In [2]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.datasets import mnist
from keras.layers import Dense, Activation, Dropout
from keras.utils import np_utils
from sklearn.metrics import mean_squared_error
from IPython.display import clear_output
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade (https://www.tensorflow.org/guide/migrate) now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info (https://colab.research.google.com/notebooks/tensorflow_version.ipynb).

First, Let's generate the data. We set the seed to 13 to make sure we can genrate the same data each time.

In [0]:

```
# make sure numpy can generate the same array everytime , reproducible
np.random.seed(13)
x_data = np.random.uniform(-2*np.pi,2*np.pi,120000)
y_data = 2 * (2 * np.cos(x_data)**2 -1)**2 - 1
# y_data = 2 * np.cos(2*x_data)**2 - 1
```

And then, we split the data

In [0]:

```
# randomly split the data to training and testing part
X_train, X_test, y_train, y_test = train_test_split(x_data, y_data, test_size =
0.50, random_state = 45)
```

Since we are going to train three very similar but different layers model, we can build a funcion and use a parameter to decide how many layers do we need

In [0]:

```
# define the basic neural network model
def base_model(num_unit, num_layers):

    model = Sequential()
    # Units is the output,
    model.add(Dense(units = num_unit, input_dim = 1, activation='relu'))
    # add layers
    for i in range(num_layers):
        model.add(Dense(units = num_unit, activation='relu'))
    # because we are going to solve regression problem, so we need linear functi
on in the final layer
    model.add(Dense(1, activation='linear'))
    # compile the model
    model.compile(loss = 'mean_squared_error', optimizer = 'adam', metrics = ['m
ean_squared_error'])
    # fit the model
    model.fit(X_train, y_train)
    clear_output()

    return mean_squared_error(y_test,model.predict(X_test))
```

Our objective is to plot the error. It we run the model only once, we may find some outliers, so here I run each model 5 times and calculate the mean value.

In [0]:

```
# calculate the average error for different units setting
# for each setting, we repeat 5 times and report the average test error
def get_errors(lst,num_layers):
    error = []
    for n in lst:
        ttl = 0
        for i in range(5):
            mse_score = base_model(n,num_layers)
            ttl += mse_score
        ttl /= 5
        error.append(ttl)
    return error
```

Now it's time to get the error/result.

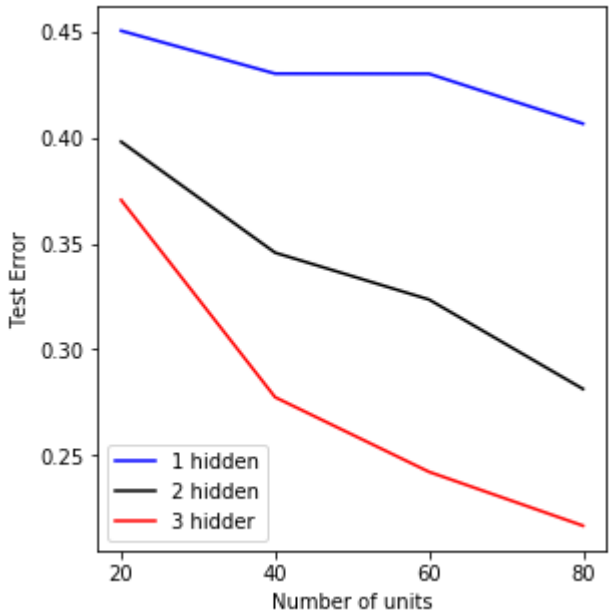
In [0]:

```
lay1_error = get_errors([20,40,60,80],1)
lay2_error = get_errors([20,40,60,80],2)
lay3_error = get_errors([20,40,60,80],3)
```

In [18]:

```
import matplotlib.pyplot as plt

plt.figure(figsize = (10,5))
plt.subplot(121)
x = [20,40,60,80]
plt.plot(x, lay1_error, color = 'b', label = '1 hidden')
plt.plot(x, lay2_error, color = 'k', label = '2 hidden')
plt.plot(x, lay3_error, color = 'r', label = '3 hidder')
plt.legend()
plt.xticks(x)
plt.xlabel('Number of units')
plt.ylabel('Test Error')
plt.show()
```



Question2

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
test.csv	just now	0 seconds	0 seconds	0.42323

Complete

Jump to your position on the leaderboard

Make a submission for X.chen

In [0]:

```
def read_image(file_path):
    img = cv2.imread(file_path, cv2.IMREAD_COLOR) #cv2.IMREAD_GRAYSCALE
    return cv2.resize(img, (ROWS, COLS), interpolation=cv2.INTER_CUBIC)

def prep_data(images):
    count = len(images)
    data = np.ndarray((count, CHANNELS, ROWS, COLS), dtype=np.uint8)

    for i, image_file in enumerate(images):
        image = read_image(image_file)
        data[i] = image.T

    return data
```

In [0]:

```
def get_result(prediction):
    img_list = []
    for img in test_images:
        img_list.append(int(img.split('/')[-1].split('.')[0]))

    result = []
    for i in range(len(img_list)):
        result.append((img_list[i],prediction[i][0]))

    col_name = ['id','label']
    df = pd.DataFrame(result, columns = col_name).sort_values(by = 'id')
    return df
```

Loading Pictures

In [0]:

```
TRAIN_DIR = 'data/train/'
TEST_DIR = 'data/test/'

ROWS = 64
COLS = 64
CHANNELS = 3

# my laptop hidder file. Contain the words 'DS'
train_images = [TRAIN_DIR + i for i in os.listdir(TRAIN_DIR)]
test_images = [TEST_DIR + i for i in os.listdir(TEST_DIR)]
random.shuffle(test_images)
random.shuffle(train_images)
```

In [0]:

```
train = prep_data(train_images)
test = prep_data(test_images)

# print the image shape
print("Train shape: {}".format(train.shape))
print("Test shape: {}".format(test.shape))
```

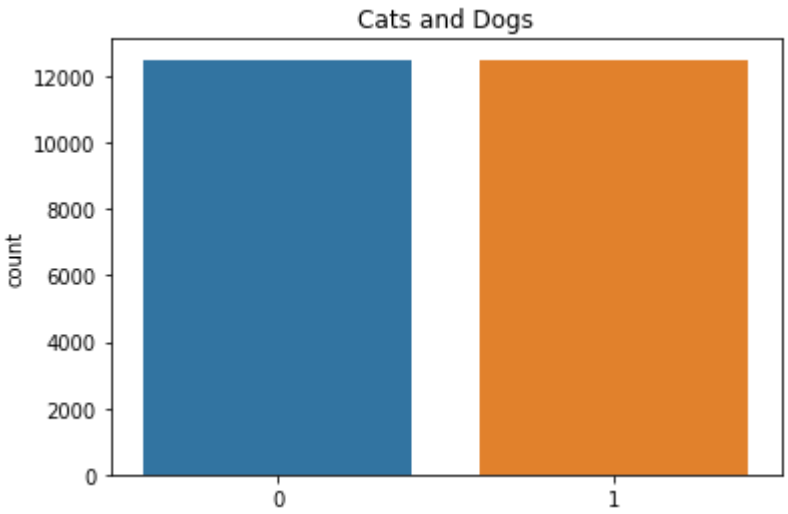
```
Train shape: (25000, 3, 64, 64)
Test shape: (12500, 3, 64, 64)
```

In [0]:

```
labels = [1 if 'dog' in i else 0 for i in train_images]
sns.countplot(labels)
plt.title('Cats and Dogs')
```

Out[0]:

```
Text(0.5, 1.0, 'Cats and Dogs')
```



Start Training

Establish my baseline model

In [0]:

```
def my_Conv2D_network(optimize):
    model = Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(3),data_format=
'channels_first'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid')) # 2 because we have cat and dog cl
asses

    model.compile(loss='binary_crossentropy', optimizer= optimize, metrics=['acc
uracy'])

    return model
# model.summary()
```

In [0]:

```
model = my_Conv2D_network('rmsprop')
```

In [0]:

```
earlystop = EarlyStopping(patience=3)
model.fit(train, labels, batch_size= 200, epochs = 20,
          validation_split=0.3, verbose=0, shuffle=True, callbacks=[early_stoppi
ng])
```

In [0]:

```
predictions = model.predict(test)
```

In [0]:

```
df = get_result(predictions)
df.to_csv('sample_result_rmsp.csv', index = False)
```

My convolutional neural network gives me a **Logloss** of 0.46234

The above method didn't try to do image normalization. All the images are colored images so that they need divided by 255 which is the total number of RGB encoder. So in the following steps, I tried normalized training data

In [0]:

```
train_255 = train / 255
test_255 = test / 255
```

In [0]:

```
model = my_Conv2D_network('rmsprop')
```

In [0]:

```
earlystop = EarlyStopping(patience = 3)
model.fit(train_255, labels, batch_size= 200, epochs = 20,
              validation_split=0.3, verbose=0, shuffle=True, callbacks=[early_st
opping])
```

In [0]:

```
predictions2 = model.predict(test_255)
```

In [0]:

```
df = get_result(predictions2)
df.to_csv('sample_result_rmsp_255.csv', index = False)
```

Only changing the training data, the same hyper-parameter settings of my convolutional neural network gives me a **Logloss** of 0.51420, which is higher than the previous one.

Here, let's try adam optimizer

In [0]:

```
model = my_Conv2D_network('adam')
```

In [0]:

```
earlystop = EarlyStopping(patience = 3)
model.fit(train255, label, batch_size= 200, epochs = 20,
              validation_split=0.3, verbose=0, shuffle=True, callbacks=[early_st
opping])
```

In [0]:

```
predictions3 = model.predict(test_255)
```

In [0]:

```
df = get_result(predictions3)
df.to_csv('sample_result_adam_255.csv', index = False)
```

Only changing the optimizer to adam, the same hyper-parameter settings of my convolutional neural network gives me a Logloss of 0.47716, which is higher than the previous one.

Try Deep Neural Network

And then, I tried some deep neural networks.

In [0]:

```
from keras.applications.resnet50 import ResNet50
from keras.applications import VGG16,VGG19
from keras import backend as K
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,GlobalAverag
ePooling2D
from keras.models import Sequential,Model,load_model
from keras.optimizers import SGD, Adam, RMSprop
K.set_image_data_format('channels_first') # use ' rather than ""
from keras.callbacks import EarlyStopping
```

In [0]:

```
def my_network(my_model,dropout,bsize):
  x = my_model.output
  x = GlobalAveragePooling2D()(x)
  x = Dropout(dropout)(x)
  predictions = Dense(1, activation= 'sigmoid')(x)
  model = Model(inputs = my_model.input, outputs = predictions)
  # sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
  adam = Adam(lr=0.0001)
  # adam = Adam(lr=0.00005)
  model.compile(optimizer= adam, loss='binary_crossentropy', metrics=['acc'])
  early_stopping = EarlyStopping(patience=5,monitor='loss')
  model.fit(train, labels, epochs = 20, batch_size = 128 , callbacks=[early_stop
ping])
  return model
```

VGG 16

Below, I am tring different hyparameter seetings with **VGG16** models.

Here is the summary: with my basic model

- droppout rate = 0.2, batch size = 128, epochs = 20, log_loss = 0.56050
- droppout rate = 0.3, batch size = 128, epochs = 20, log_loss = 0.65193
- droppout rate = 0.1, batch size = 128, epochs = 20, log_loss = 0.80925
- droppout rate = 0.6, batch size = 128, epochs = 20, log_loss = 0.80039
- droppout rate = 0.2, batch size = 64, epochs = 20, log_loss = 0.60747

In [0]:

```
vgg16 = VGG16(weights= None, include_top=False, input_shape= (3,64,64))
vgg16_model = my_network(vgg16,0.2,128)
prediction_vgg16 = vgg16_model.predict(test)
df = get_result(prediction_vgg16)
df.to_csv('/content/drive/My Drive/6420 Predictive Analytics/HOMEWORK/HW4/vgg16.csv', index = False)
```

In [0]:

```
vgg16 = VGG16(weights= None, include_top=False, input_shape= (3,64,64))
vgg16_model = my_network(vgg16,0.3,128)
prediction_vgg16 = vgg16_model.predict(test)
df = get_result(prediction_vgg16)
df.to_csv('/content/drive/My Drive/6420 Predictive Analytics/HOMEWORK/HW4/vgg16_v2.csv', index = False)
```

In [0]:

```
vgg16 = VGG16(weights= None, include_top=False, input_shape= (3,64,64))
vgg16_model = my_network(vgg16,0.1,128)
prediction_vgg16 = vgg16_model.predict(test)
df = get_result(prediction_vgg16)
df.to_csv('/content/drive/My Drive/6420 Predictive Analytics/HOMEWORK/HW4/vgg16_v3.csv', index = False)
```

In [0]:

```
vgg16 = VGG16(weights= None, include_top=False, input_shape= (3,64,64))
vgg16_model = my_network(vgg16,0.6,128)
prediction_vgg16 = vgg16_model.predict(test)
df = get_result(prediction_vgg16)
df.to_csv('/content/drive/My Drive/6420 Predictive Analytics/HOMEWORK/HW4/vgg16_v4.csv', index = False)
```

In [0]:

```
vgg16 = VGG16(weights= None, include_top=False, input_shape= (3,64,64))
vgg16_model = my_network(vgg16,0.2,64)
prediction_vgg16 = vgg16_model.predict(test)
df = get_result(prediction_vgg16)
df.to_csv('/content/drive/My Drive/6420 Predictive Analytics/HOMEWORK/HW4/vgg16_v5.csv', index = False)
```

VGG19

And the VGG19 only returns me 0.8 logloss so that I don't think VGG19 can do better in this situation. I just trained once.

In [0]:

```
vgg19 = VGG19(weights= None, include_top=False, input_shape= (3,64,64))
vgg19_model = my_network(vgg19,0.2,128)
prediction_vgg19 = vgg19_model.predict(test)
df = get_result(prediction_vgg19)
df.to_csv('/content/drive/My Drive/6420 Predictive Analytics/HOMEWORK/HW4/vgg19_v2.csv', index = False)
```

Resnet 50

Finally , I tries resnet50 with a logloss greater than 1

In [0]:

```
ResNet50 = ResNet50(weights= None, include_top=False, input_shape= (3,224,224))
ResNet50_model = my_network(ResNet50,0.2,128)
prediction_resnet50 = resnet_model.predict(test)
df = get_result(prediction_resnet50)
df.to_csv('/content/drive/My Drive/6420 Predictive Analytics/HOMEWORK/HW4/resnet.csv', index = False)
```