Report


# NEURAL NETS PROJECT

## DIABETE PREDICTION

Mini Project using neural nets to solve
classification problem

Student: Bassole Cedric-Francois
University:ISET BIZERTE
Class:Master in Robotics and Artificial Intelligence
Course: Neural Nets and Fuzzy logic
2023 Avril 7

# Abstract

Neural nets take inspiration from the learning process occurring in human brains. They consists of an artificial network of functions, called parameters, which allows the computer to learn, and to fine tune itself, by analyzing new data
Neural networks are also able to handle large amounts of data and can learn from data without being explicitly programmed with a set of rules or a decision tree. This allows them to be very flexible and adaptable, and makes them well-suited for tasks that are difficult to define using traditional programming techniques.

”The work presented in this report is our own and was obtained by my clasmate and me ”except the dataset .We obtained it from kaggle dataframe.

# Contents

# 1. Introduction

Diabetes is a chronic medical condition which is estimated to affect 415 million people in the world. 5 million deaths a year can be attributed to diabetes-related complications. Type 2 diabetes can be prevented and reversed if diagnosed early. We can use machine learning to predict diabetes in patients using vital statistics by using datasets where patients have been diagnosed. By training a neural network, we can use it to make predictions on new patients

In order to run through the example below, y ou must have installed Python, setup a notebook,and create a virtual environnement,and install all these packages:

- TensorFlow
- Keras
- Seaborn
- pydot
- scikit-learn
- pandas
- Matplotlib

For this project purpose we had use **Anaconda** ,and setup a virtual environment to use a **jupiter notebook.**
Learn how to how to install the prerequiste using the link below:

https://towardsdatascience.com/virtual-environments-in-anaconda-jupyter

# 2. The data

First, we use this data set from Kaggle which tracks diabetes in Pima Native Americans. We use it to build a predictive model of how likely someone is to get or have diabetes given their age, body mass index, glucose and insulin levels, skin thickness, etc.

## 2.1 initial observation

**-Import the dataframe with pandas**

We have several categories along with the outcome which is a binary classification.

**Importing the data set**

```
Entrée [249]: np.set_printoptions(precision=2)
Entrée [136]: df = pd.read_csv("C:/Users/user/Python/Jupiter_notebooks/diabetes.csv")
Entrée [137]: df = df.dropna(how="any", axis=0)
Entrée [138]: df.head()
```

Out[138]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | 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 |

Figure 1: An exampl image

**-browse the data, listing maximum and minimum and average values**

```
Entrée [274]: df.describe()
```

Out[274]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| count | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| mean | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.240885 | 0.348958 |
| std | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.760232 | 0.476951 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.000000 | 0.000000 |
| 25% | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.000000 | 0.000000 |
| 50% | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| 75% | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| max | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

Figure 2: An example

The data also needs to be scaled in order to allow the neural network to work effectively. we will talk about the Data cleaning and preprocessing in the following part.

**-Check correlation with heatmap graph**

That is not important for the final model but is useful to gain further insight into the data. This is the code we used for correlation map:

```
import seaborn as sns
corr = df.corr()
sns.heatmap(corr, xticklabels=corr.columns.values,
yticklabels=corr.columns.values)
```

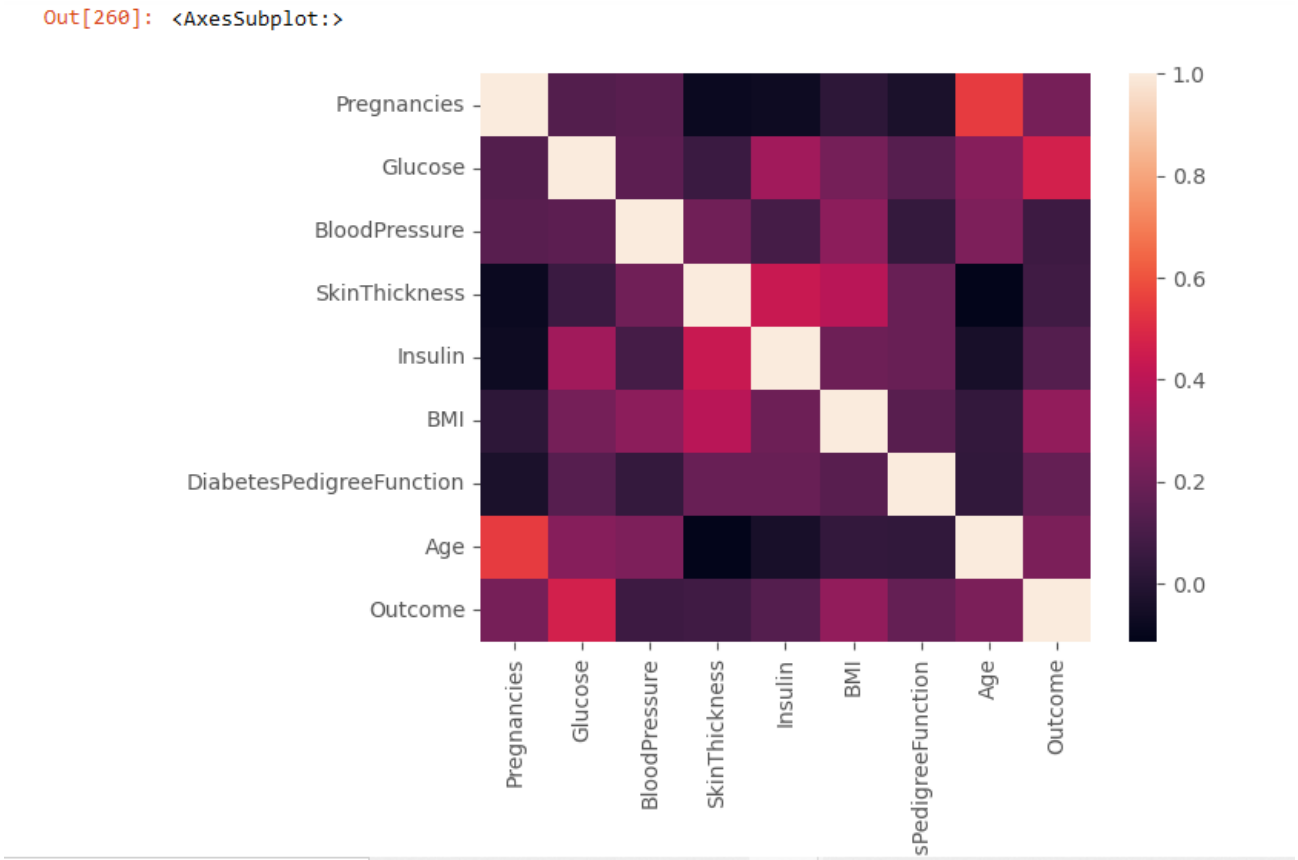


Figure 3: correlation

**There's not a lot of orange squares in the chart. So, you can say that no single value is 80%. likely to give you diabetes (outcome). There does not seem to be much correlation between these individual variables. But, we will see that when taken in the aggregate we can predict with almost 75%. accuracy who will develop diabetes given all of these factors together.**

## 2.2 Data preparation

Splitting the dataset into training set and test set

```
labels=df['Outcome']
features = df.iloc[:,1:8]
from sklearn.model_selection import train_test_split
```

```
X=features
y=np.ravel(labels)
X_train, X_test, y_train, y_test = train_test_split(X,
   y, test_size=0.33, random_state=42)
```

- **Outcome** is the column with the label (0 or 1).
- The rest of the columns exceptthe first one are the features.
- We use the scikit-learn function train-test-split(X, y, testsize=0.33, random-state=42) to split the data into training and test data sets, given 33% of the records to the test data set. The training data set is used to train the mode, meaning find the weights and biases. The test data set is used to check its accuracy.
- **Label** is not an array. It is a column in a dataset. So we use the NumPy np.ravel() function to convert that to an array.

Transform data in the same scale
StandardScaler does this in two steps:**fit()** and **transform()** .

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

# 3. Building the Neural Network

This neural network will be built with Keras by using the Sequential class.

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(7, activation='relu',input_shape=(7,)))
model.add(Dense(7, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

## 3.1 Actiavtion function

Pick an activation function for each layer

For the first two layers we use a relu (rectified linear unit) activation function. That choice means nothing, as you could have picked sigmoid. reluI is 1 for all positive values and 0 for all negative ones

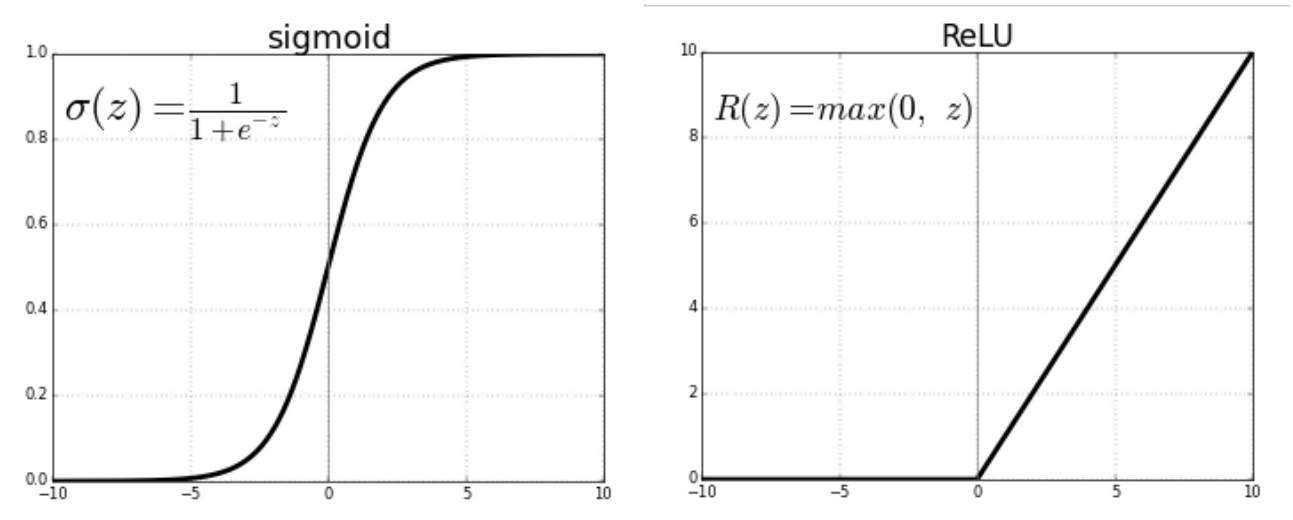


Figure 4: sigmoid and ReLu

## input-shape

we only have to give it the shape (dimensions) of the input on the first layer. It's (7,) since it's a vector of 7 features

## Dense

The first argument in the Dense function is the number of hidden units, a parameter that you can adjust to improve the accuracy of the model

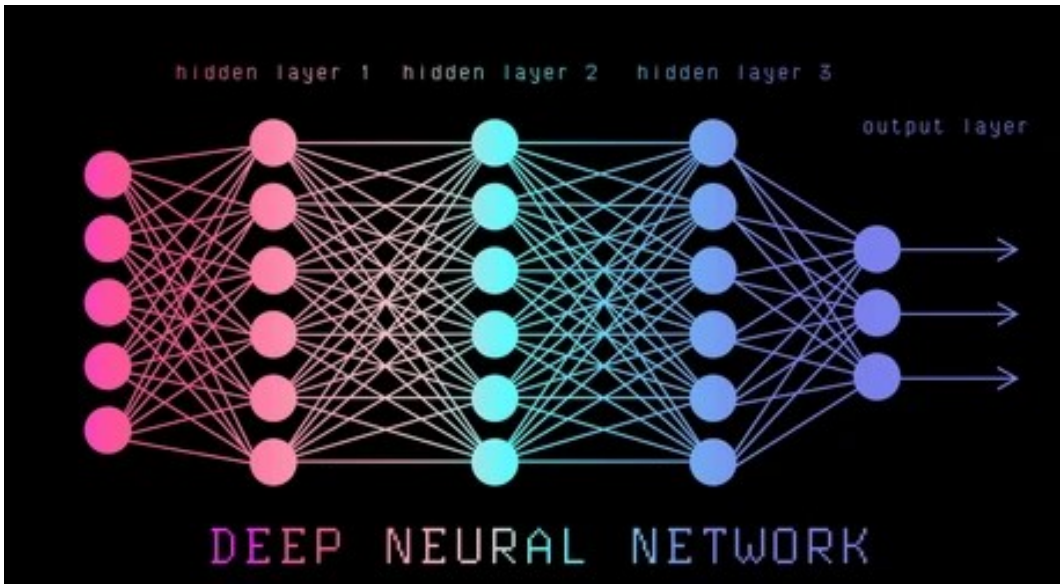


Figure 5: Neural Network

## 3.2 Keras model fit

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
H=model.fit(X_train, y_train,validation_split=0.33,epochs=200, batch_size=1, verb
print(H.history.keys())
```

Figure 6: train code

- **loss:** he goal of the neural network is to minimize the loss function, There are many functions we can use. We pick binary-crossentropy because our label data is binary (1) diabetic and (0) not diabetic.
- **optimizer-adam:** use the optimizer function adam, Adaptative moment estimation. It's an algorithm designed to minimize the loss function in the quickest way possible.
- **Epoch:** means how many times to run the model. it is an iterative process. You could add additional epochs, but the accuracy might not change much.
- **batch size:** means divide the input data into n batches and process each in parallel.
- **fit():** trains the model, meaning calculates the weights, biases, number of layers, etc.

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(7, activation='relu',input_shape
   =(7,)))
model.add(Dense(7, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Above, we talked about the iterative process of solving a neural network for weights and bias. That's done with epochs. Here is the output as it runs those. As you can see the accuracy goes up quickly then levels off.

```
        Epoch 1/30
344/344 [==============================] - 6s 16ms/step -
   loss: 0.5703 - accuracy: 0.7064 - val_loss: 0.5499 -
   val_accuracy: 0.7000
Epoch 2/30
344/344 [==============================] - 2s 5ms/step -
   loss: 0.4947 - accuracy: 0.7703 - val_loss: 0.5187 -
   val_accuracy: 0.7353
Epoch 3/30
344/344 [==============================] - 2s 5ms/step -
   loss: 0.4654 - accuracy: 0.7820 - val_loss: 0.5022 -
   val_accuracy: 0.7471
```

# 4. Results

## 4.1 Precision

### Precision

So, our predictive model is **57%** accurate while training and **85%** while Testing.

```
17/17 [==============================] - 0s 2ms/step - loss: 0.4206 - accuracy: 0.797
Training Accuracy: 79.77%

8/8 [==============================] - 0s 3ms/step - loss: 0.5373 - accuracy: 0.7480
Testing Accuracy: 74.80%
```

Figure 7: Neural Network

## 4.2 Confusion Matrix

### Confusion Matrix

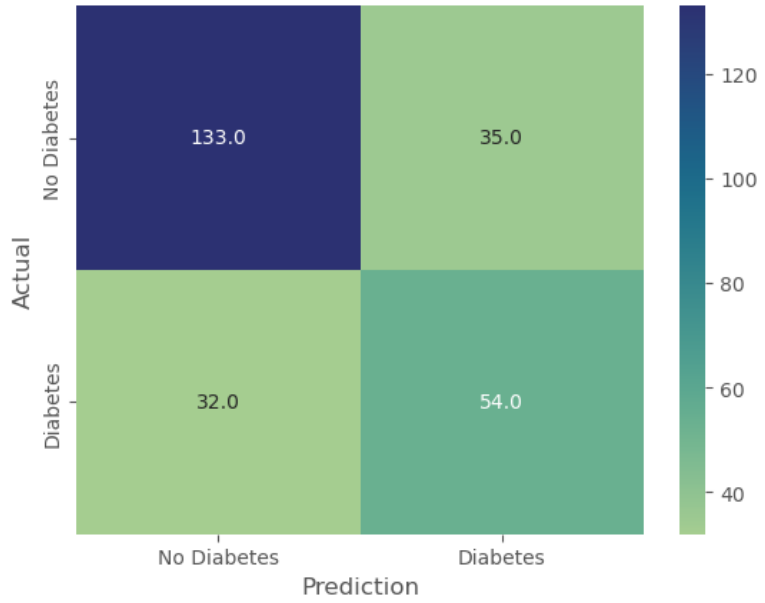


Figure 8: Confusion Matrix

- Total in dataframe(Diabetes)=86
  - **wrong prediction=33**
- Total in dataframe(No Diabetes)=168
  - **wrong prediction=40**

A carrefull approach can permit us to say the number of (1) outcomes are thery low so the model can't predict accuratly this category of persons

### 4.3 classification Report

classification Report

Here you can see the precision is 61 for Diabetic(1) and 81 for non Diabetics(0) determiation

**Show a global classification report**

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.79      0.80       168
           1       0.61      0.63      0.62        86

    accuracy                           0.74       254
   macro avg       0.71      0.71      0.71       254
weighted avg       0.74      0.74      0.74       254
```

Figure 9: Classifiaction Report

# 5. Discussion

Visualize Model Training History in Keras. You can create plots from the collected history data. he example collects the history returned from training the model and creates a chart:

**In the same graph**

- The accuracy on the training and validation datasets over training epochs
- The loss on the training and validation datasets over training epochs

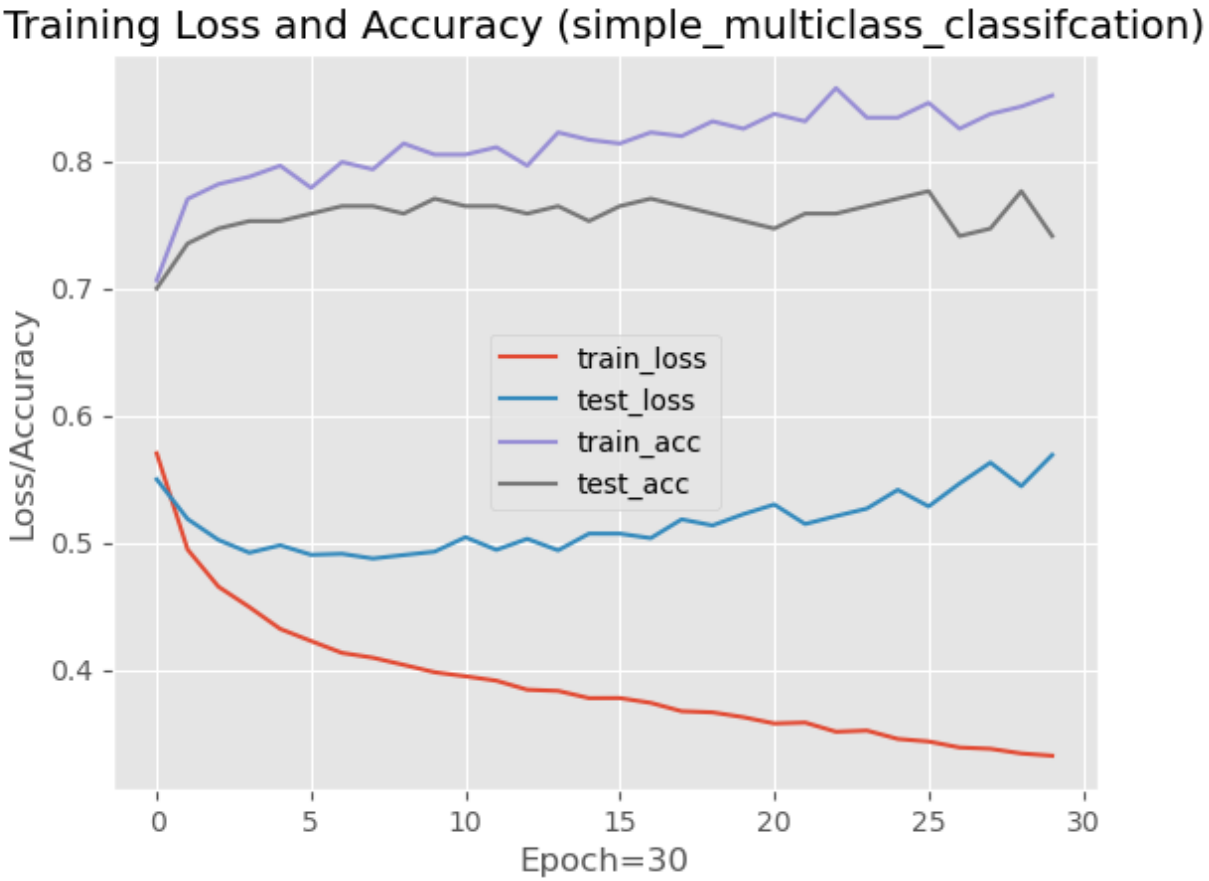


Figure 10: Confusion Matrix

-From the plot of the accuracy, you can see that the model could probably be trained a little more as the trend for accuracy on both datasets is still rising a little bit for the last few epochs

-From the plot of the loss, you can see that the model has comparable performance on both train and validation datasets (labeled test). If these parallel plots start to depart consistently, it might be a sign to stop training at an earlier epoch.

## 5.1 Limitations and Challenges

The test set accuracy was 79.87% which is not too bad for a simple neural network, but it could certainly be improved. There were only 9 false positives but 22 false negatives. There are many improvements that need to be made to this model to lower false diagnoses. This dataset only has eight categories which might not be enough to truly make accurate predictions. Feature engineering is more likely to be useful than increasing the complexity of this neural network.

# 6. Conclusion

It is important to note that neural networks can be more computationally intensive to train and may require more data and more time to achieve good performance, compared to some other classification algorithms. Additionally, they can be more difficult to interpret and understand, as they learn patterns in the data through the weights and biases of the network rather than through explicit rules or decision trees.

**There are several advantages to using neural networks for classification tasks:**

1.They are able to learn complex relationships between the input features and the target class.
2.They are able to handle large amounts of data.
3.They can learn from unstructured data.
4.They are flexible and adaptable.
5.They can be trained to perform well on a wide range of classification tasks.

**Theorem 1** (label=theorem:label)**.** *Title Some text goes here!*