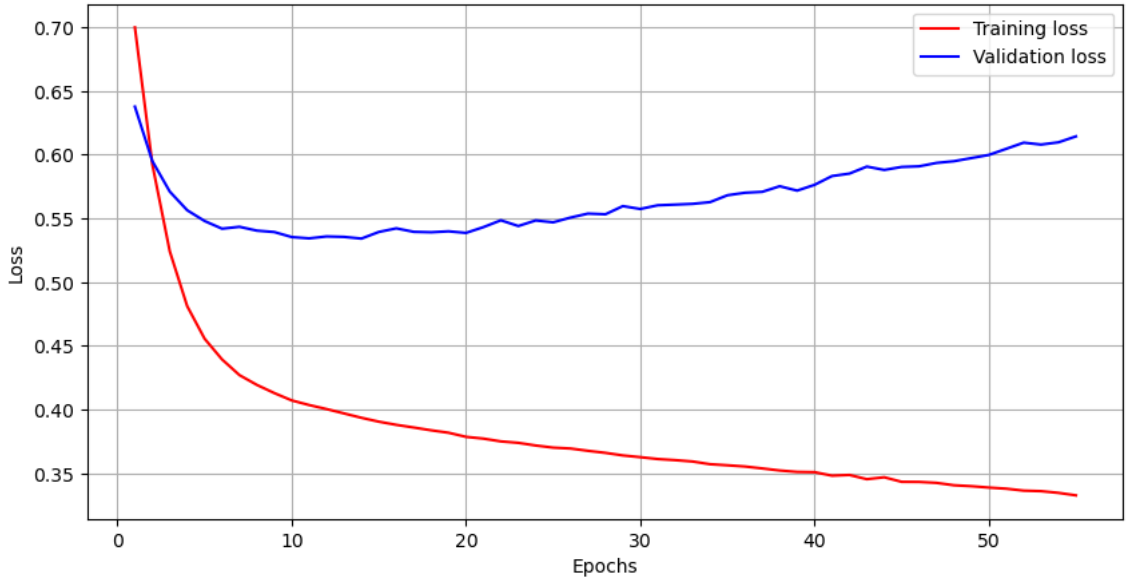
```
loss = history.history["loss"]
val_loss = history.history["val_loss"]
```

In [26]:

```
epochs = range(1,n_epochs+1)
```

In [27]:

```
plt.figure(figsize=(10,5))
plt.plot(epochs, loss, "r",
         label="Training loss")
plt.plot(epochs, val_loss, "b",
         label="Validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
```



In [73]:

```
# The model starts overfitting after x epochs
```

In [74]:

```
## Retrain model with all train data (with x epochs)
```

In [28]:

```
model = keras.Sequential([
    layers.Dense(32,activation='relu'),
    layers.Dense(16,activation='relu'),
    layers.Dense(1,activation='sigmoid')
])

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_scaled, y_train, epochs=10, batch_size=20)
```

```
Epoch 1/10
31/31 [==============================] - 1s 3ms/step - loss: 0.6612 - accuracy: 0.6270
Epoch 2/10
31/31 [==============================] - 0s 3ms/step - loss: 0.5824 - accuracy: 0.7231
Epoch 3/10
31/31 [==============================] - 0s 3ms/step - loss: 0.5307 - accuracy: 0.7427
Epoch 4/10
31/31 [==============================] - 0s 4ms/step - loss: 0.4952 - accuracy: 0.7573
Epoch 5/10
31/31 [==============================] - 0s 4ms/step - loss: 0.4734 - accuracy: 0.7704
Epoch 6/10
31/31 [==============================] - 0s 3ms/step - loss: 0.4589 - accuracy: 0.7736
Epoch 7/10
31/31 [==============================] - 0s 4ms/step - loss: 0.4494 - accuracy: 0.7785
Epoch 8/10
31/31 [==============================] - 0s 3ms/step - loss: 0.4439 - accuracy: 0.7769
Epoch 9/10
31/31 [==============================] - 0s 3ms/step - loss: 0.4382 - accuracy: 0.7801
Epoch 10/10
31/31 [==============================] - 0s 3ms/step - loss: 0.4358 - accuracy: 0.7769
```

Out[28]:

```
<keras.callbacks.History at 0x27a897bcdc0>
```

test model

In [29]:

```
test_loss,test_acc = model.evaluate(X_test_scaled, y_test)
```

```
5/5 [==============================] - 0s 3ms/step - loss: 0.5030 - accuracy: 0.7597
```

In [30]:

```
test_acc  # 0.7597402334213257
```

Out[30]:

```
0.7597402334213257
```

In [31]:

```
test_loss # 0.5030236840248108
```

Out[31]:

```
0.5030236840248108
```