# Instructions for Lab Session7: Deep Q Network

For this lab session, you will be following and filling out a Jupyter notebook.

You will have to submit back this notebook, with your plots visible, and an analysis in a cell below containing

- A short summary of the experiment (or what is different from before, explanation of the new method)
- An analysis of the plot – performance, speed, and why you think it is so
- If you changed a hyper-parameter (or did a study), an explanation of it.
- A final single plot comparing the performance of all agents playing cartpole 30 times. (use test function to get a agent's performance)

Section wise detailed instructions follow on next page.

**The deadline of assignment is 9th of april**.

**For those who are using jupyter for first time**

You can run the notebook document step-by-step (one cell a time) by pressing **shift + enter**.

You can run the whole notebook in a single step by clicking on the menu **Run -> Run All Cells**.

To restart the kernel (i.e. the computational engine), click on the menu **Kernel -> Restart Kernel**. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc…).

## Environment

The CartPole environment is quite simple: you have to balance a pole on a moving cart.

General Description: https://gym.openai.com/envs/CartPole-v0/

Use https://github.com/openai/gym/wiki/CartPole-v0 to understand the state space, action space, dynamics and reward function

## Useful Reminders and details

### *Loss function's done signal*

QLearning used so far used target R + gamma*max_a' Q(s',a').

When s' is a terminal state, we know by construction that q*(s', . )=0. Until now, we initialized our Q estimates at 0, so this was verified. However, our network will be initialized randomly, so it will most likely not give a 0 value to terminal states. Note that we do never update a terminal state (learn from a s=terminal, s'=?? Transition), therefore we cannot learn this. In order to incorporate this information, the learn method takes in the **done** signal, which is a boolean indicating whether next_state is a terminal state.

Do not forget to use this signal in the computation of the expected_q_value, or the agent will rely on wrong estimates of the terminal state values!

## Instructions

Go through the boxes one by one to understand what is going on in each of them.

You will have to implement the missing parts indicated by **TODO** flags in the code. A bullet in the following list indicated a cell you have to fill in:

- In **Computing Temporal Difference Loss** (in compute_td_loss_dqn function). There, you have to
    - Predict the q_values for this state
    - Predict the q_values for the next state
    - Extract the q_value of the action performed
    - Compute the QLearning target “expected q value”
    - Compute the MSE loss
- The associated plots, and in a cell below, your analysis
- **DQN with Replay Buffer / Compute TD Loss** (in compute_td_loss_batch): same work as before, but now the inputs come in a batch, since we’re sampling experiences from the buffer.
    - Predict the q_values for this state
    - Predict the q_values for the next state
    - Extract the q_value of the action performed
    - Compute the QLearning target “expected q value”
    - Compute the MSE loss
- The associated plots, and in a cell below, your analysis
- **DQN with Target Network / Compute TD Loss** (in compute_td_loss_target): same work as before, but now using a target network. Be careful, you now need to write the next_q_value yourself!
    - Predict the q_values for this state
    - Predict the q_values for the next state
    - Extract the q_value of the action performed
    - !NEW! Next Q value
    - Compute the QLearning target “expected q value”
    - Compute the MSE loss
- The associated plots, and in a cell below, your analysis
- **DQN with Target Network and Replay Buffer / Compute TD Loss** (in compute_td_loss_target_batch): same work as DQN alone, but now using both improvements.
    - Predict the q_values for this state
    - Predict the q_values for the next state
    - Extract the q_value of the action performed
    - Next Q value
    - Compute the QLearning target “expected q value”
    - Compute the MSE loss

- The associated plots, and in a cell below, your analysis

From now on, always include both improvements in the following

- **Double DQN / Compute TD Loss** (in compute_td_loss_doubleDQN): using the target net to estimate the value of the best action according to the model.
    - Predict the q_values for this state
    - Predict the q_values for the next state using model
    - Predict the q_values for the next state using target
    - Next Q value
    - Compute the QLearning target “expected q value”
    - Compute the MSE loss
- The associated plots, and in a cell below, your analysis
- **Dueling DQN / Compute TD Loss** (in compute_td_loss_duelingDQN): decoupling q value into state value and action advantage.
    - Predict the q_values for this state
    - Predict the q_values for the next state
    - Extract the q_value of the action performed
    - Next Q value
    - Compute the QLearning target “expected q value”
    - Compute the MSE loss
- The associated plots, and in a cell below, your analysis
- **Plot comparison of all agents**: Use the **test** function defined at the starting of notebook to collect each agent’s performance individually. Plot their comparison here.
- **Playing Atari with Double DQN:** last panel: Based on Td_loss functions you have worked with write or choose a function to reuse here and justify your choice?
- The associated plots, and in a cell below, your analysis