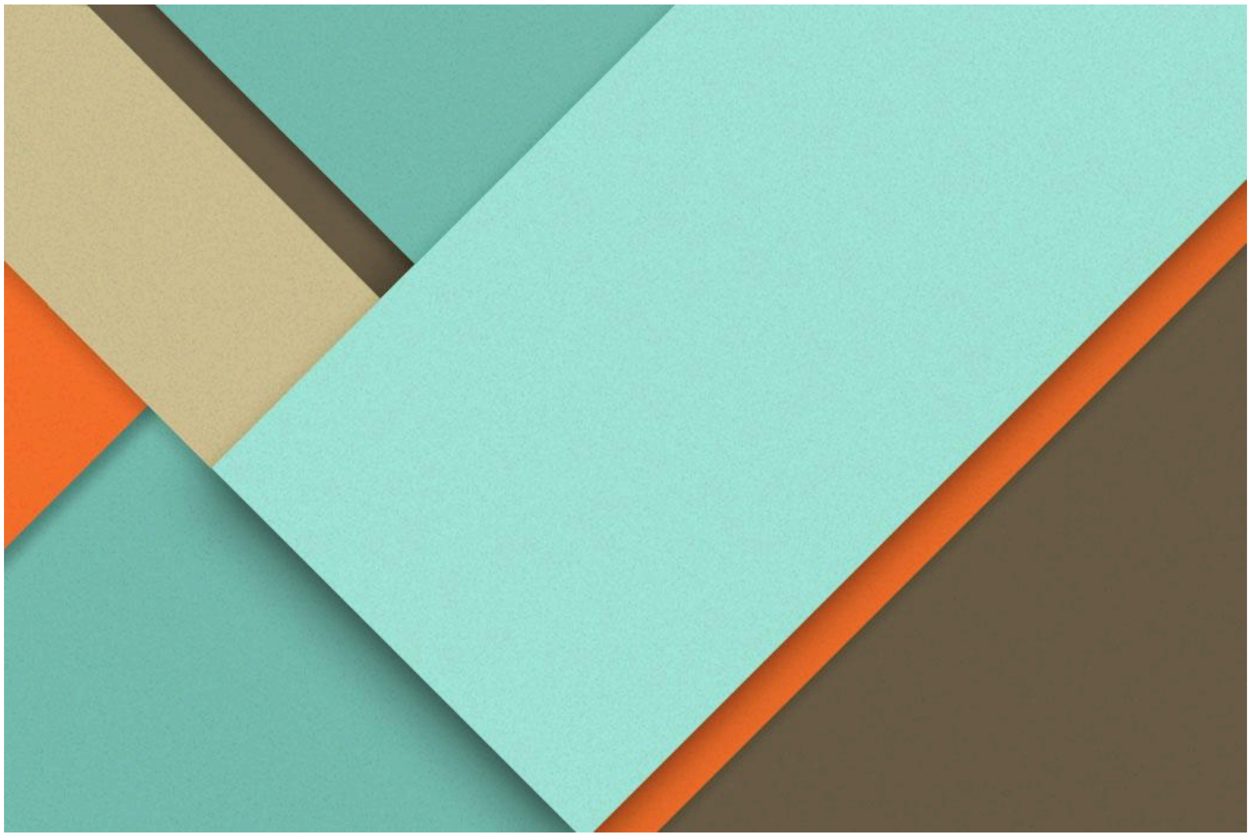


Assignment 1

Question 2

—

Abhinav Prakash
21CH10001
Course Code: CH620003

Problem Statement

Link to the data set (excel file):
https://www.dropbox.com/scl/fi/zlkxnyaptvc48xiu76xpy/data_file.xlsx?rlkey=754ozed754fyw4dhnlugjw7fa&dl=0

The data are tabulated into rows, one for each compound, and columns that include a common name, molecular weight, critical temperature (K), acentric factor, and the normal boiling point (K). Extract the data and place it into a matrix (array) **AllData** to use further on.

(a) Extract the boiling points from the data and place them in a column vector. Similarly, extract the molecular weights in a second vector. Plot the normal boiling point as a function of the molecular weight of the compounds in the database. Fit a straight line through the data and find the R^2 value.

(b) While we could use all the data we have to produce a meaningful correlation, let us assume that not all of the data were available. We will select a training set, or set of data that will be used to inform our ML algorithm. Extract from **AllData** 100 random data compounds. Using these selected compounds, build a 100 × 3 matrix, which we call **X**, where each row corresponds to a training example (i.e., the data for a given compound). For each one of these rows the first column is the number 1 (corresponding to x0) and the next two columns correspond to the values of x1 and x2 (molecular weight and acentric factor). Create another column matrix y with all the corresponding expected results; i.e., each of the elements of the vector is the expected reduced boiling point (Tb/Tc) of each training example. The resulting $\boldsymbol{y}$ vector is a 100 × 1 vector.

The "solution" of this problem can be found directly as

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\,\boldsymbol{y}$$

Calculate the values of the coefficients θ_0, θ_1 and θ_2, to evaluate the quality of the correlation.

(c) From the matrix **AllData** retrieve two column vectors, one called **InputData**, which will have two columns, the MW and the acentric factor of all the data in the original set and a second matrix called **TargetData** which contains the reduced boiling point data.

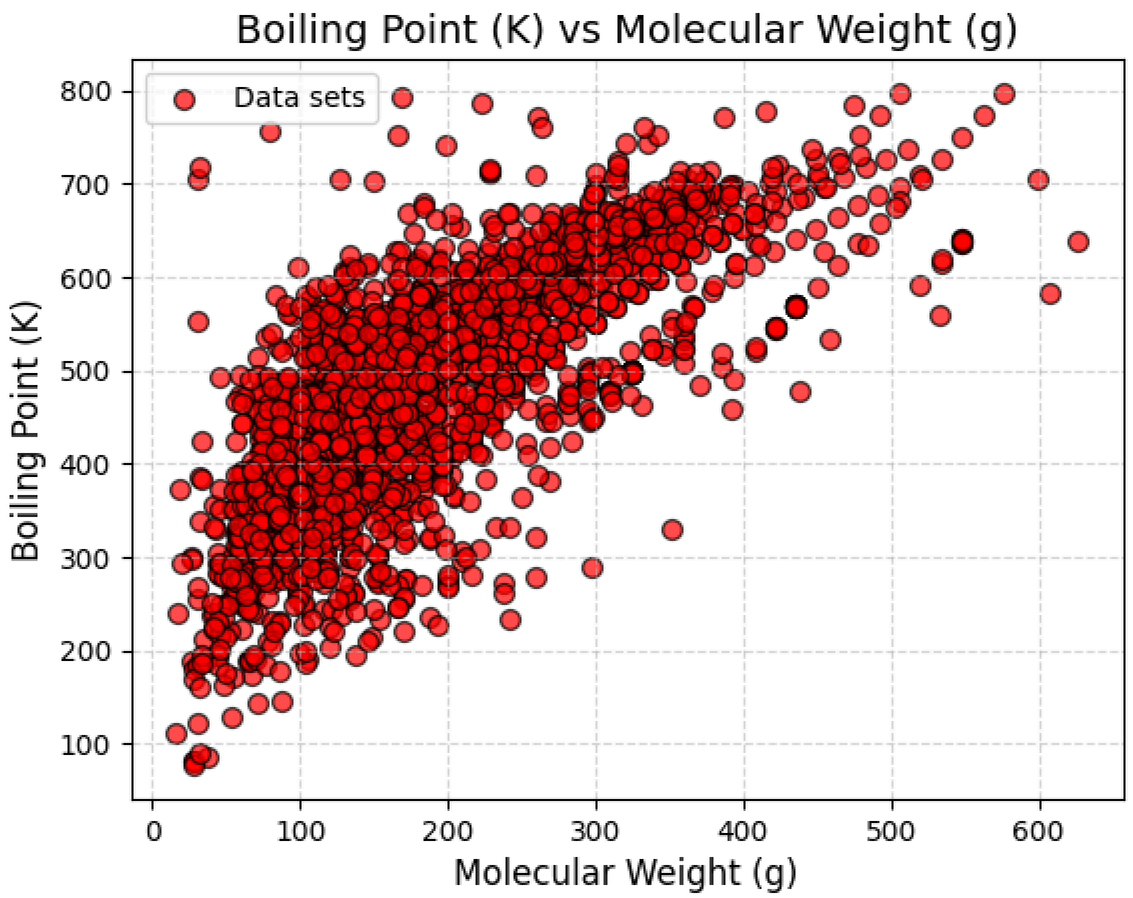
Develop a neural network for this data, and train the network, using only 10% of the data. Present the results of the output.

You may want to explore the effect of changing the number of layers. In the above case, only a 10% of the available data is used to train the network (some 600 data points) while the rest is used for validations and testing. Explore what happens when these ratios are changed.

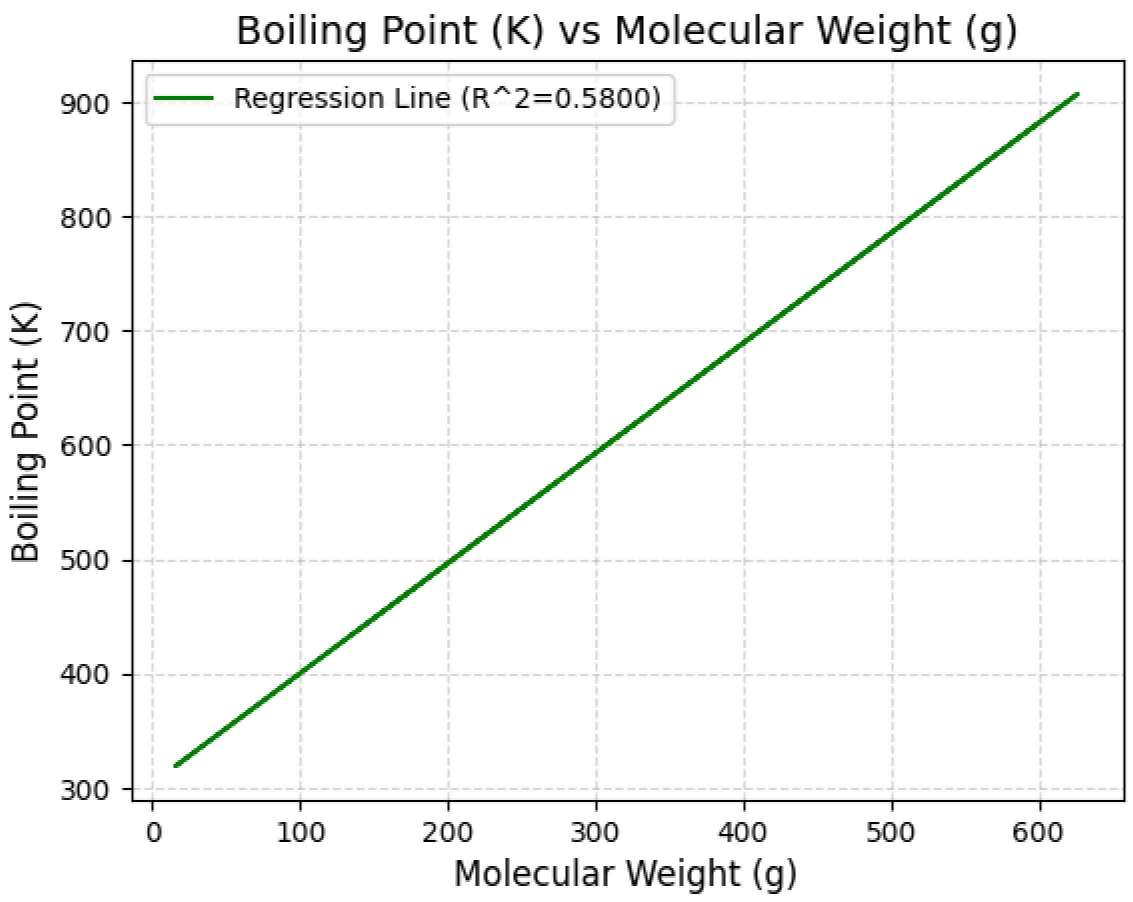
Google Colab Link

https://colab.research.google.com/drive/1INoHybCifEAblrG7wsHaoQBWVCPSXuTH?usp=sharing

Graph - BP vs Mw



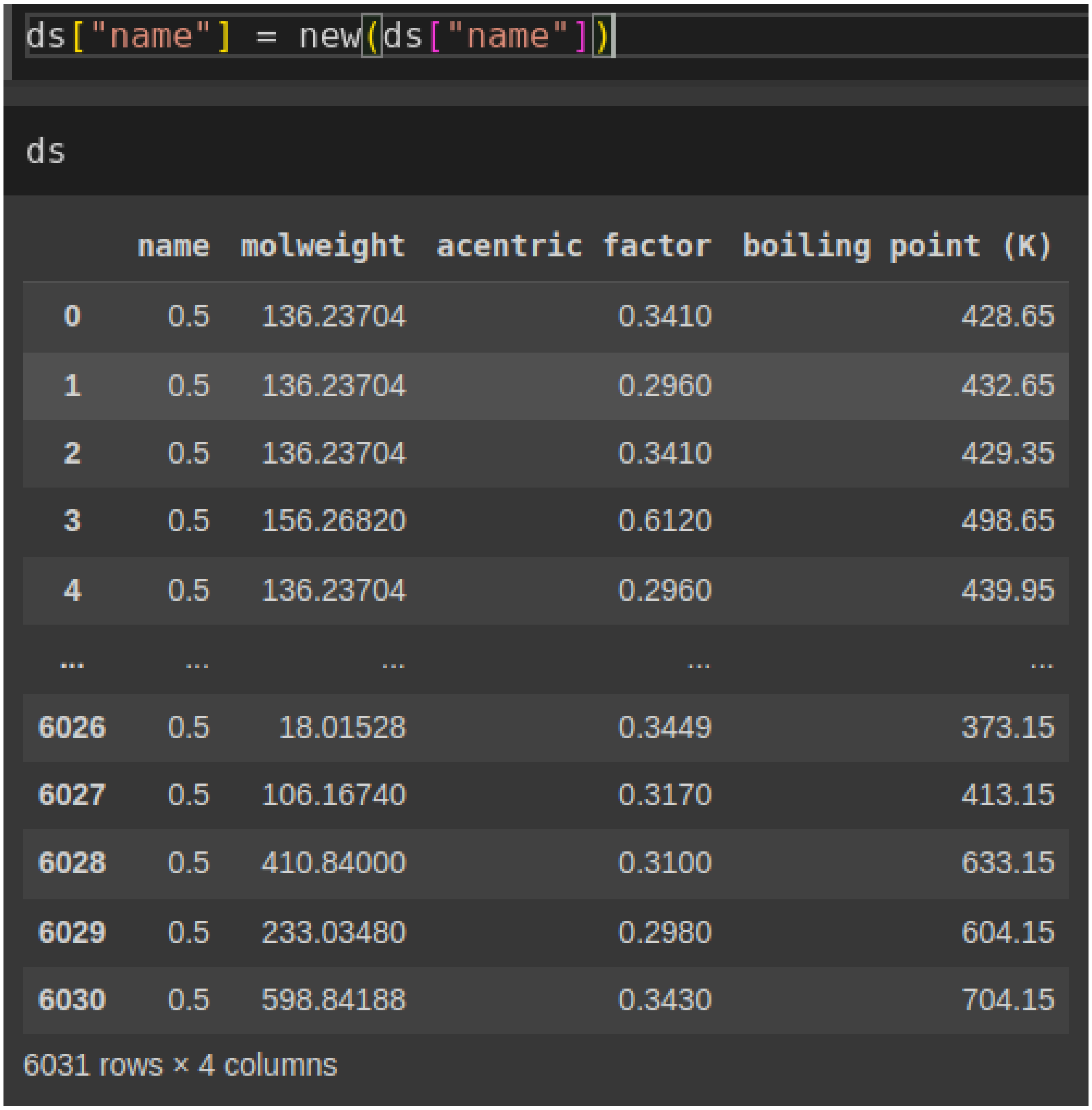
Fit a straight line to corresponding scatter plot



The R2 value for the plotted line is notably low, measuring **below 0.6**. Consequently, a linear regression may not be deemed suitable for this dataset.

Further Data Processing

In my approach to data preprocessing, my objective was to extract three key columns: compound name, molecular weight, and acentric factor. To address chiral compounds identified by '+' and '-', I introduced a binary representation (0 for '+', 1 for '-'). Following this, I removed duplicate entries based on both the compound name and the chiral flag, ensuring that the dataset retained only unique values for subsequent analysis. The resultant dataset now features the following columns: Name (with corresponding float values), Molecular Weight, and Acentric Factor.



```
ds["name"] = new(ds["name"])
```

```
ds
```

	name	molweight	acentric factor	boiling point (K)
0	0.5	136.23704	0.3410	428.65
1	0.5	136.23704	0.2960	432.65
2	0.5	136.23704	0.3410	429.35
3	0.5	156.26820	0.6120	498.65
4	0.5	136.23704	0.2960	439.95
...	...	...	...	...
6026	0.5	18.01528	0.3449	373.15
6027	0.5	106.16740	0.3170	413.15
6028	0.5	410.84000	0.3100	633.15
6029	0.5	233.03480	0.2980	604.15
6030	0.5	598.84188	0.3430	704.15

6031 rows × 4 columns

Theta Values:

Theta0: 496.4208942392646

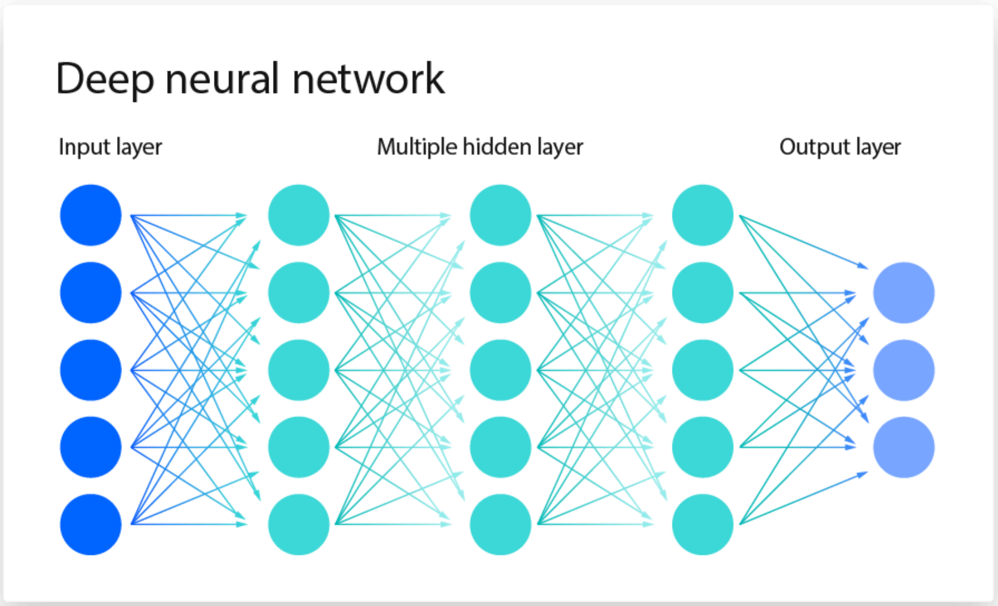
Theta1: 0.7933792601750705

Theta2: 161.5616492950219

For the linear regression model, the following equation will be used:
Boiling point = Theta0 x Name + Theta 1 x molweight + Theta2 x acentric factor

ANN Theory:

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of the human brain. Comprising interconnected nodes or "neurons," organized into layers, ANNs excel in learning complex patterns and relationships within data. The network undergoes a training process where it adjusts its internal parameters to minimize the difference between predicted and actual outcomes. ANNs find widespread applications in various fields, including image and speech recognition, natural language processing, and predictive analytics, leveraging their ability to capture intricate non-linear dependencies in data for efficient decision-making and pattern recognition.



Input layer

The input layer in an Artificial Neural Network (ANN) represents the initial stage where raw data or features are introduced to the network. Each node in this layer corresponds to a specific feature, and the layer serves as the entry point for information into the neural network.

Hidden layer

A hidden layer in an ANN resides between the input and output layers, and its nodes collectively contribute to the network's ability to capture complex relationships within the data. These hidden layers enable the network to learn hierarchical representations, transforming input features into higher-level abstractions through weighted connections and activation functions

Output layer

The output layer is the final layer in an ANN responsible for producing the network's output or predictions. The nodes in this layer generate the model's output based on the patterns and relationships learned during the training process. The number of nodes in the output layer depends on the nature of the task, such as binary classification, multi-class classification, or regression, with each node corresponding to a specific class or predicted value.

Activation Function

The **activation function** in an Artificial Neural Network (ANN) serves a crucial role in introducing non-linearity to the network, enabling it to learn complex relationships within data. It operates on the weighted sum of inputs and determines the output of a node or neuron. Common activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). Sigmoid and tanh functions squash the output between specific ranges, making them suitable for binary classification and normalization tasks. ReLU, on the other hand, introduces non-linearity by allowing positive values to pass through and setting negative values to zero, contributing to the network's ability to handle intricate patterns and improve training efficiency. The choice of activation function depends on the nature of the problem and the desired characteristics of the network's learned representations.

ANN ARCHITECTURE:

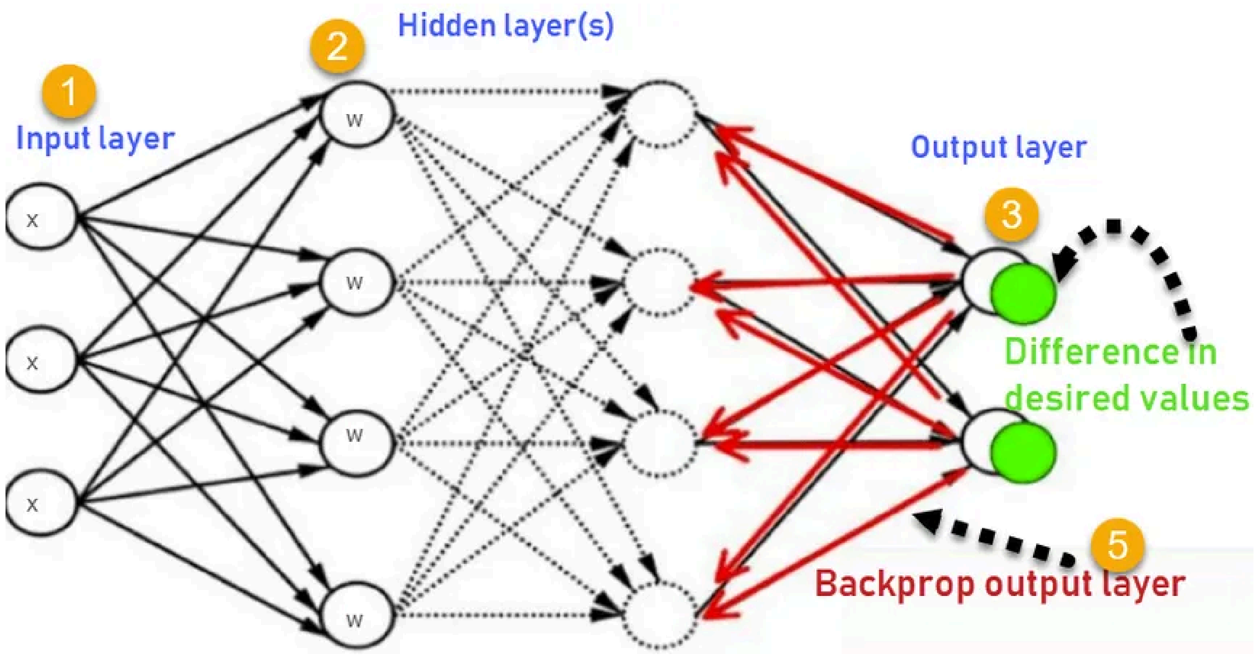
ANN Architecture:

The architecture of an Artificial Neural Network (ANN) defines its organizational structure, encompassing the arrangement and connectivity of its layers of neurons. A typical ANN architecture consists of three main components: the **input layer**, the **hidden layers**, and the **output layer**. The **input layer** receives the initial data, and each neuron in this layer represents a feature of the input. The **hidden layers**, positioned between the input and output layers, process and transform the input through weighted connections, applying activation functions to introduce non-linearity. Multiple hidden layers allow the network to learn hierarchical representations of data. The **output layer** produces the final result of the network's computation, providing the desired output based on the learned features. The number of neurons in each layer and the connectivity between them define the network's capacity to learn and generalize from the given data. The design and size of the architecture are crucial considerations, impacting the network's ability to comprehend complex patterns and perform effectively on diverse tasks.

1. Assign Random weights to all the linkages to start the algorithm
2. Using the inputs and the (Input ->Hidden node) linkages find the activation rate of Hidden Nodes
3. Using the activation rate of Hidden nodes and linkages to Output, find the activation rate of Output Nodes
4. Find the error rate at the output node and recalibrate all the linkages between Hidden Nodes and Output Nodes
5. Using the Weights and error found at Output node, cascade down the error to Hidden Nodes
6. Recalibrate the weights between hidden node and the input nodes
7. Repeat the process till the convergence criterion is met
8. Using the final linkage weights score the activation rate of the output nodes

Backward propagation:

Backpropagation, short for "backward propagation of errors," is a critical process in training Artificial Neural Networks (ANNs). Imagine you have a teacher guiding you on a task. You make a guess, and the teacher corrects your mistakes. Backpropagation works similarly for ANNs. When the network makes a prediction, we compare it to the actual result. If there's a difference (an error), we go backward through the network, adjusting the connection weights to reduce future errors. It's like learning from your mistakes and getting better at the task with each attempt. Backpropagation helps ANNs improve and fine-tune their predictions, making them more accurate over time.



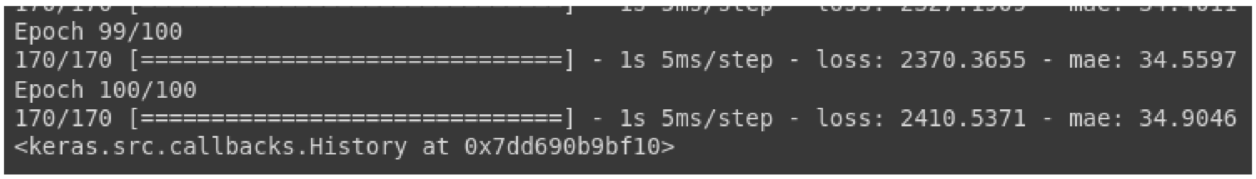
ANN Build <Code>:

<Code>

```
import tensorflow as tf


# Define the model
ann = tf.keras.Sequential([
   tf.keras.layers.Dense(128, activation="relu",
input_shape=(2,)),
   tf.keras.layers.Dropout(0.2),
   tf.keras.layers.BatchNormalization(),
   tf.keras.layers.Dense(64, activation="relu"),
   tf.keras.layers.Dropout(0.1),
   tf.keras.layers.BatchNormalization(),
   tf.keras.layers.Dense(units=1)
])

# model configure
optimizer = tf.keras.optimizers.Nadam(learning_rate=0.001)
ann.compile(loss="mse", optimizer=optimizer, metrics=["mae"])

# Train model
# Replace "features" and "labels" with your actual data
ann.fit(X_train, y_train, epochs=100, batch_size=32)
```

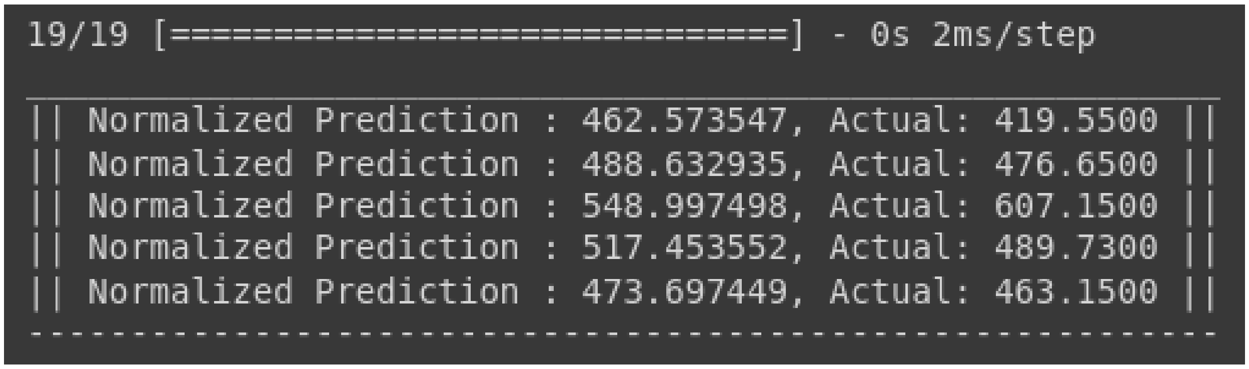
This piece of code does **two main things**: it gets a computer ready to learn from data and then teaches it. First, it takes information about molecular weight and acentric factor to predict boiling points. The data is split into parts for training (90%) and testing (10%). The computer is taught to understand this data through a neural network, which is like a virtual brain. This network has layers that help in learning patterns. The goal is to make accurate predictions about boiling points. The computer goes through this learning process 100 times, making adjustments each time to get better at making predictions. Overall, it's about making a smart system that can predict boiling points based on given information.

<Output>



```
Epoch 99/100
170/170 [==============================] - 1s 5ms/step - loss: 2370.3655 - mae: 34.5597
Epoch 100/100
170/170 [==============================] - 1s 5ms/step - loss: 2410.5371 - mae: 34.9046
<keras.src.callbacks.History at 0x7dd690b9bf10>
```

The Mean Absolute Error (MAE) of the improved neural network model has dropped to 34.9081, thanks to using dropout and batch normalization techniques. In the original model without these adjustments, the MAE was very high, around 104. It's important to mention that artificial neural networks (ANNs) might not work as well with smaller datasets, like the one here with only 6000 data points. For smaller datasets, other models might give better results, as ANNs could have trouble fitting the data too closely or making general predictions.



```
19/19 [==============================] - 0s 2ms/step
______________________________________________________________
|| Normalized Prediction : 462.573547, Actual: 419.5500 ||
|| Normalized Prediction : 488.632935, Actual: 476.6500 ||
|| Normalized Prediction : 548.997498, Actual: 607.1500 ||
|| Normalized Prediction : 517.453552, Actual: 489.7300 ||
|| Normalized Prediction : 473.697449, Actual: 463.1500 ||
--------------------------------------------------------------
```

It's evident that our model is not capturing the complexity of the data well, indicating underfitting. To improve results, more data will be necessary.