Ονοματεπώνυμο: Χαράλαμπος Ραφαήλ Δημητρίου

1)

Επιλέξαμε το περιβάλλον MountainCar-v0

Action space: Το περιβάλλον έχει έναν διακριτό action space (discrete action space) με 3 δυνατές ενέργειες:

- 0: Επιτάχυνση προς τα πίσω (αριστερά).
- 1: Μηδενική επιτάχυνση.
- 2: Επιτάχυνση προς τα εμπρός (δεξιά).

Observation Space: αποτελείται από μία δυάδα (θέση, ταχύτητα)

- Η θέση του αυτοκινήτου στην πίστα με τιμές στο διάστημα [-1.2, 0.6] και μονάδα μέτρησης m.
- Η ταχύτητα του αυτοκινήτουμε με τιμές στο διάστημα [-0.07, 0.07] και μονάδα μέτρησης m/sec.

Reward signals:

Ο πράκτορας λαμβάνει -1 reward για κάθε βήμα που δεν φτάνει τον στόχο.

΄Οταν το αυτοκίνητο φτάσει στην κορυφή (position ≥0.5), το επεισόδιο τερματίζει.

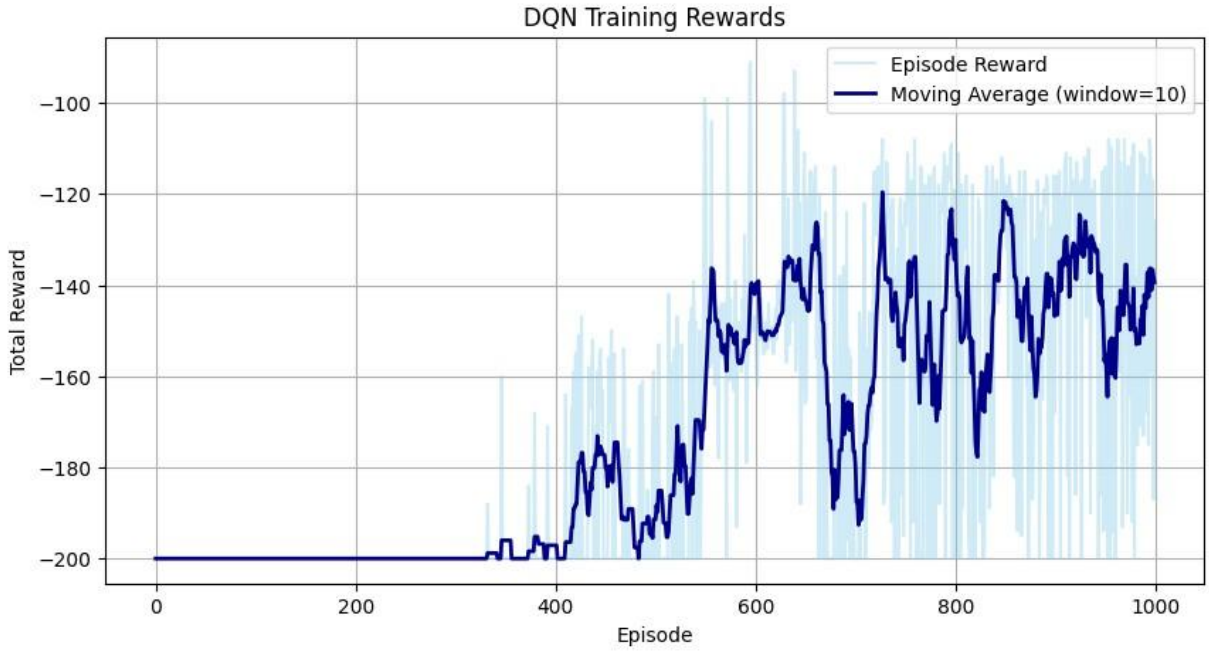
2.1)

Για την επίλυση του προβλήματος με τη χρήση του αλγορίθμου dqn έχουμε τον κώδικα που βρίσκεται στο αρχείο dqn.py

Παρακάτω παρουσιάζεται το διάγραμμα συνολικής αμοιβής και μεσης συνολικής αμοιβής (των 10 τελευταιων επισοδιων) σε συνάρτηση με τα επεισόδια. Με τις παραμέτρους του προβλήματος να παίρνουν τις παρακάτω τιμές.

batch_size = 64 gamma
= 0.99
epsilon_start = 1.0 epsilon_end =
0.01 epsilon_decay = 0.995
target_update = 10
memory_capacity = 10000
learning_rate = 0.001

num_episodes = 1000
MAX_STEPS = 200 (default)



2.2) Ανάλυση ευαισθησίας DQN ως προς τις υπερπαραμέτρους

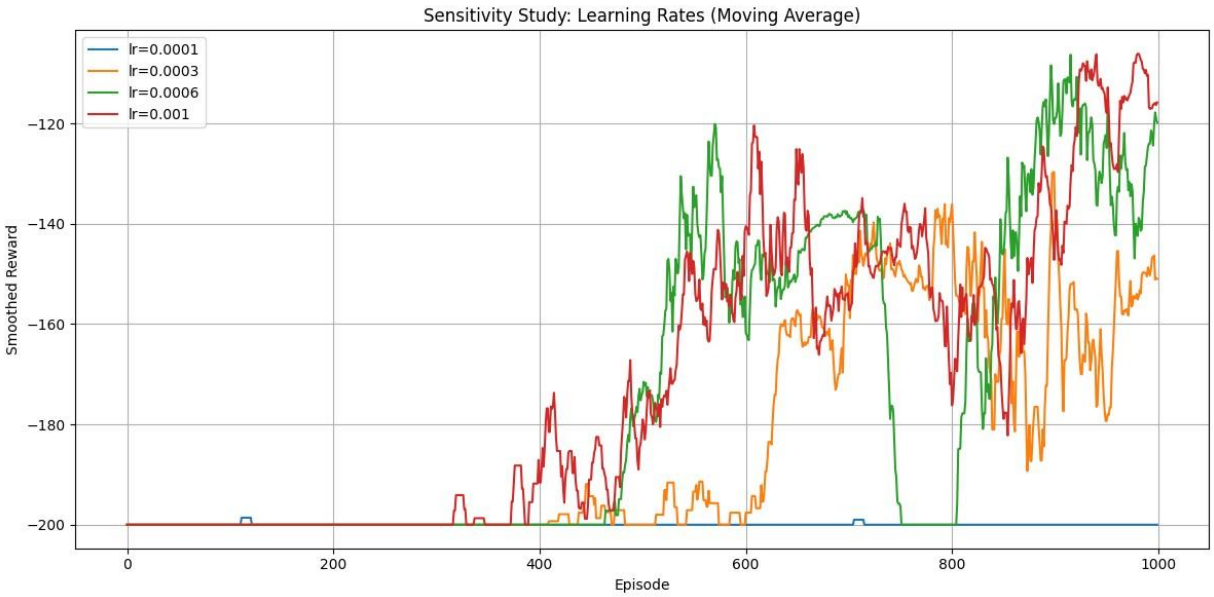
Για να μελετήσουμε την συμπεριφορά του αλγορίθμου σε συνάρτηση με 4 παραμέτρους έχουμε υλοποιήσει τον κώδικα που βρίσκεται στο αρχείο dqn_4parameters.py Επιλεγεγουμε τις παρακατω τιμες:

1. learning_rates = {0.0001, 0.0003, 0.0006, 0.001}
2. gammas = {0.9, 0.95, 0.99, 0.999}
3. batch_sizes = {32, 64, 128, 256}
4. epsilon_decays = {0.98, 0.99, 0.995, 0.999}

Σε κάθε περίπτωση οι υπόλοιπες μεταβλητες σταθεροποιούνται με:
base_lr = 0.001 base_gamma =
0.99 base_batch_size = 64
base_epsilon_decay = 0.995

1. learning_rates.

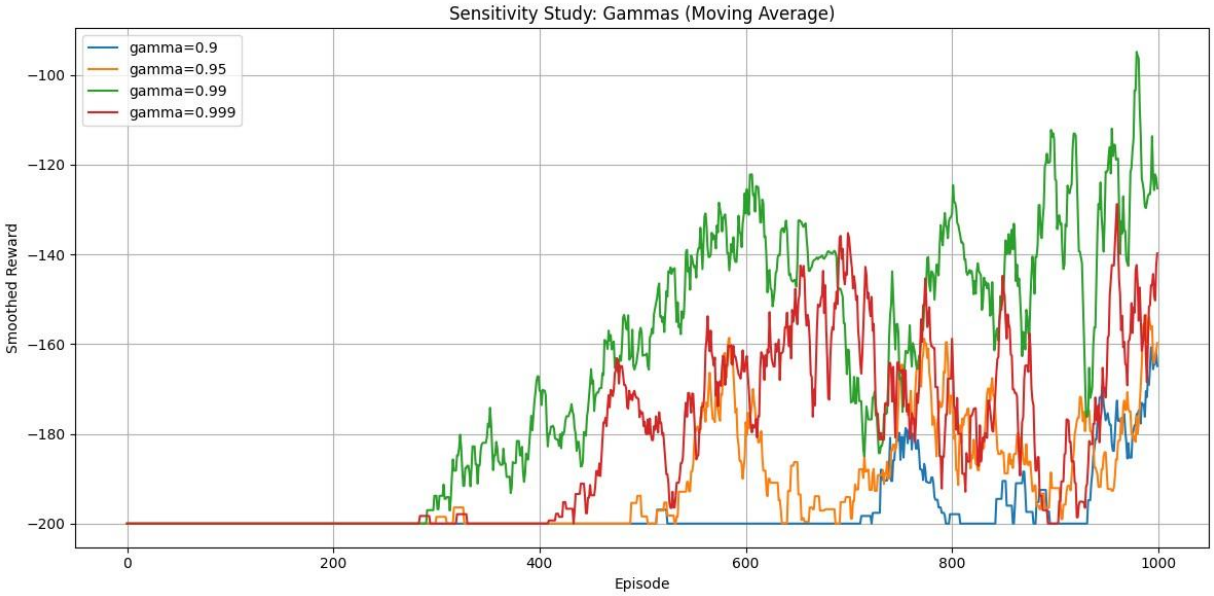
Μελέτη ως προς

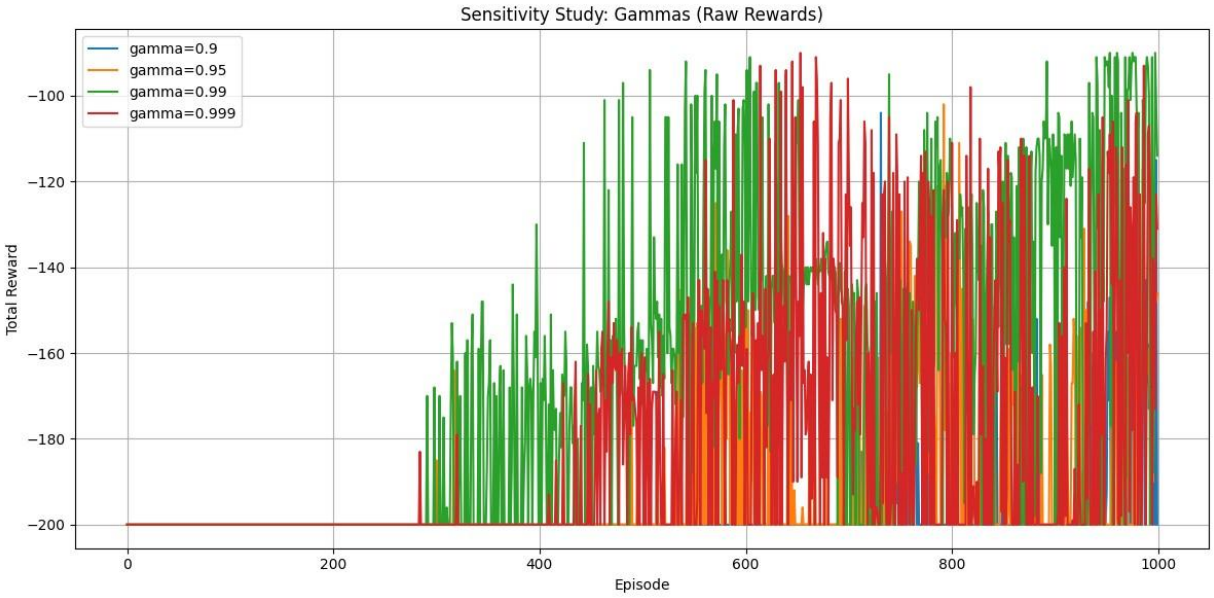


Η ανάλυση ευαισθησίας για το learning rate ανέδειξε πως οι τιμές 0.001 και 0.0006 προσφέρουν τις καλύτερες επιδόσεις στο περιβάλλον MountainCar-v0, με την τιμή 0.001 να παρουσιάζει σταθερή και ταχεία βελτίωση του συνολικού reward. Αντίθετα, μικρότερες τιμές όπως 0.0003 και ιδιαίτερα 0.0001 οδηγούν σε πολύ αργή μάθηση, με αποτέλεσμα ο πράκτορας να αποτυγχάνει να ξεφύγει από τις χειρότερες δυνατές επιδόσεις (reward κοντά στο -200). Συνεπώς, για το συγκεκριμένο πρόβλημα, η επιλογή learning rate στην περιοχή [0.0006, 0.001] θεωρείται καταλληλότερη, προσφέροντας καλύτερη ισορροπία μεταξύ ταχύτητας εκπαίδευσης και σταθερότητας του μοντέλου.

Μελέτη ως προς

2. gammas

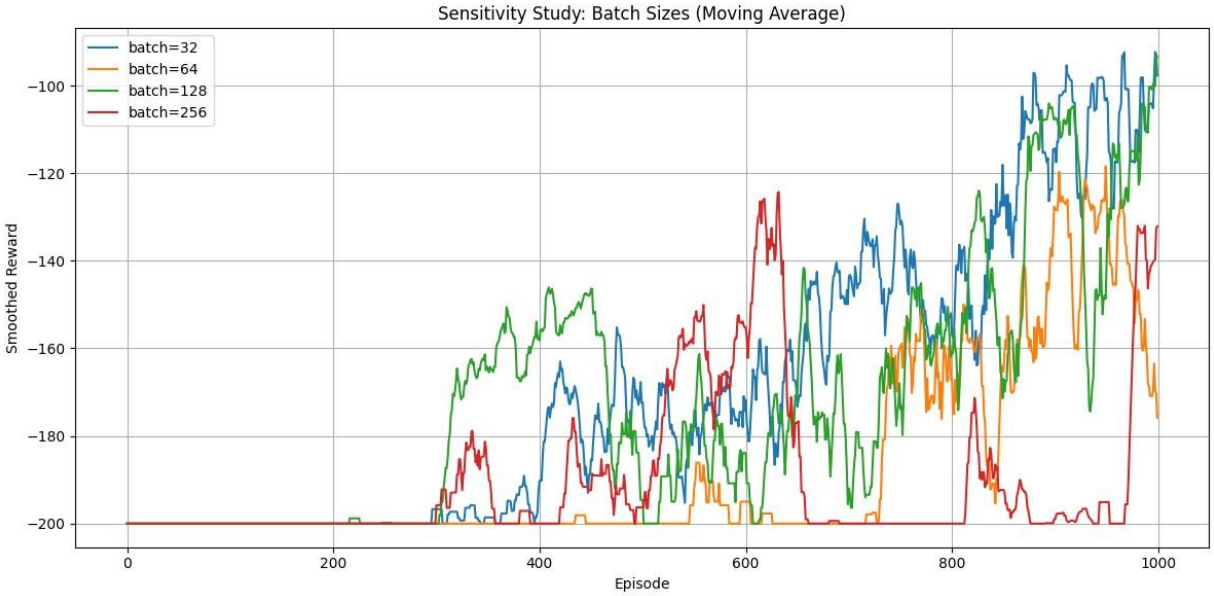


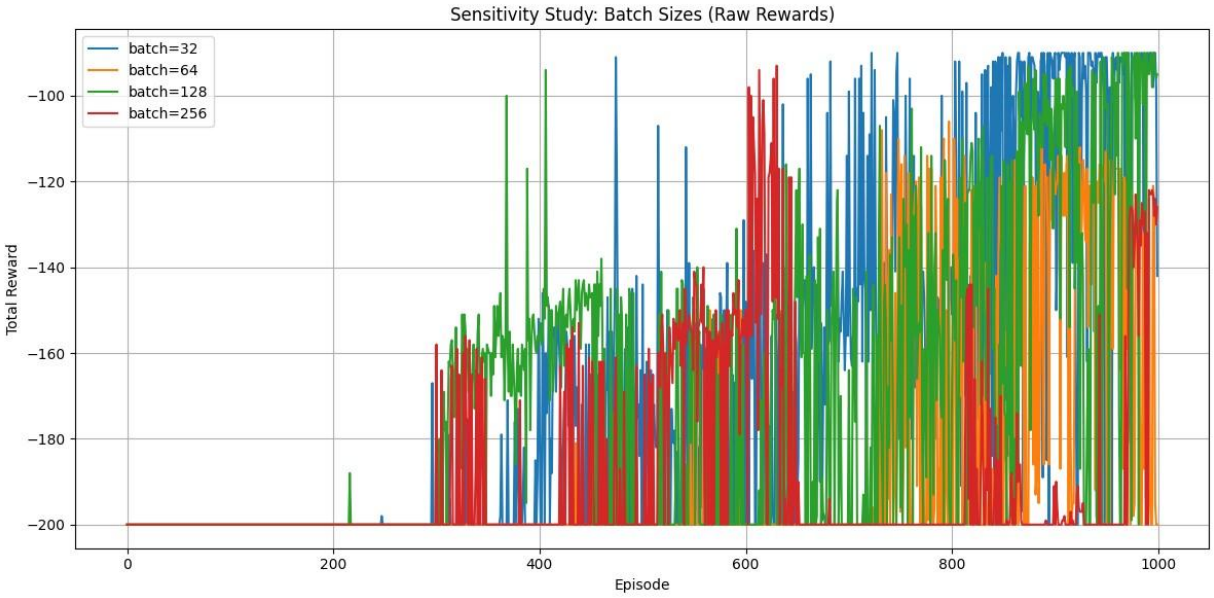


Η παραμετρική διερεύνηση του παράγοντα προεξόφλησης (γ) αποκάλυψε ότι οι τιμές κοντά στο 0.99 παρείχαν σημαντικά καλύτερη απόδοση σε σχέση με χαμηλότερες τιμές, όπως 0.9 ή 0.95. Συγκεκριμένα, το γ = 0.99 οδήγησε σε πιο γρήγορη σύγκλιση και υψηλότερα rewards, υποδεικνύοντας ότι η εκτίμηση μελλοντικών ανταμοιβών ήταν κρίσιμη για την επιτυχή πλοήγηση στο περιβάλλον MountainCar. Αντιθέτως, χαμηλότερες τιμές του γ περιόρισαν τη δυνατότητα του πράκτορα να αξιοποιήσει μακροπρόθεσμες ανταμοιβές, με αποτέλεσμα ανεπαρκή μάθηση. Το γ = 0.999, αν και διατηρεί καλή απόδοση, παρουσιάζει μεγαλύτερη αστάθεια, γεγονός που πιθανώς οφείλεται στην υπερβολική έμφαση στο μέλλον, με αντίτιμο την καθυστέρηση στην ενίσχυση βραχυπρόθεσμων στρατηγικών. Συνεπώς, το γ = 0.99 αναδεικνύεται ως η πλέον ισορροπημένη επιλογή για το συγκεκριμένο πρόβλημα.

Μελέτη ως προς

3. batch_sizes



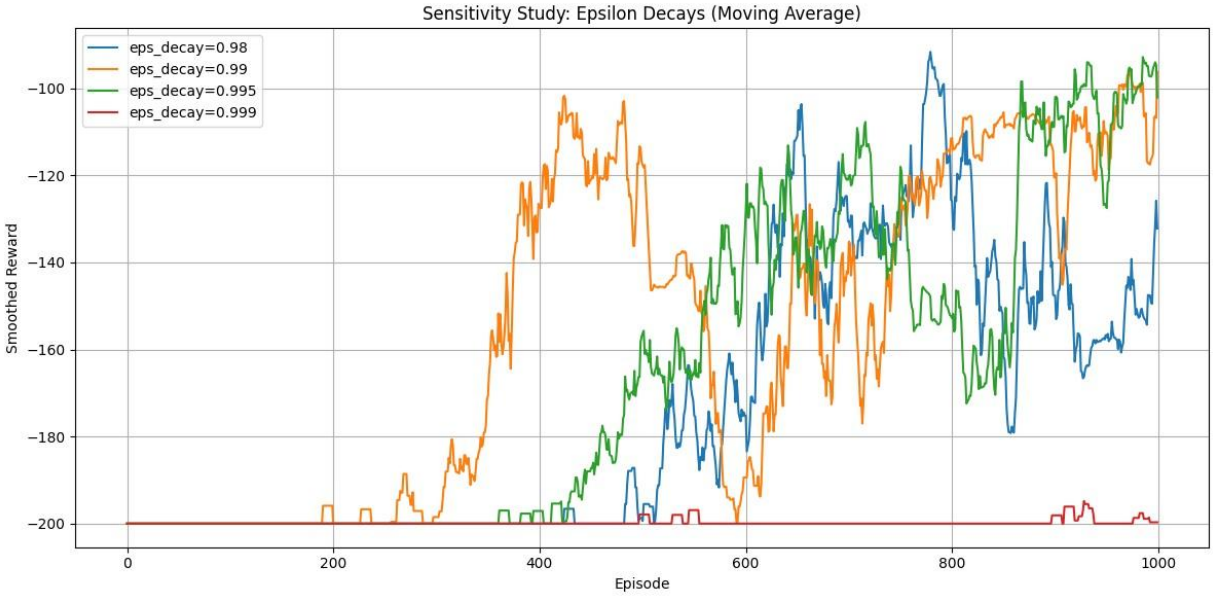


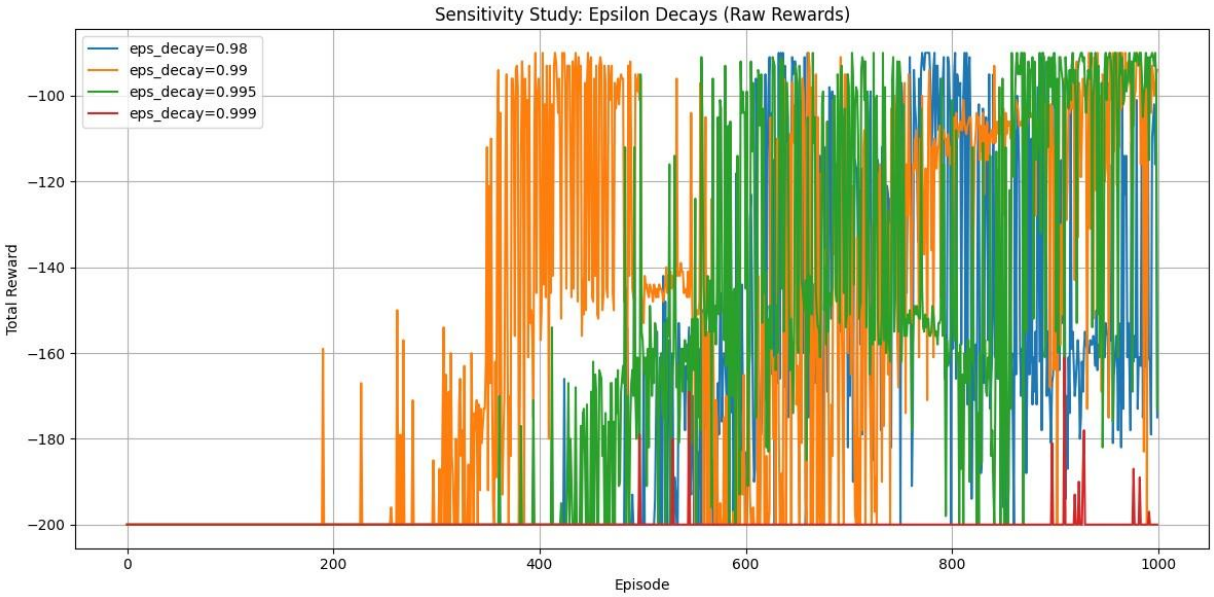
Η επιλογή του μεγέθους βατση φαίνεται να επηρεάζει σημαντικά την αποδοτικότητα και σταθερότητα της μάθησης του DQN πράκτορα. ΄Οπως παρατηρείται, το μέγεθος batch_size = 128 προσφέρει έναν καλό συμβιβασμό μεταξύ σταθερότητας και απόδοσης, επιτυγχάνοντας υψηλές ανταμοιβές με σχετικά ομαλή σύγκλιση. Αντιθέτως, το batch_size = 32 παρουσιάζει γρηγορότερη αρχική βελτίωση, αλλά συνοδεύεται από υψηλότερη διακύμανση, γεγονός που υποδηλώνει λιγότερο σταθερή συμπεριφορά του μοντέλου. Οι μεγαλύτερες τιμές, και ειδικά το batch_size = 256, υστερούν ξεκάθαρα, παρουσιάζοντας καθυστερημένη εκμάθηση και αστάθεια, πιθανώς λόγω της μειωμένης συχνότητας ενημερώσεων των παραμέτρων. Καταλήγουμε στο συμπέρασμα ότι για το συγκεκριμένο περιβάλλον (MountainCar-v0), τα ενδιάμεσα μεγέθη batch (όπως το 128) ενδείκνυνται για

Μελέτη ως προς

βέλτιστη ισορροπία μεταξύ ταχύτητας μάθησης και σταθερότητας, ενώ πολύ μικρές ή πολύ μεγάλες τιμές ενδέχεται να περιορίσουν την τελική απόδοση του αλγορίθμου.

4. epsilon_decays





Η παράμετρος epsilon_decay επηρεάζει τον ρυθμό με τον οποίο μειώνεται η εξερευνητική συμπεριφορά του αλγορίθμου κατά τη διάρκεια της μάθησης. Από τα αποτελέσματα, παρατηρείται ότι η τιμή epsilon_decay=0.99 οδηγεί σε ταχύτερη και σταθερότερη βελτίωση της απόδοσης, καθώς επιτυγχάνει υψηλότερες τιμές ρεωαρδ σε μικρότερο αριθμό επεισοδίων. Αντίθετα, η τιμή epsilon_decay=0.999 οδηγεί σε πολύ αργή σύγκλιση, καθυστερώντας σημαντικά την εκμάθηση, γεγονός που αποτυπώνεται στην παραμονή των ρεωαρδς κοντά στο -200 σε όλη τη διάρκεια της εκπαίδευσης.

Η τιμή epsilon_decay = 0.995 εμφανίζει ισορροπημένη συμπεριφορά, με ανοδική πορεία και αρκετά καλή απόδοση σε μετέπειτα επεισόδια, ενώ η πιο χαμηλή τιμή 0.98 φαίνεται να

Μελέτη ως προς
οδηγεί σε πρόωρη εκμετάλλευση, χωρίς να επιτρέπεται επαρκής εξερεύνηση του περιβάλλοντος, με αποτέλεσμα ασταθή και υποδεέστερη μάθηση.

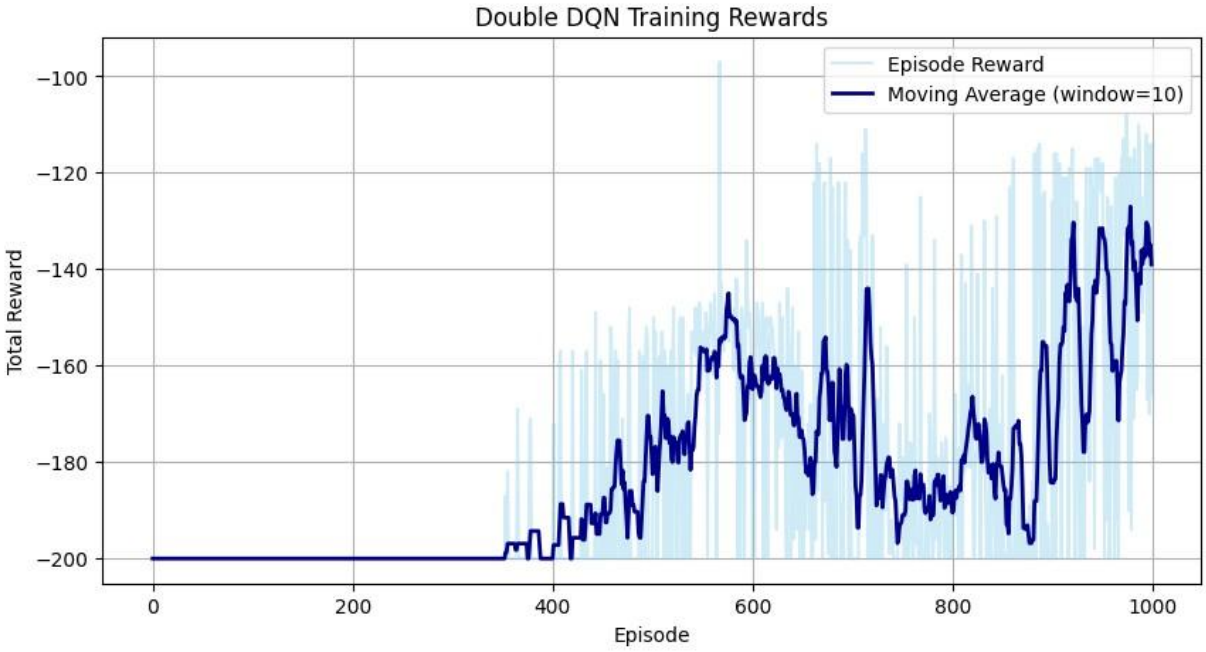
Συνολικά, φαίνεται ότι οι ενδιάμεσες τιμές epsilon_decay = 0.99 και 0.995 προσφέρουν την καλύτερη επίδοση, ενώ ακραίες τιμές (πολύ μικρή ή πολύ μεγάλη απόσβεση) δυσχεραίνουν τη μάθηση του πράκτορα.

3.1) Double DQN

Για την επίλυση του προβλήματος με τη χρήση του αλγορίθμου Double DQN έχουμε τον κώδικα που βρίσκεται στο αρχείο doubleDQN.py

Παρακάτω παρουσιάζεται το διάγραμμα συνολικής αμοιβής και μεσης συνολικής αμοιβής (των 10 τελευταιων επισοδιων) σε συνάρτηση με τα επεισόδια. Με τις παραμέτρους του προβλήματος να παίρνουν τις παρακάτω τιμές.

batch_size = 64 gamma
= 0.99
epsilon_start = 1.0 epsilon_end =
0.01 epsilon_decay = 0.995
target_update = 10
memory_capacity = 10000
learning_rate = 0.001
num_episodes = 1000
MAX_STEPS = 200 (default)



Το κλασικό DQN τείνει να υπερεκτιμά τις τιμές Q, επειδή το ίδιο δίκτυο χρησιμοποιείται τόσο για την επιλογή της καλύτερης επόμενης ενέργειας όσο και για την εκτίμηση της αντίστοιχης τιμής της. Η υπερεκτίμηση οδηγεί σε ασταθή εκπαίδευση και υποδεέστερες πολιτικές.

Για να επιλύσουμε αυτό το πρόβλημα σχεδιάζουμε τον Double DQN, ο οποιος διαχωρίζει την επιλογή από την αξιολόγηση της δράσης: η επιλογή γίνεται από το policy network, ενώ η αξιολόγηση από το target network.

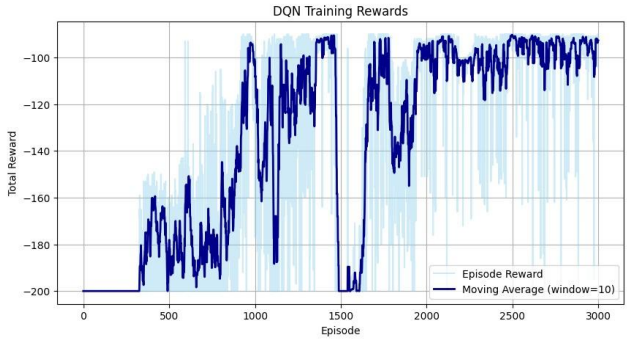
Ο στόχος από $y = r + \gamma \cdot \max_{a'} Q_{\text{target}}(s',a')$, γίνεται: $y = r + \gamma \cdot Q_{\text{target}}\left(s', \text{argmax}_a Q_{\text{policy}}(s',a)\right)$

Μελέτη ως προς

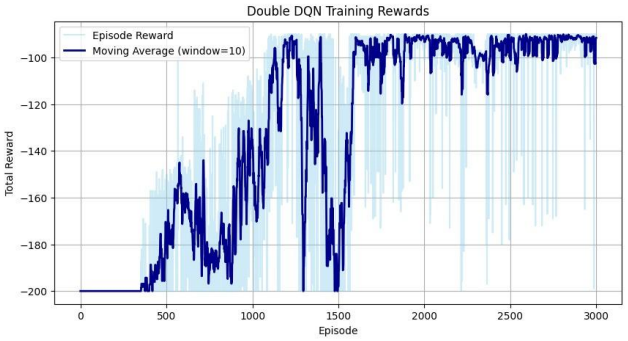
όπου Q_{target} είναι το (παγωμένο) target network, r η άμεση ανταμοιβή και γ ο παράγοντας βαρύτητας μελλοντικών ανταμοιβών. Έτσι πετυχαίνουμε να μειώσουμε το (bias) και να βελτιώσουμε τη σταθερότητα της μάθησης.

Για την υλοποίηση του αλγορίθμου Double DQN, η μόνη αλλαγή που χρειάστηκε να κάνουμε στον βασικό κώδικα είναι η τροποποίηση του υπολογισμού του στόχου (target Q-value). Αντί να επιλέγουμε την επόμενη δράση με το target network όπως στο κλασικό DQN, επιλέγουμε τη δράση που μεγιστοποιεί την τιμή του policy network και υπολογίζουμε την αντίστοιχη τιμή με βάση το target network. Αυτό γίνεται με την εντολή: next_actions = policy_net(next_states).argmax(1)
next_q_values = target_net(next_states).gather(1,next_actions.unsqueeze(1)).squeeze(1)

Για την σύγκριση των DQN και Double DQN στο MountainCar-v0 εκτελούμε τους δύο αλγόριθμους για 3000 επεισόδια και με τις ίδιες υπερπαραμέτρους. Έτσι παίρνουμε τα παρακάτω διαγράμματα.



(α') DQN (baseline).



(β') Double DQN.

Από τα παραπάνω διαγράμματα παρατηρούμε ότι ο αλγόριθμος Double DQN επιτυγχάνει ταχύτερη και πιο σταθερή σύγκλιση συγκριτικά με τον απλό DQN. Οι διακυμάνσεις του συνολικού reward μειώνονται σημαντικά, ενώ η μέση απόδοση σταθεροποιείται κοντά στο βέλτιστο (-110 έως -100) σε μικρότερο αριθμό επεισοδίων. Αυτό τεκμηριώνει εμπειρικά την θεωρητική υπόθεση ότι το Double DQN μειώνει την υπερεκτίμηση των τιμών Q, οδηγώντας σε πιο αποτελεσματική εκπαίδευση.

3.2) Ανάλυση ευαισθησίας Double DQN ως προς τις υπερπαραμέτρους

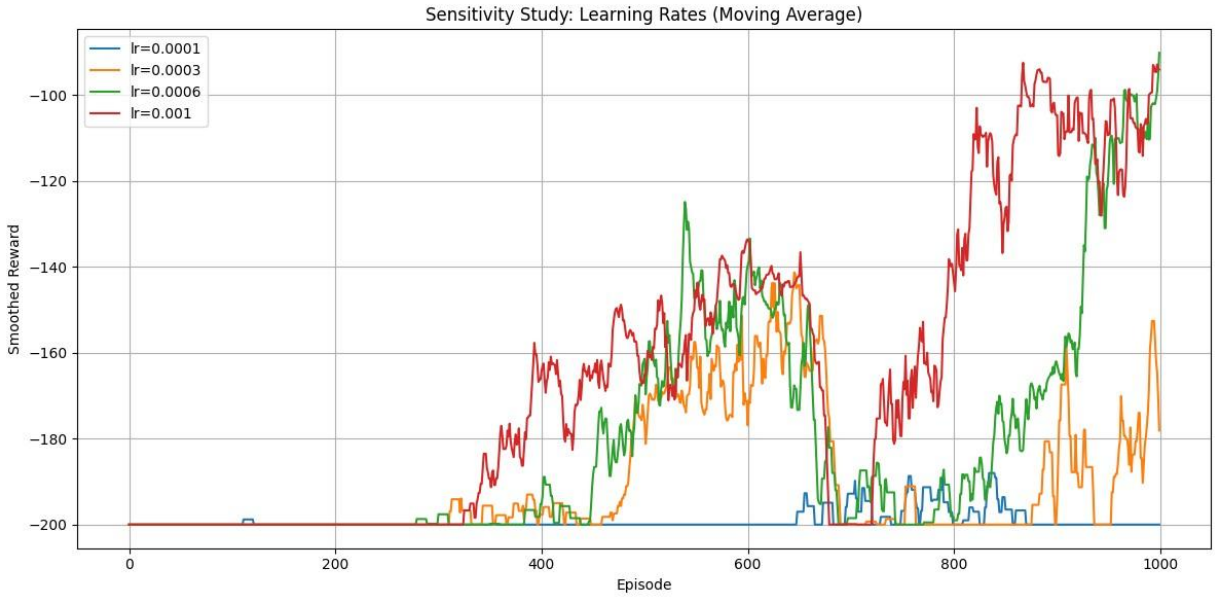
Για την ανάλυση ευαισθησίας του Double DQN ακολουθήσαμε τη μεθοδολογία του ερωτήματος 2.2, αλλά εφαρμόζοντας το βελτιωμένο μοντέλο με διαχωρισμό επιλογής και αξιολόγησης δράσεων. Χρησιμοποιήθηκε αντίστοιχος κώδικας που βρίσκεται στο αρχείο doubleDqn_4parameters.py, με τροποποίηση στο target update ώστε να υλοποιεί τον υπολογισμό στόχου ως:

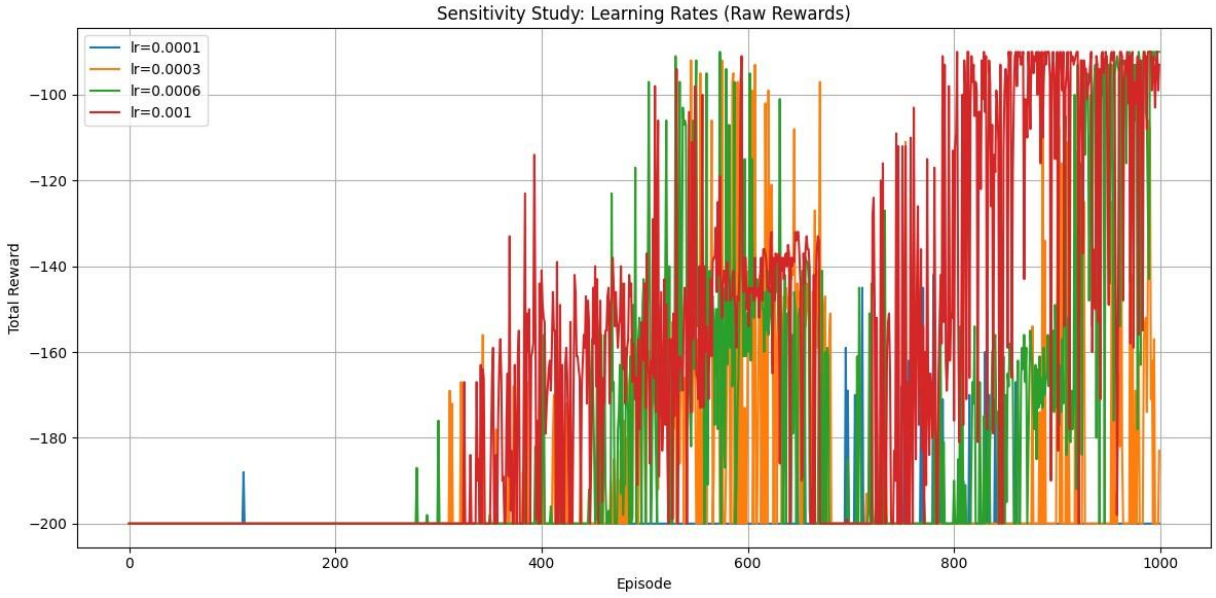
$$y = r + \gamma \cdot Q_{\text{target}}(s', \underset{a}{\text{argmax}} Q_{\text{policy}}(s', a))$$

Οι τιμές των υπερπαραμέτρων που εξετάζουμε παραμένουν ίδιες με πριν:

1. learning_rates = {0.0001, 0.0003, 0.0006, 0.001}
2. gammas = {0.9, 0.95, 0.99, 0.999}
3. batch_sizes = {32, 64, 128, 256}
4. epsilon_decays = {0.98, 0.99, 0.995, 0.999}

1. learning_rates.





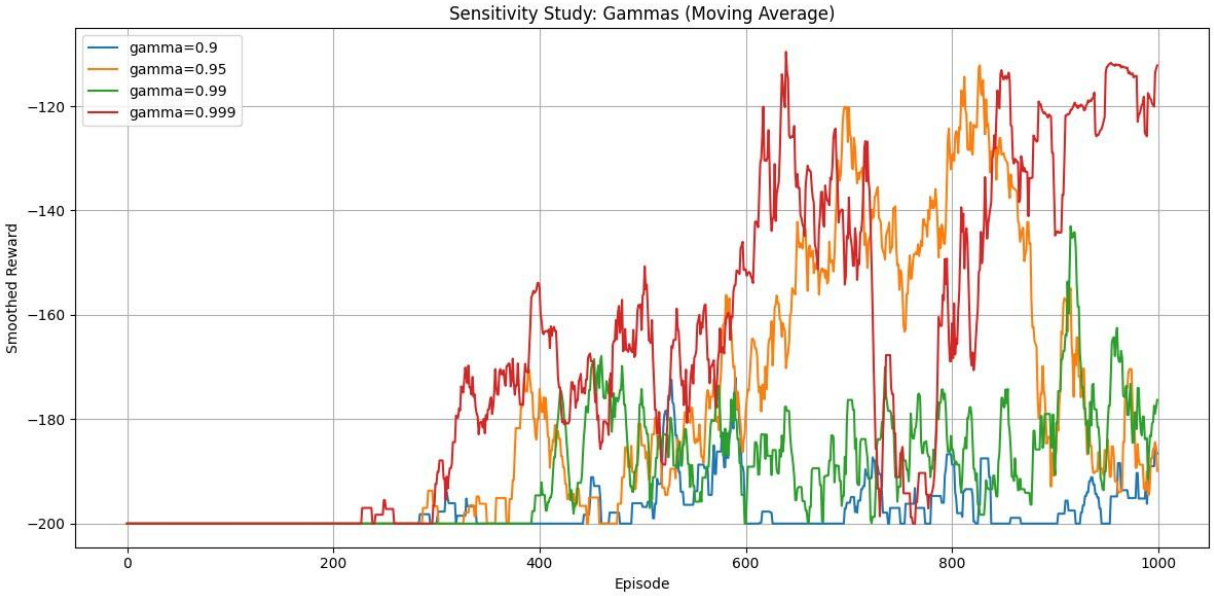
Η επίδραση της παραμέτρου learning rate είναι εμφανής και στο Double DQN, με τις τιμές 0.001 και 0.0006 να οδηγούν σε ταχύτερη σύγκλιση και καλύτερη σταθερότητα. Αντίθετα, χαμηλότερες τιμές όπως 0.0001 προκαλούν σημαντική καθυστέρηση στην εκμάθηση. Σε

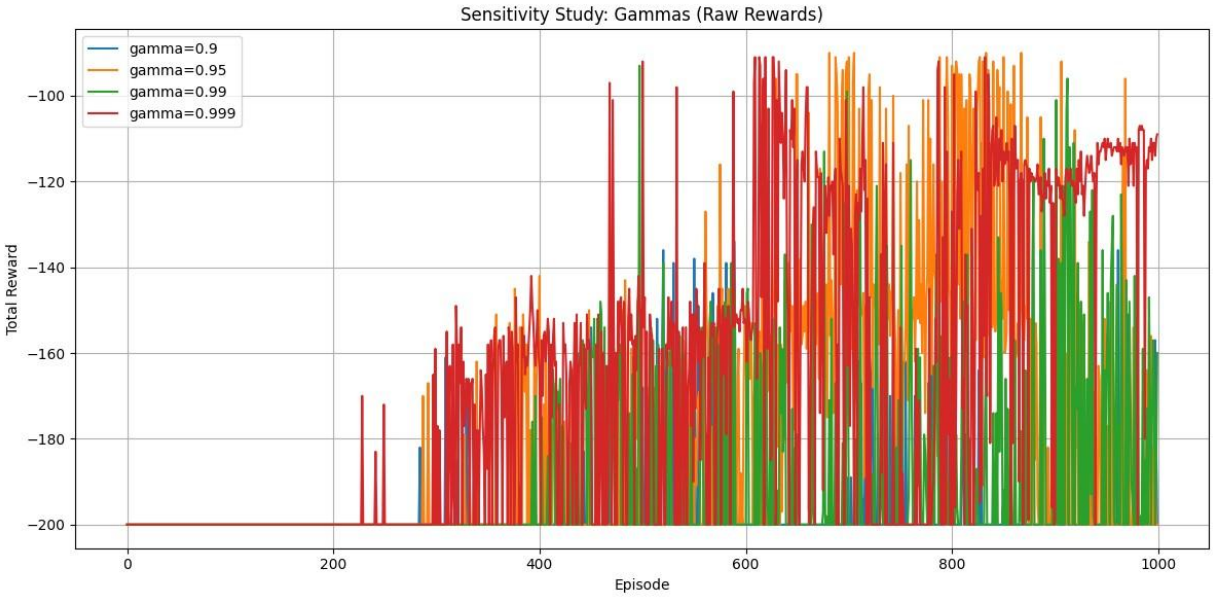
Μελέτη ως προς

σύγκριση με το απλό DQN, παρατηρούμε συνολικά μεγαλύτερη σταθερότητα για ίδιες τιμές, γεγονός που επιβεβαιώνει ότι η αρχιτεκτονική του Double DQN προσφέρει μειωμένη διακύμανση κατά την ενημέρωση των τιμών Q και καλύπτει την αστάθεια που προσφέρουν μεγαλύτερες τιμές της παραμέτρου learning rate.

Μελέτη ως προς

2. gammas.



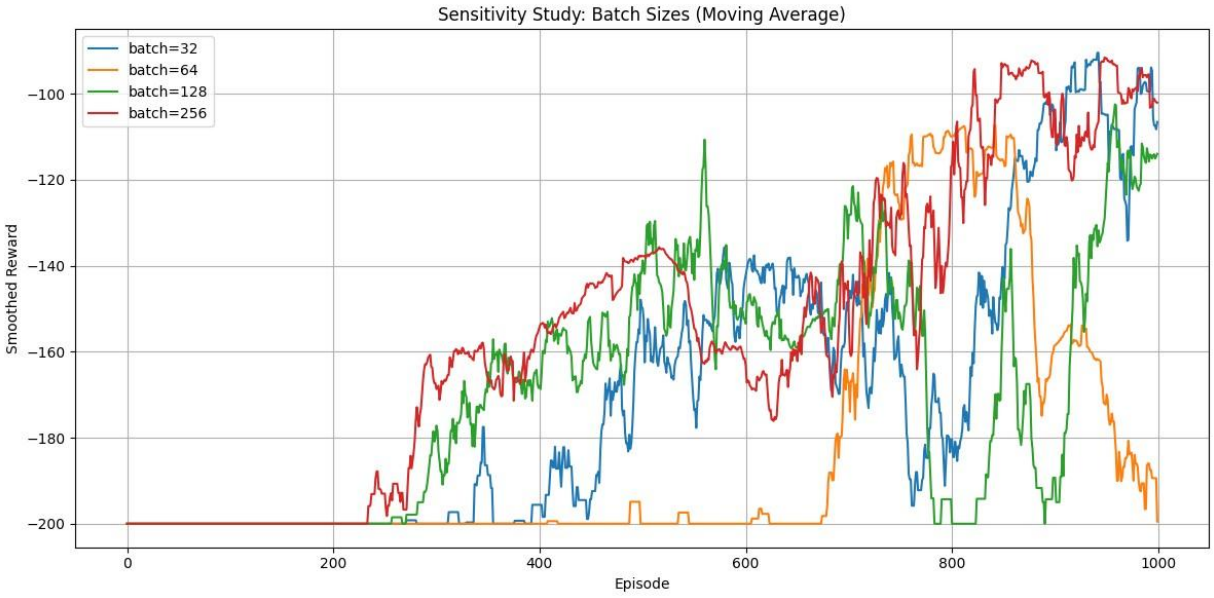


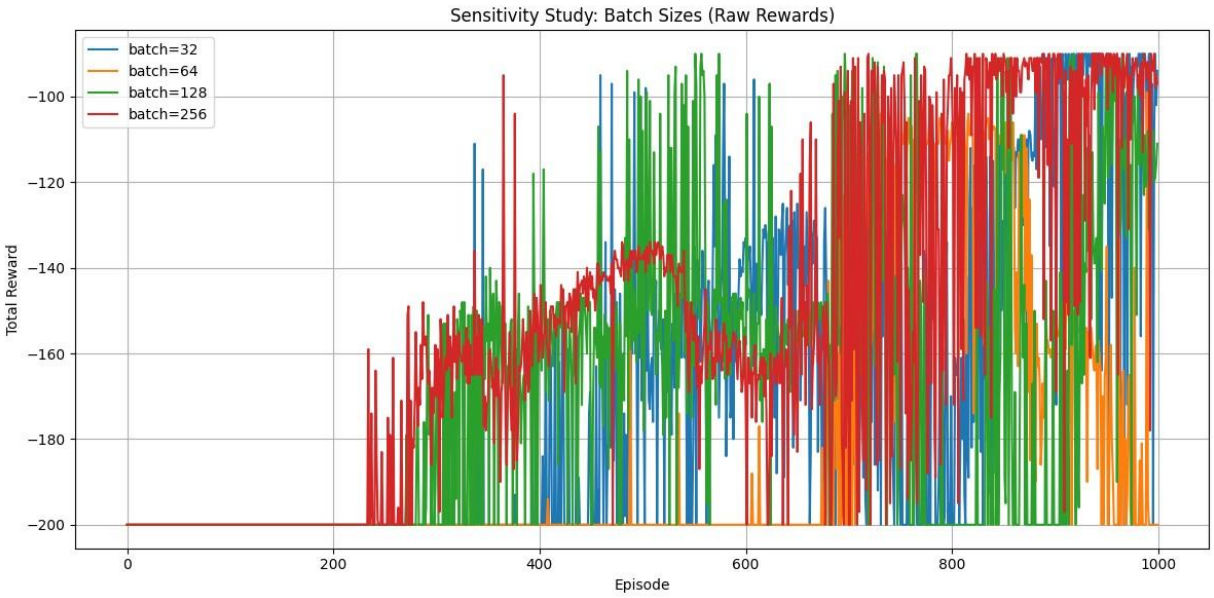
Η ευαισθησία ως προς τον παράγοντα προεξόφλησης (γ) παρουσιάζει παρόμοια μοτίβα με το DQN, με τις τιμές 0.99 και 0.999 να υπερέχουν. Ωστόσο, ο Double DQN εμφανίζει λιγότερες διακυμάνσεις, ιδιαίτερα για την τιμή 0.999, η οποία στο απλό DQN προκαλούσε υπερβολική

Μελέτη ως προς

αστάθεια. Αυτό δείχνει ότι το Double DQN μπορεί να χειριστεί καλύτερα υψηλές τιμές γ, αξιοποιώντας μελλοντικές ανταμοιβές χωρίς να υπερεκτιμά τις τιμές Q.

3. batch_sizes.





Η επίδοση του Double DQN φαίνεται να βελτιστοποιείται για μεγέθη batch μεταξύ 128 και

Μελέτη ως προς

256, με την τιμή 256 να εμφανίζει ιδιαίτερα καλή σταθερότητα. Αυτό διαφοροποιείται από

δείχνει να επωφελείται από μεγαλύτερα βατση λόγω της σταθερότερης εκτίμησης και της μειωμένης διασποράς κατά την επιλογή δράσης.

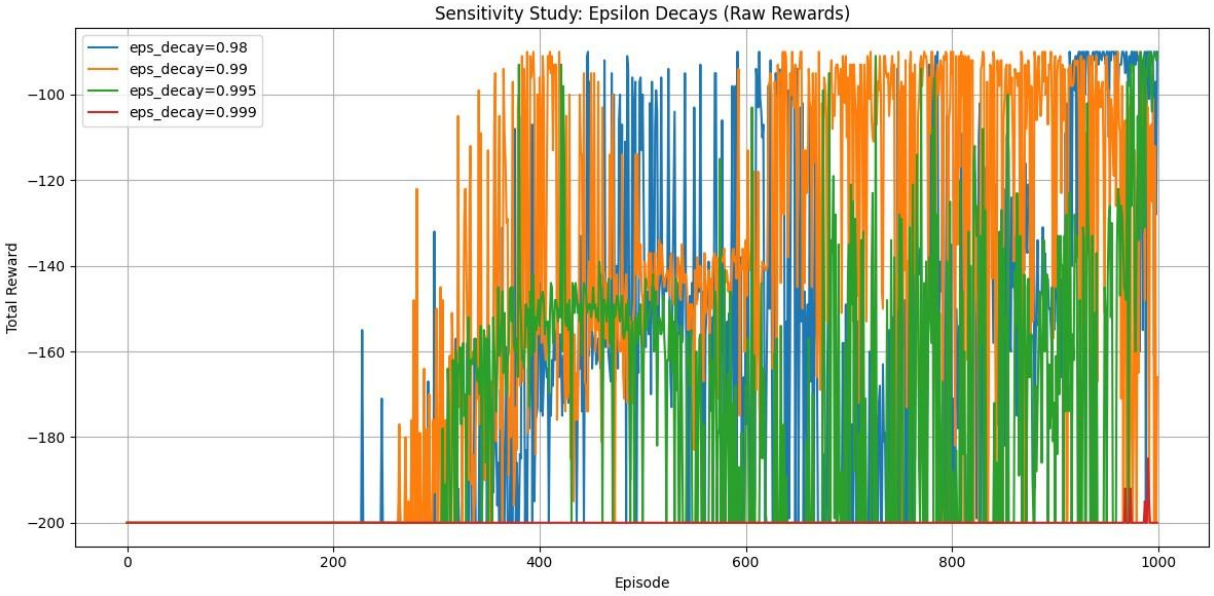
το

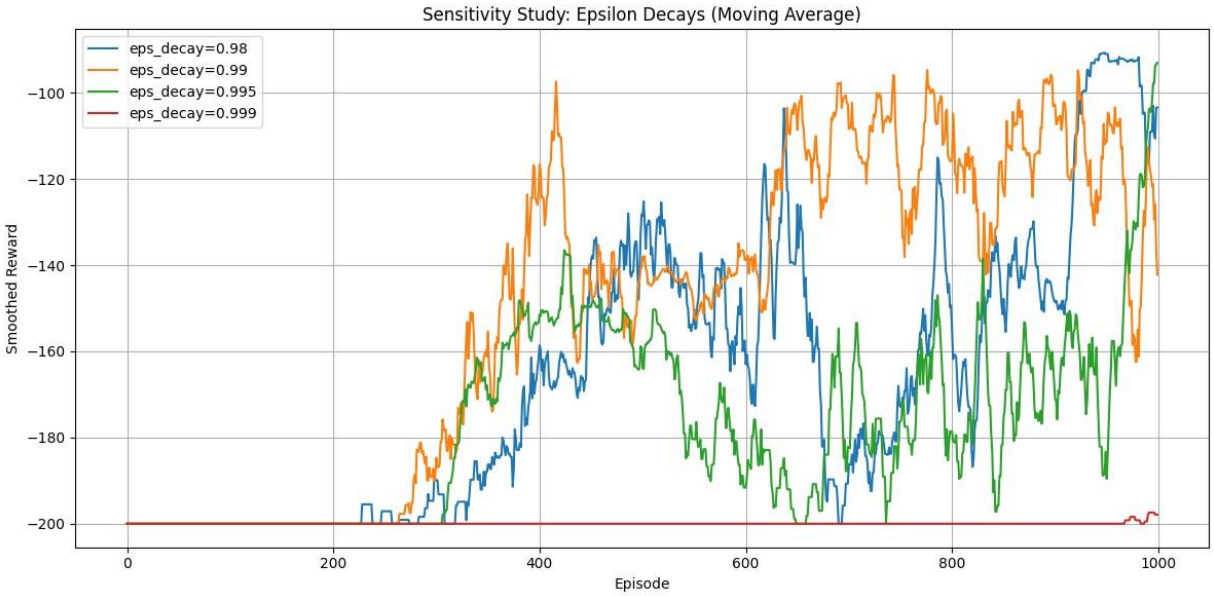
απλό

DQN Q

DQN, όπου τα μεγαλύτερα batch προκαλούσαν καθυστέρηση και αστάθεια. Το Double

4. epsilon_decays.





Μελέτη ως προς

΄Οσον αφορά τη στρατηγική εξερεύνησης, οι ενδιάμεσες τιμές epsilon_decay = 0.99, 0.995 προσφέρουν και εδώ την καλύτερη απόδοση, επιτρέποντας στον πράκτορα να εξερευνήσει επαρκώς στην αρχή και να συγκλίνει έγκαιρα σε επιτυχημένες πολιτικές. Αντίθετα, η πολύ αργή απόσβεση της εξερεύνησης για epsilon_decay = 0.999 οδηγεί σε αποτυχία της μάθησης. Η τιμή epsilon_decay = 0.98 προκαλεί πρόωρη εκμετάλλευση, περιορίζοντας την εξερεύνηση και οδηγώντας σε χαμηλότερη συνολική απόδοση. Συνολικά, επιβεβαιώνεται ότι οι ενδιάμεσες τιμές προσφέρουν τον βέλτιστο συμβιβασμό μεταξύ εξερεύνησης και εκμετάλλευσης, ακόμη και στην πιο σταθερή εκδοχή του Double DQN.

4.1) DRQN (LSTM-based DQN)

Για τη διερεύνηση της επίδρασης διαφορετικών αρχιτεκτονικών στον DQN, επανασχεδιάζουμε τον εκτιμητή Q_θ ως Deep Recurrent Q-Network (DRQN) με LSTM. Η βασική ιδέα είναι ότι, αντί για MLP που λαμβάνει ένα μεμονωμένο s_t, το δίκτυο δέχεται μία μικρή ακολουθία καταστάσεων $(s_{t-L+1},\ldots,s_t)$, περνά από LSTM (για να ενσωματώσει χρονική εξάρτηση) και παράγει $Q(s_t)$ για το τελευταίο βήμα. Έτσι, το μοντέλο μπορεί να εκμεταλλευτεί καλύτερα τις βραχυπρόθεσμες εξαρτήσεις.

Αρχιτεκτονική. Το δίκτυο αποτελείται από:

- Γραμμικό επίπεδο εισόδου Linear(obs_dim,hidden) με ReLU,
- LSTM με batch_first = True που λαμβάνει σειρές μήκους L,
- Γραμμικό επίπεδο εξόδου Linear(hidden,$|A|$) που δίνει Q-τιμές.

Κατά το forward επιστρέφουμε τις Q-τιμές του τελευταίου χρονικού βήματος στην ακολουθία (τύπικη πρακτική στο DRQN).

Ενημέρωση στόχου (target) και μάθηση. Χρησιμοποιούμε ξεχωριστό target network με παραμέτρους θ^- και κλασικό DQN στόχο:

$$y_t = r_t + \gamma \max_{a'} Q_{\theta^-}(s'_t, a')$$

Η απώλεια είναι μέσο τετραγωνικό σφάλμα:

$$\mathcal{L}(\theta) = \big(Q_\theta(s_t, a_t) - y_t\big)^2$$

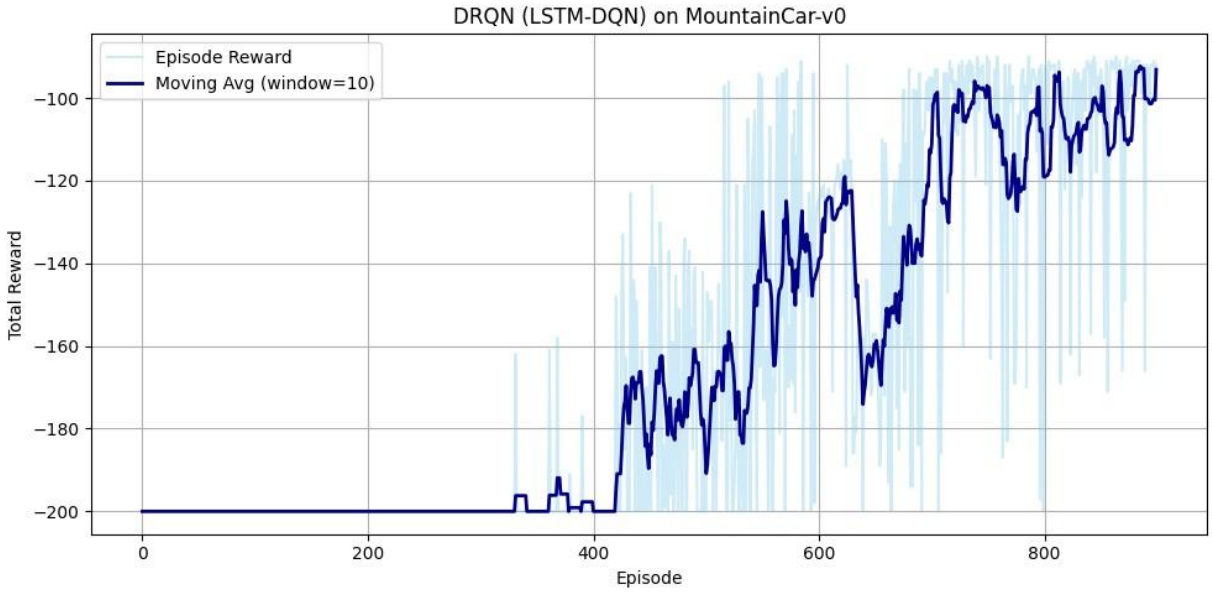
Υπερπαράμετροι που χρησιμοποιήθηκαν:

Μεταβλητή	Τιμή
random_update	True
batch_size	32
hidden_size	128
learning_rate	10^{-3}
γ	0.99
epsilon start / end / decay	0.5 / 0.01 / 0.995
target_update_frequency	10

Μελέτη ως προς

soft update τ	0.05

Πίνακας 1: Υπερπαράμετροι DRQN (LSTM).



Σχήμα 2: DRQN (LSTM) στο MountainCar-v0 με Bootstrapped Random Updates.

Από το παραπάνω γράφημα παρατηρούμε ότι η προσθήκη αναδρομικής μνήμης μέσω LSTM βελτιώνει τη σταθερότητα και την απόδοση. Στο MountainCar-v0, όπου είναι πολύ σημαντικός ο συνδυασμός θέσεις ταχύτητας και επιτάχυνσης, ο DRQN μαθαίνει ταχύτερα πότε να αυξήσει ταχύτητα πριν την ανάβαση, οδηγώντας σε καλύτερη επίλυση από το αντίστοιχο DQN-MLP με ίδιες υπερπαραμέτρους.

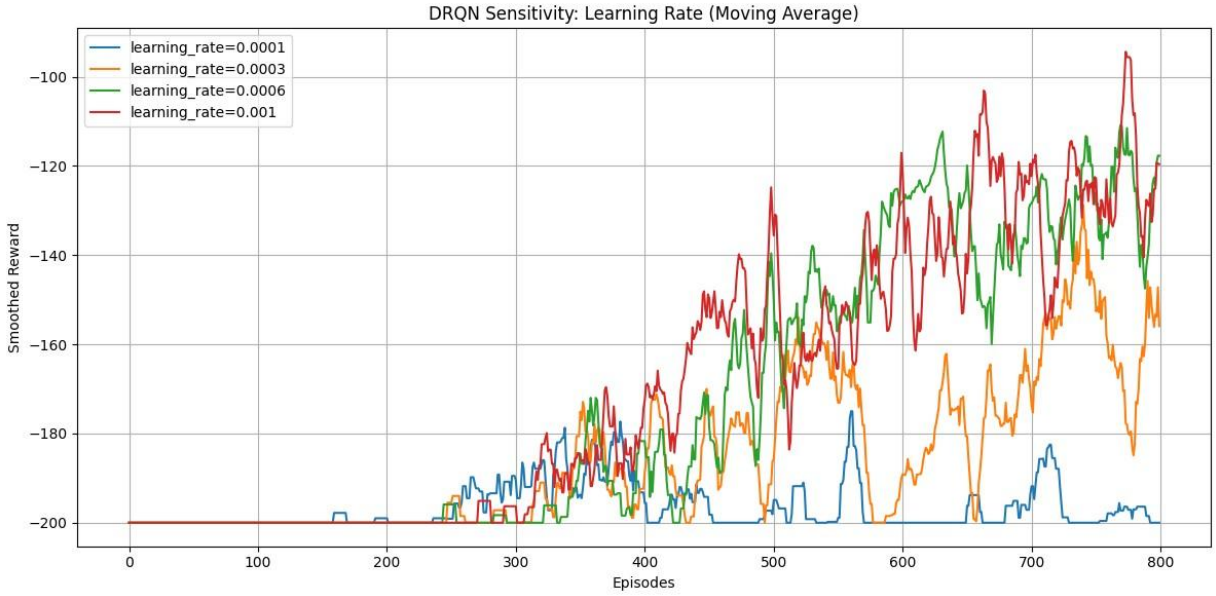
4.2) Ανάλυση ευαισθησίας Double DRQN ως προς τις υπερπαραμέτρους

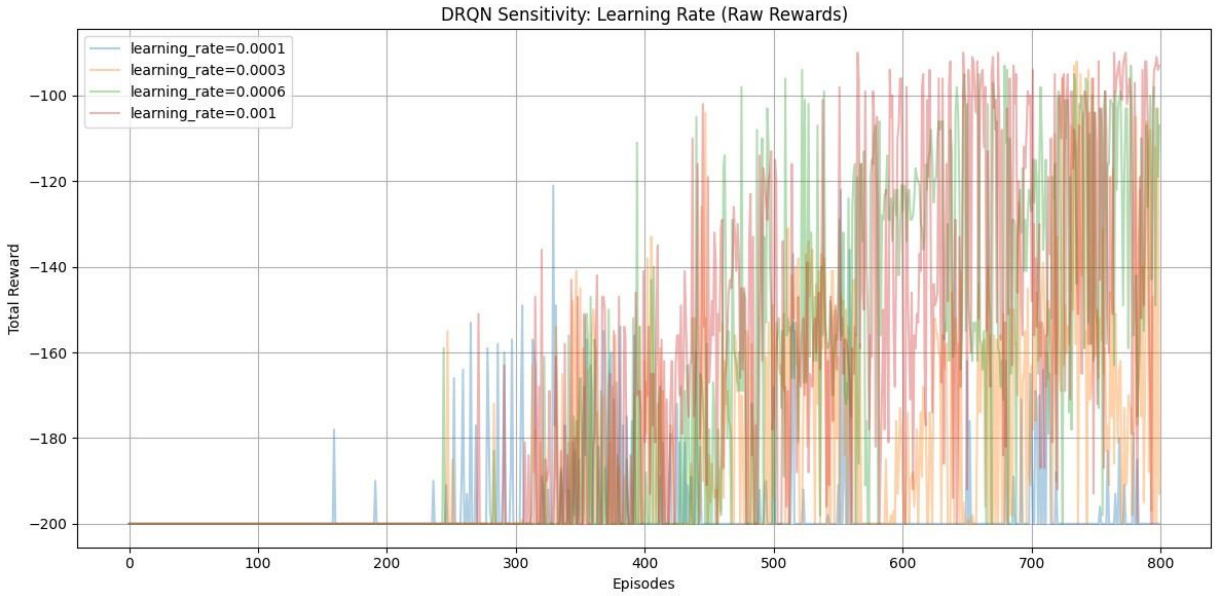
Για να μελετήσουμε την συμπεριφορά του αλγορίθμου σε συνάρτηση με 4 παραμέτρους έχουμε υλοποιήσει τον κώδικα που βρίσκεται στο αρχείο drqn_4parameters.py Επιλεγεγουμε τις παρακατω τιμες:

1. learning_rates = {0.0001, 0.0003, 0.0006, 0.001}
2. gammas = {0.9, 0.95, 0.99, 0.999}

3. batch_sizes = {32, 64, 128, 256}
4. epsilon_decays = {0.98, 0.99, 0.995, 0.999}

1. learning_rates.



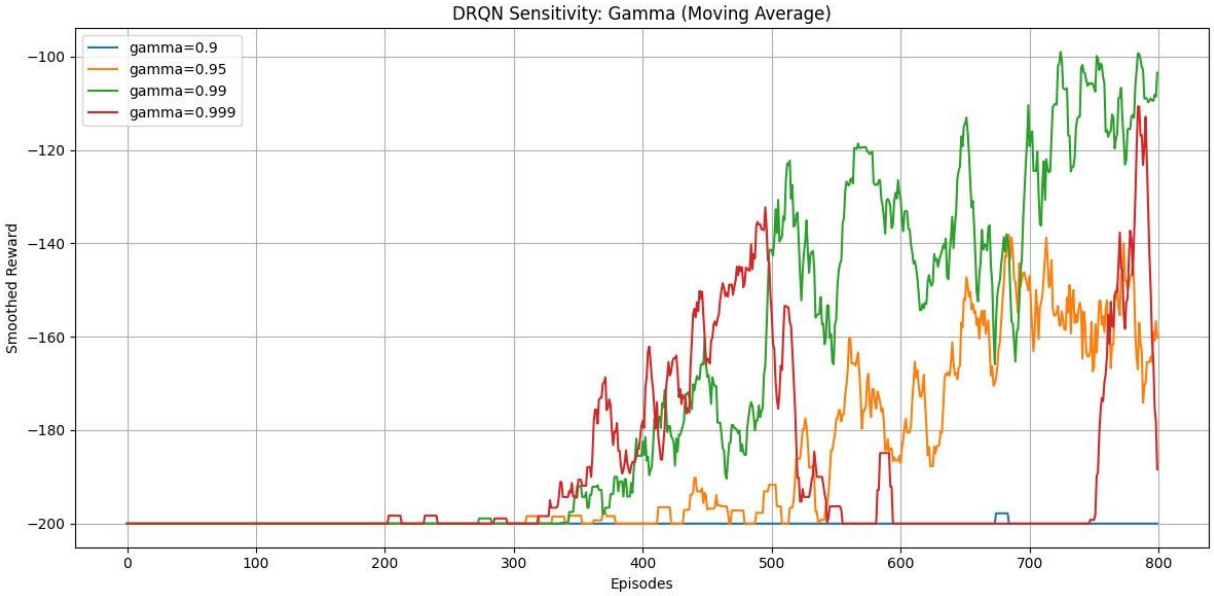


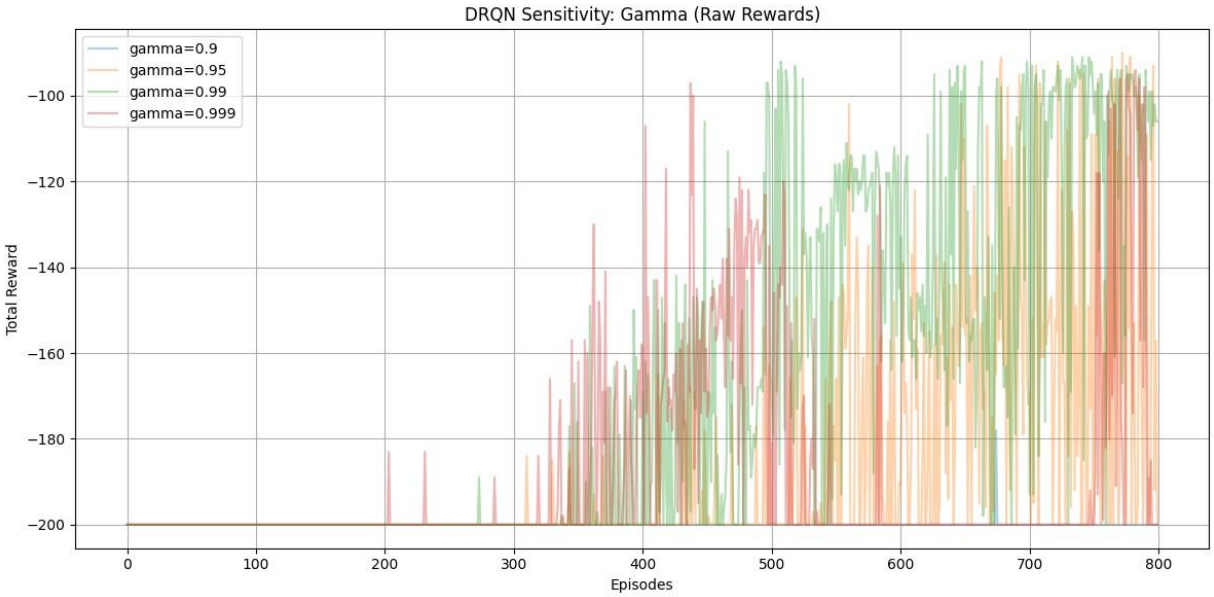
Μελέτη ως προς

Από το παραπάνω γράφημα παρατηρούμε ότι και στον συγκεκριμένο αλγόριθμο θα μπλέξουμε μεγαλύτερες τιμές για την παράμετρολearning_rates καθώς έχουμε πιο γρήγορη εκμάθηση χωρίς να υπάρχει πρόβλημα με την σταθερότητα του αλγορίθμου την οποία διορθώνει την οποία διορθώνει ο DRQN

Μελέτη ως προς

2. gammas.



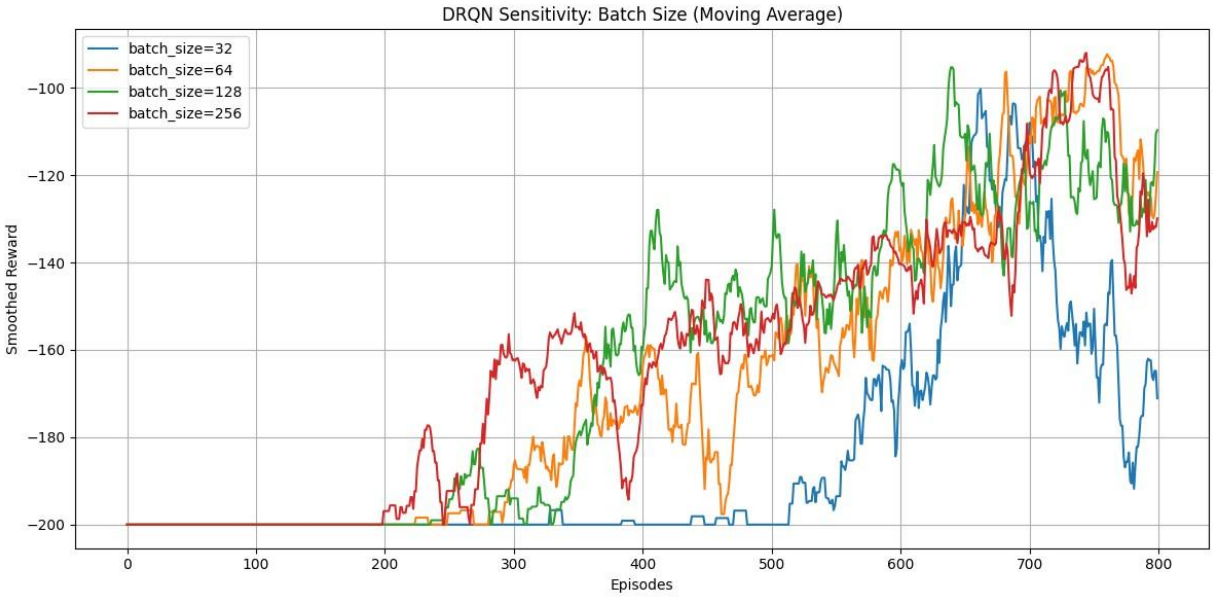


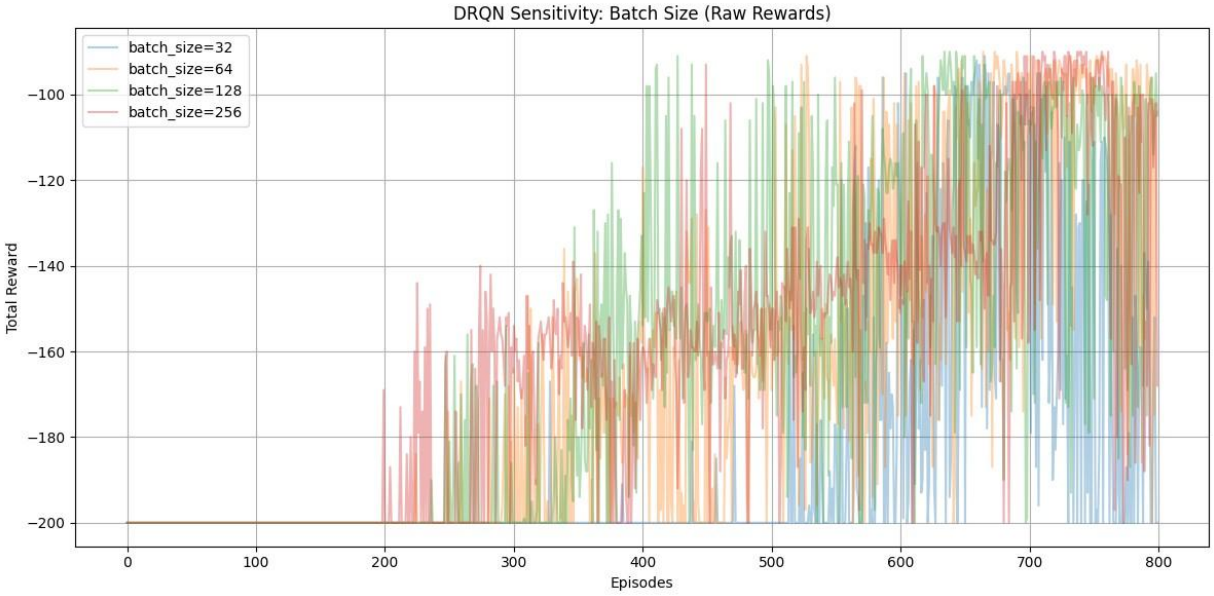
Παρατηρούμε ότι σε αντίθεση από τους προηγούμενους αλγόριθμους που οι αποδόσεις ήταν πιο κοντά μεταξύ τους στον DRQN η τιμή της παραμέτρου 0.99 απέδωσε αρκετά καλύτερα από τις υπόλοιπες το οποίο ερμηνεύεται από τις καλύτερες αποφάσεις που λαμβάνει ο

Μελέτη ως προς

αλγόριθμος με βάση την χρονική εξάρτηση. Αντίστοιχα η τιμή 0.999 ήταν υπερβολικά αισιόδοξη για την προεξόφλησή της ανταμοιβής, συνεπώς είχαμε την μεγάλη αστάθεια που φαίνεται στα διαγράμματα.

3. batch_sizes.

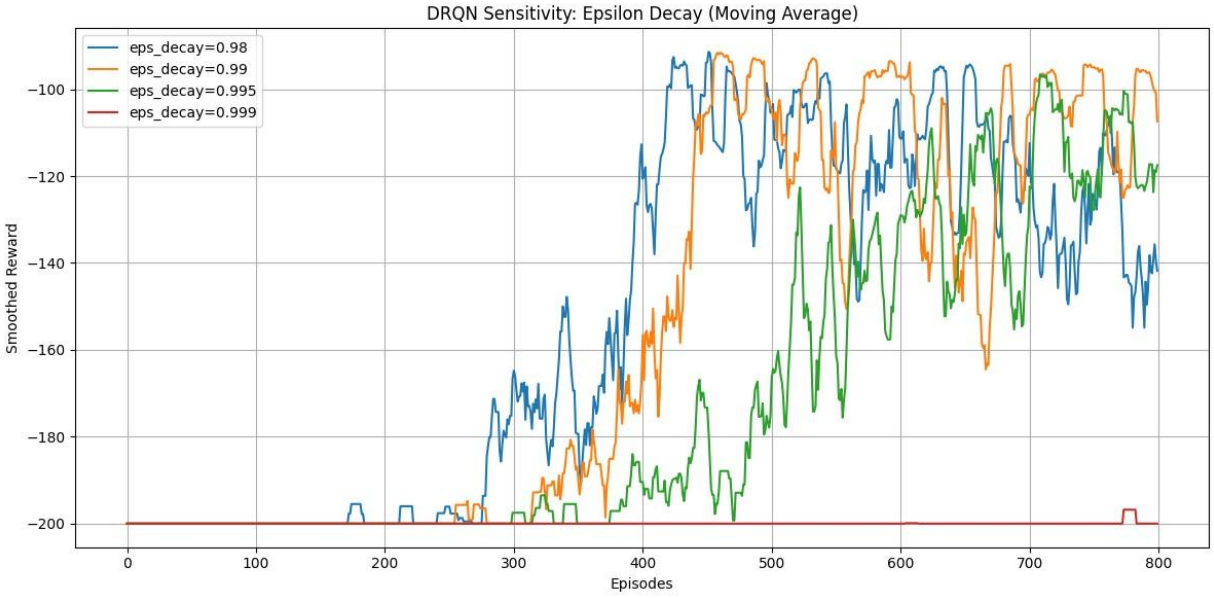


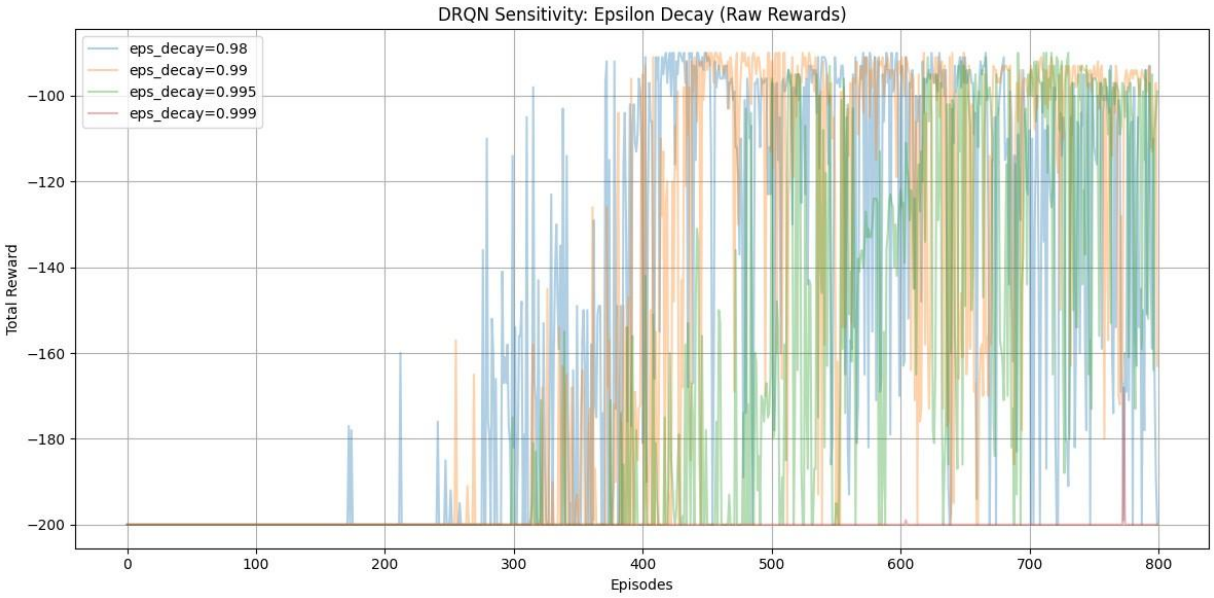


Μελέτη ως προς

Από το παραπάνω γράφημα παρατηρούμε ότι όλες οι τιμές εκτός από την 32 έχουν μία παρόμοια συμπεριφορά του προς την απόδοση και την σταθερότητα. Τα παραπάνω αποτελέσματα ήταν αναμενόμενο καθώς αν μέγεθος του batch_sizes είναι 32 δεν υπάρχει επαρκής χώρος για αρκετή πληροφορία την οποία χρειάζεται ένα δίκτυο LSTM.

4. epsilon_decays.





Μελέτη ως προς

Από τις παραπάνω γραφικές παραστάσεις παρατηρούμε ότι η απόδοση του αλγορίθμου είναι καλή για τιμές μικρότερες οι ίσες από το 0.995, καθώς επιτυγχάνει την καλύτερη ισορροπία ανάμεσα στην εξερεύνηση και την εκμετάλλευση. Οι μεγαλύτερες τιμές διατηρούν την τυχαιότητα για πολύ περισσότερα επεισόδια, καθυστερώντας την εκμάθηση. Επομένως, η τιμή

0.995 είναι πιο κατάλληλη, καθώς συνδυάζει σταθερή εκμάθηση με ικανοποιητική απόδοση.

4.3) PPO and A2C

Proximal Policy Optimization (PPO)

Ο αλγόριθμος Proximal Policy Optimization (PPO) ανήκει στην κατηγορία των μεθόδων Policy Gradient και έχει σχεδιαστεί για να εκπαιδεύει πράκτορες με μεγαλύτερη σταθερότητα σε σχέση με παραδοσιακές προσεγγίσεις. Η βασική ιδέα είναι ότι αντί να πραγματοποιούμε πολύ μεγάλες ενημερώσεις στην πολιτική (που μπορεί να αποσταθεροποιήσουν τη μάθηση), το PPO εφαρμόζει έναν περιορισμένο στόχο βελτιστοποίησης, ώστε η νέα πολιτική να μην αποκλίνει υπερβολικά από την παλιά.

Ο αντικειμενικός σκοπός της πολιτικής δίνεται αρχικά ως:

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min r_t(\theta)\hat{A}_t, \text{clip}^{(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t}\right],$$

όπου:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{ολδ}}}(a_t|s_t)}$ είναι το πηλίκο πιθανοτήτων μεταξύ της νέας πολιτικής και της παλιάς.

- $\hat{A}_t$ είναι η εκτίμηση του advantage, δηλαδή πόσο καλύτερη ήταν η ενέργεια a_t από τον μέσο όρο των πιθανών ενεργειών.

- ϵ είναι μία μικρή σταθερά που περιορίζει πόσο μπορεί να αποκλίνει η νέα πολιτική.

Για μεγαλύτερη σταθερότητα, το τελικό αντικειμενικό κριτήριο δεν περιλαμβάνει μόνο τον policy loss, αλλά και δύο πρόσθετους όρους: έναν όρο σφάλματος της Value Function και έναν όρο Entropy Bonus. Συνεπώς, το πλήρες κριτήριο είναι:

$$L(\theta) = \hat{E}_t\left[L_{\text{CLIP}t}(\theta) + c_v \cdot (V_\theta(s_t) - V_{t\text{target}})_2 + c_e \cdot H[\pi_\theta](s_t)\right],$$

όπου:

- $V_\theta(s_t)$ είναι η εκτίμηση της συνάρτησης αξίας (Value Function) από το δίκτυο.

- $V_{t^{\text{ταργετ}}}$ είναι ο στόχος (αναμενόμενη αθροιστική ανταμοιβή) που χρησιμοποιείται για την εκπαίδευση.

- $H[\pi_\theta](s_t)$ είναι η εντροπία της πολιτικής, που μετρά πόσο στοχαστική είναι.

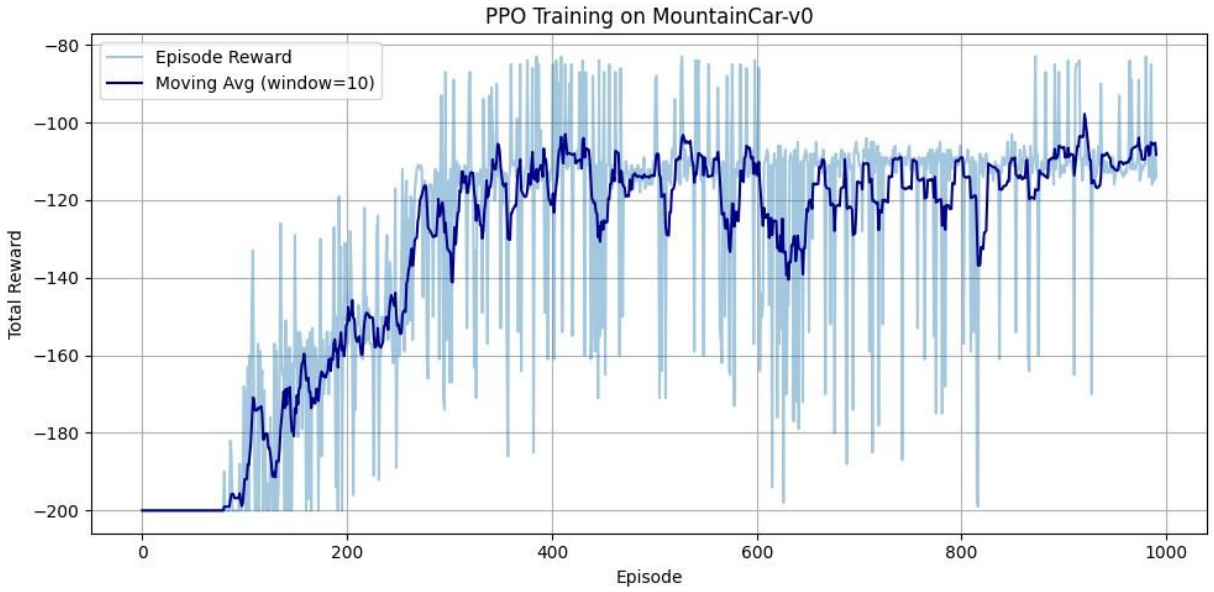
- c_v, c_e είναι βάρη υπερπαραμέτρων που ρυθμίζουν τη συμβολή των δύο επιπλέον όρων.

Συμπεράσματα: Ο αλγόριθμος PPO συνδυάζει την απλότητα στην υλοποίηση με υψηλή απόδοση και σταθερότητα. Με τον περιορισμό μέσω της παραμέτρου ϵ, αποφεύγει μεγάλες αποκλίσεις της νέας πολιτικής από την παλιά, διατηρώντας τη μάθηση σταδιακή και ασφαλή. Η προσθήκη του όρου Value Loss επιτρέπει στο μοντέλο να μαθαίνει καλύτερες εκτιμήσεις της μελλοντικής

ανταμοιβής, ενώ το Entropy Bonus ενισχύει την εξερεύνηση και αποτρέπει την πρόωρη σύγκλιση. Τα παραπάνω επαληθεύονται από το διάγραμμα που παίρνουμε εκτελώντας τον κώδικα με τις παραμέτρους του πίνακα που ακολουθεί.

Μεταβλητή	Τιμή
learning_rate	$5 \cdot 10^{-4}$
gamma	0.99
gae_lambda	0.90
n_steps	2048
batch_size	64
n_epochs	20
clip_range	0.2
seed	42

Πίνακας 2: Υπερπαράμετροι του αλγορίθμου PPO στη συγκεκριμένη υλοποίηση.



Σχήμα 3: PPO Τραινινγ στο MountainCar-v0

Advantage Actor-Critic (A2C)

Ο αλγόριθμος Advantage Actor-Critic (A2C) ανήκει στην οικογένεια actor–critic, όπου συνδυάζονται δύο βασικές συνιστώσες: ο actor, που εκπαιδεύει την πολιτική $\pi_\theta(a|s)$, και ο critic, που εκτιμά τη συνάρτηση αξίας $V_w(s)$ και παρέχει ανατροφοδότηση για την ποιότητα των ενεργειών. Η διάκριση αυτή επιτρέπει πιο σταθερή εκπαίδευση σε σχέση με απλές μεθόδους policy gradient, καθώς μειώνεται η διακύμανση των ενημερώσεων.

Η σωρευτική προεξοφλημένη ανταμοιβή ορίζεται ως:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

όπου γ είναι ο παράγοντας προεξόφλησης και r_{t+k} η άμεση ανταμοιβή.

Με βάση αυτήν, το πλεονέκτημα (advantage) ορίζεται ως:

$$A(s_t,a_t) = R_t - V_w(s_t),$$

το οποίο δείχνει πόσο καλύτερη ήταν μια ενέργεια a_t σε σχέση με την αναμενόμενη αξία της κατάστασης s_t.

Η loss function της πολιτικής είναι:

$$L^{\text{policy}}(\theta) = -\log\pi_\theta(a_t|s_t) \cdot A(s_t,a_t),$$

ενώ του critic για την εκτίμηση της συνάρτησης αξίας είναι:

$$L^{\text{value}}(w) = \left(R_t - V_w(s_t)\right)^2.$$

Για να ενθαρρυνθεί η εξερεύνηση και να αποτραπεί η πρόωρη σύγκλιση, προστίθεται και ένας όρος εντροπίας:

$$L_{\text{entropy}}(\theta) = H[\pi_\theta](s_t).$$

Συνεπώς, το συνολικό αντικειμενικό κριτήριο του A2C είναι:

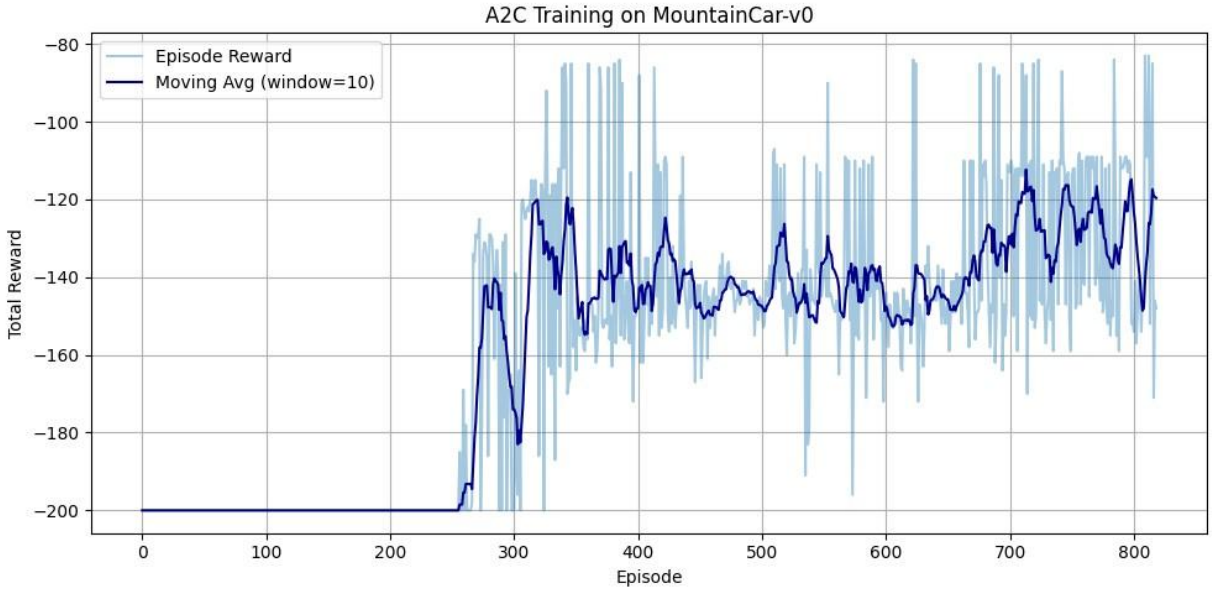
$$L(\theta,w) = L_{\text{policy}}(\theta) + c_v \cdot L_{\text{value}}(w) - c_e \cdot L_{\text{entropy}}(\theta),$$

όπου c_v, c_e είναι βάρη υπερπαραμέτρων.

Εκτελουμε τον αλγοριθμο, με τους παρακάτω υπερπαραμέτρους:

Μεταβλητή	Τιμή
learning_rate	10^{-3}
gamma	0.99
gae_lambda	0.95
n_steps	5
seed	42

Πίνακας 3: Υπερπαράμετροι του A2C στη συγκεκριμένη υλοποίηση.



Σχήμα 4: A2C Training στο MountainCar-v0

Από το παραπάνω γράφημα παρατηρούμε ότι η μέθοδος έχει αρκετά καλά αποτελέσματα, με την απόδοση να σταθεροποιείται μετά το 300 επεισόδιο. Το οποίο επαληθεύει τα αναμενόμενα αποτελέσματα που έχει ο αλγόριθμος A2C που αναφέραμε στην αρχική περιγραφή του αλγορίθμου.