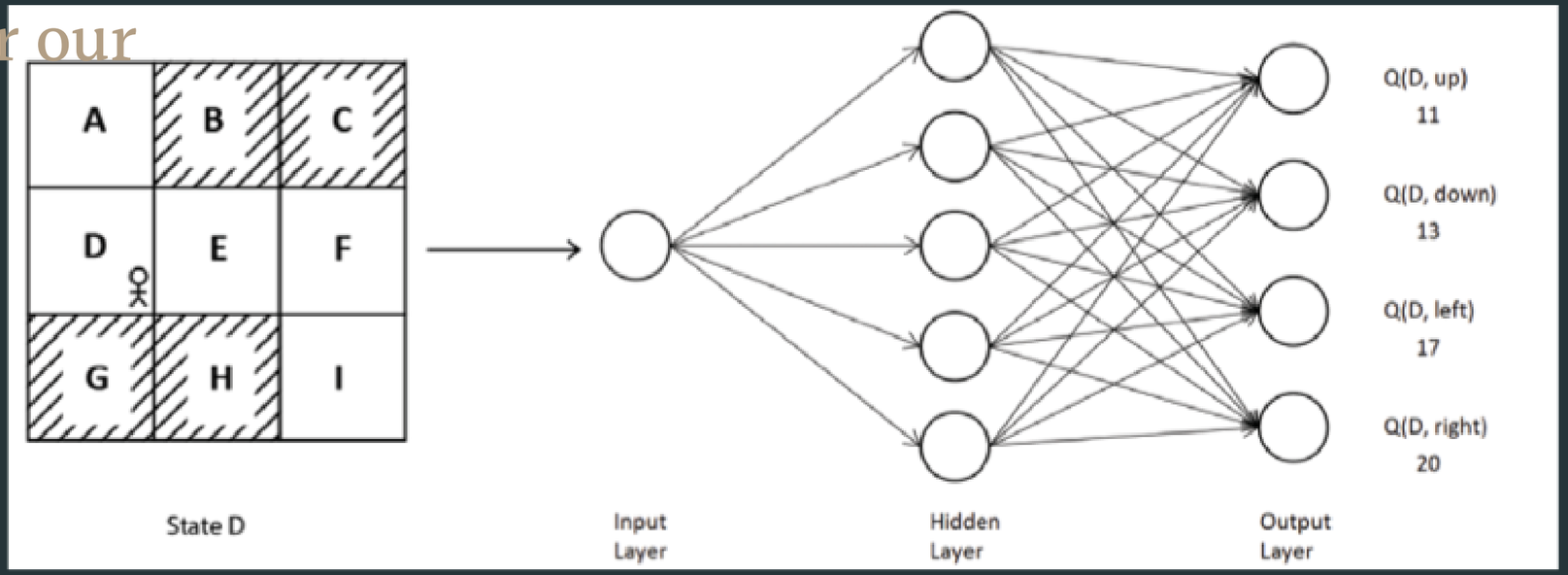
Deep Q Network

Deep Reinforcement Learning (DRL)

What is DQN?

The objective of reinforcement learning is to find the optimal policy, that is, the policy that gives us the maximum return (the sum of rewards of the episode). In order to compute the policy, first we compute the Q function. Once we have the Q function, then we extract the policy by selecting an action in each state that has the maximum Q value.

For example, let's consider our grid world environment



In scenarios with numerous states and actions, calculating Q values for all potential state-action pairs exhaustively can be very costly.

Instead of computing Q values in this way, can we approximate them using any function approximator, such as a neural network

In this process, we input the environmental state into a neural network, which then outputs the Q values for all available actions in that state. After acquiring the Q values, we can choose the optimal action based on the one with the highest Q value.

02

- We name the neural network used for Q value approximation as the Q network.
- When a deep neural network is used for Q value approximation, it is referred to as a deep Q network (DQN).
- The Q function is denoted as Q(s, a) with parameter theta (θ) in subscript indicates that our Q function is parameterized by θ, and θ is just the parameter of our neural network.
- We initialize the network parameter θ with random values and approximate the Q function (Q values), but since we initialized θ with random values, the approximated Q function will not be optimal.
- So, we train the network for several iterations by finding the optimal parameter θ. Once we find the optimal $\theta\theta$, we will have the optimal Q function. Then we can extract the optimal policy from the optimal Q function.

Understanding DQN

Proliferation of Chat Bots

In the future there will be a way to automate responses based on pre-written keywords and through machine learning.

Presentations are communication tools that can be used as demonstrations, lectures, speeches, reports, and more. It is mostly presented before an audience.

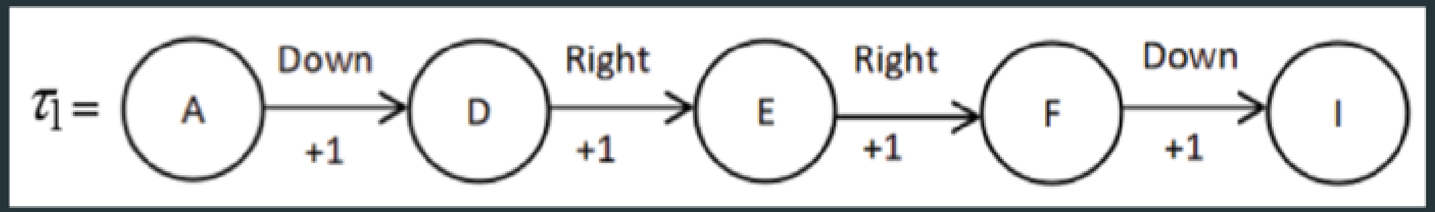
Understanding DQN

- We discovered that DQN is utilized to estimate the Q value for all actions in a specific input state. Since the Q value is a single continuous number, our use of DQN essentially involves carrying out a regression task.
- We use a buffer called a replay buffer to collect the agent's experience and, based on this experience, we train our network.

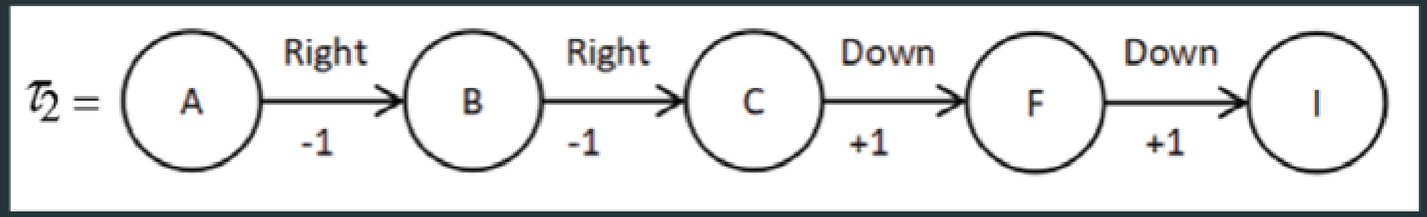
Replay Buffer

- The agent transitions from one state, s, to another state, s', by taking a specific action, a, and then receives a reward, r. This transition data (s, a, r, s') is stored in a buffer known as a **replay buffer** or experience replay, typically represented by D.
- This transition data essentially represents the agent's learning.
- By accumulating the agent's experiences from various episodes in the replay buffer, we can train our DQN using sampled experiences (transitions) from this buffer.
- The replay buffer serves as a crucial tool in storing and utilizing the agent's experiences effectively.

Episode 1:

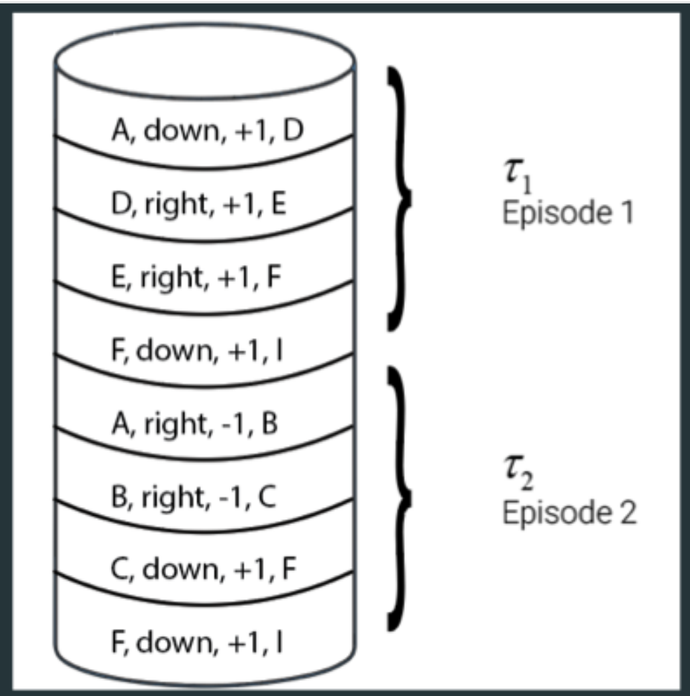


Episode 2:



How we store the transition information incthe replay buffer $\mathscr{D}$:

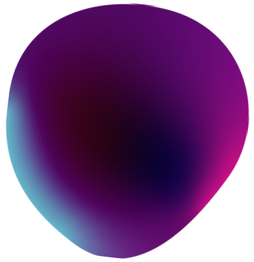
1. Initialize the replay buffer $\mathscr{D}$.
2. For each episode perform step 3.
3. For each step in the episode:
   1. Make a transition, that is, perform an action a in the state s, move to the next state s', and receive the reward r.
   2. Store the transition information (s, a, r, s') in the replay buffer $\mathscr{D}$.



Replay buffer

03

Loss function



$$\text{MSE} = \frac{1}{K}\sum_{i=1}^{K}(y_i - \hat{y}_i)^2$$

We learned that in DQN, our goal is to predict the Q value, which is just a continuous value. Thus, in DQN we basically perform a regression task. We generally use the mean squared error (MSE) as the loss function for the regression task. MSE can be defined as the average squared difference between the target value and the predicted value, as shown here:

Where y is the target value, ŷ is the predicted value, and K is the number of training samples.
We can train our network by minimizing the MSE between the target Q value and predicted Q value.

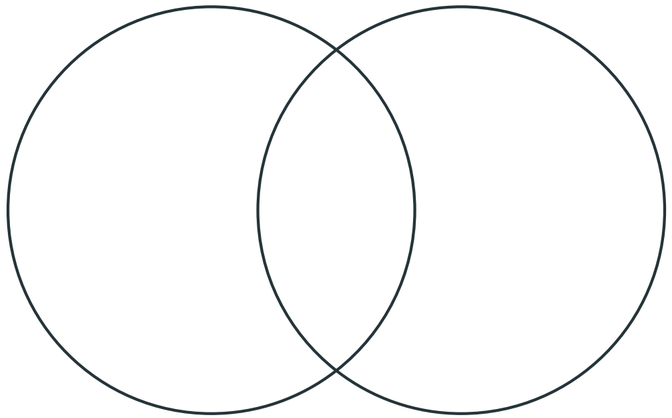
Step-by-step explanation of how the Deep Q-Network (DQN) algorithm works:

1. Initialize the Q-Network: Start by initializing a neural network to approximate the Q-function. This network takes the state as input and outputs the Q-values for each action.
2. Initialize the Replay Buffer: Create a replay buffer to store the agent's experiences. Each experience consists of a tuple (state, action, reward, next state, done), representing the state the agent was in, the action it took, the reward it received, the next state it transitioned to, and whether the episode ended.
3. Select an Action: For each step in the environment, select an action using an epsilon-greedy policy. With probability epsilon, choose a random action to explore the environment; otherwise, choose the action that maximizes the Q-value predicted by the network for the current state.
4. Execute the Action: Take the selected action and observe the next state and the reward received from the environment.
5. Store the Experience: Store the experience tuple (state, action, reward, next state, done) in the replay buffer.
6. Sample from the Replay Buffer: Periodically, sample a batch of experiences from the replay buffer to train the Q-network. This helps to break the correlation between consecutive experiences and stabilize the learning process.
7. **Compute the Target Q-Values**: For each sampled experience, compute the target Q-value using the formula: $y_i = r_i + \gamma \max_{a'} Q_{\theta'}(s_i', a')$
8. **Compute the loss:** Minimize the mean squared error between the predicted Q-values and the target Q-values: $L(\theta) = \frac{1}{K}\sum_{i=1}^{K}(y_i - Q_\theta(s_i, a_i))^2$
9. **Repeat:** Continue interacting with the environment, selecting actions, storing experiences, and updating the Q-network until the desired performance is achieved.

Understanding Double DQN

1. **Q-Learning:** Q-learning is a popular RL algorithm used for making decisions in an environment. It learns a policy by estimating the value of being in a certain state and taking a certain action, denoted as Q(state, action). The goal is to maximize the total reward over time.
2. **Deep Q-Network (DQN):** DQN is an extension of Q-learning that uses a deep neural network to approximate the Q-values. This allows DQN to handle high-dimensional state spaces like images, making it suitable for tasks like playing Atari games.
3. **Challenges with Traditional DQN:** Traditional DQN can overestimate Q-values, leading to suboptimal policies. This is because the same network is used to select and evaluate actions, which can lead to a bias in the Q-value estimates.
4. **Double DQN:** Double DQN addresses the overestimation issue by decoupling the action selection and evaluation. Instead of using a single network to estimate Q-values, Double DQN uses two separate networks: one for action selection (the policy network) and one for action evaluation (the target network).

How Double DQN Works:



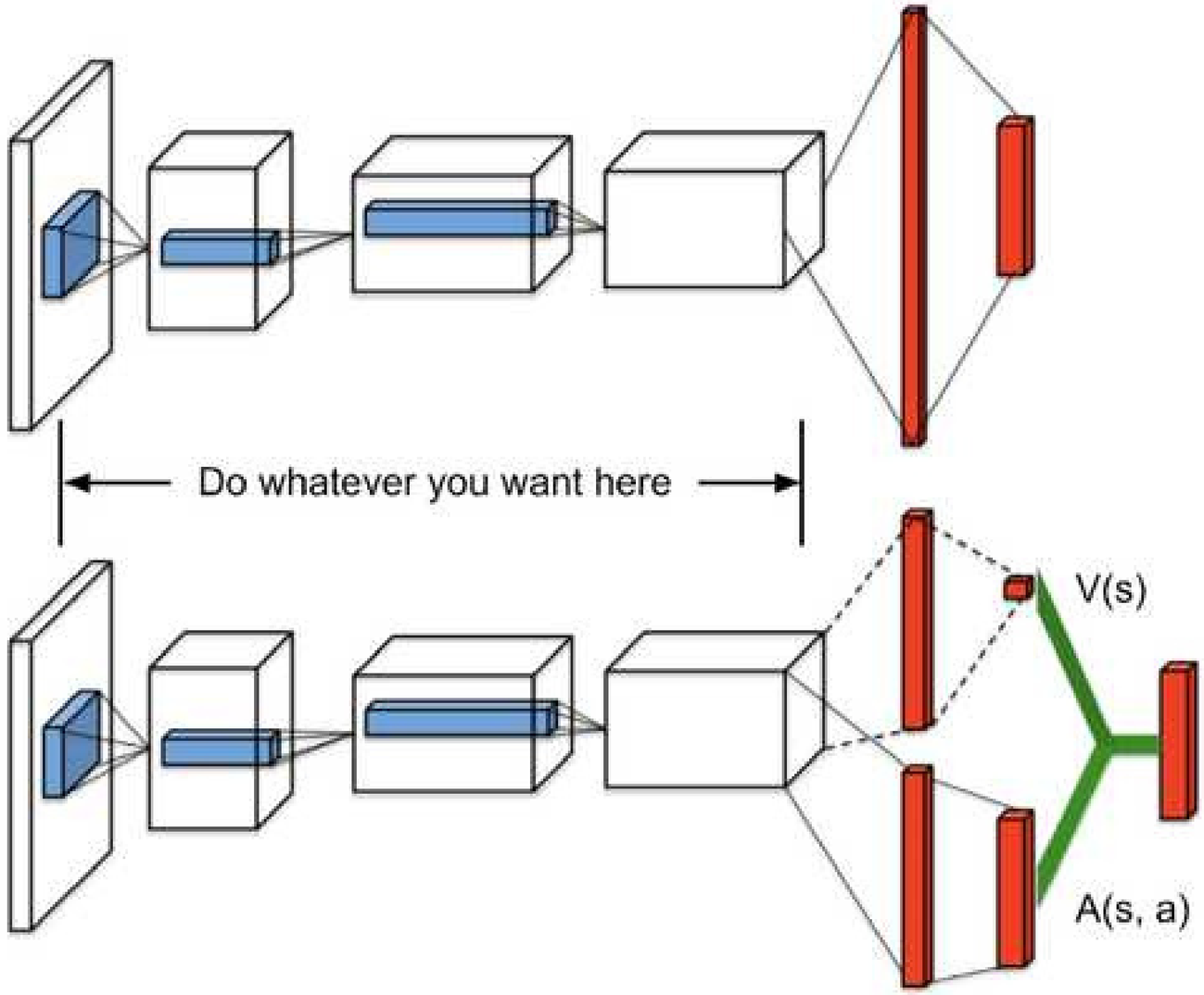
Double DQN

1. The policy network is used to select actions based on the current state.
2. The target network is used to evaluate the Q-values of the selected actions.
3. Periodically, the weights of the target network are updated to match the weights of the policy network, stabilizing the learning process.

Benefits of Double DQN:

Reduces overestimation of Q-values, leading to more stable and accurate policy updates.

Improves the learning process, especially in environments where Q-values can be easily overestimated.

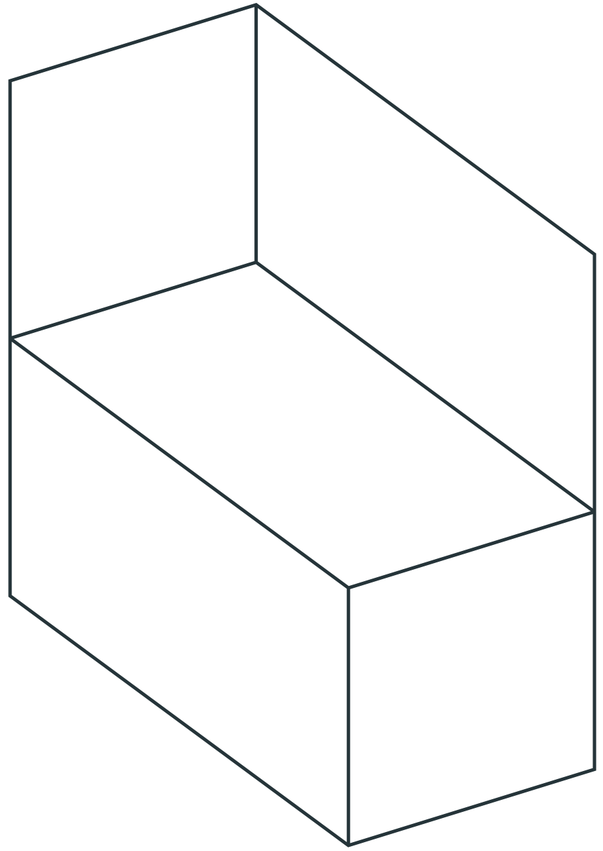


Vanilla Q network

Dueling Q network

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

DQN with prioritized experience replay



Types of prioritization
We can prioritize our transition using the following two methods:

- Proportional prioritization
- Rank-based prioritization

We learned that in DQN, we randomly sample a minibatch of K transitions from the replay buffer and train the network. Instead of doing this, can we assign some priority to each transition in the replay buffer and sample the transitions that had high priority for learning?

The dueling DQN

Before we move forward, let's explore a fundamental concept in reinforcement learning called the advantage function. This function represents the difference between the Q function and the value function, defined as:

$$A(s, a) = Q(s, a) - V(s).$$

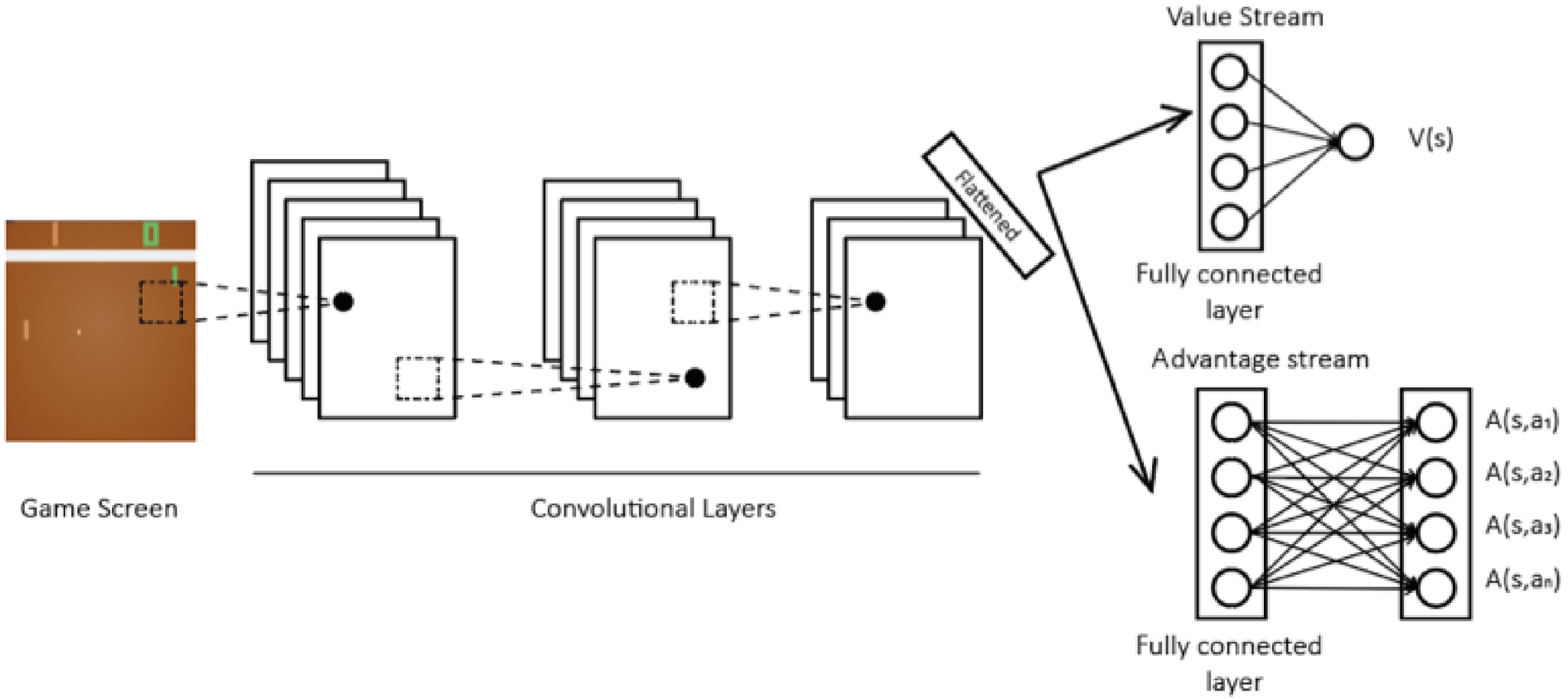
But why is the advantage function essential? What does it signify? To gain a better understanding, let's review the definitions of the Q function and the value function:

- Q function: Predicts the expected return an agent would get by starting from state s, taking action a, and following policy π.
- Value function: Predicts the expected return an agent would receive by starting from state s and following policy π.

In simpler terms, the Q function evaluates the worth of a state-action pair, while the value function assesses the value of a state irrespective of the action taken. The difference between these functions shows how advantageous action a is compared to the average actions in state s.

Therefore, the advantage function demonstrates the effectiveness of action a compared to average actions in state s. After understanding the core concept of the advantage function, let's delve into its importance and application in the realm of DQN.

Architecture of the dueling DQN

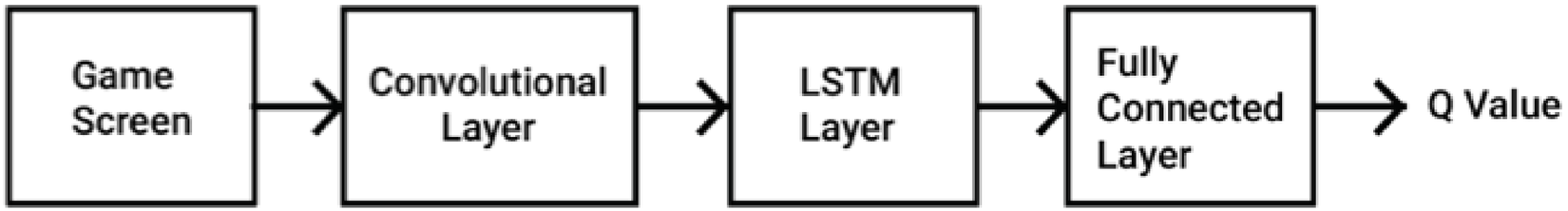


The final layer of the network splits into two streams: one calculates the value function and the other computes the advantage function. The value stream provides the state's value, and the advantage stream determines the advantage of all potential actions in that state.

The deep recurrent Q network

- The DRQN is a DQN with recurrent layers, useful for Partially Observable Markov Decision Processes (POMDPs) where only limited information is available about the environment. Retaining information about past states helps the agent understand the environment better and determine the optimal action in POMDP scenarios.
- Long Short-Term Memory Recurrent Neural Networks (LSTM RNN) can retain information as needed, useful for addressing the issue of Partially Observable Markov Decision Process (POMDP).
- LSTM layer in Deep Q-Networks (DQN) helps retain past states information.

The architecture of a DRQN



We input the game screen into the convolutional layer. This layer convolves the image to create a feature map, which is then forwarded to the LSTM layer. The LSTM layer possesses memory to retain crucial details, preserving significant past game states and adjusting its memory as needed during time steps. Subsequently, the hidden state from the LSTM layer is fed into the fully connected layer, which generates the Q value.

Summary

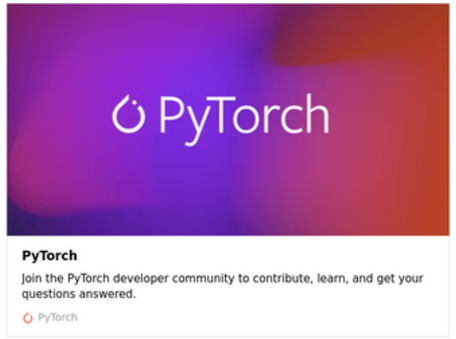
A key component of a DQN is the replay buffer, which stores the agent's experiences. These experiences are randomly sampled to train the network by minimizing the Mean Squared Error (MSE).

We addressed the issue of overestimation in DQN due to the max operator, leading to the introduction of double DQN with two Q functions for more accurate target value computation. One Q function, parameterized by the main network parameter θ, is used for action selection, while the other, parameterized by the target network parameter θ, computes the Q value.

We covered DQN with prioritized experience replay, emphasizing prioritization based on TD error through proportional and rank-based methods. Another variant discussed was dueling DQN, where Q values are computed using two streams: the value stream and the advantage stream.

The Lecture concluded by introducing DRQN and its application in solving challenges posed by partially observable Markov decision processes. The upcoming chapter will focus on exploring another widely-used algorithm known as policy gradient.

EL Framework



Tensorforce

IntelLabs/coach

Stable Baselines

Playing Atari games using DQN

Lab 8: **Playing Atari games using DQN**

Thank you