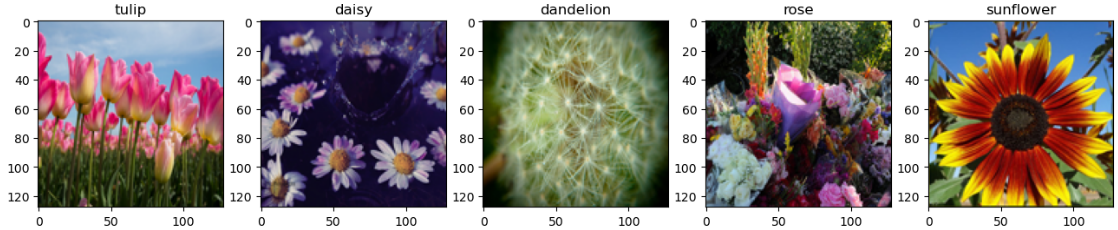
ANN_model

April 16, 2023

0.1 ANN Model (Flower Prediction)

Flower Recognition where we have to classify the flower species ,The Images are divided into five classes: chamomile, tulip, rose, sunflower, dandelion.

[6]:
```
from PIL import Image
import os
import matplotlib.pyplot as plt
import numpy as np
folder_path = 'flowers'
fig_size = (16, 8)
num_cols = 5
fig, axes = plt.subplots(nrows=1, ncols=num_cols, figsize=fig_size)

# access to the subfolder
for i, subfolder_name in enumerate(os.listdir(folder_path)):
    count=0
    # path
    subfolder_path = os.path.join(folder_path, subfolder_name)
    # get the image
    count=0
    for filename in os.listdir(subfolder_path):
        if filename.endswith('.jpg'):
            img_path = os.path.join(subfolder_path, filename)
        count+=1
        if(count==5): break
        # load and resize the image
        img = Image.open(img_path)
        img = img.resize((128, 128))
        img_array = np.array(img)

        # flower name
        ax = axes[i % num_cols]
        ax.imshow(img_array)
        ax.set_title(subfolder_name)

for j in range(i+1, num_cols):
    axes[j].axis('off')
plt.show()
```



0.1.1 Step 1: load the dataset

[7]:
```
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split

def load_datasets(img_size=(28, 28)):
    data_dir = 'flowers'
    flower_species = os.listdir(data_dir)
    images = []
    labels = []
    for species in flower_species:
        species_dir = os.path.join(data_dir, species)
        for img_file in os.listdir(species_dir):
            img_path = os.path.join(species_dir, img_file)
            img = cv2.imread(img_path)
            img = cv2.resize(img, img_size)
            images.append(img)
            labels.append(species)
    images = np.array(images)
    labels = np.array(labels)
    X_train, X_test, y_train, y_test = train_test_split(images, labels,␣
 ↪test_size=0.2, random_state=42)
    return X_train, X_test, y_train, y_test
```

[8]:
```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Load the datasets
X_train, X_test, y_train, y_test = load_datasets()

# Convert the labels to numerical values
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
num_classes = len(label_encoder.classes_)
```

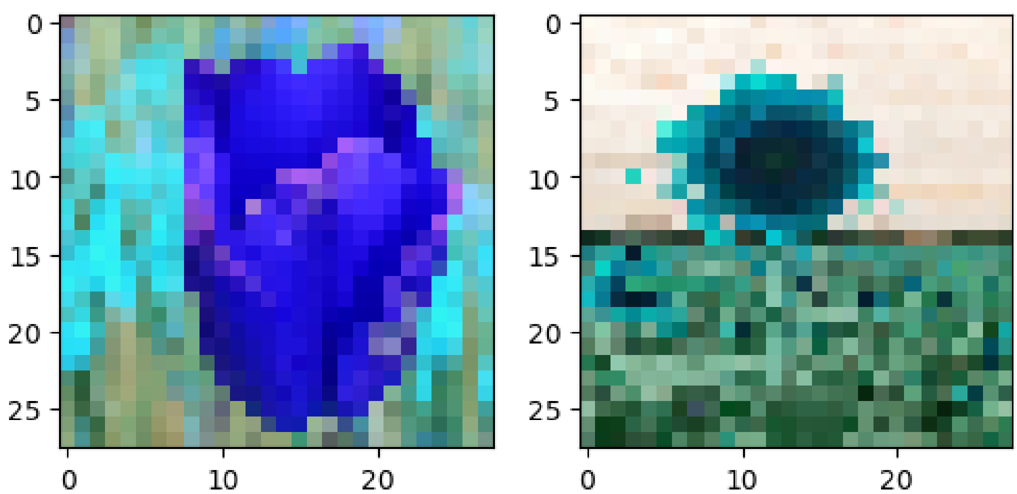
```
y_train = to_categorical(y_train, num_classes=num_classes).T
y_test = to_categorical(y_test, num_classes=num_classes).T
```

[9]:
```
X_train = X_train.reshape((X_train.shape[0], -1)).T
X_test=X_test.reshape((X_test.shape[0],-1)).T
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2352, 3309)
(2352, 828)
(5, 3309)
(5, 828)
```

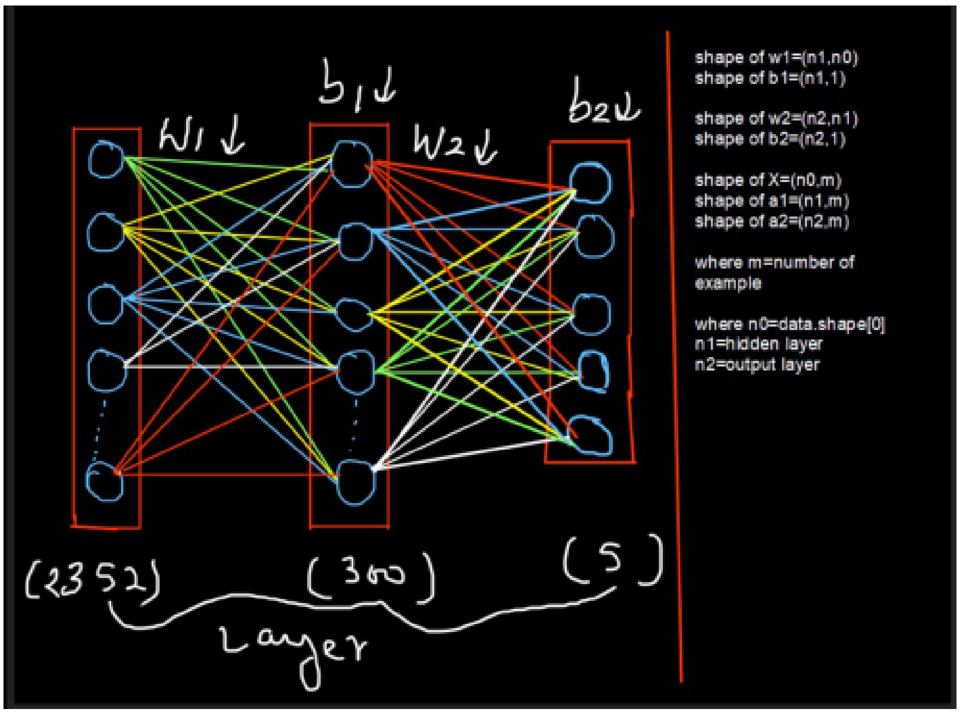
0.1.2 Step 2: show two (28*28) pixel images

[10]:
```
import random
import matplotlib.pyplot as plt

index1 = random.randrange(0, X_train.shape[1])
index2 = random.randrange(0, X_train.shape[1])

fig, axes = plt.subplots(nrows=1, ncols=2)
axes[0].imshow(X_train[:, index1].reshape(28,28,3), cmap='gray')
axes[1].imshow(X_train[:, index2].reshape(28,28,3), cmap='gray')
plt.show()
```



0.1.3 Network

[11]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('C:/Users/Umang/Downloads/last_one/neural_.jpeg')
# Show image
plt.imshow(img)
plt.axis('off')
plt.show()
```



0.1.4 Step 3: activation function

[12]:
```
def relu(Z):
    return np.maximum(Z,0)
def softmax(Z):
    exp=np.exp(Z)
    return exp/np.sum(exp,axis=0)
```

[13]:
```
def relu_backward(Z):
    return np.array(Z>0,dtype=np.float32)
```

0.1.5 Step 4: initialize the weight and bias variable

[14]:
```
def init_w_b(layers):
    w1=np.random.randn(layers[1],layers[0])*0.01
    b1=np.zeros((layers[1],1))
    w2=np.random.randn(layers[2],layers[1])*0.01
    b2=np.zeros((layers[2],1))
    w_b={
        'W1':w1,'b1':b1,'W2':w2,'b2':b2
    }
    return w_b
```

0.1.6 Step 5: forward propagation

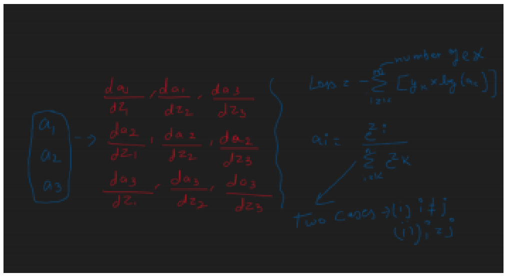
[15]:
```
def forward_prop(X,w_b):
    W1=w_b['W1']
    b1=w_b['b1']
    W2=w_b['W2']
    b2=w_b['b2']

    Z1=np.dot(W1,X)+b1
    a1=relu(Z1)
    Z2=np.dot(W2,a1)+b2
    a2=softmax(Z2)
    cache={
        'Z1':Z1,'a1':a1,'Z2':Z2,'a2':a2
    }

    return cache
```

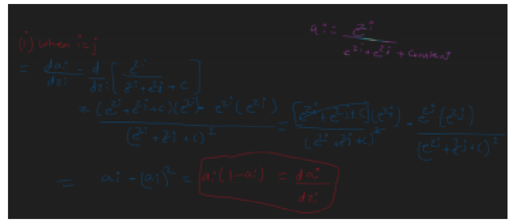
0.1.7 Step 6: compute_cost

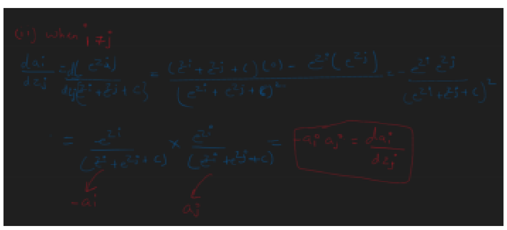
[16]:
```
def compute_cost(AL, Y,w_b,lambd):
    m = Y.shape[1]
    epsilon = 1e-8
    cost = -(1/m)*np.sum(np.multiply(Y, np.log(AL+epsilon)) + np.multiply(1-Y,␣
↪np.log(1-AL+epsilon)))
    L2= (lambd/(2*m)) * (np.sum(np.square(w_b['W1'])) + np.sum(np.
↪square(w_b['W2'])))
    cost = np.squeeze(cost+L2)
    return cost
```

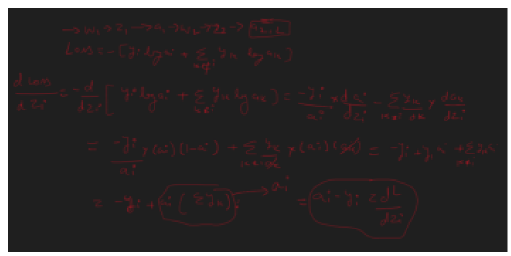
0.1.8 derivative methods for backpropagation for softmax

[31]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Read the images
img1 = mpimg.imread('methods/10.png')
img2 = mpimg.imread('methods/11.png')
img3 = mpimg.imread('methods/22.png')
img4 = mpimg.imread('methods/33.png')

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Plot the images in separate subplots
axs[0, 0].imshow(img1)
axs[0, 0].axis('off')
axs[0, 1].imshow(img2)
axs[0, 1].axis('off')
axs[1, 0].imshow(img3)
axs[1, 0].axis('off')
axs[1, 1].imshow(img4)
axs[1, 1].axis('off')

# Show the plot
plt.show()
```

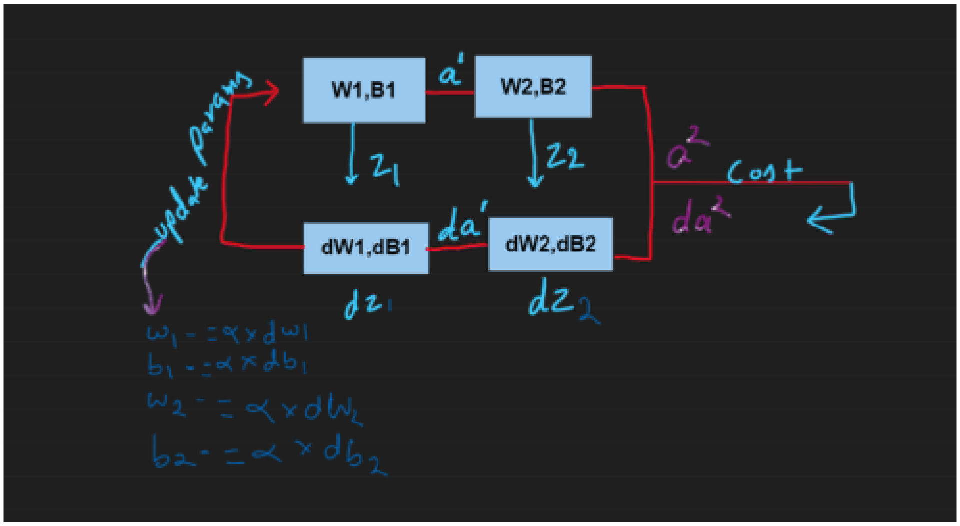








[18]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('C:/Users/Umang/Downloads/last_one/back_propagation.png')
# Show image
plt.imshow(img)
plt.axis('off')
plt.show()
```



0.1.9 Step 7: Back propagation

[19]:
```
def back_prop(X,y,w_b,cache,lambd):
    W1=w_b['W1']
    b1=w_b['b1']
    W2=w_b['W2']
    b1=w_b['b2']

    a1=cache['a1']
    a2=cache['a2']

    m=X.shape[1]
    dZ2=(a2-y)
    dW2=(1/m)*np.dot(dZ2,a1.T)+lambd/m * W2 # where (lambd/m*weight is L2 regularization)
    db2=(1/m)*np.sum(dZ2,axis=1,keepdims=True)

    dZ1=(1/m)*np.dot(W2.T,dZ2)*relu_backward(a1)
    dW1=(1/m)*np.dot(dZ1,X.T)+lambd/m * W1
    db1=(1/m)*np.sum(dZ1,axis=1,keepdims=True)

    grads={
        'dW1':dW1,'db1':db1,'dW2':dW2,'db2':db2
    }
    return grads
```

0.1.10 Step 8: update weight and bias parameter

[20]:
```
def update_w_b(w_b,grads,alpha):
    w1=w_b['W1']
    b1=w_b['b1']
    w2=w_b['W2']
    b2=w_b['b2']

    dw1=grads['dW1']
    db1=grads['db1']
    dw2=grads['dW2']
    db2=grads['db2']

    w1-=(dw1*alpha)
    b1-=(db1*alpha)
    w2-=(dw2*alpha)
    b2-=(db2*alpha)

    w_b={
        'W1':w1,'b1':b1,'W2':w2,'b2':b2
    }
    return w_b
```

0.1.11 Step 9: train the model

[21]:
```
def model(X,y,alpha,iters,layer):
    costt=[]
    w_b=init_w_b(layer)
    for i in range(iters+1):
        cache=forward_prop(X,w_b)
        cost=compute_cost(cache['a2'],y,w_b,0.7)
        grads=back_prop(X,y,w_b,cache,0.7)
        w_b=update_w_b(w_b,grads,alpha)
        costt.append(cost)
        nn=iters/5
        if(i%nn==0 or i==20000):
            print('Cost after',i,'iters is: ',cost)
    return w_b,costt
```

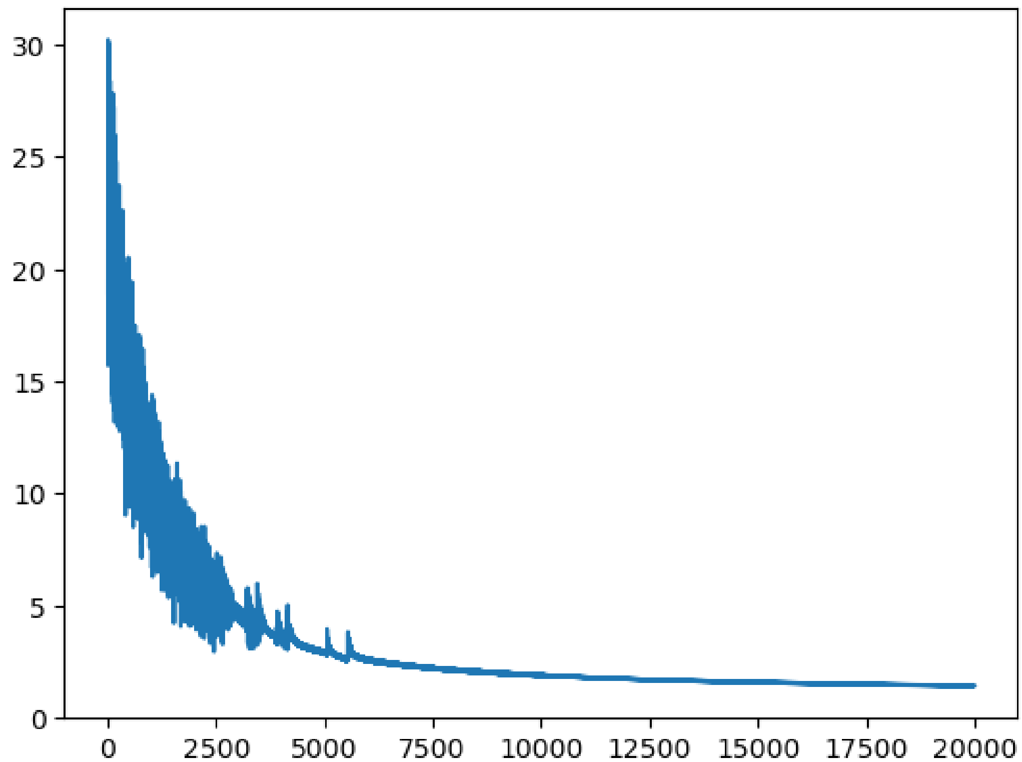
[26]:
```
layer=[X_train.shape[0],400,y_train.shape[0]]
params,costt=model(X_train,y_train,0.001,20000,layer)
```

```
Cost after 0 iters is:  18.94025249928416
Cost after 4000 iters is:  3.3461034274810193
Cost after 8000 iters is:  2.1735879741207906
Cost after 12000 iters is:  1.753008779101652
Cost after 16000 iters is:  1.5572281503495047
```

```
Cost after 20000 iters is:  1.4383941660916968
```

0.1.12 Step 10: cost per iteration graph

[27]:
```
t=np.arange(0,20001)
plt.plot(t,costt)
plt.show()
```



0.1.13 Accuracy of Model

[28]:
```
forr=forward_prop(X_train,params)
a_out=forr['a2']
a_out=np.argmax(a_out,0)
y_out=np.argmax(y_train,0)
a_out==y_out
acc=np.mean(a_out==y_out)*100
print(acc)
```

```
67.39196131761862
```