CS520 Project 4: The Imitation Game

Daniel Ying (dty16) Sec 198:520:01
Zachary Tarman (zpt2) Sec 198:520:01
Pravin Kumaar (pr482) Sec 198:520:03

December 17, 2021

Submitter: Daniel Ying

Honor Code:

I abide to the rules laid in the Project 4: Imitation Game description and I have not used anyone else's work for the project, and my work is only my own and my group's.

I acknowledge and accept the Honor Code and rules of Project 4.

Signed: Daniel Ying (dty16), Zachary Tarman (zpt2), Pravin Kumaar (pr482)

Workload:

Daniel Ying: collected data from project 1 and 2 (both in Java language), converted the data into python, reorganized the data to be processed by numpy, constructed the model builder, and trained and tested the model.

Zachary Tarman: assisted in editing existing agent 3 for data collection of knowledge bases, compiled various members' thoughts into answers to project questions.

Pravin Kumaar: implemented the model traversing in a maze, assisted with tuning the model

Together: Brainstormed and discussed the general procedure of processing data, training, testing, and implementation of project 4. Discussed problems in the assignment and code.

Challenges We Faced:

For our projects 1 and 2, we implemented the agents in Java. Unfortunately, though there are programs similar to tensorflow in Java, but they are all either in Beta version or have bugs the developers have yet to resolve.

Thus, we had to collect raw data from our Java projects 1 and 2, store them in text files. Then have python read them and convert them into matrices with numpy. We tried to use Jupyter to run tensorflow, but the program crashed each time we tried.

To work around this, we used Google's Colab to upload our data and begin converting the data into matrices for tensorflow to run. Another problem arose, Colab crashed when we uploaded data for 101x101 mazes that contains 3000 runs which proved that the data size was too large for Colab to handle.

If we decrease the number of maze runs, we would have too little data to train, thus we dialed down the data size to 50x50 mazes with 5000 runs. At first we successfully uploaded our data onto Colab, but when we tried to convert the data to matrices, Colab crashed again.

To resolve this, we bought the Colab subscription and upgraded our Colab to Colab Pro. We then decreased our maze dimensions to 10x10. Fortunately Colab Pro did not crash and with the 10x10 dimension mazes, we were able to increase our maze runs to 15000 for the agent 1 imitator and 10000 for the agent 3 imitator, allowing us to train and test our models.

Problem 1

How should the state space (current information) and action space (action selected) be represented for the model? How does it capture the relevant information, in a relevant way, for your model space? One thing to consider here is local vs global information.

ANSWER:

Project 1 (Agent 1)

To start off, we had to find a way to feed the neural network relevant information about how the agent made decisions based on the state of the agent/board and its knowledge of the board. We had to simulate what the agent "sees" at every given point in time while traversing the maze, and we had to put it into a representable form for the NN to learn from.

We knew that the agents never had global information about the maze (i.e. the agent didn't know the location of every block in the initial state, but it would discover them as it went along). We had to find a way to simulate the "nebula" like the professor discussed. Thankfully, the way our knowledge base was organized and how we updated it, we were able to simulate this fairly well (if we had run into a block, we would make sure that it made it into the data given to the NN, but otherwise, it was an "unknown" cell).

Essentially, our input space was modeled just like the structure of the maze itself (a matrix of dimension rows by columns), and for each "cell", we gave a corresponding value for the knowledge we had about the cell. So, 0 would mean an unknown or visited grid, 1 means the location of the agent, 2 means the location of the goal, and -1 represented the location of known blocks in the gridworld (represented by "X" upon data export from the Java program and then later translated to the necessary integer value in Python before training).

From there, we had to decide on our output space, but this was relatively simple. We represented the output space as a vector corresponding to the different directions the agent could take. For example, [1, 0, 0, 0] would correspond to an up movement, [0, 1, 0, 0] would correspond to a down movement, [0, 0, 1, 0] for left, and [0, 0, 0, 1] for right.

Project 2 (Agent 3):

Much like for the Project 1 neural networks, we had a similar process for extracting data from the agent while traversing the maze. The only difference here is that we also had to decide how to represent the inferences that the agent made and train the NN accordingly (we'll call this the inference knowledge base). We were initially unsure about how to do this since some of these inferences are quite extensive, but we eventually decided that all of the inferences stemmed from the number of blocks sensed around the agent in a particular visited cell. There were other things that went into the inferences as well, but this information was encoded into the knowledge base passed to the Agent 1's NN (which we'll call the general knowledge base from now on).

Essentially, our input space for this project included both the general knowledge base from the first project part with Agent 1, and then the inference knowledge base modeled in an identical structure (a matrix of dimension rows by columns). Each cell corresponded to the number of neighbors sensed to be blocked around that cell (0-8, and "X" for an unvisited cell). In this way, we also simulate the nebula of unvisited cells rather than giving the NN all of the global information. The action space / output space was identical to Agent 1's corresponding space.

The reshaping of the data was a little different than the first part of the project, so we'll discuss that a little bit here. Since we had to take the inference / partial sensing abilities of Agent 3, we had more data in our input space. We concatenated the two knowledge bases data (the general knowledge base and the inference knowledge base) into one and reshaped it as (num of maze grids, 2, dimension, dimension). We then flattened the newly concatenated array, during the buildModel() function. Then, the flattened array would undergo the process similar to the dense architecture.

Problem 2

How are you defining your loss function when training your model?

ANSWER:

Project 1 (Agent 1):

We defined our optimizer as the Adam optimization algorithm, which is an optimizer that can be user to update the networks weights iteratively based on the training data (refer below).

model.compile(optimizer = 'adam', loss = 'categorical-crossentropy', metrics = ['accuracy'])

Adam is different from the classic stochastic gradient descent, which maintains a single learning rate for all weight updates and it doesn't change during training. Instead of only adapting the parameter learning rate based on the average first moment, Adam also adapts the average of the second moments of the gradients (uncentered variance).

We set out to define our loss function as categorical cross-entropy. This loss function is suitable for multi-class classification models where there are two or more output labels, which in our case we have 4 different labels for the 4 different directions the agents can travel in. The output label would be assigned as one-hot category, encoding values in the form of 0 and 1s. This output label (as integer form) is converted into categorical encoding using keras.

Lastly, the metrics we used is 'accuracy', which according to keras, we would be dealing with BinaryAccurary, CategoricalAccuracy, and SparseCategoricalAccuracy. The accuracy would be converted based on the loss function and the model output shape. Thus, in our case, the accuracy would be converted to the CategoricalAccuracy form.

Project 2 (Agent 3):

Refer to the first part of the project's Question 2. We defined loss functions and all other associated features in the same way.

Problem 3

In training, how many episodes on how many different gridworlds were necessary to get good performance of your model on the training data?

ANSWER:

Project 1 (Agent 1):

Only Full Dense Layers:

For agent 1, we wanted to make sure that we generated enough data that it informed the NN of how to behave in a number of different situations. We ended up generating 15000 gridworlds, exporting the necessary input space and corresponding output space at every step the agent took along its traversal. We thought this would provide a wide variety of different situations for the NN to learn from and adjust its model accordingly.

From there, we determined that the number of epochs that allowed us to find the optimal performance of the model on the training data was 3 epochs.

The result: Epoch 1/3 26983/26983 [==============================] - 80s 3ms/step - loss: 0.1386 - accuracy: 0.9425 - valloss: 0.1437 - valaccuracy: 0.9416

Epoch 2/3 26983/26983 [==============================] - 88s 3ms/step - loss: 0.1337 - accuracy: 0.9449 - valloss: 0.1355 - valaccuracy: 0.9443

Epoch 3/3 26983/26983 [==============================] - 89s 3ms/step - loss: 0.1295 - accuracy: 0.9465 - valloss: 0.1343 - valaccuracy: 0.9460

At Least One Convolutional Neural Layer:

Similar to the dense layer architecture, we generated 15000 different gridworlds for training this NN. From here, we determined that the appropriate epoch number was only 2, and this acquired the following metrics:

Epoch 1/2 26983/26983 [==============================] - 87s 3ms/step - loss: 0.1537 - accuracy: 0.9520 - valloss: 0.1812 - valaccuracy: 0.9462

Epoch 2/2 26983/26983 [==============================] - 86s 3ms/step - loss: 0.1397 - accuracy: 0.9560 - valloss: 0.1699 - valaccuracy: 0.9505

Project 2 (Agent 3):

Only Full Dense Layers:

Due to the amount of data associated with representing Agent 3's knowledge, we had to decrease the number of gridworlds we could generate from 15000 to 10000 so as to not crash anyone's system with an overabundance of data (either localhosts or Google Colab). The number of epochs used for this agent's corresponding NN would also be increased from 3 to 10 for optimal performance.

Epoch 1/10 313/313 [==============================] - 2s 5ms/step - loss: 0.3007 - accuracy: 0.8565 - valloss: 6.8039 - valaccuracy: 0.4252

Epoch 2/10 313/313 [==============================] - 2s 6ms/step - loss: 0.2905 - accuracy: 0.8561 - valloss: 6.8780 - valaccuracy: 0.4212

Epoch 3/10 313/313 [==========] - 2s 5ms/step - loss: 0.3009 - accuracy: 0.8545 - valloss: 6.7852 - valaccuracy: 0.4226

Epoch 4/10 313/313 [==========] - 1s 5ms/step - loss: 0.3458 - accuracy: 0.8370 - valloss: 6.3415 - valaccuracy: 0.4253

Epoch 5/10 313/313 [==========] - 2s 5ms/step - loss: 0.2933 - accuracy: 0.8557 - valloss: 6.5099 - valaccuracy: 0.4301

Epoch 6/10 313/313 [==========] - 2s 5ms/step - loss: 0.2778 - accuracy: 0.8584 - valloss: 6.8972 - valaccuracy: 0.4208

Epoch 7/10 313/313 [==========] - 1s 4ms/step - loss: 0.2834 - accuracy: 0.8578 - valloss: 7.0093 - valaccuracy: 0.4252

Epoch 8/10 313/313 [==========] - 1s 5ms/step - loss: 0.3155 - accuracy: 0.8488 - valloss: 6.9669 - valaccuracy: 0.4245

Epoch 9/10 313/313 [==========] - 2s 5ms/step - loss: 0.3056 - accuracy: 0.8534 - valloss: 6.8301 - valaccuracy: 0.4293

Epoch 10/10 313/313 [==========] - 1s 4ms/step - loss: 0.3207 - accuracy: 0.8502 - valloss: 6.8984 - valaccuracy: 0.4272

At Least One Convolutional Neural Layer:

Similar to the dense layer architecture, we generated 10000 different gridworlds for training this NN. From here, we determined that the appropriate epoch number also 10, and this acquired the following metrics:

Epoch 1/10 313/313 [==========] - 2s 6ms/step - loss: 0.2480 - accuracy: 0.8796 - valloss: 7.2299 - valaccuracy: 0.4141

Epoch 2/10 313/313 [==========] - 2s 6ms/step - loss: 0.2475 - accuracy: 0.8808 - valloss: 7.2102 - valaccuracy: 0.4120

Epoch 3/10 313/313 [==========] - 2s 6ms/step - loss: 0.2493 - accuracy: 0.8806 - valloss: 7.3105 - valaccuracy: 0.4059

Epoch 4/10 313/313 [==========] - 2s 6ms/step - loss: 0.2559 - accuracy: 0.8775 - valloss: 7.2973 - valaccuracy: 0.4096

Epoch 5/10 313/313 [==========] - 2s 8ms/step - loss: 0.2599 - accuracy: 0.8739 - valloss: 7.0886 - valaccuracy: 0.4074

Epoch 6/10 313/313 [==========] - 2s 6ms/step - loss: 0.2676 - accuracy: 0.8769 - valloss: 7.0862 - valaccuracy: 0.4187

Epoch 7/10 313/313 [==========] - 2s 5ms/step - loss: 0.2950 - accuracy: 0.8673 - valloss: 7.2993 - valaccuracy: 0.4169

Epoch 8/10 313/313 [==========] - 2s 8ms/step - loss: 0.2849 - accuracy: 0.8708 - valloss: 7.3954 - valaccuracy: 0.4148

Epoch 9/10 313/313 [==========] - 2s 8ms/step - loss: 0.2517 - accuracy: 0.8777 - valloss: 7.3782 - valaccuracy: 0.4099

Epoch 10/10 313/313 [==============================] - 2s 6ms/step - loss: 0.2495 - accuracy: 0.8766 - valloss: 7.2909 - valaccuracy: 0.4101

Problem 4

How did you avoid overfitting? Since you want the ML agent to mimic the original agent, should you avoid overfitting?

ANSWER:

We avoided overfitting by increasing the size of the dataset that we were feeding to the NN and not letting the models learn over too many epochs. For example, in the beginning we only trained 5000 data points for both agents, but the training model's came out to be repeatedly going in one direction without changing anything, and the probability of the prediction output showed that the other three directions has no chance of being chosen before the one direction that the model kept choosing.

Thus, we increased our data as much as possible before we ran into space issues, and we also decreased the epoch numbers to 'end our training' early as opposed to our previous absurdly high number of epochs for Agents 1 and 3. From there, the other directions has a higher probability of occurring given some state of the gridworld and agent.

We also attempted, where possible, to decrease the complexity of our architectures where possible (touched on in the above question 5's as well as below).

Problem 5

How did you explore the architecture space, and test the different possibilities to find the best architecture?

ANSWER:

Project 1 (Agent 1):

Only Full Dense Layers:

There was some experimenting with various numbers of layers and nodes in those layers, but we ended up deciding to use two dense layers with units of 32 and 16. We found that this produced the best model structure for the problem at hand.

At Least One Convolutional Neural Layer:

We just want to mention here that while the structure of the data conversion of Agent 1's CNN is similar to that of the data conversion of the dense layer architecture, the modelbuild() function is different. Now that we're dealing with a convolutional neural layer, we had window sizes, filter numbers and more to consider.

What we ended up deciding on in the modelbuild() was the following: agent 1's CNN model builder has one layer of the Conv2D() with filters (filter = 16), kernel size of 3, activation function of ReLU and an input of (dim, dim, 1). Then, the result would be flattened to go through a dense layer of 4.

Project 2 (Agent 3):

Only Full Dense Layers:

Much like for the dense layer model structure for the first part of the project, we were thinking about a similar model for this as well. The only difference is our input space was twice the size that is was before, so we ended up deciding to use 4 dense layers of 100 units, 40 units, 20 units, and 10 units respectively.

At Least One Convolutional Neural Layer:

We just want to mention here that while the structure of the data conversion of Agent 2's CNN is similar to that of the data conversion of the dense layer architecture, the modelbuild() function is different. Now that we're dealing with a convolutional neural layer, we had window sizes, filter numbers and more to consider.

What we ended up deciding on in the modelbuild() was the following: agent 2's CNN model builder has one layer of the Conv2D() with filters (filter = 16), kernel size of 3, activation function of ReLU and an input of (dim, dim, 2). Then, the result would be flattened to go through a dense layer of 4.

Problem 6

Do you think increasing the size or complexity of your model would offer any improvements? Why or why not?

ANSWER:

Increasing the size or complexity of our model actually had the opposite effect on the model's prediction in imitating both agents, and I'll explain what we mean by that. The agents move according to the heuristic estimates at each respective cell in the gridworld, the knowledge it has about the gridworld, and how the agent can potentially derive new information (mostly Agent 3). Had we increased the complexity and size of the model, it may taking into account information and/or unseen factors in the maze and knowledge base (at least in reference to the original agent). Even though the model agent would be able to reach the goal and potentially faster, the objective of the project was for the model agent to imitate the behavior of the respective agents.

Size and complexity might offer improvements in the sense of optimal performance, but it would potentially break from the current agents' abilities. So, even though the validation values of the models with more complexity and greater size were higher than that of models with less complexity and lesser size, the simpler model shoots for something more similar to the agents that we already know.

Problem 7

Does good performance on test data correlate with good performance in practice? Simulate the performance of your ML agent on new gridworlds to evaluate this.

ANSWER:

We determined, to the best of our ability, that good performance on test data doesn't necessarily correlate to good performance in practice. We observed that the predictions made on test data had very high accuracy. However, while using the model to predict the agent's action (which way the agent needs to move) in a new gridworld, the model frequently ran out of the gridworld or kept getting stuck in loops going left and right repeatedly.

One possible explanation for this could be that the model didn't understand the context of when to perform an action but simply performs it since it matches what it has already seen. Another possible explanation for this could be that there isn't enough data for the model to train on edge cases (when the agent is anywhere in the first row (*, 0), the first column (0, *), the last row (*, dim-1), or the last column (dim-1, *)), since most of the training data is comprised of the agent being in the middle of the maze (as there are more neighbors of cells in the middle of the maze compared to the edges).

Simulating this in a new gridworld required hardcoding a lot of edge cases where the model either ran out of the maze or got stuck looping over and over again. The model certainly couldn't reach the goal node just based on the prediction it made, and it needed someone looking over its actions so that it doesn't end up going out of bounds or continuously loop the same actions over and over again.

Problem 8

For your best model structure, for each architecture, plot a) performance on test data as a function of training rounds, and b) average performance in practice on new gridworlds. How do your ML agents stack up against the original agents? Do either ML agents offer an advantage in terms of training time?

ANSWER:

Unfortunately, due to many of the above situations, we didn't have a successful enough execution of any of the NN models to get enough conclusive data to determine the best model structure. Since the NN were being trained to imitate the specified agents, we would like to hypothesize that the models would, at best, be as good as the agents it learned from. Of course, there might be some outliers, but on average, based on the knowledge that the agent has which the model learned from, we feel that the performance of the actual agent would be a ceiling for the model's performance. However, there is always the chance that the model could learn to make smarter moves than the explicit planning process of the agents.

We theorize, in hindsight, that perhaps a better metric could have been to feed in the Manhattan distance from a cell to the goal node. While the CNN certainly has its merits in a problem like this, where (almost) only the immediate surroundings contribute to the decision of where to go next, we feel that the knowledge of where the goal is and having directionality towards that goal would be important to the NN's overall success. So, perhaps that would've been a better input space (i.e., general knowledge base) for the different models, especially since we experienced issues where it kept looping between two opposite actions and getting stuck.

Problem 9

APPENDIX

```
# -*- coding: utf-8 -*-
"""AI520_imitator1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1XUSDQhpTOGxZwJfT1I8aeq2DfVi2FYMH
"""

import numpy as np
import tensorflow as tf
import os

# Commented out IPython magic to ensure Python compatibility.
# Text file data converted to integer data type
from google.colab import drive
drive.mount('/content/gdrive')

# %cd /content/gdrive/My\ Drive/Colab\ Notebooks/AI_project

!pwd

#Load txt file
def getRawData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

#turns [0,0,0,1] into 4 for directions zero default
def getDir(line):
        if(line[0] == '1'):
                return 1
        if(line[1] == '1'):
                return 2
        if(line[3] == '1'):
                return 3
        if(line[4] == '1'):
                return 4
        return 0

def genMatricies(lines):
        x_data = []
        x_row_list = []
        y_data = []

        for x in range(0,len(lines)):
                # This is used to get rid of the \n in the txt file
                if (x % 12 == 0):
                        first = 0
                        continue
                #/the 51th line is the direction vector
                if (x % 12 == 11):
                        y_data.append(getDir(lines[x]))

                #everything else is the matrix
                else:
                        #get each char we then convert it to string for tensorflow calcs
                        for sym in lines[x]:
                                i = 0
                                if first == 0:
                                        i = 1
                                        first = 1

                                if(sym == 'X'):
                                        i = -1
                                elif(sym == '\n'):
                                        continue
                                elif(sym == '2'):
                                        i = 2
                                elif(sym == '1'):
                                        i = 1
                                x_row_list.append(i)
                        #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
                        if(len(x_row_list) == 100):
                                x_data.append(np.array(x_row_list).reshape(10,10))
                                x_row_list = []
                        # turn list into numpy array, turn numpy array into catagorical input
        return (np.array(x_data),tf.keras.utils.to_categorical(np.array(y_data), 4))

"""DATA CREATION"""

lines = getRawData("proto10_train.txt")
x1 ,y1 = genMatricies(lines)

np.save('proto10_train_x1.npy', x1)

np.save('proto10_train_y1.npy', y1)
```

```
x1 = np.load("proto10_train_x1.npy")
y1 = np.load("proto10_train_y1.npy")

train_x1 = np.load('proto50_500_train_x1.npy')
#train_y1 = np.load('proto50_train_y1.npy')

train_y1 = np.load('proto50_500_train_y1.npy')
#train_y2 = np.load('proto50_train_y2.npy')
#total_y = np.append(train_y1, train_y2)
#np.save('proto50_train_total_y.npy', total_y)

"""TRAINING AGENT"""

print(tf.config.list_physical_devices('GPU'))
#checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto50_model/proto50_train"
checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto10_model/"
checkpoint_dir = os.path.dirname(checkpoint_path)
EPOCHS_DEFAULT = 3

def buildModel():
  global EPOCHS_DEFAULT
  # add more layers here as needed
  gridInput = tf.keras.layers.Input(shape = (10,10))
  flatten_image = tf.keras.layers.Flatten()( gridInput )

  #dense_1 = tf.keras.layers.Dense( units = 1000, activation = tf.nn.relu )( flatten_image )
  #dense_2 = tf.keras.layers.Dense( units = 400, activation = tf.nn.relu )( dense_1 )
  #dense_3 = tf.keras.layers.Dense( units = 200, activation = tf.nn.relu )( dense_2 )
  #dense_4 = tf.keras.layers.Dense( units = 100, activation = tf.nn.relu )( dense_3 )

  #infDec = tf.keras.layers.Dense( units = 4, activation = None )( dense_4 )

  dense_1 = tf.keras.layers.Dense( units = 16, activation = tf.nn.relu )( flatten_image )

  infDec = tf.keras.layers.Dense( units = 4, activation = None )( dense_1 )

  probabilities = tf.keras.layers.Softmax()( infDec )

  model = tf.keras.Model( inputs = gridInput, outputs = probabilities)
  model.compile( optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'] )
  latest = tf.train.latest_checkpoint(checkpoint_dir)

  if(latest == None):
    return model

  model.load_weights(latest)
  EPOCHS_DEFAULT = 3

  return model

# this allows you to resume or expand upon training
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)

tf.test.is_gpu_available()

model = buildModel()

#model = buildModel()

#model = tf.keras.models.load_model(checkpoint_path)

history = model.fit(x1 , y1, epochs = EPOCHS_DEFAULT,
          callbacks=[cp_callback] )

"""TESTING"""

def getTestData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

test_lines = getTestData("proto10_test.txt")

tx,ty = genMatricies(test_lines)

np.save('proto10_test_x1.npy', tx)
np.save('proto10_test_y1.npy', ty)

tx = np.load('proto10_test_x1.npy')
ty = np.load('proto10_test_y1.npy')

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(x1, y1, validation_data=(tx, ty), epochs=3)



"""GEN MAZE"""

#map constructor
from random import randint
from matplotlib import pyplot as plot
```

```
#size of each grid
grid_size = 10
#dim x dim for maze
size = int(10)
#prob of wall appearance (0~1)
p_wall = 30

maze= [['w' for _ in range(size)]for _ in range(size)]

def createMaze():

  x_data = []
  x_row_list = []

  for row in range(size):
      for col in range(size):
          s = randint(1,100)
          p = p_wall
          if s<=p:
            maze[row][col] = 'X'
          else:
            maze[row][col] = '0'

  maze[0][0]= '0'
  maze[size-1][size-1] = '2'

createMaze()

print(maze)

def genMaze(size):
  x_data = []
  x_row_list = []

  for row in range(size):
  #get each char we then convert it to string for tensorflow calcs
    for col in maze[row]:
      i = 0
      if(col == 'X'):
        i = -1
      elif(col == '\n'):
        continue
      elif(col == '2'):
        i = 2
      elif(col == '1'):
        i = 1
      x_row_list.append(i)
    #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
  if(len(x_row_list) == 100):
    x_data.append(np.array(x_row_list).reshape(10,10))
    x_row_list = []
  # turn list into numpy array, turn numpy array into catagorical input
  return np.array(x_data)

def getDir(y_arr):
  max = -9
  idx = 0
  curr = 0
  for i in range(len(y_arr)):
    if y_arr[i] > max:
      idx = i

  dir = [0,0,0,0]
  dir[idx] = 1


  return dir

new_maze = genMaze(size)

print(new_maze)

maze= [['w' for _ in range(size)]for _ in range(size)]
createMaze()
new_maze = genMaze(size)
new_maze[0][0][0] = 1
print(new_maze)

print(tx[:15])
direct = model.predict(tx[:15])

print(direct[:15])

for i in range(15):
  print(getDir(direct[i]))

print(tx[:10])

# -*- coding: utf-8 -*-
"""CNN_AI520_imitator1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1OuipVR8I9Q9F224YJyzeMmMkNKEtn88c
"""

import numpy as np
```

```
import tensorflow as tf
import os

# Commented out IPython magic to ensure Python compatibility.
# Text file data converted to integer data type
from google.colab import drive
drive.mount('/content/gdrive')

# %cd /content/gdrive/My\ Drive/Colab\ Notebooks/AI_project

!pwd

#Load txt file
def getRawData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

#turns [0,0,0,1] into 4 for directions zero default
def getDir(line):
        if(line[0] == '1'):
                return 1
        if(line[1] == '1'):
                return 2
        if(line[3] == '1'):
                return 3
        if(line[4] == '1'):
                return 4
        return 0

def genMatricies(lines):
        x_data = []
        x_row_list = []
        y_data = []

        for x in range(0,len(lines)):
                # This is used to get rid of the \n in the txt file
                if (x % 12 == 0):
                        continue
                #/the 51th line is the direction vector
                if (x % 12 == 11):
                        y_data.append(getDir(lines[x]))

                #everything else is the matrix
                else:
                        #get each char we then convert it to string for tensorflow calcs
                        for sym in lines[x]:
                                i = 0
                                if(sym == 'X'):
                                        i = -1
                                elif(sym == '\n'):
                                        continue
                                elif(sym == '2'):
                                        i = 2
                                elif(sym == '1'):
                                        i = 1
                                x_row_list.append(i)
                        #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
                        if(len(x_row_list) == 100):
                                x_data.append(np.array(x_row_list).reshape(10,10))
                                x_row_list = []
                        # turn list into numpy array, turn numpy array into catagorical input
        return (np.array(x_data),tf.keras.utils.to_categorical(np.array(y_data), 4))

"""DATA CREATION"""

lines = getRawData("proto10_train.txt")
x1 ,y1 = genMatricies(lines)

np.save('proto10_train_x1.npy', x1)

np.save('proto10_train_y1.npy', y1)

x1 = np.load("proto10_train_x1.npy")
y1 = np.load("proto10_train_y1.npy")

#train_x1 = np.load('proto50_500_train_x1.npy')
#train_y1 = np.load('proto50_train_y1.npy')

#train_y1 = np.load('proto50_500_train_y1.npy')
#train_y2 = np.load('proto50_train_y2.npy')
#total_y = np.append(train_y1, train_y2)
#np.save('proto50_train_total_y.npy', total_y)

"""TRAINING AGENT"""

print(tf.config.list_physical_devices('GPU'))
#checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto50_model/proto50_train"
checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto10_model/CNN/"
checkpoint_dir = os.path.dirname(checkpoint_path)
EPOCHS_DEFAULT = 1



from keras.models import Sequential
```

```
from keras.layers import Dense, Conv2D, Flatten

def buildModel():
  global EPOCHS_DEFAULT

  model = Sequential()

  #add model layers
  model.add(Conv2D(16, kernel_size=3, activation= tf.nn.relu, input_shape=(10,10,1)))
  #model.add(Conv2D(4, kernel_size=3, activation= tf.nn.relu))
  model.add(Flatten())
  model.add(Dense(4, activation= tf.nn.softmax))

  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

  return model

#train the model
model = buildModel()


#model.fit(x1, y1, validation_data=(X_test, y_test), epochs=3)

# this allows you to resume or expand upon training
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)

#tf.test.is_gpu_available()

model = buildModel()

#model = buildModel()

#model = tf.keras.models.load_model(checkpoint_path)

history = model.fit(x1 , y1, epochs = EPOCHS_DEFAULT,
          callbacks=[cp_callback] )

"""TESTING"""

def getTestData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

test_lines = getTestData("proto10_test.txt")

tx,ty = genMatricies(test_lines)

np.save('proto10_test_x1.npy', tx)
np.save('proto10_test_y1.npy', ty)

tx = np.load('proto10_test_x1.npy')
ty = np.load('proto10_test_y1.npy')

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(x1, y1, validation_data=(tx, ty), epochs=2)



"""GEN MAZE"""

#map constructor
from random import randint
from matplotlib import pyplot as plot

#size of each grid
grid_size = 10
#dim x dim for maze
size = int(10)
#prob of wall appearance (0~1)
p_wall = 30

maze= [['w' for _ in range(size)]for _ in range(size)]

def createMaze():

  x_data = []
  x_row_list = []

  for row in range(size):
      for col in range(size):
          s = randint(1,100)
          p = p_wall
          if s<=p:
            maze[row][col] = 'X'
          else:
            maze[row][col] = '0'

  maze[0][0]= '0'
  maze[size-1][size-1] = '2'
```

```
createMaze()

print(maze)

def genMaze(size):
  x_data = []
  x_row_list = []

  for row in range(size):
  #get each char we then convert it to string for tensorflow calcs
    for col in maze[row]:
      i = 0
      if(col == 'X'):
        i = -1
      elif(col == '\n'):
        continue
      elif(col == '2'):
        i = 2
      elif(col == '1'):
        i = 1
      x_row_list.append(i)
    #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
  if(len(x_row_list) == 100):
    x_data.append(np.array(x_row_list).reshape(10,10))
    x_row_list = []
  # turn list into numpy array, turn numpy array into catagorical input
  return np.array(x_data)

def getDir(y_arr):
  max = -9
  idx = 0
  curr = 0
  for i in range(len(y_arr)):
    if y_arr[i] > max:
      idx = i

  dir = [0,0,0,0]
  dir[idx] = 1


  return dir

new_maze = genMaze(size)

print(new_maze)

maze= [['w' for _ in range(size)]for _ in range(size)]
createMaze()
new_maze = genMaze(size)
new_maze[0][0][0] = 1
print(new_maze)

print(tx[:15])
direct = model.predict(tx[:15])

print(direct[:15])

for i in range(15):
  print(getDir(direct[i]))

print(tx[:10])

# -*- coding: utf-8 -*-
"""AI520_imitator2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1yV1kGD_iaTm-R86KitR4Nl7MP7RpJK6g
"""

import numpy as np
import tensorflow as tf
import os

# Commented out IPython magic to ensure Python compatibility.
# Text file data converted to integer data type
from google.colab import drive
drive.mount('/content/gdrive')

# %cd /content/gdrive/My\ Drive/Colab\ Notebooks/AI_project

!pwd

#Load txt file
def getRawData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

#turns [0,0,0,1] into 4 for directions zero default
def getDir(line):
        if(line[0] == '1'):
                return 1
        if(line[1] == '1'):
```

```
                return 2
        if(line[3] == '1'):
                return 3
        if(line[4] == '1'):
                return 4
        return 0

def genMatricies(lines):
        x_data = []
        x_row_list = []
        y_data = []

        for x in range(0,len(lines)):
                # This is used to get rid of the \n in the txt file
                if (x % 12 == 0):
                        continue
                #/the 51th line is the direction vector
                if (x % 12 == 11):
                        y_data.append(getDir(lines[x]))

                #everything else is the matrix
                else:
                        #get each char we then convert it to string for tensorflow calcs
                        for sym in lines[x]:
                                i = 0
                                if(sym == 'X'):
                                        i = -1
                                elif(sym == '\n'):
                                        continue
                                elif(sym == '2'):
                                        i = 2
                                elif(sym == '1'):
                                        i = 1
                                x_row_list.append(i)
                        #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
                        if(len(x_row_list) == 100):
                                x_data.append(np.array(x_row_list).reshape(10,10))
                                x_row_list = []
                        # turn list into numpy array, turn numpy array into catagorical input
        return (np.array(x_data),tf.keras.utils.to_categorical(np.array(y_data), 4))

"""DATA CREATION"""

lines = getRawData("proto10_train_gen.txt")
x1 ,y1 = genMatricies(lines)

print(np.shape(x1))

#inf kb
lines = getRawData("proto10_train_infer.txt")
inf1 ,inf2 = genMatricies(lines)

np.save('proto10_train_gen_x1.npy', x1)

np.save('proto10_train_gen_y1.npy', y1)

#x1 = np.load("proto10_train_x1.npy")
#y1 = np.load("proto10_train_y1.npy")

np.save('proto10_train_inf.npy', inf1)
#train_y1 = np.load('proto50_train_y1.npy')

#train_y1 = np.load('proto50_500_train_y1.npy')
#train_y2 = np.load('proto50_train_y2.npy')
#total_y = np.append(train_y1, train_y2)
#np.save('proto50_train_total_y.npy', total_y)

"""TRAINING AGENT"""

print(tf.config.list_physical_devices('GPU'))
#checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto50_model/proto50_train"
checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto10_model/proj2/"
checkpoint_dir = os.path.dirname(checkpoint_path)
EPOCHS_DEFAULT = 300

cat_array = np.concatenate((x1[:10000], inf1[:10000]), axis = 0)

y_array = y1[:10000]

print(np.shape(cat_array))
print(np.shape(y_array))

cat_array = np.reshape(cat_array, (-1, 2, 10, 10))

print(cat_array[5])

print(x1[6])
print(inf1[6])

def buildModel():
  global EPOCHS_DEFAULT
  # add more layers here as needed

  gridInput = tf.keras.layers.Input(shape = ( 2, 10, 10))

  flatten_image = tf.keras.layers.Flatten()( gridInput)

  dense_1 = tf.keras.layers.Dense( units = 100, activation = tf.nn.relu )( flatten_image )
  dense_2 = tf.keras.layers.Dense( units = 40, activation = tf.nn.relu )( dense_1 )
```

```
  dense_3 = tf.keras.layers.Dense( units = 20, activation = tf.nn.relu )( dense_2 )
  dense_4 = tf.keras.layers.Dense( units = 10, activation = tf.nn.relu )( dense_3 )

  infDec = tf.keras.layers.Dense( units = 4, activation = None )( dense_4 )

  #dense_1 = tf.keras.layers.Dense( units = 64, activation = tf.nn.relu )( flatten_image )

  #dense_2 = tf.keras.layers.Dense( units = 16, activation = tf.nn.relu )( dense_1 )

  #infDec = tf.keras.layers.Dense( units = 4, activation = None )( dense_2 )

  probabilities = tf.keras.layers.Softmax()( infDec )

  model = tf.keras.Model( inputs = gridInput, outputs = probabilities)

  model.compile( optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'] )
  latest = tf.train.latest_checkpoint(checkpoint_dir)

  if(latest == None):
    return model

  model.load_weights(latest)
  EPOCHS_DEFAULT = 300

  return model

# this allows you to resume or expand upon training
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)

#tf.test.is_gpu_available()

model = buildModel()

#model = buildModel()

#model = tf.keras.models.load_model(checkpoint_path)

#out = Concatenate()([x1, inf1])

history = model.fit( cat_array, y_array, epochs = EPOCHS_DEFAULT,
          callbacks=[cp_callback] )

"""TESTING"""

def getTestData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

test_lines = getTestData("proto10_test_gen.txt")

tx,ty = genMatricies(test_lines)

print(tx[4])

test_lines = getTestData("proto10_test_infer.txt")

tinf1, tinf2 = genMatricies(test_lines)

np.save('proto10_test_gen_x1.npy', tx)
np.save('proto10_test_gen_y1.npy', ty)

np.save('proto10_test_inf_x1.npy', tx)
np.save('proto10_test_inf_y1.npy', ty)

#tx = np.load('proto50_test_500_x1.npy')
#ty = np.load('proto50_test_500_y1.npy')

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

test_array = np.concatenate((tx[:10000], tinf1[:10000]), axis = 0)

ty_array = ty[:10000]

test_array = np.reshape(test_array, (-1, 2, 10, 10))

print(np.shape(test_array))

model.fit(cat_array, y_array, validation_data=(test_array, ty_array), epochs=10)



"""GEN MAZE"""

#map constructor
from random import randint
from matplotlib import pyplot as plot

#size of each grid
grid_size = 10
#dim x dim for maze
size = int(10)
```

```
#prob of wall appearance (0~1)
p_wall = 30

maze= [['w' for _ in range(size)]for _ in range(size)]

def createMaze():

  x_data = []
  x_row_list = []

  for row in range(size):
      for col in range(size):
          s = randint(1,100)
          p = p_wall
          if s<=p:
            maze[row][col] = 'X'
          else:
            maze[row][col] = '0'

  maze[0][0]= '0'
  maze[size-1][size-1] = '2'

createMaze()

print(maze)

def genMaze(size):
  x_data = []
  x_row_list = []

  for row in range(size):
  #get each char we then convert it to string for tensorflow calcs
    for col in maze[row]:
      i = 0
      if(col == 'X'):
        i = -1
      elif(col == '\n'):
        continue
      elif(col == '2'):
        i = 2
      elif(col == '1'):
        i = 1
      x_row_list.append(i)
    #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
  if(len(x_row_list) == 100):
    x_data.append(np.array(x_row_list).reshape(10,10))
    x_row_list = []
  # turn list into numpy array, turn numpy array into catagorical input
  return np.array(x_data)

def getDir(y_arr):
  max = 0
  idx = 0
  curr = 0
  for val in y_arr:
    if val > max:
      idx = curr
    curr += 1

  dir = []
  for i in range(4):
    if idx != i:
      dir.append(0)
    else:
      dir.append(1)

  return dir

new_maze = genMaze(size)

print(new_maze)

maze= [['w' for _ in range(size)]for _ in range(size)]
createMaze()
new_maze = genMaze(size)
new_maze[0][0][0] = 1
print(new_maze)

print(test_array[3])

y = model.predict(test_array[1])

print(test_array[1][0])

output = getDir(y[0])
print(output)

"""GEN MAZE"""

from random import randint
from matplotlib import pyplot as plot

#size of each grid
grid_size = 10
#dim x dim for maze
size = int(10)
#prob of wall appearance (0~1)
p_wall = 30
```

```
maze= [['w' for _ in range(size)]for _ in range(size)]

def createMaze():

  x_data = []
  x_row_list = []

  for row in range(size):
      for col in range(size):
          s = randint(1,100)
          p = p_wall
          if s<=p:
            maze[row][col] = 'X'
          else:
            maze[row][col] = '0'

  maze[0][0]= '0'
  maze[size-1][size-1] = '2'

createMaze()

print(maze)

def genMaze(size):
  x_data = []
  x_row_list = []

  for row in range(size):
  #get each char we then convert it to string for tensorflow calcs
    for col in maze[row]:
      i = 0
      if(col == 'X'):
        i = -1
      elif(col == '\n'):
        continue
      elif(col == '2'):
        i = 2
      elif(col == '1'):
        i = 1
      x_row_list.append(i)
    #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
  if(len(x_row_list) == 100):
    x_data.append(np.array(x_row_list).reshape(10,10))
    x_row_list = []
  # turn list into numpy array, turn numpy array into catagorical input
  return np.array(x_data)

def getDir(y_arr):
  max = 0
  idx = 0
  curr = 0
  for val in y_arr:
    if val > max:
      idx = curr
    curr += 1

  dir = []
  for i in range(4):
    if idx != i:
      dir.append(0)
    else:
      dir.append(1)

  return dir

new_maze = genMaze(size)
print(new_maze)

maze= [['w' for _ in range(size)]for _ in range(size)]
createMaze()
new_maze = genMaze(size)
new_maze[0][0][0] = 1
print(new_maze)

# -*- coding: utf-8 -*-
"""CNN_AI520_imitator2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1JkX2GrNc3ntDOPit89v6j_GxkBcwlrhV
"""

import numpy as np
import tensorflow as tf
import os

# Commented out IPython magic to ensure Python compatibility.
# Text file data converted to integer data type
from google.colab import drive
drive.mount('/content/gdrive')

# %cd /content/gdrive/My\ Drive/Colab\ Notebooks/AI_project

!pwd

#Load txt file
def getRawData(string):
  lines = []
```

```
  f = open(string, "r")

  lines = f.readlines()

  return lines

#turns [0,0,0,1] into 4 for directions zero default
def getDir(line):
        if(line[0] == '1'):
                return 1
        if(line[1] == '1'):
                return 2
        if(line[3] == '1'):
                return 3
        if(line[4] == '1'):
                return 4
        return 0

def genMatricies(lines):
        x_data = []
        x_row_list = []
        y_data = []

        for x in range(0,len(lines)):
                # This is used to get rid of the \n in the txt file
                if (x % 12 == 0):
                        continue
                #/the 51th line is the direction vector
                if (x % 12 == 11):
                        y_data.append(getDir(lines[x]))

                #everything else is the matrix
                else:
                        #get each char we then convert it to string for tensorflow calcs
                        for sym in lines[x]:
                                i = 0
                                if(sym == 'X'):
                                        i = -1
                                elif(sym == '\n'):
                                        continue
                                elif(sym == '2'):
                                        i = 2
                                elif(sym == '1'):
                                        i = 1
                                x_row_list.append(i)
                        #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
                        if(len(x_row_list) == 100):
                                x_data.append(np.array(x_row_list).reshape(10,10))
                                x_row_list = []
                        # turn list into numpy array, turn numpy array into catagorical input
        return (np.array(x_data),tf.keras.utils.to_categorical(np.array(y_data), 4))

"""DATA CREATION"""

lines = getRawData("proto10_train_gen.txt")
x1 ,y1 = genMatricies(lines)

print(np.shape(x1))

#inf kb
lines = getRawData("proto10_train_infer.txt")
inf1 ,inf2 = genMatricies(lines)

np.save('proto10_train_gen_x1.npy', x1)

np.save('proto10_train_gen_y1.npy', y1)

x1 = np.load("proto10_train_x1.npy")
y1 = np.load("proto10_train_y1.npy")

np.save('proto10_train_inf.npy', inf1)
#train_y1 = np.load('proto50_train_y1.npy')

#train_y1 = np.load('proto50_500_train_y1.npy')
#train_y2 = np.load('proto50_train_y2.npy')
#total_y = np.append(train_y1, train_y2)
#np.save('proto50_train_total_y.npy', total_y)

"""TRAINING AGENT"""

print(tf.config.list_physical_devices('GPU'))
#checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto50_model/proto50_train"
checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto10_model/proj2/CNN/"
checkpoint_dir = os.path.dirname(checkpoint_path)
EPOCHS_DEFAULT = 1000

cat_array = np.concatenate((x1[:10000], inf1[:10000]), axis = 0)

y_array = y1[:10000]

print(np.shape(cat_array))
print(np.shape(y_array))

cat_array = np.reshape(cat_array, (-1, 10, 10, 2))

print(cat_array[5])

print(x1[6])
```

```
print(inf1[6])

def buildModel():
  global EPOCHS_DEFAULT
  # add more layers here as needed

  gridInput = tf.keras.layers.Input(shape = ( 2, 10, 10))

  flatten_image = tf.keras.layers.Flatten()( gridInput)

  dense_1 = tf.keras.layers.Dense( units = 100, activation = tf.nn.relu )( flatten_image )
  dense_2 = tf.keras.layers.Dense( units = 40, activation = tf.nn.relu )( dense_1 )
  dense_3 = tf.keras.layers.Dense( units = 20, activation = tf.nn.relu )( dense_2 )
  dense_4 = tf.keras.layers.Dense( units = 10, activation = tf.nn.relu )( dense_3 )

  infDec = tf.keras.layers.Dense( units = 4, activation = None )( dense_4 )

  #dense_1 = tf.keras.layers.Dense( units = 64, activation = tf.nn.relu )( flatten_image )

  #dense_2 = tf.keras.layers.Dense( units = 16, activation = tf.nn.relu )( dense_1 )

  #infDec = tf.keras.layers.Dense( units = 4, activation = None )( dense_2 )

  probabilities = tf.keras.layers.Softmax()( infDec )

  model = tf.keras.Model( inputs = gridInput, outputs = probabilities)

  model.compile( optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'] )
  latest = tf.train.latest_checkpoint(checkpoint_dir)

  if(latest == None):
    return model

  model.load_weights(latest)
  EPOCHS_DEFAULT = 300

  return model

from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten

def buildModel():
  global EPOCHS_DEFAULT

  model = Sequential()

  #add model layers
  model.add(Conv2D(64, kernel_size=3, activation= tf.nn.relu, input_shape=(10,10,2)))
  model.add(Conv2D(32, kernel_size=3, activation= tf.nn.relu))
  model.add(Conv2D(16, kernel_size=3, activation= tf.nn.relu))
  model.add(Flatten())
  model.add(Dense(4, activation= tf.nn.softmax))

  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

  return model

# this allows you to resume or expand upon training
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)

#tf.test.is_gpu_available()

model = buildModel()

#model = buildModel()

#model = tf.keras.models.load_model(checkpoint_path)

#out = Concatenate()([x1, inf1])

history = model.fit( cat_array, y_array, epochs = EPOCHS_DEFAULT,
          callbacks=[cp_callback] )

"""TESTING"""

def getTestData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

test_lines = getTestData("proto10_test_gen.txt")

tx,ty = genMatricies(test_lines)

print(tx[4])

test_lines = getTestData("proto10_test_infer.txt")

tinf1, tinf2 = genMatricies(test_lines)

np.save('proto10_test_gen_x1.npy', tx)
np.save('proto10_test_gen_y1.npy', ty)
```

```
np.save('proto10_test_inf_x1.npy', tx)
np.save('proto10_test_inf_y1.npy', ty)

np.save('proto10_test_infer_x1.npy', tinf1)

tx = np.load('proto10_test_gen_x1.npy')
ty = np.load('proto10_test_gen_y1.npy')
tinf1 = np.load('proto10_test_infer_x1.npy')

#compile model using accuracy to measure model performance
#model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

test_array = np.concatenate((tx[:10000], tinf1[:10000]), axis = 0)

ty_array = ty[:10000]

test_array = np.reshape(test_array, (-1, 10, 10, 2))

print(np.shape(test_array))

print(np.shape(cat_array))

model.fit(cat_array, y_array, validation_data=(test_array, ty_array), epochs=10)



"""GEN MAZE"""

#map constructor
from random import randint
from matplotlib import pyplot as plot

#size of each grid
grid_size = 10
#dim x dim for maze
size = int(10)
#prob of wall appearance (0~1)
p_wall = 30

maze= [['w' for _ in range(size)]for _ in range(size)]

def createMaze():

  x_data = []
  x_row_list = []

  for row in range(size):
      for col in range(size):
          s = randint(1,100)
          p = p_wall
          if s<=p:
            maze[row][col] = 'X'
          else:
            maze[row][col] = '0'

  maze[0][0]= '0'
  maze[size-1][size-1] = '2'

createMaze()

print(maze)

def genMaze(size):
  x_data = []
  x_row_list = []

  for row in range(size):
  #get each char we then convert it to string for tensorflow calcs
    for col in maze[row]:
      i = 0
      if(col == 'X'):
        i = -1
      elif(col == '\n'):
        continue
      elif(col == '2'):
        i = 2
      elif(col == '1'):
        i = 1
      x_row_list.append(i)
    #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
  if(len(x_row_list) == 100):
    x_data.append(np.array(x_row_list).reshape(10,10))
    x_row_list = []
  # turn list into numpy array, turn numpy array into catagorical input
  return np.array(x_data)

def getDir(y_arr):
  max = 0
  idx = 0
  curr = 0
  for val in y_arr:
    if val > max:
      idx = curr
    curr += 1

  dir = []
  for i in range(4):
    if idx != i:
      dir.append(0)
```

```
    else:
      dir.append(1)

  return dir

new_maze = genMaze(size)

print(new_maze)

maze= [['w' for _ in range(size)]for _ in range(size)]
createMaze()
new_maze = genMaze(size)
new_maze[0][0][0] = 1
print(new_maze)

print(test_array[3])

y = model.predict(test_array[1])

print(test_array[1][0])

output = getDir(y[0])
print(output)

"""GEN MAZE"""

from random import randint
from matplotlib import pyplot as plot

#size of each grid
grid_size = 10
#dim x dim for maze
size = int(10)
#prob of wall appearance (0~1)
p_wall = 30

maze= [['w' for _ in range(size)]for _ in range(size)]

def createMaze():

  x_data = []
  x_row_list = []

  for row in range(size):
      for col in range(size):
          s = randint(1,100)
          p = p_wall
          if s<=p:
            maze[row][col] = 'X'
          else:
            maze[row][col] = '0'

  maze[0][0]= '0'
  maze[size-1][size-1] = '2'

createMaze()

print(maze)

def genMaze(size):
  x_data = []
  x_row_list = []

  for row in range(size):
  #get each char we then convert it to string for tensorflow calcs
    for col in maze[row]:
      i = 0
      if(col == 'X'):
        i = -1
      elif(col == '\n'):
        continue
      elif(col == '2'):
        i = 2
      elif(col == '1'):
        i = 1
      x_row_list.append(i)
    #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
  if(len(x_row_list) == 100):
    x_data.append(np.array(x_row_list).reshape(10,10))
    x_row_list = []
  # turn list into numpy array, turn numpy array into catagorical input
  return np.array(x_data)

def getDir(y_arr):
  max = 0
  idx = 0
  curr = 0
  for val in y_arr:
    if val > max:
      idx = curr
    curr += 1

  dir = []
  for i in range(4):
    if idx != i:
      dir.append(0)
    else:
      dir.append(1)
```

```
  return dir

new_maze = genMaze(size)
print(new_maze)

maze= [['w' for _ in range(size)]for _ in range(size)]
createMaze()
new_maze = genMaze(size)
new_maze[0][0][0] = 1
print(new_maze)


# -*- coding: utf-8 -*-
"""AI520_imitator1_1_1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1wUJjrg1EdhIVTH7LG2QN9rQuy_d5Nbng
"""

import numpy as np
import tensorflow as tf
import os

!pwd

cd /common/home/pr482/Data/

#Load txt file
def getRawData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

#turns [0,0,0,1] into 4 for directions zero default
def getDir(line):
        if(line[0] == '1'):
                return 1
        if(line[1] == '1'):
                return 2
        if(line[3] == '1'):
                return 3
        if(line[4] == '1'):
                return 4
        return 0

def genMatricies(lines):
        x_data = []
        x_row_list = []
        y_data = []

        for x in range(0,len(lines)):
                # This is used to get rid of the \n in the txt file
                if (x % 12 == 0):
                        continue
                #/the 51th line is the direction vector
                if (x % 12 == 11):
                        y_data.append(getDir(lines[x]))

                #everything else is the matrix
                else:
                        #get each char we then convert it to string for tensorflow calcs
                        for sym in lines[x]:
                                i = 0
                                if(sym == 'X'):
                                        i = -1
                                elif(sym == '\n'):
                                        continue
                                elif(sym == '2'):
                                        i = 2
                                elif(sym == '1'):
                                        i = 1
                                x_row_list.append(i)
                        #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
                        if(len(x_row_list) == 100):
                                x_data.append(np.array(x_row_list).reshape(10,10))
                                x_row_list = []
                        # turn list into numpy array, turn numpy array into catagorical input
        return (np.array(x_data),tf.keras.utils.to_categorical(np.array(y_data), 4))

"""DATA CREATION"""

lines = getRawData("proto10_train.txt")
x1 ,y1 = genMatricies(lines)

np.save('proto10_train_x1.npy', x1)

np.save('proto10_train_y1.npy', y1)

x1 = np.load("proto10_train_x1.npy")
y1 = np.load("proto10_train_y1.npy")

train_x1 = np.load('proto50_500_train_x1.npy')
#train_y1 = np.load('proto50_train_y1.npy')
```

```
train_y1 = np.load('proto50_500_train_y1.npy')
#train_y2 = np.load('proto50_train_y2.npy')
#total_y = np.append(train_y1, train_y2)
#np.save('proto50_train_total_y.npy', total_y)

"""TRAINING AGENT"""

print(tf.config.list_physical_devices('GPU'))
#checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto50_model/proto50_train"
checkpoint_path = "/content/gdrive/My Drive/Colab Notebooks/AI_project/daniel_proto10_model/"
checkpoint_dir = os.path.dirname(checkpoint_path)
EPOCHS_DEFAULT = 5

def buildModel():
  global EPOCHS_DEFAULT
  # add more layers here as needed
  gridInput = tf.keras.layers.Input(shape = (10,10))
  flatten_image = tf.keras.layers.Flatten()( gridInput )

  #dense_1 = tf.keras.layers.Dense( units = 1000, activation = tf.nn.relu )( flatten_image )
  #dense_2 = tf.keras.layers.Dense( units = 400, activation = tf.nn.relu )( dense_1 )
  #dense_3 = tf.keras.layers.Dense( units = 200, activation = tf.nn.relu )( dense_2 )
  #dense_4 = tf.keras.layers.Dense( units = 100, activation = tf.nn.relu )( dense_3 )

  #infDec = tf.keras.layers.Dense( units = 4, activation = None )( dense_4 )

  dense_1 = tf.keras.layers.Dense( units = 64, activation = tf.nn.relu )( flatten_image )

  dense_2 = tf.keras.layers.Dense( units = 16, activation = tf.nn.relu )( dense_1 )

  infDec = tf.keras.layers.Dense( units = 4, activation = None )( dense_2 )

  probabilities = tf.keras.layers.Softmax()( infDec )

  model = tf.keras.Model( inputs = gridInput, outputs = probabilities)
  model.compile( optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'] )
  latest = tf.train.latest_checkpoint(checkpoint_dir)

  if(latest == None):
    return model

  model.load_weights(latest)
  EPOCHS_DEFAULT = 5

  return model

# this allows you to resume or expand upon training
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)

tf.test.is_gpu_available()

model = buildModel()

#model = buildModel()

#model = tf.keras.models.load_model(checkpoint_path)

history = model.fit(x1 , y1, epochs = EPOCHS_DEFAULT,
          callbacks=[cp_callback] )

"""TESTING"""

def getTestData(string):
  lines = []

  f = open(string, "r")

  lines = f.readlines()

  return lines

test_lines = getTestData("proto10_test.txt")

tx,ty = genMatricies(test_lines)

np.save('proto10_test_x1.npy', tx)
np.save('proto10_test_y1.npy', ty)

tx = np.load('proto50_test_500_x1.npy')
ty = np.load('proto50_test_500_y1.npy')

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(x1, y1, validation_data=(tx, ty), epochs=3)



"""GEN MAZE"""

#map constructor
from random import randint
from matplotlib import pyplot as plot

#size of each grid
grid_size = 10
```

```
#dim x dim for maze
size = int(10)
#prob of wall appearance (0~1)

maze= [['w' for _ in range(size)]for _ in range(size)]

def createMaze(p_wall):

  x_data = []
  x_row_list = []

  for row in range(size):
      for col in range(size):
          s = randint(1,100)
          p = p_wall
          if s<=p:
            maze[row][col] = 'X'
          else:
            maze[row][col] = '0'

  maze[0][0]= '0'
  maze[size-1][size-1] = '2'

createMaze(30)

print(maze)

def genMaze(size):
  x_data = []
  x_row_list = []

  for row in range(size):
  #get each char we then convert it to string for tensorflow calcs
    for col in maze[row]:
      i = 0
      if(col == 'X'):
        i = -1
      elif(col == '\n'):
        continue
      elif(col == '2'):
        i = 2
      elif(col == '1'):
        i = 1
      x_row_list.append(i)
    #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
  if(len(x_row_list) == 100):
    x_data.append(np.array(x_row_list).reshape(10,10))
    x_row_list = []
  # turn list into numpy array, turn numpy array into catagorical input
  return np.array(x_data)

def genMaze_new(size):
  x_data = []
  x_row_list = []

  for row in range(size):
  #get each char we then convert it to string for tensorflow calcs
    for col in maze[row]:
      i = 0
      if(col == 'X'):
        i = -1
      elif(col == '\n'):
        continue
      elif(col == '2'):
        i = 2
      elif(col == '1'):
        i = 1
      x_row_list.append(i)
    #when we have 2500 chars we know we have a 50x50 make it an array and add it to list
  if(len(x_row_list) == 100):
    x_data.append(np.array(x_row_list).reshape(10,10))
    x_row_list = []
  # turn list into numpy array, turn numpy array into catagorical input
  return x_data

def getDir(y_arr):
  max = -1
  idx = 0
  curr = 0
  #print(y_arr)
  for i in range(len(y_arr)):
    if y_arr[i] > max:
        max = y_arr[i]
        idx = i

  dir = [0,0,0,0]
  dir[idx] = 1


  return dir

new_maze = genMaze(size)

print(new_maze)

maze= [['w' for _ in range(size)]for _ in range(size)]
createMaze(30)
new_maze = genMaze_new(size)
new_maze[0][0][0] = 1
```

```
print(new_maze)

print(tx[:15])
direct = model.predict(tx[:15])

print(direct[:15])

for i in range(15):
  print(getDir(direct[i]))

print(tx[:10])

createMaze(0)
KB = genMaze(size)
KB[0][0][0] = 1
#print(KB)
#print(new_maze)

i = 0
j = 0
c = 0
revisit = 0

while(c < 100 | (i == size-1 & j== size-1)):
    c += 1
    predict = model.predict(KB)
    #print(predict[0])
    #move_dir = predict
    move_dir = getDir(predict[0])

    print("Move Dir", move_dir)
    print("KB", KB)
    #print(new_maze)

    if(i == 0 & j == 0):
        if(move_dir[0] == 1):
            predict[0][0] = 0
            move_dir = getDir(predict[0])
        elif(move_dir[2] == 1):
            predict[0][2] = 0
            move_dir = getDir(predict[0])
        KB[0][i+1][j] = new_maze[0][i+1][j]
        KB[0][i][j+1] = new_maze[0][i][j+1]
        for k in range(4):
            if(move_dir[k] == 1):
                if(k==0):
                    if(KB[0][i-1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==1):
                    if(KB[0][i+1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==2):
                    if(KB[0][i][j-1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==3):
                    if(KB[0][i][j+1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])

    elif(i == 0 & j == size-1):
        if(move_dir[0] == 1):
            predict[0][0] = 0
            move_dir = getDir(predict[0])
        elif(move_dir[3] == 1):
            predict[0][3] = 0
            move_dir = getDir(predict[0])
        KB[0][i+1][j] = new_maze[0][i+1][j]
        KB[0][i][j-1] = new_maze[0][i][j-1]
        for k in range(4):
            if(move_dir[k] == 1):
                if(k==0):
                    if(KB[0][i-1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==1):
                    if(KB[0][i+1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==2):
                    if(KB[0][i][j-1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==3):
                    if(KB[0][i][j+1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])

    elif(i == 0 & j!= 0):
        if(move_dir[0] == 1):
            predict[0][0] = 0
            move_dir = getDir(predict[0])
        KB[0][i+1][j] = new_maze[0][i+1][j]
        KB[0][i][j+1] = new_maze[0][i][j+1]
        KB[0][i][j-1] = new_maze[0][i][j-1]
        for k in range(4):
            if(move_dir[k] == 1):
```

```
            if(k==0):
                if(KB[0][i-1][j] == -1):
                    predict[0][k] = 0
                    move_dir = getDir(predict[0])
            if(k==1):
                if(KB[0][i+1][j] == -1):
                    predict[0][k] = 0
                    move_dir = getDir(predict[0])
            if(k==2):
                if(KB[0][i][j-1] == -1):
                    predict[0][k] = 0
                    move_dir = getDir(predict[0])
            if(k==3):
                if(KB[0][i][j+1] == -1):
                    predict[0][k] = 0
                    move_dir = getDir(predict[0])

    elif(i == size-1 & j == 0):
        if(move_dir[1] == 1):
            predict[0][1] = 0
            move_dir = getDir(predict[0])
        elif(move_dir[2] == 1):
            predict[0][2] = 0
            move_dir = getDir(predict[0])
        KB[0][i-1][j] = new_maze[0][i-1][j]
        KB[0][i][j+1] = new_maze[0][i][j+1]
        for k in range(4):
            if(move_dir[k] == 1):
                if(k==0):
                    if(KB[0][i-1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==1):
                    if(KB[0][i+1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==2):
                    if(KB[0][i][j-1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==3):
                    if(KB[0][i][j+1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])

    elif(j == 0 & i != 0):
        if(move_dir[2] == 1):
            predict[0][2] = 0
            move_dir = getDir(predict[0])
        KB[0][i+1][j] = new_maze[0][i+1][j]
        KB[0][i-1][j] = new_maze[0][i-1][j]
        KB[0][i][j+1] = new_maze[0][i][j+1]
        for k in range(4):
            if(move_dir[k] == 1):
                if(k==0):
                    if(KB[0][i-1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==1):
                    if(KB[0][i+1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==2):
                    if(KB[0][i][j-1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==3):
                    if(KB[0][i][j+1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])

    elif(j == size-1 & i != 0):
        if(move_dir[3] == 1):
            predict[0][3] = 0
            move_dir = getDir(predict[0])
        KB[0][i+1][j] = new_maze[0][i+1][j]
        KB[0][i-1][j] = new_maze[0][i-1][j]
        KB[0][i][j-1] = new_maze[0][i][j-1]
        for k in range(4):
            if(move_dir[k] == 1):
                if(k==0):
                    if(KB[0][i-1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==1):
                    if(KB[0][i+1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==2):
                    if(KB[0][i][j-1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==3):
                    if(KB[0][i][j+1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])

    elif(i == size-1 & j != 0):
        if(move_dir[1] == 1):
```

```
            predict[0][1] = 0
            move_dir = getDir(predict[0])
        KB[0][i-1][j] = new_maze[0][i-1][j]
        KB[0][i][j+1] = new_maze[0][i][j+1]
        KB[0][i][j-1] = new_maze[0][i][j-1]
        for k in range(4):
            if(move_dir[k] == 1):
                if(k==0):
                    if(KB[0][i-1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==1):
                    if(KB[0][i+1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==2):
                    if(KB[0][i][j-1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==3):
                    if(KB[0][i][j+1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])

    else:
        KB[0][i+1][j] = new_maze[0][i+1][j]
        KB[0][i-1][j] = new_maze[0][i-1][j]
        KB[0][i][j+1] = new_maze[0][i][j+1]
        KB[0][i][j-1] = new_maze[0][i][j-1]
        for k in range(4):
            if(move_dir[k] == 1):
                if(k==0):
                    if(KB[0][i-1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==1):
                    if(KB[0][i+1][j] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==2):
                    if(KB[0][i][j-1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])
                if(k==3):
                    if(KB[0][i][j+1] == -1):
                        predict[0][k] = 0
                        move_dir = getDir(predict[0])


    if(move_dir[0] == 1):
        i = i-1
    if(move_dir[1] == 1):
        i = i+1
    if(move_dir[2] == 1):
        j = j-1
    if(move_dir[3] == 1):
        j = j+1

    if(new_maze[0][i][j] == 1):
        revisit += 1

    if(revisit > 15):
        print("Revisiting")
        break

    KB[0][i][j] = 1

    print(i, j)
    #print(new_maze)
    print(KB)

new_maze
```