ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΌΓΩΝ ΜΗΧΑΝΙΚΏΝ ΚΑΙ ΜΗΧΑΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΏΝ

[PLH 423]-Reinforcement Learning & Dynamic Optimization

Αναφορά 2ης εργασίας

Ιωάννης Περίδης

A.M. 2018030069

1 Εισαγωγή:

Στην προγραμματιστική άσκηση αυτή, μας δίνεται ένα σύνολο δεδομένων, πραγματικών φορτίων κίνησης με την πάροδο του χρόνου , για διαφορετικούς servers. Συγκεκριμένα, μας δίνονται τα φορτία για k=30 servers, σε έναν ορίζοντα T ίσο με 7000 timesteps. Μας ζητείται η υλοποίηση αλγορίθμων multiplicative weights σε περιβάλλοντα με experts και με bandits, αλλά και η προσαρμογή του αλγορίθμου UCB στο πρόβλημα αυτό. Έπειτα πρέπει να γίνουν συγκρίσεις μεταξύ της λειτουργίας του και του regret τους, όπως και σχολιασμός της επίδρασης διαφορετικών χρονικών οριζόντων.

2 Βασική Λειτουργία - Ομοιότητες & Διαφορές:

Και οι τρείς αλγόριθμοι, προσπαθούν με διαφορετικούς τρόπους, να μαθαίνουν να προβλέπουν τον server με το μικρότερο φορτίο για κάθε χρονική στιγμή.

Ο ***Multiplicative Weights σε Experts environment***, λειτουργεί ως εξής. Αντιμετωπίζει τους servers σαν k experts, με κάθε έναν από αυτούς να έχει διαφορετικό βάρος, σχετιζόμενο με το loss του, δηλαδή στην περίπτωση μας με το load του server. Το βάρος αυτό, μας υποδεικνύει τον βαθμό εμπιστοσύνης, που έχει ο expert, δηλαδή διαμορφώνει το πόσο μεγάλη ή μικρή θα είναι η πιθανότητα που επιλέγεται. Συγκεκριμένα, κάθε expert, αρχίζει για τη χρονική στιγμή 0 με βάρος ίσο με 1. Έπειτα, για κάθε γύρο t, υπολογίζεται η πιθανότητα επιλογής κάθε server I και είναι ίση με το προσωπικό του βάρος, δια το αθροιστικό: $\boldsymbol{p_i = \frac{w_i}{\sum w_i^t} = \frac{w_i}{W_i}}$. Στην συνέχεια ανάλογα με το φορτίο του κάθε server, αναναιώνοται αντίστοιχα το βάρος του από την σχέση $\boldsymbol{w_i^{t+1} = (1-\eta)^{li^t} w_i^t, \forall i}$, όπου παράμετρος $\boldsymbol{\eta = \left(\frac{lnk}{T}\right)^{\frac{1}{2}}}$, βοηθάει στο discount των weights και $\boldsymbol{l_i^t = loss = load}$ του server για κάθε χορνική στιγμή, το οποίο γνωστό για όλους του servers και όχι μόνο αυτόν που επιλέγεται. Ο συντελεστής έκπτωσης βάρους $\boldsymbol{(1-\eta)^{li^t}}$, είναι μεγαλύτερος για servers που είχαν μεγάλο load , έτσι ώστε να γίνεται μεγαλύτερη μείωση βάρους και να μην επιλέγονται με μικρότερη πιθανότητα. Αντίστοιχα η πιθανότητα επιλογής για τους servers με μικρό load, μειώνεται ελάχιστα, έτσι ώστε να επιλέγονται συχνότερα. Ο αλγόριθμος, για το επιλεγμένο η, υποθέτοντας οτι το optimal Load είναι ίσο με T, πετυχαίνει sublinerar regret και ο λόγος που δουλεύει τόσο εύστοχα σε adversarial περιβάλλοντα, είναι το γεγονός ότι είναι πιθανοτικός, δηλαδή περιέχει και τυχαιότητα η οποία είναι απαραίτητη σε τέτοια περιβάλλοντα

Ο ***Multiplicative Weights σε Bandits environment*** λειτουργεί με αρκετά παρόμοιο τρόπο με τους experts, αλλά με την πολύ σημαντική διαφορά ότι πλέον τα loads των servers δεν ειναι όλα γνωστά, αλλά είναι γνωστό μόνο το load του server που επιλέγεται. Επομένως, δεν είναι εφικτό να εφαρμοστεί ο MW απευθείας. Πλέον, η πιθανότητα επιλογής κάθε bandit, είναιλειτουργεί με αρκετά παρόμοιο τρόπο με τους experts, αλλά με την πολύ σημαντική διαφορά ότι πλέον τα loads των servers δεν ειναι όλα γνωστά, αλλά είναι γνωστό μόνο το load του server που επιλέγεται. Επομένως, δεν είναι εφικτό να εφαρμοστεί ο MW απευθείας. Πλέον, η πιθανότητα επιλογής κάθε bandit, είναι: $\boldsymbol{q_i^t = (1-\varepsilon)p_i^t + \frac{\varepsilon}{k}}$, όπου το pi είναι το ίδιο με πριν και εξαρτάται από τα weights, και η παράμετρος : $\boldsymbol{\varepsilon = \left(\frac{k*lnk}{T}\right)^{\frac{1}{3}}}$, προσθέτει το exploration στην λύση. Στην συνέχεια, μέσο της διαδικασίας importance sampling, υπολογίζεται το unbiased estimate του load: $\boldsymbol{\hat{l_i^t} = \frac{l_i^t}{q_i^t}}$ για το επιλεγμένο bandit i και 0 αλλού , το οποίο χρησιμοποιείται για να ανανεωθεί το βάρος του επιλεγμένου bandit, ενώ τα άλλα βάρυ μένουν σταθερά: $\boldsymbol{w_i^{t+1} = (1-\eta)^{\hat{l_i^t}} w_i^t}$. Όπως και πριν όσο μεγαλύτερο load είχε ο server Που επιλέχθηκε, τόσο πιο πολύ θα μειωθεί το βάρος του. Χρησιμοποιώντας, το κατάλληλο ε με τον παραπάνω τύπο, επίσης επιτυγχάνεται sublinear regret.

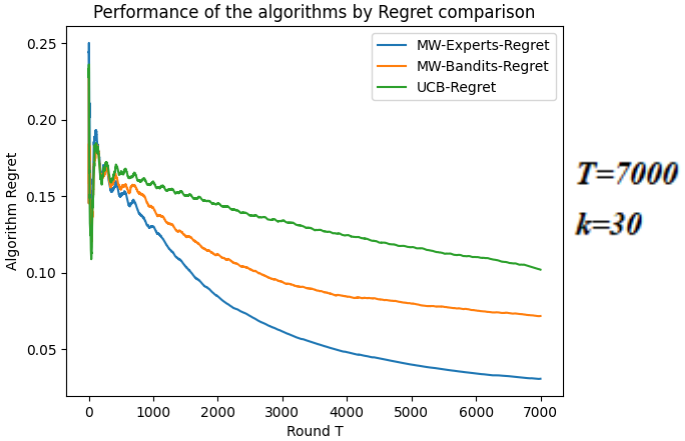
Ο ***Upper Confidence Bound,*** είναι ο γνωστός αλγόριθμος που χρησιμοποιήθεικε σε προηγούμενη εργασία , οπότε δεν θα ξαναγίνει αναλυτική επεξήγηση. Με την σημαντική διαφορά πως τώρα πλέον λειτουργέι με losses , δηλαδή τα loads και όχι rewards, επομένως, θα πρέπει να τροποποιηθεί κατάλληλα. Αρχικά, πως ο πρώτος όρος του exploitation, θα είναι το άθροισμα των losses (loads) και όχι των rewards, ενώ δεύτερος όρος του exploration, θα παραμείνει ίδιος. Για να λειτουργήσει σωστά ο UCB, θα πρέπει και οι δύο όροι, να συμβάλουν ταυτόχρονα θετικά ή αρνητικά στην τιμή UCB με την οποία θα επιλεχθεί ο κάθε server και να μην έρχονται σε αντιπαράθεση ο ένας με τον άλλο. Επομένως, αφού πλέον ο πρώτος όρος συμβάλει αρνητικά , έτσι θα πρέπέι και ο δεύτερος όρος να συμβάλει αρνητικά. Αυτό επιτυγχάνεται , αντικαθιστώντας την πρόθεση του δεύτερου όρου με αφαίρεση. Τέλος, ο αλγόριθμος θα πρέπει να επιλέγει τον server, πλέον με την μικρότερη τιμή UCB και όχι την μεγαλύτερη , αφού αυτός θα έχει το ελάχιστο load. Πετυχαίνει επίσης και αυτός sublinear regret.

3 Υλοποίηση Κώδικα:

Ο κώδικας ήταν αρκετά απλός. Αρχικά φορτώθηκαν τα loads απο το csv αρχείο σε έναν πίνακα loads[T][k], οι γραμμές αντιστοιχούν στα timesteps και οι στήλες στους διαφορετικούς servers και αρχικοποιήθκαν όλες οι παράμετροι. Ο UCB είχε

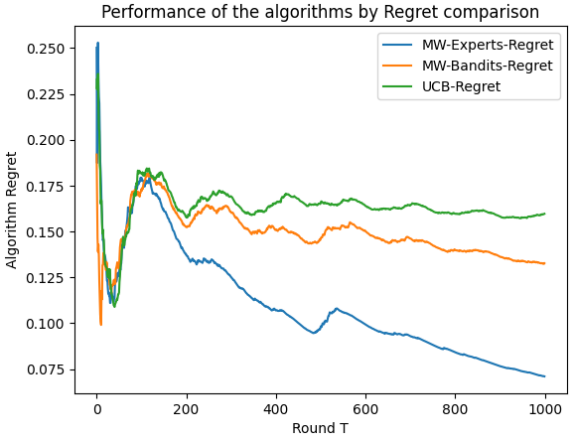
την ίδια υλοποίηση με την πρώτη άσκηση, με τις διαφορές που προαναφέρθηκαν.Για την υλοποίηση του MW, χρησιμοποιήθηκαν κοινές συναρτήσεις ,με διαφορετικές υλοποίήσεις στο σώμα, για τους experts και τα bandits, εισάγοντας κάθε φορά σαν όρισμα για ποιον από τους 2 αλγορίθμους πρόκειται η εκτέλεση. Η συνάρτηση multiplicative_weights, καλεί με την σειρά τις συναρτήσεις update_probabilities, η οποία ανανεώνει τα διανύσματα των πιθανοτήτων επιλογής server με τους προαναφαίροντες τύπους για κάθε περίπτωση. Έπειτα, καλείται η select_sever, στην οποία επιλέγεται και επιστρέφεται μέσω της random choice και των probabilities, ο αριθμός του server και το load που έχει. Ύστερα , καλείται η update_weights η οποία ανάλογα τον αλγόριθμο, ανανεώνει τις τιμές όλων των weights για τους experts και για τα bandits, ανανεώνει μόνο την τιμή του επιλεγμένου server, αφού πρώτα υπολογίσει το estimated load. Σε ένα for loop διάρκειας T, με timestep 1, καλούνται και οι 3 αλγόριθμοι και υπολογίζονται το αθροισιτκο total_load των επιλεγμένων server , το αθροιστικο total_optimal_load(δηλαδή το ελάχιστο σε κάθε επανάληψη) και το regret του κάθε ενός, μέσω του τύπου: $\boldsymbol{regret^t = Load^t_{alg} - Load^t_{opt}}$. Τέλος, εκτυπώνονται αυτές οι μετρήσεις και συγκρίνονται σε κοινές γραφικές παραστάσεις. Να σημειωθεί πως αν ο αναγνώστης θέλει να εκτυπωθούν για κάθε βήμα ο αριθμός επιλεγμένου server και το load του, για κάθε αλγόριθμο και για τις optimal λύσεις, πρέπει να κάνει το αντίστοιχο κομμάτι του κώδικα unccoment.

4 Αποτελέσματα & Συγκρίσεις:



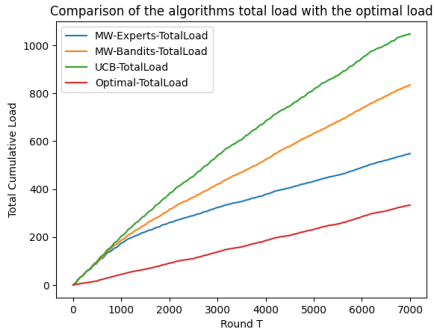
T=7000

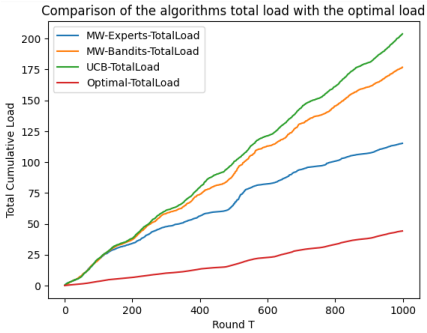
k=30



T=1000

k=30





Στις γραφικές παραστάσεις ,παραπάνω απεικονίζονται τα regrets στον χρόνο των 3 αλγορίθμων και τα αθροιστικά loads τους σοτν χρόνο, σε σχέση με το βέλτιστο load.

Παρατηρείται και στις 2 περιπτώσεις πως οι experts, έχουν μικρότερο regret και μικρότερο load, από τα bandits.Γεγονός που ήταν αναμενόμενο από την θεωρία ,αφού σε bandit environment, ο αλγόριθμος, θα συγκλίνει πιο αργά κατά παράγοντα k, ο οποίος δημιουργείται λόγο της μερικής παρατηρησιμότητας (γνωστό μόνο το ένα load). Αυτό, διότι τα bandits, χρειάζονται k timesteps, για να λάβουν την ίδια πληροφορία που λαμβάνουν οι experts σε ένα timestep. Επομένως, γίνεται κατανοητή η διαφορά πως ο συντελεστής discount των weights, εφαρμόζεται σε κάθε expert και άρα συμβαίνει μεγαλύτερη και πιο γρήγορη μείωση βάρους στους experts που έχουν μεγάλο loss. Όσον αφορά τον UCB, ήταν αναμενόμενο, να είναι χειρότερος και από τους 2 άλλους αλγορίθμους, διότι το περιβάλλον είναι μεταβαλλόμενο και ο UCB βασίζεται σε ισχυρά assumptions και άρα είναι βέλτιστος σε αυστηρά μη μεταβαλλόμενα περιβάλλοντα. Παρόλα αυτά, αποτελεί μια καλή εναλλακτική όταν το load όλων των servers δεν είναι διαθέσιμο, καθώς εξισορροπεί αποτελεσματικά την εξερεύνηση και την εκμετάλλευση. Συνολικά, αυτά τα αποτελέσματα υποδηλώνουν ότι ο αλγόριθμος MW των experts είναι η καλύτερη επιλογή για αυτό το πρόβλημα, καθώς είναι σε θέση να λαμβάνει σχεδόν βέλτιστες αποφάσεις με πρόσβαση στα loads όλων των servers.

Στην **1η περίπτωση για T=7000**, regret: experts:0,25→0,05/ bandits:0,25→0,1/ UCB: 0,25→0,15
Στην **2η περίπτωση για T=1000**, regret: experts:0,25→0,075/ bandits:0,25→0,15/ UCB: 0,25→0,175

Παρατηρείται, πως με την μείωση του ορίζοντα, το regret και τα loads και των 3 αλγορίθμων, αυξάνονται. Παρόλα αυτά, φαίνεται πως οι experts, επηρεάζονται λιγότερο από τον ορίζοντα, σε σχέση με τα bandits και το UCB. Γεγονός που ήταν αναμενόμενο, διότι ο UCB δεν προλαβαίνει να εξισοροπίσει σωστά το exploitation με το exploration και έτσι δεν βρίσκει τις βέλτιστες λύσεις, ενώ τα bandits, δεν προλαβαίνουν σε τόσο μικρό χρονικό διάστημα να συλλέξουν όλη την απαραίτητη πληροφορία που έχουν κατευθείαν οι experts,για βέλτιστες αποφάσεις.Ωστόσο, η σχετική απόδοση των αλγορίθμων παρέμεινε σταθερή και ικανοποιητική και στους δύο ορίζοντες.