# King Fahd University of Petroleum and Minerals Introduction to Artificial Intelligence (COE292-05) Quiz 2 (10 marks)

**Name:\_\_Mujtaba Zaman\_\_ Major:\_\_MIS\_\_\_ Student ID:\_\_202132090\_\_**

**Instruction(s): Fill in the table below with appropriate answers. Use internet resources.**

# Deadline: 28th November 2023

Q1. Find a code on Github (or other sources available online) on **any 2** of the previous 5 application areas of Machine Learning related to your major from the previous take home quiz.

Transport and Logistics

|  |
| --- |
| actor = network(STATIC\_SIZE,DYNAMIC\_SIZE,kwargs['encoder\_hidden\_size'],kwargs['decoder\_hidden\_size'],kwargs['use\_cuda'],kwargs['input\_type'],kwargs['allow\_rot'],kwargs['container\_width'],kwargs['container\_height'],kwargs['obj\_dim'],kwargs['reward\_type'],kwargs['decoder\_input\_type'],kwargs['heightmap\_type'],kwargs['packing\_strategy'],pack.update\_dynamic,pack.update\_mask,kwargs['num\_layers'],kwargs['dropout'],kwargs['unit'])    critic = realCritic(STATIC\_SIZE, DYNAMIC\_SIZE, kwargs['encoder\_hidden\_size'], kwargs['num\_layers'],kwargs['n\_process\_blocks'], kwargs['dropout'])    if use\_cuda:  actor = actor.cuda()  critic = critic.cuda()      kwargs['train\_data'] = train\_data  kwargs['valid\_data'] = valid\_data  kwargs['reward\_fn'] = pack.reward  kwargs['render\_fn'] = pack.render    if kwargs['checkpoint']:  path = os.path.join(kwargs['checkpoint'], 'actor.pt')  actor.load\_state\_dict(torch.load(path))    path = os.path.join(kwargs['checkpoint'], 'critic.pt')  critic.load\_state\_dict(torch.load(path))    print('Loading pre-train model', path)    train(actor, critic, \*\*kwargs)    if \_\_name\_\_ == '\_\_main\_\_':    parser = argparse.ArgumentParser(description='Transport and Pack')    # Task settings  parser.add\_argument('--task', default='test', type=str) # train, test, generate  parser.add\_argument('--note', default='debug', type=str)  parser.add\_argument('--use\_cuda', default=True, type=str2bool)  parser.add\_argument('--cuda', default='0', type=str)  parser.add\_argument('--cpu\_threads', default=0, type=int)  parser.add\_argument('--checkpoint', default=None)  parser.add\_argument('--seed', default=12345, type=int)    # Training/testing settings  parser.add\_argument('--train\_size',default=13, type=int)  parser.add\_argument('--valid\_size', default=10, type=int)  parser.add\_argument('--epoch\_num', default=2, type=int)  parser.add\_argument('--batch\_size', default=128, type=int)    # Data settings  parser.add\_argument('--obj\_dim', default=2, type=int)  parser.add\_argument('--nodes', dest='num\_nodes', default=10, type=int)  parser.add\_argument('--total\_obj\_num', default=10, type=int) # if more, do Rolling, TODO  parser.add\_argument('--dataset', default='RAND', type=str) # RAND, PPSG, MIX  # sizes of blocks and containers  parser.add\_argument('--unit', default=1.0, type=float)  parser.add\_argument('--arm\_size', default=1, type=int) # size of robotic arm to pass and rotate a block  parser.add\_argument('--min\_size', default=1, type=int)  parser.add\_argument('--max\_size', default=5, type=int)  parser.add\_argument('--container\_width', default=5, type=int)  parser.add\_argument('--container\_length', default=5, type=int) # for 3D, TODO  parser.add\_argument('--container\_height', default=50, type=int)  parser.add\_argument('--initial\_container\_width', default=7, type=int)  parser.add\_argument('--initial\_container\_length', default=7, type=int) # for 3D, TODO  parser.add\_argument('--initial\_container\_height', default=50, type=int)    # Packing settings  parser.add\_argument('--packing\_strategy', default='LB\_GREEDY', type=str)  parser.add\_argument('--reward\_type', default='C+P+S-lb-soft', type=str)    # Network settings  # ---- TODO: network reward  parser.add\_argument('--input\_type', default='bot', type=str)  parser.add\_argument('--allow\_rot', default=True, type=str2bool)  parser.add\_argument('--decoder\_input\_type', default='shape\_heightmap', type=str) # shape\_heightmap, shape\_only, heightmap\_only  parser.add\_argument('--heightmap\_type', default='diff', type=str) # full, zero, diff  parser.add\_argument('--no\_precedence', default=False, type=str2bool) # if true, set all deps to 0    # Network parameters  parser.add\_argument('--dropout', default=0.1, type=float)  parser.add\_argument('--actor\_lr', default=5e-4, type=float)  parser.add\_argument('--critic\_lr', default=5e-4, type=float)  parser.add\_argument('--max\_grad\_norm', default=2., type=float)  parser.add\_argument('--n\_process\_blocks', default=3, type=int)  parser.add\_argument('--layers', dest='num\_layers', default=1, type=int)  parser.add\_argument('--encoder\_hidden', dest='encoder\_hidden\_size', default=128, type=int)  parser.add\_argument('--decoder\_hidden', dest='decoder\_hidden\_size', default=256, type=int)      args = parser.parse\_args()    if args.cpu\_threads != 0:  torch.set\_num\_threads(args.cpu\_threads)    print('Reward type: %s' % args.reward\_type)  print('Input type: %s' % args.input\_type)  print('Dataset: %s' % args.dataset)  print('Decoder input: %s' % args.decoder\_input\_type)  print('Heightmap\_type: %s' % args.heightmap\_type)  print('Target container: %s' % args.container\_width)  print('Init container: %s' % args.initial\_container\_width)  print('Packing strategy: %s' % args.packing\_strategy)  print('note: %s' % args.note)  os.environ["CUDA\_VISIBLE\_DEVICES"] = args.cuda    kwargs = vars(args)  train\_pack(\*\*kwargs) |

|  |
| --- |
| #  #Helthcare  # Import necessary libraries  import numpy as np  import os  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras.utils import to\_categorical  from itertools import product  from sklearn import metrics    from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import LSTM, Dense    # Set the path to the data directory  PATH = os.path.join('data')    # Create an array of actions (signs) labels by listing the contents of the data directory  actions = np.array(os.listdir(PATH))    # Define the number of sequences and frames  sequences = 30  frames = 10    # Create a label map to map each action label to a numeric value  label\_map = {label:num for num, label in enumerate(actions)}    # Initialize empty lists to store landmarks and labels  landmarks, labels = [], []    # Iterate over actions and sequences to load landmarks and corresponding labels  for action, sequence in product(actions, range(sequences)):  temp = []  for frame in range(frames):  npy = np.load(os.path.join(PATH, action, str(sequence), str(frame) + '.npy'))  temp.append(npy)  landmarks.append(temp)  labels.append(label\_map[action])    # Convert landmarks and labels to numpy arrays  X, Y = np.array(landmarks), to\_categorical(labels).astype(int)    # Split the data into training and testing sets  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.10, random\_state=34, stratify=Y)    # Define the model architecture  model = Sequential()  model.add(LSTM(32, return\_sequences=True, activation='relu', input\_shape=(10,126)))  model.add(LSTM(64, return\_sequences=True, activation='relu'))  model.add(LSTM(32, return\_sequences=False, activation='relu'))  model.add(Dense(32, activation='relu'))  model.add(Dense(actions.shape[0], activation='softmax'))    # Compile the model with Adam optimizer and categorical cross-entropy loss  model.compile(optimizer='Adam', loss='categorical\_crossentropy', metrics=['categorical\_accuracy'])  # Train the model  model.fit(X\_train, Y\_train, epochs=100)    # Save the trained model  model.save('my\_model')    # Make predictions on the test set  predictions = np.argmax(model.predict(X\_test), axis=1)  # Get the true labels from the test set  test\_labels = np.argmax(Y\_test, axis=1)    # Calculate the accuracy of the predictions  accuracy = metrics.accuracy\_score(test\_labels, predictions) |

Q2. **Briefly** describe the first 20 lines of the code **in your own words**. (20 lines means the actual code excluding commented lines).

Transportation and logistics – TAP-Net Transport and Pack using Reinforcement Learning

Introduction:

The project is for finding the most optimal way to pack a place to minimize the space that will be used. To do this, it has many different files codded in python. The main files are a functions file, a file for generating training and testing data, a network structure defining file, a file to load the train and test data and a train file that is the main file that I will be focusing on as it is the main code that is used to run the network.

Code Explanation:

The code starts in the main function, the main function then calls the train\_pack fnction ith kwargs argument. Kwargs is a variable that has all the parameters that the network needs to run the model, these arguments can be settings like train/test settings (line519, trainer.py), data settings, packing settings, network settings and parameters, and task settings.

In the train pack from line 320 to 342 are explaining the dimensions of the input data, it defines static type and dynamic type based on input type, this will be helpful in creation of actor and crtic models.

Line 345 defines variable of use\_cuda. Cuda is a Nvidia toolkit that helps in deep learning, helps code run efficient [cuda nvidia](https://developer.nvidia.com/cuda-toolkit).

Line 347 is a variable about size range that is a list inputting the maximum and minimum size helping in configuration of the model

Line 350 to 450 is concerned with loading or generating datasets based on the kwarg variables inputs

Line 461 is defining a network for actor that takes inputs of lines 320 to 342 and gives output for a decision

Line482 is defining the critic variable that is used to determine the complexity or quality of the problem.

Line 485 and 486 are checking if the network can use Cuda engine, if so, it will use it

Line 490 and 491 is creating instances of train\_data and valid\_data

Line492 and 493 is calling functions reward and render in pack class

Line495 checks if checkpoint is created in kwargs, if so, it takes the previous trained data to train them again for actor and critic networks

Line 504, the train function is called with parameters of actor and critic networks and the kwargs variable

Healthcare and IoMT – Sign Language Translator – model.py

1to 12 import libraries

15, set the path of the directory the program will be in

21 and 22, define the sequences and frames that will be analyzed

31 is a loop that finds the landmarks in the sequences and stores them in landmark array

40 create NumPy arrays to store the landmark array and labels array to help in processing the information later

43 it to split the train and test data into 10% test size and 90% train size

46 the model is sequential

47 the model has an input layer with shape of 10 frames and 126 features, a LTSM long short term memory of 32 units, return sequence true, and activation function of relu

Line 48 and 49 are hidden layers

Line 50 is Dence layer

Line 51 is output layer

Line 54 compile the model, optimization model is Adam, loss function is categorical cross entropy that is good for multi class classification, also some metrics recorded to provide accuracy of model

55 train the model, epochs is the number of turns taken to pass the dataset through the model

59 save model

62 make predictions n test set

64 get the true labels using one hot encoding

67calculate the accuracy and compare it with predictions

Q3. **Share** the link to the code (as a hyperlink in your document so it is easy to click and verify). Verify it is working before submitting the assignment. Dead links will amount to zero marks.

<https://github.com/Juzhan/TAP-Net/blob/master/trainer.py>

<https://github.com/dgovor/Sign-Language-Translator/blob/main/model.py>

References:

Used help from ChatGPT to understand some concepts