Εθνικο Μετσοβιο Πολυτεχνειο

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Νευρωνικά Δίκτυα και Ευφυή Υπολογιστικά Συστήματα

4η Εργαστηριακή Άσκηση

Αχλάτης Στέφανος-Σταμάτης (03116149)

<el16149@mail.ntua.gr>

Ηλιακοπούλου Νικολέτα-Μαρκέλα (03116111)

<el116111@mail.ntua.gr>

Σταυροπούλου Γεωργία (03116162)

<el116162@mail.ntua.gr>

Μάρτιος 2021

Γενική Εισαγωγή

Το Reinforcement Learning (RL) επανήλθε στο ερευνητικό προσκήνιο τα τελευταία χρόνια καθώς κατάφερε να φτάσει και πολλές φορές να ξεπεράσει την ανθρώπινη επίδοση σε πολλά προβλήματα. Χαρακτηριστικά παραδείγματα είναι η επίδοση του RL στα Atari 2600 games (Mnih et al., 2015), στην νίκη του απέναντι στον πρωταθλητή του Go (Silver et al., 2017) και η επίδοση του παίζοντας ανταγωνιστικά DOTA για 5 χρήστες (OpenAI, 2018b). Η ιδέα των πρώτον αυτών εφαρμογών βασίστηκε στο stacking ενός συγκεκριμένου αριθμού απο fixed consecutive frames για να αντιμετωπίσουν την μερική παρατηρησιμότητα στο Atari 2600 games. Ωστόσο, με την πρόοδο σε όλο και δυσκολότερα προβλήματα υπάρχει η ανάγκη για πιο προηγμένα memory-based αναπαραστάσεις, που οδήγησε στην ερευνητική κοινότητα να στραφει στα Recurrent Neural Networks (RNNs) και στις παραλλαγές τους όπως τα LSTMs οι οποίες μέχρι σήμερα είναι οι state-of-the-art στον χώρο.

Σε αυτή την άσκηση θα μελετήσουμε την συμπεριφορά ορισμένων απλών και κλασικών αλγορίθμων RL πάνω σε ένα περιβάλλον των Atari 2600 games, το Asterix(Taz). Πιο συγκεκριμένα θα μελετήσουμε αλγορίθμους στους εξείς τρεις χώρους: value optimization, policy optimization και actor critic optimization με στόχο να βρούμε το καλύτερο αλγόριθμό για την επίλυση του συγκεκριμένου προβλήματος στο συγκεκριμένο περιβάλλον. Σημαντικό σημείο της εργασίας μας είναι η πεπερασμένη χρήση Accelerator (GPU) και το γεγονός ότι αυτά τα προβλήματα απαιτούν πολύ μεγάλη υπολογιστική ισχύς για να μελετηθούν σωστά.

Περιβάλλον Μελέτης

Παρακάτω δίνουμε την αναφορά του RetroGames για το Asterix Atari 2600 games: Asterix (Taz) was released by Atari in 1983 for the Atari 2600 and features the Looney Tunes character the Tasmanian Devil in a food frenzy. Within the game, Taz only appears as a tornado. The same game was released outside the United States featuring Asterix instead of Taz. The gameplay is rather simple. The player guides Taz between the stage lines in order to eat hamburgers and avoid the dynamites. The game does not use any buttons and the difficulty increases by increasing the speed of the objects on screen. As the game progresses, the burgers may change into other edible or drinkable objects such as beer kegs, hot dogs, etc. There are not many sound effects in the game except a blipping sound when the player hits an edible object and another sound that resembles an explosion when the player hits dynamite.

Μετά μπορούμε να έχουμε υποκατηγορίες περιβαλλοντων καθώς έχουμε δύο παράγοντες που επηρεάζουν το περιβάλλον, το Frame Skip και το Sticky Actions ή Repeat action probability. Το Frame Skip καθορίζει πόσα frames διαρκεί η κάθε ενέργεια του πράκτορα. Αν έχει “Deterministic” στο όνομα, η ενέργεια είναι η ίδια για 4 frames. Αν έχει NoFrameskip στο όνομα, τότε η ενέργεια αποφασίζεται για κάθε frame ξεχωριστά. Αν δεν έχει τίποτε από τα δύο, τότε η διάρκεια της κάθε ενέργειας αποφασίζεται στοχαστικά και έχει διάρκεια 2 με 4 frames.

Το σύστημα Atari δεν έχει εγγενώς πηγές εντροπίας και είναι ντετερμινιστικό. Συνεπώς, αν ξεκινήσεις από το ίδιο φύτρο (seed) της γεννήτριας ψευδοτυχαίων και κάνεις τις ίδιες ενέργειες παίρνεις πάντα τα ίδια αποτελέσματα. Αυτό είναι πλεονέκτημα για αλγόριθμους χωρίς μάθηση που βασίζονται στη μνήμη. Ταυτόχρονα, δεν μας δίνει την μικρή στοχαστικότητα που θέλουμε στην ενισχυτική μάθηση. Ένας τρόπος που μπορούμε να προσθέσουμε στοχαστικότητα είναι με τα sticky actions: ο πράκτορας στην επόμενη κίνηση δεν θα κάνει απαραίτητα αυτό που έχει αποφασίσει αλλά με μια μικρή πιθανότητα p θα κάνει την προηγούμενη ενέργειά του (κολλημένη ενέργεια, sticky action). Αν το όνομα του περιβάλλοντος έχει v0 υπάρχει η δυνατότητα sticky action με πιθανότητα 0.25. Αν το όνομα έχει v4, δεν έχουμε sticky actions.

Random Agent

Αρχικά μελετάμε την βασική περίπτωση που ο παίκτης παίζει εντελώς τυχαία χωρίς να έχει καμία απολύτως γνώση για το περιβάλλον μελέτης. Επομένως δεν έχει προυπάρξει κάποια μορφή εκπαίδευσης και το score αυτού θα έχουμε ως μέτρο σύγκρισης για να δούμε "πόσο έμαθε" το μοντέλο κατα την διάρκεια της εκπαίδευσης. Προφανώς στην θεωρητική περίπτωση που ένα μοντέλο έχει επίδοση χαμηλότερη της τυχαίας έχουμε ότι η εκπαίδευση έβαψε το agent

Το αποτέλεσμα είναι το εξής:

```
Eval reward: 175.0 (+/-33.54101966249684)
```

Μελέτη Απόδοσης ενός τυπικού χρήστη

Στην συνέχεια παίξαμε εμείς οι ίδιοι το παιχνίδι online στο free80sarcade και πετύχαμε score: 23,200. Το game play μπορείτε να το δείτε εδώ και να το συγκρίνεται με το αντίστοιχο game play που θα μάθει ο RL agent. Η τιμή αυτή είναι ένα score που θέλουμε μέσω της εκπαίδευσης να το φτάσουμε και να το ξεπεράσουμε για να μπορέσουμε να έχουμε ένα σύστημα το οποίο παίζει εξίσου καλά ή και καλύτερα από έναν μέσο άνθρωπο.

Μελέτη Απόδοσης του καλύτερου ανθρώπινου χρήστη

Μελετώντας την βιβλιογραφία γι' αυτό το πρόβλημα μπορουμε να δούμε ότι ο καλύτερος παίκτης έχει καταφέρει score ίσο με: 1,000,000.

Μελέτη Απόδοσης του καλύτερου artificial agent

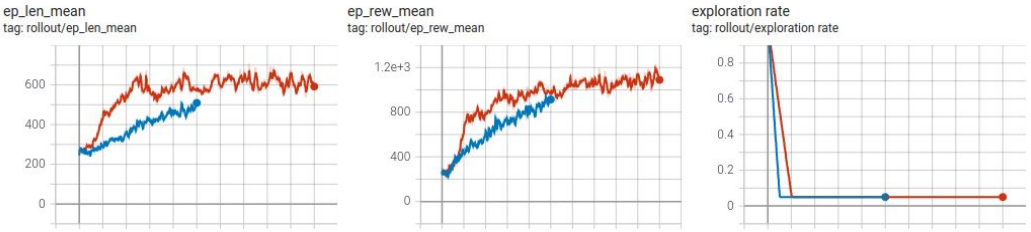
Μελετώντας την βιβλιογραφία γι' αυτό το πρόβλημα μπορουμε να δούμε ότι ο καλύτερος artificial agent έχει καταφέρει score ίσο με: 428,200. Το μηχάνημα που πέτυχε αυτό το score λέγεται Rainbow και η έρευνα αυτή δημοσιεύτηκε τον Οκτώβριο του 2017, δηλαδή φαίνεται σαν η ερευνητική κοινότητα να μην έχει να προτείνει κάτι καλύτερο από τότε.

Αξίζει να σημειωθεί ότι το καλύτερο score σε αυτό το παιχνίδι το πετυχαίνει **άνθρωπος** και όχι μηχανή.

Αρχικοποίηση Μοντέλου: Μελέτη πλήθους Timestep και Policy

Σε αυτό το σημείο θέλουμε να βρούμε το ιδανικό πλήθος timestep για την επίλυση του προβλήματος μας και καλούμαστε να λύσουμε το εξής tradeoff: η συντομη εκπαίδευση είναι γρήγορη αλλά κατα πάσα πιθανότητα όχι τόσο αποδοτική ενώ η πιο μακροχρόνια εκπαίδευση είναι πιο αργή αλλά κατά πάσα πιθανότητα πιο αποδοτική. Γι' αυτό τον λόγο τρέξαμε δυο πειράματα στο περιβάλλον AsterixDeterministic-v4 με 1.000.000 και με 2.000.000 timestep με Deep Q-learning με MlpPolicy και default παραμέτρους.

Τα αποτελέσματα στο TensorBoard είναι τα εξείς:



Τα αποτελέσματα στο Testing είναι τα εξής για το πρώτο και το δεύτερο πείραμα αντίστοιχα:

```
Eval reward: 1025.0 (+/-186.07794065928394)
```

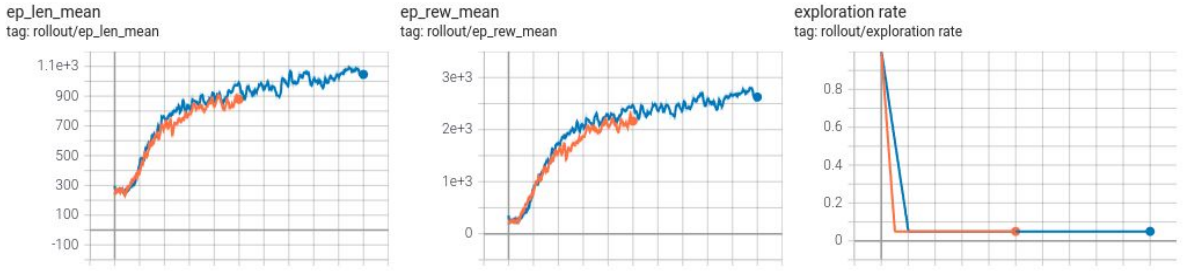
```
Eval reward: 1540.0 (+/-602.8266749240614)
```

Οι χρόνοι που χρειαστηκαν για να ολοκληρωθούν οι εκπαίδευσης του κάθε πειράματος είναι οι εξης:

- Το πείραμα με 1.000.000 timestep χρειάστηκε 1h 36m 57s
- Το πείραμα με 2.000.000 timestep χρειάστηκε 3h 4m 20s

Επειδή όμως το περιβάλλον μας έχει να κάνει σχέση με εικόνες είναι θεμιτό να πειραματιστούμε επί της αρχής με CnnPolicy, καθώς δίνουν καλύτερα αποτελέσματα σε εικόνες σε σχέση με το MlpPolicy. Γι' αυτό τον λόγο τρέξαμε δυο πειράματα στο περιβάλλον AsterixDeterministic-v4 με 1.000.000 και με 2.000.000 timestep με Deep Q-learning με το CnnPolicy και default παραμέτρους.

Τα αποτελέσματα στο TensorBoard είναι τα εξείς:



Τα αποτελέσματα στο Testing είναι τα εξής για το πρώτο και το δεύτερο πείραμα αντίστοιχα:

```
Eval reward: 2465.0 (+/-553.1952638987431)
```

```
Eval reward: 3090.0 (+/-816.639455328972)
```

Οι χρόνοι που χρειαστηκαν για να ολοκληρωθούν οι εκπαίδευσης του κάθε πειράματος είναι οι εξης:

- Το πείραμα με 1.000.000 timestep χρειάστηκε 1h 41m 2s
- Το πείραμα με 2.000.000 timestep χρειάστηκε 3h 21m 41s

Έτσι, αξίζει να παρατηρήσουμε ότι το CnnPolicy αποδίδει πολύ καλύτερα για το δεδομένο περιβάλλον. Ωστόσο, παρατηρούμε ότι η τελική τιμή στο tests έχει αρκετά μεγαλύτερη διασπορά σε σχέση με την αντίστοιχη διασπορά του MlpPolicy, άρα η CnnPolicy δίνει λιγότερο προβλέψιμα αποτελέσματα αλλά ακόμη και το χειρότερο αποτέλεσμα της είναι καλύτερο από το καλύτερο

αποτέλεσμα της MlpPolicy. Έτσι, επιλέγουμε να συνεχίσουμε την μελέτη μας με χρήση της CnnPolicy.

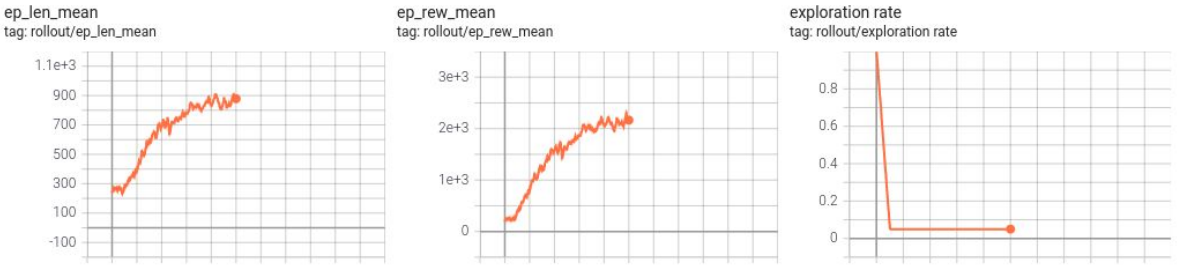
Όσον αφορά το timestep,είναι γνωστό ότι οι RL αλγόριθμοι θέλουν αρκετό χρόνο εκπαίδευσης επομένως γνωρίζουμε a priori ότι και τα δυο timesteps δεν είναι αρκετά. Ωστόσο για το CnnPolicy βλέπουμε ότι και τα δύο μοντέλα ξεπερνούν την "μεταβατική κατάσταση" και υπάρχει μια μερική ισορροπία στην συνάρτηση μετρική της εκπαίδευσης, δηλαδή η παράγωγος της είναι όλο και μικρότερη με το πέρας του χρόνου και αυτό δηλώνει ότι το μοντέλο έχει αρχίσει και μαθαίνει, σίγουρα όχι αρκετά. Επίσης ένα ακόμη επιχείρημα για το γεγονός ότι ο πράκτορας μαθαίνει είναι το γεγονός ότι πετυχαίνει δεκαπλάσια με εικοσαπλάσιο καλύτερη επίδοση από ότι ο random agent. Συγκρίνοντας τα δύο μοντέλα επιλέγουμε να συνεχίσουμε την μελέτη μας με 1.000.000 timesteps καθώς ο λόγος κέρδους προς κόστος για αυτή την τιμή είναι μεγαλύτερη σε σχέση με τα 2.000.000 timesteps.

Στο επόμενο στάδιο των πειραμάτων θα μελετήσουμε τρία διαφορετικά περιβάλλοντα τα: AsterixDeterministic-v4, AsterixNoFrameskip-v4, Asterix-v4 και τέσσερις διαφορετικούς αλγορίθμους τους DQN,A2C,QR-DQN,PPO, άρα συνολικά θα τρέξουμε 12 πειράματα.

Πειράματα στο περιβαλλον **AsterixDeterministic-v4**:

DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

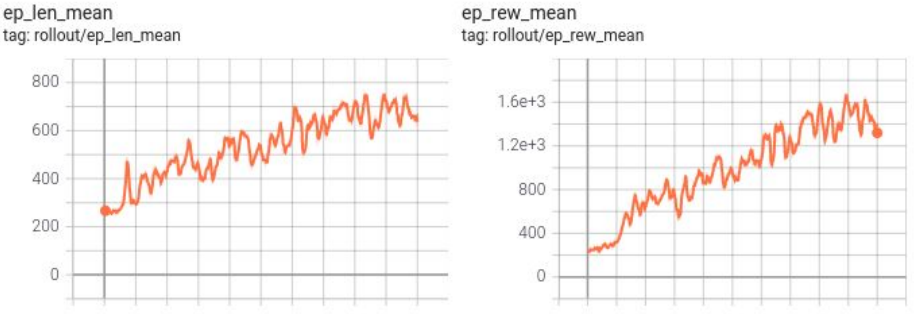


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2465.0 (+/-553.1952638987431)
```

A2C

Τα αποτελέσματα στο TensorBoard είναι το εξης:

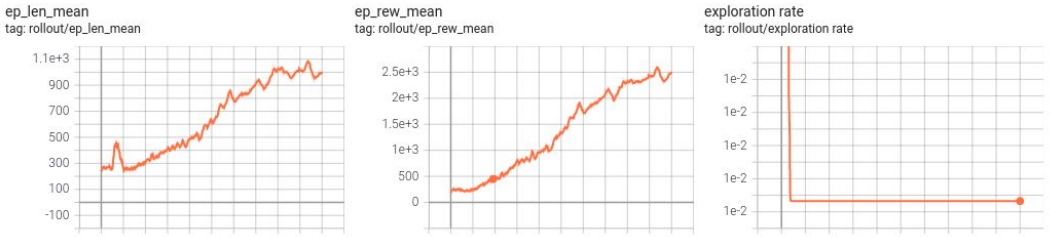


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 1590.0 (+/-528.5830114561004)
```

QR-DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

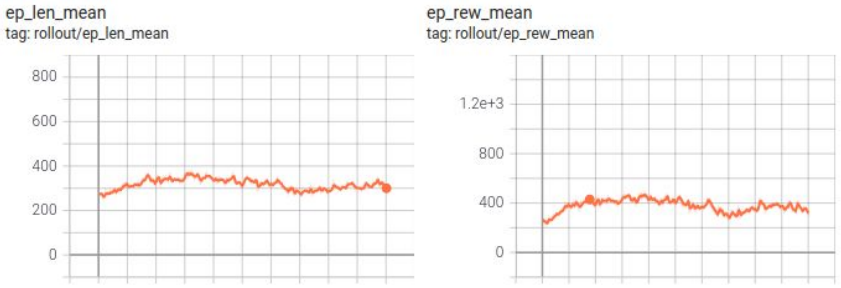


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2495.0 (+/-656.6772418776213)
```

PPO

Τα αποτελέσματα στο TensorBoard είναι το εξης:



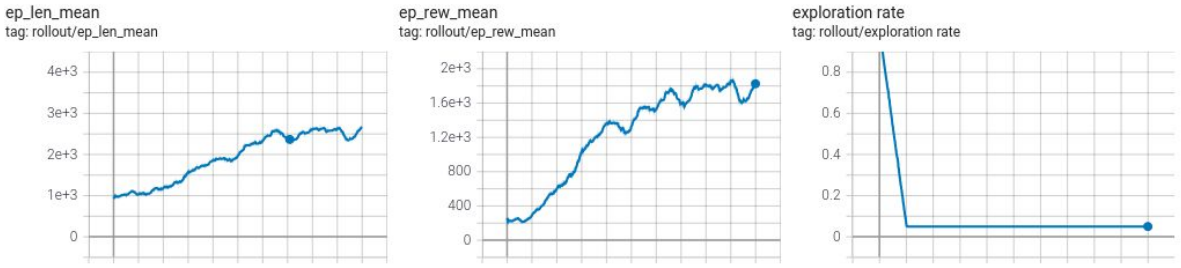
Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 320.0 (+/-110.0)
```

Πειράματα στο περιβαλλον **AsterixNoFrameskip-v4**:

DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

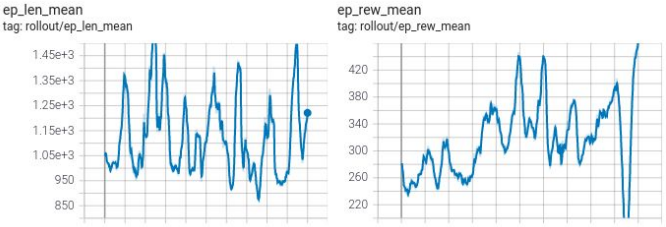


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2130.0 (+/-699.7142273814361)
```

A2C

Τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 480.0 (+/-249.19871588754225)
```

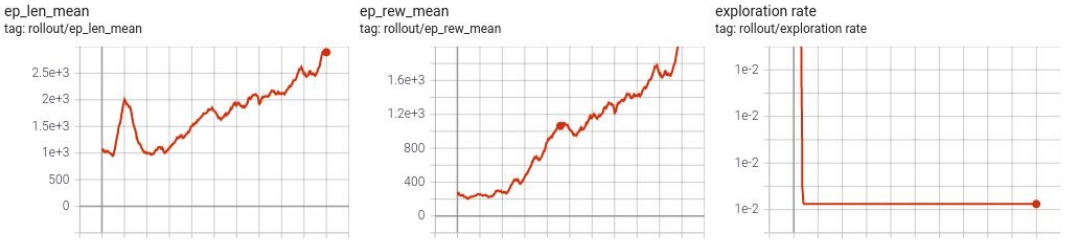
Η μη ομαλή συμπεριφορά της καμπύλης καθώς και η τόσο χαμηλή επίδοση δείχνει ότι ο πράκτορας δεν έμαθε αφού το αποτέλεσμα είναι συγκρίσιμο με το αποτέλεσμα του random agent. Αυτό ώθησε να επαναλάβουμε το αντίστοιχο πείραμα και πετύχαμε την εξής , εντυπωσιακά διαφορετική, επίδοση:

```
Eval reward: 1110.0 (+/-37.416573867739416)
```

Με αυτό μπορούμε να συμπεράνουμε ότι ο A2C είναι αρκετά στοχαστικός σαν αλγόριθμος.

QR-DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

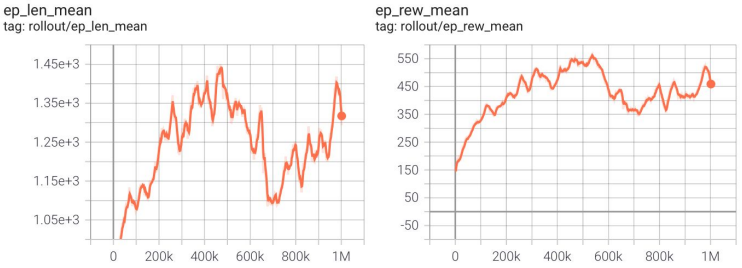


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2120.0 (+/-918.4770002564027)
```

PPO

Τα αποτελέσματα στο TensorBoard είναι το εξης:



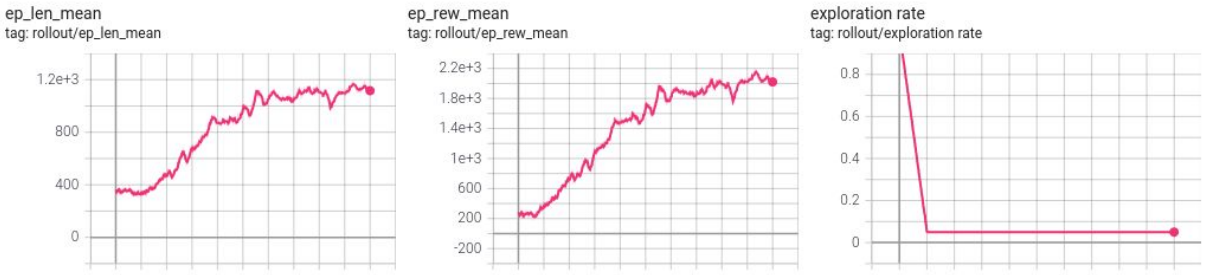
Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 485.0 (+/-167.40669042783207)
```

Πειράματα στο περιβαλλον **Asterix-v4**:

DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

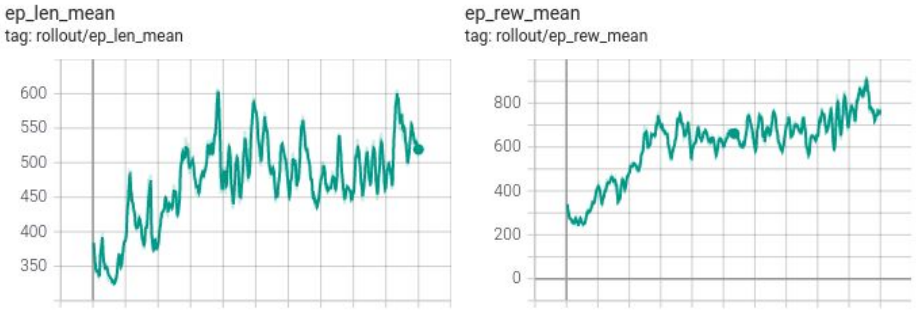


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2535.0 (+/-714.8601261785413)
```

A2C

Τα αποτελέσματα στο TensorBoard είναι το εξης:

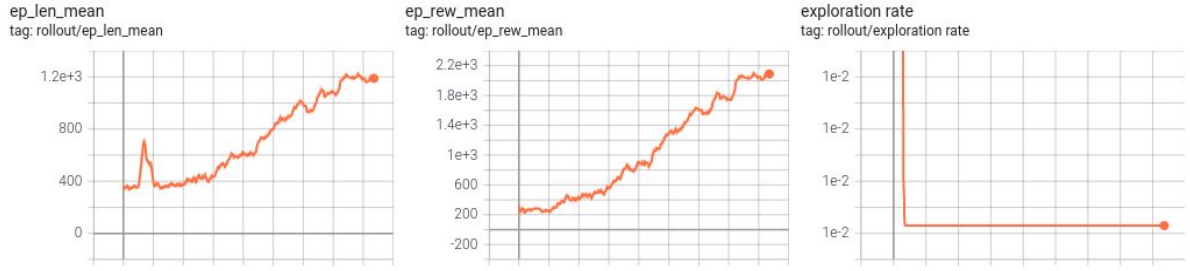


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 1105.0 (+/-346.01300553591915)
```

QR-DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

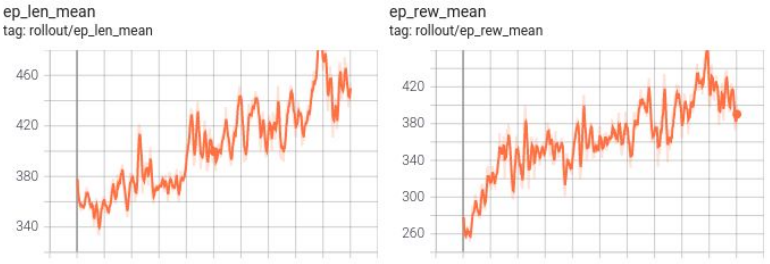


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2615.0 (+/-563.0497313737038)
```

PPO

Τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 515.0 (+/-196.27786426390523)
```

Συμπεράσματα

Από τα πειράματα που λάμβανε χώρα με συνολική διάρκεια περίπου 24 ωρών μπορούμε να βγάλουμε τα παρακάτω συμπεράσματα. Αρχικά βλέπουμε ότι τα πειράματα που έλαβαν χώρα στο NoFrameskip περιβάλλον απέδωσαν χειρότερα από τα πειράματα στο απλό και στο Deterministic περιβάλλον, με τα δύο τελευταία περιβάλλοντα να πετυχαίνουν παρόμοιες επιδόσεις με ένα μικρό προβάδισμα στο απλό περιβάλλον. Όσον αφορά τους αλγορίθμους βλέπουμε ότι ο ppo πετυχαίνει την χειρότερη απόδοση σε όλα τα περιβάλλοντα και αμέσως μετά ο a2c ενώ οι άλλοι δύο επιτυγχάνουν σχεδόν παρόμοιες επιδόσης αν και ο Quantile Regression Deep Q-Network φαίνεται από τις γραφικές να έχει την δυναμική να "μάθει" ακόμη καλύτερα αν γίνει εκπαίδευση για μεγαλύτερο χρονικό διάστημα. Αξίζει να σημειωθεί ότι ο ppo έχει επιδόσεις συγκρίσιμες με τον random agent επομένως μπορούμε να θεωρήσουμε ότι δεν εκπαιδεύεται.

Έτσι, θα κρατήσουμε το απλό περιβάλλον και τους αλγορίθμους Deep Q-learning, Quantile Regression Deep Q-Network.

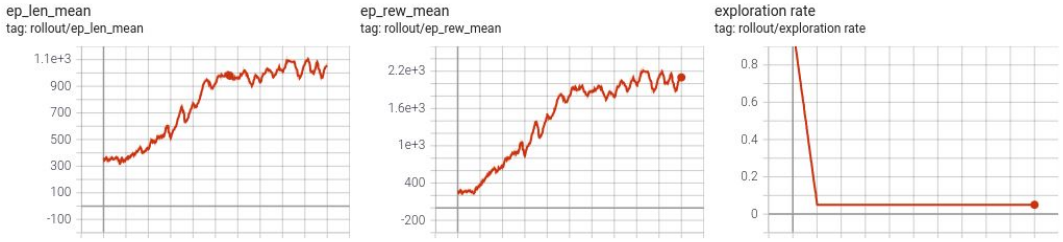
Στην συνέχεια θα κάνουμε hyperparameter optimization για τους καλύτερους αλγορίθμους στο καλύτερο περιβάλλον. Ένα σύγχρονο framework που χρησιμοποιείται για το fine tuning είναι το Optuna: A hyperparameter optimization framework το οποίο βασίζεται στο εξής paper.

Στην παρούσα εργασία θα μελετήσουμε κάποιες βασικές παραμέτρους όπως το γ και το batch size, το tau και το learning rate για να δούμε πόσο επηρεάζουν την επίδοση του πράκτορα.

Εφαρμόζοντας το γ=0.95

DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

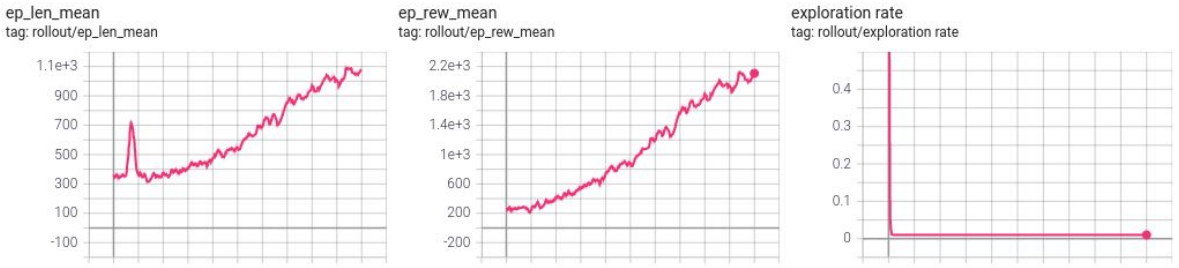


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2600.0 (+/-426.6145801540308)
```

QR-DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2530.0 (+/-417.85164831552356)
```

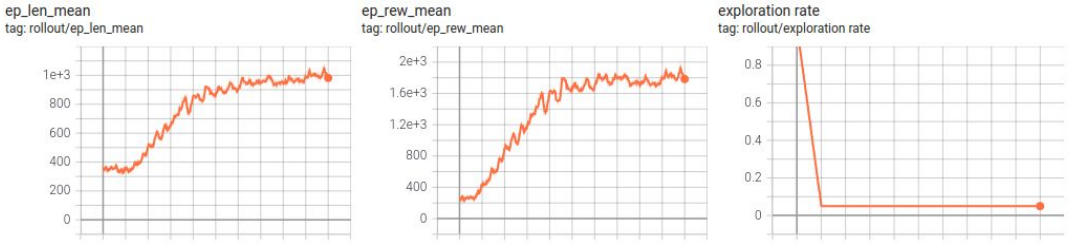
Σχολιασμός

Σχετικά με την παράμετρο γ παρατηρούμε ότι η μετάβαση από γ=0.99 σε γ=0.95 για τις ίδιες εποχές εκπαίδευσης δεν άλλαξε ιδιαίτερα την μέση απόδοση των αντίστοιχων πειραμάτων. Ωστόσο μείωσε την διακύμανση της απόδοσης, μια ιδιότητα επιθυμητή, αφού έτσι το αποτέλεσμα είναι πιο προβλέψιμο. Επομένως **κρατάμε το γ=0.95** σε όλα τα υπόλοιπα πειράματα.

Εφαρμόζοντας το batch_size = 64

DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

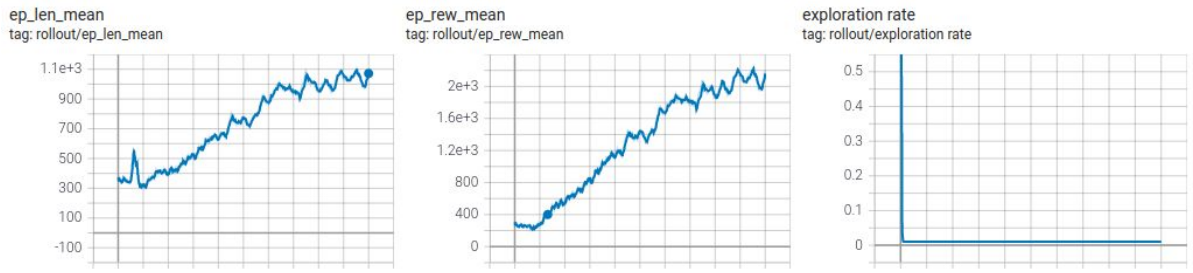


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 1940.0 (+/-654.9045732013176)
```

Quantile Regression Deep Q-Network (QR-DQN):

Τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2235.0 (+/-567.4724662924184)
```

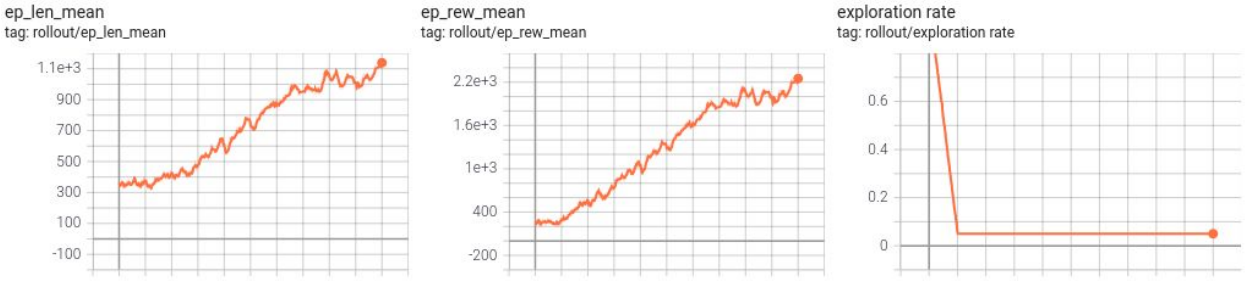
Σχολιασμός

Επομένως η αύξηση του batch size επιδρά αρνητικά στην επίδοση του πράκτορα. Για αυτό τον λόγο θα κάνουμε μια ακόμη σειρά πειραμάτων με μικρότερο batch size, μια λογική τιμή πειραματισμού είναι το batch_size =16

Εφαρμόζοντας το batch_size = 16

DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:

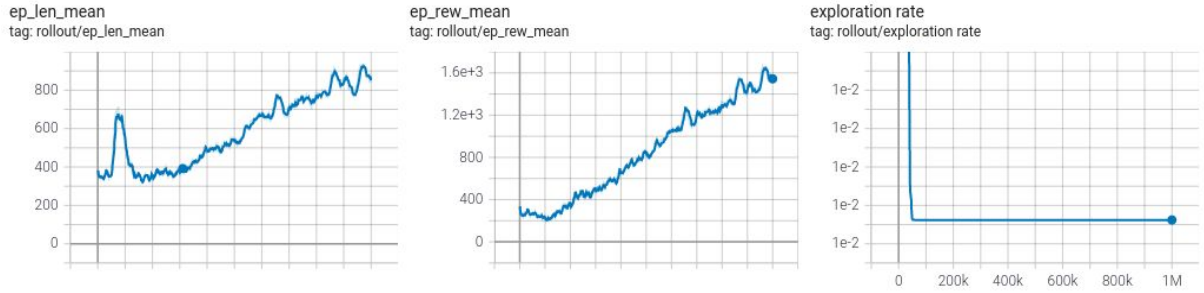


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2870.0 (+/-310.0)
```

QR-DQN

Τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 1805.0 (+/-457.9574216016157)
```

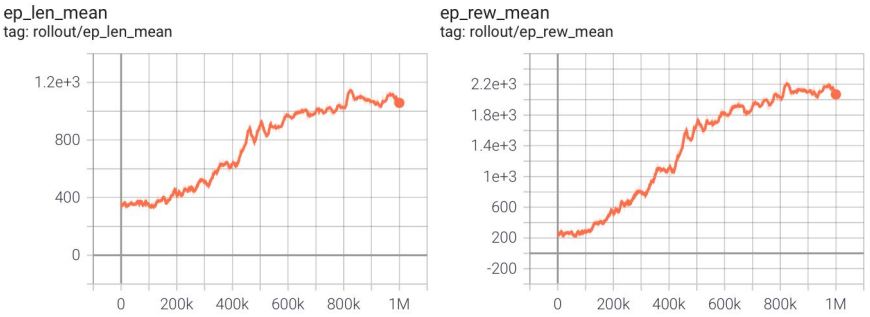
Σχολιασμός

Επομένως βλέπουμε ότι οι δύο διαφορετικοί αλγόριθμοι έχουν διαφορετική επίδοση ανάλογα με το batch size που ορίζεται. Πιο συγκεκριμένα ο dqn αποδίδει καλύτερα με batch size ίσο με 16 ενώ ο qrdqn με batch size ίσο με 32.

Εφαρμόζοντας το tau = 0.95

DQN (με γ=0.95, batch_size=16)

Τα αποτελέσματα στο TensorBoard είναι το εξης:

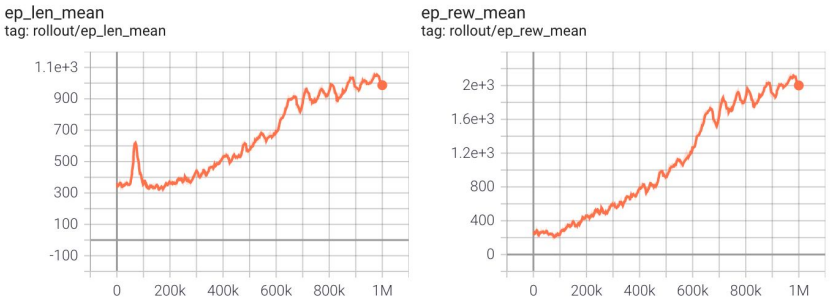


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2435.0 (+/-791.5333221033717)
```

QR-DQN (με γ=0.95, batch_size=32):

Τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2170.0 (+/-741.3501197140256)
```

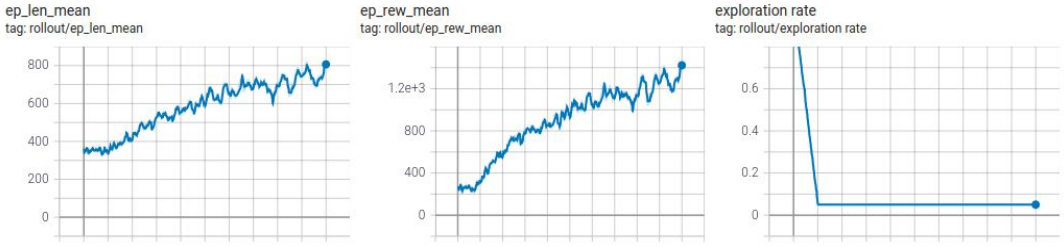
Σχολιασμός

Επομένως συμπερένουμε ότι η υπερπαράμτερος tau = 0.95 συμβάλει αρνητικά επομένως αφήνουμε το tau = 1 στην default τιμή του.

Εφαρμόζοντας το learning_rate = 0.0005

DQN(με γ=0.95, batch_size=16)

Τα αποτελέσματα στο TensorBoard είναι το εξης:

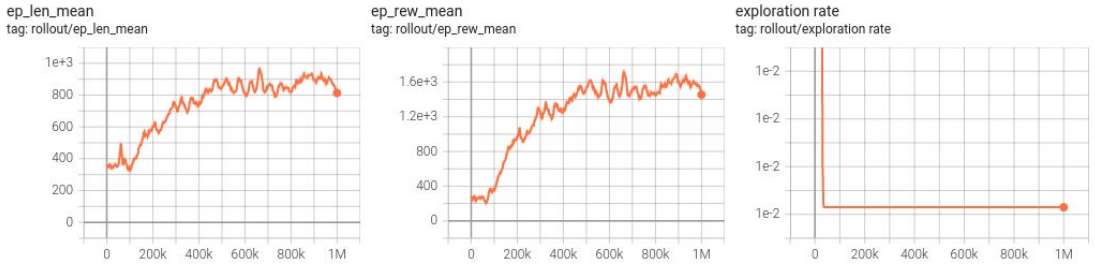


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 1870.0 (+/-451.7742799230607)
```

QR-DQN(με γ=0.95, batch_size=32):

Τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 1440.0 (+/-552.6300751859239)
```

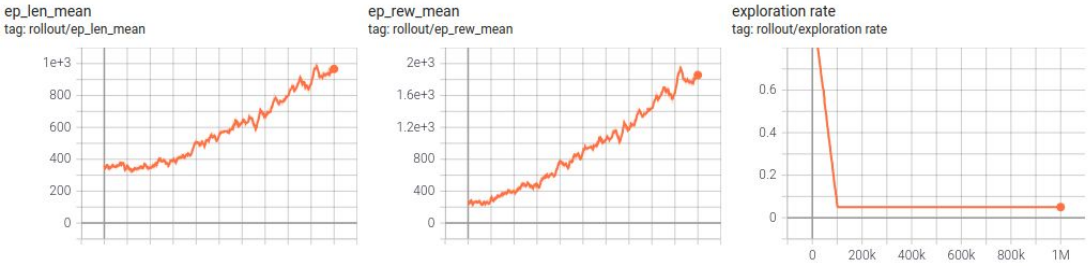
Σχολιασμός

Επομένως συμπερένουμε ότι η υπερπαράμτερος learning_rate = 0.0005 συμβάλει αρνητικά και στην συνέχεια θα μελετήσουμε την συμπεριφορά των πρακτόρων για learning_rate=0.00005.

Εφαρμόζοντας το learning_rate=0.00005

DQN (με γ=0.95, batch_size=16)

Τα αποτελέσματα στο TensorBoard είναι το εξης:

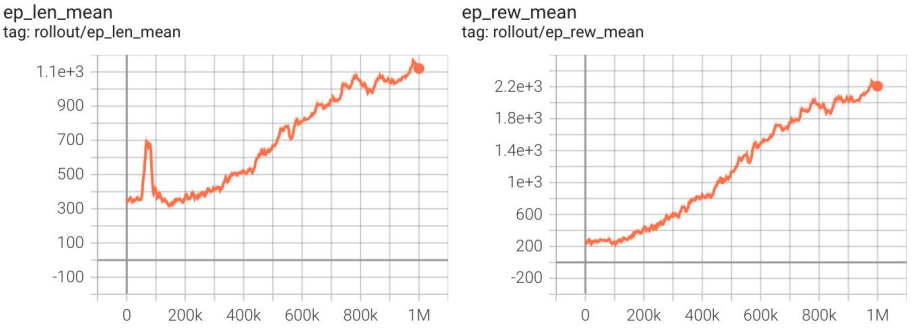


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 1840.0 (+/-663.2495759516172)
```

QR-DQN (με γ=0.95, batch_size=32):

Τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 2560.0 (+/-728.6288492778748)
```

Σχολιασμός

Επομένως συμπερένουμε ότι οι αλλαγές της τιμής της υπερπαράμτερου learning_rate συνέβαλαν αρνητικά επομένως αφήνουμε το learning_rate στην default τιμή του.

Συμπεράσματα

Άρα στον χώρο Asterix-v4 το **καλύτερο μοντέλο είναι το DQN με γ=0.95 και batch_size=16** και όλες οι υπόλοιπες παρατροι στις default τιμές με το εξής score:

```
Eval reward: 2870.0 (+/-310.0)
```

Αξίζει να σημειωθεί (αν και δεν ζητείται) ότι ο DQN είναι ο αλγόριθμος που προτάθηκε από την μελέτη των (Mnih et al., 2013) και χρησιμοποιούμε την υλοποίηση του stable-baselines3 και μπορεί να περιγραφεί ως εξής:

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Επίσης, αξίζει να σχολιάσουμε σύντομα τον τρόπο λειτουργίας του QR-DQN όπως περιγράφεται από αυτό το paper. Για πιο αναλυτικά δείτε το Appendix.

For a concrete algorithm we will build on the DQN architecture (Mnih et al. 2015). We focus on the minimal changes necessary to form a distributional version of DQN. Specifically, we require three modifications to DQN. First, we use a nearly identical neural network architecture as DQN, only changing the output layer to be of size $|\mathcal{A}| \times N$, where N is a hyper-parameter giving the number of quantile targets. Second, we replace the Huber loss used by DQN[4], $L_\kappa(r_t + \gamma \max_{a'} Q(x_{t+1}, a') - Q(x_t, a_t))$ with $\kappa = 1$, with a quantile Huber loss (full loss given by Algorithm 1). Finally, we replace RMSProp (Tieleman and Hinton 2012) with Adam (Kingma and Ba 2015). We call this new algorithm quantile regression DQN (QR-DQN).

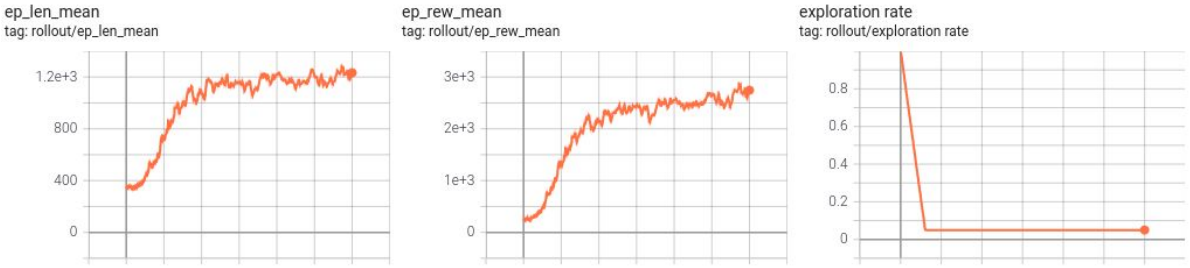
Algorithm 1 Quantile Regression Q-Learning

Require: N, κ
input $x, a, r, x', \gamma \in [0, 1)$
  # Compute distributional Bellman target
  $Q(x', a') := \sum_j q_j \theta_j(x', a')$
  $a^* \leftarrow \arg\max_{a'} Q(x, a')$
  $\mathcal{T}\theta_j \leftarrow r + \gamma \theta_j(x', a^*), \quad \forall j$
  # Compute quantile regression loss (Equation 10)
output $\sum_{i=1}^{N} \mathbb{E}_j \left[\rho_{\hat{\tau}_i}^{\kappa}(\mathcal{T}\theta_j - \theta_i(x, a)) \right]$

Τωρα θα τρέξουμε τα τελικά πειράματα σε περιβάλλον Asterix-vo με περισσότερες εποχές εκπαίδευσης. Επίσης θα τρέξουμε το αντίστοιχο πείραμα σε Asterix-v4 για να συγκρίνουμε αν πηγαίνει καλύτερα.

Τελικό Πείραμα

Για το περιβάλλον **Asterix-v4** με τον καλύτερο αλγόριθμο DQN σε 3M εποχές τα αποτελέσματα στο TensorBoard είναι το εξης:

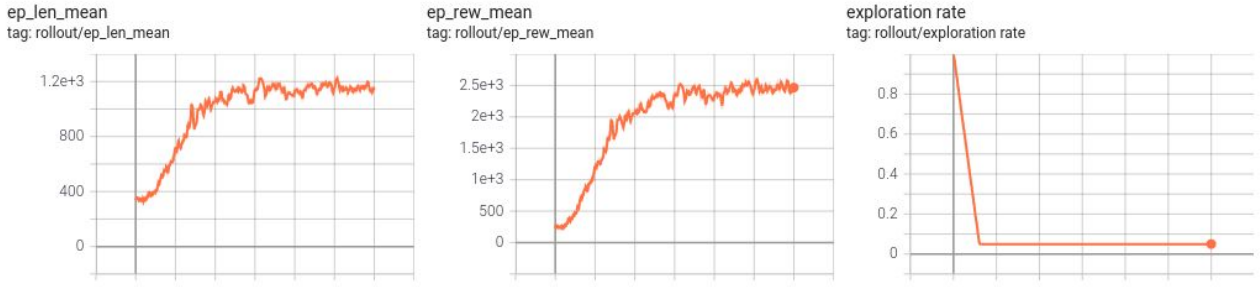


Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 3125.0 (+/-811.5571452461004)
```

Μπορείτε να δείτε ένα στιγμιότυπο από το testing του μοντέλου αυτού εδώ.

Για το περιβάλλον **Asterix-v0** με τον καλύτερο αλγόριθμο DQN σε 3M εποχές τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

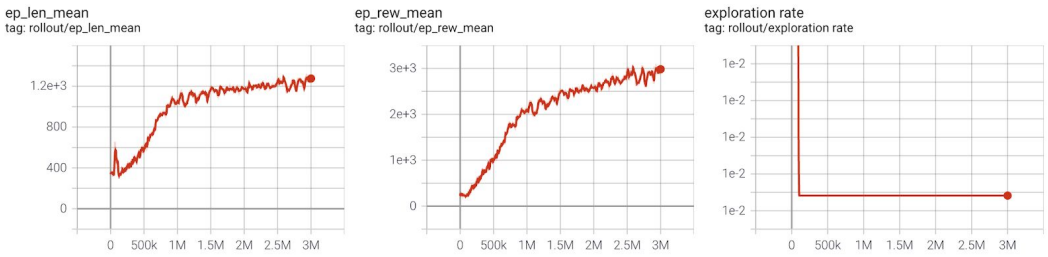
```
Eval reward: 2845.0 (+/-845.1183349093783)
```

Μπορείτε να δείτε ένα στιγμιότυπο από το testing του μοντέλου αυτού εδώ.

Μερικά Συμπεράσματα

Γενικά, περιμέναμε υψηλότερη τελική επίδοση δεδομένων καθώς στα στάδια της βελτιστοποίησης των παραμέτρων σε λίγες εποχές έχουμε δει score περίπου της τάξεως αυτών των τελικών πειραμάτων. Ίσως σε αυτό να οφείλεται και η απλότητα της ιδέας του DQN που μπορεί εν τέλει να μην ήταν ο ιδανικός αλγόριθμος για το συγκεκριμένο πείραμα και γι αυτόν τον λόγο θα τρέξουμε ένα ακόμη πείραμα με 3M εποχές στον πιο σύνθετο αλγόριθμο Quantile Regression Deep Q-Network.

Για το περιβάλλον **Asterix-v0** με τον καλύτερο συνδυασμό υπερ παραμέτρων για τον αλγόριθμο QR-DQN σε 3M εποχές τα αποτελέσματα στο TensorBoard είναι το εξης:



Το αποτέλεσμα στο Testing είναι το εξής:

```
Eval reward: 3165.0 (+/-460.4617247937118)
```

Μπορείτε να δείτε ένα στιγμιότυπο από το testing του μοντέλου αυτού εδώ.

Συμπεράσματα

Παρατηρούμε ότι ο πράκτορας με 3M εποχές εμφανίζει μικρή βελτίωση σε σχέση με το καλύτερο πράκτορα του προηγούμενου σταδίου της μελέτης αυτής, αλλά βλέπουμε ότι οι καμπύλες μάθησης είναι αρκετά πιο smooth που δείχνει ότι ο πράκτορας έχει αρχίσει και μαθαίνει το περιβάλλον που μελετάει. Εντυπωσιακό είναι ότι υπάρχει έντονη variance στα scores, πράγμα που μπορούμε να θεωρήσουμε ότι είναι λογικό επακόλουθο του περιβάλλοντος μελέτης.

Αξίζει να σημειωθεί ότι στο περιβάλλον v0 η εκπαίδευση ήταν πιο χρονοβόρα, συγκριτικά η εκπαίδευση στο περιβάλλον v0 κράτησε 4:51:22 ενώ στο περιβάλλον v4 κράτησε 4:22:52. Επίσης, παρατηρούμε ότι η επίδοση στο περιβάλλον v0 είναι χειρότερη από την επίδοση στο περιβάλλον v4, όπως αναμέναμε απο την θεωρία.

Συγκρίνοντας την επίδοση του μέσου χρήστη με την επίδοση των πρακτόρων στα προηγούμενα video μπορούμε να αντιληφθούμε μια πρωταρχική διαφοροποίηση. Ο μέσος χρήστης καταλαβαίνει **σημασιολογικά** τι συμβαίνει στο παιχνίδι, δηλαδή καταλαβαίνει ότι πρέπει να πάρει το φαγητό και να αποφύγει τους εισβολείς, ενώ ο πράκτορας μαθαίνει στατιστικά κάποιες κινήσεις και χώρους που είναι ασφαλείς από εχθρούς και τείνει να λάβει περισσότερο φαγητό. Ακριβώς γι αυτόν τον λόγο υπάρχει και αυτή η έντονη διακύμανση του score, επειδή άλλες φορες το παιχνίδι δημιουργεί αυτους τους χώρους ενώ άλλες φορές δημιουργεί άλλους χώρους που δεν έχει μάθει ο πράκτορας. Αυτός είναι και ο λόγος που η καλύτερη μηχανή δεν κατάφερε να ξεπεράσει το score του καλύτερου ανθρωπου, και έτσι μπορούμε να θεωρήσουμε το παιχνίδι του Asterix ως ένα **δύσκολο** παιχνίδι.

Συγκρίνοντας το score του agent με το score του μέσου χρήστη βλέπουμε ότι ο μέσος χρήστης επιτυγχάνει αρκετά υψηλότερο score και επίσης το state of the art είναι πολύ καλύτερο από τον μέσο χρήστη όπως και από τον πράκτορα που κατασκευάσαμε. Από την άλλη το score είναι πολύ υψηλότερο από το score του random agent, επομένως ο πράκτορας μαθαίνει.

Όπως, τονίστηκε και στην αρχή της εργασίας η στρατηγική που ακολουθήθηκε και μας οδήγησε να θεωρήσουμε ότι ο καταλληλότερος αλγόριθμος είναι ο DQN δεν είναι απαραίτητα ορθή. Αυτό συμβαίνει καθώς δεν μπορούμε να κάνουμε ένα πλήρες grid search λόγω έλλειψης υπολογιστικών πόρων. Επίσης είναι δύσκολο να κρίνουμε τους αλγορίθμους με λίγες εποχές καθώς σε διαφορετικά περιβάλλοντα κάποιοι μπορεί να αυξάνουν την επιδοση της πολύ γρήγορα στην αρχή και μετά να σταθεροποιούνται ενώ άλλοι αλγόριθμοι μπορεί να έχουν συνεχείς πιο αργή βελτίωση και μετά από αρκετό χρόνο εκπαίδευσης να επιτυγχάνουν τελικά καλύτερο score. Στην πρώτη κατηγορία σε αυτό το περιβάλλον μπορούμε να θεωρήσουμε ότι εμπίπτει ο DQN και στην δεύτερη περίπτωση ο QR-DQN.

Συγκριτικά βλέπουμε ότι ο QR-DQN απέδωσε καλύτερα στο πιο δύσκολο περιβάλλον Asterix-v0 δείχνοντας ότι η μέθοδος που ακολουθήθηκε δεν εγγυάται την επιτυχημένη έκβαση. Έτσι, πετύχαμε στο **Asterix-v0 με 3M** εποχές με τον QR-DQN score ίσο με **3165** με διακύμανση 460.

Appendix: Distributional Reinforcement Learning (C51 and QR-DQN)

Motivation

Value-based reinforcement learning methods like DQN try to model the expectation of total returns, or value.

That is, the value of an action a in state s describes the expected return, or discounted sum of rewards, obtained from beginning in that state, choosing action a, and subsequently following a prescribed policy.

All the state transitions, actions, and rewards, that are used to calculate the value or long-term return can induce randomness, if sampled probabilistically. This makes it useful to display the returns in a distribution: value distribution. The distribution of the random return received by a reinforcement learning agent.

However, traditional value-based reinforcement learning algorithms like DQN average over randomness to estimate the value.

Distributional reinforcement learning methods model this distribution over returns explicitly instead of only estimating the mean. This can lead to more insights and knowledge for the agent. And indeed results show the usefulness of modeling the value distribution in and of itself. Leading to a much faster and more stable learning of the agent.

Over the past years, since starting to develop these new distributional reinforcement learning algorithms, constant progress has been done. Which led to ever-improving algorithms.

C51 ⟶ QR-DQN ⟶ IQN ⟶ FQF

The first two popular distributional RL algorithms that were:

- C51
- QR-DQN

Both algorithms were published in 2017. However, C51 appeared in Juli 2017 and QR-DQN in October 2017. Further QR-DQN is built on C51, takes up some problems, and tries to solve them.

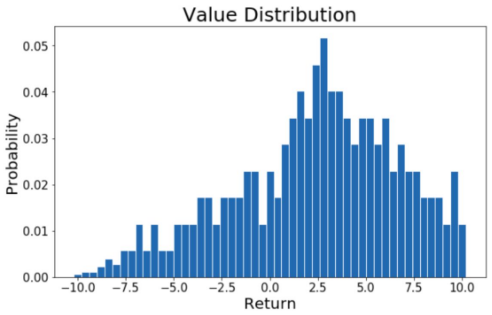
One of the theoretical contributions of the C51 work was a proof that the distributional Bellman operator is a contraction in a maximal form of the Wasserstein metric between probability distributions. The problem is, that the Wasserstein metric, viewed as a loss, cannot generally be minimized using stochastic gradient methods and thus does not lead directly to a practical algorithm.

C51 algorithm first performs a heuristic projection step, and then minimizes the KL divergence between projected Bellman update and prediction. By that C51 left the question open, if it is possible to devise an online distributional reinforcement learning algorithm that takes advantage of the contraction result.

QR-DQN tries to answer on that open question of the existence of a distributional algorithm that operates end-to-end on the Wasserstein metric, by appealing to the theory of quantile regression.

How does C51 work

In Categorical DQN (C51) the possible returns are limited to a discrete set of fixed values (51), and the probability of each value is learned through interacting with environments.



The value distribution is modeled by using a discrete distribution with the parameter N=51, V_max = 10, V_min = -10, and the set of atoms who support the distribution:

$$z_i = V_{MIN} + i\Delta z \,, with\ 0 \leq i < N \qquad \Delta z := \frac{V_{MAX} - V_{MIN}}{N-1}$$

You can interpret these atoms as the "canonical returns" of the value distribution. The probability of each atom is given by our neural network:

$$Z_\theta(x,a) = z_i \quad \text{w.p.} \;\; p_i(x,a) := \frac{e^{\theta_i(x,a)}}{\sum_j e^{\theta_j(x,a)}}.$$

Using a discrete distribution poses a problem: the Bellman update TZ and our parametrization Z almost always have disjoint supports. To overcome this issue, the sample Bellman update is projected onto the support of Z and thereby effectively reducing the Bellman update to multiclass classification.

First, calculate the Bellman update for each atom z_j:

$$\mathcal{T} z_j := r + \gamma z_j$$

Followed by distributing the probability of each atom to its immediate neighbors. This defines the projection update:

$$(\Phi\hat{\mathcal{T}} Z_\theta(x,a))_i = \sum_{j=0}^{N-1} \left[1 - \frac{|[\hat{\mathcal{T}} z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}} - z_i|}{\triangle z} \right]_0^1 p_j(x', \pi(x'))$$

Minimizing the loss, which is the cross-entropy term of the KL divergence:

$$D_{\text{KL}}(\Phi\hat{\mathcal{T}} Z_{\tilde{\theta}}(x,a) \,\|\, Z_\theta(x,a))$$

With that, the distributional RL algorithm C51 outperforms all previous variants of DQN on a set of 57 Atari 2600 games. Link to my implementation of C51. C51 was used for the Rainbow version of DQN which combined all advancements for DQN algorithms.

How does QR-DQN work

QR-DQN differs from C51 in that it learns the quantile values directly. QR-DQN computes the return quantiles on fixed, uniform quantile fractions using quantile regression and minimizes the quantile Huber loss between the Bellman updated distribution and current return distribution.

Compared to C51, QR-DQN is not restricted or bound for value, and therefor improves significantly over C51. But, both C51 and QR-DQN approximate the distribution function or quantile function on fixed locations, either value or probability.

$$\tau_i = \frac{i}{N} \text{ for } i = 1, \ldots, N$$

But how does that work? As we know from C51 we have 51 fixed locations for the value distribution and we learn the probabilities of the fixed location. QR-DQN "transpose" this parametrization by considering fixed probabilities.

Only a few changes need to be done to transform DQN to a QR-DQN network. First, changing the output layer to be of size |A|* N, where N=32 is a hyper-parameter giving the number of quantile targets, and A is the action space.

$$Z_\theta(x,a) := \frac{1}{N}\sum_{i=1}^{N}\delta_{\theta_i(x,a)}$$

Z_theta can be seen as the regular Q-Value which is the mean over the estimated Dirac δ by our network. The second adjustment we have to do is to change our loss function, which might be the Huber loss or the MSE loss, and replace it with the quantile Huber loss:

$$\mathcal{L}_\kappa(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \le \kappa \\ \kappa(|u| - \frac{1}{2}\kappa), & \text{otherwise} \end{cases} \qquad \rho_\tau^\kappa(u) = |\tau - \delta_{\{u<0\}}|\mathcal{L}_\kappa(u)$$

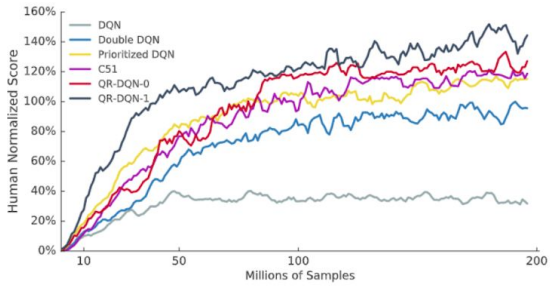
Where the constant kappa k is set to 1, u is the TD error:

$$u = \mathcal{T}\theta_j - \theta_i(x,a)$$

with the distributional Bellman target:

$$\begin{aligned} &Q(x',a') := \textstyle\sum_j q_j\theta_j(x',a') \\ &a^* \leftarrow \arg\max_{a'} Q(x,a') \\ &\mathcal{T}\theta_j \leftarrow r + \gamma\theta_j(x',a^*), \qquad \forall j \end{aligned}$$

and theta_i the expected estimated value for the current state-action tuple. With these slight changes, QR-DQN not only outperforms DQN but also gains an impressive 33% median score increment over the back then state-of-the-art C51.



	Mean	Median	>human	>DQN
DQN	228%	79%	24	0
DDQN	307%	118%	33	43
DUEL.	373%	151%	37	50
PRIOR.	434%	124%	39	48
PR. DUEL.	592%	172%	39	44
C51	701%	178%	40	50
QR-DQN-0	881%	199%	38	52
QR-DQN-1	**915%**	**211%**	**41**	**54**