```
import numpy as np
import pandas as pd
```

Double-click (or enter) to edit

```
data=pd.read_csv('/content/Iris.csv')
data.columns=['Id','Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Species']
data.head(10)
```

	Id	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

```
def activation_func(value):    #Tangent Hypotenuse
    #return (1/(1+np.exp(-value)))
    return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
```

```
def perceptron_train(in_data,labels,alpha):
    X=np.array(in_data)
    y=np.array(labels)
    weights=np.random.random(X.shape[1])
    original=weights
    bias=np.random.random_sample()
    for key in range(X.shape[0]):
        a=activation_func(np.matmul(np.transpose(weights),X[key]))
        yn=0
        if a>=0.7:
            yn=1
        elif a<=(-0.7):
            yn=-1
        weights=weights+alpha*(yn-y[key])*X[key]
        print('Iteration '+str(key)+': '+str(weights))
    print('Difference: '+str(weights-original))
    return weights
```

```
def perceptron_test(in_data,label_shape,weights):
```

```
    X=np.array(in_data)
    y=np.zeros(label_shape)
    for key in range(X.shape[1]):
        a=activation_func((weights*X[key]).sum())
        y[key]=0
        if a>=0.7:
            y[key]=1
        elif a<=(-0.7):
            y[key]=-1
    return y
```

```
def score(result,labels):
    difference=result-np.array(labels)
    correct_ctr=0
    for elem in range(difference.shape[0]):
        if difference[elem]==0:
            correct_ctr+=1
    score=correct_ctr*100/difference.size
    print('Score='+str(score))
```

```
# Dividing DataFrame "data" into "d_train" (60%) and "d_test" (40%)
divider = np.random.rand(len(data)) < 0.70
d_train=data[divider]
d_test=data[~divider]
```

```
# Dividing d_train into data and labels/targets
d_train_y=d_train['Species']
d_train_X=d_train.drop(['Species'],axis=1)

# Dividing d_train into data and labels/targets
d_test_y=d_test['Species']
d_test_X=d_test.drop(['Species'],axis=1)
```

```
# Learning rate
alpha = 0.01

# Train
weights = perceptron_train(d_train_X, d_train_y, alpha)
```

```
# Test
result_test=perceptron_test(d_test_X,d_test_y.shape,weights)
```

```
# Calculate score
score(result_test,d_test_y)
```

```
Score=28.571428571428573
```

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
```

```
import matplotlib.pyplot as plt
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mni
11490434/11490434 [==============================] - 0s 0us/step
```

```
# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# normalize image pixel values by dividing
# by 255
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale

```

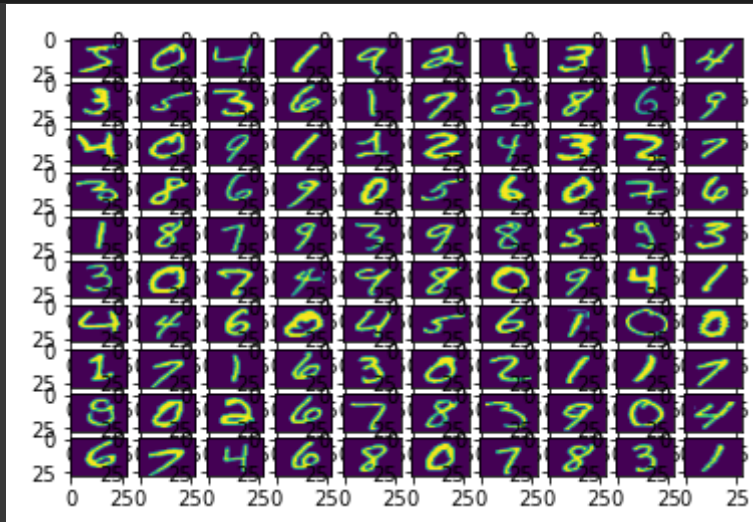
```
print("Feature matrix:", x_train.shape)
print("Target matrix:", x_test.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_test.shape)

```

```
Feature matrix: (60000, 28, 28)
Target matrix: (10000, 28, 28)
Feature matrix: (60000,)
Target matrix: (10000,)
```

```
fig, ax = plt.subplots(10, 10)
k = 0
for i in range(10):
    for j in range(10):
        ax[i][j].imshow(x_train[k].reshape(28, 28),
                        aspect='auto')
        k += 1
plt.show()

```



```
model = Sequential([

    # reshape 28 row * 28 column data to 28*28 rows
    Flatten(input_shape=(28, 28)),

    # dense layer 1
    Dense(256, activation='sigmoid'),

    # dense layer 2
    Dense(128, activation='sigmoid'),

    # output layer
    Dense(10, activation='sigmoid'),
])

```

```
model.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

```

```
model.fit(x_train, y_train, epochs=10,
        batch_size=2000,
        validation_split=0.2)

```

```
Epoch 1/10
24/24 [==============================] - 2s 57ms/step - loss: 2.1259 - accuracy: 0.3
Epoch 2/10
24/24 [==============================] - 1s 48ms/step - loss: 1.4742 - accuracy: 0.7
Epoch 3/10
24/24 [==============================] - 1s 50ms/step - loss: 0.9484 - accuracy: 0.8
Epoch 4/10
24/24 [==============================] - 1s 50ms/step - loss: 0.6579 - accuracy: 0.8
Epoch 5/10
24/24 [==============================] - 1s 48ms/step - loss: 0.5093 - accuracy: 0.8
Epoch 6/10
24/24 [==============================] - 1s 48ms/step - loss: 0.4255 - accuracy: 0.8
Epoch 7/10
24/24 [==============================] - 1s 48ms/step - loss: 0.3738 - accuracy: 0.9
Epoch 8/10
24/24 [==============================] - 1s 48ms/step - loss: 0.3379 - accuracy: 0.9
Epoch 9/10
24/24 [==============================] - 1s 48ms/step - loss: 0.3118 - accuracy: 0.9
Epoch 10/10
24/24 [==============================] - 1s 49ms/step - loss: 0.2907 - accuracy: 0.9
<keras.callbacks.History at 0x7fa0595f94d0>
```

```
results = model.evaluate(x_test, y_test, verbose = 0)
print('test loss, test acc:', results)
```

```
test loss, test acc: [0.2749628722667694, 0.9228000044822693]
```

```
1
```

```
1 import numpy as np
2 import pandas as pd
```

```
1 data=pd.read_csv('/content/Iris.csv')
2 data.columns=['Id','Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Species']
3 data.head(10)
```

	Id	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

```
1 from sklearn.neural_network import MLPClassifier
2 from sklearn.datasets import make_classification
3 from sklearn.model_selection import train_test_split
4 X, y = make_classification(n_samples=100, random_state=1)
5 X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,random_state=1)
6 clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
7 clf.predict_proba(X_test[:1])
8 clf.predict(X_test[:5, :])
9 clf.score(X_test, y_test)
```

```
0.88
```

```
1 clf
```

```
MLPClassifier(max_iter=300, random_state=1)
```

```
1 import tkinter as tk
2 from tkinter import *
3 import cv2
4 from PIL import Image, ImageTk
5 import os
6 import numpy as np
7
8
```

```
global last_frame1
last_frame1 = np.zeros((480, 640, 3), dtype=np.uint8)
global last_frame2
last_frame2 = np.zeros((480, 640, 3), dtype=np.uint8)
global cap1
global cap2
cap1 = cv2.VideoCapture(-1)
cap2 = cv2.VideoCapture(-1)

def show_vid():
    if not cap1.isOpened():
        print("cant open the camera1")
    flag1, frame1 = cap1.read()
    frame1 = cv2.resize(frame1,(100,100))
    if flag1 is None:
        print ("Major error!")
    elif flag1:
        global last_frame1
        last_frame1 = frame1.copy()
        pic = cv2.cvtColor(last_frame1, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(pic)
        imgtk = ImageTk.PhotoImage(image=img)
        lmain.imgtk = imgtk
        lmain.configure(image=imgtk)
        lmain.after(10, show_vid)


if __name__ == '__main__':
    root=tk.Tk()
    lmain = tk.Label(master=root)
    lmain2 = tk.Label(master=root)

    lmain.pack(side = LEFT)
    lmain2.pack(side = RIGHT)
    root.title("Lane-line detection")
    root.geometry("900x700+100+10")
    exitbutton = Button(root, text='Quit',fg="red",command=   root.destroy).pack(side = BOTTOM
    show_vid()
    root.mainloop()
    cap.release()
```

```
---------------------------------------------------------------------------
TclError                                  Traceback (most recent call last)
<ipython-input-1-2d2e8826dcdc> in <module>
     35
     36 if __name__ == '__main__':
---> 37     root=tk.Tk()
     38     lmain = tk.Label(master=root)
     39     lmain2 = tk.Label(master=root)

/usr/lib/python3.7/tkinter/__init__.py in __init__(self, screenName, baseName,
className, useTk, sync, use)
   2021                 baseName = baseName + ext
   2022         interactive = 0
-> 2023         self.tk = _tkinter.create(screenName, baseName, className,
interactive, wantobjects, useTk, sync, use)
   2024         if useTk:
   2025             self._loadtk()

TclError: no display name and no $DISPLAY environment variable
```

SEARCH STACK OVERFLOW

Colab paid products - Cancel contracts here