Ατομική Διπλωματική Εργασία

ΓΕΝΙΚΕΥΜΕΝΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ ΓΙΑ ΤΕΧΝΗΤΟΥΣ ΠΑΙΚΤΕΣ TETRIS

Χρύσης Ευτυχίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Γενικευμένη αρχιτεκτονική νευρωνικών
δικτύων για τεχνητούς παίκτες Tetris

Χρύσης Ευτυχίου

Επιβλέπων Καθηγητής
Χριστοδούλου Χρίστος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Δρ. Χρίστο Χριστοδούλου, καθώς και τον Δρ. Βασίλη Βασιλειάδη που μου έδωσαν την ευκαιρία να εκπονήσω τη συγκεκριμένη διπλωματική εργασία. Με τη βοήθεια και τη σωστή καθοδήγηση που μου παρείχαν, καθώς και με την υπομονή που έχουν υποδείξει, κατάφερα να φέρω εις πέρας μεγάλο κομμάτι της εργασίας.

Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου για την ψυχολογική και οικονομική υποστήριξη που μου παρείχαν καθ' όλη την διάρκεια των σπουδών μου.

Περίληψη

Στόχος της διπλωματικής μου εργασίας είναι η υλοποίηση μια νέας αρχιτεκτονικής νευρωνικών δικτύων στοχευμένη για τεχνητούς παίκτες του Tetris αλλά που θα μπορεί να γενικευτεί και να χρησιμοποιηθεί και σε άλλα παιχνίδια τύπου πίνακα (game board), όπως είναι για παράδειγμα το 2048.

Η αρχιτεκτονική αυτή χρησιμοποιεί έναν one-piece controller και afterstate evaluator στο παιχνίδι Tetris και μαθαίνει πώς να αναλύει τον πίνακα του παιχνιδιού (low-level features) και να εξάγει high-level features παρόμοια με αυτά του Dellacherie-Thierry (Thiery & Scherrer, 2009). Η γενική ιδέα είναι η δημιουργία έξυπνων εξαγωγών χαρακτηριστικών με τη μορφή μιας γενικής παραμετροποιημένης μονάδας που θα μπορεί να βγάζει διαφορετικά χαρακτηριστικά ανάλογα με τις παραμέτρους που θα δεχθεί. Μέρος της έρευνας είναι η εύρεση και βελτιστοποίηση των παραμέτρων καθώς και των στοιχείων που θα συνδέουν την γενική αυτή μονάδα. Το δίκτυο θα μπορεί να μάθει χαρακτηριστικά υψηλού επιπέδου, καθώς και να βρει τον καλύτερο συνδυασμό (π.χ. χρησιμοποιώντας ένα χαρακτηριστικό αντί για ένα άλλο) για να επιτύχει το καλύτερο αποτέλεσμα.

Τα αποτελέσματα που πάρθηκαν είναι αισιόδοξα, αλλά ελλιπής για αυτό χρειάζεται περαιτέρω έρευνα και πειράματα χρησιμοποιώντας την συγκεκριμένη αρχιτεκτονική. Παρόμοιες έρευνες δεν εμφανίζονται συχνά στην βιβλιογραφία, για αυτό η επίτευξη καλών αποτελεσμάτων θα έχει μεγάλη σημασία για την παγκόσμια κοινότητα μηχανικής μάθησης στον τρόπο που σχεδιάζονται και υλοποιούνται κάποια νευρωνικά δίκτυα.

Περιεχόμενα

Κεφάλαιο 1

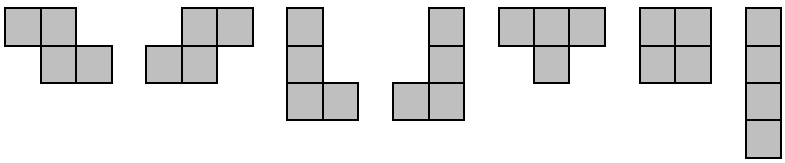
Εισαγωγή

1.1 Tetris

1.1.1 Γενικά

Το Tetris είναι ένα διάσημο tile-matching βιντεοπαιχνίδι παζλ, το οποίο αρχικά αναπτύχθηκε και σχεδιάστηκε από τον Ρώσο σχεδιαστή βιντεοπαιχνιδιών Αλεξέι Πάζιτνοβ. Το παιχνίδι κυκλοφόρησε στις 6 Ιουνίου του 1984, ενώ ο Πάζιτνοβ εργαζόταν στην Dorodnitsyn Computing Centre της Ρωσικής Ακαδημίας Επιστημών, στη Μόσχα. Η ονομασία του παιχνιδιού προήλθε από το ελληνικό πρόθημα "τέτρα", που σημαίνει τέσσερα, καθώς κι από το τένις, αγαπημένο άθλημα του Πάζιτνοβ. Το παιχνίδι (και πολυάριθμες εκδόσεις του) είναι διαθέσιμο σχεδόν σε οποιαδήποτε κονσόλα και λειτουργικό σύστημα υπολογιστή, καθώς και σε κινητά, σε graphing calculator, σε portable media player, σε PDA και σε network music player. Έχει χρησιμοποιηθεί έως έμπνευση μέχρι και για πιάτα, ενώ έχει παιχτεί κι επάνω σε διάφορα πραγματικά κτίρια.

Στόχος του παιχνιδιού είναι η τοποθέτηση τυχαίων γεωμετρικών σχημάτων πάνω σε ένα 10x20 πίνακα (game board) συμπληρώνοντας όσο περισσότερες γραμμές είναι δυνατόν. Αυτά τα σχήματα ονομάζονται ‘tetriminos’, βλέπε Σχήμα 2.2 (έτσι ονομάζονται τα ‘tetrominoes’ στο παιχνίδι αυτό), αποτελούνται από τέσσερα τετράγωνα συνδεδεμένα μαζί και πέφτουν από την κορυφή του πίνακα. Υπάρχουν επτά (7) tetriminos στο αυθεντικό παιχνίδι και παρομοιάζουν τα εξής λατινικά γράμματα : Z, S, L, J, T, O, I. Κάθε σχήμα μπορεί να περιστραφεί καθώς πέφτει και να μετακινηθεί από αριστερά προς δεξιά για να τοποθετηθεί στην επιθυμητή θέση. Με την τοποθέτηση ενός σχήματος, ένα νέο εμφανίζεται. Όταν μια γραμμή συμπληρώνεται, εξαφανίζεται και όλα τα σχήματα από πάνω της μετακινούνται κατά ένα κελί προς τα κάτω. Είναι δυνατό να συμπληρωθούν πολλαπλές γραμμές την ίδια στιγμή και όταν τέσσερις γραμμές συμπληρωθούν μαζί, η συγκεκριμένη κίνηση ονομάζεται ‘Tetris’ και είναι και η βάση του ονόματος του παιχνιδιού. Όταν τα σχήματα φτάσουν την κορυφή του πίνακα και δεν μπορούν νέα σχήματα να προστεθούν, τότε το παιχνίδι τελειώνει και η βαθμολογία του παίκτη είναι βασισμένη στον αριθμό των γραμμών που κατάφερε να συμπληρώσει.



Σχήμα 1.1: Τα 7 tetriminos και τα γράμματα που παρομοιάζουν Z, S, L, J, T, O, I (αριστερά προς δεξιά)

Δεν υπάρχει τρόπος να ‘Κερδίσεις’ το παιχνίδι ή να ‘παίζεις για πάντα’ καθώς ερευνητές έχουν καταλήξει ότι το παιχνίδι είναι στατιστικά καταδικασμένο να τελειώσει . Αυτό ευθύνεται στο γεγονός ότι μια μεγάλη ακολουθία από εναλλασσόμενα Z και S tetriminos αναπόφευκτα θα οδηγήσει σε ‘τρύπες’ στον πίνακα και αυτό με την σειρά του θα οδηγήσει τα σχήματα να γεμίζουν τον πίνακα χωρίς να συμπληρώνονται γραμμές και έτσι τερματίζεται το παιχνίδι (Burgiel, 1997). Παρόλα αυτά το Tetris ήταν και εξακολουθεί να αποτελεί ένα διάσημο παιχνίδι αναφοράς (benchmark) για αλγόριθμους τεχνητής νοημοσύνης.

1.1.2 Τεχνητοί παίκτες

Στην υπάρχουσα βιβλιογραφία (Bertsekas and Tsitsiklis 1996; Boumaza 2009; Kakade 2001) που αφορά τους τεχνητούς παίκτες για το Tetris, οι συγγραφείς συνήθως αναφέρονται σε μια απλοποιημένη έκδοση του προβλήματος όπου δεν λαμβάνεται υπόψη η δυναμική της πτώσης των σχημάτων. Σε αυτήν την απλοποιημένη εκδοχή ο τεχνητός παίκτης λαμβάνει το τρέχον σχήμα και την τρέχουσα κατάσταση και πρέπει να αποφασίσει πού να το τοποθετήσει. Μια απόφαση σε αυτή την περίπτωση είναι μια περιστροφή (ο προσανατολισμός του σχήματος) και μια μετάφραση (τη στήλη) για την εφαρμογή στο τρέχον σχήμα. Ο παίκτης ελέγχει όλες τα δυνατές περιστροφές και όλες τις πιθανές μεταφράσεις και αποφασίζει, με βάση μια συνάρτηση λήψης αποφάσεων, την καλύτερη απόφαση που θα επιφέρει το καλύτερο αποτέλεσμα. Αυτό ονομάζεται afterstate evaluator. Το καλύτερο αποτέλεσμα βγαίνει από την συνάρτηση εκτίμησης η οποία δίνει ψηλές τιμές σε καταστάσεις με ‘καλούς’ πίνακες και χαμηλές τιμές σε ‘κακούς΄. Στο σενάριο του Tetris καλοί πίνακες είναι αυτοί που αυξάνουν τις πιθανότητες να συμπληρωθούν γραμμές σε μελλοντικές κινήσεις. Παρεμπιπτόντως, αυτή η απλούστευση δεν επηρεάζει την απόδοση τεχνητών παικτών στο πρωτότυπο παιχνίδι καθώς οι αποφάσεις τους γίνονται συνήθως μέσα στους συντομότερους γύρους του χρόνου που ένα σχήμα παίρνει για να πέσει. Επιπλέον, εκτός από λίγους συγγραφείς, οι περισσότερες από τις υπάρχοντες μελέτες απευθύνονται στις αποκαλούμενες «στρατηγικές ενός σχήματος», όπου μόνο το τρέχον σχήμα είναι γνωστό και όχι το επόμενο, όπως συμβαίνει στο πρότυπο παιχνίδι.

Παρόλο που το Tetris είναι ένα σχετικά εύκολο παιχνίδι με απλούς κανόνες, απαιτεί πολλή εξάσκηση και περίπλοκες στρατηγικές ώστε ανθρώπινοι παίκτες να παίξουν καλά. Παρόμοια ο σχεδιασμός τεχνητών παικτών παρουσιάζει μεγάλη πρόκληση στην κοινότητα της τεχνητής νοημοσύνης, καθώς ο αριθμός των ΄καταστάσεων΄ του παιχνιδιού εκτείνεται σε μεγάλο μέγεθος (περίπου 2^{199}). Το μεγάλο αυτό μέγεθος δεν μπορεί να εξερευνηθεί άμεσα, για αυτό και υπάρχει η ανάγκη χρήσης τεχνικών προσέγγισης για να μειωθεί το μέγεθος και να σχεδιαστούν στρατηγικές για τεχνητούς παίκτες. Ακόμα και με τέτοιες τεχνικές το πρόβλημα εύρεσης στρατηγικών για μεγιστοποίηση της βαθμολογίας στο παιχνίδι είναι NPComplete (Demaine et al., 2003).

1.2 Προηγούμενες Έρευνες

1.2.1 Πρώτες προσπάθειες

Οι Tsitsiklis & Van Roy (1996) χρησιμοποίησαν το Tetris ως δοκιμαστικό παιχνίδι αναφοράς για δυναμικό προγραμματισμό μεγάλης κλίμακας με χρήση χαρακτηριστικών. Χρησιμοποίησαν δύο απλά χαρακτηριστικά : τον αριθμό των τρυπών και το ύψος της υψηλότερης στήλης. Επέτυχαν βαθμολογία περίπου 30 συμπληρωμένες γραμμές σε πίνακα 16 × 10.

Οι Bertsekas & Tsitsiklis (1996) πρόσθεσαν δύο σύνολα χαρακτηριστικών: το ύψος κάθε στήλης και την διαφορά ύψους μεταξύ διαδοχικές στήλες. Πέτυχαν βαθμολογία περίπου 2.800 γραμμών χρησιμοποιώντας την πολιτική lambda, η οποία είναι μια μέθοδος που συνδυάζει την κλασσική value iteration και policy iteration μεθόδους και σχετίζεται στενά με την οπτιμιστική policy iteration μέθοδο, όπου κάθε εκτίμηση της πολιτικής γίνεται προσεγγιστικά χρησιμοποιώντας ένα πεπερασμένο αριθμό value iterations (Bertsekas D. and Ioffe S. 1996; Bertsekas D. 2011). Σημειώστε, ωστόσο, ότι η εφαρμογή τους για τον τερματισμό του παιχνιδιού μειώνει τον πίνακα σε μέγεθος 19 × 10.

Οι Lagoudakis et al. (2002) πρόσθεσαν επιπλέον χαρακτηριστικά, συμπεριλαμβανομένων των : το μέσο ύψος των στηλών και το άθροισμα των διαφορών στα διαδοχικά ύψη των στηλών. Χρησιμοποιώντας την πολιτική των ελάχιστων τετραγώνων, πέτυχαν μια μέση βαθμολογία μεταξύ 1.000 και 3.000 γραμμές.

Ο Kakade (2001) χρησιμοποίησε έναν αλγόριθμο πολιτικής κλίσης για να επιτύχει 6.800 γραμμές κατά μέσο όρο, χρησιμοποιώντας τα ίδια χαρακτηριστικά με τους Bertsekas & Tsitsiklis (1996).

Οι Farias & Van Roy (2006) χρησιμοποίησαν έναν αλγόριθμο με δειγματοληψία περιορισμών στην μορφή εξισώσεων Bellman (Bellman R. 1952)για ένα γραμμικό επιλυτή. Ο επιλυτής αυτός βρίσκει μια πολιτική που συμπληρώνει περίπου 4.500 γραμμές χρησιμοποιώντας τα χαρακτηριστικά που χρησιμοποίησαν οι Bertsekas & Tsitsiklis (1996).

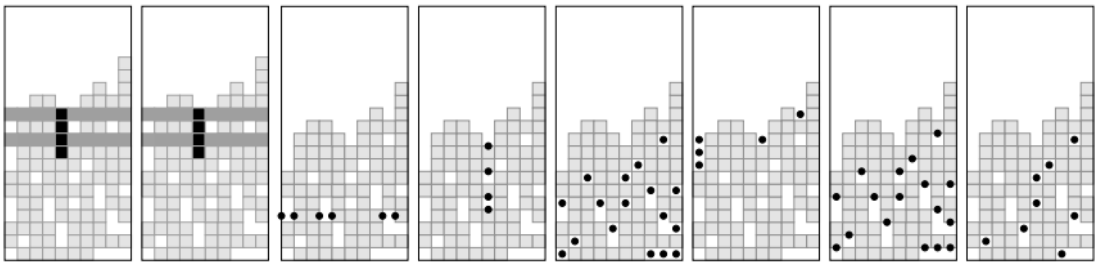
Οι Ramon & Driessens (2004) χρησιμοποίησαν σχεσιακή ενισχυτική μάθηση μαζί με ένα Γκαουσιανό πυρήνα και πέτυχαν μια βαθμολογία περίπου 50 συμπληρωμένες γραμμές. Οι Romdhane & Lamontagne (2008) συνδύασαν ενισχυτική μάθηση και η συλλογιστική των περιπτώσεων χρησιμοποιώντας μοτίβα μικρών τμημάτων του πίνακα. Οι βαθμολογίες τους ήταν επίσης περίπου 50 συμπληρωμένες γραμμές.

1.2.2 Χειροποίητος παίκτης

Μέχρι το 2008, ο καλύτερος τεχνητός παίκτης Tetris ήταν χειροποίητος, όπως ανέφερε ο Fahey (2003). Ο Pierre Dellacherie, ένας αυτοαποκαλούμενος μέσος παίκτης Tetris, αναγνώρισε έξι απλά χαρακτηριστικά και ρύθμισε τα βάρη με την μέθοδο δοκιμής-σφάλματος. Αυτά τα χαρακτηριστικά ήταν :

1. το ύψος προσγείωσης του σχήματος,
2. ο αριθμός των τρυπών,
3. ο αριθμός των μεταβάσεων των σειρών (μεταβάσεις από ένα τετράγωνο σε ένα κενό ή αντίστροφα, εξέταση κάθε σειράς από άκρο σε άκρο),
4. αριθμός μεταβάσεων στηλών,
5. αθροιστικός αριθμός πηγαδιών και
6. διαβρωμένα κελιά (αριθμός συμπληρωμένων γραμμών πολλαπλασιασμένος με τον αριθμό των τρυπών τους που κάλυψε το παρόν Tetrimino).

Η λειτουργία αξιολόγησης ήταν η εξής:
– 4 × τρύπες - αθροιστικός αριθμός πηγαδιών – μεταβάσεις σειρών – μεταβάσεις στηλών – ύψος προσγείωσης + διαβρωμένα κελιά
Αυτή η γραμμική συνάρτηση αξιολόγησης πέτυχε κατά μέσο όρο 660.000 συμπληρωμένες γραμμές στο κανονικό πίνακα (10 × 20). Οι βαθμολογίες αυτές όμως ήταν βασισμένες σε μια εκδοχή όπου το παιχνίδι τελείωνε αν το νέο σχήμα δεν είχε χώρο για να εμφανιστεί στον πίνακα (στο κέντρο της άνω σειράς). Στην απλουστευμένη εκδοχή που χρησιμοποιείται από τις προσεγγίσεις που συζητήθηκαν νωρίτερα, τα παιχνίδια θα είχαν συνεχιστεί περαιτέρω, μέχρις ότου κάθε τοποθέτηση θα γεμίσει τον πίνακα. Επομένως, αυτή η αναφορά υπονομεύει αυτόν τον απλό γραμμικό κανόνα σε σύγκριση με άλλους αλγορίθμους.



Σχήμα 1.2: Τα 8 Dellacherie-Thierry (DT) χαρακτηριστικά: Από αριστερά προς δεξιά: ύψος προσγείωσης του σχήματος, διαβρωμένα κελιά, αριθμός μεταβάσεων των σειρών, αριθμός μεταβάσεων των στηλών, αριθμός των τρυπών, αθροιστικός αριθμός πηγαδιών, βάθος τρυπών και γραμμές με τρύπες.

1.2.3 Εξελικτικοί αλγόριθμοι

Οι Bohm et al. (2005) χρησιμοποίησαν εξελικτικούς αλγορίθμους για την ανάπτυξη τεχνητού παίκτη Tetris. Στην εκδοχή τους, ο πράκτορας γνωρίζει όχι μόνο το παρόν Tetrimino που πέφτει αλλά και το επόμενο (two-piece controller). Αυτό καθιστά τα αποτελέσματά τους ασύγκριτα με εκείνα που επιτυγχάνονται σε εκδοχές με γνώση μόνο του τρέχοντος Tetrimino (one-piece controller). Ανέφεραν 480.000.000 συμπληρωμένες γραμμές χρησιμοποιώντας μια γραμμική συνάρτηση και 34.000.000 χρησιμοποιώντας μια εκθετική συνάρτηση, και τα δύο στον κανονικό πίνακα. Παρουσίασαν νέα χαρακτηριστικά όπως ο αριθμός των συνδεδεμένων τρυπών, ο αριθμός των κατεχομένων κελιών και ο αριθμός των κατεχόμενων κελιών σε σχέση με το ύψος τους. Αυτά τα επιπρόσθετα χαρακτηριστικά δεν χρησιμοποιήθηκαν σε μεταγενέστερες έρευνες.

Οι Szita & Lorincz (2006) χρησιμοποίησαν τον αλγόριθμο διασταυρούμενης εντροπίας (cross-entropy) και πέτυχαν 350.000 συμπληρωμένες γραμμές. Ο αλγόριθμος ανιχνεύει διανύσματα τυχαίων παραμέτρων σε αναζήτηση της γραμμικής πολιτικής που μεγιστοποιεί τη βαθμολογία. Για κάθε διάνυσμα παραμέτρων, εφαρμόζεται ένας αριθμός παιχνιδιών. Η μέση και τυπική απόκλιση των καλύτερων διανυσμάτων χρησιμοποιούνται για τη δημιουργία μιας νέας γενιάς. Ένας θόρυβος που συνεχώς μειώνεται επιτρέπει μια αποτελεσματική εξερεύνηση του χώρου των παραμέτρων.

Ακολουθώντας αυτήν την έρευνα, οι Thiery & Scherrer (2009) πρόσθεσαν δύο επιπλέον χαρακτηριστικά (βάθος τρυπών και γραμμές με τρύπες) και ανέπτυξαν τον παίκτη Building Controllers for Tetris (BCTS) χρησιμοποιώντας τον αλγόριθμο cross-entropy. Επέτυχαν μια μέση βαθμολογία 35.000.000 συμπληρωμένων γραμμών. Με την προσθήκη ενός νέου χαρακτηριστικού, της ποικιλομορφίας μοτίβων, ο BCTS κέρδισε τον διαγωνισμό ενισχυτικής μάθησης του 2008.

Το 2009, ο Boumaza εισήγαγε έναν άλλο εξελικτικό αλγόριθμο στον Tetris, τον Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen & Ostermeier, 2001). Αυτή είναι η πιο γνωστή εξελικτική στρατηγική. Παρατήρησε ότι τα βάρη που προέκυψαν από την εκπαίδευση ήταν πολύ κοντά σε εκείνα του Dellacherie και επίσης συμπλήρωσε 35.000.000 γραμμές κατά μέσο όρο.

Η επιτυχία των γενετικών αλγορίθμων αξίζει την προσοχή μας, δεδομένης της πρόσφατης ανέγερσης των εξελικτικών στρατηγικών ως ισχυροί ανταγωνιστές των αλγόριθμων ενισχυτικής μάθησης (Salimans et al., 2017). Συγκεκριμένα, είναι ευκολότερο να παραλληλιστούν από τους αλγόριθμους ενισχυτικής μάθησης.

1.2.4 Ενισχυτική Μάθηση

Οι Gabillon et al. (2013) βρήκαν ένα διάνυσμα βαρών που πέτυχε 51.000.000 συμπληρωμένες γραμμές χρησιμοποιώντας έναν αλγόριθμο με πολιτική classification, εμπνευσμένος από τους Lagoudakis & Parr (2003). Αυτός είναι ο πρώτος αλγόριθμος ενισχυτικής μάθησης που έχει απόδοση συγκρίσιμη με εκείνη των γενετικών αλγορίθμων. Η ιδέα των Lagoudakis & Parr ήταν να γίνει χρήση εξελιγμένων classifiers μέσα στον βρόχο αλγορίθμων ενισχυτικής μάθησης για τον εντοπισμό καλών ενεργειών. Οι Gabillon et al. (2013) εκτίμησαν τις τιμές των ζευγών κατάσταση-δράση με τη χρήση των rollouts και στη συνέχεια ελαχιστοποίησαν μια σύνθετη συνάρτηση αυτών των rollouts χρησιμοποιώντας τον αλγόριθμο CMA-ES. Στο πλαίσιο αυτού του αλγορίθμου, ο CMA-ES εκτελεί cost-sensitive classification, εξ ου και το όνομα του αλγορίθμου: Classification-Based Modified Policy Iteration (CBMPI).

1.3 Η σημασία και ο σκοπός της έρευνας

Ο στόχος αυτής της έρευνας ήταν η δημιουργία μιας αρχιτεκτονικής νευρωνικού δικτύου χρησιμοποιώντας έναν one-piece controller και afterstate evaluator στο παιχνίδι Tetris, η οποία μαθαίνει πώς να αναλύει τον πίνακα του παιχνιδιού (low-level features) και να εξάγει high-level features παρόμοια με αυτά του Dellacherie-Thierry, βλέπε Σχήμα 1.2. (Thiery & Scherrer, 2009). Η γενική ιδέα είναι η δημιουργία έξυπνων εξαγωγών χαρακτηριστικών με τη μορφή μιας γενικής παραμετροποιημένης μονάδας που θα μπορεί να βγάζει διαφορετικά χαρακτηριστικά ανάλογα με τις παραμέτρους που θα δεχθεί. Μέρος της έρευνας είναι η εύρεση και βελτιστοποίηση των παραμέτρων καθώς και των στοιχείων που θα συνδέουν την γενική αυτή μονάδα. Το δίκτυο θα μπορεί να μάθει χαρακτηριστικά υψηλού επιπέδου, καθώς και να βρει τον καλύτερο συνδυασμό (π.χ. χρησιμοποιώντας ένα χαρακτηριστικό αντί για ένα άλλο) για να επιτύχει το καλύτερο αποτέλεσμα.

Επιτυγχάνοντας καλά αποτελέσματα στο παιχνίδι του Tetris ακολουθώντας αυτήν την προσέγγιση, θα μπορούσε επίσης να σημαίνει ότι οι εφαρμογές σε άλλα παιχνίδια τύπου πίνακα (game board) είναι εφικτές, εφόσον η αρχιτεκτονική γενικεύεται και δεν είναι συγκεκριμένη για το παιχνίδι Tetris.

Προηγουμένως περιγράψαμε το πρόβλημα του Tetris και της εύρεσης/δημιουργίας τεχνητών παικτών καθώς και συνοψίσαμε τις υπάρχουσες βιβλιογραφίες και τα αποτελέσματα που επέτυχαν. Ακολούθως σε αυτήν την έρευνα θα εξηγήσουμε σύντομα τα Νευρωνικά δίκτυα, την Ενισχυτική μάθηση και τον αλγόριθμο CMA-ES (κεφάλαιο 2) και θα αναλύσουμε την μεθοδολογία που ακολούθησα τόσο στον σχεδιασμό όσο και στην υλοποίηση της αρχιτεκτονικής (κεφάλαιο 3). Έπειτα θα παρουσιάσουμε τα αποτελέσματα που πάρθηκαν και θα τα συγκρίνουμε με την υπάρχουσα βιβλιογραφία (κεφάλαιο 4), καθώς και τα συμπεράσματα που εξαγάγαμε (κεφάλαιο 5).

Κεφάλαιο 2

Υπόβαθρο

2.1 Νευρωνικά Δίκτυα

2.1.1 Γενικά

Τα νευρωνικά δίκτυα (neural networks) αποτελούν μια σχετικά νέα περιοχή στις φυσικές επιστήμες, καθ' όσον έχουν γίνει γνωστά και έχουν αναπτυχθεί μόνο κατά τα τελευταία σαράντα περίπου χρόνια. Για αυτό, η περιοχή αυτή έχει δει μια μεγάλη άνθηση, κρίνοντας από την μεγάλη ανάπτυξη που έχει παρατηρηθεί, από τον αριθμό των επιστημόνων που ασχολούνται με αυτά τα θέματα, και βέβαια από τα πολύ σημαντικά επιτεύγματα, που έχουν συμβάλλει στο να γίνουν γνωστά σε ένα ευρύτερο κύκλο. Αποτελούν επομένως ένα θέμα με μεγάλο ενδιαφέρον στις τεχνολογικές επιστήμες. Το κύριο χαρακτηριστικό τους είναι ότι οι πρώτες αρχές και λειτουργίες τους βασίζονται στο νευρικό σύστημα των ζωντανών οργανισμών (και φυσικά του ανθρώπου), αλλά η μελέτη και η χρήση τους έχει προχωρήσει πολύ πέρα από τους βιολογικούς οργανισμούς, και σήμερα τα νευρωνικά δίκτυα χρησιμοποιούνται για να λύσουν κάθε είδους προβλήματα με ηλεκτρονικό υπολογιστή. Η φιλοσοφία τους όμως είναι διαφορετική από τον τρόπο με τον οποίο δουλεύουν οι κλασσικοί υπολογιστές. Η λειτουργία τους προσπαθεί να συνδυάσει τον τρόπο σκέψης του ανθρώπινου εγκεφάλου με τον αφηρημένο μαθηματικό τρόπο σκέψης. Έτσι στα νευρωνικά δίκτυα χρησιμοποιούμε ιδέες όπως, π.χ. ένα δίκτυο μαθαίνει και εκπαιδεύεται, θυμάται ή ξεχνά μια αριθμητική

τιμή, κλπ. πράγματα που μέχρι τώρα τα αποδίδουμε μόνο στην ανθρώπινη σκέψη. Αλλά βέβαια μπορούν και χρησιμοποιούν επί πλέον και περίπλοκες μαθηματικές συναρτήσεις και κάθε είδους εργαλεία από την μαθηματική ανάλυση.

Όπως ανέφερα προηγουμένως τα τελευταία χρόνια έχει υπάρξει μία έκρηξη ενδιαφέροντος για τα νευρωνικά δίκτυα καθώς εφαρμόζονται με μεγάλη επιτυχία σε ένα ασυνήθιστα μεγάλο φάσμα τομέων της επιστήμης και της τεχνολογίας, όπως τα χρηματοοικονομικά, η ιατρική, η επιστήμη μηχανικού, η γεωλογία, η φυσική, η ρομποτική, η επεξεργασία σήματος κτλ. Στην πραγματικότητα, τα νευρωνικά δίκτυα εισάγονται οπουδήποτε τίθεται θέμα πρόβλεψης, ταξινόμησης ή ελέγχου. Αυτό δείχνει ότι για την ανάπτυξή τους απαιτούνται ταυτόχρονα γνώσεις και θέματα από πολλές περιοχές, ενώ το ίδιο ισχύει και για τις τεχνικές και τις μεθόδους που χρησιμοποιούνται. Έτσι καταλαβαίνει κανείς ότι τα νευρωνικά δίκτυα δίνουν μια νέα πρόκληση στις επιστήμες, καθ' όσον οι νέες γνώσεις που απαιτούνται είναι από τις πιο χρήσιμες στον άνθρωπο, τόσο για την ζωή και την ιατρική, όσο και στην τεχνολογία. Καμία άλλη επιστήμη σήμερα δεν συνδυάζει έτσι γνώσεις από τόσο διαφορετικές περιοχές με τόσο άμεσο τρόπο. Η σαρωτική αυτή επιτυχία, μπορεί να αποδοθεί σε δύο βασικά στοιχεία: την ισχύ και την ευχρηστία.

Ισχύς: Τα νευρωνικά δίκτυα είναι πολύ εξελιγμένες τεχνικές μη γραμμικής μοντελοποίησης, ικανές να μοντελοποιήσουν εξαιρετικά πολύπλοκες λειτουργίες. Η γραμμική μοντελοποίηση υπήρξε ευρέως διαδεδομένη για πολύ καιρό, δεδομένου ότι στα γραμμικά μοντέλα εφαρμόζονται πολύ γνωστές στρατηγικές βελτιστοποίησης. Στις συνήθεις, όμως, περιπτώσεις όπου η γραμμική προσέγγιση δεν ήταν έγκυρη, τα μοντέλα αυτά αποτύγχαναν αναλόγως. Τα νευρωνικά δίκτυα βέβαια, αν και επιτρέπουν τη μη γραμμικότητα μέσω χρήσης μη γραμμικών συναρτήσεων ενεργοποίησης, μεταθέτουν με τη σειρά τους το πρόβλημα στο ζήτημα της διάστασης (του πλήθους των διαφορετικών εισόδων και εξόδων), το οποίο αποτελεί αγκάθι στις προσπάθειες μοντελοποίησης μη γραμμικών συναρτήσεων με μεγάλο αριθμό μεταβλητών.

Ευχρηστία: Τα νευρωνικά δίκτυα εκπαιδεύονται με παραδείγματα. Ο χρήστης συγκεντρώνει αντιπροσωπευτικά δεδομένα και στη συνέχεια, καθώς τα τροφοδοτεί συστηματικά στο δίκτυο μέσω των κατάλληλων αλγορίθμων εκπαίδευσης, το δίκτυο

«αντιλαμβάνεται» αυτομάτως τη δομή των δεδομένων και η «γνώση» αυτή εκφράζεται ως κατάλληλες επιλογές συναπτικών βαρών. Επομένως το τελικό αποτέλεσμα της εκπαίδευσης με ένα συγκεκριμένο σύνολο παραδειγμάτων είναι ο προσδιορισμός των κατάλληλων βαρών του δικτύου. Ο χρήστης χρειάζεται να έχει κάποιες ουσιώδεις γνώσεις σχετικά με τον τρόπο επιλογής και προετοιμασίας των δεδομένων, τον τρόπο εκλογής του κατάλληλου νευρωνικού δικτύου και στο πως θα ερμηνευτούν τα αποτελέσματα. Παρά ταύτα, το επίπεδο των γνώσεων του χρήστη που απαιτούνται για μια επιτυχημένη εφαρμογή των νευρωνικών δικτύων, είναι πολύ χαμηλότερο συγκριτικά με κάποια περίπτωση που θα χρησιμοποιούνταν ορισμένες πιο παραδοσιακές, μη γραμμικές στατιστικές μέθοδοι.

2.1.1 Ενισχυτική Μάθηση

Η ενισχυτική μάθηση (reinforcement learning) (Sutton R. and Barto A., 2018) στην επιστήμη των υπολογιστών είναι ένας γενικός όρος που έχει δοθεί σε μια οικογένεια τεχνικών στις οποίες το σύστημα μάθησης (πράκτορας) προσπαθεί να μάθει μέσα από την άμεση αλληλεπίδραση με το περιβάλλον. Εφαρμόζεται στον έλεγχο κίνησης ρομπότ, στη βελτιστοποίηση εργασιών σε εργοστάσια, στη μάθηση επιτραπέζιων και ηλεκτρονικών παιχνιδιών, κτλ. Η έννοια της ενισχυτικής μάθησης είναι εμπνευσμένη από τα αντίστοιχα ανάλογα της μάθησης με επιβράβευση και τιμωρία που συναντώνται ως μοντέλα μάθησης των έμβιων όντων. Σκοπός του συστήματος μάθησης είναι να μεγιστοποιήσει μια συνάρτηση του αριθμητικού σήματος ενίσχυσης (ανταμοιβή), για παράδειγμα την αναμενόμενη τιμή του σήματος ενίσχυσης στο επόμενο βήμα. Το σύστημα δεν καθοδηγείται από κάποιον εξωτερικό επιβλέποντα για το ποια ενέργεια θα πρέπει να ακολουθήσει αλλά πρέπει να ανακαλύψει μόνο του ποιες ενέργειες είναι αυτές που θα του αποφέρουν το μεγαλύτερο κέρδος.

Ο πράκτορας επιλέγει ενέργειες που μπορούν να επιφέρουν αλλαγή στην κατάσταση του περιβάλλοντος και το περιβάλλον παρουσιάζει νέες καταστάσεις στον πράκτορα, βλέπε Σχήμα 2.1. Πέραν του πράκτορα και του περιβάλλοντος, βασικές έννοιες που αφορούν την Ενισχυτική Μάθηση είναι οι εξής:

- η πολιτική (policy)
- η συνάρτηση ανταμοιβής (reward function) και

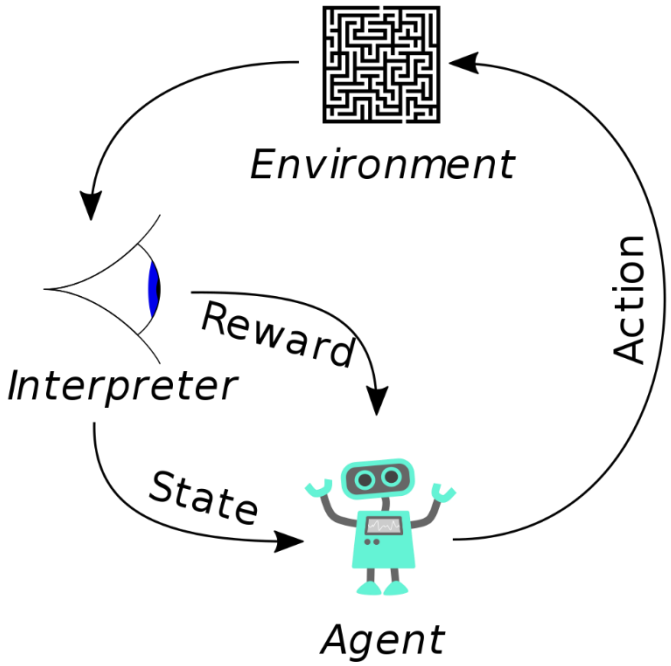
- η συνάρτηση αξίας (value function)

Η **πολιτική** ορίζει τον τρόπο συμπεριφοράς του πράκτορα ανά δεδομένη χρονική στιγμή. Χονδρικά, η πολιτική είναι μια αντιστοίχιση των καταστάσεων (states) ή παρατηρήσεων (observations) που αντιλαμβάνεται ο πράκτορας, με τις ενέργειες (actions) που επιλέγει όταν βρίσκεται σε αυτές. Σε κάποιες περιπτώσεις, μια πολιτική μπορεί να υλοποιηθεί απλά ως ένας πίνακας αντιστοιχίσεων (lookup table), ενώ σε άλλες ο ορισμός της μπορεί να εμπλέκει εκτενείς υπολογιστικές διαδικασίες. Η πολιτική ουσιαστικά αποτελεί τον πυρήνα ενός πράκτορα Ενισχυτικής Μάθησης, καθώς υπό μία έννοια, από μόνη της αρκεί για να καθορίσει την συμπεριφορά του. Στη γενική περίπτωση, οι πολιτικές είναι στοχαστικές, δηλαδή ορίζουν τις πιθανότητες με τις οποίες επιλέγονται οι ενέργειες από τον πράκτορα σε κάθε κατάσταση.

Η **συνάρτηση ανταμοιβής** ορίζει τον στόχο σε ένα πρόβλημα Ενισχυτικής Μάθησης και αντιστοιχίζει κάθε κατάσταση (ή και ζεύγος κατάστασης-ενέργειας) του περιβάλλοντος, σε έναν αριθμό, την ανταμοιβή (reward), που δηλώνει πόσο επιθυμητό είναι να βρισκόμαστε στην κατάσταση αυτή (ή να επιλέγουμε την συγκεκριμένη ενέργεια, βρισκόμενοι στην συγκεκριμένη κατάσταση, αντίστοιχα). Η μοναδική αποστολή ενός πράκτορα ΕΜ είναι η μεγιστοποίηση της συνολικής ανταμοιβής που λαμβάνει μακροπρόθεσμα. Ο πράκτορας μπορεί να βασιστεί στην συνάρτηση ανταμοιβής για να αλλάξει την πολιτική του. Για παράδειγμα, αν μια ενέργεια που επιλέγεται βάσει της τρέχουσας πολιτικής επιφέρει χαμηλή ανταμοιβή, τότε η πολιτική μπορεί να τροποποιηθεί ώστε να επιλέγει κάποια άλλη ενέργεια, όταν ο πράκτορας αντιμετωπίσει μελλοντικά την ίδια περίσταση / κατάσταση. Στη γενική περίπτωση, όπως και οι πολιτικές, οι συναρτήσεις ανταμοιβής είναι στοχαστικές.

Σε αντίθεση με την συνάρτηση ανταμοιβής που καθορίζει τί είναι καλό άμεσα, μια **συνάρτηση αξίας** καθορίζει τί είναι καλό μακροπρόθεσμα. Η αξία μιας κατάστασης ορίζεται ως το συνολικό ποσό ανταμοιβής που μπορεί να συγκεντρώσει μελλοντικά ο πράκτορας, ξεκινώντας από αυτή την κατάσταση. Η αξία μιας κατάστασης καταδεικνύει πόσο επιθυμητή είναι μια κατάσταση μακροπρόθεσμα, λαμβάνοντας υπόψιν τις καταστάσεις που θα ακολουθήσουν και τις διαθέσιμες ανταμοιβές που μπορούν να ληφθούν κατά τη μετάβαση, από και προς αυτές. Για παράδειγμα, μπορεί μια κατάσταση

να επιφέρει πάντα μικρή ανταμοιβή, ωστόσο η αξία της μπορεί να είναι μεγάλη, επειδή συνήθως ακολουθείται από καταστάσεις που επιφέρουν μεγάλες ανταμοιβές (ή και το αντίστροφο). Οι ανταμοιβές είναι υπό μια έννοια πρωταρχικές, ενώ οι αξίες, ως προβλέψεις των ανταμοιβών, δευτερεύουσες. Χωρίς τις ανταμοιβές, δε θα μπορούσαν να υπάρξουν αξίες, ενώ ο μόνος σκοπός για τον οποίο γίνεται η εκτίμηση των αξιών, είναι η προσπάθεια για την συλλογή περισσότερης ανταμοιβής.



Σχήμα 2.1 Τυπικό σενάριο Ενισχυτικής Μάθησης : ο πράκτορας επιλέγει ενέργειες σε ένα περιβάλλον, το οποίο μεταφράζεται πίσω με μια ανταμοιβή και μια αναπάρασταση της καινούριας κατάστασης και επιστρέφονται στον πράκτορα.

2.1.3 CMA-ES

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001; Hansen, 2016) όπως υποδεικνύει το όνομα είναι μια στρατηγική εξέλιξης. Οι στρατηγικές εξέλιξης (ES) είναι στοχαστικές, χωρίς παράγωγα μέθοδοι, για την αριθμητική βελτιστοποίηση μη γραμμικών ή μη κυρτών προβλημάτων συνεχούς βελτιστοποίησης. Ανήκουν στην τάξη των εξελικτικών αλγορίθμων και του εξελικτικού υπολογισμού. Ένας εξελικτικός αλγόριθμος βασίζεται σε γενικές γραμμές στην αρχή της βιολογικής εξέλιξης, δηλαδή στην επαναλαμβανόμενη αλληλεπίδραση παραλλαγής

(μέσω ανασυνδυασμού και μετάλλαξης) και επιλογής: σε κάθε γενιά (επανάληψη) νέα άτομα (υποψήφιες λύσεις, που υποδηλώνονται ως x) δημιουργούνται από παραλλαγές, συνήθως με στοχαστικό τρόπο, των τρεχόντων γονικών ατόμων. Στη συνέχεια, ορισμένα άτομα επιλέγονται για να γίνουν γονείς στην επόμενη γενιά με βάση την καταλληλόλητα τους ή την τιμή μιας συναρτήσεως εκτίμησης f (x). Με αυτόν τον τρόπο, κατά τη διάρκεια της διαδικασίας γενιών, δημιουργούνται άτομα με καλύτερες και καλύτερες τιμές f.

Σε μια στρατηγική εξέλιξης, οι νέες υποψήφιες λύσεις λαμβάνονται σύμφωνα με μια κανονική κατανομή πολλαπλών παραλλαγών στο R^n. Ο ανασυνδυασμός ισοδυναμεί με την επιλογή μιας νέας μέσης τιμής για τη κατανομή. Η μετάλλαξη ισοδυναμεί με την προσθήκη ενός τυχαίου διανύσματος, μια διαταραχή με μηδενικό μέσο όρο. Οι εξαρτήσεις κατά ζεύγη μεταξύ των μεταβλητών στην κατανομή αντιπροσωπεύονται από έναν πίνακα συνδιακύμανσης (covariance matrix). Η προσαρμογή πίνακα συνδιακύμανσης (covariance matrix adaptation, CMA) είναι μια μέθοδος ενημέρωσης του πίνακα συνδιακύμανσης αυτής της κατανομής. Αυτό είναι ιδιαίτερα χρήσιμο εάν η συνάρτηση f είναι ill-conditioned, δηλαδή μικρές αλλαγές οδηγούν σε μεγάλες διακυμάνσεις στο αποτέλεσμα.

Η προσαρμογή του πίνακα συνδιακύμανσης ισοδυναμεί με την εκμάθηση μοντέλου δεύτερης τάξης της υποκείμενης συνάρτησης εκτίμησης παρόμοια με την προσέγγιση του αντίστροφου πίνακα Hessian στη μέθοδο Quasi-Newton (Broyden C, 1972) στην κλασική βελτιστοποίηση. Σε αντίθεση με τις περισσότερες κλασικές μεθόδους, γίνονται λιγότερες υποθέσεις σχετικά με τη φύση της υποκείμενης συνάρτησης εκτίμησης. Μόνο η κατάταξη μεταξύ υποψηφίων λύσεων αξιοποιείται για την εκμάθηση της κατανομής και ούτε τα παράγωγα ούτε καν οι ίδιες οι τιμές τις συνάρτησης απαιτούνται από τη μέθοδο.

Για την προσαρμογή των παραμέτρων της κατανομής αναζήτησης στον αλγόριθμο CMA-ES, αξιοποιούνται δύο βασικές αρχές.

Πρώτον, μια αρχή μέγιστης πιθανότητας, βασισμένη στην ιδέα να αυξηθεί η πιθανότητα επιτυχημένων υποψηφίων λύσεων και βημάτων αναζήτησης. Ο μέσος όρος της κατανομής ενημερώνεται έτσι ώστε να μεγιστοποιείται η πιθανότητα επιτυχημένων

υποψηφίων λύσεων. Ο πίνακας συνδιακύμανσης της κατανομής ενημερώνεται (σταδιακά) έτσι ώστε να αυξάνεται η πιθανότητα προηγούμενων επιτυχημένων βημάτων αναζήτησης. Και οι δύο ενημερώσεις μπορούν να ερμηνευθούν ως μια φυσική κατάβαση κλίσης (natural gradient descent). Επίσης, κατά συνέπεια, η CMA πραγματοποιεί μια επαναλαμβανόμενη ανάλυση βασικών συστατικών των επιτυχημένων βημάτων αναζήτησης. διατηρώντας ταυτόχρονα όλους τους βασικούς άξονες. Οι αλγόριθμοι εκτιμήσεων κατανομής και η μέθοδος Cross-Entropy βασίζονται σε πολύ παρόμοιες ιδέες, αλλά εκτιμούν (μη σταδιακά) τον πίνακα συνδιακύμανσης μεγιστοποιώντας την πιθανότητα επιτυχημένων σημείων λύσης αντί επιτυχημένων βημάτων αναζήτησης.

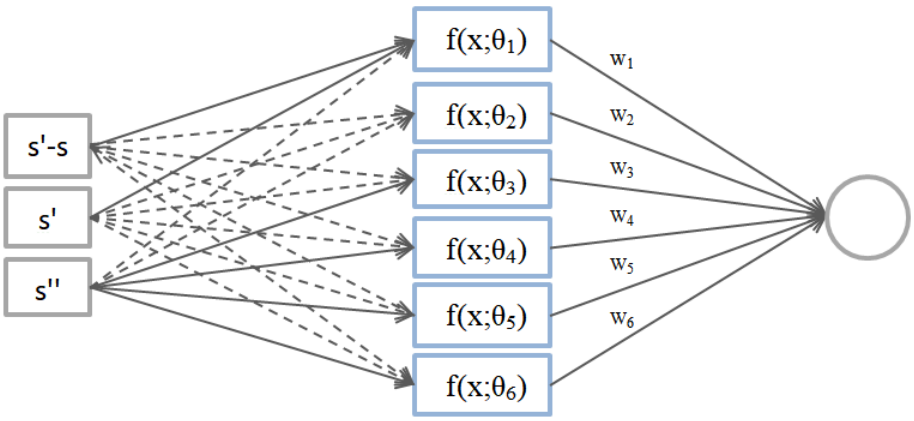
Δεύτερον, καταγράφονται δύο διαδρομές της χρονικής εξέλιξης του μέσου κατανομής της στρατηγικής, που ονομάζονται διαδρομές αναζήτησης ή εξέλιξης. Αυτές οι διαδρομές περιέχουν σημαντικές πληροφορίες σχετικά με τη συσχέτιση μεταξύ διαδοχικών βημάτων. Συγκεκριμένα, εάν ληφθούν διαδοχικά βήματα σε παρόμοια κατεύθυνση, οι διαδρομές εξέλιξης καθίστανται μεγάλες. Τα μονοπάτια εξέλιξης αξιοποιούνται με δύο τρόπους. Χρησιμοποιείται ένα μονοπάτι για τη διαδικασία CMA στη θέση μεμονωμένων επιτυχημένων βημάτων αναζήτησης και προκαλεί μια πιθανώς πολύ ταχύτερη αύξηση στην διακύμανση των ευνοϊκών κατευθύνσεων. Η άλλη διαδρομή χρησιμοποιείται για τη διεξαγωγή ενός πρόσθετου ελέγχου στο μέγεθος βημάτων. Αυτός ο έλεγχος μεγέθους βημάτων στοχεύει να κάνει τις διαδοχικές κινήσεις του μέσου όρου της κατανομής να είναι ορθογώνιες. Ο έλεγχος μεγέθους βημάτων αποτρέπει αποτελεσματικά την πρόωρη σύγκλιση, επιτρέποντας όμως τη γρήγορη σύγκλιση στο βέλτιστο.

Κεφάλαιο 3

Μεθοδολογία

3.1 Σχεδιασμός

Για να αντιμετωπιστεί το παιχνίδι του Tetris, σχεδιάστηκε ένα feedforward νευρωνικό δίκτυο (δίκτυο που δεν έχει κύκλους μεταξύ των κόμβων) που λαμβάνει ως είσοδο την κατάσταση του παιχνιδιού στην μορφή τριών πινάκων (***s'-s***, ***s'*** και ***s''***), βλέπε Σχήμα 3.1. Το κρυφό στρώμα του δικτύου αποτελείται από πολλαπλές γενικές μονάδες και κάθε μονάδα συμβάλει με βάσει το βάρος του στην τελική εκτίμηση. Στην ουσία το παρόν νευρωνικό δίκτυο αποτελεί την συνάρτηση εκτίμησης του afterstate evaluator για την εύρεση της καλύτερης τοποθέτησης του σχήματος (περιστροφή και μετάφραση).



Σχήμα 3.1: Αρχιτεκτονική του δικτύου. Το πρώτο στρώμα είναι το στρώμα εισόδου, το δεύτερο είναι το κρυφό στρώμα με τις γενικές μονάδες και δεξιά είναι η έξοδος με την εκτίμηση της συγκεκριμένης κίνησης (περιστροφή και μετάφραση σχήματος).

Οι γενικές μονάδες στο κρυφό στρώμα αποτελούνται από τα εξής συστατικά, βλέπε Παράρτημα Α :

1. Επιλογή του πίνακα εισόδου που θα χρησιμοποιηθεί στην αξιολόγηση,
2. Εφαρμογή padding στον πίνακα με 0 ή 1 σε ζευγάρια (πάνω-κάτω, δεξιά-αριστερά),
3. Προαιρετική επιλογή για εφαρμογή του αντίστροφου του φίλτρου στον πίνακα ως προς τον κάθετο ή οριζόντιο άξονα,
4. Εφαρμογή του φίλτρου στον πίνακα εισόδου,
5. Προαιρετική επιλογή για ενεργοποίηση του βρόγχου όπου στο αποτέλεσμα του φίλτρου θα κινείται προς μια κατεύθυνση μετρώντας 0 ή 1,
6. Προαιρετική επιλογή για εύρεση της μεσαίας τιμής του ύψους/πλάτους του αποτελέσματος της εφαρμογής του φίλτρου.

Όλα αυτά τα συστατικά επιλέχθηκαν και αναπτυχθήκαν βασισμένα στα χαρακτηριστικά του Dellacherie, βλέπε Σχήμα 1.2. Έγινε δηλαδή μια προσπάθεια μετάφρασης των χαρακτηριστικών του Dellacherie σε συνελίξεων για χρήση στο Tensorflow μέσα σε γενική μονάδα.

Τα τρία φίλτρα που μπορούν να χρησιμοποιήσουν οι γενικές μονάδες είναι τα παρακάτω:

3x3 φίλτρο

0	0	0
1	-1	0
0	0	0

20x1 φίλτρο

1
1
1
0
0
0
0
0
0
0
0
-1
1
1
1
1
1
1
1
1

1x10 φίλτρο

0	1	0	1	1	1	-1	-1	0	0

Με το 3x3 φίλτρο μπορούν να υπολογιστούν οι μεταβάσεις των σειρών (από κατειλημμένο κελί σε κενό), με την μορφή που έχει πιο κάτω στα αριστερά και τις μεταβάσεις των στηλών, με την μορφή στα δεξιά. Με την μορφή στα δεξιά μπορεί επίσης να υπολογιστεί και ο αριθμός τρυπών, δηλαδή τα κενά κελιά που έχουν κατειλημμένο κελί ακριβώς από πάνω τους.

0	0	0
1	-1	0
0	0	0

0	1	0
0	-1	0
0	0	0

Ακόμα με το 3x3 φίλτρο μπορούν να ευρεθούν τα πηγάδια, δηλαδή τα κενά κελιά όπου τα γειτονικά δεξιά και αριστερά είναι κατειλημμένα, με την παρακάτω μορφή.

0	0	0
1	-1	1
0	0	0

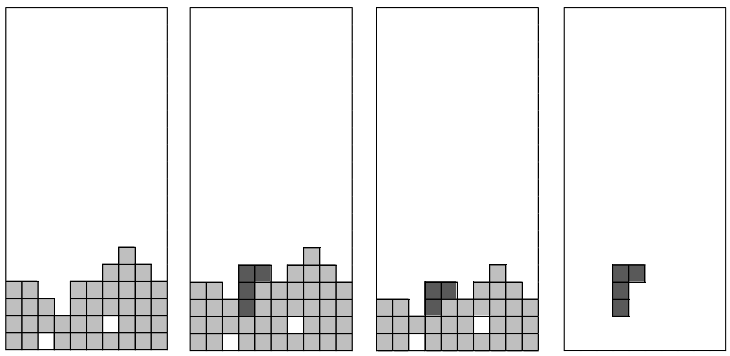
Με την χρήση του 1x10 φίλτρου μπορούν να υπολογιστούν ακόμη δύο χαρακτηριστικά του Dellacherie, το ύψος προσγείωσης του σχήματος και οι συμπληρωμένες γραμμές (στα πραγματικά χαρακτηριστικά του Dellacherie οι συμπληρωμένες γραμμές είναι τα διαβρωμένα κελιά δηλαδή ο αριθμός συμπληρωμένων γραμμών πολλαπλασιασμένος με τον αριθμό των τρυπών τους που κάλυψε το παρόν Tetrimino, αλλά επειδή δεν ήταν δυνατός ο υπολογισμός τους με την χρήση συνελίξεων μέσα σε μια γενικά μονάδα έγινε αυτός ο συμβιβασμός). Και τα δύο χαρακτηριστικά χρησιμοποιούν την παρακάτω μορφή, αλλά η εφαρμογή τους γίνεται σε διαφορετικούς πίνακες, που θα εξηγήσουμε παρακάτω.

1	1	1	1	1	1	1	1	1	1

Το 20x1 φίλτρο είναι το μοναδικό φίλτρο που επιλέχθηκε χωρίς να βασίζεται σε κάποιο υπάρχων χαρακτηριστικό του Dellacherie ή κάποιου άλλου τεχνητού παίκτη. Η επιλογή του έγινε με συμπληρωματικό σκοπό, δηλαδή αφού υπάρχει ένα οριζόντιο φίλτρο τότε λογικό ήταν να υπάρχει ένα κάθετο με την πιθανότητα εύρεσης νέων ή υπάρχοντών χαρακτηριστικών ως προς τις στήλες του πίνακα παιχνιδιού (game board).

Επιλογή πίνακα εισόδου

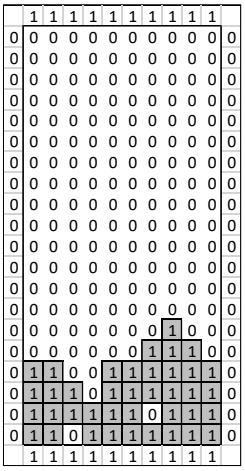
Όπως ανάφερα προηγουμένως, χρησιμοποιούνται τρεις διαφορετικοί πίνακες για την εκτίμηση της συγκεκριμένης κίνησης (***s'-s***, ***s'*** και ***s''***). Ο ***s*** πίνακας είναι ο πίνακας πριν την πτώση κάποιου σχήματος, ο ***s'*** είναι ο πίνακας μετά την πτώση κάποιου σχήματος και ο ***s''*** είναι η τελική κατάσταση της πτώσης του σχήματος (μετά την αφαίρεση συμπληρωμένων γραμμών), βλέπε Σχήμα 3.2. Οι τρεις αυτοί πίνακες είναι το αποτέλεσμα της πτώσης ενός σχήματος, όπως μας τους επιστρέφει ο Mdp Tetris. Ο s πίνακας δεν μπορεί να χρησιμοποιηθεί απευθείας για την εκτίμηση μιας περιστροφής και μετάφρασης ενός σχήματος αφού είναι η κατάσταση πριν την πτώση, που είναι η ίδια για όλες τις περιστροφές και μεταφράσεις που θα εκτιμηθούν. Για αυτό αφαιρείται από τον s' πίνακα με αποτέλεσμα ένα πίνακα (s'-s) που περιέχει μόνο το νέο tetrimino που τοποθετήθηκε στην θέση που έπεσε. Η μόνη χρήση αυτή του πίνακα είναι φυσικά η εύρεση του ύψος προσγείωσης του σχήματος. Ο πίνακας s' παρόλο που δεν είναι η τελική κατάσταση της πτώσης του σχήματος είναι εξίσου σημαντικός. Εφόσον οι συμπληρωμένες γραμμές στην τελική κατάσταση εξαφανίζονται και τα κελιά από πάνω μετακινούνται κατά ένα κελί προς τα κάτω, δεν υπάρχει τρόπος εύρεσης των συμπληρωμένων γραμμών. Έτσι η μόνη χρήση αυτού του πίνακα είναι η εύρεση αυτών των γραμμών. Ο πίνακας s'' είναι η τελική κατάσταση της πτώσης ενός σχήματος και λογικό είναι να τείνει της περισσότερης χρήσης. Από αυτόν τον πίνακα θα εξάγονται τα περισσότερα χαρακτηριστικά όπως οι μεταβάσεις σειρών/στηλών και ο αριθμός των τρυπών. Ο ***s'*** και ***s''*** πίνακας χρησιμοποιούνται χωρίς επεξεργασία ενώ ο ***s*** πίνακας, όπως είπαμε πιο πάνω, αφαιρείται από τον ***s'*** πριν να εισαχθεί στο δίκτυο. Όλοι οι πίνακες αντιπροσωπεύονται προγραμματιστικά με 0 και 1, 0 για κενό και 1 για κατειλημμένο κελί.



Σχήμα 3.2: Παράδειγμα κατάστασης παιχνιδιού Tetris. Από αριστερά προς δεξιά, ο πίνακας s, s', s'' και s'-s.

Εφαρμογή padding

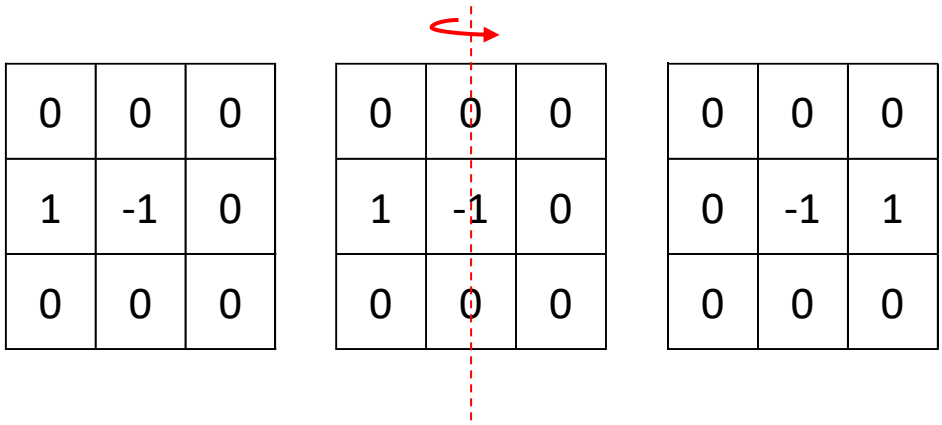
Εάν το φίλτρο που θα χρησιμοποιηθεί στην γενική μονάδα είναι 3x3, τότε για τη εξασφάλιση της σωστής εφαρμογής του φίλτρου σε όλα τα κελιά του πίνακα είναι απαραίτητη η εφαρμογή padding σε όλες τις πλευρές του πίνακα με 0 ή 1, βλέπε Σχήμα 3.3. Για τα μονοδιάστατα φίλτρα 1x10 και 20x1 δεν χρειάζεται padding αφού οι διαστάσεις τους είναι τέτοιες, ώστε να εφαρμόζονται μια φορά ανά σειρά και στήλη. Βασισμένα πάλι στα χαρακτηριστικά του Dellacherie, οι πλευρές του πίνακα ομαδοποιήθηκαν (πάνω-κάτω, δεξιά-αριστερά) και δύο τιμές χρειάζονται για την εφαρμογή του padding, μία για το κάθε ζεύγος. Η ομαδοποίηση των πλευρών έγινε με βάσει την εξάρτηση κάποιων χαρακτηριστικών σε γειτονικά κελιά, οριζόντια ή κάθετα. Για παράδειγμα, οι μεταβάσεις σειρών χρειάζονται padding με 1 αριστερά και δεξιά, διότι το περίγραμμα του πίνακα θεωρείται κατειλημμένο κελί, άρα μπορεί να υπάρχουν μεταβάσεις εάν τα κελιά που ακουμπούν στις αριστερά και δεξιά πλευρές είναι κενά. Παρομοίως, οι μεταβάσεις στηλών χρειάζονται padding με 1 πάνω και κάτω. Για την εύρεση όμως των τρυπών (κενά κελιά με κατειλημμένα κελιά ακριβώς από πάνω), χρειάζεται padding με 0 πάνω και κάτω, ώστε να μην θεωρηθούν τρύπες τα κελιά τις πρώτης γραμμής. Άλλος λόγος που γίνεται ο διαχωρισμός των πλευρών είναι η πιθανή διαφορετική εξάρτηση άγνωστων χαρακτηριστικών στην κάθε πλευρά.



Σχήμα 3.3: Παράδειγμα εφαρμογής padding σε μια κατάσταση παιχνιδιού Tetris. Σε αυτό το παράδειγμα έχουμε 0 δεξιά-αριστερά και 1 πάνω-κάτω.

Αντίστροφο του φίλτρου

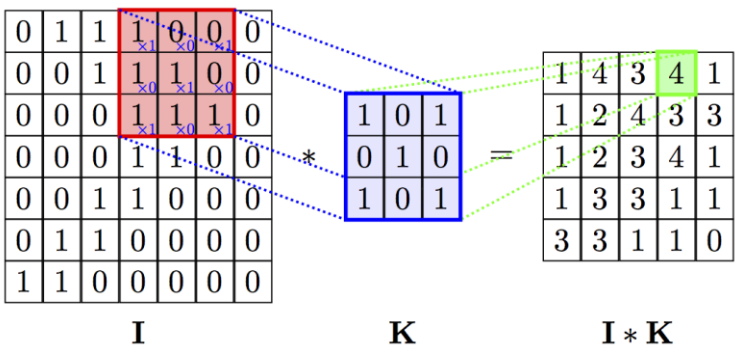
Πριν την εφαρμογή του φίλτρου πάνω στον πίνακα εισόδου υπάρχει η δυνατότητα αντιστροφής του φίλτρου, ως προς τον κάθετο ή οριζόντιο άξονα, βλέπε Σχήμα 3.4. Η αντιστροφή αυτή είναι χρήσιμη και αναγκαία για κάποια χαρακτηριστικά, κυρίως για τις μεταβάσεις σειρών και στηλών. Επειδή οι μεταβάσεις αυτές είναι από κατειλημμένο κελί σε κενό ή και αντίστροφα, δηλαδή από κενό σε κατειλημμένο, χρειάζονται δύο φίλτρα για την εύρεση όλων των μεταβάσεων. Τα δύο αυτά φίλτρα εφαρμόζονται παράλληλα στον πίνακα εισόδου και τα αποτελέσματα τους συναθροίζονται.



Σχήμα 3.4: Παράδειγμα αρχικού φίλτρου (αριστερά), αντιστροφή του ως προς τον κάθετο άξονα (μέση) και αποτέλεσμα αντιστροφής (δεξιά).

Εφαρμογή του φίλτρου

Η εφαρμογή του φίλτρου γίνεται στον πίνακα εισόδου που επιλέχθηκε και αφού εφαρμόστηκε padding και προαιρετική αντιστροφή. Η εφαρμογή γίνεται όπως η παραδοσιακή εφαρμογή μιας συνέλιξης πάνω σε δισδιάστατο πίνακα, βλέπε Σχήμα 3.5.



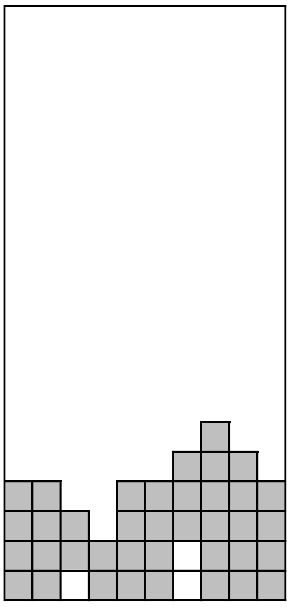
Σχήμα 3.5: Παράδειγμα εφαρμογής συνέλιξης/φίλτρου πάνω σε δισδιάστατο πίνακα.

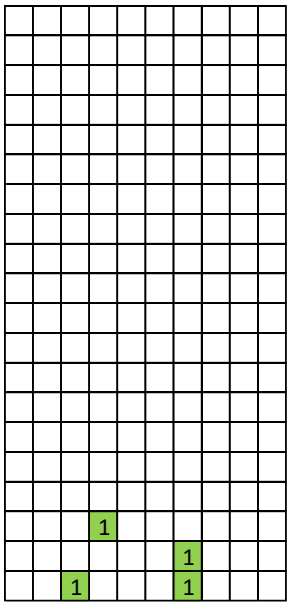
Ενεργοποίηση του βρόγχου

Η δημιουργία αυτού του συστατικού ευθύνεται το χαρακτηριστικό του Dellacherie, αθροιστικός αριθμός πηγαδιών. Με την επιλογή για ενεργοποίηση του βρόγχου μια επιπλέον επεξεργασία γίνεται στο αποτέλεσμα της εφαρμογής του φίλτρου. Αφού βρούμε τα κελιά όπου το αποτέλεσμα της εφαρμογής του φίλτρου ήταν 1, τότε για κάθε ένα από αυτά τα κελιά ξεκινούμε να μετρούμε για 0 ή 1 προς σε κάποια κατεύθυνση (πάνω, κάτω, δεξιά ή αριστερά), βλέπε Σχήμα 3.6. Σταματούμε στα όρια του πίνακα ή όταν δεν βρούμε αυτό που ψάχνουμε και το άθροισμα όλων των μετρήσεων είναι το αποτέλεσμα της εκτίμησης. Με αυτόν τον τρόπο είναι δυνατός ο υπολογισμός, περίπου του αθροιστικού αριθμού πηγαδιών. Βάσει του ορισμού του χαρακτηριστικού αυτού πηγάδια είναι τα κενά κελιά, που περικλείονται από κατειλημμένα δεξιά και αριστερά και όλα τα κελιά από πάνω τους είναι κενά. Δεν κατάφερα να το υλοποιήσω με συνελίξεις σε γενική μονάδα, έτσι το αντίστοιχο χαρακτηριστικό που μπορεί να υπολογίσει η μονάδα είναι ο αθροιστικός αριθμός όλων των κενών κελιών που περικλείονται από κατειλημμένα δεξιά και αριστερά. Για τον υπολογισμού αυτού χρειάζεται μόνο η κατεύθυνση προς τα κάτω, άλλα υπάρχουν και άλλες κατευθύνσεις για πιθανή εύρεση άλλων χαρακτηριστικών. Πιο κάτω ακολουθεί ένα παράδειγμα εκτέλεσης αυτού του βρόγχου.

Στο Σχήμα 3.6, έχουμε το φίλτρο που θα εφαρμοστεί στον πίνακα εισόδου, τον πίνακα και το αποτέλεσμα της εφαρμογής αυτού του φίλτρου.

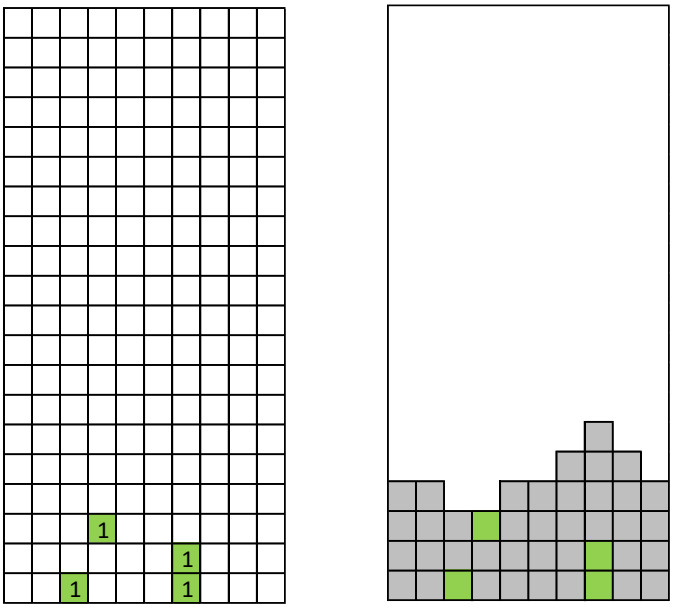
0	0	0
1	-1	1
0	0	0





Σχήμα 3.6: Παράδειγμα εφαρμογής του φίλτρου (αριστερά) πάνω στον πίνακα εισόδου (μέση) και το αποτέλεσμα τους (δεξιά).

Έπειτα στο Σχήμα 3.7, βρίσκουμε τα κελιά στον πίνακα εισόδου που είχαν αποτέλεσμα 1 (παίρνοντας τις συντεταγμένες τους) και για κάθε ένα από αυτά ξεκινούμε να μετρούμε 0 με κατεύθυνση προς τα κάτω.



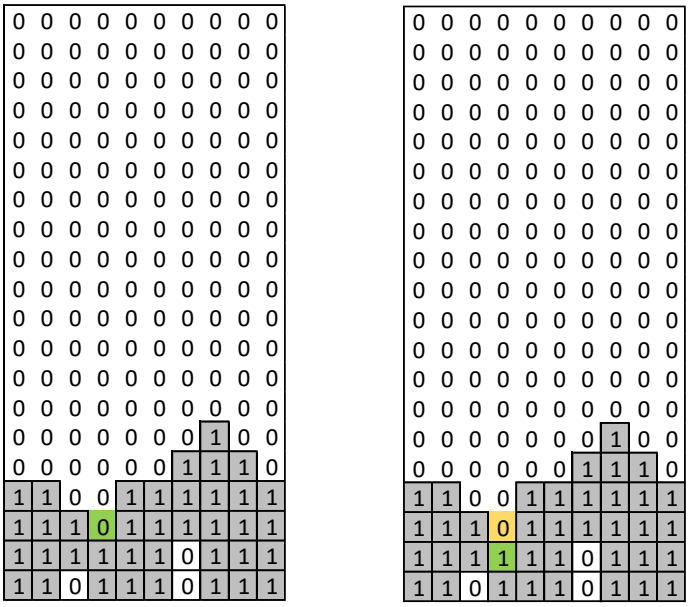
Σχήμα 3.7: Το αποτέλεσμα εφαρμογής του φίλτρου (αριστερά) και τα αντίστοιχα κελιά στον πίνακα εισόδου (δεξιά).

Στο Σχήμα 3.8, φαίνεται η πρώτη και δεύτερη επανάληψη στο πρώτο κελί που βρήκαμε προηγουμένως με συντεταγμένες [3,2] (συντεταγμένες ξεκινούν από κάτω αριστερά του πίνακα). Εφόσον ψάχνουμε για 0 και το κελί αυτό είναι μηδέν τότε αυξάνει τον μετρητή του και προχωράει στην επόμενη επανάληψη, όπου κινείται προς τα κάτω και εξετάζει εκείνο το κελί. Το κελί αυτό είναι 1, έτσι τερματίζεται ο βρόγχος για το κελί [3,2] με αποτέλεσμα 1.

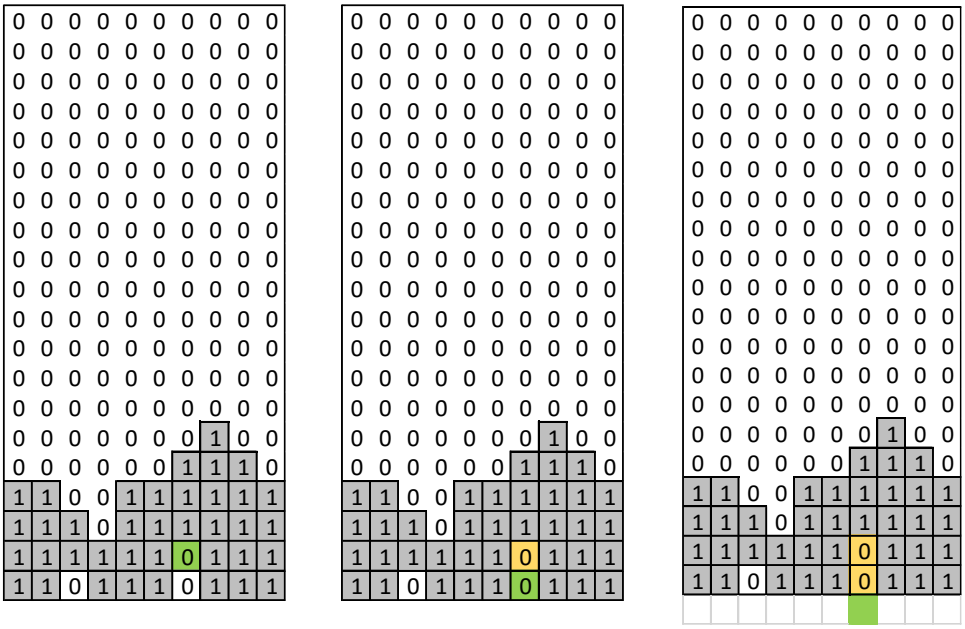
Στο Σχήμα 3.9, φαίνεται η εκτέλεση του βρόγχου στο κελί με συντεταγμένες [6,1]. Το κελί αυτό είναι 0, καθώς και το κελί ακριβώς από κάτω του. Αμέσως μετά φτάνει στα όρια του πίνακα και τερματίζει με αποτέλεσμα 2.

Στο Σχήμα 3.10, φαίνεται η εκτέλεση του βρόγχου στο κελί με συντεταγμένες [2,0]. Το κελί αυτό είναι 0 και το αμέσως από κάτω του είναι τα όρια του πίνακα, άρα τερματίζει με αποτέλεσμα 1.

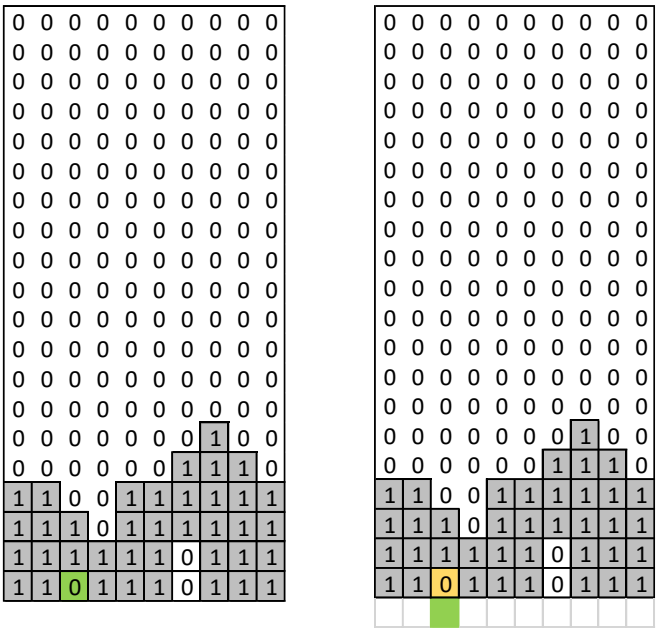
Στο Σχήμα 3.11, φαίνεται η τελευταία εκτέλεση του βρόγχου στο κελί με συντεταγμένες [6,0] και παρομοίως με το κελί [2,0] τερματίζει με αποτέλεσμα 1.



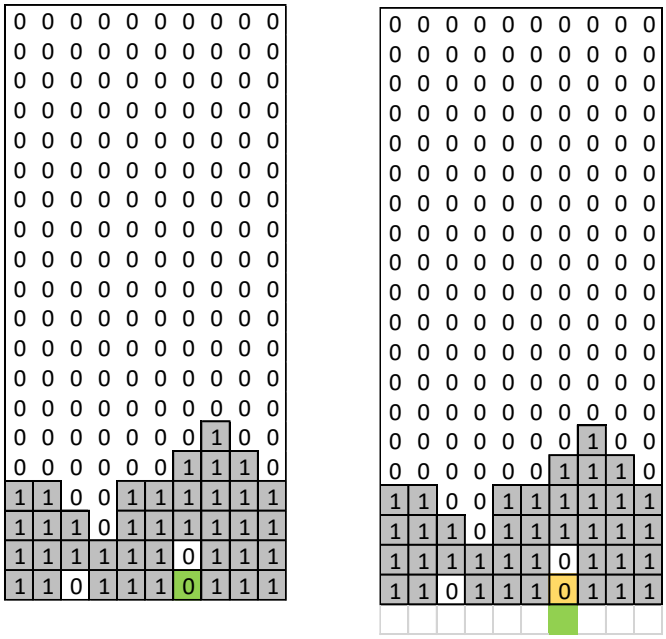
Σχήμα 3.8: Πρώτη (αριστερά) και δεύτερη (δεξιά) επανάληψη του βρόγχου στο κελί [3,2] με κατεύθυνση προς τα κάτω. Αποτέλεσμα 1.



Σχήμα 3.9: Πρώτη (αριστερά), δεύτερη (μέση) και τρίτη (δεξιά) επανάληψη του βρόγχου στο κελί [6,1] με κατεύθυνση προς τα κάτω. Αποτέλεσμα 2.



Σχήμα 3.10: Πρώτη (αριστερά) και δεύτερη (δεξιά) επανάληψη του βρόγχου στο κελί [2,0] με κατεύθυνση προς τα κάτω. Αποτέλεσμα 1.



Σχήμα 3.11: Πρώτη (αριστερά) και δεύτερη (δεξιά) επανάληψη του βρόγχου στο κελί [6,0] με κατεύθυνση προς τα κάτω. Αποτέλεσμα 1.

Αθροίζοντας όλα τα αποτελέσματα του βρόγχου (1+2+1+1), η γενική μονάδα επιστρέφει την τιμή 5.

Εύρεση μεσαίας τιμής

Ακόμη ένα συστατικό που δημιουργήθηκε με σκοπό τον υπολογισμό συγκεκριμένου χαρακτηριστικού του Dellacherie, του ύψους προσγείωσης. Εκτός από αυτό όμως πιθανή είναι και η εύρεση άλλων χαρακτηριστικών που κάνουν χρήση αυτού του συστατικού. Για να ενεργοποιηθεί το συστατικό αυτό χρειάζεται η εφαρμογή ενός μονοδιάστατου φίλτρου (20x1 ή 1x10) στον πίνακα εισόδου, έτσι ώστε το αποτέλεσμα της εφαρμογής να είναι και αυτό μονοδιάστατο. Έπειτα εύκολα υπολογίζεται η μεσαία τιμή του ύψους/πλάτους αφού προσθέσουμε το ελάχιστο και το μέγιστο και το διαιρέσουμε με το δύο, βλέπε Σχήμα 3.12.

Σχήμα 3.12: Παράδειγμα αποτέλεσμα εφαρμογής του φίλτρου και υπολογισμού της μεσαίας τιμής. Μεσαία τιμή = (min + max) / 2 = (4 + 1) / 2 = 2.5.

Μια γενική μονάδα θα έχει της εξής παραμέτρους:

- Τους 3 πίνακες εισόδου (***s'-s***, ***s'*** και ***s''***), βλέπε Σχήμα 3.2,
- Μια τιμή για την επιλογή ενός πίνακα εισόδου,
- Το φίλτρο στην μορφή πίνακα (3x3, 1x10 ή 20x1),
- Μια Boolean τιμή για εάν το φίλτρο είναι μονοδιάστατο (1x10 ή 20x1),
- Δύο τιμές για το padding (μια για αριστερά-δεξιά και μια για πάνω-κάτω),
- Μια Boolean τιμή για αντιστροφή του φίλτρου,
- Μια τιμή για την κατεύθυνση της αντιστροφής,
- Το bias που εφαρμόζεται μετά το φίλτρο,
- Μια Boolean τιμή για ενεργοποίηση του βρόγχου,
- Μια τιμή για την κατεύθυνση του βρόγχου,
- Μια τιμή την οποία θα ψάχνει κα θα μετράει ο βρόγχος και
- Μια Boolean τιμή για εύρεση της μεσαίας τιμής.

Όλες αυτοί οι παράμετροι μαζί με τα βάρη της κάθε μονάδας μαθαίνονται από τον CMAES. Επιλέχθηκε αυτός ο αλγόριθμος για αρκετούς λόγους. Ένας από αυτούς είναι η επιτυχία των εξελικτικών αλγορίθμων και συγκεκριμένα του CMAES στην βιβλιογραφία (Thiery & Scherrer, 2009; Boumaza, 2009) που αφορούν τεχνητούς παίκτες Tetris. Άλλος λόγος είναι το μέγεθος των διαστάσεων και του noisy χώρου αναζήτησης του παρόντος προβλήματος χαρακτηριστικά που μπορεί να αντιμετωπίσει ο αλγόριθμος. Ακόμα το γεγονός ότι ο CMAES δουλεύει με συνεχής τιμές, σημαίνει ότι μπορούμε να χρησιμοποιούμε τις τιμές που μας επιστρέφει ως είναι ή μετατρέποντας τις σε διακριτές.

3.2 Υλοποίηση

Απαραίτητο για την υλοποίηση ήταν η χρήση ενός αξιόπιστου περιβάλλοντος Tetris. Αυτό τον ρόλο πληροί ο προσομοιωτής Tetris, Mdp Tetris. Σκοπός αυτού του προσομοιωτή όπως αναφέρουν οι συγγραφείς του, είναι η ανάπτυξη και η σύγκριση διάφορων τεχνητών παικτών του Tetris. Αρκετές από τις προηγούμενες έρευνες κάνουν χρήση αυτού του προσομοιωτή, άρα ήταν εύκολη η απόφαση για χρήση του και στην παρούσα έρευνα. Ο Mdp Tetris χρειάζεται περιβάλλον Unix και C compiler. Εμεί όμως χρησιμοποιούμε Python έτσι χρειάζεται μια ‘ένωση’ μεταξύ C και Python. Αυτήν την ένωση την παρείχε ο Δρ. Βασίλης Βασιλειάδης με την χρήση του RL Glue, μαζί με κάποιες δοκιμές που είχε διεξάγει ο ίδιος τα οποία βοήθησαν πολύ στην υλοποίηση αυτής της έρευνας. Μαζί με τον προσομοιωτή γίνεται και χρήση του Tensorflow, ενός δωρεάν open-source λογισμικού που τείνει πολλής χρήσης σε projects μηχανικής μάθησης.

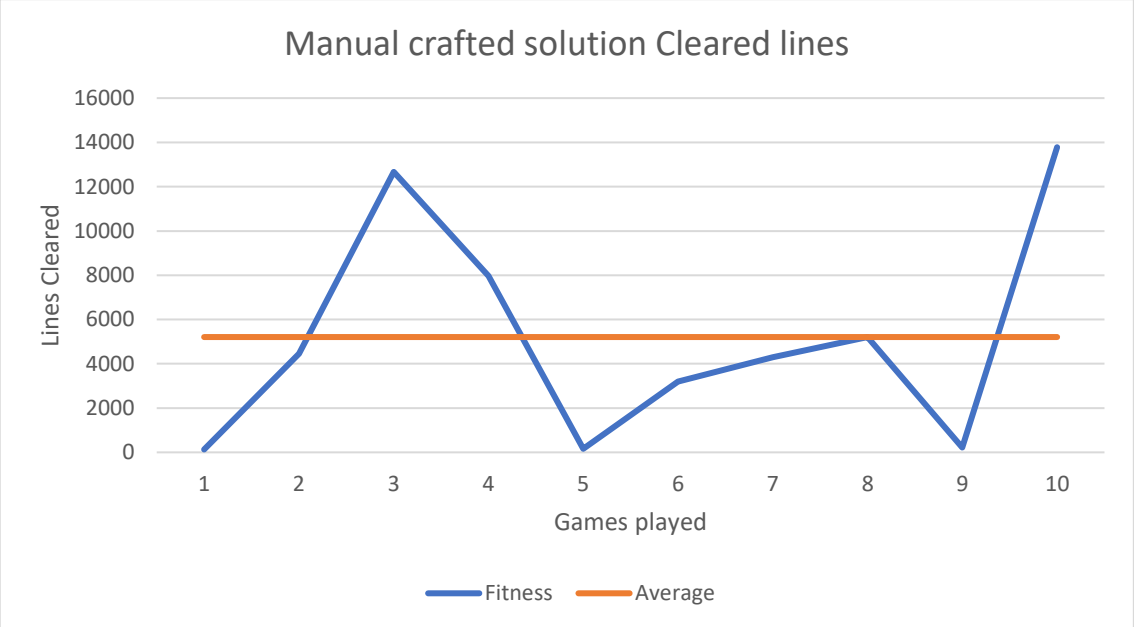
Όπως ανάφερα προηγουμένως γίνεται χρήση του CMAES για εκμάθηση των παραμέτρων και βαρών. Για κάθε επανάληψη μας παρέχει ο αλγόριθμος με πιθανές ‘λύσεις’ για εκτίμηση. Η κάθε λύση περιέχει ζεύγη 32 πραγματικών αριθμών μεταξύ του 0 και 1 οι οποίοι μεταφράζονται στους παραμέτρους μιας γενικής μονάδας όπως είδαμε προηγουμένως. Άρα ανάλογα με πόσες γενικές μονάδες θέλουμε αυξάνονται οι τιμές που πρέπει να μάθει ο αλγόριθμος (nx32). Αφού μεταφραστούν οι τιμές και δοθούν στον προσομοιωτή τότε χρησιμοποιούνται για την εκτέλεση ενός παιχνιδιού Tetris λαμβάνοντας έτσι την επίδοση του. Αυτές οι επιδόσεις δίδονται πίσω στον CMAES και αυτός με την σειρά του μας ξαναδίνει άλλες λύσεις. Αυτό επαναλαμβάνεται όσες φορές θέλουμε.

Κεφάλαιο 4

Αποτελέσματα

3.1 Αποτελέσματα

Χρησιμοποιώντας 6 γενικές μονάδες και ρυθμίζοντας τις με το χέρι ώστε να είναι όμοιες με τα χαρακτηριστικά του Dellacherie, ο τεχνητός παίκτης έχει κατά μέσο όρο ~5200 συμπληρωμένες γραμμές σε 10 παιχνίδια, βλέπε Σχήμα 4.1. Άρα με την χρήση του CMAES χρειαζόμαστε σκορ μεγαλύτερα του 5200 για να θεωρηθεί η αρχιτεκτονική αυτή χρήσιμη.



Σχήμα 4.1: Συμπληρωμένες γραμμές σε 10 παιχνίδια χρησιμοποιώντας τις χειροποίητες γενικές μονάδες.

Αρχικά υπήρχε η πρόθεση να χρησιμοποιηθεί το SZ Tetris για την εκμάθηση της καλύτερης λύσης (Szita and Szepesvári, 2010), η οποία είναι μια παραλλαγή του παραδοσιακού Tetris όπου τα tetriminoes είναι περιορισμένα στο S και στο Z. Αυτά τα δύο σχήματα είναι δύσκολα στην αντιμετώπιση τους για αυτό και το SZ Tetris είναι πολύ πιο δύσκολο από το παραδοσιακό Tetris. Ως αποτέλεσμα το υπολογιστικό κόστος για ένα παιχνίδι είναι πολύ χαμηλότερο. Δυστυχώς όμως μετά από αρκετά πειράματα παρατηρήθηκε ότι, οποιαδήποτε λύση είχε εκμάθει ο αλγόριθμος στο SZ Tetris δεν μεταφραζόταν στο παραδοσιακό παιχνίδι. Αυτό ίσως να ευθύνεται στο γεγονός ότι τα low level features που εξάγονται από το SZ να μην εφαρμόζονται στο κανονικό παιχνίδι όταν υπάρχουν και άλλα tetriminoes.

Άλλο πρόβλημα που παρατηρήθηκε είναι ότι η χρήση του συστατικού βρόγχου, ενώ δουλεύει κανονικά στην αρχή, οδηγεί το σύστημα σε αποτυχία (crash), χωρίς κανένα σήμα σφάλματος. Αυτό μάλλον ευθύνεται στον μη υπολογισμό κάποιου συνδυασμού παραμέτρων που ίσως να οδηγεί το σύστημα σε ασταμάτητο βρόγχο (infinite loop). Όλα τα επακόλουθα αποτελέσματα πάρθηκαν χωρίς την χρήση αυτού του συστατικού.
Αφού η χρήση του SZ Tetris για μείωση του υπολογιστικού κόστους δεν πέτυχε, δεν υπήρχε άλλη επιλογή από την χρήση του παραδοσιακού παιχνιδιού για την εκμάθηση καλής λύσης. Για αυτό τον σκοπό έγιναν αρκετά πειράματα, όλα με την χρήση έξι (6) γενικών μονάδων, άρα (6x32) 192 τιμών/διαστάσεων στον CMAES.

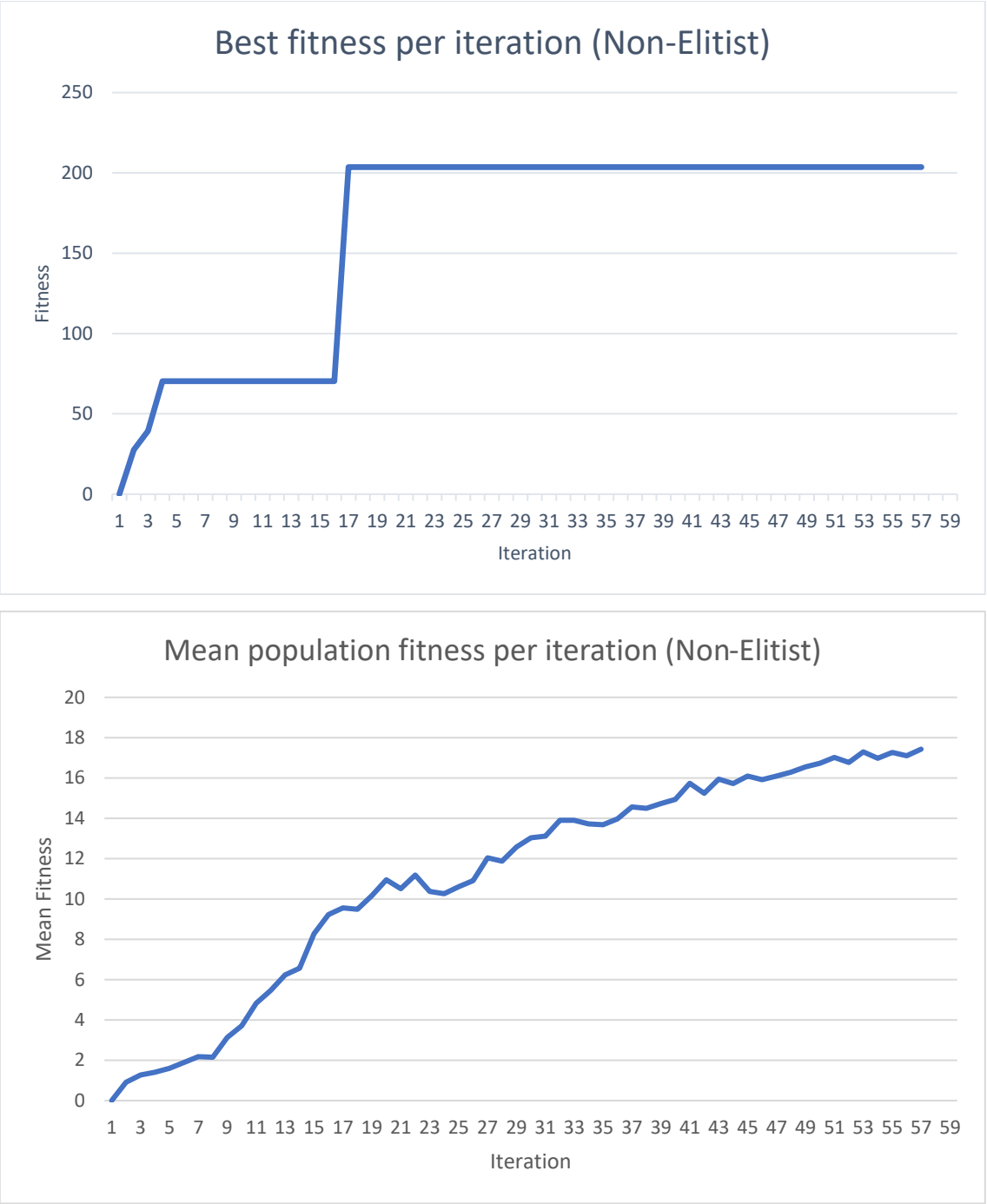
Στην αρχή δοκιμάστηκε χαμηλός πληθυσμός λύσεων (100) και επαναλήψεων (50) με μόνο ένα παιχνίδι ανά εκτίμηση λύσεως. Αυτό δεν επέφερε αποτελέσματα, καθώς ένα παιχνίδι ανά λύση δεν αντικατοπτρίζει την καταλληλόλητα της επαρκώς. Επιθυμητό θα ήταν ο αριθμός των παιχνιδιών να κυμαίνονταν μεταξύ 15 και 30, αλλά λόγω του υπολογιστικού κόστους, κατέληξα στα 5. Παρομοίως ο πληθυσμός των λύσεων ήταν μικρός αφού με τέτοιες διαστάσεις (192) χρειάζεται μεγαλύτερος αριθμός λύσεων για καλύτερη εξερεύνηση συνδυασμών παραμέτρων.

Έχοντας τα προηγούμενα υπόψη ξεκίνησα ένα πείραμα χρησιμοποιώντας τις εξής παραμέτρους :

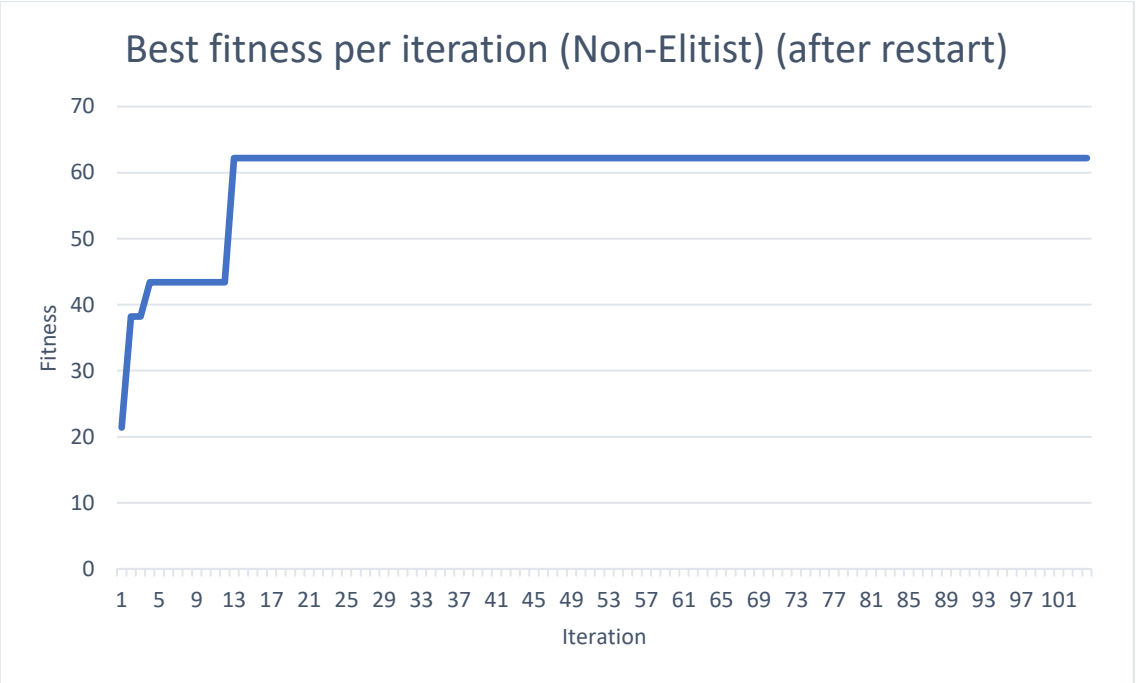
- αρχικό σίγμα 0.1, για μικρές διακυμάνσεις αφού οι τιμές του αλγορίθμου κυμαίνονται μεταξύ του 0 και 1,
- 1000 πληθυσμός λύσεων, για καλύτερη εξερεύνηση και συνδυασμών παραμέτρων,
- 5 παιχνίδια ανά λύση για καλύτερη εκτίμηση της κάθε λύσης,
- 500 επαναλήψεις/γενιές του αλγορίθμου,
- χρήση του διαγώνιου πίνακα συνδιακύμανσης, για μείωση του χρόνου εκμάθησης
- χρήση της non-elitist επιλογής, δηλαδή της επιλογής της καλύτερης λύσης μόνο από τις υποψήφιες λύσεις, αγνοώντας τους γονείς, για καλύτερη αναζήτηση,
- και χρήση ως αρχική λύση την χειροποίητη λύση, που ανάφερα στην αρχή του κεφαλαίου.

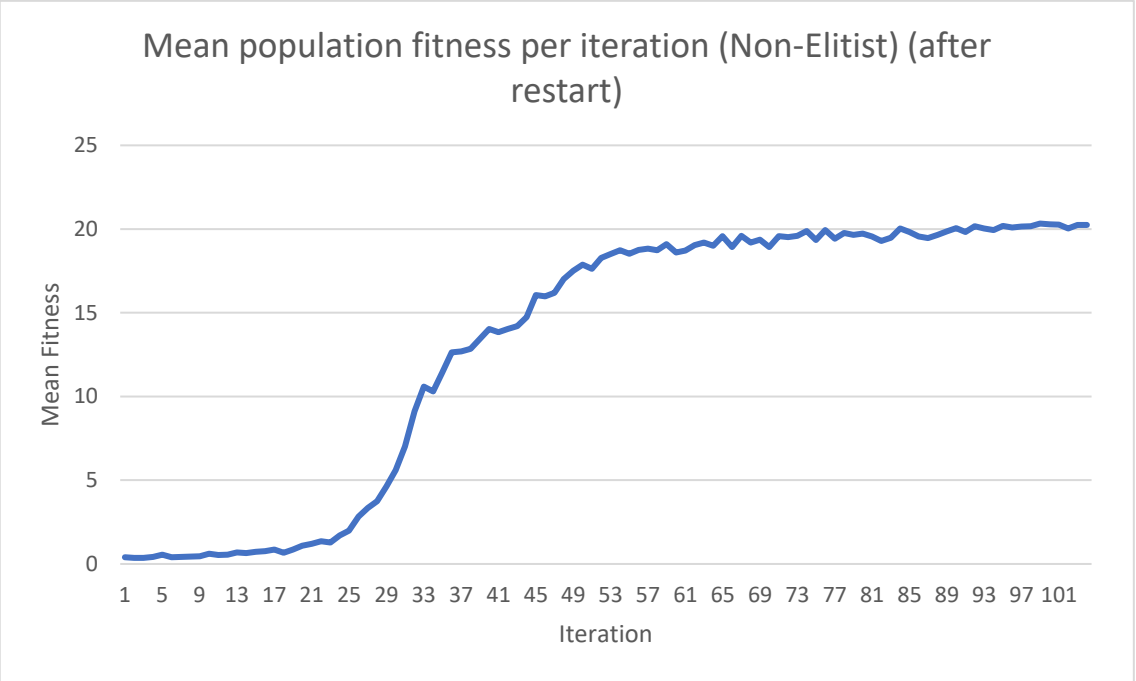
Αυτό το πείραμα, δυστυχώς διακόπηκε μετά από 56 επαναλήψεις (λόγω διακοπής ρεύματος στις εγκαταστάσεις που βρίσκεται η μηχανή, στην οποία έτρεχε το πείραμα) αλλά κατάφερε να βρει λύση που να συμπληρώνει 200 γραμμές, βλέπε Σχήμα 4.2. Μετά από επανεκκίνηση του αλγορίθμου (από την καλύτερη λύση που πέτυχε) και συνέχιση του για ακόμη 100 επαναλήψεις δεν βρέθηκε καλύτερη λύση, βλέπε Σχήμα 4.3.

Παράλληλα διεξήγαγα ένα πιο μικρό πείραμα, χρησιμοποιώντας τους ίδιου παραμέτρους με το προηγούμενο αλλά με την χρήση της Elitist επιλογής, σε περίπτωση που αυτό ευθυνόταν για τα αποτελέσματα που λάμβανα μέχρι τώρα. Όντως τα αποτελέσματα που πήρα από αυτό το πείραμα ήταν τα καλύτερα μέχρι τώρα, βλέπε Σχήμα 4.4, με 946 συμπληρωμένες γραμμές σε 33 επαναλήψεις του αλγορίθμου.

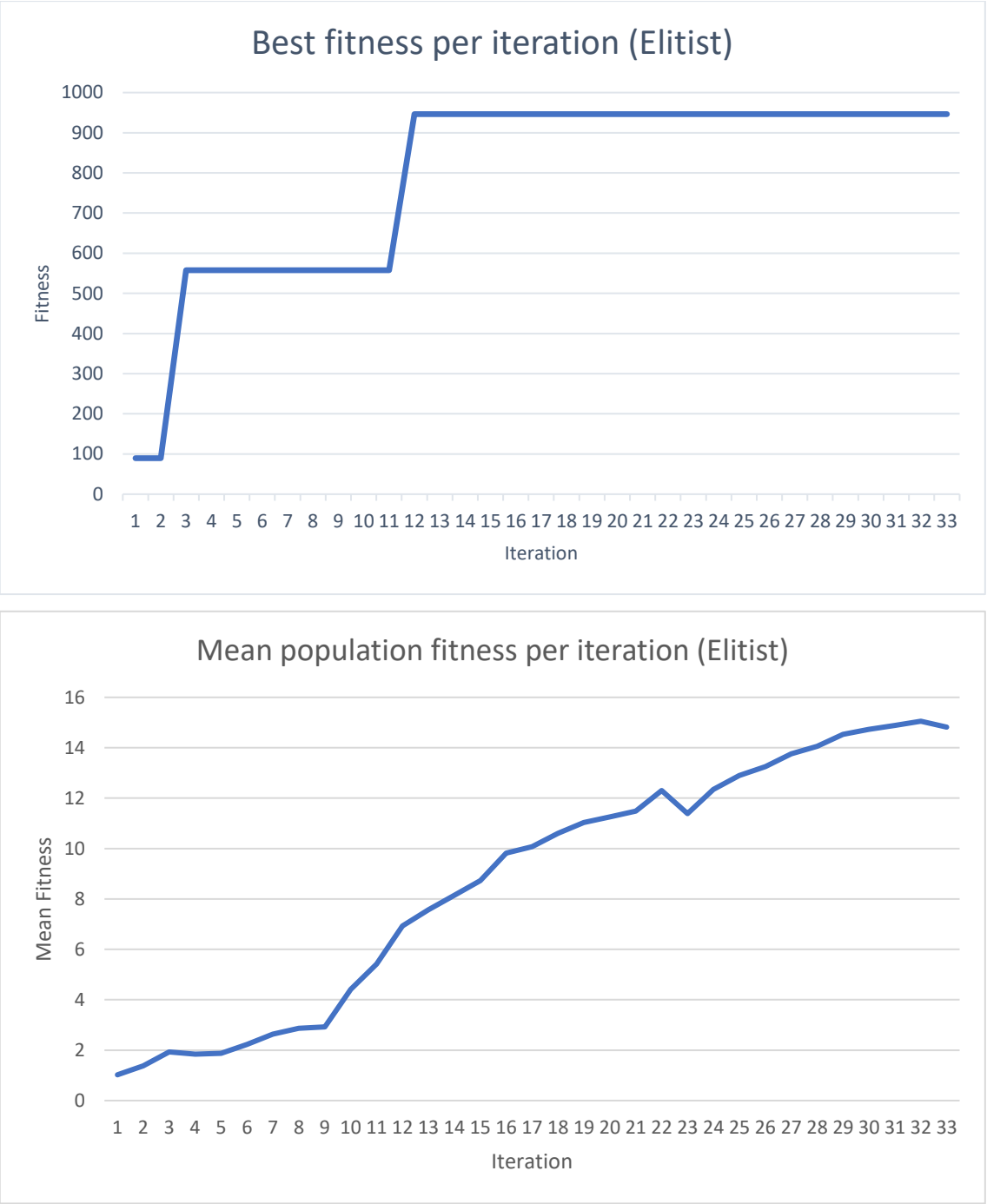


Σχήμα 4.2: Πρώτη γραφική (πάνω) παρουσιάζει την καλύτερη λύση (203 γραμμές) και η δεύτερη (κάτω) τον μέσο όρο του πληθυσμού ανά επανάληψη του αλγορίθμου (non-elitist) στις πρώτες 56 επαναλήψεις.





Σχήμα 4.3: Πρώτη γραφική (πάνω) παρουσιάζει την καλύτερη λύση (62 γραμμές) και η δεύτερη (κάτω) τον μέσο όρο του πληθυσμού ανά επανάληψη του αλγορίθμου (non-elitist) στις επόμενες 100 επαναλήψεις μετά την επανεκκίνηση του αλγορίθμου.

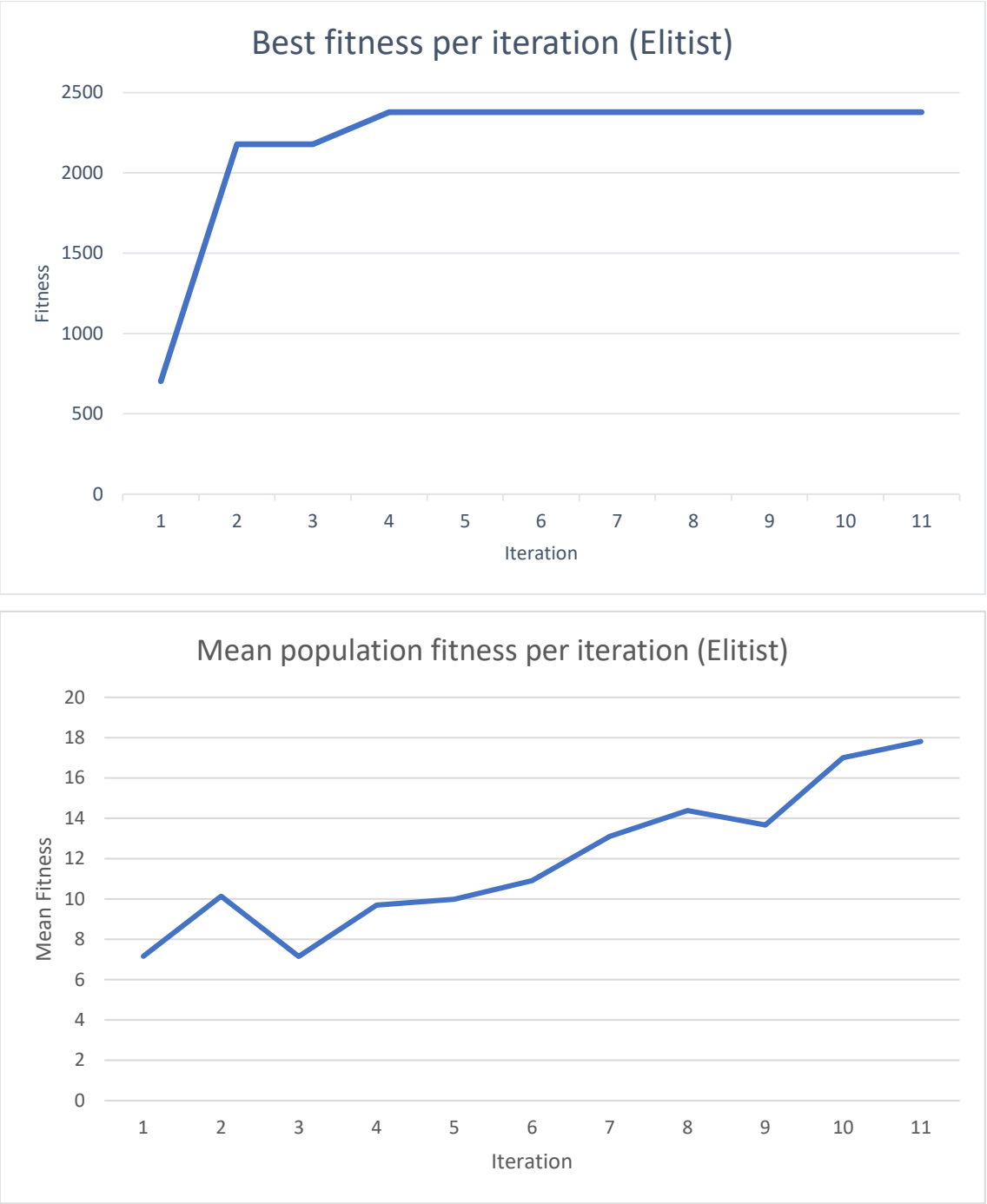


Σχήμα 4.3: Πρώτη γραφική (πάνω) παρουσιάζει την καλύτερη λύση (946 γραμμές) και η δεύτερη (κάτω) τον μέσο όρο του πληθυσμού ανά επανάληψη του αλγορίθμου (elitist) για 33 επαναλήψεις.

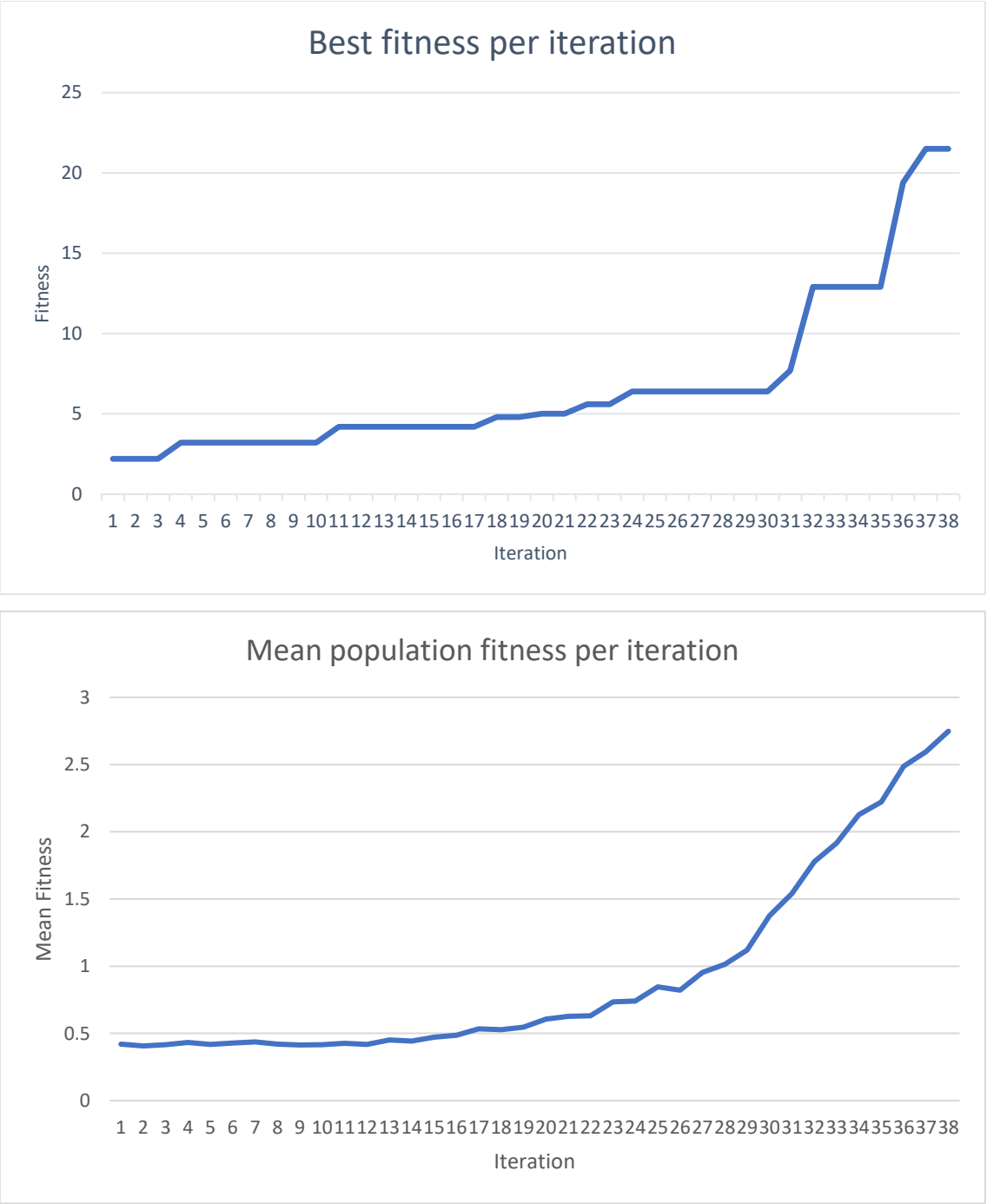
Μετά από αυτά τα αποτελέσματα διεξήγαγα δύο τελευταία πειράματα με τους ίδιους παραμέτρους, με την διαφορά όμως ότι το ένα έχει ως αρχική λύση την χειροποίητη και ο άλλος ξεκινάει από το [0.5]*192 (όλες οι τιμές αρχικοποιούνται στο 0.5 μεσαία τιμή των ορίων που έθεσα στο αλγόριθμο CMAES). Παράμετροι:

- αρχικό σίγμα 0.05, για μικρότερες διακυμάνσεις αρχικά, αφού οι τιμές του αλγορίθμου κυμαίνονται μεταξύ του 0 και 1,
- 1000 πληθυσμός λύσεων, για καλύτερη εξερεύνηση και συνδυασμών παραμέτρων,
- 10 παιχνίδια ανά λύση αντί 5, για ακόμη καλύτερη εκτίμηση της κάθε λύσης,
- 500 επαναλήψεις/γενιές του αλγορίθμου,
- χρήση του διαγώνιου πίνακα συνδιακύμανσης, για μείωση του χρόνου εκμάθησης
- χρήση της elitist επιλογής, δηλαδή της επιλογής της καλύτερης λύσης όχι μόνο από τις υποψήφιες λύσεις αλλά και από τους γονείς.

Τα αποτελέσματα αυτών των δύο πειραμάτων φαίνονται στα σχήματα 4.4 και 4.5 αντίστοιχα. Το πείραμα με την χειροποίητη λύση ως αρχική λύση έφτασε σε 2378 συμπληρωμένες γραμμές σε μόνο 11 επαναλήψεις και το άλλο έφτασε τις 21 γραμμές σε 38 επαναλήψεις.



Σχήμα 4.4: Πρώτη γραφική (πάνω) παρουσιάζει την καλύτερη λύση (2378 γραμμές) και η δεύτερη (κάτω) τον μέσο όρο του πληθυσμού ανά επανάληψη του αλγορίθμου (elitist) χρησιμοποιώντας την χειροποίητη λύση ως αρχική λύση, για 11 επαναλήψεις.



Σχήμα 4.5: Πρώτη γραφική (πάνω) παρουσιάζει την καλύτερη λύση (21 γραμμές) και η δεύτερη (κάτω) τον μέσο όρο του πληθυσμού ανά επανάληψη του αλγορίθμου (elitist) χρησιμοποιώντας ως αρχική λύση το [0.5]*192, για 38 επαναλήψεις.

Κεφάλαιο 5

Συμπεράσματα

5.1 Συμπεράσματα

Τα νευρωνικά δίκτυα και γενικότερα η μηχανική μάθηση αποτελεί μια από τις σημαντικότερες πτυχές της Τεχνητής νοημοσύνης σήμερα. Παρόλο που η εύρεση καλύτερης αρχιτεκτονικής για τεχνητούς παίκτες Tetris ή οποιουδήποτε άλλου παιχνιδιού, δεν φαίνεται σημαντική στην αρχή, αποτελεί σημαντικό κομμάτι στην παγκόσμια κοινότητα τεχνητής νοημοσύνης. Από το σκάκι και μέχρι το διάσημο παιχνίδι Dota 2, όλα τα παιχνίδια που ένας αλγόριθμος AI έχει κατακτήσει, απέφερε εξελίξεις στην επιστήμη πληροφορικής και σε άλλα πεδία.

Ο στόχος αυτής της διπλωματικής, όπως περίγραψα στο κεφάλαιο 1 ήταν η δημιουργία μιας αρχιτεκτονικής νευρωνικού δικτύου, η οποία μαθαίνει πώς να αναλύει τον πίνακα του παιχνιδιού (low-level features) και να εξάγει high-level features και πιστεύω ως ένα σημείο αυτό επιτεύχθηκε.

Βλέποντας τα αποτελέσματα στο προηγούμενο κεφάλαιο, η χρήση της Elitist επιλογή στην συγκεκριμένη αρχιτεκτονική είναι προτιμότερη. Επικεντρώνοντας κυρίως στα τελευταία δύο πειράματα, παρόλο που λόγω πολυπλοκότητας και διορίας της διπλωματικής δεν ολοκληρώθηκαν, φαίνονται να έχουν τα καλύτερα αποτελέσματα, το ένα με τις περισσότερες συμπληρωμένες γραμμές (2378) και το άλλο με την πιο σταθερή αύξηση σε best fitness αλλά και σε mean population fitness.

Επιτυγχάνοντας καλά αποτελέσματα στο παιχνίδι του Tetris ακολουθώντας αυτήν την προσέγγιση, θα μπορούσε επίσης να σημαίνει ότι οι εφαρμογές σε άλλα παιχνίδια τύπου πίνακα (game board) είναι εφικτές, εφόσον η αρχιτεκτονική γενικεύεται και δεν είναι συγκεκριμένη για το παιχνίδι Tetris.

Τέλος πιστεύω πως είναι αναγκαία να διεξαχθούν περαιτέρω έρευνα και πειράματα για να αναδειχθεί εάν η αρχιτεκτονική αυτή όντως βελτιώνει ή τουλάχιστον ανακαλύπτει τα χαρακτηριστικά του Dellacherie, στα οποία και βασίστηκε η υλοποίηση της.

Αναφορές

Bellman R. On the Theory of Dynamic Programming. Proceedings of the National Academy of Sciences of the United States of America, 38(8), 716–719.

Bertsekas D. and Tsitsiklis J. Neuro-Dynamic Programming. Nashua, NH: Athena Scientific, 1996

Bertsekas D. and Ioffe S. Temporal Differences-Based Policy Iteration and Applications in Neuro-Dynamic Programming, Technical Report LIDS-P-2349, MIT, USA, 1997.

Bertsekas D. Lambda-Policy Iteration: A Review and a New Implementation arXiv eprint arXiv: 1507.01029, 2011

Bohm N., Kokai G. and Mandl S. An Evolutionary Approach to Tetris. Proceedings of the 6th Metaheuristics International Conference (MIC2005), pp. 137–148, 2005.

Boumaza A. On the evolution of artificial Tetris players. In Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on, pp. 387–393. IEEE, 2009.

Broyden C. Quasi-Newton Methods. In Murray W. (editor) Numerical Methods for Unconstrained Optimization. London: Academic Press. pp. 87–106. ISBN 0-12-512250-0, 1972

Burgiel H. How to lose at Tetris. *The Mathematical Gazette*, 81(491):194–200, 1997

Demaine E., Hohenberger S. and Liben-Nowell D. Tetris is hard, Even to approximate. In International Computing and Combinatorics Conference, pp. 351–363. Springer, 2003

Fahey C. Tetris AI, Computer plays Tetris, June 2003. http://www.colinfahey.com/tetris/tetris.html.

Farias V. and Van Roy B. Tetris: A study of randomized constraint sampling. In Calafiore's G. and Dabbene's F. (Editors) Probabilistic and Randomized Methods for Design Under Uncertainty, pp. 189–201. Springer London, 2006.

Gabillon V., Ghavamzadeh M., and Scherrer B. Approximate dynamic programming finally performs well in the game of Tetris. In Burges C., Bottou L., Welling M., Ghahramani Z. and Weinberger K. (Editors) Advances in Neural Information Processing Systems 26, Curran Associates, Inc., pp. 1754–1762, 2013.

Hansen N. and Ostermeier A. Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation, 9(2), pp. 159-195, 2001

Hansen N. The CMA Evolution Strategy: A Tutorial. arXiv preprint arXiv: 1604.00772, 2016

Kakade S. A natural policy gradient. In Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic (NIPS'01). In Dietterich T. and Becker S. and Ghahramani Z. (Editors) Advances in Neural Information Processing Systems 14. MIT Press, Cambridge, MA, USA, 1531–1538, 2001.

Lagoudakis M., Parr R. and Littman M. Least-squares methods in reinforcement learning for control. In Vlahavas I. and Spyropoulos C. (Editors). Methods and Applications of Artificial Intelligence: Second Hellenic Conference on AI, SETN 2002, pp. 249–260. Springer, 2002.

Lagoudakis M. and Parr R. Reinforcement learning as classification: Leveraging modern classifiers. In International Conference of Machine Learning, volume 3, pp. 424–431, 2003.

MDP Tetris Documentation http://mdptetris.gforge.inria.fr/doc/index.html, 2019.

Ramon J. and Driessens K. On the numeric stability of gaussian processes regression for relational reinforcement learning. In International Conference of Machine Learning - 2004 Workshop on Relational Reinforcement Learning, pp. 10–14, 2004.

Romdhane H. and Lamontagne L. Reinforcement of local pattern cases for playing Tetris. In Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, pp. 263–268, 2008.

Salimans T., Ho J., Chen Xi, Sidor S. and Sutskever I. Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864, 2017.

Sutton R. and Barto A. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA, 2018.

Szita I. and Lorincz A. Learning Tetris using the noisy cross-entropy method. Neural Computation, 18 (12):2936–2941, 2006.

Szita I. and Szepesvári C. SZ-Tetris as a benchmark for studying key problems of reinforcement learning. In Proceedings of the ICML 2010 Workshop on Machine Learning and Games., 2010.

Thiery C. and Scherrer B. Improvements on learning Tetris with cross entropy. International Computer Games Association Journal, 32(1): 23–33, 2009.

Thiery C. and Scherrer B. Building controllers for Tetris. International Computer Games Association Journal, 32(1):3–11, 2009.

Tsitsiklis J. and Van Roy B. Feature-based methods for large scale dynamic programming. Machine Learning, 22(1-3):59–94, 1996.

Παράρτημα Α

```
def generic_module(input, choose_input, filter, one_dimensional, bias,
padding, process, loop, loop_direction, countFor, flip,
flip_direction):
    try:
        input = input[choose_input]
        input_shape = tf.shape(input)
        # print(input_shape)
        output = tf.reshape(input, shape=[1, input_shape[0],
input_shape[1], 1])

        # print(tf.shape(output))
        if (one_dimensional != True):
            if (padding[0] == 1):  # left-right
                output = tf.pad(output, [[0, 0], [0, 0], [1, 1], [0,
0]], "CONSTANT", constant_values=1)
            if (padding[0] == 0):  # left-right
                output = tf.pad(output, [[0, 0], [0, 0], [1, 1], [0,
0]], "CONSTANT", constant_values=0)
            if (padding[1] == 1):  # top-bottom
                output = tf.pad(output, [[0, 0], [1, 1], [0, 0], [0,
0]], "CONSTANT", constant_values=1)
            if (padding[1] == 0):  # top-bottom
                output = tf.pad(output, [[0, 0], [1, 1], [0, 0], [0,
0]], "CONSTANT", constant_values=0)

        conv_output = []
        filter = tf.constant(filter, dtype=tf.float32)
        filter_shape = tf.shape(filter)
        if (flip and one_dimensional != True):
            flipped_filter = np.copy(filter)
            if (flip_direction == "Vertical"):
                flipped_filter = np.flip(flipped_filter, 1)
            elif (flip_direction == "Horizontal"):
                flipped_filter = np.flip(flipped_filter, 0)

            filter = np.concatenate([[filter], [flipped_filter]])
            filter = np.expand_dims(filter, -1)
            filter = filter.transpose(1, 2, 3, 0)
            # filter = tf.reshape(filter, shape=[filter_shape[0],
filter_shape[1], 1, 2])
            # print filter
        else:
            filter = tf.reshape(filter, shape=[filter_shape[0],
filter_shape[1], 1, 1])

        # print output
        conv_output = tf.nn.conv2d(output, filter, [1, 1, 1, 1],
"VALID") + bias
        conv_output = heaviside(conv_output)
        # print conv_output

#         if (loop and one_dimensional != True and flip == False):
```

```
#             conv_output = tf.reshape(conv_output,
shape=[input_shape[0], input_shape[1]])
#             # get the indices where we have ones
#             indices = tf.where(tf.equal(conv_output, 1))
#             # print indices
#
#             count = 0
#             for index in indices:
#                 new_index = tf.Variable(index)
#
#                 new_val = 0
#                 while tf.reduce_all(tf.less(new_index,
conv_output.shape)) and tf.reduce_all(
#                         tf.greater(new_index, [-1, -1])):
#                     # print(new_index)
#                     new_val = input[list(new_index.numpy())]
#
#                     if tf.equal(new_val, countFor):
#                         count += 1
#                         if (loop_direction == "Down"):
#                             new_index[0].assign(new_index[0] + 1)
#                         if (loop_direction == "Right"):
#                             new_index[1].assign(new_index[1] + 1)
#                         if (loop_direction == "Up"):
#                             new_index[0].assign(new_index[0] - 1)
#                         if (loop_direction == "Left"):
#                             new_index[1].assign(new_index[1] - 1)
#                     else:
#                         break
#
#             return tf.constant(count, dtype=tf.float32)

        if (process and one_dimensional):
            filter_shape = tf.shape(filter)
            if filter_shape[0] > filter_shape[1]:
                coords_size = input_shape[1]
                conv_output = tf.reshape(conv_output,
shape=[input_shape[1], 1])
            else:
                coords_size = input_shape[0]
                conv_output = tf.reshape(conv_output,
shape=[input_shape[0], 1])
            coords = range(coords_size)
            coords.reverse()
            coords = tf.reshape(coords, shape=[coords_size, 1])
            coords = tf.cast(coords, 'float32')
            # print coords
            # print conv_output
            not_conv_output = heaviside(0.5 - conv_output)
            gating = heaviside(tf.reduce_sum(conv_output) + bias)
            saturating_value = 999
            not_conv_output = saturating_value * not_conv_output
            # print not_conv_output
            masked_output = tf.multiply(conv_output, coords)
            # print masked_output
            res = (not_conv_output + masked_output) * gating
            ret_min = tf.math.reduce_min(res)
            ret_max = tf.math.reduce_max(masked_output)
            # print ret_min
            # print ret_max
```

```
            return (ret_max + ret_min) / 2.0

        reduce_sum_output = tf.math.reduce_sum(conv_output)
        return reduce_sum_output
    except:
        error = open("error.txt", "a")
        e = sys.exc_info()[0]
        error.write("<p>Error: %s</p>" % e)
        error.close()
        return 0

def translate_solution(solution):
    # choose input from 3 options
    if solution[0] < 1.0 / 3:
        input = 0
    elif solution[0] < 2.0 / 3:
        input = 1
    else:
        input = 2

    # translate range [0 , 1] to -1, 0 or 1
    index = 2
    filter_values = []
    for i in range(0, 20):
        if solution[index + i] < 1.0 / 3:
            filter_values.append(-1)
        elif solution[index + i] < 2.0 / 3:
            filter_values.append(0)
        else:
            filter_values.append(1)

    # choose filter shape from 3 options
    if solution[1] < 1.0 / 3:
        filter = [[filter_values[0], filter_values[1],
filter_values[2], filter_values[3], filter_values[4],
                   filter_values[5], filter_values[6],
filter_values[7], filter_values[8], filter_values[9]]] # horizontal 1
x 10
        one_dimensional = True
    elif solution[1] < 2.0 / 3:
        filter = [[filter_values[0]], [filter_values[1]],
[filter_values[2]], [filter_values[3]], [filter_values[4]],
                  [filter_values[5]], [filter_values[6]],
[filter_values[7]], [filter_values[8]], [filter_values[9]],
                  [filter_values[10]], [filter_values[11]],
[filter_values[12]], [filter_values[13]], [filter_values[14]],
                  [filter_values[15]], [filter_values[16]],
[filter_values[17]], [filter_values[18]], [filter_values[19]]] #
vertical 20 x 1
        one_dimensional = True
    else:
        filter = [[filter_values[0], filter_values[1],
filter_values[2]],
                  [filter_values[3], filter_values[4],
filter_values[5]],
                  [filter_values[6], filter_values[7],
filter_values[8]]] # 3 x 3
        one_dimensional = False
```

```
    index = 22
    bias = np.interp(solution[index],[0.0, 1.0],[-20.0, 20.0])
    index += 1
    padding = []
    # choose left-right padding from 2 options
    if solution[index] < 1.0 / 2:
        padding.append(0)
    else:
        padding.append(1)

    index += 1
    # choose top-bottom padding from 2 options
    if solution[index] < 1.0/2:
        padding.append(0)
    else:
        padding.append(1)

    index += 1
    # choose whether or not to process one dimensional convolutional
output
    if solution[index] < 1.0 / 2:
        process = False
    else:
        process = True

    index += 1
    # choose whether to loop or not
    if solution[index] < 1.0/2:
        loop = False
    else:
        loop = True
    index += 1
    # choose loop direction
    if solution[index] < 1.0/4:
        loop_direction = "Left"
    elif solution[index] < 2.0/4:
        loop_direction = "Up"
    elif solution[index] < 3.0/4:
        loop_direction = "Right"
    else:
        loop_direction = "Down"
    index += 1
    # choose what to count in loop
    if solution[index] < 1.0/2:
        countFor = 0
    else:
        countFor = 1
    index += 1
    # choose whether to also apply the reverse of the filter
    if solution[index] < 1.0/2:
        flip = False
    else:
        flip = True
    index += 1
    # choose which direction to flip the filter
    if solution[index] < 1.0/2:
        flip_direction = "Vertical"
    else:
        flip_direction = "Horizontal"
```

```
    index += 1
    weight = np.interp(solution[index], [0.0, 1.0], [-20.0, 20.0])

    return input, filter, one_dimensional, bias, padding, process,
loop, loop_direction, countFor, flip, flip_direction, weight
```