ΠΟΛΥΤΕΧΝΕΙΟ
ΚΡΗΤΗΣ

ΠΛΗ513 - Αυτόνομοι Πράκτορες

Minesweeper (Deep) Q-Agent

Όνομα	ΑΜ
Παπαδόπουλος Αργύρης	2014030158

Περιεχόμενα

1 Εισαγωγή **2**

2 Minesweeper - Παιχνίδι και κανόνες **2**

3 Περιβάλλον και Πράκτορας **2**

4 Λύση 1: Q-Learning **3**

5 Λύση 2: Deep Q-Learning **4**

6 Αποτελέσματα **5**

7 Επίλογος και μελλοντική δουλειά **6**

1 Εισαγωγή

Η συγκεκριμένη δουλειά παρουσιάζει την διαδικασία εκπαίδευσης ενός πράκτορα με χρήση Ενισχυτικής Μάθησης ώστε να παίζει το παιχνίδι Ναρκαλιευτής ή αλλιώς Minesweeper. Παρακάτω θα αναφερθούν δύο διαφορετικοί τρόποι μάθησης, Q-Learning, Deep Q-Learning μαζί με αποτελέσματα και σχόλια.

2 Minesweeper - Παιχνίδι και κανόνες

Το Minesweeper αποτελεί μία από τις κλασσικές εφαρμογές του λειτουργικού Windows XP που ο σκοπός του είναι σχετικά απλός, αλλά οι επιλογές και οι στρατηγικές έχουν αρκετό βάθος.

Το board αποτελείται από ένα grid $M \times N$ κελιών και στην αρχή του παιχνιδιού είναι όλα άγνωστα. Οι κανόνες έχουν ως εξής. Ο χρήστης ανοίγει ένα άγνωστο κελί και εμφανίζεται η πραγματική τιμή του. Αν το κελί που πατήθηκε είναι:

- βόμβα, ο χρήστης χάνει την παρτίδα.
- κενό, ανοίγουν περισσότερα από ένα κελιά.
- αριθμός, ο παίκτης έχει γνώση για το πόσες βόμβες βρίσκονται στα γειτονικά του κελιού που εμφάνισε.

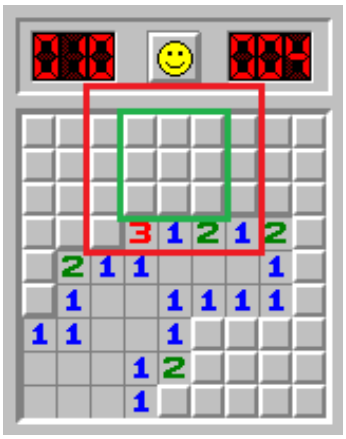
Ο παίκτης έχει την δυνατότητα να σημειώνει σε ποια κελιά θεωρεί πως υπάρχουν βόμβες, και κερδίζει το παιχνίδι εφόσον ανοίξει όλα τα κελία που δεν έχουν βόμβες. Οι κανόνες είναι αρκετά απλοί. Μια κλασσική στρατηγική είναι να αναλύονται με πιθανότητες τα κελιά, έτσι ώστε να επιδιώκεται η σιγουρότερη κίνηση (αν ο αριθμός ενός κελιού είναι ίσος με τον αριθμό των γειτονικών άγνωστων, τότε όλα τα άγνωστα είναι βόμβες), καθώς και αφαιρετική λογική (αν ο αριθμός ενός κελιού είναι ίσος με τις σίγουρες σημειωμένες βόμβες τότε όλα τα υπόλοιπα γειτονικά άγνωστα κελιά είναι σίγουρα όχι βόμβες). Προφανώς στην περίπτωση που υπάρχει αβεβαιότητα τότε επιλέγεται η μεγαλύτερη πιθανότητα να μην υπάρχει βόμβα.

3 Περιβάλλον και Πράκτορας

Η επικοινωνία μεταξύ πράκτορα και περιβάλλον προκύπτει από τις έτοιμες βιβλιοθήκες της python που επιτρέπουν την χρήση του ποντικιού, πληκτρολόγιου (pynput) για να δίνει εντολές ο πράκτορας (κινήσεις) και για να λαμβάνει δεδομένα χρησιμοποιεί mss που επιτρέπει screen capture σε συγκεκριμένα σημεία της οθόνης. Όταν αναγνωρίζει πως ένα παιχνίδι έχει τερματίσει, είτε από νίκη είτε από ήττα, κανονίζει το reset πατώντας το πλήκτρο F2 αυτόματα.

4 Λύση 1: Q-Learning

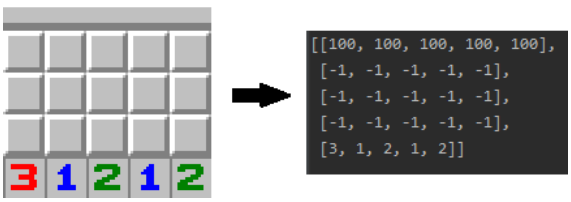
Το μεγαλύτερο θέμα σε αυτή την δουλειά ήταν ο χαρακτηρισμός states, actions. Για το Beginner mode που είναι 9×9 grid οι διαφορετικές καταστάσεις board φτάνουν αστρονομικό νούμερο. Πιο συγκεκριμένα αν υπάρχουν 11 διαφορετικές τιμές που μπορεί να πάρει ένα κελί (1-8 αριθμοί, άγνωστο κελί, κενό κελί, βόμβα), τότε υπάρχουν 11^{81} διαφορετικές καταστάσεις με 81 actions όπου το κάθε ένα αντιστοιχεί σε ένα κελί (πχ action 0: $\rightarrow$ open cell at (0, 0)).
Επομένως η λύση που σκέφτηκα ήταν κάποιο είδος διαμοίρασης. Δηλαδή δεν θα λαμβάνεται όλο το board ως κατάσταση, αλλά μια περιοχή του, 5×5. Τα actions που μπορεί να λάβει είναι 9 και χαρακτηρίζονται από το 3×3 που υπάρχει στο κέντρο ενός state. Θεώρησα σημαντικό να υπάρχει ένα layer παρατήρησης επιπλέον και να μην χαρακτηρίζεται ένα state από 3×3 διότι μου φάνηκε περιοριστικό.



Σχήμα 1: Μια κατάσταση της παρτίδας

Παρατηρείται πως διαβάζεται και ο τοίχος εφόσον βοηθάει στην πληροφορία. Είναι απαραίτητο να ληφθούν οι καταστάσεις έτσι ώστε να μπορεί να λάβει action ο πράκτορας σε κάθε κελί. Άρα στην προκειμένη περίπτωση της παρτίδας πριν από κάθε action θα λαμβάνει 9 καταστάσεις **(το πράσινο κουτάκι θα περνάει από κάθε 9-άδα μία φορά)**.

Για την αναγνώριση μιας κατάστασης χρησιμοποιήθηκε πολύ απλή σύγκριση χρωμάτων, για την μετονομασία των καταστάσεων από εικόνα σε λίστα, έτσι ώστε να καταγραφτεί σε python dictionary με μικρότερη πληροφορία και όχι με pixels εικόνας. Όπως φαίνεται και στην παρακάτω εικόνα μία κατάσταση μεταφράζεται σε λίστα.



Σχήμα 2: Μετάφραση κατάστασης για εισαγωγή στο dictionary

Η αντιστοιχία είναι ως εξής:

- αριθμοί 1-8: 1-8
- τοίχος: 100

- άγνωστο κελί: -1
- κενό κελί: 0

Και ένα κλειδί του dictionary αποτελείται από μία λίστα (όπως φαίνεται και στην εικόνα παραπάνω) η οποία γίνεται cast σε string.

Σε κάθε μία κατάσταση, προφανώς αν κάποιο κελί από τα 9 κεντρικά δεν είναι άγνωστο, που σημαίνει πως δεν μπορεί να παίξει εκεί ο πράκτορας, λαμβάνει τιμή $Q[state][action] = -\infty$ για το συγκεκριμένο state, action έτσι ώστε να μην επιλεγεί ποτέ. Αν κανένα από τα 9 κεντρικά κελιά μιας κατάστασης δεν έχει άγνωστο κελί, η κατάσταση προσπερνάται και δεν καταγράφεται στο dictionary. Λαμβάνεται ως action εκείνο που έχει τη μέγιστη τιμή Q ανάμεσα σε όλες τις καταστάσεις. Αφού ληφθεί η ενέργεια, ανανεώνεται το Q για εκείνη την κατάσταση ως εξής:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right)$$

ή για τερματική κατάσταση (εύρεση βόμβας, είτε η επόμενη κατάσταση δεν έχει στα κεντρικά 9 κελιά άγνωστο, είτε νίκη της παρτίδας):

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r - Q(s,a) \right)$$

Όπου:

- α, learning-rate, καθορίζει την επιρροή στην τιμή $Q(s,a)$ από ένα νέο tuple (s, a, r, s'), συγκεκριμένα στον κώδικα τέθηκε $\alpha = .75$
- γ, discount factor, καθορίζει το πόσο θα επηρεάσει την επιλογή του πράκτορα το καλύτερο action στην νέα κατάσταση, συγκεκριμένα τέθηκε $\gamma = .95$
- r, reward, καθορίζει την ανταμοιβή κάθε κίνησης, συγκεκριμένα τέθηκε -5 για ανακάλυψη βόμβας, +1 αλλιώς.

Για να υπάρξει exploration και να μην επιλέξει ο πράκτορας μια συγκεκριμένη τακτική ενώ μπορεί να υπάρξει κάποια καλύτερη, επιλέγει να παίξει τυχαία αν ένας τυχαίος αριθμός είναι μικρότερος του ϵ. Η τιμή του ϵ αρχικά είναι 1.0 και στην πορεία πολλαπλασιάζεται επί κάποιο decay (συγκεκριμένα το έβαλα 0.999975 διότι ο αριθμός των παρτίδων που απαιτεί να παίξει για να πραγματοποιηθεί σύγκλιση είναι μεγάλος) έτσι ώστε να μειώνεται η πιθανότητα τυχαίας επιλογής με την πάροδο των παρτίδων. Σημειώνεται πως ακόμα και για 5×5 καταστάσεις, ο αριθμός κάθε πιθανού συνδυασμού είναι 11^{25}, αρκετά μεγάλος. Παρ΄ όλα αυτά ένα μεγάλο ποσοστό αυτού του αριθμού αποκόβεται εφόσον πολλές καταστάσεις δεν υφίστανται (πχ. 25 άσσοι) και επίσης πολλές αγνοούνται (πχ. αν κανένα από τα κεντρικά 9 κελιά δεν έχει άγνωστο, την αγνοεί).
Το μεγαλύτερο ρίσκο σε αυτή τη λύση ήταν η σύγκλιση στον ρυθμό αύξησης των καταστάσεων ανά παιχνίδι, εφόσον είναι άγνωστος ο πραγματικός αριθμός των πιθανών καταστάσεων που θα κατέγραφε ο πράκτορας στο Q-table. Από περιέργια και μόνο η συγκεκριμένη μορφή πράκτορα εκπαιδεύτηκε στο μεγαλύτερο χρονικό διάστημα της δουλειάς.

5 Λύση 2: Deep Q-Learning

Η χρήση ενός νευρωνικού δικτύου για ένα πρόβλημα ενισχυτικής μάθησης, και πιο συγκεκριμένα για ένα πρόβλημα με υπερβολικά μεγάλο αριθμό καταστάσεων board και κινήσεων, καθώς και σε περιβάλλον που είναι δύσκολο να ληφθούν τα δεδομένα του, παρά μόνο η εικονική παρατήρηση, καθορίζεται ως ιδανική. Πλέον ο πράκτορας δεν διατηρεί ένα dictionary που αντιστοιχεί τιμές για κάθε κλειδί-κατάσταση σε μία λίστα με τιμές Q για κάθε πιθανό action, αλλά δημιουργεί μία αρχιτεκτονική και αποθηκεύει τα βάρη διασυνδέσεων του δικτύου.
Η αρχιτεκτονική αυτή χρησιμοποιεί τεχνικές και layers όπως:

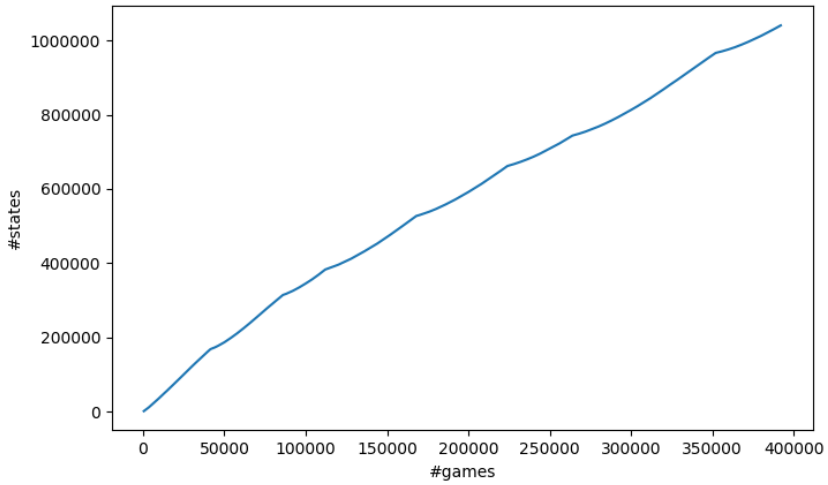
- Convolutional 2D layer που χρησιμοποιείται κυρίως για την συσχέτιση των εικόνων ως είσοδοι

- MaxPooling 2D layer που πραγματοποιεί downscale (οριζόντια και κάθετα) σε μια εικόνα
- Dropout 20% που μηδενίζει τις τιμές του 20% των νευρώνων συγκεκριμένου layer για να αποφευχθεί overfitness
- Flatten, απαραίτητη διαδικασία για την μείωση των dimensions των εικόνων (συνήθως από κάποιες διαστάσεις πχ. λίστα $64 \times 32 \times 32$ που είναι πλήθος εικόνων-batch, μήκος, πλάτος σε 65536 στοιχεία), συνήθως τοποθετείται μετά τα Conv2D layers και πριν τα Dense για την έξοδο του δικτύου.
- Dense layer, κλασσικό βαθύ layer που δημιουργεί κάθε πιθανή διασύνδεση από το προηγούμενο layer με τον επιλεγμένο αριθμό νευρώνων του

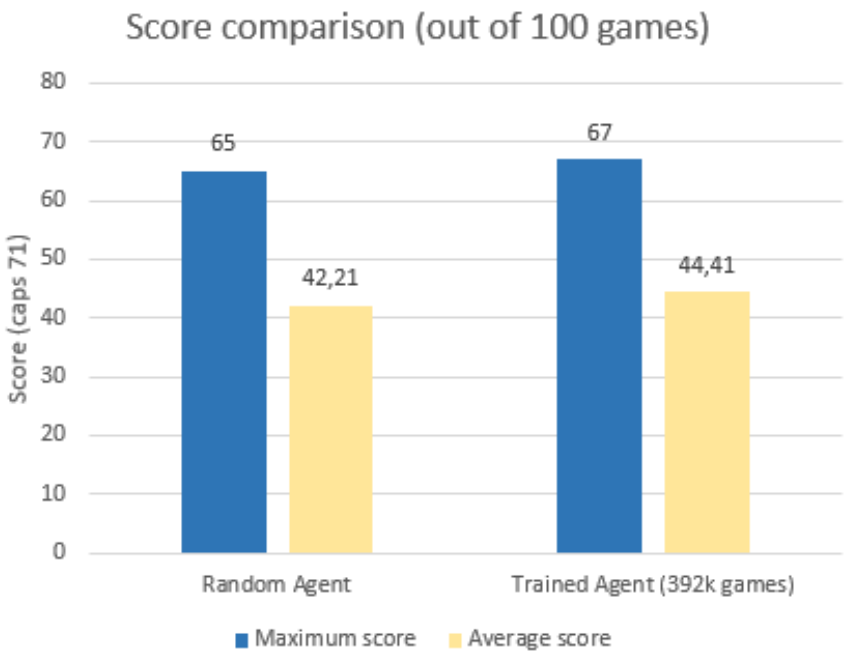
Συγκεκριμένα η είσοδος του νευρωνικού είναι εικόνες διαστάσεων $(9 \cdot 16) \times (9 \cdot 16) \times 1$ (16 pixels ανά κελί), δηλαδή όλο το board, και οι έξοδοι είναι τα Q για κάθε κίνηση δηλαδή 81.
Ο πράκτορας κατά την διάρκεια του παιχνιδιού διατηρεί στην μνήμη transitions, δηλαδή tuples για (s, a, r, s') και επιλέγει σε κάθε κύκλο ένα δείγμα, batch που εμπεριέχει διάφορα transitions και πραγματοποιεί fit στα βάρη του δικτύου σύμφωνα με αυτά. Η μνήμη αυτή δεν απειρίζεται αλλά εφόσον φτάσει κάποια συγκεκριμένη ποσότητα με transitions τα ξεχνάει, για να μην κάνει overfit. Όπως και προηγουμένως υπάρχει η μεταβλητή ϵ και *decay* έτσι ώστε να υπάρχει exploration

6 Αποτελέσματα

Και για τις δύο λύσεις δεν παρουσιάστηκαν ορατά αποτελέσματα καθώς σε κάθε περίπτωση προέκυψαν προβλήματα. Συγκεκριμένα έγραψα πολλές διαφορετικές εκδοχές μέχρι να καταλήξω σε αυτές τις δύο και θεωρώ πως υπάρχει χώρος προς βελτίωση ακόμα.
Q-Agent
Η λύση αυτή είναι καλή διότι προσφέρει αρκετή πληροφορία και δεν καθυστερεί το πρόγραμμα να λάβει αποφάσεις. Άλλες εκδοχές που δοκίμασα είτε κόστιζαν σε καταστάσεις είτε σε χρόνο. Βέβαια το κάθε παιχνίδι λάμβανε 1-3 δευτερόλεπτα. Ο πράκτορας έπαιξε 392.000 παιχνίδια, εξερεύνησε 1.040.292 καταστάσεις και έτρεξε για περίπου 163 ώρες (όχι ακριβές). Ο ρυθμός καταστάσεων ανά παιχνίδι παρέμενε σταθερός καθ' όλη την εκπαίδευση πράγμα που σημαίνει πως ήθελε περισσότερο train.



Σχήμα 3: Γράφημα αριθμού καταστάσεων σε σχέση με τον αριθμό των παιχνιδιών



Σχήμα 4: Γράφημα μέγιστου και μέσου score για 100 παιχνίδια (τυχαίος και εκπαιδευμένος πράκτορας)

Σε αυτή την φάση ο πράκτορας δύσκολα συναντάει καταστάσεις που ήδη γνωρίζει, επομένως είναι σαν να παίζει ακόμα τυχαία. Έτσι το score σαν μετρική δεν έχει νόημα εφόσον δίδει ίδια αποτελέσματα με το να παίζει τυχαία.

Εφόσον εκπαιδευτεί και υπάρχει στην κατοχή μας ένα table με τιμές από πολλά παιχνίδια τότε μπορεί να εφαρμοστεί η ίδια λογική για να παίξει και σε υψηλότερες δυσκολίες χωρίς ιδιαίτερη καθυστέρηση. Συγκεκριμένα ταιριάζει να παίζει σε παρτίδες με ύψος και πλάτος που είναι πολλαπλάσια του 3. Δηλαδή στο 9×9 σε κάθε κίνηση αναλύει 9 καταστάσεις, στο 15×12 αναλύει 20 καταστάσεις, πράγμα που δεν αυξάνει την πολυπλοκότητα σε ορατό βαθμό.

Deep Q-Agent

Η συγκεκριμένη λύση θεωρώ πως είναι η καλύτερη από άποψη προσέγγισης. Όμως περιορίζεται σε συγκεκριμένο $M \times N$ board και δεν μπορεί να εφαρμοστεί σε άλλες διαστάσεις καθώς και είναι πολύ ακριβή όσον αφορά το processing power για την εκπαίδευση του νευρωνικού δικτύου. Ακόμη και με μικρό αριθμό νευρώνων ανά layer, το δίκτυο για κάθε κίνηση και κάθε εικόνα από το batch κάνει update εκατομμύρια βάρη και με το δικό μου μηχάνημα ένα παιχνίδι μπορεί να έπαιρνε και 20-30 δευτερόλεπτα. Η ολοκλήρωση του κώδικα για Deep Reinforcement Learning έγινε πριν τον προηγούμενο (του Q-Learning) και μετά από ένα βράδυ με 8ωρη εκπαίδευση δεσμεύτηκα να προχωρήσω και να ασχοληθώ περισσότερο με τον Q-Agent εφόσον δεν άξιζε για λόγους processing power.

7 Επίλογος και μελλοντική δουλειά

Στην πορεία της δουλειάς αυτής συνειδητοποίησα ποσο εύκολα μπορεί να λυθεί αυτό το πρόβλημα με hardcoded λογική, παρ' όλα συνέχισα την δουλειά για ακαδημαϊκούς λόγους και λόγω περιέργειας. Προφανώς τα αποτελέσματα δεν είναι αρκετά και θεωρώ πως δεν έχει ολοκληρώσει η ασχολία μου με την συγκεκριμένη δουλειά.

- Στο μέλλον θα συνεχίσω να εκπαιδεύω τον Q-Agent για να δω αν θα συγκλίνουν ποτέ οι καταστάσεις ή αν θα συνεχίσουν να αυξάνονται για μεγάλο διάστημα ακόμη.
- Επίσης θα αναζητήσω μήπως μπορώ να χρησιμοποιήσω GPU version της python, διότι ειδικά

για νευρωνικά δίκτυα που απαιτούν τεράστιο αριθμό απλών πράξεων, όπως προσθέσεις και πολλαπλασιασμοί, οι κάρτες γραφικών είναι ιδανικές.

- Επιπλέον θα δοκιμάσω και διαφορετικές εκδοχές του Q-Agent με λιγότερη πληροφορία (3×3 καταστάσεις κτλ) έτσι ώστε να παρατηρηθεί (συντομότερη) σύγκλιση στην αύξηση των καταστάσεων

Βιβλιοθήκες Python 3.7:

- pynput
- time
- datetime
- keyboard
- mss
- random
- numpy
- pickle
- sys
- tensorflow
- keras

Οι δεξιά βιβλιοθήκες χρειάζονται μονάχα για τον DQ-Agent που προτείνω να μην εκτελεστεί διότι δεν είναι εκπαιδευμένος.
Πηγές:

- https://pythonprogramming.net/
- https://www.geeksforgeeks.org/
- https://docs.python.org/3/