Advanced Machine Learning Assignment — 2024

Submission Submit (1) a report as a PDF[1] and (2) your code and results data zipped into one single file, without the report. To zip your code and data, use zip or tar/gzip, do not use rar. Please **include the signed and completed cover sheet in your report that you can find at the end of the document**.

Submission is due on 23 Oct 2024, 11:59pm.

MiniPong

"Pong" was an early computer game (first released in November 1972), and became such a classic that it is now part of the permanent collection of the Smithsonian group of museums [https://en.wikipedia.org/wiki/Pong].

In this assignment, we will work with data and a simulation of a simplified version of "Pong". Two objects appear on the field: a "ball" $^{+}$ and a paddle^{-}, which can occupy different positions but only along the bottom row. The pixels of the two objects are represented by the values -1 and 1, respectively, while the background pixels have the value 0. The two markers in the top corners are fixed (with values -1 and $+1$, respectively) and appear in every frame.

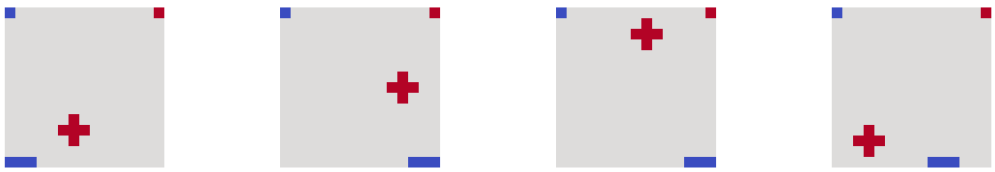

Figure 1: 4 frames from our data, using $+1$ valued pixels for $+$, -1 for the $-$ paddle.

Preparation Download the `sprites.py` and `mini_pong.py` Python files. The program `sprites.py` will create datasets of `MiniPong` screenshots for your first task. The class `MiniPongEnv` in the second file uses `gymnasium` to implement the simulation for your RL tasks. You can use `MiniPongEnv` like any other `gymnasium` environment:

```
from mini_pong import MiniPongEnv
# Create instance of the environment at given level and size
env = MiniPongEnv(level=1, size=5)
```

You can also run the file directly to view a brief demo and examine how it works.

The `level` parameter determines the amount of information returned, while `size` specifies the size of the game (in terms of the number of different paddle positions). Both the ball^{+} and the paddle^{-} are 3 pixels wide and cannot leave the field. A game with size 5 is represented by a 15×15 pixel grid, and the ball's x- and y-coordinates can range from 1 to 13. The paddle can occupy 5 different positions (from 0 to 4).

[1] Yes, you have to submit a separate PDF report even if you use Jupyter notebooks for your code. Notebooks are not ideal for long training runs, we suggest using `.py` scripts instead.

Task 1: Train a CNN to predict object positions

15 points

The Python program `sprites.py` generates training and test sets of MiniPong scenes: `trainingpix.csv` (676 samples) and `testingpix.csv` (169 samples). Each row represents a 15×15 screenshot, flattened in row-major order. The corresponding labels are stored in separate files, `traininglabels.csv` and `testinglabels.csv`. Each label file contains three values per example (x, y, z): the x- and y-coordinates of the $+$ marker, which range from 1 to 13, and z, the position of the $-$ paddle, which ranges from 0 to 4. Your task is to create a CNN to predict the labels from the inputs.

Steps

1. Run the `sprites.py` code to generate the datasets.
2. Create a CNN that predicts the x-coordinate of the $+$ marker.
   - You may use an architecture similar to what we used for classifying MNIST. However, be aware that the input dimensions and outputs differ, so you will need to make some modifications.
   - Consider normalising or standardising the data if it helps improve the training.
3. Create a CNN that predicts all three outputs (x, y, z) from each input.[2]
4. Describe your approach and your results in your report.

What to submit for task 1

- **Submit**: the Python code for your solutions (both versions), the predicted labels for the test sets (2 csv files), description of approach and your results.
- Your approach to solving the tasks and your explanations are more important than the absolute performance of your code. If the models do not perform as expected, submit your steps and describe the specific issues you encountered.

Provide a brief description of your model architectures and the steps you took to create them. What approach did you use? In your description, also include answers to the following questions:

- What loss function did you use and why? What changes did you make for the second model?
- For how long did you train both models (number of epochs and total time)? What is the performance (accuracy) for each, on the test set?

[2] As an intermediate step, consider training three separate networks, one for each output. Then, attempt to merge these into a single network with three outputs.

Task 2: Train a convolutional autoencoder

10 points

Instead of predicting positions, create a convolutional autoencoder that compresses the MiniPong screenshots to a small number of bytes (the encoder), and then transforms them back to the original form (in the decoder part).

Steps

1. Create and train an (undercomplete) convolutional autoencoder using the training dataset from the first task.
2. It is up to you to choose the architecture of the network and the size of the representation $\mathbf{h} = f(x)$. The goal is to learn a representation that is smaller than the original image while still leading to recognisable reconstructions.
3. For the encoder, you may use the same architecture as in the first task, but you cannot use the labels for training. Alternatively, you may choose a completely different architecture.
4. (No programming): In theory, what is the absolute minimal size of the hidden layer representation that allows perfect reconstruction of the original image?

What to submit for task 2: the Python code for your solution and a description of it.

- In your report, provide a brief description of your steps to create the models and your observations. Describe what you did (e.g., what loss function you used, the size of the encoded image in your architecture, and the number of training steps).
- Include screenshots of 1-2 output images next to the original inputs (e.g., select one good and one bad example).

Task 3: Create an RL agent for MiniPong

15 points

The code in `mini_pong.py` provides an environment similar in style to other `gymnasium` environments, with different levels and sizes. Below is a description of the environment dynamics:

- The ball^{+} marker moves diagonally at each step. When it hits the paddle or a wall (top, left, or right), it reflects.
- The agent controls the paddle^{-}, moving it one 3-pixel slot per step. The agent has three actions available: do nothing, move left, or move right. The paddle cannot move outside the boundaries.
- The agent receives a positive reward when the $+$ marker reflects off the paddle. In this case, the $+$ may also move randomly by 1 or 2 pixels left or right.
- An episode ends when the $+$ reaches the bottom row without reflecting off the paddle.

In the *level 1* version of the environment, the observed state (the information available to the agent after each step) consists of a single value, dz: the relative position of the $+$ marker to the centre of the paddle^{-}. This value is negative if the $+$ marker is on one side, and positive if on the other.

For this task, you should use `level=1` and `size=5`. You can choose to normalise the state information to values between -1 and 1 by setting `normalise=True`, or use the original (integer) values in the range $-13...13$ by setting `normalise=False`.

```
env = MiniPongEnv(level=1, size=5, normalise=True)
```

Steps

1. First, create a simple Python program (without RL) that plays MiniPong. The program should select an action based on the value of dz. You may use and modify the code at the end of `mini_pong.py` to complete this task.
2. Create an RL agent (tabular or deep) that learns to play MiniPong. If you use a TD-learning approach with ϵ-greedy action selection, set $\epsilon = 1$ initially, and slowly reduce it during training to a minimum of 0.1.
3. Run your training, resetting the environment after each episode. Store the sum of rewards for each episode. After or during training, plot the total sum of rewards per episode. This plot — the *Training Reward* plot — indicates the extent to which your agent is improving its cumulative reward. It is your decision when to stop training. It is not required to submit a perfectly performing agent, but show how it learns.
4. After deciding the training is complete, run 50 test episodes using your trained policy with $\epsilon = 0.0$ for all 50 episodes. Reset the environment at the beginning of each episode. Calculate the average of the sum of rewards per episode (call this the *Test-Average*) and the standard deviation (the *Test-Standard-Deviation*). These values indicate how your trained agent performs.
5. (No programming required) Describe: In level 2 of MiniPong, the observed state would consist of 2 values instead of 1: it now contains the ball's y-coordinate as well as the relative position dz (that we also get in level 1). Would this additional information help or hinder learning?

What to submit:

- Submit the Python code for your solutions (both the manual strategy and the RL agent).
- In your report, describe your solution, mention the *Test-Average* and *Test-Standard-Deviation*, and include the *Training Reward* plot described above. Indicate how many episodes you ran before stopping training and how long the process took.
- Include your response to the question about the *level 2* version of the environment.

Task 4: Create an RL agent for MiniPong (level 3)

10 points

In the *level 3* version of the game, the observed state (the information available to the agent after each step) consists of three values: y, dx, and dz. These represent the following:

- y: the ball's y-coordinate.
- dx: the change in the ball's x-coordinate from the previous step to the current step.
- dz: the relative position of the ball to the centre of the paddle (same as in previous levels).

Similar to previous tasks, you can initialise MiniPong in two ways for this task:

```
env = MiniPongEnv(level=3, size=5, normalise=True)
```

or

```
env = MiniPongEnv(level=3, size=5, normalise=False)
```

In the first version, `step()` returns normalised values for y and dz (between -1 and 1), while in the second version these values are unnormalised. The dx values are always unnormalised (but should be -1 or 1 in most cases, except after the paddle has been hit).

Steps

1. Create a neural network-based RL agent that finds a policy using all *level 3* state information. Tip: typical values for the learning rate α (e.g., when using Adam) range from 0.0001 to 0.01, and the discount factor γ typically ranges from 0.9 to 0.999.
2. Choose an algorithm (e.g., deep TD or deep policy gradient).
3. Train an agent that achieves a running reward > 300.
4. Keep the model small to avoid long training times. Start with a single hidden layer and a small number of units.
5. Write a description explaining your approach and its performance. If some (or all) of your attempts are unsuccessful, describe what did not work and which changes made a difference.

What to submit:

- Submit the Python code for your solutions, and a description of your approach.
- In your report, describe your solution, mention the *Test-Average* and *Test-Standard-Deviation*, and include the *Training Reward* plot as described above.

Tips

1. RL tasks can take time to show progress. If the agent is not learning, experiment with learning rates. For Adam, start with values between `5e-3` (faster) and `1e-4` (slower).
2. Even if learning does not work as expected, use the task to demonstrate your own understanding. Describe the ideas you tried and submit your code, along with an explanation of any issues encountered.

Bonus questions (extra points)

I can do it (Neural Networks): Train a neural network to predict the dz variable from the input images.

Bring it on (Pong level 3): Modify the learning process or adjust the reward structure in the environment to discourage the agent from moving the paddle unnecessarily. Compare the learned policies before and after your changes.

Hardcore (Autoencoder-RL): Train an autoencoder where the encoded image (the middle layer) is used as input to an RL agent that successfully plays MiniPong.

Nightmare (Level 0 Challenge): Solve `MiniPong(level = 0)` using PyTorch. In this level, the state is represented as the difference between the current image and the previous image. Check out Karpathy's blog for tips.

Brief overview mini_pong.py

MiniPong environment interface The `MiniPongEnv` class provides a simplified version of Pong implemented using the `gymnasium` framework. The environment supports multiple levels of state information and varying paddle sizes, allowing flexibility for different RL tasks.

- **Action space:** The agent can choose between three discrete actions: move the paddle left (1), move the paddle right (2), or do nothing (0). The paddle moves one 3-pixel slot per action and cannot go beyond the boundaries.
- **Observation space:** The format of the observation depends on the chosen level:
  - Level 0: A 15×15 pixel difference image between the current and previous states.
  - Levels 1-5: A vector of length equal to the level number, containing various state features such as the ball's position and velocity relative to the paddle.
- **Reward:** The agent receives a positive reward when the ball reflects off the paddle. The episode ends when the ball reaches the bottom row without hitting the paddle.
- **Environment dynamics:** The ball moves diagonally at each step. When it collides with the walls or the paddle, it changes direction based on reflection rules. If the ball hits the paddle, it may randomly adjust its horizontal direction by 1-2 pixels.
- **Reset and step functions:**
  - `reset()`: Resets the environment to the initial state, returning the first observation.
  - `step(action)`: Executes the specified action, returning a tuple containing the next observation, reward, a boolean indicating if the episode has ended, and an empty info dictionary.
- **Rendering:** The `render()` method provides a visual representation of the environment using `matplotlib`.

If you put the `mini_pong.py` file into your working directory, you can import the class like this:

```
from mini_pong import MiniPongEnv

env = MiniPongEnv(level=1, size=5)
state = env.reset()
...
```

Assignment Cover Sheet

School of Computer, Data, and Mathematical Sciences

Student Name	
Student Number	
Subject	INFO7001: Advanced Machine Learning
Title of Assignment	Project
Due Date	23 Oct 2024
Date Submitted	

DECLARATION

I hold a copy of this assignment that I can produce if the original is lost or damaged.

I hereby certify that no part of this assignment/product has been copied from any other student's work or from any other source except where due acknowledgement is made in the assignment. No part of this assignment/product has been written/produced for me by another person except where such collaboration has been authorised by the subject lecturer/tutor concerned.

Signature: .....

(Note: An examiner or lecturer/tutor has the right not to mark this assignment if the above declaration has not been signed)

	Task 1	Task 2	Task 3	Task 4	Bonus	Total
Mark						
Possible	15	10	15	10	?	50

The maximum points possible for this assignment is 50 (including any bonus points).