CS4287 - Assignment 3

Cathal Kelly - 18244513, Rioghan Lowry - 18226531, Alannah Ryan - 18232132

Reinforced Learning is the machine learning paradigm of choice for this task because it is an effective learning and training method based on rewarding desired behaviors and/or punishing bad ones. In general, reinforced learning is used to be able perceive and interpret its environment, take actions and learn via trial and error. It differs from supervised learning in not needing labeled input and output pairs to be presented and in not needing sub-optimal actions to be explicitly corrected. Instead, it puts the focus on finding a balance between exploration and exploitation. Partially supervised reinforced learning algorithms can combine the advantages and disadvantages of supervised and reinforced learning algorithms. The environment is typically stated in the form of a Markov decision process (MDP), because many reinforcement learning algorithms for this context use dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

The purpose of reinforcement learning is for the agent to learn an optimal, or nearly-optimal, policy that maximizes the "reward function" or other user-provided reinforcement signal that accumulates from the immediate rewards. This is similar to processes that appear to occur in animal psychology. For example, biological brains are hardwired to interpret signals such as pain and hunger as negative reinforcements, and interpret pleasure and food intake as positive reinforcements. In some circumstances, animals can learn to engage in behaviors that optimize these rewards.

The Environment

Selected Game: Space Invaders
Inputs Received from OpenAI gym Environment: OpenAI Gym is a toolkit that provides a wide variety of simulated environments (Atari games, board games, 2D and 3D physical simulations, and so on), so you can train agents, compare them, or develop new Machine Learning algorithms (Reinforcement Learning).

In our case, the inputs received from our OpenAI gym environment are the height and width of the enemy “Invaders”, and the channels in which they move down, which is usually denoted by the rightmost and leftmost remaining enemies.

Control Settings for Joystick:
We could not implement the control settings for the joystick into our Deep Q Network.

Implementation

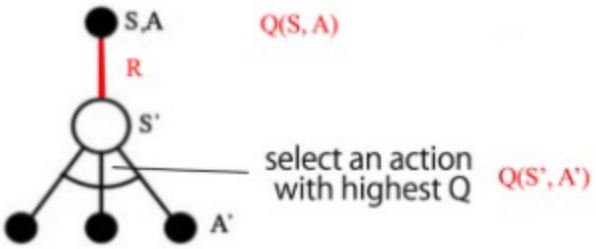
Capture and Pre-Processing of the Data

Input images of the game space representing initial states s, initial probability distribution of actions or Q-values. Before training, they will appear random and sub-optimal. Preprocessing can also include stacking and composition.

We crop images from our gym environment and convert them into 1D tensors. We then want to add stacking and input composition to the preprocessing pipeline. We take two frames(they will be the same) and store them in a deque which automatically will remove older entries as new entries are introduced. We use the preprocessed maxframe to fill out the deque at first. As we progress through the episode, we create new maxframes by taking out the new frame element-wise and maximum summing it with the most recent entry in the deque. We then append the new maxframe to our deque.

The Network Structure

The structure of our DQN is that of a classical DQN architecture which uses a single NN to predict directly the value of all possible actions $Q\theta(s,a)$. The value of an action depends on two factors: the value of the underlying state s: in some states, all actions are bad, you lose whatever you do.



We select an action a and evaluate our decision in the gym environment to receive information on the new state s', the reward r, and whether the episode has been finished,This would be stored in a buffer in the list form <s, a, r, s', d> or <state, action, reward, new state, d?> and do this a preset number of times to build up a large enough buffer dataset.

We then move to create our target-y values, R' and A' that are required for the loss calculation. R' is simply discounted from R, and we get A' by feeding S' into our network. We then calculate the loss to train the network..

```
def build_model(height, width, channels, actions):
    model = Sequential()
    model.add(Convolution2D(32, (8,8), strides=(4,4), activation='relu', input_shape=(3,height, width, channels)))
    model.add(Convolution2D(64, (4,4), strides=(2,2), activation='relu'))
    model.add(Convolution2D(64, (3,3), activation='relu'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(actions, activation='linear'))
    return model
```

We create a model using Sequential. We have three 2D Convolutional Layers to reduce the inputs gained from the environment. We use an activation of relu for them all. The strides of the layers move from a 4 to a 2 to none. We follow up with a Flatten layer to put any multi-dimensional tensors into a single dimension.
Finally we add three Dense layers, using an activation of relu for the first two and one more using linear. The first two have units of 512 and 256 respectively, and the final layer takes our actions for the number of units.

The Q Learning Update Applied to the Weights

Deep Q-Learning agents use Experience Replay to learn about their environment and update the Main and Target networks. To summarize, the main network samples and trains on a batch of past experiences every 4 steps. The main network weights are then copied to the target network weights every 100 steps. In our network we adjusted the weights to every 1000 steps and the steps warm-up to 100.

Other Concepts of Significance

```
def build_agent(model, actions):
    policy = LinearAnnealedPolicy(EpsGreedyQPolicy(), attr='eps', value_max=1., value_min=.1, value_test=.2, nb_steps=1000)
    memory = SequentialMemory(limit=1000, window_length=3)
    dqn = DQNAgent(model=model, memory=memory, policy=policy, enable_dueling_network=True, dueling_type='avg', nb_actions=actions, nb_steps_warmup=100)
    return dqn
```

```
dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-4))
dqn.fit(env, nb_steps=1000, visualize=False, verbose=2)
```

Results

We managed to get results before and after training our DQN. The results however do not show much of an improvement and this could mean that our DQN is not learning. In fact if you look at the results it seems the training has made the DQN worse. The results of pre-training are shown on the left below and the post training results are on the right.

Pre-training

```
Episode:1 Score:75.0
Episode:2 Score:210.0
Episode:3 Score:210.0
Episode:4 Score:155.0
Episode:5 Score:110.0
```

Post-training

```
Episode:1 Score:145.0
Episode:2 Score:65.0
Episode:3 Score:45.0
Episode:4 Score:105.0
Episode:5 Score:50.0
```

Evaluation of the Results

How Does One Evaluate the Performance of the RL Agent?

One would be able to check the reward of the DQN playing the game before training it and then one could compare them to the results after training. To compare the results one could have put them into a graph with one line showing before and one line showing after the training. We focused our time on trying to get the DQN to learn and so we do not have any graphs to show.

Is the Agent Learning?

Much to our dismay, our agent is not learning.

References

https://stackoverflow.com/questions/60690327/typeerror-keyword-argument-not-understood-inputs

https://stackoverflow.com/questions/67656740/exception-rom-is-missing-for-ms-pacman-see-https-github-com-openai-atari-py

https://github.com/openai/atari-py#roms

https://stackoverflow.com/questions/14844687/invalid-character-in-identifier

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten

https://github.com/openai/atari-py#roms

https://stackoverflow.com/questions/67656740/exception-rom-is-missing-for-ms-pacman-see-https-github-com-openai-atari-py

https://github.com/NicMag/Reinforcement-Learning/blob/master/GYM_SPACE_INVADERS.py

https://www.google.com/search?q=NameError%3A%20name%20%27Monitor%27%20is%20not%20defined+site:stackoverflow.com

https://gym.openai.com/

https://stackoverflow.com/questions/64077941/nameerror-name-screen-is-not-defined

https://stackoverflow.com/questions/51089334/what-is-the-difference-between-tf-keras-layers-versus-tf-layers/51089448

https://stackoverflow.com/questions/57963649/error-when-using-the-keras-api-on-convelutional-layers

https://towardsdatascience.com/optimized-deep-q-learning-for-automated-atari-space-invaders-an-implementation-in-tensorflow-2-0-80352c744fdc

https://keras.io/api/layers/convolution_layers/convolution2d/

https://stackoverflow.com/questions/47822154/layer-conv2d-3-was-called-with-an-input-that-isnt-a-symbolic-tensor

https://towardsdatascience.com/optimized-deep-q-learning-for-automated-atari-space-invaders-an-implementation-in-tensorflow-2-0-80352c744fdc