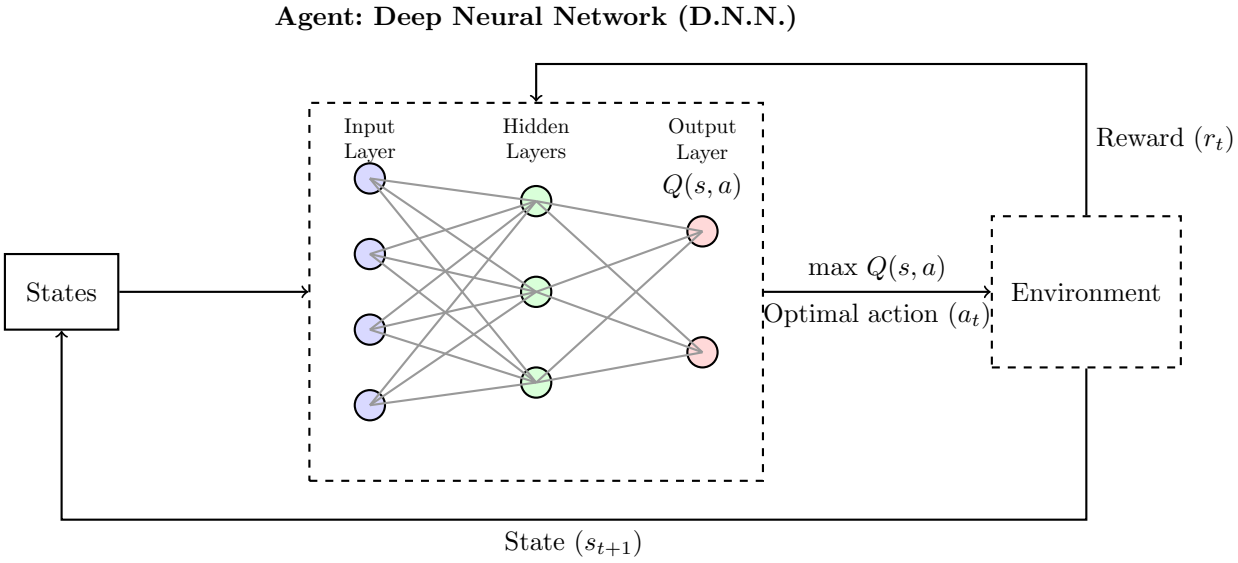
DQN – Basic Structure Idea



Q-learning function:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

Problem: I need $Q(s',a')$, $\forall(s',a') \rightarrow$ inefficiency.
Solution: Deep Q-Network.

Updating Q-learning function (Bellman Equation):

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big]$$

where:

- α: learning rate
- r: immediate reward
- γ: discount factor (how much we value the future)
- $Q(s,a)$: current Q-value
- $\max_{a'} Q(s',a')$: target Q-value (what Q should be)

Deep Q-Networks: Loss Function and Target Network

Καταλαβαίνουμε ότι το νευρωνικό μας πρέπει να ελαχιστοποιήσει την:

$$\text{Loss function: } \Big(Q(s,a) - \big[r + \gamma \max_{a'} Q(s',a')\big]\Big)$$

Target network

Τα $Q(s,a)$ και $Q(s',a')$ δεν πρέπει να υπολογίζονται με τον ίδιο τρόπο — it doesn't make sense! Φτιάχνουμε δύο όμοια νευρωνικά:

$$\text{policy-net} \leftarrow (\text{ανανεώνεται συνεχώς κατά το training})$$

$$\text{target-net} \leftarrow Q(s',a')$$

Τα βάρη του policy-net ανανεώνονται συνεχώς, ενώ του target-net περιοδικά.

Notation: policy weights θ, target weights θ'.

DQN Bellman Equation

$$Q(s,a;\theta) \leftarrow Q(s,a;\theta) + \alpha\Big[r + \gamma \max_{a'} Q(s',a';\theta') - Q(s,a;\theta)\Big]$$

$$\text{Loss function: } \Big[r + \gamma \max_{a'} Q(s',a';\theta') - Q(s,a;\theta)\Big]^2$$

Μέχρι τώρα:

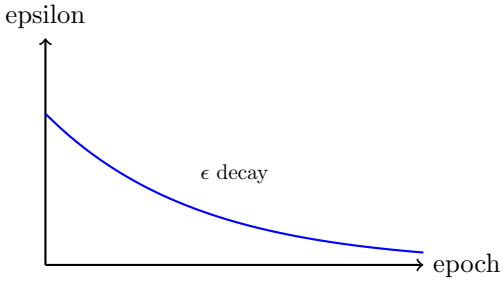
- Αρχιτεκτονική ✓
- Loss function για υπολογισμό των βαρών ✓

Το target-net θα ενημερώνεται ομαλά (*soft update*) ως εξής:

$$\Theta' \leftarrow \tau\Theta + (1-\tau)\Theta', \quad \text{όπου } \tau \ll 1$$

Πως κάνουμε train:

Selecting action: Choose an ϵ, and draw a random number $x \in [0,1]$.
If $x < \epsilon$, then perform a **random action** → exploration.
Otherwise, use the **policy-net** → exploitation.
This is called epsilon-greedy policy.



Calculating the Reward

This depends on the environment. However, there are some general guiding principles:

- Frequent and small rewards are usually better than sparse and large ones.
- Penalties should be introduced to help control and stabilize learning.
- Our general intuition should be to design a reward/penalty system that guides the learning process toward convergence at a desired target behavior or state.

In summary: the reward function acts as the teacher for the agent — it shapes the policy by providing consistent feedback.

Experience Replay

Αυτό αφορά τον τρόπο με τον οποίο γίνεται το *training*. Αποθηκεύουμε στη μνήμη ένα διάνυσμα της μορφής:

$$(state,\ action,\ next\_state,\ reward)$$

για κάθε δράση (*action*) που εκτελούμε.
This allows us to:

- Store past experiences in a replay buffer (memory).
- Sample random batches of these experiences during training.
- Compute the loss function on these batches, and
- Update the network weights more efficiently and stably.

Με αυτόν τον τρόπο, το νευρωνικό μας δίκτυο μαθαίνει από ένα πλήθος παλαιότερων εμπειριών και όχι μόνο από την πιο πρόσφατη, βελτιώνοντας έτσι τη γενίκευση και τη σταθερότητα του αλγορίθμου.