**Deep Learning :**

Total Marks: 60

Each question 20 marks

**Question: 1**

(a) Explain how you can implement DL in a real-world application.10 Real World Applications of Deep Leaning

Here are ten ways deep learning is already being used in diverse industries.

1. Computer vision

High-end gamers interact with deep learning modules on a very frequent basis. Deep neural networks power bleeding-edge object detection, image classification, image restoration, and image segmentation. So much so, they even power the recognition of hand-written digits on a computer system. To wit, deep learning is riding on an extraordinary neural network to empower machines to replicate the mechanism of the human visual agency.

2. Sentiment based news aggregation

Carolyn Gregorie writes in her Huffington Post piece: “the world isn’t falling apart, but it can sure feel like it.” And we couldn’t agree more. I am not naming names here, but you cannot scroll down any of your social media feed without stumbling across a couple of global disasters – with the exception of Instagram perhaps.

News aggregators are now using deep learning modules to filter out negative news and show you only the positive stuff happening around. This is especially helpful given how blatantly sensationalist a section of our media has been of late.

3. Bots based on deep learning

Take a moment to digest this – Nvidia researchers have developed an AI system that helps robots learn from human demonstrative actions. Housekeeping robots that perform actions based on artificial intelligence inputs from several sources are rather common. Like human brains process actions based on past experiences and sensory inputs, deep-learning infrastructures help robots execute tasks depending on varying AI opinions.

4. Automated translations

Automated translations did exist before the addition of deep learning. But deep learning is helping machines make enhanced translations with the guaranteed accuracy that was missing in the past. Plus, deep learning also helps in translation derived from images – something totally new that could not have been possible using traditional text-based interpretation.

5. Customer experience

Many businesses already make use of machine learning to work on customer experience. Viable examples include online self-service platforms. Plus, many organizations now depend on deep learning to create reliable workflows. Most of us are already familiar with the use of chatbots by organizations. As this application of deep leering matures, we can expect to see further enhancements in this field.

6. Autonomous vehicles

The next time you are lucky enough to witness an autonomous vehicle driving down, understand that there are several AI models working simultaneously. While some models pin-point pedestrians, others are adept at identifying street signs. A single car can be informed by millions of AI models while driving down the road. Many have considered AI-powered car drives safer than human riding.

7. Coloring illustrations

At one point, adding colors to black and white videos used to be one of the most time-consuming jobs in media production. But thanks to deep learning models and artificial intelligence, adding color to b/w photos and videos is now easier than ever. As you read, hundreds of black and white illustrations are being recreated in colored form.

8. Image analysis and caption generation

One of the greatest feats of deep learning is the ability to identify images and generate intelligent captions for them. In fact, image caption generation powered by AI is so accurate that many online publications are already making use of such techniques to save time and cost.

9. Text generation

Machines now have the power to generate new text from the scratch. They can learn the punctuation, grammar, and style of a piece of text and pen down effective news pieces. Robo-journalists riding on deep learning models have been producing accurate match reports for at least three years now. And the skill isn’t limited to match report writing exclusively.

AI-based text generation is fully equipped to handle the complexity of opinion pieces on issues concerning you and myself. As of now, text generation has helped create entries on just about everything from children’s rhymes to scholarly topics.

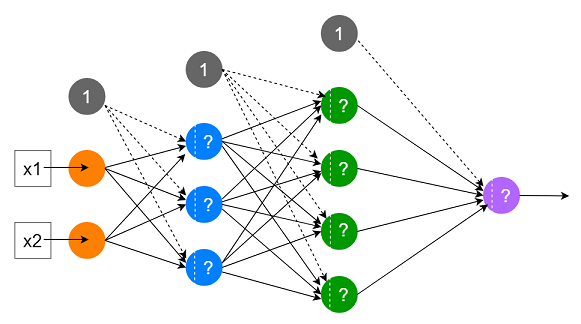
10. Language identification

At this point, we are looking at a preliminary stage where deep learning machines can differentiate between different dialects. For example, a machine will make the decision that someone is speaking in English. It will then make a distinction based on the dialect. Once the dialect has been established, further processing will be handled by another AI that specializes in the particular language. Not to mention, there is no human intervention in any of these steps.

**(b) What is the use of Activation function in Artificial Neural Networks? What would be the problem if we don't use it in ANN networks.**

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Consider the following neural network model with two hidden layers.



we represent the input data and parameter values in matrices. In the following mathematical expressions, the X represents the input data. The W1, W2 and W3 are weight matrices and the b1, b2 and b3 are bias vectors.

Let’s perform the calculations inside the above neural network model without using any activation function.

Hidden layer 1: Here, the input is X.

X\*W1 + b1

Hidden layer 2: Here, the input is X\*W1 + b1.

(X\*W1 + b1)\*W2 + b2

Output layer: Here, the input is (X\*W1 + b1)\*W2 + b2.

((X\*W1 + b1)\*W2 + b2)\*W3 + b3

The final output is ((X\*W1 + b1)\*W2 + b2)\*W3 + b3. We can simplify this expression further.

(X\*W1\*W2+ b1\*W2 + b2)\*W3 + b3

(X\*W1\*W2\*W3 + b1\*W2\*W3 + b2\*W3) + b3

(X\*W1\*W2\*W3) + (b1\*W2\*W3 + b2\*W3 + b3)

Finally, this can be denoted as:

(X\*W) + B

Where, W = W1\*W2\*W3 and B = b1\*W2\*W3 + b2\*W3 + b3

Here is the answer to the above question.

If you do not use any activation function in a neural network, it would become a giant linear regression model as (X\*W1\*W2\*W3) + (b1\*W2\*W3 + b2\*W3 + b3) that can be simplified into (X\*W) + B.

Moreover, the hidden layers would be useless and the model will not learn any non-linear relationship in the data. The model will only learn linear relationships that hardly exist in real-world data.

Therefore, we do need to use a non-linear activation function in the hidden layer(s) and also a suitable activation function in the output layer to allow our neural network models to learn complex non-linear relationships in real-world data.

**Question: 2**

Train a Pure ANN with less than 10000 trainable parameters using the MNIST Dataset

Image classification using Deep Learning

For this tutorial, we are going to use MNIST Fashion Dataset. Fashion-MNIST is a dataset of Zalando’s article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

The class labels are:



Figure : Unique classes in MNIST Fashion dataset

Import the required python libraries for Image Classification

#install required libraries

import pandas as pd

import numpy as np

#data visualization packages

import matplotlib.pyplot as plt

#keras packages

import keras

from keras.models import Sequential

from keras.layers import Convolution2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense

from keras.wrappers.scikit\_learn import KerasClassifier

from keras.layers import Dropout

#model evaluation packages

from sklearn.metrics import f1\_score, roc\_auc\_score, log\_loss

from sklearn.model\_selection import cross\_val\_score, cross\_validate

Load an MNIST Fashion dataset from the Keras.dataset library

#read mnist fashion dataset

mnist = keras.datasets.fashion\_mnist

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

print(X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape)

(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)

**Question: 3**

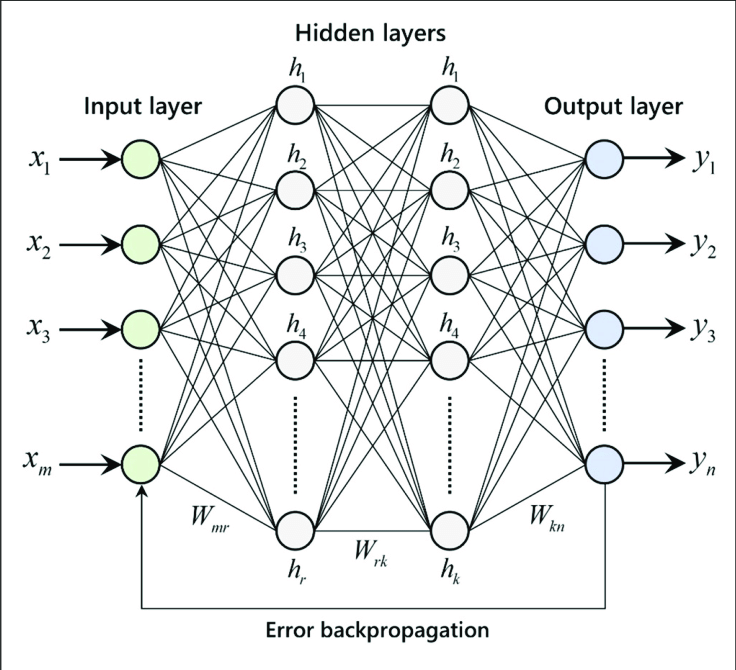
Perform Regression Task using ANN

The purpose of using Artificial Neural Networks for Regression over Linear Regression is that the linear regression can only learn the linear relationship between the features and target and therefore cannot learn the complex non-linear relationship. In order to learn the complex non-linear relationship between the features and target, we are in need of other techniques. One of those techniques is to use Artificial Neural Networks. Artificial Neural Networks have the ability to learn the complex relationship between the features and target due to the presence of activation function in each layer. Let’s look at what are Artificial Neural Networks and how do they work.

Artificial Neural Networks

Artificial Neural Networks are one of the deep learning algorithms that simulate the workings of neurons in the human brain. There are many types of Artificial Neural Networks, Vanilla Neural Networks, Recurrent Neural Networks, and Convolutional Neural Networks. The Vanilla Neural Networks have the ability to handle structured data only, whereas the Recurrent Neural Networks and Convolutional Neural Networks have the ability to handle unstructured data very well. In this post, we are going to use Vanilla Neural Networks to perform the Regression Analysis.

Structure of Artificial Neural Networks



The Artificial Neural Networks consists of the Input layer, Hidden layers, Output layer. The hidden layer can be more than one in number. Each layer consists of n number of neurons. Each layer will be having an Activation Function associated with each of the neurons. The activation function is the function that is responsible for introducing non-linearity in the relationship. In our case, the output layer must contain a linear activation function. Each layer can also have regularizers associated with it. Regularizers are responsible for preventing overfitting.

Artificial Neural Networks consists of two phases,

Forward Propagation

Backward Propagation

Forward propagation is the process of multiplying weights with each feature and adding them. The bias is also added to the result. Backward propagation is the process of updating the weights in the model. Backward propagation requires an optimization function and a loss function.

Regression Analysis Using Tensorflow

The entire code was executed in Google Colab. The data we use is the California housing prices dataset, in which we are going to predict the median housing prices. The data is available in the Colab in the path /content/sample\_data/california\_housing\_train.csv. We are going to use TensorFlow to train the model.

Import the required libraries.

import math

import pandas as pd

import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow.keras import Model

from tensorflow.keras import Sequential

from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.layers import Dense, Dropout

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.losses import MeanSquaredLogarithmicError

TRAIN\_DATA\_PATH = '/content/sample\_data/california\_housing\_train.csv'

TEST\_DATA\_PATH = '/content/sample\_data/california\_housing\_test.csv'

TARGET\_NAME = 'median\_house\_value'

The training data is in the path /content/sample\_data/california\_housing\_train.csv and the test data is in the path /content/sample\_data/california\_housing\_test.csv in Google Colab. The target name in the data is median\_house\_value.

# x\_train = features, y\_train = target

train\_data = pd.read\_csv(TRAIN\_DATA\_PATH)

test\_data = pd.read\_csv(TEST\_DATA\_PATH)

x\_train, y\_train = train\_data.drop(TARGET\_NAME, axis=1), train\_data[TARGET\_NAME]

x\_test, y\_test = test\_data.drop(TARGET\_NAME, axis=1), test\_data[TARGET\_NAME]

Read the train and test data and split the data into features and targets. The x\_train would look like the following image.

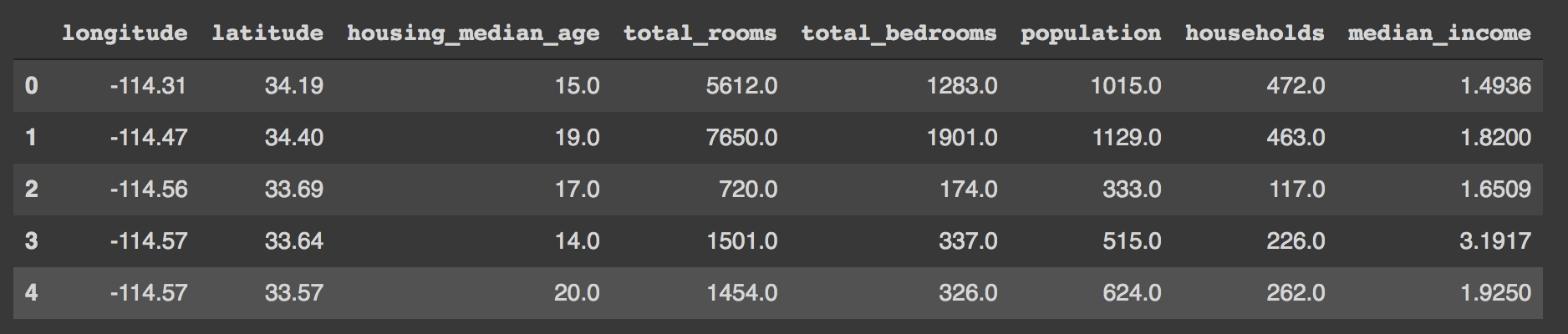


Image Source: Author’s Google Colab

def scale\_datasets(x\_train, x\_test):

"""

Standard Scale test and train data

Z - Score normalization

"""

standard\_scaler = StandardScaler()

x\_train\_scaled = pd.DataFrame(

standard\_scaler.fit\_transform(x\_train),

columns=x\_train.columns

)

x\_test\_scaled = pd.DataFrame(

standard\_scaler.transform(x\_test),

columns = x\_test.columns

)

return x\_train\_scaled, x\_test\_scaled

x\_train\_scaled, x\_test\_scaled = scale\_datasets(x\_train, x\_test)

The function scale\_datasets is used to scale the data. Scaling the data would result in faster convergence to the global optimal value for loss function optimization functions. Here we use Sklearn’s StandardScaler class which performs z-score normalization. The z-score normalization subtracts each data from its mean and divides it by the standard deviation of the data. We are going to train with the scaled dataset.

hidden\_units1 = 160

hidden\_units2 = 480

hidden\_units3 = 256

learning\_rate = 0.01

# Creating model using the Sequential in tensorflow

def build\_model\_using\_sequential():

model = Sequential([

Dense(hidden\_units1, kernel\_initializer='normal', activation='relu'),

Dropout(0.2),

Dense(hidden\_units2, kernel\_initializer='normal', activation='relu'),

Dropout(0.2),

Dense(hidden\_units3, kernel\_initializer='normal', activation='relu'),

Dense(1, kernel\_initializer='normal', activation='linear')

])

return model

# build the model

model = build\_model\_using\_sequential()

We are going to build the Neural Network model using TensorFlow’s Sequential class. Here we have used four layers. The first layer consists of 160 hidden units/ neurons with the ReLU activation function. ReLU stands for Rectified Linear Units. The second layer consists of 480 hidden units with the ReLU activation function. The third layer consists of 256 hidden units with the ReLU activation function. The final layer is the output layer which consists of one unit with a linear activation function.

# loss function

msle = MeanSquaredLogarithmicError()

model.compile(

loss=msle,

optimizer=Adam(learning\_rate=learning\_rate),

metrics=[msle]

)

# train the model

history = model.fit(

x\_train\_scaled.values,

y\_train.values,

epochs=10,

batch\_size=64,

validation\_split=0.2

)

After building the model with the Sequential class we need to compile the model with training configurations. We use Mean Squared Logarithmic Loss as loss function and metric, and Adam loss function optimizer. The loss function is used to optimize the model whereas the metric is used for our reference. After compiling the model we need to train the model. For training, we use the function fit which takes in the following parameters, features, target, epochs, batch size, and validation split. We have chosen 10 epochs to run. An epoch is a single pass through the training data. A batch size of 64 is used.

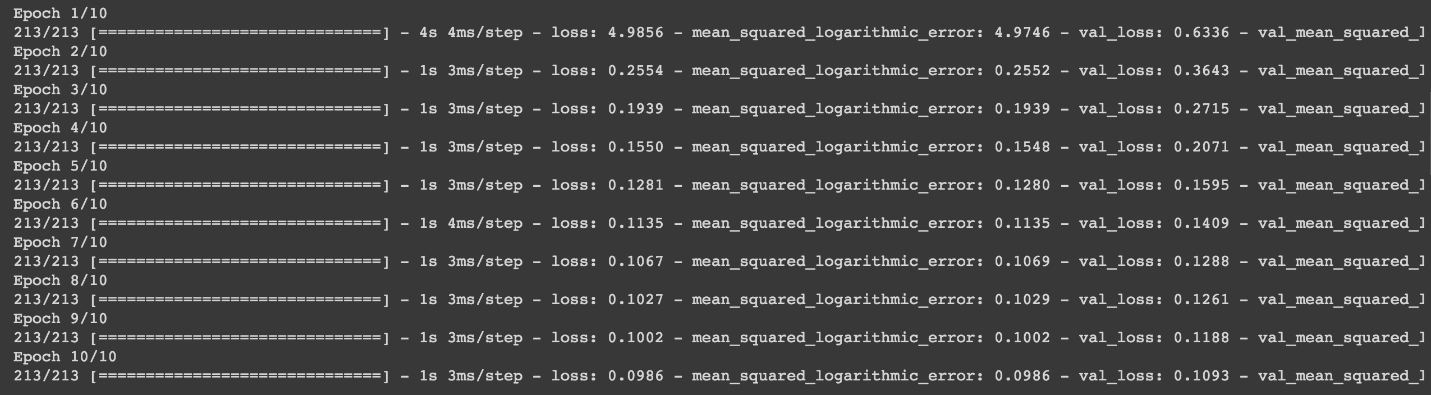


Image Source: Author’s Google Colab

After training, plot the history.

def plot\_history(history, key):

plt.plot(history.history[key])

plt.plot(history.history['val\_'+key])

plt.xlabel("Epochs")

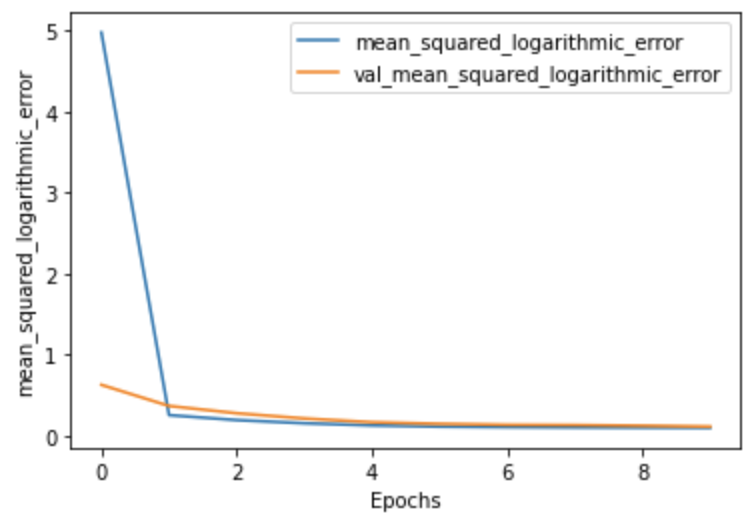
plt.ylabel(key)

plt.legend([key, 'val\_'+key])

plt.show()

# Plot the history

plot\_history(history, 'mean\_squared\_logarithmic\_error')



The image above shows that as the epochs increase the loss decreases, which is a good sign of progress. Now that we have trained the model, we can use that model to predict the unseen data, in our case, the test data.

x\_test['prediction'] = model.predict(x\_test\_scaled)

