Assignment Deep Learning AI23 HT24

1 Introduction

In this assignment you will develop and train a Reinforcement Learning algorithm to play the Atari 2600 game "Space Invaders", originally released in 1980 (Taito's original arcade version was released 1978). If you would like to try the real thing, there's a miniature Atari 2600 console in the lounge room in one end of the ITHS premises.

To solve this assignment, you can use cloud compute such as Kaggle or Colab, but this task can also be run on a home computer (powerful laptop or gaming rig) – although it might take quite a long time (hours to days, depending on how good you want the result to be).

(a) Box art for Space Invaders, 1980

(b) Gameplay

The algorithm in question is Deep Q-Learning, as with so many other things in this area originally developed by Google DeepMind. Deep Q-Learning was an early breakthrough in the current wave of machine learning. Game playing systems were suddenly routinely able to beat humans in most games, including Chess and Go. Computer games turned out to be somewhat more challenging from an interface perspective. In this assignment the use of Q-Learning will highlight the inner workings of neural networks as implemented in `tensorflow`. The original paper describing this approach can be found here: DQNNaturePaper.pdf(2015)

2 Atari 2600 Emulation

The package `gymnasium`, a fork of OpenAI's `gym`, offers full emulation of various software environments suitable for reinforcement learning. Among them is an Atari 2600 emulator with an out-of-the-box toolchain that makes experimenting with game playing RL easy! Use `pip install gymnasium[atari] ale-py` to install the library. Here's a script that, when you

have trained and saved a model, will load said model and play a game of Space Invaders on screen:

```
import keras
import gymnasium as gym
import ale_py
from gymnasium.wrappers.frame_stack import FrameStack
from gymnasium.wrappers.atari_preprocessing import AtariPreprocessing

gym.register_envs(ale_py)

model_file = "<path_to_your_model_here>"
agent = keras.models.load_model(model_file)

env = gym.make("SpaceInvadersNoFrameskip-v4", render_mode="human")
env = AtariPreprocessing(env)
env = FrameStack(env, 4)

state, _ = env.reset()
done = False
while not done:
    # first convert to a tensor for compute efficiency
    state_tensor = keras.ops.convert_to_tensor(state)
    # shape of state is 4, 84, 84, but we need 84, 84, 4
    state_tensor = keras.ops.transpose(state_tensor, [1, 2, 0])
    # Add batch dimension
    state_tensor = keras.ops.expand_dims(state_tensor, 0)
    # 'predict' method is for large batches, call as function instead
    action_probs = agent(state_tensor, training=False)
    # Take 'best' action
    action = keras.ops.argmax(action_probs[0]).numpy()

    state, reward, done, _, _ = env.step(action)
```

A Q-learning model that is suitable for this problem is the following (from the DeepMind paper):

```
num_actions = 6 # see atari gym documentation for SpaceInvaders

def create_q_model():
    return keras.Sequential(
        [
            layers.Input(shape=(84, 84, 4)),
            layers.Conv2D(32, kernel_size=8, strides=4, activation="relu"),
            layers.Conv2D(64, kernel_size=4, strides=2, activation="relu"),
            layers.Conv2D(64, kernel_size=3, strides=1, activation="relu"),
            layers.Flatten(),
            layers.Dense(512, activation="relu"),
            layers.Dense(num_actions, activation="linear")
        ]
    )
```

The last axis of input here is *not* colour information, but a set of four frames. The network sees four frames of gameplay at a time, and since the game renders at 29.97 FPS (so-called beam-chasing on analog TVs), that's roughly a human-equivalent reaction-time of 125 ms or eight actions per second. Quite fast, but not at the limit of human capability. Professional StarCraft players are consistently faster than that, many times more in bursts.

3 Your Task

Implement the Deep Q-Learning approach given in the example from the lecture and train a model. You will have to adapt the example here and there to work with Space Invaders. DeepMind ran their training for 50 million frames – we will not be able to run anything close to that with the compute capabilities at hand. Consider optimizations or other time-saving measures. At the end, just for fun, we will collect all the trained models and run an AI-tournament.

The limiting factor with the original Deep Q-learning is how very expensive it is to train – you have to simulate the game during training! Another factor is the quality of the original observation space. For example, consider the example of Chess. When the original observation space was constructed it included literally every rated tournament game in the history of modern Chess. The best of the best of human proficiency in a classic game! Later approaches started using detailed reward-mappings extracted from known Chess games – referred to as "teaching the computer the rules first" at the time. We don't have access to anything like that for Space Invaders.

You will need to save the model often during training and decide on what conditions you want to call "success". If you do run it for something like 24 hours on a relatively powerful computer, the network should reach human-level ability, or better.

Submit your code as link to a repo and upload your trained `keras` model to ITHS. Better to save it there than in github, but either is ok if there's a problem with one or the other.

Document in the repo what you have done – for example as a `README.md` or in a jupyter notebook as you see fit. Rather than having a mandatory form, you should create something using the skills you have built so far and present it through the repo in your hand-in – make it your own and present it through some kind of report, app or documentation. Make it obvious to me what I'm supposed to be looking at!