ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών

ΙΔΡΥΘΕΝ ΤΟ 1837

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

Τελική Αναφορά Project

Ομάδα

Κωνσταντίνος Μαραγκός, 1115201400095

Νικόλαος Σέντης, 1115200700156

Κωνσταντίνος Γκόγκας, 1115201200027

Επιμελητής

Κωνσταντίνος Κεχαγιάς

Καθηγητής

Ιωάννης Ιωαννίδης

Εισαγωγή

Η συγκεκριμένη αναφορά έχει να κάνει με το project “Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα”. Το project αυτό έχει να κάνει με μηχανική μάθηση και τις τεχνικές της, ενώ χρησιμοποιεί και άλλες τεχνικές προγραμματισμού, όπως multi-threaded programming, unit-testing κ.α. Δίνεται έναν dataset X από αρχεία μορφής .json, τα οποία περιέχουν πληροφορία σχετικά με specs φωτογραφικών μηχανών. Επίσης δίνεται και αρχείο μορφής .csv, το οποίο δείχνει για μια μερίδα των αρχείων .json, αν εννοούν την ίδια μηχανή ή όχι. Στόχος είναι, να χρησιμοποιήσουμε την πληροφορία που δίνεται, ώστε να εκπαιδεύσουμε ένα μοντέλο που θα μπορεί να κάνει πρόβλεψη εάν δύο αρχεία του dataset X εννοούν την ίδια φωτογραφική μηχανή. Για να το πετύχουμε αυτό, χρησιμοποιούμε τεχνικές μηχανικές μάθησης όπως το Bag Of words και το TF-IDF, καθώς και τη στατιστική μέθοδο της Λογιστικής Παλινδρόμησης σε συνδυασμό με τη συνάρτηση LLF (log-likelihood function).

Περιγραφή της λειτουργίας του προγράμματος

Φάση 1: Αποθήκευση των δεδομένων

Σε αυτή τη φάση αποθηκεύουμε τα δεδομένα του dataset στις δομές μας. Διαβάζουμε κάθε αρχείο και το κάνουμε parse στη δομή Specs η οποία περιέχει key-value pairs. Αφού αποθηκευτεί, γίνεται ένας “καθαρισμός” της πρότασης, όπου αφαιρούμε λέξεις μικρής σημασίας που μπορεί να εμφανίζονται συχνά (stopwords), διπλότυπες λέξεις ή λέξεις με διαφορετικό case type, σημεία στίξης κ.λ.π. Έτσι, ορίζουμε κάθε αρχείο ως ένα σύνολο λέξεων και πόσες φορές εμφανίζονται σε αυτό. Στη συνέχεια, αφού κρατήσουμε το αρχείο σε μια λίστα με όλα τα διαφορετικά αρχεία του dataset, το εισάγουμε σε έναν πίνακα κατακερματισμού, ο οποίος συμβολίζει τις διαφορετικές κλίκες του γράφου μας. Για αρχή κάθε αρχείο εισάγεται σε μια κλίκα μόνο του, και έπειτα, σε επόμενη φάση, η δομή του γράφου αλλάζει.

Φάση 2: Δημιουργία λεξιλογίου

Σ’ αυτή τη φάση, αρχικά περνάμε όλες τις λέξεις των specs, τις οποίες παίρνουμε από μια λίστα με τα specs, σε μια δομή, οποία χρησιμοποιείται για το λεξιλόγιό μας. Στη συνέχεια, αφού υπολογίσουμε το μέσο tf-idf για κάθε λέξη του λεξιλογίου μας, δημιουργούμε έναν πίνακα από λέξεις, ο οποίος αποτελεί ένα υποσύνολο με τις πιο “σημαντικές” λέξεις του λεξιλογίου. Όπου πιο σημαντικές, εννοούμε τις λέξεις με τιμή tf-idf πάνω από ένα όριο.

Φάση 3: Ανακατάταξη του γράφου & δημιουργία των συνόλων

Εδώ, αφού διαβάσουμε το .csv αρχείο με τις συσχετίσεις των προϊόντων,δημιουργούμε τις κλίκες των αρχείων μέσα στο γράφο. Αυτό σημαίνει, πως εάν π.χ. διαβαστεί a,b,1 και στη συνέχεια a,c,1, τότε προφανώς πρέπει να ισχύει και b,c,1. Άρα θα πρέπει τα a, b, c να ανήκουν στην ίδια κλίκα. Αντίστοιχα, εάν ένα στοιχείο μιας κλίκας A δείξει πως είναι αρνητικό με ένα στοιχείο μιας κλίκας B, τότε όλα τα στοιχεία μεταξύ των δύο κλικών θα πρέπει να είναι αρνητικά μεταξύ τους. Με αυτόν τον τρόπο, καταλήγουμε σε ένα νέο σύνολο συνδυασμών, που είναι το δυναμοσύνολο που προκύπτει από τους συνδυασμούς που δόθηκαν. Το σύνολο αυτό στη συνέχεια το χωρίζουμε σε 3 μέρη, 60%, 20% και 20% αντίστοιχα: ένα για το training, ένα για το validation και ένα για το testing.

Φάση 4: Εκπαίδευση του μοντέλου

Χρησιμοποιώντας το training set, δημιουργούμε για κάθε γραμμή έναν πίνακα με τις εμφανίσεις των πιο σημαντικών λέξεων στο αρχείο. Αυτό είναι και το input μας στο μοντέλο της λογιστικής παλινδρόμησης, στο οποίο ανανεώνουμε τα βάρη προς τα πίσω, χρησιμοποιώντας τη συνάρτηση υπολογισμού λάθους και τον αλγόριθμο LLF. Για να επιταχυνθεί αυτή η διαδικασία χρησιμοποιήσαμε πολυνηματικό προγραμματισμό, χωρίσαμε δηλαδή το set σε batches, και δημιουργήσαμε n threads. Κάθε thread αναλαμβάνει από ένα batch. Μόλις ολοκληρώσουν όλα τα threads, υπολογίζουμε ένα μέσο όρο από τα νέα βάρη που ανατέθηκαν και συνεχίζουμε με τα επόμενα n batches, εως ότου καλύψουμε όλο το training set. Αυτή η διαδικασία επαναλαμβάνεται για κάποιον αριθμό επαναλήψεων (epochs). Όταν τελειώσει, θεωρούμε το μοντέλο μας εκπαιδευμένο.

Φάση 5: Validation & επανεκπαίδευση

Σε αυτή τη φάση χρησιμοποιούμε τα βάρη που έχουμε καταλήξει για να εξετάσουμε την ύπαρξη conflicts. Χρησιμοποιώντας το validation set, βγάζουμε προβλέψεις για κάθε γραμμή. Στη συνέχεια προσπαθούμε να εισάγουμε τα αποτελέσματα σε ένα νέο γράφο και εξετάζουμε εάν παράγονται conflicts. Για παράδειγμα, εάν κάνουμε την πρόβλεψη a,b,1 και a,c,1 τότε, εάν κάπου στη συνέχεια του set γίνει η πρόβλεψη b,c,0 αυτό θα ήταν ένα conflict. Τα conflicts τα επιλύουμε ανάλογα με το ποιά πρόβλεψη ήταν η πιο ισχυρή: Ταξινομούμε όλες τις προβλέψεις με βάση την ισχύ τους, χρησιμοποιώντας ένα δέντρο δυαδικής αναζήτησης, και στη συνέχεια όταν προκύψει το conflict σημαίνει πως η πρόβλεψη που γίνεται τώρα είναι η πιο “ανίσχυρη” και άρα η γραμμή πρέπει να γίνει retrain ως την αντίθετη τιμή.

Φάση 6: Testing

Στην τελική αυτή φάση, ελέγχουμε το testing set, για να δούμε πόσες από τις προβλέψεις γίνονται σωστά.

Bias ως προς το μηδέν

Επειδή το dataset είχε εξαιρετικό bias ως προς το μηδέν. Για να το αντιμετωπίσουμε αυτό έπρεπε να πειράζουμε κάπως τους αριθμούς, γιατί διαφορετικά τα αποτελέσματα έβγαιναν όλα, ή σχεδόν όλα ως μηδενικά.

Ο ένας τρόπος ήταν να κάνουμε retrain τις γραμμές με 1 περισσότερες φορές για να αποκτήσει ένα bias το μοντέλο μας ως προς το ένα. Επιλέξαμε να το αποφύγουμε, επειδή δε θεωρήσαμε σωστό να αλλάζουμε το bias της εκπαίδευσης.

Ο άλλος τρόπος, αυτός που τελικά επιλέξαμε να το αντιμετωπίσουμε, ήταν ή να μειώσουμε λίγο το threshhold του τι θεωρείται σωστό ή λάθος, ή να δίνουμε ένα “bonus” στην ισχύ της απόφασης για να κάνει rank πιο ψηλά και να μη γίνεται retrain στη φάση του validation.

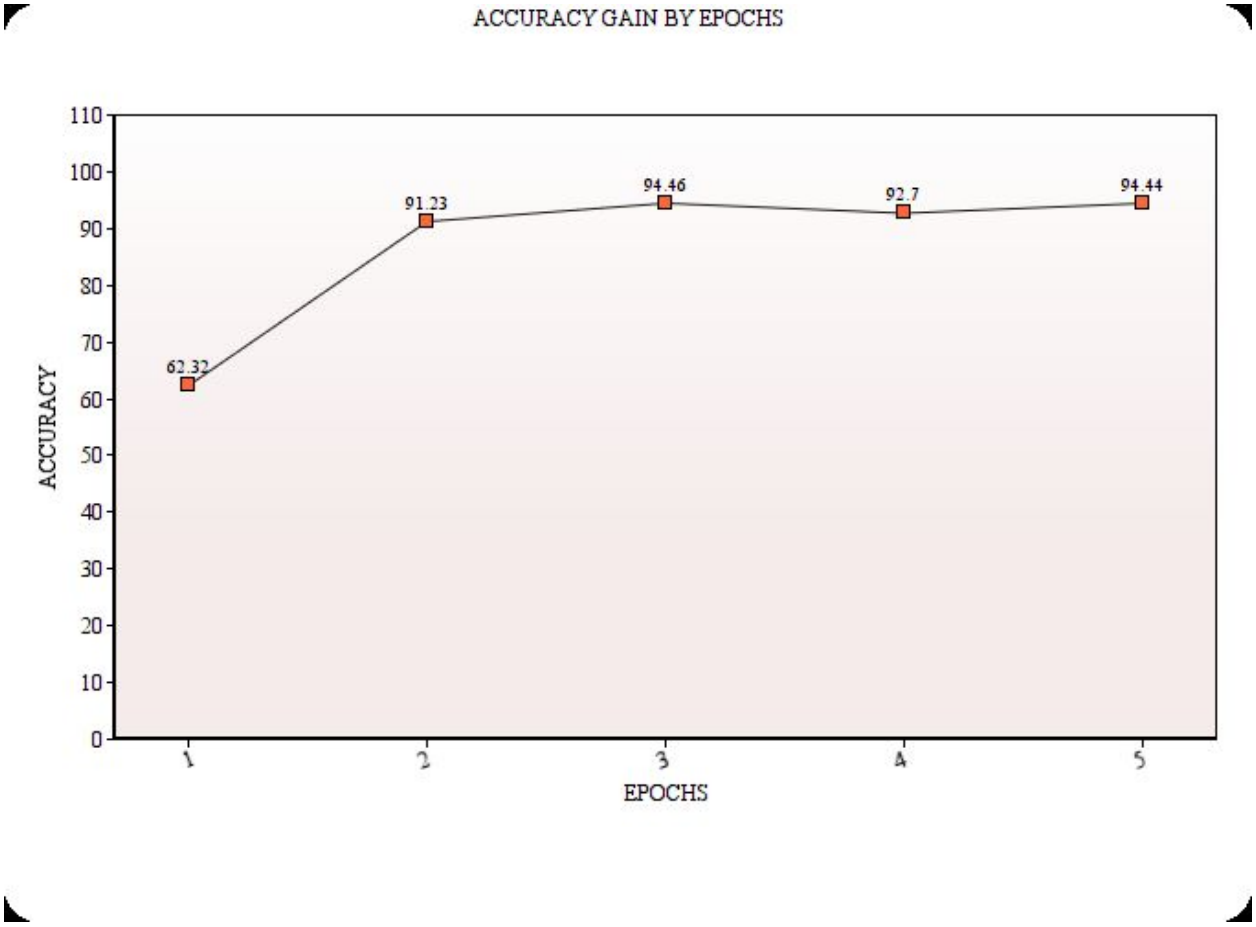
Παρατηρήσεις

Για να κάνουμε παρατηρήσεις επάνω στη λειτουργία του μοντέλου ορίσαμε ένα base case, που θα χρησιμοποιούσαμε σα βάση για τις δοκιμές μας. Οι τιμές που αλλάζαμε ήταν το margin με το οποίο δεχόμασταν λέξεις στον πίνακα των μέσων tf-idf (άρα το μέγεθος του πίνακα), το batch size, ο αριθμός των epochs και ο αριθμός των επαναλήψεων του gradient-descent μέσα στον αλγόριθμο της λογιστικής παλινδρόμησης (iterations).

Οι base case τιμές που επιλέξαμε ήταν οι εξής:

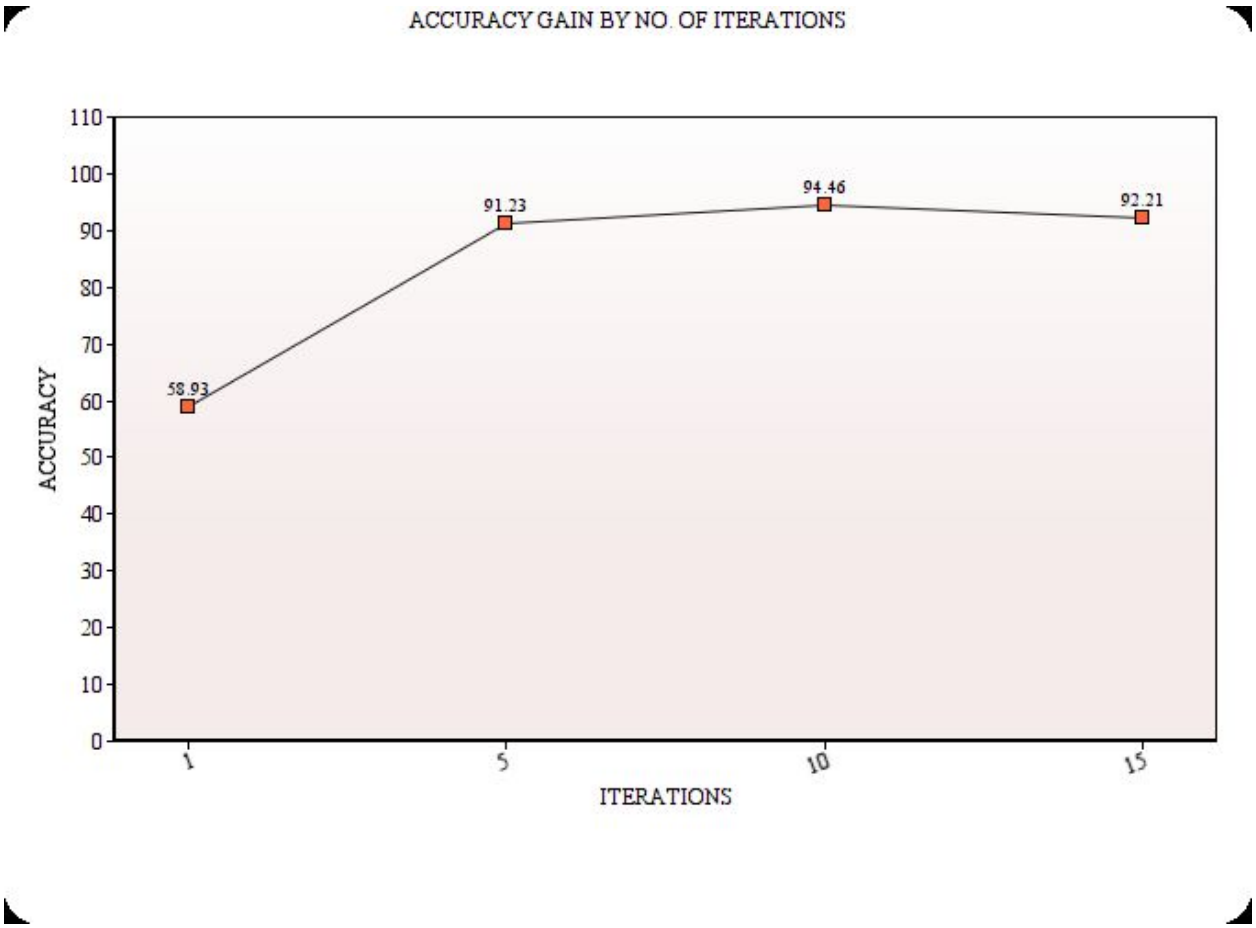
ARRAY SIZE 216
BATCH SIZE 256
ITERATIONS 5
EPOCHS 2

Epochs



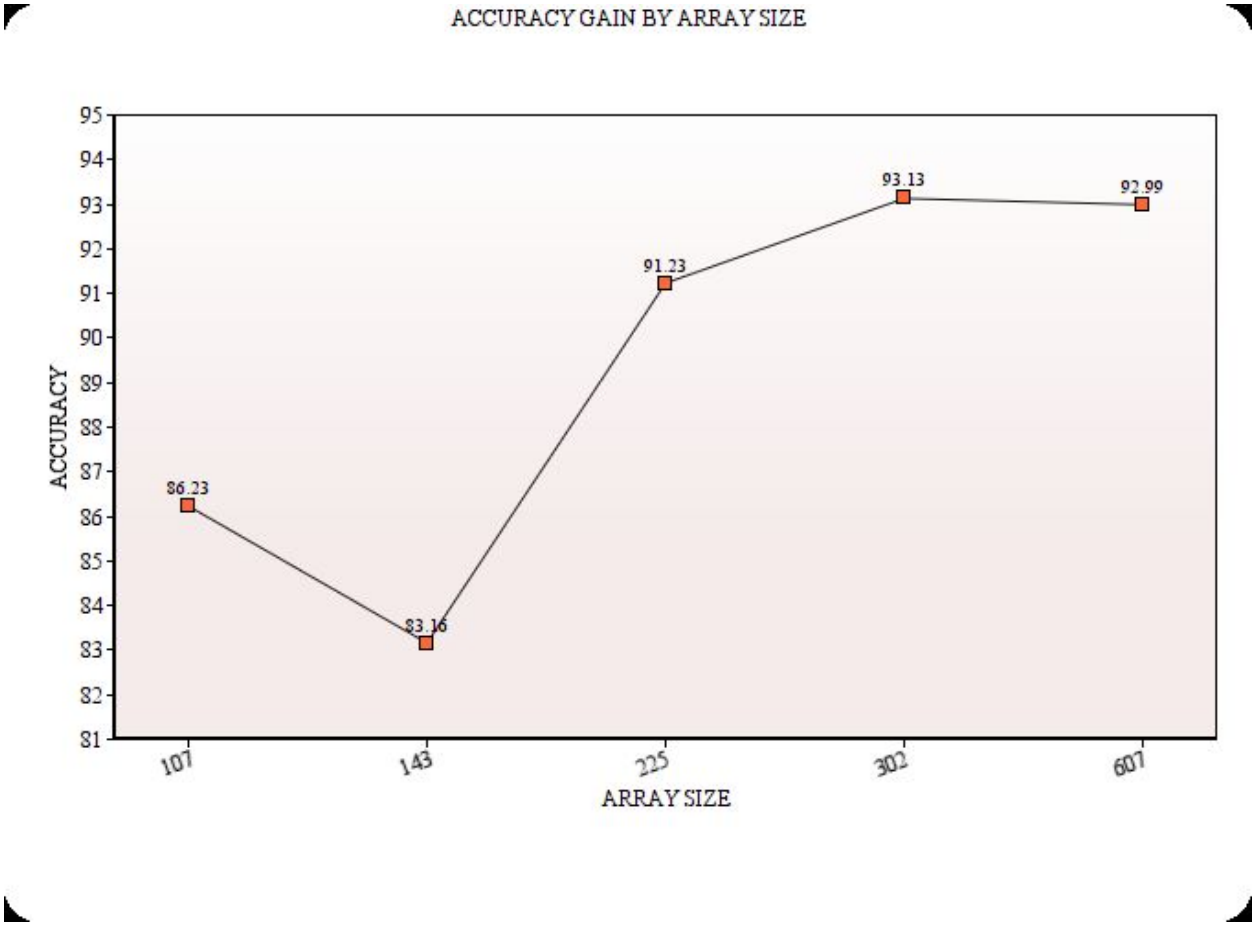
Στην περίπτωση των epochs, προφανώς όσο πιο πολλά τόσο το καλύτερο, πρέπει όμως να αναλογιζόμαστε και αν αξίζει τον έξτρα χρόνο. Παρατηρήσαμε πως μετά τις 3 επαναλήψεις σταματούσε να έχει μεγάλη επίδραση στην ακρίβεια των προβλέψεων.

Iterations



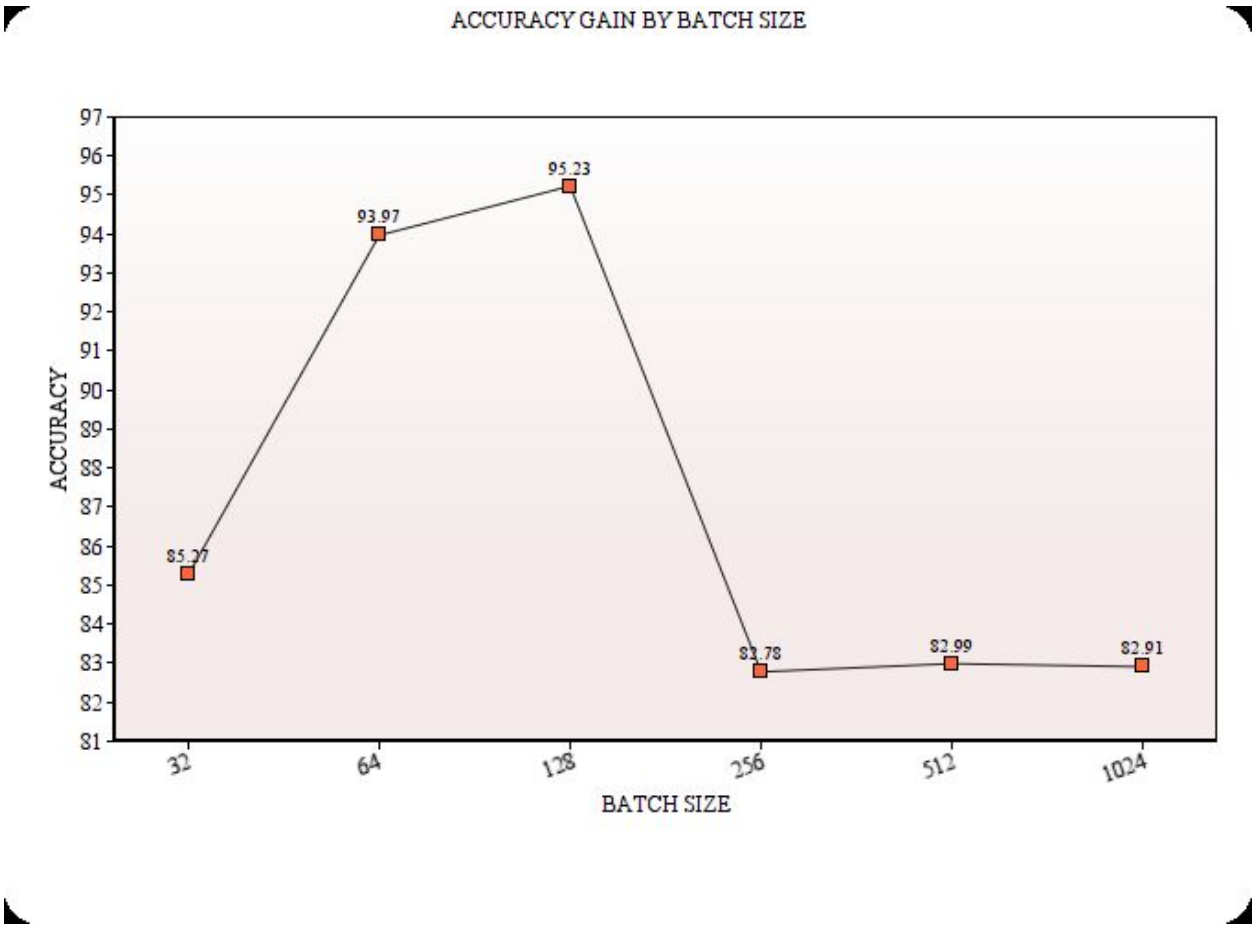
Το ίδιο που ισχύει για τα epochs ίσχυε και για τον αριθμό των επαναλήψεων του gradient descent. Σίγουρα 1 δε λειτουργεί όπως πρέπει, όμως με το learning rate 0.1 που χρησιμοποιήσαμε, από τις 5 και μετά μπορούσαμε να πούμε πως τα βάρη σύγκλιναν προς μία κατεύθυνση.

Array Size



Όσον αφορά το μέγεθος του πίνακα με τις λέξεις αρχικά ένας μεγαλύτερος πίνακας δείχνει να έχει καλύτερα αποτελέσματα. Παρατηρήσαμε όμως, πως, τουλάχιστον στο δεδομένο dataset, το accuracy στις θετικές προβλέψεις έπεφτε και μάλλον τα αποτελέσματα αυτά είναι εικονικά και προιόν ενός overfeed του dataset

Batch Size



Ενδιαφέρουσα ήταν η συμπεριφορά του μοντέλου με την αλλαγή του batch size. Απ’ ότι φαίνεται, μικρότερα batches λειτουργούν ακριβέστερα χωρίς να επηρεάζεται η ταχύτητα του προγράμματος.