ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Τεχνικές Επιβλεπόμενης και Ενισχυτικής Μάθησης στο Παιχνίδι Checkers

Διπλωματική Εργασία

Ιωάννης Ν. Θεοδοσίου

Επιβλέπων Καθηγητής: Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής Ε.Μ.Π

Αθήνα, Μάιος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Τεχνικές Επιβλεπόμενης και Ενισχυτικής Μάθησης στο Παιχνίδι Checkers

Διπλωματική Εργασία

Ιωάννης Ν. Θεοδοσίου

Επιβλέπων Καθηγητής: Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26/5/2021:

.....	.....	.....
Ανδρέας Σταφυλοπάτης Καθηγητής Ε.Μ.Π	Στέφανος Κολιας Καθηγητής Ε.Μ.Π	Γεώργιος Στάμου Καθηγητής Ε.Μ.Π

Αθήνα, Μάιος 2021

.....
Ιωάννης Ν. Θεοδοσίου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα τελευταία χρόνια έχει γίνει εμφανές ότι η μηχανική και η ενισχυτική μάθηση αποτελούν ισχυρό τρόπο αντιμετώπισης παιχνιδιών ανάμεσα σε δύο παίκτες και γενικότερα περιβάλλοντα στα οποία υπάρχει εναλλαγή κινήσεων και η πλεονεκτική θέση του ενός φέρνει σε μειονεκτική θέση τον άλλο. Συγκεκριμένα το 2016 έχουμε ίσως ένα από τα μεγαλύτερα επιτεύγματα στον χώρο της επιστήμης των υπολογιστών όταν μηχανή σχεδιασμένη από τη DeepMind της Google κατάφερε για πρώτη φορά να νικήσει σε μία σειρά αγώνων Go τον νυν παγκόσμιο πρωταθλητή. Κάτι τέτοιο θεωρούνταν από πολλούς ακατόρθωτο μέσω κλασικών τεχνικών Τεχνητής νοημοσύνης κυρίως λόγω της αχανής φύσης του παιχνιδιού με περισσότερες από 10^{172} διαφορετικές θέσεις, περισσότερες δηλαδή από όσα είναι τα άτομα στο σύμπαν. Σκοπός της παρούσας εργασίας είναι η μελέτη των αλγορίθμων μάθησης, επιβλεπόμενης και ενισχυτικής, πάνω στο παιχνίδι checkers ,γνωστό και ως ντάμα, το οποίο αν και απλούστερο, με 10^{20} θέσεις παρουσιάζει μεγάλο υπολογιστικό ενδιαφέρον.

λέξεις κλειδιά: Επιβλεπόμενη μάθηση, Ενισχυτική μάθηση, Βαθιά Ενισχυτική Μάθηση, Τεχνητά Νευρωνικά Δίκτυα, Αλγόριθμοι αναζήτησης δέντρου, Θεωρία Παιγνίων, Policy iteration, Monte Carlo Tree Search

Abstract

The last few years it has become obvious that machine and reinforcement learning are a powerful way of confronting games between two players and in general environments in which there is an interchange of movements and the advantageous position of one player leads the other to a disadvantageous one. More specifically in 2016 one the biggest achievements in the field of computer science took place when a machine designed by Deep Mind of Google managed for the first time to beat the world champion in a series of games. This was considered by many impossible to achieve through classical methods of artificial intelligence, mainly due to the vast nature of the game, with more than 10^{172} different positions, that is more than the number of atoms in the universe. The purpose of this report is to study the algorithms of learning, supervised and reinforcement, on the game of checkers, which although it is simpler, with 10^{20} positions, it is of high computational interest.

Keywords: Reinforcement Learning , Deep Reinforcement Learning, Artificial Neural Networks, Tree Search Algorithms, Game Theory, Policy Iteration, Monte Carlo Tree Search

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Η διαδικασία της μάθησης στον άνθρωπο γίνεται αντιληπτή από τα πρώτα στάδια της ζωής του. Σε μόλις βρεφική ηλικία μαθαίνει να αλληλεπιδρά με το περιβάλλον του και με τα ερεθίσματα που δέχεται να προσαρμόζει τη μετέπειτα πορεία του και τη συμπεριφορά του. Τα ερεθίσματα που δέχεται μπορούν να αφορούν είτε απευθείας παρατήρηση των γονιών και των τριγύρω ανθρώπων ώστε να μιμηθεί τη συμπεριφορά τους είτε το χώρο στον οποίο βρίσκεται. Έτσι για παράδειγμα παρατηρώντας τα χείλη από ανθρώπους θα μάθει να μιλάει την πρώτη του γλώσσα ενώ επίσης καθώς περιπλανιέται στη βεράντα θα καταλάβει ότι δεν πρέπει να αγγίζει αιχμηρά αντικείμενα αφού πρώτα τρυπηθεί μια φορά από αυτά. Παρόλη την τεράστια διαφορά ανάμεσα σε μηχανές και ανθρώπους η διαδικασία της εκπαίδευσης τους με την ευρύτερη έννοια παρουσιάζει αρκετές ομοιότητες. Η επιβλεπόμενη και η ενισχυτική μάθηση αν και δυο διαφορετικές μέθοδοι κινούνται προς την εκμάθηση κάποιας λειτουργίας είτε με τη μορφή μίμησης έτοιμων παραδειγμάτων είτε μέσω αλληλεπίδρασης με το περιβάλλον και αξιολόγηση των αποτελεσμάτων.

1.1 Ορισμός Μηχανικής Μάθησης

Σαν γενικότερος ορισμός η έννοια της μηχανικής μάθησης είναι υποσύνολο της τεχνητής νοημοσύνης και περιλαμβάνει την μελέτη και την ανάπτυξη αλγορίθμων με τους οποίους μοντελοποιούνται στατιστικά μοντέλα τα οποία καλούνται να επιτελέσουν κάποια εργασία βασιζόμενα σε προηγούμενη εμπειρία και δικών τους μοτίβων και κανόνων που έχουν προκύψει από αυτή. Η προηγούμενη «εμπειρία» λέμε ότι αποτελεί το στάδιο εκπαίδευσης του μοντέλου το οποίο απλά επεξεργάζεται στατιστικά μια βάση δεδομένων που του παρέχουμε σε αντίθεση από τον ρητό προγραμματισμό του για την εκτέλεση της διαδικασίας. Στη συνέχεια το μοντέλο καλείται να κάνει κάποια γενίκευση και να δώσει μια έξοδο με βάση ό,τι έχει αποκομίσει από τα δεδομένα εκπαίδευσης.

1.2 Πλεονεκτήματα Μηχανικής Μάθησης

- Ο τεράστιος όγκος δεδομένων στις σύγχρονες εφαρμογές μπορεί να κρύβει συσχετίσεις οι οποίες είναι δύσκολο να ληφθούν υπόψη από τον άνθρωπο και με κλασσικές μεθόδους στατιστικής.
- Αυτοματοποίηση με την έννοια ότι η παρέμβαση του προγραμματιστή είναι ελάχιστη και εμφανίζεται κυρίως στην επιλογή κάποιων υπερπαραμέτρων για την απόδοση του μοντέλου.

- Συνεχής βελτίωση: Όσο περισσότερα (διαφορετικά) δεδομένα τροφοδοτούνται στο μοντέλο τόσο καλύτερα προσεγγίζουμε την διαδικασία που θέλουμε να εκτελέσουμε.
- Ικανότητα χειρισμού δεδομένων και επίλυση προβλημάτων σε πολυδιάστατους χώρους.

1.3 Εφαρμογές Μηχανικής Μάθησης

- Αναγνώριση εικόνας (Image recognition): Σε συνδυασμό με τεχνικές και εργαλεία από την όραση υπολογιστών (computer vision) η μηχανική μάθηση έχει επιτύχει αξιοσημείωτα αποτελέσματα στην αναγνώριση αντικειμένων μέσα από εικόνες.
- Πρόβλεψη χρονοσειράς (series forecasting): Μιλάμε συγκεκριμένα για ανάπτυξη μοντέλων τα οποία δεδομένης μιας χρονοσειράς έχουν εκπαιδευτεί να προβλέπουν την επομένη (η επόμενες) τιμή ελαχιστοποιώντας κάποιο σφάλμα. Τέτοιου είδους εφαρμογές χρησιμοποιούνται είτε για την πρόβλεψη μετοχών είτε γενικότερα για καμπύλες πωλήσεων ή ζημιών σε διάφορες επιχειρήσεις.
- Συστήματα προτάσεων βασιζόμενα στο προφίλ χρηστών: Εδώ για παράδειγμα έχουμε εφαρμογές συλλογής προηγούμενων αγορών και αναζητήσεων χρηστών και είναι σε θέση να προτείνουν νέα προϊόντα κοντά στο ενδιαφέρον του καταναλωτή αυξάνοντας έτσι την επιχειρηματικότητα.
- Παιχνίδια: Καθόλου σπάνια δεν είναι και η ανάπτυξη ευφυών παικτών οι οποίοι είναι σε θέση να μάθουν στρατηγικές νίκης παιχνιδιών μέσα από την αλληλεπίδραση με το περιβάλλον που τους παρέχεται και γνώση των κανόνων του.

1.4 Τα Είδη της Μάθησης

Η εκπαίδευση των μοντέλων διαχωρίζεται ανάλογα με την προσέγγισή τους, με τον τύπο των δεδομένων που απαιτεί ο κάθε αλγόριθμος και φυσικά την φύση του προβλήματος που καλείται το σύστημα να επιλύσει.

1.4.1 Μάθηση με Εκπαιδευτή

Αναφέρεται επίσης και ως επιβλεπόμενη μάθηση. Σε αυτού του είδους την μάθηση μπορούμε να θεωρήσουμε ότι η επίβλεψη αποτελεί τη γνώση του περιβάλλοντος που θέλουμε να επιλύσουμε και αντιπροσωπεύεται από ένα σύνολο δειγμάτων εισόδου εξόδου. Να σημειώσουμε ότι το μοντέλο δεν γνωρίζει το ίδιο το περιβάλλον αλλά μόνο ένα κομμάτι του μέσω από τα δείγματα που του παρέχονται σε ζεύγη της μορφής είσοδος - επιθυμητή έξοδος. Έτσι το πρόβλημα λοιπόν ορίζεται ως εξής: Έχοντας X δεδομένα εισόδου και Υ δεδομένα εξόδου χρησιμοποιούμε κάποιον αλγόριθμο ώστε να κατασκευάσουμε συνάρτηση F του περιβάλλοντος τέτοια ώστε $\Upsilon = \mathrm{F}(\mathrm{X})$. Ο στόχος είναι να προσεγγίσουμε την συνάρτηση F όσο καλύτερα γίνεται έτσι ώστε για νέα δεδομένα εισόδου του περιβάλλοντος που δεν γνωρίζουμε η έξοδος του μοντέλου να προσεγγίζει το Y για την αντίστοιχη είσοδο. Η ονομασία μάθηση με εκπαιδευτή προέρχεται από τη διαδικασία της εκπαίδευσης. Ο εκπαιδευτής δηλαδή το σύνολο δεδομένων X,Υ δίνοντας στο μοντέλο κάποιο δείγμα , λαμβάνει μια πρόβλεψη και γνωρίζοντας την σωστή απάντηση y κάνει την αντίστοιχη διόρθωση στο μοντέλο. Η διαδικασία επαναλαμβάνεται μέχρι να φτάσουμε ένα επιθυμητό στάδιο απόδοσης καθώς ελαχιστοποιούμε μια αντικειμενική συνάρτηση κόστους.

1.4.2 Μάθηση Χωρίς εκπαιδευτή

Σε αντίθεση με το προηγούμενο είδος μάθησης όπως υποδηλώνει και η λέξη δεν υπάρχει εκπαιδευτής δηλαδή δεν έχουμε χαρακτηρισμένα παραδείγματα ή ζεύγη της λειτουργίας που πρέπει να μάθει να επιτελεί το μοντέλο. Έχουμε τις εξής δύο υποκατηγορίες:

Μη επιβλεπόμενη Μάθηση Στη μη επιβλεπόμενη μάθηση [1] δεν υπάρχει κάποιος εκπαιδευτής η κριτής που να επιβλέπει την διαδικασία μάθησης. Υπάρχει ένα ανεξάρτητο από την εργασία μέτρο της ποιότητας της αναπαράστασης που καλείται να μάθει το δίκτυο και οι ελεύθερες παράμετροι του μοντέλου βελτιστοποιούνται με βάση αυτό. Για ένα συγκεκριμένο και ανεξάρτητο από την εργασία μέτρο αφού το δίκτυο προσαρμοστεί στις στατιστικές κανονικότητες των δεδομένων εισόδου αναπτύσσει την δυνατότητα να να σχηματίζει εσωτερικές αναπαραστάσεις για την κωδικοποίηση χαρακτηριστικών της εισόδου και μέσω αυτών να δημιουργεί νέες κλάσεις αυτόματα. Συνήθως για την εκτέλεση αυτής της μάθησης χρησιμοποιούμε έναν κανόνα ανταγωνιστικής μάθησης.

Ενισχυτική Μάθηση Στην ενισχυτική μάθηση [1] η εκμάθηση της σχέσης F που εκφράζει την αντιστοίχιση εισόδου – εξόδου γίνεται μέσω συνεχούς αλληλεπίδρασης με το περιβάλλον. Το μοντέλο αφήνεται στο περιβάλλον και πραγματοποιώντας κάποιες δράσεις δέχεται επιβραβεύσεις ή ποινές με τη μορφή κάποιων σημάτων ανάλογα με την διαδικασία που του έχει ανατεθεί. Το feedback αυτό με τη μορφή πολλαπλών επαναλήψεων καθώς το μοντέλο επιδιώκει την μακροχρόνια αύξηση της επιβράβευσης και αποφυγή της ποινής οδηγεί με μαθηματική σύγκλιση στην εκμάθηση της ζητούμενης συμπεριφοράς. Ως κλάδος συγγενεύει σημαντικά με τη θεωρία παιγνίων, η θεωρία ελέγχου, η βελτιστοποίηση βάσει προσομοίωσης, τα συστήματα πολλαπλών πρακτόρων, γενετικοί αλγόριθμοι, στατιστικές κτλ.

Κεφάλαιο 2

Τεχνητά Νευρωνικά Δίκτυα

2.1 Σύντομη Ιστορική Αναδρομή

Το πρώτο άρθρο σχετικά με νευρωνικά δίκτυα δημοσιεύθηκε το από τον Warren McCulloch και τον Walter Pitts το 1943 όπου μοντελοποίησαν το πρώτο απλό νευρωνικό δίκτυο με ηλεκτρικά κυκλώματα. Το 1949 ο Donald Heb ενίσχυσε την παραπάνω ιδέα στο βιβλίο του "The Organization Behavior" όπου και έδειξε ότι οι διαδρομές μεταξύ των νευρώνων γίνονται πιο ισχυρές κάθε φορά που χρησιμοποιούνται.

Το 1958 ο Frank Rosenblatt ένας νευροβιολόγος απο το Cornell ξεκίνησε να δουλεύει πάνω στο γνωστό perceptron το πιο παλιό νευρωνικό το οποίο μάλιστα χρησιμοποιείται μέχρι σήμερα. Παρόλη την απλότητα στην σχεδίαση του με μόνο ένα επίπεδο νευρώνων αποδείχθηκε ικανό να κατηγοριοποιήσει με μεγάλη επιτυχία ένα σετ από συνεχής τιμές σε δύο κλάσεις. Η έξοδος του ήταν απλά ένα σταθμισμένο άθροισμα των εισόδων του.

Μόλις ένα χρόνο αργότερα Ο Bernard Widrow και Marcian Hoff από το standford ανέπτυξαν τα πρώτα μοντέλα που χρησιμοποιήθηκαν για την επίλυση πραγματικών προβλημάτων. Τα μοντέλα πήραν την ονομασία τους απο Adaptive Linear Elements και ονομάστηκαν αντίστοιχα Adaline και Madaline. Λειτούργησαν ως προσαρμοστικό φίλτρο για την εξάλειψη της ήχους στις τηλεφωνικές γραμμές.

Ακολούθησε ένα διάστημα μέχρι το 1981 όπου η έρευνα πάνω στα τεχνητά νευρωνικά δίκτυα διακόπηκε κυρίως λόγω των περιορισμένων δυνατοτήτων που θεωρήθηκαν ότι υπήρχαν. Το 1982 ο John Hopfield παρουσίασε ένα άρθρο με υποσχόμενες ιδέες που έστρεψε το ενδιαφέρων πάλι σε αυτά. Ακολούθησε το 1985 η καθιέρωση του πρώτου μεγάλου ετήσιου συνεδρίου για νευρωνικά δίκτυα. Αργότερα το 1997 προτάθηκε και το πρώτο αναδρομικό νευρωρνικό δίκτυο LSTM το οποίο είχε τη δυνατότητα διατήρησης μνήμης από προηγούμενα βήματα και επεξεργασία έτσι χρονοσειρών.

2.2 Ορισμός

Τα τεχνητά νευρωνικά δίκτυα (Artificial Neural Networks) αποτελούν υπολογιστικά μοντέλα των οποίων η λειτουργία είναι βασισμένη στους νευρώνες του ανθρωπίνου εγκεφάλου. Βασικό τους χαρακτηριστικό είναι η επίλυση διεργασιών χωρίς να απαιτούν ρητό προγραμματισμό. Έχουν τον ρόλο ενός εκτιμητή και μπορούν να προσεγγίσουν εξαιρετικά μη γραμμικές συναρτήσεις καθιστώντας τα βασικό εργαλείο για την επίλυση προβλημάτων στα οποία είναι από δύσκολο έως αδύνατο να δοθεί αναλυτική λύση.

2.3 Ο Απλός Νευρώνας

Βασικό δομικό κύτταρο των νευρωνικών δικτύων αποτελεί ο απλός νευρώνας, η δομή του οποίου παρουσιάζεται στο παρακάτω σχήμα.

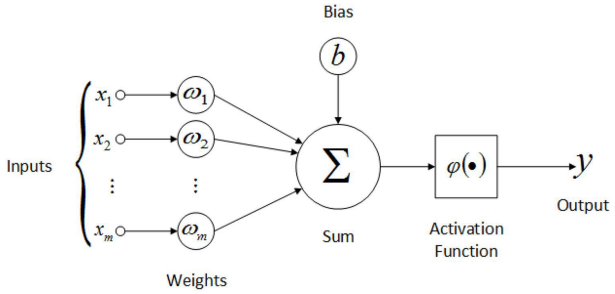

Figure 2.1: Ο Απλός Νευρώνας

Η έξοδος όπως βλέπουμε δίνεται από την παρακάτω σχέση:

$$y = \varphi\left(\sum_{i=1}^{i=n} x_i * w_i + b\right) \tag{2.1}$$

Αναλυτικά λοιπόν στη γενική περίπτωση έχουμε ως είσοδο ένα ν-διάστατο διάνυσμα εισόδου χ το οποίο εκτελεί εσωτερικό γινόμενο με ένα διάνυσμα βαρών w. Στο άθροισμα προστίθεται ένας επιπλέον όρος πόλωσης (bias) και το τελικό αποτέλεσμα περνάει από μια επιπλέον συνάρτηση η οποία καλείται συνάρτηση ενεργοποίησης του νευρώνα (activation function) ώστε να αποκτήσουμε την έξοδο y. Το επαυξημένο με τη πόλωση διάνυσμα βαρών λέμε ότι αποτελεί τις παραμέτρους του δικτύου και είναι αυτό το οποίο αλλάζει κατά τη διάρκεια της μάθησης όπως θα περιγράψουμε αργότερα.

2.4 Συνάρτηση Ενεργοποίησης Νευρώνα

Όπως περιγράψαμε παραπάνω, πριν την έξοδο, η πολλαπλασιασμένη με το διάνυσμα βαρών είσοδος, περνάει ως όρισμα σε μία συνάρτηση ενεργοποίησης φ.Ονομάζεται συνάρτηση ενεργοποίησης καθώς αυτή επιτρέπει στον νευρώνα να δώσει την επιθυμητή έξοδο ή να παραμείνει απενεργοποιημένος. Έχουν προταθεί πολλές διαφορετικές συναρτήσεις για αυτή τη λειτουργία και γενικότερα η επιλογή της αποτελεί κομμάτι της σχεδίασης του δικτύου που θέλουμε να σχεδιάσουμε με άλλες συναρτήσεις να πετυχαίνουν καλύτερα και άλλες χειρότερα αποτελέσματα ανάλογα τη λειτουργία και την υπόλοιπη αρχιτεκτονική. Αναφέρουμε ενδεικτικά τις συνηθέστερες συναρτήσεις ενεργοποίησης που υπάρχουν στη βιβλιογραφία.

Βηματική συνάρτηση ενεργοποίησης

$$y = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.2}$$

Σιγμοειδής συνάρτηση ενεργοποίησης

$$y = \frac{1}{1 - e^{-x}} \tag{2.3}$$

Έχει τα εξής πλεονεκτήματα:

- Ομαλή κλίση με αποτέλεσμα να αποτρέπει μεγάλα άλματα στην έξοδο του νευρώνα
- Οι τιμές τής εξόδου είναι μεταξύ 0 και 1 δηλαδή είναι κανονικοποιημένες
- Δίνει αρκετά καθαρές προβλέψεις καθώς για τιμές πάνω από το 2 η χαμηλότερα από το -2 η έξοδος είναι κοντά στο 1 ή στο 0 αντίστοιχα

Και τα εξής μειονεκτήματα:

- Για πολύ μεγάλες ή αντίστοιχα πολύ μικρές τιμές η αλλαγή στη κλίση είναι ανεπαίσθητη με αποτέλεσμα να δημιουργείται πρόβλημα στον αλγόριθμο οπισθοδιάδοσης που θα εξετάσουμε αργότερα.(vanishing gradient problem).
- Η έξοδος δεν είναι κεντραρισμένη στο 0.

Υπερβολική εφαπτομένη

$$y = tanh(x) \tag{2.4}$$

Όμοια συμπεριφορά με την την σιγμοειδή αλλά κεντραρισμένη στο μηδέν καθιστώντας πιο εύκολη την κωδικοποίηση εισόδων με ισχυρά θετικές , ουδέτερες αλλά και αρνητικές τιμές

ReLU (Rectified Linear Unit)

$$y = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} = max(0, x) \tag{2.5}$$

Αποτελεί ίσως τη δημοφιλέστερη συνάρτηση ενεργοποίησης και έχει παρατηρηθεί ότι βοηθάει το δίκτυο να συγκλίνει γρηγορότερα. Ως αδυναμία έχει το γεγονός ότι για εισόδους που πλησιάζουν το μηδέν η παράγωγος μηδενίζεται και μπορεί να δυσκολέψει έτσι τη μάθηση του δικτύου.

Leaky ReLU

$$y = max(0.1x, x) \tag{2.6}$$

Έχει παρόμοια συμπεριφορά με την ReLU αλλά υπάρχει μια θετική κλίση για αρνητικές τιμές. Αυτό μπορεί να βοηθήσει τον αλγόριθμο οπισθοδιάδοσης ακόμα και για αρνητικές τιμές αν και για αυτές τις τιμές οι προβλέψεις δεν μπορούν να θεωρηθούν σταθερές.

Παραμετρική ReLU

$$y = max(ax, x) \tag{2.7}$$

Εδώ η κλίση για το αρνητικό κομμάτι αποτελεί παράμετρο και μας δίνεται η δυνατότητα να μάθουμε στο μοντέλο την κατάλληλη τιμή α καθώς εκτελούμε τον αλγόριθμο οπισθοδιάδοσης

Softmax

$$y(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.8}$$

Αποτελεί συνάρτηση που χρησιμοποιείται κυρίως στο τελευταίο επίπεδο νευρώνων και συγκεκριμένα σε προβλήματα κατηγοριοποίησης όπου η έξοδος του νευρωνικού είναι μια κατανομή πιθανοτήτων με κάθε πιθανότητα να εκφράζει τη βεβαιότητα του δικτύου με την οποία κατατάσσει το δείγμα στην κάθε κλάση.

2.5 Δομή Απλού Νευρωνικού Δικτύου

Σχετικά με τη δομή ενός νευρωνικού δικτύου δηλαδή τον τρόπο με τον οποίο συνδέονται οι νευρώνες μεταξύ τους έχουμε δυο μεγάλες κατηγορίες:

- Δίκτυα πρόσθιας τροφοδότησης
- Δίκτυα με ανάδραση στα οποία μπορεί να υπάρχουν συνάψεις νευρώνων και με το προηγούμενο επίπεδο και όχι μόνο με το επόμενο.

Στη παράγραφο αυτή θα εξετάσουμε την πιο απλή δομή Νευρωνικού Δικτύου που μπορούμε να έχουμε, το Multilayer Perceptron η αλλιώς feedforward network.Αν και απλό από θέμα αρχιτεκτονικής και λειτουργίας θα μας βοηθήσει να παρουσιάσουμε παρακάτω περισσότερο σύνθετες υλοποιήσεις. Το δίκτυο μπορούμε να θεωρήσουμε ότι αποτελείται από τρία επίπεδα: Ένα στάδιο εισόδου το οποίο αποτελείται από νευρώνες οι οποίοι δέχονται απευθείας (η μετά από κάποια προεπεξεργασία) το διάνυσμα εισόδου χ , ένα ενδιάμεσο επίπεδο το οποίο ονομάζεται κρυμμένο επίπεδο (hidden layer) και από ένα επίπεδο εξόδου. Το ενδιάμεσο επίπεδο μπορεί να αποτελείται από πολλά επίπεδα νευρώνων και αποτελεί το στάδιο όπου γίνονται όλες οι μη γραμμικές απεικονίσεις μεταξύ των δεδομένων.

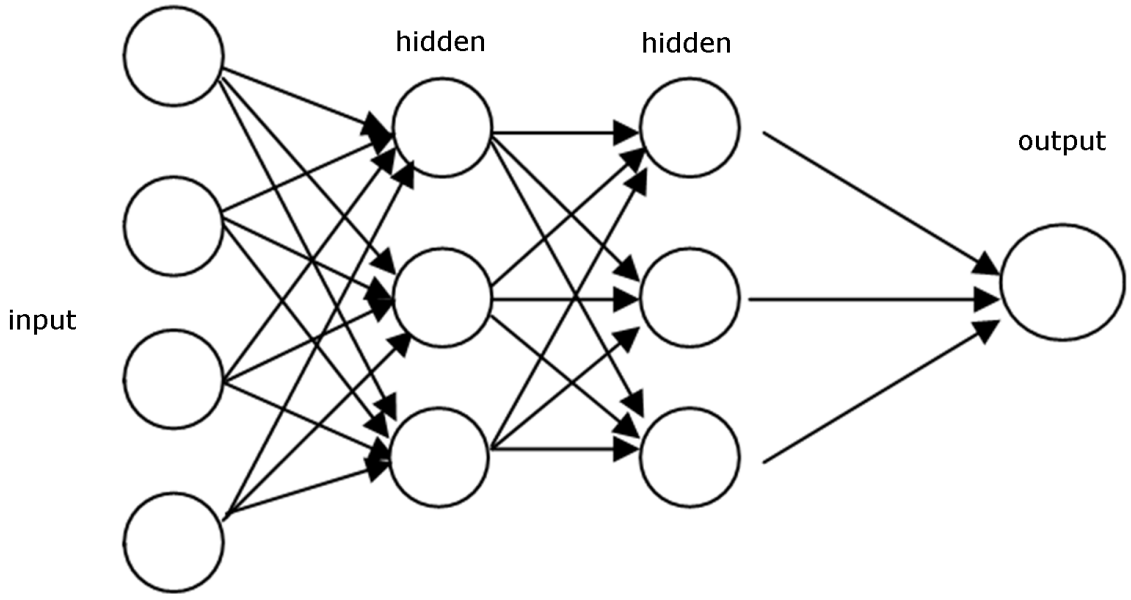

Figure 2.2: Feedforward Δίκτυο

Ουσιαστικά το Νευρωνικό Δίκτυο αναπαριστάται ως ένας γράφος όπου κάθε κόμβος(νευρώνας) συνδέεται με τους υπόλοιπους μέσω συνάψεων. Οι συνάψεις αυτές αποτελούν τις ακμές του γράφου και ονομάζονται βάρη του νευρωνικού δικτύου.

Όπως φαίνεται από το παραπάνω διάγραμμα η ροή πληροφορίας έχει μονή κατεύθυνση από την είσοδο προς την έξοδο εξ ου και το όνομα δίκτυο εμπρόσθιας διάδοσης. Το κρυφό επίπεδο στο παραπάνω νευρωνικό αποτελείται από δύο συστοιχίες νευρώνων, αριθμός ο οποίος όπως είπαμε μπορεί να διαφέρει. Όσο περισσότερο βαθύ είναι αυτό το στάδιο τόσο περισσότερο αυξάνεται και το βάθος των χαρακτηριστικών τα οποία εξάγονται.

Σε μητρική μορφή αν θέλουμε να παρουσιάσουμε την ροή πληροφορίας από ένα επίπεδο νευρώνων στο επόμενο έχουμε: Αν θεωρήσουμε ως $\boldsymbol{a}^{(l)}$ το διάνυσμα εξόδου από το l-οστό επιπέδου νευρώνων τότε το διάνυσμα $\boldsymbol{a}^{(l+1)}$ από την έξοδο του l+1 επιπέδου δίνεται ως:

$$\varphi\left(\begin{bmatrix} w_{00} & w_{01} & ... & w_{0n} \\ w_{11} & w_{11} & ... & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k0} & w_{k1} & ... & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(l)} \\ a_1^{(l)} \\ \vdots \\ a_n^{(l)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right) = \begin{bmatrix} a_0^{l+1} \\ a_1^{l+1} \\ \vdots \\ a_n^{l+1} \end{bmatrix} \tag{2.9}$$

Η σε συμπτυγμένη μορφή:

$$\varphi(\boldsymbol{W} \cdot \boldsymbol{a}^l + \boldsymbol{b}) = \boldsymbol{a}^{l+1} \tag{2.10}$$

όπου φ η συνάρτηση ενεργοποίησης για το συγκεκριμένο επίπεδο νευρώνων.
Τα βάρη w_{ij} υποδηλώνουν τη σύναψη του i-οστού νευρώνα στο l επίπεδο με τον j-οστό νευρώνα στο l+1 επίπεδο

2.6 Μέθοδοι Βελτιστοποίησης

Πριν εξετάσουμε τον τρόπο με τον οποίο γίνεται η ενημέρωση των βαρών του δικτύου θα παρουσιάσουμε την βασική αρχή με την οποία μαθαίνουν τα νευρωνικά δίκτυα. Οι αλγόριθμοι που χρησιμοποιούνται ονομάζονται μέθοδοι κλίσης.
Ο ορισμός του προβλήματος είναι ο εξής: ΄
Δοθείσας συνάρτησης κόστους $F : R^n \to R^m$ θέλουμε να βρούμε το

$$x_0 = argminF(x) \tag{2.11}$$

Για συναρτήσεις μίας μεταβλητής το πρόβλημα θεωρείται τετριμμένο αλλά σε πολυδιάστατους χώρους, η διάσταση των οποίων καθορίζεται από το πλήθος των βαρών w του δικτύου, στο πρόβλημά πολύ δύσκολα μπορεί να δοθεί κλειστή λύση,οπότε εφαρμόζουμε κάποιον επαναληπτικό προσεγγιστικό αλγόριθμό που μπορεί να μας εξασφαλίσει κάποια σύγκλιση. Η βασική λογική στην οποία στηρίζονται οι αλγόριθμοι κλίσης είναι ότι προκειμένου να καταλήξουμε στο ελάχιστο (η και μέγιστο) μιας συνάρτησης η οποία είναι καλώς ορισμένη και διαφορίσιμη δεν έχουμε παρά να κινηθούμε με γνώμονα τη κλίση από το σημείο στο οποίο βρισκόμαστε. Συγκεκριμένα ο αλγόριθμος **κατάβασης κλίσης (gradient descent)** αξιοποιεί ότι για

$$\mathbf{a_{t+1}} = \mathbf{a_t} - \gamma \nabla \mathbf{F}(\mathbf{a_t}) \tag{2.12}$$

τότε

$$F(\mathbf{a_{t+1}}) \leq \mathbf{F}(\mathbf{a_t}) \tag{2.13}$$

για κάποιο $\gamma \in R_+$
όπου:
$\boldsymbol{\alpha}_t$ το σημείο που βρισκόμαστε τη χρονική στιγμή t
$\boldsymbol{\alpha}_{t+1}$ το σημείο που βρισκόμαστε τη χρονική στιγμή t+1
γ το "βήμα" της μετακίνησης προς τη κατεύθυνση της παραγώγου
Με τη λογική αυτή ξεκινάμε με μία υπόθεση ελαχίστου έστω x_0 και σχηματίζουμε ακολουθία $x_0, x_1, ...x_n$
τέτοια ώστε

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \gamma_\mathbf{n} \nabla \mathbf{F}(\mathbf{x_n}), \mathbf{n} \geq \mathbf{0} \tag{2.14}$$

και έχουμε μονοτονική ακολουθία
$F(x_n) \leq F(x_{n-1}) \leq ... \leq F(x_0)$ Παρακάτω βλέπουμε ένα παράδειγμα κατάβασης κλίσης στο χώρο R^3 όπου η συνάρτηση που θέλουμε να ελαχιστοποιήσουμε αποτελεί ένα παραβολοειδές που εκτείνεται κάθετα προς τα έξω της σελίδας.

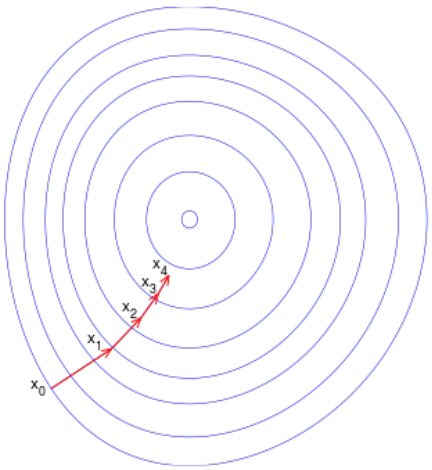

Figure 2.3: Κατάβαση κλίσης

Στο σημείο αυτό αξίζει να δώσουμε λίγο περισσότερη προσοχή στη παράμετρο γ η οποία είναι γνωστή και ώς ρυθμός μάθησης (learning rate).Για πολύ μικρές τιμές η σύγκλιση μπορεί να είναι απαγορευτικά αργή για τους διαθέσιμους υπολογιστικούς μας πόρους ενώ από την άλλη αρκετά μεγάλη τιμή μπορεί να οδηγήσει σε υπερύψωση (overshoot) με την οποία να έχουμε διαρκή ταλάντωση μεταξύ θέσεων δεξιά και αριστερά του ελαχίστου το οποίο ψάχνουμε.

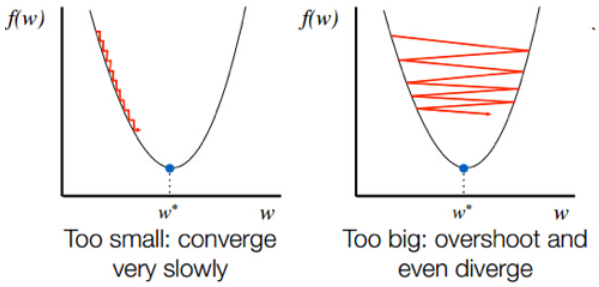

Figure 2.4: Ρυθμός μάθησης

2.7 Ο Αλγόριθμος Οπισθοδιάδοσης (Backpropagation)

Στην παράγραφο αυτή θα εξετάσουμε τον τρόπο με τον οποίο παίρνουν τις τιμές τους τα βάρη w_{ij} τα οποία συνδέουν τους νευρώνες όπως παρουσιάζεται και στο αντίστοιχο κεφάλαιο του [1] . Αποτελούν τις παραμέτρους του δικτύου και καθορίζουν την έξοδο του και γενικότερα την απόδοση του. Όπως αναφέραμε και στην προηγούμενη παράγραφο ο στόχος μας είναι η ελαχιστοποίηση μιας συνάρτησης κόστους L. Για την περιγραφή του αλγορίθμου ο συμβολισμός που θα χρησιμοποιήσουμε φαίνεται στο παρακάτω σχήμα:

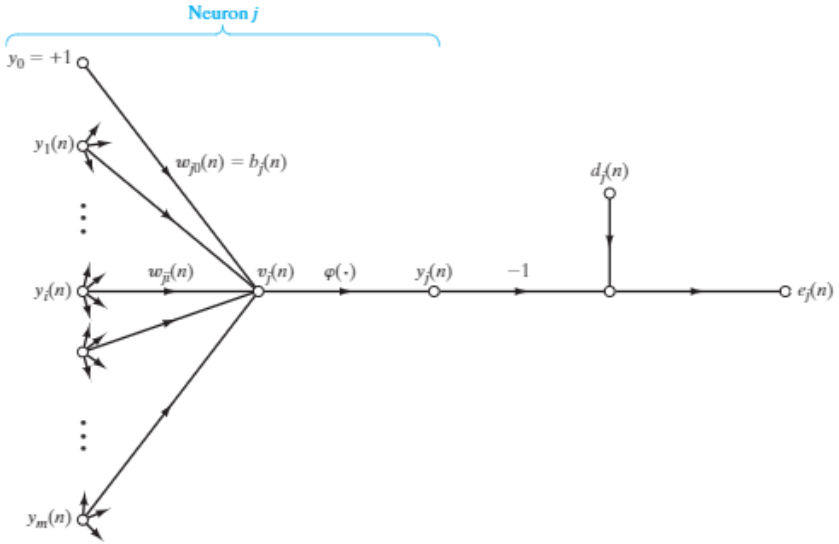

Figure 2.5: Ροή σήματος που περιγράφει τις λεπτομέρειες του νευρώνα εξόδου j

Ένα σύνολο σημάτων που παράγονται από νευρώνες στα αριστερά τροφοδοτούν τον νευρώνα j. Η είσοδος της συνάρτησης ενεργοποίησης του νευρώνα j είναι:

$$v_j = \sum_{i=0}^{m} w_{ji} y_i \tag{2.15}$$

Ο συνολικός αριθμός εισόδων που εφαρμόζονται στο νευρώνα j είναι m αν δεν συμπεριλάβουμε το κατώφλι. Επιπλέον το βάρος w_{j0} αποτελεί το κατώφλι η αλλιώς b_j που εφαρμόζεται στον νευρώνα j. Η έξοδος λοιπόν του νευρώνα δίνεται από τη σχέση:

$$y_j = \varphi_j(v_j) \tag{2.16}$$

Βασική ιδέα του αλγορίθμου οπισθοδιάδοσης αποτελεί η διόρθωση του βάρους w_{ji} με έναν όρο Δw_{ji} ο οποίος είναι ανάλογος με την μερική παράγωγο $\frac{\partial L}{\partial w_{ji}}$.
Για τον υπολογισμό αυτής της κλίσης εφαρμόζουμε τον κανόνα της αλυσίδας:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial e_j} \frac{\partial e_j}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}} \tag{2.17}$$

Η μερική παράγωγος λέμε εκφράζει έναν συντελεστή ευαισθησίας ο οποίος μας κατευθύνει στην αναζήτηση ελαχίστου στο χώρο των βαρών για το βάρος w_{ji}

Διαφορίζοντας και τις δύο πλευρές της εξίσωσης $E(n) = \sum_{j \in C} E_j(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$,που αποτελεί την συνολική στιγμιαία ενέργεια σφάλματος του δικτύου, ως προς e_j
παίρνουμε:

$$\frac{\partial L}{\partial e_j} = e_j \tag{2.18}$$

Διαφορίζοντας τώρα την εξίσωση του σφάλματος που παράγεται στην έξοδο του νευρώνα j
$e_j(n) = d_j(n) - y_j(n)$
ως προς y_j παίρνουμε:

$$\frac{\partial e_j}{\partial y_j} = -1 \tag{2.19}$$

Διαφορίζοντας την 2.16 ως προς την έξοδο του νευρώνα έχουμε:

$$\frac{\partial y_j}{\partial v_j} = \varphi_j^{'}(v_j) \tag{2.20}$$

Διαφορίζοντας την 2.15 ως προς w_{ji} παίρνουμε:

$$\frac{\partial v_j}{w_{ij}} = y_i \tag{2.21}$$

Η χρήση όλων των παραπάνω στη 2.17 μας δίνει:

$$\frac{\partial L}{\partial w_{ji}} = -e_j \varphi_j^{'}(v_j) y_i \tag{2.22}$$

Η διόρθωση τώρα που εφαρμόζεται Δw_{ji} προκύπτει από τον κανόνα της κλίσης όπως περιγράψαμε στη προηγούμενη παράγραφο:

$$\Delta w_{ji} = -\gamma \frac{\partial L}{\partial w_{ji}} \tag{2.23}$$

όπου γ ο ρυθμός μάθησης για τον αλγόριθμο ενώ το αρνητικό πρόσημο προκύπτει από τη λογική της κατάβασης κλίσης με κατεύθυνση προς το ελάχιστο της L στο χώρο των βαρών.Αντικαθιστώντας λοιπόν έχουμε:

$$\Delta w_{ji} = e_j \varphi_j^{'}(v_j) \tag{2.24}$$

Παρατηρούμε δηλαδή ότι παίζει σημαντικό ρόλο το σφάλμα e_j στην έξοδο του νευρώνα j.Πρέπει να γνωρίζουμε ποιοί νευρώνες συμπεριφέρονται σωστά ώστε να ανταμειφθούν ή σε αντίθετη περίπτωση να λάβουν ποινή. Διακρίνουμε λοιπόν 2 περιπτώσεις για τον τύπο του κάθε νευρώνα προκειμένου να υπολογίσουμε το σφάλμα e_j

Ο νευρώνας j είναι κόμβος εξόδου του δικτύου: Αυτή είναι η πιο απλή περίπτωση καθώς ο νευρώνας τροφοδοτείται απευθείας με την επιθυμητή έξοδο και ο υπολογισμός του σφάλματος είναι τετριμμένος. Για την διόρθωση χρησιμοποιούμε απευθείας τον τύπο 2.24

Ο νευρώνας j είναι ενδιάμεσος κόμβος: Σε αυτή τη περίπτωση δεν έχουμε απευθείας την επιθυμητή απόκριση για αυτόν τον νευρώνα καθώς όπως έχουμε αναφέρει τα δείγματα εκπαίδευσης είναι της μορφής (είσοδος,επιθυμητή έξοδος στο τέλος του δικτύου).Αν και αυτοί οι νευρώνες δεν είναι άμεσα προσπελάσιμοι έχουν "ευθύνη" για οποιοδήποτε σφάλμα προκύπτει στην έξοδο του δικτύου.Το σήμα σφάλματος για έναν τέτοιο νευρώνα θα καθοριστεί αναδρομικά ακολουθώντας πορεία από την έξοδο προς τα πίσω περνώντας από νευρώνες που συνδέονται άμεσα με τον κόμβο αυτόν. Η λειτουργία του κρυφού νευρώνα φαίνεται και στο παρακάτω διάγραμμα:

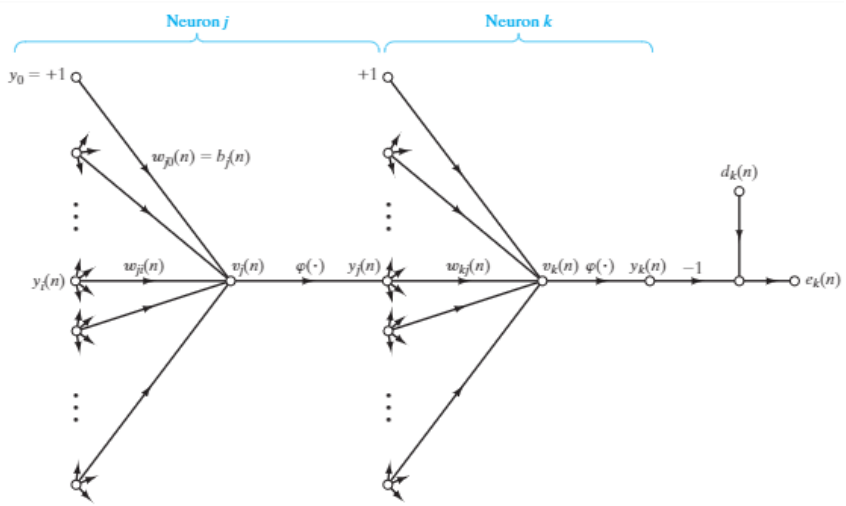

Figure 2.6: Λεπτομέρειες σύνδεσης του νευρώνα εξόδου k με τον κρυφό νευρώνα j

Αρχικά γράφουμε την τοπική κλίση δ_j της συνάρτησης λάθους για τον νευρώνα j ως:

$$\delta_j = -\frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial v_j} = -\frac{\partial L}{\partial y_j}\varphi_j^{'}(v_j) \tag{2.25}$$

Στο σημείο αυτό για να συνεχίσουμε θεωρούμε συνάρτηση λάθους L:

$$L = \frac{1}{2}\sum_{k\varepsilon C} e_k^2 \tag{2.26}$$

Δηλαδή το συνολικό σφάλμα προκύπτει ως άθροισμα των τετραγώνων των λαθών για τους νευρώνες εξόδου (σύνολο C).Διαφορίζοντας ως προς το σήμα y_j και χρησιμοποιώντας τον κανόνα της αλυσίδας λαμβάνουμε:

$$\frac{\partial L}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial v_k}\frac{\partial v_k}{\partial y_i} \tag{2.27}$$

Επιπλέον έχουμε:

$$e_k = d_k - y_k = d_k - \varphi_k(v_k) \Rightarrow \frac{\partial e_k}{\partial v_k} = -\varphi_k^{'}(v_k) \tag{2.28}$$

και

$$v_k = \sum_{j=0}^{m} w_{kj} y_j \tag{2.29}$$

όπου m ο αριθμός των νευρώνων που εφαρμόζονται στον νευρώνα k με το βάρος w_{j0} να είναι η πόλωση b_k που εφαρμόζεται στο νευρώνα k και η αντίστοιχη είσοδος είναι σταθερή και ίση με -1
Διαφορισμός της παραπάνω ως προς y_j προκύπτει:

$$\frac{\partial v_k}{\partial y_j} = w_{kj} \tag{2.30}$$

οπότε είμαστε σε θέση να γράψουμε την μερική παράγωγο ως:

$$\frac{\partial L}{\partial y_j} = -\sum_k e_k \varphi_\kappa^{'}(v_k) w_{kj} = -\sum_k \delta_\kappa w_{kj} \tag{2.31}$$

Συνδυάζοντάς όλες τις παραπάνω σχέσεις μπορούμε να γράψουμε την τοπική κλίση ως:

$$\delta_j = \varphi_j^{'}(v_j) \sum_k \delta_\kappa w_{kj} \tag{2.32}$$

Παρακάτω παρουσιάζουμε συγκεντρωτικά τα βήματα για την εκτέλεση του αλγορίθμου.

1. **Αρχικοποίηση**: Αρχικοποιούμε τα βάρη του δικτύου με τυχαίες τιμές από κάποια κατανομή.

2. **Πέρασμα των παραδειγμάτων εκπαίδευσης**: Για κάθε ένα δείγμα (η ομάδα δειγμάτων) εκτέλεσε ένα πέρασμα προς τα εμπρός και ένα πέρασμα προς τα πίσω όπως περιγράφεται παρακάτω.

3. **Πέρασμα προς τα εμπρός**: Έστω ότι έχουμε δείγμα εκπαίδευσης $(\mathbf{x}, \mathbf{d})$ όπου το διάνυσμα εισόδου $\mathbf{x}$ εφαρμόζεται στους νευρώνες εισόδου του δικτύου και οι επιθυμητές αποκρίσεις $\mathbf{d}$ στο στάδιο εξόδου.Υπολογίζουμε τα σήματα εξόδου του νευρώνα j στο επίπεδο l:
$$y_i^{(l)} = \varphi(\sum_i w_{ji}^{(l)} y_i^{(l-1)}) \tag{2.33}$$
με $y_i^{(l-1)}$: το σήμα εξόδου του νευρώνα i του προηγούμενου επιπέδου l-1 στην n-οστη επανάληψη και $w_{ji}^{(l)}(n)$ το βάρος μεταξύ του νευρώνα j στο επίπεδο l με τον νευρώνα i επίπεδο l-1
Για όλους τους j νευρώνες που βρίσκονται στο στάδιο εξόδου L υπολογίζουμε το σφάλμα:
$$e_j = d_j - y_j^{(L)} \tag{2.34}$$
με d_j η j-οστη συνιστώσα του διανύσματος $\mathbf{d}$

4. **Πέρασμα προς τα πίσω**: Υπολογίζουμε τις τοπικές κλίσεις ως:
$$\delta_j^l = \begin{cases} e_j^{(L)} \varphi_i^{'}(v_j^{(L)}) & \sigma\tau\delta\iota o - \varepsilon\xi\delta o u \\ \varphi_j^{'}(v_i^{(l)}) \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} & \varepsilon\nu\delta\iota\mu\varepsilon\sigma o - \sigma\tau\delta\iota o \end{cases} \tag{2.35}$$
Τα βάρη στη συνέχεια ενημερώνονται σύμφωνα με το κανόνα της κατάβασης κλίσης:
$$w_{jinew}^{(l)} = w_{jiold}^{(l)} + \gamma \delta_j^{(l)} y_i^{(l-1)} \tag{2.36}$$
όπου γ ο ρυθμός μάθησης.

5. **Επανάληψη**: Επαναλαμβάνουμε τα παραπάνω βήματα μέχρι να ικανοποιηθεί κάποιο κριτήριο σύγκλισης η χρονικού περιορισμού.

Συνολικά μπορούμε να πούμε ότι έχουμε ένα πέρασμα προς τα μπροστά όταν δίνουμε στο δίκτυο μια είσοδο ώστε να λάβουμε μια πρόβλεψη και στη συνέχεια χρησιμοποιώντας αυτή τη πρόβλεψη για τον καθορισμό του σφάλματος έχουμε ένα πέρασμα προς τα πίσω ενημερώνοντας τα βάρη για την ελαχιστοποίηση αυτού.

2.8 Συναρτήσεις Σφάλματος

Παρουσιάζουμε τώρα τις συνηθέστερες συναρτήσεις σφάλματος που χρησιμοποιούνται στις δυο μεγαλύτερες εφαρμογές νευρωνικών δικτύων.
Αρχικά εξετάζουμε την περίπτωση του προβλήματος **παλινδρόμησης (regression)** στο οποίο η έξοδος του νευρωνικού είναι μία συνεχής τιμή σε ένα διάστημα συνήθως κανονικοποιημένη στο [-1,1] ή [0,1] Η έξοδος του δικτύου για είσοδο χ δηλώνεται ως f(x) και η επιθυμητή απόκριση y.

- **Συνάρτηση τετραγωνικού σφάλματος**

$$L = \frac{1}{2}(y - f(x))^2 \tag{2.37}$$

  όπου ο όρος 1/2 υπάρχει απλά για ευκολότερο χειρισμό της παραγώγου ως προς το όρισμα.

- **Συνάρτηση απολύτου σφάλματος**

$$L = |y - f(x)| \tag{2.38}$$

- **Συνάρτηση Huber**

$$L = \begin{cases} \frac{1}{2}(y - f(x))^2 & |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \alpha\lambda\lambda\iota\varsigma \end{cases} \tag{2.39}$$

  Είναι πρακτικά συνδυασμός των δύο παραπάνω έτσι ώστε να βρισκόμαστε σε γραμμική περιοχή για μεγάλα σφάλματα ενώ σε τετραγωνική για μικρά.

Για προβλήματα **ταξινόμησης (classification)** στα οποία δηλαδή θέλουμε το δίκτυο να κατατάσσει τα δείγματα εισόδου του σε κάποιες κλάσεις έχουμε τις παρακάτω συναρτήσεις λάθους οι οποίες βασίζονται στην έννοια της εντροπίας. Για μια τυχαία μεταβλητή X με κατανομή p(x) η εντροπία ορίζεται ως:

$$S = \begin{cases} -\int p(x)log(p(x))dx & \text{x συνεχής} \\ -\sum_x p(x)log(p(x)) & \text{x διακριτή} \end{cases} \tag{2.40}$$

Όπου το αρνητικό πρόσημο χρησιμοποιείται προκειμένου να γίνει η συνολική ποσότητα θετική.
Υψηλές τιμές της εντροπίας φανερώνουν ασάφεια ως προς την κατανομή που ακολουθεί η τυχαία μεταβλητή.

- **Σε προβλήματα δυαδικής κατάταξης** όπου ένα δείγμα μπορεί να ανήκει σε μία από δύο κλάσεις ορίζουμε την συνάρτηση λάθους διασταυρούμενης εντροπίας ως:

$$L = -ylog(p) - (1-y)log(p) = \begin{cases} -log(1-p) & y = 0 \\ -log(p) & y = 1 \end{cases} \tag{2.41}$$

  όπου y οι δύο πιθανές κλάσεις και p η πιθανότητα στην έξοδο του δικτύου που εκφράζει τη σιγουριά του για την κατάταξη του δείγματος εισόδου σε αυτές τις κλάσεις.

- **Σε προβλήματα κατάταξης σε πολλαπλές κλάσης** έχουμε γενίκευση της προηγούμενης περίπτωσης ως εξής:

$$L(X_i, Y_i) = -\sum_{j=1}^{c} y_{ij} log(p_{ij}) \tag{2.42}$$

  όπου:
  c: ο αριθμός των κλάσεων
  X_i : Το διάνυσμα εισόδου
  Y_i : Το διάνυσμα επιθυμητής απόκρισης το οποίο ορίζεται ως $(y_{i1}, y_{i2}, ..., y_{ic})$ με

$$y_{ij} = \begin{cases} 1 & i_{th} \text{ element} \in \text{class-j} \\ 0 & \text{otherwise} \end{cases} \tag{2.43}$$

  $p_{ij} = f(X_i) =$ Πιθανότητα ότι το i-οστό στοιχείο ανήκει στη κλάση j
  Το διάνυσμα κατανομής πιθανοτήτων **p** το λαμβάνουμε με μια συνάρτηση softmax στο τελευταίο στάδιο του δικτύου.

Τέλος να σημειώσουμε την διαφορά μεταξύ συναρτήσεων λάθους και συναρτήσεων κόστους οι οποίες πολλές φορές συγχέονται. Οι συναρτήσεις λάθους όπως αναφέρθηκαν παραπάνω αφορούν μεμονωμένα δείγματα εισόδου εξόδου. Αντιθέτως οι συναρτήσεις κόστους αφορούν τον μέσο όρο των παραπάνω συναρτήσεων σε όλα τα δείγματα και αποτελεί το μέτρο που καλούμαστε να ελαχιστοποιήσουμε εφαρμόζοντας κατάβαση κλίσης και τον αλγόριθμο απισθοδιάδοσης.

2.9 Συνελικτικά νευρωνικά Δίκτυα

Θα ασχοληθούμε τώρα με μια ειδική αρχιτεκτονική νευρωνικών δικτύων με μεγάλη εφαρμογή στην όραση υπολογιστών. Στην κατηγοριοποίηση δηλαδή ενός μεγάλου συνόλου εικόνων, ανίχνευση αντικειμένων και ταυτοποίηση προσώπων. Η εμπειρία έχει δείξει ότι τα συνελικτικά νευρωνικά δίκτυα ως βάση είναι τα πλέον κατάλληλα για την αντιμετώπιση παιχνιδιών, αντιμετωπίζοντας την κατάσταση κατά κύριο λόγο ως εικόνα και μπορούν να αντιληφθούν και να μάθουν εξαρτήσεις και μοτίβα απευθείας από τα Pixel της εικόνας.
Η επεξεργασία εικόνας βασίζεται σε φίλτρα τα οποία μπορούν για παράδειγμα να εξάγουν τις άκρες και το περίγραμμα από ένα αντικείμενο. Τα φίλτρα αυτά εκτελούν με την εικόνα ένα συνελικτικό γινόμενο δηλαδή άθροισμα των πολλαπλασιασμένων στοιχείων προς στοιχείων σε κάθε τμήμα.Ένα τέτοιο φίλτρο το οποίο ανιχνεύει τις κάθετες ακμές (Vertical edge filter VEF) μπορεί να οριστεί μαθηματικά ως:

$$VEF = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.44}$$

Αν μετακινούμε δηλαδή το παραπάνω φίλτρο πάνω σε μια εικόνα δύο διαστάσεων και εκτελούμε συνελίξεις λαμβάνουμε τις κάθετες ακμές της εικόνας.Με αυτή τη λογική η παραπάνω μέθοδος επεκτείνεται έτσι ώστε τα στοιχεία του φίλτρου να είναι βάρη τα οποία καλείται να μάθει το νευρωνικό και να εξάγει κάθε φορά χαρακτηριστικά αντίστοιχα της διαδικασίας που θέλουμε να εκτελέσει. Τα χαρακτηριστικά αυτά δεν είναι πάντα προφανή ούτε εύκολα αντιληπτά από τον άνθρωπο.

Ένα συνελικτικό νευρωνικό δίκτυο αποτελείται από συνελικτικά και pooling επίπεδα.Πρωτού παρουσιάσουμε τις μαθηματικές σχέσεις που διέπουν το δίκτυο όπως παρουσιάζονται στο [2], αναφέρουμε κάποιες βασικές διεργασίες οι οποίες χρησιμοποιούνται.

Padding: Καθώς μετακινούμε τα φίλτρα πάνω στην εικόνα και εκτελούμε την συνέλιξη οι τιμές των ακριανών pixel της εικόνας χρησιμοποιούνται λιγότερες φορές σε σχέση με αυτές που βρίσκονται στο κέντρο της εικόνας. Αυτό σημαίνει ότι ενδεχομένως έχουμε κάποια απώλεια πληροφορίας κατά την εξαγωγή χαρακτηριστικών. Για να λύσουμε αυτό το πρόβλημα πολλές φορές προσθέτουμε μηδενικά γύρω από την εικόνα και αναπαριστούμε τον αριθμό των μηδενικών που προστίθεται σε κάθε μια από τις 4 πλευρές της εικόνας με τον αριθμό p.

Stride: Ορίζουμε ως stride (s) το βήμα μετακίνησης του φίλτρου πάνω στην εικόνα. Έτσι ένα μεγάλο βήμα μειώνει το μέγεθος της εικόνας εξόδου της συνέλιξης και αντίστροφα.
Οι εικόνες που επεξεργάζονται τα δίκτυα στην γενικότερη τους μορφή είναι τριών διαστάσεων δηλαδή:

$$dim(image) = (n_H, n_W, n_C)$$

όπου n_H το ύψος της εικόνας , n_W το πλάτος και n_C ο αριθμός των καναλιών. Για 3 κανάλια έχουμε RGB εικόνες (red,green,blue) ενώ για ένα κανάλι έχουμε ασπρόμαυρες εικόνες (grayscale). Έχει επικρατήσει το φίλτρο K να είναι τετραγωνικό και περιττής διάστασης f η οποία επιτρέπει το κάθε pixel να είναι στο κέντρο του φίλτρου και να λαμβάνονται υπόψιν στη συνέλιξη όλα τα γύρω του.Επίσης είναι απαραίτητο το φίλτρο να έχει τον ίδιο αριθμό καναλιών με την εικόνα $dim(filter) = (f, f, n_C)$

Pooling Αποτελεί το βήμα στο οποίο πραγματοποιούμε μείωση της διάστασης των εξαγόμενων χαρακτηριστικών από τη συνέλιξη και είναι μια πράξη η οποία γίνεται κατά μήκος των καναλιών και επηρεάζει μόνο τις δύο πρώτες διαστάσεις της εικόνας.Δεδομένης της εικόνας πραγματοποιούμε μια μετακίνηση φίλτρου χωρίς παραμέτρους προς μάθηση και με δεδομένο βήμα εφαρμόζουμε μια συνάρτηση φ στα στοιχεία που καλύπτει το φίλτρο. Οι δύο συνηθέστερες συναρτήσεις είναι η average pooling η οποία επιστρέφει τον μέσο όρο των στοιχειών και η max pooling που επιστρέφει απλά το μέγιστο.

Συνέλιξη Για εικόνα I και φίλτρο K έχουμε:

$$conv(I,K)(x,y) = \sum_{i=1}^{n_H}\sum_{j=1}^{n_W}\sum_{k=1}^{n_C} K_{i,j,k} I_{x+i-1,y+j-1,k} \tag{2.45}$$

ενώ οι διαστάσεις της εξόδου δίνονται από:

$$dim(conv(I,K)) = (\lfloor \frac{n_H + 2p - f}{s} + 1 \rfloor), \lfloor \frac{n_W + 2p - f}{s} + 1 \rfloor)$$

όπου $\lfloor x \rfloor$ η στρογγυλοποίηση προς τα πάνω του x

Είμαστε σε θέση τώρα να παρουσιάσουμε τα κομμάτια τα οποία αποτελούν ένα συνελικτικό νευρωνικό δίκτυο (CNN)

Συνελικτικό επίπεδο

Στο επίπεδο l έχουμε τα παρακάτω:

- είσοδος: a^{l-1} με διαστάσεις $(n_H^{l-1}, n_W^{l-1}, n_C^{l-1})$ με a^0 η είσοδος στο αρχικό στάδιο.
- Padding: p^l , stride s^l
- αριθμός φίλτρων n_C^l όπου κάθε K^n έχει διαστάσεις (f^l, f^l, n_C^{l-1})
- πόλωση της n-οστης συνέλιξης: b_n^l
- συνάρτηση ενεργοποίησης ψ^l
- έξοδος a^l με διαστάσεις (n_H^l, n_W^l, n_C^l)

Έτσι $\forall n \in [1,2,..n_C^l]$ έχουμε:

$$conv(a^{l-1}, K^n)_{x,y} = \psi^l \left(\sum_{i=1}^{n_H^{l-1}} \sum_{j=1}^{n_W^{l-1}} \sum_{k=1}^{n_C^{l-1}} K_{i,j,k}^n a_{x+i-1,y+j-1,k}^{l-1} + b_n^l \right)$$
$$dim(conv(a^{l-1}, K^n)) = (n_H^l, n_W^l)$$

οπότε:

$$a^l = [\psi^l(conv(a^{l-1}, K^1)), [\psi^l(conv(a^{l-1}, K^2))..., \psi^l(conv(a^{l-1}, K^{n_C^l}))]$$
$$dim(a^l) = (n_H^l, n_W^l, n_C^l)$$

με:

$$n^l_{H/W} = \left(\lfloor \frac{n^{l-1}_{H/W} + 2p^l - f^l}{s^l} + 1 \rfloor \right)$$
$$n^l_C = \text{αριθμός φίλτρων}$$

οι παράμετροι προς μάθηση αποτελούν τα φίλτρα και τα αντίστοιχα κατώφλια b.

Pooling επίπεδο

Το επίπεδο αυτό όπως εξηγήσαμε μειώνει τη διάσταση της εικόνας χωρίς να επηρεάζει τον αριθμό των καναλιών.Συγκεκριμένα έχουμε:

- είσοδος: a^{l-1} με διαστάσεις $(n^{l-1}_H, n^{l-1}_W, n^{l-1}_C)$ με a^0 η είσοδος στο αρχικό στάδιο.
- Padding: p^l (χρησιμοποιείται σπάνια) , stride s^l
- Διάσταση του φίλτρου: f^l
- συνάρτηση pooling: φ^l
- έξοδος: a^l με διαστάσεις $(n^l_H, n^l_W, n^l_C = n^{l-1}_C)$

και έχουμε:

$$a^l_{x,y,z} = pool(a^{l-1})_{x,y,z} = \varphi^l((a^{l-1}_{x+i-1,y+j-1,z})_{(i,j)\in[1,2,..,f^l]^2})$$
$$dim(a^l) = (n^l_H, n^l_W, n^l_C)$$

με

$$n^l_{H/W} = \left(\lfloor \frac{n^{l-1}_{H/W} + 2p^l - f^l}{s^l} + 1 \rfloor \right)$$
$$n^l_C = n^{l-1}_C$$

Αυτά τα επίπεδα στο τέλος συνήθως καταλήγουν σε ένα απλό feedforward επίπεδο όπως παρουσιάστηκε στην προηγούμενη παράγραφο και χειριζόμαστε τα χαρακτηριστικά που έχουν εξάγει τα συνελικτικά επίπεδα ως ένα μονοδιάστατο διάνυσμα.Η Λογική είναι ότι σε ένα βαθύ νευρωνικό δίκτυο όσο πιο βαθιά πάμε μειώνουμε τις διαστάσεις της εικόνας και αυξάνουμε τον αριθμό των φίλτρων δηλαδή τα χαρακτηριστικά που εξάγουμε.

Κεφάλαιο 3

Ενισχυτική Μάθηση

Η κυρίαρχη διαφορά στην ενισχυτική από την επιβλεπόμενη μάθηση είναι ότι μετά τον ορισμό του ευφυούς πράκτορα αυτός αφήνεται σε ένα περιβάλλον να αλληλεπιδράσει και τα ερέθισμα που δέχεται από αυτό είναι που τον βοηθάνε στις μετέπειτα κινήσεις του χωρίς να του παρέχεται κάποια έτοιμη βάση δεδομένων. Παραδείγματα ενισχυτικής μάθησης αποτελούν:

- Η εκμάθηση ενός ελικοπτέρου στην εκτέλεση περίπλοκων κινήσεων στον αέρα.
- Η εκμάθηση ενός ευφυούς πράκτορα στη διαχείριση και επένδυση κεφαλαίου σε χρηματιστηριακά περιβάλλοντα.
- Διαχείριση και έλεγχος ενός σταθμού παροχής ενέργειας
- Πράκτορες που παίζουν βιντεοπαιχνίδια με υψηλό επίπεδο ικανοτήτων.

Ο συμβολισμός και η περιγραφή που ακολουθούμε βασίζεται στο έργο των [3] και [4]

3.1 Ορολογία Ενισχυτικής Μάθησης

3.1.1 Το περιβάλλον του Πράκτορα

Στην ενισχυτική μάθηση έχουμε το παρακάτω γενικότερο σχεδιάγραμμα.

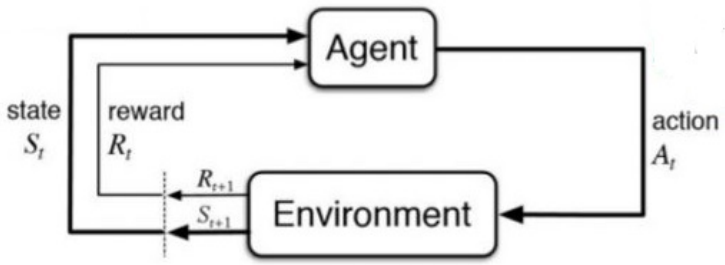

Figure 3.1: Σχεδιάγραμμα ενισχυτικής μάθησης

- R_t : Είναι το σήμα ανάδρασης τη χρονική στιγμή t που δέχεται ο πράκτορας απο το περιβάλλον.Είναι γνωστό ως αμοιβή(reward signal) και μας δείχνει το πόσο καλά τα πηγαίνει ο πράκτορας μας την χρονική στιγμή t λαμβάνοντας θετικές τιμές +v για κάποια θετική δράση του σχετικά με τη λειτουργία που θέλουμε να επιτελέσει και -v αλλιώς.

- O_t ¨: Αποτελεί την παρατήρηση του πράκτορα από το περιβάλλον την χρονική στιγμή t.

- A_t : Είναι η δράση που εκτελεί ο πράκτορας την χρονική στιγμή t στο περιβάλλον του.

- $H_t = O_1R_1A_1.....A_{t-1}O_tR_t$ αποτελεί το ιστορικό του πράκτορα με την ακολουθία όλων των παρατηρήσεων,αμοιβών και δράσεων έως την χρονική στιγμή t.

- $S_t = f(H_t)$ και αποτελεί την κατάσταση (state) του πράκτορα στο περιβάλλον. Στην γενικότερη περίπτωση αποτελεί συνάρτηση του ιστορικού του και περιέχει όλη τη χρήσιμη πληροφορία για τους αλγορίθμους ενισχυτικής μάθησης.

Στην ακολουθιακή λήψη αποφάσεων ο στόχος είναι να να μεγιστοποιήσουμε την ανταμοιβή λαμβάνοντας όμως υπόψιν ότι οι δράσεις μας έχουν και μακροπρόθεσμες επιπτώσεις και η αμοιβή μπορεί να έρχεται με καθυστέρηση. Οι σωστά εκπαιδευμένοι ευφυείς πράκτορες θα θυσιάσουν άμεση αμοιβή για μεγαλύτερη επιτυχία στο μέλλον.Στη συνέχεια θα ασχοληθούμε αποκλειστικά με μαρκοβιανές καταστάσεις.

Ορισμός

Μια κατάσταση S_t ονομάζεται μαρκοβιανή αν και μόνο αν:

$$P(S_t|S_1, S_2, ...S_{t-1}) = P(S_t|S_{t-1}) \tag{3.1}$$

Αυτό πρακτικά σημαίνει ότι η πιθανότητα να βρεθούμε στη κατάσταση S_t την χρονική στιγμή t εξαρτάται μόνο από την κατάσταση στην οποία βρισκόμασταν την ακριβώς προηγούμενη στιγμή και όχι από το υπόλοιπο ιστορικό καταστάσεων.

3.1.2 Στοιχεία ενός πράκτορα ενισχυτικής μάθησης

Τα βασικότερα στοιχεία ενός ευφυούς πράκτορα ενισχυτικής μάθησης είναι τα εξής:

- **Πολιτική (policy)**: Μοντελοποιεί την συμπεριφορά του πράκτορα και αντιστοιχίζει καταστάσεις σε δράσεις.

  $$\pi(\alpha|s) = P(A_t = \alpha|S_t = s) \tag{3.2}$$

  Δηλαδή μια κατανομή πιθανοτήτων για όλες τις δράσεις α που μπορεί να εκτελέσει ο πράκτορας ενώ βρίσκεται στην κατάσταση s.

- **Συνάρτηση Αμοιβής (Reward function)**: Αποτελεί μια πρόβλεψη της μελλοντικής αμοιβής με την οποία αξιολογούμε κατά πόσο συμφέρει να βρισκόμαστε στη κάθε κατάσταση και διαλέγουμε δράσεις αναλόγως.

  $$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + ... + \gamma^2 R_{t+3}] \tag{3.3}$$

  Δηλαδή λαμβάνουμε τη μέση αμοιβή όλων των αμοιβών αν ακολουθήσουμε πολιτική π από την κατάσταση s. Η παράμετρος γ λειτουργεί ως συντελεστής απόσβεσης για λόγους σύγκλισης και λαμβάνει τιμές μικρότερες από 1 κάνοντας έτσι το πράκτορα να λαμβάνει υπόψιν του περισσότερο τις άμεσες αμοιβές παρά τις ασαφείς αμοιβές του μακρινού μέλλοντος.

- **Μοντέλο (model)** το οποίο προσομοιάζει τη συμπεριφορά του περιβάλλοντος.

$$P^a_{ss'} = P[S_{t+1} = s' | A_t = a, S_t = s] \tag{3.4}$$

και

$$R^a_s = E[R_{t+1} | S_t = s, A_t = a] \tag{3.5}$$

όπου το P μας προβλέπει την επόμενη κατάσταση και το R μας προβλέπει την άμεση αμοιβή τής επόμενης χρονικής στιγμής.

Ένας ευφυής πράκτορας μπορεί να είναι βασισμένος είτε στη πολιτική είτε στη συνάρτηση αμοιβής είτε και στις δύο (actor-critic). Επιπλέον μπορεί να διαθέτει μοντέλο περιβάλλοντος(model based agent) η και όχι (model free agent).

3.1.3 Το δίλημμα της εξερεύνησης-εκμετάλλευσης (exploration exploitation dilemma)

Σαν διαδικασία η ενισχυτική μάθηση απαιτεί προσαρμογή στο λάθος μετά από κάθε προσπάθεια (trial and error). Η προσαρμογή στο λάθος συνίσταται στη βελτίωση της πολιτικής που ακολουθούμε και της συνάρτησης αμοιβής. Κατά τη διάρκεια της εκπαίδευσης θέλουμε να αξιοποιούμε την υπάρχουσα πολιτική που διαθέτει ο πράκτορας από την εκπαίδευση (exploitation) του ενώ ταυτόχρονα θέλουμε να ενθαρρύνουμε τον πράκτορα και σε νέες δράσεις που δεν έχει δοκιμάσει μέχρι στιγμής και μπορεί να αποδειχθούν καλύτερες (exploration).

3.2 Μαρκοβιανές Διαδικασίες

Όπως αναφέραμε και προηγουμένως θα ασχοληθούμε με καταστάσεις οι οποίες ικανοποιούν την μαρκοβιανή ιδιότητα 3.1. Το μοναδικό πράγμα που μας ενδιαφέρει για πιθανές μελλοντικές καταστάσεις είναι μόνο η παρούσα κατάσταση. Τα στοιχεία $P_{ss'}$ σχηματίζουν τον παρακάτω πίνακα μετάβασης:

$$P = \begin{bmatrix} P_{11} & \dots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \dots & P_{nn} \end{bmatrix} \tag{3.6}$$

Όπου ο πρώτος υποδείκτης δείχνει σε ποια κατάσταση βρισκόμαστε και ο δεύτερος σε ποία πρόκειται να πάμε.

Μαρκοβιανή διαδικασία η Μαρκοβιανή αλυσίδα καλείται μια δυάδα της μορφής (S, P) όπου:
S ένα πεπερασμένο σύνολο από καταστάσεις που ικανοποιούν την μαρκοβιανή ιδιότητα και
P ο πίνακας με τις πιθανότητες μετάβασης. $P_{ss'} = P[S_{t+1} = s' | S_t = s]$
Μπορούμε να επεκτείνουμε τον παραπάνω ορισμό και να τον προσαρμόσουμε στις έννοιες της ενισχυτικής μάθησης:

Μια **Μαρκοβιανή αλυσίδα αποφάσεων (Markov decision Process) MDP** καλείται μια πεντάδα (S, A, P, R, γ) όπου

- S¨ένα πεπερασμένο σύνολο καταστάσεων που ικανοποιούν την μαρκοβιανή ιδιότητα.

- Ρ Ο πίνακας με τις πιθανότητες μετάβασης. Αν ακολουθούμε πολιτική π τότε οι ζητούμενες πιθανότητες δίνονται ως:

$$P^{\pi}_{ss'} = \sum_{a \in A} \pi(\alpha|s) P^{a}_{ss'} \tag{3.7}$$

- Α: ένα πεπερασμένο σύνολο από δράσεις.

- $$R^{a}_{s} = E[R_{t+1}|S_t = s, A_t = a] \tag{3.8}$$

  Η μέση τιμή της αμοιβής που λαμβάνει ο πράκτορας αν εκτελέσει δράση α από τη κατάσταση s. Αν ακολουθούμε πολιτική π τότε αθροίζοντας πάνω σε όλες τις δυνατές δράσεις έχουμε:

$$R^{\pi}_{s} = \sum_{a \in A} \pi(\alpha|s) R^{a}_{s} \tag{3.9}$$

- γ ο συντελεστής απόσβεσης με $\gamma \in [0, 1]$

Στο σημείο αυτό ορίζουμε την "επιστροφή" (return) ως:

$$G_t = R_{t+1} + R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^{\kappa} R_{t+k+1} \tag{3.10}$$

Σε μια MDP η συνάρτηση αμοιβής(state value function) ορίζεται ως:

$$V_{\pi}(s) = E_{\pi}[G_t|S_t = s] \tag{3.11}$$

Επίσης εδώ ορίζουμε μια νέα συνάρτηση την συνάρτηση δράσης-αμοιβής (action value function) ως:

$$q_{\pi}(s, a) = E_{\pi}[G_t|S_t = s, A_t = a] \tag{3.12}$$

η οποία συμπεριλαμβάνει στη συνάρτηση αμοιβής την δράση α.Και οι δύο παραπάνω συναρτήσεις μπορούν να αποσυντεθούν ως:

$$V_{\pi}(s) = E_{\pi}[R_{t+1} + \gamma V_{\pi}(S_{t+1})|S_t = s] \tag{3.13}$$

και:

$$q_{\pi}(s, \alpha) = E_{\pi}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{3.14}$$

Μπορούμε να γράψουμε τώρα την εξίσωση **Bellman** η οποία αποτελεί θεμελιώδη σχέση που χρησιμοποιείται για την επίλυση MDPs.

$$V_{\pi}(s) = \sum_{a \in A} \pi(a|s) q_{\pi}(s, a) \tag{3.15}$$

και αντίστοιχα για τη συνάρτηση δράσης-αμοιβής:

$$q_{\pi}(s, a) = R^{a}_{s} + \gamma \sum_{s' \in S} P^{a}_{ss'} V_{\pi}(s') \tag{3.16}$$

Αντικαθιστώντας την μία στην άλλη και αντίστροφα λαμβάνουμε

$$V_{\pi}(s) = \sum_{a \in A} \pi(a|s) \left(R^{a}_{s} + \gamma \sum_{s' \in S} P^{a}_{ss'} V_{\pi}(s') \right) \tag{3.17}$$

και

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \left(\sum_{a' \in A} \pi(a'|s') q_\pi(s',a') \right) \tag{3.18}$$

Βέλτιστη συνάρτηση αμοιβής:
Είναι πρακτικά η μέγιστη V για όλες τις πολιτικές π.

$$V_*(s,a) = \max_\pi V_\pi(s) \tag{3.19}$$

και ομοίως για την **βέλτιστη συνάρτηση q**:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{3.20}$$

και εκφράζουν την καλύτερη στρατηγική σε μια MDP προκειμένου να λάβουμε τη μεγαλύτερη δυνατή αμοιβή.Με αυτή τη λογική η βέλτιστη πολιτική είναι αυτή που μας δίνει την βέλτιστη V δηλαδή:
$\pi \geq \pi'$ αν $V_\pi(s) \geq V_{\pi'}(s) \forall s$
Αποδεικνύεται ότι για κάθε MDP υπάρχει μία βέλτιστη πολιτική π_* με την οποία πετυχαίνουμε τις βέλτιστες συναρτήσεις q και V. Μια βέλτιστη πολιτική μπορεί να βρεθεί μεγιστοποιώντας πάνω στη $q_*(s,a)$ ως εξής:

$$\pi_*(a|s) = \begin{cases} 1 & a = \underset{a \in A}{\operatorname{argmax}}\, q_*(s,a) \\ 0 & \alpha\lambda\lambda\iota\varsigma \end{cases} \tag{3.21}$$

Υπάρχει πάντα μια ντετερμινιστική βέλτιστη πολιτική για κάθε MDP και εφόσον γνωρίζουμε την βέλτιστη q_* μπορούμε να την εξάγουμε άμεσα. Οι βέλτιστες συναρτήσεις q και v συνδέονται αναδρομικά μέσω της εξίσωσης βελτιστοποίησης Bellman. Συνδυάζοντάς τις:

$$\begin{cases} V_*(s) = \max_a q_*(s,a) \\ q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s) \end{cases}$$

λαμβάνουμε

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a') \tag{3.22}$$

Η παραπάνω εξίσωση είναι γνωστή ως Bellman Optimality equation και στις επόμενες παραγράφους θα ασχοληθούμε με επαναληπτικές μεθόδους επίλυσης της λόγω της μη γραμμικότητας της και της δυσκολίας της να βρεθεί λύση με κλειστή μορφή.

3.3 Δυναμικός Προγραμματισμός

Ο Δυναμικός προγραμματισμός αποτελεί έναν τρόπο επίλυσης της εξίσωσης Bellman. Η βασική ιδέα είναι ότι σπάμε το πρόβλημα σε υποπροβλήματα τα οποία λύνονται ευκολότερα και ο συνδυασμός των οποίων μπορεί να μας δώσει τη συνολική λύση.Για να μπορεί ένα πρόβλημα να λυθεί με μεθόδους δυναμικού προγραμματισμού θα πρέπει να ικανοποιεί τις παρακάτω ιδιότητες.

- Η βέλτιστη λύση να μπορεί να αποσυντεθεί σε υποπροβλήματα
- Τα υποπροβλήματα να εμφανίζονται πολλές φορές και η λύση τους να μπορεί να αποθηκεύεται και να επαναχρησιμοποιείται.

Αυτοί οι αλγόριθμοι απαιτούν πλήρη γνώση του περιβάλλοντος δηλαδή των MDPs και χρησιμοποιούνται για σχεδιασμό πάνω σε αυτό. Επειδή στη πλειοψηφία των εφαρμογών το μοντέλο του περιβάλλοντος δεν μας είναι πλήρως γνωστό ο δυναμικός προγραμματισμός δεν χρησιμοποιείται ιδιαίτερα σήμερα άλλα μας βοηθάει να κατανοήσουμε την ενισχυτική μάθηση. Οι MDPs ικανοποιούν τα παραπάνω με την εξίσωση bellman να έχει την απαιτούμενη αναδρομική μορφή και η συναρτήσεις V και q να αποθηκεύουν και να επαναχρησιμοποιούν τιμές.
Θα ασχοληθούμε με δύο προβλήματα:

- **Το πρόβλημα της πρόβλεψης (prediction problem)**
  Για είσοδο μια MDP $< S, A, P, R, \gamma >$ και πολιτική π ψάχνουμε την συνάρτηση V.
- **Το πρόβλημα του ελέγχου (control problem)**
  Για είσοδο μια MDP $< S, A, P, R, \gamma >$ ψάχνουμε την βέλτιστη V_* και π_*

3.3.1 Παράλληλος Δυναμικός Προγραμματισμός

Ονομάζεται παράλληλος γιατί όλες οι ενημερώσεις των καταστάσεων γίνονται ταυτόχρονα.

Επαναληπτική αξιολόγηση πολιτικής (iterative policy evaluation) Έχουμε δεδομένη την πολιτική π και κατασκευάζουμε την συνάρτηση v. Ο αλγόριθμός εκτελείται ως εξής:

1. Για κάθε επανάληψη κ+1
2. Για όλα τα $s \in S$
3. ενημέρωση του $V_{k+1}(s)$ από το $V_k(s')$ όπου s' θυγατρική κατάσταση της s

Αποδεικνύεται ότι η διαδικασία αυτή συγκλίνει στην V_π. Η ενημέρωση στο βήμα 3 γίνεται μέσω της εξίσωσης Bellman.

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s') \right)$$

Η απλούστερα σε μορφή μητρών:

$$\mathbf{V^{k+1}} = \mathbf{R}^\pi + \gamma \mathbf{P}^\pi \mathbf{V^k}$$

Βελτίωση μιας πολιτικής (policy improvement) Θέλουμε να ενημερώσουμε μια πολιτική π προς την κατεύθυνση της π_*. Τα βήματα που ακολουθούμε είναι τα εξής:

- Αξιολογούμε την π μέσω της τιμής της συνάρτησης V.
  $V_\pi(s) = E_\pi[G_t|S_t = s]$
- Βελτιώνουμε ακολουθώντας μια άπληστη στρατηγική $\pi' = greedy(V_\pi)$

Πρακτικά αυτό σημαίνει ότι η πολιτική κατασκευάζεται ως εξής:
Βρισκόμαστε σε ένα state και απλά πραγματοποιούμε την δράση που μας πηγαίνει με μεγαλύτερη πιθανότητα στη πιο συμφέρουσα κατάσταση s' με βάση το $V_\pi(s)$ που έχουμε υπολογίσει από το προηγούμενο βήμα. Σχηματικά δηλαδή:

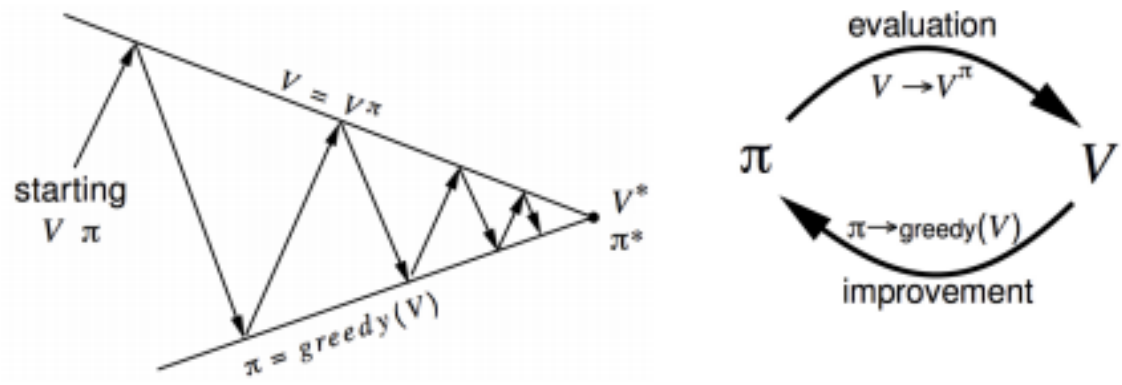

Figure 3.2: Policy Iteration

Αναλυτικότερα για το βήμα της βελτίωσης αν έχουμε μια ντετερμινιστική πολιτική $\pi(s)$ τότε η βελτίωση της πολιτικής γίνεται ώς $\pi'(s) = \underset{a\in A}{\operatorname{argmax}}\, q_\pi(s,a)$. Έτσι βελτιώνεται η αξία για κάθε κατάσταση s στο βήμα t.Δηλαδή:

$$q_\pi(s,\pi(s')) = \max_{a\in A} q_\pi(s,a) \geq q_\pi(s,\pi(s)) = V_\pi(s)$$

Έχουμε $V_{\pi'}(s) \geq V_\pi(s)$ και αποδεικνύεται ώς:

$$\begin{aligned}
&V_\pi(s) \leq q_\pi(s,\pi(s')) = \\
&E_{\pi'}[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s] \leq \\
&E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1},\pi'(S_{t+1})|S_t = s] \leq \\
&E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2},\pi'(S_{t+2})|S_t = s]
\end{aligned}$$

Όταν οι βελτιώσεις σταματήσουν τότε τερματίζουμε την διαδικασία και παίρνουμε:

$$q_\pi(s,\pi(s')) = \max_{a\in A} q_\pi(s,a) = q_\pi(s,\pi(s)) = V_\pi(s)$$

και η εξίσωση βελτιστοποίησης Bellman ικανοποιείται οπότε αφού $V_*(s) = V_\pi(s)\forall S$ τότε η πολιτική π είναι βέλτιστη.

Μια βέλτιστη πολιτική αποτελείται από μια βέλτιστη αρχική δράση A_* η οποία ακολουθείται από μια βέλτιστη πολιτική από την θυγατρική κατάσταση που προκύπτει S'.
Διατυπώνουμε το παρακάτω θεώρημα.

Θεώρημα 1. *Μία πολιτική $\pi(\alpha|s)$ πετυχαίνει την βέλτιστη συνάρτηση $V_\pi(s) = V_*(s)$ στη κατάσταση s αν και μόνο αν για κάθε κατάσταση s' προσβάσιμη από την κατάσταση s η πολιτική π πετυχαίνει τη βέλτιστη $V_\pi(s') = V_*(s')$*

Αν λοιπόν γνωρίζουμε τη λύση στα υποπροβλήματα που προκύπτουν από τις θυγατρικές καταστάσεις s' $V_*(s')$ τότε η λύση βρίσκεται απλά με εξέταση του μέλλοντος για ένα βήμα μπροστά (one step lookahead).

$$V_*(s) \leftarrow \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s')) \tag{3.23}$$

Διαισθητικά ξεκινάμε απο τις τελικές αμοιβές (rewards) και δουλεύουμε προς τα πίσω.

Επανάληψη συνάρτησης V (value iteration) Τα βήματα του αλγορίθμου είναι:

1. Για κάθε επανάληψη κ+1
2. Για όλα τα $s \in S$
3. ενημέρωση του $V_{k+1}(s)$ από το $V_k(s')$ όπου s' θυγατρική κατάσταση της s

όπου η ενημέρωση όμως τώρα γίνεται από τη σχέση:

$$V_{k+1}(s) = \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'} V_k(s)) \tag{3.24}$$

η σε μορφή πινάκων:

$$\mathbf{V_{k+1}} = \max_{\mathbf{a} \in \mathbf{A}}(\mathbf{R^a} + \gamma \mathbf{P^a V_k}) \tag{3.25}$$

Εδώ σε αντίθεση με τον αλγόριθμο policy iteration δεν έχουμε συγκεκριμένη πολιτική και εργαζόμαστε απευθείας με την συνάρτηση V.Αυτό σημαίνει ότι μέχρι να έχουμε σύγκλιση στη V_* ενδιάμεσες V μπορεί να μην αντιστοιχούν σε κάποια πολιτική. Συγκεντρωτικά για τον Παράλληλο δυναμικό προγραμματισμό έχουμε τον παρακάτω πίνακα:

Πρόβλημα	Εξίσωση Bellman	Αλγόριθμος
Prediction	Bellman Expectation Equation	Iterative policy evaluation
Control	Bellman Expectation Equation Policy Iteration + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Για κάθε επανάληψη έχουμε πολυπλοκότητα $O(mn^2)$ ενώ για συναρτήσεις q_π και q_* έχουμε $O(m^2n^2)$ όπου m ο αριθμός των actions και n ο αριθμός των states.

3.3.2 Ασύγχρονος Δυναμικός προγραμματισμός

Εδώ δεν ενημερώνουμε όλα τα states παράλληλα όπως προηγουμένως αλλά ένα ένα και οι αλγόριθμοι συγκλίνουν με την προϋπόθεση ότι όλες οι καταστάσεις εξακολουθούν να επιλέγονται αλλά μπορεί να μειώσει σημαντικά τον χρόνο υπολογισμού.

In-place Δυναμικός προγραμματισμός Αποθηκεύει μόνο ένα αντίγραφο της V για όλα τα $s \in S$. Η ενημέρωση γίνεται ως:

$$V(s) \leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V(s') \right) \tag{3.26}$$

σε αντίθεση με τον σύγχρονο ΔΠ:

$$V_{new}(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_{old}(s') \right)$$

και $V_{ols} \leftarrow V_{new}$

Prioritised Sweeping Μπορεί να μοντελοποιηθεί αποδοτικά με μια ουρά προτεραιότητας. Χρησιμοποιούμε το βαθμό του λάθους Bellman:

$$e = |\max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V(s') \right) - V(s)| \tag{3.27}$$

και ενημερώνουμε το λάθος για επηρεασμένες καταστάσεις μετά από κάθε ενημέρωση.Πάραυτα χρειάζεται να γνωρίζουμε την αντίστροφη δυναμική του συστήματος που μοντελοποιούμε.

Δυναμικός προγραμματισμός πραγματικού χρόνου Η ιδέα εδώ είναι ότι ασχολούμαστε μόνο με καταστάσεις που σχετίζονται άμεσα με το πράκτορα και η επιλογή νέων καταστάσεων γίνεται μέσω της εμπειρίας του. Για κάθε S_t, A_t, R_{t+1} έχουμε ενημέρωση του S_t

$$V(S_t) \leftarrow \max_{a \in A} (R_{s_t}^a + \gamma \sum_{s' \in S} P_{s_t s'}^a V(s')) \tag{3.28}$$

Στο κλείσιμο αυτής της ενότητας θα θέλαμε να αναφέρουμε την γενικότερη αδυναμία του δυναμικού προγραμματισμού στην επίλυση της εξίσωσης Bellman για χώρους με μεγάλο αριθμό καταστάσεων.Όπως φαίνεται και από το σχεδιάγραμμα για κάθε κατάσταση εξετάζουμε κάθε θυγατρική κατάσταση και κάθε δράση που μπορεί να εκτελέσει ο πράκτορας.Έτσι ακόμα και μία οπισθοδιάδοση για ενημέρωση μπορεί να είναι υπολογιστικά ακριβή.

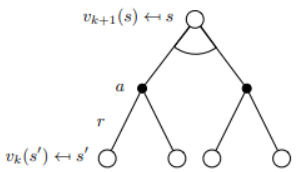

Figure 3.3: Dynamic programming tree

3.4 Η Πρόβλεψη για Άγνωστο Μοντέλο

Στην παράγραφο αυτή θα εξετάσουμε το πρόβλημα στο οποίο για άγνωστο το μοντέλο του συστήματος δηλαδή την MDP θέλουμε να εκτιμήσουμε την συνάρτηση V.

3.4.1 Μέθοδοι Monte Carlo

Οι κατηγορία αυτών των αλγορίθμων στηρίζεται στο ότι είναι εφικτό να μάθουμε την $V_\pi(s)$ μέσα από εμπειρία (experience). Δηλαδή εκτελούμε N προσομοιώσεις ή επεισόδια στο περιβάλλον του πράκτορα (episodes) με την προϋπόθεση πάντα ότι η διαδικασία ή το παιχνίδι που παίζουμε έχει κάποια συνθήκη τερματισμού η οποία χαρακτηρίζεται από επιτυχία η αποτυχία. Από αυτά τα επεισόδια μαζεύουμε κάθε φορά την "επιστροφή" και στο τέλος βρίσκουμε την μέση τιμή τους.

Monte Carlo πρώτης επίσκεψης Την πρώτη χρονική στιγμή t που επισκεπτόμαστε μια κατάσταση s τότε αυξάνουμε τους δύο στατιστικούς δείκτες¨:
$N(s) \leftarrow N(s) + 1$ ώς τον συνολικό αριθμό των επισκέψεων και
$S(s) \leftarrow S(s) + G_t$ ως το συνολικό άθροισμα των αμοιβών.
Τέλος απλά λαμβάνουμε τον μέσο όρο όπως αναφέραμε.
$V(s) = \frac{S(s)}{N(s)}$

Monte Carlo κάθε επίσκεψης Η ιδέα είναι ακριβώς όμοια με προηγουμένως μόνο που η ενημέρωση των στατιστικών δεικτών γίνεται κάθε φορά που βρισκόμαστε σε μία κατάσταση s σε ένα επεισόδιο και όχι μόνο την πρώτη φορά που θα βρεθούμε σε αυτή.

Αυξητικός Monte Carlo Ενημερώνουμε την V(s) αυξητικά μετά από κάθε επεισόδιο. Στηριζόμαστε απλά σε έναν άλλο τρόπο γραφής του μέσου όρου ως εξής:

$$\mu_\kappa = \frac{1}{\kappa}\sum_{j=0}^{\kappa} x_j = \mu_{\kappa-1} + \frac{1}{\kappa}(x_k - \mu_{k-1}) \tag{3.29}$$

Για κάθε $V(s)$ με επιστροφή G_t ενημερώνουμε:
$N(S_t) \leftarrow N(S_t) + 1$ και
$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$

Ο νόμος μεγάλων αριθμών μας εξασφαλίζει την ζητούμενη σύγκλιση , δηλαδή για:
$N(s) \to \infty$ τότε $V(s) \to V_\pi(s)$

3.4.2 Μάθηση Προσωρινών διαφορών (Temporal Difference learning)

Αποτελεί μια άλλη κατηγορία αλγορίθμων στους οποίους δεν είναι απαραίτητη η γνώση της μαρκοβιανής διαδικασίας που περιγράφει το μοντέλο.Η διαφορά με τις μεθόδους Monte Carlo είναι ότι η μάθηση γίνεται καθ όλη την διάρκεια του επεισοδίου χωρίς να περιμένουμε να τερματίσει και δεν ενημερώνουμε προς το αληθινό G_t. Έτσι χρησιμοποιείται για εκμάθηση πρακτόρων σε συνεχή περιβάλλοντα και όχι σε επεισοδιακά.

Ο αλγόριθμος TD(0) Σε αντίθεση με τον incremental Monte Carlo που κάνουμε ενημέρωση προς καλύτερες προσεγγίσεις της αληθινής επιστροφής μέσα από επεισόδια εδώ ενημερώνουμε το $V(S_t)$ προς τον όρο $R_{t+1} + \gamma V(S_{t+1})$ δηλαδή:

$$V(S_t) \leftarrow V(S_t) + a \overbrace{(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))}^{TDerror} \tag{3.30}$$

Ο όρος $R_{t+1} + \gamma V(S_{t+1})$ ονομάζεται TD-target και ο ορός $\delta_t = (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$ ονομάζεται TD-error. Η παράμετρος α είναι αντίστοιχη με την παράμετρο γ που εξετάσαμε στο κεφάλαιο των νευρωνικών δικτύων και αποτελεί το βήμα μάθησης (learning rate) του πράκτορα. Από τον στόχο προς τον οποίο γίνεται η ενημέρωση καταλαβαίνουμε διαισθητικά οτι ενημερώνουμε μια υπόθεση προς μια άλλη υπόθεση.
Σε αντίθεση με το πραγματικό TD-target $R_{t+1} + \gamma V_\pi(S_{t+1})$ το οποίο αποτελεί αμερόληπτη εκτίμηση του $V_\pi(S_t)$, το TD-target που έχουμε αποτελεί μια biased εκτίμηση του $V_\pi(S_t)$ η οποία όμως έχει πολύ μικρότερη διασπορά από την επιστροφή G_t των μεθόδων Monte carlo η οποία εξαρτάται από πολλές τυχαίες δράσεις,μεταβάσεις και αμοιβές. Εδώ σε κάθε ενημέρωση έχουμε εξάρτηση μόνο απο μια τυχαία δράση-μετάβαση-αμοιβή.

Ο αλγόριθμος TD(0) συγκλίνει στη λύση της μέγιστης πιθανοφάνειας μαρκοβιανού μοντέλου που προσεγγίζει καλύτερα τα δεδομένα μας. MDP με $< S, A, \hat{P}, \hat{R}, \gamma >$

$$\hat{P^a_{ss'}} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=0}^{T_k} \mathbf{1}(\mathbf{s_t^k}, \mathbf{a_t^k}, \mathbf{s_{t+1}^k} = \mathbf{s}, \mathbf{a}, \mathbf{s'})$$

και

$$\hat{R^a_s} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=0}^{T_k} \mathbf{1}(\mathbf{s_t^k}, \mathbf{a_t^k} = \mathbf{s}, \mathbf{a}) \mathbf{r_t^k}$$

Ο αλγόριθμος TD(λ) Τον αλγόριθμο TD(0) μπορούμε να τον επεκτείνουμε έτσι ώστε να εξετάζει n-βήματα στο μέλλον.Αναλυτικά ορίζουμε την "επιστροφή" για n-βήματα ως εξής:

$$\begin{aligned} G_t^1 &= R_{t+1} + \gamma V(S_{t+1}) \quad n = 1 \\ G_t^2 &= R_{t+1} + R_{t+2} + \gamma V(S_{t+2}) \quad n = 2 \\ &\vdots \\ G_t^\infty &= R_{t+1} + ... + \gamma^{\tau-1} R_\tau \quad n = \infty \end{aligned}$$

οπότε ο αλγόριθμος λαμβάνει αντίστοιχα την μορφή:

$$V(S_t) \leftarrow V(S_t) + a(G_t^n - V(S_t) \tag{3.31}$$

Για n=1 έχουμε τον απλό αλγόριθμο TD(0) ενώ για $n = \infty$ έχουμε τον αλγόριθμο monte carlo από την προηγούμενη ενότητα. Μπορούμε να συνδυάσουμε όλες τις επιστροφές για όλα τα n με βάρη. $(1-\lambda)\lambda^{n-1}$ και ορίζουμε την λ-επιστροφή ως:

$$G_t^\lambda = (1-\lambda)\sum_{n=0}^{\infty} \lambda^{n-1} G_t^n \tag{3.32}$$

οπότε ο TD γίνεται:

$$V(S_t) \leftarrow V(S_t) + a(G_t^\lambda - V(S_t)) \tag{3.33}$$

Η παραπάνω εξίσωση ενημέρωσης αποτελεί τον εμπρόσθιο αλγόριθμο TD(λ).

Ο οπίσθιος αλγόριθμος TD(λ) Η βασική ιδέα είναι ότι για κάθε state διατηρούμε ένα ίχνος και έτσι ενημερώνουμε το $V(s)$ με έναν όρο ανάλογο του λάθους δ_t αλλά και του ίχνους $E(s)$. Το ίχνος συνδυάζει δύο ευριστικές, μία συχνότητας η οποία δίνει βάρος σε καταστάσεις που επισκέπτονται πιο συχνά και μια που δίνει βάρος στις πιο πρόσφατες καταστάσεις. Έχουμε:

$$\begin{aligned} E_0(s) &= 0 \\ E_t(s) &= \gamma\lambda E_{t-1}(s) + \mathbf{1}(\mathbf{S_t} = \mathbf{s}) \end{aligned}$$

Η λύση της παραπάνω αναδρομικής σχέσης μας δίνει:

$$E_t(s) = \begin{cases} 0 & t < k \\ (\gamma\lambda)^{t-k} & t \geq k \end{cases} \tag{3.34}$$

και η ενημέρωση γίνεται:

$$V(s) \leftarrow V(s) + a\delta_t E_t(s) \tag{3.35}$$

με

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Για λ=0 ενημερώνουμε μόνο την τρέχουσα κατάσταση και προκύπτει ο forward TD(0). Ενδιαφέρον παρουσιάζει το γεγονός ότι το άθροισμα όλων των ενημερώσεων είναι ίδιο για τον εμπρόσθιο και τον οπίσθιο αλγόριθμο TD(λ) δηλαδή:

$$\sum_{t=1}^{T} a\delta_t E_t(s) = \sum_{t=1}^{T} a(G_t^\lambda - V(S_t))\mathbf{1}(\mathbf{S_t} = \mathbf{s}) \tag{3.36}$$

3.5 Η Βελτιστοποίηση για Άγνωστο Μοντέλο

Στην προηγούμενη ενότητα εξετάσαμε πως μπορούμε να βρούμε την συνάρτηση V δεδομένης μιας πολιτικής π ενώ μας είναι άγνωστη η μαρκοβιανή διαδικασία που περιγράφει το μοντέλο του περιβάλλοντος.Σε αυτή τη παράγραφο θα εξετάσουμε τρόπους βελτιστοποίησης της συνάρτησης V.Θα ασχοληθούμε με δύο μεθόδους:
On policy μέθοδος: Μαθαίνουμε μια πολιτική π μέσω δειγματοληψίας από εμπειρία από την ίδια την π.
off policy μέθοδος: Μαθαίνουμε μέσω δειγματοληψίας από πολιτική μ.

3.5.1 On Policy έλεγχος

Προκειμένου να αγνοήσουμε τις δυναμικές του συστήματος δηλαδή το μαρκοβιανό μοντέλο το οποίο δεν μας είναι γνωστό χρησιμοποιούμε για την βελτίωση της π ένα άπληστο κριτήριο πάνω στην συνάρτηση q και όχι στη συνάρτηση V. Έτσι έχουμε:

$$\pi' = \underset{a \in A}{\operatorname{argmax}}\, q(s,a) \tag{3.37}$$

Εισάγουμε επίσης την έννοια της **άπληστης εξερεύνησης** ως:

$$\pi(\alpha|s) = \begin{cases} \frac{\varepsilon}{m} + 1 - \varepsilon & \alpha^* = \underset{a \in A}{\operatorname{argmax}}\, q(s,a) \\ \frac{\varepsilon}{m} & \alpha\lambda\lambda\iota\varsigma \end{cases} \tag{3.38}$$

Η παραπάνω σχέση δηλώνει απλά ότι αν έχουμε m διαθέσιμες δράσεις τότε η καλύτερη δράση σύμφωνα με τη συνάρτηση q που έχουμε σε εκείνο το βήμα επιλέγεται με πιθανότητα (1-ε) ενώ μια τυχαία δράση επιλέγεται με πιθανότητα ε. Με αυτό το τρόπο εξασφαλίζουμε ότι όλες οι δυνατές δράσεις θα πραγματοποιηθούν. Παραθέτουμε το παρακάτω θεώρημα σύμφωνα με το οποίο εξασφαλίζουμε τη βελτίωση:

Θεώρημα 2. *Για κάθε ϵ-άπληστη πολιτική π τότε η ϵ-άπληστη πολιτική π' ως προς την συνάρτηση q_π αποτελεί μια βελτίωση με $V_{\pi'}(s) \geq V_\pi(s)$.*

Proof.

$$V_{\pi'}(s) = q_\pi(s, \pi'(s)) = \tag{3.39}$$

$$\sum_{a \in A} \pi'(a|s) q_\pi(s,a) = \tag{3.40}$$

$$\frac{\varepsilon}{m} \sum_{a \in A} q_\pi(s,a) + (1-\varepsilon) \max_{a \in A} q_\pi(s,a) \geq \tag{3.41}$$

$$\frac{\varepsilon}{m} \sum_{a \in A} q_\pi(s,a) + (1-\varepsilon) \sum_{a \in A} \frac{\pi(a|s) - e/m}{1-\varepsilon} q_\pi(s|a) = \tag{3.42}$$

$$\sum_{a \in A} \pi(a|s) q_\pi(s,a) = V_\pi(s) \tag{3.43}$$

□

3.5.1.1 Βελτιστοποίηση με Monte Carlo

Στην βελτιστοποίηση με Monte carlo εκτελούμε διαδοχικά τα βήματα της αξιολόγησης πολιτικής με $Q = q_\pi$ και ε-άπληστη βελτίωση για κάθε επεισόδιο. GLIE (Greedy in the limit of infinite exploration)

- Όλα τα ζευγάρια δράσης-κατάστασης εξερευνούνται άπειρες φορές.

$$\lim_{k\to\infty} N(s,a) = \infty$$

- Η πολιτική συγκλίνει σε μία άπληστη πολιτική.

$$\lim_{k\to\infty} \pi_\kappa(a|s) = \mathbf{1}(\mathbf{a} = \underset{\mathbf{a'}\in\mathbf{A}}{\operatorname{argmax}}\, \mathbf{Q_k}(\mathbf{s},\mathbf{a'}))$$

Για παράδειγμα μια ε-άπληστη στρατηγική είναι GLIE αν πηγαίνει στο μηδέν για $\varepsilon_k = \frac{1}{k}$

GLIE βελτιστοποίηση Monte carlo Ο αλγόριθμος λειτουργεί ως εξής:

1. δειγματοληπτούμε το κ-οστο επεισόδιο χρησιμοποιώντας πολιτική π και παίρνουμε $[S_1, A_1, R_2, .., S_\tau]$
2. Για κάθε κατάσταση S_t και δράση A_t κάνουμε τις εξής ενημερώσεις:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$$

3. Βελτιώνουμε την πολιτική βασιζόμενοι στην νέα συνάρτηση q με

$$\varepsilon \leftarrow \frac{1}{k}$$
$$\pi' \leftarrow \varepsilon - greedy(Q)$$

3.5.1.2 Ο Αλγόριθμος SARSA

Η λογική του αλγορίθμου SARSA είναι παρόμοια με τον TD που εξετάσαμε στη προηγούμενη ενότητα για το πρόβλημα της αξιολόγησης μιας πολιτικής. Η ονομασία του προκύπτει από την αλληλουχία των διαδοχικών καταστάσεων και των μεταξύ τους δράσεων και σημάτων αμοιβών. Η σχέση ενημέρωσης για την συνάρτηση q δίνεται ως:

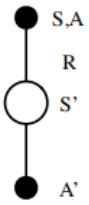

Figure 3.4: Sarsa

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A)) \tag{3.44}$$

και αναφέρεται στην συνάρτηση q σε αντίθεση με τον TD που αναφερόταν στην συνάρτηση V.

Algorithm 1: Ο αλγόριθμος SARSA

Initialize Q(s,a) $\forall s \in S, a \in A$ and Q(terminal,a)=0;
for *each episode* **do**
  Initialize S;
  Pick action A from S following policy derived from Q (ε-greedy);
  while *step of episode is not terminal* **do**
    Take action A, observe R and arrive at S';
    Choose action A' from S' using policy derived from Q;
    $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$;
    $S \leftarrow S'$;
    $A \leftarrow A'$
  end
end

Αποδεικνύεται ότι ο παραπάνω αλγόριθμος συγκλίνει στην βέλτιστη συνάρτηση q δηλαδή $Q(S,A) \rightarrow q_*(s,a)$.

Ο εμπρόσθιος SARSA(λ) Μπορούμε πάλι να επεκτείνουμε τον SARSA όπως τον TD(0) ώστε να εξετάζει περισσότερα βήματα μπροστά ως εξής:

$$\begin{aligned} q_t^1 &= R_{t+1} + \gamma Q(S_{t+1}) \quad n = 1 \\ q_t^2 &= R_{t+1} + R_{t+2} + \gamma Q(S_{t+2}) \quad n = 2 \\ &\vdots \\ q_t^\infty &= R_{t+1} + ... + \gamma^{\tau-1} R_\tau \quad n = \infty \end{aligned}$$

Αν ορίσουμε την q-επιστροφή για n βήματα ώς:

$$q_t^n = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}) \tag{3.45}$$

τότε ο SARSA n-βημάτων ενημερώνει την Q(S,A) προς το q_t^n ως

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(q_t^n - Q(S_t, A_t) \tag{3.46}$$

Αν συνδυάσουμε πάλι την επιστροφή για όλα τα βήματα n με βάρη (1-λ) έχουμε την επιστροφή:

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)} \tag{3.47}$$

Έτσι λοιπόν παίρνουμε την εξίσωση ενημέρωσης για τον **forward view SARSA(λ)**

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(q_t^\lambda - Q(S_t, A_t) \tag{3.48}$$

Αντίστοιχα για τον backward view SARSA έχουμε όμοια λογική με τον backward view TD(0).Ορίζουμε τα ίχνη:

$$E_0(s,a) = 0$$
$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(\mathbf{S_t} = \mathbf{s}, \mathbf{A_t} = \mathbf{a})$$

και η συνάρτηση Q(s,a) ενημερώνεται για κάθε ζεύγος κατάστασης-δράσης.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E(s,a)$$
$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

Είμαστε σε θέση τώρα να παρουσιάσουμε τον αλγόριθμο **SARSA(λ)**

Algorithm 2: Ο αλγόριθμος SARSA(λ)

Initialize Q(s,a) $\forall s \in S, a \in A$ and Q(terminal,a)=0;
for *each episode* **do**
  E(s,a)=0 $\forall s \in S, a \in A$ Initialize S,A;
  while *step of episode is not terminal* **do**
    Take action A, observe R and arrive at S';
    Choose action A' from S' using policy derived from Q (ε-greedy);
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$;
    $E(S, A) \leftarrow E(S, A) + 1$;
    for *all* $s \in S, a \in A$ **do**
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$;
      $E(s,a) \leftarrow \gamma\lambda E(s,a)$;
    end
    $S \leftarrow S'$;
    $A \leftarrow A'$;
  end
end

3.5.2 Off Policy έλεγχος

Η ιδέα είναι ότι αξιολογούμε πολιτική π(α—s) ώστε να υπολογίσουμε το $V_\pi(s)$ ή $q_\pi(s,a)$ ενώ ακολουθούμε πολιτική συμπεριφοράς $\mu(\alpha|s)$ $[S_1, A_1, ..., S_\tau]$ μ. Με άλλα λόγια η μάθηση μπορεί να γίνεται είτε παρατηρώντας κάποια σχεδιασμένη ανθρώπινη πολιτική είτε ακόμα από πολιτικές άλλων πρακτόρων.Μπορούμε επίσης να επαναχρησιμοποιούμε πολιτικές που έχουν προκύψει από παλιότερα βήματα $\pi_1, \pi_2, ...\pi_{t-1}$ και να αξιοποιούμε προηγούμενη εμπειρία του πράκτορα.
Για πολιτική π και πολιτική συμπεριφοράς μ ορίζουμε την επιστροφή:

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)\pi(A_{t+1}|S_{t+1})...\pi(A_\tau|S_\tau)}{\mu(A_t|S_t)\mu(A_{t+1}|S_{t+1})...\mu(A_\tau|S_\tau)} G_t \tag{3.49}$$

όπου G_t είναι η επιστροφή από την πολιτική μ την οποία ακολουθήσαμε πολλαπλασιασμένη με ένα συντελεστή διόρθωσης ο οποίος μας δείχνει την ομοιότητα της πολιτικής μ με την π σε κάθε βήμα μέχρι το τέλος του επεισοδίου με την προϋπόθεση πάντα ότι μ δεν είναι μηδέν όταν δεν έχουμε και π=0
Στον **off policy montecarlo** ενημερώνουμε προς τη $G_t^{\pi/\mu}$:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\pi/\mu} - V(S_t)) \tag{3.50}$$

Τον **temporal difference TD** τον προσαρμόζουμε απλά σταθμίζοντας το στόχο μας $R_{t+1} + \gamma V(S_{t+1})$ με τη διόρθωση μόνο για ένα βήμα:

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\overbrace{\frac{\pi(A_t|S_t}{\mu(A_t|S_t)}(R_{t+1} + \gamma V(S_{t+1})}^{off-policyTD-error} - V(S_t) \right) \tag{3.51}$$

Όπως και στο πρόβλημα της αξιολόγησης μιας πολιτικής η ενημέρωση με αυτό το τρόπο έχει πολύ λιγότερη διασπορά σε σύγκριση με την μέθοδο Monte Carlo.

3.5.2.1 Ο αλγόριθμος Q-learning

Με αυτόν τον αλγόριθμο ο στόχος μας είναι να μάθουμε τις τιμές Q(s,a) off policy. Χωρίς να χρησιμοποιούμε συντελεστές διόρθωσης όπως προηγουμένως διαλέγουμε από την κατάσταση S την δράση A ακολουθώντας την πολιτική συμπεριφοράς μ δηλαδή $A_{t+1} \sim \mu(\cdot|S_t)$ αλλά λαμβάνουμε υπόψιν μας και δράση A' ακολουθώντας την πολιτική π $A' \sim \pi(\cdot|S_t)$ και στη συνέχεια ενημερώνουμε το $Q(S_t, A_t)$ χρησιμοποιώντας την εναλλακτική δράση.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\overbrace{R_{t+1} + \gamma Q(S_{t+1}, A')}^{target} - Q(S_t, A_t)) \tag{3.52}$$

Με αυτό το τρόπο επιτρέπουμε και στην πολιτική συμπεριφοράς μ αλλά και στην πολιτική στόχο π να βελτιωθούν. Η πολιτική π είναι άπληστη ως προς Q(s,a) δηλαδή $\pi(S_t) = \underset{a' \in A}{\operatorname{argmax}} Q(S_{t+1}, a')$ και η πολιτική μ είναι ε-άπληστη ως προς την Q(s,a) και αυτή. Τον

στόχο μπορούμε να τον γράψουμε ως:

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q(S_{t+1}, \underset{a' \in A}{\operatorname{argmax}}\, Q(S_{t+1}, a') \\ &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \\ Q(S, A) &\leftarrow Q(S, A) + a(target - Q(S, A)) \end{aligned}$$

Παρακάτω περιγράφουμε συνολικά τον αλγόριθμο βήμα προς βήμα.

Algorithm 3: Ο αλγόριθμος Q-learning

```
Initialize Q(s,a) ∀s ∈ S, a ∈ A and Q(terminal,a)=0;
for each episode do
    Initialize S;
    while step of episode is not terminal do
        Choose action A from S using policy derived from Q (ε-greedy);
        Take action A, observe R,S';
        Q(S,A) ← Q(S,A) + α(R + γ max_a Q(S',a) − Q(S,A));
        S ← S';
    end
end
```

Κεφάλαιο 4

Βαθιά Ενισχυτική Μάθηση

4.1 Προσέγγιση συναρτήσεων αξίας

Στην διάρκεια της προηγούμενης ενότητας όλοι οι αλγόριθμοι που παρουσιάσαμε μεταχειρίζονταν τη συνάρτηση αξίας V αλλά και τη συνάρτηση δράσης-αξίας Q(s,a) ως έναν πίνακα τον οποίο χρησιμοποιούσαμε για να αποθηκεύουμε τις τιμές αξίας που αντιστοιχούσαν σε κάθε δράση και σε κάθε κατάσταση και κάναμε την αντίστοιχη ενημέρωση όταν αυτό απαιτούνταν.Όπως γίνεται αντιληπτό μια τέτοια πρακτική εκτός από μη αποδοτική είναι και μη υγροποιήσιμη για το φάσμα τον περισσότερων εφαρμογών στις οποίες είτε ο αριθμός των δράσεων και καταστάσεων είναι απαγορευτικά μεγάλος ώστε ένας τέτοιος πίνακας να χωράει στη μνήμη είτε μιλάμε για περιβάλλοντα στα οποία ο χώρος των καταστάσεων είναι συνεχής.Για πολύ μεγάλες λοιπόν μαρκοβιανές διαδικασίες, είτε τις γνωρίζουμε ακριβώς (model based) είτε όχι (model free) χρησιμοποιούμε έναν εκτιμητή (aproximator) για τις συναρτήσεις v και q ο οποίος έχει την δυνατότητα να γενικεύει τις προβλέψεις του ακόμα και για καταστάσεις που δεν έχει ξανασυναντήσει με βάση καταστάσεις που έχει συναντήσει στο παρελθόν.Ο εκτιμητής είναι συνάρτηση ενός διανύσματος βαρών **w** δηλαδή:

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

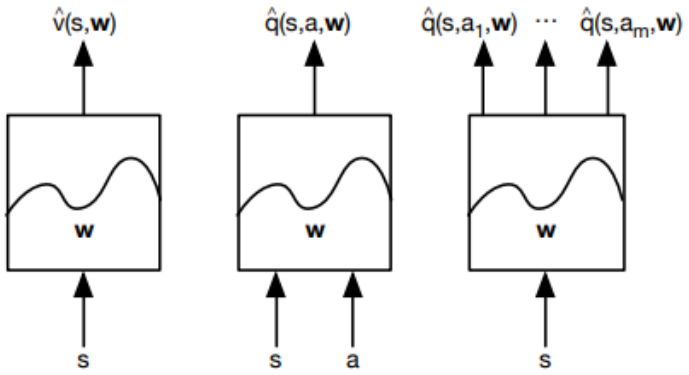

Figure 4.1: Τύποι συναρτήσεων προσέγγισης

4.1.1 Αυξητικοί αλγόριθμοι πρόβλεψης

Όπως παρουσιάζεται και στο [4], στόχος μας είναι να προσεγγίσουμε όσο καλύτερα γίνεται την συνάρτηση v η q οπότε καλούμαστε να ορίσουμε ένα κριτήριο σφάλματος ανάμεσα στον εκτιμητή και στην πραγματική συνάρτηση το οποίο καλούμαστε να ελαχιστοποιήσουμε.Ορίζουμε λοιπόν συνάρτηση κόστους:

$$J(\boldsymbol{w}) = E_\pi[(\hat{v}(s, \boldsymbol{w}) - v_\pi(s))^2] \tag{4.1}$$

Έτσι το πρόβλημα μετατρέπεται σε βέλτιστο έλεγχο δηλαδή εύρεση του διανύσματος $\boldsymbol{w}$ έτσι ώστε να ελαχιστοποιείται το μέσο τετραγωνικό σφάλμα μεταξύ του εκτιμητή και της v_π
Για να προσεγγίσουμε το ελάχιστο χρησιμοποιούμε μέθοδο κατάβασης κλίσης όπως περιγράψαμε στην ενότητα των νευρωνικών δικτύων.

$$\begin{aligned}\Delta \boldsymbol{w} &= -\frac{1}{2}\alpha \nabla_w J(\boldsymbol{w}) \\ &= \alpha E_\pi[(V_\pi(s) - \hat{V}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w})]\end{aligned}$$

Για ευκολία στην ενημέρωση των βαρών δειγματολειπτούμε την κλίση και παίρνουμε απλά (stohastic gradient decent):

$$\Delta \boldsymbol{w} = \alpha(V_\pi(s) - \hat{V}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w})$$

Γενικά αναπαριστούμε την κατάσταση s ως ένα διάνυσμα με n χαρακτηριστικά (features)

$$x(s) = \begin{bmatrix} x_1(s) \\ x_2(s) \\ \vdots \\ x_n(s) \end{bmatrix} \tag{4.2}$$

Γραμμικός συνδυασμός χαρακτηριστικών Ο πιο απλός τρόπος να κατασκευάσουμε έναν εκτιμητή κατάστασης είναι απλά να σταθμίσουμε με βάρη το διάνυσμα καταστάσεων x(s). Η δυσκολία αυτής της μεθόδου συνήθως έγκειται στο γεγονός οτι τα χαρακτηριστικά του διανύσματος κατάστασης είναι αρκετά δύσκολο να εξαχθούν και συνήθως απαιτούν προγραμματισμό από ειδικούς στη διεργασία που θέλουμε να εκτελέσει ο πράκτορας μας.

$$\hat{v}(s, \boldsymbol{w}) = x^T(s)\boldsymbol{w} = \sum_{j=1}^{n} x_j(s) w_j \tag{4.3}$$

και

$$J(\boldsymbol{w}) = E_\pi[(v_\pi(s) - x^T(s)\boldsymbol{w})^2] \tag{4.4}$$

Έτσι ο κανόνας της ενημέρωσης των βαρών γίνεται με βάση την κλίση:

$$\nabla_w \hat{v}(s, \boldsymbol{w}) = x(s) \tag{4.5}$$

οπότε:

$$\Delta \boldsymbol{w} = \alpha(V_\pi(s) - \hat{V}(s, \boldsymbol{w}))x(s) \tag{4.6}$$

Όσον αφορά την εκπαίδευση στα προβλήματα που αντιμετωπίζουμε δεν έχουμε διαθέσιμη την πραγματική τιμή του $V_\pi(s)$ αλλά μόνο ένα σήμα αμοιβής. Έτσι ανάλογα με τον αλγόριθμο ενισχυτικής μάθησης που χρησιμοποιούμε έχουμε και διαφορετικό στόχο. Στον Monte carlo χρησιμοποιούμε το G_t ενώ στον TD(0) και TD(λ) έχουμε για targets τα $R_{t+1} + \gamma\hat{v}(S_{t+1})$ και G_t^λ αντίστοιχα.

Έτσι έχοντας ζεύγη της μορφής $< S_1, G_1 >, ... < S_m, G_m >$ μπορούμε να εφαρμόσουμε γραμμική αξιολόγηση πολιτικής με Monte Carlo ως επιβλεπόμενη μάθηση με τα G_i να αποτελούν ένα αμερόληπτο θορυβώδες δείγμα από την V_π:

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha(G_t - \hat{V}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w}) \\ &= \alpha(G_t - \hat{V}(s, \boldsymbol{w}))x(S_t)\end{aligned}$$

Δεν είναι απαραίτητο να χρησιμοποιούμε γραμμικό συνδυασμό των βαρών και μια μη γραμμική σχέση να μπορεί να προσεγγίσει καλύτερα την εργασία του πράκτορα. Χρησιμοποιήσαμε ενδεικτικά γραμμικό συνδυασμό για λόγους απλότητας και ευκολίας στο χειρισμό της παραγώγου. Με την παραπάνω μέθοδο μας εξασφαλίζεται η σύγκλιση σε τοπικό ακρότατο και όχι σε ολικό ακόμα και αν χρησιμοποιήσουμε κάποια μη γραμμική απεικόνιση ανάμεσα στα χαρακτηριστικά του state και της τιμής της κάθε κατάστασης.
Αν έχουμε ζεύγη της μορφής: $< S_1, R_2 + \gamma V(S_2, w) >, ... < S_m, R_{m+1} + \gamma V(S_{m+1}, w) >$ μπορούμε να εφαρμόσουμε γραμμικό TD(0) (ομοίως και TD(λ)) ως:

$$\Delta \boldsymbol{w} = \alpha(R + \gamma\hat{V}(S', w) - \hat{V}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w})$$

Σε αντίθεση με την μέθοδο monte carlo ο TD(0) συγκλίνει προς το ολικό ελάχιστο της συνάρτησης κόστους στο χώρο των βαρών.

4.1.2 Αυξητικοί αλγόριθμοι ελέγχου.

Είτε ασχολούμαστε με προσεγγίσεις των συναρτήσεων αξίας είτε όχι η λογική παραμένει ή ίδια όταν ασχολούμαστε με το πρόβλημα του ελέγχου σύμφωνα με το οποίο θυμίζουμε ο στόχος μας είναι η εύρεση βέλτιστης πολιτικής.Εκτελούμε δηλαδή επαναληπτικά τα βήματα αξιολόγησης πολιτικής (policy evaluation) και βελτίωση πολιτικής (policy improvement). Προσεγγίζουμε πρώτα την

$$\hat{q}(s, a, w) \approx q_\pi(s, a)$$

και ύστερα ελαχιστοποιούμε την:

$$J(w) = E_\pi[(q_\pi(s, a) - \hat{q}(s, a, w))^2]$$

Η ενημέρωση των βαρών γίνεται πάλι ακολουθώντας την τοπική κλίση της J και έχουμε:

$$\Delta \boldsymbol{w} = \alpha(q_\pi(s, a) - \hat{q}(s, a, \boldsymbol{w}))\nabla_w \hat{q}(s, a, \boldsymbol{w}) \tag{4.7}$$

Επειδή πάλι δεν έχουμε διαθέσιμες τις πραγματικές τιμές για την συνάρτηση q, για τον monte carlo χρησιμοποιούμε το G_t ως στόχο και για τον TD(0) το $R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w)$

4.1.3 Μέθοδος ελαχίστων τετραγώνων

Οι παραπάνω μέθοδοι αν και εύκολες στην κατανόηση και υλοποίηση δεν είναι αποδοτικές λόγο της ποιότητας της δειγματοληψίας που πραγματοποιείται σε περιβάλλοντα ενισχυτικής μάθησης.Πριν προχωρήσουμε στα DQN δίκτυα τα οποία θεωρούνται από τις βασικότερες μεθόδους βαθιάς ενισχυτικής μάθησης παρουσιάζουμε μια άλλη μέθοδο συνόλου (batch method) που καλείται να προσαρμόσει μια συνάρτηση αξίας σε ένα **σύνολο** από δείγματα αξιοποιώντας δηλαδή την εμπειρία που έχει αποκτήσει ο πράκτορας [4].
Έχουμε να επιλύσουμε λοιπόν το εξής πρόβλημα: Δεδομένης μιας συνάρτησης προσέγγισης $\hat{v}(s,w) \approx v_\pi(s)$ και ενός συνόλου δειγμάτων $D = \{(S_1, V_1^\pi), ...(S_\tau, V_\tau^\pi)\}$ ποιοί παράμετροι w ελαχιστοποιούν την ποσότητα:

$$LS(w) = \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, w))^2 = E_D[(v_t^\pi - \hat{v}(s_t, w))] \tag{4.8}$$

Έτσι λοιπόν κάθε φορά δειγματολειπτούμε από το D ζεύγος $< s, v^\pi >$ και εφαρμόζουμε στοχαστική κατάβαση κλίσης για να ενημερώσουμε τα βάρη κατά

$$\Delta \boldsymbol{w} = \alpha(v^\pi - \hat{V}(s, \boldsymbol{w}))\nabla_w \hat{v}(s, \boldsymbol{w}) \tag{4.9}$$

και στο τέλος έχουμε σύγκλιση στο $\boldsymbol{w}^\pi = arg\min_w LS(\boldsymbol{w})$ Στο ελάχιστο της LS(w) περιμένουμε η αναμενόμενη ενημέρωση να είναι μηδενική οπότε:

$$E_D[\Delta \boldsymbol{w}] = 0$$

$$a \sum_{t=1}^{T} x(s_t)(v_t^\pi - x(s_t)^T \boldsymbol{w}) = 0$$

$$\sum_{t=1}^{T} v_t^\pi x(s_t) = \sum_{t=1}^{T} x(s_t) x(s_t)^T \boldsymbol{w}$$

$$\boldsymbol{w} = \left(\sum_{t=1}^{T} x(s_t) x(s_t)^T \right)^{-1} \sum_{t=1}^{T} v_t^\pi x(s_t)$$

Επειδή δεν γνωρίζουμε τις αληθινές τιμές για το v_t^π χρησιμοποιούμε θορυβώδη ή δείγματα με μεροληψίακαι έχουμε τις τρεις περιπτώσεις:

- **LS Monte carlo** με $v_t^\pi \approx G_t$
- **LSTD** με $v_t^\pi \approx R_{t+1} + \gamma \hat{V}(S_{t+1}, \boldsymbol{w})$
- **LSTD(λ)** με $v_t^\pi \approx G_t^\lambda$

Για το πρόβλημα του ελέγχου εφαρμόζουμε τον αλγόριθμο q learning με την ενημέρωση των βαρών ως:

$$\Delta \boldsymbol{w} = \alpha \delta x(s_t, a_t)$$
$$\delta = R_{t+1} + \gamma \hat{q}(s_{t+1}, \pi(s_{t+1}), \boldsymbol{w}) - \hat{q}(s_t, a_t, \boldsymbol{w})$$

Για να λάβουμε κλειστή μορφή λύνουμε την εξίσωση με την αναμενόμενη τιμή της ενημέρωσης και με όμοιο τρόπο όπως στο πρόβλημα της πρόβλεψης παραπάνω λαμβάνουμε (least squares TD Q learning):

$$\boldsymbol{w} = \left(\sum_{t=1}^{T} (\boldsymbol{x}(s_t, a_t)(\boldsymbol{x}(s_t, a_t) - \gamma \boldsymbol{x}(s_{t+1}, \pi(a_{t+1})))^T \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(s_t) R_{t+1} \tag{4.10}$$

4.2 Κλίση πολιτικής (Policy Gradients)

4.2.1 Το θεώρημα κλίσης πολιτικής

Ένας άλλος τρόπος να προσεγγίσουμε το πρόβλημα εύρεσης βέλτιστης πολιτικής είναι αντί να παραμετροποιήσουμε τη συνάρτηση δράσης-αξίας Q και να διαλέγουμε ύστερα άπληστα την καλύτερη δράση είναι η απευθείας παραμετροποίηση της πολιτικής π και μοντελοποίηση της μέσω ενός νευρωνικού δικτύου.Δηλαδή:

$$\pi_\theta(s|a) = P[a|s,\theta]$$

Ο στόχος της ενισχυτικής μάθησης είναι να μεγιστοποιήσουμε την συνολική αμοιβή ενώ ακολουθούμε πολιτική π. Για να εμπλέξουμε την μηχανική μάθηση ορίζουμε ένα σετ παραμέτρων ϑ (λόγου χάρη τα βάρη ενός νευρωνικού δικτύου) οι οποίοι παραμετροποιούν την πολιτική π. Για λόγους καθαρότητας οπότε χρησιμοποιείται η πολιτική π ως δείκτης θα παραλείπουμε την παράμετρο ϑ η οποία απλά εννοείται. Υπάρχουν τρία είδη κριτηρίων τα οποία μπορεί να κληθούμε να ελαχιστοποιήσουμε:

$$J_1(\theta) = E_\pi[V_1] = V^\pi(S_0)$$

Αν έχουμε επεισοδιακό περιβάλλον.

$$J_2(\theta) = \sum_s d^\pi(s)V^\pi(s)$$

Αν έχουμε μη επεισοδιακό περιβάλλον και

$$J_3(\theta) = \sum_s d^\pi(s)\sum_a \pi_\theta(s,a)R_s^a$$

αν θέλουμε ένα μέσο όρο για κάθε χρονικό βήμα.
Ως $d^\pi(s)$ έχουμε την στάσιμη κατανομή για μια μαρκοβιανή διαδικασία ως προς την πολιτική π. $d^\pi(s) = \lim_{t\to\infty} P[s_t = s|s_0,\pi_\theta]$ Ως συνηθισμένος τρόπος επίλυσης του προβλήματος είναι η ανάβαση κλίσης η οποία χρησιμοποιείται στους περισσότερους αλγόριθμους μηχανικής μάθησης και ενημερώνουμε το διάνυσμα παραμέτρων ϑ κατά τα γνωστά ως:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\nabla_\theta J(\boldsymbol{\theta}_t])$$

Ωστόσο η παράγωγος του J είναι δύσκολο να υπολογιστεί λόγω της ύπαρξης της παραγώγου της κατανομής d. Παρουσιάζουμε παρακάτω ένα πολύ βασικό θεώρημα με το οποίο καθίσταται δυνατός ο υπολογισμός της κλίσης [4].

Θεώρημα 3. *Για κάθε διαφορίσιμη πολιτική $\pi_\theta(a,s)$ και για κάθε ένα από τα αντικειμενικά κόστη J1,J2,J3 η κλίση ως προς ϑ εκφράζεται ως*

$$\nabla_\theta J(\theta) = E_\pi[\nabla_\theta log\pi_\theta(s,a)Q^\pi(s,a)] \tag{4.11}$$

4.2.2 Κλίση πολιτικής Monte carlo (REINFORCE)

Θα επιχειρήσουμε να παρουσιάσουμε τα βήματα με τα οποία προκύπτει ο αλγόριθμος κλίσης monte carlo.Αρχικά θα εξετάσουμε επεισοδιακά περιβάλλοντα όπως και το παιχνίδι που εξετάζουμε σε αυτή την εργασία.Έχοντας νευρωνικό δίκτυο με παράμετρο ένα διάνυσμα ϑ θέλουμε να εξετάσουμε την καταλληλότητα μιας πολιτικής π_θ για μέγιστη επιστροφή αμοιβών.Ορίζουμε δηλαδή το αντικειμενικό κριτήριο:

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right] \tag{4.12}$$

Δηλαδή μας επιστρέφει την αναμενόμενη αμοιβή που πετυχαίνει μια πολιτική π_θ πάνω σε ένα χρονικό ορίζοντα t. Ο συμβολισμός $\tau \sim \pi_\theta(\tau)$ δηλώνει ότι δειγματολειπτούμε τροχιές τ από την κατανομή πιθανοτήτων που μοντελοποιεί το νευρωνικό μας δίκτυο. Η κατανομή αυτή μπορεί να αποσυντεθεί ως:

$$\begin{aligned}\pi_\theta(\tau) &= p(s_1, a_1, ..., s_\tau, a_\tau) \\ &= p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t, s_t)p(s_{t+1}|a_t, s_t)\end{aligned}$$

και ορίζουμε την βέλτιστη πολιτική ως:

$$\pi* = \pi_{\theta*} = argmax_\theta J(\theta) \tag{4.13}$$

Θέλουμε πρακτικά να απαλλαγούμε από το ολοκλήρωμα που υπάρχει στη σχέση της αναμενόμενης τιμής για το κόστος και χρησιμοποιούμε την προσέγγιση Monte carlo η οποία όπως γνωρίζουμε μας λέει ότι:

$$\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} f(x_i)_{x_i\sim\pi(x)} = E_\pi[f(x)] \tag{4.14}$$

Δηλαδή αν αν δειγματοληπτούμε $f(x_i)$ παίρνοντας τα x_i από την κατανομή p(x) N φορές τότε αν το N είναι αρκετά μεγάλο έχουμε μια καλή προσέγγιση της μέσης τιμής του f(x).Οπότε ξαναγράφουμε το κριτήριο κόστους ως:

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r(s_{i,t}, a_{i,t}) \tag{4.15}$$

όπου τα N δείγματα λαμβάνονται από την πιθανοτική κατανομή π_θ απλά τρέχοντας την πολιτική N φορές.Έτσι το κριτήριο κόστους λαμβάνει πιο εύχρηστη μορφή. Για την εύρεση του βέλτιστου ϑ* χρησιμοποιούμε μέθοδο ανάβαση κλίσης.Για να απλοποιήσουμε τον συμβολισμό ορίζουμε την συνολική αμοιβή μιας τροχιάς τ ως:

$$r(\tau) = \sum_t r(s_{i,t}, a_{i,t})$$

Χρησιμοποιώντας την σχέση παραγώγισης σύνθετης συνάρτησης λογαρίθμου μπορούμε να γράψουμε την κλίση του κόστους ως:

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau)r(\tau)d\tau \tag{4.16}$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta\pi_\theta(\tau)r(\tau)d\tau = \int \pi_\theta(\tau)\nabla_\theta log\pi_\theta(\tau)r(\tau)d\tau \tag{4.17}$$

$$= E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta log\pi_\theta(\tau)r(\tau)] \tag{4.18}$$

Αν πάρουμε το λογάριθμο της τροχιάς έχουμε:

$$log\pi_\theta = logp(s_1) + \sum_{t=1}^{T} log\pi_\theta(a_t, s_t) + logp(s_{t+1}|a_t, s_t)$$

οπότε η παράγωγος γράφεται:

$$\nabla_\theta log\pi_\theta = \nabla_\theta \left[logp(s_1) + \sum_{t=1}^{T} log\pi_\theta(a_t, s_t) + logp(s_{t+1}|a_t, s_t) \right]$$

Η παράγωγος του πρώτου και του τελευταίου όρου προφανώς μηδενίζεται οπότε συνολικά έχουμε:

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta log\pi_\theta(a_t, s_t) \right) \left(\sum_t r(s_t, a_t) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta log\pi_\theta(a_t, s_t) \right) \left(\sum_t r(s_{i,t}, a_{i,t}) \right) \right]$$

Βασική ιδέα αυτού του αλγορίθμου όπως και σε όλες τις μεθόδους monte carlo είναι η χρήση της επιστροφής G_t ως ένα αμερόληπτο δείγμα της $Q^\pi(s, a)$ και χρησιμοποιώντας το θεώρημα κλίσης πολιτικής ενημερώνουμε τα βάρη του νευρωνικού κατά $\Delta\theta = \alpha\nabla_\theta log\pi_\theta(s, a)G_t$ με στοχαστική ανάβαση κλίσης [4]. Συνολικά ο αλγόριθμος παρουσιάζεται παρακάτω:

Algorithm 4: Ο Αλγόριθμος REINFORCE

```
Initialize ϑ;
for each episode i sample trajectory τ {s_1, a_1, r_2, ..., r_τ} ~ π_θ do
    Initialize S;
    for t=0 to t=τ-1 do
        recall G_t;
        θ ← θ + α∇_θ log π_θ(s_t, a_t) G_t;
    end
end
```

4.2.3 Βελτιώσεις στον Αλγόριθμο REINFORCE

Χρήση Βάσης (Baseline) Η μέθοδος για ανάβαση κλίσης όπως περιγράφηκε παραπάνω για μεγιστοποίηση του αντικειμενικού κριτηρίου J αν και απλή ως υλοποίηση έχει κάποια μειονεκτήματα.Όταν αντιμετωπίζουμε ένα πρόβλημα με μεθόδους μηχανικής μάθησης θέλουμε τα δεδομένα να είναι όσον δυνατόν ασυσχέτιστα και με μικρή ευαισθησία. Στη μέθοδο Monte carlo η κλίση της πολιτικής υποφέρει από μεγάλη διασπορά και δυσκολία στη σύγκλιση. Αυτό συμβαίνει γιατί παίζουμε το παιχνίδι για μια ολόκληρη τροχιά και κρατάμε τις ακριβείς αμοιβές που προκύπτουν σε κάθε βήμα. Ωστόσο η στοχαστική πολιτική μπορεί να διαλέξει διαφορετικές δράσεις σε διαφορετικά επεισόδια και μια μικρή αλλαγή μπορεί να επηρεάσει σε μεγάλο βαθμό το αποτέλεσμα. Για αυτό μπορεί να μην έχουμε κάποια μεροληψία αλλά έχουμε πολύ μεγάλη διασπορά η οποία αποτρέπει το νευρωνικό να μάθει. Κάποιες αμοιβές μπορεί να προσπαθούν να αυξήσουν το λογάριθμο της πιθανοφάνειας ενώ άλλες να τον μειώσουν με αποτέλεσμα να έχουμε αστάθεια. Οπότε για να μειώσουμε την ευαισθησία που υπάρχει όσον αφορά τις δράσεις μειώνουμε την ευαισθησία για τις αμοιβές που δειγματολειπτούμε.
Η λογική είναι ότι μπορούμε να αφαιρέσουμε έναν όρο από την συνάρτηση αμοιβής που έχουμε να μεγιστοποιήσουμε αρκεί να μην εξαρτάται από την παράμετρο θ.Ένας καλός όρος είναι η συνάρτηση V(s) οπότε αντί να χρησιμοποιούμε την συνολική αμοιβή έχουμε την διαφορά της με την συνάρτηση αξίας.Έτσι αντί να έχουμε:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta log\pi_\theta(a_t,s_t)\right)\underbrace{\left(\sum_{t} r(s_{i,t},a_{i,t})\right)}_{Q(s,a)}\right] \tag{4.19}$$

έχουμε:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta log\pi_\theta(a_t,s_t)\right)\left(Q(s_{i,t},a_{i,t})-V(s_i)\right)\right] \tag{4.20}$$

Ορίζουμε την συνάρτηση πλεονεκτήματος:

$$A^\pi(s_t,a_t) = Q(s_t,a_t) - V(s_t)$$

Στη βαθιά μηχανική μάθηση θέλουμε τα δεδομένα μας να είναι κεντραρισμένα στο μηδέν.Επίσης διαισθητικά στην ενισχυτική μάθηση θέλουμε να ξέρουμε πότε μια δράση έχει καλύτερα αποτελέσματα από το μέσο όρο.Η κλίση πολιτικής προσπαθεί να αυξήσει την πιθανότητα μιας τροχιάς ακόμα και αν έχει μικρή αμοιβή. Σε ένα παράδειγμα όπου η τροχιά Α έχει συνολική αμοιβή +10 και η τροχιά Β έχει συνολική αμοιβή -10 ο αλγόριθμος θα αυξήσει την πιθανότητα της τροχιάς Α και θα μειώσει την Β. Αντίθετα αν οι δυο τροχιές είχαν αμοιβές +10 και +1 αντίστοιχα τότε θα αυξήσει την πιθανότητα και των δύο παρόλο που οι άνθρωποι διαισθητικά θα μείωναν την πιθανότητα της τροχιάς Β. Η συνάρτηση V αποτελεί για ακριβώς αυτόν το λόγο ένα καλό μέτρο σύγκρισης καθώς μας βοηθάει να συγκρίνουμε με τον μέσο όρο. Παρουσιάζουμε τον συνολικό αλγόριθμο κλίσης με monte carlo εντάσσοντας ένα μέτρο σύγκρισης b.

Algorithm 5: "Vanilla" policy gradient algorithm

Initialize policy parameter ϑ, baseline b;
for *iteration=1,2,...* **do**
  collect a set of trajectories by executing the current policy At each timestep in each trajectory ,compute the return $G_t = \sum_{t'=t}^{\tau-1} \gamma^{t'-t} r^{t'}$ and the advantage estimate $\hat{A}_t = G_t - b(s_t)$;
  Re-fit the baseline by minimizing $\|\|b(s_t) - G_t\|\|^2$ summed over all trajectories and timesteps;
  update the policy parameter ϑ using gradient estimate as a sum of terms $\nabla_\theta log\pi_\theta(a_t|s_t,\theta)\hat{A}_t$
end

Αιτιατότητα Σε πολλές εφαρμογές όπως τα περιβάλλοντα παιχνιδιών μελλοντικές δράσεις δεν θέλουμε να αλλάζουν δράσεις του παρελθόντος. Μια δράση σε μία παρούσα κατάσταση θέλουμε να επηρεάζει μόνο το μέλλον οπότε μπορούμε να τροποποιήσουμε την αντικειμενική συνάρτηση κριτηρίου ως:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta log\pi_\theta(a_t, s_t) \right) \left(\sum_{t' \nmid t}^{T} r(s_{i,t'}, a_{i,t'}) \right) \right] \tag{4.21}$$

Η Ελάττωση στις αμοιβές Η ελάττωση στις μακροπρόθεσμες αμοιβές είναι ένας άλλος τρόπος να μειώσουμε την ευαισθησία στις διακριτές δράσεις. Κατά τα γνωστά χρησιμοποιούμε για τον υπολογισμό των συνολικών αμοιβών τη σχέση:

$$Q^{\pi,\gamma}(s,a) \leftarrow r_0 + \gamma r_1 + \gamma^2 r_2 + ...|s_0 = s, a_0 = a$$

και το αντικειμενικό κριτήριο γίνεται:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta log\pi_\theta(a_t, s_t) \right) \left(\sum_{t' \nmid t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) \right) \right] \tag{4.22}$$

4.2.4 Μέθοδοι Δράστη-Κριτή (Actor-Critic

Παρόλο που η μέθοδος REINFORCE με μέτρο σύγκρισης μαθαίνει τόσο την πολιτική όσο και τη συνάρτηση της τιμής κατάστασης, δεν θεωρούμε ότι είναι μια μέθοδος κριτή-δράστη, επειδή η συνάρτηση της κατάστασης τιμής χρησιμοποιείται μόνο ως γραμμή βάσης και όχι ως κριτής. Δηλαδή, δεν χρησιμοποιείται για bootstrapping (ενημέρωση της εκτίμησης αξίας για μια κατάσταση από τις εκτιμώμενες τιμές των επόμενων καταστάσεων), αλλά μόνο ως βάση για την κατάσταση της οποίας η εκτίμηση ενημερώνεται. Αυτή είναι μια χρήσιμη διάκριση, διότι μόνο με την τεχνική bootstrap εισάγουμε προκατάληψη και ασυμπτωτική εξάρτηση από την ποιότητα της προσέγγισης της συνάρτησης. Δυο βασικά στοιχεία στην κλίση πολιτικής είναι το μοντέλο πολιτικής και η συνάρτηση αξίας. Μπορούμε να επιχειρήσουμε να "μάθουμε" την συνάρτηση αξίας σε σχέση με την πολιτική καθώς γνωρίζοντας την συνάρτηση αξίας μπορεί να βοηθήσει στην ενημέρωση της πολιτικής μειώνοντας τη διακύμανση που υπάρχει στον απλό αλγόριθμο κλίσης πολιτικής.Στις μεθόδους δράστη-κριτή έχουμε δύο μοντέλα τα οποία μπορούν να έχουν ακόμα και κοινές παραμέτρους.

- **Ο κριτής** ενημερώνει ένα σετ παραμέτρων w από το οποίο εξαρτάται ανάλογα με τον αλγόριθμο είτε η συνάρτηση δράσης αξίας $Q_w(s,a)$ είτε η συνάρτηση αξίας $V_w(s)$
- **Ο δράστης** ενημερώνει τις παραμέτρους ϑ της πολιτικής $\pi_\theta(a|s)$ στην κατεύθυνση που υποδεικνύει ο κριτής

Παρουσιάζουμε τον πιο απλό αλγόριθμο δράστη κριτή [4] στον οποίο με ένα δεύτερο νευρωνικό δίκτυο μοντελοποιούμε την συνάρτηση Q

Algorithm 6: Q actor critic algorithm

Initialize s,ϑ,w;
sample $a \sim \pi_\theta(a|s)$;
for *t=1...T* **do**
  sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$;
  Sample the next action $a' \sim \pi_\theta(a'|s')$;
  update policy parameter ϑ $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta log\pi_\theta(a|s)$;
  compute the correction TD error for action-value at time t
   $\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$ and use it to update w
   $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$;
  $a \leftarrow a'$;
  $s \leftarrow s'$;
end

Asynchronous Advantage actor-critic (A3C) Ο αλγόριθμος A3C [5] αποτελεί μια από τις πιο κλασικές μεθόδους για κλίση πολιτικής και εστιάζει στη παράλληλη μάθηση.Οι κριτές μαθαίνουν την συνάρτηση αξίας ενώ πολλαπλοί δράστες εκπαιδεύονται παράλληλα και συγχρονίζονται με καθολικές παραμέτρους ανά χρονικά βήματα.Θα χρησιμοποιήσουμε ως παράδειγμα την συνάρτηση αξίας .Καλούμαστε να ελαχιστοποιήσουμε το μέσο τετραγωνικό σφάλμα $J_v(w) = (G_t - V_w(s))^2$ και εφαρμόζουμε κατάβαση κλίσης.Αυτή η συνάρτηση αξίας είναι που χρησιμοποιείται ως μέτρο σύγκρισης (baseline) στην ενημέρωση κλίσης πολιτικής. Παρακάτω παρουσιάζουμε συνολικά τον αλγόριθμο.Τα βάρη w,ϑ ανανεώνονται κάθε φορά κατά λίγο στη κατεύθυνση της κάθε διεργασίας ανεξάρτητα.

Algorithm 7: A3C algorithm

We have global parameters ϑ,w and similar thread specific w',ϑ';
Initialize timestep t=1;
while $T \leq T_{max}$ **do**
  reset gradient dϑ=0 and dw=0;
  synchronize thread-specific parameters with global ones w=w',ϑ=ϑ';
  $t_{start} = t$ and sample a starting state s_t;
  while s_t*!=terminal and* $t - t_{start} \leq T_{max}$ **do**
    pick the action $A_t \sim \pi_\theta(A_t|S_t)$ and receive a new reward R_t and new state s_{t+1};
    t=t+1 and T=T+1;
  end
  Initialize the variable that holds the the return estimation $R = V_{w'}(s_t)$ (zero if s terminal);
  for *i=t-1,...*t_s*tart* **do**
    $R \leftarrow \gamma R + R_i$ *R is a MC measure of* G_t*;*
    accumulate gradients w.r.t ϑ*'* $d\theta \leftarrow d\theta + \nabla_{\theta'} log \pi_{\theta'}(a_i|s_i)(R - V_{w'}(s_i))$*;*
    accumulate gradients w.r.t w' dw $\leftarrow dw + 2(R - V_{w'}(s_i))\nabla_{w'}(R - V_{w'}(s_i))$*;*
  end
  update asynchronously ϑ *using* $d\vartheta$ *and w using dw;*
end

4.3 Deep Q Networks

4.3.1 DQN με επαναφορά εμπειρίας

Όταν προσπαθούμε να εφαρμόσουμε μεθόδους βαθιάς μάθησης στην ενισχυτική μάθηση αντιμετωπίζουμε κάποιες δυσκολίες.Για αρχή η εκπαίδευση νευρωνικών δικτύων απαιτεί τεράστιο αριθμό δειγμάτων μαρκαρισμένα με τον αντίστοιχο στόχο που θέλουμε να εκπαιδεύσουμε το δίκτυο.Αντίθετα στην ενισχυτική μάθηση η εκπαίδευση γίνεται με βάση απλά ένα σήμα αμοιβής το οποίο εκτός από θόρυβο μπορεί να έρχεται και με καθυστέρηση πολλαπλών βημάτων στο μέλλον. Για παράδειγμα σε παιχνίδια όπως το σκάκι το σήμα της αμοιβής μας είναι μοναδικό και το λαμβάνουμε μόνο κατά τη λήξη του παιχνιδιού εφόσον κερδίσαμε η χάσαμε. Ο ενδιάμεσος αυτός χρόνος μεταξύ των δράσεων για τη λήψη της αμοιβής σε συνδυασμό με το γεγονός ότι στα δείγματα που λαμβάνουμε υπάρχει ισχυρή συσχέτιση μεταξύ τους καθιστά την εκπαίδευση νευρωνικών δύσκολη. Σε αυτά έρχεται να προστεθεί και το γεγονός ότι οι κατανομές που ακολουθούν τα δεδομένα δεν είναι στατικές αλλά αλλάζουν συνεχώς καθώς ο πράκτορας μας μαθαίνει νέες συμπεριφορές και προσθέτει ακόμα μεγαλύτερη αστάθεια στους αλγορίθμους βαθιάς μάθησης που υποθέτουν σταθερές κατανομές.Ο αλγόριθμος DQN όπως παρουσιάστηκε στο άρθρο της deepmind [6] μπόρεσε να παρακάμψει στη πράξη αυτά τα προβλήματα και να πετύχει αξιοσημείωτα αποτελέσματα σε παιχνίδια atari χρησιμοποιώντας ένα συνελικτικό νευρωνικό δίκτυο που δέχεται ως είσοδο κατευθείαν τα Pixels της εικόνας του παιχνιδιού.Αποτελεί μια επέκταση πάνω στον αλγόριθμο Q-learning που παρουσιάσαμε χρησιμοποιώντας ένα νευρωνικό για την προσέγγιση της συνάρτησης Q καθώς η πινακοειδής μορφή της είναι αδύνατο να επιλύσει προβλήματα οπού ο συνδυασμός δράσεων καταστάσεων είναι αρκετά μεγάλος.
Αρχικά υπενθυμίζουμε ότι η αναμενόμενη αμοιβή ορίζεται ως:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{4.23}$$

ως το άθροισμα όλων των ελλατωμένων αμοιβών που λαμβάνει ο πράκτορας από την χρονική στιγμή t μέχρι το τέλος του επεισοδίου. Η συνάρτηση Q για μια πολιτική π ως:

$$Q_\pi(s,a) = E_\pi[R_t | s_t = s, a_t = a, \pi] \tag{4.24}$$

ενώ η βέλτιστη Q* δίνεται απο την εξίσωση bellman ως:

$$Q^*(s,a) = E_{s' \sim \varepsilon}[r_t + \gamma \max_a Q(s',a') | s, a] \tag{4.25}$$

Παραμετροποιούμε την βέλτιστη συνάρτηση Q με τα βάρη του νευρωνικού δικτύου ϑ δηλαδή $Q^*(s,a) \approx Q(s,a,\theta)$ και ορίζουμε ως κόστος προς ελαχιστοποίηση το:

$$L(\theta_i) = E[(y_i - Q(s,a,\theta_i))^2] \tag{4.26}$$

όπου:

$$y_i = E_{s' \sim \varepsilon}[r_t + \gamma \max_{a'} Q(s',a,\theta_{i-1}) | s, a] \tag{4.27}$$

είναι ο στόχος στην επανάληψη i

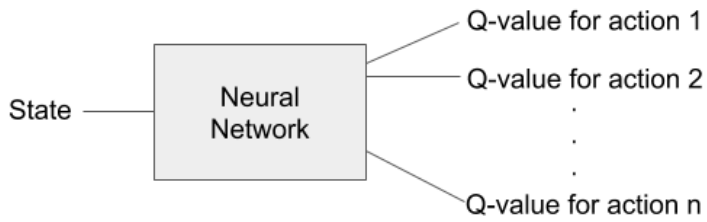

Figure 4.2: Q network

Όπως φαίνεται και από την παραπάνω εικόνα στο δίκτυο περνάει η κωδικοποιημένη κατάσταση και το επίπεδο εξόδου διαθέτει ένα κόμβο για κάθε δράση που μπορεί να εκτελέσει ο πράκτορας στο περιβάλλον.Το πρόβλημα που προκύπτει τώρα είναι αν ορίσουμε το αντικειμενικό κριτήριο το οποίο θα κληθούμε να ελαχιστοποιήσουμε για το κομμάτι της βελτιστοποίησης. Αν ακολουθούσαμε την συμβατική μέθοδο ορίζοντας απλά ως¨$L = (target - output)^2$ με τον στόχο να προκύπτει από την εξίσωση bellman τότε η εκπαίδευση του δικτύου γίνεται προβληματική. Αυτό συμβαίνει γιατί ο στόχος περιέχει όπως βλέπουμε έναν όρο της μορφής $\gamma \max_{a'} Q(s', a', \theta)$ τον οποίο για να υπολογίσουμε περνάμε απλά από το νευρωνικό την διαδοχική κατάσταση s' και διαλέγουμε τον κόμβο με την μεγαλύτερη τιμή στο επίπεδο εξόδου. Στη συνέχεια θα εφαρμόζαμε κατάβαση κλίσης της συνάρτησης κόστους και τον αλγόριθμο οπισθοδιάδοσης για ενημέρωση των βαρών.Αυτό έχει ως αποτέλεσμα η έξοδος του δικτύου να προσπαθεί να έρθει κοντύτερα στο δεξί μέλος της εξίσωσης bellman. Όμως κάθε φορά που αλλάζουν τα βάρη και πλησιάζουν την προσέγγιση της εξίσωσης αλλάζει και ο στόχος καθώς εμπεριέχει την ίδια την έξοδο του νευρωνικού. Αυτό δημιουργεί μια κατάσταση ατέρμονης εκπαίδευσης στην οποία δεν μπορεί να υπάρξει σύγκλιση προς την βέλτιστη q.

Διπλό DQN Για τον λόγο αυτό ο αλγόριθμος κάνει χρήση ενός δευτέρου δικτύου (target network) πανομοιότυπης αρχιτεκτονικής με το αρχικό και βάρη θ^- το οποίο χρησιμοποιείται αποκλειστικά για τον υπολογισμό του στόχου από την εξίσωση bellman. Η διαφορά τώρα είναι ότι για ένα προκαθορισμένο αριθμό από περάσματα καταστάσεων στο αρχικό δίκτυο το δεύτερο διατηρεί τα βάρη του αμετάβλητα δίνοντας έτσι την ευκαιρία στο πρώτο δίκτυο να προσεγγίσει την έξοδο του δεύτερου. Ύστερα τα βάρη του αρχικού Q netwrork αντιγράφονται στο target network και η διαδικασία επαναλαμβάνεται. Με αυτό το τρόπο ανάγεται το πρόβλημα της ενισχυτικής μάθησης σε βαθιά μάθηση.

Επαναφορά εμπειρίας (experience replay) Όταν μαθαίνουμε απευθείας από τροχιές τότε διαδοχικές καταστάσεις είναι προσωρινά συσχετισμένες ενώ επιπλέον οι κατανομές στα δείγματα αλλάζουν γρήγορα καθώς οι παράμετροι του δικτύου καθορίζουν το επόμενο δείγμα πάνω στο οποίο θα ενημερωθούν οι παράμετροι αυτές καθ αυτές. Για να σπάσουμε αυτή την εξάρτηση χρησιμοποιούμε την επαναφορά εμπειρίας. Πρακτικά η επαναφορά εμπειρίας μοντελοποιείται με έναν buffer στον οποίο αποθηκεύεται ένα υποσύνολο από προηγούμενα δείγματα και τους αντίστοιχους στόχους τους που προέκυψαν από την αλληλεπίδραση του πράκτορα με το περιβάλλον.Για κάθε χρονικό βήμα αποθηκεύουμε συγκεκριμένα την κατάσταση , την δράση από αυτή τη κατάσταση , την αμοιβή που λάβαμε και την νέα κατάσταση στην οποία βρεθήκαμε.Για την ενημέρωση των βαρών λαμβάνουμε ένα σύνολο

(batch) από αυτό τον Buffer διαλέγοντας τυχαία δείγματα από τον buffer παρακάμπτοντας την αλληλεξάρτηση που υπάρχει μεταξύ τους. Ο buffer είναι κυκλικός και δείγματα τα οποία βρίσκονται μέσα για έναν αριθμό ενημέρωσης των βαρών του δικτύου δίνουν τη θέση τους στα επόμενα. Επίσης με αυτό το τρόπο τα δείγματα μπορούν να επαναχρησιμοποιηθούν για την ενημέρωση των βαρών όσο βρίσκονται στον buffer.Παρακάτω έχουμε το dataset $D\bar{e}_1, e_2, ...e_N$ που δημιουργούν δείγματα της μορφής $(e_t, a_t, r_{t+1}, s_{t+1})$

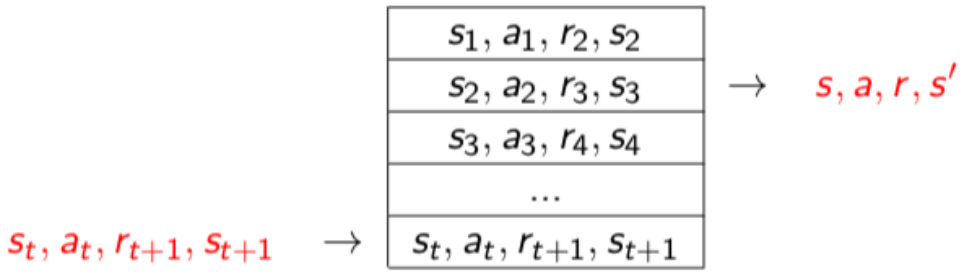

Figure 4.3: Memory Replay Buffer

Παρακάτω παρουσιάζουμε τον συνολικό αλγόριθμο DQN με επαναφορά εμπειρίας.

Algorithm 8: DQN with experience replay

Initialize replay memory D to capacity N;
Initialize action-value function Q with random weights ϑ;
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$;
for *episode=1,...M* **do**
  Initilize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\varphi_1 = \varphi(s_1)$;
  for *t=1,..T* **do**
    with probability ε select a random action a_t;
    otherwise select $a_t = argmax_a Q(\varphi(s_t), a; \theta)$;
    execute action a_t in emulator and observe reward r_t and image x_{t+1};
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\varphi_{t+1} = \varphi(s_{t+1})$;
    store transition $(\varphi_t, a_t, r_t, \varphi_{t+1})$ in D;
    Sample random mini-batch of transitions $(\varphi_j, a_j, r_j, \varphi_{j+1})$ from D;
    set $y_j = \begin{cases} \mathrm{r}_j \text{ if episode terminates at step j+1} \\ \mathrm{r}_j + \gamma \max_{a'} \hat{Q}(\varphi_{j+1}, a'; \theta^-) \text{ otherwise} \end{cases}$ Perform a gradient descent step on $(y_j - Q(\varphi_j, a_j; \theta))^2$ with respect to the network parameters ϑ;
    Every C steps $\hat{Q} = Q$;
  end
end

Όπου η συνάρτηση φ χρησιμοποιείται για την αναπαράσταση της κατάστασης από έναν αριθμό από προηγούμενα frames ώστε να συμπεριλαμβάνεται η έννοια της κινήσεις στα παιχνίδια atari.

4.3.2 DQN Μονομαχίας

Η τροποποίηση (βλέπε [7]) που παρουσιάζουμε σε αυτή την παράγραφο σχετικά με τον αλγόριθμό DQN βασίζεται στο γεγονός ότι σε κάποιες εφαρμογές και παιχνίδια δεν είναι αναγκαίο να γνωρίζουμε την αξία κάθε δράσης σε κάθε χρονικό βήμα. Η αρχιτεκτονική του δικτύου που χρησιμοποιείται εδώ διαχωρίζει την αναπαράσταση της αξίας για κάθε κατάσταση και του **πλεονεκτήματος** της δράσης για την αντίστοιχη κατάσταση. σε δύο κανάλια.Υπενθυμίζουμε ότι η συνάρτηση του πλεονεκτήματος προκύπτει αφαιρώντας από την συνάρτηση Q την συνάρτηση V:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s,a) \tag{4.28}$$

Αφού η συνάρτηση Q αναπαριστά την αξία να διαλέξουμε μια δράση δεδομένης μιας κατάστασης και η συνάρτηση V αναπαριστά την αξία να βρισκόμαστε σε αυτή τη κατάσταση ανεξάρτητα της δράσης που θα επιλέξουμε τότε διαισθητικά μπορούμε να καταλάβουμε ότι η συνάρτηση A μας δείχνει πόσο καλύτερη είναι η δράση a σε σχέση με τις άλλες δράσης από την ίδια κατάσταση s.

Έτσι λοιπόν διαχωρίζοντας δυο εκτιμητές η αρχιτεκτονική μονομαχίας μπορεί να μάθει ποίες καταστάσεις είναι και ποιες δεν είναι συμφέρουσες χωρίς να χρειάζεται να λαμβάνει υπόψιν κάθε δράση από αυτές. Αυτό είναι εξαιρετικά χρήσιμο όταν οι δράσεις δεν έχουν κάποιο φυσικό νόημα στο περιβάλλον για παράδειγμα προσπαθούμε με ένα αυτοκίνητο να αποφεύγουμε εμπόδια στο δρόμο αλλά οι κινήσεις του οχήματος δεν έχουν κάποια σημασία όταν ο δρόμος είναι άδειος.Ενδεικτικά παρουσιάζουμε μια αρχιτεκτονική DQN μονομαχίας αποτελούμενης από 3 συνελικτικά και δυο εμπρόσθιας τροφοδότησης επίπεδα. Έτσι λοιπόν

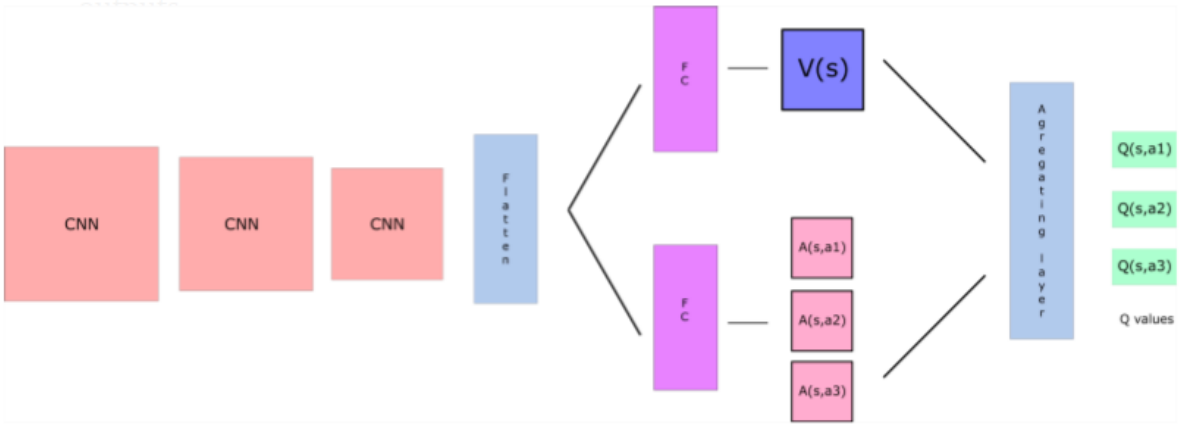

Figure 4.4: Αρχιτεκτονική DQN μονομαχίας

θα μπορούσαμε για να πάρουμε την συνάρτηση Q απλά να προσθέσουμε τις δύο τιμές:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) \tag{4.29}$$

Ωστόσο η απλή πρόσθεση μπορεί να αποδειχθεί προβληματική καθώς δεν γνωρίζουμε πόσο ακριβής είναι οι προσεγγίσεις στο ενδιάμεσο στάδιο και το άθροισμα τους μπορεί να αυξάνει το σφάλμα. Επίσης από την συνολική Q του τελευταίου σταδίου μετά τον συνδυασμό της V και της A είναι δύσκολο να αποσυντεθεί πάλι στα επιμέρους κομμάτια της.

Έτσι το τελευταίο επίπεδο πραγματοποιεί την παρακάτω αντιστοίχηση:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \max_{a' \in A} A(s,a;\theta,\alpha)) \tag{4.30}$$

Με αυτό το τρόπο μπορούμε να μετριάσουμε το πρόβλημα της αναγνωρισιμότητας καθώς αναγκάζουμε την συνάρτηση Q να είναι ίση με την συνάρτηση V για την δράση μεγιστοποίησης.Ένας άλλος τρόπος όπως προτείνεται στην αντίστοιχη αναφορά είναι να αφαιρέσουμε από την συνάρτηση πλεονεκτήματος μια ποσότητα μέσου όρου πάνω στις δράσεις:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a' \in A} A(s,a;\theta,\alpha)) \tag{4.31}$$

Το κριτήριο προς ελαχιστοποίηση είναι όμοιο με την κλασσική αρχιτεκτονική DQN:

$$L(\Theta) = \frac{1}{N}\sum_{i=1}^{N}(Q(s_i,a_i;\theta) - \hat{Q}(s_i,a_i,\theta))^2 \tag{4.32}$$

όπου:

$$Q(s_i,a_i;\theta) = R(s_i,a_i) + \gamma \max_{a'} Q(s_i',a_i') \tag{4.33}$$

4.3.3 DQN Με προτεραιότητα επαναφοράς εμπειρίας

Η βασική ιδέα στην οποία στηρίζεται αυτή η τροποποίηση (βλέπε [8]) είναι το γεγονός ότι καθώς δειγματολειπτούμε εμπειρίες από την μνήμη κάποιες είναι σημαντικότερες από τις άλλες. Καθώς λοιπόν δειγματολειπτούμε ομοιόμορφα όλες οι εμπειρίες έχουν την ίδια πιθανότητα να επιλεγούν για εκπαίδευση του νευρωνικού. Αν αντίθετα επιλέγουμε εμπειρίες με βάρη τότε οι πιο εποικοδομητικές θα επιλέγονται περισσότερο από άλλες χαμηλότερης ποιότητας. Στόχος μας είναι λοιπόν να κατασκευάσουμε μια κατανομή δειγματοληψίας η οποία θα είναι ανάλογη του λάθους που προκύπτει μετά από ένα πέρασμα των δειγμάτων από το δίκτυο.Αυτό σημαίνει ότι θέλουμε να κρατάμε εμπειρίες των οποίων η διαφορά μεταξύ της αναμενόμενης αμοιβής και της πραγματικής αμοιβής είναι σημαντική και προσφέρει μάθηση στο δίκτυο. Η λογική αυτή μπορεί να αποδειχθεί εξαιρετικά χρήσιμη στην αρχή της εκπαίδευσης όπου για παράδειγμα εκπαιδεύουμε έναν πράκτορα σε ένα παιχνίδι connect 3 όπου αρχικά θα παρατηρούνται συνέχεια ήττες αλλά στην σπάνια περίπτωση νίκης θέλουμε να προσανατολίσουμε το δίκτυο προς αυτή την κατεύθυνση δίνοντας βάση στις δράσεις που οδήγησαν στη νίκη.
Στην επαναφορά εμπειρίας με προτεραιότητα εκτός από τον απλό container χρειάζεται να αντιστοιχίσουμε κάθε εμπειρία με την αντίστοιχη πιθανότητα προτεραιότητας και βάρος. Η προτεραιότητα ανανεώνεται σύμφωνα το σφάλμα του νευρωνικού ενώ τα βάρη ανανεώνονται σύμφωνα με αυτές τις πιθανότητες.Επίσης έχουμε δυο υπερπαραμέτρους α και β που ελέγχουν την προτεραιότητα με την έννοια ότι προς το τέλος της εκπαίδευσης θέλουμε να έχουμε ομοιόμορφη δειγματοληψία για να αποφύγουμε το overfitting από εμπειρίες που διαλέγονται συνεχώς εις βάρος κάποιον άλλων. Συγκεκριμένα οι εξισώσεις για τη στοχαστική κατανομή και τα βάρη δίνονται ως:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{4.34}$$

και

$$w_i = \left(\frac{1}{N}\frac{1}{p_i}\right)^\beta \tag{4.35}$$

Όσον αφορά τις εκ των προτέρων πιθανότητες μπορούμε να έχουμε δυο περιπτώσεις. Είτε να θεωρήσουμε $p_i = |\delta_i| + \varepsilon$ όπου δ_i το TD-σφάλμα είτε μια έμεση ιεράρχηση προτεραιότητας με $p_i = \frac{1}{rank(i)}$ οπου rank(i) η τάξη της μετάβασης i όταν η μνήμη επανάληψης ταξινομείται σύμφωνα με το δ_i Η εκτίμηση της αναμενόμενης τιμής με στοχαστικές ενημερώσεις βασίζεται σε αυτές τις ενημερώσεις που αντιστοιχούν στην ίδια κατανομή με την προσδοκία τους. Η αναπαραγωγή με προτεραιότητα εισάγει προκατάληψη επειδή αλλάζει αυτή την κατανομή με ανεξέλεγκτο τρόπο και ως εκ τούτου αλλάζει τη λύση με την οποία οι εκτιμήσεις θα συγκλίνουν (ακόμα και αν η πολιτική και η κατανομή των καταστάσεων είναι σταθερές). Μπορούμε να διορθώσουμε αυτή την προκατάληψη χρησιμοποιώντας τα βάρη της δειγματοληψίας όπως ορίστηκαν παραπάνω αντικαθιστώντας το TD σφάλμα δ_i στον αλγόριθμο Q-learning με $w_i\delta_i$. Παρακάτω παρουσιάζουμε τον συνολικό αλγόριθμο DQN με προτεραιότητα επαναφορά εμπειρίας.

Algorithm 9: DQN with prioritized experience replay

Input: minibatch k, step-size η, replay period K and size N ,exponents α and β, budget T.;
Initialize replay Memory M=∅ Δ=0,$p_1 = 1$;
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$;
for *t=1 to T* **do**
  Observe S_t, R_t, γ_t;
  Store Transition $(S_{t-1}, A_{t-1}, R_t, \gamma, S_t)$ in Memory with maximal priority $p_t = \max_{i \leq t} p_i$;
  if $t \equiv 0 mod K$ **then**
    for *j=1 to K* **do**
      Sample transition $j \sim P(j) = \frac{p_j^\alpha}{\sum_k p_k^\alpha}$;
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$;
      Compute TD-error:

$$\delta_j = R_j + \gamma Q_{target}(S_{j,a} Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$$

      Update transition priority: $p_j \leftarrow |\delta_j|$;
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \delta_j \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end
    Update weights $\theta \leftarrow \theta + \eta\Delta$, reset Δ=0;
    From time to time copy weights into target network $\theta \leftarrow \theta_{target}$;
  end
  choose action $A_t \sim \pi_\theta(S_t)$;
end

Κεφάλαιο 5

Πειραματικό μέρος στο παιχνίδι Checkers

5.1 Το παιχνίδι Checkers

Για τους σκοπούς αυτής της εργασίας εξετάζουμε την εκδοχή English draughts η οποία παίζεται σε σκακιέρα 8x8 με τα κομμάτια να τοποθετούνται στην αρχική θέση ως εξής:

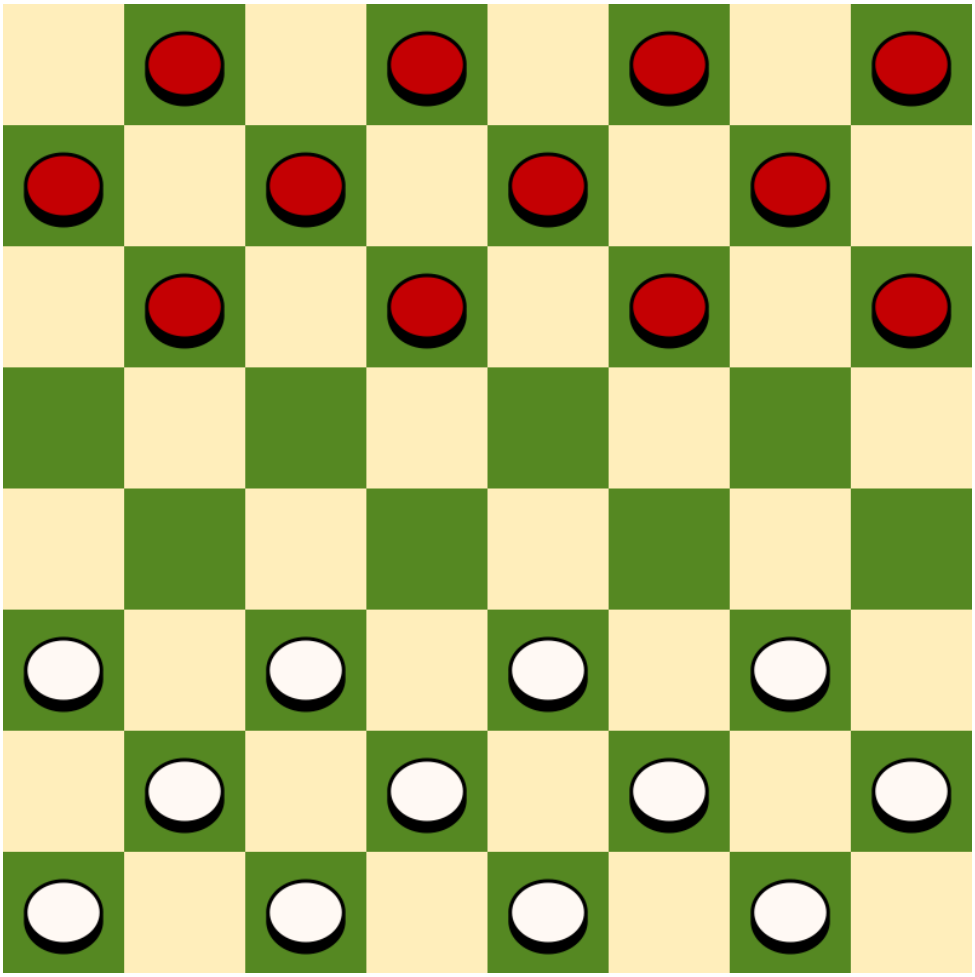

Figure 5.1: Checkers αρχική θέση

Οι κανόνες έχουν ώς εξής

- Ο λευκός ξεκινάει πρώτος και οι παίκτες εναλλάσσουν κινήσεις μέχρι κάποιος να ξεμείνει από κομμάτια είτε από αποδεκτές κινήσεις οπότε χάνει το παιχνίδι.
- Τα απλά κομμάτια (στρατιώτες) κινούνται διαγώνια κατά 1 προς τη κατεύθυνση του αντιπάλου.
- Τα απλά κομμάτια μπορούν να αιχμαλωτίσουν ένα αντίπαλο κομμάτι αν αυτό βρίσκεται στο διαγώνιο μπροστινό τους τετράγωνο και το ακριβώς επόμενο τετράγωνο προς την ίδια κατεύθυνση είναι κενό.Έτσι το κομμάτι που εκτελεί την αιχμαλώτιση περνάει πάνω από το αντίπαλο κομμάτι και καταλήγει στη κατά σειρά επόμενη θέση.Το κομμάτι που αιχμαλωτίζεται απομακρύνεται από το παιχνίδι.
- Στην εκδοχή παιχνιδιού που εξετάζουμε κάθε αιχμαλώτιση εφόσον υπάρχει διαθέσιμη είναι υποχρεωτική και σε περίπτωση πολλών πιθανών αιχμαλωτήσεων γίνεται επιλογή από τον παίκτη.
- Αν μετά από μια αιχμαλώτιση υπάρχει διαθέσιμη επόμενη αιχμαλώτιση από το ίδιο κομμάτι τότε η σειρά του παίκτη συνεχίζεται. Σε περίπτωση πολλαπλών αιχμαλωτίσεων ο παίκτης διαλέγει το μονοπάτι με τις περισσότερες αιχμαλωτίσεις.
- Όταν ένα κομμάτι φτάσει στην απέναντι τερματική γραμμή τότε αναβαθμίζεται σε βασιλιάς και έχει τις ίδιες δυνατότητες με τον απλό στρατιώτη ενώ επιπρόσθετα οι διαγώνιες κινήσεις του (και οι αιχμαλωτίσεις) μπορούν να γίνουν και προς τα πίσω.

Το παιχνίδι αυτό εντάσσεται στην κατηγορία των perfect information zero sum games. Αυτό σημαίνει ότι σε κάθε φάση του παιχνιδιού δεν μας αποκρύπτεται καμία πληροφορία προκειμένου να αποφασίσουμε ποια θα είναι η επόμενη μας κίνηση αντιθέτως για παράδειγμα με ότι συμβαίνει σε ένα παιχνίδι poker. Η έννοια του μηδενικού αθροίσματος υποδεικνύει ότι το πλεονέκτημα ενός παίκτη αντιστοιχεί σε μειονέκτημα ίσου μέτρου για τον αντίπαλο παίκτη και αφορά μια ευρεία κατηγορία παιχνιδιών στρατηγικής όπως το σκάκι το Go το shogi κτλ.

5.2 Αλγόριθμοι Αναζήτησης Δέντρου

Οι αλγόριθμοι αναζήτησης δέντρου αποτελούν βασικό κομμάτι κάθε μηχανής σχεδιασμένης για ένα ανταγωνιστικό παιχνίδι μεταξύ δυο παικτών και βοηθούν στην επιλογή της κίνησης από μια δεδομένη θέση. Η κλασσική αναπαράσταση ενός παιχνιδιού στρατηγικής γίνεται με τη βοήθεια ενός δέντρου.Κάθε κόμβος στο δέντρο αναπαριστά μια θέση στο παιχνίδι (state) ενω οι ακμές από έναν κόμβο δηλώνουν τις διαθέσιμες κινήσεις από το αντίστοιχο state και τα παιδιά στον οποίο καταλήγουν αποτελούν τις θέσεις που προκύπτουν αν παιχτεί η κίνηση της ακμής στη θέση του κόμβου γονέα. Δεδομένης μιας συνάρτησης V κάθε κόμβος λαμβάνει μια τιμή ο οποίος δηλώνει την αξία της κατάστασης στην οποία βρισκόμαστε.Κατα σύμβαση θεωρούμε ότι θετικές τιμές σημαίνει πλεονεκτική κατάσταση για το λευκό ενώ αρνητικές πλεονεκτική κατάσταση για τον μαύρο.

5.2.1 Ο αλγόριθμος minimax

Ο αλγόριθμος minimax αποτελεί έναν αναδρομικό αλγόριθμο ο οποίος ελαχιστοποιεί την απώλεια (η μεγιστοποιεί το ελάχιστο κέρδος) στο χειρότερο δυνατό σενάριο έκβασης του παιχνιδιού για τον παίκτη ο οποίος έχει τη σειρά του. Σε παιχνίδια μεταξύ δυο παικτών η λύση που δίνει είναι η ίδια που προκύπτει από την ισοσταθμία του Nash. Ο αλγόριθμος δέχεται ως είσοδο την θέση του παιχνιδιού και κατασκευάζει το δέντρο του παιχνιδιού προσθέτοντας σε κάθε κόμβο τα παιδιά του από όλες τις αποδεκτές κινήσεις.Καθώς πηγαίνουμε από το ένα επίπεδο στο άλλο αλλάζει και η σειρά του κάθε παίκτη. Βασική λογική αυτού του αλγορίθμου είναι ότι οι δύο παίκτες κινούνται προς αντίθετες κατευθύνσεις δηλαδή ο ένας στοχεύει σε μέγιστη θετική αμοιβή ενώ ο άλλος σε μέγιστη κατ απόλυτη τιμή αρνητική αμοιβή.Αν κατασκευάσουμε εξαντλητικά το δέντρο που προκύπτει με ρίζα την τρέχουσα θέση του παιχνιδιού και οι τερματικοί κόμβοι λαμβάνουν τιμές +inf για νίκη λευκού , -inf για νίκη μαύρου και 0 για ισοπαλία τότε βλέποντας όλα τα μονοπάτια μέχρι το τέλος του παιχνιδιού μπορούμε να διαλέξουμε αυτό που θα μας εξασφαλίσει τη νίκη υποθέτοντας ότι ο αντίπαλος ακολουθεί βέλτιστη στρατηγική με τον ίδιο αντίστοιχο στόχο.(Στο παιχνίδι checkers έχει αποδειχθεί ότι η υιοθέτηση βέλτιστης πολιτικής και από τους δύο παίκτες οδηγεί σε ισοπαλία).

Ωστόσο για το συγκεκριμένο παιχνίδι με συνολικές θέσεις της τάξης του 10^{20} η κατασκευή ολόκληρου του δέντρου είναι αδύνατη. Περιοριζόμαστε λοιπόν δεδομένης μιας θέσης της κατασκευής του υπόδεντρου μέχρι ένα συγκεκριμένο βάθος. Τα φύλλα του υπόδεντρου μπορεί να είναι τερματικοί κόμβοι για το παιχνίδι για τους οποίους γνωρίζουμε την αξία τους ή μπορεί να είναι κάποιοι ενδιάμεσοι κόμβοι. Η ανάθεση τιμής σε αυτούς τους κόμβους γίνεται μέσω κάποιας ευριστικής ή ενός νευρωνικού το οποίο προσεγγίζει την συνάρτηση $V^*(s)$ όπως θα αναφέρουμε στην επόμενη παράγραφο.

Παρακάτω παρουσιάζουμε σε ψευδοκώδικα τον αλγόριθμο.

Algorithm 10: Ο αλγόριθμος minimax

```
Function minimax(node, depth,maximizer):
    if depth=0 or node is terminal then
        return Heuristic(node);
    end
    if maximizer then
        value ← −∞;
        for each child of node do
            value ← max(value, minimax(child, depth − 1, FALSE));
        end
        return value;
    end
    else
        value ← +∞;
        for each child of node do
            value ← min(value, minimax(child, depth − 1, TRUE));
        end
        return value;
    end
```

Ο αλγόριθμος καλείται με μία ρίζα, το βάθος που θέλουμε να εξετάσουμε την θέση και τον παίκτη ο οποίος παίζει (true για τον maximizer δηλαδή τον λευκό ή False αλλιώς). Καθώς αυξάνουμε το βάθος προκειμένου να μπορέσουμε να επιλέξουμε την πιο συμφέρουσα κίνηση σε ένα παιχνίδι όπως το checkers με branching factor 7 δηλαδή κατά μέσο όρο 7 κινήσεις ανά διαθέσιμη θέση ο αλγόριθμος γίνεται μη αποδοτικός. Μια πολύ σημαντική βελτίωση μπορεί να επιτευχθεί αν σταματάμε να εξετάζουμε μια κίνηση αν έχει βρεθεί έστω ένα ενδεχόμενο που αποδεικνύει ότι η κίνηση αυτή είναι χειρότερη από μια προηγούμενη κίνηση του ίδιου επιπέδου που εξετάστηκε προηγουμένως.Για να το πετύχουμε αυτό χρησιμοποιούμε δύο παραμέτρους a,b με τις οποίες κρατάμε το ελάχιστο σκορ που έχει εξασφαλίσει ο maximizer και το μέγιστο σκορ που έχει εξασφαλίσει ο minimizer. Έτσι καθώς διασχίζουμε το δέντρο αν ισχύει η συνθήκη $b < a$ δεν χρειάζεται να εξετάσουμε περαιτέρω παιδιά του κόμβου καθώς δεν πρόκειται (με βέλτιστη στρατηγική) να υπάρξει θέση από αυτό το υπόδεντρο. Παρουσιάζουμε παρακάτω τον αλγόριθμο alpha beta του οποίου η αρχική κλήση γίνεται με μια ρίζα, το βάθος που θέλουμε να ψάξουμε και τις τιμές -inf και +inf για τα α,β αντίστοιχα.

Algorithm 11: Ο αλγόριθμος alphabeta

```
Function alphabeta(node, depth,a,b,maximizer):
    if depth=0 or node is terminal then
        return Heuristic(node);
    end
    if maximizer then
        value ← −∞;
        for each child of node do
            value ← max(value, alphabeta(child, depth − 1, a, b, FALSE));
            a ← max(a, value);
            if a ≥ b then
                Break;
            end
        end
        return value;
    end
    else
        value ← +∞;
        for each child of node do
            value ← min(value, alphabeta(child, depth − 1, a, b, TRUE));
            b ← min(b, value);
            if a ≥ b then
                Break;
            end
        end
        return value;
    end
```

Παρακάτω φαίνεται ένα σχηματικό παράδειγμα που δείχνει πώς ανεβαίνουν οι τιμές από τα φύλλα προς την κορυφή ώστε να διαλέξει ο παίκτης 1(maximizer) που είναι στη ρίζα το πιο υποσχόμενο παιδί του.

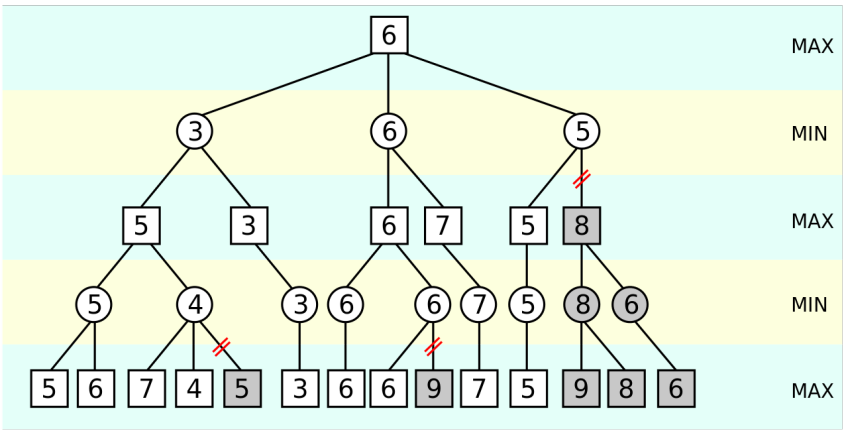

Figure 5.2: Minimax with alpha beta pruning

5.2.2 Monte Carlo Tree search (MCTS)

Σε σύγκριση με τον ντετερμινιστικό αλγόριθμο minimax η διάσχιση δέντρου Monte carlo υπολογίζει την πιο υποσχόμενη κίνηση με εξ ολοκλήρου διαφορετική λογική και υπό τη σκοπιά της ενισχυτικής μάθησης, χωρίς να κατασκευάζεται όλο το δέντρο από τον αρχικό κόμβο. Όπως υποδεικνύει και η ονομασία κατά την εκτέλεση του αλγορίθμου για μια δεδομένη θέση πραγματοποιούνται προσομοιώσεις και η τελική κίνηση προκύπτει από τα στατιστικά που προέκυψαν από αυτές [9].

Θα ξεκινήσουμε διευκρινίζοντας κάποιες βασικές έννοιες για τον αλγόριθμο:

Η προσομοίωση (simulation) αποτελεί μια αλληλουχία από κινήσεις μεταξύ δύο παικτών η οποία ξεκινάει από ένα κόμβο n μέχρι να τελειώσει το παιχνίδι και να αποκτήσουμε το σήμα αμοιβής (reward) ανάλογα με την έκβαση του παιχνιδιού. Κατ αυτό το τρόπο μπορεί να ανατεθεί μια τιμή προσεγγιστικής αξίας στον αρχικό κόμβο από τον οποίο ξεκίνησε η προσομοίωση.Οι κινήσεις κατά τη διάρκεια της προσομοίωσης επιλέγονται από τη πολιτική προσομοίωσης:

$$\pi_{rollout} : s_i \rightarrow a_i$$

όπου κάθε κατάσταση s_i αντιστοιχίζεται στη δράση a_i. Στην πράξη θέλουμε αυτή η πολιτική να είναι γρήγορη ώστε να μπορούν να εκτελούνται όσο δυνατόν περισσότερες προσομοιώσεις και ως default χρησιμοποιούμε την ομοιόμορφη πολιτική η οποία επιλέγει με ίση πιθανότητα κάθε μία από τις αποδεκτές κινήσεις.

Visited κόμβος: Ένας κόμβος θεωρούμε ότι τον έχουμε επισκεφθεί αν έχει ξεκινήσει τουλάχιστον μια προσομοίωση από αυτόν.Δηλαδή έχει αξιολογηθεί και του έχει ανατεθεί μια τιμή όπως προέκυψε από το σήμα αμοιβής.

Πλήρης επεκτεταμένος κόμβος (fully expanded): Έναν κόμβο τον θεωρούμε πλήρη επεκτεταμένο αν όλα του τα παιδιά είναι visited. Αλλιώς λέμε ότι επιδέχεται περαιτέρω επέκταση.Στη αρχή λοιπόν που ξεκινάμε με έναν κόμβο ρίζα δεν έχουμε επισκεφθεί κανένα από τα παιδιά του οπότε διαλέγουμε έναν και ξεκινάει μια προσομοίωση με αφετηρία αυτό το κόμβο παιδί.Αξίζει να επισημάνουμε ότι κατά τη διάρκεια μιας προσομοίωσης οι κόμβοι που διατρέχονται μέχρι να φτάσουμε σε τερματική κατάσταση δεν θεωρούμε ότι τους έχουμε επισκεφθεί και δεν εμφανίζονται στο δέντρο που δημιουργούμε σε κάθε βήμα. Από πλευράς υλοποίησης μπορούμε να πούμε ότι οι προσομοιώσεις ξεκινάνε κάθε φορά από ένα προσωρινό αντίγραφο του κόμβου.

Οπισθοδιάδοση (Backpropagation) Είναι η διαδικασία μέσω της οποίας κάθε φορά ένας που κόμβος αξιολογείται μέσω προσομοίωσης το σήμα αμοιβής επιστρέφει ακολουθώντας το μονοπάτι προς την ρίζα του δέντρου και ενημερώνει όλους τους ενδιάμεσους κόμβους που συναντάει.

Στον κάθε κόμβο n αποθηκεύουμε τα εξής στατιστικά:

- $Q(n)$: Αποτελεί τον συνολικό άθροισμα από τις αμοιβές ως αποτέλεσμα προσομοιώσεων που είτε ξεκίνησαν από αυτό το κόμβο ή προστέθηκαν ως αποτέλεσμα ενημέρωσης από το στάδιο της οπισθοδιάδοσης.

- $N(n)$: Αποτελεί τον αριθμό των επισκέψεων σε αυτό το κόμβο δηλαδή πόσες φορές έχει περάσει μια τιμή αμοιβής από αυτόν με αποτέλεσμα να ενημερώσει το συνολικό άθροισμα

Αυτές οι δύο τιμές μένουν αποθηκευμένες στον κόμβο και αποτελούν μια ένδειξη κατά πόσο έχει εξερευνηθεί και αξιοποιηθεί κατά τη διάσχιση του δέντρου (exploration-exploitation).Με άλλα λόγια το Q μας δείχνει πόσο υποσχόμενος είναι ένας κόμβος υπό την έννοια ότι υψηλή τιμή είναι ένδειξη επιθυμητής κατάστασης ενώ το N μας υποδεικνύει ότι καταστάσεις που δεν έχουμε επισκεφθεί πολύ, μπορεί να έχουν και αυτές ενδιαφέρον και αξίζει να εξεταστούν.

Μένει λοιπόν να εξετάσουμε πως δημιουργείται το δέντρο και με ποιόν τρόπο διασχίζεται.Όταν βρισκόμαστε σε ένα κόμβο n προχωράμε στο παιδί n_i που μεγιστοποιεί την παρακάτω ποσότητα:

$$UCT(n, n_i) = \frac{Q(n_i)}{N(n_i)} + c\sqrt{\frac{log(N(n))}{N(n_i)}}$$

Όπου c μια υπερπαράμετρος που ελέγχει την προτεραιότητα που δίνεται υπέρ της εξερεύνησης.Μια σημαντική λεπτομέρεια είναι ότι ο όρος Q_i στα ανταγωνιστικά παιχνίδια υπολογίζεται πάντα από την πλευρά του παίκτη του οποίου είναι η σειρά να παίξει στον κόμβο i οπότε καθώς εναλλασσόμαστε στα επίπεδα του δέντρου ο όρος αυτός αντιστρέφεται. Μια επανάληψη του αλγορίθμου αποτελείται απο τα 4 εξής βήματα:

- **selection** Ξεκινάμε από τη ρίζα και διασχίζουμε το δέντρο επιλέγοντας κάθε φορά τον κόμβο που μεγιστοποιεί την ποσότητα UCT μέχρι να φτάσουμε σε φύλλο δηλαδή κόμβο που δεν έχει επεκταθεί.

- **expansion** Αν το φύλλο στο οποίο φτάσαμε δεν είναι τερματικός κόμβος σύμφωνα με τους κανόνες του παιχνιδιού τότε τον επεκτείνουμε προσθέτοντας τα παιδιά του.

- **simulation** Διαλέγουμε τυχαία ένα από τα παιδιά του (όλα έχουν άπειρο UCT με N=0) και εκτελούμε ένα simulation από αυτό.

- **backpropagation** Η τιμή αμοιβής που λάβαμε από το simulation ενημερώνει όλους τους κόμβους στο μονοπάτι από το παιδί που ξεκίνησε μέχρι τη ρίζα.

Ο αλγόριθμος δεν έχει κάποια συνθήκη τερματισμού και γίνεται πιο ακριβής όσο αυξάνεται ο αριθμός των επαναλήψεων ο οποίος εξαρτάται από τους χρονικούς περιορισμούς που λαμβάνουμε υπόψιν. Μετά τον προκαθορισμένο αριθμό επαναλήψεων διαλέγουμε από την ρίζα το παιδί με το υψηλότερο αριθμό επισκέψεων ως το πιο υποσχόμενο.

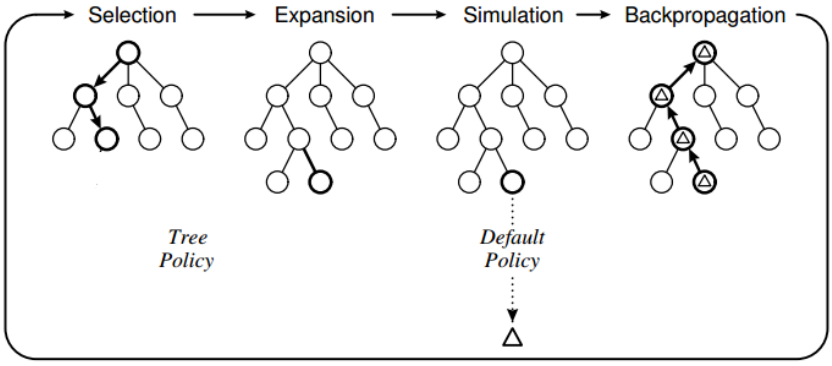

Figure 5.3: Monte Carlo Tree Search

5.3 Επιβλεπόμενη μάθηση

Στο κομμάτι αυτό προσπαθήσαμε να προσεγγίσουμε την συνάρτηση αξίας V(s) η οποία μας δείχνει πόσο επιθυμητό είναι να βρισκόμαστε στην κατάσταση s καθώς και μια πολιτική $\pi(a|s)$ με τη βοήθεια νευρωνικών δικτύων.

5.3.1 Κωδικοποίηση Κατάστασης και πολιτικής

Σύμφωνα με τους επίσημους κανονισμούς του παιχνιδιού τα τετράγωνα της σκακιέρας ακολουθούν την παρακάτω αρίθμηση με τα λευκά στο κάτω μέρος: Επομένως μια κατάσταση

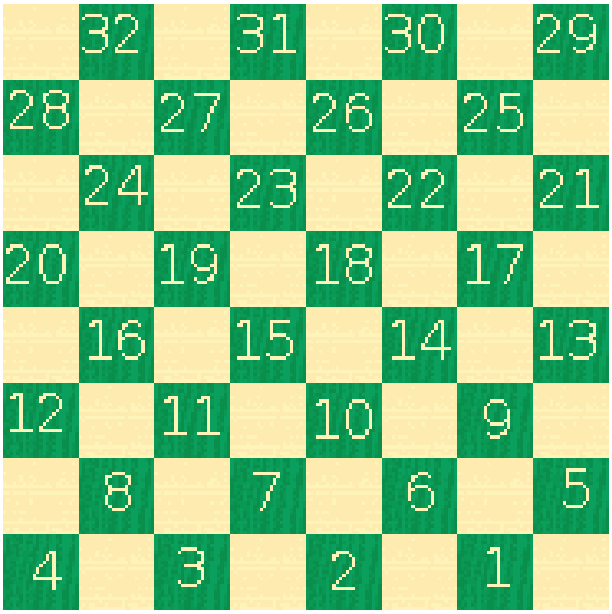

Figure 5.4: Board Notation

μπορεί να αντιμετωπιστεί ως ένα διάνυσμα 32 διαστάσεων ως εξής:

$$state = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{32} \end{pmatrix} \tag{5.1}$$

με¨

$$x_i = \begin{cases} 0 & \text{i-οστο τετράγωνο άδειο} \\ 1 & \text{i-οστο τετράγωνο από λευκό κομμάτι} \\ -1 & \text{i-οστο τετράγωνο από μαύρο κομμάτι} \\ 3 & \text{i-οστο τετράγωνο από λευκό βασιλιά} \\ -3 & \text{i-οστο τετράγωνο από μαύρο βασιλιά} \end{cases} \tag{5.2}$$

Αξίζει να σημειώσουμε ωστόσο το εξής: Για να απλοποιήσουμε την κωδικοποίηση της κατάστασης καθώς και να διευκολύνουμε την εκπαίδευση νευρωνικών δικτύων στο state δεν εμφανίζεται η πληροφορία για την σειρά του κάθε παίκτη. Δεδομένης μιας θέσης ο παίκτης ο οποίος παίζει έχει την προοπτική του λευκού έτσι μετά την κωδικοποίηση του state έχουμε ένα rotation της θέσης και πολλαπλασιάζουμε τα στοιχεία του διανύσματος κατάστασης με

-1. Επιπλέον σε αρχιτεκτονικές δικτύων που έχουν συνελικτικά επίπεδα στην αρχή, το 1x32 διάνυσμα κατάστασης μετασχηματίζεται σε 8x4 εικόνα με ένα κανάλι.

Τώρα όσον αφορά τις κινήσεις ο επίσημος συμβολισμός που ακολουθείται είναι ο εξής για ένα κομμάτι:

$$move = [start, end] \tag{5.3}$$

οπού start η αρχική θέση του κομματιού και end η τελική του θέση σύμφωνα με τον παραπάνω συμβολισμό της 5.4. Σε περίπτωση που έχουμε πολλαπλές αιχμαλωτίσεις αυτές αντιμετωπίζονται ως διακριτές κινήσεις οι οποίες πραγματοποιούνται διαδοχικά χωρίς να αλλάζει η σειρά του παίκτη.Να αναφέρουμε ότι ομοίως με το rotation που πραγματοποιούμε σε περίπτωση κίνησης του μαύρου έτσι και για τις κινήσεις για να συνεχίζουν να συμβαδίζουν με την αρίθμηση της σκακιέρας μετά την κωδικοποίηση κάνουμε την εξής τροποποίηση:

$$move_{black} = [33 - start, 33 - end]$$

Για την αναπαράσταση της πολιτικής σύμφωνα με την οποία μπορούμε να αναπαραστήσουμε κάθε κίνηση εργαστήκαμε ως εξής:

Αρχικά παρατηρούμε ότι σε κάθε θέση κάθε βασιλιάς που μπορεί να βρεθεί εκεί έχει 4 δυνατές κινήσεις: Πάνω δεξιά, πάνω αριστερά , κάτω δεξιά, κάτω αριστερά. Οι αιχμαλωτίσεις αντιμετωπίζονται απλά ως κινήσεις προς την αντίστοιχη κατεύθυνση.Έτσι η πολιτική μπορεί να αντιμετωπιστεί ως ένα ένα σταθερό διάνυσμα 32x4

$$\pi(a|s) = \begin{pmatrix} \pi_{1,1} & \dots & \pi_{1,4} \\ \vdots & \ddots & \vdots \\ \pi_{32,1} & \dots & \pi_{32,4} \end{pmatrix} \tag{5.4}$$

όπου: $\pi_{i,j}$= πιθανότητα το κομμάτι στη θέση i να κινηθεί προς την κατεύθυνση j. Προφανώς η κωδικοποίηση αυτή εμπεριέχει και συνδυασμούς θέσεων-κινήσεων όπως τα άκρα της σκακιέρας και οι γωνίες όπου δεν είναι δυνατές οι κινήσεις προς όλες τις κατευθύνσεις (illeagal moves) και το δίκτυο πρέπει να μπορεί να το μάθει αυτό.Ο παραπάνω πίνακας μετάβασης κατάστασης 32x4 μετασχηματίζεται σε ένα διάνυσμα

$$p = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_{128} \end{pmatrix}$$

το οποίο αποτελεί την έξοδο του δικτύου.Για να εξάγουμε κινήσεις από αυτό ακολουθούμε αντίστροφη πορεία μετασχηματίζοντας το σε 32x4 και είτε διαλέγουμε την θέση με την μεγαλύτερη πιθανότητα είτε δειγματολειπτούμε από την κατανομή πιθανοτήτων. Σε κάθε περίπτωση οι συντεταγμένες i,j μας επιτρέπουν να κάνουμε αποκωδικοποίηση και να επιστρέψουμε στη μορφή move=[start,end]. Να σημειωθεί ότι η αποκωδικοποίηση θα επιστρέψει 2 κινήσεις: την απλή κίνηση προς την j κατεύθυνση και την αιχμαλώτιση αντίπαλου κομματιού προς την j κατεύθυνση. Από αυτές τις δύο εκτελείται πάντα η νόμιμη.

5.3.2 Το Dataset

Για τους σκοπούς αυτής της εργασίας το dataset που χρησιμοποιήθηκε αποτελούνταν από πάνω από 20.000 παιχνίδια μεταξύ "ειδικών" στο παιχνίδι, σε τουρνουά που πραγματοποιήθηκε τον 19ο αιώνα και καταγράφηκαν με το χέρι με ελάχιστα λάθη. Το αρχείο έχει την παρακάτω μορφή:

```
[Event "Manchester 1841"]
[Date "1841-??-??"]
[Black "Moorhead, W."]
[White "Wyllie, J."]
[Site "Manchester"]
[Result "0-1"]
1. 11-15 24-20 2. 8-11 28-24 3. 9-13 22-18 4. 15x22 25x18 5. 4-8 26-22 6. 10-14
18x9 7. 5x14 22-18 8. 1-5 18x9 9. 5x14 29-25 10. 11-15 24-19 11. 15x24 25-22 12.
24-28 22-18 13. 6-9 27-24 14. 8-11 24-19 15. 7-10 20-16 16. 11x20 18-15 17. 2-6
15-11 18. 12-16 19x12 19. 10-15 11-8 20. 15-18 21-17 21. 13x22 30-26 22. 18x27
26x17x10x1 0-1

[Event "Alwick 1842"]
[Date "1842-??-??"]
[Black "Hay, W."]
[White "Wyllie, J."]
[Site "Alwick"]
[Result "1/2-1/2"]
1. 11-15 24-20 2. 8-11 28-24 3. 4-8 23-19 4. 9-14 22-17 5. 15-18 26-23 6. 5-9
17-13 7. 1-5 32-28 8. 14-17 21x14 9. 10x17 23x14 10. 9x18 25-21 11. 6-10 21x14
```

Figure 5.5: Dataset format

Έχουμε δηλαδή σειριακά τα παιχνίδια υπό τη μορφή:

- Τουρνουά
- Ημερομηνία
- Ονόματα παικτών
- Τελικό αποτέλεσμα
- Κινήσεις που παίχτηκαν.

Προφανώς μας ενδιαφέρουν μόνο οι κινήσεις και το τελικό αποτέλεσμα.Κατασκευάζουμε τρεις παράλληλες λίστες state,policy,value έτσι ώστε σε κάθε θέση να αντιστοιχεί η κίνηση η οποία παίχτηκε καθώς και το τελικό αποτέλεσμα που προέκυψε από το παιχνίδι της θέσης.

Algorithm 12: Data Parser

```
Initialize empty Dataset: d;
for each game in database do
    Initialize Game() object g;
    for each move in game do
        s ← encode(g);
        move_encoded ← encode(move);
        r ← result of game;
        play move in g;
        store (s,move_encoded,r) in d;
    end
end
```

Αυτό που αξίζει να σημειωθεί είναι όπως αναφέραμε οι συμμετρίες που αξιοποιούμε όταν αποθηκεύουμε θέσεις στις οποίες παίζει ο μαύρος καθώς και την αντίστοιχη κίνηση. Κάθε state είναι ανηγμένο στο συμμετρικό του (canonical representation) δηλαδή κάθε παίκτης αντιλαμβάνεται τον εαυτό του ώς λευκό. Επίσης το αποτέλεσμα v αποθηκεύεται με μορφή αμοιβής με τιμές 1 για νίκη -1 για ήττα , 0 για ισοπαλία και πάντα υπό τη σκοπιά της σειράς του παίκτη στη θέση s. Η κίνηση κωδικοποιείται ως διάνυσμα 128 θέσεων από τον μετασχηματισμό του πίνακα 32x4 ως

$$move_{encoded} = unroll\left(\begin{pmatrix} \pi_{1,1} & \dots & \pi_{1,4} \\ \vdots & \ddots & \vdots \\ \pi_{32,1} & \dots & \pi_{32,4} \end{pmatrix}\right)$$

με:

$$\pi_{i,j} = \begin{cases} 1 & \text{κίνηση του κομματιού i προς τη j κατεύθυνση} \\ 0 & \text{αλλιώς} \end{cases}$$

Τέλος για την καλύτερη οργάνωση του dataset και την ευκολότερη εκπαίδευση των δικτύων εργαστήκαμε ως εξής:

- Όλες οι ίδιες θέσεις που προκύπτουν αντικαθιστώνται από μια s_{unique}
- Κάθε s_{unique} αντιστοιχεί σε ένα διάνυσμα πολιτικής $\boldsymbol{p}$ το οποίο σε κάθε θέση του p_i περιέχει το πλήθος εμφάνισης της κίνησης i σε όσες φορές εμφανίστηκε η s_{unique} κανονικοποιημένο ως προς το πλήθος εμφάνισης της θέσης αυτής.Έτσι προκύπτει μια κατανομή πιθανοτήτων πάνω στις δράσεις δεδομένης της κατάστασης.
- Ομοίως όλα τα αποτελέσματα που αντιστοιχούν στην κάθε s_{unique} προστίθενται και διαιρούνται με το πόσες φορές εμφανίστηκε η θέση.Κατ αυτόν τον τρόπο προκύπτει η αξία v της θέσης με $v \in [-1, 1]$ ως συνεχής τιμή και εκτίμηση του πλεονεκτήματος της θέσης.

Στο εξής και για την εκπαίδευση των δικτύων θα χρησιμοποιούμε δείγματα της μορφής (s,p,v).

5.3.3 Αρχιτεκτονικές δικτύων

5.3.3.1 Απλά συνελικτικά δίκτυα

Παρουσιάζουμε εν συντομία δύο απλές δομές δικτύων που χρησιμοποιήθηκαν σε πρώτο στάδιο.

Convolutional Policy Net

- Συνελικτικό επίπεδο : Κανάλια εισόδου 1 , κανάλια εξόδου 128 , kernel 2x2 , stride 1, padding 1.
- BatchNormalization επίπεδο.
- ReLu συνάρτηση ενεργοποίησης
- Συνελικτικό επίπεδο : Κανάλια εισόδου 128 , κανάλια εξόδου 1 , kernel 3χ3 , stride 1, padding 1.
- BatchNormalization επίπεδο.
- ReLu συνάρτηση ενεργοποίησης
- Γραμμικό επίπεδο: 45 νευρώνες εισόδου 512 νευρώνες εξόδου.
- BatchNormalization επίπεδο.
- ReLu συνάρτηση ενεργοποίησης
- Γραμμικό επίπεδο: 512 νευρώνες εισόδου, 128 νευρώνες εξόδου.
- BatchNormalization επίπεδο.
- Softmax συνάρτηση

Convolutional Value Net

- Συνελικτικό επίπεδο : Κανάλια εισόδου 1 , κανάλια εξόδου 128 , kernel 2x2 , stride 1, padding 1.
- BatchNormalization επίπεδο.
- ReLu συνάρτηση ενεργοποίησης
- Συνελικτικό επίπεδο : Κανάλια εισόδου 128 , κανάλια εξόδου 1 , kernel 3χ3 , stride 1, padding 1.
- BatchNormalization επίπεδο.
- ReLu συνάρτηση ενεργοποίησης
- Μετασχηματισμος σε διάνυσμα 45 διαστάσεων.
- Γραμμικό επίπεδο: 45 νευρώνες εισόδου 1 νευρώνας εξόδου.
- tanh συνάρτηση ενεργοποίησης.

5.3.3.2 Αρχιτεκτονική τύπου Resnet

Το πρόβλημα που προσπαθεί να αντιμετωπίσει αυτή η αρχιτεκτονική [10] είναι η εξαφάνιση κλίσης (vanishing gradient problem). Καθώς αυξάνουμε τον αριθμό των επιπέδων σε ένα νευρωνικό δίκτυο προκειμένου να πετύχουμε καλύτερη απόδοση φτάνουμε σε ένα σημείο κορεσμού και ακολουθεί ύστερα απότομη πτώση. Συγκεκριμένα η εξαφάνιση κλίσης αποτελεί ένα πρόβλημα κατά το οποίο ένα βαθύ δίκτυο δεν μπορεί να μεταδώσει προς τα πίσω χρήσιμη πληροφορία κατά την ενημέρωση των βαρών. Η κλίση γίνεται ολοένα και μικρότερη και έτσι τα βάρη δεν ενημερώνονται δυσκολεύοντας την εκπαίδευση του δικτύου. Η αρχιτεκτονική αυτή είχε μεγάλη επιτυχία την κατηγοριοποίηση εικόνων οπότε επιχειρήσαμε να την εφαρμόσουμε αντιμετωπίζοντας το state του παιχνιδιού ως εικόνα.Δομική μονάδα αυτής της αρχιτεκτονικής είναι η δομή υπολοίπου (residual) που υλοποιείται με μια ταυτοτική σύνδεση όπως βλέπουμε παρακάτω.

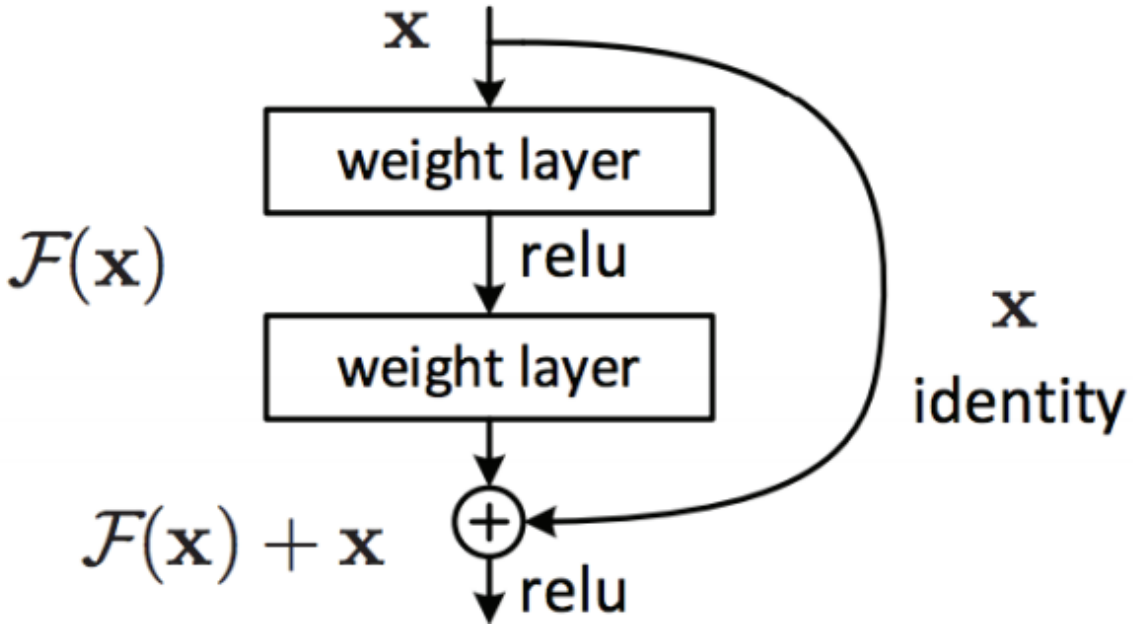

Figure 5.6: Residual learning: a building block.

Βασική λογική είναι ότι επειδή τα δίκτυα δυσκολεύονται να αναγνωρίσουν την ταυτότητα, αν θεωρήσουμε ότι θέλουμε να μάθουμε την απεικόνιση Η αντί αυτής μοντελοποιούμε με το δίκτυο βαρών την $F(x) = H(x) - x$ και η έξοδος λαμβάνεται όπως βλέπουμε παραπάνω.

Για την υλοποίηση χρησιμοποιήσαμε τα παρακάτω δομικά κομμάτια ως κλάσεις:

Convolutional Block

- Συνελικτικό επίπεδο: κανάλι εισόδου 1, κανάλια εξόδου 128,kernel 3x3, stride 1, padding 1.
- BatchNormalization2D επίπεδο.
- ReLu συνάρτηση ενεργοποίησης.

Residual Block

- Είσοδος 128 κανάλια
- Συνελικτικό επίπεδο: κανάλια εισόδου 128, κανάλια εξόδου 128, kernel 3x3 ,stride 1, padding 1
- BatchNormalization2D επίπεδο.
- Συνελικτικό επίπεδο: κανάλια εισόδου 128, κανάλια εξόδου 128, kernel 3x3 ,stride 1, padding 1
- BatchNormalization2D επίπεδο.
- Skip connection: + Είσοδος
- ReLu συνάρτηση ενεργοποίησης.

Output Block1 (έξοδος πολιτική p)

- Συνελικτικό επίπεδο: κανάλια εισόδου 128, κανάλια εξόδου 32, kernel 1x1 , stride 1 padding 1.
- BatchNormalization2D επίπεδο.
- ReLu συνάρτηση ενεργοποίησης.
- Μετασχηματισμός σε διάνυσμα $32 \cdot 8 \cdot 4$
- Γραμμικό επίπεδο $32 \cdot 8 \cdot 4$ νευρώνες εισόδου 128 νευρώνες εξόδου.
- Softmax συνάρτηση για αναγωγή σε κατανομή πιθανοτήτων.

Output Block2 (έξοδος συνάρτηση αξίας V)

- Συνελικτικό επίπεδο: κανάλια εισόδου 128, κανάλια εξόδου 1, kernel 1x1 , stride 1 padding 1.
- BatchNormalization2D επίπεδο.
- ReLu συνάρτηση ενεργοποίησης.
- Μετασχηματισμός σε διάνυσμα 32 διαστάσεων
- Γραμμικό επίπεδο 32 νευρώνες εισόδου 32 νευρώνες εξόδου.
- ReLu συνάρτηση ενεργοποίησης.
- Γραμμικό επίπεδο 32 νευρώνες εισόδου 1 νευρώνας εξόδου.
- tanh συνάρτηση ενεργοποίησης για έξοδο στο διάστημα [-1,1]

5.3.4 Εκπαίδευση των δικτύων

Στόχος μας είναι η εκμάθηση μιας πολιτικής $\pi(a|s)$ και μιας συνάρτησης αξίας $V(s)$ όπως προκύπτει από γνώστες του παιχνιδιού και με βάση το dataset που δημιουργήσαμε. Υπενθυμίζουμε τα δείγματα έχουνε την μορφή (s_t,π_t,z_t) με s το state, π η κατανομή πιθανοτήτων πάνω στις δράσης της συγκεκριμένης θέσης και z η αξία της θέσης με $z \in [-1, 1]$

Για την εκπαίδευση των δικτύων χρησιμοποιούμε τις εξής συναρτήσεις κόστους με τις εξόδους από τα δίκτυα να είναι παραμετροποιημένες ως προς τα βάρη ϑ.

Value Network

$$L_1 = \sum_t (v_\theta(s_t) - z_t)^2 \tag{5.5}$$

Policy Network

$$L_2 = -\sum_t \overrightarrow{\pi_t} \cdot log(\overrightarrow{p_\theta}(s_t)) \tag{5.6}$$

Double Head Network

$$L_3 = \sum_t (v_\theta(s_t) - z_t)^2 - \sum_t \overrightarrow{\pi_t} \cdot log(\overrightarrow{p_\theta}(s_t)) \tag{5.7}$$

Το dataset χωρίστηκε με την εξής αναλογία:

- Training set 90%
- Validation set 5%
- Test set 5%

Ο βρόγχος εκπαίδευσης υλοποιήθηκε σε pytorch με optimizer τον Adam και ο ρυθμός μάθησης να επιλέγεται μετά από δοκιμές στο 0.001.

Early stopping Κατα τη διάρκεια της εκπαίδευσης παρακολουθούσαμε την μεταβολή για το Loss στο Validation set. Το validation set δεν χρησιμοποιείται για την ενημέρωση των βαρών και το δίκτυο δεν μαθαίνει από αυτά τα δείγματα αλλά παρεμβάλλονται μετά από κάθε πέρασμα από όλο το training set και αποτελούν ένα κριτήριο ελέγχου βοηθώντας μας να καταλάβουμε αν αυξάνεται η μειώνεται η ικανότητα του δικτύου να γενικεύει σε states που δεν έχει συναντήσει.Ξεκινάμε με validation loss = $-\infty$ και κάθε φορά που μειώνεται κρατάμε την νέα ελάχιστη τιμή και αποθηκεύουμε την τελευταία έκδοση του δικτύου. Η κλάση early stopping έχει μια παράμετρο patience η οποία εκφράζει τον αριθμό των εποχών για τις οποίες συνεχίζουμε την εκπαίδευση όταν το validation loss αρχίσει να αυξάνεται. Ύστερα τερματίζουμε την εκπαίδευση.Το tradeoff είναι ότι χάνουμε ένα κομμάτι από το training dataset άλλα δεν χρειάζεται να ασχοληθούμε άμεσα με τον συνολικό αριθμών των εποχών ώστε να μεγιστοποιήσουμε την ακρίβεια ενώ αποφεύγουμε το overfitting.Παρακάτω βλέπουμε τις καμπύλες μάθησης για τις αρχιτεκτονικές δικτύων που χρησιμοποιήθηκαν.

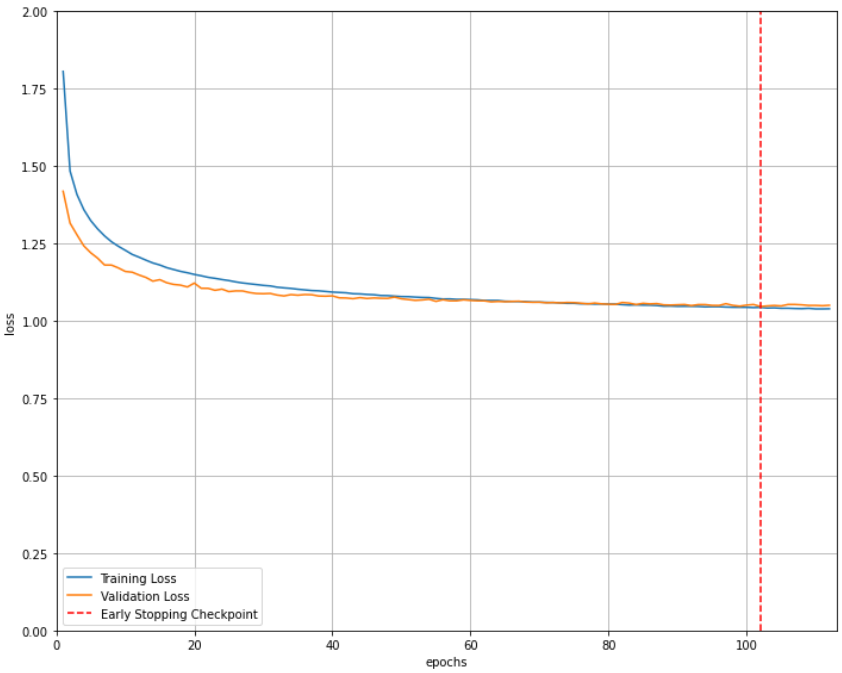

Figure 5.7: Simple Convolutional Policy Model Loss

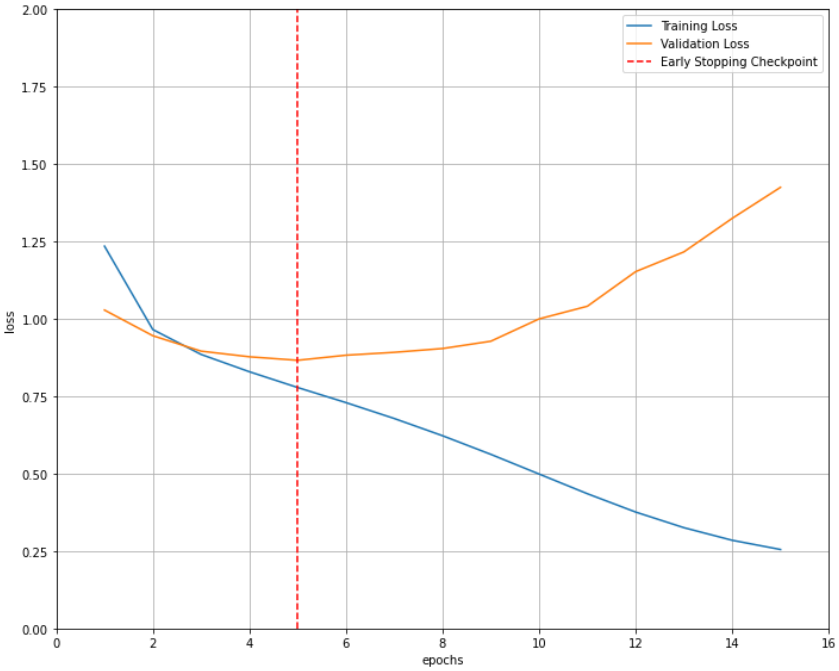

Figure 5.8: 20 residual blocks Policy Model Loss

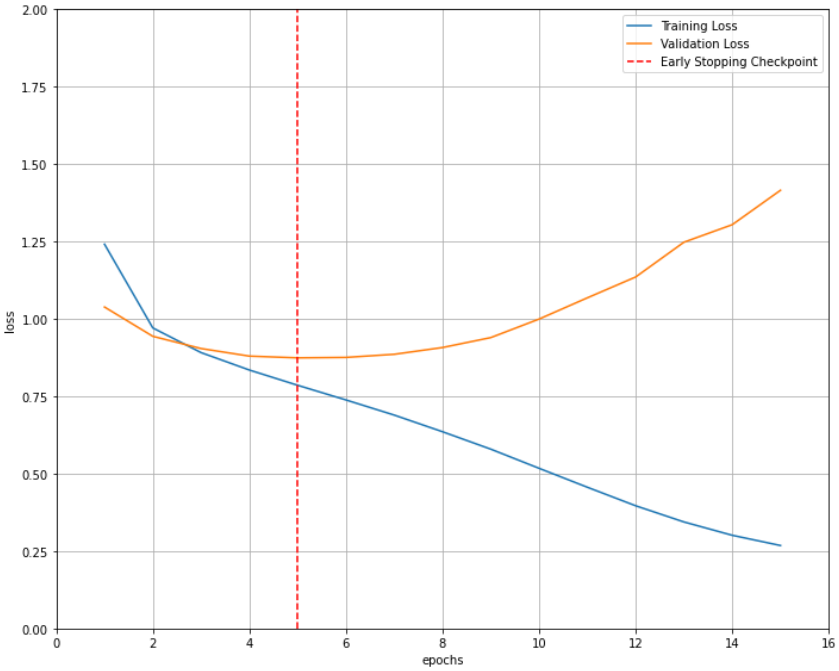

Figure 5.9: 10 residual blocks Policy Model Loss

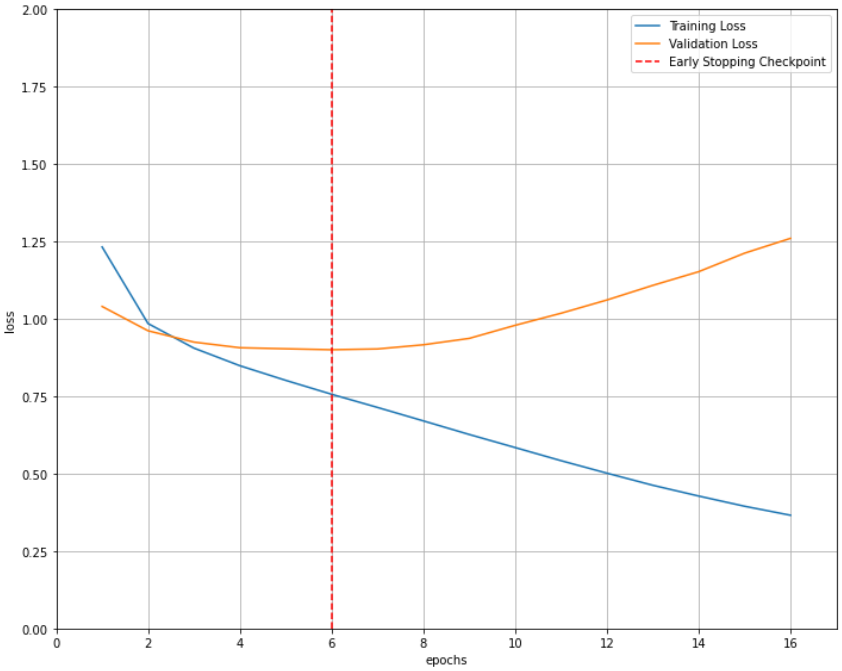

Figure 5.10: 5 residual blocks Policy Model Loss

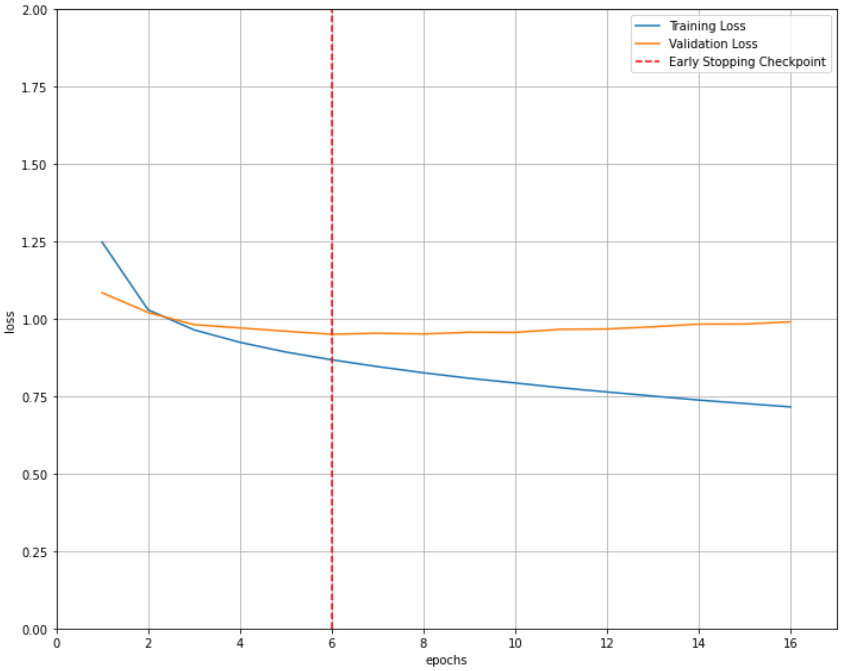

Figure 5.11: 1 residual block Policy Model Loss

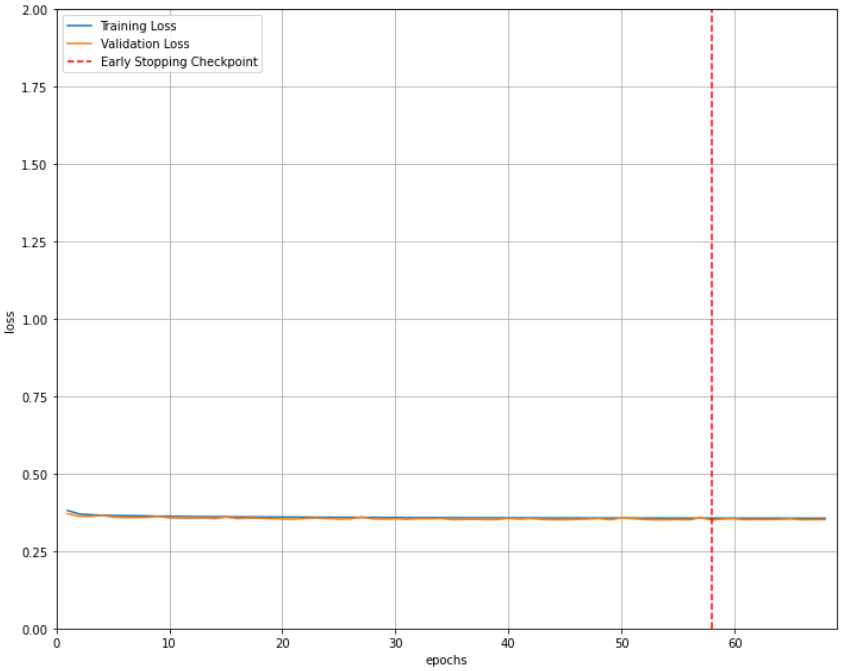

Figure 5.12: Simple Convolutional Value Model Loss

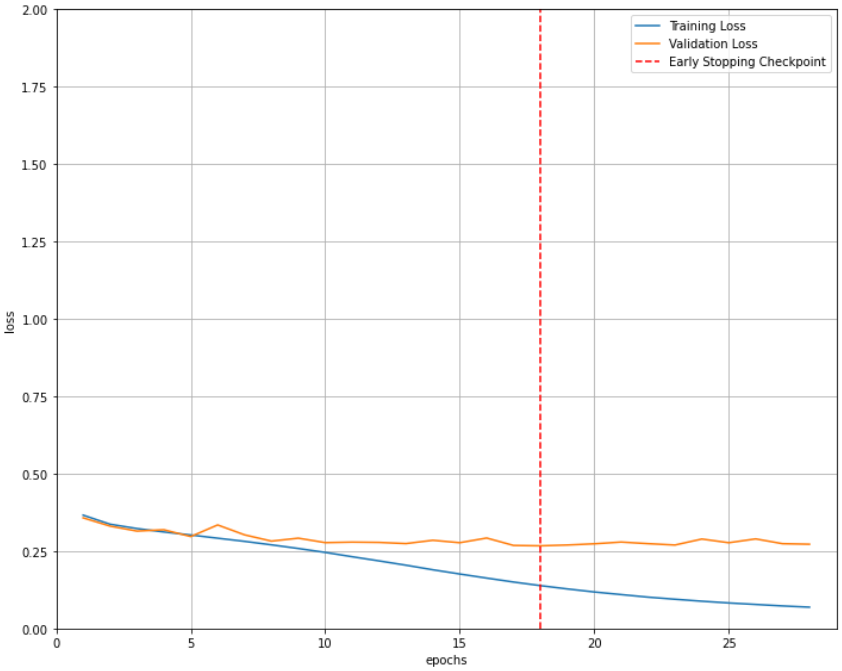

Figure 5.13: 20 residual blocks Value Model Loss

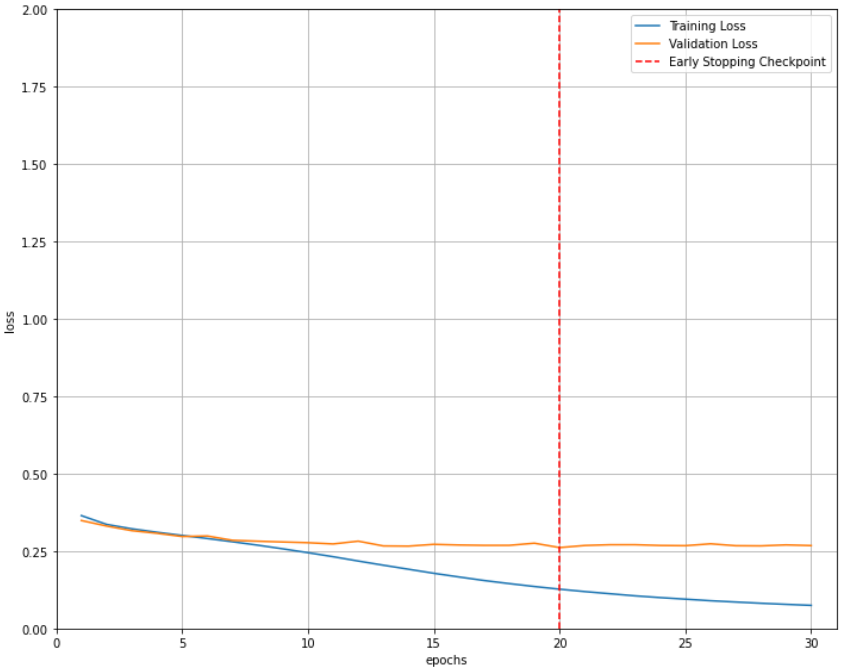

Figure 5.14: 10 residual blocks Value Model Loss

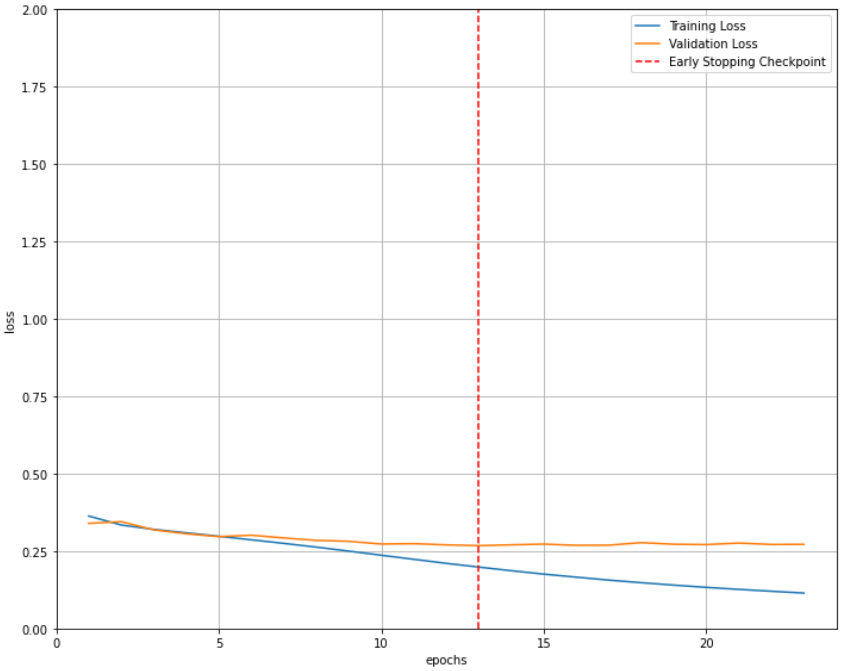

Figure 5.15: 5 residual blocks Value Model Loss

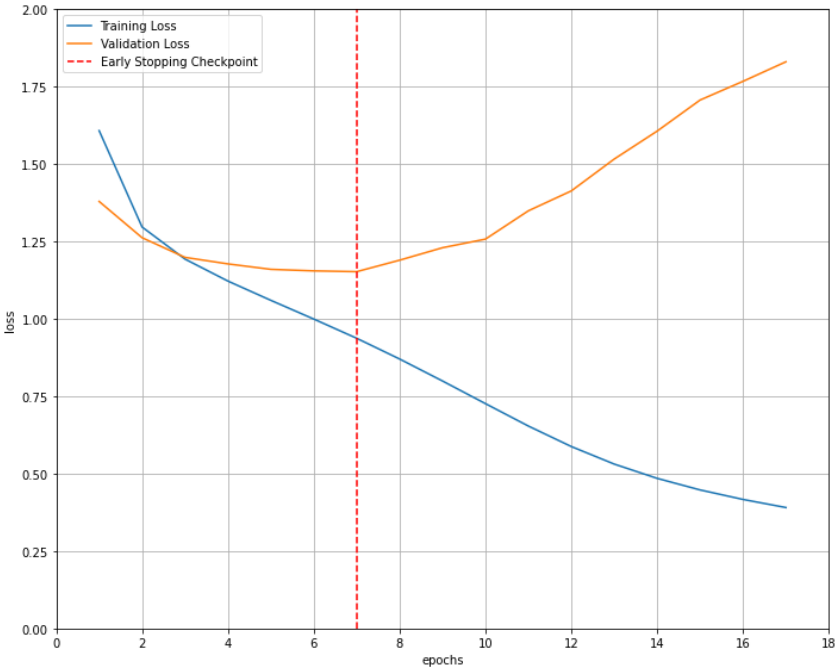

Figure 5.16: 20 residual blocks Double Model Loss

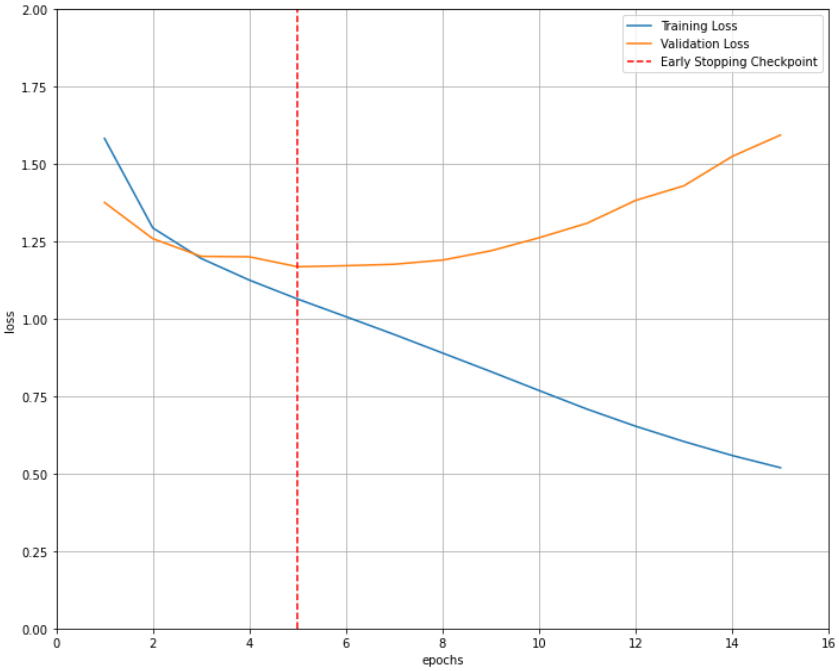

Figure 5.17: 10 residual blocks Double Model Loss

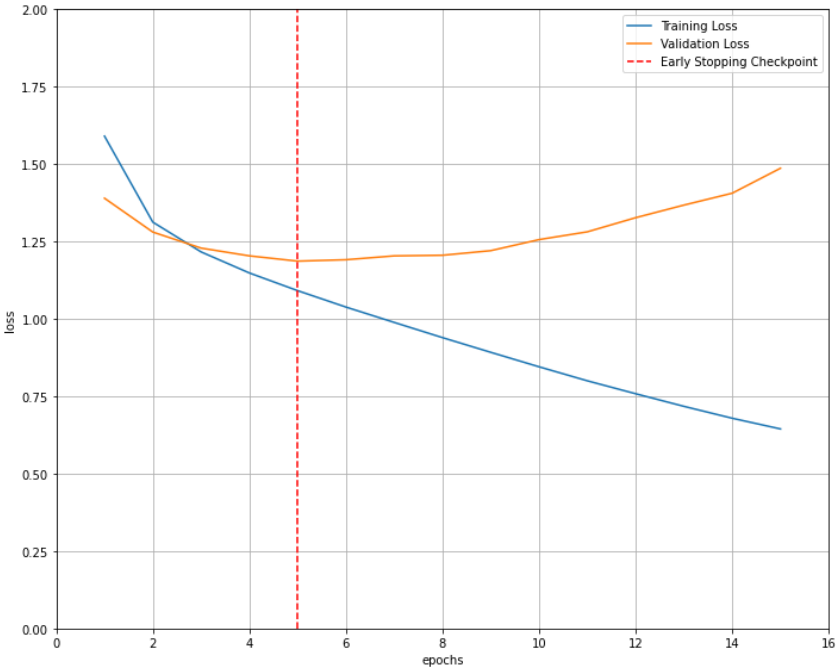

Figure 5.18: 5 residual blocks Double Model Loss

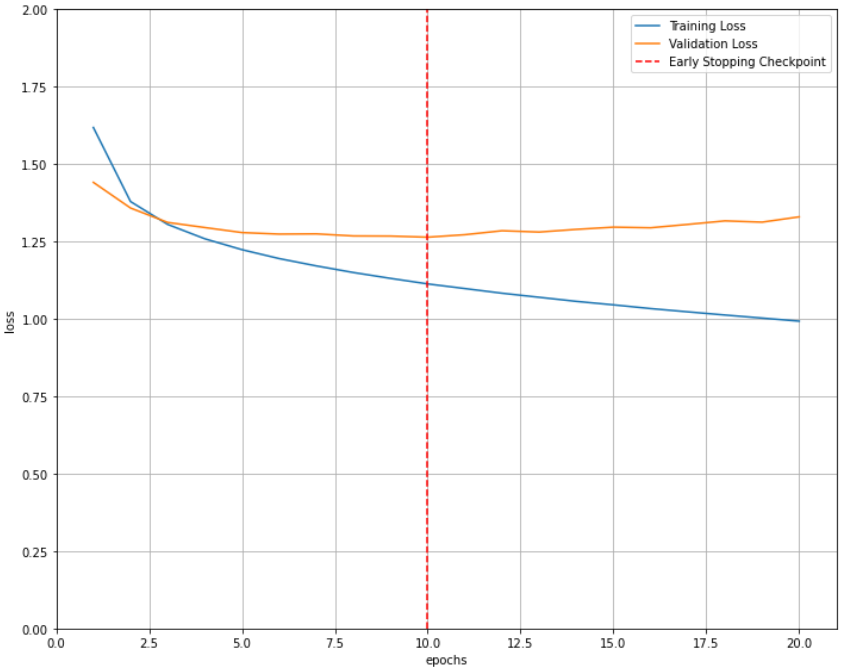

Figure 5.19: 1 residual block Double Model Loss

5.3.5 Αξιολόγηση Εκπαίδευσης

Για να αξιολογήσουμε την επίδοση των δικτύων χρησιμοποιούμε τις θέσεις που δεν χρησιμοποιήσαμε από την διάρκεια της εκπαίδευσης από το Test Set. Για το value δίκτυο το οποίο επιλύει ένα πρόβλημα regression δηλαδή προβλέπει μια συνεχής τιμή στο [-1,1] χρησιμοποιούμε ώς μετρική αξιολόγησης απλά το μέσο τετραγωνικό σφάλμα (MSE).

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{v}_i - v_i)^2$$

Για την αξιολόγηση του policy δικτύου το οποίο έχει έξοδο μια κατανομή πιθανοτήτων ελέγχουμε αν η κίνηση που προτείνει με τη μέγιστη πιθανότητα είναι ίδια με τη κίνηση της μέγιστης πιθανότητας όπως προκύπτει από το test set.

$$accuracy = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}(argmax_a\pi_\theta(a|s_i) = argmax_a\pi(a|s_i))$$

Value Networks	Mean Square Error
Simple Convolutional Net	0.4631
20 Residual Blocks Net	0.3965
10 Residual Blocks Net	0.3968
5 Residual Blocks Net	0.4120
1 Residual Block Net	0.4318

Table 5.1: Value Nets Test

Policy Networks	accuracy
Simple Convolutional Net	61.30%
20 Residual Blocks Net	67.064%
10 Residual Blocks Net	66.81%
5 Residual Blocks Net	66.18%
1 Residual Block Net	63.85%

Table 5.2: Policy Nets Test

Double Head Networks	accuracy	Mean Square Error
20 Residual Blocks Net	67.23%	0.4136
10 Residual Blocks Net	66.48%	0.4266
5 Residual Blocks Net	65.66%	0.4281
1 Residual Block Net	64.60%	0.4444

Table 5.3: Double Nets Test

5.4 MCTS + Νευρωνικά δίκτυα

Ο αλγόριθμος Monte Carlo tree search αν και ισχυρός και μπορεί να πετύχει καλά αποτελέσματα καθώς αυξάνουμε τον αριθμό των επαναλήψεων είναι αργός. Ο κύριος λόγος καθυστέρησης οφείλεται στο χρονοβόρο simulation που πραγματοποιούμε όταν διαλέξουμε ένα από τα παιδιά του φύλλου που μόλις έχουμε επεκτείνει μέχρι να φτάσουμε σε end-state και να λάβουμε το σήμα της αμοιβής.Μια τροποποίηση που μπορούμε να κάνουμε είναι να αντικαταστήσουμε το simulation με μια συνάρτηση αξίας που θα μας δίνει για τη θέση το expected reward ως μια συνεχής τιμή στο [-1,1]. Έτσι κάθε φορά που επεκτείνουμε έναν κόμβο αναθέτουμε αυτόματα στα παιδιά του μια τιμή αξίας.Επιπλέον μπορούμε αν γνωρίζουμε μια πολιτική να τη χρησιμοποιήσουμε για καθοδήγηση στο tree traversal και να μας βοηθήσει πιθανώς στο να αγνοήσουμε αδιάφορες εκβάσεις του παιχνιδιού και να εστιάσουμε στις σημαντικότερες. Η τροποποίηση αυτή του MCTS στηρίζεται στο έργο των [11],[12] Σε κάθε κόμβο λοιπόν εκτός από την αξία του, του καταχωρείται και η εκ των προτέρων πιθανότητα να προκύψει από τον γονικό κόμβο με βάση τη διαθέσιμη πολιτική. Δεδομένων λοιπόν δύο νευρωνικών δικτύων έχουμε τα εξής:

$$V(s) \approx V_\theta(s)$$
$$\pi(a|s) \approx \pi_w(s|a)$$

Μπορούμε να έχουμε και 1 νευρωνικό με τις δύο παραπάνω εξόδους και να μοιράζεται τις παραμέτρους για τις συνάρτηση αξίας και πολιτική.

Το στάδιο selection του αλγορίθμου όπως το περιγράψαμε σε προηγούμενη παράγραφο τώρα γίνεται διαλέγοντας πάντα τον θυγατρικό κόμβο που μεγιστοποιεί την παρακάτω παραλλαγή του UCT:

$$PUCT(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c_{puct} P(v, v_i) \frac{\sqrt{N(v)}}{1 + N(v_i)} \tag{5.8}$$

όπου:

- $Q(v_i)$ Το συνολικό άθροισμα από αμοιβές που έχει συσσωρευτεί στο κόμβο v_i
- $N(v_i)$ Ο συνολικός αριθμός ενημέρωσης της αμοιβής στο κόμβο v_i.
- $P(v, v_i)$ η εκ των προτέρων πιθανότητα να βρεθούμε από τον κόμβο v στο θυγατρικό κόμβο v_i
- $N(v)$ Ο συνολικός αριθμός ενημέρωσης της αμοιβής στο κόμβο v (πατέρας του κόμβου v_i).
- c_{puct} υπερπαράμετρος που ορίζει το βαθμό της εξερεύνησης. Τυπικά χρησιμοποιήσαμε την τιμή 1.

Όπως έχουμε αναφέρει ο πρώτος όρος αφορά το exploitation κομμάτι και στρέφει το ενδιαφέρον σε κόμβους που έχουν φανεί μέχρι στιγμής αποδοτικοί από πλευράς αμοιβής ενώ το δεύτερο ενώ ο δεύτερος όρος αφορά το exploration και αντισταθμίζει την διαδικασία επιλογής ώστε να ελέγχονται και υποσχόμενοι κόμβοι που δεν έχουν εξερευνηθεί ακόμα.

5.5 Αυτο-Ενισχυτική Μάθηση

Η βασική ιδέα αυτού του αλγορίθμου είναι ότι Ο MCTS μετά από επαρκή αριθμό επαναλήψεων μπορεί να παρέχει πολιτική εξόδου ισχυρότερη από την πολιτική που λαμβάνουμε απευθείας από το νευρωνικό δίκτυο [12].Έχουμε δηλαδή μια διαρκή επανάληψη κατά την οποία το νευρωνικό βοηθάει το MCTS να προτείνει καλύτερες κινήσεις και οι καλύτερες κινήσεις χρησιμοποιούνται ως δείγματα για να επανεκπαίδευση του νευρωνικού. Θυμίζουμε ότι για μια δεδομένη θέση ως ρίζα για είσοδος στον αλγόριθμο του MCTS μετά από κάποιον αριθμό επαναλήψεων και τον τερματισμό του, στα παιδιά της ρίζας έχουν ανατεθεί τιμές N_i που μας δείχνουν πόσες φορές επισκεφτήκαμε το παιδί i και προφανώς όσο μεγαλύτερη είναι αυτή η τιμή τόσο περισσότερο υποσχόμενο είναι όσον αφορά την έκβαση του παιχνιδιού. Συγκεκριμένα αποκτάμε μια πολιτική για την ρίζα $\overrightarrow{\pi}(s)$ με :

$$\pi(a_i|s) = \frac{N_i}{\sum_j N_j} \tag{5.9}$$

Ο πράκτορας μας ο οποίος παίζει το παιχνίδι αποτελείται από τον αλγόριθμο MCTS και τα δύο νευρωνικά με τα οποία τον καθοδηγούν. Η εκπαίδευση γίνεται με τον πράκτορα να παίζει παιχνίδια με τον εαυτό του και σε κάθε στάδιο να αποθηκεύει ένα δείγμα (s,p,·) όπου s η εκάστοτε θέση και p η πολιτική εξόδου του MCTS. Η τελευταία συνιστώσα ενημερώνεται μετά την λήξη του παιχνιδιού και είναι η έκβαση του με +1 , -1, 0 για αντίστοιχα νίκη, ήττα η ισοπαλία του παίκτη ο οποίος είχε σειρά στην θέση. Θυμίζουμε ότι τα states αποθηκεύονται με τη λογική ότι ο παίκτης ο οποίος παίζει έχει την προοπτική του λευκού και προσαρμόζονται ανάλογα οι κινήσεις της πολιτικής.

Τα δείγματα που αποθηκεύονται είναι ισχυρά συσχετισμένα. Σύμφωνα με την αντίστοιχη δημοσίευση της deepmind της Google από κάθε παιχνίδι πρέπει να λαμβάνουμε μόνο ένα δείγμα (s,p,z) και όχι όλη την παρτίδα. Λόγω των υπολογιστικών πόρων που διαθέτουμε και της μεγάλης διάρκειας που απαιτείται για να τελειώσει ένα self-play αγνοήσαμε αυτή την υπόδειξη. Για να αποσυσχετίσουμε ως εναν μικρό βαθμό τα δείγματα εισάγαμε έναν memory replay buffer που επιτελεί λειτουργία παρόμοια όπως στον DQN. Ο buffer είναι κυκλικός και απομακρύνονται τα παλαιότερα δείγματα για να εισαχθούν τα νέα. Από τον buffer δειγματολειπτούμε στη συνέχεια τυχαία τα δείγματα για την εκπαίδευση του δικτύου σε διαδικασία όμοια με την επιβλεπόμενη μάθηση αλλά χωρίς validation set.

Παρουσιάζουμε την διαδικασία που ακολουθήσαμε για ενισχυτική μάθηση σε ψευδοκώδικα καθώς και τις υπερπαραμέτρους της.

Algorithm 13: Reinforcement Learning Pipeline

```
Initialize Buffer = ReplayMemory;
Initialize Best model = model;
while Buffer not Full do
    Execute Self play;
    Buffer←(s,p,v) samples;
end
Train Best model;
new model ← Trained Net;
for i in range(max iterations) do
    while new model not imporved do
        update buffer;
        Train Best model;
    end
end
```

Self play Σε αυτό το στάδιο έχουμε δύο όμοιους πράκτορες οι οποίοι χρησιμοποιούν τον MCTS και τα νευρωνικά δίκτυα για να διαλέξουν κινήσεις. Από την πολιτική εξόδου του MCTS δεν παίζουμε την κίνηση που δίνεται με την μεγαλύτερη πιθανότητα αλλά δειγματολειπτούμε τυχαία από την πιθανοτική κατανομή για να μην είναι όλα τα παιχνίδια ίδια μεταξύ τους για δεδομένα νευρωνικά δίκτυα.

Αξιολόγηση Εξετάζουμε αν το δίκτυο βελτιώθηκε βάζοντας το να παίξει 20 παιχνίδια εναντίον του καλύτερου νευρωνικού μέχρι στιγμής. Αν έχει ποσοστό νίκης πάνω από 55% τότε θεωρούμε ότι βελτιώθηκε. Για τα παιχνίδια οι κινήσεις λαμβάνονται απευθείας από την πολιτική εξόδου του νευρωνικού χωρίς MCTS και για να εξασφαλίσουμε ότι είναι διαφορετικά οι 4 πρώτες κινήσεις γίνονται τυχαία.

Update Buffer Ο buffer επιλέξαμε να είναι 1000 θέσεων υπολογίζοντας το γεγονός ότι χωράει περίπου δέκα παιχνίδια. Αν για την εκπαίδευση του δικτύου πάνω στα δείγματα που έχουν αποθηκευθεί το νευρωνικό δεν βελτιώνεται, παίζουμε 5 νέα παιχνίδια με self play και αποθηκεύουμε τα νέα δείγματα ενώ διώχνουμε τα παλαιότερα με κυκλικό τρόπο.

Παρακάτω παρουσιάζουμε ενδεικτικά αποτελέσματα μετά από 10 επανεκπαιδεύσεις του νευρωνικού πολιτικής από την επιβλεπόμενη μάθηση.

Retrained Net	Best Model vs New model
1	96-1 3 draws
2	100-0
3	99-0 1 draw
4	95-3 2 draws
5	99-1
6	99-1
7	98-0 2 draws
8	100-0
9	99-1
10	99-1

Table 5.4: Επανεκπαιδεύσεις μέσω ενισχυτικής μάθησης

Ενώ θεωρητικά θα περιμέναμε κάποια βελτίωση καθώς όπως θα φανεί και στην επόμενη παράγραφο Ο MCTS σε συνδυασμό με το νευρωνικό δίνει ισχυρότερη πολιτική εξόδου , το νευρωνικό χειροτερεύει σημαντικά μετά από την εκπαίδευση της αυτοενισχυτικής μάθησης και δεν προσεγγίζει καθόλου ποσοστό νικών 55% . Οι νίκες που καταφέρνει μπορεί να θεωρηθούν και τυχαίες ή κυρίως λόγω των καταλοίπων της επιβλεπόμενης μάθησης του πιο ισχυρού μοντέλου πριν αρχίσει να χαλάει.Δηλαδή πρακτικά δεν καταφέρνουμε να ξεφύγουμε από το *while new model not improved* όπως περιγράφεται στο ψευδοκώδικα παραπάνω. Ο λόγος για τον οποίον συμβαίνει αυτό είναι η εξάρτηση μεταξύ των δειγμάτων εκπαίδευσης. Οι συγγραφείς του σχετικού άρθρου της deepmind για την ενισχυτική μάθηση του alpha zero αναφέρουν ότι από κάθε παιχνίδι επιλέγεται τυχαία μόνο μία θέση για εκπαίδευση του δικτύου μαζί με την πολιτική που της αντιστοιχεί και το αποτέλεσμα που προέκυψε. Λόγω των υπολογιστικών πόρων που διαθέτουμε κάτι τέτοιο είναι απαγορευτικό αν λάβουμε υπόψιν μας οτι για 700 επαναλήψεις που εκτελούμε τον MCTS κατά τη διάρκεια του self play και που είναι απαραίτητες ώστε να έχουμε πολιτική εξόδου καλύτερη από την πολιτική του υπάρχοντος καλύτερου νευρωνικού η κάθε κίνηση γίνεται σε 12sec. Ένα μέσο παιχνίδι selfplay διαρκεί περίπου 90 κινήσεις οπότε βλέπουμε ότι για την απόκτηση ενός και μόνου δείγματος ο χρόνος είναι τεράστιος.

5.6 Τύποι Παικτών-Αποτελέσματα

Συνολικά στο πλαίσιο αυτής της εργασίας δημιουργήσαμε κάποιους agents συνδυάζοντας όλες τις μεθόδους που έχουμε αναφέρει μέχρι στιγμής και αξιολογήσαμε τις επιδόσεις τους σε παιχνίδια μεταξύ τους.

Πράκτορας 1 Ο πράκτορας αυτός διαλέγει τις κινήσεις του τυχαία από ομοιόμορφη κατανομή.

Πράκτορας 2 Ο πράκτορας αυτός βασίζεται στον αλγόριθμό alpha beta. Όσον αφορά την ευριστική στους κόμβους φύλλα η έξοδος του νευρωνικού αξίας δεν μπόρεσε να συνδυαστεί με αυτόν τον αλγόριθμό, δίνοντας άσχημα αποτελέσματα ακόμα και με τον πράκτορα 1. Αντί για αυτό η ευριστική που χρησιμοποιήσαμε μετράει τα κομμάτια της θέσης προκειμένου να αναθέσει μια τιμή αξιολόγησης.

$$F(state) = \sum_i a_i$$

$$a_i = \begin{cases} 1 & \text{λευκός στρατιώτης} \\ -1 & \text{μαύρος στρατιώτης} \\ 3 & \text{λευκός βασιλιάς} \\ -3 & \text{μαύρος βασιλιάς} \end{cases}$$

Το βάθος του αλγορίθμου ορίστηκε ως 5.

Πράκτορας 3 Ο πράκτορας αυτός διαλέγει κινήσεις από την πολιτική εξόδου του απλού MCTS χωρίς νευρωνικά δίκτυα, όπου εκτελούνται προσομοιώσεις τυχαίων κινήσεων στα φύλλα για την απόκτηση της αμοιβής στα φύλλα.

Πράκτορας 4 Ο πράκτορας αυτός διαλέγει τις κινήσεις του απευθείας από την πολιτική εξόδου του νευρωνικού πολιτικής όπως εκπαιδεύτηκε με επιβλεπόμενη μάθηση.

Πράκτορας 5 Ο πράκτορας αυτός λειτουργεί όπως ο πράκτορας 3 άλλα οι προσομοιώσεις δεν γίνονται με τυχαίο τρόπο αλλά οι κινήσεις διαλέγονται από την πολιτική ενός ρηχού νευρωνικού. (του απλού συνελικτικού)

Πράκτορας 6 Χρησιμοποιεί τον MCTS ο οποίος καθοδηγείται από το δίκτυο πολιτικής από επιβλεπόμενη μάθηση και την κανονικοποιημένη ευριστική που χρησιμοποιεί ο πράκτορας 2

Πράκτορας 7 Όμοια με τον πράκτορα 6 αλλά η τιμή της ευριστικής αντικαθίσταται από μια τυχαία προσομοίωση.

Πράκτορας 8 Όμοια με τον πράκτορα 6 αλλά το νευρωνικό πολιτικής που καθοδηγεί τον MCTS προέρχεται από κάποιες εκπαιδεύσεις με αυτό-ενισχυτική μάθηση.

Πράκτορας 2	
Βάθος minimax	χρόνος κίνησης (sec)
Βάθος=1	0.0204
Βάθος=2	0.0373
Βάθος=3	0.1545
Βάθος=4	0.3841
Βάθος=5	1.1346
Βάθος=6	2.4676
Βάθος=7	7.6707
Βάθος=8	15.9327
Βάθος=9	36.9126
Βάθος=10	89.8176
Βάθος=11	184.3146

Table 5.5: Χρόνοι κινήσεων Για Πράκτορα 2

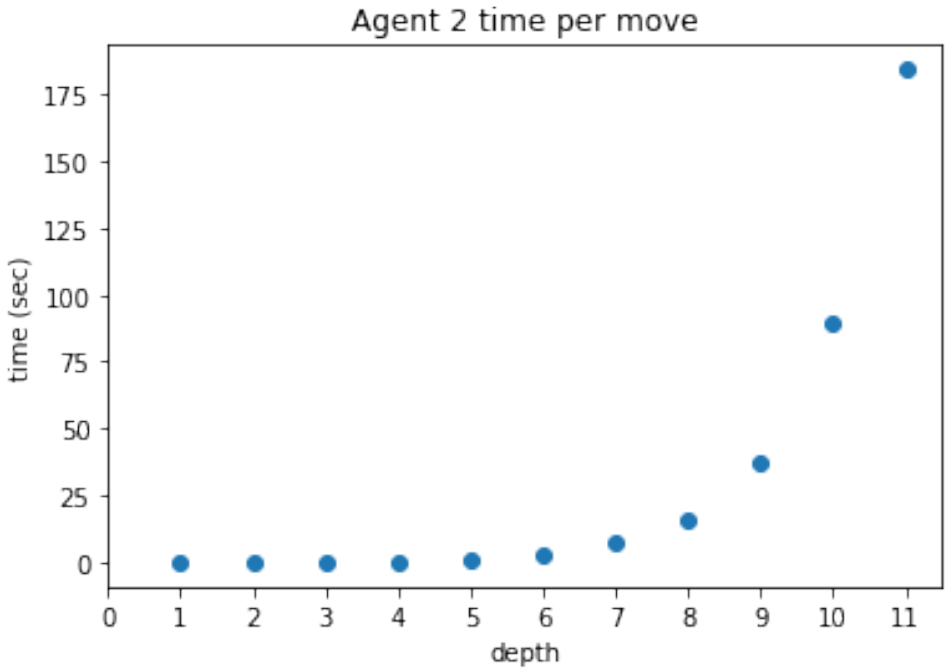

Figure 5.20: Agent 2 time per move

Πράκτορας 3	
Αριθμός Επαναλήψεων MCTS	χρόνος κίνησης (sec)
10	1.4900
100	34.1548
200	124.4183
300	275.5695
400	484.8207
500	761.5103
600	1112.2500
700	1538.8846
800	1965.4644
900	2659.7014
1000	3186.7178

Table 5.6: Χρόνοι κινήσεων Για Πράκτορα 3

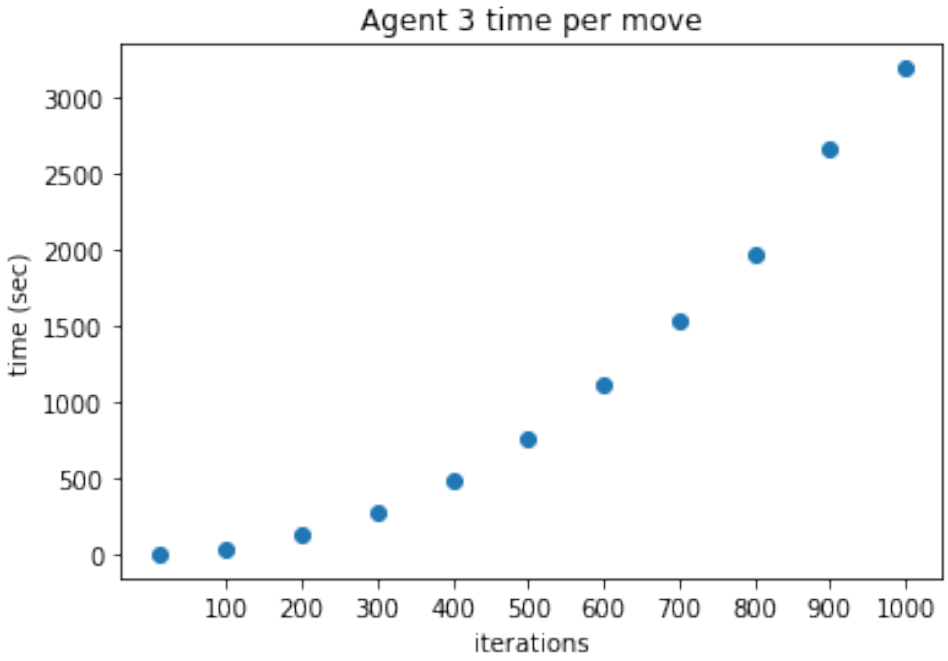

Figure 5.21: Agent 3 time per move

Πράκτορας 4		
Τύπος Νευρωνικού	χρόνος κίνησης σε CPU (sec)	χρόνος κίνησης σε GPU (sec)
Simple Convolutional Net	0.0025	0.0022
1 Res Block	0.0037	0.0022
5 Res Blocks	0.0066	0.0032
10 Res Blocks	0.0113	0.0046
20 Res Blocks	0.0249	0.0064

Table 5.7: Χρόνοι κινήσεων Για Πράκτορα 4

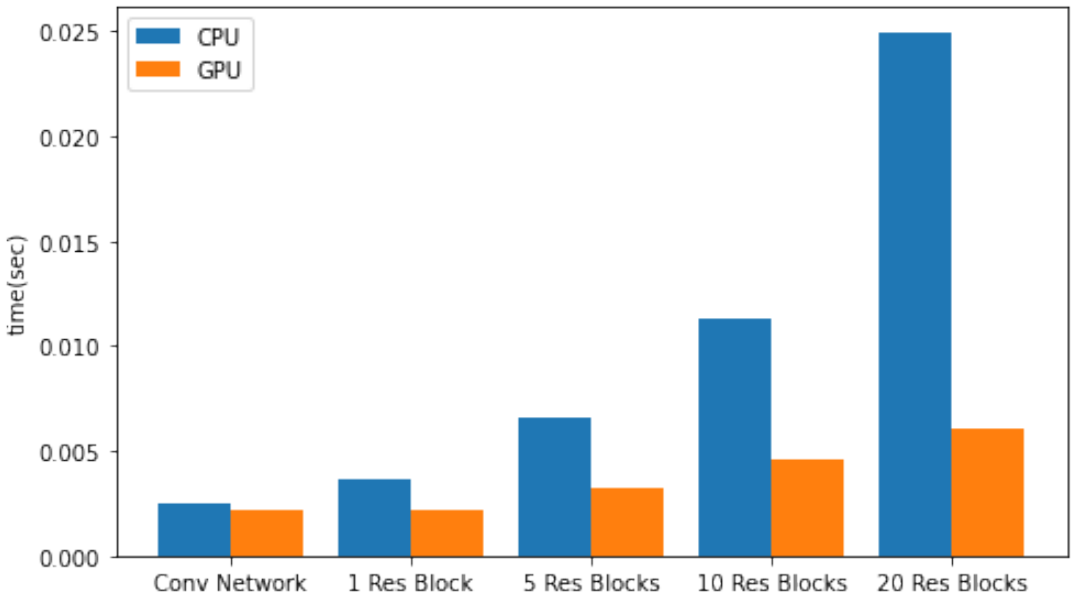

Figure 5.22: Agent 4 time per move

Πράκτορας 5		
Αριθμός Επαναλήψεων MCTS	χρόνος κίνησης σε CPU (sec)	χρόνος κίνησης σε GPU (sec)
10	3.020	2.6434
100	69.7192	43.2697
200	193.1637	135.2212
300	364.0799	278.2663
400	609.5468	477.0521
500	904.5464	686.3005
600	1321.0298	978.0201
700	1821.0298	1333.5503
800	2301.3382	1754.5278
900	2923.9374	2319.4980

Table 5.8: Χρόνοι κινήσεων Για Πράκτορα 5

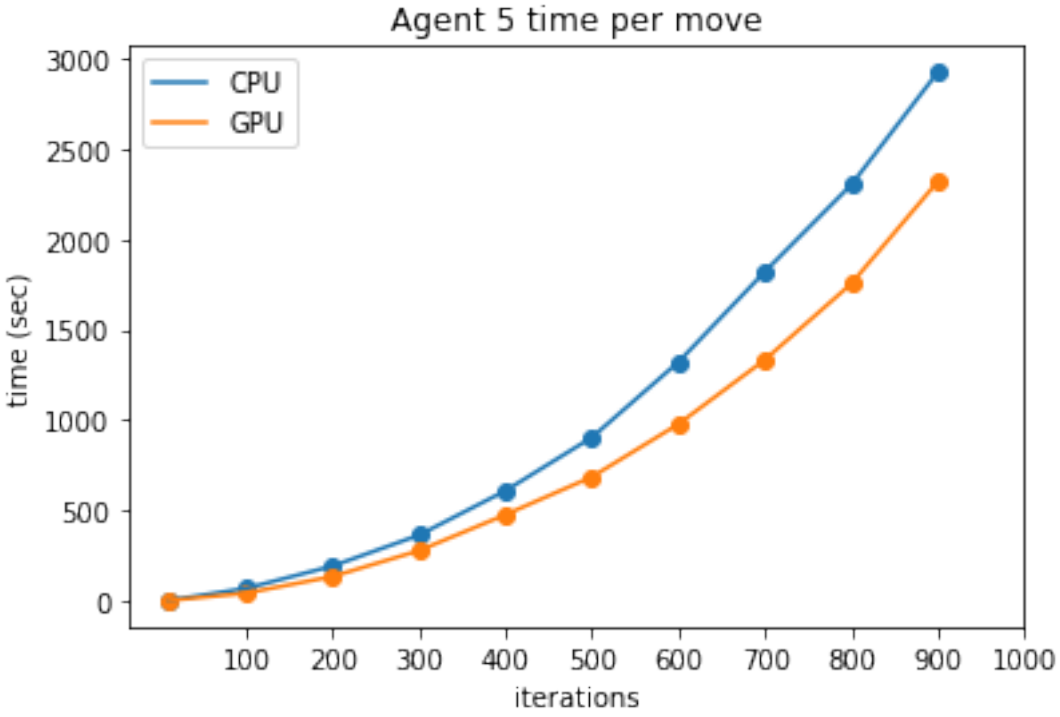

Figure 5.23: Agent 5 time per move

Πράκτορας 6		
Αριθμός Επαναλήψεων MCTS	χρόνος κίνησης σε CPU	χρόνος κίνησης σε GPU
10	0.3002	0.2370
100	1.9724	1.3514
200	3.6709	2.4284
300	5.4600	3.5847
400	7.1161	4.8323
500	8.9658	6.2354
600	10.5366	7.1421
700	12.6203	8.5813
800	14.3717	9.7600
900	15.9705	10.5077
1000	17.5118	12.0698

Table 5.9: Χρόνοι κινήσεων Για Πράκτορα 6

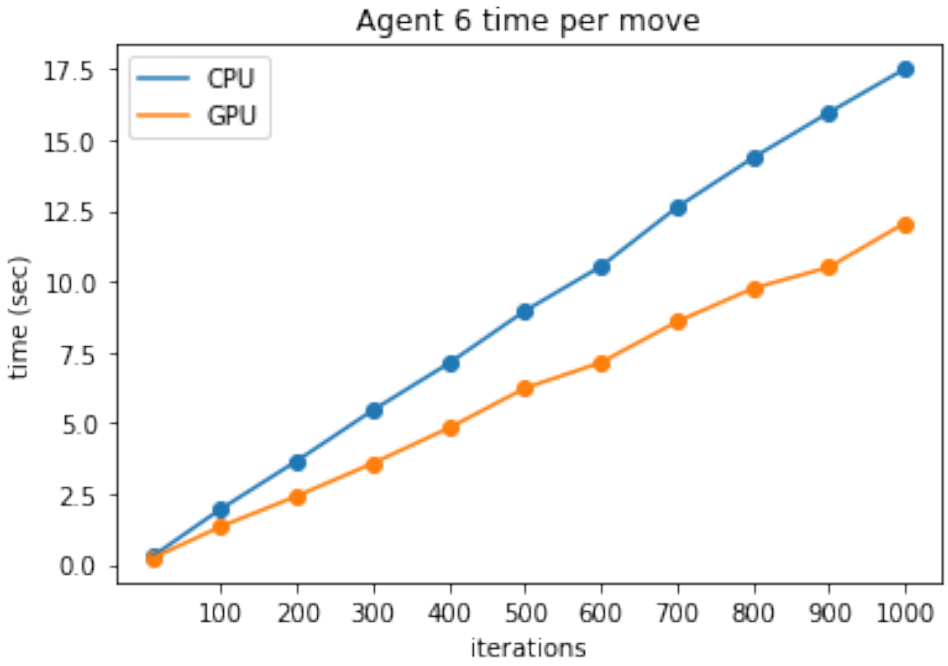

Figure 5.24: Agent 6 time per move

Πράκτορας 7		
Αριθμός Επαναλήψεων MCTS	χρόνος κίνησης σε CPU (sec)	χρόνος κίνησης σε GPU (sec)
10	1.2736	1.097
100	39.6595	32.8868
200	142.3332	113.2410
300	336.6219	245.6334
400	617.2375	416.8263
500	879.8589	645.6867
600	1263.4751	1136.5665
700	1872.5272	1506.0471
800	2319.7489	1907.2093
900	3414.0825	2633.1753

Table 5.10: Χρόνοι κινήσεων Για Πράκτορα 7

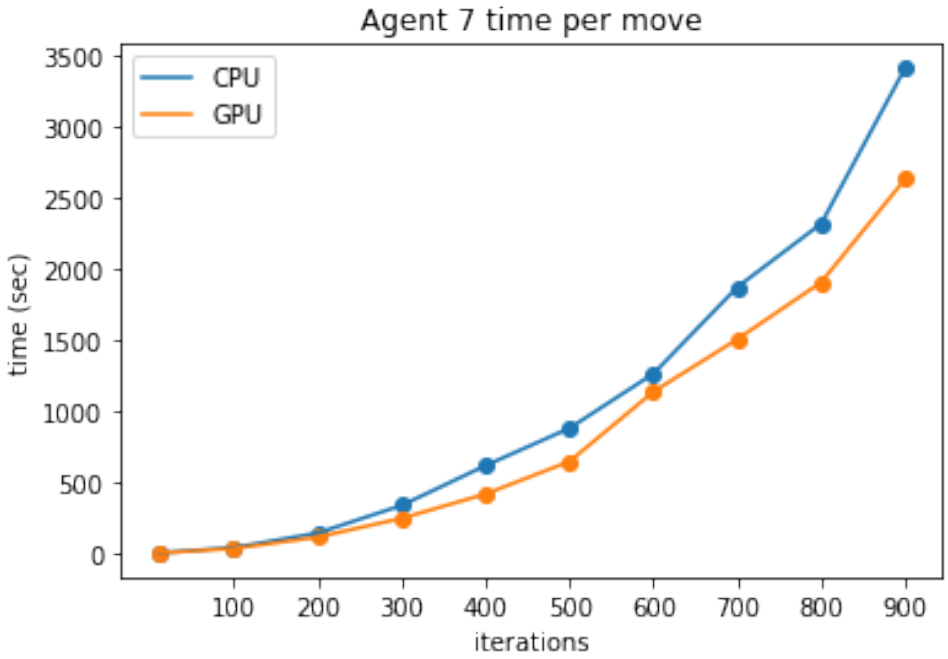

Figure 5.25: Agent 7 time per move

White/Black	Π1	Π2	Π3	Π4
Π1		0%-100% Π2 depth=5	0%-86% 14% draws Π3 iter=10	0%-100%
Π2	96%-0% 4% draws Π2 depth=5		10%-10% 80% draws Π3 iter=100	12%-28% 60% draws Π2 depth=5
Π3	90%-4% 6% draws Π3 iter=10	20%-10% 70% draws Π3 iter=100 Π2 depth=5		10%-80% 10% draw Π3 iter=100
Π4	100%-0%	30%-2% 68% draws Π2 depth=5	60%-0% 40% draws Π3 iter=100	
Π5	90%-4% 6% draws Π5 iter=100	0%-40% 60% draws Π5 iter=100 Π2 depth=5	23%-27% 50% draws Π5 iter=100 Π3 iter=100	0%-70% 30% draws Π5 iter=100
Π6	96%-0 4% draws Π6 iter=20	16%-4% 80% draws Π2 depth=5 Π6 iter=100	30%-10% 60% draws Π6 iter=100 Π3 iter=100	28%-6% 66% draws Π6 iter=100
Π7	97%-0% 3% draws Π7 iter=10	30%-10% 60% draws Π7 iter=100 Π2 depth=5	50%-0% 50% draws Π7 iter=50 Π3 iter=50	20%-20% 60% draws Π7 iter=100

Table 5.11: Αποτελέσματα παιχνιδιών 1

White/Black	Π5	Π6	Π7
Π1	4%-90% 6% draws Π5 iter=10	0%-93% 7% draws Π6 iter=10	1%-98% 1% draw Π7 iter=10
Π2	0%-20% 80% draws Π2 depth=5 Π5 iter=100	2%-16% 82% draws Π2 depth=5 Π6 iter=100	10%-40% 50% draws Π2 depth=5 Π7 iter=100
Π3	21%-24% 55% draws Π3 iter=10 Π5 iter=10	0%-50% 50% draws Π6 iter=100 Π3 iter=100	6%-68% 26% draws Π3 iter=50 Π7 iter=50
Π4	70%-0% 30% draws Π5 iter=100	12%-26% 62% draws Π6 iter=100	0%-40% 60% draws Π7 iter=100
Π5		2%-30% 68% draws Π6 iter=20 Π5 iter=20	10%-60% 30% draws Π5 iter=50 Π7 iter=50
Π6	36%-0% 64% draws Π6 iter=20 Π5 iter=20		14%-8% 78% draws Π6 iter=20 Π7 iter=20
Π7	30%-0% 70% draws Π5 iter=50 Π7 iter=50	14%-14% 72% draws Π6 iter=20,Π7 iter=20	

Table 5.12: Αποτελέσματα παιχνιδιών 2

5.7 Συμπεράσματα - Επιπλέον Κατευθύνσεις

Τα παιχνίδια μεταξύ των πρακτόρων καθώς και οι πίνακες με τους χρόνους κινήσεων καθώς αυξάνουμε το βάθος για τον minimax ή τις επαναλήψεις για τον MCTS μας βοηθάνε να αποκτήσουμε μια καλή εποπτεία όσον αφορά την σύγκριση μεταξύ τους. Επειδή η επιλογή των κινήσεων σε κάποιους πράκτορες γίνεται ντετερμινιστικά στα παιχνίδια που διεξάγονται θέσαμε έναν αριθμό κινήσεων στην αρχή του παιχνιδιού να γίνονται τυχαίες ώστε να εξασφαλίσουμε διαφορετικές εκβάσεις.Τα πειράματα έγιναν στο google colab και οι χρόνοι κινήσεων αναφέρονται σε CPU: Intel(R) Xeon(R) CPU @ 2.30GHz και GPU: Tesla P100-PCIE-16GB

Για αρχή όπως ήταν αναμενόμενο ο πράκτορας 1 (ο random player) συνετρίβη από όλους τους πράκτορες γεγονός που μας επιβεβαιώνει ότι οι αλγόριθμοι δουλεύουν προς τη σωστή κατεύθυνση για τη νίκη στο παιχνίδι.

Ο πράκτορας 8 που χρησιμοποιεί το νευρωνικό από την αύτο-ενισχυτική μάθηση στον MCTS είναι απλά μια χειρότερη έκδοση του πράκτορα 6.Ενώ το νευρωνικό του πράκτορα 4 καταφέρνει το απόλυτο 100-0 εναντίον του πράκτορα 1, το νευρωνικό του πράκτορα 8 πετυχαίνει ένα σκορ 72-16 με 12 ισοπαλίες. Για τον λόγο αυτό δεν ασχοληθήκαμε παραπάνω με αυτόν σε παιχνίδια με τους υπολοίπους πράκτορες.

Ο πράκτορας 2 που αποτελεί τον κλασικό τρόπο λειτουργίας των περισσότερων μηχανών ανάλυσης για παιχνίδια όπως το σκάκι το shogi, το οθελο κτλ βλέπουμε οτι αν και είχε αρνητικό σκορ με τους υπόλοιπους παίκτες μπορεί να συγκριθεί με τις παραλλαγές του mcts και αποτελεί ισχυρό παίκτη αν λάβουμε υπόψιν μας ότι η ευριστική που χρησιμοποιεί για την αξιολόγηση των κόμβων-φύλλων είναι η απλούστερη δυνατή και το βάθος του έχει οριστεί ως 5 εκτελώντας κινήσεις σε περίπου 1sec για ταχύτητα στην εκτέλεση των αγώνων. Αυξάνοντας του το βάθος ο παίκτης αυτός χάνει με περισσότερη δυσκολία τα κομμάτια του και χάνει πολύ δύσκολα αλλά με κόστος την εκθετική αύξηση του χρόνου ανά κίνηση καθώς εξετάζεται όλο και μεγαλύτερο μέρος στο χώρο καταστάσεων του παιχνιδιού.

Ο πράκτορας 3 είναι ο μοναδικός πράκτορας που δεν χρησιμοποιεί καμία απολύτως γνώση του παιχνιδιού παρά μόνο των κανόνων του. Ο σκέτος MCTS εκτελούμενος με μόνο 10 επαναλήψεις ανά κίνηση φαίνεται ότι είναι ικανός να φέρει στη χειρότερη περίπτωση μόνο μερικές ισοπαλίες με τον random player φανερώνοντας έτσι την αποτελεσματικότητα του αλγορίθμου να δειγματοληπτεί τον χώρο καταστάσεων μέσα από τυχαίες προσομοιώσεις με μοναδικό σήμα αμοιβής αυτό της νίκης ή της ήττας.Παρατηρούμε ότι φέρνει αρνητικά σκορ ενάντια των παραλλαγών του στους πράκτορες 5,6 και 7 που είναι λογικό καθώς ενσωματώνουν περισσότερη πληροφορία για το παιχνίδι. Ως μεγάλο του μειονέκτημα μπορούμε να δούμε ότι είναι αρκετά αργός καθώς αυξάνουμε τον αριθμό των επαναλήψεων για να αυξήσουμε την δύναμη του και αντισταθμίζει έτσι την έλλειψη ενσωματωμένης γνώσης.

Ο πράκτορας 5 που αποτελεί ελάχιστη τροποποίηση ως προς τον πράκτορα 3 και ενώ αυξάνει κατά πολύ τον χρόνο κινήσεων δεν παρατηρούμε κάποια βελτίωση στην ακρίβεια των προτεινόμενων κινήσεων για ίδιο αριθμό επαναλήψεων στον MCTS.Πρακτικά φαίνεται λόγω της πολυπλοκότητας του παιχνιδιού οι προσομοιώσεις συμφέρει να γίνονται τυχαία και να μην έχουν την απόλυτη εξάρτηση από μια rollout policy η οποία μάλιστα απέχει κατά πολύ από τη βέλτιστη. Για τον ίδιο χρόνο ανά κίνηση συμφέρει δηλαδή η στρατηγική του πράκτορα 3.

Ο πράκτορας 6 φαίνεται ότι είναι ο ισχυρότερος συνδυάζοντας δυνατές κινήσεις με τον λιγότερο χρόνο ανά κίνηση. Ο MCTS καθοδηγούμενος από ένα ισχυρό νευρωνικό πολιτικής και μια συνάρτηση αξίας,που ανταποκρίνεται στη στρατηγική του παιχνιδιού όπως

φάνηκε και από την εφαρμογή της στον minimax,δειγματοληπτεί αποτελεσματικά τον χώρο καταστάσεων εστιάζοντας στις πιο ενδιαφέρουσες εκβάσεις του παιχνιδιού.Νικάει με ευκολία τους υπόλοιπους πράκτορες ακόμα και αν χρησιμοποιούν περισσότερο χρόνο ανά κίνηση με εξαίρεση τον πράκτορα 7 που βρίσκονται περίπου στο ίδιο επίπεδο.Με μόλις 100 επαναλήψεις στην εκτέλεση του MCTS για την επιλογή της κίνησης είναι ικανός να νικήσει σχετικά εύκολα το μεμονωμένο νευρωνικό πολιτικής που είναι ενσωματωμένο σε αυτόν. Παράγει δηλαδή καλύτερη πολιτική εξόδου και για αυτό με 800 επαναλήψεις μπορεί να χρησιμοποιηθεί για αυτο-ενισχυτική μάθηση και βελτίωση του νευρωνικού πολιτικής,όπως παρουσιάσαμε σε προηγούμενη παράγραφο, αν διαθέτουμε και τους απαραίτητους υπολογιστικούς πόρους.

Ο πράκτορας 7 ως προς τον πράκτορα 6 έχει παρόμοια συμπεριφορά όπως ο 3 με τον 5. Πάλι φαίνεται ότι συμφέρει για τον ίδιο αριθμό επαναλήψεων το σήμα αμοιβής στον MCTS να λαμβάνεται από τυχαίες προσομοιώσεις και όχι σε απόλυτη εξάρτηση από μια ευριστική. Επειδή όμως στο στάδιο επέκτασης του κάθε κόμβου αναθέτουμε τιμές αξίας σε κάθε παιδί του αυξάνεται σημαντικά ο χρόνος ανά κίνηση και για παράδειγμα με τους δύο πράκτορες να εκτελούν τον MCTS για 600 επαναλήψεις ο πράκτορας 7 γίνεται 120 φορές πιο αργός. Μια τέτοια χρονική διαφορά υποδεικνύει ότι στον ίδιο χρόνο ανά κίνηση συμφέρει προφανώς κατά πολύ η στρατηγική του πράκτορα 6.

Ο πράκτορας 4 είναι εντελώς διαφορετικής λογικής από τους υπόλοιπους πράκτορες και δεν μπορούμε να κάνουμε άμεση σύγκριση μεταξύ τους καθώς δεν πραγματοποιεί κάποια δειγματοληψία ή αναζήτηση στο χώρο καταστάσεων. Για τα παραπάνω παιχνίδια χρησιμοποιήσαμε το νευρωνικό με 5 Residual Blocks και σε σχέση με τους υπόλοιπους πράκτορες όλες οι κινήσεις μπορούμε να θεωρήσουμε ότι γίνονται στιγμιαία. Χωρίς να χρειάζεται να βλέπει μπροστά κινήσεις είναι ικανό να νικήσει 100-0 τον random player και να φέρει το αξιοσημείωτο 15-1 με τον πράκτορα 2. Νικάει με ευκολία τους πράκτορες 3 και 5 εφόσον δεν τους αυξάνουμε σημαντικά τον αριθμό επαναλήψεων στον MCTS και έχει αρνητικό σκορ μόνο με τους πράκτορες 6 και 7. Αξίζει να σημειώσουμε ότι το νευρωνικό εκπαιδεύτηκε να αναγνωρίζει μοτίβα και κινήσεις ανά θέση όπως προέκυψαν από το dataset των παιχνιδιών και επομένως δεν γνωρίζει τους κανόνες του παιχνιδιού με κάποιες φορές προτεινόμενες κινήσεις από την κατανομή πιθανοτήτων εξόδου του να είναι και μη έγκυρες για αυτό και η έξοδος φιλτράρεται σε σχέση με τις διαθέσιμες κινήσεις στη θέση. Η αδυναμία του δικτύου βρίσκεται κυρίως στην τελευταία φάση του παιχνιδιού (φινάλε) στην οποία ακόμα και με υλικό πλεονέκτημα μπορεί να αδυνατεί να αιχμαλωτίσει τα τελευταία κομμάτια του αντιπάλου παίζοντας πολλές φορές άσκοπες κινήσεις. Αυτό οφείλεται στο ότι κατά την εκπαίδευσή του οι θέσεις που του παρέχονταν από την τελική φάση του παιχνιδιού ήταν ελάχιστες καθώς μεταξύ ανθρώπων το υλικό πλεονέκτημα με ελάχιστα εναπομείναντα κομμάτια θεωρείται θέμα τεχνικής όσον αφορά την νίκη, με συγκεκριμένες στρατηγικές και κινήσεις έτσι ώστε ο παίκτης σε μειονεκτική θέση να εγκαταλείπει στην πλειοψηφία των περιπτώσεων. Η αδυναμία αυτή του δικτύου μεταφέρεται λοιπόν και στους πράκτορες που το χρησιμοποιούν για καθοδήγηση στον MCTS.

Κάποιος γνώστης του παιχνιδιού και των στρατηγικών του θα μπορούσε να ενσωματώσει στο διάνυσμα κατάστασης για την εκπαίδευση και επιπλέον πληροφορία που ενδεχομένως να αύξανε την αποτελεσματικότητα του δικτύου, ή να εφαρμόσει κάποια κατηγοριοποίηση των κόμβων τροποποιώντας την διαδικασία επιλογής τους στον MCTS. Κάτι τέτοιο όμως δεν είναι στο πνεύμα αυτής της εργασίας που εξετάζει μεθόδους εκμάθησης με ελάχιστη παρέμβαση από τον προγραμματιστή. Μια μεγαλύτερη συλλογή από παιχνίδια ίσως βοηθούσε και στην εκπαίδευση του νευρωνικού αξίας το οποίο είχε να προσεγγίσει μια εξαιρετικά μη γραμ-

μική σχέση. Η αξιολόγηση της θέσης μπορεί από νίκη να οδηγεί σε ήττα με μια ελάχιστη αλλαγή στην κατάσταση επομένως αν για την εκπαίδευση του δικτύου αξίας βασιζόμαστε μόνο στα τελικά αποτελέσματα των θέσεων ο αριθμός δειγμάτων που απαιτούνται είναι τεράστιος. Τα νευρωνικά δίκτυα και η ευριστική αξίας παίζουν μεγάλο ρόλο στην εκτέλεση του MCTS οπότε ακόμα και μικρές βελτιώσεις τους θα έχουν σημαντικό αποτέλεσμα. Τέλος η αποτελεσματικότητα στον αλγόριθμο ενισχυτικής μάθησης έρχεται με κόστος ως προς τους υπολογιστικούς πόρους και κυρίως όσον αφορά το self-play απαιτούνται πολλαπλοί πυρήνες για παράλληλη εκτέλεση ενώ μια παραλληλοποιημένη υλοποίηση του MCTS ενδεχομένως να βοηθούσε κάπως.

Αναφορές

[1] Simon Haykin, *Neural Networks and Learning Machines*.

[2] Andrew Ng Coursera, *Deep Learning Specialization*.

[3] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*.

[4] David Silver, *Ucl course on rl*. [Online]. Available: https://www.davidsilver.uk/teaching/.

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning",

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, "Playing atari with deep reinforcement learning",

[7] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas, "Dueling network architectures for deep reinforcement learning",

[8] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, "Prioritized experience replay",

[9] Xiaobai Ma, Katherine Driggs-Campbell, Zongzhang Zhang, and Mykel J. Kochenderfer1, "Monte-carlo tree search for policy optimization",

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition",

[11] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis, "Mastering the game of go with deep neural networks and tree search",

[12] David Silver, Thomas Hubert1, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez1, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play",