Table of Contents

Machine Learning – Energy Systems Course at University of Calgary

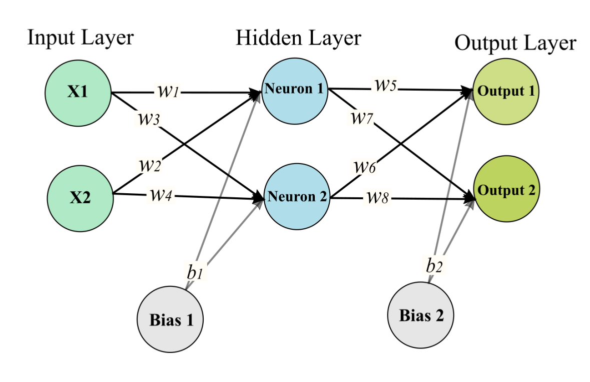
- Instructor: Mehdi Rezvandehy

Artificial Neural Network (ANN)

We are inspired by birds to fly but not exactly doing the same as birds do; planes do not flap their wings. This is the key for Artificial Neural Network (ANN). ANN looks at the human brain's architecture to build an intelligent machine. However, ANN has gradually become quite different from the biological term. ANN is the very core of Deep Learning. Deep Learning usually involves much more successive layers of representations. It is a major field of Machine Learning for tackling very complex problems that Machine Learning is unable to resolve such as classifying billions of images, speech recognition, computer vision. Deep Learning requires big data in order to receive higher performance than Machine Learning and ANN.

How ANN Works

The Figure below shows a simple ANN architecture. It consists of input layers, hidden layer and output layer. Input layer is number of futures in training set. Hidden layer is made up of neurons; here we have two neurons, and final layer is output layer, which is number of target for each training instance. When all the neurons in a layer are connected to every neuron in the previous layer, it is called a **fully connected layer** or **a dense layer**. A bias neuron is also considered for hidden layer and output layer.



The outputs of a layer of artificial neurons for several instances at once is calculated by:

$O_{\mathbf{w,b}}(\mathbf{x})=\varphi(\mathbf{x}\mathbf{w}+\mathbf{b})$

- **X** is the matrix of input features.
- **W** is the connection weights for the neurons except the bias neuron.
- **b** is the all weights for biased neurons.
- φ is a function called activation function.

For the ANN architecture above: $\mathbf{x}=x_{1},x_{2}$, $\mathbf{w}=w_{1},w_{2},w_{3},w_{4},w_{5},w_{6},w_{7},w_{8}$ and $\mathbf{b}=b1_{N1},b1_{N2},b2_{O1},b2_{O2}$.

The weights $\mathbf{w}$ and $\mathbf{b}$ should be optimized by minimizing the calculated error between the calculated output from ANN ($O_{\mathbf{w,b}}(\mathbf{x}$) and actual values. This can be done by **backpropagation** algorithm; it is simply Gradient Descent discussed before. Backpropagation algorithm applies two passes through the network (one forward, one backward) to calculate the gradient of the network's error with regards to every single weight. It measures how much each weight and bias term should be tweaked in order to mitigate the error from the actual values. After computing the gradients, simple Gradient Descent is applied by iterating over the entire network until the network converges to the solution.

Lets discuss this approach in more details as below:

1- One mini-batch (mini-batch discussed in previous lecture) is applied at the time. For example, random selection of 32 training instances can be applied each time. As we discussed before, each iteration is called **epoch**.

2- Each mini-batch is passed to the first hidden layer. The output of all the neurons in the first layer is calculated and then pass on to the next layer; the output of all the neurons in the new layer are computed and passed again to the next layer. This process is repeated until reaching to the output layer. This is the **forward pass**. All the weights for the unknown weights and bias terms are randomly generated in this step.

3- Next step is to calculate the ANN output error by using a cost function such as mean square error.

4- Then, the contribution of each parameter (weight) to the error is calculated. This step can be fast and precise by applying *chain rule*. It starts from the last hidden layer; the *chain rule* is repeated again until reaching to the input layer. By going through this reverse pass, the error gradient across all the connection weights in the network is efficiently measured: the error gradient is propagated backward through the network. That is why it is called backpropagation.

5- Finally, the algorithm goes through multiple Gradient Descent step to optimize all weights and biases in ANN using the error gradients.

In order to have higher performance for very complex problem and achieve non-linear relations; an activation function φ should be used. Since $\mathbf{x}\mathbf{w}+\mathbf{b}$ is a linear transformation, we need some non-linearity between layers to solve complex problems, otherwise even using very deep network will be inefficient.

The common activation functions are:

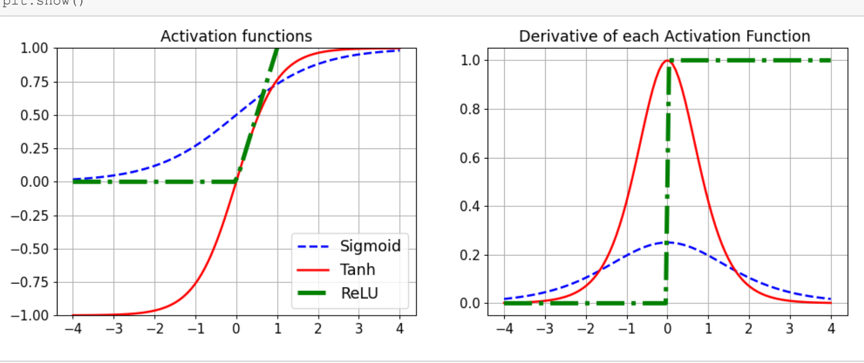
1- *Rectified Linear Unit function: ReLU(z) = max(0, z)*. Although its derivative is 0 for z<0, *ReLU* is the fastest and most popular one activation function.

2- *Logistic Regression*: $\frac{1}{1+e^{-z}}$

3-*Tangent function*: tanh(z)

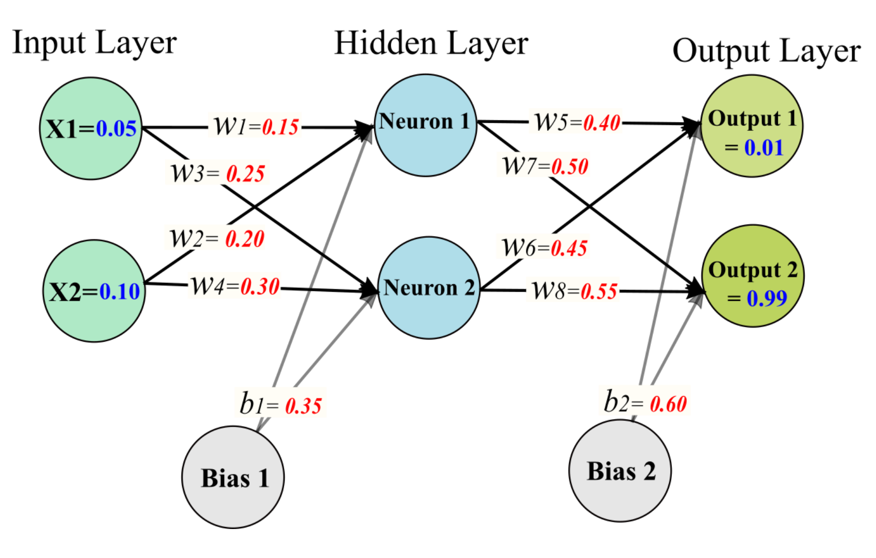
See the following plots for these activation function with the derivate of each function:

In [1]:
```
from matplotlib.colors import ListedColormap
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from numpy import random

font = {'size'   : 12}
plt.rc('font', **font)
fig = plt.subplots(figsize=(13.0, 4.5), dpi= 90, facecolor='w', edgecolor='k')

z1 = np.linspace(-4, 4, 100)
#
def sgd(y):
    return 1 / (1 + np.exp(-y))

def relu(y):
    return np.maximum(0, y)

def drivtve(f, y):
    return (f(z1 + 0.000001) - f(z1 - 0.000001))/(2 * 0.000001)
#
plt.subplot(1,2,1)

plt.plot(z1, sgd(z1), "b--", linewidth=2, label="Sigmoid")
plt.plot(z1, np.tanh(z1), "r-", linewidth=2, label="Tanh")
plt.plot(z1, relu(z1), "g-.", linewidth=4, label="ReLU")
plt.grid(True)
plt.legend(loc=4, fontsize=14)
plt.title("Activation functions", fontsize=14)
plt.ylim(-1,1)

plt.subplot(1,2,2)

plt.plot(z1, drivtve(sgd, z1), "b--", linewidth=2, label="Sigmoid")
plt.plot(z1, drivtve(np.tanh, z1), "r-", linewidth=2, label="Tanh")
plt.plot(z1, drivtve(relu, z1), "g-.", linewidth=4, label="ReLU")
plt.grid(True)
plt.title("Derivative of each Activation Function ", fontsize=14)

plt.show()
```



In [ ]:

Lets apply ANN for a simple example below. Assume for one training instance, we have the value X1=0.05, X2= 0.10 for features X1 and X2, respectively. And we have two outputs 0.01 and 0.99. The activation function is logistic regression. First, the weights $w_{1},w_{2},w_{3},w_{4},w_{5},w_{6},w_{7},w_{8}$ and $b1_{N1},b1_{N2},b2_{O1},b2_{O2}$ are randomly generate between 0 to 1 as below:



The Forward Pass

Neuron 1:

$z_{N1}=x_{1}*w_{1}+x_{2}*w_{2}+b1_{N1}*1= 0.05*0.15+0.1*0.2+0.35*1 =0.3775$.

$ out_{N1}=\frac{1}{1+e^{-z_{N1}}}=0.593269992$

Neuron 2:

$z_{N2}=x_{1}*w_{3}+x_{2}*w_{4}+b1_{N2}*1= 0.05*0.25+0.1*0.3+0.35*1 =0.39249$.

$ out_{N2}=\frac{1}{1+e^{-z_{N2}}}=0.596884378$

Output 1:

$z_{O1}=out_{N1}*w_{5}+out_{N2}*w_{6}+b2_{O1}*1= 0.593269992*0.4+0.596884378*0.45+0.60*1 =1.1059059667$.

$ out_{O1}=\frac{1}{1+e^{-z_{O1}}}=0.75136507$

Output 2:

$z_{O2}=out_{N1}*w_{7}+out_{N2}*w_{8}+b2_{O2}*1= 0.593269992*0.5+0.596884378*0.55+0.60*1 =1.2249214039$.

$ out_{O2}=\frac{1}{1+e^{-z_{O2}}}=0.772928465$

We calculated Output 1 and Output 2, so the mean square error can be calculated as $MSE=\frac{1}{2}\sum_{i=1}^{2} (Target_{i}-Predicted_{i})^{2}=\frac{1}{2}((0.01-0.75136507)^{2}+(0.99-0.772928465)^{2})=0.298371109$.

The Backwards Pass

The main aim of backpropagation is to update each the weight in the ANN to make the calculated outputs ($ out_{O1}$,$ out_{O2}$) close to the target values (0.01 and 0.99) by minimizing the error (MSE).

Lets do it for w_{5}. We want to know how much a very little change in w_{5} impacts MSE. So, we should calculate the partial derivative of MSE with respect to w_{5}. It is also called the gradient with respect to w_{5}: $\frac{\partial MSE}{\partial w_{5}}$. This partial derivative can be calculated by using chain rule:

$\Large \frac{\partial MSE}{\partial w_{5}}= \frac{\partial MSE}{\partial out_{O1}}\times \frac{\partial out_{O1}}{\partial z_{O1}}\times \frac{\partial z_{O1}}{\partial w_{5}}$

Each partial derivative can be calculated as:

$\frac{\partial MSE}{\partial out_{O1}} \rightarrow MSE=\frac{1}{2} \sum_{i=1}^{2} (Target_{out_{Oi}}-out_{Oi})^{2} \,\,\,\,\,\,\,\frac{\partial MSE}{\partial out_{O1}}=2\times \frac{1}{2}(Target_{out_{O1} }-out_{O1})^{2-1}\times -1+0 \\=-(0.01-0.75136507)=0.74136507$

$\frac{\partial out_{O1}}{\partial z_{O1}} \rightarrow out_{O1}=\frac{1}{1+e^{-z_{O1}}} \rightarrow \frac{\partial out_{O1}}{\partial z_{O1}}= \frac{e^{-z_{O1}}}{(1+e^{-z_{O1}})^{2}} =0.1868156018$

$\frac{\partial z_{O1}}{\partial w_{5}} \rightarrow z_{O1}=out_{N1}*w_{5}+out_{N2}*w_{6}+b_{1} \rightarrow \frac{\partial z_{O1}}{\partial w_{5}}=0.593269992$

We have calculated all partial derivatives. Now we can calculate the gradient of MSE with respect to w_{5}: $\frac{\partial MSE}{\partial w_{5}}$:

$\Large \frac{\partial MSE}{\partial w_{5}}= \frac{\partial MSE}{\partial out_{O1}}\times \frac{\partial out_{O1}}{\partial z_{O1}}\times \frac{\partial z_{O1}}{\partial w_{5}}$$=0.74136507\times 0.1868156018 \times 0.593269992=0.0821670401$

$\Large \frac{\partial MSE}{\partial w_{5}}$$=0.0821670401$

This partial derivatives of MSE with respect to other weights $w_{1},w_{2},w_{3},w_{4},w_{6},w_{7},w_{8}$ and bias $b1_{N1},b1_{N2},b1_{O1},b1_{O2}$ can be calculated by using chain rule as we did.

The next step is to decrease the MSE by tuning w_{5} using Gradient Descent:

$\large w_{5}^{u}=w_{5}-\alpha \frac{\partial MSE}{\partial w_{5}}$

If learning rate α=0.5, then the updated weight $ w_{5}^{u}$ to decrease MSE is calculated as:

$\large w_{5}^{u}$$=0.4-0.5* 0.0821670401=0.35891648$.

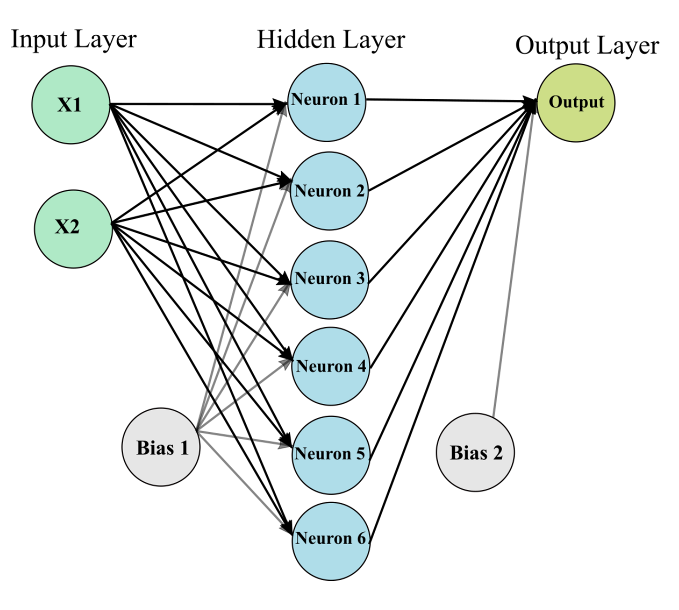
This process should be repeated for many iteration to optimize w_{5}. See lecture before for more information about Gradient Descent. The entire process should be applied to uptimize other weights $w_{1},w_{2},w_{3},w_{4},w_{6},w_{7},w_{8}, b1_{N1},b1_{N2},b1_{O1},b1_{O2}$.

Small Examples

We start to build a ANN using tensorflow library.

In [2]:
```
import tensorflow as tf
from tensorflow import keras
```

We want to build a neural network structure for a binary classification and regression. The network structure has 1 hidden layers with 6 neurons (see Figure below).



Classification (binary)

Here is a synthetic data set with two features and only 4 training instances.

In [3]:
```
X=[[1.5,2.2],
   [-1.1,1.6],
   [-2.7,-0.5],
   [3.1,2.5]]
y=[1,
   0,
   0,
   1]
```

In [ ]:

In [4]:
```
np.random.seed(42)
tf.random.set_seed(42)

keras.backend.clear_session() # Clear the previous model

# create model
model = keras.models.Sequential()

# Input & Hidden Layer 1
model.add(keras.layers.Dense(6, input_dim=np.array(X).shape[1], activation='sigmoid'))

# Output Layer
model.add(keras.layers.Dense(1,activation='sigmoid'))
# use activation="softmax" for multicalss classification

# Compile model
model.compile(optimizer='sgd',loss="binary_crossentropy",metrics=['accuracy'])
# sgd is stochastic gradient descent
# loss="binary_crossentropy" is for binary classification
# use loss="sparse_categorical_crossentropy" for multiclass classification

history=model.fit(X,y,verbose=1,epochs=200)
```

```
Epoch 1/200
1/1 [==============================] - 0s 1ms/step - loss: 0.8143 - accuracy: 0.5000
Epoch 2/200
1/1 [==============================] - 0s 998us/step - loss: 0.8119 - accuracy: 0.5000
Epoch 3/200
1/1 [==============================] - 0s 999us/step - loss: 0.8094 - accuracy: 0.5000
Epoch 4/200
1/1 [==============================] - 0s 998us/step - loss: 0.8070 - accuracy: 0.5000
Epoch 5/200
1/1 [==============================] - 0s 997us/step - loss: 0.8046 - accuracy: 0.5000
Epoch 6/200
1/1 [==============================] - 0s 996us/step - loss: 0.8022 - accuracy: 0.5000
Epoch 7/200
1/1 [==============================] - 0s 996us/step - loss: 0.7999 - accuracy: 0.5000
Epoch 8/200
1/1 [==============================] - 0s 997us/step - loss: 0.7975 - accuracy: 0.5000
Epoch 9/200
1/1 [==============================] - 0s 997us/step - loss: 0.7952 - accuracy: 0.5000
Epoch 10/200
1/1 [==============================] - 0s 996us/step - loss: 0.7929 - accuracy: 0.5000
Epoch 11/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7906 - accuracy: 0.5000
Epoch 12/200
1/1 [==============================] - 0s 997us/step - loss: 0.7883 - accuracy: 0.5000
Epoch 13/200
1/1 [==============================] - 0s 998us/step - loss: 0.7860 - accuracy: 0.5000
Epoch 14/200
1/1 [==============================] - 0s 996us/step - loss: 0.7838 - accuracy: 0.5000
Epoch 15/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7816 - accuracy: 0.5000
Epoch 16/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7793 - accuracy: 0.5000
Epoch 17/200
1/1 [==============================] - 0s 996us/step - loss: 0.7772 - accuracy: 0.5000
Epoch 18/200
1/1 [==============================] - 0s 998us/step - loss: 0.7750 - accuracy: 0.5000
Epoch 19/200
1/1 [==============================] - 0s 995us/step - loss: 0.7728 - accuracy: 0.5000
Epoch 20/200
1/1 [==============================] - 0s 998us/step - loss: 0.7707 - accuracy: 0.5000
Epoch 21/200
1/1 [==============================] - 0s 995us/step - loss: 0.7685 - accuracy: 0.5000
Epoch 22/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7664 - accuracy: 0.5000
Epoch 23/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7643 - accuracy: 0.5000
Epoch 24/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7622 - accuracy: 0.5000
Epoch 25/200
1/1 [==============================] - 0s 995us/step - loss: 0.7602 - accuracy: 0.5000
Epoch 26/200
1/1 [==============================] - 0s 998us/step - loss: 0.7581 - accuracy: 0.5000
Epoch 27/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7561 - accuracy: 0.5000
Epoch 28/200
1/1 [==============================] - 0s 997us/step - loss: 0.7540 - accuracy: 0.5000
Epoch 29/200
1/1 [==============================] - 0s 997us/step - loss: 0.7520 - accuracy: 0.5000
Epoch 30/200
1/1 [==============================] - 0s 998us/step - loss: 0.7500 - accuracy: 0.5000
Epoch 31/200
1/1 [==============================] - 0s 997us/step - loss: 0.7481 - accuracy: 0.5000
Epoch 32/200
1/1 [==============================] - 0s 998us/step - loss: 0.7461 - accuracy: 0.5000
Epoch 33/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7441 - accuracy: 0.5000
Epoch 34/200
1/1 [==============================] - 0s 988us/step - loss: 0.7422 - accuracy: 0.5000
Epoch 35/200
1/1 [==============================] - 0s 998us/step - loss: 0.7403 - accuracy: 0.5000
Epoch 36/200
1/1 [==============================] - 0s 998us/step - loss: 0.7384 - accuracy: 0.5000
Epoch 37/200
1/1 [==============================] - 0s 994us/step - loss: 0.7365 - accuracy: 0.5000
Epoch 38/200
1/1 [==============================] - 0s 997us/step - loss: 0.7346 - accuracy: 0.5000
Epoch 39/200
1/1 [==============================] - 0s 0s/step - loss: 0.7327 - accuracy: 0.5000
Epoch 40/200
1/1 [==============================] - 0s 3ms/step - loss: 0.7309 - accuracy: 0.5000
Epoch 41/200
1/1 [==============================] - 0s 996us/step - loss: 0.7290 - accuracy: 0.5000
Epoch 42/200
1/1 [==============================] - 0s 2ms/step - loss: 0.7272 - accuracy: 0.5000
Epoch 43/200
1/1 [==============================] - 0s 999us/step - loss: 0.7254 - accuracy: 0.5000
Epoch 44/200
1/1 [==============================] - 0s 996us/step - loss: 0.7235 - accuracy: 0.5000
Epoch 45/200
1/1 [==============================] - 0s 1ms/step - loss: 0.7218 - accuracy: 0.5000
Epoch 46/200
1/1 [==============================] - 0s 997us/step - loss: 0.7200 - accuracy: 0.5000
Epoch 47/200
1/1 [==============================] - 0s 983us/step - loss: 0.7182 - accuracy: 0.5000
Epoch 48/200
1/1 [==============================] - 0s 998us/step - loss: 0.7164 - accuracy: 0.5000
Epoch 49/200
1/1 [==============================] - 0s 998us/step - loss: 0.7147 - accuracy: 0.5000
Epoch 50/200
1/1 [==============================] - 0s 996us/step - loss: 0.7130 - accuracy: 0.5000
Epoch 51/200
1/1 [==============================] - 0s 997us/step - loss: 0.7112 - accuracy: 0.5000
Epoch 52/200
1/1 [==============================] - 0s 1000us/step - loss: 0.7095 - accuracy: 0.5000
Epoch 53/200
1/1 [==============================] - 0s 998us/step - loss: 0.7078 - accuracy: 0.5000
Epoch 54/200
1/1 [==============================] - 0s 997us/step - loss: 0.7061 - accuracy: 0.5000
Epoch 55/200
1/1 [==============================] - 0s 997us/step - loss: 0.7045 - accuracy: 0.5000
Epoch 56/200
1/1 [==============================] - 0s 999us/step - loss: 0.7028 - accuracy: 0.5000
Epoch 57/200
1/1 [==============================] - 0s 993us/step - loss: 0.7011 - accuracy: 0.5000
Epoch 58/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6995 - accuracy: 0.5000
Epoch 59/200
1/1 [==============================] - 0s 997us/step - loss: 0.6979 - accuracy: 0.5000
Epoch 60/200
1/1 [==============================] - 0s 998us/step - loss: 0.6962 - accuracy: 0.5000
Epoch 61/200
1/1 [==============================] - 0s 993us/step - loss: 0.6946 - accuracy: 0.5000
Epoch 62/200
1/1 [==============================] - 0s 995us/step - loss: 0.6930 - accuracy: 0.5000
Epoch 63/200
1/1 [==============================] - 0s 0s/step - loss: 0.6914 - accuracy: 0.5000
Epoch 64/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6899 - accuracy: 0.5000
Epoch 65/200
1/1 [==============================] - 0s 959us/step - loss: 0.6883 - accuracy: 0.5000
Epoch 66/200
1/1 [==============================] - 0s 997us/step - loss: 0.6867 - accuracy: 0.5000
Epoch 67/200
1/1 [==============================] - 0s 999us/step - loss: 0.6852 - accuracy: 0.5000
Epoch 68/200
1/1 [==============================] - 0s 998us/step - loss: 0.6836 - accuracy: 0.5000
Epoch 69/200
1/1 [==============================] - 0s 964us/step - loss: 0.6821 - accuracy: 0.5000
Epoch 70/200
1/1 [==============================] - 0s 996us/step - loss: 0.6806 - accuracy: 0.5000
Epoch 71/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6791 - accuracy: 0.5000
Epoch 72/200
1/1 [==============================] - 0s 998us/step - loss: 0.6776 - accuracy: 0.5000
Epoch 73/200
1/1 [==============================] - 0s 997us/step - loss: 0.6761 - accuracy: 0.5000
Epoch 74/200
1/1 [==============================] - 0s 995us/step - loss: 0.6746 - accuracy: 0.5000
Epoch 75/200
1/1 [==============================] - 0s 996us/step - loss: 0.6731 - accuracy: 0.5000
Epoch 76/200
1/1 [==============================] - 0s 996us/step - loss: 0.6717 - accuracy: 0.5000
Epoch 77/200
1/1 [==============================] - 0s 996us/step - loss: 0.6702 - accuracy: 0.5000
Epoch 78/200
1/1 [==============================] - 0s 0s/step - loss: 0.6687 - accuracy: 0.5000
Epoch 79/200
1/1 [==============================] - 0s 995us/step - loss: 0.6673 - accuracy: 0.5000
Epoch 80/200
1/1 [==============================] - 0s 996us/step - loss: 0.6659 - accuracy: 0.5000
Epoch 81/200
1/1 [==============================] - 0s 998us/step - loss: 0.6645 - accuracy: 0.5000
Epoch 82/200
1/1 [==============================] - 0s 997us/step - loss: 0.6630 - accuracy: 0.5000
Epoch 83/200
1/1 [==============================] - 0s 993us/step - loss: 0.6616 - accuracy: 0.5000
Epoch 84/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6602 - accuracy: 0.5000
Epoch 85/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6589 - accuracy: 0.5000
Epoch 86/200
1/1 [==============================] - 0s 999us/step - loss: 0.6575 - accuracy: 0.5000
Epoch 87/200
1/1 [==============================] - 0s 998us/step - loss: 0.6561 - accuracy: 0.5000
Epoch 88/200
1/1 [==============================] - 0s 996us/step - loss: 0.6547 - accuracy: 0.5000
Epoch 89/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6534 - accuracy: 0.5000
Epoch 90/200
1/1 [==============================] - 0s 1000us/step - loss: 0.6520 - accuracy: 0.5000
Epoch 91/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6507 - accuracy: 0.5000
Epoch 92/200
1/1 [==============================] - 0s 995us/step - loss: 0.6494 - accuracy: 0.5000
Epoch 93/200
1/1 [==============================] - 0s 997us/step - loss: 0.6480 - accuracy: 0.5000
Epoch 94/200
1/1 [==============================] - 0s 996us/step - loss: 0.6467 - accuracy: 0.5000
Epoch 95/200
1/1 [==============================] - 0s 997us/step - loss: 0.6454 - accuracy: 0.5000
Epoch 96/200
1/1 [==============================] - 0s 996us/step - loss: 0.6441 - accuracy: 0.5000
Epoch 97/200
1/1 [==============================] - 0s 995us/step - loss: 0.6428 - accuracy: 0.5000
Epoch 98/200
1/1 [==============================] - 0s 995us/step - loss: 0.6415 - accuracy: 0.5000
Epoch 99/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6402 - accuracy: 0.5000
Epoch 100/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6389 - accuracy: 0.5000
Epoch 101/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6377 - accuracy: 0.5000
Epoch 102/200
1/1 [==============================] - 0s 995us/step - loss: 0.6364 - accuracy: 0.5000
Epoch 103/200
1/1 [==============================] - 0s 993us/step - loss: 0.6352 - accuracy: 0.5000
Epoch 104/200
1/1 [==============================] - 0s 995us/step - loss: 0.6339 - accuracy: 0.5000
Epoch 105/200
1/1 [==============================] - 0s 998us/step - loss: 0.6327 - accuracy: 0.5000
Epoch 106/200
1/1 [==============================] - 0s 0s/step - loss: 0.6314 - accuracy: 0.5000
Epoch 107/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6302 - accuracy: 0.5000
Epoch 108/200
1/1 [==============================] - 0s 999us/step - loss: 0.6290 - accuracy: 0.5000
Epoch 109/200
1/1 [==============================] - 0s 996us/step - loss: 0.6278 - accuracy: 0.5000
Epoch 110/200
1/1 [==============================] - 0s 997us/step - loss: 0.6266 - accuracy: 0.5000
Epoch 111/200
1/1 [==============================] - 0s 994us/step - loss: 0.6254 - accuracy: 0.5000
Epoch 112/200
1/1 [==============================] - 0s 998us/step - loss: 0.6242 - accuracy: 0.5000
Epoch 113/200
1/1 [==============================] - 0s 998us/step - loss: 0.6230 - accuracy: 0.5000
Epoch 114/200
1/1 [==============================] - 0s 992us/step - loss: 0.6218 - accuracy: 0.5000
Epoch 115/200
1/1 [==============================] - 0s 1ms/step - loss: 0.6206 - accuracy: 0.5000
Epoch 116/200
1/1 [==============================] - 0s 0s/step - loss: 0.6194 - accuracy: 0.5000
Epoch 117/200
1/1 [==============================] - 0s 997us/step - loss: 0.6183 - accuracy: 0.5000
Epoch 118/200
1/1 [==============================] - 0s 999us/step - loss: 0.6171 - accuracy: 0.5000
Epoch 119/200
1/1 [==============================] - 0s 2ms/step - loss: 0.6160 - accuracy: 0.5000
Epoch 120/200
1/1 [==============================] - 0s 995us/step - loss: 0.6148 - accuracy: 0.5000
Epoch 121/200
1/1 [==============================] - 0s 998us/step - loss: 0.6137 - accuracy: 0.5000
Epoch 122/200
1/1 [==============================] - 0s 988us/step - loss: 0.6125 - accuracy: 0.5000
Epoch 123/200
1/1 [==============================] - 0s 997us/step - loss: 0.6114 - accuracy: 0.5000
Epoch 124/200
1/1 [==============================] - 0s 999us/step - loss: 0.6103 - accuracy: 0.5000
Epoch 125/200
1/1 [==============================] - 0s 997us/step - loss: 0.6091 - accuracy: 0.5000
Epoch 126/200
1/1 [==============================] - 0s 0s/step - loss: 0.6080 - accuracy: 0.5000
Epoch 127/200
1/1 [==============================] - 0s 997us/step - loss: 0.6069 - accuracy: 0.5000
Epoch 128/200
1/1 [==============================] - 0s 0s/step - loss: 0.6058 - accuracy: 0.5000
Epoch 129/200
1/1 [==============================] - 0s 997us/step - loss: 0.6047 - accuracy: 0.5000
Epoch 130/200
1/1 [==============================] - 0s 996us/step - loss: 0.6036 - accuracy: 0.5000
Epoch 131/200
1/1 [==============================] - 0s 1000us/step - loss: 0.6025 - accuracy: 0.5000
Epoch 132/200
1/1 [==============================] - 0s 998us/step - loss: 0.6014 - accuracy: 0.5000
Epoch 133/200
1/1 [==============================] - 0s 995us/step - loss: 0.6004 - accuracy: 0.5000
Epoch 134/200
1/1 [==============================] - 0s 0s/step - loss: 0.5993 - accuracy: 0.5000
Epoch 135/200
1/1 [==============================] - 0s 1ms/step - loss: 0.5982 - accuracy: 0.5000
Epoch 136/200
1/1 [==============================] - 0s 997us/step - loss: 0.5971 - accuracy: 0.5000
Epoch 137/200
1/1 [==============================] - 0s 996us/step - loss: 0.5961 - accuracy: 0.7500
Epoch 138/200
1/1 [==============================] - 0s 997us/step - loss: 0.5950 - accuracy: 0.7500
Epoch 139/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5940 - accuracy: 0.7500
Epoch 140/200
1/1 [==============================] - 0s 998us/step - loss: 0.5929 - accuracy: 0.7500
Epoch 141/200
1/1 [==============================] - 0s 997us/step - loss: 0.5919 - accuracy: 0.7500
Epoch 142/200
1/1 [==============================] - 0s 997us/step - loss: 0.5909 - accuracy: 0.7500
Epoch 143/200
1/1 [==============================] - 0s 964us/step - loss: 0.5898 - accuracy: 0.7500
Epoch 144/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5888 - accuracy: 0.7500
Epoch 145/200
1/1 [==============================] - 0s 997us/step - loss: 0.5878 - accuracy: 0.7500
Epoch 146/200
1/1 [==============================] - 0s 996us/step - loss: 0.5868 - accuracy: 0.7500
Epoch 147/200
1/1 [==============================] - 0s 999us/step - loss: 0.5857 - accuracy: 0.7500
Epoch 148/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5847 - accuracy: 0.7500
Epoch 149/200
1/1 [==============================] - 0s 997us/step - loss: 0.5837 - accuracy: 0.7500
Epoch 150/200
1/1 [==============================] - 0s 998us/step - loss: 0.5827 - accuracy: 0.7500
Epoch 151/200
1/1 [==============================] - 0s 996us/step - loss: 0.5817 - accuracy: 0.7500
Epoch 152/200
1/1 [==============================] - 0s 997us/step - loss: 0.5807 - accuracy: 0.7500
Epoch 153/200
1/1 [==============================] - 0s 998us/step - loss: 0.5797 - accuracy: 0.7500
Epoch 154/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5788 - accuracy: 0.7500
Epoch 155/200
1/1 [==============================] - 0s 999us/step - loss: 0.5778 - accuracy: 0.7500
Epoch 156/200
1/1 [==============================] - 0s 998us/step - loss: 0.5768 - accuracy: 0.7500
Epoch 157/200
1/1 [==============================] - 0s 998us/step - loss: 0.5758 - accuracy: 0.7500
Epoch 158/200
1/1 [==============================] - 0s 999us/step - loss: 0.5749 - accuracy: 0.7500
Epoch 159/200
1/1 [==============================] - 0s 0s/step - loss: 0.5739 - accuracy: 0.7500
Epoch 160/200
1/1 [==============================] - 0s 997us/step - loss: 0.5729 - accuracy: 0.7500
Epoch 161/200
1/1 [==============================] - 0s 999us/step - loss: 0.5720 - accuracy: 0.7500
Epoch 162/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5710 - accuracy: 0.7500
Epoch 163/200
1/1 [==============================] - 0s 998us/step - loss: 0.5701 - accuracy: 0.7500
Epoch 164/200
1/1 [==============================] - 0s 998us/step - loss: 0.5691 - accuracy: 0.7500
Epoch 165/200
1/1 [==============================] - 0s 996us/step - loss: 0.5682 - accuracy: 0.7500
Epoch 166/200
1/1 [==============================] - 0s 998us/step - loss: 0.5672 - accuracy: 0.7500
Epoch 167/200
1/1 [==============================] - 0s 997us/step - loss: 0.5663 - accuracy: 0.7500
Epoch 168/200
1/1 [==============================] - 0s 959us/step - loss: 0.5654 - accuracy: 0.7500
Epoch 169/200
1/1 [==============================] - 0s 997us/step - loss: 0.5644 - accuracy: 0.7500
Epoch 170/200
1/1 [==============================] - 0s 1ms/step - loss: 0.5635 - accuracy: 0.7500
Epoch 171/200
1/1 [==============================] - 0s 993us/step - loss: 0.5626 - accuracy: 0.7500
Epoch 172/200
1/1 [==============================] - 0s 996us/step - loss: 0.5617 - accuracy: 0.7500
Epoch 173/200
1/1 [==============================] - 0s 999us/step - loss: 0.5608 - accuracy: 0.7500
Epoch 174/200
1/1 [==============================] - 0s 962us/step - loss: 0.5599 - accuracy: 0.7500
Epoch 175/200
1/1 [==============================] - 0s 998us/step - loss: 0.5589 - accuracy: 0.7500
Epoch 176/200
1/1 [==============================] - 0s 998us/step - loss: 0.5580 - accuracy: 0.7500
Epoch 177/200
1/1 [==============================] - 0s 997us/step - loss: 0.5571 - accuracy: 0.7500
Epoch 178/200
1/1 [==============================] - 0s 997us/step - loss: 0.5562 - accuracy: 0.7500
Epoch 179/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5553 - accuracy: 0.7500
Epoch 180/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5545 - accuracy: 0.7500
Epoch 181/200
1/1 [==============================] - 0s 998us/step - loss: 0.5536 - accuracy: 0.7500
Epoch 182/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5527 - accuracy: 0.7500
Epoch 183/200
1/1 [==============================] - 0s 995us/step - loss: 0.5518 - accuracy: 0.7500
Epoch 184/200
1/1 [==============================] - 0s 958us/step - loss: 0.5509 - accuracy: 0.7500
Epoch 185/200
1/1 [==============================] - 0s 965us/step - loss: 0.5501 - accuracy: 0.7500
Epoch 186/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5492 - accuracy: 0.7500
Epoch 187/200
1/1 [==============================] - 0s 988us/step - loss: 0.5483 - accuracy: 0.7500
Epoch 188/200
1/1 [==============================] - 0s 996us/step - loss: 0.5474 - accuracy: 0.7500
Epoch 189/200
1/1 [==============================] - 0s 996us/step - loss: 0.5466 - accuracy: 0.7500
Epoch 190/200
1/1 [==============================] - 0s 984us/step - loss: 0.5457 - accuracy: 0.7500
Epoch 191/200
1/1 [==============================] - 0s 970us/step - loss: 0.5449 - accuracy: 0.7500
Epoch 192/200
1/1 [==============================] - 0s 997us/step - loss: 0.5440 - accuracy: 0.7500
Epoch 193/200
1/1 [==============================] - 0s 997us/step - loss: 0.5432 - accuracy: 0.7500
Epoch 194/200
1/1 [==============================] - 0s 999us/step - loss: 0.5423 - accuracy: 0.7500
Epoch 195/200
1/1 [==============================] - 0s 970us/step - loss: 0.5415 - accuracy: 0.7500
Epoch 196/200
1/1 [==============================] - 0s 998us/step - loss: 0.5406 - accuracy: 0.7500
Epoch 197/200
1/1 [==============================] - 0s 2ms/step - loss: 0.5398 - accuracy: 0.7500
Epoch 198/200
1/1 [==============================] - 0s 996us/step - loss: 0.5389 - accuracy: 0.7500
Epoch 199/200
1/1 [==============================] - 0s 997us/step - loss: 0.5381 - accuracy: 0.7500
Epoch 200/200
1/1 [==============================] - 0s 961us/step - loss: 0.5373 - accuracy: 0.7500
```

In [5]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6)                 18
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 25
Trainable params: 25
Non-trainable params: 0
_________________________________________________________________
```

In [6]:
```
i=1
weights_input = model.layers[i].get_weights()[0]
print('Weights: ',weights_input)
Bias = model.layers[i].get_weights()[1]
print('Bias:',Bias)
```

```
Weights:  [[ 0.37828332]
 [-0.15197238]
 [ 0.57736015]
 [-0.6945995 ]
 [ 0.02347528]
 [ 0.78492385]]
Bias: [-0.2640285]
```

In [7]:
```
pred=model.predict(X)
pred
```

Out[7]:
```
array([[0.6444869 ],
       [0.53500885],
       [0.42337704],
       [0.6769317 ]], dtype=float32)
```

The predict function gives probability of each class. We can simply convert to integer:

In [8]:
```
pred=[1 if i >= 0.5 else 0 for i in pred]
pred
```

Out[8]:
```
[1, 1, 0, 1]
```

Regression

In [9]:
```
X=[[1.5,2.2],
   [-1.1,1.6],
   [-2.7,-0.5],
   [3.1,2.5]]
y=[1.6,
   0.4,
   0.98,
   1.3]
```

In [10]:
```
np.random.seed(42)
tf.random.set_seed(42)

keras.backend.clear_session() # Clear the previous model

# create model
model = keras.models.Sequential()

# Input & Hidden Layer 1
model.add(keras.layers.Dense(6, input_dim=np.array(X).shape[1], activation='relu'))

# Output Layer
model.add(keras.layers.Dense(1))

# Compile model
model.compile(optimizer='sgd',loss="mse")

history=model.fit(X,y,verbose=1,epochs=300)
```

```
Epoch 1/300
1/1 [==============================] - 0s 2ms/step - loss: 0.2352
Epoch 2/300
1/1 [==============================] - 0s 987us/step - loss: 0.2024
Epoch 3/300
1/1 [==============================] - 0s 2ms/step - loss: 0.1784
Epoch 4/300
1/1 [==============================] - 0s 2ms/step - loss: 0.1604
Epoch 5/300
1/1 [==============================] - 0s 997us/step - loss: 0.1467
Epoch 6/300
1/1 [==============================] - 0s 999us/step - loss: 0.1361
Epoch 7/300
1/1 [==============================] - 0s 996us/step - loss: 0.1278
Epoch 8/300
1/1 [==============================] - 0s 2ms/step - loss: 0.1211
Epoch 9/300
1/1 [==============================] - 0s 998us/step - loss: 0.1157
Epoch 10/300
1/1 [==============================] - 0s 997us/step - loss: 0.1113
Epoch 11/300
1/1 [==============================] - 0s 997us/step - loss: 0.1076
Epoch 12/300
1/1 [==============================] - 0s 0s/step - loss: 0.1045
Epoch 13/300
1/1 [==============================] - 0s 997us/step - loss: 0.1019
Epoch 14/300
1/1 [==============================] - 0s 998us/step - loss: 0.0996
Epoch 15/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0977
Epoch 16/300
1/1 [==============================] - 0s 997us/step - loss: 0.0960
Epoch 17/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0944
Epoch 18/300
1/1 [==============================] - 0s 997us/step - loss: 0.0931
Epoch 19/300
1/1 [==============================] - 0s 996us/step - loss: 0.0919
Epoch 20/300
1/1 [==============================] - 0s 998us/step - loss: 0.0908
Epoch 21/300
1/1 [==============================] - 0s 997us/step - loss: 0.0898
Epoch 22/300
1/1 [==============================] - 0s 998us/step - loss: 0.0889
Epoch 23/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0881
Epoch 24/300
1/1 [==============================] - 0s 994us/step - loss: 0.0873
Epoch 25/300
1/1 [==============================] - 0s 0s/step - loss: 0.0866
Epoch 26/300
1/1 [==============================] - 0s 998us/step - loss: 0.0859
Epoch 27/300
1/1 [==============================] - 0s 984us/step - loss: 0.0853
Epoch 28/300
1/1 [==============================] - 0s 999us/step - loss: 0.0848
Epoch 29/300
1/1 [==============================] - 0s 0s/step - loss: 0.0842
Epoch 30/300
1/1 [==============================] - 0s 997us/step - loss: 0.0837
Epoch 31/300
1/1 [==============================] - 0s 998us/step - loss: 0.0833
Epoch 32/300
1/1 [==============================] - 0s 998us/step - loss: 0.0828
Epoch 33/300
1/1 [==============================] - 0s 958us/step - loss: 0.0824
Epoch 34/300
1/1 [==============================] - 0s 997us/step - loss: 0.0820
Epoch 35/300
1/1 [==============================] - 0s 996us/step - loss: 0.0817
Epoch 36/300
1/1 [==============================] - 0s 997us/step - loss: 0.0813
Epoch 37/300
1/1 [==============================] - 0s 997us/step - loss: 0.0810
Epoch 38/300
1/1 [==============================] - 0s 986us/step - loss: 0.0806
Epoch 39/300
1/1 [==============================] - 0s 996us/step - loss: 0.0803
Epoch 40/300
1/1 [==============================] - 0s 999us/step - loss: 0.0800
Epoch 41/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0798
Epoch 42/300
1/1 [==============================] - 0s 996us/step - loss: 0.0795
Epoch 43/300
1/1 [==============================] - 0s 998us/step - loss: 0.0792
Epoch 44/300
1/1 [==============================] - 0s 999us/step - loss: 0.0790
Epoch 45/300
1/1 [==============================] - 0s 958us/step - loss: 0.0788
Epoch 46/300
1/1 [==============================] - 0s 995us/step - loss: 0.0785
Epoch 47/300
1/1 [==============================] - 0s 998us/step - loss: 0.0783
Epoch 48/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0781
Epoch 49/300
1/1 [==============================] - 0s 997us/step - loss: 0.0779
Epoch 50/300
1/1 [==============================] - 0s 997us/step - loss: 0.0777
Epoch 51/300
1/1 [==============================] - 0s 995us/step - loss: 0.0775
Epoch 52/300
1/1 [==============================] - 0s 998us/step - loss: 0.0773
Epoch 53/300
1/1 [==============================] - 0s 996us/step - loss: 0.0771
Epoch 54/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0770
Epoch 55/300
1/1 [==============================] - 0s 0s/step - loss: 0.0768
Epoch 56/300
1/1 [==============================] - 0s 998us/step - loss: 0.0767
Epoch 57/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0765
Epoch 58/300
1/1 [==============================] - 0s 0s/step - loss: 0.0764
Epoch 59/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0762
Epoch 60/300
1/1 [==============================] - 0s 995us/step - loss: 0.0761
Epoch 61/300
1/1 [==============================] - 0s 998us/step - loss: 0.0759
Epoch 62/300
1/1 [==============================] - 0s 995us/step - loss: 0.0758
Epoch 63/300
1/1 [==============================] - 0s 997us/step - loss: 0.0757
Epoch 64/300
1/1 [==============================] - 0s 994us/step - loss: 0.0756
Epoch 65/300
1/1 [==============================] - 0s 973us/step - loss: 0.0754
Epoch 66/300
1/1 [==============================] - 0s 0s/step - loss: 0.0753
Epoch 67/300
1/1 [==============================] - 0s 997us/step - loss: 0.0752
Epoch 68/300
1/1 [==============================] - 0s 997us/step - loss: 0.0751
Epoch 69/300
1/1 [==============================] - 0s 0s/step - loss: 0.0750
Epoch 70/300
1/1 [==============================] - 0s 994us/step - loss: 0.0749
Epoch 71/300
1/1 [==============================] - 0s 999us/step - loss: 0.0748
Epoch 72/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0747
Epoch 73/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0746
Epoch 74/300
1/1 [==============================] - 0s 995us/step - loss: 0.0745
Epoch 75/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0744
Epoch 76/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0743
Epoch 77/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0742
Epoch 78/300
1/1 [==============================] - 0s 997us/step - loss: 0.0741
Epoch 79/300
1/1 [==============================] - 0s 0s/step - loss: 0.0741
Epoch 80/300
1/1 [==============================] - 0s 996us/step - loss: 0.0740
Epoch 81/300
1/1 [==============================] - 0s 997us/step - loss: 0.0739
Epoch 82/300
1/1 [==============================] - 0s 998us/step - loss: 0.0738
Epoch 83/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0738
Epoch 84/300
1/1 [==============================] - 0s 997us/step - loss: 0.0737
Epoch 85/300
1/1 [==============================] - 0s 999us/step - loss: 0.0736
Epoch 86/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0735
Epoch 87/300
1/1 [==============================] - 0s 997us/step - loss: 0.0735
Epoch 88/300
1/1 [==============================] - 0s 998us/step - loss: 0.0734
Epoch 89/300
1/1 [==============================] - 0s 998us/step - loss: 0.0733
Epoch 90/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0733
Epoch 91/300
1/1 [==============================] - 0s 998us/step - loss: 0.0732
Epoch 92/300
1/1 [==============================] - 0s 994us/step - loss: 0.0731
Epoch 93/300
1/1 [==============================] - 0s 998us/step - loss: 0.0731
Epoch 94/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0730
Epoch 95/300
1/1 [==============================] - 0s 0s/step - loss: 0.0730
Epoch 96/300
1/1 [==============================] - 0s 997us/step - loss: 0.0729
Epoch 97/300
1/1 [==============================] - 0s 998us/step - loss: 0.0728
Epoch 98/300
1/1 [==============================] - 0s 995us/step - loss: 0.0728
Epoch 99/300
1/1 [==============================] - 0s 993us/step - loss: 0.0727
Epoch 100/300
1/1 [==============================] - 0s 998us/step - loss: 0.0727
Epoch 101/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0726
Epoch 102/300
1/1 [==============================] - 0s 998us/step - loss: 0.0726
Epoch 103/300
1/1 [==============================] - 0s 997us/step - loss: 0.0725
Epoch 104/300
1/1 [==============================] - 0s 996us/step - loss: 0.0725
Epoch 105/300
1/1 [==============================] - 0s 945us/step - loss: 0.0724
Epoch 106/300
1/1 [==============================] - 0s 997us/step - loss: 0.0724
Epoch 107/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0723
Epoch 108/300
1/1 [==============================] - 0s 997us/step - loss: 0.0723
Epoch 109/300
1/1 [==============================] - 0s 998us/step - loss: 0.0722
Epoch 110/300
1/1 [==============================] - 0s 998us/step - loss: 0.0722
Epoch 111/300
1/1 [==============================] - 0s 997us/step - loss: 0.0721
Epoch 112/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0721
Epoch 113/300
1/1 [==============================] - 0s 993us/step - loss: 0.0720
Epoch 114/300
1/1 [==============================] - 0s 997us/step - loss: 0.0720
Epoch 115/300
1/1 [==============================] - 0s 998us/step - loss: 0.0719
Epoch 116/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0719
Epoch 117/300
1/1 [==============================] - 0s 995us/step - loss: 0.0718
Epoch 118/300
1/1 [==============================] - 0s 998us/step - loss: 0.0718
Epoch 119/300
1/1 [==============================] - 0s 0s/step - loss: 0.0718
Epoch 120/300
1/1 [==============================] - 0s 997us/step - loss: 0.0717
Epoch 121/300
1/1 [==============================] - 0s 996us/step - loss: 0.0717
Epoch 122/300
1/1 [==============================] - 0s 998us/step - loss: 0.0716
Epoch 123/300
1/1 [==============================] - 0s 994us/step - loss: 0.0716
Epoch 124/300
1/1 [==============================] - 0s 995us/step - loss: 0.0715
Epoch 125/300
1/1 [==============================] - 0s 998us/step - loss: 0.0715
Epoch 126/300
1/1 [==============================] - 0s 0s/step - loss: 0.0715
Epoch 127/300
1/1 [==============================] - 0s 985us/step - loss: 0.0714
Epoch 128/300
1/1 [==============================] - 0s 0s/step - loss: 0.0714
Epoch 129/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0713
Epoch 130/300
1/1 [==============================] - 0s 0s/step - loss: 0.0713
Epoch 131/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0713
Epoch 132/300
1/1 [==============================] - 0s 997us/step - loss: 0.0712
Epoch 133/300
1/1 [==============================] - 0s 998us/step - loss: 0.0712
Epoch 134/300
1/1 [==============================] - 0s 960us/step - loss: 0.0711
Epoch 135/300
1/1 [==============================] - 0s 0s/step - loss: 0.0711
Epoch 136/300
1/1 [==============================] - 0s 0s/step - loss: 0.0711
Epoch 137/300
1/1 [==============================] - 0s 996us/step - loss: 0.0710
Epoch 138/300
1/1 [==============================] - 0s 997us/step - loss: 0.0710
Epoch 139/300
1/1 [==============================] - 0s 997us/step - loss: 0.0709
Epoch 140/300
1/1 [==============================] - 0s 0s/step - loss: 0.0709
Epoch 141/300
1/1 [==============================] - 0s 998us/step - loss: 0.0709
Epoch 142/300
1/1 [==============================] - 0s 998us/step - loss: 0.0708
Epoch 143/300
1/1 [==============================] - 0s 997us/step - loss: 0.0708
Epoch 144/300
1/1 [==============================] - 0s 996us/step - loss: 0.0708
Epoch 145/300
1/1 [==============================] - 0s 996us/step - loss: 0.0707
Epoch 146/300
1/1 [==============================] - 0s 951us/step - loss: 0.0707
Epoch 147/300
1/1 [==============================] - 0s 0s/step - loss: 0.0706
Epoch 148/300
1/1 [==============================] - 0s 999us/step - loss: 0.0706
Epoch 149/300
1/1 [==============================] - 0s 998us/step - loss: 0.0706
Epoch 150/300
1/1 [==============================] - 0s 0s/step - loss: 0.0705
Epoch 151/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0705
Epoch 152/300
1/1 [==============================] - 0s 997us/step - loss: 0.0705
Epoch 153/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0704
Epoch 154/300
1/1 [==============================] - 0s 998us/step - loss: 0.0704
Epoch 155/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0704
Epoch 156/300
1/1 [==============================] - 0s 960us/step - loss: 0.0703
Epoch 157/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0703
Epoch 158/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0703
Epoch 159/300
1/1 [==============================] - 0s 996us/step - loss: 0.0702
Epoch 160/300
1/1 [==============================] - 0s 957us/step - loss: 0.0702
Epoch 161/300
1/1 [==============================] - 0s 997us/step - loss: 0.0701
Epoch 162/300
1/1 [==============================] - 0s 997us/step - loss: 0.0701
Epoch 163/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0701
Epoch 164/300
1/1 [==============================] - 0s 998us/step - loss: 0.0700
Epoch 165/300
1/1 [==============================] - 0s 997us/step - loss: 0.0700
Epoch 166/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0700
Epoch 167/300
1/1 [==============================] - 0s 997us/step - loss: 0.0699
Epoch 168/300
1/1 [==============================] - 0s 996us/step - loss: 0.0699
Epoch 169/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0699
Epoch 170/300
1/1 [==============================] - 0s 998us/step - loss: 0.0698
Epoch 171/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0698
Epoch 172/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0698
Epoch 173/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0697
Epoch 174/300
1/1 [==============================] - 0s 997us/step - loss: 0.0697
Epoch 175/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0697
Epoch 176/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0696
Epoch 177/300
1/1 [==============================] - 0s 999us/step - loss: 0.0696
Epoch 178/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0696
Epoch 179/300
1/1 [==============================] - 0s 998us/step - loss: 0.0695
Epoch 180/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0695
Epoch 181/300
1/1 [==============================] - 0s 998us/step - loss: 0.0695
Epoch 182/300
1/1 [==============================] - 0s 996us/step - loss: 0.0694
Epoch 183/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0694
Epoch 184/300
1/1 [==============================] - 0s 997us/step - loss: 0.0694
Epoch 185/300
1/1 [==============================] - 0s 997us/step - loss: 0.0693
Epoch 186/300
1/1 [==============================] - 0s 998us/step - loss: 0.0693
Epoch 187/300
1/1 [==============================] - 0s 998us/step - loss: 0.0693
Epoch 188/300
1/1 [==============================] - 0s 996us/step - loss: 0.0692
Epoch 189/300
1/1 [==============================] - 0s 997us/step - loss: 0.0692
Epoch 190/300
1/1 [==============================] - 0s 997us/step - loss: 0.0692
Epoch 191/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0691
Epoch 192/300
1/1 [==============================] - 0s 997us/step - loss: 0.0691
Epoch 193/300
1/1 [==============================] - 0s 996us/step - loss: 0.0691
Epoch 194/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0691
Epoch 195/300
1/1 [==============================] - 0s 998us/step - loss: 0.0690
Epoch 196/300
1/1 [==============================] - 0s 998us/step - loss: 0.0690
Epoch 197/300
1/1 [==============================] - 0s 997us/step - loss: 0.0690
Epoch 198/300
1/1 [==============================] - 0s 997us/step - loss: 0.0689
Epoch 199/300
1/1 [==============================] - 0s 997us/step - loss: 0.0689
Epoch 200/300
1/1 [==============================] - 0s 3ms/step - loss: 0.0689
Epoch 201/300
1/1 [==============================] - 0s 0s/step - loss: 0.0688
Epoch 202/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0688
Epoch 203/300
1/1 [==============================] - 0s 998us/step - loss: 0.0688
Epoch 204/300
1/1 [==============================] - 0s 962us/step - loss: 0.0687
Epoch 205/300
1/1 [==============================] - 0s 998us/step - loss: 0.0687
Epoch 206/300
1/1 [==============================] - 0s 991us/step - loss: 0.0687
Epoch 207/300
1/1 [==============================] - 0s 0s/step - loss: 0.0686
Epoch 208/300
1/1 [==============================] - 0s 995us/step - loss: 0.0686
Epoch 209/300
1/1 [==============================] - 0s 0s/step - loss: 0.0686
Epoch 210/300
1/1 [==============================] - 0s 998us/step - loss: 0.0685
Epoch 211/300
1/1 [==============================] - 0s 996us/step - loss: 0.0685
Epoch 212/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0685
Epoch 213/300
1/1 [==============================] - 0s 996us/step - loss: 0.0684
Epoch 214/300
1/1 [==============================] - 0s 997us/step - loss: 0.0684
Epoch 215/300
1/1 [==============================] - 0s 998us/step - loss: 0.0684
Epoch 216/300
1/1 [==============================] - 0s 999us/step - loss: 0.0684
Epoch 217/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0683
Epoch 218/300
1/1 [==============================] - 0s 990us/step - loss: 0.0683
Epoch 219/300
1/1 [==============================] - 0s 996us/step - loss: 0.0683
Epoch 220/300
1/1 [==============================] - 0s 0s/step - loss: 0.0682
Epoch 221/300
1/1 [==============================] - 0s 998us/step - loss: 0.0682
Epoch 222/300
1/1 [==============================] - 0s 997us/step - loss: 0.0682
Epoch 223/300
1/1 [==============================] - 0s 998us/step - loss: 0.0681
Epoch 224/300
1/1 [==============================] - 0s 998us/step - loss: 0.0681
Epoch 225/300
1/1 [==============================] - 0s 993us/step - loss: 0.0681
Epoch 226/300
1/1 [==============================] - 0s 986us/step - loss: 0.0680
Epoch 227/300
1/1 [==============================] - 0s 0s/step - loss: 0.0680
Epoch 228/300
1/1 [==============================] - 0s 996us/step - loss: 0.0680
Epoch 229/300
1/1 [==============================] - 0s 997us/step - loss: 0.0679
Epoch 230/300
1/1 [==============================] - 0s 998us/step - loss: 0.0679
Epoch 231/300
1/1 [==============================] - 0s 997us/step - loss: 0.0679
Epoch 232/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0679
Epoch 233/300
1/1 [==============================] - 0s 998us/step - loss: 0.0678
Epoch 234/300
1/1 [==============================] - 0s 995us/step - loss: 0.0678
Epoch 235/300
1/1 [==============================] - 0s 998us/step - loss: 0.0678
Epoch 236/300
1/1 [==============================] - 0s 996us/step - loss: 0.0677
Epoch 237/300
1/1 [==============================] - 0s 999us/step - loss: 0.0677
Epoch 238/300
1/1 [==============================] - 0s 997us/step - loss: 0.0677
Epoch 239/300
1/1 [==============================] - 0s 998us/step - loss: 0.0676
Epoch 240/300
1/1 [==============================] - 0s 997us/step - loss: 0.0676
Epoch 241/300
1/1 [==============================] - 0s 998us/step - loss: 0.0676
Epoch 242/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0676
Epoch 243/300
1/1 [==============================] - 0s 996us/step - loss: 0.0675
Epoch 244/300
1/1 [==============================] - 0s 995us/step - loss: 0.0675
Epoch 245/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0675
Epoch 246/300
1/1 [==============================] - 0s 927us/step - loss: 0.0674
Epoch 247/300
1/1 [==============================] - 0s 996us/step - loss: 0.0674
Epoch 248/300
1/1 [==============================] - 0s 998us/step - loss: 0.0674
Epoch 249/300
1/1 [==============================] - 0s 998us/step - loss: 0.0673
Epoch 250/300
1/1 [==============================] - 0s 996us/step - loss: 0.0673
Epoch 251/300
1/1 [==============================] - 0s 998us/step - loss: 0.0673
Epoch 252/300
1/1 [==============================] - 0s 997us/step - loss: 0.0672
Epoch 253/300
1/1 [==============================] - 0s 996us/step - loss: 0.0672
Epoch 254/300
1/1 [==============================] - 0s 998us/step - loss: 0.0672
Epoch 255/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0672
Epoch 256/300
1/1 [==============================] - 0s 998us/step - loss: 0.0671
Epoch 257/300
1/1 [==============================] - 0s 1000us/step - loss: 0.0671
Epoch 258/300
1/1 [==============================] - 0s 998us/step - loss: 0.0671
Epoch 259/300
1/1 [==============================] - 0s 3ms/step - loss: 0.0670
Epoch 260/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0670
Epoch 261/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0670
Epoch 262/300
1/1 [==============================] - 0s 995us/step - loss: 0.0669
Epoch 263/300
1/1 [==============================] - 0s 995us/step - loss: 0.0669
Epoch 264/300
1/1 [==============================] - 0s 996us/step - loss: 0.0669
Epoch 265/300
1/1 [==============================] - 0s 997us/step - loss: 0.0669
Epoch 266/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0668
Epoch 267/300
1/1 [==============================] - 0s 997us/step - loss: 0.0668
Epoch 268/300
1/1 [==============================] - 0s 0s/step - loss: 0.0668
Epoch 269/300
1/1 [==============================] - 0s 996us/step - loss: 0.0667
Epoch 270/300
1/1 [==============================] - 0s 0s/step - loss: 0.0667
Epoch 271/300
1/1 [==============================] - 0s 1ms/step - loss: 0.0667
Epoch 272/300
1/1 [==============================] - 0s 998us/step - loss: 0.0667
Epoch 273/300
1/1 [==============================] - 0s 999us/step - loss: 0.0666
Epoch 274/300
1/1 [==============================] - 0s 0s/step - loss: 0.0666
Epoch 275/300
1/1 [==============================] - 0s 996us/step - loss: 0.0666
Epoch 276/300
1/1 [==============================] - 0s 997us/step - loss: 0.0665
Epoch 277/300
1/1 [==============================] - 0s 1000us/step - loss: 0.0665
Epoch 278/300
1/1 [==============================] - 0s 997us/step - loss: 0.0665
Epoch 279/300
1/1 [==============================] - 0s 995us/step - loss: 0.0664
Epoch 280/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0664
Epoch 281/300
1/1 [==============================] - 0s 996us/step - loss: 0.0664
Epoch 282/300
1/1 [==============================] - 0s 999us/step - loss: 0.0664
Epoch 283/300
1/1 [==============================] - 0s 994us/step - loss: 0.0663
Epoch 284/300
1/1 [==============================] - 0s 965us/step - loss: 0.0663
Epoch 285/300
1/1 [==============================] - 0s 995us/step - loss: 0.0663
Epoch 286/300
1/1 [==============================] - 0s 997us/step - loss: 0.0662
Epoch 287/300
1/1 [==============================] - 0s 999us/step - loss: 0.0662
Epoch 288/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0662
Epoch 289/300
1/1 [==============================] - 0s 997us/step - loss: 0.0661
Epoch 290/300
1/1 [==============================] - 0s 999us/step - loss: 0.0661
Epoch 291/300
1/1 [==============================] - 0s 996us/step - loss: 0.0661
Epoch 292/300
1/1 [==============================] - 0s 997us/step - loss: 0.0661
Epoch 293/300
1/1 [==============================] - 0s 996us/step - loss: 0.0660
Epoch 294/300
1/1 [==============================] - 0s 995us/step - loss: 0.0660
Epoch 295/300
1/1 [==============================] - 0s 0s/step - loss: 0.0660
Epoch 296/300
1/1 [==============================] - 0s 968us/step - loss: 0.0659
Epoch 297/300
1/1 [==============================] - 0s 998us/step - loss: 0.0659
Epoch 298/300
1/1 [==============================] - 0s 996us/step - loss: 0.0659
Epoch 299/300
1/1 [==============================] - 0s 2ms/step - loss: 0.0659
Epoch 300/300
1/1 [==============================] - 0s 998us/step - loss: 0.0658
```

In [11]:
```
pred=model.predict(X)
pred
```

Out[11]:
```
array([[1.1854893 ],
       [0.5455201 ],
       [0.97490805],
       [1.5648893 ]], dtype=float32)
```

In [ ]:

You can see by increasing number of epochs, the model will greatly overfit training data. The easiest technique is to apply early stopping as we discussed before; it stops iteration when error in validation set start to increase while error in training set keep decreasing. So, first you need to divide data into training set and test set; then the training set should be divided to smaller training set and validation. The test set should be applied in the end only for measuring the error of the model.

In [ ]:

Visit the TensorFlow Playground at https://playground.tensorflow.org/, play with the parameters, change Hidden layera, Neurons.. choose regression and classification task to understand the problem

Energy Efficiency Data Set

Lets apply ANN for Energy Efficiency data set for both regression and classification. First get training data:

Binary Classification

In [12]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

df = pd.read_csv('./Data/Building_Heating_Load.csv',na_values=['NA','?',' '])
np.random.seed(32)
df=df.reindex(np.random.permutation(df.index))
df.reset_index(inplace=True, drop=True)
#
df_binary=df.copy()
df_binary.drop(['Heating Load','Multi-Classes'], axis=1, inplace=True)
#
df_binary['Binary Classes']=df_binary['Binary Classes'].replace('Low Level', 0)
df_binary['Binary Classes']=df_binary['Binary Classes'].replace('High Level', 1)

# Training and Test
spt = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in spt.split(df_binary, df_binary['Binary Classes']):
    train_set = df_binary.loc[train_idx]
    test_set  = df_binary.loc[test_idx]

#
X_train = train_set.drop("Binary Classes", axis=1)
y_train = train_set["Binary Classes"].values
#
scaler = StandardScaler()
X_train_Std=scaler.fit_transform(X_train)

# Smaller Training
# You need to divid your data to smaller training set and validation set for early stopping.
Training_c=np.concatenate((X_train_Std,np.array(y_train).reshape(-1,1)),axis=1)
Smaller_Training, Validation = train_test_split(Training_c, test_size=0.2, random_state=100)
#
Smaller_Training_Target=Smaller_Training[:,-1]
Smaller_Training=Smaller_Training[:,:-1]
#
Validation_Target=Validation[:,-1]
Validation=Validation[:,:-1]
```

Lets now train a ANN for binary classification with 3 hidden layers; each layer has 50 neurons. Th following code implement early stopping to avoid overfitting.

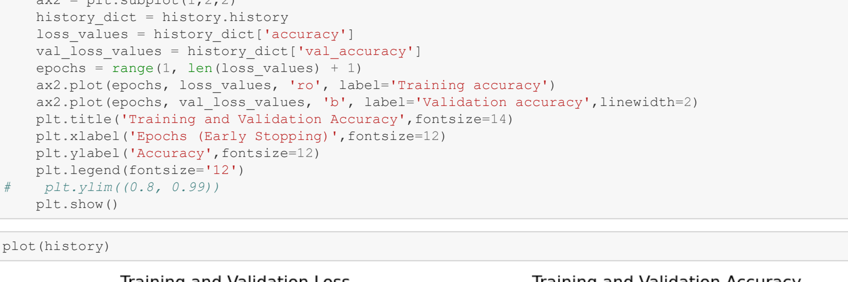
In [13]:
```
np.random.seed(42)
tf.random.set_seed(42)

keras.backend.clear_session() # Clear the previous model

# create model
model = keras.models.Sequential()

# Input & Hidden Layer 1
model.add(keras.layers.Dense(50, input_dim=np.array(Smaller_Training).shape[1], activation='relu'))

# Hidden Layer 2
model.add(keras.layers.Dense(50,activation='relu'))

# Hidden Layer 3
model.add(keras.layers.Dense(50,activation='relu'))

# Output Layer
model.add(keras.layers.Dense(1,activation='sigmoid'))
# use activation="softmax" for multicalss classification

# Compile model
model.compile(optimizer='adam',loss="binary_crossentropy",metrics=['accuracy'])
#'adam' is another optimization approach is more efficient than sgd. We will talk about it in more deta
ils.

# Early stopping to avoid overfitting
monitor= keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=1e-3,patience=3, mode='auto')
history=model.fit(Smaller_Training,Smaller_Training_Target,batch_size=32,validation_data=
         (Validation,Validation_Target),callbacks=[monitor],verbose=1,epochs=1000)
```

```
Epoch 1/1000
16/16 [==============================] - 0s 10ms/step - loss: 0.5822 - accuracy: 0.7576 - val_loss:
0.4568 - val_accuracy: 0.7886
Epoch 2/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3800 - accuracy: 0.8187 - val_loss: 0.
3279 - val_accuracy: 0.8293
Epoch 3/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2747 - accuracy: 0.8758 - val_loss: 0.
2663 - val_accuracy: 0.8780
Epoch 4/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2281 - accuracy: 0.9022 - val_loss: 0.
2334 - val_accuracy: 0.8943
Epoch 5/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2021 - accuracy: 0.9104 - val_loss: 0.
2132 - val_accuracy: 0.8943
Epoch 6/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1897 - accuracy: 0.9145 - val_loss: 0.
2023 - val_accuracy: 0.8943
Epoch 7/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1851 - accuracy: 0.9124 - val_loss: 0.
2091 - val_accuracy: 0.9024
Epoch 8/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1731 - accuracy: 0.9185 - val_loss: 0.
1928 - val_accuracy: 0.8943
Epoch 9/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1686 - accuracy: 0.9226 - val_loss: 0.
1849 - val_accuracy: 0.9187
Epoch 10/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1549 - accuracy: 0.9328 - val_loss: 0.
1845 - val_accuracy: 0.9024
Epoch 11/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1480 - accuracy: 0.9389 - val_loss: 0.
1828 - val_accuracy: 0.9106
Epoch 12/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1444 - accuracy: 0.9246 - val_loss: 0.
1801 - val_accuracy: 0.9024
Epoch 13/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1325 - accuracy: 0.9430 - val_loss: 0.
1873 - val_accuracy: 0.9187
Epoch 14/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1255 - accuracy: 0.9409 - val_loss: 0.
1787 - val_accuracy: 0.9106
Epoch 15/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1152 - accuracy: 0.9450 - val_loss: 0.
1816 - val_accuracy: 0.9106
Epoch 16/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1037 - accuracy: 0.9511 - val_loss: 0.
1774 - val_accuracy: 0.9024
Epoch 17/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1006 - accuracy: 0.9511 - val_loss: 0.
2134 - val_accuracy: 0.9350
Epoch 18/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1055 - accuracy: 0.9532 - val_loss: 0.
1943 - val_accuracy: 0.9106
Epoch 19/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0948 - accuracy: 0.9593 - val_loss: 0.
2048 - val_accuracy: 0.9187
```

In [ ]:

In [14]:
```python
import matplotlib.pyplot as plt

def plot(history):

    font = {'size'   : 10}
    plt.rc('font', **font)
    fig, ax=plt.subplots(figsize=(12, 4), dpi= 110, facecolor='w', edgecolor='k')

    ax1 = plt.subplot(1,2,1)
    history_dict = history.history
    loss_values = history_dict['loss']
    val_loss_values = history_dict['val_loss']

    epochs = range(1, len(loss_values) + 1)
    plt.plot(epochs, loss_values, 'bo', label='Training loss')
    plt.plot(epochs, val_loss_values, 'r', label='Validation loss',linewidth=2)
    plt.title('Training and Validation Loss',fontsize=14)
    plt.xlabel('Epochs (Early Stopping)',fontsize=12)
    plt.ylabel('Loss',fontsize=11)
    plt.legend(fontsize='12')
#    plt.ylim((0, 0.8))

    ax2 = plt.subplot(1,2,2)
    history_dict = history.history
    loss_values = history_dict['accuracy']
    val_loss_values = history_dict['val_accuracy']
    epochs = range(1, len(loss_values) + 1)
    ax2.plot(epochs, loss_values, 'ro', label='Training accuracy')
    ax2.plot(epochs, val_loss_values, 'b', label='Validation accuracy',linewidth=2)
    plt.title('Training and Validation Accuracy',fontsize=14)
    plt.xlabel('Epochs (Early Stopping)',fontsize=12)
    plt.ylabel('Accuracy',fontsize=12)
    plt.legend(fontsize='12')
#    plt.ylim((0.8, 0.99))
    plt.show()
```

In [15]:
```python
plot(history)
```



In [ ]:

In [16]:
```python
# Prediction using trained ANN for entire training
pred=model.predict(Smaller_Training[:5])
pred=[1 if i >= 0.5 else 0 for i in pred]
pred
```

Out[16]:
```
[0, 0, 1, 0, 0]
```

In [17]:
```python
# Actual values
Smaller_Training_Target[:5]
```

Out[17]:
```
array([0., 0., 1., 0., 0.])
```

Lets apply the trained model on the for the entire training set:

In [18]:
```python
from sklearn.metrics import accuracy_score

pred=model.predict(X_train_Std)
pred=[1 if i >= 0.5 else 0 for i in pred]

acr=accuracy_score(y_train, pred)
print(acr)
```

```
0.9592833876221498
```

Lets apply the model on test set which is the data that model has never seen before.

In [19]:
```python
X_test = test_set.drop("Binary Classes", axis=1)
y_test = test_set["Binary Classes"].values
#
X_test_Std=scaler.transform(X_test)
```

In [20]:
```python
from sklearn.metrics import accuracy_score

pred=model.predict(X_test_Std)
pred=[1 if i >= 0.5 else 0 for i in pred]

acr=accuracy_score(y_test, pred)
print(acr)
```

```
0.948051948051948
```

In [ ]:

Multiclass Classification

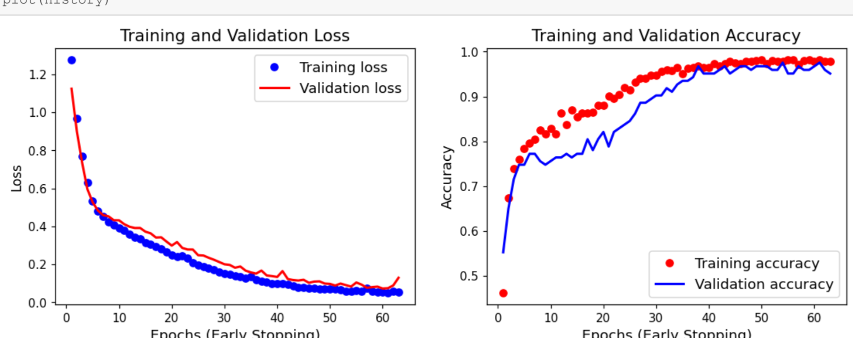
In [21]:
```python
from sklearn.preprocessing import OrdinalEncoder

df_multi=df.copy()
df_multi.drop(['Binary Classes','Heating Load'], axis=1, inplace=True)
#
ordinal_encoder = OrdinalEncoder()
Multi_Classes_encoded = ordinal_encoder.fit_transform(df_multi[['Multi-Classes']])
df_multi['Multi-Classes']=Multi_Classes_encoded

# Training and Test
spt = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in spt.split(df_multi, df_multi['Multi-Classes']):
    train_set = df_multi.loc[train_idx]
    test_set  = df_multi.loc[test_idx]

#
X_train = train_set.drop("Multi-Classes", axis=1)
y_train = train_set["Multi-Classes"].values
#
scaler = StandardScaler()
X_train_Std=scaler.fit_transform(X_train)

# Smaller Training
# You need to divid your data to smaller training set and validation set for early stopping.
Training_c=np.concatenate((X_train_Std,np.array(y_train).reshape(-1,1)),axis=1)
Smaller_Training, Validation = train_test_split(Training_c, test_size=0.2, random_state=100)
#
Smaller_Training_Target=Smaller_Training[:,-1]
Smaller_Training=Smaller_Training[:,:-1]
#
Validation_Target=Validation[:,-1]
Validation=Validation[:,:-1]
```

In [ ]:

In [22]:
```python
np.random.seed(42)
tf.random.set_seed(42)

keras.backend.clear_session() # Clear the previous model

# create model
model = keras.models.Sequential()

# Input and Hidden Layer 1
model.add(keras.layers.Dense(50, input_dim=np.array(Smaller_Training).shape[1], activation='relu'))

# Hidden Layer 2
model.add(keras.layers.Dense(50,activation='relu'))

# Hidden Layer 3
model.add(keras.layers.Dense(50,activation='relu'))

# Output Layer
model.add(keras.layers.Dense(4,activation='softmax'))
# we have 4 classes

# Compile model
model.compile(optimizer='adam',loss="sparse_categorical_crossentropy",metrics=['accuracy'])
#'adam' is another optimization approach is more efficient than sgd. We will talk about it in more deta
ils.

# Early stopping to avoid overfitting
monitor= keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=1e-3,patience=3, mode='auto')
history=model.fit(Smaller_Training,Smaller_Training_Target,batch_size=32,validation_data=
          (Validation,Validation_Target),callbacks=[monitor],verbose=1,epochs=1000)
```

```
Epoch 1/1000
16/16 [==============================] - 0s 15ms/step - loss: 1.2768 - accuracy: 0.4623 - val_loss:
1.1251 - val_accuracy: 0.5528
Epoch 2/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.9694 - accuracy: 0.6741 - val_loss: 0.
9023 - val_accuracy: 0.6504
Epoch 3/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.7677 - accuracy: 0.7393 - val_loss: 0.
7371 - val_accuracy: 0.7154
Epoch 4/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.6307 - accuracy: 0.7597 - val_loss: 0.
5969 - val_accuracy: 0.7480
Epoch 5/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.5354 - accuracy: 0.7841 - val_loss: 0.
5278 - val_accuracy: 0.7480
Epoch 6/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.4801 - accuracy: 0.7963 - val_loss: 0.
4840 - val_accuracy: 0.7724
Epoch 7/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.4529 - accuracy: 0.8045 - val_loss: 0.
4595 - val_accuracy: 0.7724
Epoch 8/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.4235 - accuracy: 0.8248 - val_loss: 0.
4535 - val_accuracy: 0.7561
Epoch 9/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.4083 - accuracy: 0.8167 - val_loss: 0.
4330 - val_accuracy: 0.7480
Epoch 10/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3919 - accuracy: 0.8289 - val_loss: 0.
4326 - val_accuracy: 0.7561
Epoch 11/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3798 - accuracy: 0.8167 - val_loss: 0.
4119 - val_accuracy: 0.7642
Epoch 12/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3606 - accuracy: 0.8635 - val_loss: 0.
3986 - val_accuracy: 0.7642
Epoch 13/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3448 - accuracy: 0.8371 - val_loss: 0.
3921 - val_accuracy: 0.7724
Epoch 14/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3343 - accuracy: 0.8697 - val_loss: 0.
3922 - val_accuracy: 0.7642
Epoch 15/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3136 - accuracy: 0.8554 - val_loss: 0.
3728 - val_accuracy: 0.7724
Epoch 16/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3072 - accuracy: 0.8635 - val_loss: 0.
3639 - val_accuracy: 0.7724
Epoch 17/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2942 - accuracy: 0.8635 - val_loss: 0.
3418 - val_accuracy: 0.8049
Epoch 18/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2804 - accuracy: 0.8656 - val_loss: 0.
3431 - val_accuracy: 0.7805
Epoch 19/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2649 - accuracy: 0.8798 - val_loss: 0.
3206 - val_accuracy: 0.8049
Epoch 20/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2517 - accuracy: 0.8798 - val_loss: 0.
2995 - val_accuracy: 0.8211
Epoch 21/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2427 - accuracy: 0.9002 - val_loss: 0.
3180 - val_accuracy: 0.7886
Epoch 22/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2450 - accuracy: 0.8961 - val_loss: 0.
2872 - val_accuracy: 0.8211
Epoch 23/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2337 - accuracy: 0.9043 - val_loss: 0.
2789 - val_accuracy: 0.8293
Epoch 24/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2107 - accuracy: 0.9206 - val_loss: 0.
2786 - val_accuracy: 0.8374
Epoch 25/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1970 - accuracy: 0.9145 - val_loss: 0.
2488 - val_accuracy: 0.8455
Epoch 26/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1872 - accuracy: 0.9328 - val_loss: 0.
2473 - val_accuracy: 0.8618
Epoch 27/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1805 - accuracy: 0.9409 - val_loss: 0.
2354 - val_accuracy: 0.8862
Epoch 28/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1745 - accuracy: 0.9409 - val_loss: 0.
2249 - val_accuracy: 0.8862
Epoch 29/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1622 - accuracy: 0.9470 - val_loss: 0.
2130 - val_accuracy: 0.8943
Epoch 30/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1530 - accuracy: 0.9470 - val_loss: 0.
2011 - val_accuracy: 0.9024
Epoch 31/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1472 - accuracy: 0.9552 - val_loss: 0.
1976 - val_accuracy: 0.9024
Epoch 32/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1415 - accuracy: 0.9593 - val_loss: 0.
1829 - val_accuracy: 0.9187
Epoch 33/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1345 - accuracy: 0.9572 - val_loss: 0.
1894 - val_accuracy: 0.9106
Epoch 34/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1290 - accuracy: 0.9654 - val_loss: 0.
1676 - val_accuracy: 0.9268
Epoch 35/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1345 - accuracy: 0.9511 - val_loss: 0.
1584 - val_accuracy: 0.9350
Epoch 36/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1185 - accuracy: 0.9633 - val_loss: 0.
1518 - val_accuracy: 0.9350
Epoch 37/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1124 - accuracy: 0.9654 - val_loss: 0.
1680 - val_accuracy: 0.9431
Epoch 38/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1066 - accuracy: 0.9674 - val_loss: 0.
1424 - val_accuracy: 0.9675
Epoch 39/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1000 - accuracy: 0.9654 - val_loss: 0.
1387 - val_accuracy: 0.9512
Epoch 40/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1010 - accuracy: 0.9654 - val_loss: 0.
1344 - val_accuracy: 0.9512
Epoch 41/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0984 - accuracy: 0.9735 - val_loss: 0.
1652 - val_accuracy: 0.9512
Epoch 42/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0963 - accuracy: 0.9674 - val_loss: 0.
1236 - val_accuracy: 0.9593
Epoch 43/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0880 - accuracy: 0.9735 - val_loss: 0.
1179 - val_accuracy: 0.9675
Epoch 44/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0811 - accuracy: 0.9776 - val_loss: 0.
1154 - val_accuracy: 0.9512
Epoch 45/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0784 - accuracy: 0.9756 - val_loss: 0.
1194 - val_accuracy: 0.9593
Epoch 46/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0766 - accuracy: 0.9735 - val_loss: 0.
1036 - val_accuracy: 0.9675
Epoch 47/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0751 - accuracy: 0.9776 - val_loss: 0.
1098 - val_accuracy: 0.9675
Epoch 48/1000
16/16 [==============================] - ETA: 0s - loss: 0.0284 - accuracy: 1.00 - 0s 2ms/step - los
s: 0.0725 - accuracy: 0.9776 - val_loss: 0.1112 - val_accuracy: 0.9593
Epoch 49/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0693 - accuracy: 0.9796 - val_loss: 0.
0996 - val_accuracy: 0.9675
Epoch 50/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0692 - accuracy: 0.9817 - val_loss: 0.
0977 - val_accuracy: 0.9675
Epoch 51/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0704 - accuracy: 0.9735 - val_loss: 0.
0891 - val_accuracy: 0.9675
Epoch 52/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0677 - accuracy: 0.9796 - val_loss: 0.
0996 - val_accuracy: 0.9593
Epoch 53/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0599 - accuracy: 0.9776 - val_loss: 0.
0915 - val_accuracy: 0.9593
Epoch 54/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0592 - accuracy: 0.9776 - val_loss: 0.
0837 - val_accuracy: 0.9756
Epoch 55/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0612 - accuracy: 0.9817 - val_loss: 0.
1052 - val_accuracy: 0.9512
Epoch 56/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0580 - accuracy: 0.9817 - val_loss: 0.
0938 - val_accuracy: 0.9512
Epoch 57/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0736 - accuracy: 0.9715 - val_loss: 0.
0774 - val_accuracy: 0.9675
Epoch 58/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0572 - accuracy: 0.9796 - val_loss: 0.
0807 - val_accuracy: 0.9593
Epoch 59/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0540 - accuracy: 0.9817 - val_loss: 0.
0834 - val_accuracy: 0.9593
Epoch 60/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0546 - accuracy: 0.9776 - val_loss: 0.
0733 - val_accuracy: 0.9675
Epoch 61/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0503 - accuracy: 0.9817 - val_loss: 0.
0752 - val_accuracy: 0.9756
Epoch 62/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0597 - accuracy: 0.9776 - val_loss: 0.
0897 - val_accuracy: 0.9593
Epoch 63/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0556 - accuracy: 0.9776 - val_loss: 0.
1302 - val_accuracy: 0.9512
```

In [23]:
```python
plot(history)
```



In [ ]:

In [24]:
```python
# Prediction using trained ANN
pred=model.predict(Smaller_Training[:5])
pred=[np.argmax(prob) for prob in pred]
pred
```

Out[24]:
```
[0, 0, 3, 2, 2]
```

In [25]:
```python
# Actual values
Smaller_Training_Target[:5]
```

Out[25]:
```
array([0., 0., 3., 2., 2.])
```

Apply the trained model for the entire training set:

In [26]:
```python
pred=model.predict(X_train_Std)
pred=[np.argmax(prob) for prob in pred]

acr=accuracy_score(y_train, pred)
print(acr)
```

```
0.9739413680781759
```

Lets apply the model on test set which is the data that model has never seen before.

In [27]:
```python
X_test = test_set.drop("Multi-Classes", axis=1)
y_test = test_set["Multi-Classes"].values
#
X_test_Std=scaler.transform(X_test)
```

In [28]:
```python
from sklearn.metrics import accuracy_score

pred=model.predict(X_test_Std)
pred=[np.argmax(prob) for prob in pred]

acr=accuracy_score(y_test, pred)
print(acr)
```

```
0.9090909090909091
```

Regression

In [29]:
```python
df_reg=df.copy()
df_reg.drop(['Binary Classes','Multi-Classes'], axis=1, inplace=True)

# Training and Test
train_set, test_set = train_test_split(df_reg, test_size=0.2, random_state=42)

#
X_train = train_set.drop("Heating Load", axis=1)
y_train = train_set["Heating Load"].values
#
scaler = StandardScaler()
X_train_Std=scaler.fit_transform(X_train)

# Smaller Training
# You need to divid your data to smaller training set and validation set for early stopping.
Training_c=np.concatenate((X_train_Std,np.array(y_train).reshape(-1,1)),axis=1)
Smaller_Training, Validation = train_test_split(Training_c, test_size=0.2, random_state=100)
#
Smaller_Training_Target=Smaller_Training[:,-1]
Smaller_Training=Smaller_Training[:,:-1]
#
Validation_Target=Validation[:,-1]
Validation=Validation[:,:-1]
```

In [ ]:

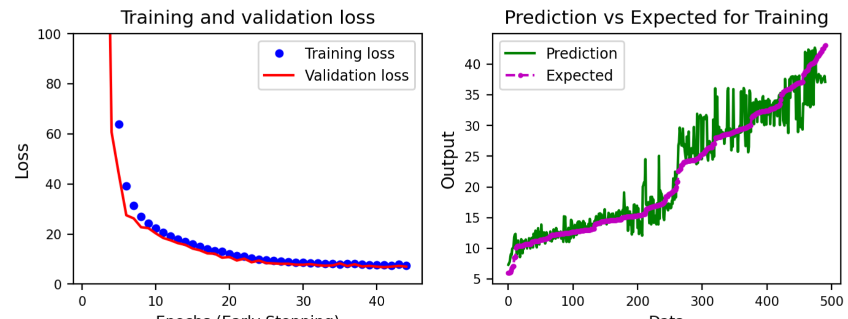
In [30]:
```python
np.random.seed(42)
tf.random.set_seed(42)

keras.backend.clear_session() # Clear the previous model

# create model
model = keras.models.Sequential()

# Input and Hidden Layer 1
model.add(keras.layers.Dense(50, input_dim=np.array(Smaller_Training).shape[1], activation='relu'))

# Hidden Layer 2
model.add(keras.layers.Dense(50,activation='relu'))

# Hidden Layer 3
model.add(keras.layers.Dense(50,activation='relu'))

# Output Layer
model.add(keras.layers.Dense(1))

# Compile model
model.compile(optimizer='adam',loss="mse")
# 'adam' is another optimization approach is more efficient than sgd. We will talk about it in more det
ails.
# 'mse' is mean square error which is applied for regression

# Early stopping to avoid overfitting
monitor= keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=1e-3,patience=3, mode='auto')
history=model.fit(Smaller_Training,Smaller_Training_Target,batch_size=32,validation_data=
          (Validation,Validation_Target),callbacks=[monitor],verbose=1,epochs=1000)
```

```
Epoch 1/1000
16/16 [==============================] - 0s 7ms/step - loss: 556.5300 - val_loss: 617.5810
Epoch 2/1000
16/16 [==============================] - 0s 2ms/step - loss: 487.8694 - val_loss: 495.5008
Epoch 3/1000
16/16 [==============================] - 0s 2ms/step - loss: 348.5084 - val_loss: 263.0908
Epoch 4/1000
16/16 [==============================] - 0s 2ms/step - loss: 143.5860 - val_loss: 60.7297
Epoch 5/1000
16/16 [==============================] - 0s 2ms/step - loss: 63.8592 - val_loss: 43.6602
Epoch 6/1000
16/16 [==============================] - 0s 2ms/step - loss: 39.3157 - val_loss: 27.5265
Epoch 7/1000
16/16 [==============================] - 0s 2ms/step - loss: 31.4174 - val_loss: 26.2661
Epoch 8/1000
16/16 [==============================] - 0s 2ms/step - loss: 26.9751 - val_loss: 22.7469
Epoch 9/1000
16/16 [==============================] - 0s 2ms/step - loss: 24.2055 - val_loss: 22.3535
Epoch 10/1000
16/16 [==============================] - 0s 2ms/step - loss: 22.3141 - val_loss: 20.2912
Epoch 11/1000
16/16 [==============================] - 0s 2ms/step - loss: 20.5691 - val_loss: 18.4587
Epoch 12/1000
16/16 [==============================] - 0s 2ms/step - loss: 19.2199 - val_loss: 17.5179
Epoch 13/1000
16/16 [==============================] - 0s 2ms/step - loss: 18.0143 - val_loss: 16.3742
Epoch 14/1000
16/16 [==============================] - 0s 2ms/step - loss: 16.9375 - val_loss: 15.7136
Epoch 15/1000
16/16 [==============================] - 0s 2ms/step - loss: 16.0073 - val_loss: 14.2242
Epoch 16/1000
16/16 [==============================] - 0s 2ms/step - loss: 15.0520 - val_loss: 13.4981
Epoch 17/1000
16/16 [==============================] - 0s 2ms/step - loss: 14.0176 - val_loss: 12.2931
Epoch 18/1000
16/16 [==============================] - 0s 2ms/step - loss: 13.3079 - val_loss: 12.1714
Epoch 19/1000
16/16 [==============================] - 0s 2ms/step - loss: 12.9443 - val_loss: 10.5775
Epoch 20/1000
16/16 [==============================] - 0s 2ms/step - loss: 11.9876 - val_loss: 10.8459
Epoch 21/1000
16/16 [==============================] - 0s 2ms/step - loss: 11.4372 - val_loss: 9.4545
Epoch 22/1000
16/16 [==============================] - 0s 2ms/step - loss: 11.0544 - val_loss: 10.1898
Epoch 23/1000
16/16 [==============================] - 0s 2ms/step - loss: 10.3113 - val_loss: 8.7403
Epoch 24/1000
16/16 [==============================] - 0s 2ms/step - loss: 9.9459 - val_loss: 9.4022
Epoch 25/1000
16/16 [==============================] - 0s 2ms/step - loss: 9.6834 - val_loss: 8.4257
Epoch 26/1000
16/16 [==============================] - 0s 2ms/step - loss: 9.3336 - val_loss: 8.2730
Epoch 27/1000
16/16 [==============================] - 0s 2ms/step - loss: 9.0927 - val_loss: 8.1060
Epoch 28/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.8698 - val_loss: 8.1399
Epoch 29/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.7586 - val_loss: 7.7792
Epoch 30/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.5881 - val_loss: 7.7569
Epoch 31/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.5016 - val_loss: 7.9986
Epoch 32/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.3818 - val_loss: 7.5497
Epoch 33/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.1607 - val_loss: 7.3497
Epoch 34/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.2662 - val_loss: 7.4865
Epoch 35/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.8319 - val_loss: 8.2898
Epoch 36/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.1632 - val_loss: 7.2159
Epoch 37/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.0737 - val_loss: 7.9438
Epoch 38/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.8347 - val_loss: 7.0586
Epoch 39/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.7487 - val_loss: 7.2865
Epoch 40/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.6372 - val_loss: 6.9797
Epoch 41/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.6165 - val_loss: 6.7889
Epoch 42/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.5040 - val_loss: 6.9834
Epoch 43/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.9819 - val_loss: 7.2834
Epoch 44/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.4035 - val_loss: 6.9784
```

In [ ]:

In [31]:
```python
def plot_NN(model,history,x_train,y_train):
    """ Plot training loss versus validation loss and
    training accuracy versus validation accuracy"""

    font = {'size'   : 7.5}
    plt.rc('font', **font)
    fig, ax=plt.subplots(figsize=(7, 5), dpi= 200, facecolor='w', edgecolor='k')

    ax1 = plt.subplot(2,2,1)
    history_dict = history.history
    loss_values = history_dict['loss']
    val_loss_values = history_dict['val_loss']
    epochs = range(1, len(loss_values) + 1)
    ax1.plot(epochs, loss_values, 'bo',markersize=4, label='Training loss')
    ax1.plot(epochs, val_loss_values, 'r-', label='Validation loss')
    plt.title('Training and validation loss',fontsize=11)
    plt.xlabel('Epochs (Early Stopping)',fontsize=9)
    plt.ylabel('Loss',fontsize=10)
    plt.legend(fontsize='8.5')
    plt.ylim((0, 100))

    ax2 = plt.subplot(2,2,2)
    pred=model.predict(x_train)
    t = pd.DataFrame({'pred': pred.flatten(), 'y': y_train.flatten()})
    t.sort_values(by=['y'], inplace=True)

    epochs = range(1, len(loss_values) + 1)
    ax2.plot(t['pred'].tolist(), 'g', label='Prediction')
    ax2.plot(t['y'].tolist(), 'm--o',markersize=2, label='Expected')
    plt.title('Prediction vs Expected for Training',fontsize=11)
    plt.xlabel('Data',fontsize=9)
    plt.ylabel('Output',fontsize=10)
    plt.legend(fontsize='8.5')
    fig.tight_layout(w_pad=1.42)
    plt.show()
```

In [32]:
```python
plot_NN(model,history,Smaller_Training,Smaller_Training_Target)
```



Lets get RMSE for training set:

In [33]:
```python
from sklearn.metrics import mean_squared_error

pred=model.predict(X_train_Std)
mse = mean_squared_error(y_train, pred)
rmse= np.sqrt(mse)
rmse
```

Out[33]:
```
2.676985308315499
```

Get RMSE for test set:

In [34]:
```python
X_test = test_set.drop("Heating Load", axis=1)
y_test = test_set["Heating Load"].values
#
X_test_Std=scaler.transform(X_test)
```

In [35]:
```python
pred=model.predict(X_test_Std)
mse = mean_squared_error(y_test, pred)
rmse= np.sqrt(mse)
rmse
```

Out[35]:
```
2.9973878342651545
```

You should always expect less performance on test set since the model has not seen this data set before.

In [ ]:

Fine-tune Neural Network Hyperparameters

Classification: GridSearchCV

You can use Scikit-Learn to fune-tune ANN's hyperparameters. You need a function for ANN to fine-tune hyperparameters. The following code is ANN with 3 hidden layers. You can adjust any hyperparameter.

In [36]:
```python
def ANN (input_dim,neurons=50,loss="binary_crossentropy",activation="relu",Nout=1,
             metrics=['accuracy'],activation_out='sigmoid'):
    """ Function to run Neural Network for different hyperparameters"""
    np.random.seed(42)
    tf.random.set_seed(42)

    # create model
    model = keras.models.Sequential()

    # Input and Hidden Layer 1
    model.add(keras.layers.Dense(neurons,input_dim=input_dim, activation=activation))

    # Input Layer 2
    model.add(keras.layers.Dense(neurons,activation=activation))

    # Input Layer 3
    model.add(keras.layers.Dense(neurons,activation=activation))

    # Output Layer
    model.add(keras.layers.Dense(Nout,activation=activation_out))

    # Compile model
    model.compile(optimizer='adam',loss=loss,metrics=metrics)
    return model
```

Then you can fine-tune number of neurons with the following code for the binary classification

In [37]:
```python
# Training and Test
spt = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in spt.split(df_binary, df_binary['Binary Classes']):
    train_set = df_binary.loc[train_idx]
    test_set  = df_binary.loc[test_idx]

#
X_train = train_set.drop("Binary Classes", axis=1)
y_train = train_set["Binary Classes"].values
#
scaler = StandardScaler()
X_train_Std=scaler.fit_transform(X_train)

# Smaller Training
# You need to divid your data to smaller training set and validation set for early stopping.
Training_c=np.concatenate((X_train_Std,np.array(y_train).reshape(-1,1)),axis=1)
Smaller_Training, Validation = train_test_split(Training_c, test_size=0.2, random_state=100)
#
Smaller_Training_Target=Smaller_Training[:,-1]
Smaller_Training=Smaller_Training[:,:-1]
#
Validation_Target=Validation[:,-1]
Validation=Validation[:,:-1]
```

The following code fine-tune only number of neuron, but you can simply apply for all parameters. The only issue is the process can be extremely computationally expensive and large number of parameters to adjust.

In [38]:
```python
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV


# define the grid search parameters
param_grid = {'neurons' : [50,100,150]}

# Run Keras Classifier
model = KerasClassifier(build_fn=ANN,input_dim=Smaller_Training.shape[1])

# Apply Scikit Learn GridSearchCV
grid = GridSearchCV(estimator=model,param_grid=param_grid, cv=2)

# Early stopping to avoid overfitting
monitor= keras.callbacks.EarlyStopping(min_delta=1e-3,patience=5, verbose=0)
grid_result = grid.fit(Smaller_Training,Smaller_Training_Target,batch_size=32,validation_data=
                       (Validation,Validation_Target),callbacks=[monitor],
                    verbose=0,epochs=1000)
```

```
8/8 [==============================] - 0s 998us/step - loss: 0.2535 - accuracy: 0.9065
8/8 [==============================] - 0s 873us/step - loss: 0.1597 - accuracy: 0.9061
8/8 [==============================] - 0s 873us/step - loss: 0.2367 - accuracy: 0.9065
8/8 [==============================] - 0s 997us/step - loss: 0.1443 - accuracy: 0.9102
8/8 [==============================] - 0s 997us/step - loss: 0.2278 - accuracy: 0.9106
8/8 [==============================] - 0s 999us/step - loss: 0.1179 - accuracy: 0.9306
```

In [39]:
```python
# Best result
print("Best parameters: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best parameters: 0.920591 using {'neurons': 150}
```

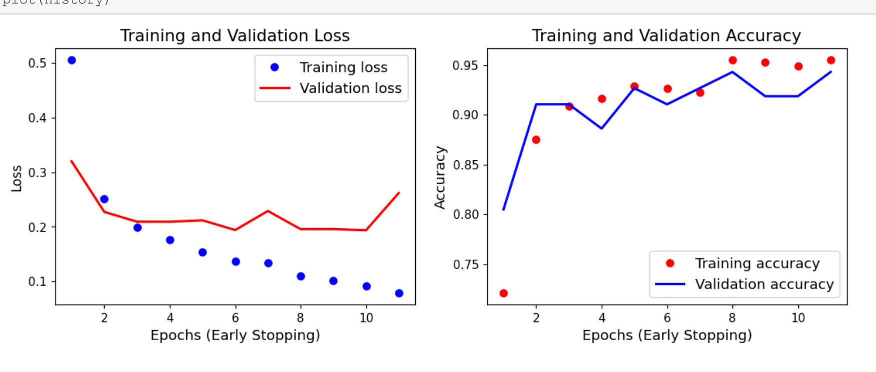
In [ ]:

You can fune-tune any ANN hyperparameter. Then call the function used fine-tuned neurons.

In [40]:
```python
# Call Function with fined-tune numebr of neurons
model_ft=ANN (input_dim=Smaller_Training.shape[1],neurons=150)

# Early stopping to avoid overfitting
monitor= keras.callbacks.EarlyStopping(min_delta=1e-3,patience=5)
history=model_ft.fit(Smaller_Training,Smaller_Training_Target,batch_size=32,validation_data=
          (Validation,Validation_Target),callbacks=[monitor],verbose=1,epochs=1000)
```

```
Epoch 1/1000
16/16 [==============================] - 0s 15ms/step - loss: 0.5053 - accuracy: 0.7210 - val_loss:
0.3199 - val_accuracy: 0.8049
Epoch 2/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2507 - accuracy: 0.8758 - val_loss: 0.
2273 - val_accuracy: 0.9106
Epoch 3/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1992 - accuracy: 0.9084 - val_loss: 0.
2093 - val_accuracy: 0.9106
Epoch 4/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1765 - accuracy: 0.9165 - val_loss: 0.
2093 - val_accuracy: 0.8862
Epoch 5/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1533 - accuracy: 0.9287 - val_loss: 0.
2119 - val_accuracy: 0.9268
Epoch 6/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1375 - accuracy: 0.9267 - val_loss: 0.
1941 - val_accuracy: 0.9106
Epoch 7/1000
16/16 [==============================] - ETA: 0s - loss: 0.1135 - accuracy: 1.00 - 0s 2ms/step - los
s: 0.1347 - accuracy: 0.9226 - val_loss: 0.2290 - val_accuracy: 0.9268
Epoch 8/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1097 - accuracy: 0.9552 - val_loss: 0.
1957 - val_accuracy: 0.9431
Epoch 9/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1014 - accuracy: 0.9532 - val_loss: 0.
1958 - val_accuracy: 0.9187
Epoch 10/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0919 - accuracy: 0.9491 - val_loss: 0.
1937 - val_accuracy: 0.9187
Epoch 11/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.0789 - accuracy: 0.9552 - val_loss: 0.
2618 - val_accuracy: 0.9431
```

In [41]:
```
plot(history)
```



In [42]:
```
X_test = test_set.drop("Binary Classes", axis=1)
y_test = test_set["Binary Classes"].values
#
X_test_Std=scaler.transform(X_test)
```

In [43]:
```
from sklearn.metrics import accuracy_score

pred=model_ft.predict(X_test_Std)
pred=[1 if i >= 0.5 else 0 for i in pred]

acr=accuracy_score(y_test, pred)
print(acr)
```

```
0.9675324675324676
```

You can see the accuracy you calculated is higher than "grid_result.best*score*" because of some overfiting. So, it is always a good idea

In [ ]:

Regression: GridSearchCV

In [44]:
```
df_reg=df.copy()
df_reg.drop(['Binary Classes','Multi-Classes'], axis=1, inplace=True)

# Training and Test
train_set, test_set = train_test_split(df_reg, test_size=0.2, random_state=42)

X_train = train_set.drop("Heating Load", axis=1)
y_train = train_set["Heating Load"].values
#
scaler = StandardScaler()
X_train_Std=scaler.fit_transform(X_train)

# Smaller Training
# You need to divid your data to smaller training set and validation set for early stopping.
Training_c=np.concatenate((X_train_Std,np.array(y_train).reshape(-1,1)),axis=1)
Smaller_Training, Validation = train_test_split(Training_c, test_size=0.2, random_state=100)
#
Smaller_Training_Target=Smaller_Training[:,-1]
Smaller_Training=Smaller_Training[:,:-1]
#
Validation_Target=Validation[:,-1]
Validation=Validation[:,:-1]
```

In [ ]:

In [45]:
```
from keras.wrappers.scikit_learn import KerasRegressor

# define the grid search parameters
param_grid = {'neurons' : [50,100,150]}

# Run Keras Classifier
model = KerasRegressor(build_fn=ANN,input_dim=Smaller_Training.shape[1],loss='mse'
                       ,metrics=None,activation_out=None)
#model. _estimator_type = "regressor"

# Apply Scikit Learn GridSearchCV
grid = GridSearchCV(estimator=model,param_grid=param_grid, cv=2)

# Early stopping to avoid overfitting
monitor= keras.callbacks.EarlyStopping(min_delta=1e-3,patience=5, verbose=0)
grid_result = grid.fit(Smaller_Training,Smaller_Training_Target,batch_size=32,validation_data=
                       (Validation,Validation_Target),callbacks=[monitor],
                  verbose=0,epochs=1000)
```

```
8/8 [==============================] - 0s 628us/step - loss: 3.2498
8/8 [==============================] - 0s 502us/step - loss: 10.2170
8/8 [==============================] - 0s 623us/step - loss: 5.1038
8/8 [==============================] - 0s 499us/step - loss: 8.7048
8/8 [==============================] - 0s 624us/step - loss: 3.0345
8/8 [==============================] - 0s 628us/step - loss: 6.0309
```

In [46]:
```
# Best result
print("Best parameters: %f using %s" % (-1*grid_result.best_score_, grid_result.best_params_))
```

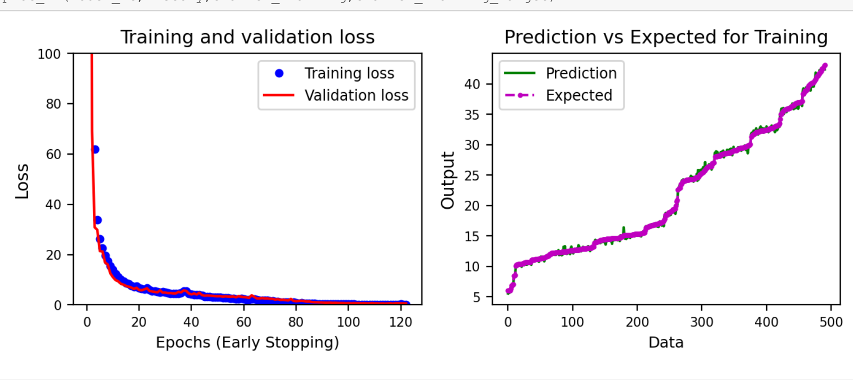
```
Best parameters: 4.532704 using {'neurons': 150}
```

You can fune-tune any ANN hyperparameter. Then call the function used fine-tuned neurons.

In [47]:
```
# Call Function with fined-tune numebr of neurons
model_ft=ANN (input_dim=Smaller_Training.shape[1],neurons=150,loss='mse',metrics=None,activation_out=No
ne)

# Early stopping to avoid overfitting
monitor= keras.callbacks.EarlyStopping(min_delta=1e-3,patience=5)
history=model_ft.fit(Smaller_Training,Smaller_Training_Target,batch_size=32,validation_data=
          (Validation,Validation_Target),callbacks=[monitor],verbose=1,epochs=1000)
```

```
Epoch 1/1000
16/16 [==============================] - 0s 7ms/step - loss: 496.0419 - val_loss: 411.0409
Epoch 2/1000
16/16 [==============================] - 0s 2ms/step - loss: 189.4804 - val_loss: 69.1375
Epoch 3/1000
16/16 [==============================] - 0s 2ms/step - loss: 61.8730 - val_loss: 30.8594
Epoch 4/1000
16/16 [==============================] - 0s 2ms/step - loss: 33.7987 - val_loss: 29.8524
Epoch 5/1000
16/16 [==============================] - 0s 2ms/step - loss: 26.3349 - val_loss: 21.3596
Epoch 6/1000
16/16 [==============================] - 0s 2ms/step - loss: 22.6852 - val_loss: 21.2456
Epoch 7/1000
16/16 [==============================] - 0s 2ms/step - loss: 19.7035 - val_loss: 16.8350
Epoch 8/1000
16/16 [==============================] - 0s 2ms/step - loss: 17.5764 - val_loss: 16.0825
Epoch 9/1000
16/16 [==============================] - 0s 2ms/step - loss: 15.4212 - val_loss: 12.5194
Epoch 10/1000
16/16 [==============================] - 0s 2ms/step - loss: 14.0032 - val_loss: 11.0032
Epoch 11/1000
16/16 [==============================] - 0s 2ms/step - loss: 12.4288 - val_loss: 9.8423
Epoch 12/1000
16/16 [==============================] - 0s 2ms/step - loss: 11.0770 - val_loss: 9.3910
Epoch 13/1000
16/16 [==============================] - 0s 2ms/step - loss: 10.1157 - val_loss: 8.4061
Epoch 14/1000
16/16 [==============================] - 0s 2ms/step - loss: 9.3192 - val_loss: 8.0439
Epoch 15/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.6984 - val_loss: 7.5356
Epoch 16/1000
16/16 [==============================] - 0s 2ms/step - loss: 8.3997 - val_loss: 7.2205
Epoch 17/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.6843 - val_loss: 6.8987
Epoch 18/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.2509 - val_loss: 6.3502
Epoch 19/1000
16/16 [==============================] - 0s 2ms/step - loss: 7.4551 - val_loss: 6.8005
Epoch 20/1000
16/16 [==============================] - 0s 2ms/step - loss: 6.6192 - val_loss: 5.8998
Epoch 21/1000
16/16 [==============================] - 0s 2ms/step - loss: 6.5088 - val_loss: 5.8347
Epoch 22/1000
16/16 [==============================] - 0s 2ms/step - loss: 6.1192 - val_loss: 6.1785
Epoch 23/1000
16/16 [==============================] - 0s 2ms/step - loss: 6.6946 - val_loss: 6.6095
Epoch 24/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.9191 - val_loss: 5.5249
Epoch 25/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.5505 - val_loss: 5.3439
Epoch 26/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.4321 - val_loss: 5.1484
Epoch 27/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.2984 - val_loss: 4.9999
Epoch 28/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.0261 - val_loss: 5.7480
Epoch 29/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.2190 - val_loss: 4.8931
Epoch 30/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.0271 - val_loss: 4.9039
Epoch 31/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.7875 - val_loss: 4.6973
Epoch 32/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.5833 - val_loss: 4.6911
Epoch 33/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.6143 - val_loss: 4.5598
Epoch 34/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.4251 - val_loss: 4.5542
Epoch 35/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.4548 - val_loss: 5.1648
Epoch 36/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.8674 - val_loss: 4.9474
Epoch 37/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.5750 - val_loss: 5.7298
Epoch 38/1000
16/16 [==============================] - 0s 2ms/step - loss: 5.4152 - val_loss: 5.5497
Epoch 39/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.3875 - val_loss: 4.5470
Epoch 40/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.2398 - val_loss: 4.1655
Epoch 41/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.1187 - val_loss: 4.5292
Epoch 42/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.0220 - val_loss: 4.1258
Epoch 43/1000
16/16 [==============================] - 0s 2ms/step - loss: 4.0955 - val_loss: 4.8214
Epoch 44/1000
16/16 [==============================] - 0s 2ms/step - loss: 3.7867 - val_loss: 3.9332
Epoch 45/1000
16/16 [==============================] - 0s 2ms/step - loss: 3.3385 - val_loss: 3.6392
Epoch 46/1000
16/16 [==============================] - 0s 2ms/step - loss: 3.3016 - val_loss: 3.7124
Epoch 47/1000
16/16 [==============================] - 0s 2ms/step - loss: 3.2104 - val_loss: 3.6684
Epoch 48/1000
16/16 [==============================] - 0s 2ms/step - loss: 3.0862 - val_loss: 3.5787
Epoch 49/1000
16/16 [==============================] - ETA: 0s - loss: 1.752 - 0s 2ms/step - loss: 3.0164 - val_los
s: 3.4757
Epoch 50/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.9203 - val_loss: 3.4317
Epoch 51/1000
16/16 [==============================] - 0s 2ms/step - loss: 3.0320 - val_loss: 3.4689
Epoch 52/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.7935 - val_loss: 3.2752
Epoch 53/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.7513 - val_loss: 3.4164
Epoch 54/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.6770 - val_loss: 3.3035
Epoch 55/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.5766 - val_loss: 3.2466
Epoch 56/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.5639 - val_loss: 3.3499
Epoch 57/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.3649 - val_loss: 3.0051
Epoch 58/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.2593 - val_loss: 3.4642
Epoch 59/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.4544 - val_loss: 3.2325
Epoch 60/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.2106 - val_loss: 3.0975
Epoch 61/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.1582 - val_loss: 2.9797
Epoch 62/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.1414 - val_loss: 2.7830
Epoch 63/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.1905 - val_loss: 3.8061
Epoch 64/1000
16/16 [==============================] - 0s 2ms/step - loss: 2.3123 - val_loss: 3.1691
Epoch 65/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.9668 - val_loss: 2.5556
Epoch 66/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.7237 - val_loss: 2.5010
Epoch 67/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.7079 - val_loss: 2.7022
Epoch 68/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.8957 - val_loss: 2.5126
Epoch 69/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.8670 - val_loss: 2.5414
Epoch 70/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.6417 - val_loss: 2.4126
Epoch 71/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.5882 - val_loss: 2.3989
Epoch 72/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.4380 - val_loss: 2.0954
Epoch 73/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.2311 - val_loss: 2.2355
Epoch 74/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.1962 - val_loss: 1.9829
Epoch 75/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.1219 - val_loss: 2.0183
Epoch 76/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.0019 - val_loss: 1.8860
Epoch 77/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.9830 - val_loss: 1.7563
Epoch 78/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.9107 - val_loss: 2.3790
Epoch 79/1000
16/16 [==============================] - 0s 2ms/step - loss: 1.0232 - val_loss: 1.5938
Epoch 80/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.8119 - val_loss: 1.4614
Epoch 81/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.7030 - val_loss: 1.5426
Epoch 82/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.7883 - val_loss: 1.6369
Epoch 83/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.7123 - val_loss: 1.5083
Epoch 84/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.6877 - val_loss: 1.3337
Epoch 85/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.6299 - val_loss: 1.3608
Epoch 86/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.5525 - val_loss: 1.1083
Epoch 87/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.4775 - val_loss: 1.0285
Epoch 88/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.4445 - val_loss: 1.0157
Epoch 89/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.4305 - val_loss: 0.9644
Epoch 90/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3969 - val_loss: 0.9314
Epoch 91/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3768 - val_loss: 0.8610
Epoch 92/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3506 - val_loss: 0.8625
Epoch 93/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3450 - val_loss: 0.8950
Epoch 94/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.3313 - val_loss: 0.8003
Epoch 95/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2967 - val_loss: 0.8246
Epoch 96/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2823 - val_loss: 0.7612
Epoch 97/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2723 - val_loss: 0.7309
Epoch 98/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2665 - val_loss: 0.7157
Epoch 99/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2620 - val_loss: 0.6482
Epoch 100/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2504 - val_loss: 0.7040
Epoch 101/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2573 - val_loss: 0.6372
Epoch 102/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2572 - val_loss: 0.6856
Epoch 103/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2301 - val_loss: 0.6433
Epoch 104/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2233 - val_loss: 0.6534
Epoch 105/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1974 - val_loss: 0.6036
Epoch 106/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1832 - val_loss: 0.6474
Epoch 107/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1815 - val_loss: 0.5545
Epoch 108/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1812 - val_loss: 0.5798
Epoch 109/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1837 - val_loss: 0.6729
Epoch 110/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1784 - val_loss: 0.5668
Epoch 111/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1533 - val_loss: 0.5680
Epoch 112/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1482 - val_loss: 0.5305
Epoch 113/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1654 - val_loss: 0.5589
Epoch 114/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1782 - val_loss: 0.5046
Epoch 115/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1684 - val_loss: 0.5452
Epoch 116/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1734 - val_loss: 0.6055
Epoch 117/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1625 - val_loss: 0.4919
Epoch 118/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1517 - val_loss: 0.6197
Epoch 119/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1842 - val_loss: 0.6176
Epoch 120/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.2609 - val_loss: 0.5866
Epoch 121/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1672 - val_loss: 0.5615
Epoch 122/1000
16/16 [==============================] - 0s 2ms/step - loss: 0.1937 - val_loss: 0.5411
```

In [48]:
```
plot_NN(model_ft,history,Smaller_Training,Smaller_Training_Target)
```



In [49]:
```
X_test = test_set.drop("Heating Load", axis=1)
y_test = test_set["Heating Load"].values
#
X_test_Std=scaler.transform(X_test)
```

In [50]:
```
pred=model_ft.predict(X_test_Std)
mse = mean_squared_error(y_test, pred)
rmse= np.sqrt(mse)
rmse
```

Out[50]:
```
0.5701598221707747
```

In [ ]: