COMP3411/9814 Artificial Intelligence
Term 1, 2025

Assignment 2 – Neural Nets, Reinforcement Learning

Due: Tuesday 22 April, 10pm
Marks: 25% of final assessment

In this assignment you will be examining two machine learning methods. Part A requires you to implement a neural network for classification. You will also implement for Part B the method of Q-learning, with an extension to allow the learner to receive "advice" from a "teacher", actually another agent trained by reinforcement learning.

For both Part A and Part B you will implement and submit code plus a written report.

Note: You will make separate submissions for Parts A and B ! Submission details are at the end of this specification.

Part A: Neural Network Classification (11 marks)

Problem overview

Credit decisions are a long-standing application area for machine learning. If you have ever supplied information on your financial history when applying for a loan (or other credit facility) from a financial institution then your data has probably been run through a model trained by machine learning.

In this part of the assignment, you will train an artificial neural network on historical data to model the risk of default on loans to small businesses. The task is framed in terms of *classification*, i.e., predicting whether the outcome of a loan is likely to result in default, or not.

LendingClub was an early mover in peer-to-peer (P2P) marketplace lending. The idea behind P2P lending was to match borrowers with investors who fund loans. Loans could be made for various purposes. The company is no longer involved in P2P lending but has released some of its historical data for research. There are many versions of these datasets available online. Note: for this assignment we will use a version of a dataset for small business loans which is not generally available for download.

Dataset

The small business loan dataset used in this assignment has 17 variables (features):

Index	Variable	Description
1	term_36m	loan term of 36 months
2	term_60m	loan term of 60 months
3	grade_n	LendingClub assigned loan grade
4	sub_grade_n	LendingClub assigned loan sub-grade
5	revol_util_n	Sum of revolving credit utilisation
6	int_rate_n	Interest Rate on the loan
7	installment_n	Monthly payment owed
8	tot_hi_cred_lim_n	Total installment high credit/credit limit
9	emp_length_n	Employment length in years
10	dti_n	Calculated from total monthly debt payments
11	avg_cur_bal_n	Average current balance of all accounts
12	all_util_n	Balance to credit limit on all trades
13	acc_open_past_24mths_n	Number of trades opened in past 24 months
14	annual_inc_n	Self-reported annual income
15	loan_amnt_n	Listed amount of the loan
16	cover	Derived feature
17	Outcome	Loan outcome: default (1) or discharged (0)

Target variable

In this dataset the 'Outcome' feature is used as the target variable. Since Outcome has only two values the task is *binary* or *two-class* classification. Like many real-world datasets the class ratio is *unbalanced*. In this case, the majority of loans were discharged (paid back) with relatively few examples of loans that defaulted. In the dataset, default is represented as 1, with 0 meaning the loan was successfully discharged.

Classification task

In this classification task, the goal is for the neural network to predict whether the loan will default or not, based on the 16 features characterising each loan.

Solution overview

For Part A you will implement a neural network for the classification task, evaluate its performance, submit a short written report describing the network and its performance, and submit your neural network for automarking on a (random sample of) a hidden test set.

Getting started

Download the file `hw2nn.zip` from this directory:
`https://www.cse.unsw.edu.au/~cs3411/25T1/hw2/`

(or download it from here).

Unzip the file and change directory to `hw2nn`:

```
unzip hw2nn.zip
cd hw2nn
```

You should see in this directory the following files:

```
hw2main.py student.py hw2nn_data.csv
```

Model training

(a) The downloaded data you will use for training is in `hw2nn_data.csv`

(b) Implement your neural network using PyTorch in `student.py` by defining its architecture and hyperparameters. This includes: loss function, optimiser, batch size, learning rate, and number of epochs. It is recommended that your network has fewer parameters than the number of samples divided by 10.

(c) Run `hw2main.py` to train the neural network model. Monitor the evolution of the loss values during training.

Validation set to evaluate model performance

In `student.py` you can specify a split of the dataset `hw2nn_data.csv` into training and validation sets. The default is to use 80% of the data for training and 20% for validation, but you can change this.

(d) Train on the training set and use the validation set to estimate model accuracy.

(e) Repeat steps (b), (c) and (d) to see if you can improve the performance of your model.

Results from evaluating your classification model

When you think you have achieved the best accuracy you can through training, you need to collect the results as follows and include them in your report.

(f) Create a plot showing the accuracy (y-axis) versus the number of epochs (x-axis) on the training set. Save the plot for inclusion in the report.

(g) Use your model to predict the class on the validation set:

  - Compute and plot a confusion matrix. Calculate this in terms of predictions where the positive class is 'Outcome' = 1, i.e., default.
  - Note the "class ratio", i.e., the ratio of the number of positive examples to the size of the entire dataset.

  Include your confusion matrix, together with a brief discussion of these results, in the report.

Additional notes

- You will need to set the random seed to ensure that your results are reproducible
- Note: the provided code will automatically save your model weights in the file `model.pth`
- In step (f), you only need to plot the result(s) for your best-performing neural network configuration.

Questions These questions should be answered in your written report.

Question A1 [2 marks] Describe your neural network architecture, including, but not limited to: the number of inputs, number of outputs, number of hidden layer(s), if any, an all hyperparameters. Justify the choices made, and any experimentation that led to those.

Question A2 [1 mark] You will see in the file "`hw2main.py`" that scaling of dataset values is applied using the `sklearn` method StandardScaler. You can remove this scaling by setting `scale_inputs` to `False` in `student.py` Try doing this, and compare what happens to performance with and without scaling. Explain the difference (if any) in outcomes you observe.

Marking outline:

Training and validation set performance [3 marks] These marks are given for the plot obtained in step (f), and the confusion matrix and discussion in step (g), above. These should be provided in the report.

Automarked test set performance [3 marks] When you submit, your model will be run on a random sample from a separate validation set, and will report results. Note that this gives an **indication** of the performance your model will obtain on the test set, but the final performance may vary from this. Note that the actual marks will be allocated based on performance on the test set once submission has closed.

Answers to questions [3 marks] Answers to Questions A1 and A2 above.

Code quality [2 marks] This will be based on criteria of working code, well-structured and with good style, assessed by human markers.

What to submit for Part A
You will need to submit three files: `student.py` which contains your network, `model.pth` which contains the model weights, and a file called `report.pdf` with your results (plot, confusion matrix plus commentary) and answers to the questions. Full submission instructions at the end of this specification.

Part B: Reinforcement Learning with a Teacher (14 marks)

For Part B of this assignment, you will implement the Q-learning reinforcement learning algorithm. You will also implement a variant of this algorithm where an existing agent acts as a "teacher" to help train a new agent.

The environment for these tasks is a grid world consisting of a 10×10 grid within which the agent must navigate from a random starting position to a designated goal, while avoiding fixed obstacles.

You will first develop the Q-learning algorithm, implementing action selection policies that allow an agent to choose actions using an ε-greedy approach to balance exploration and exploitation. After training the agent, you will introduce the teacher-student framework. In this setup, a pre-trained agent (the teacher) provides advice to a new agent (the student) during its training.

The teacher's advice will be configurable in terms of its **availability** (probability of offering advice) and **accuracy** (probability that the advice is correct). You will evaluate the impact of teacher feedback on the student's learning performance by running experiments with varying levels of availability and accuracy. This involves comparing how the agent learns with varying degrees of teacher guidance. The goal is to understand how such teacher interactions influence the learning efficiency of the new agent.

Environment

For detailed information about the environment including setup instructions, key functions, agent movement and actions, movement constraints, and the reward structure please refer to the **Environment User Guide** provided separately as a PDF file along with the `env.py` file. Ensure you familiarize yourself with the environment before proceeding with the assignment.

We have made two small changes to the environment: first, the arrangement of obstacles has been altered, as shown in Figure 1; second, the reward for reaching the goal has been increased to 80.

Download the file `hw2grid.zip` from this directory:
`https://www.cse.unsw.edu.au/~cs3411/25T1/hw2/`
(or download it from here).

Unzip the file and change directory to `hw2grid`:

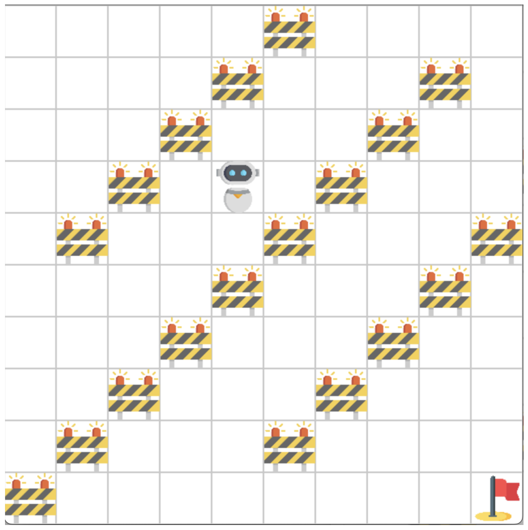

Figure 1: Grid World Environment, showing the agent, target and obstacles.

```
unzip hw2grid.zip
cd hw2grid
```

You should see in this directory the following files:

```
teach.py env.py utils.py
env_doc.pdf images.zip
```

Important: Make sure that `env.py`, `utils.py`, and the (unzip'ed) `images` folder are placed in the same directory as your solution. These files are needed in order to import the environment, use the utility function, and render the environment visuals properly.

Task 1: Implement Q-Learning (4 marks)

Your first task is to implement (in the file `teach.py`) a function `train_agent()` which will train an agent by Q-learning to operate in the grid-world environment. This function should take a parameter called `max_total_steps` (you may add additional parameters, if you wish). Training consists of a number of episodes. Each episode begins with the command:

```
state = env.reset()
```

which will initialize the environment and place the agent in a random location. Actions are executed by calling:

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

Each episode should terminate when either the goal has been reached (indicated by the return value `done` being `True`) or the maximum of 100 steps per episode has been exceeded. At the end of each episode, your code should check whether the total number of steps executed across all episodes is less than `max_total_steps`, in which case a new episode should begin; otherwise, your code should exit the loop and terminate the training. For Task 1, the value for `max_total_steps` should be 50000.

Your function should return four values: `q_table`, `reward_per_episode`, `success_rate`, `avg_steps_per_episode`, where:

- **q_table** is the Q-table of the agent after training.
- **reward_per_episode** is an array, indexed by episode, specifying the total reward received during each episode.
- **success_rate** is the percentage of episodes in which the agent successfully reached the goal.
- **avg_steps_per_episode** is the total number of steps divided by the number of episodes.

In this assignment, you have the flexibility to choose the learning rate (α), discount factor (γ), action selection method (e.g., ε-greedy, softmax) and exploration rate (e.g., fixed ε, initial ε and ε-decay rate, or temperature in softmax). However, the maximum number of steps per episode must be fixed at 100. After training the agent, you are required to:

(a) Generate a plot that displays the total reward for each episode (and include it in your Report),

(b) Include in your Report the Average Reward per Episode, Success Rate, and Average Steps per Episode, using the following formulas:

- **Average Reward per Episode**

$$\text{Average Reward} = \frac{1}{N}\sum_{i=1}^{N} R_i$$

where R_i is the total reward for the i-th episode.

- **Success Rate**

$$\text{Success Rate} = \left(\frac{\text{Number of Successful Episodes}}{N}\right) \times 100\%$$

where N is the total number of episodes.

- **Average Steps per Episode**

$$\text{Average Steps per Episode} = \frac{\text{Total Number of Steps}}{N}$$

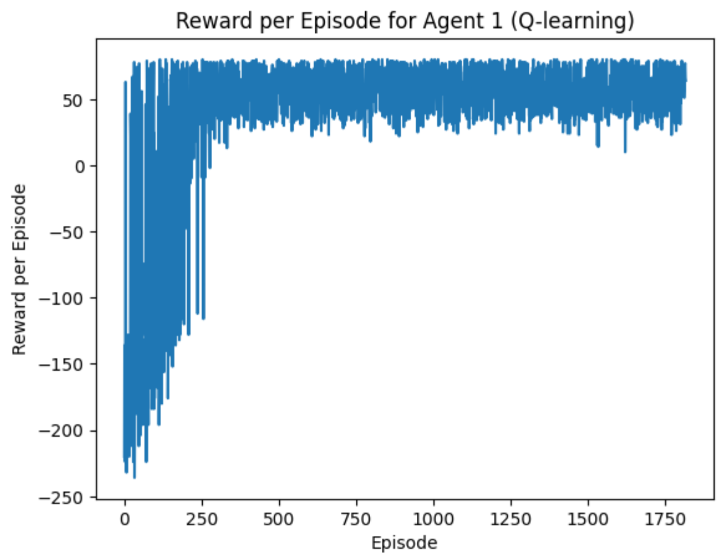

Figure 2: Example plot of Reward per Episode for Q-learning Agent.

Task 2: Evaluation of Trained Agent (3 marks)

As well as collecting statistics during training, we also wish to evaluate the performance of the agent after training has finished.

Your second task is to implement a function `evaluate_agent()` which allows a previously trained agent to act in the grid world environment, with the actions always being selected according to the Q-table of the agent.

Your function should take as parameters `q_table` and `max_total_steps`, where `q_table` is the Q-table of the trained agent, and `max_total_steps` is the maximum total number of steps across all evaluation episodes (you may include other parameters, if you wish).

Evaluation should terminate when, at the end of an episode, the total number of steps has exceeded `max_total_steps`. Your function should then return `avg_reward_per_episode`, `success_rate` and `avg_steps_per_episode`, where `avg_reward_per_episode` is the reward per episode averaged across all episodes.

Use your code to evaluate the agent trained in Task 1, and copy the returned values into your report.

Teacher Feedback Mechanism

A **teacher feedback system** is a valuable addition to the training process of Q-learning agents. In this system, a pre-trained agent (teacher) assists a new agent by offering advice during training. The advice provided by the teacher is based on two key probabilities:

- The **availability** factor determines whether advice is offered by the teacher at any step.
- The **accuracy** factor dictates whether the advice given is correct or incorrect.

Pseudo-code for Teacher Feedback

This pseudo-code illustrates how the teacher's advice can be implemented:

```
function provide_teacher_advice(teacher_q_table,state,availability,accuracy):
   if random_value_1 < availability:  // random chance based on availability
      correct action = best action for state according to teacher_q_table
      if random_value_2 < accuracy:   // random chance based on accuracy
         return correct_action        // return correct advice
      else:                           // return incorrect advice
         possible actions = all actions excluding the correct_action
         return a randomly chosen action from possible_actions
   else:
      return None                     // no advice given
```

Two separate random value variables (each a number between 0 and 1) are generated in this process. First, `random_value_1` is compared to `availability` to determine whether the teacher provides advice. If advice is given, a second `random_value_2` is used to check if the advice is correct by comparing it to `accuracy`. These two checks ensure that advice is provided probabilistically and may not always be accurate. The effect of the teacher's advice on the agent's action is as follows:

- If teacher provides **correct** advice, it will recommend the best action (according to its own Q-table), and the agent will take this action.
- If teacher provides **incorrect** advice, it will recommend a random action (excluding the correct action), and the agent will perform this action.
- If teacher provides **no advice** (or `None`), the agent must choose its own action, according to its own Q-table and exploration strategy (e.g., ε-greedy or softmax).

Task 3: Q-Learning Agent with Teacher Advice (3 marks)

In this task, you will use the agent trained in Task 1 by Q-learning as the teacher. This teacher will provide advice to a new agent, which will also be trained using Q-learning. The new agent will be trained in the same environment as in Task 1, and with the same parameters except for `max_total_steps` (which is passed as a parameter).

Your task is to write a function `train_agent_with_teacher()` which will train a new agent by Q-learning with Teacher Advice.
Your function should take as parameters:

- **teacher_q_table**: the Q-table of the teacher (normally derived from a previously trained agent)
- **max_total_steps**: the maximum total number of steps across all training episodes
- **availability** (between 0 and 1)
- **accuracy** (between 0 and 1)

You may add additional parameters to your function, if you wish.

Your function should return four values: `agent_q_table`, `avg_reward_per_episode`, `success_rate`, `steps_per_episode` (where `agent_q_table` is the Q-table of the newly trained agent, and `avg_reward_per_episode` is the reward per episode, averaged over all episodes).

Task 4: Analysis of Teacher Training (4 marks)

In this task, you will be exploring how well the agent can learn with teacher training, for various values of availability and accuracy. Since we are primarily interested in the early stages of training, the value of `max_total_steps` should be smaller for Task 4 than it was for Task 1 (specifically, it should be somewhere in the range between 6000 to 12000).

Write code to train and evaluate a series of agents, using two nested loops that account for all pairwise combinations of the following values:

- `availability`: [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
- `accuracy`: [0.2, 0.4, 0.6, 0.8, 1.0]

For each combination, you should train an agent using `train_agent_with_teacher()` from Task 3, and store the `avg_reward_per_episode` into an array `avg_reward_train` indexed by availability and accuracy. You should also evaluate the trained agent using `evaluate_agent()` from Task 2, and store the `avg_reward_per_episode` into an array `avg_reward_eval` (also indexed by availability and accuracy).

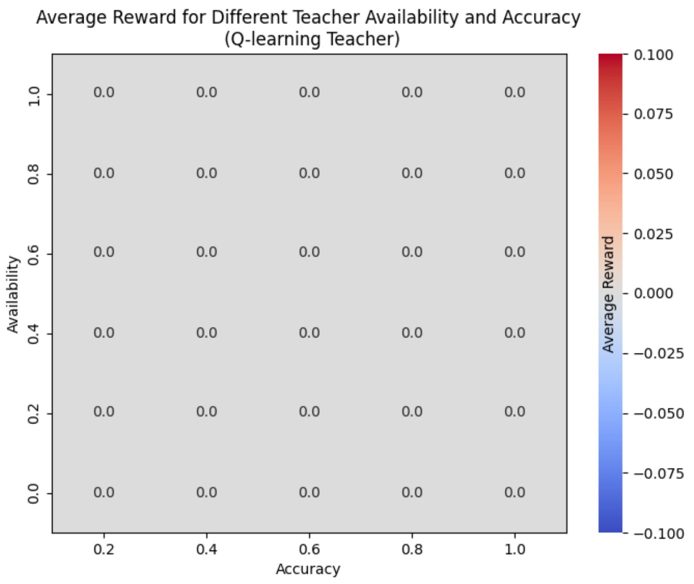

Figure 3: Illustration of heatmap showing zeros where the average reward scored for each combination of availability and accuracy should appear.

After training and evaluating in all combinations, you should produce two heat maps — one showing `avg_reward_train` and the other showing `avg_reward_eval`.

You must store these metrics for every combination in a DataFrame and use the average reward to plot a heatmap. The heatmap should represent all combinations of availability and accuracy, where: – The y-axis represents the availability values. – The x-axis represents the accuracy values. An illustration of the desired heatmap is shown in Figure 3.

Add these two heat maps to your report, together with a discussion of the values you observe in these two heat maps based on your understanding of the training setup. Your report should be named `grid.pdf`.

Submission

Parts A and B should be submitted separately.

You should submit your solutions for Part A by typing:

```
give cs3411 hw2nn student.py model.pth report.pdf
```

You should submit your solutions for Part B by typing:

```
give cs3411 hw2grid teach.py grid.pdf
```

You can submit as many times as you like – later submissions will overwrite earlier ones. You can check that your submission has been received by using one of this command:

```
3411 classrun -check
```

The submission deadline is **Tuesday 22 April, 10 pm**.

A penalty of 5% will be applied to the mark for every 24 hours late after the deadline, up to a maximum of 5 days (in accordance with UNSW policy).

Additional information may be found in the FAQ and will be considered as part of the specification for the project. Questions relating to the project can also be posted to the course Forums. If you have a question that has not already been answered on the FAQ or the Forums, you can email it to `cs3411@cse.unsw.edu.au`

Plagiarism Policy Group submissions will not be allowed. Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar assignments in previous years, if applicable) and serious penalties will be applied, including an entry on UNSW's plagiarism register.

You are also not allowed to submit code obtained with the help of ChatGPT, Claude, GitHub Copilot, Gemini or similar automatic tools.

- Do not copy code from others; do not allow anyone to see your code.
- Do not copy code from the Internet; do not develop or upload your own code on a publicly accessible repository.
- Code generated by ChatGPT, Claude, GitHub Copilot, Gemini and similar tools will be treated as plagiarism.

Please refer to the on-line resources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Academic Integrity and Plagiarism
- UNSW Plagiarism Policy

Good luck!