# Deep learning Lab programs

**Note:**

**1-4 programs should be completed using numpy and matplot lib**

**5-16 programs should be implemented using TensorFlow and keras**

# LAB-1:

Prelab:

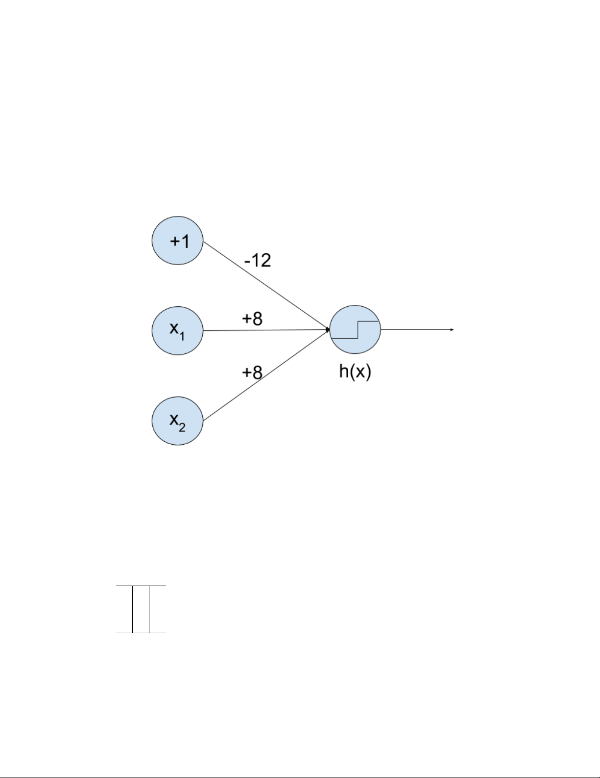
1. What is covex function?
2. What is threshold logic?

Inlab:

1a)Implement the basic logic gates AND & OR using McCullough Pit s model

1b) implement this perceptron which takes two binary valued inputs x1, x2∈ {0,1} and the activation function is the threshold function (h(x) = 1 if x > 0;

=0 if x <=0;



import numpy as np

class Perceptron:

def \_\_init\_\_(self,inodes,bias=0.2,lr=0.1):

self.w=np.random.rand(inodes)

self.bias=bias

self.learning\_rate=lr

def get\_weight(self):

return self.w

def set\_weights(self,w):

self.w=w

def train(self,X,y,epochs):

Y\_predict=np.zeros(len(y))

for t in range(epochs):

for i,x in enumerate(X):

if(np.dot(X[i],self.w))+self.bias <=0:

Y\_predict[i]=0

else:

Y\_predict[i]=1

self.bias=self.bias+self.learning\_rate\*(y[i]-Y\_predict[i])

self.w=self.w+self.learning\_rate\*X[i]\*(y[i]-Y\_predict[i])

print("error",y[i]-Y\_predict[i],"epoch:",t,"bias:",self.bias)

return self.w

def predict(self,x\_new):

if ((np.dot(x\_new,self.w))+self.bias) <=0:

return 0

else:

return 1

if \_\_name\_\_=="\_\_main\_\_":

X=np.array([[0,0],[0,1],[1,0],[1,1]])

y=np.array([0,0,1,1])

mlpcp=Perceptron(2,-0.2,0.1)

print(mlpcp.get\_weight())

trained\_weights=mlpcp.train(X,y,30)

print("Trained",mlpcp.get\_weight())

print(mlpcp.predict(x\_new=[0,1]))

Post lab

1) Solve the given polynomial equation using Tensorflow X2 -4X+4=0

**import** numpy **as** np  
import tensorflow **as** tf  
x=tf.Variable(initial\_value=0,name=**'x'**,trainable=**True**,dtype=tf.float32)  
coefficients =tf.constant(value=[1,-4,4],shape=(3,1),dtype=tf.float32)  
def compute\_loss():  
 **return** coefficients[0][0]\*x\*\*2+coefficients[1][0]\*x+coefficients[2][0]  
  
optimizer= tf.keras.optimizers.SGD(learning\_rate=0.1)  
  
for i **in** range(100):  
 optimizer.minimize(loss=compute\_loss, var\_list=[x])  
  
print(x.numpy())

-------------------------------------------------------------------------------------

# LAB-2

Pre lab:

1. What is Vanishing Gradient problem?
2. The nodes in the i/p layer is 10 and that in the hidden layer is 5. The max connections from the i/p layer to the hidden layer are?

Inlab:

Analyse the forward pass and backward pass of back propagation algorithm for the network using your own initiate, forward, backward and loss functions . Only use numpy , matplot libraries only.

import numpy as np

class MLP:

def \_\_init\_\_(self, input\_nodes, n\_hidden\_nodes, n\_y\_nodes):

self.w1 = np.random.rand(n\_hidden\_nodes, input\_nodes)

self.w2 = np.random.rand(n\_y\_nodes, n\_hidden\_nodes)

self.b1 = np.random.rand(n\_hidden\_nodes)

self.b2 = np.random.rand(n\_y\_nodes)

def get\_weights(self):

return (self.w1,self.w2)

def sigmoid(self, z):

z = 1/(1+np.exp(-z))

return z

def forward\_prop(self, x):

h = np.dot(self.w1, x)

hout = self.sigmoid(h)

y = np.dot(self.w2, hout)

yout = self.sigmoid(y)

return h,hout,y,yout

def back\_prop(self, m, h, hout, y, yout, yp):

dz2 = yout-yp

dw2 = np.dot(dz2, hout.T)/m

dz1 = np.dot(self.w2.T, dz2) \* hout\*(1-hout)

dw1 = np.dot(dz1, x.T)/m

dw1 = np.reshape(dw1, self.w1.shape)

dw2 = np.reshape(dw2, self.w2.shape)

return dz2, dw2, dz1, dw1

def train(self, x, y, iterations, lr):

losses = []

for i in range(iterations):

h, hout, yp, yout = self.forward\_prop(x)

print("Epoch",i," : ",yout)

loss = -(1/len(x))\*np.sum(y\*np.log(yout)+(1-y)\*np.log(1-yout))

losses.append(loss)

m = 1/len(x)

da2, dw2, dz1, dw1 = self.back\_prop(m,h, hout, y, yout, y)

self.w2 = self.w2-lr\*dw2

self.w1 = self.w1-lr\*dw1

def predict(self, input):

h, hout, y, yout = self.forward\_prop(input)

yout = np.squeeze(yout) # to remove single dimensional entries

print("output : ", yout)

x = np.array([[0, 0, 1, 1], [0, 1, 0, 1],[1,0,0,1]])

y = np.array([[0, 1, 1, 0]])

n\_x = 3

n\_y = 1

n\_h = 3

model = MLP(n\_x, n\_h, n\_y)

print("wt in network: before", model.get\_weights())

model.train(x, y, 5, 0.2)

print("wt in network: after", model.get\_weights())

model.predict(np.array([[0, 1,1], [1,0, 1],[0,0,1]]))

Postlab:

1. Design model to predict credit card risk data using multi-layer perceptron. Display weights in each layer.

// without implementation of dataset

from keras.models import Sequential

from keras.layers import Dense

# Define the model architecture

model = Sequential()

model.add(Dense(units=64, activation='relu', input\_dim=X\_train.shape[1]))

model.add(Dense(units=32, activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

# Compile the model with appropriate optimizer and loss function

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model on the training data

model.fit(X\_train, y\_train, epochs=10, batch\_size=32)

# Evaluate the model on the test data

loss, accuracy = model.evaluate(X\_test, y\_test)

# Display the model weights in each layer

for layer in model.layers:

weights = layer.get\_weights()

print(weights)

----------------------------------------------------------------------------------------------------------------

# LAB-3

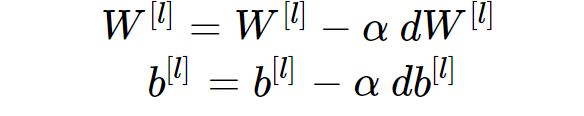
Pre lab:

1.Which of the SGD variants are based on both momentum and adaptive learning?

2. what is global minima?

Inlab:

1 Implement the gradient descent update rule. The gradient descent rule is,

for l=1,..., L layers. Assume number of nodes in each layer are 2 and number of nodes in input layer are 4.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import mean\_squared\_error

data=pd.read\_csv('iris.csv')

data.info()

from sklearn.preprocessing import StandardScaler

scale = data.iloc[:,[0,4]]

sc = StandardScaler()

scale = sc.fit\_transform(scale)

scale = pd.DataFrame(scale)

scale.columns = ['SepalLengthCm','PetalWidthCm']

cur\_x = 3 # The algorithm starts at x=3

rate = 0.01 # Learning rate

precision = 0.000001 #This tells us when to stop the algorithm

previous\_step\_size = 1 #

max\_iters = 10000 # maximum number of iterations

iters = 0 #iteration counter

df = lambda x: 2\*(x+5) #Gradient of our function

while previous\_step\_size > precision and iters < max\_iters:

prev\_x = cur\_x #Store current x value in prev\_x

cur\_x = cur\_x - rate \* df(prev\_x) #Grad descent

previous\_step\_size = abs(cur\_x - prev\_x) #Change in x

iters = iters+1 #iteration count

print("Iteration",iters,"\nX value is",cur\_x) #Print iterations

print("The local minimum occurs at", cur\_x)

Postlab:

1) Use gradient descent optimiser function in TensorFlow and perform classification of iris data set.

from sklearn.preprocessing import LabelEncoder

# Load the iris dataset

df = pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)

# Preprocess the data

X = df.iloc[:, :4].values

y = df.iloc[:, 4].values

# Encode the labels

encoder = LabelEncoder()

y = encoder.fit\_transform(y)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Scale the data

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Convert labels to one-hot encoding

y\_train = tf.keras.utils.to\_categorical(y\_train, num\_classes=3)

y\_test = tf.keras.utils.to\_categorical(y\_test, num\_classes=3)

# Define the model

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(10, input\_shape=(4,), activation='relu'))

model.add(tf.keras.layers.Dense(3, activation='softmax'))

# Define the optimizer

optimizer = tf.keras.optimizers.SGD(learning\_rate=0.01)

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=0)

print('Test Accuracy:', test\_acc)

----------------------------------------------------------------------------------------------------------------

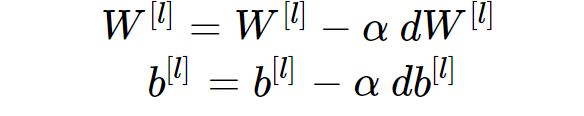
# LAB-4

Pre lab:

1. Write down the activation functions whose output is zero centered?
2. What is local minima?

Inlab:

Implement the **Stochastic Gradient Descent** update rule. The gradient descent rule is,

for l=1,..., L layers. Assume number of nodes in each layer are 2 and number of nodes in input layer are 4., use only numpy and matplotlib.\

**import** numpy **as** np  
# import math.pyplot as plt  
  
def cost\_fun(theta,x,y):  
 m=len(y)  
 prediction=x.dot(theta)  
 error=(1/2\*m)\*np.sum(np.square((prediction-y)))  
 **return** error  
  
def SGD(x,y,theta,lr,epochs):  
 m=len(y)  
 cal\_history=np.zeros([epochs])  
 **for** i **in** range(epochs):  
 cost=0  
 *# print("epoch",i)*  
**for** j **in** range(m):  
 rand\_x=np.random.randint(0,m)  
 x[j]=x[rand\_x,:]  
 y[j]=y[rand\_x].reshape(1,1)  
 predict\_y=np.dot(x[j],theta)  
 theta=theta-(1/m)\*(lr\*np.dot((predict\_y-y),x[j].T))  
 cost+=cost\_fun(theta,x,y)  
 *# print(cost)*  
 *cal\_history[i]=cost*  
  
**return** cal\_history,theta  
  
l=0.01  
theta=0.1  
x = np.array([[0, 0, 1, 1], [0, 1, 0, 1], [1, 0, 0, 1],[0,1,1,0]])  
y = np.array([0, 1, 1,0])  
cal\_history,theta=SGD(x,y,theta,l,10)  
print(cal\_history)  
print(theta)

**from** sklearn.datasets **import** load\_iris  
from sklearn.model\_selection **import** train\_test\_split  
from sklearn.preprocessing **import** StandardScaler  
from keras.models **import** Sequential  
from keras.layers **import** Dense  
from tensorflow.keras.utils **import** to\_categorical  
  
# Load the Iris dataset  
iris = load\_iris()  
X = iris.data  
y = iris.target  
  
# Split the dataset into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Normalize the data using the StandardScaler  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)  
  
# Convert the target variable to a one-hot encoded *format*  
*y\_train = to\_categorical(y\_train)*  
*y\_test = to\_categorical(y\_test)*  
  
*# Build an ANN model*  
*model = Sequential()*  
*model.add(Dense(*8, input\_dim=4, activation=**'relu'**))  
model.add(Dense(3, activation=**'softmax'**))  
model.compile(loss=**'categorical\_crossentropy'**, optimizer=**'adam'**, metrics=[**'accuracy'**])  
  
# Train the model  
model.fit(X\_train, y\_train, epochs=30, batch\_size=5)  
  
# Evaluate the performance of the model on the tes*ting set*  
*loss, accuracy = model.evaluate(X\_test, y\_test)*  
*print*(**'Accuracy: %.2f'** % (accuracy\*100))

Post lab:

Normalise the data in the dataset and perform classification using ANN.

from sklearn.datasets import load\_iris

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

# Load the iris dataset

iris = load\_iris()

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2)

# One-hot encode the target variable

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

# Define the model

model = Sequential()

model.add(Dense(10, input\_shape=(4,), activation='relu'))

model.add(Dense(3, activation='softmax'))

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model to the training data

model.fit(X\_train, y\_train, epochs=50, batch\_size=10)

# Evaluate the model on the testing data

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test loss: {loss}, Test accuracy: {accuracy}')

----------------------------------------------------------------------------------------------------------------

# LAB-5

Pre lab:

1. Which is the fastest gradient descent
2. What is saddle point

Inlab:

1) Perform Multi class classification on MNIST dataset using Deep NN with tensorflow (Note: Use softmax activation function for the output layer with 3-4 hidden layers consisting of ReLU activation function also evaluate the performance by using confusion matrix.)

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data()

# normalizing

x\_train = x\_train / 255.0 # pixel ranges from 0-255 to make it to range of 0-1

x\_test = x\_test / 255.0

x\_train = x\_train.reshape(-1, 28 \* 28)

x\_test = x\_test.reshape(-1, 28 \* 28)

# One-hot encode the labels

y\_train = keras.utils.to\_categorical(y\_train, 10) # to convert the categorical variables

y\_test = keras.utils.to\_categorical(y\_test, 10)

model = keras.Sequential([

layers.Dense(256, activation='relu', input\_shape=(28 \* 28,)),

layers.Dense(128, activation='relu'),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=10, batch\_size=32) # training the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test) # evaluating the model

print('Test accuracy:\n', test\_acc)

from sklearn.metrics import confusion\_matrix

import numpy as np

y\_pred\_classes = np.argmax(y\_pred, axis=1) # returns the index of max ele in array from each row as axis=1

y\_test\_classes = np.argmax(y\_test, axis=1)

con\_mat = confusion\_matrix(y\_test\_classes, y\_pred\_classes)

print(con\_mat)

Postlab:

1) Applications of image classification in real world.

----------------------------------------------------------------------------------------------------------------

# LAB-6

Pre lab

1. Which optimization algorithm processes all the training examples for each iteration of gradient descent.
2. CNN is faster to train than a DNN. Why?

Inlab:

Build and train a ConvNet in TensorFlow for a classification problem. Don’t use keras.

import numpy as np

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

# Load the MNIST dataset

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

# Preprocess the data

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

# Convert class vectors to binary class matrices

num\_classes = 10

y\_train = keras.utils.to\_categorical(y\_train, num\_classes) # one hot encoding

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

# Define the model

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),activation='relu',input\_shape=(28, 28, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5)) # deactivating 50% neurons

model.add(Dense(num\_classes, activation='softmax'))

# Compile the model

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

# Train the model

batch\_size = 128

epochs = 5

model.fit(x\_train, y\_train,batch\_size=batch\_size,epochs=epochs,verbose=1,validation\_data=(x\_test, y\_test))

# Evaluate the model on the test data

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

Postlab:

1. Use 3\*3 filter and perform convolution operation for the matrix given below.

----------------------------------------------------------------------------------------------------------------

LAB-7

Pre lab:

1. What is multiclass classification?
2. What is the function of pooling layer

InLab:

1)Build a deep learning model which classifies cats and dogs using CNN.

Post-lab

Read and display an image using opencv

# LAB-8

Pre lab:

What is Object Recognition?

How greedy approach is useful in designing selective search algorithm for R-CNN object detection

In lab:

1) Build The R-CNN model to extracts many regions from the input image and use a CNN to perform forward propagation on each region proposal to extract its features, then use these features to predict the class and bounding box of this region proposal. (multi class classification)

Post Lab:

Implement Fast R-CNN on the same problem and see how CNN forward propagation is performed on the entire image.

# LAB-9

Pre lab

## Which neural network has only one hidden layer between the input and output?

What is the use of forget gate?

Inlab:

Implement a simple RNN model based on following recurrence relation.

ht=σ(W([ht−1,xt])+b)

Post Lab:

Build RNN model by taking breast cancer dataset from UCI repository.

# LAB-10

Pre lab:

#### What is 'gradient' when you are talking about RNN?

1. What is backpropagation and how it is useful in LSTM

In lab:

1) Using LSTM train the model with train dataset , predict the weather for the dates given in the test dataset and visualize the actual ,predicte data using matplotlib. use 60 days time stamp.

# LAB-11:

Pre lab:

1.What are the applications of autoencoders?

2. what is the difference between auto encoders and PCA

Inlab:

1)Build a simple auto encoder on MNIST data using keras library and help them in building a simple Auto Encoder. (note: encode the image into from 784 to 32 pixels)

# Lab-12:

Pre lab:

# How deep learning models are built on Keras

1. Which deep learning model generate data randomly that looks very similar to training data?

Inlab:

Build a simple denoisy auto encoder using keras on MNIST.

Post lab:

Build a Autoencoders for Feature Extraction for the mnist data set. Display the feature set.