Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

Γραπτή Αναφορά

Μέλη Ομάδας:

Νιώτης Νικόλαος	1115201700106
Παπαθυμιόπουλος Θωμάς	1115201700119

Αθήνα

Ιανουάριος 2021

Σύνοψη

Η πρωτη διαδικασία που γίνεται από το πρόγραμμα είναι η ανάγνωση των datasets και η δημιουργία των κατάλληλων αντικλικών και κλικών από αυτά. Κατά την διαδικασία αυτή για κάθε entry γίνεται καθαρισμός των specs του και η εισαγωγή όλων των μοναδικών λέξεων και των idf τους στο λεξιλόγιο μας.

Με την λήξη της ανάγνωσης έχουμε το πλήρες λεξιλόγιο σε μορφή AVL δέντρου του οποίου κάθε κόμβος αποτελεί μια λέξη. Το δέντρο αυτό μετατρέπεται σε πίνακα. Κατά την μετατροπή αυτή σε κάθε λέξη του δέντρου φυλάσσεται η θέση που της αντιστοιχεί στον πίνακα.

Έπειτα δημιουργούμε ένα sparse matrix για όλα τα entries του προγράμματος. Το sparse matrix είναι ένας πίνακας από λίστες λέξεων. Κάθε στοιχείο του sparse matrix κρατάει τις μοναδικές λέξεις των specs ενός αρχείου μαζί με τις τιμές bow και tfidf που του αντιστοιχούν.

Ο Job Scheduler δημιουργεί το πλήθος των thread που έχουμε ορίσει και τα φυλάει σε ένα thread pool και αρχικοποιεί την ουρά προτεραιότητας που θα περιέχει τα jobs. Όταν θέλουμε να παραλληλοποιήσουμε μια διαδικασία γεμίζουμε την ουρά με τα αντίστοιχα κομμάτια των συναρτήσεων και όταν η διαδικασία αυτή ολοκληρωθεί αφήνουμε τα threads να τα εκτελέσουν.

Όσον αφορά την εκπαίδευση του μοντέλου λογιστικής παλινδρόμησης, αυτή γίνεται με mini batches τα οποία τροφοδοτούμε στον Job Scheduler . Μόλις ολοκληρωθεί η εισαγωγή, αυτά εκτελούνται και το καθένα προσαρμόζει ανάλογα τις παραγώγους θj(w). Μόλις τελειώσει η εκτέλεση όλων των jobs (δηλαδή όλου του batch) τροποποιούνται τα βάρη σύμφωνα με τις τιμές των παραγώγων που σχηματίστηκαν.

Στην επανεκπαίδευση του μοντέλου, κάνουμε predict την τιμή κάθε ζεύγους προϊόντων τα οποία δε βρίσκονται στο προηγούμενο train_set και αν αυτές πλησιάζουν τις τιμές στόχους (0 ή 1) κατά ένα ορισμένο threshold τότε το ζευγάρι αυτό γίνεται υποψήφιο για ένταξη στο νέο train_set. Όλα αυτά τα ζευγάρια αποθηκεύονται σε μια δομή, γίνοται sort, ελέγχονται κατά φθίνον prediction value και αν δεν έχουμε πληροφορία διαφορετική αυτής που προέκυψε απο το prediction ομαδοποιούνται στην ίδια κλίκα προϊόντων ή διαφορετικών προϊόντων ανάλογα. Όταν αυτή η διαδικασία ολοκληρωθεί για όλα τα υποψήφια ζευγάρια, εξάγουμε το σύνολο των πληροφοριών που έχει σχηματιστεί σε ένα output αρχείο και το χρησιμοποιούμε σαν νέο train_set.

Παρατηρήσεις & Αποφάσεις

AVL δεντρα

Επιλέξαμε να χρησιμοποιήσουμε δέντρα στις περισσότερες περιπτώσεις όπου θα βάζαμε λίστες. Αυτό μας επιτρέπει αρκετά ταχύτερη αναζήτηση, μειώνοντας έτσι σημαντικά τον χρόνο εκτέλεσης πολλών εργασιών όπως την αναζήτηση ενός entry στο hashtable, την αναζήτηση λέξεων και πολλά άλλα. Έπειτα καταλήξαμε στην απόφαση τα δέντρα αυτά να είναι self-balancing ώστε να διατηρούν σταθερό ύψος και κατ' επέκταση σταθερό χρόνο αναζήτησης.

Sparse Matrix

Αρχικά φυλασσόταν πίνακας τιμών για όλες τις τιμές bow και tf-idf των καλύτερων λέξεων του κάθε αρχείου, ωστόσο ακόμα και κρατώντας ένα σχετικά μικρό πλήθος λέξεων παρατηρήθηκε πως υπήρχαν ασύγκριτα πολλές μηδενικές τιμές σε σχέση με τις μη μηδενικές για το κάθε αρχείο. Αυτό προκαλούσε τεράστια σπατάλη μνήμης αλλά και χρόνου για να διατρέξουμε αυτούς τους πολύ αραιούς πίνακες όταν αυτό χρειαζόταν. Έτσι καταλήξαμε στη χρήση ενός sparse matrix το οποίο έλυσε τόσο το πρόβλημα του χώρου όσο και του χρόνου.

Για την παραγωγή των tfidf διανυσμάτων κάθε entry χρειάζεται το σύνολο των μοναδικών λέξεων των specs του, που ανήκουν στις καλύτερες λέξεις. Το sparse matrix είναι ένας πίνακας από λίστες λέξεων με διάσταση όσο και το πλήθος των entries. Σε κάθε κόμβο των λιστών φυλάσσεται μια λέξη μαζί με τις τιμές bow και tfidf που της αντιστοιχούν και την θέση της λέξης αυτής στον πίνακα των καλύτερων λέξεων. Μια σειρά του sparse matrix καταναλώνει λιγότερη μνήμη από μια αντίστοιχη σειρά κανονικού matrix καθώς φυλάσσονται μονάχα οι λέξεις που αντιστοιχούν στο συγκεκριμένο αρχείο (δηλαδή μονάχα όσες δεν έχουν μηδενική τιμή tfidf). Για την μετατροπή μια σειράς του sparse matrix σε tfidf vector απαιτείται μονάχα μια προσπέλαση της λίστας κάτι με time complexity O(N) όπου N είναι το μέγεθος της.

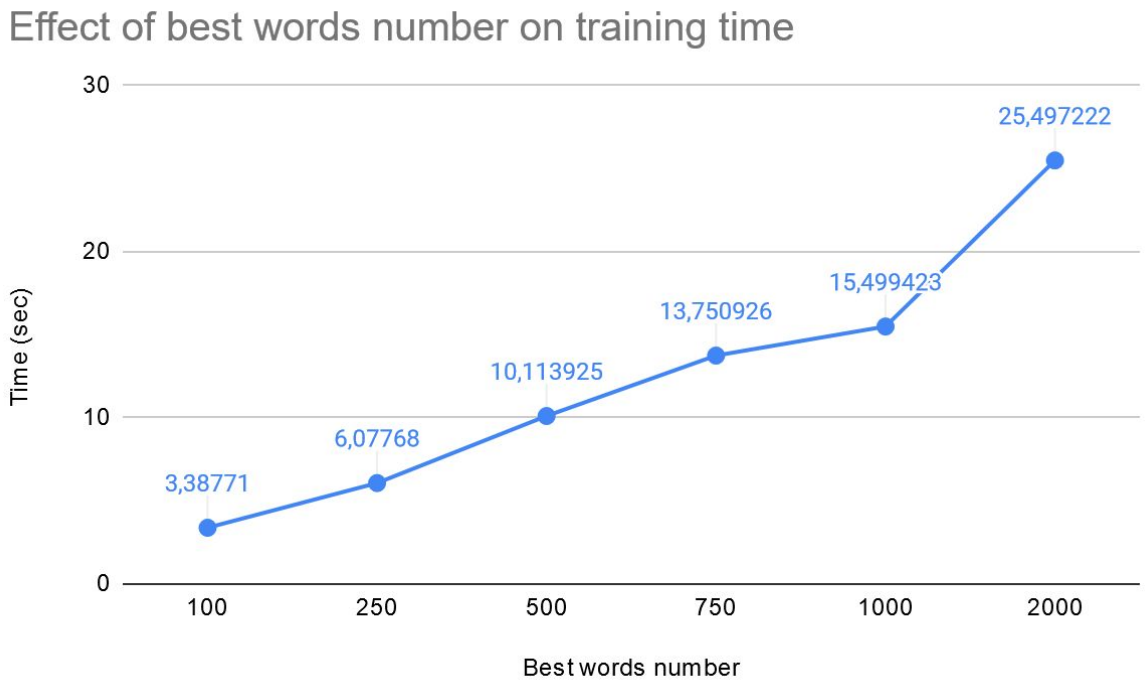
Πειράματα

Το μηχάνημα των εκτελέσεων:
CPU: Intel Core i5 6600 (4 πυρήνες, 4 threads, 3.3GHz base clock)
Ram: 16GB
Λογισμικό: Fedora 32
gcc version 10.2.1 20201125 (Red Hat 10.2.1-9) (GCC)

Ορισμένα από τα πειράματα που διεξήχθησαν, εφαρμόστηκαν σε 1 κύκλο εκπαίδευσης καθώς τα αποτελέσματα αρκούν και μπορούν εύκολα να αναχθούν σε μεγαλύτερο πλήθος εκπαιδεύσεων, οι οποίες μάλιστα θα έχουν και περισσότερες πληροφορίες.

Number of Words

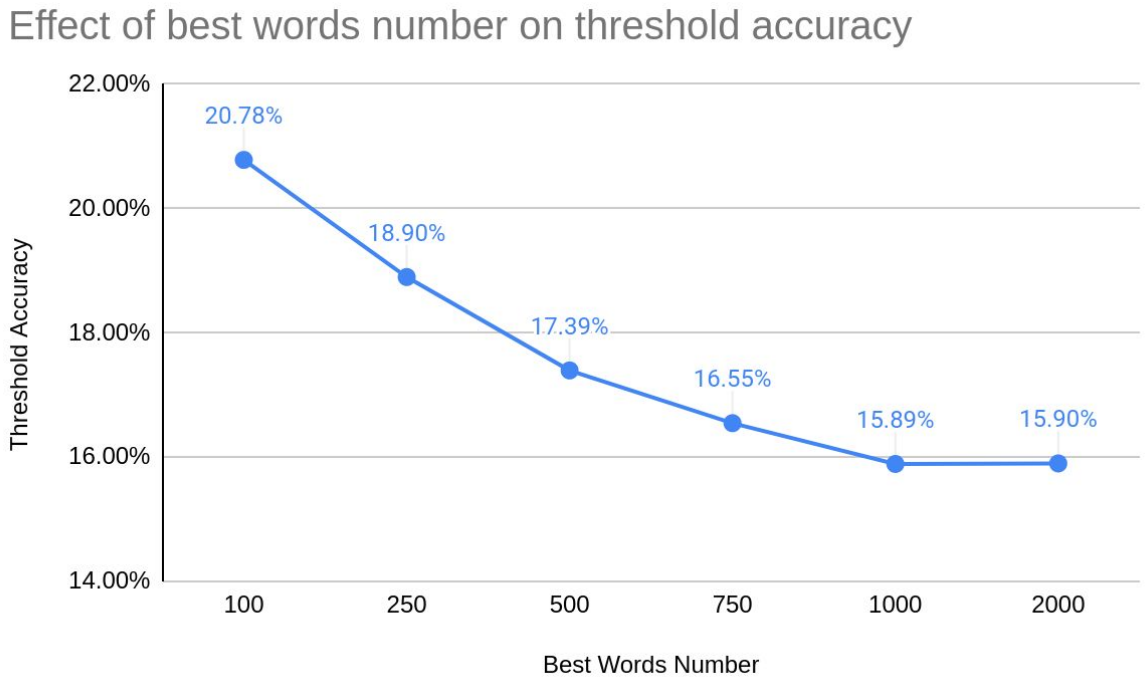
Αρχικά ελέγξαμε το ιδανικό πλήθος σημαντικών λέξεων που έπρεπε να κρατήσουμε. Συγκρίνοντας διάφορες τιμές και παρατηρώντας αποκλειστικά τον χρόνο εκτέλεσης παρατηρούμε μεγάλη βελτίωση στον χρόνο εκτέλεσης του training.



Στο παραπάνω διάγραμμα φαίνονται τα αποτελέσματα μερικών δοκιμαστικών εκτελέσεων (με μία μόνο εκπαίδευση του μοντέλου). Το συμπέρασμα ότι λιγότερες λέξεις θα επιτάχυναν την εκτέλεση τόσο της εκπαίδευσης όσο και των test και validate συναρτήσεων ήταν απολύτως αναμενόμενο καθώς κρατώντας λιγότερες λέξεις μειώνεται σημαντικά το συνολικό πλήθος συγκρίσεων του προγράμματος. Τα συμπεράσματα αυτά γιγαντώθηκαν διατηρώντας μικρότερο πλήθος λέξεων για την επανεκπαίδευση του μοντέλου όπου με 100 λέξεις

παρατηρήθηκε 50% μείωση του χρόνου εκτέλεσης του training έναντι της επανεκπαίδευσης με 500 λέξεις.

Σημαντική διαφορά βέβαια προκάλεσε και στο accuracy του μοντέλου

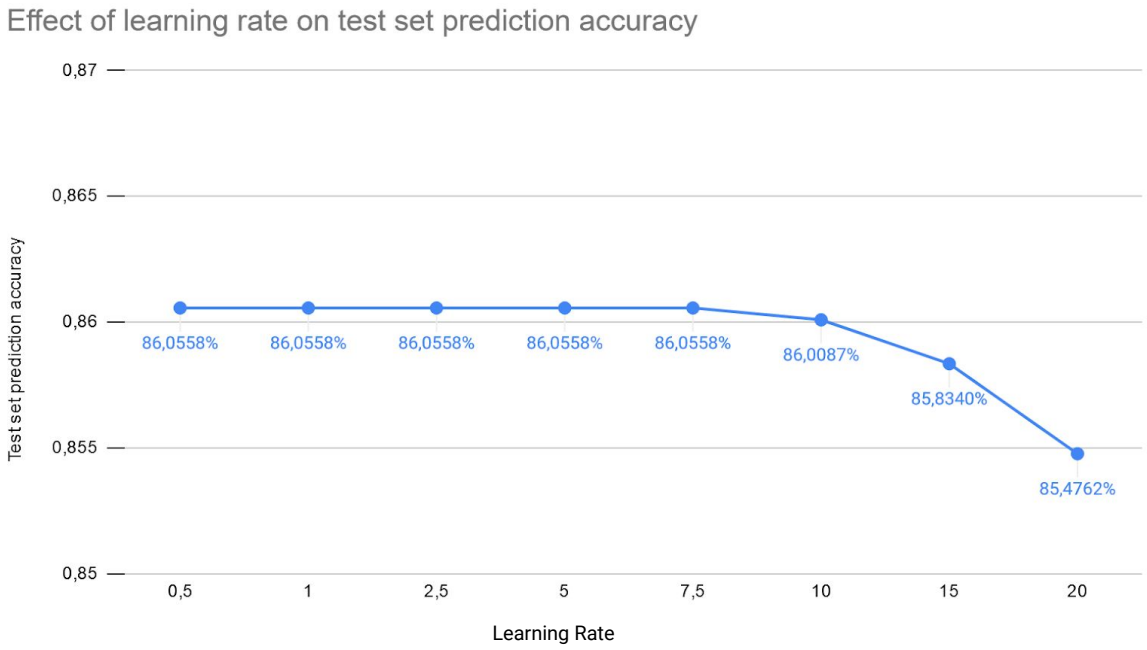


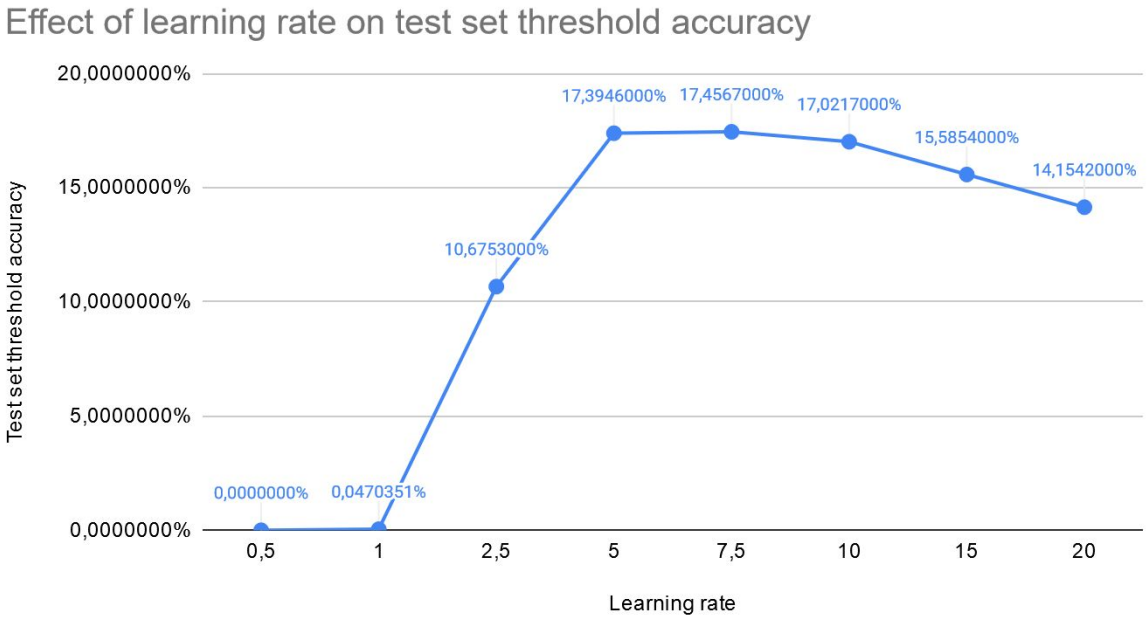
Στα παραπάνω διαγράμματα οπτικοποιήσαμε την επιρροή του πλήθους των λέξεων στο accuracy του με threshold 0,1. Παρά το γεγονός ότι τα αποτελέσματα δείχνουν τα μικρά πλήθη λέξεων να υπερτερούν και σε αυτόν τον τομέα, θεωρούμε ότι το συγκεκριμένο αποτέλεσμα οφείλεται στο γεγονός ότι το μοντέλο είναι ευνοηκότερο προς τις αρνητικές συσχετίσεις καθώς το πλήθος τους ξεπερνά κατά πολύ αυτό των θετικών στο train_dataset. Χρησιμοποιώντας μικρότερα πλήθη λέξεων και στην επανεκπαίδευση παρουσιάστηκε και στο accuracy σημαντική βελτίωση όπως και στους χρόνους. Επειδή όμως φτάνοντας στα όρια αυτού του ελέγχου, ορίζοντας δηλαδή το πλήθος των λέξεων σε 10 παρατηρήσαμε ακόμα μεγαλύτερη βελτίωση αποτελεσμάτων παραμένουμε καχύποπτοι για την ουσιαστική συνεισφορά του, καθώς με τόσο λίγες λέξεις θα χανόταν εντελώς το νόημα και η διαφοροποίηση των αρχείων, σε ένα πιο ισορροπημένο train_set και έτσι συνεχίζουμε τους ελέγχους μας διατηρώντας το πλήθος λέξεων στο 500.

Learning Rate

Από τους πιο σημαντικούς παράγοντες για το μοντέλο της λογιστικής παλινδρόμησης αποτελεί το learning rate. Δοκιμάζοντας τιμές από 0,5 έως και 20 για ένα train με batch size 4096 και 5 epochs προέκυψαν τα παρακάτω γραφήματα.

Στο πρώτο παρατηρούμε την ουσιαστικά αμυδρή επίδρασή του στο γενικό prediction accuracy του test_set. Αυτό κατά πάσα πιθανότατα οφείλεται στο bias που προαναφέραμε καθώς παρατηρήθηκε πώς ακόμα και με ελάχιστη ή ελλιπή εκπαίδευση του μοντέλου αυτό μπορούσε να πετύχει παρόμοια αποτελέσματα.



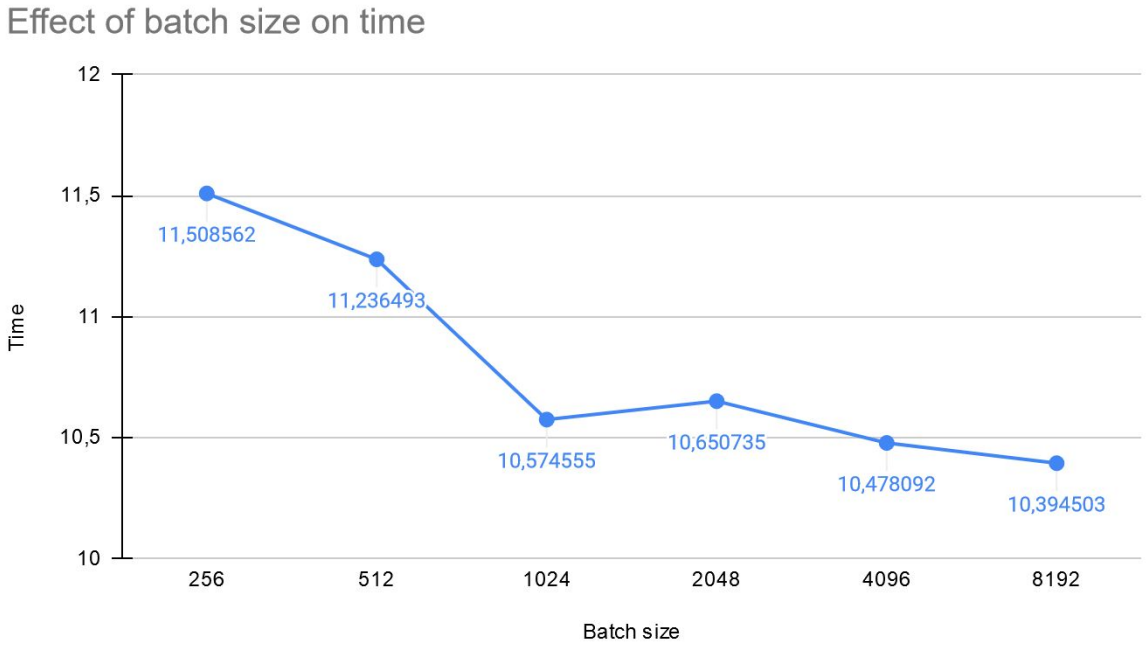


Στο δεύτερο γράφημα ωστόσο αναπαραστήσαμε την επίδρασή του στο prediction accuracy του test_set με threshold 0,1. Εδώ πλέον μπορούμε να δούμε εμφανείς διαφορές ανάμεσα στις τιμές. Για τιμές μικρότερες του ιδανικού, τα predictions δεν παρουσιάζουν μεγάλη

“βεβαιότητα” για το αποτέλεσμα της σύγκρισης ενώ για μεγαλύτερα learning rate παρατηρούμε το accuracy να φθίνει, προφανώς διότι τα βάρη των διανυσμάτων μεταβάλλονται περισσότερο απ’ ότι θα έπρεπε. Καταλήξαμε λοιπόν να κρατήσουμε το ελάχιστο δυνατό learning rate καθώς στις επανεκπαιδεύσεις, όπου τα δεδομένα όλο και θα πληθαίνουν, το μεγαλύτερο learning rate θα προκαλεί μειωμένη απόδοση στις “σίγουρες” προβλέψεις, όπως είδαμε και στο παραπάνω πείραμα.

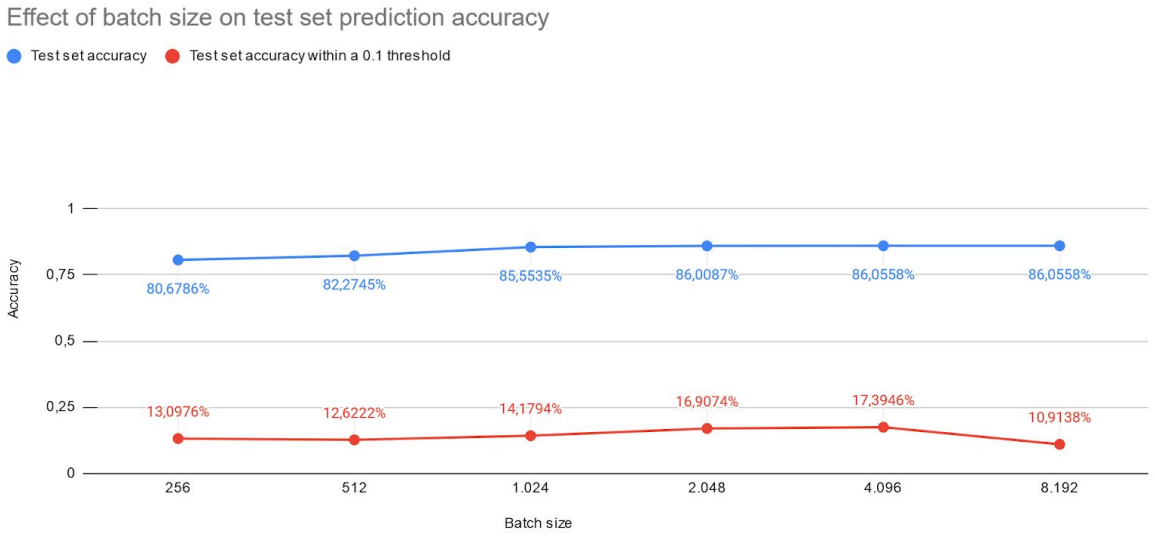
Batch Size

Άλλος ένας σημαντικός παράγοντας χρόνου και απόδοης του μοντέλου είναι το batch size. Το παρακάτω πείραμα πραγματοποιήθηκε για 1 train με learning rate 5.



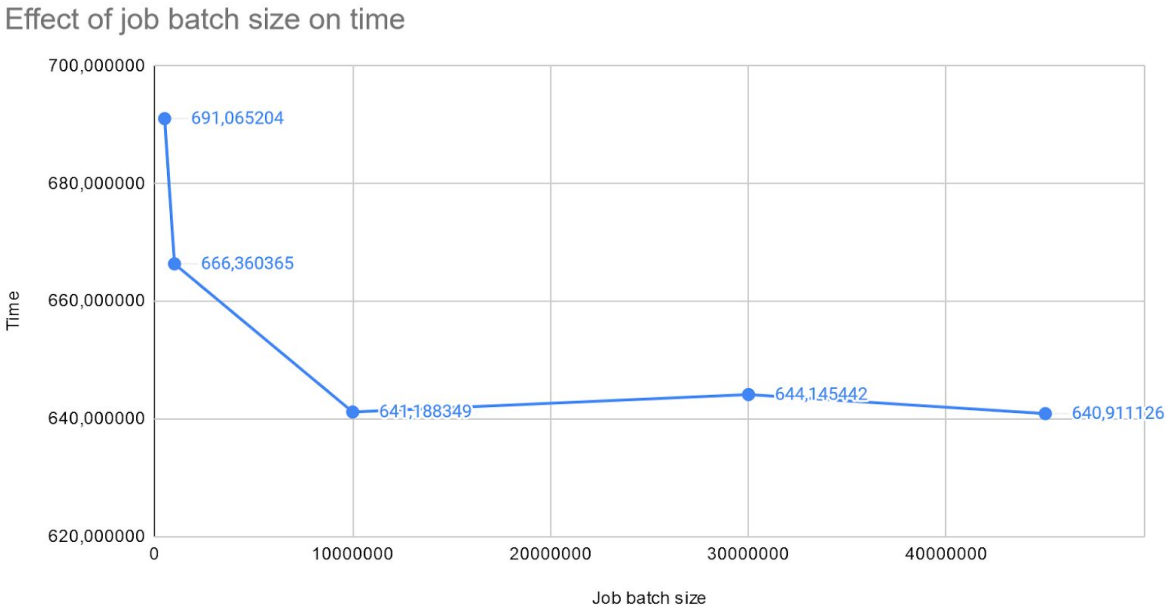
Στο ανωτέρω γράφημα εμφανίζεται η χρονική απόδοση (σε λεπτά) των διαφόρων batch size. Με εξαίρεση τον χρόνο του μεγέθους 2048, το οποίο πρόκειται για κάποιο πειραματικό σφάλμα κατά πάσα πιθανότητα λόγω κάποιας μικρής απασχόλησης του επεξεργαστή από άλλες διεργασίες, παρατηρούμε την αναμενόμενη ελάττωση του χρόνου εκτέλεσης της εκπαίδευσης. Αυτό οφείλεται στο γεγονός ότι το μοντέλο προσαρμόζει λιγότερες φορές τα βάρη, πράγμα που σημαίνει λιγότερες αναγνώσεις και γραψίματα στη μνήμη. Αυξάνοντας το μέγεθος του batch size αυξάνεται και η κατανάλωση μνήμης στο σύστημά μας ωστόσο δεν παρατηρήθηκε καμία αξιόλογη αύξηση απο τα 256 στα 8192 tasks.

Όσο για την απόδοση του μοντέλου, όπως μπορούμε να δούμε στο παρακάτω γράφημα, είχε ανοδική πορεία μέχρι το μέγεθος των 4096 και μετά μια απότομη πτώση. Το ιδανικό μέγεθος του batch size σχετίζεται σε πολύ μεγάλο βαθμό με το learning rate και το πλήθος των epochs. Με τα δικά μας δεδομένα λοιπόν το 4096 φαίνεται ιδανικό τόσο από άποψη χρόνου όσο και από ποιότητα αποτελεσμάτων και πορευόμαστε με αυτό.



Job Batch Size

Ως Job Batch Size ορίζουμε το πλήθος συγκρίσεων που δίνουμε ανά εκτέλεση από τα threads στον Job Scheduler, καθώς δεν ήταν εφικτό να φορτώσουμε και τα 900 εκατομμύρια συγκρίσεις στη μνήμη. Εδώ τίθεται σε μεγάλο βαθμό το θέμα της μνήμης που καταναλώνει από το πρόγραμμά μας. Έχοντας 16 gb διαθέσιμης μνήμης στο σύστημά μας μπορούσαμε να φτάσουμε το πολύ μεχρι 60-80 περίπου εκατομμύρια προτού αυτή γεμίσει. Ωστόσο δε χρειάστηκε να ωθήσουμε το σύστημα στα όρια του για να πάρουμε το μέγιστο.

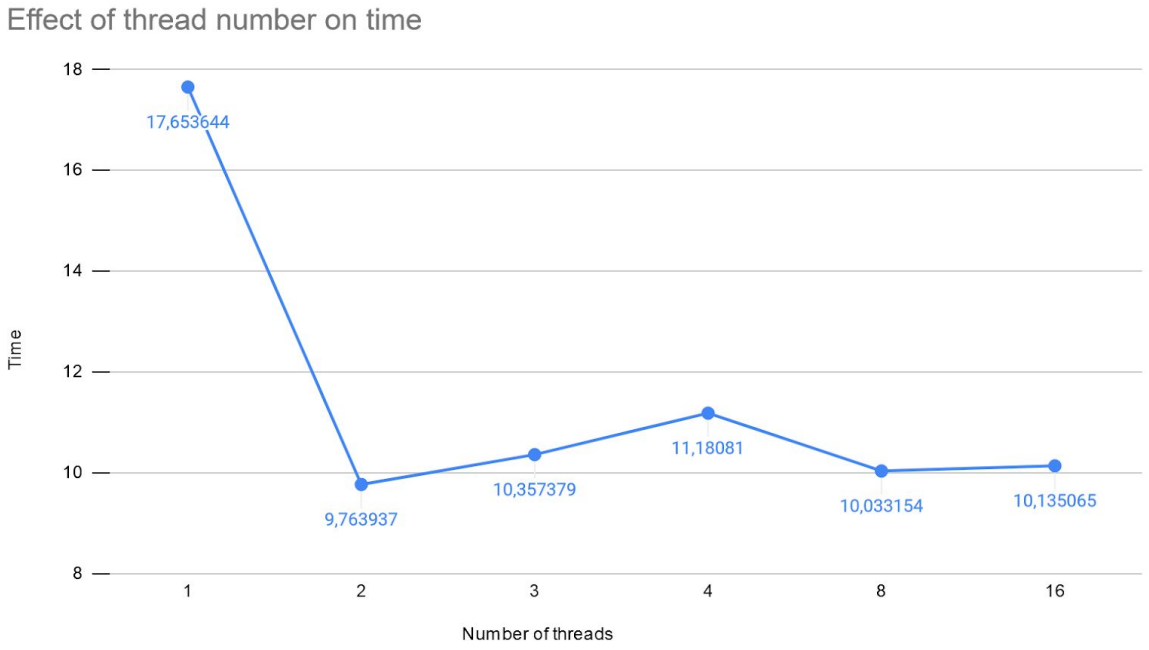


Για το παραπάνω γράφημα εκτελέστηκε 1 επανεκπαίδευση του μοντέλου μετά την αρχική για να μετρήσουμε τις επιπτώσεις του πλήθους των jobs (χρόνος σε sec) στην ταχύτητα της επανεκπαίδευσης (ο χρόνος αποτελεί το άθροισμα χρόνου εκπαίδευσης και επανεκπαίδευσης). Βλέπουμε λοιπόν ότι, μετά τα 10 εκατομμυρια jobs η κλίσης της

γραφικής παράστασης μηδενίζεται και κατ' επέκταση ο χρόνος εκτέλεσης των εκπαιδεύσεων σταθεροποιείται. Επομένως επιλέξαμε να χρησιμοποιούμε το 10 εκατομμύρια ως μέγεθος αυτής της μεταβλητής και να μη σπαταλάμε μνήμη από το σύστημα αφιλοκερδώς.

Number of Threads

Ο πολυνηματισμός επέτρεψε στο μοντέλο μας να εκτελέσει δύσκολες, χρονοβόρες, δουλειές σε πολύ καλύτερους χρόνους. Παραδόξως, πειραματιζόμενοι με το πλήθος των threads παρατηρήσαμε πως ο μοντέλο έτρεχε σαφώς πιο γρήγορα με 2 threads παρά με 4 (το μέγιστο του μηχανήματος εκτέλεσης) ή περισσότερα. Για το αποτέλεσμα αυτό υποπτευόμαστε ότι ευθύνεται ο καλύτερος συγχρονισμός τους ανά τα batch size το πλήθος tasks που τους αποδίδονταν σε τρεξίματα 1 μόνο εκπαίδευσης. Τα αποτελέσματα αυτά φαίνονται στο παρακάτω διάγραμμα.



Ωστόσο όταν κάναμε παρόμοιο πείραμα με περισσότερα από 1 trainings πήραμε πλέον τα αναμενόμενα αποτελέσματα, λόγω του πολυνηματισμού των 900 εκατομμυρίων συγκρίσεων που πρέπει να γίνουν πριν την κάθε επανεκπαίδευση. Τα 4 threads ελάττωσαν κατά πολύ, σχεδόν κατά το ήμισυ, τον συνολικό χρόνο εκτέλεσης των 2 training, διαφορά όπου θα γιγαντωνόταν για περισσότερες ακόμα επανεκπαίδευσης.

Πληθος των thread	Χρόνος με 1 επανεκπαίδευση
2	1126.348872 sec
4	613.968390 sec

Αποτελέσματα Επανεκπαίδευσης

Για τις επανεκπαιδεύσεις του μοντέλου χρησιμοποιήσαμε ως αρχικό threshold για την πιθανή ένταξη ζευγαριών στο νεο train_set το 0,025 με βήμα αύξησης 0,05. Στον πρώτο κύκλο επανεκπαίδευσης υπήρχαν 2686 πιθανά ζευγάρια για εισαγωγή στο νεο train_set και στον δεύτερο κύκλο άλλα 95665. Τα αποτελέσματα χρόνου και accuracy αναγράφονται στον παρακάτω πίνακα.

Αριθμός επανεκπαιδεύσεων	Χρόνος εκπαιδεύσεων	Test Accuracy Threshold 0.1
2	585.368313 sec	20.9978%
3	1185.552621 sec	48.888%

Σε ένα πιο ακραίο σενάριο εκτέλεσης, έχοντας αρχικό το threshold στο 0.03 και διατηρώντας το σταθερο (επειδή με μεγαλύτερο threshold υπήρχαν πολλά πιθανά ζευγάρια για εισαγωγή και το πρόγραμμα χρειαζόταν αρκετές ώρες για να τελειώσει) στο πρώτο κύκλο είχε 23185 πιθανά ζευγάρια για εισαγωγή ενώ στον δεύτερο άλλα 9658991. Ο χρόνος εκτέλεσης ήταν 2 ώρες και 15 λεπτά και το τελικό μοντέλο κατάφερε να φτάσει το test accuracy με threshold 0,1 στο 84,2%, ενώ το γενικό accuracy παρέμεινε στο 86%.