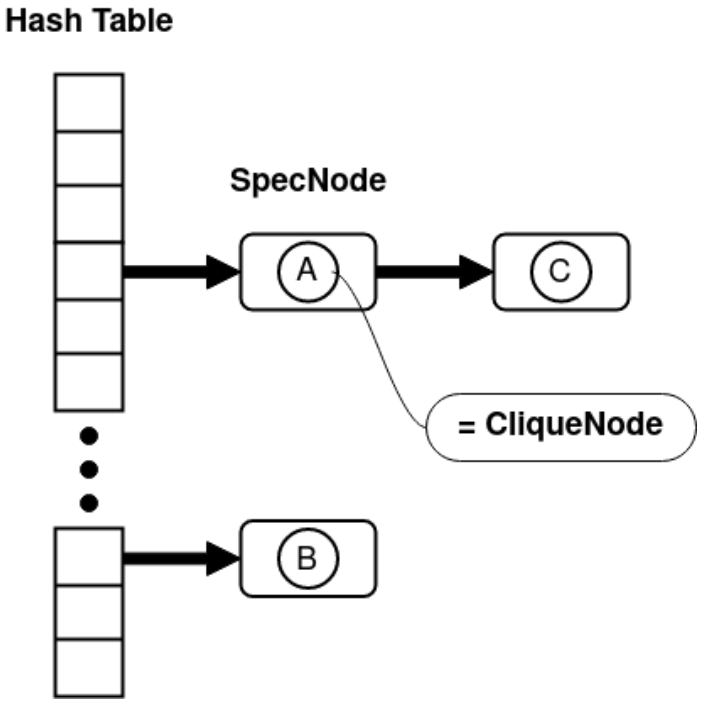
Project - Ανάπτυξη λογισμικού για Πληροφοριακά Συστήματα

Γρηγόρης Καλλίνικος - sdi1500056
Θεοδόσης Παιδάκης - sdi1500118

Δομές / Σχόλια υλοποίησης:

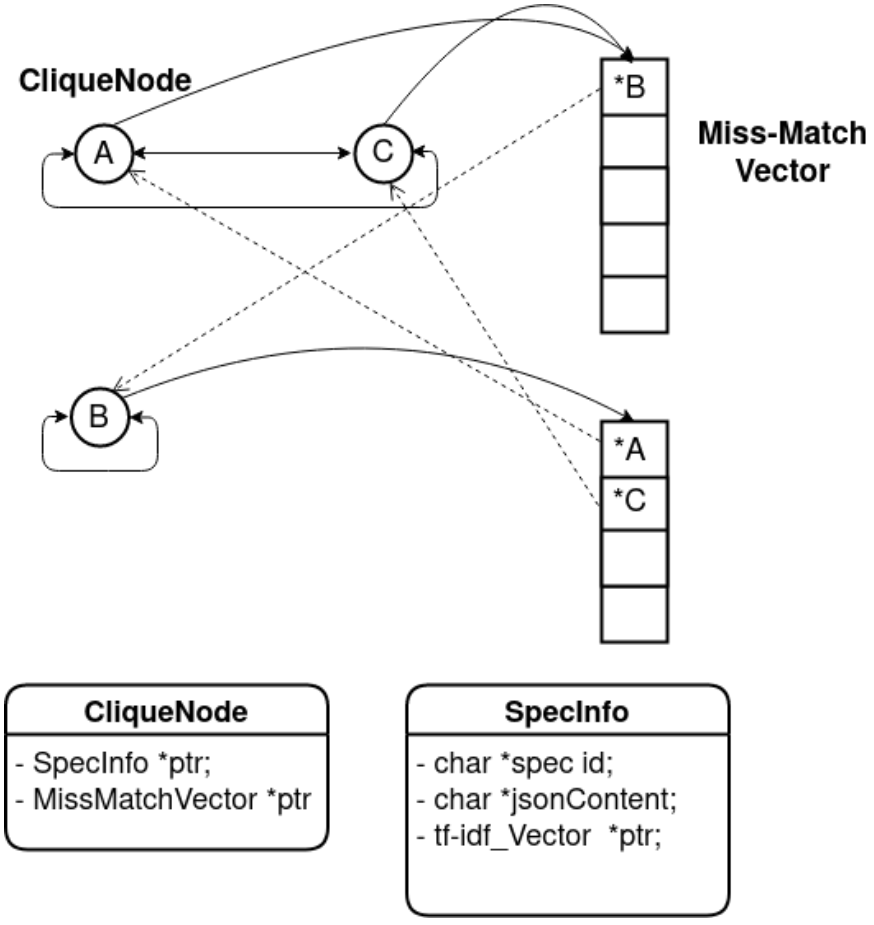
Τα αρχεία json διαβάζονται και εισχωρούνται ένα ένα στο Hash Table χρησιμοποιώντας το spec id για το hashing. Λύνουμε τα collisions με separate chaining.



Όπως φαίνεται στο παρακάτω σχήμα, η κάθε κλίκα υλοποιείται με μια κυκλική διπλά συνδεδεμένη λίστα. Κάθε μέλος την κλίκας, είναι επίσης συνδεδεμένο με έναν κοινό Miss-match vector, ο οποίος περιέχει δείκτες σε κόμβους κλίκας με τους οποίους γνωρίζουμε ότι δεν ταιριάζουν.

Αφού δημιουργηθούν οι κλίκες, κάνουμε process λέξη λέξη όλα τα json κείμενα και μέσα από μια συγκεκριμένη διαδικασία, εξάγουμε τους tf-idf vectors για κάθε spec.

Στο συγκεκριμένο παράδειγμα, τα specs A, C ταιριάζουν, ενώ δεν ταιριάζουν με το B.



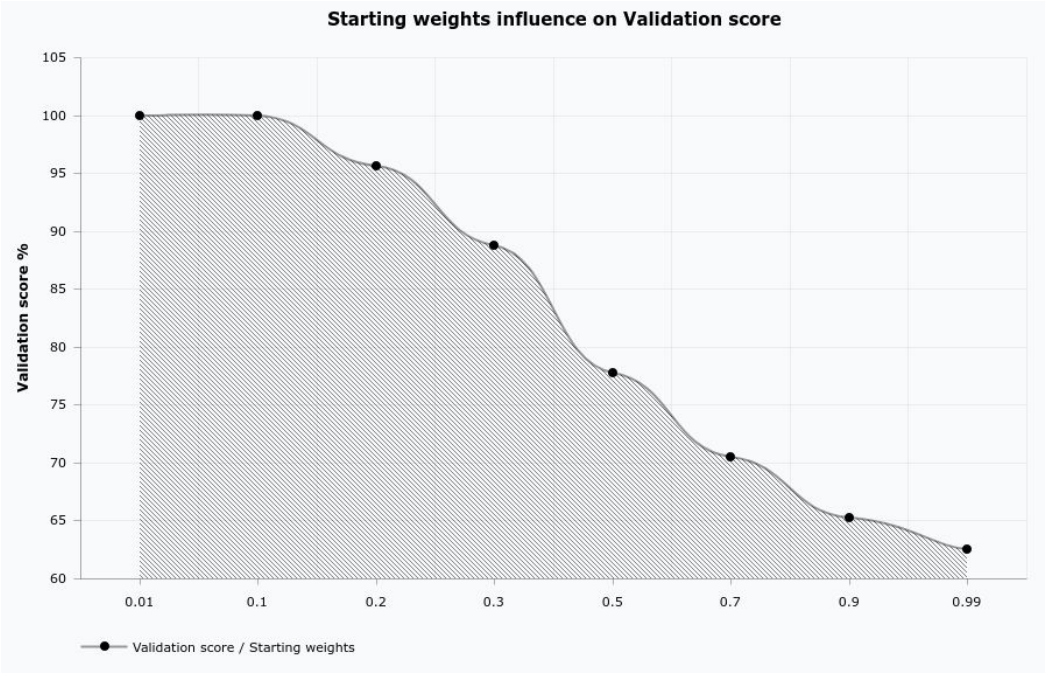
Όσων αφορά το κομμάτι της παραλληλίας, έχει υλοποιηθεί ένας scheduler, ο οποίος με την βοήθεια ουράς, αποθηκεύει και δίνει jobs στα threads. Όταν τελειώσει ένα thread την δουλειά που εκτελεί, διαβάζει από την αρχή της ουράς μία νέα μέχρι αυτές να εξαντληθούν.

Μετρήσεις / Παρατηρήσεις:

Προτού αναφερθούμε στην απόδοση των threads στο πρόγραμμά μας, θα πειραματιστούμε με διάφορες τιμές στο training κομμάτι, και θα αξιολογήσουμε το καθένα πώς επηρεάζει την εκμάθηση και γιατί.

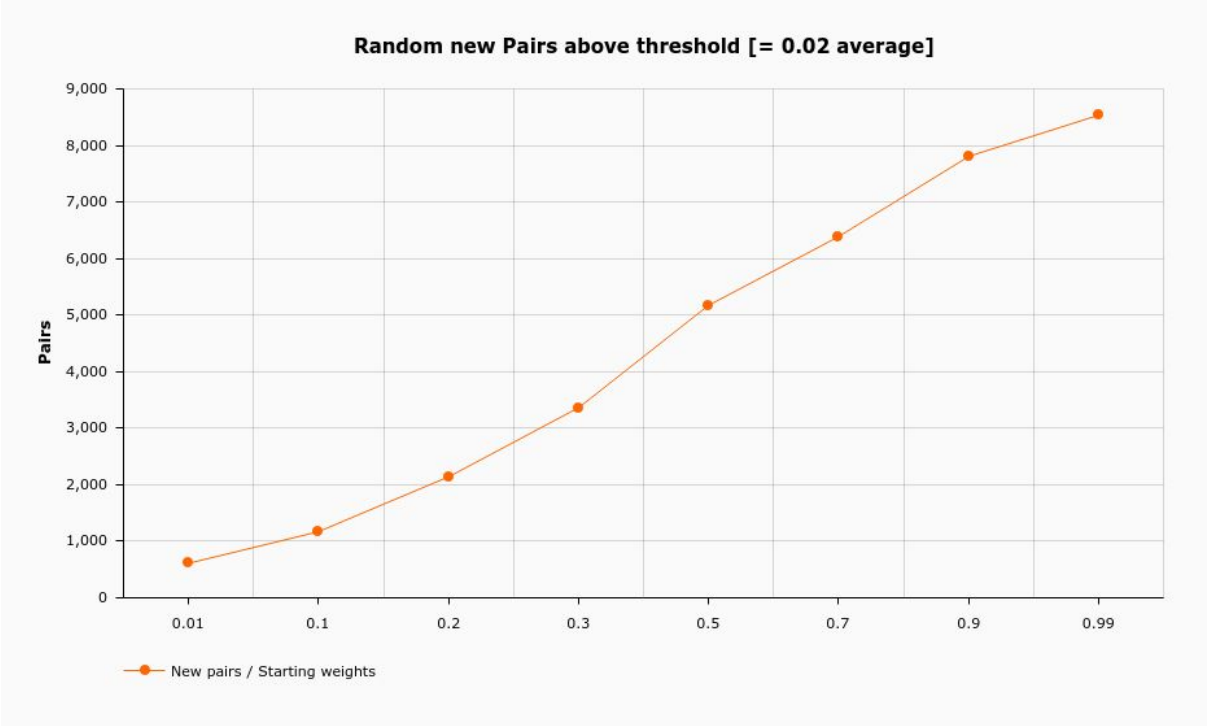
Starting weight values:

Παρατηρούμε πως κατά πολύ μεγάλο ποσοστό, στο dataset W υπερτερούν τα ζευγάρια "0". Αυτό δημιουργεί ένα bias στο μοντέλο μας, καθώς έχει πολλές περισσότερες πληροφορίες για τα κριτήρια που μπορεί να έχουν δύο Specs που δεν ταιριάζουν. Επομένως πρόκειται να τα βρίσκει με περισσότερη ευκολία και πιο επιτυχημένα.
Το παραπάνω έχει σαν αποτέλεσμα, αν δοθεί αρχική τιμή των βαρών πολύ κοντά στο μηδέν, να δημιουργείται η ψευδαίσθηση πως η συνάρτηση λογιστικής παλινδρόμησης είναι καλά εκπαιδευμένη, διότι θα μας επιστρέφει ένα υψηλό validation score.



weight train loops = 200, batch = 500, learning rate = 0.7

Βλέπουμε πόσο σημαντικά επηρεάζεται λοιπόν το validation score. Αυξάνοντας τις επαναλήψεις του training των βαρών, εξάγουμε αρκετά καλύτερα score, έχοντας όμως έναν γραμμικά μεγαλύτερο χρόνο εκτέλεσης.
Μια σημαντική παρατήρηση που κάνουμε και η οποία φαίνεται στο παρακάτω διάγραμμα, είναι πως όσο αυξάνουμε τα αρχικές τιμές βαρών, τόσο αυξάνονται και τα τυχαία νέα ζευγάρια που "περνάνε" το threshold. Αυτό σημαίνει πως η p(x), εξάγει αποτελέσματα για ένα ζευγάρι με μεγαλύτερη "αυτοπεποίθηση" για την μεταξύ τους σχέση.

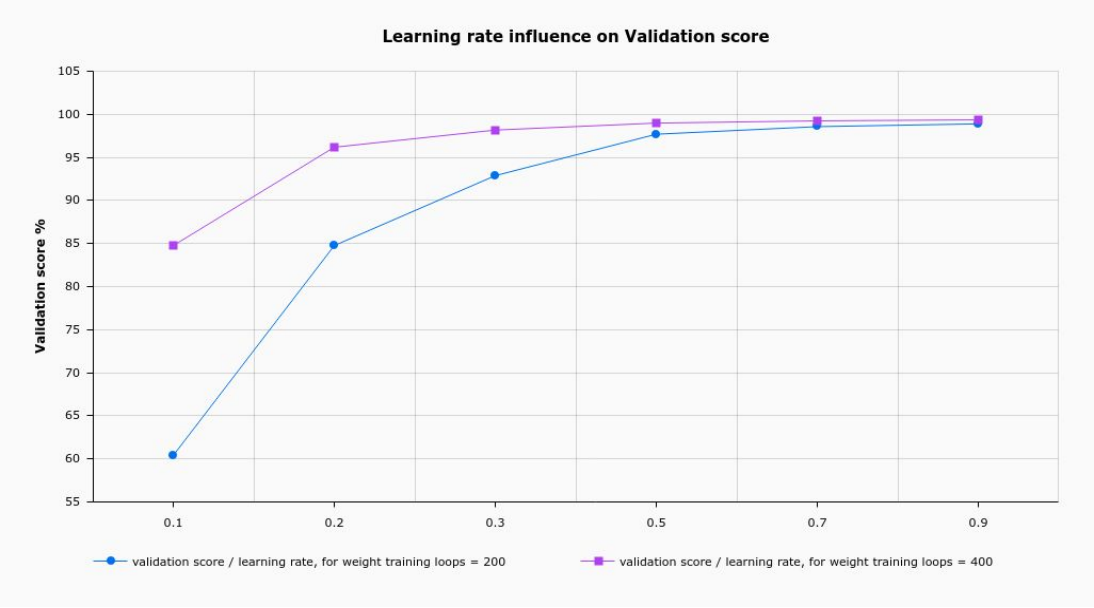


weight train loops = 200, batch = 500, learning rate = 0.7, average threshold = 0.02

Εν τέλει, μια πιο σωστή και ισορροπημένη προσέγγιση θα ήταν να αρχικοποιούμε τα βάρη με διαφορετικές τυχαίες τιμές στο διάστημα [0,1].

Learning rate:

Το learning rate θεωρείται απο πολλούς η πιο σημαντική μεταβλητή, με την οποία θα πρέπει να πειραματιστεί ο δημιουργός ενός μοντέλου, για να παράξει τα πιο συνεπή αποτελέσματα.



batch = 500, initial weights = 0.15

Όπως βλέπουμε στις παραπάνω μετρήσεις του μοντέλου μας, τα μικρότερα learning rates έχουν σαν αποτέλεσμα να αυξάνεται ο χρόνος εκμάθησης. Αυτό συμβαίνει διότι κάνουμε πολύ μικρά "βήματα" στην ενημέρωση των βαρών μας. Με τις αντίστοιχες τιμές, αν κάνουμε τις διπλάσιες επαναλήψεις εκμάθησης βαρών, τα αποτελέσματα είναι αυτά που θα αναμέναμε, αρκετά καλύτερο validation score.

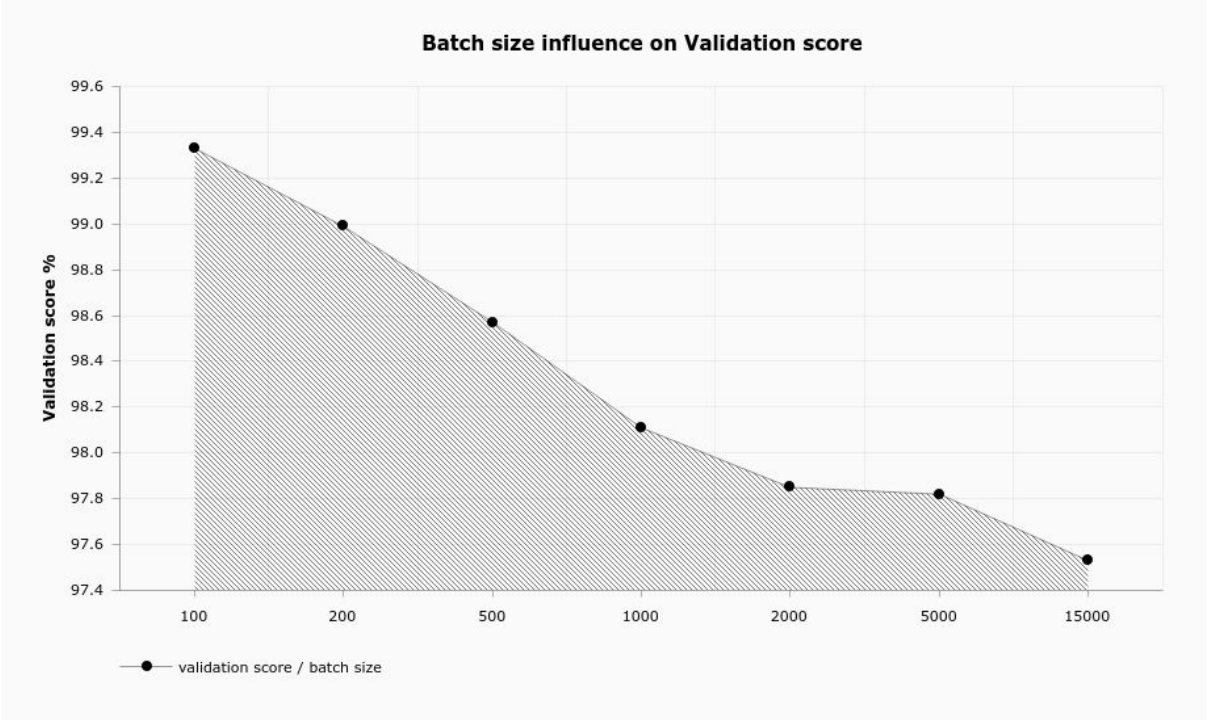
Από την άλλη, ένα μεγάλο learning rate, μπορεί να προκαλέσει δραστικές αλλαγές στα βάρη, το οποίο οδηγεί σε αποκλίνουσες συμπεριφορές στο loss function μας.

Batch size:

Το μέγεθος των batch, είναι το μέγεθος του δείγματος που δίνουμε κάθε φορά στο μοντέλο μας για εκμάθηση. Για παράδειγμα, αν έχουμε συνολικά 2000 δείγματα και batch size 150, δίνουμε τα πρώτα 150 για επεξεργασία, μετά τα επόμενα και πάει λέγοντας μέχρι να εξαντληθούν.

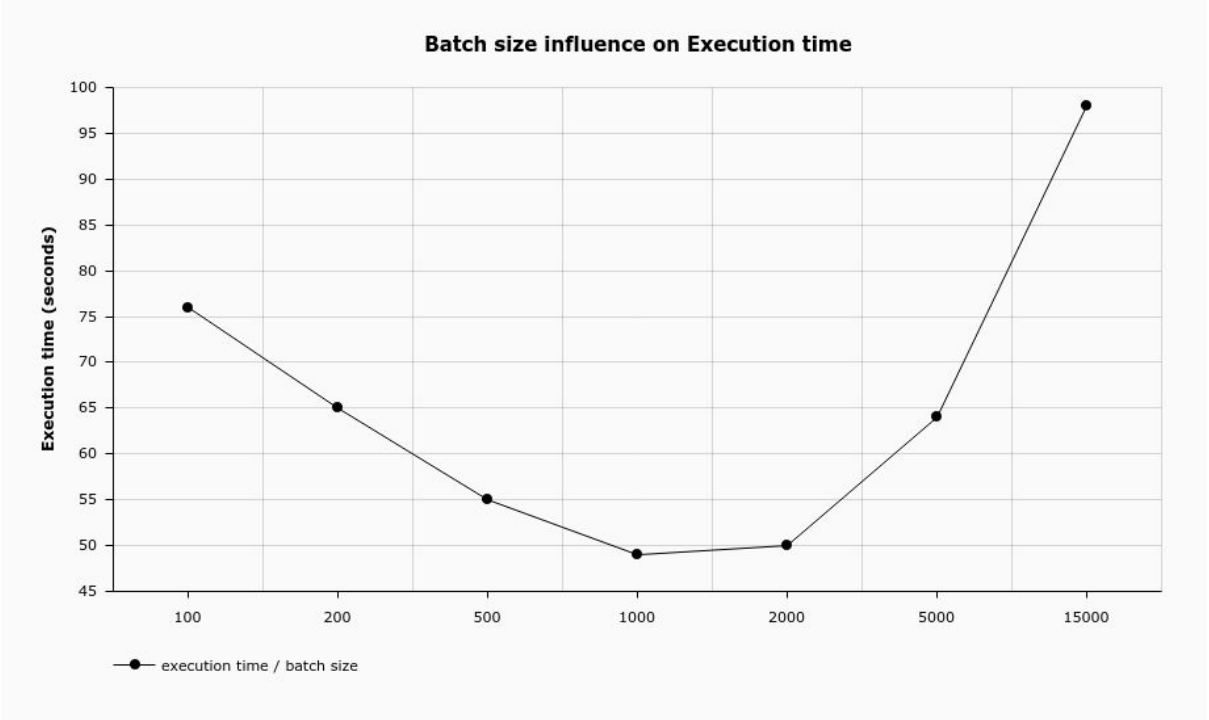
Ένα θετικό είναι ότι δεν χρειάζεται να χωρέσουμε στην μνήμη ολόκληρο το dataset, οπότε οι πόροι της ram που χρησιμοποιούνται είναι αρκετά μικρότεροι.
Επίσης, συνήθως η εκμάθηση γίνεται πιο γρήγορα, διότι ενημερώνονται τα βάρη μας μετά από κάθε batch.

Ερχόμενοι στο μοντέλο μας, δεν παρατηρήθηκε κάποια σημαντική διαφορά στην χρήση μνήμης σε σχέση με το batch size. Σε όλα τα τεστ μας, ήταν περίπου 700 MB.
Για διαφορετικά batch sizes, μετρήθηκαν ακόμα ο χρόνος και το validation score:



threads = 20, initial weights = 0.15, learning rate = 0.7, weight training loops = 200

Τα δεδομένα του παραπάνω διαγράμματος, υποδεικνύουν ότι πράγματι, χαμηλότερα batch sizes βοηθούν την εκμάθηση να γίνει αποδοτικότερα. Όσο ανεβαίνουμε σε batch size, η διαφορά αρχίζει και γίνεται λιγότερο φανερή.



threads = 20, initial weights = 0.15, learning rate = 0.7, weight training loops = 200

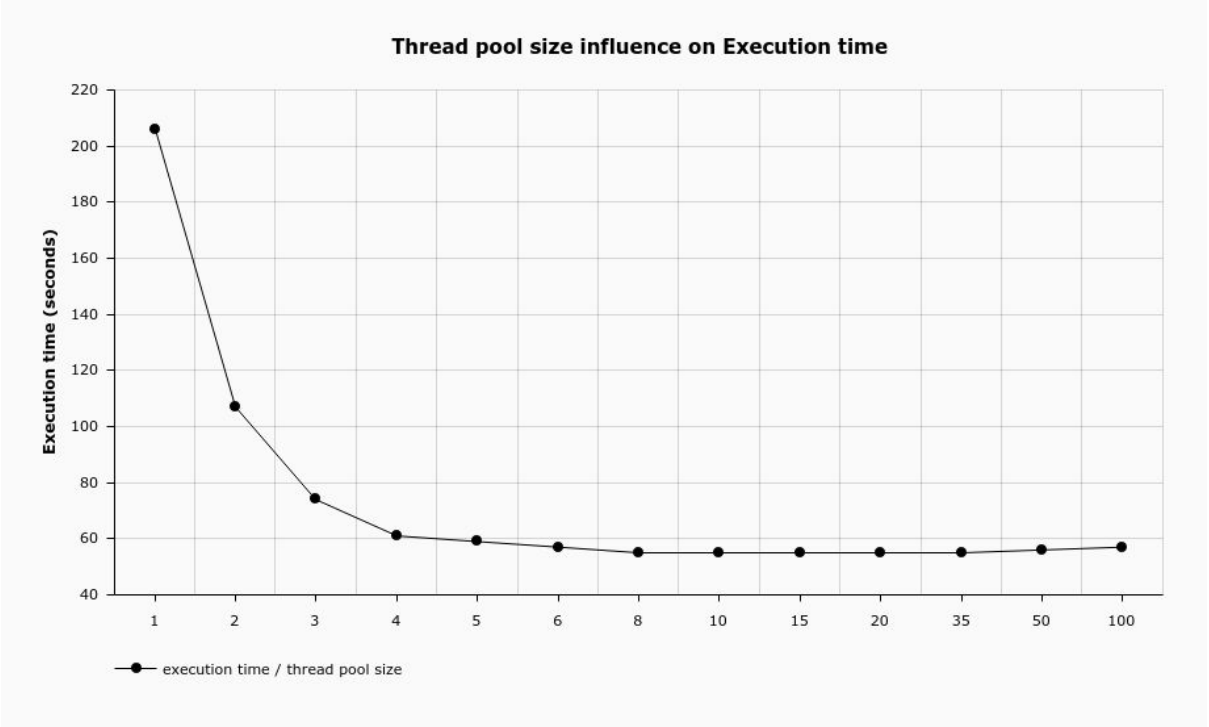
Βλέπουμε μια σχετικά μεγάλη πτώση στον χρόνο εκτέλεσης, όσο αυξάνεται το batch size. Αυτό πιθανότατα συμβαίνει, επειδή δημιουργούνται υπο-πολλαπλάσια jobs που δίνονται στα threads, οπότε υπάρχει λιγότερο thread-switching.
Μετά το 2.000 batch size, παρατηρείται το αντίθετο, αύξηση του χρόνου εκτέλεσης. Αυτό οφείλεται πιθανότατα στο γεγονός ότι μοιράζεται περισσότερη δουλειά στο κάθε thread, οπότε δεν χρησιμοποιείται προς όφελος μας αρκετά η παραλληλία.

Threads:

Στις εφαρμογές μας, χρησιμοποιούμε πολλές φορές threads, για να τις παραλληλοποιήσουμε και να γίνουν αποδοτικότερες. Πιο συγκεκριμένα, υλοποιούμε μεγάλο αριθμό threads σε εφαρμογές UI, για να είναι πιο responsive. Επίσης, σε εφαρμογές που χρειάζονται network επικοινωνία, αλλά και I/O (input output).

Στην δική μας περίπτωση, η εκμάθηση του μοντέλου είναι αρκετά CPU intensive διαδικασία, που σημαίνει ότι κατά την διάρκεια της εκτέλεσης, θα έχουμε συνεχόμενα 100% χρήση πυρήνων.
Θεωρητικά μιλώντας οπότε, αν ο επεξεργαστής μας έχει N πυρήνες, δεν πρόκειται να δούμε κάποια σημαντική βελτίωση στην απόδοση για > N+1 thread pool size.

Κάναμε ορισμένες μετρήσεις για διαφορετικά thread pool sizes:



batch = 500, initial weights = 0.15, learning rate = 0.7, weight training loops = 200

Βλέπουμε λοιπόν από τα αποτελέσματα των τεστ μας, ότι η παραπάνω θεωρία αποδεικνύεται. Το μοντέλο "έτρεξε" πάνω σε επεξεργαστή με 4 φυσικούς πυρήνες ή 8 λογικούς (hyper-threading).
Όπως αναμέναμε, η χρονική απόδοση πλατιάζει από το πλήθος των 8 threads και μετά. Σε λιγότερα threads, η διαφορά στην απόδοση είναι μεγάλη και πάντα προτιμάται.
Όσο ανεβαίνουμε σε αρκετά μεγάλους αριθμούς threads μάλιστα, παρατηρούνται χειρότεροι χρόνοι. Ο λόγος είναι ότι δημιουργείται ένα overhead από το context-switching.

<u>Σημειώσεις:</u>

- Όλες οι παραπάνω μετρήσεις, έγιναν για το medium dataset.
- Χρησιμοποιήθηκε υπολογιστής με λειτουργικό Manjaro KDE, με specifications:
  CPU: Intel i7 6700K, 4.5GHz OC
  RAM: 16 GB DDR4 2400 MHz