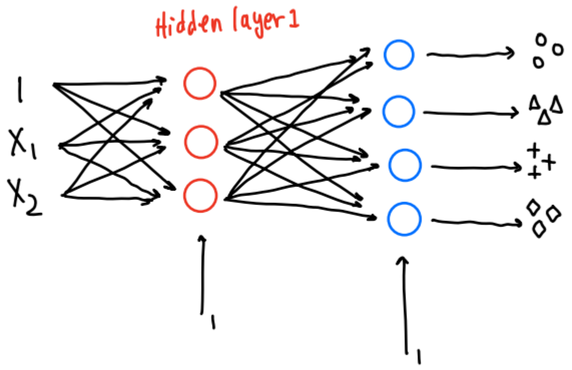
CAP6619 Deep Learning Assignment 2
Benjamin Luo (6/1/2024)

Problem 1.

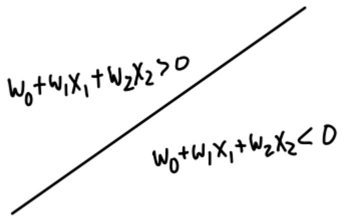


Solution. I chose two nodes (x_1, x_2) in the input layer, as we are dealing with a two-dimensional feature space. Since the classes can be roughly delineated using three lines, I chose to have three nodes in our one hidden layer. As for output nodes, there are 4 unique classes that we are trying to classify for, so each of the four output nodes are responsible for classifying one of the classes (circle, triangle, plus, and diamond). ☐

Problem 2.

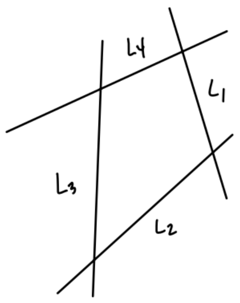
Solution.

a) Single layer neural network



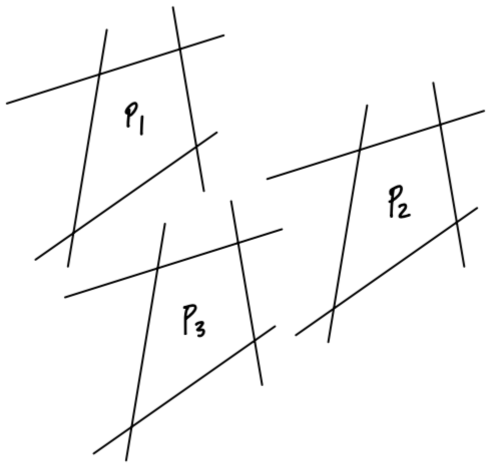
The single layer neural network classifies samples into two possible categories, based on if the output is above or below the line. If the output created by the inputs of the sample is greater than 0, then it is classified as class C_1, and otherwise, it classifies the sample as class C_2.

b) One hidden layer neural network



The decision region of a one hidden layer neural network is represented by the region that is within the lines that each hidden layer node represents. The network, over iterations will adjust weight values based on the lines represented by each hidden layer node.

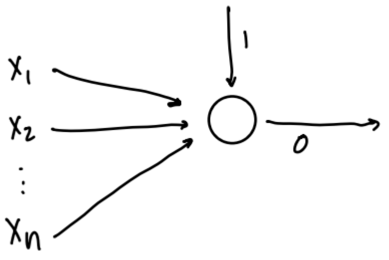
c) Two hidden layer neural network



The decision regions for this two hidden layer look like the three regions like shown in part b), but instead there are three regions instead of only 1, one for each new hidden node of the additional hidden layer.

□

Problem 3.



Solution.

The gradient descent learning objective function of the neuron is as follows:

$$f(w_0, w_1, w_2, ..., w_i) = \frac{1}{2}(d - [w_0 + w_1x_1 + w_2x_2 + \ldots + w_ix_i])^2$$

$$\text{Gradient } = \frac{\partial f(w_0, w_1, w_2, \ldots w_i)}{\partial w_i}$$
$$= \frac{1}{2} \cdot 2 \cdot (d - [w_0 + w_1x_1 + w_2x_2 + \ldots + w_ix_i]) \cdot \frac{\partial(d - [w_0 + w_1x_1 + w_2x_2 + \ldots + w_ix_i])}{\partial w_{i+1}}$$
$$= (d - [w_0 + w_1x_1 + w_2x_2 + \ldots + w_ix_i]) \cdot (-1)$$

The gradient descent learning weight update rule for weight w_i can be derived as follows:

$$\Delta w_i = -\eta \frac{\partial f(W)}{\partial w_i}$$

$$\Delta w_i = -\eta[d(n) - w_0 - w_1x_1(n) - w_2x_2(n) - \ldots - w_ix_i(n)] \cdot x_i(n)$$

$\square$

Problem 4.

Solution.

a) To calculate actual output of nodes e and f, we need to first find the outputs of the nodes a, b, c, and d, as those are the nodes whose outputs are the input values for nodes e and f. To do that, we need to first find v_a, v_b, v_c and v_d. This is done by calculating the values of all the input values and their corresponding weights together.

$v_a = x_0w_{0a} + x_1w_{1a} + x_2w_{2a}$
$v_a = 1(0.1) + 0.5(0.1) + 0.5(0.1)$
$v_a = 0.1 + 0.05 + 0.05 = 0.2$

$v_b = x_0w_{0b} + x_1w_{1b} + x_2w_{2b}$
$v_b = 1(0.2) + 0.5(0.2) + 0.5(0.2)$
$v_b = 0.2 + 0.1 + 0.1 = 0.4$

$v_c = x_0w_{0c} + x_1w_{1c} + x_2w_{2c}$
$v_c = 1(0.3) + 0.5(0.3) + 0.5(0.3)$
$v_c = 0.3 + 0.15 + 0.15 = 0.6$

$v_d = x_0w_{0d} + x_1w_{1d} + x_2w_{2d}$
$v_d = 1(0.4) + 0.5(0.4) + 0.5(0.4)$
$v_d = 0.4 + 0.2 + 0.2 = 0.8$

Then, using the sigmoid activation function with $a = 1.0$, we then find o_a, o_b, o_c, and o_d.

$o_a = \frac{1}{1+e^{-v_a}} = \frac{1}{1+e^{-0.2}} = 0.55$
$o_c = \frac{1}{1+e^{-v_c}} = \frac{1}{1+e^{-0.6}} = 0.60$

$o_a = \frac{1}{1+e^{-v_b}} = \frac{1}{1+e^{-0.4}} = 0.65$
$o_a = \frac{1}{1+e^{-v_d}} = \frac{1}{1+e^{-0.8}} = 0.69$

After finding o_a, o_b, o_c, o_d, we can calculate v_e and v_f in order to find actual outputs o_e and o_f.

$v_e = x_0w_{0e} + o_aw_{ae} + o_bw_{be} + o_cw_{ce} + o_dw_{de}$
$v_e = 1(0.2) + 0.55(0.5) + 0.60(0.5) + 0.65(0.5) + 0.69(0.5)$
$v_e = 1.445$
$o_e = \frac{1}{1+e^{-v_e}} = \frac{1}{1+e^{-1.445}}$

$o_e = 0.81$

$v_f = x_0 w_{0f} + o_a w_{af} + o_b w_{bf} + o_c w_{cf} + o_d w_{df}$ $\quad v_f = 1.694$
$v_f = 1(0.2) + 0.55(0.6) + 0.60(0.6) + 0.65(0.6) + 0.69(0.6)$ $\quad o_f = \frac{1}{1+e^{-v_e}} = \frac{1}{1+e^{-1.694}}$
$o_f = 0.845$

b) We calculate the network error of the instance as follows:
Network error is defined as the sum of the squared errors of the output neurons. To accomplish this, we find the error of output neurons e and f. We know that the instance $[1, 0.5, 0.5$ is labelled as "dog", so $d_e = 1$ and $d_f = 0$.

$e_e = d_e - o_e$ $\quad e_f = d_f - o_f$
$e_e = 1 - 0.81 = 0.19$ $\quad e_f = 0 - 0.845 = -0.845$

$E = \frac{1}{2}\sum_j e_j^2$ for j is the set of output neurons
$E = \frac{1}{2}[(0.19)^2 + (-0.845)^2]$
The network error of instance $[1, 0.5, 0.5]$ is $E = 0.375$.

c) We calculate local gradient for nodes e and f as follows:
Using the local gradient formula for output neurons $\delta_k = o_k(1 - o_k)(d_k - o_k)$,

$\delta_e = o_e(1 - o_e)(d_e - o_e)$ $\quad \delta_f = o_f(1 - o_f)(d_f - o_f)$
$\delta_e = 0.81(1 - 0.81)(1 - 0.81) = 0.029$ $\quad \delta_f = 0.845(1 - 0.845)(0 - 0.845) = -0.111$

d) We calculate local gradient for nodes a and b as follows:
Using the local gradient formula for hidden layer neurons $\delta_h = o_h(1 - o_h)\sum_k = w_{h,k}\delta_k$ for k is a neuron in the output layer:

$\delta_a = o_a(1 - o_a)\sum_k = w_{a,k}\delta_k$ $\quad \delta_b = o_b(1 - o_b)\sum_k = w_{b,k}\delta_k$
$\delta_a = o_a(1 - o_a)[w_{ae}\delta_e + w_{af}\delta_f]$ $\quad \delta_b = o_h(1 - o_h)[w_{be}\delta_e + w_{bf}\delta_f]$
$\delta_a =$ 0.55(1-0.55)[0.5(0.029)+0.6(-0.111)] $\quad \delta_b =$ 0.6(1-0.6)[0.5(0.029)+0.6(-0.111)]
$\delta_a = -0.0129$ $\quad \delta_b = -0.0125$

□

Problem 5.

Solution.

a) We find the actual outputs of each instance from the network by inputting the values of the instance into the network and going forward and the results of our output neurons are the actual output of the instance. There are three instances I_1, I_2, I_3, their actual outputs are calculated as follows:

i) Instance I_1:

$v = x_0w_{0a}+x_1w_{1a}+x_2w_{2a}$	$v_b = x_0w_{0b}+x_1w_{1b}+x_2w_{2b}$	$v_c = x_0w_{0c}+x_1w_{1c}+x_2w_{2c}$
$v_a = 1(1) + 1(1) + 0.5(1)$	$v_b = 1(1) + 1(1) + 0.5(1)$	$v_c = 1(1) + 1(1) + 0.5(1)$
$v_a = 2.5$	$v_b = 2.5$	$v_c = 2.5$
$o_a = \frac{1}{1+e^{-2.5}} = 0.924$	$o_b = \frac{1}{1+e^{-2.5}} = 0.924$	$o_c = \frac{1}{1+e^{-2.5}} = 0.924$

ii) Instance I_2:

$v = x_0w_{0a}+x_1w_{1a}+x_2w_{2a}$	$v_b = x_0w_{0b}+x_1w_{1b}+x_2w_{2b}$	$v_c = x_0w_{0c}+x_1w_{1c}+x_2w_{2c}$
$v_a = 1(1) + 0(1) + 1(1)$	$v_b = 1(1) + 0(1) + 1(1)$	$v_c = 1(1) + 0(1) + 1(1)$
$v_a = 2$	$v_b = 2$	$v_c = 2$
$o_a = \frac{1}{1+e^{-2}} = 0.881$	$o_b = \frac{1}{1+e^{-2}} = 0.881$	$o_c = \frac{1}{1+e^{-2}} = 0.881$

iii) Instance I_3:

$v = x_0w_{0a}+x_1w_{1a}+x_2w_{2a}$	$v_b = x_0w_{0b}+x_1w_{1b}+x_2w_{2b}$	$v_c = x_0w_{0c}+x_1w_{1c}+x_2w_{2c}$
$v_a = 1(1)+0.5(1)+0.5(1)$	$v_b = 1(1)+0.5(1)+0.5(1)$	$v_c = 1(1)+0.5(1)+0.5(1)$
$v_a = 2$	$v_b = 2$	$v_c = 2$
$o_a = \frac{1}{1+e^{-2}} = 0.881$	$o_b = \frac{1}{1+e^{-2}} = 0.881$	$o_c = \frac{1}{1+e^{-2}} = 0.881$

b) Finding the mean squared error of the network with respect to the three instances means that we will need to find the total squared error for each instance and then divide it by the number of instances. The mean squared error with respect to these instances is calculated as follows:

$E(I_1) = \frac{1}{2}\sum_j e_j^2(I_1)$	$E(I_2) = \frac{1}{2}\sum_j e_j^2(I_2)$	$E(I_3) = \frac{1}{2}\sum_j e_j^2(I_3)$
$E(I_1) = \frac{1}{2}[(d_c(I_1)-o_c(I_1))^2]$	$E(I_2) = \frac{1}{2}[(d_c(I_2)-o_c(I_2))^2]$	$E(I_3) = \frac{1}{2}[(d_c(I_3)-o_c(I_3))^2]$
$E(I_1) = \frac{1}{2}[(1-0.924)^2]$	$E(I_2) = \frac{1}{2}[(1-0.881)^2]$	$E(I_3) = \frac{1}{2}[(0-0.881)^2]$
$E(I_1) = 0.0029$	$E(I_2) = 0.0071$	$E(I_3) = 0.3881$

$$E(W) = \frac{1}{3}(0.029 + 0.0071 + 0.3881) = 0.1327$$

c) Using I_1 to update the weight of the network, the updated weights are calculated as follows:

- Step 1 is to calculate δ_k for output layers, which ends up being δ_c
  $\delta_c = o_c(1-o_c)(d_c-o_c)$
  $\delta_c = 0.924(1-0.924)(1-0.924)$
  $\delta_c = 0.0053$
- Step 2 is to calculate δ_h for hidden layers, which ends up being δ_a and δ_b:

$\delta_a = o_a(1-o_a)(w_{ac}\delta_c)$	$\delta_b = o_b(1-o_b)(w_{bc}\delta_c)$
$\delta_a = 0.924(1-0.924)(1 \cdot 0.0053)$	$\delta_b = 0.924(1-0.924)(1 \cdot 0.0053)$
$\delta_a = 0.00037$	$\delta_b = 0.00037$

- Now, final step is to update the weights using the formula $w_{ij} = w_{ij} + \eta \delta_j x_{ij}$:

$w_{0a} = 1.000037$ $\quad w_{0b} = 1.000037$ $\quad w_{0c} = 1.00053$
$w_{1a} = 1.000037$ $\quad w_{1b} = 1.000037$ $\quad w_{1c} = 1.00053$
$w_{2a} = 1$ $\quad w_{2b} = 1$ $\quad w_{2c} = 1$

$\square$

CAP6619 Assignment #2 Question 6

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.utils import shuffle
import tensorflow as tf
from tensorflow import keras
print(tf.__version__)
```

```
2.10.0
```

In [ ]:
```
# Loading data into notebook
data = np.load("olivetti_faces.npy")
target = np.load("olivetti_faces_target.npy")
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(400, 64, 64)
```

In [ ]:
```
def show_face_img_for_each_class(images, unique_ids):
    fig, ax = plt.subplots(nrows=4, ncols=10, figsize=(18,9))
    ax = ax.flatten()

    rand = np.random.randint(10)
    for unique_id in unique_ids:
        image_index = unique_id * 10 + rand
        ax[unique_id].imshow(images[image_index], cmap='gray')
        ax[unique_id].set_xticks([])
        ax[unique_id].set_yticks([])
        ax[unique_id].set_title("Class {}".format(unique_id))
    plt.suptitle("A face for each class in the Olivetti faces dataset")
```

In [ ]:
```
# Show at least one face image for each class in the Olivetti face dataset
show_face_img_for_each_class(data, np.unique(target))
```

A face for each class in the Olivetti faces dataset

In [ ]:
```
# Randomly split the dataset into 60% training and 40% test samples
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.4, ra
```

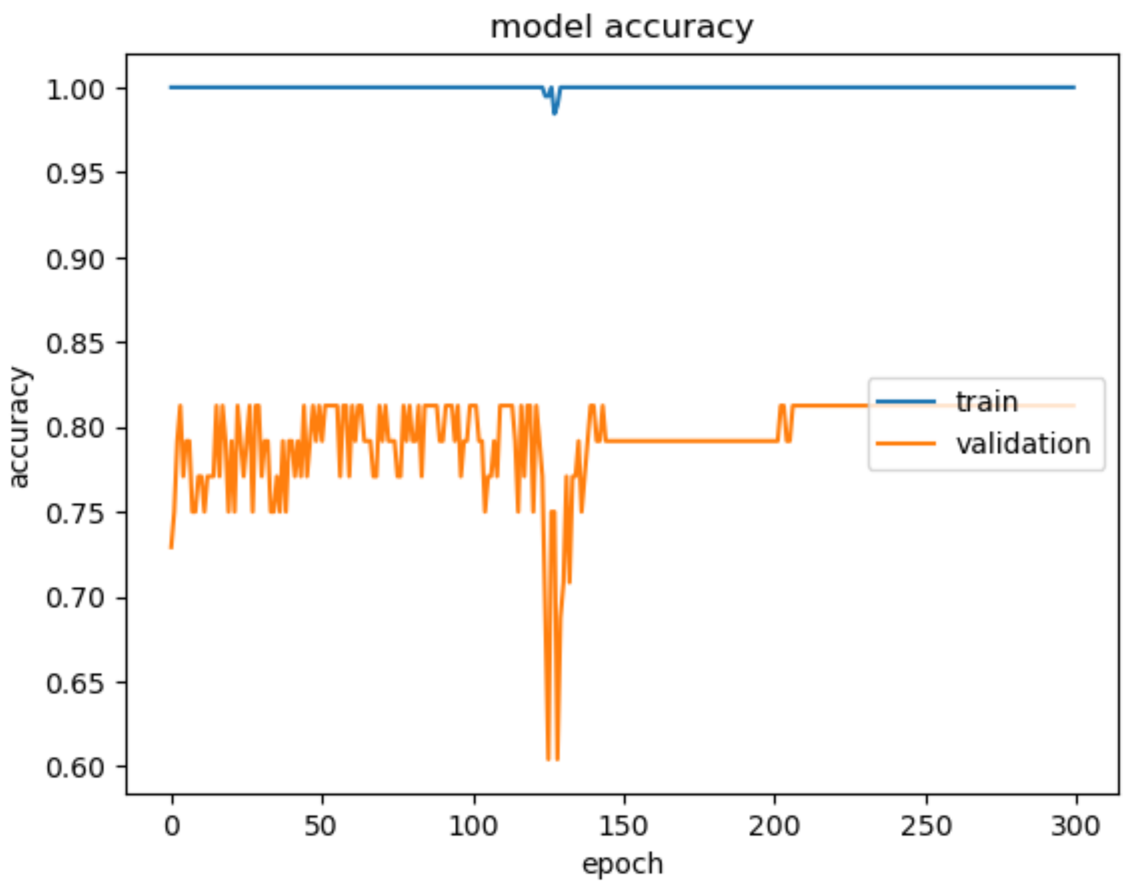
In [ ]:
```
# Create our one hidden layer neural network
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(64,64,1), name="Input"),
    keras.layers.Dense(10, activation='sigmoid', name='Hidden'),
    keras.layers.Dense(40, name="Output"),
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (Flatten)             (None, 4096)              0

 Hidden (Dense)              (None, 10)                40970

 Output (Dense)              (None, 40)                440

=================================================================
Total params: 41,410
Trainable params: 41,410
Non-trainable params: 0
_________________________________________________________________
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (Flatten)             (None, 4096)              0

 Hidden (Dense)              (None, 10)                40970

 Output (Dense)              (None, 40)                440

=================================================================
Total params: 41,410
Trainable params: 41,410
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# Train a one-hidden layer neural network with 10 hidden nodes.
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_split=0.2, epochs=300, verbose=1)
```

In [ ]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='center right')
plt.show()
```



```
In [ ]: # Report the classification accuracy of the classifier on the test set
        test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
        print('\nTest accuracy:', test_acc)
```

```
5/5 - 0s - loss: 1.4169 - accuracy: 0.7250 - 35ms/epoch - 7ms/step

Test accuracy: 0.7250000238418579

Test accuracy: 0.7250000238418579
```

```
In [ ]: # Use one time 10-fold cross validation to compare the performance
        # using different neural network architectures
```

```
In [ ]: # cross-validation framework
        from sklearn.model_selection import KFold
        kf = KFold(n_splits=10, shuffle=True)
        kf.get_n_splits(data)
        Acc = []
```

```
In [ ]: from sklearn.metrics import accuracy_score
```

```
In [ ]: # function to conduct cross validation of a given model
        def run_model(model):
            for train_index, test_index in kf.split(data):
                X_train, X_test = data[train_index], data[test_index]
                y_train, y_test = target[train_index], target[test_index]
                model = model
                y_pred=model.predict(X_test)
                y_pred=np.argmax(y_pred,axis=1)
                Acc.append(accuracy_score(y_test, y_pred))
```

In [ ]:
```
# 1) one-hidden layer NN with 10 hidden nodes
Acc = []
run_model(model)
print(np.mean(Acc), np.std(Acc))
```

```
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 21ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 16ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 41ms/step
2/2 [==============================] - 0s 4ms/step
0.8675 0.03716517186829629
```

In [ ]:
```
# 2) one-hidden layer NN with 50 hidden nodes
model50 = keras.Sequential([
    keras.layers.Flatten(input_shape=(64,64,1), name="Input"),
    keras.layers.Dense(50, activation='sigmoid', name='Hidden'),
    keras.layers.Dense(40, name="Output"),
])
model50.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
model50.fit(X_train, y_train, validation_split=0.2, epochs=300,verbose=0)
Acc = []
run_model(model50)
print(np.mean(Acc), np.std(Acc))
```

```
2/2 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 43ms/step
2/2 [==============================] - 0s 12ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 6ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 39ms/step
2/2 [==============================] - 0s 9ms/step
0.9199999999999999 0.04153311931459037
```

In [ ]:
```
# 3) one-hidden layer NN with 500 hidden nodes
model500 = keras.Sequential([
    keras.layers.Flatten(input_shape=(64,64,1), name="Input"),
    keras.layers.Dense(500, activation='sigmoid', name='Hidden'),
    keras.layers.Dense(40, name="Output"),
])
model500.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model500.fit(X_train, y_train, validation_split=0.2, epochs=300,verbose=0)
Acc = []
run_model(model500)
print(np.mean(Acc), np.std(Acc))
```

```
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 4ms/step
0.9475000000000001 0.02610076627227636
```

In [ ]:
```
# 4) two-hidden layer NN with 50 hidden nodes (1st) and 10 hidden nodes (2nd)
model2hl = keras.Sequential([
    keras.layers.Flatten(input_shape=(64,64,1), name="Input"),
    keras.layers.Dense(50, activation='sigmoid', name='HiddenLayer1'),
    keras.layers.Dense(10, activation='sigmoid', name='HiddenLayer2'),
    keras.layers.Dense(40, name="Output"),
])
model2hl.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
model2hl.fit(X_train, y_train, validation_split=0.2, epochs=300,verbose=0)
Acc = []
run_model(model2hl)
print(np.mean(Acc), np.std(Acc))
```

```
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 4ms/step
0.31 0.04769696007084728
```

Report and compare the cross-validation accuracy of the four neural networks

- NN #1 accuracy: 0.8675
- NN #2 accuracy: 0.9199
- NN #3 accuracy: 0.9475
- NN #4 accuracy: 0.31

From the results of these cross-validation, NN #3 (one-hidden layer with 500 hidden nodes) seems to be the most accurate in classifying the faces to the correct classes. It seems to be a general trend to have slightly improved accuracy as the number of hidden nodes increased within a one-hidden layer NN. NN #4's results suggest that the two-hidden layer NN has a risk of overfitting, although that may be able to remedied with other solutions or techniques.

CAP6619 Assignment #2 Question 7

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.utils import shuffle
import tensorflow as tf
from tensorflow import keras
print(tf.__version__)

from os import listdir
from numpy import asarray, save
from tensorflow.keras.utils import load_img, img_to_array
from tensorflow.keras.layers import BatchNormalization
from keras.layers import Activation, Flatten, Dense, Dropout
from tensorflow.keras import layers
from matplotlib.image import imread
```

```
2.10.0
```

In [ ]:
```
# get dataset
folder = 'dogvscat1000/'
photos, labels = list(), list()
for file in listdir(folder):
    output = 0.0
    if file.startswith('cat'):
        output = 1.0
    photo = load_img(folder + file, target_size=(32,32), color_mode='rgb')
    photo = img_to_array(photo)
    photos.append(photo)
    labels.append(output)
photos = asarray(photos)
labels = asarray(labels)
print(photos.shape, labels.shape)
save('dogs_vs_cats_photos.npy', photos)
save('dogs_vs_cats_labels.npy', labels)
```

```
(1000, 32, 32, 3) (1000,)
```

In [ ]:
```
print(photos[1].shape)
```

```
(32, 32, 3)
```

In [ ]:
```
# Create training and test datasets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(photos, labels, test_size=0.4,
```

In [ ]:
```
# Use 10-fold cross validation to split the 1000 images into training vs test
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, shuffle=True)
kf.get_n_splits(photos)
Acc, Acc2 = [], []
```

```
In [ ]: from sklearn.metrics import accuracy_score
```

```
In [ ]: # Create feedforward NN
        model = keras.Sequential([
            keras.layers.Flatten(input_shape=(32,32,3), name='Input'),
            keras.layers.Dense(50, activation='sigmoid', name='HiddenLayer1'),
            keras.layers.Dense(2, name='Output'),
        ])
```

```
In [ ]: # Create feedforward NN with batchnorm and/or dropout
        model2 = keras.Sequential([
            keras.layers.Flatten(input_shape=(32,32,3), name='Input'),
            keras.layers.Dense(100, activation='sigmoid', name='HiddenLayer1'),
            keras.layers.Dropout(rate=0.5),
            keras.layers.BatchNormalization(),
            keras.layers.Dense(2, name='Output'),
        ])
```

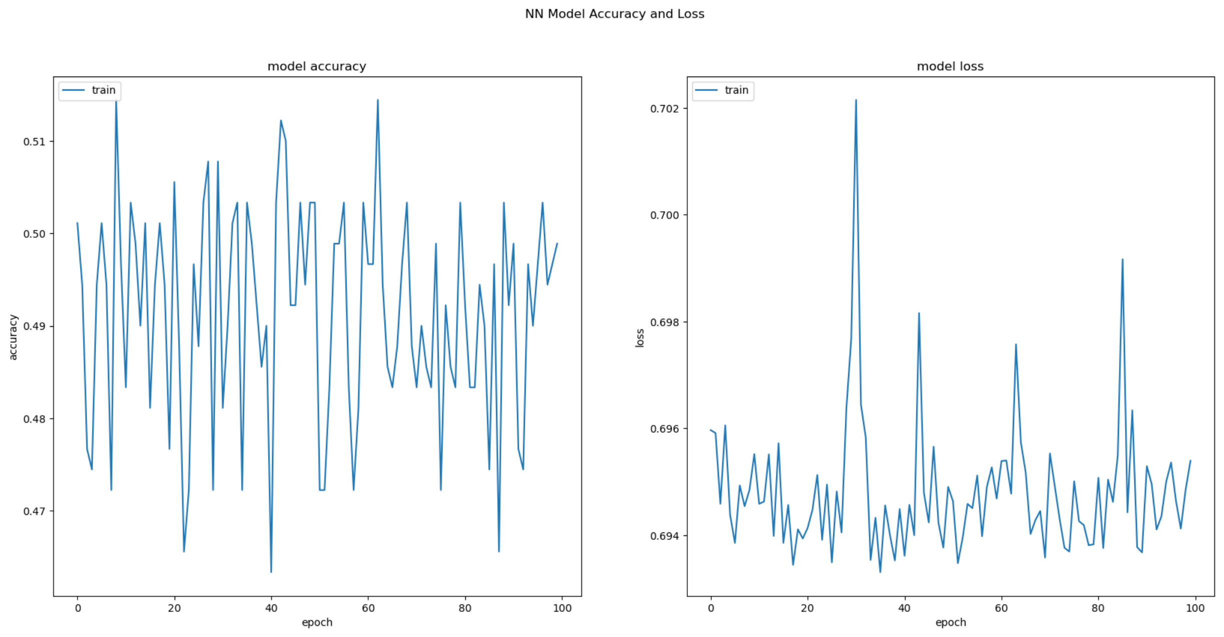
```
In [ ]: # function to conduct cross validation of a given model
        for train_index, test_index in kf.split(photos):
            X_train, X_test = photos[train_index], photos[test_index]
            y_train, y_test = labels[train_index], labels[test_index]
            model.compile(optimizer='adam',
                          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                          metrics=['accuracy'])
            model2.compile(optimizer='adam',
                          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                          metrics=['accuracy'])
            history = model.fit(X_train, y_train, epochs=100, verbose=1)
            history2 = model2.fit(X_train, y_train, epochs=100, verbose=1)
            y_pred=model.predict(X_test)
            y_pred2=model2.predict(X_test)
            y_pred=np.argmax(y_pred,axis=1)
            y_pred2=np.argmax(y_pred,axis=0)
            Acc.append(accuracy_score(y_test, y_pred))
            Acc2.append(accuracy_score(y_test, y_pred))
```

```
In [ ]: # Show accuracy of the models
        test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
        print('\nTest accuracy:', test_acc)

        test_loss, test_acc = model2.evaluate(X_test, y_test, verbose=2)
        print('\nTest accuracy:', test_acc)
```

```
4/4 - 0s - loss: 0.7004 - accuracy: 0.4700 - 179ms/epoch - 45ms/step

Test accuracy: 0.4699999988079071
4/4 - 0s - loss: 0.6939 - accuracy: 0.4700 - 240ms/epoch - 60ms/step

Test accuracy: 0.4699999988079071
```

```
In [ ]: # NN Model 1 Loss and Accuracy
        fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(20,9))
        ax1.plot(history.history['accuracy'])
        ax1.set_title('model accuracy')
```

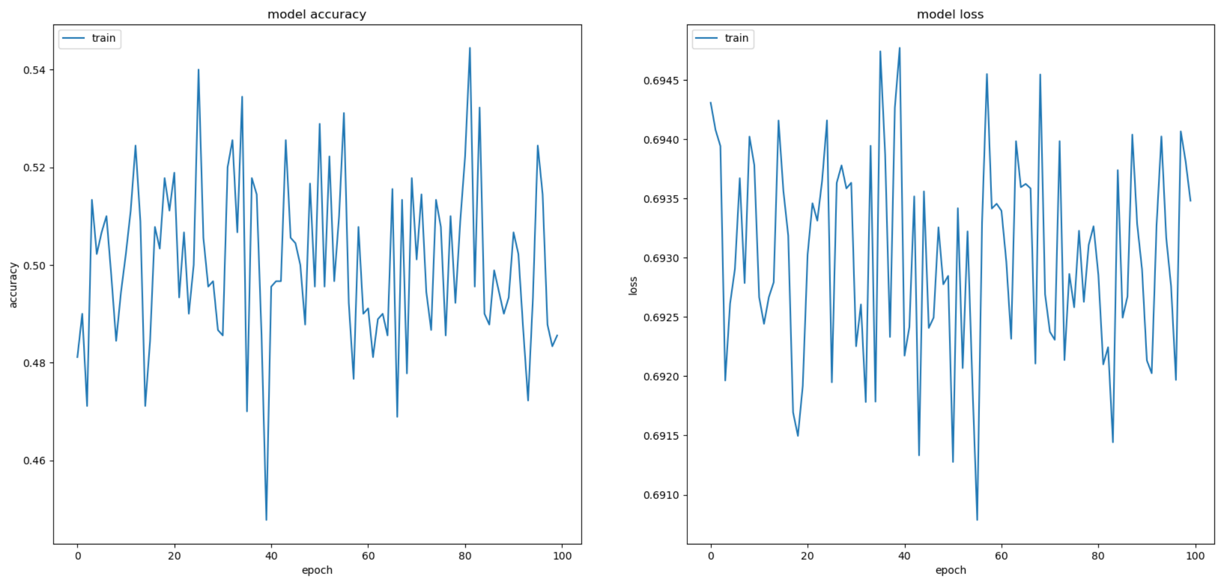
```
ax1.set_xlabel('epoch')
ax1.set_ylabel('accuracy')
ax1.legend(['train', 'validation'], loc='upper left')
ax2.plot(history.history['loss'])
ax2.set_title('model loss')
ax2.set_xlabel('epoch')
ax2.set_ylabel('loss')
ax2.legend(['train', 'validation'], loc='upper left')
fig.suptitle('NN Model Accuracy and Loss')
plt.show()
```



In [ ]:

```
# NN Model 2 (with BatchNorm and Dropout) Loss and Accuracy
fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(20,9))
ax1.plot(history2.history['accuracy'])
ax1.set_title('model accuracy')
ax1.set_xlabel('epoch')
ax1.set_ylabel('accuracy')
ax1.legend(['train', 'validation'], loc='upper left')
ax2.plot(history2.history['loss'])
ax2.set_title('model loss')
ax2.set_xlabel('epoch')
ax2.set_ylabel('loss')
ax2.legend(['train', 'validation'], loc='upper left')
fig.suptitle('NN Model (with BatchNorm and Dropout) Accuracy and Loss')
plt.show()
```

NN Model (with BatchNorm and Dropout) Accuracy and Loss



In [ ]: