Basic Instructions

1. Enter your Name and UID in the provided space.
2. Do the assignment in the notebook itself
3. you are free to use Google Colab

Name: Paaras Bhandari
UID: 116191739

I have used Keras to implement a 2 layer binary classification neural network. Report is at the end of the notebook.

Importing Packages

In [192]:

```
# libraries for data processing and analysis
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
import math
from PIL import Image
from scipy import ndimage
import os
import tensorflow as tf
import random as rn
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import History
```

Model - Cat vs Non-Cat

Loading Data

In [193]:

```
def load_data(train_file, test_file):   # Code from HW2
    # Load the training data
    train_dataset = h5py.File(train_file, "r")

    # Separate features(x) and labels(y) for training set
    train_set_x_orig = np.array(train_dataset["train_set_x"][:])
    train_set_y_orig = np.array(train_dataset["train_set_y"][:])

    # Load the test data
    test_dataset = h5py.File(test_file, "r")

    # Separate features(x) and labels(y) for training set
    test_set_x_orig = np.array(test_dataset["test_set_x"][:])
    test_set_y_orig = np.array(test_dataset["test_set_y"][:])

    classes = np.array(test_dataset["list_classes"][:]) # the list of classes

    train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes
```

In [194]:

```
train_file="data/train_catvnoncat.h5"    #  loading training dataset
test_file="data/test_catvnoncat.h5"      #  loading test datatset
train_x_orig, train_y, test_x_orig, test_y, classes = load_data(train_file, test_file)  # seperating input and la
bels
```

Pre-processing Data

In [206]:

```
# Reshape the training and test examples
train_x_flatten = train_x_orig.reshape(train_x_orig.shape[0], -1).T   # The "-1" makes reshape flatten the remain
ing dimensions
test_x_flatten = test_x_orig.reshape(test_x_orig.shape[0], -1).T

# Standardize data to have feature values between 0 and 1.

train_x = train_x_flatten/255.
test_x = test_x_flatten/255.

print ("train_x's shape: " + str(train_x.shape))
print ("test_x's shape: " + str(test_x.shape))
```

```
train_x's shape: (12288, 209)
test_x's shape: (12288, 50)
```

Building Model

In [207]:

```
### CONSTANTS DEFINING THE MODEL ####
def buildMode(layers_dim):
    (n_x, n_h, n_y) = layers_dim

    model = Sequential()
    model.add(Dense(n_h, input_dim=n_x, activation='relu'))    # adding first hidden layer with relu activation
    model.add(Dense(n_y, activation='sigmoid'))                # adding output layer with sigmoid output

    return model

model1 = buildMode((12288, 10, 1)) # n_h = 10

## using stochastic gradient descent with an initial learning rate of 0.2
from keras.optimizers import SGD
opt = SGD(lr=0.02)
model1.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

Training Model

In [208]:

```
# function for printing loss and accuracy after every 100 epochs
def print_test_loss(history):

    loss = history.history["loss"]
    accuracy = history.history["accuracy"]

    for i in range (0,len(loss),100):
        print("Loss after " +str(i) + " epochs: " +str(loss[i]) + " | Accuracy: "+str(accuracy[i]))

history = History()
history = model1.fit(train_x.T, train_y.T, verbose=0, epochs=2500, batch_size=209, shuffle=False)
print_test_loss(history)
```

```
Loss after 0 epochs: 0.711380124092102 | Accuracy: 0.43540668
Loss after 100 epochs: 0.5643119812011719 | Accuracy: 0.75598085
Loss after 200 epochs: 0.4663430452346802 | Accuracy: 0.8229665
Loss after 300 epochs: 0.36939671635627747 | Accuracy: 0.861244
Loss after 400 epochs: 0.3206654489040375 | Accuracy: 0.8755981
Loss after 500 epochs: 0.24638144671916962 | Accuracy: 0.9186603
Loss after 600 epochs: 0.13446632027626038 | Accuracy: 0.98564595
Loss after 700 epochs: 0.08569730073213577 | Accuracy: 0.9952153
Loss after 800 epochs: 0.05594748631119728 | Accuracy: 0.9952153
Loss after 900 epochs: 0.03838104382157326 | Accuracy: 1.0
Loss after 1000 epochs: 0.02819713018834591 | Accuracy: 1.0
Loss after 1100 epochs: 0.02181633561849594 | Accuracy: 1.0
Loss after 1200 epochs: 0.017559180036187172 | Accuracy: 1.0
Loss after 1300 epochs: 0.014531448483467102 | Accuracy: 1.0
Loss after 1400 epochs: 0.012306074611842632 | Accuracy: 1.0
Loss after 1500 epochs: 0.010609985329210758 | Accuracy: 1.0
Loss after 1600 epochs: 0.009285074658691883 | Accuracy: 1.0
Loss after 1700 epochs: 0.008216540329158306 | Accuracy: 1.0
Loss after 1800 epochs: 0.00735136866569519 | Accuracy: 1.0
Loss after 1900 epochs: 0.006640611216425896 | Accuracy: 1.0
Loss after 2000 epochs: 0.006034592166543007 | Accuracy: 1.0
Loss after 2100 epochs: 0.005526149179786444 | Accuracy: 1.0
Loss after 2200 epochs: 0.00508833397179842 | Accuracy: 1.0
Loss after 2300 epochs: 0.004707040265202522 | Accuracy: 1.0
Loss after 2400 epochs: 0.004374540410935879 | Accuracy: 1.0
```

Testing Model

In [209]:

```
# evaluate the keras model
_, accuracy = model1.evaluate(test_x.T, test_y.T)
print('Accuracy: %.2f' % (accuracy*100))
```

```
50/50 [==============================] - 0s 625us/step
Accuracy: 80.00
```

Model - IMDB Reviews

Importing Packages

In [210]:

```
import re
```

Loading Data

In [211]:

```
def load_data(train_file, test_file):
    train_dataset = []
    test_dataset = []

    # Read the training dataset file line by line
    for line in open(train_file, 'r'):
        train_dataset.append(line.strip())

    for line in open(test_file, 'r'):
        test_dataset.append(line.strip())
    return train_dataset, test_dataset
```

In [212]:

```
train_file = "data/train_imdb.txt"
test_file = "data/test_imdb.txt"
train_dataset, test_dataset = load_data(train_file, test_file)
y = [1 if i < len(train_dataset)*0.5 else 0 for i in range(len(train_dataset))]
```

Pre-Processing Data

In [213]:

```
REPLACE_NO_SPACE = re.compile("(\.)|(\;)|(\:)|(\!)|(\')|(\?)|(\,)|(\")|(\()|(\))|(\[)|(\])|(\d+)")
REPLACE_WITH_SPACE = re.compile("(<br\s*/><br\s*/>)|(\-)|(\/)")
NO_SPACE = ""
SPACE = " "

def preprocess_reviews(reviews):

    reviews = [REPLACE_NO_SPACE.sub(NO_SPACE, line.lower()) for line in reviews]
    reviews = [REPLACE_WITH_SPACE.sub(SPACE, line) for line in reviews]

    return reviews

train_dataset_clean = preprocess_reviews(train_dataset)
test_dataset_clean = preprocess_reviews(test_dataset)
```

Vectorization

In [214]:

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(binary=True, stop_words="english", max_features=2000)
cv.fit(train_dataset_clean)
X = cv.transform(train_dataset_clean)

X_test = cv.transform(test_dataset_clean)
X = np.array(X.todense()).astype(float)
X_test = np.array(X_test.todense()).astype(float)
y = np.array(y)
```

Building Model

In [258]:

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(
    X, y, train_size = 0.80
)

X_train = X_train.T
X_val = X_val.T
y_train = y_train.reshape(1,-1)
y_val = y_val.reshape(1,-1)

n_x = X_train.shape[0]
n_h = 10
n_y = 1
layers_dims = (n_x, n_h, n_y)

model2 = buildMode(layers_dims)

## using stochastic gradient descent with an initial learning rate of 0.2
from keras.optimizers import SGD
opt = SGD(lr=0.01)
model2.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

Training Model

In [259]:

```
history = History()
history = model2.fit(X_train.T, y_train.T, verbose=0, epochs=2400, batch_size=X_train.shape[0], shuffle=True)
print_test_loss(history)
```

```
Loss after 0 epochs: 0.699784517288208 | Accuracy: 0.46375
Loss after 100 epochs: 0.641793429851532 | Accuracy: 0.6725
Loss after 200 epochs: 0.5812547206878662 | Accuracy: 0.7825
Loss after 300 epochs: 0.5163273811340332 | Accuracy: 0.84375
Loss after 400 epochs: 0.456033855676651 | Accuracy: 0.875
Loss after 500 epochs: 0.4040308892726898 | Accuracy: 0.8875
Loss after 600 epochs: 0.3599049746990204 | Accuracy: 0.91125
Loss after 700 epochs: 0.3221399188041687 | Accuracy: 0.92625
Loss after 800 epochs: 0.2897164821624756 | Accuracy: 0.93875
Loss after 900 epochs: 0.26175281405448914 | Accuracy: 0.95125
Loss after 1000 epochs: 0.23742519319057465 | Accuracy: 0.9525
Loss after 1100 epochs: 0.21612773835659027 | Accuracy: 0.9575
Loss after 1200 epochs: 0.19734108448028564 | Accuracy: 0.97
Loss after 1300 epochs: 0.1806963086128235 | Accuracy: 0.975
Loss after 1400 epochs: 0.16590608656406403 | Accuracy: 0.98
Loss after 1500 epochs: 0.1527162343263626 | Accuracy: 0.98375
Loss after 1600 epochs: 0.14092588424682617 | Accuracy: 0.98625
Loss after 1700 epochs: 0.13036030530929565 | Accuracy: 0.98625
Loss after 1800 epochs: 0.1208539679646492 | Accuracy: 0.98875
Loss after 1900 epochs: 0.11228891462087631 | Accuracy: 0.9925
Loss after 2000 epochs: 0.10455211997032166 | Accuracy: 0.99375
Loss after 2100 epochs: 0.09754742681980133 | Accuracy: 0.995
Loss after 2200 epochs: 0.09119144082069397 | Accuracy: 0.995
Loss after 2300 epochs: 0.08541058748960495 | Accuracy: 0.995
```

Testing Model

In [260]:

```
# evaluate the keras model
_, accuracy = model2.evaluate(X_val.T, y_val.T)
print('Accuracy: %.2f' % (accuracy*100))
```

```
201/201 [==============================] - 0s 561us/step
Accuracy: 90.05
```

Report

I have used Keras library to implement a 2 layer neural network. The first hidden layer uses a ReLU activation and the second layer (final output layer) uses sigmoid.

I first tried to train the model with adam optimizer as it was the most reccomended optimizer (from online resources). However, the loss seemed to converge too early before reaching a minima. I also tried to use the adam optimizer with a custom starting learning rate but it didn't show much improvement. I then tried the SGD (Stochastic gradient descent) optimizer, starting with an initial learning rate of 0.1. The training loss was now converging faster than before, so I raised the initial learning rate to 0.2, and then 0.4. At 0.4 initial learning rate, the loss started to fluctuate/oscillate. So I finally decided to use an initial learning rate of 0.2 with the SGD optimizer.

I then tried to play aroud with the batch size and started with a batch size of 32. The training was fluctuating, and this also made sense since we learned in the lectures that if decrease the batch size, we trust the gradient less. Hence, as we increase the learning rate, we must also increase the batch size. I then increased the batch size to 64, and the training loss was still fluctuating. Finally, I decided to use the entire dataset in one batch.

I also adjusted the size of the hidden layer, by increasing it starting from 5. As I increased the size of the hidden layer, my training seemed to get slower while my test accuracy started to improve. As I increased the size of the hidden layer further, I could see my training accuracy reach 1 and the test accuracy started to decrease. So, I decided to set the size of my hidden layer as 10.

Then, I adjusted the epochs by increasing it starting from 600. At first, my model seemed to have a low test accuracy (perhaps underfitting). As I increased the epochs, my final training loss converged and the test accuracy improved. After a certain point, my test accuracy increased to 1.0 and my test accuracy started to decrease (overfitting). I then chose a reasonable number of epochs (2600).

After trying out several permutations and combinations of the above hyperparameters, I was able to achieve an accuracy of 80% on the cat vs non-cat example, and 90.5% on the IMDB reviews example.