Πανεπιστήμιο Ιωαννίνων

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ: Πληροφορικής και Τηλεπικοινωνιών

ΠΜΣ : Πληροφορική και Δίκτυα

Γιωτάκης Ιωάννης

Διπλωματική εργασία

Η ενισχυτική μάθηση στην επίλυση προβλημάτων βελτιστοποίησης

Επιβλέπων καθηγητής: Γκόγκος Χρήστος

Νοέμβριος 2022

Πρόλογος

Η παρούσα διπλωματική εργασία μελετά την εφαρμογή της ενισχυτικής μάθησης στην βελτιστοποίηση προβλημάτων συνδυαστικότητας.

Συγκεκριμένα πραγματοποιείται μελέτη μεθόδων ενισχυτικής μάθησης και βαθιάς ενισχυτικής μάθησης. Στα πλαίσιά της παρουσιάζονται προσεγγίσεις προβλημάτων όπως η εύρεση βέλτιστης εξόδου σε περιβάλλον λαβυρίνθου, η εύρεση συντομότερης διαδρομής καθώς και μια αποδοτική επίλυση του προβλήματος του περιπλανώμενου πωλητή με τις μεθόδους της ενισχυτικής και βαθιάς ενισχυτικής μάθησης

ΕΥΧΑΡΙΣΤΙΕΣ

Ξεκινώντας θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κ. Γκόγκο Χρήστο για την εμπιστοσύνη, την καθοδήγηση κατά την διάρκεια της διπλωματικής εργασίας αλλά και στο σύνολο της φοίτησης μου.

Επιπλέον θα ήθελα να ευχαριστήσω τους συμφοιτητές μου για την συνεργασία κατά την διάρκεια του προηγούμενου έτους.

Πίνακας περιεχομένων

Κεφάλαιο 1

Εισαγωγή

1.1 Μηχανική Μάθηση

Η Μηχανική μάθηση είναι πεδίο της επιστήμης των υπολογιστών που αναπτύχθηκε από τη μελέτη της αναγνώρισης προτύπων και της υπολογιστικής θεωρίας μάθησης στην τεχνητή νοημοσύνη και διερευνά τη μελέτη και την κατασκευή αλγορίθμων που μπορούν να μαθαίνουν από τα δεδομένα και να κάνουν προβλέψεις σχετικά με αυτά. Τέτοιοι αλγόριθμοι λειτουργούν κατασκευάζοντας μοντέλα από πειραματικά δεδομένα, προκειμένου να κάνουν προβλέψεις βασιζόμενες στα δεδομένα ή να εξάγουν αποφάσεις που εκφράζονται ως το αποτέλεσμα.

Οι εργασίες μηχανικής μάθησης συνήθως ταξινομούνται σε τρεις μεγάλες κατηγορίες, ανάλογα με τη φύση του εκπαιδευτικού «σήματος» ή την «ανατροφοδότηση» που είναι διαθέσιμη σε ένα σύστημα εκμάθησης [1]. Αυτές είναι

- *Επιτηρούμενη μάθηση (αλλιώς επιβλεπόμενη μάθηση ή μάθηση με επίβλεψη) (supervised learning):* Το υπολογιστικό πρόγραμμα δέχεται τις παραδειγματικές εισόδους καθώς και τα επιθυμητά αποτελέσματα από έναν «δάσκαλο», και ο στόχος είναι να μάθει έναν γενικό κανόνα προκειμένου να αντιστοιχίσει τις εισόδους με τα αποτελέσματα.
- Μη επιτηρούμενη μάθηση (αλλιώς μη επιβλεπόμενη μάθηση ή μάθηση χωρίς επίβλεψη (unsupervised learning): Χωρίς να παρέχεται κάποια εμπειρία στον αλγόριθμο μάθησης, πρέπει να βρεί την δομή των δεδομένων εισόδου. Η μη επιτηρούμενη μάθηση μπορεί να είναι αυτοσκοπός (ανακαλύπτοντας κρυμμένα μοτίβα σε δεδομένα) ή μέσο για ένα τέλος (χαρακτηριστικό της μάθησης).
- Ενισχυτική μάθηση: Ένα πρόγραμμα υπολογιστή αλληλοεπιδρά με ένα δυναμικό περιβάλλον στο οποίο πρέπει να επιτευχθεί ένας συγκεκριμένος στόχος χωρίς κάποιος δάσκαλος να του λέει ρητά αν έχει φτάσει κοντά στο στόχο του.

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι η ενισχυτική μάθηση, ένας τομέας της μηχανικής μάθησης που εστιάζει στο πώς μπορεί ένα σύστημα να εκπαιδευτεί μέσα από την άμεση αλληλεπίδραση με το περιβάλλον. Οι αλγόριθμοι ενισχυτικής μάθησης μελετούν τη συμπεριφορά του συστήματος σε ένα περιβάλλον και προσπαθούν να βελτιστοποιούν τη συμπεριφορά του. Η έννοια της ενισχυτικής μάθησης είναι εμπνευσμένη από την μάθηση με επιβράβευση και τιμωρία που συναντάται ως μοντέλο μάθησης των έμβιων όντων.

1.2 Ιστορικά στοιχεία

Η ιστορία της ενισχυτικής μάθησης έχει δύο βασικούς τομείς, που εξελίχθηκαν ανεξάρτητα πριν συνδυαστούν στη σύγχρονη ενισχυτική μάθηση [2]. Ο Πρώτος τομέας αφορά τη μάθηση με δοκιμή και λάθος και είναι υπεύθυνος τις πρώτες εργασίες στην τεχνητή νοημοσύνη. Ο δεύτερος τομέας αφορά το πρόβλημα του βέλτιστου ελέγχου και την επίλυσή του χρησιμοποιώντας συναρτήσεις τιμών και δυναμικό προγραμματισμό.

Η αρχή έγινα στα μέσα της δεκαετίας του 1950 ο Richard Bellman μέσω της επέκτασης, της θεωρίας των Hamilton και Jacobi προσέγγισε το πρόβλημα του σχεδιασμού ενός ελεγκτή για την ελαχιστοποίηση ενός μέτρου στη συμπεριφορά ενός δυναμικού συστήματος με την πάροδο του χρόνου και έθεσε τα θεμέλια της ενισχυτικής μάθησης.

Το 1957 ο Bellman εισήγαγε επίσης τη διακριτή στοχαστική εκδοχή του προβλήματος βέλτιστου ελέγχου που είναι γνωστό ως Markovian διαδικασίες απόφασης (MDPs) και ο Ron Howard (1960) επινόησε τη μέθοδο επαναληπτικής πολιτικής για τα MDP. Όλα αυτά είναι βασικά στοιχεία που διέπουν τη θεωρία και τους αλγόριθμους της σύγχρονης ενισχυτικής μάθησης.

Σημαντικό ρόλο στην διαμόρφωση της ενισχυτικής μάθησης όπως την γνωρίζουμε σήμερα έχει ο ιδέα της μάθησης με δοκιμή και σφάλμα. Ο πρώτος που συνέλαβε την ιδέα της μάθησης δοκιμής και σφάλματος ήταν ο Edward Thorndike. Εκφράζοντας την άποψη ότι η τάση επιλογής των ενέργειων αλλάζει ανάλογα με τα αποτελέσματα (καλά ή κακά) που επιφέρουν.

1.3 Εφαρμογές

Η ενισχυτική μάθηση είναι μια από τις πιο σύγχρονες τεχνολογίες μηχανικής μάθησης αυτό όμως δεν έχει σταθεί εμπόδιο καθώς βρίσκει εφαρμογή σε πληθώρα τομέων της καθημερινότητας. [3]

- Αυτόνομη οδήγηση: Η ενισχυτική μάθηση μπορεί να είναι πολύ αποτελεσματική σε ένα σύνθετο περιβάλλον οδήγησης στο οποίο πρέπει να ληφθούν υπόψη διάφορες παράμετροι, όπως τα όρια ταχύτητας, οι ζώνες οδήγησης, η αποφυγή συγκρούσεων.
- Συναλλαγές και χρηματοδότηση: Μπορεί να αποφασίσει για μια ενέργεια, είτε να κρατήσει, να αγοράσει ή να πουλήσει. Το μοντέλο RL αξιολογεί χρησιμοποιώντας τα υπάρχοντα πρότυπα της αγοράς προκειμένου να διασφαλίσει τη βέλτιστη απόδοση.
- Ενισχυτική μάθηση στο gaming: Χρησιμοποιώντας την ενισχυτική μάθηση το μοντέλο μπορεί να μάθει το παιχνίδι από την αρχή, παίζοντας ενάντια στον εαυτό του.
- Επεξεργασία Φυσικής Γλώσσας: Η ενισχυτική μάθηση μπορεί να χρησιμοποιηθεί στη σύνοψη κειμένου, στην απάντηση ερωτήσεων και στη μετάφραση.
- Στην ρομποτική: Με την χρήση της βαθιάς μάθησης και της ενισχυτικής μάθησης μπορούν να εκπαιδευτούν ρομπότ έτσι ώστε να έχουν την ικανότητα να αλληλοεπιδρούν με διάφορα αντικείμενα του περιβάλλοντος. Ακόμη και αν αυτά δεν υπάρχουν κατά τη διάρκεια της εκπαίδευσης, βρίσκοντας έτσι χρήση στις κατασκευές και σε γραμμές συναρμολόγησης.

Κεφάλαιο 2

2.1 Reinforcement learning

Στην μάθηση με ενίσχυση το σύστημα δεν καθοδηγείται από κάποιον εξωτερικό επιβλέποντα έτσι παρουσιάζει μεγαλύτερη χρονική καθυστέρηση σε σύγκριση με την εποπτευόμενη καθώς το σύστημα εκμάθησης δεν έχει ρητά καλή και κακή συμπεριφορά αλλά πρέπει να ανακαλύψει μόνο του ποιες ενέργειες θα πραγματοποιηθούν [4]. Αυτό την καθιστά πολύ πιο κατάλληλη για προβλήματα στα οποία είναι δύσκολο να περιγράφει επαρκώς η βέλτιστη συμπεριφορά.

Για την εμβάθυνση στην υλοποίηση των αλγόριθμων reinforcement learning θα παρατεθούν οι βασικοί ορισμοί.

Agent: Σύστημα μάθησης, είναι υπεύθυνο λήψης αποφάσεων, αλληλοεπιδρά με το περιβάλλον και επιλεγεί την επόμενη ενέργεια.

State: Η κατάσταση που λαμβάνει το σύστημα εκπαίδευσης για την αλληλεπίδραση με το περιβάλλον.

Reward: Η ανταμοιβή/ποινή που λαμβάνει το σύστημα μάθησής για την ενέργεια που πραγματοποίησε.

Policy: Υποδεικνύει τη συμπεριφορά του συστήματος εκμάθησης. Είναι μια συνάρτηση που αντιστοιχίζει την ενέργεια στην κατάσταση (state-action).

Return: Είναι οι εκτιμώμενες σωρευτικές ανταμοιβές για ένα δεδομένο χρονικό βήμα t με γ την έκπτωση για τις μελλοντικές ανταμοιβές

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma 2 r_{t+3} + \cdots + \gamma m r_{t+m+1},\ \gamma \in [0,1]$$

Λειτουργία: Ένας agent θα λάβει την κατάσταση του συστήματος και μια ανταμοιβή που σχετίζεται με την τελευταία κατάσταση μετάβασης. Στη συνέχεια υπολογίζει-επιλέγει μια ενέργεια που επιστρέφεται στο σύστημα. Ως απόκριση, το σύστημα αλλάζει σε μια νέα κατάσταση και επαναλαμβάνει τη διαδικασία .

2.2 Markov Decision Process

Ακρογωνιαίος λίθος της ενισχυτικής μάθησης είναι η διαδικασία απόφασης Markov, MDP. Για το σκοπό αυτό ορίζεται μια συνάρτηση, η οποία αντιστοιχεί στην εκτέλεση της ενέργειας και στη βέλτιστη συνέχιση σύμφωνα με οποιαδήποτε πολιτική. [7]

Βασίζεται στην υπόθεση ότι η παρούσα κατάσταση, S_t, είναι αρκετή για να προβλέψει το μέλλον για ένα σύνολο καταστάσεων.

$$P\,[S_{t+1}|S_t, S_{t-1}, \ldots, S_1] = P\,[S_{t+1}|S_t]$$

Ένα MDP αποτελείται από:

- S = {s1, s2, . . . , sn} οι καταστάσεις του περιβάλλοντος.
- A = {α1,α2,...,αn} οι ενέργειες που μπορεί να εκτελέσει το σύστημα εκμάθησης οποιαδήποτε στιγμή.
- P = {s,α,s′} είναι η πιθανότητα μετάβασης στην κατάσταση s′ όταν εκτελούμε ενέργεια α από την κατάσταση s, έτσι ώστε να ικανοποιούν την ιδιότητα Markov
- R = είναι η συνάρτηση ανταμοιβής της διαδικασίας, μπορεί να είναι η στιγμιαία ή η αναμενόμενη στιγμιαία ανταμοιβή (για στοχαστικές ανταμοιβές) σε κάθε χρονικό βήμα.
- $Rs\alpha = E[R_{t+1}\, S_t = s_t, A_t = \alpha]$

Για S, A, R και χρονικές καταστάσεις t=1,2,3...

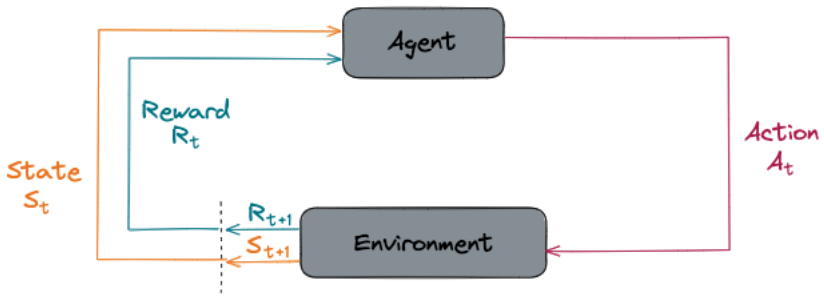
1 Τη χρονική στιγμή t το περιβάλλον είναι στην κατάσταση S_t.

2 Agent με βάση την τωρινή κατάσταση S_t επιλέγει μια ενέργεια A_t.

3 Η μετάβαση στην κατάσταση S_{t+1} έχει μια ανταμοιβή R_{t+1}

4 ίδια διαδικασία επαναλαμβάνεται για την κατάσταση περιβάλλοντος S_{t+1}

Στην παρακάτω εικόνα 1.1 αναπαρίσταται η τυπική λειτουργία agent-state-reward



Εικόνα 1 agent-state-reward [1]

Ο στόχος σε μια διαδικασία απόφασης Markov είναι να βρεθεί μια πολιτική π, έτσι ώστε να καθορίζει την ενέργεια που θα επιλέξει το σύστημα στην κατάσταση S. Μόλις μια διαδικασία απόφασης Markov συνδυαστεί με μια πολιτική τότε διορθώνει την ενέργεια για κάθε κατάσταση.

Σκοπός: Το συστήματα μάθησης να επιλέξει μια πολιτική π, έτσι ώστε να μεγιστοποιήσει (ή ελαχιστοποιήσει) τη συνάρτηση ανταμοιβής.

Value Functions: Η συνάρτηση εκτιμάει για το σύστημα εκμάθησης τον αντίκτυπο του να βρίσκεται σε μια δεδομένη κατάσταση ή στην εκτέλεση μιας δεδομένης ενέργειας σε μια δεδομένη κατάσταση. Τυπικά, η αξία της ενέργειας α δεδομένης της κατάσταση s υπό μια πολιτική π είναι η αναμενόμενη επιβράβευση G από την έναρξη έως την συγκεκριμένη κατάσταση s τη χρονική στιγμή t και τη συνέχεια της πολιτικής π με βάση την ενέργεια α που πραγματοποιήθηκε.

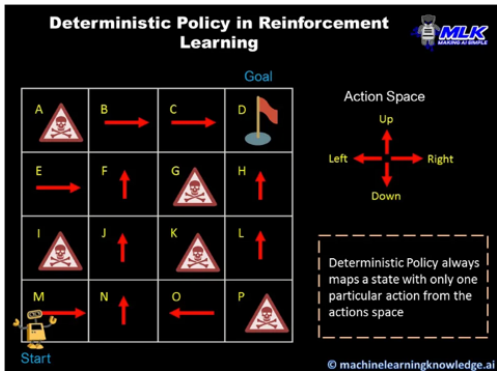
$$q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = \alpha]$$
$$= E_\pi[\sum_{k=0}^{\infty} \gamma^\kappa R_{t+k+1} | S_t = s, A_t = \alpha]$$

Όσον αφορά την απόδοση, μια πολιτική θεωρείται ότι είναι η καλύτερη, εάν η αναμενόμενη ανταμοιβή της είναι μεγαλύτερη από την αναμενόμενη ανταμοιβή των υπολοίπων πολιτικών. Μια τέτοια πολιτική ονομάζεται Optimal Policy π΄.

Η βέλτιστη πολιτική καθορίζεται από την βέλτιστη Value function, η οποία αναφέρεται και ως Q-function, q* και ορίζεται ως:

$$q_*(s,a) = max_\pi q_\pi(s,a)$$

Στην *Εικόνα 2* φαίνεται η βέλτιστη πολιτική για οποιαδήποτε κατάσταση, ο Agent-ρομπότ αποφύγει τις νάρκες και φτάνει στον προορισμό



Εικόνα 2: Βέλτιστη πολιτική [2]

2.3 Bellman Optimality Equation

Το σύστημα μάθησης προσπαθεί από οποιαδήποτε κατάσταση να λάβει την υψηλότερη αναμενόμενη ανταμοιβή. Για τον λόγο αυτό πρέπει να επιτευχθεί η συνάρτηση βέλτιστης τιμής, q*, δηλαδή το μέγιστο άθροισμα των ανταμοιβών [5].

Έτσι λοιπόν για οποιοδήποτε ζεύγος κατάστασης-ενέργειας, η αναμενόμενη επιβράβευση από την κατάσταση s επιλέγοντας μια ενέργεια a και ακολουθώντας την πολιτική π, θα είναι η αναμενόμενη ανταμοιβή που θα λάβουμε από τη λήψη της ενέργειας a στην κατάσταση s (Rt+1) συν τη μέγιστη αναμενόμενη τιμή που μπορεί να επιτευχθεί από οποιοδήποτε πιθανό επόμενο ζεύγος κατάστασης-ενέργειας(s',a') και ονομάζονται Q-values.

$$q_*(s,a) = E\left[R_{t+1} + \gamma max_{a'} q_*(s',a')\right]$$

Οι τιμές που επιστρέφει η Bellman Equation για κάθε ζεύγος s-a, αποθηκεύονται σε έναν πίνακα, Q-table. Για να αποφευχθεί η εξολοκλήρου αντικατάσταση της προηγούμενης τιμής του πίνακα με την καινούργια εισάχθηκε ο ρυθμός εκμάθησης lr δηλαδή πόσο γρήγορα το σύστημα εγκαταλείπει την προηγούμενη τιμή και ορίζεται ως:

$$q^{new}(s,a) = (1-lr)q(s,a) + lr(R_{t+1} + \gamma max_{a'} q(s',a'))$$

2.4 Epsilon Greedy Strategy

Το σύστημα εκμάθησης κατά την έναρξη της διαδικασίας εκμάθησης δεν έχει καμία αλληλεπίδραση με το περιβάλλον και καμία εικόνα για το ποιες ενέργειες πρέπει να πραγματοποιήσει [5].

Ακολουθώντας την Epsilon Greedy Strategy έρχεται σε αλληλεπίδραση με το περιβάλλον. Αρχικά για κάθε επεισόδιο πραγματοποιεί τυχαίες ενέργειες έως ότου φτάσει σε κάποια τελική κατάσταση είτε ολοκληρώσει κάποιον πεπερασμένο αριθμό βημάτων. Σταδιακά με την πάροδο των επεισοδίων οι τυχαίες ενέργειες μειώνονται (exploration) καθώς το σύστημα εκμάθησης αποκτά εικόνα για το περιβάλλον και οι ενέργειες που επιλέγονται συμφώνα με την τεχνική εκμετάλλευση αυξάνονται (exploitation).

2.5 Q-learning

Το Q-learning είναι ένας off-policy αλγόριθμος ελέγχου που ενημερώνει έναν πίνακα τιμών Q για κάθε ζεύγος κατάστασης-ενέργειας [4]. Μια off-policy μέθοδος εκμάθησης βελτιώνει μια πολιτική που είναι διαφορετική από την πολιτική που χρησιμοποιείται για την επιλογή ενεργειών, αυτό σημαίνει ότι το σύστημα εκμάθησης μπορεί να προχωρήσει στη συνεχή εξερεύνηση ενώ μαθαίνει τη βέλτιστη πολιτική.

Βήματα αλγόριθμου Q-learning :

1. Δημιουργία ενός Q-table και αρχικοποίηση όλων των τιμών του στο 0 (Q-values)
2. Ξεκίνησε από μια αρχική κατάσταση (state)
3. Για κάθε βήμα ενός επεισοδίου:
    a) Επέλεξε μια ενέργεια με βάση το state (exploitation) ή μια τυχαία ενέργεια (exploration).
    b) Πάρε την νέα κατάσταση(state_next) και την επιβράβευση στην οποία οδήγησε η ενέργεια.
    c) Ενημέρωσε τις τιμές του Q-table σύμφωνα με την εξίσωση bellman: $Q(S,A) \leftarrow Q(S,A) + \alpha\,[R + \gamma\, max_a\, Q(S',a) - Q(S,A)]$
    d) Κάνε το state=state_next και επανέλαβε την διαδικασία.
4. Η εκπαίδευση σταματάει με την ολοκλήρωση των επεισοδίων.

Σημαντικό ρόλο στην λειτουργία του αλγορίθμου, όπως θα δούμε και στις δοκιμές που πραγματοποιήθηκαν, έχουν οι μεταβλητές: ποσοστό εκμάθησης (α), συντελεστής έκπτωσης (γ) , και $max(a)$, που είναι η μέγιστη ανταμοιβή που μπορεί να λάβει ο agent μετά τη λήψη της βέλτιστης ενέργειας στην τρέχουσα κατάσταση. Ως ρυθμός εκμάθησης(α) ορίζεται ένας αριθμός μεταξύ 0 και 1, όπου το μηδέν σημαίνει ότι οι τιμές Q δεν ενημερώνονται ποτέ και δεν μαθαίνει τίποτα. Αντίθετα, το να είναι κοντά

στο ένα σημαίνει ότι το σύστημα μπορεί να μάθει γρήγορα, ωστόσο αυτό μπορεί να οδηγήσει σε ασταθή εκπαίδευση. Ο συντελεστής έκπτωσης (γ) είναι επίσης μια τιμή μεταξύ 0 και 1, όπου μηδέν σημαίνει ότι το σύστημα χρησιμοποιεί μόνο τις ανταμοιβές του εγγύς μέλλοντος, και όταν είναι κοντά στο ένα, χρησιμοποιεί τις μακροπρόθεσμες ανταμοιβές

Κεφάλαιο 3

3.1 Deep Reinforcement learning

Για την εμβάθυνση στην βαθιά ενισχυτική μάθηση θα χρειαστεί να αποδοθούν κάποιοι περαιτέρω ορισμοί και τρόποι λειτουργίας.

Τεχνητό Νευρωνικό δίκτυο

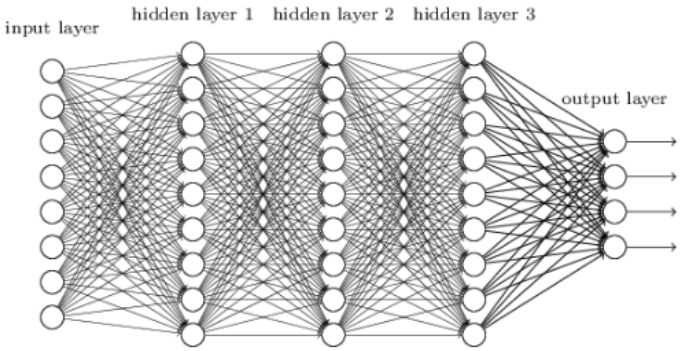
Νευρωνικό δίκτυο ονομάζεται ένα κύκλωμα διασυνδεδεμένων νευρώνων [8]. Στην περίπτωση βιολογικών νευρώνων, πρόκειται για ένα τμήμα νευρικού ιστού. Στην περίπτωση τεχνητών νευρώνων, πρόκειται για ένα αφηρημένο αλγοριθμικό μοντέλο το οποίο εμπίπτει στον τομέα της υπολογιστικής νοημοσύνης. Είναι ένα δίκτυο από απλούς υπολογιστικούς κόμβους (νευρώνες), διασυνδεδεμένους μεταξύ τους. Είναι εμπνευσμένο από το Κεντρικό Νευρικό Σύστημα, το οποίο προσπαθεί να προσομοιώσει. Οι νευρώνες είναι τα δομικά στοιχεία του δικτύου. Κάθε τέτοιος κόμβος δέχεται ένα σύνολο αριθμητικών εισόδων από διαφορετικές πηγές (είτε από άλλους νευρώνες, είτε από το περιβάλλον), επιτελεί έναν υπολογισμό με βάση αυτές τις εισόδους και παράγει μία έξοδο. Η εν λόγω έξοδος είτε κατευθύνεται στο περιβάλλον, είτε τροφοδοτείται ως είσοδος σε άλλους νευρώνες του δικτύου.

Αρχιτεκτονική ενός τεχνητού νευρωνικού δικτύου.

Ένα τεχνητό νευρωνικό δίκτυο, όπως φαίνεται στην Εικόνα 3 είναι ένα σύνολο συνδεδεμένων νευρώνων σε επίπεδα:

- **Input Layer**: Εισάγει τα αρχικά δεδομένα για περαιτέρω επεξεργασία στο σύστημα μέσω των επόμενων επίπεδων.
- **Hidden Layer**: Ένα επίπεδο σταθμισμένων εισόδων από τεχνητούς νευρώνες για την παραγωγή της εξόδου μέσω της συνάρτησης ενεργοποίησης μεταξύ των επιπέδων εισόδου και των επιπέδων εξόδου.
- **Output Layer:** Το τελευταίο στρώμα νευρώνων που παράγει τις εξόδους του δίκτυο

Ένα βαθύ νευρωνικό δίκτυο (DNN) αποτελείται από ένα επίπεδο εισόδου x, ένα ή περισσότερα κρυφά επίπεδα h1, h2,…,hn και ένα επίπεδο εξόδου y.



Εικόνα 3: Νευρωνικό δίκτυο [3]

Κάθε επίπεδο k μετατρέπει τη δραστηριότητα του προηγούμενου επιπέδου (διάνυσμα h_{k-1}) σε ένα καινούργιο διάνυσμα h_k πολλαπλασιάζοντάς το με έναν πίνακα βαρών W_k, προσθέτοντας ένα διάνυσμα πόλωσης b_k (bias) και εφαρμόζοντας μια συνάρτηση ενεργοποίησης f .

$$\mathbf{h_k}=f(W_k \times \mathbf{h_{k-1}}+\mathbf{b_k})$$

Συναρτήσεις ενεργοποίησης

Η συνάρτηση ενεργοποίησης επαναπροσδιορίζει την έξοδο του κόμβου δεδομένης της εισόδου ή του συνόλου των εισόδων [10]

Sigmoid: Είναι μια συνάρτηση με τυπική καμπύλη "S". Η σιγμοειδής συνάρτηση δημιουργεί ένα σύνολο εξόδων πιθανότητας μεταξύ 0 και 1.

$$Sigmoid(x) = \frac{1}{1+e^{(-x)}} = \frac{e^x}{1+e^x}$$

ReLU: Όταν η είσοδος είναι κάτω από το 0 η διορθωμένη έξοδος είναι 0 αλλιώς παραμένει η τιμή της εισόδου. Αυτό σημαίνει ότι εάν η είσοδος είναι μεγαλύτερη από 0, το αποτέλεσμα ισούται με την είσοδο.

$$ReLU(x) = \max(0, x)$$

Linear: Η είσοδος παραμένει ως έχει ή αλλάζει σύμφωνα με οποιαδήποτε γραμμική συνάρτηση.

$$\phi(x) = x$$

Συνάρτηση απώλειας

Είναι μια συνάρτηση που συγκρίνει τον στόχο και τις προβλεπόμενες τιμές εξόδου. Κατά την εκπαίδευση, στοχεύει να ελαχιστοποιήσει την απώλεια μεταξύ των προβλεπόμενων και των στοχευόμενων αποτελεσμάτων με έτσι ώστε να βρεθούν τα κατάλληλα βάρη και πολώσεις θ. Στα προβλήματα παλινδρόμησης, ελαχιστοποιείται το μέσο τετραγωνικό σφάλμα (mse):

$$\mathcal{L}(\theta) = \mathrm{E}_{x,t\in D}[||t - y||^2]$$

Με x την είσοδο, t την έξοδο (που ορίζεται στο σετ εκπαίδευσης) και y την πρόβλεψη του NN για την είσοδο x. Όσο πιο κοντά είναι η πρόβλεψη στην πραγματική τιμή, τόσο μικρότερο είναι το mse.

Αφού οριστεί η συνάρτηση απώλειας, πρέπει να ελαχιστοποιηθεί αναζητώντας τις βέλτιστες τιμές για τις παραμέτρους θ. Αυτή η διαδικασία βελτιστοποίησης βασίζεται στην κλίση κατάβασης (gradient descent), η οποία είναι μια επαναληπτική διαδικασία που τροποποιεί τις εκτιμήσεις των παραμέτρων στην αντίθετη κατεύθυνση της κλίσης της συνάρτησης απώλειας:

$$\Delta\theta = -\eta\nabla_\theta\mathcal{L}(\theta) = -\eta\frac{d\mathcal{L}(\theta)}{d\theta}$$

Backpropagation

Η οπίσθια διάδοση είναι η εφαρμογή του κανόνα της αλυσίδας

$$\frac{dz}{dx} = \frac{dz}{dy}\cdot\frac{dy}{dx}$$

στα νευρωνικά δίκτυα:

$$\frac{\partial \mathcal{L}(\theta)}{\partial W_k} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{y}} \times \frac{\partial \mathbf{y}}{\partial \mathbf{h_n}} \times \frac{\partial \mathbf{h_n}}{\partial \mathbf{h_{n-1}}} \times \ldots \times \frac{\partial \mathbf{h_k}}{\partial W_k}$$

Η μέθοδος χρησιμοποιείται στον προσδιορισμό της κλίσης (Gradient) για τον υπολογισμό των βαρών του δικτύου, καθώς ένα σφάλμα εξόδου υπολογίζεται και κατανέμεται στα πίσω επίπεδα του δικτύου. Χρησιμοποιείται για εκπαίδευση σε βαθιά νευρωνικά δίκτυα.

Αυτή η προς τα πίσω ροή των πληροφοριών σφάλματος επιτρέπει τον αποτελεσματικό προσδιορισμό της κλίσης (gradient) σε κάθε επίπεδο.

Αλγόριθμοι βελτιστοποίησης κλίσης (Gradient Descent Optimization algorithms)

Το gradient descent είναι ο προτιμώμενος τρόπος για τη βελτιστοποίηση των νευρωνικών δικτύων και πολλών άλλων αλγορίθμων μηχανικής μάθησης.

RMSProp

Είναι βελτιστοποιητής που έχει σχεδιαστεί για εκπαίδευση νευρωνικών δικτυών και βρίσκεται στη σφαίρα των μεθόδων προσαρμοστικού ρυθμού μάθησης [9], οι οποίες έχουν αυξηθεί σε δημοτικότητα τα τελευταία χρόνια, είναι η επέκταση του αλγορίθμου Stochastic Gradient Descent (SGD) και της μεθόδου momentum και βάση του αλγορίθμου Adam.

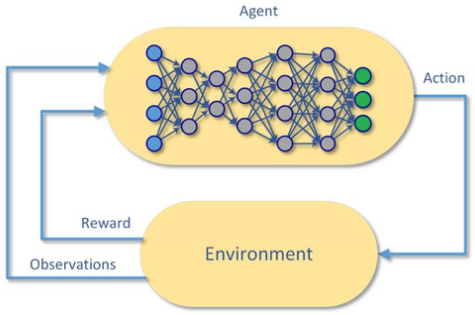
Adam

Είναι ένας αλγόριθμος βελτιστοποίησης που μπορεί να αποτελέσει εναλλακτική για τη διαδικασία της στοχαστικής κλίσης καθόδου. [10]. Υπολογίζεί τον εκθετικό κινητό μέσο όρο των κλίσεων και των τετραγωνικών κλίσεων συνδυάζοντας έτσι την μέθοδο καθόδου κλίσης(gradient descent methods) και την RMSP.

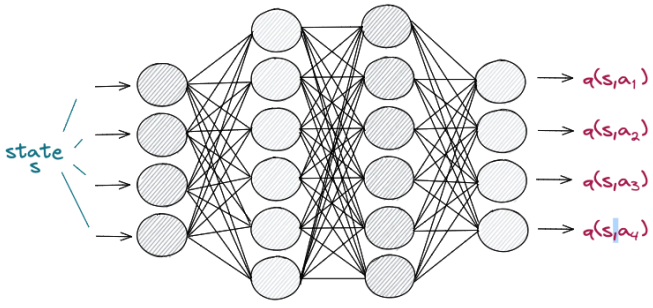
3.2 Deep Q-learning

Η προσέγγιση που συνδυάζει το Q-learning με ένα βαθύ νευρωνικό δίκτυο ονομάζεται deep Q-learning, κατά την οποία το νευρωνικό δίκτυο προσεγγίζει τη βέλτιστη συνάρτηση Q, υπολογίζοντας τις Q-values για κάθε ζεύγος action-state σε ένα δεδομένο περιβάλλον [11].

Το αρχικό βαθύ νευρωνικό δίκτυο με τυχαία βάρη, δέχεται καταστάσεις από ένα δεδομένο περιβάλλον ως είσοδο. Για κάθε κατάσταση εισόδου, το δίκτυο εξάγει εκτιμώμενες τιμές Q για κάθε ενέργεια που μπορεί να ληφθεί από αυτήν την κατάσταση. Ο στόχος αυτού του δικτύου είναι να προσεγγίσει τη βέλτιστη συνάρτηση Q που ικανοποιεί την εξίσωση Bellman. Η απώλεια από το δίκτυο υπολογίζεται συγκρίνοντας τις εξαγόμενες τιμές του νευρωνικού δικτύου με τις τιμές-στόχους της εξίσωσης Bellman έτσι ώστε να ελαχιστοποιηθεί η απώλεια.



Εικόνα 4: Ο agent έχει αντικατασταθεί από το νευρωνικό δίκτυο. [3]



Εικόνα 5: Το νευρωνικό δίκτυο παίρνει ως είσοδο μια κατάσταση-διάνυσμα και επιστρέφει μια πρόβλεψη- διάνυσμα για κάθε πιθανή ενέργεια. [4]

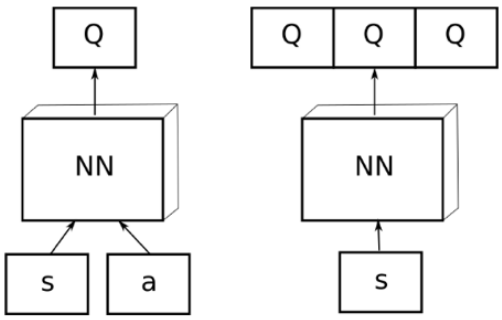
Η διαδικασία εκπαίδευσης είναι παρόμοια με την επιβλεπόμενη μάθηση. Κάθε επίπεδο μετατρέπει τη δραστηριότητα του προηγούμενου επιπέδου σε ένα άλλο διάνυσμα

πολλαπλασιάζοντάς το με τα βάρη του, προσθέτοντας την πόλωση και εφαρμόζοντας μια συνάρτηση ενεργοποίησης.

Μετά τον υπολογισμό της απώλειας, τα βάρη του δικτύου ενημερώνονται. Αυτή η διαδικασία επαναλαμβάνεται για κάθε κατάσταση του περιβάλλοντος μέχρι να ελαχιστοποιηθεί επαρκώς η απώλεια και να ληφθεί μια κατά προσέγγιση βέλτιστη συνάρτηση Q.

3.3 Deep Q-Network

Η χρήση ενός νευρωνικού δικτύου για την εκτίμηση των Q-Values είναι γνωστή ως DQN, ανήκει στις Value-based μεθόδους, δηλαδή στις μεθόδους που βασίζονται στις τιμές κάθε πιθανού ζεύγους state-action για να προσεγγίσουν τη βέλτιστη συνάρτηση Q. Το δίκτυο μπορεί είτε να λάβει ένα ζεύγος state-action ως είσοδο και να επιστρέψει μια ενιαία τιμή εξόδου, είτε να λάβει μόνο το state ως είσοδο και να επιστρέψει την τιμή Q όλων των πιθανών ενεργειών (εάν ο χώρος ενεργειών είναι διακριτός), και στις δύο περιπτώσεις, ο στόχος είναι οι εκτιμήσεις του δικτύου, $Q_\theta(s,a)$, με θ τις παραμέτρους του νευρωνικού δικτύου [12].



Εικόνα 6: DQN ζεύγος κατάστασης-ενέργειας (s,a) με μια τιμή εξόδου Q (αριστερά), DQN κατάστασης s με τις τιμές Q όλων των ενεργειών που είναι δυνατές σε αυτήν την κατάσταση (δεξιά).

Στην παρούσα εργασία χρησιμοποιήθηκε η δεύτερη προσέγγιση, όπου το νευρωνικό παίρνει σαν είσοδο μια κατάσταση και επιστρέφει μια τιμή-πρόβλεψη για κάθε πιθανή ενέργεια.

Value network: Κύριο δίκτυο, οι προβλέψεις του καθορίζουν την επιλογή της καταλληλότερης ενέργειας-πολιτικής.

$$y=r(s,a,s')+\gamma \max_{a'} Q_\theta(s',a')$$

Target network: Δευτερεύον δίκτυο, οι προβλεπόμενες τιμές Q, χρησιμοποιούνται για να εκπαιδεύσουν το κύριο δίκτυο Q. Οι παράμετροι του target network δεν εκπαιδεύονται, αλλά συγχρονίζονται περιοδικά με τις παραμέτρους του κύριου δικτύου Q (Value network). Η ιδέα είναι ότι η χρήση των τιμών Q του target network για την εκπαίδευση του Value network θα βελτιώσει τη σταθερότητα της εκπαίδευσης και συμβολίζεται ως $Q_{\theta*}$ με παραμέτρους θ* τις παραμέτρους του target Network.

Υπολογισμός target : $Y_i = r_i + \gamma max Q\theta^*(s_i', a)$

Το δίκτυο εκπαιδεύεται έτσι ώστε ελαχιστοποιηθεί η απώλεια :

$$L(\theta) = \mathbb{E}_\pi[(rt + \gamma max_{a'} Q_\theta(s',a') - Q_\theta(s,a))^2]$$

Στην απλή περίπτωση δικτύου DQN τα δύο δίκτυα ταυτίζονται και έτσι δεν χρησιμοποιείται το Target network

Κατά την υλοποίηση ενός DQN οι μεταβάσεις αποθηκεύονται σε ένα buffer που ονομάζεται μνήμη επαναληπτικής εμπειρίας (experience replay memory). Όταν το buffer είναι γεμάτο, νέες μεταβάσεις αντικαθιστούν τις παλιές. Για την εκπαίδευση του δικτύου χρησιμοποιείται ένα mini batch συνήθως μεγέθους 32, το οποίο δειγματίζει τυχαίες εγγραφές από την experience replay memory.

Αλγόριθμος Deep Q-Network (DQN)

1. Αρχικοποίηση αρχικού νευρωνικού δικτύου με τυχαία βάρη
2. Δημιουργία experience replay memory D μεγέθους N
3. Αποθήκευση αρχικής κατάστασης S_0
4. Για κάθε επεισόδιο e ∈ [0, E_{end}]
    a. Επίλεξε μια ενέργεια a_t με βάση την πολιτική συμπεριφοράς του δικτύου, $Q_\theta(s_t,a)$
    b. Εφάρμοσε την ενέργεια a_t και πάρε την κατάσταση S_t στην οποία οδηγεί η ενέργεια και την επιβράβευση R_t
    c. Αποθήκευσε $(s_t,a_t,r_{t+1},s_{t+1})$ στην experience replay memory.
    d. Για κάθε βήμα t του επεισοδίου

i. Πάρε ένα τυχαίο δείγμα μιας *mini batch Ds* από το *D*.
ii. Για κάθε μετάβαση (s,a,r,s΄) της *mini batch*
    1. Προέβλεψε τις Q-values των ενεργειών της επομένης κατάστασης βάση του $\max_{a'}Q_\theta(s',a')$ του δικτύου
    2. Υπολόγισε την target Value $y=r+\gamma\max_{a'}Q_\theta(s',a')$.
iii. Εκπαίδευσε το δίκτυο Q_θ στο Ds έτσι ώστε να ελαχιστοποιεί την $Loss(\theta)=\mathbb{E}Ds[(y-Q_\theta(s,a))^2]$

3.4 Double Deep Q-Network

Είναι μια τροποποίηση του DQN η οποία χρησιμοποιεί 2 ανεξάρτητα νευρωνικά δίκτυα (Value Network και Target Network) [13], σε μια προσπάθεια να λυθεί το πρόβλημα της υπερεκτίμησης των τιμών από το κύριο νευρωνικό δίκτυο που συναντάται συχνά κατά την αρχή της εκμάθησης όταν οι τιμές Q απέχουν πολύ από το να είναι σωστές. (το $Q_{\theta'}(s',a)$ είναι υψηλότερο από την πραγματική Q τιμή).

Λειτουργία: Χρήση δυο ανεξάρτητων δίκτυών, το Target network θα χρησιμοποιηθεί για την εύρεση της ενέργειας (η ενέργεια με τη μέγιστη τιμή Q), το Value network για την εκτίμηση της τιμής Q της συγκεκριμένης ενέργειας.

Εφαρμογή: Το εκπαιδευόμενο δίκτυο θ (Value network) χρησιμοποιείται για την επιλογή της ενέργειας $a^*=\text{argmax}_{a'}Q_\theta(s',a')$, ενώ το Target network θ* εκτιμά μόνο την τιμή Q. Η τιμή στόχος γίνεται: $y=r(s,a,s')+\gamma Q_{\theta'}(s',\text{argmax}_{a'}Q_\theta(s',a'))$

Αλγοριθμικά υπάρχει μια τροποποίηση σε σχέση με τον DQN:

Αλγόριθμος Double Deep Q-Network (DDQN)

- Για κάθε βήμα t του επεισοδίου
    - Πάρε ένα τυχαίο δείγμα μιας mini batch Ds από το D.
    - Για κάθε μετάβαση (s,a,r,s΄) της mini batch
        - Επίλεξε μια ενέργεια $a*=\text{argmax}a'Q_\theta(s',a^*)$ με βάση το value network
        - Προέβλεψε τις Q-values των ενεργειών της επομένης κατάστασης βάση του $\max_{a'}Q_{\theta'}(s',a')$ του target network

- Υπολόγισε την target Value $y=r+\gamma Q\theta'(s',a*)$.

3.5 Prioritized Experience Replay

Στο DQN η δειγματοληψία από την experience replay memory γίνεται ομοιόμορφα. Οι νέες και καλύτερα προβλεπόμενες μεταβάσεις επιλέγονται με την ίδια πιθανότητα με τις παλιές μεταβάσεις, κάτι που επιβραδύνει τη μάθηση [14]. Η κύρια ιδέα της Prioritized Experience Replay είναι να ταξινομηθούν οι μεταβάσεις στη μνήμη επανάληψης με φθίνουσα σειρά του σφάλματος TD:

$$\delta = r(s, a, s') + \gamma\, Q_{\theta'}(s', \operatorname{argmax}_{a'} Q_\theta(s', a')) - Q_\theta(s, a)$$

και να επιλέγονται με μεγαλύτερη συχνότητα.

3.6 Duelling Deep Q-Network

Η κλασική αρχιτεκτονική DQN χρησιμοποιεί ένα νευρωνικό δίκτυο για να προβλέψει άμεσα την τιμή όλων των πιθανών ενεργειών Qθ(s,a). Η αξία μιας ενέργειας εξαρτάται από δύο παράγοντες [15].

- Την αξία της κατάστασης s: Σε κάποιες καταστάσεις οποιαδήποτε ενέργεια δεν είναι επιθυμητή.
- Την επιλογή της ενέργειας: Σε μια δεδομένη κατάσταση κάποιες ενέργειες είναι καλύτερες

Αυτό οδηγεί στον ορισμό του πλεονεκτήματος μιας ενέργειας Aπ(s,a).

$$Aπ(s,a)=Qπ(s,a)−Vπ(s)$$

Ένα μηδενικό πλεονέκτημα βέλτιστης ενέργειας στην κατάσταση s σημαίνει πως: Η αναμενόμενη επιβράβευση της κατάσταση s (Vπ(s)) είναι ίδια με την αναμενόμενη επιβράβευση όταν από την s επιλέγεται η βέλτιστη ενέργεια a (Qπ(s,a)).
Αυτό σημαίνει ότι το πλεονέκτημα όλων των άλλων ενεργειών είναι αρνητικό καθώς φέρνουν μικρότερη επιβράβευση από τη βέλτιστη ενέργεια, επομένως είναι λιγότερο συμφέρουσες. Απαραίτητη προϋπόθεση για την σωστή λειτουργία είναι η σωστή εκτίμηση της Vπ(s) τιμής.

Η συνάρτηση Aπ(s,a) έχει μικρότερη διακύμανση από τις τιμές Q, κάτι που είναι μια πολύ καλή ιδιότητα όταν χρησιμοποιούνται νευρωνικά δίκτυα. Η διακύμανση των

τιμών Q προέρχεται από το γεγονός ότι προβλέπονται με βάση άλλες εκτιμήσεις, οι οποίες εξελίσσονται κατά τη διάρκεια της εκμάθησης (μη σταθερότητα των στόχων) και μπορούν να αλλάξουν δραστικά κατά την εξερεύνηση (στοχαστικές πολιτικές).
Έτσι η Aπ(s,a) παρακολουθεί μόνο τη αλλαγή της αξίας μιας ενέργειας σε σύγκριση με την κατάστασή της, κάτι που θα είναι πολύ πιο σταθερό με την πάροδο του χρόνου.

Κεφάλαιο 4

Περιγραφή προβλημάτων που μελετήθηκαν και των δομών, βιβλιοθηκών που χρησιμοποιήθηκαν.

4.1 Δομές και προβλήματα που μελετήθηκαν

Γράφημα

Ένα γράφημα είναι ένα διατεταγμένο ζεύγος G = (V, E), αποτελούμενο από ένα σύνολο V κόμβων και ένα σύνολο E ακμών, οι οποίες είναι υποσύνολα δύο στοιχείων V (δηλαδή, μια ακμή σχετίζεται με δύο κορυφές και η σχέση απεικονίζεται ως μη ταξινομημένο ζεύγος των κορυφών σε σχέση με τη συγκεκριμένη ακμή.

Πρόβλημα συντομότερης διαδρομής.

Στη θεωρία γραφημάτων, το πρόβλημα της συντομότερης διαδρομής είναι το πρόβλημα της εύρεσης μιας διαδρομής μεταξύ δύο κορυφών (ή κόμβων) σε ένα γράφημα έτσι ώστε το άθροισμα των βαρών των ακμών που το αποτελούν να ελαχιστοποιείται.

Το πρόβλημα της εύρεσης της συντομότερης διαδρομής μεταξύ δύο σημείων σε έναν οδικό χάρτη μπορεί να μοντελοποιηθεί ως ειδική περίπτωση του προβλήματος της συντομότερης διαδρομής σε γραφήματα, όπου οι κορυφές αντιστοιχούν σε σημεία και οι ακμές αντιστοιχούν σε τμήματα δρόμου, το καθένα σταθμισμένο με το μήκος του τμήματος.

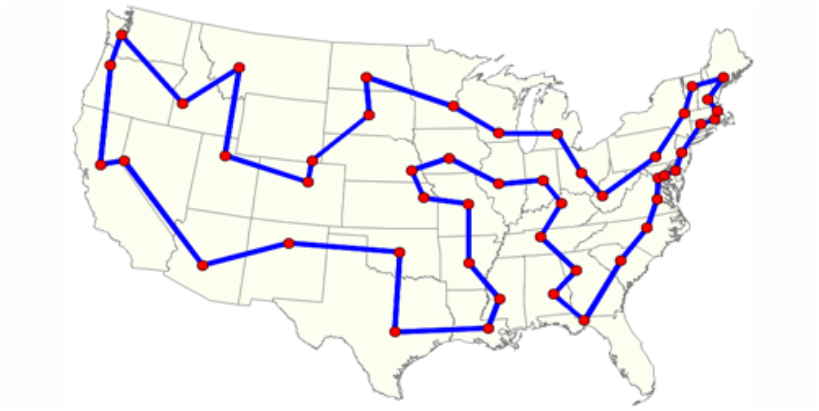
Προβλήματα συνδυαστικής βελτιστοποίησης

Τα προβλήματα συνδυαστικής βελτιστοποίησης είναι προβλήματα που περιλαμβάνουν την εύρεση του βέλτιστου αντικειμένου από ένα πεπερασμένο σύνολο αντικειμένων. Σε αυτό το πλαίσιο, το βέλτιστο υπολογίζεται από μια δεδομένη συνάρτηση αξιολόγησης που αντιστοιχίζει τα αντικείμενα σε κάποια βαθμολογία ή κόστος και ο στόχος είναι να βρεθεί το αντικείμενο που έχει το χαμηλότερο κόστος. Τα περισσότερα προβλήματα συνδυαστικής βελτιστοποίησης έχουν μεγάλο βαθμό δυσκολίας, καθώς ο

αριθμός των αντικειμένων του συνόλου αυξάνεται εξαιρετικά γρήγορα, καθιστώντας την αναζήτηση μη πρακτική [16].

Πρόβλημα περιπλανώμενου πωλητή

Η εργασία μελέτησε το Πρόβλημα του περιπλανώμενου πωλητή, γνωστό και ως TSP. Σε αυτό το πρόβλημα έχουμε Κ πόλεις και ο πωλητής πρέπει να τις επισκεφτεί όλες και να επιστρέψει στην πόλη προέλευσης. Ωστόσο, το ταξίδι μεταξύ πόλεων συνεπάγεται με κάποιο κόστος και πρέπει να βρεθεί η περιήγηση που ελαχιστοποιεί το συνολικό συσσωρευμένο κόστος.



Εικόνα 7: βέλτιστη περιήγηση στις πρωτεύουσες πολιτειών των ΗΠΑ [5]

Το πρόβλημα βρίσκει εφαρμογές στον σχεδιασμό, στις υπηρεσίες παράδοσης, στις αλληλουχίες DNA και πολλές άλλες. Η εύρεση καλύτερων περιηγήσεων μπορεί μερικές φορές να έχει σοβαρές οικονομικές επιπτώσεις, ωθώντας την επιστημονική κοινότητα και τις επιχειρήσεις να επενδύσουν σε καλύτερες μεθόδους για τέτοια προβλήματα.

Έστω ότι πρέπει να γίνει επίσκεψη σε Κ πόλεις, σε κάθε στάδιο της περιήγησης αφαιρείται μια πόλη, έως ότου δεν απομείνει καμία. Στο πρώτο στάδιο υπάρχουν Κ πόλεις προς επιλογή, στο δεύτερο στάδιο Κ-1 και ούτω καθεξής. Ο αριθμός των πιθανών περιηγήσεων είναι το γινόμενο του αριθμού των επιλογών κάθε σταδίου, έτσι η πολυπλοκότητά του είναι O(K!) και ανήκει στην κατηγορία των NP-hard προβλημάτων.

4.2 Βιβλιοθήκες

TensorFlow

Η TensorFlow είναι μια βιβλιοθήκη λογισμικού βαθιάς εκμάθησης ανοιχτού κώδικα που κυκλοφόρησε τον Νοέμβριο του 2015 από την Google [17]. Χρησιμοποιείται για τη μοντελοποίηση και την εκπαίδευση τεχνητών νευρωνικών δικτύων. Η TensorFlow επιτρέπει τη δημιουργία γραφήματος ροής δεδομένων και δομών για να δείξει πώς τα δεδομένα κινούνται μέσα σε ένα γράφημα.

Το νευρωνικό δίκτυο που χρησιμοποιήθηκε στην παρούσα εργασία δημιουργήθηκε χρησιμοποιώντας τη μονάδα Keras, η οποία είναι ένας εύκολος τρόπος υλοποίησης νευρωνικών δικτύων και βαθιάς μάθησης.

Networkx

Είναι ένα πακέτο λογισμικού στην Python για τη δημιουργία, το χειρισμό και τη μελέτη της δομής, της δυναμικής και της λειτουργίας πολύπλοκων δικτύων[18]. Χρησιμοποιείται για τη μελέτη μεγάλων σύνθετων δικτύων που αντιπροσωπεύονται σε μορφή γραφημάτων με κόμβους και ακμές. Χρησιμοποιώντας το networkx μπορούμε να φορτώσουμε και να αποθηκεύσουμε πολύπλοκα δίκτυα. Μπορούμε να δημιουργήσουμε πολλούς τύπους τυχαίων και κλασικών δικτύων, να αναλύσουμε τη δομή του δικτύου, να δημιουργήσουμε μοντέλα δικτύου, να σχεδιαστούν νέοι αλλά και να γίνει χρήση κλασσικών αλγόριθμων δικτύων.

OpenAI Gym

Το OpenAI Gym είναι μια εργαλειοθήκη για την ανάπτυξη και τη σύγκριση αλγορίθμων ενισχυτικής μάθησης [18]. Υποστηρίζει την εκπαίδευση παιχνιδιών όπως το pong ή το flipper. Αυτά τα παιχνίδια, ονομάζονται περιβάλλοντα και παρέχουν στον χρήστη τον χώρο παρατήρησης (observation-space) και τον χώρο δράσης (action-space) του περιβάλλοντος, με βάση τους οποίους ο agent μαθαίνει να παίζει το παιχνίδι. Ο πρωταρχικός σκοπός του OpenAI Gym είναι να παρέχει μια συλλογή διαφορετικών περιβαλλόντων που μοιράζονται μια παρόμοια διεπαφή που επιτρέπει τη σύγκριση χρησιμοποιώντας τους ίδιους αλγόριθμους

Τα πιο δημοφιλή περιβάλλοντα ονομάζονται «Κλασικού ελέγχου» και περιλαμβάνουν το CartPole-v0 ενώ τα περιβάλλοντα Box2D περιλαμβάνουν LunarLander-v2.

CartPole

Στο περιβάλλον αυτό ένας στύλος είναι συνδεδεμένος σε ένα καρότσι που κινείται κατά μήκος μιας τροχιάς χωρίς τριβές [19].

Εικόνα 8: Αρχική θέση, Θέση αποτυχίας

Περιβάλλον: Ο στύλος ξεκινά όρθιος και ο στόχος είναι να αποτραπεί η πτώση του ασκώντας μια δύναμη -1 ή +1 στο καρότσι για 500 επεισόδια. Μια ανταμοιβή +1 δίνεται για κάθε βήμα που παραμένει όρθιος και -100 αν πέσει. Ένα επεισόδιο τελειώνει όταν ο στύλος απέχει περισσότερο από 15 μοίρες από την κατακόρυφο ή το καρότσι μετακινηθεί περισσότερο από 2,4 μονάδες από το κέντρο. Το observation-space είναι (θέση, ταχύτητα, γωνία, γωνιακής ταχύτητά) και action-space είναι (δεξιά-αριστερά)

LunarLander

Το περιβάλλον αυτό είναι ένα πρόβλημα βελτιστοποίησης τροχιάς πυραύλων. Ο κινητήρα ενεργοποιείται ή απενεργοποιείται. Αυτός είναι ο λόγος για τον οποίο αυτό το περιβάλλον έχει διακριτές ενέργειες: ενεργοποίηση ή απενεργοποίηση κινητήρα [20].

Για το Περιβάλλον ισχύουν οι ακόλουθοι κανόνες: Το σημείο προσγείωσης είναι πάντα στις συντεταγμένες (0,0). Επιβράβευση για τη μετακίνηση από το πάνω μέρος της οθόνης έως το σημείο προσγείωσης και η μηδενική ταχύτητα είναι περίπου 100-140 πόντοι. Εάν το σκάφος απομακρυνθεί από το σημείο προσγείωσης, χάνει την ανταμοιβή. Το επεισόδιο τελειώνει εάν το σκάφος προσγειωθεί ή ακινητοποιηθεί, λαμβάνοντας επιπλέον -100 ή +100 πόντους. Κάθε επαφή με το έδαφος είναι +10. Η ενεργοποίηση του κύριου κινητήρα πυροδότησης επιφέρει ποινή -0,3 πόντων. Η επιτυχής προσγείωση είναι 200 βαθμοί. Είναι δυνατή η προσγείωση έξω από το σημείο προσγείωσης. Το καύσιμο είναι άπειρο, έτσι ένας agent μπορεί να μάθει να πετάει και στη συνέχεια να προσγειωθεί στην πρώτη του προσπάθεια. Το observation-space είναι ένα διάνυσμα μεγέθους 8 και αποτελείται: (θέση_x, θέση_y, ταχύτητα_y, ταχύτητα_x, γωνιακής ταχύτητά, γωνία, επαφή με έδαφος_1, επαφή με έδαφος_2). Το action-space αποτελείται από τέσσερις διακριτές ενέργειες: (μην κάνεις τίποτα, πήγαινε αριστερά ,πήγαινε πάνω(κύρια μηχανή), πήγαινε δεξιά)

Κεφάλαιο 5

Πειράματα

Τα αρχικά Πειράματα πραγματοποιήθηκαν στο OpenAI Gym, για τα περιβάλλοντα του CartPole και του LunarLander.

5.1 CartPole & LunarLander

Τα προβλήματα προσεγγίστηκαν με την χρήση της DQN αρχιτεκτονικής

Neural Networks

CartPole NN	*LunarLander NN*
input_dim=4 action=2	input_dim=8 action=4
2 hidden layer (256,64)	3 hidden layer (512,256,64)

Κώδικας Νευρωνικού

```
def OurModel(input_shape, action_space):
    model = Sequential()
    model.add(Dense(512, input_dim=input_shape, activation='relu'))
    model.add(Dense(256, activation='relu')) # 2nd hidden layer
    model.add(Dense(64, activation='relu')) # 3nd hidden layer
    model.add(Dense(action_space, activation='linear')) # 4 actions,
                        so 4 output neurons: 0 and 1 (L/R)
    # model.compile(loss="mse",optimizer="Adam", metrics=["accu-
racy"])
    model.compile(loss="mse", optimizer=RMSprop(learning_rate=0.001,
rho=0.95, epsilon=0.01), metrics=["accuracy"])
    model.summary()
    return model
```

Αλγόριθμος εκπαίδευσης DQN CartPole & LunarLander

1) Αρχικοποίηση νευρωνικού δικτύου με τυχαία βάρη.
2) Πάρε την αρχική κατάσταση S_0 *(S_0 = env.reset()).*
   a) Για κάθε επεισόδιο e ∈ [0, E_{end}]. *(CartPole_E_{end}=200, LunarLander _E_{end}=1000).*
   b) Επίλεξε μια ενέργεια a_t με βάση το state. *(agent.act(state)).*

c) Εφάρμοσε την ενέργεια a_t και πάρε την κατάσταση S_t στην οποία οδηγεί η ενέργεια και την επιβράβευση R_t *(env.step(action))*.

d) Αποθήκευσε $(s_t,a_t,r_{t+1},s_{t+1})$ στην experience replay memory.

e) Για κάθε βήμα t ∈ [0, T_{end}]. *(CartPole_ T_{end} =200, LunarLander _ T_{end} =1000)*.

   i) Πάρε ένα τυχαίο δείγμα μιας *mini batch Ds* από το *D*.

   ii) Για κάθε μετάβαση (s,a,r,s΄) της *mini batch*

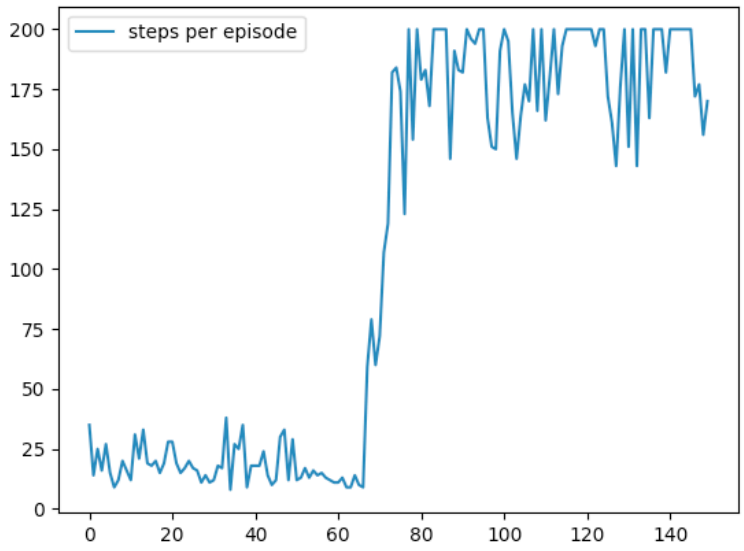
      (1) Προέβλεψε τις Q-values των ενεργειών της επομένης κατάστασης για το $\max_{a'}Q_\theta(s',a')$ του δικτύου

      (2) Υπολόγισε την target Value $y=r+\gamma\max_{a'}Q_\theta(s',a')$.

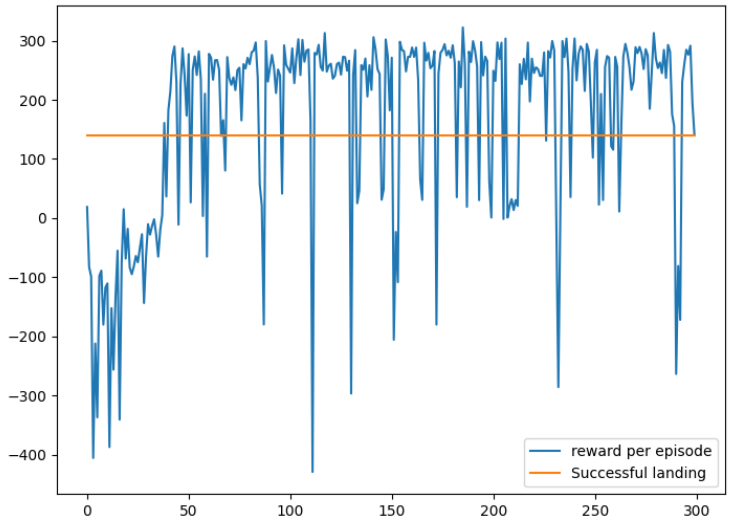
   iii) Εκπαίδευσε το δίκτυο Q_θ στο Ds (train_on_batch(state, target))

Διαγράμματα εκπαίδευσης

CartPole



Στο CartPole η εκπαίδευση πραγματοποιήθηκε για 150 επεισόδια, όπως φαίνεται και στο διάγραμμα ο agent είναι σε θέση σταδιακά να ισορροπεί τον στύλο και για τα 200 βήματα.



Στο LunarLander η εκπαίδευση πραγματοποιήθηκε για 300 επεισόδια καθώς πρόκειται για ένα πιο πολύπλοκο περιβάλλον. Ο agent προσγείωσε με επιτυχία το Lander στο 71% των προσπαθειών.

Κώδικας DQNAgent

```
class DQNAgent:
    def __init__(self):
        self.state_size = env.observation_space.shape[0] #8
        self.action_size = env.action_space.n#4
        self.memory = deque(maxlen=200000)
        self.gamma = 0.99          #discount_rate
        self.epsilon = 1.0         #exploration_rate
        self.epsilon_min = 0.01    #1/100 θα κάνει explore
        self.epsilon_decay = 0.99
        self.batch_size = 64
        self.train_start=1000
        self.model = OurModel(input_shape=self.state_size,
                              action_space = self.action_size)


   def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))
        if len(self.memory) > self.train_start:
            if self.epsilon > self.epsilon_min:
                self.epsilon *= self.epsilon_decay

   def act(self, state):# return action
        if np.random.rand() <= self.epsilon:
            return random.randrange(self.action_size)
        else:
            act_values = self.model.predict(state,verbose = 0)
            return np.argmax(act_values[0])

   def train(self):
```

```
 if len(self.memory) < self.train_start:
     return
minibatch = random.sample(self.memory, min(len(self.memory),
                                           self.batch_size))
 state = np.zeros((self.batch_size, self.state_size))
 next_state = np.zeros((self.batch_size, self.state_size))
 action, reward, done = [], [], []
 target =self.model.predict_on_batch(state)
 target_next =self.model.predict_on_batch(next_state)

 for i in range(self.batch_size):
     if done[i]:
         target[i][action[i]] = reward[i]
     else:
         target[i][action[i]] = reward[i] + self.gamma *
                                    (np.amax(target_next[i]))
 self.model.train_on_batch(state, target)
```

5.2.1 Εύρεση βέλτιστης εξόδου σε Λαβύρινθο

Οι ακόλουθες δοκιμές πραγματοποιηθήκαν εκτός των περιβαλλόντων της βιβλιοθήκης OpenAI. Τα περιβάλλοντα των πειραμάτων αναπαραστάθηκαν με πίνακες μεταβάσεων και γραφήματα.

Περιγραφή προβλήματος: Ο agent έχοντας ως αφετηρία το κελί start πρέπει να βρει τη συντομότερη διαδρομή έτσι ώστε να φτάσει στον προορισμό, κελί End αποφεύγοντας όλα τα κελιά με το x. Η αναζήτηση που πραγματοποιείται είναι ντετερμινιστική, καθώς όταν επιλεχθεί μια ενέργεια ο agent θα την πραγματοποιήσει με πιθανότητα 100%.

Start		x	
	x		
	x		
			End

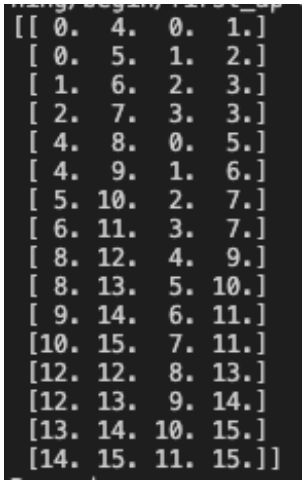
Εικόνα 9:Περιβάλλον πειράματος

Οι μεταβάσεις στα κελιά πραγματοποιούνται με την αρίθμηση της Εικόνας 10.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Εικόνα 10: Αρίθμηση κελίων

Δημιουργία πίνακα μεταβάσεων έτσι ώστε να γνωρίζουμε που οδηγεί η κάθε ενέργεια (left,down,up,right)



Εικόνα 11: Πίνακας μεταβάσεων

Παράδειγμα μεταβάσεων: Έστω ότι βρισκόμαστε στο κελί 0, [0, 4, 0, 1], αν επιλεγεί η ενέργεια left οδηγεί ξανά στο κελί 0 καθώς δεν υπάρχει αριστερός γείτονας, ομοίως και αν επιλεχθεί η ενέργεια up, αν επιλεχθεί η ενέργεια down οδηγεί στο κελί 4 και αν η ενέργεια right στο κελί 1.

Λειτουργία αλγορίθμου

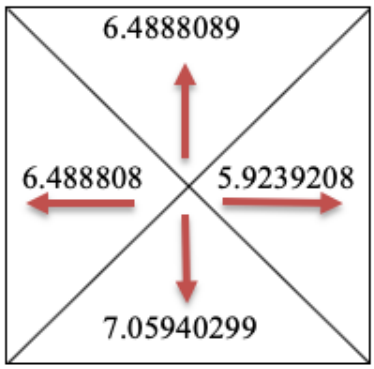
1. Δημιουργία πίνακα μεταβάσεων
2. Δημιουργία πίνακα ανταμοιβών (όλες οι θέσεις -1 εκτός από προορισμό 10)
3. Δημιουργία πίνακα τελικών καταστάσεων (προορισμός ή κελί με x)
    i. Για κάθε επεισόδιο e ∈ [0, E_{end}]
    ii. Ξεκινά από το κελί αφετηρία
    iii. Για κάθε βήμα t ∈ T
        1. Πραγματοποίησε μεταβάσεις (τυχαίες ή συμφώνα με το Q-table)
        2. Για κάθε ενέργεια που πραγματοποιείται ενημέρωσε τον Q-table σύμφωνα με τον τύπο: Q[current,next] = (1-lr)*Q[current,next] + lr*(Reward+discount*max(Q[next])

Το επεισόδιο τελειώνει όταν φτάσουμε σε τελική κατάσταση ή όταν πραγματοποιηθούν όλα τα βήματα. Με την πάροδο των επεισοδίων οι τυχαίες ενέργειες μειώνονται και επιλέγονται ενέργειες με βάση τον Q table.

```
[[ 6.48880896  7.05940299  6.48880896  5.92392087]
 [ 6.48880896 -0.5         5.92392087 -0.5       ]
 [ 0.          0.          0.          0.        ]
 [-0.4         0.         -0.4        -0.4       ]
 [ 7.05940299  7.6357606   6.48880896 -0.5       ]
 [ 0.          0.          0.          0.        ]
 [-0.499968    8.806      -0.4992      0.        ]
 [ 0.          9.32459644 -0.4        -0.496     ]
 [ 7.6357606   8.21794     7.05940299 -0.5       ]
 [ 0.          0.          0.          0.        ]
 [-0.5         9.4         8.21794     9.4       ]
 [ 8.80587951 10.          7.83777165  9.39433595]
 [ 8.21794     8.21794     7.6357606   8.806     ]
 [ 8.21794     8.806      -0.5         9.4       ]
 [ 8.806       9.4         8.806      10.        ]
 [ 0.          0.          0.          0.        ]]
```

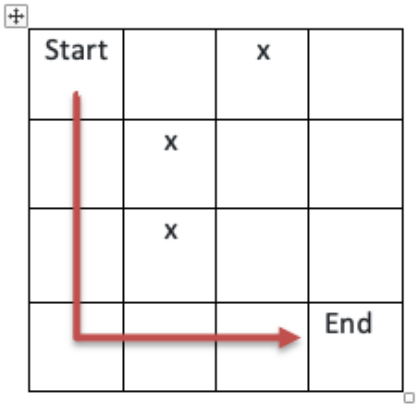
Εικόνα 12: Τελικό Q table

Το τελικό Q table καθορίζει την πολιτική: Όταν είμαστε στο κελί 0 η ενέργεια που θα επιλεχθεί (left,down,up,right) θα είναι down (κελί 4) καθώς αυτή η τιμή του Q table είναι η μεγαλύτερη.



Εικόνα 13: Actions-Values για κελί 0

Το τελικό μονοπάτι: 0, 4, 8, 12, 13, 14, 15



5.2.2 Εύρεση εξόδου σε Λαβύρινθο ΙΙ

Σε ένα δεύτερο πείραμα δόθηκε στον Agent ένα πιο απαιτητικό περιβάλλον και αρχικά εκπαιδεύτηκε για 2000 επεισόδια

Start	x	End	
	x		
	x	x	

Εικόνα 14: Περιβάλλον δευτέρου πειράματος

Ο agent σε κανένα από τα επεισόδια δεν μπόρεσε να βρει το κελί προορισμού.

```
[[-0.995      -0.995      -0.995      -0.5       ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [-0.995      -0.995      -0.995      -0.5       ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [-0.995      -1.48504999 -0.995      -0.5       ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.         -0.4         0.          0.        ]
 [-1.48504993 -1.48504993 -0.995      -0.995     ]
 [-1.48449851 -0.99492549 -0.5        -0.98711142]
 [-0.98641306 -0.99024794 -0.49984    -0.89584128]
 [-0.99348664 -0.892032   -0.4        -0.892032  ]]
```

Εικόνα 15: Q table, δεν βρέθηκε διαδρομή που οδηγεί στο προορισμό

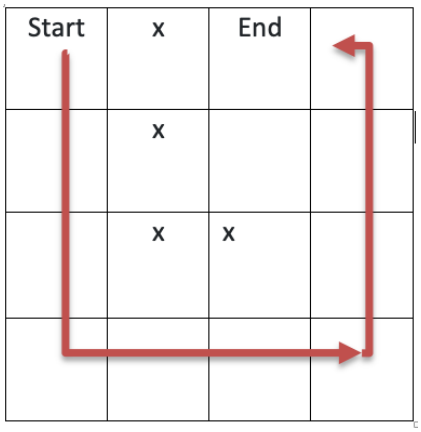
Σε αυτή την περίπτωση, μη λαμβάνοντας κάποια θετική επιβράβευση πάρα μόνο αρνητικές (-0.5 για κάθε βήμα), αποφασίζει να ακολουθήσει την διαδρομή που θα τερματίσει γρηγορότερα την διαδικασία, αρά θα του δώσει τις λιγότερες αρνητικές επιβραβεύσεις. Δηλαδή θα "αυτοκτονήσει" και μάλιστα με τον πιο σύντομο δρόμο, πέφτοντας κατευθείαν στην τρύπα του κελιού 1.

Για να ξεπεραστεί αυτή η συμπεριφορά αυξήθηκαν τα επεισόδια από 2.000 σε 10.000 έτσι ώστε να αυξηθεί το πλήθος των ενεργειών εξερεύνησης (τυχαίες ενέργειες).

```
[[ 4.2629245   4.81103485  4.2629245  -0.5        ]
 [ 0.          0.          0.          0.         ]
 [ 0.          0.          0.          0.         ]
 [10.          8.806       9.4         9.4        ]
 [ 4.81103485  5.36468167  4.2629245  -0.5        ]
 [ 0.          0.          0.          0.         ]
 [-0.5        -0.5        10.          8.806      ]
 [ 9.4         8.21794     9.4         8.806      ]
 [ 5.36468167  5.92392087  4.81103485 -0.5        ]
 [ 0.          0.          0.          0.         ]
 [ 0.          0.          0.          0.         ]
 [-0.5         7.6357606   8.806       8.21794    ]
 [ 5.92392087  5.92392087  5.36468167  6.48880896]
 [ 5.92392087  6.48880896 -0.5         7.05940299]
 [ 6.48880896  7.05940299 -0.5         7.6357606 ]
 [ 7.05940299  7.6357606   8.21794     7.6357606 ]]
```

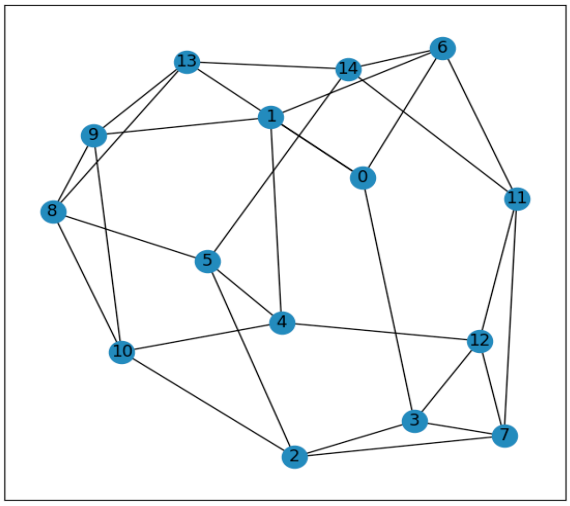
Εικόνα 16: Q table 10.000 επεισοδίων

Το τελικό μονοπάτι : 0, 4, 8, 12, 13, 14, 15, 11, 7, 3, 2



5.3 Εύρεση ελάχιστου μονοπατιού Q-Learning

Περιγραφή προβλήματος: Σε ένα μη κατευθυνόμενο γράφημα με βάρη, ο agent ψάχνει τη συντομότερη διαδρομή μεταξύ του κόμβου αφετηρίας και του κόμβου προορισμού. Το γράφημα δημιουργήθηκε με τυχαία βάρη ακμών και τα αποτελέσματα συγκρίνονται με τα αντίστοιχα του αλγορίθμου dijkstra.



Δημιουργία πίνακα Reward στον οποίο αποθηκεύονται οι ανταμοιβές. Για κάθε ακμή που οδηγεί σε κόμβο η θέση στον πίνακα Reward ενημερώνεται με το αντίστοιχο **αρνητικό βάρος**. Για ακμή 0->2 με βάρος 9 => Reward[0][2] = Reward[2][0] = -9. Οι γείτονες του κόμβου προορισμού παίρνουν την τιμή 100 στην σχετική θέση. Έστω κόμβος 9 τελικός, στον 9 φθάνουμε από τους κόμβους (4, 5, 8, 10), Reward[9][4] = Reward[9][5]….=100

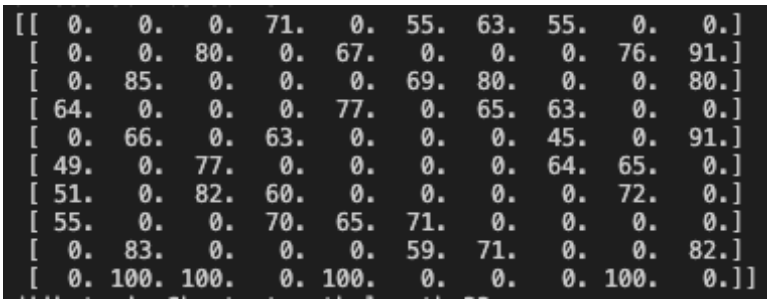
```
[[  0. -22.   0.  -6.  -3. -10.   0.   0.   0.   0.   0.]
 [-22.   0.   0.   0.   0.   0.  -9.   0.   0.  -7.  -8.]
 [  0.   0.   0.   0. -19. -24.   0. -22.  -7.   0.   0.]
 [ -6.   0.   0.   0.   0. -19.   0.   0. -20.   0. -11.]
 [ -3.   0. -19.   0.   0.   0.   0.   0.   0. -17. -16.]
 [-10.   0. -24. -19.   0.   0.   0.  -9.   0.   0.   0.]
 [  0.  -9.   0.   0.   0.   0.   0. -27. -16. -29.   0.]
 [  0.   0. -22.   0.   0.  -9. -27.   0. -12.   0.   0.]
 [  0.   0.  -7. -20.   0.   0. -16. -12.   0.   0.   0.]
 [  0.  -7.   0.   0. -17.   0. -29.   0.   0.   0.  -8.]
 [  0. 100.   0. 100. 100.   0.   0.   0.   0. 100.   0.]]
```

Εικόνα 17: Πίνακας Reward

Κατά την εκπαίδευση του αλγορίθμου θα επιλέγεται μια ενέργεια-βήμα (τυχαία ή σύμφωνα με τον Q-table) και βάση της επιλεγμένης ενέργειας και της ανταμοιβής που

επιστρέφει θα πραγματοποιηθεί η ανανέωση των τιμών του Q-table σύμφωνα με τον τύπο:

*Q[current,next]= (1-lr)*Q[current,next] + lr*(Reward+max(Q[next])*



Εικόνα 18: Q-table

Βήματα αλγορίθμου εκπαίδευσης

1. Δημιουργία γράφου με τυχαία βάρη
2. Δημιουργία πίνακα Reward
    i. Για κάθε επεισόδιο e ∈ [0, E_{end}]
    ii. Ξεκίνα από το κελί αφετηρία
    iii. Για κάθε βήμα t ∈ T μέχρι να φτάσεις στον προορισμό.
        1. Πραγματοποίησε μεταβάσεις (τυχαίες ή συμφώνα με το Q-table)
        2. Για κάθε ενέργεια που πραγματοποιείται ενημέρωσε τον Q-table σύμφωνα με τον τύπο: Q[current,next]= (1-lr)*Q[current,next] + lr*(Reward+ discount *max(Q[next])

b. Κάθε επεισόδιο τελειώνει μόλις γίνει επίσκεψη στον προορισμό ή ολοκληρωθούν όλα τα επεισόδια

Αποτελέσματα

Γράφημα με 11 κόμβους

```
dijkstra's Shortest path length 10
Reinforcement Learning Shortest path length 10
Reinforcement Learning Shortest path [0, 7, 10]
dijkstra's Shortest path [0, 7, 10]
```

Γράφημα με 100 κόμβους

```
dijkstra's Shortest path length 9
Reinforcement Learning Shortest path length 9
Reinforcement Learning Shortest path [0, 11, 37, 88]
dijkstra's Shortest path [0, 11, 37, 88]
```

Γράφημα με 500 κόμβους

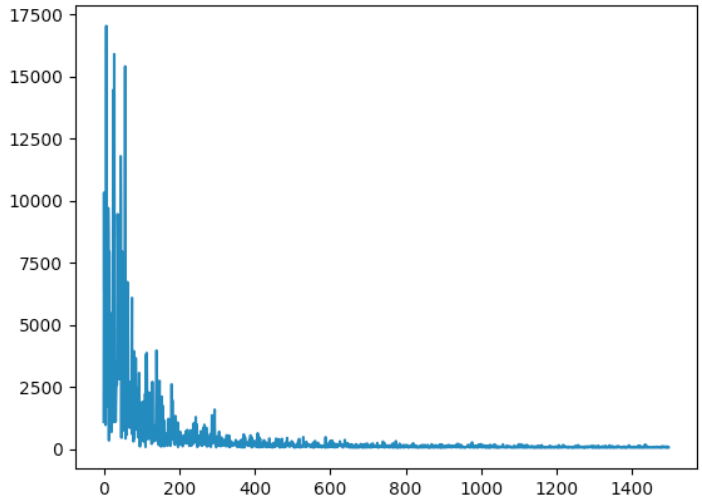
```
dijkstra's Shortest path length 25
Reinforcement Learning Shortest path length 25
Reinforcement Learning Shortest path [0, 461, 307, 349, 218, 456, 455]
dijkstra's Shortest path [0, 461, 307, 349, 218, 456, 455]
```

Γράφημα με 700 κόμβους

```
dijkstra's Shortest path length 20
Reinforcement Learning Shortest path length 20
Reinforcement Learning Shortest path [0, 96, 296, 229, 675, 411, 501, 683]
dijkstra's Shortest path [0, 96, 296, 229, 675, 411, 501, 683]
```

Γράφημα με 1000 κόμβους

```
dijkstra's Shortest path length 90
Reinforcement Learning Shortest path length 90
Reinforcement Learning Shortest path [0, 50, 629, 137, 535, 887, 703, 669, 261, 596, 366, 990]
dijkstra's Shortest path [0, 50, 629, 137, 535, 887, 703, 669, 261, 596, 366, 990]
```

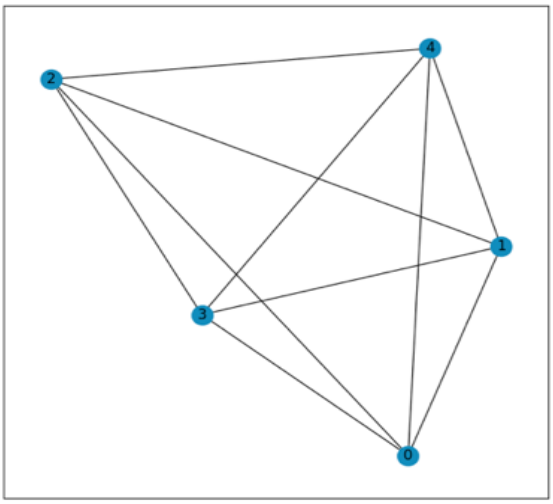


Εικόνα 19: Διάγραμμα εκπαίδευσης για γράφημα 500 κόμβων, 1500 επεισόδια, με ελάχιστο μονοπάτι 25.

Στην Εικόνα 5.9 βλέπουμε το μήκος του μονοπατιού σε κάθε επεισόδιο. Στα πρώτα επεισόδια οι ενέργειες είναι τυχαίες και έτσι επιστρέφει πολύ μεγάλες τιμές, σταδιακά με την πάροδο των επεισοδίων ο agent εκπαιδεύεται και συγκλίνει στην ελάχιστη διαδρομή.

5.4 Εύρεση βέλτιστης διαδρομής TSP σε Γράφο με Reinforcement learning

Περιγραφή προβλήματος: Σε ένα πλήρη μη κατευθυνόμενο γράφημα με βάρη, ο agent ψάχνει την συντομότερη διαδρομή έτσι ώστε, ξεκινώντας από έναν κόμβο αφετηρία να επισκεφθεί όλους τους υπολοίπους και να επιστρέψει στην αφετηρία. Το γράφημα δημιουργήθηκε με τυχαία βάρη ακμών και τα αποτελέσματα συγκρίνονται με τα αντίστοιχα του αλγορίθμου greedyTSP της βιβλιοθήκης networkx.



Πίνακας Reward: Στο πρόβλημα TSP θέλουμε οι επιβραβεύσεις να είναι μεγαλύτερες για τις μικρότερες αποστάσεις, έτσι ο πίνακας Reward ενημερώνεται με τιμή ίση με το μεγαλύτερο βάρος ακμής του γραφήματος αφαιρώντας το βάρος της αντίστοιχης ακμής.

$$Reward[i,j]=max(edge\_weight\ )-current\_weight[i,j]$$

Κατά την εκπαίδευση του ο αλγορίθμου ξεκινάει από την αφετηρία και μπαίνει στην travel_list. Στο available_actions υπάρχουν όλοι οι κομβοι που δεν έχουμε ακόμα επισκεφθεί και ενημερώνεται κάθε φορά που επιλέγεται ένας.

Έως ότου το available_actions αδειάσει (έχει γίνει επίσκεψη σε όλους τους κόμβους) επιλέγεται μια ενέργεια, είτε τυχαία (exploration), είτε με βάση το Q_table (exploitation) με τη γνωστή διαδικασία. Η διαφορά σε αυτή την προσέγγιση είναι ότι θα επιλεχθεί η μεγαλύτερη τιμή του Q-table από το possible_actions (κόμβους που δεν έχουμε επισκεφθεί ακόμα) και όχι από όλες τις τιμές του Q-table.

Παράδειγμα λειτουργίας: Έστω action=0 και possible_actions=[1,2,5].Η επιλογή θα γινεί βάση της μεγαλύτερης τιμής των Q_table[0][1], Q_table[0][2] και Q_table[0][5]. Όταν πραγματοποιηθεί η επιλογή του action θα ανανεωθεί η τιμή του Q-table και θα προστεθεί ο κόμβος στην travel_list.

Βήματα αλγορίθμου εκπαίδευσης

- Δημιουργία πλήρη γράφου με τυχαία βάρη
- Δημιουργία πίνακα Reward
  - Για κάθε επεισόδιο e ∈ [0, E_{end}]
  - Ξεκινά από τον κόμβο αφετηρία
  - Για κάθε βήμα t ∈ T μέχρι να επισκεφθείς όλους τους κόμβους
    - πραγματοποίησε μεταβάσεις
    - για κάθε ενέργεια που πραγματοποιείται ενημέρωσε τον Q-table σύμφωνα με τον τύπο: Q[current,next] = (1-lr)*Q[current,next] + lr*(Reward+discount*max(Q[next])
  - Μόλις γίνει επίσκεψη σε όλους τους κόμβους, επέστρεψε στον αρχικό και ενημέρωσε τον Q-table

Κατά την υλοποίηση παρουσιάστηκε ένα από τα κύρια προβλήματα των αλγορίθμων RL, το οποίο είναι η θέσπιση των παραμέτρων: ποσοστό εκμάθησης (lr), συντελεστής έκπτωσης (discount).

Δοκιμές
Nodes=10

Learning rate	discount	path size
0.5	0.1	75
0.5	0.5	90
0.5	0.8	105

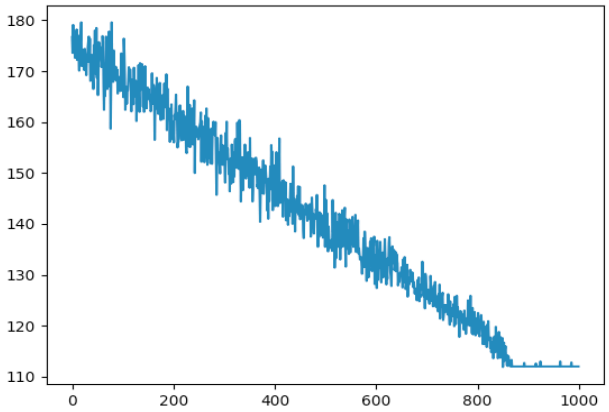
Μετά από δοκιμές οι παράμετροι οριστήκαν σε Learning rate = 0.5, Discount = 0.9 ωστόσο ενδέχεται σχετικές τροποποιήσεις να φέρνουν καλύτερα αποτελέσματα σε ορισμένες περιπτώσεις

Αποτελέσματα

Γράφος 25 κόμβων

RL path size=110 Greedy tsp path size=113

```
Best trajectory found by Reinforcement_Learning_tsp:
[0, 20, 15, 13, 16, 6, 12, 9, 17, 7, 2, 5, 10, 23, 18, 19, 3, 22, 8, 24, 14, 11, 21, 4, 1, 0]
With path size:
110
Best trajectory found by greedy_tsp:
[0, 2, 5, 10, 16, 6, 9, 4, 12, 3, 13, 15, 20, 24, 8, 19, 11, 14, 1, 23, 18, 7, 17, 21, 22, 0]
With path size:
113
```

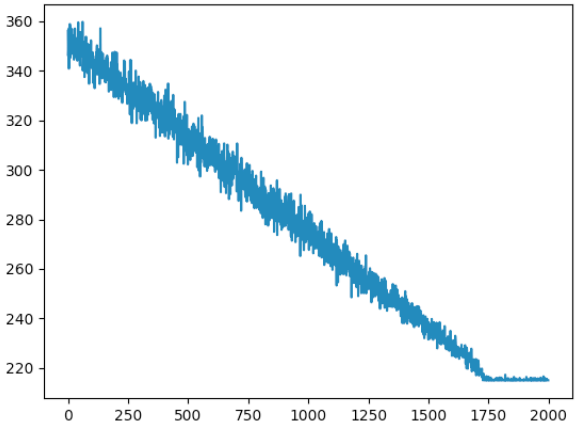


Εικόνα 20: Μέσο μήκος διαδρομής ανά πέντε επεισόδιά για 25 κόμβους

Γράφος με 50 κόμβους

RL path size=212 Greedy tsp path size=215

```
Best trajectory found by Reinforcement_Learning_tsp:
[0, 24, 12, 34, 3, 2, 20, 18, 44, 19, 31, 14, 39, 16, 35, 17, 7, 42, 27, 23, 45, 46, 11, 49, 15, 25, 10, 43, 13, 36, 21,
 32, 22, 48, 8, 29, 41, 37, 1, 30, 5, 6, 28, 26, 9, 38, 47, 4, 40, 33, 0]
With path size:
212
Best trajectory found by greedy_tsp:
[0, 2, 1, 4, 31, 7, 5, 6, 11, 16, 3, 34, 12, 8, 20, 9, 26, 17, 13, 23, 27, 14, 25, 10, 21, 19, 29, 30, 24, 15, 39, 47, 4
4, 18, 46, 37, 35, 40, 22, 32, 33, 43, 42, 49, 28, 36, 48, 41, 45, 38, 0]
With path size:
215
```

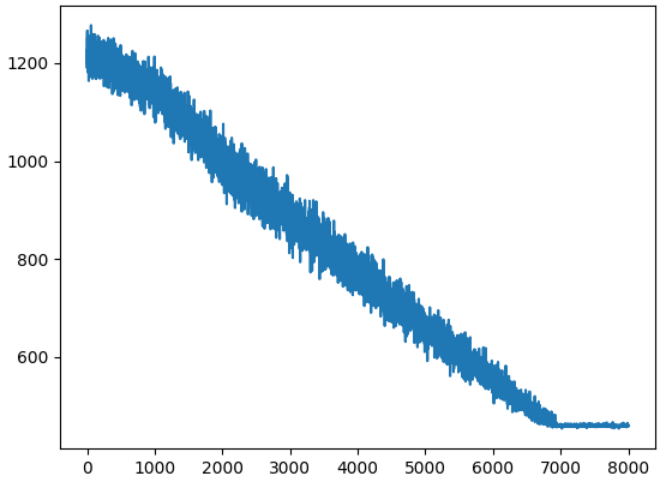


Εικόνα 21: Μέσο μήκος διαδρομής για 50 κόμβου

Γράφος 100 κόμβων

RL path size=459 Greedy tsp path size=469

```
Best trajectory found by Reinforcement_Learning_tsp:
[0, 35, 42, 23, 9, 10, 20, 30, 69, 32, 40, 89, 4, 16, 6, 8, 43, 7, 12, 53, 34, 31, 25, 58, 44, 2, 26, 21, 62, 13, 27, 5, 63, 37, 83, 75, 14, 48, 33, 67, 29, 38, 45
, 22, 54, 80, 28, 19, 18, 15, 88, 66, 47, 36, 59, 3, 68, 51, 11, 79, 85, 81, 52, 1, 24, 46, 17, 73, 70, 55, 76, 65, 93, 82, 56, 50, 49, 87, 84, 94, 78, 64, 86, 41,
 72, 92, 39, 61, 74, 77, 90, 95, 97, 91, 57, 71, 60, 99, 96, 98, 0]
With path size:
459
Best trajectory found by greedy_tsp:
[0, 4, 16, 6, 8, 43, 7, 12, 53, 34, 23, 9, 10, 20, 30, 69, 32, 40, 3, 55, 14, 48, 26, 2, 44, 5, 27, 13, 62, 21, 24, 1, 52, 46, 17, 73, 58, 25, 31, 60, 71, 47, 18,
15, 88, 66, 51, 11, 79, 85, 81, 92, 63, 37, 19, 28, 61, 22, 35, 42, 82, 33, 67, 29, 38, 45, 56, 50, 49, 75, 41, 72, 78, 64, 80, 39, 89, 90, 54, 98, 74, 36, 59, 57,
 91, 70, 86, 97, 95, 84, 94, 83, 77, 93, 65, 76, 68, 99, 87, 96, 0]
With path size:
469
```



Εικόνα 22: Μέσο μήκος διαδρομής για 100 κόμβους

Όπως φαίνεται από τα διαγράμματα εκπαίδευσης, στα πρώτα επεισόδια οι ενέργειες είναι τυχαίες και έτσι οι διαδρομές έχουν μεγάλες τιμές, με την πάροδο των επεισοδίων ο agent εκπαιδεύεται επιστρέφοντας διαδρομές μικρότερου μήκους.

5.5 Εύρεση βέλτιστης διαδρομής TSP με Deep Reinforcement learning

Η υλοποίηση του αλγορίθμου Deep-Q-Learning έγινε με την χρήση νευρωνικών δικτυών. Η πρόβλεψη του καταλληλότερου κόμβου προς επίσκεψη δεν βασίζεται στο Q-table (όπως στο προηγούμενο παράδειγμα) αλλά σε ένα νευρωνικό δίκτυο και στις παραμέτρους θ που εκπαιδεύονται έτσι ώστε να επιστρέψουν το αποτέλεσμα.

Υλοποίηση αλγορίθμου Deep-Q-Learning

State space: Αρχικά για την αλληλεπίδραση του agent-δικτύου με το περιβάλλον ορίστηκε ένα διάνυσμα S, με μήκος όσο οι κόμβοι M του γραφήματος και τιμή η σειρά με την οποία έχει γίνει επίσκεψη σε κάθε κόμβο. Αν δεν έχει γίνει επίσκεψη η αντίστοιχη τιμή είναι μηδέν.

$S_t = [V_1, V_2, \ldots, V_M] \text{ για } s_t \in S$

Action space: Οι πιθανές ενέργειες από τις οποίες ο agent θα επιλέξει την καλύτερη, ορίστηκε ένα διάνυσμα A_t, με μήκος όσο οι κόμβοι M του γραφήματος.

Πίνακας Reward: Στο πρόβλημα TSP θέλουμε οι επιβραβεύσεις να είναι μεγαλύτερες για τις μικρότερες αποστάσεις, έτσι ο πίνακας Reward ενημερώνεται με τιμή ίση με το μεγαλύτερο βάρος ακμής του γραφήματος αφαιρώντας το βάρος της αντίστοιχης ακμής.

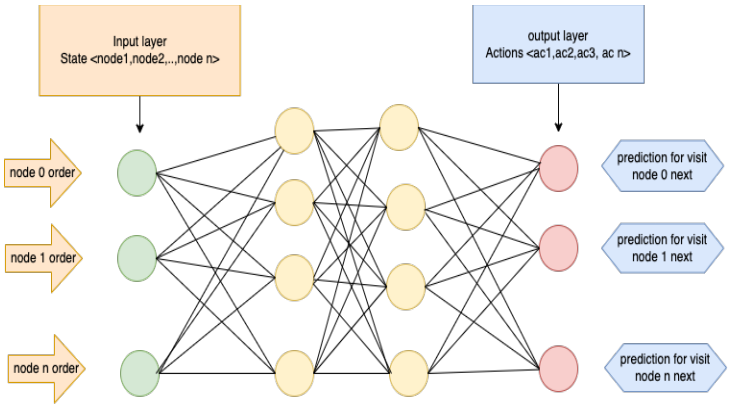
Reward[i,j]=max(edge_weight)-current_weight[i,j]

Agent: Ένα νευρωνικό δίκτυο τριών κρυφών επίπεδων (1024,512,124) με είσοδο ένα διάνυσμα S_t και έξοδο ένα διάνυσμα A_t. Ως συνάρτηση απώλειας ορίστηκε το μέσο τετραγωνικό σφάλμα (MSE) και ως βελτιστοποιητής ο RMSprop.

Λειτουργία εκπαίδευσης: Σε κάθε βήμα (επιλογή ενέργειας) αποθηκεύεται στην μνήμη-ουρά μια εγγραφή της μορφής <S_t, action, reward, $S_{t\text{-next}}$>, η οποία στην συνέχεια δειγματοληπτικά θα χρησιμοποιηθεί στην εκπαίδευση του δικτύου. Κατά την εκπαίδευση οι προβλέψεις του δικτύου με βάση το state θα συνδυαστούν με τις προβλέψεις του δικτύου με βάση το state-next:
Predicted_values_by_state[action]=reward+gamma*best_predicted_val-
ues_by_next_state

Στόχος της εκπαίδευσης του νευρωνικού δικτύου, δεδομένου ενός state εισόδου, είναι να προσεγγίσει τις αναμενόμενες συνολικές επιβραβεύσεις για κάθε μια από τις διαθέσιμες ενέργειες. Με αυτόν τον τρόπο προβλέπει ποια ενέργεια πρέπει να επιλεχθεί έτσι ώστε να αποφέρει τις καλύτερες επιβραβεύσεις.

Λειτουργία επιλογής ενέργειας: Έστω γράφημα 6 κόμβων με αρχικό κόμβο τον 0 και αρχικό State <0, 0, 0, 0, 0, 0> και έστω ότι βρίσκεται στην μέση της διαδρομής έχοντας επισκεφθεί τους κόμβους 2, 5 το State θα είναι <0, 0, 1, 0, 0, 2> . Το δίκτυο θα λάβει ως είσοδο το διάνυσμα State και ως έξοδο θα επιστρέψει ένα διάνυσμα με τις προβλέψεις του σχετικά με το ποιος θα είναι ο επόμενος κόμβος κατεύθυνσης. Η θέση με την μεγαλύτερη τιμή πρόβλεψης θα επιλεχθεί ως ενέργεια .



Εικόνα 23: Αναπαράσταση λειτουργείας Agent

Βήματα αλγορίθμου εκπαίδευσης deep Q learning

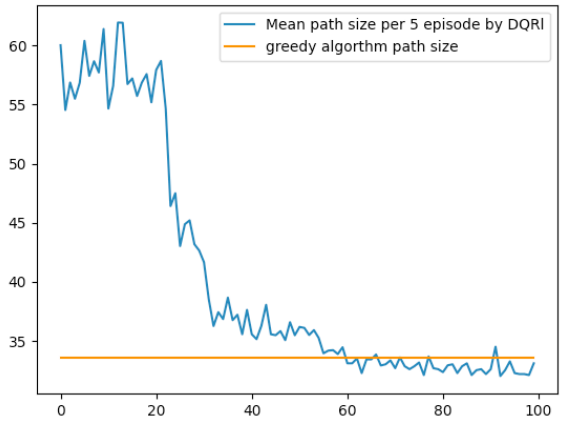
- Δημιουργία πλήρη γράφου με τυχαία βάρη
- Δημιουργία πίνακα Reward
  - Για κάθε επεισόδιο e ∈ [0, E_{end}]
  - Ξεκινά από το κελί αφετηρία
  - Μέχρι να επισκεφθείς τους κόμβους και να επιστρέψεις στην αφετηρία
    - Πραγματοποίησε ενέργειες.
    - Για κάθε ενέργεια-βήμα που πραγματοποιείται
      - Αποθήκευσε στην μνήμη μια εγγραφή της μορφής [S_t, action, reward, $S_{t\text{-next}}$]
      - Πάρε ένα τυχαίο δείγμα μιας *mini batch*(32) από την μνήμη.
      - Προέβλεψε τις Q-values των ενεργειών της $S_{t\text{-next}}$ ($'Q_\theta(s',a')$).

- Υπολόγισε την target value $y=r+\gamma \max_{a'} Q_{\theta}(s',a')$.
- Εκπαίδευσε το δίκτυο (state, target value

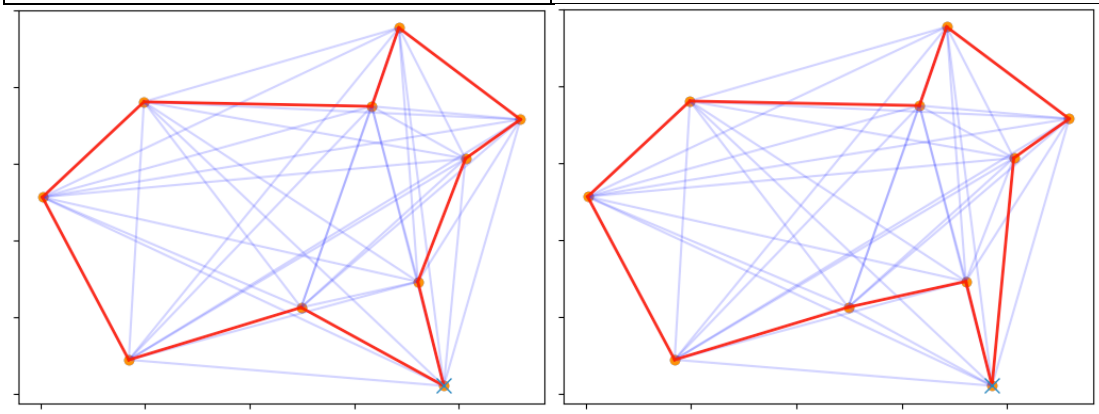
Αποτελέσματα

Σε αυτή την ενότητα θα παρατεθούν τα αποτελέσματα των αλγόριθμων και τα αντίστοιχα του greedy TSP, τα διαγράμματα εκπαίδευσης και η οπτικοποίηση της διαδρομής.

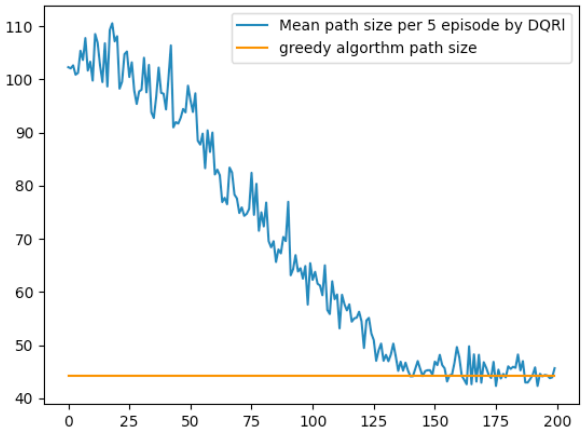
Γράφος 10 κόμβων



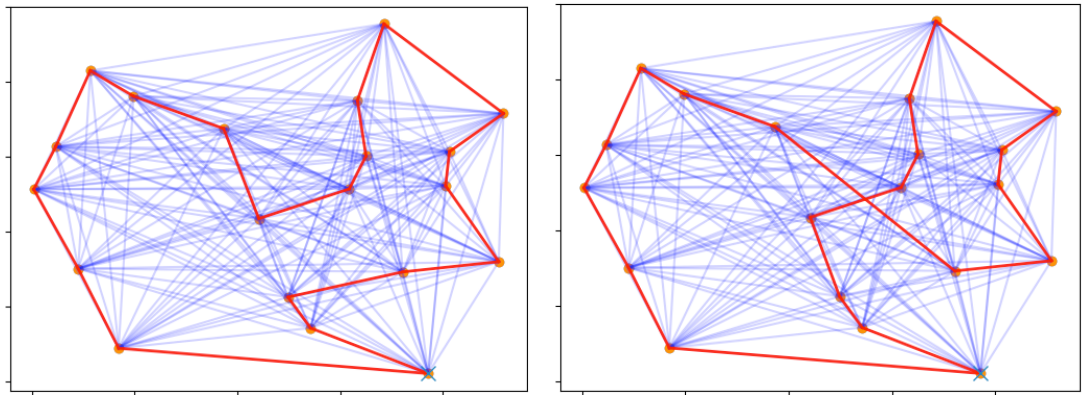
Διαδρομή DQN	Διαδρομή greedy-tsp
32	33.5



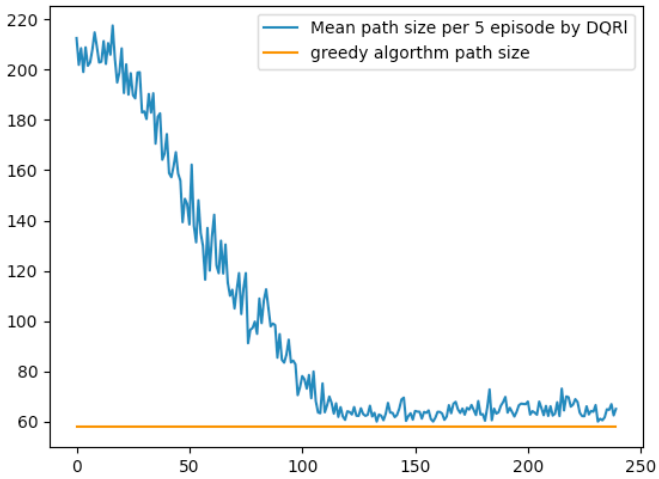
Γράφος 20 κόμβων



Διαδρομή DQN	Διαδρομή greedy-tsp
41.7	44.1

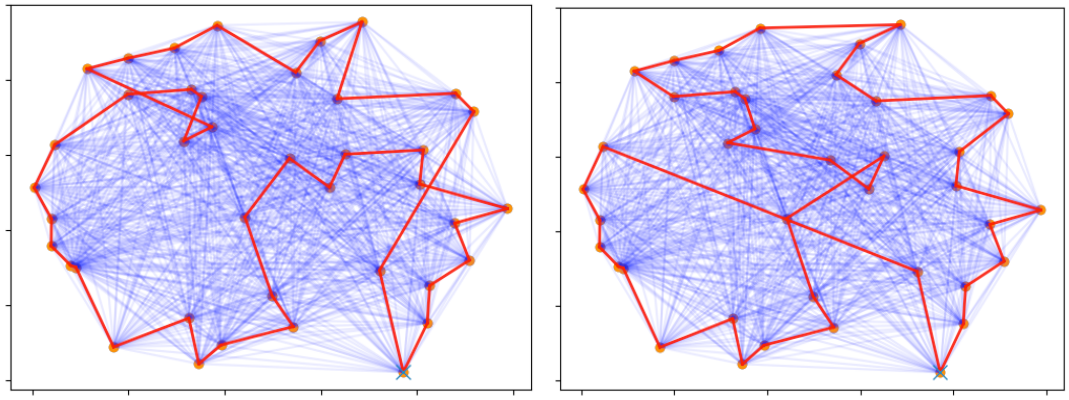


Γράφος 40 κόμβων

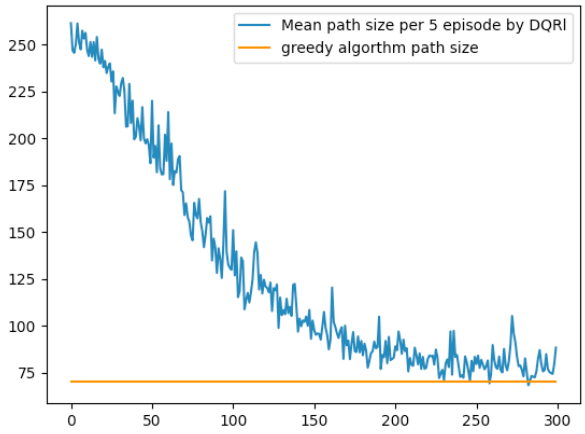


Διαδρομή DQN	Διαδρομή greedy-tsp

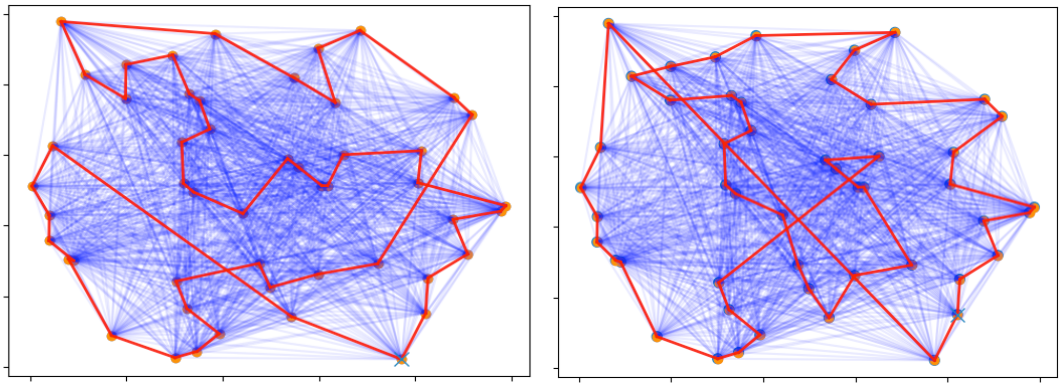
56.6	57.7



Γράφος 50 κόμβων



Διαδρομή DQN	Διαδρομή greedy-tsp
66.3	70.3



Κατά την εκκίνηση οι διαδρομές που επιστρέφονται είναι αρκετά μεγαλύτερες από τις αντίστοιχες των τελευταίων επεισοδίων, μια συμπεριφορά που έρχεται σε συμφωνία με τα διαγράμματα εκπαίδευσης, υποδεικνύοντας ότι το δίκτυο εκπαιδεύεται στην αναζήτηση συντομότερης διαδρομής.

Στα πλαίσια της εργασίας πραγματοποιήθηκε δοκιμή σύγκρισης των τριών αρχιτεκτονικών (Deep Q-Network, Double Deep Q-Network, Duelling Deep Q-Network) προσαρμόζοντάς τες στον αλγόριθμο TSP.

Υλοποίηση αλγορίθμου Double Deep Q-Network

Η υλοποίηση του Double Deep Q-Network διαφοροποιείται από την αρχική στα εξής σημεία.

1. Χρησιμοποιούνται δυο πανομοιότυπα νευρωνικά δίκτυα.
    a. Το κύριο δίκτυο, το οποίο είναι υπεύθυνο για την επιλογή της ενέργειας και εκπαιδεύεται.
    b. Το δευτερεύων νευρωνικό δίκτυο το οποίο εκτίμα την τιμή της επιλεγμένης ενέργειας. Σε αντίθεση με το κύριο δίκτυο δεν εκπαιδεύεται αλλά συντονίζει περιοδικά τα βάρη του με αυτά του κύριου δίκτυο
    c. Ο συντονισμός των δικτύων πραγματοποιείται ανά έναν συγκεκριμένο αριθμό επεισοδίων, επιδιώκοντας με αυτό τον τρόπο την σταθερότητα στις προβλέψεις. Στις δοκιμές που πραγματοποιήθηκαν ο ρυθμός αυτός είναι τα 20 επεισόδια.
    d. Όσο αφορά τον συντονισμό των βαρών μπορεί να επιτευχθεί με 2 μεθόδους
        i. Hard update: Το δευτερεύων ανανεώνει πλήρως τα βάρη του με βάση το κύριο.
        ii. Soft update: Το δευτερεύων ανανεώνει ποσοστιακά τα βάρη του με βάση το κύριο (συνήθως 10%)

        Στις δοκιμές χρησιμοποιήθηκε η Hard update μέθοδος

Υλοποίηση αλγορίθμου Duelling Deep Q-Network

Η Duelling Deep Q-Network προσέγγιση αλλάζει τον τρόπο με τον οποίο το νευρωνικό κάνει την πρόβλεψη και μπορεί να εφαρμοστεί στην DQN και στην DDQN αρχιτεκτονική.Ο αλγόριθμος χωρίζει τις τιμές Q σε δύο διαφορετικά μέρη, τη συνάρτηση τιμής V(s) και τη συνάρτηση πλεονεκτήματος A(s, a)
Η συνάρτηση τιμής V(s) επιστρέφει την συνολική ανταμοιβή που θα συλλεχθεί από

την κατάσταση s. Η συνάρτηση A(s, a) επιστρέφει πόσο καλύτερη είναι μια ενέργεια σε σύγκριση με τις υπόλοιπες.

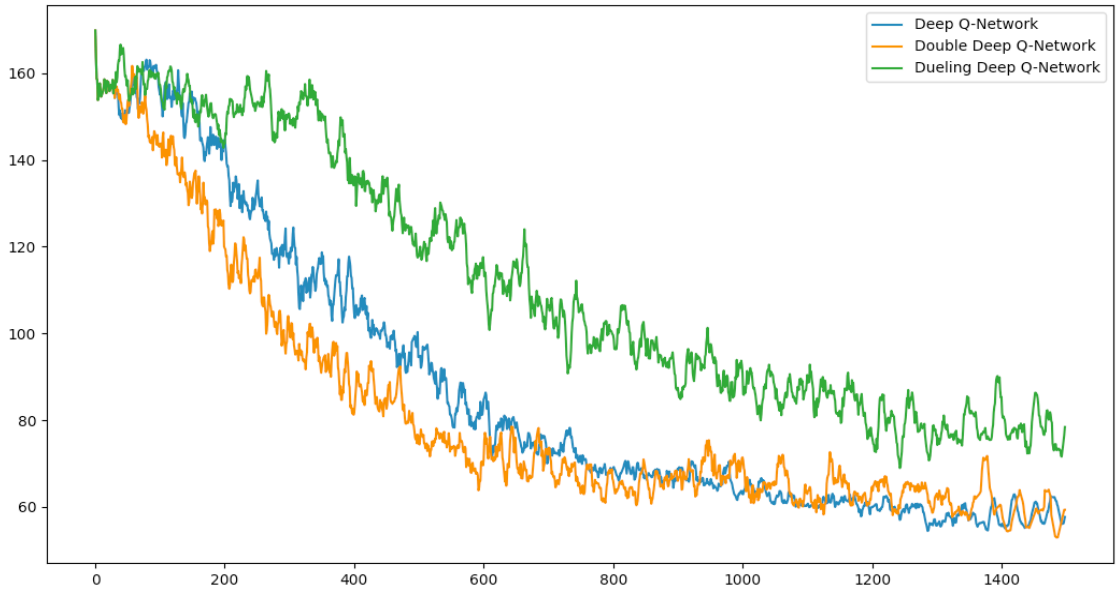
Αλγοριθμικά στον Duelling DQN το ίδιο νευρωνικό δίκτυο χωρίζει το τελευταίο του επίπεδο σε δύο μέρη.

- Το πρώτο μέρος: Έχει μια έξοδο V και πραγματοποιεί την συνολική εκτίμηση της συνάρτηση value-state για την κατάσταση s.
- Το δεύτερο μέρος : Έχει έξοδούς όσες και ο αριθμός τον ενεργειών και η εκτίμηση A(s, a) πραγματοποιείται για κάθε διαθέσιμη ενέργεια.
- Στο τέλος θα επιστέψει τις εκτιμήσεις A(s, a) αλλά και τις τιμές Q για κάθε ενέργεια, συνδυάζοντας τα δύο μέρη σε μια ενιαία έξοδο.

$$Q_n = V + (A_n - avg\_A) \text{ για } n \in Actions$$

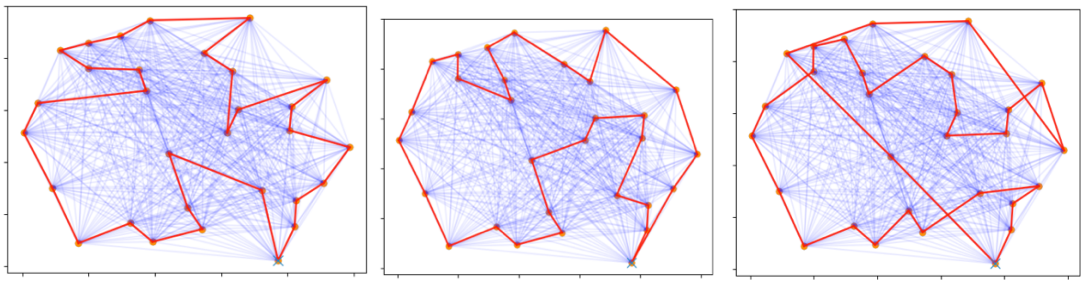
Deep Q-Network vs Double Deep Q-Network vs Duelling Deep Q-Network

Ως είσοδος δόθηκε ένα πλήρης γράφημα 30 κόμβων και εκπαιδευτήκαν για 1500 επεισόδια. Στην Εικόνα 5.7 παρουσιάζεται το διάγραμμα σύγκρισης με βάση το μήκος της διαδρομής που επιστρέφει ο κάθε αλγόριθμος ανά επεισόδιο.



Εικόνα 24 Διάγραμμα σύγκρισης διαδρομής.

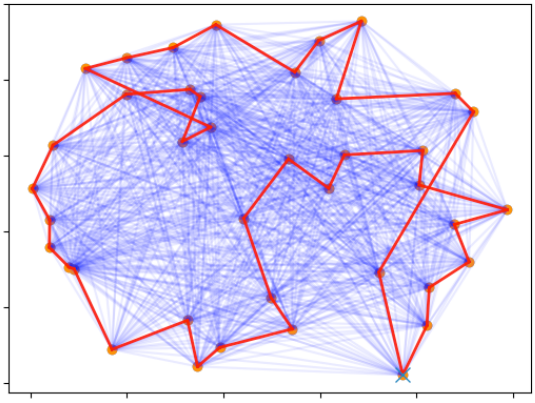
Ελάχιστη διαδρομή DQN	Ελάχιστη διαδρομή DDQN	Ελάχιστη διαδρομή Duelling DQN
50.4	49.9	60.4

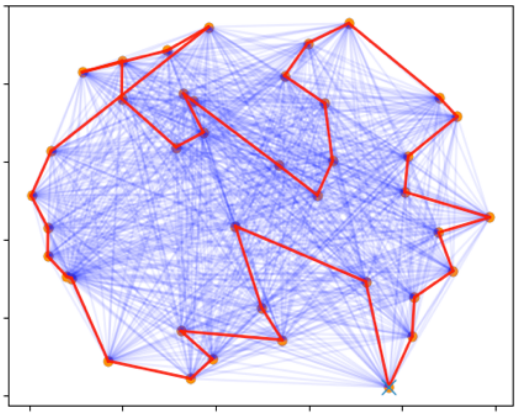


Η Double Deep Q-Network αρχιτεκτονική φαίνεται να υπερτερεί οριακά της Deep Q-Network, όμως οι διάφορες που παρουσιάζουν στο διαγράμματα εκπαίδευσης και στις διαδρομές είναι πολύ μικρές. Από την άλλη η Duelling Deep Q-Network συγκλίνει με την πάροδο των επεισοδίων σε διαδρομές μικρότερου μήκους όμως αποδοτικά υστερεί.

Το πείραμα επαναλήφθηκε για γράφημα 40 κόμβων με παρόμοια αποτελέσματα.

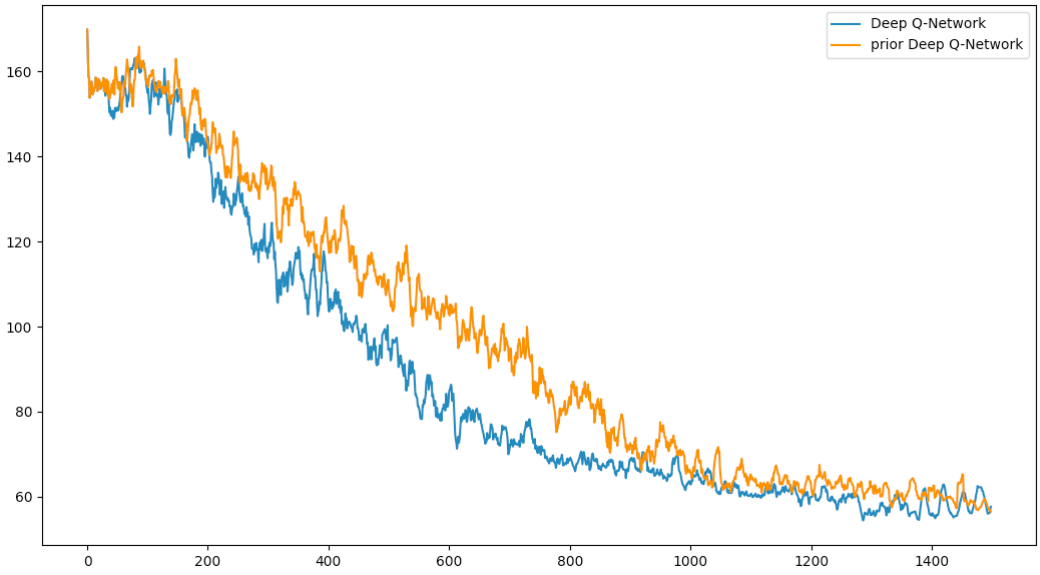
Διαδρομή DQN	Διαδρομή DDQN
56.6	56.2





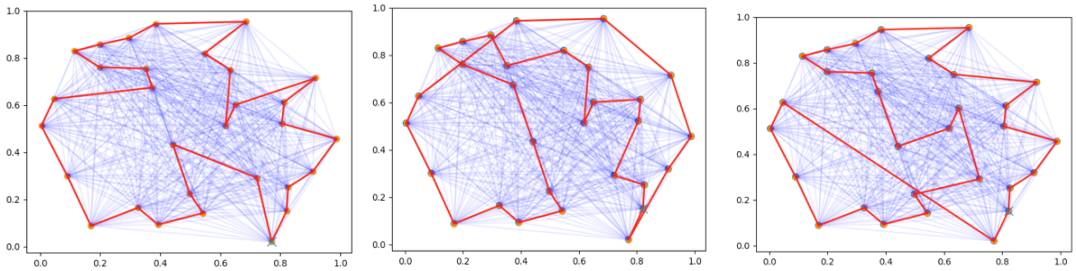
Deep Q-Network vs Prioritized experience Deep Q-Network

Σε μια δεύτερη δοκιμή τροποποιήθηκαν οι δομές αποθήκευσης της DQN σύμφωνα με την Prioritized experience προσέγγιση και συγκρίθηκε με το πρωτότυπο DQN. Ως είσοδος δόθηκε επίσης ένας πλήρης γράφημα 30 κόμβων και εκπαιδεύτηκαν για 1500 επεισόδια. Στην Εικόνα 5.8 παρουσιάζεται το διάγραμμα σύγκρισης του μήκους της διαδρομής.



Εικόνα 25: Διάγραμμα σύγκρισης διαδρομής.

Deep Q-Network:	Prioritized experience DQN	Greedy tsp
50.4	54.6	56.1



Η αλλαγή δομής όπως φαίνεται και στο διάγραμμα εκπαίδευσης, αλλά και στο ελάχιστο μονοπάτι δεν βελτίωσε την αποδοτικότητα του αλγορίθμου.

Παρατηρήσεις

Κάθε γράφημα στον οποίο εφαρμόζεται ο αλγόριθμος αποτελεί ένα καινούργιο περιβάλλον, ως εκ τούτου οι παράμετροι θα πρέπει να διαφοροποιηθούν, καθώς τόσο το μέγεθος όσο και η φύση του προβλήματος αλλάζει και όπως είδαμε και στις δοκιμές οι αλγόριθμοι παρουσιάζουν ευαισθησία ως προς την ρύθμιση των παραμέτρων, μια ευαισθησία που μπορεί να οδηγήσει σε τοπικά μέγιστα.

Παράμετροι

- Το μέγεθος της μνήμης
- Οι επιβραβεύσεις
- Το μέγεθος του minibatch
- Ο ρυθμός μείωσης τυχαίων ενεργειών(e),
- Το ποσοστό έκπτωσης(γ)
- Ο ρυθμός μάθησης (r)
- Αριθμός επεισοδίων εκπαίδευσης

Εξίσου σημαντική στην περίπτωση της βαθιάς ενισχυτικής μάθησης είναι η δομή του νευρωνικού δικτύου καθώς αυτό καλείται να επιλέξει τις ενέργειες.

Παράμετροι νευρωνικού δικτύου

- Κρυφά επίπεδα
- Νευρώνες
- Συναρτήση ενεργοποίησης
- Συνάρτηση σφάλματος
- Βελτιστοποιητές

Η Deep Q-Network παρουσιάζει συχνά, κυρίως σε μεγάλα γραφήματα, το πρόβλημα της υπερεκτίμησης, δηλαδή το δίκτυο κατά την αρχή της εκπαίδευσης προβλέπει τιμές πολύ μεγαλύτερες από τις αναμενόμενες. Για παράδειγμα κατά τις πρώτες εκπαιδεύσεις και ενώ τα reward παίρνουν τιμές από 0 έως 12, το νευρωνικό επιστρέφει τιμές της τάξεως του 10^6 καθιστώντας έτσι την εκπαίδευση αδύνατη.

Λύσεις του προβλήματος της υπερεκτίμησης.

- Μείωση του ποσοστού έκπτωσης(γ), παρατηρήθηκε πως ένα gamma κοντά στο 0.5 αντιμετωπίζει το πρόβλημα.
- Χρήση της Double Deep Q-Network αρχιτεκτονικής η οποίας δημιουργήθηκε για να αντιμετωπίζει το συγκεκριμένο πρόβλημα

Κεφάλαιο 6

Σε αυτό το κεφάλαιο παρατίθενται οι δομές που χρησιμοποιούνται και περιγράφεται ο κώδικας που υλοποιήθηκε για την κατασκευή των αλγορίθμων ενισχυτικής μάθησης.

Εφαρμογή Ι: Εύρεση βέλτιστης εξόδου σε λαβύρινθο

Neighbors: Πίνακας στον οποίο είναι αποθηκευμένες οι πιθανές ενέργειες για κάθε κελί.

Reward: Πίνακας ανταμοιβών για κάθε πιθανή ενέργεια, -1 για κάθε βήμα,+10 για τον προορισμό.

Done: Πίνακας τελικών καταστάσεων.

Class Agent: Δημιουργεί και εκπαιδεύει τον agent.

Πεδία

learning rate: Ο ρυθμός μάθησης ο οποίος χρησιμοποιείται από την *Bellman Optimality Equation.*

discount: Το ποσοστό έκπτωσης, δηλαδή την βαρύτητα που έχουν οι μελλοντικές τιμές του Q-table.

Epsilon: Αριθμός βάση του οποίου αποφασίζεται αν θα πραγματοποιηθεί τυχαία ενέργεια.

Epsilon_decay: Ο ρυθμός μείωσης του epsilon έτσι ώστε να μειωθούν και οι τυχαίες ενέργειες.

Epsilon_min: Ένας πολύ μικρός αριθμός που χρησιμοποιείται ως όριο στην τιμή του epsilon. Σκοπός του είναι να πραγματοποιείται με μικρή συχνότητα κάποια τυχαία ενέργεια εξερεύνησης, ακόμα και όταν ο αλγόριθμος έχει συγκλίνει

Μέθοδοι

- *Build_Q_table:* Κατασκευάζει το Q-table ο οποίος αρχικοποιείται στο 0 ενώ ενημερώνονται μόνο θέσεις από τις οποίες μπορούν να πραγματοποιηθούν ενέργειες.
- *Action:* Επιλεγεί μια ενέργεια είτε τυχαία είτε με βάση το *Q-table και μειώνει το epsilon.*
- *Update_Q*: Ενημερώνει του *Q-table* σύμφωνα με την *Bellman Optimality Equation*

- *Train:* Για έναν συγκεκριμένο αριθμό επεισοδίων εκπαιδεύει τον Agent (ενημερώνει το Q-table)
- *Find_path:* Καλείται μόλις τελειώσει η εκπαίδευση, επιστρέφει την τελική διαδρομή.

Κώδικας agent

```
class RLAgent:
    def __init__(self,nodes,start,end):
        self.end=end
        self.learning_rate=0.8
        self.discount=0.8
        self.start=start
        self.exploration_treshold=1
        self.Q_table=self.build_Q_table()

    def build_Q_table(self):
        Q_table= np.zeros((Size,Size_2))
        return Q_table

    def Action(self,current):
        decision = random.random()
        if decision<self.exploration_treshold:
            available_actions=neighbors[current] #geitones tou current
        else:
            available_actions_index = [index for index, val in enumer-
ate(self.Q_table[current]) if val == np.max(self.Q_table[current])]
            available_actions=neighbors[current,available_actions_index]
        next_cell=int(np.random.choice(available_actions))

        return next_cell

    def update_Q(self,node1,node2):
        highst_Q_value_indexes = [index for index, val in enumer-
ate(self.Q_table[node2]) if val == np.max(self.Q_table[node2])]
        if len(highst_Q_value_indexes)>1:
            max_index=int(np.random.choice(highst_Q_value_indexes))
        else:
            max_index=highst_Q_value_indexes
        max_value=self.Q_table[node2,max_index]
        node2_index=np.where(neighbors[node1]==node2)
        self.Q_table[node1,node2_index]=(1-self.learning_rate)*self.Q_ta-
ble[node1,node2_index]+self.learning_rate*(Reward[node2]+self.dis-
count*max_value)

    def train(self):
        episodes=2000
        for e in range(episodes):
            current=0
            if e==50 or e==500 or e==1000:
                print(e,self.exploration_treshold)
                print(self.Q_table)
                print()
            if self.exploration_treshold>0.01:
                self.exploration_treshold=self.exploration_treshold*0.999
            for i in range(100):
                next=self.Action(current)
                self.update_Q(current,next)
                if done[next]==1:
                    break
                current=next
```

Εφαρμογή ΙΙ: Εύρεση ελάχιστου μονοπατιού

Η υλοποίηση είναι παρόμοια με την Εφαρμογή Ι

Class Agent: Δημιουργεί και εκπαιδεύει τον agent, ορίζονται τα πεδία τα learnig rate, discount, epsilon, epsilon min και εμπεριέχει τις μεθόδους:

- *Build_Graph:* Δημιουργεί ένα μη κατευθυνόμενο γράφημα με τυχαία βαρύ με την χρήση της βιβλιοθήκης networkx
- *Build_Reward:* Κατασκευάζει τον πίνακα Reward
- *Build_Q_table:* Κατασκευάζει το Q-table, οπού υπάρχει ακμή-ενέργεια η αντιστοιχεί τιμή του κόμβου που οδηγεί η ακμή γίνεται 0 ενώ όλες οι άλλες θέσεις -100
- *Action:* Επιλεγεί μια ενέργεια είτε τυχαία είτε με βάση το *Q-table και μειώνει το Epsilon.*
- *Update_Q*: Ενημερώνει των *Q-table* σύμφωνα με την *Bellman Optimality Equation*
- *Train:* Για έναν συγκεκριμένο αριθμό επεισοδίων εκπαιδεύει τον Agent (ενημερώνει το Q-table)
- *Find_path:* Καλείται μόλις τελειώσει η εκπαίδευση και επιστρέφει την τελική διαδρομή.

Κώδικας agent

```
class SPAgent:
    def __init__(self,nodes,start,end):
        self.nodes=nodes
        self.end=end
        self.learning_rate=0.9
        self.discount=0.9
        self.start=start
        self.exploration_treshold=1
        self.Graph=self.build_Graph()
        self.Q_table=self.build_Q_table()
        self.Reward=self.build_Reward()
    def build_Graph(self):
        #seed=44,randint(2,30)nodes=185,50,start=0,end=135
        G = nx.random_regular_graph(4,self.nodes,seed=44)
        i=0
        for (u, v) in G.edges():
            random.seed(i*3)
            G.edges[u,v]['weight']=random.randint(2,30)
            i+=1
        return G

    def Display_Graph(self):
        plt.figure(figsize=(8, 8))
        pos=nx.spring_layout(self.Graph)
        nx.draw_networkx_nodes(self.Graph,pos)
```

```
        nx.draw_networkx_edges(self.Graph,pos)
        nx.draw_networkx_labels(self.Graph,pos)
        nx.draw_networkx_edge_labels(self.Graph,pos)
        plt.show()

    def build_Reward(self):
        reward= np.zeros((self.nodes,self.nodes))

        for (u, v) in self.Graph.edges():
            reward[u,v]=-(self.Graph.edges[u,v]['weight'])
            reward[v,u]=-(self.Graph.edges[u,v]['weight'])

        for x in self.Graph[self.end]:
            # reward[x,self.end]=10
            reward[self.end,x]=100
        return reward

    def build_Q_table(self):
        Q_table= np.zeros((self.nodes,self.nodes))
        for i in range(self.nodes):
            for j in range(self.nodes):
                Q_table[i][j]=0

        available_actions=list()
        for node in self.Graph.nodes:
            n=list()
            for neighbour in self.Graph[node]:
                Q_table[node,neighbour]=0
                Q_table[neighbour,node]=0
        for x in self.Graph[self.end]:
            Q_table[self.end,x]=100
        return Q_table

    def Action(self,current):
        decision = random.random()
        if decision<self.exploration_treshold:
            available_actions=self.Graph[current] #geitones tou current
            next_node=int(np.random.choice(available_actions))
        else:
            available_actions=self.Graph[current] #geitones tou current
            available_actions = np.array(available_actions)
            best_action_index = self.Q_table[current][available_ac-
tions].argmax()    #apo tous diathesimoyw auton me to max q
            # print('best_action_index',best_action_index)

            next_node=available_actions[best_action_index]
            # print("next_node",next_node)
        return next_node

    def update_Q(self,node1,node2):
        highst_Q_value_indexes = [index for index, val in enumer-
ate(self.Q_table[node2]) if val == np.max(self.Q_table[node2])]
        if len(highst_Q_value_indexes)>1:
            max_index=int(np.random.choice(highst_Q_value_indexes))
        else:
            max_index=highst_Q_value_indexes
        max_value=self.Q_table[node2,max_index]
        self.Q_table[node1,node2]=(1-self.learning_rate)*self.Q_ta-
ble[node1,node2]+self.learning_rate*(self.discount*self.Re-
ward[node1,node2]+max_value)

    def train(self):
        All_path_size=list()
        episodes=300
        for e in range(episodes):
            print(e)
            start=0
            sum_path=0
```

```
            if self.exploration_treshold>0.01:
                self.exploration_treshold=self.exploration_treshold*0.985
            for s in range(1200):

                next_node=self.Action(start)

                self.update_Q(start,next_node)
                sum_path+=self.Graph.edges[start,next_node]['weight']
                if next_node==self.end:
                    All_path_size.append([e,sum_path])
                    break
                start=next_node

        print(self.exploration_treshold)
        return All_path_size
```

Εφαρμογή ΙΙΙ: Εύρεση βέλτιστης διαδρομής TSP

Οι συναρτήσεις έχουν παρόμοια χρήση με τις προαναφερθείσες εφαρμογές με τις εξής διαφοροποιήσεις:

Build_Q_table: Κατασκευάζει και αρχικοποιεί το Q-table σε 0, καθώς πλέον βρισκόμαστε σε πλήρη γράφημα και όλες οι ενέργειες είναι πιθανές.

Available_actions: Είναι η λίστα στην οποία είναι αποθηκευμένοι όλοι οι διαθέσιμοι προς επίσκεψη κόμβοι. Ενημερώνεται μετά από κάθε ενέργεια, κάθε επεισόδιο σταματάει όταν δεν υπάρχουν πλέον διαθέσιμοι κόμβοι.

Κώδικας agent

```
class TSPagent:
    def __init__(self,nodes,start,end=0):
        self.nodes=nodes
        self.end=end
        self.learning_rate=0.3
        self.discount=0.9
        self.start=start
        self.epsilon=1
        self.Graph=self.build_Graph()
        self.Q_table=self.build_Q_table()
        self.Reward=self.build_Reward()

    def build_Graph(self):
        G = nx.complete_graph(self.nodes)
        i=0
        for (u, v) in G.edges():
            G.edges[u,v]['weight']=random.randint(4,10)
            i+=1
        return G

    def Display_Travel(self,path):
        plt.figure(figsize=(8, 8))
        pos=nx.spring_layout(self.Graph)
        nx.draw_networkx_nodes(self.Graph,pos)
        nx.draw_networkx_labels(self.Graph,pos)
        # nx.draw_networkx_edge_labels(self.Graph,pos)
        # nx.draw_networkx_edges(self.Graph,pos)
```

```
        labels = {}
        j=1
        for i in path:
            i=tuple(i)
            # print(i)
            labels[i] = self.Graph.edges[i[0],i[1]]['weight']
            j+=1
            # print(labels)
        nx.draw_networkx_edges(
            self.Graph,
            pos,
            edgelist=path,
            width=3,
            alpha=0.8,
            edge_color="tab:red",
        )
        nx.draw_networkx_edge_labels(self.Graph, pos, labels, font_size=8,
font_color="black")
        plt.show()
        # plt.show()

    def build_Reward(self):
        reward= np.zeros((self.nodes,self.nodes))
        for (u, v) in self.Graph.edges():
            # reward[u,v]=1/self.Graph.edges[u,v]['weight']
            # reward[v,u]=1/self.Graph.edges[u,v]['weight']
            reward[u,v]=12-(self.Graph.edges[u,v]['weight'])
            reward[v,u]=12-(self.Graph.edges[u,v]['weight'])
        return reward

    def build_Q_table(self):
        Q_table= np.zeros((self.nodes,self.nodes))
        return Q_table

    def Action(self,current,available_actions):
        # print("available_actions",available_actions)
        decision = random.random()
        if decision<self.epsilon:
            next_node=random.choice(available_actions)
            # print(available_actions,"->random->",next_node)
        else:
            best_action_index = self.Q_table[current, available_ac-
tions].argmax()#apo tous diathesimoyw auton me to max q
            next_node=available_actions[best_action_index]
            # print('best',next_node)
        return next_node

    def update_Q(self,node1,node2):
        max_value=np.max(self.Q_table[node2])
        self.Q_table[node1,node2]=(1-self.learning_rate)*self.Q_ta-
ble[node1,node2]+self.learning_rate*(self.Reward[node1,node2]+self.dis-
count*max_value)

    def train_salesman(self):
        All_path_size=list()
        episodes=400*self.nodes
        for e in range(episodes):
            current=self.start
            sum_path=0
            i=0
            Travel_list=list()
            Path_edges_for_plot=list()
            Travel_list.append(current)
            available_actions=list()
            path_reward=0
            all_Reward_list=list()
            for no in range(self.nodes):
                if no not in Travel_list:
```

```
            available_actions.append(no)
        while available_actions and i<self.nodes+2:
            next_node=self.Action(current,available_actions)
            # print(current,next_node)
            path_reward+=self.Reward[current,next_node]
            self.update_Q(current,next_node)
            sum_path+=self.Graph.edges[current,next_node]['weight']
            Travel_list.append(next_node)
            Path_edges_for_plot.append([current,next_node])
            current=next_node
            available_actions.remove(current)

        if self.epsilon>0.0005:
            self.epsilon-=1.15/episodes
        next_node=self.start
        sum_path+=self.Graph.edges[current,next_node]['weight']
        path_reward+=self.Reward[current,next_node]
        All_path_size.append(sum_path)
        self.update_Q(current,next_node)
        Travel_list.append(next_node)

    # print(self.epsilon)
    return All_path_size,Travel_list,sum_path
```

Εφαρμογή IV: Εύρεση βέλτιστης διαδρομής TSP με Deep-Q-learning.

Build_Graph: Δημιουργεί το γράφημα

Reward_funcion: Δημιουργεί τις επιβραβεύσεις

Build_Model: Δημιουργεί τα νευρωνικά δίκτυα με την χρήση της keras βιβλιοθήκης

Class DQNAgent: Εκπαιδεύει τον agent.

Πεδία

***state_size*:** Το μέγεθος του state που λαμβάνει ως είσοδο το NN

***action_size*:** Το πλήθος των ενεργειών-έξοδοι του NN

memory : Το μέγεθος της μνήμη(deque) αποθήκευσης μεταβάσεων

gamma*:* Το ποσοστό έκπτωσης,

epsilon, epsilon_min, epsilon_decay : Ίδια με τις προηγούμενες εφαρμογές

***batch_size*:** Μέγεθος του batch που θα χρησιμοποιηθεί για την εκπαίδευση

train_start : Το Μέγεθος της μνήμης από το οποίο θα αρχίσει να πραγματοποιείται η εκπαίδευση

Model: Το μοντέλο του νευρωνικού που θα χρησιμοποιηθεί.

Τα παρακάτω πεδία χρησιμοποιούνται μόνο *κατά την* ***double deep Q*-network** μέθοδο.

Soft_Update: Boolean μεταβλητή καθορίζει τον τύπο του update που θα πραγματοποιηθεί

TAU: Ποσοστό των βαρών που θα γίνουν update

target_model: Το μοντέλο του δευτερεύον νευρωνικού δικτύου

Μέθοδοι

- *Remember:* Αποθηκεύει μια εγγραφή στην memory με την μορφή [state,action,next_state,reward]
- update_target_model: Καλείται μόνο κατά την χρήση της Double DQN αρχιτεκτονικής για να συγχρονίζει τα βάρη του κύριου NN με το δευτερεύον. Έχει δυο τρόπους update τον soft και τον hard.
- Act: Επιλέγει μια ενέργεια είτε τυχαία είτε με βάση το *Q-table*
- Replay: Αυτή η μέθοδος πραγματοποιεί την τυχαία δειγματοληψία από την memory(minbatch), μειώνει το Epsilon και εκπαιδεύει το NN με βάση το minibatch.
- Train: Είναι η κύρια μέθοδος που καλείται για να υλοποιηθεί η εκπαίδευση του agent. Αποθηκεύει τα NN που φέρνουν τα καλυτέρα αποτελέσματα.
- Test: Μόλις ολοκληρωθεί η εκπαίδευση φορτώνει το αποθηκευμένο δίκτυο και επιστρέφει το κόστος της διαδρομής.

Κώδικας agent

```
class DQNAgent:
    def __init__(self):
        self.state_size = Nodes+1#8
        self.action_size = Nodes#4
        self.memory = deque(maxlen=2000)#4500
        self.gamma = 0.8 # discount rate
        self.epsilon = 1.0      # exploration rate
        self.epsilon_min = 0.001    #5/1000 θα κανει explore
        self.epsilon_decay = 0.997 #9998
        self.batch_size = 16
        self.train_start=1000

        self.Soft_Update = False
        self.TAU = 0.1 # target network soft update hyperparameter
        self.model = OurModel(input_shape=self.state_size,
                              action_space = self.action_size)
        self.target_model = OurModel(input_shape=self.state_size,
                                     action_space = self.action_size)

    def remember(self, state, action, reward, next_state,done,
                                              available_actions,e):
        available_actions = np.array(available_actions)

        self.memory.append((state, action, reward, next_state, done, availa-
ble_actions,e))
        if len(self.memory) > self.train_start
```

```
            if self.epsilon > self.epsilon_min:
                self.epsilon *= self.epsilon_decay

    def update_target_model(self):
        if not self.Soft_Update:
            self.target_model.set_weights(self.model.get_weights())
            return
        if self.Soft_Update:
            q_model_theta = self.model.get_weights()
            target_model_theta = self.target_model.get_weights()
            counter = 0
            for q_weight, target_weight in zip(q_model_theta,
                                               target_model_theta):
                target_weight = target_weight * (1-self.TAU)
                                               + q_weight * self.TAU
                target_model_theta[counter] = target_weight
                counter += 1
            self.target_model.set_weights(target_model_theta)

    def act(self, state, available_actions,l):
        if np.random.rand() <= self.epsilon:    #explore
            return random.choice(available_actions)
        else:
            act_values = self.model.predict(state,verbose = 0)
            # print("episode=",l)
            # print("state=",state)
            # print("act_values=",act_values)
            act_values  = act_values[0]
            best_action_index = act_values[available_
                                                   actions].argmax()q
            best_action=available_actions[best_action_index]
            return best_action

    def replay(self):
        if len(self.memory) < self.train_start:
            return
        # Randomly sample minibatch from the memory
        minibatch = random.sample(self.memory, min(len(self.memory),
self.batch_size))
        state = np.zeros((self.batch_size, self.state_size))
        next_state = np.zeros((self.batch_size, self.state_size))
        action, reward, done_list, available_actions, episodes= [], [], [],
[], []
        for i in range(self.batch_size):
            state[i] = minibatch[i][0]  #array
            action.append(minibatch[i][1])  #list
            reward.append(minibatch[i][2])
            next_state[i] = minibatch[i][3]
            done_list.append(minibatch[i][4])
            available_actions.append(minibatch[i][5])
            # episode.append(minibatch[i][6])

        target =self.model.predict_on_batch(state)
        target_next =self.model.predict_on_batch(next_state)
        target_val = self.target_model.predict_on_batch(next_state)

        for i in range(self.batch_size):
            if action[i]==0:
                if done_list[i]==1:
                    target[i][action[i]] = reward[i]*3
                else :
                    target[i][action[i]] = reward[i]
            else:
                act_values  = target_next[i]
                best_predict_action_value=np.amax(act_values[available_ac-
tions[i]])
                target[i][action[i]] = reward[i] + self.gamma * best_pre-
dict_action_value
```

```
                #DDQL
                # best_predict_action_index= np.argmax(act_values[availa-
ble_actions[i]])
                # target[i][action[i]] = reward[i] + self.gamma * (tar-
get_val[i][best_predict_action_index])

                # a = np.argmax(target_next[i])
                # target[i][action[i]] = reward[i] + self.gamma * (tar-
get_val[i][a])

        self.model.train_on_batch(state, target)
```

Συμπεράσματα

Στην παρούσα εργασία μελετήθηκαν αλγόριθμοι ενισχυτικής μάθησης και βαθιάς ενισχυτικής μάθησης και το πως μπορούν να εφαρμοστούν σε προβλήματα βελτιστοποίησης. Πραγματοποιήθηκε υλοποίηση και πειραματική δοκιμή αλγορίθμων Q-learning και deep Q-learning , καθώς και σύγκριση με παραδοσιακούς αλγορίθμους, στα πλαίσια της οποίας τα αποτελέσματα των αλγορίθμων της ενισχυτικής μάθησης συγκρίθηκαν με τα αντίστοιχα του αλγορίθμου dijkstra έτσι ώστε να ελεγχθούν ως προς την ορθότητα αλλά και με του greedy TSP έτσι ώστε να ελεγχθούν ως προς την αποδοτικότητα, φέροντας πολλές φορές καλύτερα αποτελέσματα.

Όπως παρατηρήθηκε από τα πειράματα, πρωταρχικό ρολό στην απόδοση των αλγορίθμων έχει η θέσπιση των καταλλήλων τιμών στις μεταβλητές τόσο στην ενισχυτική μάθηση (ποσοστό εκμάθησης (α), συντελεστής έκπτωσης (γ)) όσο και στην βαθιά ενισχυτική (παράμετροι νευρωνικού δικτύου). Παρατηρήθηκε ότι αλλαγές στις μεταβλητές σε συνδυασμό με την φύση του προβλήματος ενδέχεται να φέρουν διαφορετικά αποτελέσματα. Η διαδικασία θέσπισης των καταλλήλων μεταβλητών είναι πολυπαραγοντική και θα μπορούσε να αποτελέσει μελλοντικό θέμα διερεύνησης, καθώς πρόκειται για ένα σύγχρονο τομέα της μηχανικής μάθησης με πρόσφατη βιβλιογραφία και πολλά περιθώρια έρευνας.

Παρόλα αυτά παρουσιάζει ταχεία ανάπτυξη, γνωρίζοντας μεγάλη δημοφιλία σε πληθώρα εφαρμογών, όπως η αυτόνομη οδήγηση και η ρομποτική, όντας εξαιρετικά χρήσιμη σε προβλήματα τα οποία είναι πολύ δύσκολο να περιγράφει επαρκώς η βέλτιστη συμπεριφορά. Επιπλέον σε αντίθεση με την επιβλεπόμενη μάθηση, οπού το σύστημα μαθαίνει από κάποιον δάσκαλο, θέτοντας έτσι όρια στις δυνατότητες του, στην μάθηση με ενίσχυση το σύστημα μαθαίνει αλληλοεπιδρώντας με το περιβάλλον, μια ιδιότητα που το καθιστά αποδοτικότερο σε προβλήματα βελτιστοποίησης.

Βιβλιογραφία

1) *Russell, Stuart J. (2002). Artificial intelligence : a modern approach. New Delhi: Prentice-Hall of India*
2) *History of Reinforcement Learning. (n.d.). Retrieved October 3, 2022, from http://incompleteideas.net/book/ebook/node12.html*
3) *Mwiti, D. (2022, July 21). 10 Real-Life Applications of Reinforcement Learning. Neptune.Ai. Retrieved September 18, 2022, from https://neptune.ai/blog/reinforcement-learning-applications*
4) *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction.*
5) *Reinforcement Learning: Bellman Equation and Optimality*
6) *Exploration vs. Exploitation - Learning the Optimal Reinforcement Learning Policy. (n.d.). Deeplizard. Retrieved October 3, 2022, from https://deeplizard.com/learn/video/mo96Nqlo1L8*
7) *Wikipedia contributors. (2022, September 29). Markov decision process. Wikipedia. Retrieved October 3, 2022, from https://en.wikipedia.org/wiki/Markov_decision_process*
8) *Wikipedia contributors. (2022a, September 13). Artificial neural network. Wikipedia. Retrieved October 3, 2022, from https://en.wikipedia.org/wiki/Artificial_neural_network*
9) *Athaiya A., Sharma S. & Sharma S. (2020) Activation Functions in Neural Networks, International Journal of Engineering Applied Sciences and Technology (Vol 4. Issue 12, pp. 310-316).*
10) *RMSProp - Cornell University Computational Optimization Open Textbook - Optimization Wiki. (n.d.). Retrieved October 3, 2022, from https://optimization.cbe.cornell.edu/index.php?title=RMSProp*
11) *Adam - Cornell University Computational Optimization Open Textbook - Optimization Wiki. (n.d.). Retrieved October 3, 2022, from https://optimization.cbe.cornell.edu/index.php?title=Adam11 Deep Q-learning: https://www.mlq.ai/deep-reinforcement-learning-q-learning/*
12) *Foy, P. (2022, August 6). Deep Reinforcement Learning: Guide to Deep Q-Learning. MLQ.ai. Retrieved October 3, 2022, from https://www.mlq.ai/deep-reinforcement-learning-q-learning/*
13) *Vitay, J. (n.d.-b). Deep Reinforcement Learning. Retrieved September 18, 2022, from https://julien-vitay.net/deeprl/Valuebased.html*
14) *H. Van Hasselt, A. Guez, and D. Silver. (2016) Deep reinforcement learning with double q-learning. In Thirtieth AAAI conference on artificial intelligence, arXiv:1509.06461v3.*
15) *prioritized-experience-replay: Schaul et al. - Prioritized Experience Replay, arXiv:1511.05952, 2015*
16) *Rivlin, O. (2021, December 9). Reinforcement Learning for Combinatorial Optimization. Medium. Retrieved October 3, 2022, from https://towardsdatascience.com/reinforcement-learning-for-combinatorial-optimization-d1402e396e91*

17) *Mnih et al. - Human-level control through deep reinforcement learning, Nature 518, 2015*
18) *Introduction to. (n.d.). TensorFlow. Retrieved October 3, 2022, from https://www.tensorflow.org/learn18*
19) *NetworkX — NetworkX documentation. (n.d.). https://networkx.org/*
20) *Gym Documentation. (n.d.). Retrieved October 3, 2022, from https://www.gymlibrary.dev*
21) *Cart Pole - Gym Documentation. (n.d.). Retrieved October 3, 2022, from https://www.gymlibrary.dev/environments/classic_control/cart_pole/*
22) *Cart Pole - Gym Documentation. (n.d.). Retrieved October 3, 2022, from https://www.gymlibrary.dev/environments/classic_control/cart_pole/*

ΕΙΚΟΝΕΣ

1) *Markov Decision Processes (MDPs) - Structuring a Reinforcement Learning Problem. (n.d.). Deeplizard. Retrieved October 3, 2022, from* https://deeplizard.com/learn/video/my207WNoeyA
2) *Manning Publications. (2019, July 27). Neural Network Architectures. Manning. Retrieved October 3, 2022, from* https://freecontent.manning.com/neural-network-architectures/
3) *Manning Publications. (2019, July 27). Neural Network Architectures. Manning. Retrieved October 3, 2022, from* https://freecontent.manning.com/neural-network-architectures/
4) *Ring, R. (2020, January 13). Deep Reinforcement Learning With TensorFlow 2.1. Roman Ring. Retrieved October 3, 2022, from* http://inoryy.com/post/tensorflow2-deep-reinforcement-learning/
5) *Rivlin, O. (2021, December 9). Reinforcement Learning for Combinatorial Optimization. Medium. Retrieved October 3, 2022, from https://towardsdatascience.com/reinforcement-learning-for-combinatorial-optimization-d1402e396e91*