Department of Computer Engineering
Faculty of Engineering, University of Peradeniya
CO542

# Neural Networks and Fuzzy Systems
2023
# Lab 05 - Multi Layer Perceptrons (MLP)

---

## Objectives

- Implement and train multi layer perceptron using python libraries.
- Practice the usage of neural networks in real world applications.
- Familiarize data prepossessing functions and effect of changing hyper-parameters.

## Multi Layer Perceptrons

- A multi layer perceptron (i.e. feedforward neural networks with hidden layers) contains at least two layers of functional units.
- This means that at least one layer contains hidden units, which do not communicate with the environment.
- If the number of hidden units is appropriately chosen, multilayer perceptrons are universal approximators, i.e. they can solve, at least theoretically, any association problem. In other words, they can approximate any function, given enough inputs and target outputs (e.g.: nonlinearly separable classification, nonlinear regression and prediction problems).

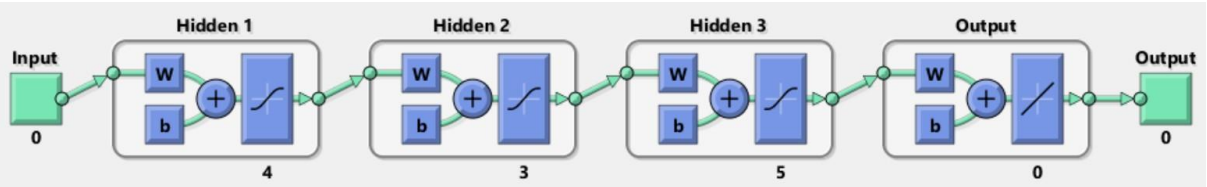


Figure 1: Multi Layer Perceptron

Multilayer Perceptrons are modeled using Scikit-learn in a 6-step process:

1. **Step 1:** Like always first we will import the modules.
   (Note: Here we have loaded the data using sklearn.datasets for explanation. However, you can also load your data using any preferred way, e.g., external urls,.csv files, etc.)

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
```

```
#Importing the iris dataset
from sklearn.datasets import load_iris

#Here are some additional useful imports
import pandas as pd
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

2. **Step 2:** Separate data frames “X” and “y”, the values of the independent and dependent variables.

```
iris_data = load_iris()
X = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
y = iris_data.target
```

3. **Step 3:** Split the dataset into train and test dataset.
   (We have reserved 20% of the dataset for checking the accuracy of the trained model. Independent train and test dataset are further scaled to make sure that the input data is standard normally distributed are centered around zero and have variance in the same order.)

```
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    random_state=1,
                                                    test_size=0.2)
sc_X = StandardScaler()
X_trainscaled=sc_X.fit_transform(X_train)
X_testscaled=sc_X.transform(X_test)
```

4. **Step 4:** Model the MLP.
   (We have used four hidden layers with different neurons in each layer. Considering the input and output layer, we have a total of 6 layers in the model. In case any optimizer is not mentioned then “Adam” is the default optimizer.)

```
model = MLPClassifier(hidden_layer_sizes=(256,128,64,32),
                      activation="relu",
                      random_state=1)
```

5. **Step 5:** Train the model and make predictions.

```
clf = model.fit(X_trainscaled, y_train)
y_pred=clf.predict(X_testscaled)
```

6. **Step 6:** Evaluate the accuracy of the classifier.

```
print(clf.score(X_testscaled, y_test))

fig=plot_confusion_matrix(clf, X_testscaled,y_test,
                          display_labels=["Setosa","Versicolor","Virginica"])
fig.figure_.suptitle("Confusion Matrix for Iris Dataset")
plt.show()
```

For more information refer to:

- sklearn.neural  network.MLPClassifier.
- sklearn.neural network.MLPRegressor.

# Task 1

## Create and train a neural network that solves the XOR problem.

1. What is the number of inputs and outputs of the neurons?
2. Why is this problem NOT linearly separable?
3. What is the input training vector and target vector?
4. Create a network named 'netXOR' with 2 neurons in the input layer, 5 neurons in the hidden layer and 1 output neuron.

***Note:*** *Usually the selection of the,*

- *number of hidden layers*
- *number of neurons in each hidden layer*
- *type of activation functions*
- *the training algorithm*

# Task 2

## Create and train a neural network that predicts Concrete Compressive Strength.

- **In this exercise you will train an MLP to predict the Concrete Compressive Strength**

  *Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients.*

  ***Additional Information***
  - ***Number of instances*** *1030*
  - ***Number of Attributes*** *9*
  - ***Attribute breakdown*** *8 quantitative input variables, and 1 quantitative output variable*

- **Please use the provided data set in FEeLS to complete the task.**

1. Load the dataset as a Pandas dataframe.
2. Separate the dataset into features and labels.
3. Split the data set as train and test data accordingly (E.g.: 90% training data, 10% test data)
4. Scale the independent train and test datasets using Sklearn Standard Scaler.
5. Model the MLP using MLPRegressor (Use hidden layers as (256,128,64,32))
6. Train the model with 300 iterations
7. Find the predicted concrete compressive strength for previously reserved test data sets and find the accuracy of the test set.
8. Do a cross validation for this model. Use the number of folds as 5

Useful Link : https://scikit-learn.org/stable/modules/cross_validation.html

9. Plot the data using *matplotlib.pyplot.boxplot*
10. Repeat training with 400, 500 and 600 and plot the boxplot in same plot

## Submission

You must submit 2 folders (for each exercise) containing,

1. All required .py files (or .ipynb files).
2. A pdf containing all required outputs (images of output curves, network training window, confusion matrix, etc...) and explanations before the deadline (announced in the course page)

Note: Please submit all the files (pdfs, code files etc.) in a single zipped file renamed as lab 4 E XXX.zip where XXX stands for your index no.