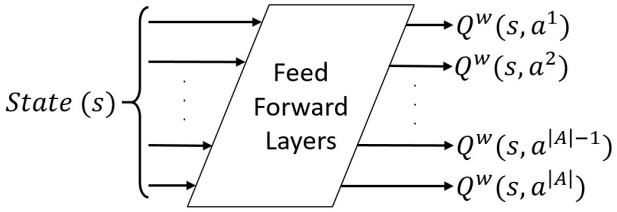
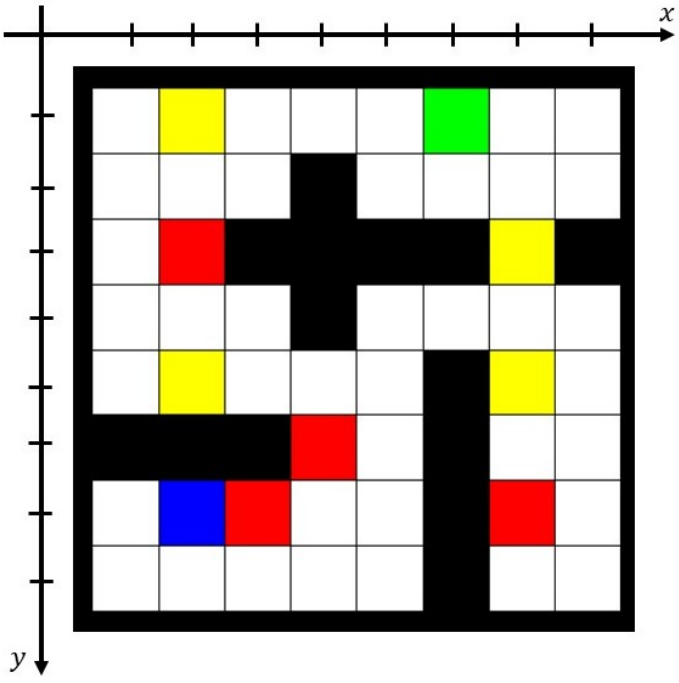
A **Deep Q-Network (DQN)** is a reinforcement learning algorithm that approximates the Q-value function using a neural network. Instead of storing a table of Q-values for every state-action pair (which becomes infeasible for large state spaces), DQN uses neural network to estimate the Q-values. This is achieved using two fully connected feed-forward neural networks with similar architectures in terms of input, output, layers, and neurons. The first network, referred to as the Q-network, is denoted by Q^w, while the second, known as the target network, is denoted by Q^{w^-}, where w and w^- represent the weights of the respective networks. The input to these networks is the state s, and the outputs correspond to the Q-values for all actions $a \in A$, i.e., $\left(Q^w(s, a^1), \ldots, Q^w(s, a^{|A|})\right)$, as depicted in the following diagram.

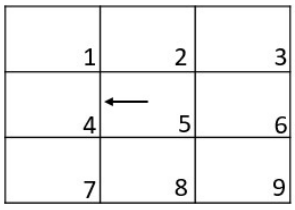


In this project, you are going to use DQN and some of its variations to learn the following maze using (x, y) coordinates as the state (i.e. $s = [x, y]^T$). The goal state is shown with green color, and the start with blue.

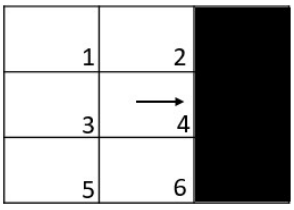


For the state representation, you are free to define the variables (x, y) in the way that suits your implementation. For example, you can use integer coordinates (1,2,...,8) or normalize these values to a range between 0 and 1. In your report, explain how you represented the state.

After taking any action, with a probability of 1-P, the agent moves to the anticipated state and, with an equal probability of P/2, will move to one of the perpendicular neighboring cells. Set the transition probability parameter to $P = 0.025$. Notice that if the next state is wall, the agent stays in the current cell. Consider the following examples:

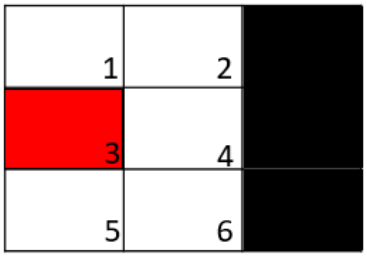


$$p(s' = 4|s = 5, a = L) = 1 - P$$
$$p(s' = 2|s = 5, a = L) = \frac{P}{2}$$
$$p(s' = 8|s = 5, a = L) = \frac{P}{2}$$
$$p(s' = 5|s = 5, a = L) = 0$$
$$p(s' = 6|s = 5, a = L) = 0$$



$$p(s' = 4|s = 4, a = R) = 1 - P$$
$$p(s' = 2|s = 4, a = R) = \frac{P}{2}$$
$$p(s' = 6|s = 4, a = R) = \frac{P}{2}$$
$$p(s' = 3|s = 4, a = R) = 0$$

The primary objective is to find the optimal policy that leads the agent to the goal state with the maximum accumulated reward. Therefore, we define the reward of taking any action -1 (delay), the reward for ending up to the yellow states -5, the reward of hitting the wall -0.8, the reward for ending up to the red states -10, and the reward for reaching to the goal state +100. For instance, this can be expressed for the following example as:



$$\begin{cases} R(s = 4, L, s' = 3) = -11 & (-10 \text{ for red and } -1 \text{ for taking an action}) \\ R(s = 4, L, s' = 2) = -1 & (\text{for taking an action}) \\ R(s = 4, L, s' = 6) = -1 & (\text{for taking an action}) \\ R(s = 4, R, s' = 4) = -1.8 & (-1 \text{ for taking an action and } -0.8 \text{ for hitting wall}) \end{cases}$$

The training process of the DQN is carried out over multiple episodes, with each episode starting from a random initial state $s_0 = [x_0, y_0]^T$ (Do not start every episode from the blue start state). At each time step t within an episode, the current state s_t is given as the input to the Q-network. The output of the Q-network is the Q-values $Q^w(s_t, a)$ for all possible actions $a \in A$, where in maze problem $A = \{Up, Right, Down, Left\}$. The action a_t is then chosen based on the Q-values using the following epsilon-greedy exploration policy:

$$a_t \sim \begin{cases} \arg\max_{a \in A} Q^w(s_t, a) & \text{w.p. } 1 - \epsilon \\ \text{Random}\{Up, Right, Down, Left\}, & \text{w.p. } \epsilon \end{cases} \tag{1}$$

Upon applying the selected action a_t according to the transition probabilities (with a probability of 1-P moving to the anticipated state and, with an equal probability of P/2, moving to one of the perpendicular neighboring cells), the new state s_{t+1}, and the reward r_t are observed. Also, you need to keep track of another variable which is commonly denoted as *done*. If s_{t+1} is the goal state, $done = 1$ and otherwise $done = 0$. These transitions are stored in a large replay memory $\mathcal{D}$, which holds the history of experiences in the form of the current sate (s), action (a), reward (r), next sate (s'), and *done* as:

$$(s, a, r, s', done) \tag{2}$$

where the tuple is stored at the end of the replay memory. If the replay memory reaches its capacity, the oldest experience is replaced to make room for new ones. The maximum length of episode is donated as T_{epi}, but the moment that the agent reaches to the goal state, the episode ends. So, each episode ends when it reaches the goal state or the maximum allowed steps T_{epi}.
The Q-network Q^w is updated every N_{QU} steps during each episode. This is done by randomly selecting a minibatch of size N_{batch} from the experiences stored in the replay memory $\mathcal{D}$. The minibatch can be represented as:

$$Z = \{s_i, a_i, r_i, s'_i, done_i\}_{i=1}^{N_{batch}} \sim \mathcal{D} \tag{3}$$

Given the selected minibatch samples, the Q-network parameters are updated by minimizing the following loss function:

$$L(Z, w, w^-) = \sum_{i=1}^{N_{batch}} \big(z_i - Q^w(s_i, a_i)\big)^2. \tag{4}$$

where

$$z_i = r_i + \gamma \max_{a \in A} Q^{w^-}(s'_i, a)(1 - done_i) \tag{5}$$

The term $(1 - done_i)$ ensures that if s'_i in the i'th experience is the goal state, the target does not include future rewards and simply becomes the immediate reward r_i. γ is the discount factor.

This loss function can be minimized using a stochastic gradient optimization technique as follows:

$$w = w - \alpha \nabla_w L(Z, w, w^-), \tag{6}$$

where α is the learning rate, and $\nabla_w L(Z, w, w^-)$ is the gradient of the loss function in (4) with respect to w. You can compute the gradient through the backpropagation process and Adam can be used as the optimization technique.
The weights of the target network Q^{w^-} should also be updated gradually toward the Q-network's weights. This slow update helps ensure the stability of the training process, and it can be achieved through the following soft update approach:

$$w^- = \eta w + (1 - \eta) w^-, \tag{7}$$

where $\eta > 0$ is the soft update hyperparameter. The learning cycle continues until a fixed number of episodes N_{epi} is reached. In your report, explain how you implemented the training process and include the code of this part.

For your ease, the structure of the network which worked in the TA's code is provided. The first layer's input dimension is 2 (x and y) and the last layer's output dimension is 4 (the Q-values of the 4 actions). Feel free to change the network structure if you think it is needed. If you do, please mention any change you have done in your report.

```
class DQN(nn.Module):
    def __init__(self):
        super(CRITIC, self).__init__()
        self.layer1 = nn.Linear(2, 128)
        self.layer2 = nn.Linear(128, 128)
        self.layer3 = nn.Linear(128, 128)
        self.layer4 = nn.Linear(128, 4)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        x = F.relu(self.layer3(x))
        return self.layer4(x)
```

It is better to use a decreasing ϵ instead of a fixed one, because at the beginning of the training the agent needs to explore the maze a lot. As a suggestion, you can set ϵ to:

$$\epsilon \text{ at episode } k = max\Big(0.1,\ (0.95)^k\Big)$$

(Obviously 0.95 is too small and you should use a number closer to 1). In your report, mention how you implemented ϵ.

During the training process, for each episode store the episode reward (i.e. the sum of rewards obtained in the episode) and the episode loss (i.e. the sum of losses obtained in the episode) in two variables *Epi_Rewards* and *Epi_Losses*. Instead of plotting the reward and loss over individual episode, it is better to compute a moving average over the last 25 episodes to smooth out fluctuations and better visualize the learning progress. For the k'th episode, the *Avg_Reward* and *Avg_Loss* are defined as:

$$Avg\_Reward[k] = \frac{1}{m}\sum_{j=0}^{m-1} Epi\_Rewards[k-j]$$

$$Avg\_Loss[k] = \frac{1}{m}\sum_{j=0}^{m-1} Epi\_Losses[k-j]$$

where:

$$m = \min(25, k)$$

This definition ensures that during the first episodes (when fewer than 25 episodes have occurred), the average is taken over the available episodes.

Additional to what have been asked so far (shown with blue font), answer the following questions in your report:
1) Report all the parameters and hyperparameter. (TA's suggestion is mentioned in the parenthesis):

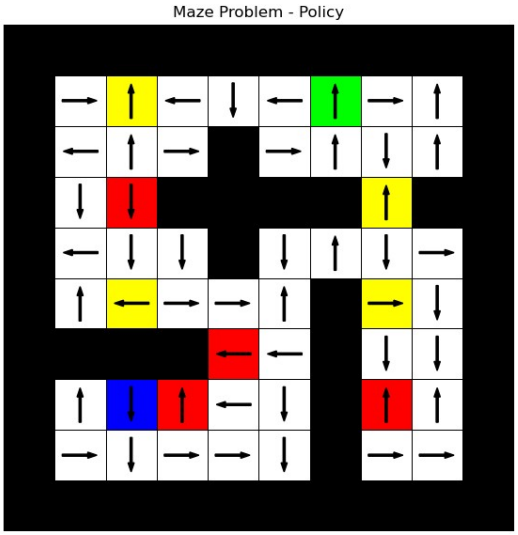
The size of the replay memory: $|\mathcal{D}|$ =? (5×10^3) The maximum length of episode: T_{epi} =? (50)

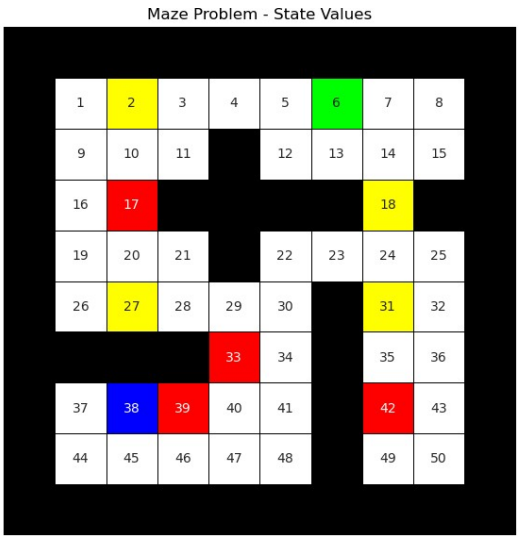
The number of steps to update the Q-network: N_{QU} =? The discount factor: γ =? ($0.95 < \gamma < 0.995$)

The size of the minibatch: N_{batch} =? (64) The learning rate: α =? ($10^{-4} < \alpha < 10^{-1}$)

The soft update hyperparameter: η =? ($10^{-4} < \eta < 10^{-1}$) The number of episodes: N_{epi} =?

2) Plot the *Avg_Reward* and *Avg_Loss* during training with respect to episode. Explain the trend you see in these plots.
3) Show the final obtained policy on the maze. Below is an example, which is clearly not the optimal policy. Explain whether the obtained policy is intuitively optimal or not.
4) Show the final state values (i.e. the maximum of the Q-values at each state) on the maze. Below is an example, which is clearly not the optimal values. Is the range of numbers what you expected? Please explain.
5) Show a path on the maze starting from the start state (blue state) and following the obtained policy. Hopefully it reaches to the goal state!





In standard DQN, the target value is computed as:

$$z_i = r_i + \gamma \max_{a \in A} Q^{w^-}(s'_i, a)(1 - done_i) \tag{8}$$

However, using the same network to both select and evaluate the action leads to overestimation bias. **Double DQN** addresses this by decoupling action selection and action evaluation as:

$$z_i = r_i + \gamma Q^{w^-}(s'_i, \arg\max_{a'} Q^{w}(s'_i, a'))(1 - done_i)$$

Here:

- The action is selected using the Q-network Q^w.
- The selected action is then evaluated using the target network Q^{w^-}.

This simple change reduces overestimation and leads to more stable and accurate learning. Implement Double DQN and mention in your report how you changed your code to do this. Also, report if you make any change in the parameters and hyperparameters.

6) Plot the results asked in question 2 to 5 (i.e 6-2: average reward and loss, 6-3: the obtained policy, 6-4: the values, and 6-5: the path) with the Double DQN implementation. Compare the obtained results with standard DQN.

In **Dueling DQN**, instead of having a single network that directly outputs Q-values for each action, the network is structured to first extract features from the input state. These features are then passed through two separate streams:
1. Value Stream (V(s)): This part of the network learns to predict how good it is to be in a given state, independent of the action taken.
2. Advantage Stream (A(s, a)): This part learns to predict the advantage of each action relative to the average action in that state.
Finally, these two streams are combined to compute the Q-values using the formula:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|} \sum_{a'} A(s,a') \right)$$

The training process, loss calculation, and optimization steps remain the same. You are expected to design this architecture themselves by:
- Creating a feature extraction layer that processes the input state.
- Designing two separate branches for value and advantage computation.
- Combining the outputs following the equation above to obtain Q-values.
No modifications are required in the loss function or training loop; they will automatically use the outputs of this new network structure. Implement Dueling DQN and mention in your report how you changed your code to do this. Also, report if you make any change in the parameters and hyperparameters.

7) Plot the results asked in question 2 to 5 (i.e 7-2: average reward and loss, 7-3: the obtained policy, 7-4: the values, and 7-5: the path) with the Dueling DQN implementation. Compare the obtained results with standard and Double DQN.