Ιωάννης Κάλεσης sdi1900069

Σύγκριση Αλγορίθμων(Cartpole-v1):

- Q-Learning
- Deep Q-Network
- Double Deep Q-Network
- Advantage-Actor-Critic (A2C)
- Asynchronous-Advantage-Actor-Critic(A3C)
- Proximal Policy Optimization(Clipped Surrogate Objective Function)

1. Q-Learning:

Όπως ήταν αναμενόμενο, ο αλγόριθμος Q-Learning μας δίνει καλύτερα αποτελέσματα όσο αυξάνουμε τον αριθμό βημάτων, καθώς και θα έχει σίγουρη σύγκλιση.

Οπότε σε αυτήν την περίπτωση που χούμε Discrete actions, άρα Non continuous-Finite action space και ο αριθμός ζευγαριών καταστάσεων/actions δεν είναι υπερβολικά μεγάλος, είναι χρήσιμος σαν εργαλείο.

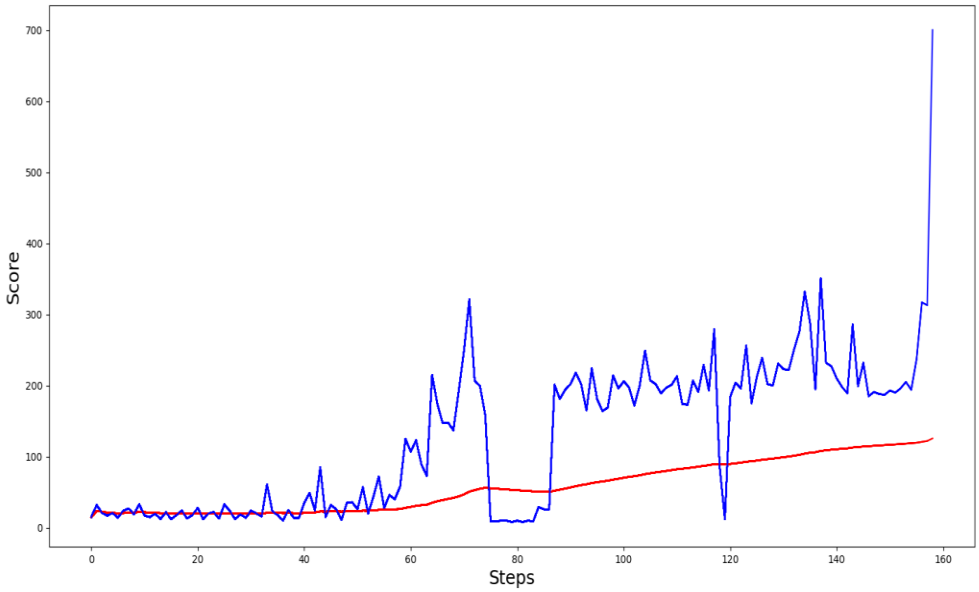
```
Episode: 6000
Mean Reward: 22.276
Episode: 12000
Mean Reward: 22.936333333333334
Episode: 18000
Mean Reward: 29.940833333333334
Episode: 24000
Mean Reward: 49.3145
Episode: 30000
Mean Reward: 77.0305
Episode: 36000
Mean Reward: 110.31766666666667
Episode: 42000
Mean Reward: 144.80366666666666
Episode: 48000
Mean Reward: 173.95833333333334
Episode: 54000
Mean Reward: 191.9035
Episode: 60000
Mean Reward: 179.76316666666668
```

2. DQN

Σε αντίθεση με το Q-Learning, όπου έχει εγγυημένη σύγκλιση, έρχεται το Deep Q Network, το οποίο κάνει χρήση νευρωτικών δικτυών για την προσέγγιση του Q function, και είναι επιρρεπές σε φαινόμενα overfitting εξαιτίας της max().

Δηλαδή, εάν έχουν χτιστεί οι παράμετροι ώστε σε κατάσταση s να μεγιστοποιούμε το reward για την πράξη “δεξιά”,(παρόλο που αυτή η τιμή μπορεί να μην είναι εφικτή), τότε αν στο replay memory εμφανιστεί μια καλή ανταμοιβή για την πράξη “αριστερά”, είναι αρκετά πιθανό αυτή να μην ληφθεί υπόψη μιας και θα έχουμε υπερεκτιμήσει την ανταμοιβή της άλλης πράξης.(Maximisation Bias)

Παρατηρούμε από τα αποτελέσματα, ότι η εφαρμογή του DQN μας δίνει καλά αποτελέσματα, πράγμα που είναι αναμενόμενο εξαιτίας του finite action space

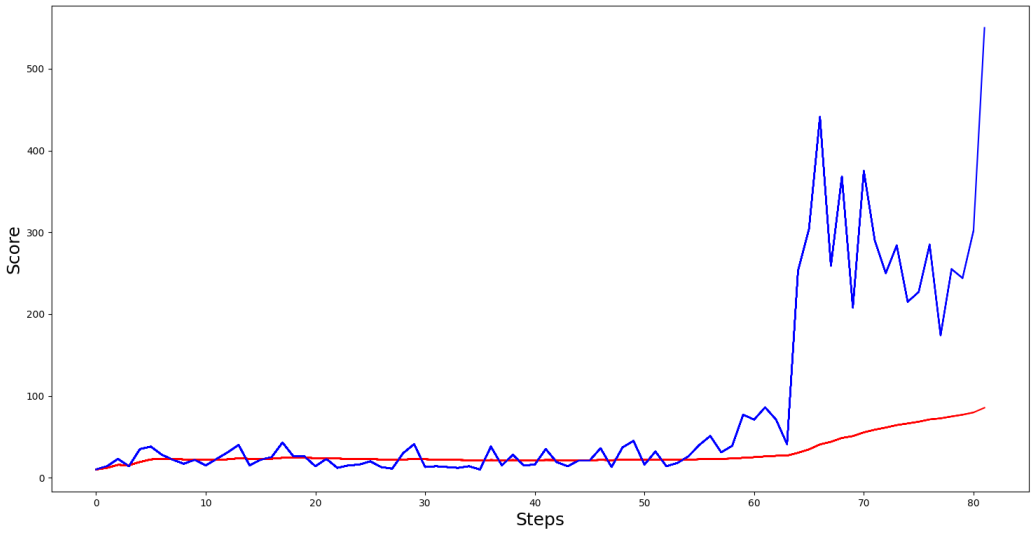


3.Double DQN

Για την επίλυση του ελαττώματος που έχει ο DQN, υλοποιούμε μια παραλλαγή αυτού, οι οποία χρησιμοποιεί δυο νευρωτικά δίκτυα, εκ των οποίων το 1ο λειτουργεί όπως και ο Standard DQN αλγόριθμος, μαθαίνοντας δηλαδή από το replay memory, και το 2ο, το οποίο είναι ένα παγωμένο στιγμιότυπο του 1ου, στο τελευταίο επεισόδιο που εκτελέστηκε.

Οπότε, χρησιμοποιώντας το index της καλύτερης Q-value από το 1ο δίκτυο, διαμορφώνουμε την Q-value μας χρησιμοποιώντας την εξίσωση belman που αντιστοιχεί στο 2ο δίκτυο και βάζοντας ως action αυτήν που επιλέξαμε βάση του 1ου δικτύου.

Πράγματι, ακόμη και σε μικρό αριθμό επαναλήψεων, σχεδόν το μισό αυτών του DQN, βλέπουμε ότι πιάνει τα αποδεκτά αποτελέσματα.



4.A2C

Σε αντίθεση με τους προηγουμένους αλγορίθμους, οπού βρίσκαμε την βέλτιστη πολιτική, υπολογίζοντας μια καλή προσέγγιση της συνάρτησης αξίας, τώρα θα υπολογίσουμε άμεσα την πολιτική, παραμετροποιωντας το π.

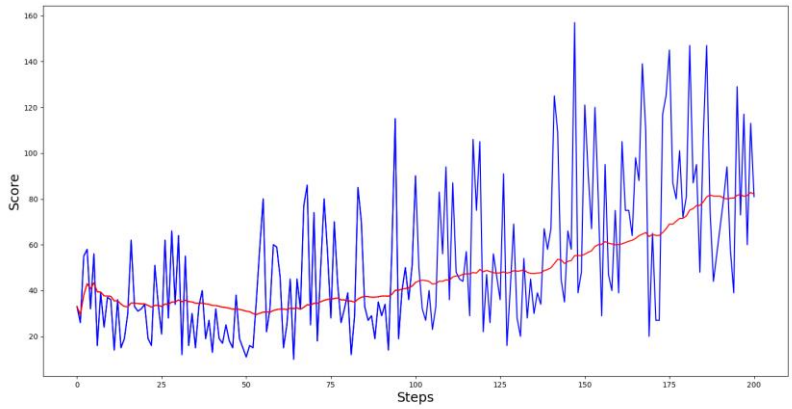
Αυτό είναι ιδιαίτερα χρήσιμο για περιπτώσεις όπου έχουμε άπειρο action space, οπότε και το εγχείρημα του να υπολογίσουμε κάθε state-action pair για να προσεγγίσουμε την value function Q, είναι αδύνατο.

Σε αντίθεση με τους αλγορίθμους που προσεγγίζουν την συνάρτηση αξίας, στους οποίους και μια πολύ μικρή αλλαγή στις τιμές, μπορεί να προκαλέσει μεγάλη αλλαγή στην σύγκλιση, οι policy-based αλγόριθμοι ακολουθούν την το gradient της πολιτικής, οπότε έχουν καλύτερη(όχι γρηγορότερη) σύγκλιση

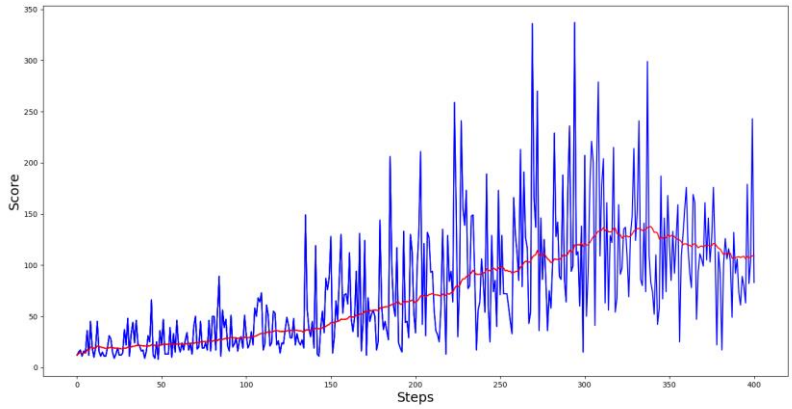
Για να επιταχύνουμε όμως την διαδικασία, καθώς οι policy-based μέθοδοι όπως ο Policy-Gradient, απαιτούν πολύ παραπάνω χρόνο για να συγκλίνουν, θα υλοποιήσουμε τον Advantage-Actor-Critic, οπού έχουμε δυο νευρωτικά δίκτυα, το Actor που λειτουργεί βάση του Policy-Gradient και καθορίζει την πολιτική, και του Critic ο οποίος ακολουθεί την λογική των DQN και προσπαθεί να εκτιμήσει το score που έχει η πολιτική που του δώσει ο Actor.

Όπως και θα περιμέναμε, βλέπουμε ότι ακόμη και σε διπλάσιο αριθμό βημάτων από ότι αυτών του DQN, ο A2C δεν έχει φέρει νίκη.

->Δόκιμη για 200 επεισόδια(learning rate=0.00025):

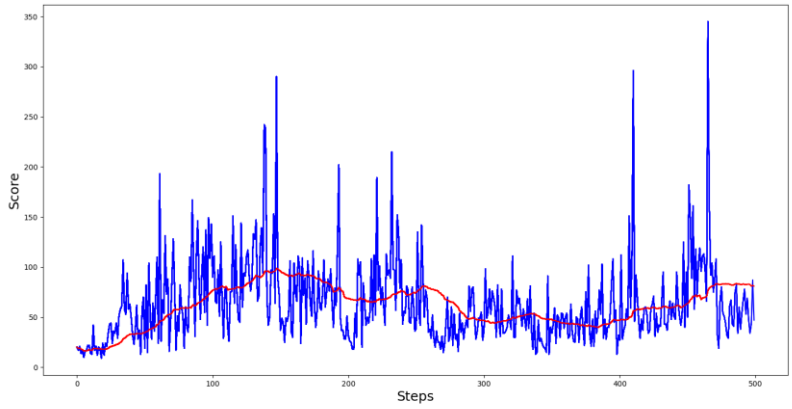


->Δοκιμή για 400 επεισόδια(learning rate=0.00025)



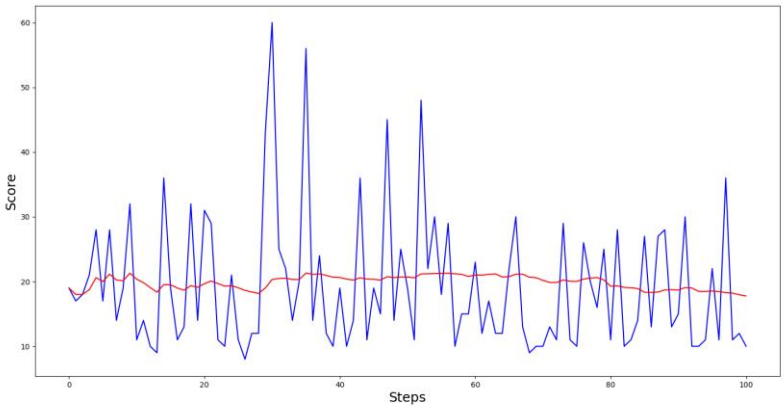
Χρόνος εκπαίδευσης: 30’

->Δοκιμή για 500 επεισόδια(learning rate=0.001):



Χρόνος εκπαίδευσης: 45’

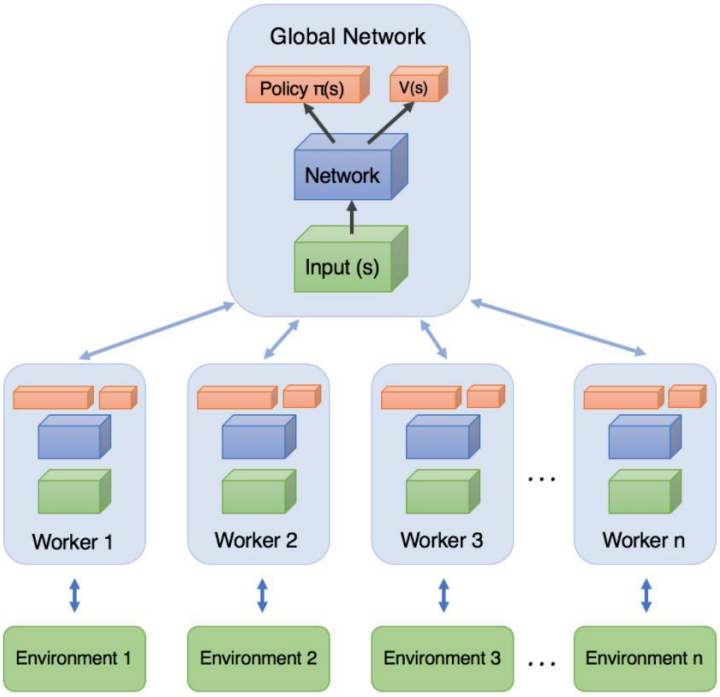
Παρακάτω είναι ένα πείραμα που χρησιμοποιούμε μόνο 1 hidden layer για το δίκτυο μας, πράγμα που κάνει πολύ κακή την επίδοση.



5.A3C

Το Asynchronous μέρος του A3C είναι μια άμεση βελτίωση στον A2C, καθώς χρησιμοποιούμε ταυτόχρονα πολλούς A2C workers οι οποίοι εκπαιδεύονται στο δικό τους local περιβάλλον, γεγονός που μας επιτρέπει όχι μόνο να επιταχύνουμε την ταχύτητα εκπαίδευσης σε συστήματα με δυνατότητα multiprocessing αλλά και παράγει ένα πιο σωστό/αντικειμενικό training set με το οποίο θα εκπαιδεύσουμε τον βασικό A3C agent, καθώς τα training data μεταξύ των πρακτόρων είναι ανεξάρτητα μεταξύ τους.

Στην ουσία αυτό που γίνεται είναι ότι σε κάθε iteration, παίρνουμε τα global parameters του network μας, τρέχουμε κάθε agent στο δικό του περιβάλλον και παίρνουμε τα values και τα losses που υπολογίζονται από την πολιτική, και τα χρησιμοποιούμε για να ανανεώσουμε το global network.



Δυστυχώς, παρόλο την προσπάθεια μου, δεν κατάφερα να υλοποιήσω το threading μέσω tesnorflow το οποίο απαιτείται για να γίνει η ταυτόχρονη εκπαίδευση πολλών A2C πρακτόρων, αλλά θεώρησα ότι άξιζε τουλάχιστον να τον αναφέρω.

6.PPO

Καθώς η βασική ιδέα από τους αλγορίθμους που χρησιμοποιούν policy gradient, είναι να κινούμαστε προς την κατεύθυνση του gradient ascent, προκύπτει το θέμα του step size.

Συγκεκριμένα, μειονέκτημα αλγορίθμων A2C και A3C είναι ότι αν το το step size είναι πολύ μικρό, θα παίρνει πολύ χρόνο η εύρεση ενός (τοπικού)μέγιστου, και από την άλλη αν το step size είναι πολύ μεγάλο, τότε θα έχουμε μεγάλο θέμα στην σύγκλιση καθώς θα διαφέρουν κατά μεγάλο βαθμό οι τιμές που προκύπτουν στην εκπαίδευση.

Ως λύση αυτού του μειονεκτήματος, έρχεται η βασική ιδέα πίσω από τον PPO, η οποία είναι να “φράξουμε” την κάθε ανανέωση που γίνεται στο δίκτυο, έτσι ώστε να μην αλλαχθεί ριζικά η πολιτική, πράγμα που θα προκαλούσε μεγάλο policy update

Αυτό επιτυγχάνεται παίρνοντας τον λόγο των δυο πολίτικων για κάθε action, και λαμβάνοντας τον υπόψη στον μέσο όρο του Loss μόνο αν είναι αναμεσά στο (1-ε,1+ε) οπού ε το clip.

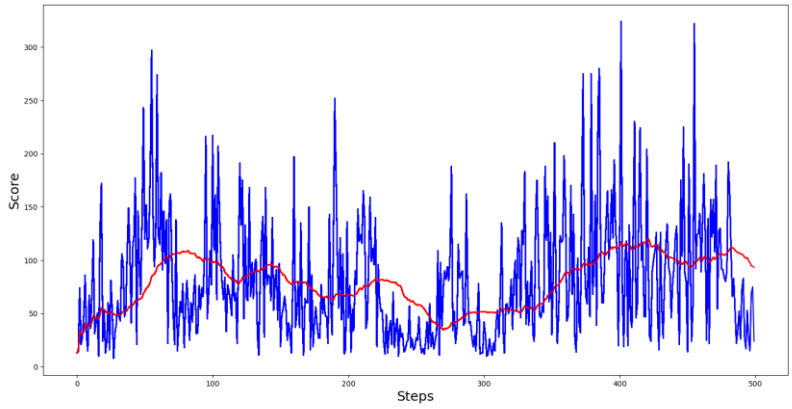
Δηλαδή για rt(θ) τον λόγο διαδοχικών πολίτικων, μετατρέπουμε το loss function από:

$$L_t(\theta) = r_t(\theta)\hat{A}_t$$

σε:

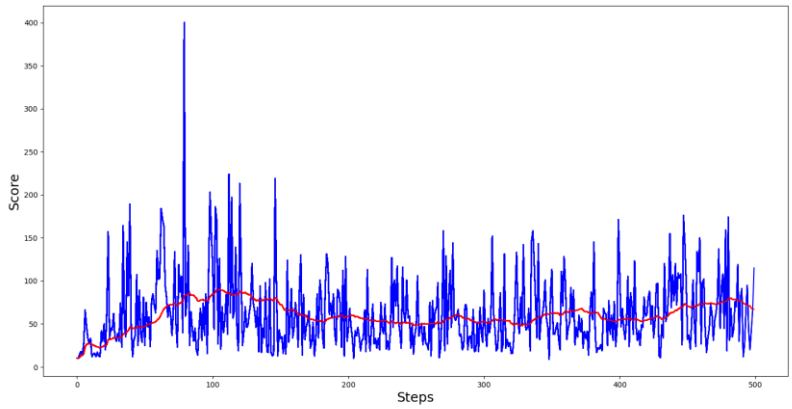
$$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$$

->Δοκιμή για 500 επεισόδια(learning rate=0.00025 και clip=0.2):



Χρόνος εκπαίδευσης: 75’

->Δοκιμή για 500 επεισόδια(learning rate=0.001 και clip=0.1):



Χρόνος εκπαίδευσης: 65’

Στο παραπάνω πείραμα διαπιστώνουμε ιδιαιτερότητα μεταξύ του learning rate και του clip.

Άμα μειώναμε το clip, δηλαδή περιορίζαμε το κατά ποσό μπορεί να αλλάξει η πολιτική, και αυξάναμε το learning rate του μοντέλου, δηλαδή τον βαθμό ανανέωσης των βαρών του μοντέλου ώστε να μαθαίνει πιο γρήγορα, θα περίμενε κάνεις ότι θα χαμε παρόμοια αποτελέσματα αν όχι και καλυτέρα σε σχέση με την αντιστροφή περίπτωση.

Η διαίσθηση αυτή προέρχεται από το γεγονός ότι θα κάναμε πολύ μεγάλα βήματα(υψηλό learning rate) μόνο σε περιπτώσεις οπού αυτό δεν οδηγεί σε μεγάλη αλλαγή πολιτικής(χαμηλό clip), οπότε θα λέγαμε ότι με το που βρει μια πολιτική οπού του δίνει καλά rewards με μικρή αλλαγή πολιτικής, τότε αυτό θα μάθαινε πολύ γρήγορα τα βάρη που πρέπει να ακολουθήσει για να την πιάσει.

Παρόλο αυτά, βλέπουμε ότι σε σχέση με το 1ο διάγραμμα, σχεδόν όλες οι δοκιμές του 2ου δεν οδηγούν σε καλή πολιτική, ενώ έχουμε πολλές περισσότερες δοκιμές με καλά rewards στο 1ο διάγραμμα, γεγονός που φαίνεται και από το υψηλότερο average αλλά και από την πυκνότητα των rewards που ναι πάνω από 150-200.

Συνάμα, το clip δεν μας δίνει την δυνατότητα να ισοβαθμήσουμε την αύξηση του learning rate με μια μείωση του clip, έτσι ώστε να επιταχυνόταν η ταχύτητα της μάθησης