COLORADO STATE UNIVERSITY GLOBAL

CSC580: CAPSTONE: INTRODUCTION TO NEURAL NETWORKS

Credit Hours: 3

Contact Hours: This is a 3-credit course, offered in accelerated format. This means that 16 weeks of material is covered in 8 weeks. The exact number of hours per week that you can expect to spend on each course will vary based upon the weekly coursework, as well as your study style and preferences. You should plan to spend 14-20 hours per week in each course reading material, interacting on the discussion boards, writing papers, completing projects, and doing research.

Faculty Information: Faculty contact information and office hours can be found on the faculty profile page.

COURSE DESCRIPTION AND OUTCOMES

Course Description:

In this course, students will be provided with an overview of the theories and models that are used to represent neural networks. Students will gain knowledge in developing constructs to evaluate and represent components associated with neural networks and will learn about machine learning algorithms. Topics for this course include propagation, feedforward networks, perceptrons, and self-organizing networks.

Course Overview:

In this course, you will be presented with the concepts of programming structures found in machine learning. These programming structures include modeling fully connected neural networks, convolutional neural networks, recurrent neural networks, and sequential neural networks. You will be required to implement programming solutions that use neural network models to model a particular problem. The nature of such problems typically involves predicting the outcome of a future result based on accumulated data that relates to the given problem. The programming language will be Python-based TensorFlow. This course is not a programming course in TensorFlow, but relative elements of TensorFlow will be introduced as needed to complete the programming assignments.

Course Learning Outcomes:

1. Evaluate basic neural network constructs.
2. Compare techniques for the processing of data in neural networks.
3. Select numerical computation components used in deep learning problems.
4. Implement components for deep learning networks.
5. Determine associations within a set of patterns.
6. Implement techniques for determining relationships in a neural network.

Participation & Attendance

Prompt and consistent attendance in your online courses is essential for your success at CSU Global Campus. Failure to verify your attendance within the first 7 days of this course may result in your withdrawal. If for some reason you would like to drop a course, please contact your advisor.

Online classes have deadlines, assignments, and participation requirements just like on-campus classes. Budget your time carefully and keep an open line of communication with your instructor. If you are having technical problems, problems with your assignments, or other problems that are impeding your progress, let your instructor know as soon as possible.

Course Materials

Required:

Ramsundar, B., & Zadeh, R. B. (2018). *TensorFlow for deep learning: From linear regression to reinforcement learning*. O'Reilly Media, Inc. ISBN: 9781491980453

Suggested:

El-Amir, H., & Hamdy, M. (2019, December). *Deep learning pipeline: Building a deep learning model with TensorFlow*. Springer Science, Apress. ISBN: 9781484253496

Tools:

TensorFlow Library

Python with PyCharm IDE

(These are external tools that do not require integration. They also are at no cost to the student.)

NOTE: *All non-textbook required readings and materials necessary to complete assignments, discussions, and/or supplemental or required exercises are provided within the course itself. Please read through each course module carefully.*

Course Schedule

Due Dates

The Academic Week at CSU Global begins on Monday and ends the following Sunday.

- **Discussion Boards:** The original post must be completed by Thursday at 11:59 p.m. MT and Peer Responses posted by Sunday 11:59 p.m. MT. Late posts may not be awarded points.
- **Critical Thinking:** Assignments are due Sunday at 11:59 p.m. MT.

Weekly Reading and Assignment Details

Module 1

Readings

- Chapter 1 from *TensorFlow for Deep Learning*
- Nassar, M. (2018). Hierarchical bipartite graph convolution networks.

- Petersen, P., & Voigtlaender, F. (2018). Equivalence of approximation by convolutional neural networks and fully connected networks.

Discussion (25 points)

Critical Thinking (75 points)

Option #1: Building a Basic Facial Recognition Program

For this assignment, you will write Python code to detect faces in an image. Use the following Python code as a starter for your program.

Supply your own image file that contains one or more faces to identify. An example output for your program should be something like the following picture with red boxes drawn around each individual's face:

```
import PIL.ImageDraw
import face_recognition

# Load the jpg file into a numpy array
# Find all the faces in the image
# Use the following Python pseudocode as guidance for your #solution.

numberOfFaces = len(faceLocations)
print("Found {} face(s) in this picture.".format(numberOfFaces))

# Load the image into a Python Image Library object so that you can
draw on top of it and display it
pilImage = PIL.Image.fromarray(image)

for faceLocation in faceLocations:

    # Print the location of each face in this image. Each face is a
    list of co-ordinates in (top, right, bottom, left) order.

    print("A face is located at pixel location Top: {}, Left: {},
    Bottom: {}, Right: {}".format(top, left, bottom, right))

     # Draw a box around the face
     drawHandle = PIL.ImageDraw.Draw(pilImage)
     drawHandle.rectangle([left, top, right, bottom], outline="red")
```

```
# Display the image on screen
pilImage.show()
```

Develop the remaining code in the section specific to face detection. Because most human faces have roughly the same structure, the pre-trained face detection model will work well for almost any image. There's no need to train a new one from scratch. Use PIL, which is the Python Image Library. Submit your image file input and completed Python source as a zip file named: CSC580_CTA_1_1_last_name_first_name.zip.

Option #2: Using Machine Learning to Perform Linear Regression

For this assignment, you will write a Python machine learning program to solve an algebraic problem. Consider the following sample training set:

X			Y
x1	x2	x3	
1	2	3	14
4	5	6	32
11	12	13	74
21	22	23	134
5	5	5	30

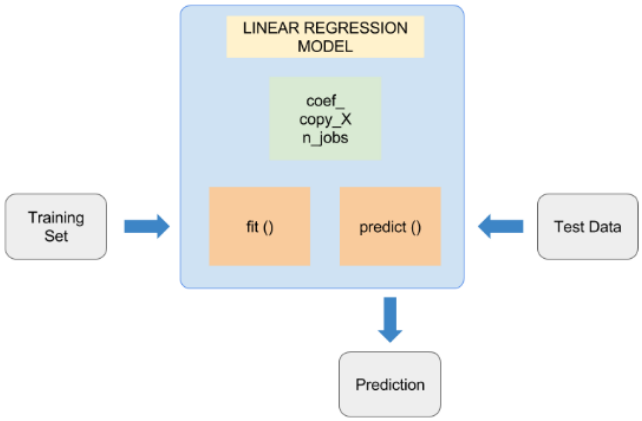
It turns out that $\square = \square_1 + 2\square_2 + 3\square_3$. The question is—given any arbitrary tuple of three numbers—what is the
predicted output from a machine-learning model that supposedly represents the algebraic equation? Keep in mind that the equation is not available for direct computation.

First, you want to generate a training data set that "conforms" to the algebraic equation. The following is an example:

```
from random import randint
TRAIN_SET_LIMIT = 1000
TRAIN_SET_COUNT = 100

TRAIN_INPUT = list()
TRAIN_OUTPUT = list()
for i in range(TRAIN_SET_COUNT):
    a = randint(0, TRAIN_SET_LIMIT)
    b = randint(0, TRAIN_SET_LIMIT)
    c = randint(0, TRAIN_SET_LIMIT)
    op = a + (2*b) + (3*c)
    TRAIN_INPUT.append([a, b, c])
    TRAIN_OUTPUT.append(op)
```

Each tuple in TRAIN_OUTPUT fits the given equation.

Second, you'll use the linear regression model from the Python package SciKit Learn (freely available) to train the model. You're essentially performing the following machine learning workflow:



(Source: Sah, 2017, para. 9)

```
from sklearn.linear_model import LinearRegression

predictor = LinearRegression(n_jobs=-1)
predictor.fit(X=TRAIN_INPUT, y=TRAIN_OUTPUT)
```

Lastly, you'll test your model with a given set of inputs, like the following:

```
X_TEST = [[10, 20, 30]]

outcome = predictor.predict(X=X_TEST)
coefficients = predictor.coef_

print('Outcome : {}\nCoefficients : {}'.format(outcome, coefficients))
```

For this assignment, implement a Python program that allows the user to enter coefficients for a linear equation that contains between four and eight variables. Next, train a model that "fits" the linear equation. Finally, prompt the user for a set of input numbers to test the model. Print out the predicted value and the actual value. Submit your Python solution as a zip file with the name: CSC580_CTA_1_2_last_name_first_name.zip.

References:

Sah, R. (2017, March 6). *Simple machine learning model in Python in 5 lines of code* [Blog post]. https://towardsdatascience.com/simple-machine-learning-model-in-python-in-5-lines-of-code-fe03d72e78c6

Module 2

Readings

- Chapter 2 in *TensorFlow for Deep Learning*
- Duan, H., Liu, Y., Wang, D., He, L., & Xiao, X. (2019). Prediction of a multi-mode coupling model based on traffic flow tensor data. *Journal of Intelligent & Fuzzy Systems*, *36*(2), 1691–1703. https://doi.org/10.3233/JIFS-18804

Discussion (25 points)

Critical Thinking (75 points)

Option #1: Classifying Handwritten Digits

For this assignment, you will use TensorFlow with Python. This assignment assumes you have read relevant literature and have TensorFlow installed.

The standard example for machine learning these days is the MNIST data set, a collection of 70,000 handwriting samples of the numbers 0-9. Your task is to predict which number each handwritten image represents.

Each image is 28x28 grayscale pixels, so you can treat each image as just a 1D array, or tensor, of 784 numbers. As long as you're consistent in how you flatten each image into an array, it will still work.

Start by importing the data set, which conveniently is part of TensorFlow itself:

```
import tensorflow as tf

sess = tf.InteractiveSession()

mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()

train_images = x_train.reshape(60000, 784)
test_images = x_test.reshape(10000, 784)
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')

x_train, x_test = train_images / 255.0, test_images / 255.0

y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

MNIST provides 60,000 samples in a training data set, 10,000 samples in a test data set, and 5,000 samples in a "validation" data set. Validation sets are used for model selection, so you use validation data to select your model, train the model with the training set, and then evaluate the model using the test data set.

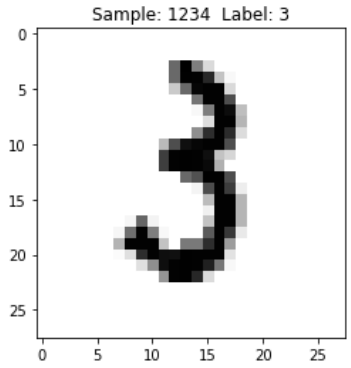
The training data, after you "flatten" it to one dimension using the reshape function, is therefore a tensor of shape [60,000, 784]: 60,000 instances of 784 numbers that represent each image.

Next, encode the label data as "one_hot" by calling the *to_categorical* function.

Define a function to visualize what the input data looks like, and pick a random training image to see what it is you're up against:

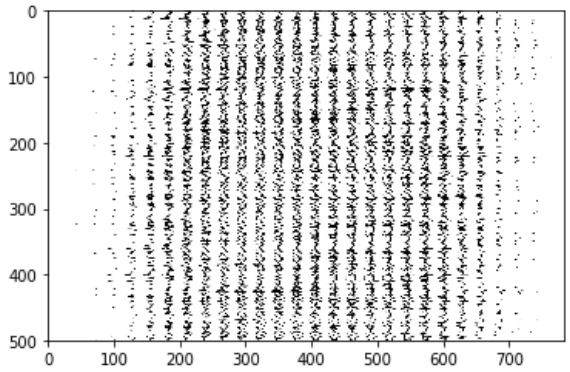
```
import matplotlib.pyplot as plt

def display_sample(num):
    #Print the one-hot array of this sample's label
    print(y_train[num])
    #Print the label converted back to a number
    label = y_train[num].argmax(axis=0)
    #Reshape the 768 values to a 28x28 image
    image = x_train[num].reshape([28,28])
    plt.title('Sample: %d  Label: %d' % (num, label))
    plt.imshow(image, cmap=plt.get_cmap('gray_r'))
    plt.show()

display_sample(1234)
```

(Note: Images were generated by the given Python code and, thus, may be included in other literary sources.)



Here's a way to visualize how the data is being fed into the model:

```
import numpy as np

images = x_train[0].reshape([1,784])
for i in range(1, 500):
    images = np.concatenate((images, x_train[i].reshape([1,784])))
plt.imshow(images, cmap=plt.get_cmap('gray_r'))
plt.show()
```



While training, you'll assign *input_images* to the training images and *target_labels* to the training labels. While testing, you'll use the test images and test labels instead.

```
input_images = tf.placeholder(tf.float32, shape=[None, 784])
target_labels = tf.placeholder(tf.float32, shape=[None, 10])
```

Now, you'll set up your deep neural network. You'll need an input layer with one node per input pixel per image, or 784 nodes. That will feed into a hidden layer of some arbitrary size, so pick 512. That hidden layer will output 10 values, corresponding to scores for each classification to be fed into *softmax*.

You'll need to reserve variables to keep track of all the weights and biases for both layers:

```
hidden_nodes = 512

input_weights = tf.Variable(tf.truncated_normal([784, hidden_nodes]))
input_biases = tf.Variable(tf.zeros([hidden_nodes]))

hidden_weights = tf.Variable(tf.truncated_normal([hidden_nodes, 10]))
hidden_biases = tf.Variable(tf.zeros([10]))
```

Next, feed that into the hidden layer, which applies the *ReLU* activation function to the weighted inputs with the learned biases added in as well.

Finally, the output layer, called *digit_weights*, multiplies in the learned weights of the hidden layer and adds in the hidden layer's bias term.

```
input_layer = tf.matmul(input_images, input_weights)
hidden_layer = tf.nn.relu(input_layer + input_biases)
digit_weights = tf.matmul(hidden_layer, hidden_weights) +
hidden_biases
```

Next, define the loss function for use in measuring progress in gradient descent: cross-entropy, which applies a logarithmic scale to penalize incorrect classifications much more than ones that are close.

```
loss_function =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=digit
_weights, labels=target_labels))
```

Now, set up the gradient descent optimizer, initializing it with an aggressive learning rate (0.5) and the loss function defined above. That learning rate is an example of a hyperparameter that may be worth experimenting with and tuning.

```
optimizer =
tf.train.GradientDescentOptimizer(0.5).minimize(loss_function)
```

Next, train the neural network and measure its accuracy. First, define some methods for measuring the accuracy of our trained model.

correct_prediction will look at the output of the neural network (in digit_weights), choose the label with the highest value, and see if that agrees with the target label given.

accuracy then takes the average of all the classifications to produce an overall score for our model's accuracy.

```
correct_prediction = tf.equal(tf.argmax(digit_weights,1),
tf.argmax(target_labels,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Now, it's time to train the model. Set up a TensorFlow session and initialize the variables. Next, train the network in 20 steps (or "epochs") with batches of 100 samples from the training data.

```
tf.global_variables_initializer().run()

EPOCH = 20
BATCH_SIZE = 100
TRAIN_DATASIZE,_ = x_train.shape
PERIOD = TRAIN_DATASIZE//BATCH_SIZE

for e in range(EPOCH):
    idxs = np.random.permutation(TRAIN_DATASIZE)
    X_random = x_train[idxs]
    Y_random = y_train[idxs]

    for i in range(PERIOD):
        batch_X = X_random[i * BATCH_SIZE:(i+1) * BATCH_SIZE]
        batch_Y = Y_random[i * BATCH_SIZE:(i+1) * BATCH_SIZE]
        optimizer.run(feed_dict = {input_images: batch_X,
target_labels:batch_Y})
```

```
        print("Training epoch " + str(e+1))
        print("Accuracy: " + str(accuracy.eval(feed_dict={input_images:
    x_test, target_labels: y_test})))
```

After you train your model, answer the following questions. You may have to write a Python script to answer a particular question.

1. What is the accuracy of the model?
2. What are some of the misclassified images?
3. How is the accuracy affected by using more hidden neurons? Fewer hidden neurons?
4. How is the accuracy affected by using different learning rates? Try a range of at least four values.
5. How is accuracy affected by adding another hidden layer?
6. How is accuracy affected by using different batch sizes? Try at least three different batch sizes.
7. What is the best accuracy you can get from this multi-layer perceptron?

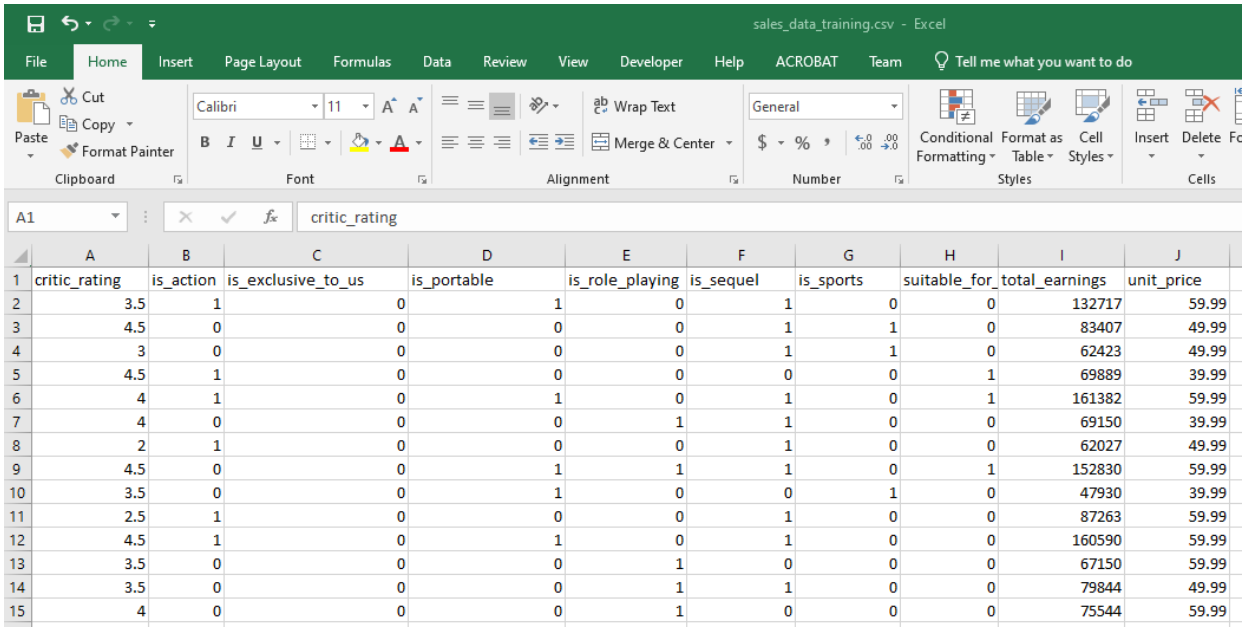
Summarize your findings in a Word document. For each finding, include the Python code and screenshots of the run-time output. Submit your Word document and Python source files as a zip file with the name: CSC580_CTA_2_1_last_name_first_name.zip.

Your paper should be a minimum of 2 pages in length and conform to the CSU Global Writing Center.

Option #2: Using Machine Learning to Perform Linear Regression

In this assignment, you will work with a neural network that can be used to predict future revenues from the sales of a new video game. A dataset is provided that you'll use to train a neural network to predict how much money you can expect future video games to earn based on historical data. The data are contained in a file named sales_data_training.csv (linked on your assignment page). In this spreadsheet, there is one row for each video game that a store has sold in the past.

Here is a screenshot of the header and first few rows:



	A	B	C	D	E	F	G	H	I	J
1	critic_rating	is_action	is_exclusive_to_us	is_portable	is_role_playing	is_sequel	is_sports	suitable_for_	total_earnings	unit_price
2	3.5	1	0	1	0	1	0	0	132717	59.99
3	4.5	0	0	0	0	1	1	0	83407	49.99
4	3	0	0	0	0	1	1	0	62423	49.99
5	4.5	1	0	0	0	0	0	1	69889	39.99
6	4	1	0	1	0	1	0	1	161382	59.99
7	4	0	0	0	1	1	0	0	69150	39.99
8	2	1	0	0	0	1	0	0	62027	49.99
9	4.5	0	0	1	1	1	0	1	152830	59.99
10	3.5	0	0	1	0	0	1	0	47930	39.99
11	2.5	1	0	0	0	1	0	0	87263	59.99
12	4.5	1	0	1	0	1	0	0	160590	59.99
13	3.5	0	0	0	1	0	0	0	67150	59.99
14	3.5	0	0	0	1	1	0	0	79844	49.99
15	4	0	0	0	1	0	0	0	75544	59.99

(Source: Retrieved from https://github.com/toniesteves/adam-geitgey-building-deep-learning-keras/tree/master/03)

The columns are defined as follows:

- critic_rating : an average star rating out of five stars.
- is_action : tells us if this was an action game.
- is_exclusive_to_us : tells us if we have an exclusive deal to sell this game.
- is_portable : tells us if this game runs on a handheld video game system.
- is_role_ playing : tells us if this is a role-playing game, which is a genre of video game.
- is_sequel : tells us if this game was a sequel to an earlier video game and part of an ongoing series.
- is_sports : tell us if this was a sports game in the sports genre.
- suitable_for_kids tells us if this game is appropriate for all ages.
- total_earnings : tells us how much money the store has earned in total from selling the game to all customers.
- unit_price tells us for how much a single copy of the game retailed.

You'll use Keras to train the neural network that will try to predict the total earnings of a new game based on these characteristics. Along with the sales_data_training.csv file, there is also a second data file called sales_data_test.csv (linked on your assignment page). This file is exactly like the first one. The machine learning system should only use the training dataset during the training phase. Then, you'll use the test data to check how well the neural network is working. To use this data to train a neural network, you first have to scale this data so that each value is between zero and one. Neural networks train best when data in each column is all scaled to the same range. Use the following Python code to scale the earning and unit price columns in both the training and test datasets. You will use Pandas to generate scaled training and test data sets.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Load training data set from CSV file
training_data_df = pd.read_csv("sales_data_training.csv")

# Load testing data set from CSV file
test_data_df = pd.read_csv("sales_data_test.csv")

# Data needs to be scaled to a small range like 0 to 1 for the neural
# network to work well.
scaler = MinMaxScaler(feature_range=(0, 1))

# Scale both the training inputs and outputs
scaled_training = scaler.fit_transform(training_data_df)
scaled_testing = scaler.transform(test_data_df)

# Print out the adjustment that the scaler applied to the total_earnings column of data
print("Note: total_earnings values were scaled by multiplying by {:.10f} and adding
{:.6f}".format(scaler.scale_[8], scaler.min_[8]))

# Create new pandas DataFrame objects from the scaled data
scaled_training_df = pd.DataFrame(scaled_training, columns=training_data_df.columns.values)
scaled_testing_df = pd.DataFrame(scaled_testing, columns=test_data_df.columns.values)

# Save scaled data dataframes to new CSV files
scaled_training_df.to_csv("sales_data_training_scaled.csv", index=False)
scaled_testing_df.to_csv("sales_data_testing_scaled.csv", index=False)
```

Now that the data has been scaled, you're ready to code the neural network. You'll code a neural network with Keras. To do this, you'll complete the given Python script named create_model.py, shown below.

```
import pandas as pd
from keras.models import Sequential
from keras.layers import *

training_data_df = pd.read_csv("sales_data_training_scaled.csv")

X = training_data_df.drop('total_earnings', axis=1).values
Y = training_data_df[['total_earnings']].values

# Define the model
model =

# Train the model
```

```
model.fit(...)

# Load the test data
# Load the separate test data set
test_data_df = pd.read_csv("sales_data_test_scaled.csv")

X_test = test_data_df.drop('total_earnings', axis=1).values
Y_test = test_data_df[['total_earnings']].values
```

First, on line five, use the Python package, Pandas library, to load the pre-scaled data from a CSV file. Each row of the dataset contains several features that describe each video game and then the total earnings value for that game. You want to split that data into two separate arrays: one with just the input features for each game and one with just the expected earnings.

On line seven, to get just the input features, we grab all of the columns of the training data but drop the total earnings column. Then, on line eight, extract just the total earnings column as shown. Now, X contains all the input features for each game, and Y contains only the expected earnings for each game. Now, you can build a neural network starting on line 11.

Incorporate the following parameters into your model definition:

- use a sequential model
- use nine inputs and one output
- make the model dense
- use the ReLU activation function for the hidden layers
- use the linear activation function for the output layer.

Train your model using both X and Y as well as the following:

- 50 epochs
- shuffle=True; this action will make Keras shuffle the data randomly during each epoch
- verbose = 2; this tells Keras to print detailed information during the processing. Take a screenshot of these messages for your submission.

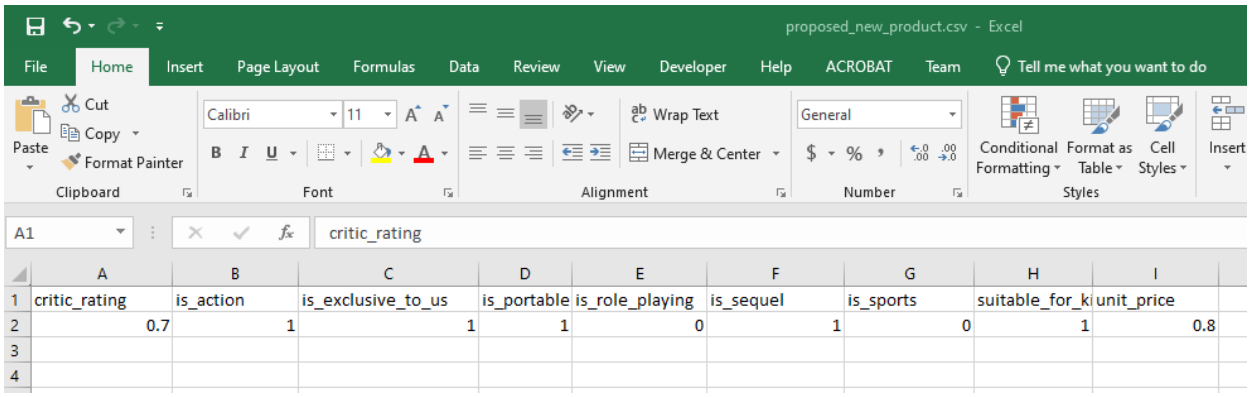
Evaluate your neural network model using model.evaluate(...) method. Print out the MSE for the test dataset.

```
test_error_rate = model.evaluate(...)
print("The mean squared error (MSE) for the test data set is: {}".format(test_error_rate))
```

Save your trained model. You will submit this model as part of your assignment.

```
# Save the model to disk
model.save("trained_model.h5")
print("Model saved to disk.")
```

Next, you will load your trained model to make predictions. The prediction information is stored in proposed_new_product.csv and consists of one row as shown below:



	A	B	C	D	E	F	G	H	I
1	critic_rating	is_action	is_exclusive_to_us	is_portable	is_role_playing	is_sequel	is_sports	suitable_for_k	unit_price
2	0.7	1	1	1	0	1	0	1	0.8
3									
4									

Complete the final segment of Python code. Be sure to rescale your final prediction using the two parameters during the scaling of the training and testing data sets.

```
import pandas as pd
from keras.models import load_model

model = load_model('trained_model.h5')

X = pd.read_csv("proposed_new_product.csv").values
prediction = model.predict(...)

# Grab just the first element of the first prediction (since we only have one)
prediction = prediction[...][...]

# Re-scale the data from the 0-to-1 range back to dollars
# These constants are from when the data was originally scaled down to the 0-to-1 range
prediction = prediction + _____
prediction = prediction / _____

print("Earnings Prediction for Proposed Product - ${}".format(prediction))
```

For your deliverable, include an introduction of your finding in a Word Document. Submit your Python code and a screenshot showing the verbose run-time outputs, the testing MSE, and your final prediction in a zip archive file. Name your archive file: CSC580_CTA_2_2_last_name_first_name.zip.

Your paper should be a minimum of two pages in length and conform to the CSU Global Writing Center.

Module 3

Readings

- Chapter 3 from *Tensorflow for Deep Learning*
- Zhang, J., Jiang, J., & Liu, Y. (2018). Tensor learning and automated rank selection for regression-based video classification. *Multimedia Tools & Applications*, *77*(22), 29213–29230.

Discussion (25 points)

Critical Thinking (75 points)

Option #1: Linear Regression Using TensorFlow

In this assignment, you will use TensorFlow to predict the next output from a given set of random inputs. Start by importing the necessary libraries. You will use Numpy along with TensorFlow for computations and Matplotlib for plotting.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

In order to make the random numbers predictable, we will define fixed seeds for both Numpy and TensorFlow.

```
np.random.seed(101)
tf.set_random_seed(101)
```

Now, let's generate some random data for training the Linear Regression Model.

```
# Generating random linear data
# There will be 50 data points ranging from 0 to 50
x = np.linspace(0, 50, 50)
y = np.linspace(0, 50, 50)

# Adding noise to the random linear data
x += np.random.uniform(-4, 4, 50)
y += np.random.uniform(-4, 4, 50)

n = len(x) # Number of data points
```

Complete the following steps:

1. Plot the training data.
2. Create a TensorFlow model by defining the placeholders X and Y so that you can feed your training examples X and Y into the optimizer during the training process.
3. Declare two trainable TensorFlow variables for the weights and bias and initialize them randomly.
4. Define the hyperparameters for the model:
   learning_rate = 0.01
   training_epochs = 1000
5. Implement Python code for:
   - the hypothesis,
   - the cost function,
   - the optimizer.
6. Implement the training process inside a TensorFlow session.
7. Print out the results for the training cost, weight, and bias.
8. Plot the fitted line on top of the original data.

For your deliverable, submit an introduction in a Word document. Submit your Python code and screenshots of your plots in a zip archive file. Name your archive file: CSC580_CTA_3_1_last_name_first_name.zip.

Option #2: Predicting Fuel Efficiency Using TensorFlow

In a *regression* problem, we aim to predict the output of a continuous value, like a price or a probability. Contrast this with a *classification* problem, where we aim to select a class from a list of classes (for example, where a picture contains an apple or an orange, recognizing which fruit is in the picture).

This assignment uses the classic Auto MPG Dataset and builds a model to predict the fuel efficiency of late-1970s and early-1980s automobiles. The data model includes descriptions of many automobiles from that time period. The description for an automobile includes attributes such as cylinders, displacement, horsepower, and weight.

For this project, you will use the tf.keras API. The Python packages you will need include the following:

```
# Use seaborn for pairplot
!pip install -q seaborn

# Use some functions from Tensorflow_docs
!pip install -q git+https://github.com/tensorflow/docs

from    future__import absolute_import, division, print_function, unicode_literals

import pathlib

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers

print(tf.   version   )
```

```
2.0.0
```

```
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import tensorflow_docs.modeling
```

Step 1: Download the dataset using keras get_file method.

```
dataset_path = keras.utils.get_file("auto-mpg.data", "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data")
dataset_path
```

```
Downloading data from http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data
32768/30286 [==============================] - 0s 1us/step

'/home/kbuilder/.keras/datasets/auto-mpg.data'
```

Step 2: Import database using Pandas.

```
column_names = ['MPG','Cylinders','Displacement','Horsepower','Weight',
                'Acceleration', 'Model Year', 'Origin']
raw_dataset = pd.read_csv(dataset_path, names=column_names,
                          na_values = "?", comment='\t',
                          sep=" ", skipinitialspace=True)

dataset = raw_dataset.copy()
dataset.tail()
```

Step 3: Take a screenshot of the tail of the dataset.

Step 4: Split the data into train and test.

```
train_dataset = dataset.sample(frac=0.8,random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

Step 5: Inspect the data.

```
sns.pairplot(train_dataset[["MPG", "Cylinders", "Displacement", "Weight"]], diag_kind="kde")
```

Step 6: Take a screenshot of the tail of the plots.

Step 7: Review the statistics.

```
train_stats = train_dataset.describe()
train_stats.pop("MPG")
train_stats = train_stats.transpose()
train_stats
```

Step 8: Take a screenshot of the tail of the statistics.

Step 9: Split features from labels.

Step 10: Separate the target value, or "label," from the features. This label is the value that you will train the model to predict.

```
train_labels = train_dataset.pop('MPG')
test_labels = test_dataset.pop('MPG')
```

Step 11: Normalize the data. It is good practice to normalize features that use different scales and ranges. Although the model *might* converge without feature normalization, it makes training more difficult, and it makes the resulting model dependent on the choice of units used in the input.

```
def norm(x):
  return (x - train_stats['mean']) / train_stats['std']
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)
```

This normalized data is what you will use to train the model.

Step 12: Build the model.

Use a *Sequential* model with two densely connected hidden layers and an output layer that returns a single, continuous value. The model building steps are wrapped in a function, build_model.

```
def build_model():
  model = keras.Sequential([
    layers.Dense(64, activation='relu',       input_shape=[len(train_dataset.keys())]),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
  ])

  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse'])
  return model

model = build_model()
```

Step 13: Inspect the model.

Use the `.summary` method to print a simple description of the model.

```
model.summary()
```

Step 14: Take a screenshot of the model summary.

Step 15: Now, try out the model. Take a batch of `10` examples from the training data and call `model.predict` on it.

Step 16: Provide a screenshot of the model summary.

Step 17: Train the model.

Step 18: Train the model for 1000 epochs, and record the training and validation accuracy in the `history` object.

```
EPOCHS = 1000

history = model.fit(
  normed_train_data, train_labels,
  epochs=EPOCHS, validation_split = 0.2, verbose=0,
  callbacks=[tfdocs.modeling.EpochDots()])
```

Step 19: Visualize the model's training progress using the stats stored in the `history` object.

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

Step 20: Provide a screenshot of the tail of the history.

```
plotter = tfdocs.plots.HistoryPlotter(smoothing_std=2)

plotter.plot({'Basic': history}, metric = "mae")
plt.ylim([0, 10])
plt.ylabel('MAE [MPG]')
```

Step 21: Provide a screenshot of the history plot.

```
plotter.plot({'Basic': history}, metric = "mse")
 plt.ylim([0, 20])
 plt.ylabel('MSE [MPG^2]')
```

Step 22: Compare the two models, one using Mean Absolute Error and the other using Mean Square Error. Which fitted model is better? Are any other models "useful?"

Step 23: For your deliverable, provide a detailed analysis using your screenshots as supporting content. Write up your analysis using a Word document. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_CTA_3_2_last_name_first_name.zip.

Module 4

Readings

- Chapter 4 in *Tensorflow for Deep Learning*
- Long, P. M., & Sedghi, H. (2019). On the effect of the activation function on the distribution of hidden nodes in a deep network.

Discussion (25 points)

Critical Thinking (75 points)

Option #1: Toxicology Testing

For this assignment, you will use a chemical dataset. Toxicologists are very interested in the task of using machine learning to predict whether a given compound will be toxic. This task is extremely complicated because science has only a limited understanding of the metabolic processes that happen in a human body. Biologists and chemists, however, have worked out a limited set of experiments that provide indications of toxicity. If a compound is a "hit" in one of these experiments, it will likely be toxic for humans to ingest.

Step 1: Load the Tox21 Dataset.

```
import numpy as np
np.random.seed(456)
import  tensorflow as tf
tf.set_random_seed(456)
import matplotlib.pyplot as plt
import deepchem as dc
from sklearn.metrics import accuracy_score

_, (train, valid, test), _ = dc.molnet.load_tox21()
train_X, train_y, train_w = train.X, train.y, train.w
valid_X, valid_y, valid_w = valid.X, valid.y, valid.w
test_X, test_y, test_w = test.X, test.y, test.w
```

Step 2: Remove extra datasets.

```
# Remove extra tasks
train_y = train_y[:, 0]
valid_y = valid_y[:, 0]
test_y = test_y[:, 0]
train_w = train_w[:, 0]
valid_w = valid_w[:, 0]
test_w = test_w[:, 0]
```

Step 3: Define placeholders that accept minibatches of different sizes.

```
# Generate tensorflow graph
d = 1024
n_hidden = 50
learning_rate = .001
n_epochs = 10
batch_size = 100

with tf.name_scope("placeholders"):
  x = tf.placeholder(tf.float32, (None, d))
  y = tf.placeholder(tf.float32, (None,))
```

Step 4: Implement a hidden layer.

```
with tf.name_scope("hidden-layer"):
  W = tf.Variable(tf.random_normal((d, n_hidden)))
  b = tf.Variable(tf.random_normal((n_hidden,)))
  x_hidden = tf.nn.relu(tf.matmul(x, W) + b)
```

Step 5: Complete the fully connected architecture.

```
with tf.name_scope("output"):
  W = tf.Variable(tf.random_normal((n_hidden, 1)))
  b = tf.Variable(tf.random_normal((1,)))
  y_logit = tf.matmul(x_hidden, W) + b
  # the sigmoid gives the class probability of 1
  y_one_prob = tf.sigmoid(y_logit)
  # Rounding P(y=1) will give the correct prediction.
  y_pred = tf.round(y_one_prob)
with tf.name_scope("loss"):
  # Compute the cross-entropy term for each datapoint
  y_expand = tf.expand_dims(y, 1)
```

```
  entropy =
tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit,
labels=y_expand)
  # Sum all contributions
  l = tf.reduce_sum(entropy)

with tf.name_scope("optim"):
  train_op = tf.train.AdamOptimizer(learning_rate).minimize(l)

with tf.name_scope("summaries"):
  tf.summary.scalar("loss", l)
  merged = tf.summary.merge_all()
```

Step 6: Add dropout to a hidden layer. Step

7: Define a hidden layer with dropout. Step

8: Implement mini-batching training.

```
train_writer = tf.summary.FileWriter('/tmp/fcnet-tox21',
                                      tf.get_default_graph())
N = train_X.shape[0]
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  step = 0
  for epoch in range(n_epochs):
    pos = 0
    while pos < N:
      batch_X = train_X[pos:pos+batch_size]
      batch_y = train_y[pos:pos+batch_size]
      feed_dict = {x: batch_X, y: batch_y}
      _, summary, loss = sess.run([train_op, merged, l],
feed_dict=feed_dict)
      print("epoch %d, step %d, loss: %f" % (epoch, step, loss))
      train_writer.add_summary(summary, step)

      step += 1
      pos += batch_size

# Make Predictions
valid_y_pred = sess.run(y_pred, feed_dict={x: valid_X})
```

Step 9: Use TensorBoard to track model convergence.

In a Word document, provide an assessment of the model accuracy. Additionally, in your Word document, include screenshots for the following:

1. a TensorBoard graph for the model, and

2. the loss curve.

Write up your analysis using a Word document. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_CTA4_Option_1_last_name_first_name.zip. Your paper should conform to the CSU Global Writing Center.

Option #2: Logistic Regression with TensorFlow

In this assignment, you will analyze the quality of a TensorFlow prediction by generating synthetic data.

1. Generate the synthetic data using the following Python code snippet.

```
# Generate synthetic data
N = 100
# Zeros form a Gaussian centered at (-1, -1)
x_zeros = np.random.multivariate_normal(
    mean=np.array((-1, -1)), cov=.1*np.eye(2), size=(N//2,))
y_zeros = np.zeros((N//2,))
# Ones form a Gaussian centered at (1, 1)
x_ones = np.random.multivariate_normal(
    mean=np.array((1, 1)), cov=.1*np.eye(2), size=(N//2,))
y_ones = np.ones((N//2,))

x_np = np.vstack([x_zeros, x_ones])
y_np = np.concatenate([y_zeros, y_ones])
```

2. Plot x_zeros and x_ones on the same graph.
3. Generate a TensorFlow graph.

```
with tf.name_scope("placeholders"):
  x = tf.compat.v1.placeholder(tf.float32, (N, 2))
  y = tf.compat.v1.placeholder(tf.float32, (N,))
with tf.name_scope("weights"):
  W = tf.Variable(tf.random.normal((2, 1)))
  b = tf.Variable(tf.random.normal((1,)))
with tf.name_scope("prediction"):
  y_logit = tf.squeeze(tf.matmul(x, W) + b)
  # the sigmoid gives the class probability of 1
  y_one_prob = tf.sigmoid(y_logit)
  # Rounding P(y=1) will give the correct prediction.
  y_pred = tf.round(y_one_prob)

with tf.name_scope("loss"):
  # Compute the cross-entropy term for each datapoint
  entropy =
tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit, labels=y)
  # Sum all contributions
  l = tf.reduce_sum(entropy)
```

```
with tf.name_scope("optim"):
  train_op = tf.compat.v1.train.AdamOptimizer(.01).minimize(l)

with tf.name_scope("summaries"):
  tf.summary.scalar("loss", l)
  merged = tf.summary.merge_all()

train_writer = tf.summary.FileWriter('logistic-train',
tf.get_default_graph())
```

4. Train the model, get the weights, and make predictions.
5. Plot the predicted outputs on top of the data.

For your deliverable, provide a detailed analysis using your screenshots as supporting content. Write up your analysis using a Word document. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_CTA_4_2_last_name_first_name.zip. Your paper should be a minimum of 2 pages in length and conform to the CSU Global Writing Center.

Portfolio Milestone (125 points)

Option #1: Implementing Facial Recognition

For this Portfolio Project Milestone, you will build on the programming requirements presented in Critical Thinking Assignment, Module 1, Option 1. In this Milestone, you will implement a Python program that uses facial recognition to determine if an individual face is present in a group of faces. For example, your program will use facial recognition to determine that the following individual:

is not in the group of people shown here:

Hints for implementing such a program are found in the following video:

LinkedIn Learning

Description: In the video, Deep Learning: Face Recognition (linked on your assignment page), you learn that face recognition is used for everything from automatically tagging pictures to unlocking cell phones. And with recent advancements in deep learning, the accuracy of face recognition has improved. In this course, learn how to develop a face recognition system that can detect faces in images, identify the faces, and even modify faces with "digital makeup" like you've experienced in popular mobile apps. Find out how to set up a development environment. Discover tools you can leverage for face recognition. See how a machine learning model can be trained to analyze images and identify facial landmarks. Learn the steps involved in coding facial feature detection, representing a face as a set of measurements, and encoding faces. Additionally, learn how to repurpose and adjust pre-existing systems.

Submit your Python code and image data using a zip file entitled:
CSC580_MidTermPortfolio_Option_1_last_name_first_name.zip.

Option #2: Improving TensorFlow Model Performance and Quality

In this Portfolio Project Milestone, you will improve the results from the work started in Module 3's Critical Thinking Assignment, Option 2. The derived models were not optimal and even present degradation after 100 epochs.

1. Update the `model.fit` call to automatically stop training when the validation score doesn't improve. Use an *EarlyStopping callback* that tests a training condition for every epoch. If a set amount of epochs elapses without showing improvement, then automatically stop the training. You can learn more about this callback by clicking on the link provided on your assignment page.

```
model = build_model()

# The patience parameter is the amount of epochs to check
for improvement
```

```
early_stop =
keras.callbacks.EarlyStopping(monitor='val_loss',
patience=10)

early_history = model.fit(normed_train_data, train_labels,
                    epochs=EPOCHS, validation_split = 0.2,
verbose=0,
                    callbacks=[early_stop,
tfdocs.modeling.EpochDots()])

plotter.plot({'Early Stopping': early_history}, metric =
"mae")
plt.ylim([0, 10])
plt.ylabel('MAE [MPG]')
```

2. Take a screenshot of the plot. What is the average error? Comment on the quality of this error.
3. Analyze how well the model generalizes by using the test set, which was not used when training the model. This tells us how well we can expect the model to predict when we use it in the real world.

```
loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=2)

print("Testing set Mean Abs Error: {:5.2f} MPG".format(mae))
```

4. Take a screenshot of the output.
5. Make predictions. Finally, predict MPG values using data in the testing set:

```
test_predictions = model.predict(normed_test_data).flatten()

a = plt.axes(aspect='equal')
plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
lims = [0, 50]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)
```

- Take a screenshot of the plot. Comment on the quality of the prediction.

6. Analyze the normality of the error distribution.

```
error = test_predictions - test_labels
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [MPG]")
_ = plt.ylabel("Count")
```

- Take a screenshot of the plot and comment on the distribution.

For your deliverable, provide a detailed analysis using your screenshots as supporting content. Write up your analysis using a Word document. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_MidTermPortfolio_Option_2_last_name_first_name.zip

Module 5

Readings

- Chapter 5 in *TensorFlow for Deep Learning*
- Zhang, X., Chen, X., Yao, L., Ge, C., & Dong, M. (2019). Deep neural network hyperparameter optimization with orthogonal array tuning. Working paper. arXiv database.

Discussion (25 points)

Critical Thinking (75 points)

Option #1: Improving the Accuracy of a Neural Network

In this assignment, you will improve the accuracy of the deep learning model of the Tox21 model from Critical Thinking Assignment, Module 4, Option 1.

1. Begin by loading the data.

```
import numpy as np
np.random.seed(456)
import matplotlib.pyplot as plt
import deepchem as dc
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

_, (train, valid, test), _ = dc.molnet.load_tox21()
train_X, train_y, train_w = train.X, train.y, train.w
valid_X, valid_y, valid_w = valid.X, valid.y, valid.w
test_X, test_y, test_w = test.X, test.y, test.w

# Remove extra tasks
train_y = train_y[:, 0]
valid_y = valid_y[:, 0]
test_y = test_y[:, 0]
train_w = train_w[:, 0]
valid_w = valid_w[:, 0]
test_w = test_w[:, 0]
```

2. Generate a TensorFlow graph using a random forest classifier.

```
# Generate tensorflow graph
sklearn_model = RandomForestClassifier(
    class_weight="balanced", n_estimators=50)
print("About to fit model on train set.")
sklearn_model.fit(train_X, train_y)

train_y_pred = sklearn_model.predict(train_X)
valid_y_pred = sklearn_model.predict(valid_X)
test_y_pred = sklearn_model.predict(test_X)

weighted_score = accuracy_score(train_y, train_y_pred,
sample_weight=train_w)
print("Weighted train Classification Accuracy: %f" %
weighted_score)
weighted_score = accuracy_score(valid_y, valid_y_pred,
sample_weight=valid_w)
print("Weighted valid Classification Accuracy: %f" %
weighted_score)
weighted_score = accuracy_score(test_y, test_y_pred,
sample_weight=test_w)
print("Weighted test Classification Accuracy: %f" %
weighted_score)
```

3. Investigate mapping hyperparameters to different Tox21 fully connected networks.

```
def eval_tox21_hyperparams(n_hidden=50, n_layers=1,
learning_rate=.001,
                           dropout_prob=0.5, n_epochs=45,
batch_size=100,
                           weight_positives=True):

  print("-----------------------------------------------")
  print("Model hyperparameters")
  print("n_hidden = %d" % n_hidden)
  print("n_layers = %d" % n_layers)
  print("learning_rate = %f" % learning_rate)
  print("n_epochs = %d" % n_epochs)
  print("batch_size = %d" % batch_size)
  print("weight_positives = %s" % str(weight_positives))
  print("dropout_prob = %f" % dropout_prob)
  print("-----------------------------------------------")

  d = 1024
  graph = tf.Graph()
```

```
  with graph.as_default():
    _, (train, valid, test), _ = dc.molnet.load_tox21()
    train_X, train_y, train_w = train.X, train.y, train.w
    valid_X, valid_y, valid_w = valid.X, valid.y, valid.w
    test_X, test_y, test_w = test.X, test.y, test.w

    # Remove extra tasks
    train_y = train_y[:, 0]
    valid_y = valid_y[:, 0]
    test_y = test_y[:, 0]
    train_w = train_w[:, 0]
    valid_w = valid_w[:, 0]
    test_w = test_w[:, 0]

    # Generate tensorflow graph
    with tf.name_scope("placeholders"):
      x = tf.placeholder(tf.float32, (None, d))
      y = tf.placeholder(tf.float32, (None,)) w
      = tf.placeholder(tf.float32, (None,))
      keep_prob = tf.placeholder(tf.float32)
    for layer in range(n_layers):
      with tf.name_scope("layer-%d" % layer):
        W = tf.Variable(tf.random_normal((d, n_hidden)))
        b = tf.Variable(tf.random_normal((n_hidden,)))
        x_hidden = tf.nn.relu(tf.matmul(x, W) + b)
        # Apply dropout
        x_hidden = tf.nn.dropout(x_hidden, keep_prob)
    with tf.name_scope("output"):
      W = tf.Variable(tf.random_normal((n_hidden, 1)))
      b = tf.Variable(tf.random_normal((1,)))
      y_logit = tf.matmul(x_hidden, W) + b
      # the sigmoid gives the class probability of 1
      y_one_prob = tf.sigmoid(y_logit)
      # Rounding P(y=1) will give the correct prediction.
      y_pred = tf.round(y_one_prob)
    with tf.name_scope("loss"):
      # Compute the cross-entropy term for each datapoint
      y_expand = tf.expand_dims(y, 1)
      entropy =
tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit,
labels=y_expand)
      # Multiply by weights
      if weight_positives:
        w_expand = tf.expand_dims(w, 1)
        entropy = w_expand * entropy
      # Sum all contributions
      l = tf.reduce_sum(entropy)
```

```
    with tf.name_scope("optim"):
      train_op =
tf.train.AdamOptimizer(learning_rate).minimize(l)

    with tf.name_scope("summaries"):
      tf.summary.scalar("loss", l)
      merged = tf.summary.merge_all()

    hyperparam_str = "d-%d-hidden-%d-lr-%f-n_epochs-%d-
batch_size-%d-weight_pos-%s" % (
        d, n_hidden, learning_rate, n_epochs, batch_size,
str(weight_positives))
    train_writer = tf.summary.FileWriter('/tmp/fcnet-func-' +
hyperparam_str,
                                         tf.get_default_graph())
    N = train_X.shape[0]
    with tf.Session() as sess:
      sess.run(tf.global_variables_initializer())
      step = 0
      for epoch in range(n_epochs):
        pos = 0
        while pos < N:
          batch_X = train_X[pos:pos+batch_size]
          batch_y = train_y[pos:pos+batch_size]
          batch_w = train_w[pos:pos+batch_size]
          feed_dict = {x: batch_X, y: batch_y, w: batch_w,
keep_prob: dropout_prob}
          _, summary, loss = sess.run([train_op, merged, l],
feed_dict=feed_dict)
          print("epoch %d, step %d, loss: %f" % (epoch, step,
loss))
          train_writer.add_summary(summary, step)

          step += 1
          pos += batch_size

      # Make Predictions (set keep_prob to 1.0 for predictions)
      valid_y_pred = sess.run(y_pred, feed_dict={x: valid_X,
keep_prob: 1.0})

    weighted_score = accuracy_score(valid_y, valid_y_pred,
sample_weight=valid_w)
    print("Valid Weighted Classification Accuracy: %f" %
weighted_score)
  return weighted_score
```

4. For each hyperparameter, select a list of values that might be good for the model weights. Write a nested for loop that tries all combinations of these values to find their validation accuracies and keep track of the best performers. However, there is one subtlety in the process: Deep networks can be fairly sensitive to the choice of random seed used to initialize the network. For this reason, it's worth repeating each choice of hyperparameter settings multiple times and averaging the results to dampen the variance.

For your deliverable, provide a detailed analysis of your model tuning. Write up your analysis using a Word document in which you explain the tuning choices you applied and the best model. Format your paper according to the guidelines in the CSU Global Writing Center. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_CTA5_Option_1_last_name_first_name.zip

Option #2: Building a Random Forest Classifier

In this assignment, you will use the iris dataset (linked on your assignment page) to classify iris plants based on measurements of their petal widths and sepal lengths. The dataset contains four variables measuring various parts of iris flowers of three related species and a fourth variable with the species name.

1. Load the data.

```
# Load the library with the iris dataset
from sklearn.datasets import load_iris

# Load scikit's random forest classifier library
from sklearn.ensemble import RandomForestClassifier

# Load pandas
import pandas as pd

# Load numpy
import numpy as np

# Set random seed
np.random.seed(0)
# Create an object called iris with the iris data
iris = load_iris()

# Create a dataframe with the four feature variables
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# View the top 5 rows
df.head()
# Add a new column with the species names; this is what we are going to try to predict
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

```
# View the top 5 rows
df.head()
```

- Make a screenshot of the head of the dataset.

2. Create training and test data.

```
# Create a new column that, for each row, generates a random number between 0 and 1,
and
# if that value is less than or equal to .75, then sets the value of that cell as True
# and false otherwise. This is a quick and dirty way of randomly assigning some rows to
# be used as the training data and some as the test data.
df['is_train'] = np.random.uniform(0, 1, len(df)) <= .75

# View the top 5 rows
df.head()
# Create two new dataframes, one with the training rows and one with the test rows
train, test = df[df['is_train']==True], df[df['is_train']==False]
# Show the number of observations for the test and training dataframes
print('Number of observations in the training data:', len(train))
print('Number of observations in the test data:',len(test))
```

- Make a screenshot of the outputs.

3. Preprocess the data.

```
# Create a list of the feature column's names
features = df.columns[:4]

# View features
features
# train['species'] contains the actual species names. Before we can use it,
# we need to convert each species name into a digit. So, in this case, there
# are three species, which have been coded as 0, 1, or 2.
y = pd.factorize(train['species'])[0]

# View target
y
```

- Make a screenshot of the outputs.

4. Train the random forest classifier.

```
# Create a random forest Classifier. By convention, clf means 'Classifier'
clf = RandomForestClassifier(n_jobs=2, random_state=0)

# Train the Classifier to take the training features and learn how they relate
# to the training y (the species)
clf.fit(train[features], y)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto',
max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2,
min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=2, oob_score=False,
random_state=0,
            verbose=0, warm_start=False)
```

5. Apply the classifier to the test data and make a screenshot of the predicted probabilities of the first 10 observations.

```
# Apply the Classifier we trained to the test data (which, remember, it has never seen
before)
clf.predict(test[features])
```

6. Evaluate the classifier by comparing the predicted and actual species for the first five observations.
7. Create a confusion matrix and use it to interpret the classification method.
   - Supply a screenshot of the confusion matrix.

```
# Create confusion matrix
pd.crosstab(test['species'], preds, rownames=['Actual Species'], colnames=['Predicted
Species'])
```

8. View the list of features and their importance scores.

```
# View a list of the features and their importance scores
list(zip(train[features], clf.feature_importances_))
```

For your deliverable, provide a detailed analysis of your model classifier. Write up your analysis using a Word document in which you explain your classifier method and model results. Format your paper according to the guidelines in the CSU Global Writing Center. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_CTA5_Option_2_last_name_first_name.zip

Module 6

Readings

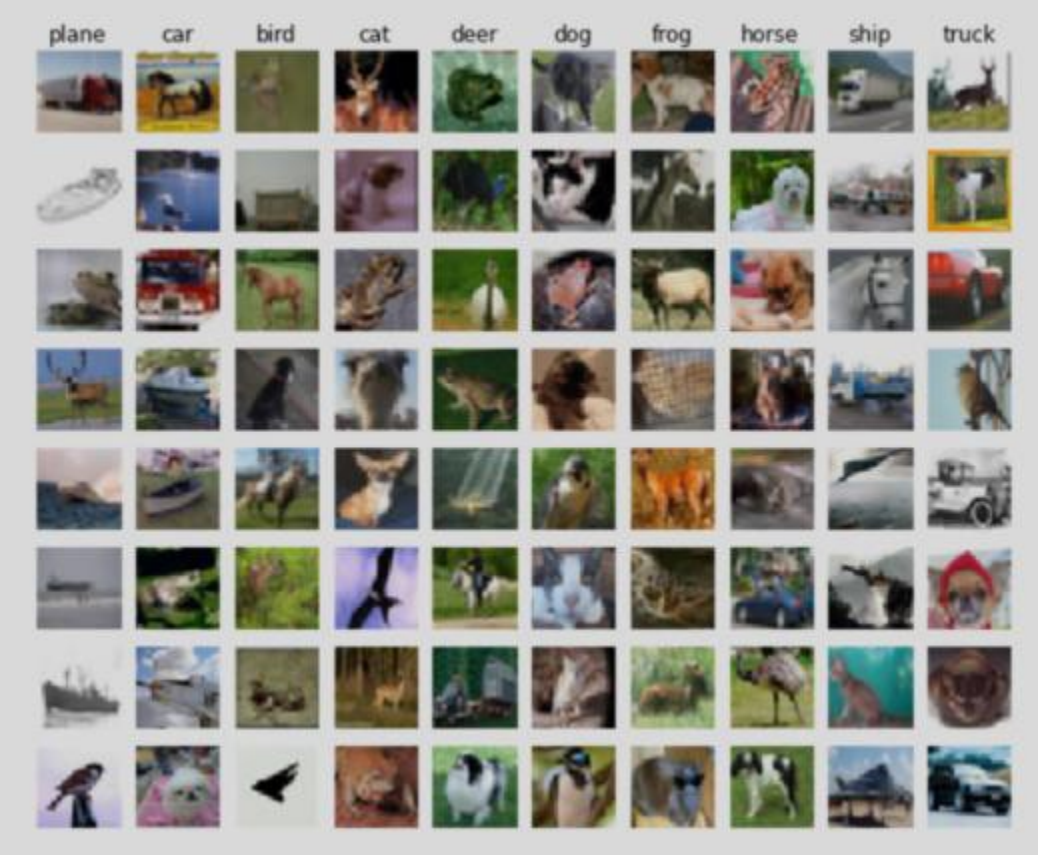
- Chapter 6 in *TensorFlow for Deep Learning*
- Zhang, Z., Zhou, X., Zhang, X., Wang, L., & Wang, P. (2018, August 6). A model based on convolutional neural network for online transaction fraud detection. *Security & Communication Networks*, 1–9.

Discussion (25 points)

Critical Thinking (75 points)

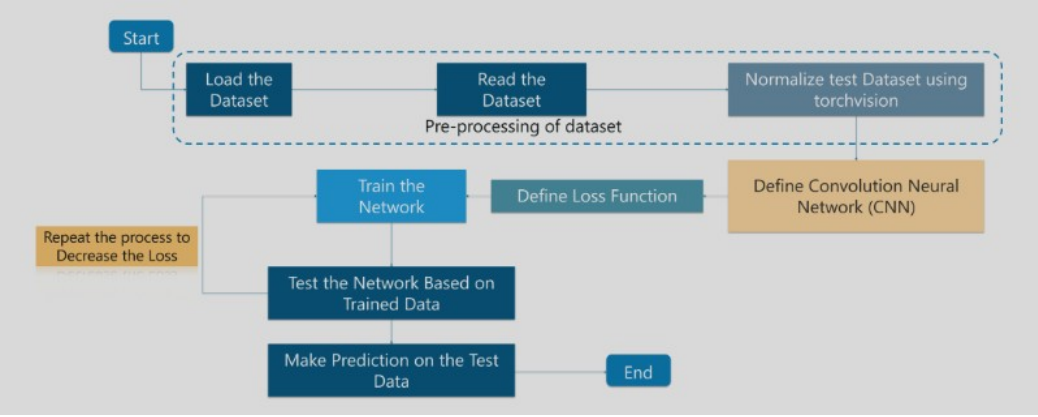
Option #1: Implementation of CIFAR10 with CNNs Using TensorFlow

For this assignment, you will train a network to classify images from the CIFAR dataset (linked on your assignment page) using a Convolutional Neural Network (CNN) built in TensorFlow. An example image follows:



(Source: Rao, 2019, para. 16)

Use the following flowchart to develop your solution:



(Source: Rao, 2019, para. 16)

In a Word document, provide an analysis of the veracity of your model. Format your paper according to the guidelines in the CSU Global Writing Center. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_ CTA6_Option_1_last_name_first_name.zip

References:

Rao, A. (2019, May 20). *Convolutional neural network tutorial (CNN) – Developing an image classifier in Python using TensorFlow*. https://www.edureka.co/blog/convolutional-neural-network/

Option #2: Distinguishing Dogs and Cats

In this assignment, you will train a Convolutional Neural Network (CNN) to predict whether an image contains a dog or a cat. To do this, use Kaggle's cats and dogs Dataset (linked on your assignment page). (Click on "Data" at the top of the homepage.) It contains 12,500 pictures of cats and 12,500 of dogs, with different resolutions.

Step 1: Load and preprocess the image data with NumPy.

```
import tensorflow as tf
import seaborn as sns
import numpy as np

from PIL import Image
import glob
from collections import defaultdict
from tensorflow import keras
from tensorflow.keras import layers
```

The most common shape size is 375×500. Divide this by four for your network's input.

Use this for the image loading code.

```
IMG_SIZE = (94, 125)
def pixels_from_path(file_path):
    im = Image.open(file_path)

    im = im.resize(IMG_SIZE)
    np_im = np.array(im)
    #matrix of pixel RGB values
    return np_im

shape_counts = defaultdict(int)
for i, cat in enumerate(glob.glob('cats/*')[:1000]):
    if i%100==0:
        print(i)
    img_shape = pixels_from_path(cat).shape
    shape_counts[str(img_shape)]= shape_counts[str(img_shape)]+ 1

shape_items = list(shape_counts.items())
shape_items.sort(key = lambda x: x[1])
shape_items.reverse()

# 10% of the data will automatically be used for validation
validation_size = 0.1
img_size = IMG_SIZE # resize images to be 374x500 (most common
shape)
```

```
num_channels = 3 # RGB
sample_size = 8192 #We'll use 8192 pictures (2**13)

pixels_from_path(glob.glob('cats/*')[5]).shape

SAMPLE_SIZE = 2048
print("loading training cat images...")
cat_train_set = np.asarray([pixels_from_path(cat) for cat in
glob.glob('cats/*')[:SAMPLE_SIZE]])
print("loading training dog images...")
dog_train_set = np.asarray([pixels_from_path(dog) for dog in
glob.glob('dogs/*')[:SAMPLE_SIZE]])

valid_size = 512
print("loading validation cat images...")
cat_valid_set = np.asarray([pixels_from_path(cat) for cat in
glob.glob('cats/*')[-valid_size:]])
print("loading validation dog images...")
dog_valid_set = np.asarray([pixels_from_path(dog) for dog in
glob.glob('dogs/*')[-valid_size:]])

x_train = np.concatenate([cat_train_set, dog_train_set])
labels_train = np.asarray([1 for _ in range(SAMPLE_SIZE)]+[0 for
_ in range(SAMPLE_SIZE)]) x_valid =
np.concatenate([cat_valid_set, dog_valid_set])
labels_valid = np.asarray([1 for _ in range(valid_size)]+[0 for _
in range(valid_size)])
```

The following code implements a CNN using a single hidden layer.

```
from tensorflow import keras
from tensorflow.keras import layers

total_pixels = img_size[0] *img_size[1] * 3
fc_size = 512

inputs = keras.Input(shape=(img_size[1], img_size[0],3),
name='ani_image')
x = layers.Flatten(name = 'flattened_img')(inputs) #turn image to
vector.

x = layers.Dense(fc_size, activation='relu',
name='first_layer')(x)
outputs = layers.Dense(1, activation='sigmoid', name='class')(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

Step 2: Do the following for this model:

- Use the AdamOptimizer,
- Use 10 epochs,
- Shuffle the training data, and
- Use MSE as the loss function.

```
customAdam = keras.optimizers.Adam(lr=0.001)
model.compile(optimizer=customAdam,  # Optimizer
              # Loss function to minimize
              loss="mean_squared_error",
              # List of metrics to monitor

metrics=["binary_crossentropy","mean_squared_error"])

print('# Fit model on training data')

history = model.fit(x_train,
                    labels_train,
                    batch_size=32,
                    shuffle = True,
                #important since we loaded cats first, dogs
second.
                    epochs=3,
                    validation_data=(x_valid, labels_valid))

#Train on 4096 samples, validate on 2048 samples
#loss: 0.5000 - binary_crossentropy: 8.0590 - mean_squared_error:
0.5000 - val_loss: 0.5000 - val_binary_crossentropy: 8.0591 -
val_mean_squared_error: 0.5000
```

Step 3: Train the CNN. Use the following configuration for the network:

- One single convolution layer with 24 kernels,
- Two fully connected layers,
- Max pooling,
- Measure the Pearson correlation between predictions and validation labels.

```
fc_layer_size = 128
img_size = IMG_SIZE

conv_inputs = keras.Input(shape=(img_size[1], img_size[0],3),
name='ani_image')
conv_layer = layers.Conv2D(24, kernel_size=3,
activation='relu')(conv_inputs)
conv_layer = layers.MaxPool2D(pool_size=(2,2))(conv_layer)
conv_x = layers.Flatten(name = 'flattened_features')(conv_layer)
#turn image to vector.
```

```
conv_x = layers.Dense(fc_layer_size, activation='relu',
name='first_layer')(conv_x)
conv_x = layers.Dense(fc_layer_size, activation='relu',
name='second_layer')(conv_x)
conv_outputs = layers.Dense(1, activation='sigmoid',
name='class')(conv_x)

conv_model = keras.Model(inputs=conv_inputs,
outputs=conv_outputs)

customAdam = keras.optimizers.Adam(lr=1e-6)
conv_model.compile(optimizer=customAdam,  # Optimizer
              # Loss function to minimize
              loss="binary_crossentropy",
              # List of metrics to monitor

metrics=["binary_crossentropy","mean_squared_error"])

#Epoch 5/5 loss: 1.6900 val_loss: 2.0413 val_mean_squared_error:
0.3688
print('# Fit model on training data')

history = conv_model.fit(x_train,
                    labels_train, #we pass it th labels
                    #If the model is taking too long to train,
make this bigger
                    #If it is taking too long to load for the
first epoch, make this smaller
                    batch_size=32,
                    shuffle = True,
                    epochs=5,
                    # We pass it validation data to
                    # monitor loss and metrics
                    # at the end of each epoch
                    validation_data=(x_valid, labels_valid))

preds = conv_model.predict(x_valid)
preds = np.asarray([pred[0] for pred in preds])
np.corrcoef(preds, labels_valid)[0][1] # 0.15292172
```

Step 4: Perform the following analysis:

1. Modify the model by adding another convolutional layer and use 48 kernels. What is the new correlation coefficient?
2. Assess the accuracy of the model.

```
sns.scatterplot(x= preds, y= labels_valid)
```

```
cat_quantity = sum(labels_valid)

for i in range(1,10):
    print('threshold :'+str(.1*i))
    print(sum(labels_valid[preds > .1*i])/labels_valid[preds >
.1*i].shape[0])
```

Step 5: Interpret the scatterplot of the summary result.

Use the following utility functions to select an image and report the probability of the image being a cat.

```
def animal_pic(index):
    return Image.fromarray(x_valid[index])
def cat_index(index):
    return conv_model.predict(np.asarray([x_valid[124]]))[0][0]
```

Step 6: Save the model.

```
conv_model.save('conv_model_big')

An example output would be:
index = 600
print("probability of being a cat: {}".format(cat_index(index)))
animal_pic(index)
```

probability of being a cat: 0.046173080801963806

(Source: Kaggle.com. Retrieved from https://www.kaggle.com/c/dogs-vs-cats)

Step 7: Write an interface to your model that allows the user to pick an index and prints out the image and the probability of the image being one of a cat. In a Word document, provide an analysis of the veracity of your model. Suggest ways of improving your model. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_ CTA6_Option_2_last_name_first_name.zip.

Format your paper according to the guidelines in the CSU Global Writing Center.

Module 7

Readings

- Chapter 7 in *TensorFlow for Deep Learning*
- Sherstinsky, A. (2018, November 4). *Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network*. https://arxiv.org/pdf/1808.03314.pdf
- Yu, A. (2019, February 26). *Getting a machine to do my English homework for me: Using recurrent neural networks for text generation*. https://towardsdatascience.com/getting-a-machine-to-do-my-english-homework-for-me-5d339470fe42

Discussion (25 points)

Program Completer Survey (25)

> Congratulations on reaching your degree Capstone! At CSU Global, we are committed to continually improving our programs and creating a culture that values quality. Because you are nearing the end of your degree program, you are in a unique position to provide valuable feedback concerning your experiences at CSU Global.

We use student feedback to assist in making improvements to our programs and student services. Your feedback is valuable; these results are reviewed each trimester and shared with multiple departments across the university.

Please complete and submit the Program Completion Survey (Links to an external site.) (Links to an external site.). This survey replaces the Module 7 Discussion and is worth 25 points. Begin the survey by clicking on the "Program Completion Survey" link above. Be sure to complete and submit the entire survey.

In order to receive the points associated with the assignment:

1. Click the "Done" button after completing the final question to submit your survey results.
2. Click "Print Screen" on the completion screen to which you are taken, which will contain a thank you message.
3. Save a copy of the screenshot with the thank you message.
4. Submit a copy of the screenshot to the Module 7 assignments area in order to receive 25 points.

Module 8

Readings

- Chapter 8 in *Tensorflow for Deep Learning*
- Huang, D. Z., Xu, K., Farhat, C., & Darve, E. (2019). Learning constitutive relations from indirect observations using deep neural networks. ArXiv database; ArXiv: 1905.12530.

Discussion (25 points)

Portfolio Project (225 points)

Option #1: Working with a Generative Adversarial Network

Part 1 (Research Write-Up):
Research and analyze the use of generative adversarial networks in industry and in applications. Your use cases should be uniquely distinct and should cover multiple areas of industry. Ensure that your paper meets the following guidelines:

- Identify at least 4 pertinent use cases in which this model is used and the benefit for using it in each.
- Your paper should be a minimum of 4 pages and include at least 4 scholarly references in APA format. Ensure that your assignment is formatted according to the CSU Global Writing Center.

Part 2 (Programming Implementation):
Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. It was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made up of two competing neural network models, which are able to analyze, capture, and copy the variations within a dataset.

Generative Adversarial Networks (GANs) can be broken down into three parts:

1. Generative: To learn a generative model, which describes how data is generated in terms of a probabilistic model.
2. Adversarial: The training of a model is done in an adversarial setting.
3. Networks: Use of deep neural networks as the artificial intelligence (AI) algorithms for training purposes.

In GANs, there is a **Generator** and a **Discriminator**. The Generator generates fake samples of data (an image, audio, etc.) and tries to fool the Discriminator. The Discriminator, on the other hand, tries to distinguish between the real and fake samples. The Generator and the Discriminator are both neural networks, and they both run in competition with each other in the training phase. The steps are repeated several times, and after each repetition, the Generator and Discriminator get better and better in their respective jobs.

Training a GAN has two parts:

Part 1: The Discriminator is trained while the Generator is idle. In this phase, the network is only forward propagated and no back-propagation is done. The Discriminator is trained on real data for n epochs to see if it can correctly predict them as real. Also, in this phase, the Discriminator is trained on the fake generated data from the Generator to see if it can correctly predict them as fake.

Part 2: The Generator is trained while the Discriminator is idle. After the Discriminator is trained by the generated fake data of the Generator, we can get its predictions and use the results to train the Generator and improve from the previous state to try and fool the Discriminator.

The above method is repeated for a few epochs, and then, we manually check the fake data if it seems genuine. If it seems acceptable, then the training is stopped; otherwise, it's allowed to continue for a few more epochs.

Sample Python code implementing a Generative Adversarial Network:

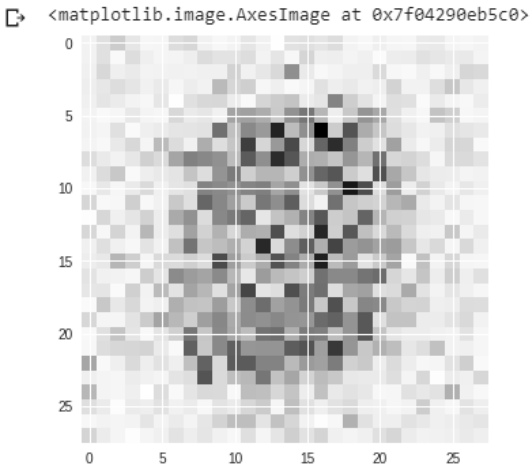
```
# importing the necessary libraries and the MNIST dataset
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data")

# defining functions for the two networks.
# Both the networks have two hidden layers
# and an output layer, which are densely or
# fully connected layers defining the
# Generator network function
def generator(z, reuse = None):
     with tf.variable_scope('gen', reuse = reuse):
          hidden1 = tf.layers.dense(inputs = z, units = 128,
```

```
                                        activation =
tf.nn.leaky_relu)

            hidden2 = tf.layers.dense(inputs = hidden1,
            units = 128, activation = tf.nn.leaky_relu)

            output = tf.layers.dense(inputs = hidden2,
                  units = 784, activation = tf.nn.tanh)

            return output

# defining the Discriminator network function
def discriminator(X, reuse = None):
      with tf.variable_scope('dis', reuse = reuse):
            hidden1 = tf.layers.dense(inputs = X, units = 128,
                                        activation =
tf.nn.leaky_relu)

            hidden2 = tf.layers.dense(inputs = hidden1,
                  units = 128, activation = tf.nn.leaky_relu)

            logits = tf.layers.dense(hidden2, units = 1)
            output = tf.sigmoid(logits)

            return output, logits

# creating placeholders for the outputs
tf.reset_default_graph()

real_images = tf.placeholder(tf.float32, shape =[None, 784])
z = tf.placeholder(tf.float32, shape =[None, 100])

G = generator(z)
D_output_real, D_logits_real = discriminator(real_images)
D_output_fake, D_logits_fake = discriminator(G, reuse = True)

# defining the loss function
def loss_func(logits_in, labels_in):
      return
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
                                    logits = logits_in, labels =
labels_in))

# Smoothing for generalization
D_real_loss = loss_func(D_logits_real,
tf.ones_like(D_logits_real)*0.9)
```

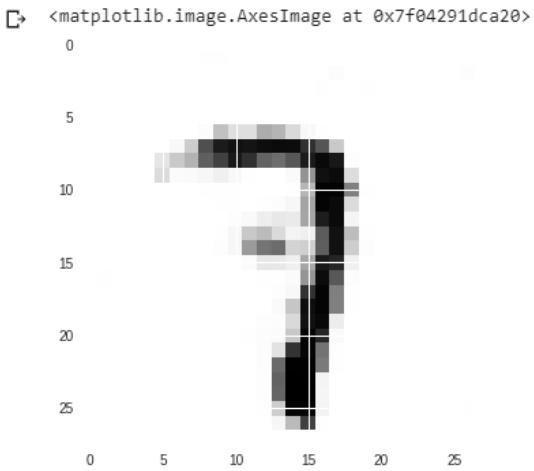
```
D_fake_loss = loss_func(D_logits_fake,
tf.zeros_like(D_logits_real))
D_loss = D_real_loss + D_fake_loss

G_loss = loss_func(D_logits_fake, tf.ones_like(D_logits_fake))

# defining the learning rate, batch size, and
# number of epochs and using the Adam optimizer
lr = 0.001 # learning rate

# Do this when multiple networks
# interact with each other

# returns all variables created(the two
# variable scopes) and makes trainable true
tvars = tf.trainable_variables()
d_vars =[var for var in tvars if 'dis' in var.name]
g_vars =[var for var in tvars if 'gen' in var.name]

D_trainer = tf.train.AdamOptimizer(lr).minimize(D_loss, var_list
= d_vars)
G_trainer = tf.train.AdamOptimizer(lr).minimize(G_loss, var_list
= g_vars)

batch_size = 100 # batch size
epochs = 500 # number of epochs. The higher the better the result
init = tf.global_variables_initializer()

# creating a session to train the networks
samples =[] # generator examples

with tf.Session() as sess:
      sess.run(init)
      for epoch in range(epochs):
            num_batches = mnist.train.num_examples//batch_size

            for i in range(num_batches):
                  batch = mnist.train.next_batch(batch_size)
                  batch_images = batch[0].reshape((batch_size,
784))
                  batch_images = batch_images * 2-1
                  batch_z = np.random.uniform(-1, 1, size
=(batch_size, 100))
                  _= sess.run(D_trainer, feed_dict
={real_images:batch_images, z:batch_z})
                  _= sess.run(G_trainer, feed_dict ={z:batch_z})
```

```
            print("on epoch{}".format(epoch))

            sample_z = np.random.uniform(-1, 1, size =(1, 100))
            gen_sample = sess.run(generator(z, reuse = True),
                                  feed_dict
={z:sample_z})

            samples.append(gen_sample)

# result after 0th epoch
plt.imshow(samples[0].reshape(28, 28))

# result after 499th epoch
plt.imshow(samples[49].reshape(28, 28))
```

The result after the 0th epoch: (Note: The figures used in this problem specification were generated by the given Python code.)



Result after the 499th epoch:



For this final Portfolio Project, you will build a GAN using the Keras library. (Keras library is a wrapper for low-level TensorFlow commands, and you will import this as a Python library.) The dataset used is the CIFAR10 IMAGE DATASET, which is preloaded into Keras. You can read about the dataset by clicking on the link provided on your assignment page.

Step 1: Import the required Python libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.layers import Input, Dense, Reshape, Flatten, Dropout
from keras.layers import BatchNormalization, Activation,
ZeroPadding2D
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.models import Sequential, Model
from keras.optimizers import Adam,SGD
```

Step 2: Load the data.

```
#Loading the CIFAR10 data
(X, y), (_, _) = keras.datasets.cifar10.load_data()

#Selecting a single class of images
#The number was randomly chosen and any number
#between 1 and 10 can be chosen
X = X[y.flatten() == 8]
```

Step 3: Define parameters to be used in later processes.

```
#Defining the Input shape
image_shape = (32, 32, 3)

latent_dimensions = 100
```

Step 4: Define a utility function to build the generator.

```
def build_generator():

        model = Sequential()

        #Building the input layer
        model.add(Dense(128 * 8 * 8, activation="relu",
                        input_dim=latent_dimensions))
```

```
        model.add(Reshape((8, 8, 128)))

        model.add(UpSampling2D())

        model.add(Conv2D(128, kernel_size=3, padding="same"))
        model.add(BatchNormalization(momentum=0.78))
        model.add(Activation("relu"))

        model.add(UpSampling2D())

        model.add(Conv2D(64, kernel_size=3, padding="same"))
        model.add(BatchNormalization(momentum=0.78))
        model.add(Activation("relu"))

        model.add(Conv2D(3, kernel_size=3, padding="same"))
        model.add(Activation("tanh"))


        #Generating the output image
        noise = Input(shape=(latent_dimensions,))
        image = model(noise)

        return Model(noise, image)
```

Step 5: Define a utility function to build the discriminator.

```
def build_discriminator():

        #Building the convolutional layers
        #to classify whether an image is real or fake
        model = Sequential()

        model.add(Conv2D(32, kernel_size=3, strides=2,
                         input_shape=image_shape,
padding="same"))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))

        model.add(Conv2D(64, kernel_size=3, strides=2,
padding="same"))
        model.add(ZeroPadding2D(padding=((0,1),(0,1))))
        model.add(BatchNormalization(momentum=0.82))
        model.add(LeakyReLU(alpha=0.25))
        model.add(Dropout(0.25))
```

```
        model.add(Conv2D(128, kernel_size=3, strides=2,
padding="same"))
        model.add(BatchNormalization(momentum=0.82))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))

        model.add(Conv2D(256, kernel_size=3, strides=1,
padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(LeakyReLU(alpha=0.25))
        model.add(Dropout(0.25))

        #Building the output layer
        model.add(Flatten())
        model.add(Dense(1, activation='sigmoid'))

        image = Input(shape=image_shape)
        validity = model(image)

        return Model(image, validity)
```

Step 6: Define a utility function to display the generated images.

```
def display_images():
        r, c = 4,4
        noise = np.random.normal(0, 1, (r * c,latent_dimensions))
        generated_images = generator.predict(noise)

        #Scaling the generated images
        generated_images = 0.5 * generated_images + 0.5

        fig, axs = plt.subplots(r, c)
        count = 0
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(generated_images[count, :,:,])
                axs[i,j].axis('off')
                count += 1
        plt.show()
        plt.close()
```

Step 7: Build the GAN.

```
# Building and compiling the discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy',
                      optimizer=Adam(0.0002,0.5),
```

```
                    metrics=['accuracy'])

#Making the discriminator untrainable
#so that the generator can learn from fixed gradient
discriminator.trainable = False

# Building the generator
generator = build_generator()

#Defining the input for the generator
#and generating the images
z = Input(shape=(latent_dimensions,))
image = generator(z)


#Checking the validity of the generated image
valid = discriminator(image)

#Defining the combined model of the generator and the
discriminator
combined_network = Model(z, valid)
combined_network.compile(loss='binary_crossentropy',
                         optimizer=Adam(0.0002,0.5))
```

Step 8: Train the network.

```
num_epochs=15000
batch_size=32
display_interval=2500
losses=[]

#Normalizing the input
X = (X / 127.5) - 1.


#Defining the Adversarial ground truths
valid = np.ones((batch_size, 1))

#Adding some noise
valid += 0.05 * np.random.random(valid.shape)
fake = np.zeros((batch_size, 1))
fake += 0.05 * np.random.random(fake.shape)

for epoch in range(num_epochs):

            #Training the Discriminator
```

```
                #Sampling a random half of images
                index = np.random.randint(0, X.shape[0], batch_size)
                images = X[index]

                #Sampling noise and generating a batch of new images
                noise = np.random.normal(0, 1, (batch_size,
latent_dimensions))
                generated_images = generator.predict(noise)


                #Training the discriminator to detect more accurately
                #whether a generated image is real or fake
                discm_loss_real =
discriminator.train_on_batch(images, valid)
                discm_loss_fake =
discriminator.train_on_batch(generated_images, fake)
                discm_loss = 0.5 * np.add(discm_loss_real,
discm_loss_fake)

                #Training the generator

                #Training the generator to generate images
                #that pass the authenticity test
                genr_loss = combined_network.train_on_batch(noise,
valid)

                #Tracking the progress
                if epoch % display_interval == 0:
                     display_images()
```

Assignment Requirements:

1. Write Python code to plot the images from the first epoch. Take a screenshot of the images from the first epoch.
2. Write Python code to plot the images from the last epoch. Take a screenshot of the images from the last epoch.
3. Comment on the network performance.

For your deliverable, provide a detailed analysis using your screenshots as supporting content. Write up your analysis in a Word document. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_FinalPortfolio_Option_1_last_name_first_name.zip

Option #2: Encoder-Decoder Model for Sequence-to-Sequence Prediction

Part 1 (Research Write up):
Research and analyze the use of encoder-decoder models in industry and in applications. Your use cases should be uniquely distinct and should cover multiple areas of industry. Ensure that your paper meets the following guidelines:

- Identify at least 4 pertinent use cases in which this model is used and the benefit for using it in each.
- Your paper should be a minimum of 4 pages and include at least 4 scholarly references in APA format. Ensure that your assignment is formatted according to the CSU Global Writing Center.

Part 2 (Programming Implementation):
For this Portfolio Project assignment, you will develop an encoder-decoder model for sequence-to-sequence prediction using Keras. (Keras library is a wrapper for low-level TensorFlow commands, and you will import this as a Python library.) The encoder-decoder model provides a pattern for using recurrent neural networks to address challenging sequence-to-sequence prediction problems such as machine translation.

Encoder-decoder models can be developed in the Keras Python deep learning library. For this assignment, you will develop a sophisticated encoder-decoder recurrent neural network for a sequence-to-sequence prediction problem with Keras.

There are three Python-based programming tasks for this Portfolio Project. The three parts are:

1. Encoder-Decoder Model in Keras
2. Scalable Sequence-to-Sequence Problem
3. Encoder-Decoder LSTM for Sequence Prediction.

For this project, you will need the following: (Note: These are free Python packages, add-ons, and you should have no problem downloading these.)

- Python 3
- Scikit-learn
- Pandas
- NumPy
- Keras
- TensorFlow.

Encoder-Decoder Model in Keras
The encoder-decoder model is a way of organizing recurrent neural networks for sequence-to-sequence prediction problems. The approach involves two recurrent neural networks: one to encode the source sequence, called the encoder, and a second to decode the encoded source sequence into the target sequence, called the decoder.

Use the following code example as a starting point to define an encoder-decoder recurrent neural network. Below is this function named *define_models()*.

```
# returns train, inference_encoder and inference_decoder models
def define_models(n_input, n_output, n_units):
	# define training encoder
	encoder_inputs = Input(shape=(None, n_input))
	encoder = LSTM(n_units, return_state=True)
	encoder_outputs, state_h, state_c = encoder(encoder_inputs)
	encoder_states = [state_h, state_c]
	# define training decoder
	decoder_inputs = Input(shape=(None, n_output))
	decoder_lstm = LSTM(n_units, return_sequences=True,
return_state=True)
	decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
initial_state=encoder_states)
	decoder_dense = Dense(n_output, activation='softmax')
	decoder_outputs = decoder_dense(decoder_outputs)
	model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
	# define inference encoder
	encoder_model = Model(encoder_inputs, encoder_states)
	# define inference decoder
	decoder_state_input_h = Input(shape=(n_units,))
	decoder_state_input_c = Input(shape=(n_units,))
	decoder_states_inputs = [decoder_state_input_h,
decoder_state_input_c]
	decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs,
initial_state=decoder_states_inputs)
	decoder_states = [state_h, state_c]
	decoder_outputs = decoder_dense(decoder_outputs)
	decoder_model = Model([decoder_inputs] + decoder_states_inputs,
[decoder_outputs] + decoder_states)
	# return all models
	return model, encoder_model, decoder_model
```

The function takes three arguments, as follows:

- n_input: The cardinality of the input sequence, i.e., number of features, words, or characters for each time step.
- n_output: The cardinality of the output sequence, i.e., number of features, words, or characters for each time step.
- n_units: The number of cells to create in the encoder and decoder models, e.g., 128 or 256.

The function then creates and returns three models, as follows:

- train: Model that can be trained if given source, target, and shifted target sequences.
- inference_encoder: Encoder model used when making a prediction for a new source sequence.
- inference_decoder: Decoder model used when making a prediction for a new source sequence.

The model is trained when given source and target sequences where the model takes both the source and a shifted version of the target sequence as input and predicts the whole target sequence.

For example, one source sequence may be [1,2,3] and the target sequence [4,5,6]. The inputs and outputs to the model during training would be:

```
Input1: ['1', '2', '3']
Input2: ['_', '4', '5']
Output: ['4', '5', '6']
```

The model is intended to be called recursively when generating target sequences for new source sequences.

The source sequence is encoded and the target sequence is generated one element at a time, using a "start of sequence" character such as '_' to start the process. Therefore, in the above case, the following input-output pairs would occur during training:

```
t,   Input1,     Input2,          Output
1,  ['1', '2', '3'],  '_',       '4'
2,  ['1', '2', '3'],  '4',       '5'
3,  ['1', '2', '3'],  '5',       '6'
```

Here, you can see how the recursive use of the model can be used to build up output sequences.

During prediction, the *inference_encoder* model is used to encode the input sequence once, which returns states that are used to initialize the *inference_decoder* model. From that point, the *inference_decoder* model is used to generate predictions step by step.

The function, below, named *predict_sequence()* can be used after the model is trained to generate a target sequence given a source sequence.

```
# generate target given source sequence
def predict_sequence(infenc, infdec, source, n_steps, cardinality):
	# encode
	state = infenc.predict(source)
	# start of sequence input
	target_seq = array([0.0 for _ in range(cardinality)]).reshape(1,
1, cardinality)
	# collect predictions
	output = list()
	for t in range(n_steps):
		# predict next char
		yhat, h, c = infdec.predict([target_seq] + state)
		# store prediction
		output.append(yhat[0,0,:])
		# update state
		state = [h, c]
		# update target sequence
		target_seq = yhat
```

```
        return array(output)
```

This function takes five arguments as follows:

1. infenc: Encoder model used when making a prediction for a new source sequence.
2. infdec: Decoder model used when making a prediction for a new source sequence.
3. source: Encoded source sequence.
4. n_steps: Number of time steps in the target sequence.
5. cardinality: The cardinality of the output sequence, i.e., the number of features, words, or characters for each time step.

The function then returns a list containing the target sequence.

Scalable Sequence-to-Sequence Problem

In this section, a contrived and scalable sequence-to-sequence prediction problem is presented for this final Portfolio option. The source sequence is a series of randomly generated integer values, such as [20, 36, 40, 10, 34, 28], and the target sequence is a reversed pre-defined subset of the input sequence, such as the first three elements in reverse order [40, 36, 20]. The length of the source sequence is configurable; so is the cardinality of the input and output sequence and the length of the target sequence. For this Portfolio Project option, you will use source sequences of six elements, a cardinality of 50, and target sequences of three elements.

Below are some more examples to make this concrete.

```
Source,                     Target
[13, 28, 18, 7, 9, 5]       [18, 28, 13]
[29, 44, 38, 15, 26, 22]                [38, 44, 29]
[27, 40, 31, 29, 32, 1]           [31, 40, 27]
...
```

Start off by defining a function to generate a sequence of random integers. Use the value of 0 as the padding or start-of-sequence character; therefore, it is reserved, and you cannot use it in your source sequences. To achieve this, add 1 to your configured cardinality to ensure the one-hot encoding is large enough (i.e., a value of 1 maps to a '1' value in index 1).

For example:

```
n_features = 50 + 1
```

You can use the *randint()* Python function to generate random integers in a range between 1 and 1-minus the size of the problem's cardinality.

The generate_sequence() below generates a sequence of random integers.

```
# generate a sequence of random integers
def generate_sequence(length, n_unique):
```

```
return [randint(1, n_unique-1) for _ in range(length)]
```

Next, you need to create the corresponding output sequence given the source sequence. To keep things simple, select the first n elements of the source sequence as the target sequence and reverse them.

```
# define target sequence
target = source[:n_out]
target.reverse()
```

You also need a version of the output sequence, shifted forward by one time step, that you can use as the mock target generated so far, including the start-of-sequence value in the first time step. You can create this from the target sequence directly.

```
# create padded input target sequence
target_in = [0] + target[:-1]
```

Now that all of the sequences have been defined, you can one-hot encode them, i.e., transform them into sequences of binary vectors. You can use the Keras built in *to_categorical()* function to achieve this. You can put all of this into a function named *get_dataset()* that will generate a specific number of sequences that we can use to train a model.

```
# prepare data for the LSTM
def get_dataset(n_in, n_out, cardinality, n_samples):
	X1, X2, y = list(), list(), list()
	for _ in range(n_samples):
		# generate source sequence
		source = generate_sequence(n_in, cardinality)
		# define target sequence
		target = source[:n_out]
		target.reverse()
		# create padded input target sequence
		target_in = [0] + target[:-1]
		# encode
		src_encoded = to_categorical([source],
num_classes=cardinality)
		tar_encoded = to_categorical([target],
num_classes=cardinality)
		tar2_encoded = to_categorical([target_in],
num_classes=cardinality)
		# store
		X1.append(src_encoded)
		X2.append(tar2_encoded)
		y.append(tar_encoded)
	return array(X1), array(X2), array(y)
```

Finally, you need to be able to decode a one-hot encoded sequence to make it readable again. This is needed for both printing the generated target sequences and for easily comparing whether the full predicted target sequence matches the expected target sequence. The *one_hot_decode()* function will decode an encoded sequence.

```
# decode a one hot encoded string
def one_hot_decode(encoded_seq):
	return [argmax(vector) for vector in encoded_seq]
```

Evaluate the given construction using the following code:

```
# configure problem
n_features = 50 + 1
n_steps_in = 6
n_steps_out = 3
# generate a single source and target sequence
X1, X2, y = get_dataset(n_steps_in, n_steps_out, n_features, 1)
print(X1.shape, X2.shape, y.shape)
print('X1=%s, X2=%s, y=%s' % (one_hot_decode(X1[0]), one_hot_decode(X2[0]), one_hot_decode(y[0])))
```

Encoder-Decoder LSTM for Sequence Prediction

You are now ready to apply the encoder-decoder LSTM model to a sequence-to-sequence prediction problem.

1. Configure the model:

```
# configure problem
n_features = 50 + 1
n_steps_in = 6
n_steps_out = 3
```

2. Define the models and compile the training model.

```
# define model
train, infenc, infdec = define_models(n_features, n_features, 128)
train.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['acc'])
```

3. Evaluate the model. Do this by making predictions for 100 source sequences and counting the number of target sequences that were predicted correctly.

```
# evaluate LSTM
total, correct = 100, 0
for _ in range(total):
	X1, X2, y = get_dataset(n_steps_in, n_steps_out, n_features, 1)
	target = predict_sequence(infenc, infdec, X1, n_steps_out,
n_features)
```

```
        if array_equal(one_hot_decode(y[0]), one_hot_decode(target)):
            correct += 1
    print('Accuracy: %.2f%%' % (float(correct)/float(total)*100.0))
```

4. Finally, generate some predictions and print the decoded source, target, and predicted target sequences to get an idea of whether the model works as expected. An example output would be something like the following:

```
X=[22, 17, 23, 5, 29, 11] y=[23, 17, 22], yhat=[23, 17, 22]
X=[28, 2, 46, 12, 21, 6] y=[46, 2, 28], yhat=[46, 2, 28]
X=[12, 20, 45, 28, 18, 42] y=[45, 20, 12], yhat=[45, 20, 12]
X=[3, 43, 45, 4, 33, 27] y=[45, 43, 3], yhat=[45, 43, 3]
X=[34, 50, 21, 20, 11, 6] y=[21, 50, 34], yhat=[21, 50, 34]
X=[47, 42, 14, 2, 31, 6] y=[14, 42, 47], yhat=[14, 42, 47]
X=[20, 24, 34, 31, 37, 25] y=[34, 24, 20], yhat=[34, 24, 20]
X=[4, 35, 15, 14, 47, 33] y=[15, 35, 4], yhat=[15, 35, 4]
X=[20, 28, 21, 39, 5, 25] y=[21, 28, 20], yhat=[21, 28, 20]
X=[50, 38, 17, 25, 31, 48] y=[17, 38, 50], yhat=[17, 38, 50]
```

Implement the Python solution for this encoder-decoder specification. Include the following items in your deliverable:

1. a comprehensive flowchart depicting the processing performed by your neural network model,
2. the Python source file that is thoroughly documented, and
3. an output text file showing the run-time predictions of your model.

For your deliverable, provide a detailed analysis using your screenshots as supporting content. Write up your analysis using a Word document. Submit your Python code and Word document in a zip archive file. Name your archive file: CSC580_FinalPortfolio _Option_2_last_name_first_name.zip

COURSE POLICIES

Grading Scale	
A	95.0 – 100
A-	90.0 – 94.9
B+	86.7 – 89.9
B	83.3 – 86.6
B-	80.0 – 83.2
C+	75.0 – 79.9
C	70.0 – 74.9
D	60.0 – 69.9
F	59.9 or below

Graduate Course Grading	
20%	Discussion Participation
45%	Critical Thinking Assignments
35%	Final Portfolio Project

IN-CLASSROOM POLICIES

For information on late work and incomplete grade policies, please refer to our **In-Classroom Student Policies and Guidelines** or the Academic Catalog for comprehensive documentation of CSU Global institutional policies.

Academic Integrity
Students must assume responsibility for maintaining honesty in all work submitted for credit and in any other work designated by the instructor of the course. Academic dishonesty includes cheating, fabrication, facilitating academic dishonesty, plagiarism, reusing /re-purposing your own work (see CSU Global Library page for Citing & APA Resources and Avoiding Common Plagiarism Mistakes for percentage of repurposed work that can be used in an assignment), unauthorized possession of academic materials, and unauthorized collaboration. The CSU Global Library provides information on how students can avoid plagiarism by understanding what it is and how to use the Library and Internet resources.

Citing Sources with APA Style
All students are expected to follow APA format for all assignments. For details, please review the APA guidelines within the CSU Global Writing Center. A link to this resource should be provided within most assignment descriptions in your course.

Disability Services Statement
CSU Global is committed to providing reasonable accommodations for all persons with disabilities. Any student with a documented disability requesting academic accommodations should contact the Disability Resource Coordinator at 720-279-0650 and/or email ada@CSUGlobal.edu for additional information to coordinate reasonable accommodations for students with documented disabilities.

Netiquette
Respect the diversity of opinions among the instructor and classmates and engage with them in a courteous, respectful, and professional manner. All posts and classroom communication must be conducted in accordance with the student code of conduct. Think before you push the Send button. Did you say just what you meant? How will the person on the other end read the words?

Maintain an environment free of harassment, stalking, threats, abuse, insults or humiliation toward the instructor and classmates. This includes, but is not limited to, demeaning written or oral comments of an ethnic, religious, age, disability, sexist (or sexual orientation), or racist nature; and the unwanted sexual advances or intimidations by email, or on discussion boards and other postings within or connected to the online classroom. If you have concerns about something that has been said, please let your instructor know.