Homework 4 - CS348 Spring 2022

Description - This assignment is intended to teach you about implementing neural networks for different tasks.

Getting Started - You should complete the assignment using your own installation of Python 3 and the tensorflow, numpy, and matplotlib packages. You can use other plotting packages as long as the graphics are clear and appear in the correct location in the notebook. Download the assignment from Moodle and unzip the file. This will create a directory with this file, 'hw4.ipynb', and a 'data' directory that contains all the required data files.

Datasets - This assignment uses two datasets. The dataset for the first question is located in the data folder. The dataset for the second question is loaded by running the cell just above question 2.1.

You should use only the training data to train your models, and only the testing data to test your models. Please note that these are small datasets that will allow you to produce results quickly. In practice, it is common to train neural networks using enormous datasets where training can often take many hours (or days).

Deliverables - The assignment has a single deliverable: this jupyter notebook file. Please answer all questions in the body of this file. Code should be written in a clean and organized fashion, and should be able to be run by the instructor and the TA by executing each code cell in order from top to bottom. It is strongly suggested that you reset the notebook and execute every cell in order before submitting.

Academic Honesty Statement — Copying solutions from external sources (books, web pages, etc.) or other students is considered cheating. Sharing your solutions with other students is considered cheating. Posting your code to public repositories such as GitHub is also considered cheating. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

This academic honesty statement does not restrict you from reading official documentation or using other web resources for understanding the syntax of python, related data science libraries, or properties of distributions.

In [1]:
```
# Do not import any other libraries other than those listed here.
import numpy as np
import tensorflow.keras as keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from sklearn.datasets import load_digits
from sklearn.utils import shuffle
from matplotlib import pyplot as plt
```

Problem 1 - Modeling non-linear functions

Neural networks are powerful modeling tools that can be very accurate in terms of their predictions, and also very flexible in terms of the data they are able to model. In this problem you will use deep neural networks to model a non-linear function with real-valued inputs and outputs.

The function you are going to model is

$$z = 0.3x^2 + 0.2x - 0.4y^2 - 0.4y + 0.1 + \epsilon$$

Where ϵ is a noise variable defined by sampling `np.random.normal(loc=0, scale=1)` for each datapoint. This function is convenient for plotting in 3 dimensions, but these methods can also be applied to higher dimensional data. Below is a plot of the function (without noise) generated using the Grapher application:

3d_plot

For this question you will build and train 3 separate neural networks. You may want to refer to the lecture slides from 2/24 for details. Specifically, slide 13 depicts how an individual node uses the inputs from many earlier nodes in the network to generate a single output value. The weights and bias depicted in that slide are the parameters that need to be learned from the data during training. Slide 15 depicts how the nodes can be connected in a network. In this case, it is depicting a "dense" network, where every node from a given layer is connected to every node in the subsequent layer. In practice, the number of layers, number of nodes per layer, and the connections between nodes are all hyperparameters of the model that need to be determined.

The "input" layer of a network has the same dimensions as the data and serves only to pass the data to the first hidden layer. The "output" layer must match the shape of the predictions you want from the network. For example, in a binary classification problem you would use an output layer with 2 nodes.

Each of the 3 networks you create will have a different architecture (i.e. configuration of nodes and layers). The first network is very simple, consisting of a single hidden layer with only 10 nodes. The second network will also have a single hidden layer, but it will be "wider," with many more nodes. The final network you will create will have 3 hidden layers that each have 10 nodes, making it a "deep" neural network.

The datasets for this question can be found in the "data" folder. The files labeled "dataset1*inputs*..." contain the x values in the first column and the y values in the second column. The files labeled "dataset1*outputs*..." contain the output of the function for each row.

Please use the keras module that is part of the tensorflow package (imported above as `keras`), and not the keras package itself. This will help avoid compatibility issues. You may find this documentation helpful:

- https://www.tensorflow.org/api_docs/python/tf/keras

1.1) (5 points) Using the `Sequential()` class, construct a neural network with the following architecture:

- Dense layer with 10 nodes and relu activation function.
- Dense layer with 1 node and linear activation function.

Use the `input_dim` argument in the first hidden layer to define the dimensions of the input data. This implicitly creates an input layer that passes the data to the first hidden layer. Since we are predicting a single continuous value, the final layer will have one node and a linear activation function.

Compile your model using `mean_squared_error` for the loss function, the `adam` optimizer, and `MeanSquaredError` as the metric.

Display the output of the `.summary()` method on your compiled model.

In [39]:
```
# 1.1 Solution

# use Sequential Class to construct Neural Network
neuralN = Sequential([
    Dense(units = 10, input_dim = 2, activation = 'relu'),
    Dense(units = 1, activation = 'linear')
])

neuralN.compile(optimizer = 'adam', loss =
keras.losses.MeanSquaredError(), metrics =
['MeanSquaredError'])
neuralN.summary()
```

```
Model: "sequential_28"
_________________________________________________________________
_____
 Layer (type)                Output Shape              Param
#
=================================================================
=====
 dense_83 (Dense)            (None, 10)                30

 dense_84 (Dense)            (None, 1)                 11

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

1.2) (5 points) The network you compiled in 1.1 uses several learned parameters. How many of the parameters are weights, and how many are bias terms?
(choose one)

- A) 30 weights, 30 bias terms
- B) 41 weights, 0 bias terms
- C) 21 weights, 20 bias terms
- D) 30 weights, 11 bias terms
- E) None of the above

Answer: D) 30 weights, 11 bias terms

1.3) (5 points) Consider a neural network with an input dimension of 3, 2 hidden layers with 4 nodes each, and an output layer with 2 nodes. If the layers are densely connected, how many parameters will this network have? (Note: do not build the network)
(choose one)

- A) 26
- B) 46
- C) 54
- D) 96
- E) 109
- F) None of the above

Answer: A) 26

Work:

i = 3, h = 4, 0 = 2 number of connections between first and second layer = 3 *4 = 12 number of connections between second and third layer = 4* 2 = 8 number of connections between first and second layers via bias = 4 number of connections betweeen second and third layer via bias = 2 number of parameters network has = 12 + 8 + 4 + 2 = 26

1.4) (5 points) Train your model from 1.1 using only the training data for dataset1. Run the training for 500 epochs with a batch size of 25. Use `validation_split=0.1` and 10% of the data will be withheld from training and used for validation at the end of each epoch. When you train your model, bind the result to a new variable (e.g. `trained_model = model.fit(...)`). That variable will be used to store useful information about the training process.

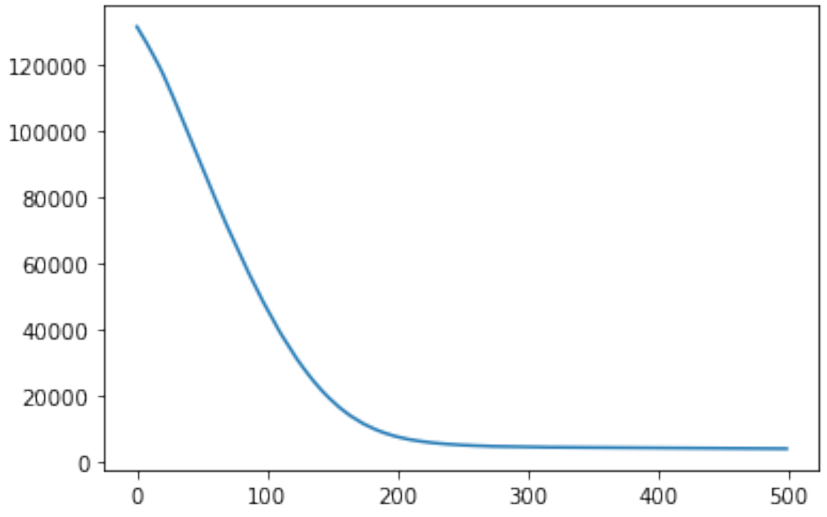
(Note: You can try `verbose=1` to see the training progress, but please specify `verbose=0` in your final submission).

In [38]:
```
# 1.4 Solution
ds1_train_input =
np.loadtxt('/Users/eeshaarabhavi/Downloads/CS348Homework/hw4/data

ds1_train_output =
np.loadtxt('/Users/eeshaarabhavi/Downloads/CS348Homework/hw4/data


trained_model = neuralN.fit(ds1_train_input, ds1_train_output,
epochs = 500, batch_size = 25, validation_split = 0.1, verbose
= 0)
```

1.5) (5 points) The variable you defined above will have a `.history` attribute that is a dictionary of different loss values for each training epoch. Plot the mean squared error on the validation set for each training epoch (`'val_mean_squared_error'`), with the training epoch on the x-axis and the loss values on the y-axis.

In [4]:
```
# 1.5 solution
plt.plot(trained_model.history['val_mean_squared_error'])
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7feb8cc2e950>]
```



1.6) (5 points) Using the `Sequential()` class, construct a new "wide" neural network with the following architecture:

- Dense layer with 30 nodes and relu activation function.
- Dense layer with 1 node and linear activation function.

Compile your model using `mean_squared_error` for the loss function, the `adam` optimizer, and `MeanSquaredError` as the metric. Then train your model on the training data using the same steps as 1.2 (remember to bind your trained model to a new variable).

In [5]:
```
# 1.6 solution
neuralNW = Sequential([
    Dense(units = 30, input_dim = 2, activation = 'relu'),
    Dense(units = 1, activation = 'linear')
])

neuralNW.compile(optimizer = 'adam', loss =
keras.losses.MeanSquaredError(), metrics =
['MeanSquaredError'])
neuralNW.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
_____
 Layer (type)                Output Shape              Param
#
=================================================================
=====
 dense_2 (Dense)             (None, 30)                90

 dense_3 (Dense)             (None, 1)                 31

=================================================================
=====
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
_____
```

1.7) (5 points) Using the `Sequential()` class, construct a new "deep" neural network with the following architecture:

- Dense layer with 10 nodes and relu activation function.
- Dense layer with 10 nodes and relu activation function.
- Dense layer with 10 nodes and relu activation function.
- Dense layer with 1 node and linear activation function.

Compile your model using `mean_squared_error` for the loss function, the `adam` optimizer, and `MeanSquaredError` as the metric. Then train your model on the training data using the same steps as 1.2 (remember to bind your trained model to a new variable).

In [6]:
```
# 1.7 solution
neuralND = Sequential([
    Dense(units = 10, input_dim = 2, activation = 'relu'),
    Dense(units = 10, input_dim = 2, activation = 'relu'),
    Dense(units = 10, input_dim = 2, activation = 'relu'),
    Dense(units = 1, activation = 'linear')
])

neuralND.compile(optimizer = 'adam', loss =
keras.losses.MeanSquaredError(), metrics =
['MeanSquaredError'])
neuralND.summary()
```

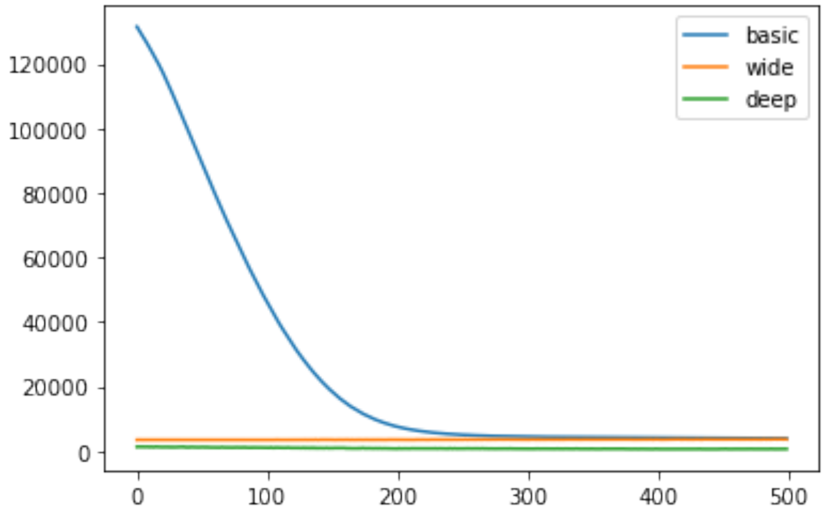
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 10)                30

 dense_5 (Dense)             (None, 10)                110

 dense_6 (Dense)             (None, 10)                110

 dense_7 (Dense)             (None, 1)                 11

=================================================================
Total params: 261
Trainable params: 261
Non-trainable params: 0
_________________________________________________________________
```

1.8) (5 points) For all three trained networks, plot the mean squared error on the validation set for each epoch on a single plot. Label the errors from the first network "basic," the second network "wide," and the third network "deep." Make sure to include a legend in the plot.

In [37]:

```python
# 1.8 Solution
w = neuralNW.fit(ds1_train_input, ds1_train_output, epochs = 500, batch_size = 25, validation_split = 0.1, verbose = 0)
d = neuralND.fit(ds1_train_input, ds1_train_output, epochs = 500, batch_size = 25, validation_split = 0.1, verbose = 0)

plt.plot(trained_model.history['val_mean_squared_error'], label = 'basic')
plt.plot(w.history['val_mean_squared_error'], label = 'wide')
plt.plot(d.history['val_mean_squared_error'], label = 'deep')
plt.legend()
```

Out[37]:
```
<matplotlib.legend.Legend at 0x7feb8d6b1b50>
```



1.9) (10 points) Briefly explain your results from 1.8 (< 5 sentences)

Answer: As the number of epochs increases, the basic model sees a significant drop in the losses until it bottoms out to a value similar to that of the wide and deep models. Both the wide and the deep models have almost the same trend except that the deep losses decrease slightly as the number of epochs increases and the wide losses increase slightly as the number of epochs increases.

1.10) (5 points) Use the `.evaluate()` method on each of the 3 networks and report the mean squared error on the test data from dataset1.

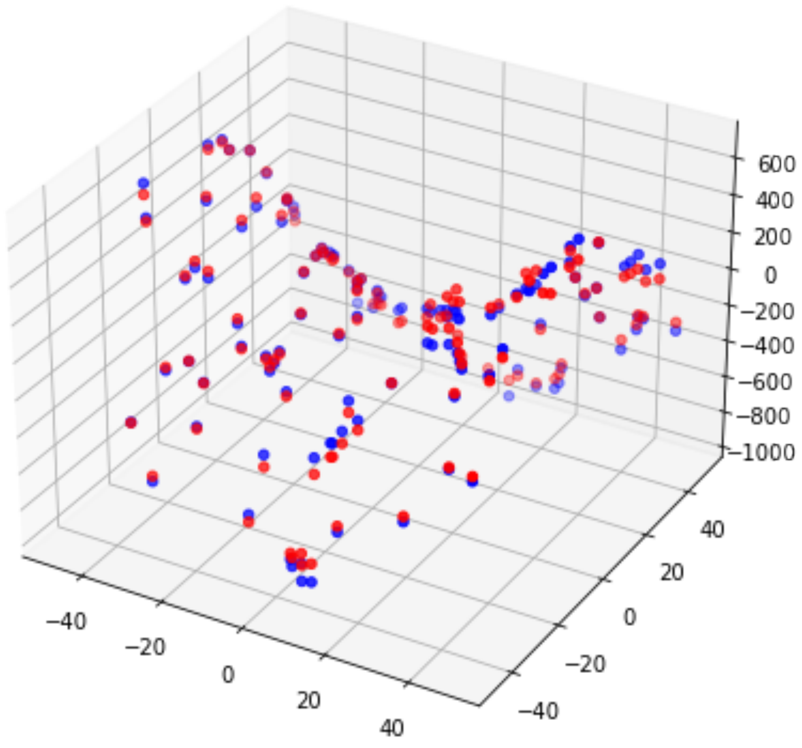
In [8]:
```
# 1.10 solution
ds1_test_input =
np.loadtxt('/Users/eeshaarabhavi/Downloads/CS348Homework/hw4/data

ds1_test_output =
np.loadtxt('/Users/eeshaarabhavi/Downloads/CS348Homework/hw4/data


bResults = neuralN.evaluate(ds1_test_input, ds1_test_output)
wResults = neuralNW.evaluate(ds1_test_input, ds1_test_output)
dResults = neuralND.evaluate(ds1_test_input, ds1_test_output)

print(' ')
print('Mean Squared Error for Data Set 1:')
print('----------------------------------')
print('Basic Neural Network Mean Squared Error: ')
print(bResults[1])
print(' ')
print('Wide Neural Network Mean Squared Error:')
print(wResults[1])
print(' ')
print('Deep Neural Network Mean Squared Error:')
print(dResults[1])
```

```
4/4 [==============================] - 0s 2ms/step - loss: 5
613.1128 - mean_squared_error: 5613.1128
4/4 [==============================] - 0s 2ms/step - loss: 5
254.5190 - mean_squared_error: 5254.5190
4/4 [==============================] - 0s 2ms/step - loss: 2
508.0798 - mean_squared_error: 2508.0798

Mean Squared Error for Data Set 1:
----------------------------------
Basic Neural Network Mean Squared Error:
5613.11279296875

Wide Neural Network Mean Squared Error:
5254.51904296875

Deep Neural Network Mean Squared Error:
2508.079833984375
```

1.11) (8 points) Use the `.predict()` method on the network with the best test set accuracy to make predictions for each test point. Then use the `scatter3D` method in matplotlib to make a 3-dimensional scatter plot showing the test data in blue and the predicted points in red.

```
In [21]: # 1.11 Solution
         pred = neuralND.predict(ds1_test_input)

         x = ds1_test_input[:, 0].reshape(-1, 1)
         y = ds1_test_input[:, 1].reshape(-1, 1)

         fig = plt.figure(figsize = (10, 7))
         ax = plt.axes(projection ="3d")

         ax.scatter3D(x, y, ds1_test_output, color = 'blue')
         ax.scatter3D(x, y, pred, color = 'red')
```

```
Out[21]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7feb90d1fe
         d0>
```



Problem 2 - Classifying images

In this problem you will use a deep neural network to classify images of hand written digits from 0 through 9. Each image is an 8x8 array of pixel values that has been flattened into a 1 dimensional feature vector. The labels identify the number that is represented in the image. Below is an example of one of the images:

If you want to plot an image you can use: `plt.imshow(image.reshape(8,8), cmap=plt.cm.binary, interpolation='nearest')` .

In [10]:
```
# load image data
digit_data = load_digits()
n = digit_data['data'].shape[0]
x = digit_data['data']
y = digit_data['target']
x, y = shuffle(x, y)
split = int(n*0.8)
x_train = np.array(x[:split,:])
y_train = keras.utils.to_categorical(y[:split],
num_classes=10)
x_test = np.array(x[split:,:])
y_test = keras.utils.to_categorical(y[split:],
num_classes=10)
```

2.1) (12 points) Write a function that uses the `Sequential()` class to construct a neural network with the following architecture:

- Dense layer with 16 nodes and relu activation function. The input dimension is 64.
- Dropout layer with a dropout rate of 0.1.
- Dense layer with 12 nodes and relu activation function.
- Dense layer with 10 nodes (one for each class) and sigmoid activation function.

Compile your model using `categorical_crossentropy` for the loss function, the `adam` optimizer, and `accuracy` as the metric.

Your function should return a compiled model.

Use your function to create a compiled model and display the output of the `.summary()` method.

In [30]:

```
# 2.1 Solution
def neural() :
    neuralNN = Sequential([
        Dense(units = 16, input_dim = 64,
activation = 'relu'),
        Dropout(0.1),
        Dense(units = 12, activation = 'relu'),
        Dense(units = 10, activation = 'sigmoid')
    ])
    neuralNN.compile(optimizer = 'adam', loss =
keras.losses.categorical_crossentropy, metrics =
['accuracy'])
    return neuralNN
nn1 = neural()
nn1.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_26 (Dense)            (None, 16)                1040      
                                                                 
 dropout_6 (Dropout)         (None, 16)                0         
                                                                 
 dense_27 (Dense)            (None, 12)                204       
                                                                 
 dense_28 (Dense)            (None, 10)                130       
                                                                 
=================================================================
Total params: 1,374
Trainable params: 1,374
Non-trainable params: 0
_________________________________________________________________
```

2.2) (25 points) Use your function from 2.1 to create 5 compiled models with the same architecture. Train each neural network for 20 epochs with a batch size of 100 and record the accuracy on the test set. When you train the first network use $1/5$ of the training data. As you train each subsequent network add an additional $1/5$ of the training data until you are using all the data to train the fifth network. Each time you test a trained network use the entire test set. Display a plot showing accuracy (on the y-axis) as a function of dataset size (on the x-axis).

In [36]:
```
# 2.2 Solution line graph
nn2 = neural()
nn3 = neural()
nn4 = neural()

traininput = np.split(ds1_train_input, 5)
trainoutput = np.split(ds1_train_output, 5)

result1 = nn1.fit(traininput[0], trainoutput[0],
epochs = 20, batch_size = 100, validation_split =
0.1, verbose = 0).evaluate(ds1_test_input,
ds1_test_output)
result2 = nn1.fit(traininput[0] + traininput[1],
trainoutput[0] + trainoutput[1], epochs = 20,
batch_size = 100, validation_split = 0.1, verbose =
0).evaluate(ds1_test_input, ds1_test_output)
```

```
-------------------------------------------------
---------------------------
ValueError
Traceback (most recent call last)
/var/folders/vf/00hc7g5d7432vm8c7sz7wqmc0000gn/T
/ipykernel_13232/2518677043.py in <module>
      7 trainoutput = np.split(ds1_train_output,
5)
      8
----> 9 result1 = nn1.fit(traininput[0],
trainoutput[0], epochs = 20, batch_size = 100,
validation_split = 0.1, verbose = 0).evaluate(ds
1_test_input, ds1_test_output)
     10 result2 = nn1.fit(traininput[0] +
traininput[1], trainoutput[0] + trainoutput[1],
epochs = 20, batch_size = 100, validation_split
= 0.1, verbose = 0).evaluate(ds1_test_input,
ds1_test_output)

~/opt/anaconda3/envs/tensorflow-env/lib/python3.
7/site-packages/keras/utils/traceback_utils.py
in error_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: di
sable=broad-except
     66       filtered_tb =
_process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb
) from None
     68     finally:
     69       del filtered_tb
```

```
~/opt/anaconda3/envs/tensorflow-env/lib/python3.
7/site-packages/keras/engine/input_spec.py in as
sert_input_compatibility(input_spec, inputs, lay
er_name)
    262         if spec_dim is not None and dim
is not None:
    263           if spec_dim != dim:
--> 264             raise ValueError(f'Input {in
put_index} of layer "{layer_name}" is '
    265                              'incompatib
le with the layer: '
    266                              f'expected
shape={spec.shape}, '

ValueError: Input 0 of layer "sequential_9" is i
ncompatible with the layer: expected shape=(None
, 64), found shape=(None, 2)
```

In [ ]: