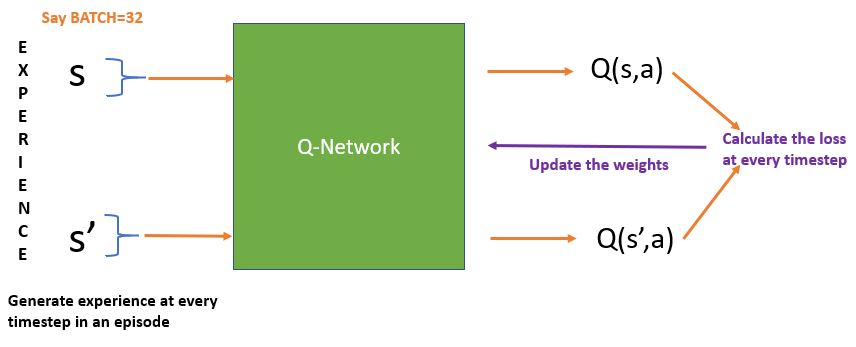


**Comprehension - DQN Code Snippet**

Let's now see how we can write the code for training a deep Q-network.



**Model Network training**

* First, make a neural network model architecture:
* **def** **build\_model**():
* model = Sequential()
* model.add(Dense(**32**, input\_dim=state\_size, activation='relu', kernel\_initializer='he\_uniform'))
* model.add(Dense(**32**, activation='relu', kernel\_initializer='he\_uniform'))
* model.add(Dense(action\_size, activation='linear', kernel\_initializer='he\_uniform'))
* model.compile(loss='mse', optimizer=Adam(lr=learning\_rate))
* **return** model
* Now initialise the memory
* memory = deque(maxlen=**1000**)
* # Defining memory with capacity = 1000
* Let's first generate an experience in an episode.
* # 'done' is for checking if the episode has ended
* **while** **not** done:
* # get action for the current state and go one step in environment
* action = agent.get\_action(state)
* next\_state, reward, done, info = env.step(action)
* In the above code, you take action using the epsilon-greedy policy.
* **def** **get\_action**(state):
* **if** np.random.rand() <= epsilon:
* **return** random.randrange(action\_size)
* **else**:
* q\_value = model.predict(state)
* **return** np.argmax(q\_value[**0**])
* Store the experience in the memory
* agent.append\_sample(state, action, reward, next\_state, done)
* **def** **append\_sample**(state, action, reward, next\_state, done):
* # Adding sample to the memory.
* memory.append((state, action, reward, next\_state, done))
* Now, let's train the model at every time step. The training step comes after saving the sample to memory.
* **while** **not** done:
* # get action for the current state and go one step in environment
* action = agent.get\_action(state)
* next\_state, reward, done, info = env.step(action)
* # save the sample <s, a, r, s',done> to the replay memory
* agent.append\_sample(state, action, reward, next\_state, done)
* # every time step do the training
* agent.train\_model()
* state = next\_state

* Let's closely look at the train\_model() function. You sample batch size from memory. Calculate the Q(s,a) and Q(s',a) and train the model. There is a little bit of Python code to make it run for a batch input.
* **def** **train\_model**():
* # Sample batch from the memory
* batch\_size = min(batch\_size, len(memory))
* mini\_batch = random.sample(memory, batch\_size)
* # Initialise the Q(s,a) with zero
* update\_input = np.zeros((batch\_size, state\_size))
* # Initialise the Q(s',a)
* update\_target = np.zeros((batch\_size, self.state\_size))
* action, reward, done = [], [], []
* **for** i **in** range(batch\_size):
* # Add state s to the Q(s,a), Q(s',a) from memory
* update\_input[i] = mini\_batch[i][**0**]
* # Add action from memory
* action.append(mini\_batch[i][**1**])
* # Add reward from the memory
* reward.append(mini\_batch[i][**2**])
* # Add next state s' to Q(s',a) from the memory
* update\_target[i] = mini\_batch[i][**3**]
* done.append(mini\_batch[i][**4**])
* # Find the Q(s,a) and Q(s',a) using state as input to the neural network
* target = model.predict(update\_input)
* target\_val = model.predict(update\_target)

* # set the target as (r + maxQ(s',a))
* **for** i **in** range(self.batch\_size):
* # Q Learning: get maximum Q value at s' from target model
* **if** done[i]:
* target[i][action[i]] = reward[i]
* **else**:
* target[i][action[i]] = reward[i] + discount\_factor \* (
* np.amax(target\_val[i]))
* # train the model
* model.fit(update\_input, target, batch\_size=batch\_size,
* epochs=**1**, verbose=**0**)

