Bill Yerkes

CS5542 Big Data Apps and Analytics

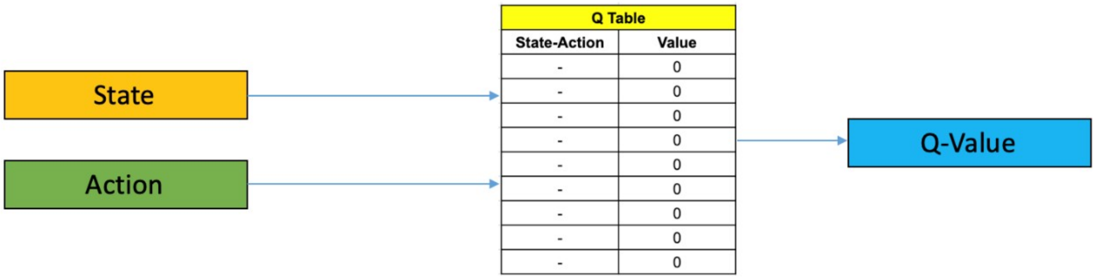
In Class Programming –10
29th October 2020
Due Date: 11/3/2020 (Tuesday by 11:59pm)

Submit ICP Feedback in Class. : [Lnik to Feed back Form]

Deep Q-Learning:

Implementing Deep Q-Learning in Python using Keras & OpenAI Gym:

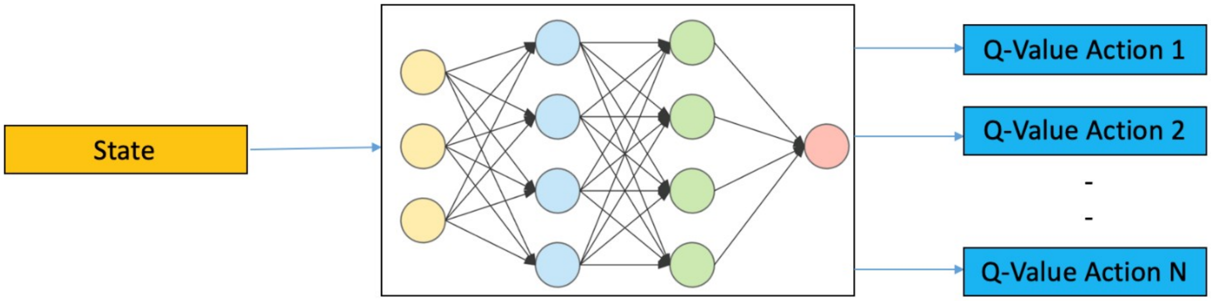
In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. The comparison between Q-learning & deep Q-learning is illustrated below:



Q Table

State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q Learning



Deep Q Learning

CartPole is one of the simplest environments in the OpenAI gym (a game simulator). The idea of CartPole is that there is a pole standing up on top of a cart. The goal is to balance this pole by moving the cart from side to side to keep the pole balanced upright.

Design a Deep Q learning Network (DQN), using Keras & OpenAI Gym , for cartpole game and visualize your results.

ICP Requirements:

1) Designing a DQN for cartpole game in python using Keras & OpenAI Gym (70 points)
2) Visualization of DQN cartpole game (10 points)
3) overall code quality (10 points)
4) Pdf Report quality, video explanation (10 points)

Submission Guidelines:

Same as previous ICPs.

ICP Report:

What I learned in the ICP:

I learned how to use Deep Learning to play and solve games. I learned some about the Gym library and how that is use for game play and deep learning. I learned how to display the cart pole environment.

Description of what task I was performing:

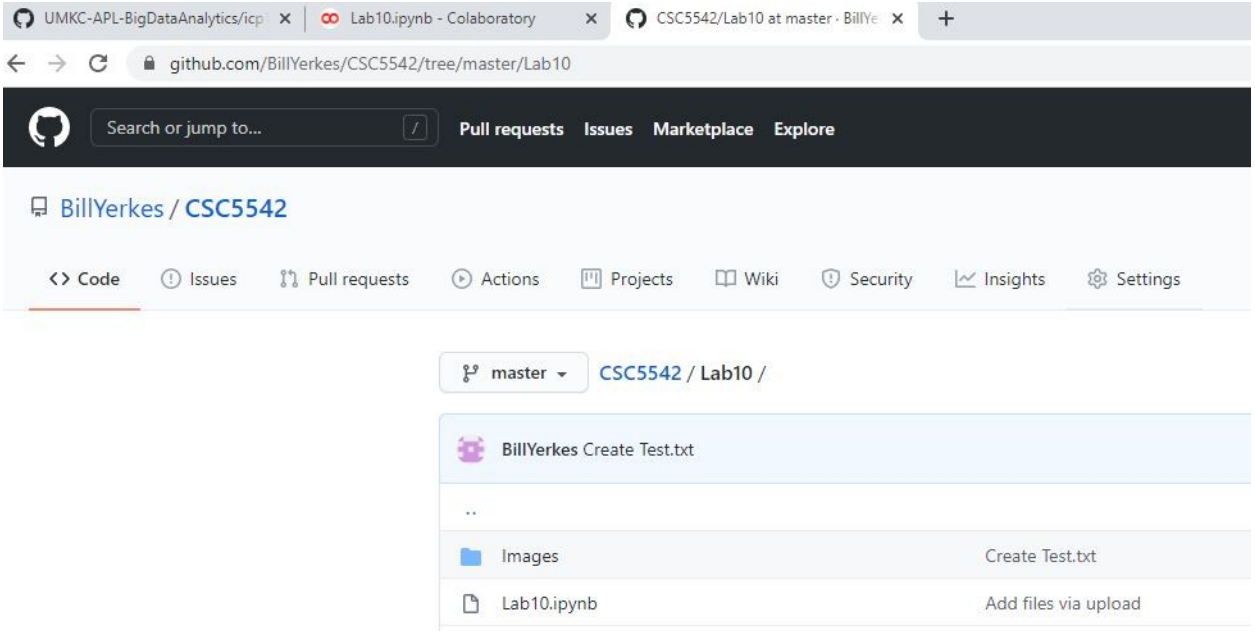
Design a deep learning solution to play the cart pole game and render it on the screen.

Challenges I faced:

One of the hardest parts was trying to figure out how to display the cart pole.

Screen Shots

GitHub:



BillYerkes Create Test.txt

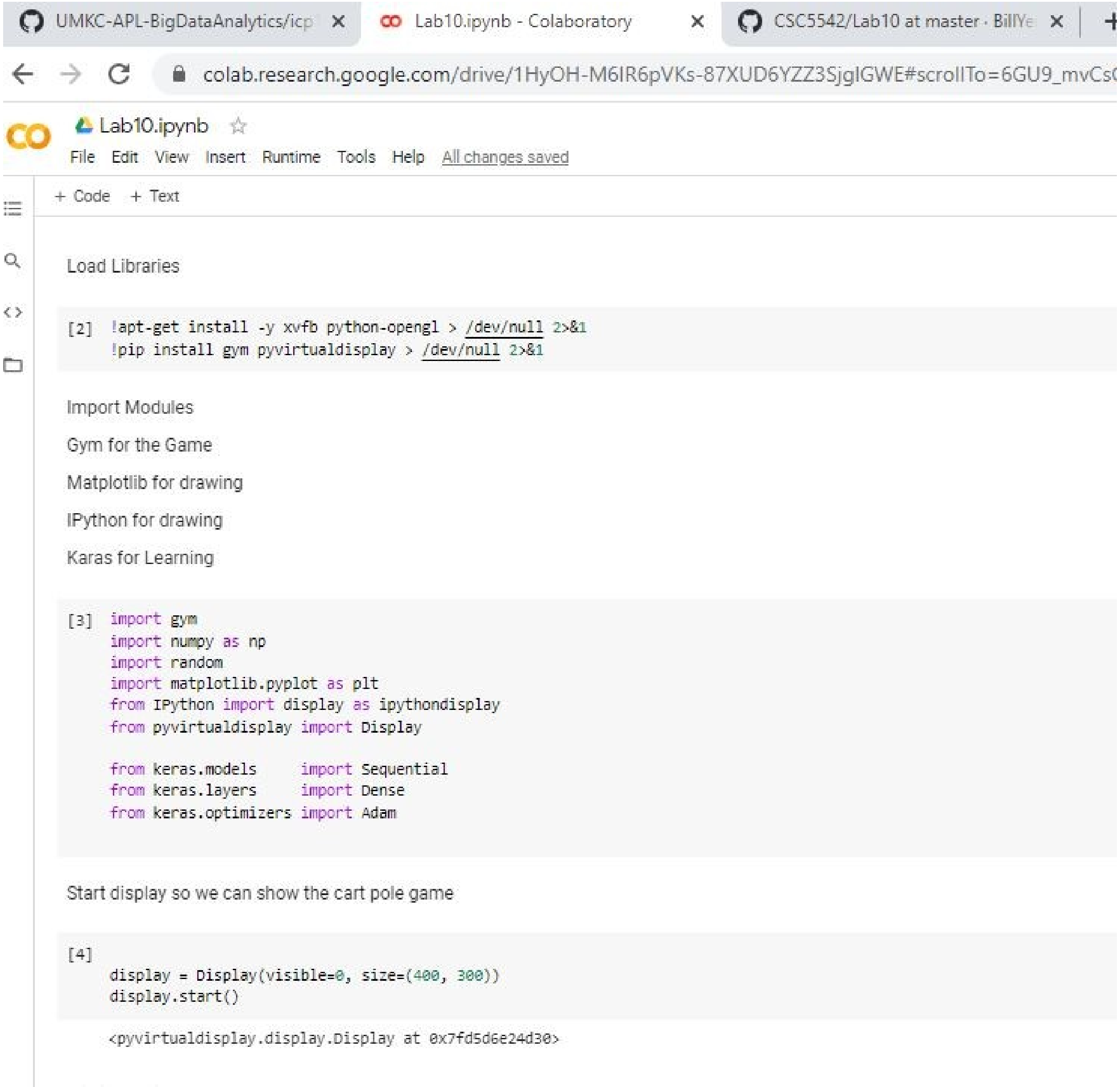
..

Images	Create Test.txt
Lab10.ipynb	Add files via upload

Load Libraries and Import Modules

Loading libraries to be able to display cart pole game.

Importing needed modules to create, play, track, and display the game.



UMKC-APL-BigDataAnalytics/icp
Lab10.ipynb - Colaboratory
CSC5542/Lab10 at master · BillYe
colab.research.google.com/drive/1HyOH-M6IR6pVKs-87XUD6YZZ3SjgIGWE#scrollTo=6GU9_mvCs
Lab10.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

Load Libraries

```
[2] !apt-get install -y xvfb python-opengl > /dev/null 2>&1
    !pip install gym pyvirtualdisplay > /dev/null 2>&1
```

Import Modules

Gym for the Game

Matplotlib for drawing

IPython for drawing

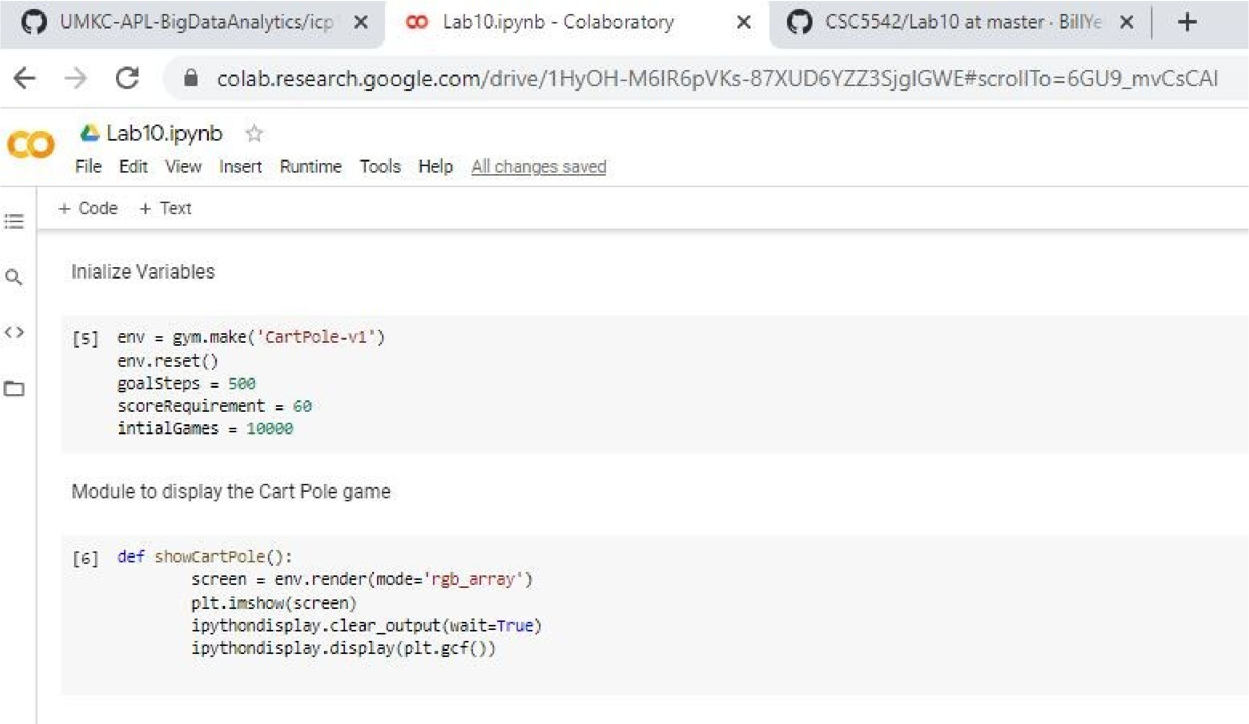
Karas for Learning

```
[3] import gym
    import numpy as np
    import random
    import matplotlib.pyplot as plt
    from IPython import display as ipythondisplay
    from pyvirtualdisplay import Display

    from keras.models     import Sequential
    from keras.layers     import Dense
    from keras.optimizers import Adam
```

Start display so we can show the cart pole game

```
[4]
    display = Display(visible=0, size=(400, 300))
    display.start()

    <pyvirtualdisplay.display.Display at 0x7fd5d6e24d30>
```

Setting up global variables for the game and creating module to display the game.



UMKC-APL-BigDataAnalytics/icp
Lab10.ipynb - Colaboratory
CSC5542/Lab10 at master · BillYe
colab.research.google.com/drive/1HyOH-M6IR6pVKs-87XUD6YZZ3SjgIGWE#scrollTo=6GU9_mvCsCAI
Lab10.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

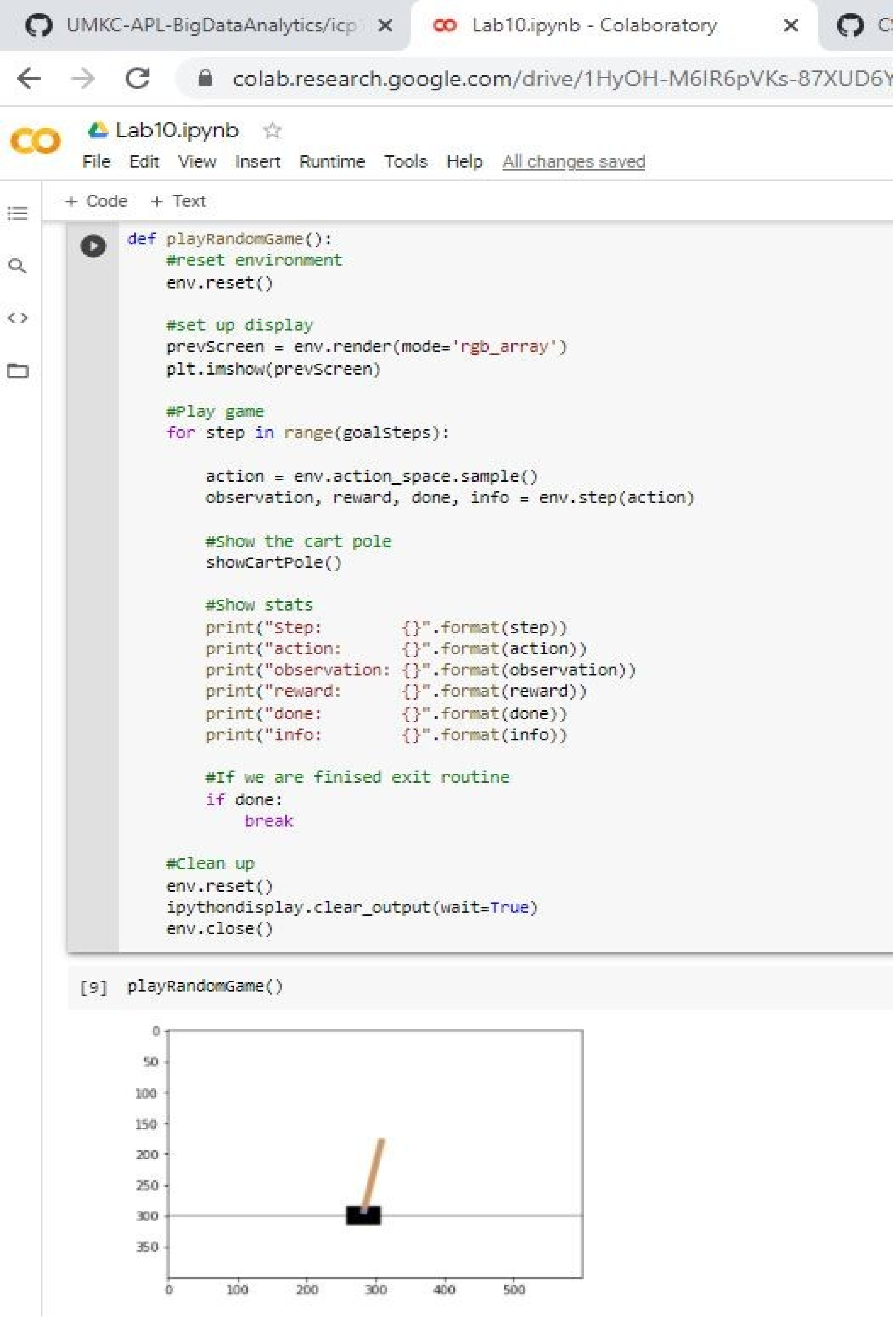
Inialize Variables

```
[5] env = gym.make('CartPole-v1')
    env.reset()
    goalSteps = 500
    scoreRequirement = 60
    intialGames = 10000
```

Module to display the Cart Pole game

```
[6] def showCartPole():
            screen = env.render(mode='rgb_array')
            plt.imshow(screen)
            ipythondisplay.clear_output(wait=True)
            ipythondisplay.display(plt.gcf())
```

Module to play random game:

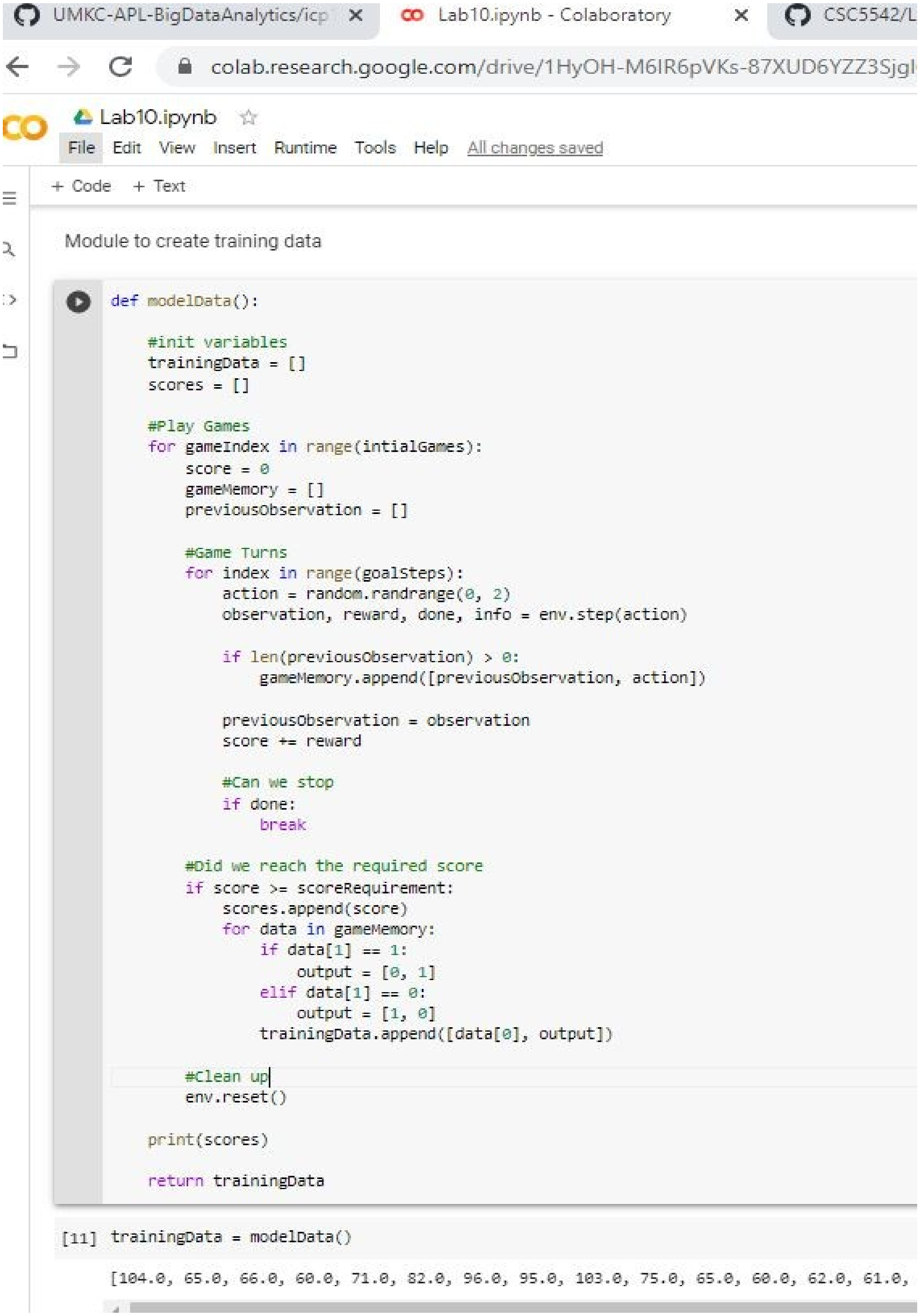


UMKC-APL-BigDataAnalytics/icp
Lab10.ipynb - Colaboratory
colab.research.google.com/drive/1HyOH-M6IR6pVKs-87XUD6Y
Lab10.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes saved
+ Code
+ Text

```
def playRandomGame():
    #reset environment
    env.reset()

    #set up display
    prevScreen = env.render(mode='rgb_array')
    plt.imshow(prevScreen)

    #Play game
    for step in range(goalSteps):

        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)

        #Show the cart pole
        showCartPole()

        #Show stats
        print("Step:        {}".format(step))
        print("action:      {}".format(action))
        print("observation: {}".format(observation))
        print("reward:      {}".format(reward))
        print("done:        {}".format(done))
        print("info:        {}".format(info))

        #If we are finised exit routine
        if done:
            break

    #Clean up
    env.reset()
    ipythondisplay.clear_output(wait=True)
    env.close()
```

```
[9]  playRandomGame()
```

Create the training data:


UMKC-APL-BigDataAnalytics/icp
Lab10.ipynb - Colaboratory
CSC5542/L
colab.research.google.com/drive/1HyOH-M6IR6pVKs-87XUD6YZZ3Sjgl
Lab10.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

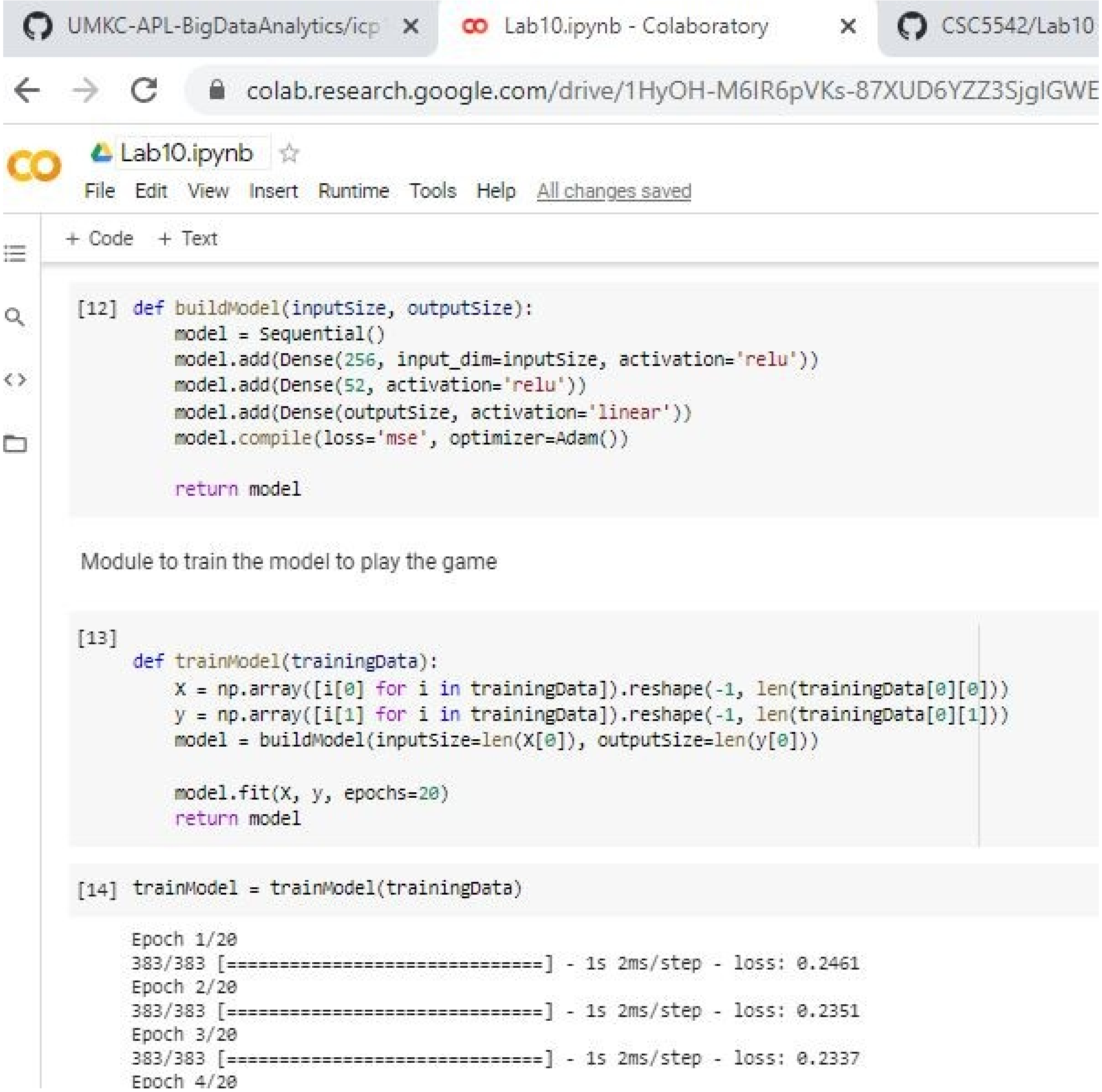
Module to create training data

```
def modelData():

    #init variables
    trainingData = []
    scores = []

    #Play Games
    for gameIndex in range(intialGames):
        score = 0
        gameMemory = []
        previousObservation = []

        #Game Turns
        for index in range(goalSteps):
            action = random.randrange(0, 2)
            observation, reward, done, info = env.step(action)

            if len(previousObservation) > 0:
                gameMemory.append([previousObservation, action])

            previousObservation = observation
            score += reward

            #Can we stop
            if done:
                break

        #Did we reach the required score
        if score >= scoreRequirement:
            scores.append(score)
            for data in gameMemory:
                if data[1] == 1:
                    output = [0, 1]
                elif data[1] == 0:
                    output = [1, 0]
                trainingData.append([data[0], output])

        #Clean up
        env.reset()

    print(scores)

    return trainingData
```

```
[11] trainingData = modelData()
```

```
[104.0, 65.0, 66.0, 60.0, 71.0, 82.0, 96.0, 95.0, 103.0, 75.0, 65.0, 60.0, 62.0, 61.0,
```

Build and Train the model on how to play the game:



UMKC-APL-BigDataAnalytics/icp
Lab10.ipynb - Colaboratory
CSC5542/Lab10
colab.research.google.com/drive/1HyOH-M6IR6pVKs-87XUD6YZZ3SjgIGWE
Lab10.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
[12] def buildModel(inputSize, outputSize):
         model = Sequential()
         model.add(Dense(256, input_dim=inputSize, activation='relu'))
         model.add(Dense(52, activation='relu'))
         model.add(Dense(outputSize, activation='linear'))
         model.compile(loss='mse', optimizer=Adam())

         return model
```

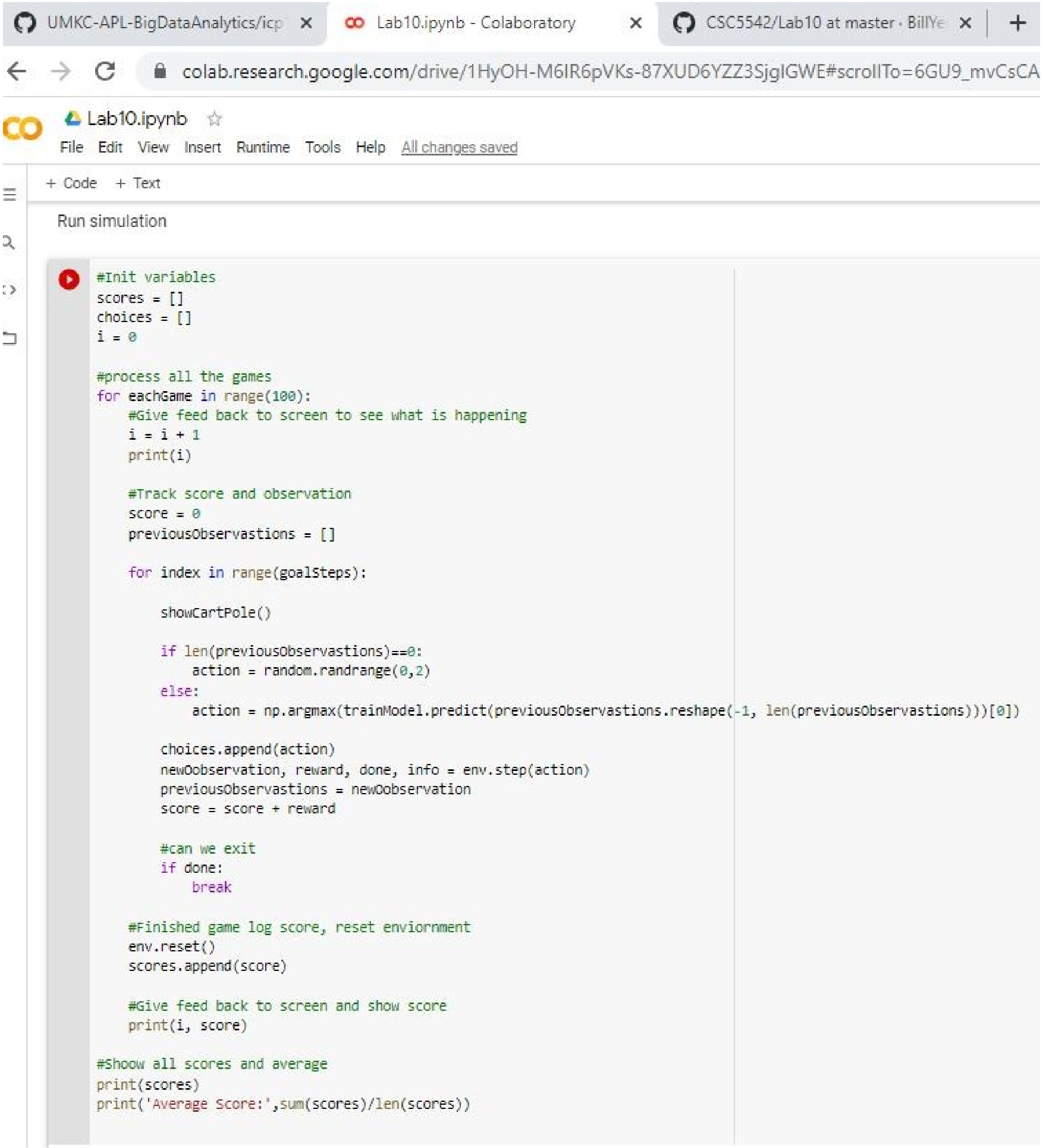
Module to train the model to play the game

```
[13]
     def trainModel(trainingData):
         X = np.array([i[0] for i in trainingData]).reshape(-1, len(trainingData[0][0]))
         y = np.array([i[1] for i in trainingData]).reshape(-1, len(trainingData[0][1]))
         model = buildModel(inputSize=len(X[0]), outputSize=len(y[0]))

         model.fit(X, y, epochs=20)
         return model
```

```
[14] trainModel = trainModel(trainingData)
```

```
Epoch 1/20
383/383 [==============================] - 1s 2ms/step - loss: 0.2461
Epoch 2/20
383/383 [==============================] - 1s 2ms/step - loss: 0.2351
Epoch 3/20
383/383 [==============================] - 1s 2ms/step - loss: 0.2337
Epoch 4/20
```

Run the simulation:


UMKC-APL-BigDataAnalytics/icp
Lab10.ipynb - Colaboratory
CSC5542/Lab10 at master · BillYe
colab.research.google.com/drive/1HyOH-M6IR6pVKs-87XUD6YZZ3SjgIGWE#scrollTo=6GU9_mvCsCA
Lab10.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes saved
+ Code
+ Text

Run simulation

```
#Init variables
scores = []
choices = []
i = 0

#process all the games
for eachGame in range(100):
    #Give feed back to screen to see what is happening
    i = i + 1
    print(i)

    #Track score and observation
    score = 0
    previousObservastions = []

    for index in range(goalSteps):

        showCartPole()

        if len(previousObservastions)==0:
            action = random.randrange(0,2)
        else:
            action = np.argmax(trainModel.predict(previousObservastions.reshape(-1, len(previousObservastions)))[0])

        choices.append(action)
        newOobservation, reward, done, info = env.step(action)
        previousObservastions = newOobservation
        score = score + reward

        #can we exit
        if done:
            break

    #Finished game log score, reset enviornment
    env.reset()
    scores.append(score)

    #Give feed back to screen and show score
    print(i, score)

#Shoow all scores and average
print(scores)
print('Average Score:',sum(scores)/len(scores))
```

Video Link

Any in site about the data or the ICP in general

Would like to do more with this type of programming.