CS601 AY24-25T3 **Submission due at 23:59 on 29/6/2025**
Assignment 2
15/6/2025

Instructions: 1. There are two questions. Each is self-contained, and all sub-questions require coding. 2. Please make a single pdf document to show your solution for all sub-questions (show either coding screenshots if the question asks for, and/or theoretical justification). 3. The solution should be typeset using professional softwares (word, keynote, latex, etc.). 4. You can consult the web resources to do quick research on related methods. However, usage of generative AI tools (such as chatGPT) is NOT allowed for any task related to this assignment.

Question 1: Approx. Q-Learning [20 points]

A robotic drone must learn to navigate a windy 2D open field to reach a charging station. The environment has continuous state features—**position (x, y)** and **wind strength (w)**—and discrete actions: {`North, South, East, West`}. The field has varying wind conditions that affect movement (e.g., stronger wind in the east slows eastward movement). The reward is +10 for reaching the charging station and -1 for each time step to encourage efficiency.

You use **tile coding**[1] to discretize the continuous state space and implement **Approximate Q-learning**. No neural networks or DQNs are allowed.

(a) Feature Representation and Update Equation [5 points]

You have been given the analytical update equations by which you would implement for $Q(s, a)$ for a given experience (s, a, r, s') using tile coding. Assume that tile IDs corresponding to the state s and action a are $\phi_1(s, a), \phi_2(s, a), \ldots, \phi_k(s, a)$. Please complete the coding lines for "Compute discrete bin indices" in drone_navigation_subparts.ipynb.

(b) Implementation and Screenshot [7 points]

Implement the `ApproxQLearning()` function (by filling up the missing lines) in drone_navigation_subparts.ipynb. Capture a screenshot of your implemented function and paste it in the solution PDF. You may assume a single grid for simplicity (multi-tiling is not required). Any reasonable implementation will be accepted for full marks.

(c) Hyperparameter Tuning [5 points]

Tune relevant hyperparameters in the code to improve the drone's performance. List below the hyperparameters you adjusted and their best-performing values. Briefly describe any dynamic adjustment strategies you used (e.g., decay of exploration rate):

Hyperparameter Name	**Tuned Value / Strategy**
Learning Rate (α)	
Exploration Rate (ϵ)	
Discount Factor (γ)	
Number of Tiles (x-dimension)	
Number of Tiles (y-dimension)	
Initialization of Weights	
Other (specify)	

[1] for which you will need to do a quick research to understand what is the tile coding method. You may start from this link: https://github.com/criteo-research/tf-tile/blob/master/doc/Tile-Coding-An-Efficient-Sparse-Coding-Method-for-Real-Valued-Data.md

(d) Performance Evaluation [3 points]

What is the best mean reward over 100 test episodes after training? Show logs in your notebook to support the number you report. Please note that we will grade your results based on the class average.

Question 2: Deep Q-Learning [20 points]

You will optimize the DQN agent for Pacman provided in pacman.ipynb. The implementation includes:

- **State**: Raw pixels $(280, 316)$ with 2-frame stacking
- **Actions**: {`Up, Down, Left, Right`}
- **Rewards**: +10 (pellet), +50 (power pellet), +200 (eat ghost), -1 (wall hit), -100 (death)
- **Network**: 3 Conv layers + 2 FC layers

Your task is to improve the agent's learning speed and final performance through careful hyperparameter tuning and implementation enhancements.

(a) Improve Algorithm [8 points]

The current implementation provides a working baseline, but there is still room in accelerating the training (i.e., reducing the training time till loss convergence). Consider exploring enhancements such as:

- Adjusting the replay buffer size or implementing a warm-up phase before learning starts
- Modifying the frame stacking strategy (currently 2 consecutive frames) to better capture motion information
- Tuning the batch size or learning frequency for more stable updates
- Experimenting with different loss functions (the code includes both MSE and Huber loss options)
- Adding simple reward shaping to guide the agent's learning

You are not limited to the above directions; any creative ideas to improve the agent's training speed are strongly encouraged. The key is to make a meaningful change and **justify why your approach should accelerate convergence**. You should report: 1) a screenshot of your code modifications with changes clearly highlighted, 2) a clear explanation of what you changed and why it helps speed up training, and 3) evidence that your change reduced training time or improved convergence speed.

(b) Hyperparameter Tuning [6 points]

Fine-tune at least TWO hyperparameters to improve the agent's convergence speed:

Hyperparameter	**Default Value**	**Description**
Epsilon start	1.0	Initial exploration rate
Epsilon end	0.01	Final exploration rate
Epsilon decay	0.999	Decay rate per episode
Learning rate	0.00025	Neural network learning rate
Batch size	128	Samples per training step
Buffer capacity	10000	Experience replay buffer size
Gamma	0.99	Discount factor
Target update frequency	1000 steps	How often to sync target network
Learning frequency	Every step	How often to train

You should provide a table showing: 1) the hyperparameter name, 2) the original value, 3) your tuned value, and 4) the observed effect. You should also provide clear justification for each hyperparameter choice based on your observations.

Hyperparameter	**Original**	**Tuned**	**Observed Effect**

(c) Performance Analysis [6 points]

Demonstrate that your improvements were effective through quantitative analysis:
1. Training Metrics: Show loss over episodes/steps demonstrating improved convergence. Compare baseline vs. optimized training curves to show faster convergence.
2. Test Performance: Run 10 test episodes with your optimized agent and report the following metrics in a table: average score, best score, worst score, and standard deviation. Note that a well-tuned agent should achieve at least an average score of 70 within a reasonable training time. Compare these against your baseline results.

Metric	**Baseline**	**Optimized**
Average Score		
Best Score		
Worst Score		
Standard Deviation		
Training Time to 70 score		