ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

Ένα Ευφυές Κατανεμημένο Σχήμα Ανάθεσης Εργασιών στις Παρυφές του Δικτύου.

Χασάπης Αθανάσιος

ΑΓΜ: Μ012017162

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Κολομβάτσος Κωνσταντίνος

Λαμία, 2021

UNIVERSITY OF THESSALY

SCHOOL OF SCIENCE

INFORMATICS AND COMPUTATIONAL BIOMEDICINE

An Intelligent Distributed Task Allocation Scheme at the Edge of the Network

Chasapis Athanasios

M012017162

Master thesis

Advisor
Kolomvatsios Konstantinos

Lamia

2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ
ΚΑΤΕΥΘΥΝΣΗ

«ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ, ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ»

Ένα Ευφυές Κατανεμημένο Σχήμα Ανάθεσης Εργασιών στις Παρυφές του Δικτύου.

Χασάπης Αθανάσιος
M012017162

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Κολομβάτσος Κωνσταντίνος

Λαμία, 2021

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση τω συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο : «Ένα Ευφυές Κατανεμημένο Σχήμα Ανάθεσης Εργασιών στις Παρυφές του Δικτύου». αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο ΔΗΛΩΝ

Ημερομηνία

Υπογραφή

Ένα Ευφυές Κατανεμημένο Σχήμα Ανάθεσης Εργασιών στις Παρυφές του Δικτύου.

Χασάπης Αθανάσιος
M012017162

Τριμελής Επιτροπή:

Κολομβάτσος Κωνσταντίνος, (επιβλέπων)

Λουκόπουλος Αθανάσιος (Επικ. Καθηγητής, ΤΠΕΒ)

Τζιρίτας Νικόλαος (Επικ. Καθηγητής, ΤΠ&Τ)

ΠΕΡΙΛΗΨΗ

Η συγκεκριμένη διπλωματική εργασία αναφέρετε στην προσομοίωση ενός ευφυούς κατανεμημένου σχήματος ανάθεσης εργασιών στις παρυφές του Δικτύου. Το παραπάνω σχήμα δημιουργήθηκε με την βοήθεια ενός νευρωνικού δικτύου. Αρχικά εκπαιδεύουμε το δίκτυο μας ώστε να είναι έτοιμο να επεξεργαστεί νέα δεδομένα στα οποία δεν έχει εκπαιδευτεί, με σκοπό την επιλογή της βέλτιστης απόφασης. Δηλαδή το νευρωνικό δίκτυο αποφασίζει που θα αναθέτει τις εργασίες μέσα από μια διαδικασία μηχανικής μάθησης. Χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python και οι βιβλιοθήκες του νευρωνικού δικτύου Keras η οποία ενσωματώνεται πλέον στη βιβλιοθήκη TensorFlow. Εργαστήκαμε με το εργαλείο Jupiter το οποίο εκτελούσαμε μέσω της πλατφόρμας ανοιχτού κώδικα για την επιστήμη των δεδομένων της Python το ‘anaconda’ σε περιβάλλον Windows.

ABSTRACT

This thesis refers to the simulation of an Intelligent Distributed task allocation scheme at the edge of the network. The model was created with the help of a neural network. Initially we train our network to be ready to process new data in which it has not been trained, in order to select the optimal decision. That is, the neural network decides where to assign tasks through a machine learning oriented process. The Python programming language and the Keras neural network libraries were used, which is now integrated into the TensorFlow library. We worked with the Jupiter tool which we ran through the open source platform for Python data science 'anaconda' of environment Windows.

.

Περιεχόμενα

ΕΙΣΑΓΩΓΗ

Στο τομέα της πληροφορικής τα πάντα αλλάζουν, ότι θεωρούταν καινοτόμο και πρωτοποριακό τα προηγούμενα χρόνια, σήμερα είναι παρωχημένο. Η συγκεκριμένη επιστήμη δημιουργεί συνεχώς νέες προκλήσεις. Η πολυπλοκότητα των προβλημάτων που έχει να αντιμετωπίσει ο σύγχρονος άνθρωπος απαιτούν λύσεις που πρέπει να εξασφαλίζουν ασφάλεια τόσο της ανθρώπινης οντότητας όσο και του περιβάλλοντος του. Η πληροφορική πλέον είναι συνυφασμένη με την καθημερινότητα του ανθρώπου και έρχεται να δώσει άμεσες λύσεις στα προβλήματα που υπήρχαν και προκύπτουν καθημερινά. Βέβαια βλέπουμε νέους τομείς να ξεπηδούν, οι οποίο δεν είναι καινούργιοι απλά έχουν νέα δυναμική, είχαν αναλυθεί προγενέστερα, πολύ πριν η πληροφορική παρεισφρήσει στην καθημερινότητα μας, χωρίς όμως απτή εφαρμογή. Παράδειγμα, η τεχνητή νοημοσύνη, δεν είναι πλέον επιστημονική φαντασία, είναι κάτι αναγκαίο και χρησιμοποιείται σήμερα για την διαχείριση μεγάλου όγκου δεδομένων. Αυτή τη στιγμή, λόγω της ανάπτυξης των δικτύων, της μεγάλης επεξεργαστικής ισχύος αλλά και της τεράστιας παραγωγής και διακίνησης δεδομένων που διαχέονται στο παγκόσμιο δίκτυο, ήρθε η στιγμή να εφαρμοστούν οι καινοτόμες αυτές πρακτικές. Τεχνητή νοημοσύνη, μηχανική μάθηση, νευρωνικά δίκτυα, blockchain, edge computing είναι μερικές από αυτές.

Σε όλα τα παραπάνω έρχεται να συνδράμει η ανάπτυξη γλωσσών προγραμματισμού όπως η Python. Πάνω στη συγκεκριμένη γλώσσα δημιουργούνται βιβλιοθήκες όπως keras, TensorFlow, όπου δημιουργούν τις συνθήκες για την επίλυση προβλημάτων μέσω τις διαδικασίας της μηχανικής μάθησης με τη βοήθεια τεχνητών νευρωνικών δικτύων που προσομοιάζουν τους νευρώνες του ανθρώπινου εγκεφάλου. Επίσης, έχουν αναπτυχθεί εργαλειοθήκες για την ανάπτυξη και σύγκριση αλγορίθμων εκμάθησης ενίσχυσης που υποστηρίζουν τους πράκτορες διδασκαλίας όπως το Gym. Αναζητώντας στο διαδίκτυο την ανάπτυξη αυτών των πρωτοποριακών τεχνικών μπορεί να δει κανείς την πληθώρα των οργανισμών που ασχολούνται και εξειδικεύονται στους παραπάνω τομείς.

Η συγκεκριμένη εργασία συνδυάζει δύο από τους παραπάνω τομείς, την μηχανική μάθηση και το edge computing. Δημιουργούμε ένα πρόγραμμα σε python και χρησιμοποιούμαι τις βιβλιοθήκες Keras και TensorFlow, ενώ η συγγραφή του κώδικα έγινε στον κειμενογράφο Jupiter μέσω της πλατφόρμας anaconda. Το παραπάνω

πρόγραμμα κάνει προσομοίωση ενός ευφυούς κατανεμημένου σχήματος ανάθεσης εργασιών στις παρυφές του δικτύου. Χρησιμοποιώντας ένα τεχνητό νευρωνικό δίκτυο βρίσκουμε τη βέλτιστη ανάθεση εργασιών ώστε να εξοικονομούνται πόροι και να μειώνονται οι χρόνοι απόκρισης. Με την βοήθεια της μηχανικής μάθησης ο κάθε κόμβος μαθαίνει να διαχειρίζεται τις εργασίες που του ανατίθενται. Εκτελεί εργασίες τοπικά, εάν αυτές είναι οι πιο δημοφιλείς και αναθέτει εργασίες η οποίες απαιτούν πολλούς πόρους στο cloud ή σε άλλους κόμβους. Στην ουσία εκπαιδεύουμε τους κόμβους ώστε να τους προετοιμάσουμε σε νέα δεδομένα στα οποία δεν έχουν εκπαιδευτεί με στόχο την ορθή απόφαση και κατ' επέκταση την βέλτιστη κατανομή εργασιών.

Στο πρώτο κεφάλαιο εξηγούμε τι σημαίνει edge computing καθώς και τα χαρακτηριστικά αυτού. Στο δεύτερο κεφάλαιο αναφερόμαστε στη μηχανική μάθηση και τα τεχνητά νευρωνικά δίκτυα και πως αυτά αλληλεπιδρούν με τα δεδομένα κατά την διάρκεια της εκπαίδευσης και κατά την διαδικασία της πρόβλεψης. Στο τρίτο κεφάλαιο επισημαίνουμε τα χαρακτηριστικά της γλώσσας προγραμματισμού Python καθώς επίσης εξηγούμε γιατί είναι συνυφασμένη με την τεχνητή νοημοσύνη και τα τεχνητά νευρωνικά δίκτυα. Τέλος, το τέταρτο κεφάλαιο αποτελεί το κύριο μέρος της εργασίας. Παρουσιάζουμε τον κώδικα του προγράμματος που δημιουργήθηκε σε γλώσσα Python ο οποίος προσομοιώνει ένα δίκτυο με κόμβους στις παρυφές του και πειραματιζόμαστε στο να πετύχουμε το βέλτιστο αποτέλεσμα του νευρωνικού μας μοντέλου. Ο πειραματισμός αφορά δύο μέρη, το πρώτο μέρος ασχολείται με την ανακάλυψη των βέλτιστων εκπαιδευτικών παραμέτρων και το δεύτερο μέρος αφορά στη δοκιμασία του νευρωνικού μας δικτύου και την εξαγωγή συμπερασμάτων.

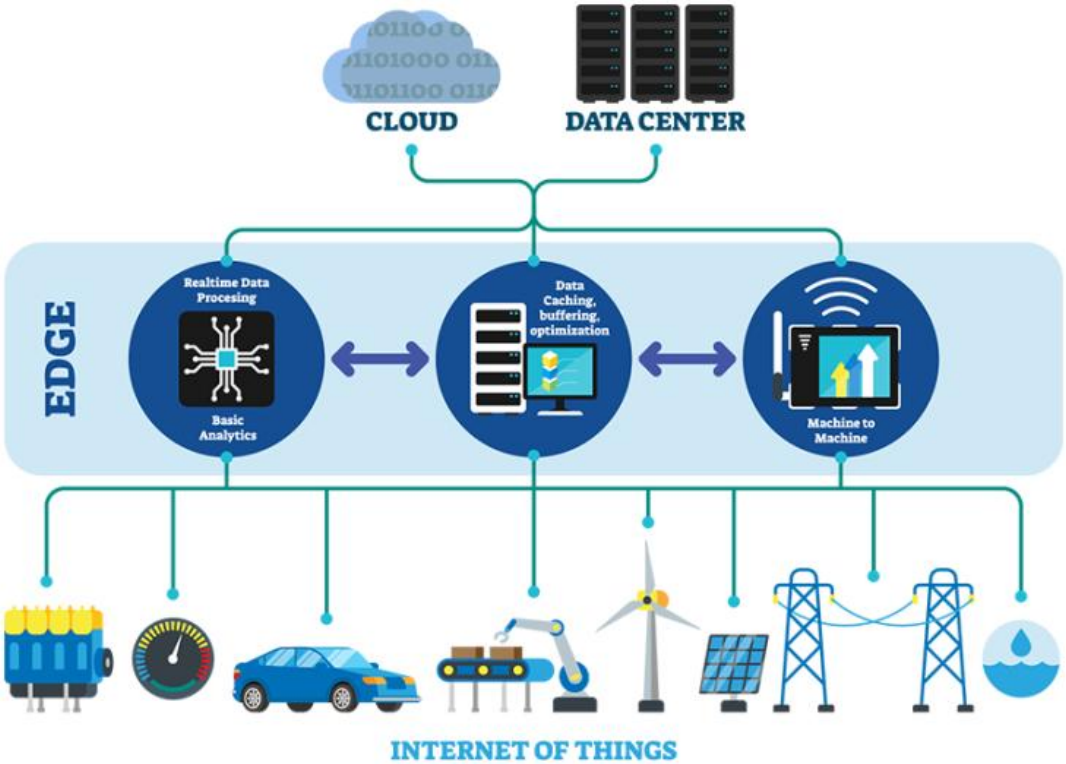
1. EDGE COMPUTING

1.1 Τι είναι το Edge computing

Το Edge computing είναι πρωτοποριακή μέθοδος αποθήκευσης και επεξεργασίας των δεδομένων πλησιέστερα στο χρήστη, έτσι οι εφαρμογές τα δεδομένα και η υπολογιστική ισχύ αποκεντρώνονται, δηλαδή δεν έχουμε την κλασική έννοια της εξυπηρέτησης των χρηστών από κεντρικούς εξυπηρετητές και όλες οι εργασίες κατανέμονται σε υπολογιστές κόμβους που βρίσκονται κοντά στην πηγή παραγωγής δεδομένων.

Η παραπάνω ιδέα έχει σα στόχο τη μείωση της μετακίνησης των δεδομένων καθώς και της απόσταση που πρέπει να διανύσει η πληροφορία, με αποτέλεσμα τη γρηγορότερη απόκριση και τη μείωση του κόστους μεταφοράς.

Εξάλλου ο υπολογισμός στις παρυφές του δικτύου είναι η χωροταξική απομάκρυνση της επεξεργασίας των δεδομένων από το κέντρο στα άκρα, ώστε να μειωθεί ο φόρτος προς τους κεντρικούς εξυπηρετητές και να αποφθεχθεί η μεταφορά του όγκου των δεδομένων από τις συσκευές που δραστηριοποιούνται στο Διαδίκτυο των Πραγμάτων (Internet of Things - IoT). Στόχος της παραπάνω τεχνικής οι μειωμένοι



Εικόνα 1 Edge Computing
innovationatwork.ieee.org

χρόνοι απόκρισης. Για να δημιουργήσουμε ένα δίκτυο όπου ο υπολογισμός θα είναι αποκεντρωμένος, θα πρέπει να παρέχουμε υπηρεσίες που παρέχονται από το cloud computing σε κέντρα δεδομένων που έχουν μετακινηθεί στα άκρα του δικτύου και εκμεταλλευόμενοι τις συσκευές του IoT και των πυλών του δικτύου.

Οι εξειδικευμένοι δρομολογητές κλάδου που τοποθετούνται στα κεντρικά σημεία του κορμού του δικτύου και λειτουργούν διαχωρίζοντας τη λειτουργία της προώθησης (forwarding) από εκείνη της δρομολόγησης (routing) καθώς επίσης και οι edge δρομολογητές δικτύου που βρίσκονται στο όριο του δικτύου συνήθως χρησιμοποιούν δυναμικές ή στατικές δυνατότητες δρομολόγησης μέσω Ethernet για την αποστολή ή λήψη δεδομένων μεταξύ του εσωτερικού και του εξωτερικού δικτύου. Η μετακίνηση των υπηρεσιών πιο κοντά σε τοποθεσίες αιχμής του δικτύου διευκολύνει την προσωρινή αποθήκευση περιεχομένου, την αποθήκευση, τη διαχείριση IoT και την παροχή υπηρεσιών, που συμβάλλουν στη βελτίωση των ποσοστών μεταφοράς και του χρόνου απόκρισης. Το δίκτυο δεδομένων υψηλής ταχύτητας αιχμής είναι το κλειδί για την επιτυχή διάθεση 5G, τα οποία απαιτούν αρχιτεκτονική υπολογιστικής αιχμής για την αποφυγή σημείων συμφόρησης. Η κορυφαία αρχιτεκτονική του δικτύου υπολογιστών μπορεί να εξοικονομήσει πόρους δικτύου αποφορτώνοντας την κίνηση του δικτύου, μειώνοντας έτσι την καθυστέρηση του δικτύου και τα σημεία συμφόρησης. Η υπηρεσία δικτύωσης γνωστών εταιρειών παρέχει περιεχόμενο εικόνας , ήχου δεδομένων σε τελικούς χρήστες με χαμηλό χρόνο απόκρισης εκμεταλλευόμενοι ένα παγκόσμιο αποκεντρωμένο δίκτυο τοποθεσιών. Εκτός από τα οφέλη μιας κατανεμημένης αρχιτεκτονικής, υπάρχουν επίσης κίνδυνοι που σχετίζονται με την αύξηση των τελικών σημείων για τη συλλογή δεδομένων και την αλληλεπίδραση με τρίτα μέρη σε όλο το παγκόσμιο δίκτυο που βρίσκεται αποκεντρωμένο. Όσο ευρύτερο είναι ένα δίκτυο, τόσο πιο ευάλωτο γίνεται στη διείσδυση. Με την άνοδο της μηχανικής μάθησης και των δικτύων που βρίσκονται στις παρυφές όπως τα IoT σε έξυπνα σπίτια και έξυπνες πόλεις, είναι πιο σημαντικό από ποτέ να εφαρμόσουμε μέτρα προστασίας στις παρυφές του δικτύου που μπορούν να βοηθήσουν στον αποκλεισμό επιθέσεων DDoS, επιθέσεων ransomware μέσω δρομολογητών, εκμεταλλεύσεων zero day και επιθέσεων μέσω εφαρμογών τρίτων. Η ασφάλεια δικτύου στο edge computing αποτελείται από περιμετρική ασφάλεια, ασφάλεια εφαρμογών, ανίχνευση απειλών, διαχείριση ευπάθειας και κύκλους επιδιόρθωσης. Το υλικό και οι υπηρεσίες που συνδυάζουν ολοκληρωμένες δυνατότητες με ολοκληρωμένες λειτουργίες ασφάλειας

δικτύου, όπως ασφαλείς πύλες ιστού και τείχη προστασίας ως υπηρεσία, προκειμένου να βοηθήσουν στην ενεργοποίηση της ασφάλειας των άκρων.

Μία υπηρεσία η οποία μοιάζει με το edge computing είναι τα CDN's. Το CDN αποτελεί ακρωνύμιο για το Content Delivery Network. Με τη CDN υπηρεσία, το στατικό περιεχόμενο ενός site αποθηκεύεται από τον κεντρικό server που φιλοξενεί την ιστοσελίδα, στο CDN δίκτυο, το οποίο αποτελείται από Servers σε όλον τον κόσμο. Τα γεωγραφικά σημεία στα οποία βρίσκονται οι Servers αυτοί, ονομάζονται PoPs (points of presence). Όταν κάποιος χρήστης, επισκεφθεί site με ενεργοποιημένη τη CDN υπηρεσία, το στατικό περιεχόμενο της σελίδας που έχει αποθηκευτεί στο CDN δίκτυο, φορτώνει από ένα server που βρίσκεται πιο κοντά γεωγραφικά σε αυτόν. Αν κάποιος χρήστης είναι π.χ. στην Αμερική, το στατικό περιεχόμενο της σελίδας θα σερβιριστεί από server του Los Angeles. Ο τρόπος που επιλέγεται ο πιο κοντινός server έχει να κάνει με τα ping times ανάμεσα στους διαθέσιμους CDN edge servers και την DNS IP διεύθυνση του τελικού χρήστη.

1.1.α Διαφορά μεταξύ δικτύου Edge και Network Core

Ο Karim Arabi, στο IEEE DAC 2014 και σε μια ομιλία του στο σεμινάριο MTL του MIT το 2015 καθόρισε όλους τους υπολογιστές, εκτός του cloud, που υπάρχουν στην άκρη του δικτύου και πιο συγκεκριμένα σε εφαρμογές όπου απαιτείται επεξεργασία δεδομένων σε πραγματικό χρόνο. Στον ορισμό του, το cloud computing λειτουργεί σε μεγάλα δεδομένα ενώ το edge computing λειτουργεί σε "στιγμιαία δεδομένα" που είναι δεδομένα σε πραγματικό χρόνο που δημιουργούνται από αισθητήρες ή χρήστες. Ένα Network Core, δηλαδή ένα κλασικό δίκτυο, λειτουργεί με τη διασύνδεση διαφόρων τμημάτων ενός δικτύου και παρέχει μια διαδρομή για την ανταλλαγή πληροφοριών εντός του κέντρου δεδομένων και μεταξύ άλλων κέντρων δεδομένων μέσω routers και switches. Η λειτουργικότητα του πυρήνα του δικτύου περιλαμβάνει έλεγχο ταυτότητας, έλεγχο κλήσεων, πύλες και κλήση υπηρεσιών. Το δίκτυο edge αποτελείται από συσκευές αιχμής, όπως υπολογιστές, σημεία πρόσβασης WiFi, switches γραφείου και rack καλωδίωσης, ακόμη και κεντρικούς υπολογιστές ή τερματικά συστήματα, που συνδέονται στις παρυφές του δικτύου. Ενώ σε ένα κλασικό δίκτυο ο πυρήνας του δικτύου βρίσκεται συχνά στα κέντρα δεδομένων, στο edge network οι πυρήνες του δικτύου είναι πολλοί και βρίσκονται στα racks καλωδίωσης.

1.2 Γιατί Edge computing

Στη σημερινή εποχή όπου το IoT έχει διαδοθεί σε ολόκληρη την υφήλιο και οι διευθύνσεις ipv4 δεν επαρκούν λόγο του πλήθους τω συσκευών, το edge computing είναι απαραίτητο περισσότερο από κάθε άλλη φορά. Όλες αυτές οι συσκευές παράγουν πλήθος δεδομένων τα οποία διαχέονται στο διαδίκτυο και απαιτούν επεξεργασία και υπολογιστική ισχύ σε κεντρικά συστήματα. Όλα αυτά τα δεδομένα απαιτούν και καταναλώνουν μεγάλο εύρος ζώνης. Παρά τις βελτιώσεις της τεχνολογίας δικτύου, τα κέντρα δεδομένων δεν μπορούν να εγγυηθούν αποδεκτούς ρυθμούς μεταφοράς και χρόνους απόκρισης, κάτι που θα μπορούσε να είναι κρίσιμη απαίτηση για πολλές εφαρμογές όπως παράδειγμα στην οδήγηση ενός οχήματος χωρίς οδηγό.

Επιπλέον, οι συσκευές στις παρυφές του δικτύου καταναλώνουν συνεχώς δεδομένα που προέρχονται από το cloud, αναγκάζοντας τις εταιρείες να δημιουργήσουν δίκτυα παράδοσης περιεχομένου για αποκέντρωση της παροχής δεδομένων και παροχής υπηρεσιών, αξιοποιώντας τη φυσική εγγύτητα με τον τελικό χρήστη.

Με παρόμοιο τρόπο, το Edge Computing έχει ως στόχο να απομακρύνει τον υπολογισμό από τα κέντρα δεδομένων προς τις παρυφές του δικτύου, αξιοποιώντας έξυπνα αντικείμενα, κινητά τηλέφωνα ή πύλες δικτύου για την εκτέλεση εργασιών και την παροχή υπηρεσιών εκ μέρους του cloud. Μετακινώντας τις υπηρεσίες στην άκρη, είναι δυνατό να παρέχεται προσωρινή αποθήκευση περιεχομένου, παροχή υπηρεσιών, αποθήκευση και διαχείριση IoT με αποτέλεσμα καλύτερους χρόνους απόκρισης και ρυθμούς μεταφοράς.

Το Edge computing περιλαμβάνει την επεξεργασία δεδομένων σε πραγματικό χρόνο κοντά στην πηγή των δεδομένων. Αυτό είναι διαφορετικό από το άκρο του δικτύου στο ότι, ενώ μπορεί να είναι ένα συστατικό του άκρου, δεν περιλαμβάνει τις άλλες συσκευές που χρησιμοποιούνται για τη μετάδοση δεδομένων από το εξωτερικό άκρο προς τον πυρήνα.

Ωστόσο, με το Edge computing, μπορούμε να απολαύσουμε βελτιωμένους χρόνους απόκρισης και εξοικονόμηση κόστους. Η συσκευή Edge computing, επειδή είναι πιο κοντά στην πηγή δεδομένων, καθιστά δυνατή την ταχύτερη μετάδοση. Μπορεί επίσης να μειώσει τις δαπάνες που σχετίζονται με τη ρύθμιση και τη συντήρηση

βασικών συσκευών, επειδή μεγάλο μέρος του υπολογιστικού φόρτου εργασίας αντιμετωπίζεται από τη φορητή υπολογιστική συσκευή.

2. ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ – ΝΕΥΡΩΝΙΚΑ ΔΚΤΥΑ

Ο γρήγορος ρυθμός προόδου στην τεχνητή νοημοσύνη οδήγησε σε προειδοποιήσεις ότι η τεχνητή νοημοσύνη αποτελεί σοβαρές απειλή για τις κοινωνίες μας, ακόμη και για την ίδια την ανθρωπότητα. Ο διάσημος επιστήμονας και πρωτοπόρος της τεχνητής νοημοσύνης Herbert Simon ανέμενε τις προειδοποιήσεις που ακούμε σήμερα σε μια παρουσίαση στο Earthware Symposium στο CMU το 2000 (Simon, 2000). Μίλησε για την αιώνια σύγκρουση μεταξύ της υπόσχεσης και των κινδύνων οποιασδήποτε νέας γνώσης, υπενθυμίζοντας μας τους ελληνικούς μύθους του Προμηθέα, τον εξιδανικευμένο ήρωα της σύγχρονης επιστήμης, ο οποίος έκλεψε φωτιά από τους θεούς προς όφελος της ανθρωπότητας και της μυθικής Πανδώρας, της οποίας το κουτί θα μπορούσε να ανοίξει με μια μικρή και αθώα δράση για να απελευθερώσει ανείπωτους κινδύνους στον κόσμο. Αν και αποδέχτηκε ότι αυτή η σύγκρουση είναι αναπόφευκτη, ο Simon μας παρότρυνε να αναγνωρίσουμε ότι ως σχεδιαστές του μέλλοντός μας και όχι απλώς θεατές, οι αποφάσεις που λαμβάνουμε μπορούν να γείρουν την κλίμακα υπέρ του Προμηθέα. Αυτό ισχύει ασφαλώς για την ενίσχυση της μάθησης, η οποία μπορεί να ωφελήσει την κοινωνία, αλλά μπορεί επίσης να παράγει ανεπιθύμητα αποτελέσματα εάν εφαρμοστεί απρόσεκτα. Έτσι, η ασφάλεια των εφαρμογών τεχνητής νοημοσύνης που περιλαμβάνει την ενίσχυση της μάθησης είναι ένα θέμα που χρήζει προσεκτικής προσοχής. (*Reinforcement Learning An introduction – Richard S. Sutton and Andrew G Barto second edition*).

2.1 Μηχανική Μάθηση

2.1.α Τι είναι η Μηχανική Μάθηση

Ενώ η τεχνητή νοημοσύνη (AI) είναι η ευρεία επιστήμη της μίμησης των ανθρώπινων ικανοτήτων, η μηχανική μάθηση είναι ένα συγκεκριμένο υποσύνολο της AI που εκπαιδεύει μια μηχανή πώς να μάθει. Η μηχανική μάθηση είναι μια μέθοδος ανάλυσης δεδομένων που αυτοματοποιεί την ανάπτυξη αναλυτικών μοντέλων. Είναι ένας κλάδος της τεχνητής νοημοσύνης που βασίζεται στην ιδέα ότι τα συστήματα μπορούν να μάθουν από δεδομένα, να προσδιορίσουν μοτίβα και να λάβουν αποφάσεις με ελάχιστη ανθρώπινη παρέμβαση.

Με άλλα λόγια η μηχανική εκμάθηση είναι η πρακτική της χρήσης αλγορίθμων για την ανάλυση δεδομένων, την εκμάθηση από αυτά τα δεδομένα και, στη συνέχεια, να κάνει έναν προσδιορισμό ή πρόβλεψη για νέα δεδομένα.

Λόγω των νέων τεχνολογιών των υπολογιστών, η μηχανική μάθηση σήμερα δεν μοιάζει με τη μηχανική εκμάθηση του παρελθόντος. Γεννήθηκε από την αναγνώριση προτύπων και τη θεωρία ότι οι υπολογιστές μπορούν να μάθουν χωρίς να προγραμματίζονται για την εκτέλεση συγκεκριμένων εργασιών. Οι ερευνητές που ενδιαφέρονται για την τεχνητή νοημοσύνη ήθελαν να δουν αν οι υπολογιστές θα μπορούσαν να μάθουν από τα δεδομένα. Η επαναληπτική πτυχή της μηχανικής μάθησης είναι σημαντική επειδή καθώς τα μοντέλα εκτίθενται σε νέα δεδομένα, είναι σε θέση να προσαρμοστούν ανεξάρτητα. Μαθαίνουν από προηγούμενους υπολογισμούς να παράγουν αξιόπιστες, επαναλαμβανόμενες αποφάσεις και αποτελέσματα. Είναι μια επιστήμη που δεν είναι καινούργια - αλλά έχει αποκτήσει νέα δυναμική.

Ενώ πολλοί αλγόριθμοι μηχανικής μάθησης υπάρχουν εδώ και πολύ καιρό, η ικανότητα αυτόματης εφαρμογής πολύπλοκων μαθηματικών υπολογισμών σε μεγάλα δεδομένα - ξανά και ξανά, γρηγορότερα και ταχύτερα - είναι μια πρόσφατη εξέλιξη.

2.1.β Κατηγορίες Μηχανικής Μάθησης

Οι προσεγγίσεις μηχανικής μάθησης χωρίζονται παραδοσιακά σε τρεις ευρείες κατηγορίες, ανάλογα με τη φύση του "σήματος" ή "ανατροφοδότησης" που είναι διαθέσιμο στο σύστημα μάθησης:

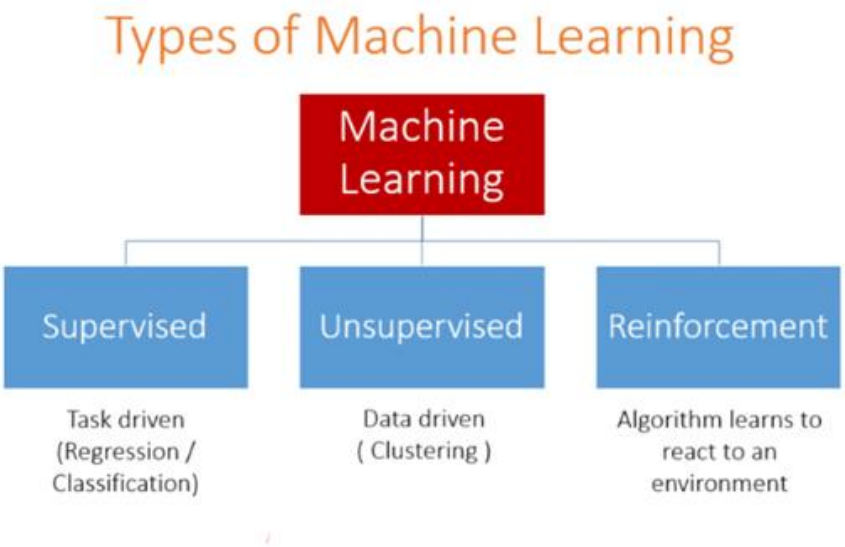
Εποπτευόμενη εκμάθηση: Ο υπολογιστής παρουσιάζεται με παραδείγματα εισόδων και τις επιθυμητές εξόδους, που δίνονται από έναν "δάσκαλο", και ο στόχος είναι να μάθει το μοντέλο έναν γενικό κανόνα που χαρτογραφεί τις εισόδους στις εξόδους.

Μη επιτηρούμενη μάθηση: Δεν δίνονται ετικέτες στον αλγόριθμο εκμάθησης, αφήνοντάς τον από μόνος του να βρει δομή στην είσοδό του. Η μη επιτηρούμενη μάθηση μπορεί να είναι ένας στόχος από μόνη της (ανακάλυψη κρυφών μοτίβων σε δεδομένα) ή μέσο προς το τέλος (λειτουργία εκμάθησης).

Εκμάθηση ενίσχυσης: Ένα πρόγραμμα υπολογιστή αλληλεπιδρά με ένα δυναμικό περιβάλλον στο οποίο πρέπει να εκτελεί ένα συγκεκριμένο στόχο (όπως οδήγηση οχήματος ή παιχνίδι εναντίον αντιπάλου). Καθώς πλοηγεί στο χώρο του

προβλήματος, το πρόγραμμα παρέχει ανατροφοδότηση ανάλογη με τις ανταμοιβές, την οποία προσπαθεί να μεγιστοποιήσει.

Μεταξύ της εποπτευόμενης και της μη επιτηρούμενης μάθησης έχουμε την ημι-επιτηρούμενη μάθηση , όπου εκεί δίνονται στον πράκτορα μέρος των δεδομένων και απουσιάζει μέρος των στόχων. Μία άλλη περίπτωση μηχανικής μάθησης η αναπτυξιακή, συντελείτε όταν καλούμε το μοντέλο να δημιουργήσει δικές του επαγωγικές μεθόδους από προηγούμενη εμπειρία. Η συγκεκριμένη λειτουργία έχει δημιουργηθεί για την εκμάθηση των ρομπότ. Έχουν αναπτυχθεί και άλλες προσεγγίσεις που δεν ταιριάζουν σε αυτή την τριπλή κατηγοριοποίηση και μερικές φορές περισσότερες από μία χρησιμοποιούνται από το ίδιο το σύστημα μηχανικής μάθησης. Από το 2020, η ενισχυτική μάθηση (Reinforcement learning) έχει γίνει η κυρίαρχη προσέγγιση στον τομέα της μηχανικής μάθησης.



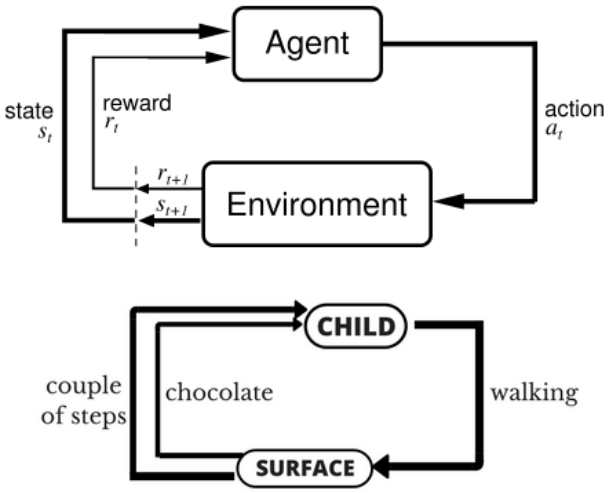
Εικόνα 2 Κατηγορίες Μηχανικής Μάθησης (www.analyticsvidhya.com)

2.1.γ Ενισχυτική Μάθησης (Reinforcement learning)

Η ενισχυτική μάθηση είναι ένας τομέας της μηχανικής μάθησης που ασχολείται με τον τρόπο με τον οποίο οι πράκτορες λογισμικού πρέπει να αναλαμβάνουν ενέργειες σε ένα περιβάλλον έτσι ώστε να μεγιστοποιείται κάποια έννοια της σωρευτικής ανταμοιβής. Λόγω της γενικότητάς του, το πεδίο μελετάται σε πολλούς άλλους κλάδους, όπως η θεωρία παιχνιδιών, η θεωρία ελέγχου, η έρευνα λειτουργίας, η θεωρία πληροφοριών, η βελτιστοποίηση με βάση την προσομοίωση, τα συστήματα πολλαπλών παραγόντων, η συλλογή πληροφοριών, στατιστικές και γενετικοί

αλγόριθμοι. Στη μηχανική μάθηση, το περιβάλλον αντιπροσωπεύεται συνήθως ως διαδικασία απόφασης Markov (MDP). Πολλοί αλγόριθμοι ενισχυτικής μάθησης χρησιμοποιούν δυναμικές τεχνικές προγραμματισμού. Οι αλγόριθμοι ενισχυτικής μάθησης δεν προϋποθέτουν γνώση ενός ακριβούς μαθηματικού μοντέλου του MDP και χρησιμοποιούνται όταν τα ακριβή μοντέλα είναι ανέφικτα. Οι αλγόριθμοι ενισχυτικής μάθησης χρησιμοποιούνται σε αυτόνομα οχήματα ή στην εκμάθηση ενός παιχνιδιού εναντίον ενός ανθρώπινου αντιπάλου.

Η ενισχυτική μάθηση μαθαίνει τι να κάνει και πώς να χαρτογραφεί τις καταστάσεις στην κάθε ενέργεια που κάνει ο μαθητής. Το τελικό αποτέλεσμα είναι να μεγιστοποιηθεί το σήμα αριθμητικής ανταμοιβής. Ο εκπαιδευόμενος δεν λέει ποια ενέργεια πρέπει να κάνει, αλλά πρέπει να ανακαλύψει ποια ενέργεια θα αποδώσει τη μέγιστη ανταμοιβή.



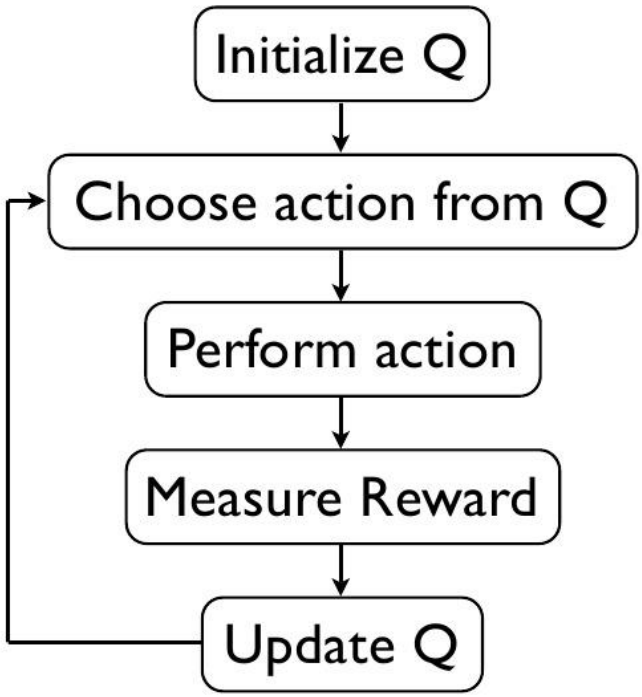
Εικόνα 3 Παράδειγμα Ενίσχυση Μάθησης (www.analyticsvidhya.com)

Ας δούμε έναν ψευδοκώδικα που αντιστοιχεί στον αλγόριθμο Q-learning:

- Αρχικοποιήστε τον πίνακα τιμών " Q (s, a)" .
- Παρατηρήστε τις τρέχουσας κατάστασης « s» .
- Επιλέξτε μια ενέργεια "a" για την κατάσταση αυτή βάσει μιας από τις πολιτικές επιλογής δράσης
- Πάρτε την ενέργεια, και παρατηρήστε την ανταμοιβή « r» , καθώς και τη νέα κατάσταση « s» .

- Ενημερώστε την τιμή για την κατάσταση χρησιμοποιώντας την παρατηρούμενη ανταμοιβή και τη μέγιστη δυνατή ανταμοιβή για την επόμενη κατάσταση.
- Ρυθμίστε την κατάσταση στην νέα κατάσταση και επαναλάβετε τη διαδικασία μέχρι να φτάσετε σε κατάσταση τερματισμού. (*www.analyticsvidhya.com*)

Μια απλή περιγραφή της Q-learning μπορεί να συνοψιστεί ως εξής:



Εικόνα 4 Περιγραφή Q learning (www.analyticsvidhya.com)

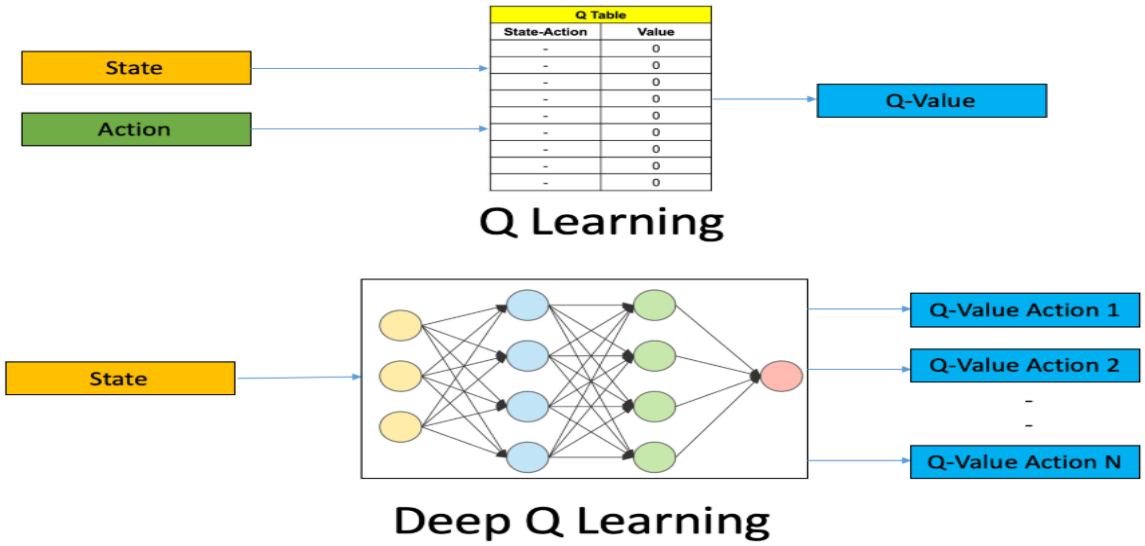
Το Q-learning είναι ένας απλός αλλά αρκετά ισχυρός αλγόριθμος για τη δημιουργία ενός cheat sheet για τον εκπαιδευόμενο μας. Αυτό βοηθά τον εκπαιδευόμενο να καταλάβει ακριβώς ποια ενέργεια θα εκτελέσει.

Τι γίνεται όμως αν αυτό το cheat sheet είναι πολύ μεγάλο; Έτσι ένα περιβάλλον με 100.000 καταστάσεων και 1.000 ενεργειών ανά κατάσταση θα δημιουργούσε έναν πίνακα 100 εκατομμυρίων κελιών.

Είναι αρκετά σαφές ότι δεν μπορούμε να συμπεράνουμε την τιμή Q των νέων καταστάσεων από τις ήδη εξερευνημένες καταστάσεις. Αυτό παρουσιάζει δύο προβλήματα:

- Πρώτον, η ποσότητα μνήμης που απαιτείται για αποθήκευση και ενημέρωση αυτού του πίνακα θα αυξανόταν καθώς ο αριθμός των καταστάσεων αυξάνεται
- Δεύτερον, ο χρόνος που απαιτείται για την εξερεύνηση κάθε κατάστασης για τη δημιουργία του απαιτούμενου πίνακα Q θα ήταν μη ρεαλιστικός

Για να αποφύγουμε τα δύο παραπάνω προβλήματα προσεγγίσουμε αυτές τις τιμές Q με μοντέλα μηχανικής μάθησης όπως ένα νευρωνικό δίκτυο.



Εικόνα 5 Μάθηση - Βαθιά Μάθηση (www.analyticsvidhya.com)

Στη βαθιά μάθηση, χρησιμοποιούμε ένα νευρωνικό δίκτυο για να προσεγγίσουμε την Q συνάρτηση (Q-value). Η κατάσταση δίνεται ως είσοδος και η τιμή Q όλων των πιθανών ενεργειών δημιουργείται ως έξοδος. Η σύγκριση μεταξύ Q-learning και deep Q-learning απεικονίζεται στην εικόνα 5.

2.2 Τεχνητά Νευρωνικά Δίκτυα

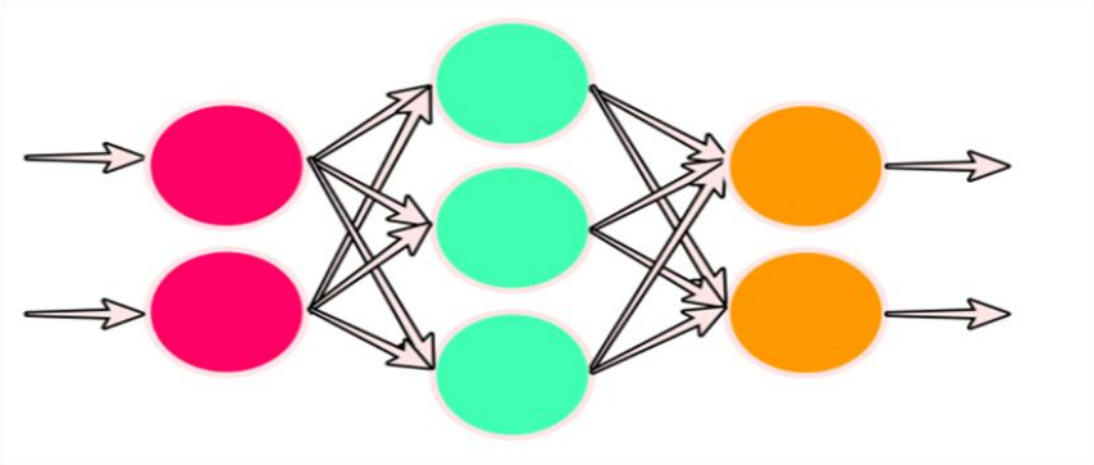
Η βαθιά μάθηση είναι ένα υπό-πεδίο της μηχανικής μάθησης που χρησιμοποιεί αλγόριθμους εμπνευσμένους από τη δομή και τη λειτουργία των νευρωνικών δικτύων του εγκεφάλου. Τα τεχνητά νευρωνικά δίκτυα είναι ένα υπολογιστικό σύστημα που αποτελείται από μια συλλογή συνδεδεμένων μονάδων που ονομάζονται νευρώνες και οργανώνονται σε αυτό που ονομάζουμε στρώματα.

Συχνά χρησιμοποιούμε επίσης άλλους όρους για να αναφερθούμε στα τεχνητά νευρωνικά δίκτυα (ANN). Στον τομέα της βαθιάς μάθησης, ο όρος τεχνητό νευρωνικό δίκτυο (ANN) χρησιμοποιείται εναλλακτικά με τα εξής:

- ✓ net
- ✓ neural net
- ✓ model

Τα ANN κατασκευάζονται χρησιμοποιώντας αυτό που ονομάζουμε νευρώνες. Οι νευρώνες σε ένα ANN οργανώνονται σε αυτό που αποκαλούμε στρώματα, layers. Τα layers, επίπεδα εντός ενός ANN (όλα εκτός από τα επίπεδα εισόδου και εξόδου) ονομάζονται κρυμμένα επίπεδα. Εάν ένα ANN έχει περισσότερα από ένα κρυμμένα επίπεδα, το ANN λέγεται ότι είναι ένα βαθύ ANN.

- ✓ Input layer - Ένας κόμβος για κάθε στοιχείο των δεδομένων εισαγωγής.
- ✓ Hidden layers - Αυθαίρετα επιλεγμένος αριθμός κόμβων για κάθε κρυφό επίπεδο.
- ✓ Output layer - Ένας κόμβος για κάθε μία από τις πιθανές επιθυμητές εξόδους



Εικόνα 6 Τεχνητό Νευρωνικό Δίκτυο (deeplizard.com)

Το ANN της εικόνας 6 έχει τρία επίπεδα συνολικά. Το επίπεδο στα αριστερά είναι το επίπεδο εισόδου. Το επίπεδο στα δεξιά είναι το επίπεδο εξόδου και το επίπεδο στη μέση είναι το κρυφό επίπεδο. Το κάθε επίπεδο αποτελείται από νευρώνες ή κόμβους. Εδώ, οι κόμβοι απεικονίζονται με τους κύκλους, οπότε σε κάθε επίπεδο υπάρχουν οι παρακάτω κόμβοι:

- Input layer (αριστερά): 2 κόμβοι
- Hidden layer (μέση): 3 κόμβοι
- Output layer (δεξιά): 2 κόμβοι

Δεδομένου ότι αυτό το δίκτυο έχει δύο κόμβους στο επίπεδο εισόδου, αυτό μας λέει ότι κάθε είσοδος σε αυτό το δίκτυο πρέπει να έχει δύο διαστάσεις, όπως για παράδειγμα διάστημα και χρόνος.

Εξάλλου, το ότι αυτό το δίκτυο έχει δύο κόμβους στο επίπεδο εξόδου, αυτό μας λέει ότι υπάρχουν δύο πιθανές έξοδοι για κάθε είσοδο που μεταφέρεται προς τα εμπρός (αριστερά προς τα δεξιά) μέσω του δικτύου. Για παράδειγμα, όριο ταχύτητας ή υπέρβαση ορίου ταχύτητας, θα μπορούσαν να είναι οι δύο κατηγορίες εξόδου. Οι κλάσεις εξόδου είναι επίσης γνωστές ως κλάσεις πρόβλεψης.

Ένα πραγματικό βάρος συνδέεται με κάθε σύνδεσμο. Ένα βάρος αντιστοιχεί περίπου στην αποτελεσματικότητα μιας συναπτικής σύνδεσης σε ένα πραγματικό νευρικό δίκτυο. Κάθε σύνδεση μεταξύ δύο κόμβων έχει σχετικό βάρος, που είναι απλώς ένας αριθμός.

Κάθε βάρος αντιπροσωπεύει τη δύναμη της σύνδεσης μεταξύ των δύο κόμβων. Όταν το δίκτυο λαμβάνει μια είσοδο σε έναν δεδομένο κόμβο στο επίπεδο εισόδου, αυτή η είσοδος μεταφέρεται στον επόμενο κόμβο μέσω μιας σύνδεσης και η είσοδος πολλαπλασιάζεται επί το βάρος που έχει αντιστοιχιστεί σε αυτήν τη σύνδεση.

Για κάθε κόμβο στο δεύτερο επίπεδο, στη συνέχεια υπολογίζεται ένα σταθμισμένο άθροισμα με καθεμία από τις εισερχόμενες συνδέσεις. Αυτό το άθροισμα μεταφέρεται στη συνέχεια σε μια συνάρτηση ενεργοποίησης, η οποία εκτελεί κάποιο είδος μετασχηματισμού στο δεδομένο άθροισμα. Για παράδειγμα, μια συνάρτηση ενεργοποίησης μπορεί να μετατρέψει το άθροισμα σε έναν αριθμό μεταξύ μηδέν και ενός. Ο πραγματικός μετασχηματισμός θα ποικίλει ανάλογα με το ποια λειτουργία ενεργοποίησης χρησιμοποιείται.

Οι κόμβοι είναι συνήθως ημι-γραμμικές μονάδες, πράγμα που σημαίνει ότι υπολογίζουν ένα σταθμισμένο άθροισμα των σημάτων εισόδου τους και στη συνέχεια εφαρμόζουν στο αποτέλεσμα μια μη γραμμική συνάρτηση, που ονομάζεται συνάρτηση ενεργοποίησης, για να παράγει την έξοδο ή ενεργοποίηση της μονάδας. Χρησιμοποιούνται διαφορετικές λειτουργίες ενεργοποίησης, αλλά συνήθως λειτουργούν σε σχήμα S ή σιγμοειδείς, όπως η λογιστική συνάρτηση $f(x) = 1 / (1 + e^{-x})$, αν και μερικές φορές ο ανορθωτής μη γραμμικότητας $f(x) = \max(0,x)$. Μια

συνάρτηση βημάτων όπως f (x) = 1 εάν x >= θ και 0 διαφορετικά, οδηγεί σε δυαδική μονάδα με κατώφλι θ. Οι μονάδες στο επίπεδο εισόδου ενός δικτύου είναι κάπως διαφορετικές στο να ρυθμίσουν τις ενεργοποιήσεις τους σε εξωτερικά παρεχόμενες τιμές που είναι οι εισόδους στη λειτουργία που προσεγγίζει το δίκτυο.

Οι περισσότερες συναρτήσεις ενεργοποίησης είναι μη γραμμικές και επιλέγονται με αυτόν τον τρόπο σκόπιμα. Η ύπαρξη λειτουργιών μη γραμμικής ενεργοποίησης επιτρέπει στα νευρωνικά μας δίκτυα να υπολογίζουν αυθαίρετα πολύπλοκες λειτουργίες.

Εάν ένα ANN έχει τουλάχιστον ένα βρόχο στις συνδέσεις του, είναι ένα επαναλαμβανόμενο παρά ένα τροφοδοτούμενο ANN. Αν και τόσο τα feedforward όσο και τα επαναλαμβανόμενα ANN έχουν χρησιμοποιηθεί στην ενισχυτική μάθηση.

Η Python είναι μια από τις πιο φιλικές γλώσσες προγραμματισμού για την υλοποίηση εφαρμογών Deep Learning. Υπάρχει υποστήριξη ενός συνόλου βιβλιοθηκών που καλύπτει κάθε περίπτωση χρήσης για τεχνητά νευρωνικά δίκτυα όπως:

- ✓ TensorFlow
- ✓ Keras
- ✓ PyTorch
- ✓ Scikit-learn
- ✓ Pandas
- ✓ NLTK
- ✓ Spark MLlib
- ✓ Theano
- ✓ MXNet
- ✓ Numpy

Μέσω των παραπάνω βιβλιοθηκών πιθανότατα χρησιμοποιούμε ήδη νευρωνικά δίκτυα σε καθημερινή βάση. Όταν ζητάμε από το βοηθό του κινητού μας να πραγματοποιήσει μια αναζήτηση ή να χρησιμοποιήσουμε ένα αυτοκίνητο που οδηγεί αυτόματα, όλα αυτά βασίζονται σε νευρωνικά δίκτυα. Τα ηλεκτρονικά παιχνίδια χρησιμοποιούν επίσης νευρωνικά δίκτυα για να προσαρμοστούν στο πως ένας παίκτης χρησιμοποιεί το παιχνίδι. Οι εφαρμογές χαρτών, για την επεξεργασία εικόνων χάρτη για να μας βοηθήσουν να βρούμε τον πιο γρήγορο τρόπο για να φτάσουμε στον προορισμό μας.

Ένα νευρωνικό δίκτυο είναι ένα σύστημα ή υλικό που έχει σχεδιαστεί για να λειτουργεί σαν ανθρώπινος εγκέφαλος. Τα νευρικά δίκτυα μπορούν να εκτελέσουν τις ακόλουθες εργασίες:

- Μετάφραση κειμένου
- Προσδιορισμό πρόσωπου
- Αναγνώριση ομιλίας
- Διάβασμα χειρόγραφου κειμένου
- Έλεγχος ρομπότ, και πολλά άλλα

3. PYTHON

Η Python είναι ένα πραγματικά εξαιρετικό εργαλείο ανάπτυξης που όχι μόνο χρησιμεύει ως γλώσσα προγραμματισμού γενικού σκοπού, αλλά και εξυπηρετεί συγκεκριμένα έργα. Είναι πιθανότατα μια από τις λιγοστές γλώσσες προγραμματισμού που είναι τόσο εύκολες όσο και ισχυρές. Αυτό είναι εξίσου καλό για τους αρχάριους όσο και για τους έμπειρους Με πολλές βιβλιοθήκες και πακέτα που επεκτείνουν τις δυνατότητες της Python και την καθιστούν μια ολοκληρωμένη και τέλεια εφαρμογή για όσους θέλουν να μπουν στην ανάπτυξη προγραμμάτων και αλγορίθμων. Με μερικές από τις σύγχρονες βιβλιοθήκες μηχανικής μάθησης και βαθιάς μάθησης.

Η επίσημη εισαγωγή στην Python είναι:

«Η Python είναι μια εύκολη στην εκμάθηση, ισχυρή γλώσσα προγραμματισμού. Έχει αποδοτικές δομές δεδομένων υψηλού επιπέδου και μια απλή αλλά αποτελεσματική προσέγγιση στον αντικειμενοστραφή προγραμματισμό. Η κομψή σύνταξη της Python και οι δυναμικοί τύποι της, μαζί με τη λειτουργία της ως διερμηνευόμενης (αντί μεταγλωττιζόμενης) γλώσσας, την καθιστούν την ιδανική γλώσσα για δημιουργία σεναρίων εντολών και για ταχεία ανάπτυξη εφαρμογών σε πολλούς τομείς και στις περισσότερες πλατφόρμες.» Ο Guido van Rossum, ο δημιουργός της γλώσσας Python, ονόμασε τη γλώσσα από την εκπομπή "Monty Python's Flying Circus" του BBC.

3.1 Χαρακτηριστικά της Python

Η Python υπήρξε η επιλογή για προγραμματιστές μηχανικής μάθησης και τεχνητής νοημοσύνης για μεγάλο χρονικό διάστημα. Προσφέρει μερικές από τις καλύτερες ευελιξίες και δυνατότητες στους προγραμματιστές που όχι μόνο αυξάνουν την παραγωγικότητά τους αλλά και την ποιότητα του κώδικα, για να μην αναφέρουμε τις εκτεταμένες βιβλιοθήκες που βοηθούν στη μείωση του φόρτου εργασίας. Παρακάτω παραθέτονται οι λειτουργίες που θέτουν την Python μεταξύ των κορυφαίων γλωσσών προγραμματισμού για μηχανική μάθηση, βαθιά μάθηση και τεχνητή νοημοσύνη όπως αυτά αναφέρονται στο βιβλίο – ***«A Byte of Python»*** και στον ιστοχώρο ***python.swaroopch.com*** :

- Η ελεύθερη και ανοιχτή φύση το καθιστά φιλικό προς την κοινότητα και εγγυάται βελτιώσεις μακροπρόθεσμα.
- Οι βιβλιοθήκες διασφαλίζουν ότι υπάρχει λύση για κάθε υπάρχον πρόβλημα.
- Η ομαλή εφαρμογή και ενσωμάτωση την καθιστά προσβάσιμη σε άτομα με διαφορετικό επίπεδο δεξιοτήτων.
- Αύξηση της παραγωγικότητας μειώνοντας τον χρόνο κωδικοποίησης και εντοπισμού σφαλμάτων.
- Μπορεί να χρησιμοποιηθεί και για Soft Computing, καθώς και για την επεξεργασία φυσικής γλώσσας.
- Λειτουργεί απρόσκοπτα με τις μονάδες κώδικα C και C++.

Απλή

Η Python είναι μια απλή και μινιμαλιστική γλώσσα. Το διάβασμα ενός καλού προγράμματος σε Python είναι σαν το διάβασμα των Αγγλικών, αλλά πολύ αυστηρών Αγγλικών! Αυτή η ομοιότητα της Python με ψευδοκώδικα είναι ένα από τα πιο ισχυρά σημεία της. Μας επιτρέπει να συγκεντρωνόμαστε στη λύση του προβλήματος αντί στην ίδια τη γλώσσα.

Εύκολη στην εκμάθηση

Η Python έχει μια ασυνήθιστα απλή σύνταξη, όπως έχει ήδη αναφερθεί.

Ελεύθερη και Ανοικτού Κώδικα

Η Python είναι ένα παράδειγμα ΕΛΛΑΚ (Ελεύθερο Λογισμικό και Λογισμικό Ανοικτού Κώδικα). Με απλά λόγια, μπορούμε να διανείμουμε αντίγραφα αυτού του λογισμικού, να διαβάσουμε τον πηγαίο κώδικά του, να κάνουμε αλλαγές σ' αυτό και να χρησιμοποιήσουμε κομμάτια του σε νέα ελεύθερα προγράμματα. Το ΕΛΛΑΚ βασίζεται στην ιδέα μιας κοινότητας που μοιράζεται τη γνώση. Αυτός είναι ένας από τους λόγους για τους οποίους η Python είναι τόσο καλή -δημιουργήθηκε και βελτιώνεται συνεχώς από μια κοινότητα που το μόνο που θέλει είναι μια καλύτερη Python.

Γλώσσα υψηλού επιπέδου

Όταν γράφουμε προγράμματα στην Python, δε χρειάζεται ποτέ να νοιαζόμαστε για τις χαμηλού επιπέδου λεπτομέρειες όπως η διαχείριση της μνήμης που χρησιμοποιείται από τα προγράμματά μας, κ.λπ.

Φορητή

Λόγω του ανοικτού της κώδικα, η Python έχει υλοποιηθεί (δηλαδή αλλάχθηκε για να λειτουργεί) σε πολλές πλατφόρμες. Όλα τα Python προγράμματά μας μπορούν να δουλέψουν σε οποιαδήποτε από αυτές τις πλατφόρμες χωρίς να χρειάζονται καθόλου αλλαγές αν είμαστε αρκετά προσεκτικοί ώστε να αποφύγουμε να χρησιμοποιήσετε χαρακτηριστικά που εξαρτούνται από κάθε σύστημα. Μπορούμε να χρησιμοποιήσουμε την Python στο Linux, στα Windows, στο FreeBSD, σε Macintosh, στο Solaris, στο OS/2, στην Amiga, στο AROS, στο AS/400, στο BeOS, στο OS/390, στο z/OS, στο Palm OS, στο QNX, στο VMS, στο Psion, στο Acorn RISC OS, στο VxWorks, σε PlayStation, στο Sharp Zaurus, στα Windows CE ακόμα και σε PocketPC.

Διερμηνευόμενη

Ένα πρόγραμμα που γράφεται σε μια μεταγλωττιζόμενη γλώσσα όπως η C ή η C++ μετατρέπεται από την πηγαία γλώσσα, για παράδειγμα τη C ή τη C++ σε μια γλώσσα που μιλάει ο υπολογιστής μας (δυαδικός κώδικας δηλαδή 0 και 1) χρησιμοποιώντας ένα μεταγλωττιστή.. Όταν τρέχουμε το πρόγραμμα, ο συνδέτης αντιγράφει το πρόγραμμα στη μνήμη και αρχίζει να το τρέχει. Η Python, από την άλλη, δε χρειάζεται μεταγλώττιση σε δυαδικό αρχείο. Απλά τρέχουμε το πρόγραμμα απ' ευθείας από τον πηγαίο κώδικα. Εσωτερικά, η Python μετατρέπει τον πηγαίο κώδικα σε μια ενδιάμεση μορφή που ονομάζεται bytecode και μετά το μεταφράζει στη γλώσσα του υπολογιστή και έπειτα το τρέχει. Όλο αυτό, στην πραγματικότητα κάνει τη χρήση της Python πολύ πιο εύκολη αφού δε χρειάζεται να ανησυχούμε για τη μεταγλώττιση του προγράμματος, τη σύνδεση με τις κατάλληλες βιβλιοθήκες. Αυτό επίσης κάνει τα προγράμματα της Python εξαιρετικά φορητά, αφού μπορούμε απλά να αντιγράψουμε το πρόγραμμα Python που φτιάξαμε σε έναν άλλο υπολογιστή και να δουλέψει έτσι απλά.

Αντικειμενοστραφής

Η Python υποστηρίζει τόσο το διαδικασιοστρεφή προγραμματισμό (procedure-oriented) όσο και τον αντικειμενοστραφή προγραμματισμό (object-oriented). Στο

διαδικασιοστρεφή προγραμματισμό, το πρόγραμμα δομείται πάνω σε διαδικασίες ή συναρτήσεις οι οποίες δεν είναι τίποτε άλλο από επαναχρησιμοποιήσιμα κομμάτια από προγράμματα. Στις αντικειμενοστραφείς γλώσσες, το προγράμματα δομούνται πάνω σε αντικείμενα τα οποία συνδυάζουν δεδομένα και λειτουργικότητα. Η Python έχει έναν πολύ ισχυρό αλλά πολύ απλό τρόπο για αντικειμενοστραφή προγραμματισμό, ειδικά όταν συγκρίνεται με μεγάλες γλώσσες όπως η C++ ή η Java.

Επεκτάσιμη

Αν χρειαζόμαστε ένα κρίσιμο κομμάτι κώδικα να τρέχει πολύ γρήγορα ή αν πρέπει να έχουμε ένα κομμάτι ενός αλγόριθμου που να μην είναι ανοικτό, τότε μπορούμε να προγραμματίσουμε εκείνο το κομμάτι σε C ή C++ και μετά να το χρησιμοποιήσουμε από το Python πρόγραμμά μας.

Ενσωματώσιμη

Μπορούμε να ενσωματώσουμε την Python μέσα στα προγράμματα σε C/C++ για να τους δώσουμε δυνατότητες 'scripting'.

Εκτεταμένες βιβλιοθήκες

Η πρότυπη βιβλιοθήκη της Python είναι πραγματικά τεράστια. Μπορεί να μας βοηθήσει να κάνουμε διάφορα πράγματα σχετικά με κανονικές εκφράσεις, δημιουργία τεκμηρίωσης, δοκιμές μονάδων, νημάτωση, βάσεις δεδομένων, περιηγητές ιστού, CGI, FTP, email, XML, XML-RPC, HTML, αρχεία WAV, κρυπτογράφηση, γραφικές διεπαφές χρήστη (GUI -graphical user interfaces), Tk, και άλλα πράγματα που εξαρτούνται από το σύστημα. Να σημειώσουμε ότι όλα τα παραπάνω είναι διαθέσιμα όποτε είναι εγκατεστημένη η Python. Αυτό ονομάζεται φιλοσοφία 'Batteries Included' της Python. Επιπλέον από την πρότυπη βιβλιοθήκη, υπάρχουν διάφορες άλλες βιβλιοθήκες υψηλής ποιότητας όπως η wxPython , η Twisted, η Python Imaging Library και πολλές άλλες.

3.2 Βιβλιοθήκες της Python

Η python μπορεί να θεωρηθεί ως μία από τις πιο διαδεδομένες γλώσσες λόγω των πολλαπλών χαρακτηριστικών της. Περιλαμβάνει μια μεγάλη ποικιλία χρήσιμων βιβλιοθηκών, εξαιρετικά τεράστια κοινότητα που αναφέρονται σε νευρωνικές παραλλαγές δικτύου για πρόσβαση στο νευρωνικό δίκτυο και σε ερευνητικούς κώδικες με βάση τη βαθιά μάθηση.

Η βαθιά μάθηση είναι ένα υποπεδίο στην αιχμή της μηχανικής μάθησης και της τεχνητής νοημοσύνης και έχει οδηγήσει σε σημαντικές ανακαλύψεις σε συναρπαστικά θέματα, όπως η οπτική του υπολογιστή, η επεξεργασία ήχου, ακόμη και στα αυτοκινούμενα οχήματα. Ουσιαστικά υπάρχει μια πληθώρα βιβλιοθηκών Python deep learning. Αυτά τα πακέτα υποστηρίζουν μια ποικιλία αρχιτεκτονικών βαθιάς μάθησης, όπως δίκτυα τροφοδοσίας, αυτόματους κωδικοποιητές, επαναλαμβανόμενα νευρωνικά δίκτυα (RNNs) και συνελικτικά νευρικά δίκτυα (CNN). Υπάρχουν πάνω από δώδεκα βιβλιοθήκες βαθιάς μάθησης στην Python, αλλά οι πιο χρήσιμές είναι πέντε.

Theano: είναι μια βιβλιοθήκη χαμηλού επιπέδου που ειδικεύεται στον αποτελεσματικό υπολογισμό. Χρησιμοποιείται απευθείας μόνο εάν χρειαζόμαστε λεπτομερής προσαρμογή και ευελιξία.

TensorFlow: είναι μια άλλη βιβλιοθήκη χαμηλού επιπέδου που είναι λιγότερο ώριμη από το Theano. Ωστόσο, υποστηρίζεται από την Google και προσφέρει κατανεμημένους υπολογιστές.

Lasagne: είναι ένα ελαφρύ περικάλυμμα για το Theano. Χρησιμοποιείται αυτό εάν χρειαζόμαστε την ευελιξία του Theano, αλλά δεν θέλουμε να γράφουμε επίπεδα νευρωνικών δικτύων από το μηδέν.

Keras: είναι ένα περικάλυμμα τόσο για το Theano όσο και για το Tensorflow. Είναι μινιμαλιστικό, αρθρωτό και καταπληκτικό για γρήγορο πειραματισμό. Αυτή είναι η βιβλιοθήκη Python για βαθιά μάθηση και το καλύτερο μέρος για να για αρχάριους.

MXNet: είναι μια άλλη βιβλιοθήκη υψηλού επιπέδου παρόμοια με την Keras. Προσφέρει δεσμεύσεις για πολλές γλώσσες και υποστήριξη για κατανεμημένους υπολογιστές.

3.3 KERAS

Η βιβλιοθήκη Keras είναι ένα API σχεδιασμένο για ανθρώπους, όχι για μηχανές. Η Keras ακολουθεί τις βέλτιστες πρακτικές για τη μείωση του γνωστικού φορτίου: προσφέρει σταθερά και απλά API, ελαχιστοποιεί τον αριθμό των ενεργειών χρήστη που απαιτούνται για κοινές περιπτώσεις χρήσης και παρέχει σαφή και ενεργά μηνύματα σφάλματος. Διαθέτει επίσης εκτενή τεκμηρίωση και οδηγούς για τους προγραμματιστές.

Το Keras δημιουργήθηκε αρχικά από τον François Chollet. Ιστορικά, το Keras ήταν ένα API υψηλού επιπέδου που βρισκόταν στην κορυφή ενός από τα τρία API

νευρικού δικτύου χαμηλότερου επιπέδου και λειτούργησε ως περικάλυμμα αυτών των βιβλιοθηκών χαμηλότερου επιπέδου. Αυτές οι βιβλιοθήκες αναφέρονται ως μηχανές υποστήριξης Keras (Keras backend engines).

Θα μπορούσαμε να επιλέξουμε TensorFlow, Theano ή CNTK ως μηχανή υποστήριξης με την οποία θα θέλαμε να εργαστούμε. Τελικά, η TensorFlow έγινε η πιο δημοφιλής μηχανή backend για την Keras.

Αργότερα, η Keras ενσωματώθηκε στη βιβλιοθήκη TensorFlow και τώρα έρχεται πλήρως συσκευασμένη με αυτήν. Τώρα, όταν εγκαθιστούμε το TensorFlow, λαμβάνουμε επίσης αυτόματα την Keras, καθώς είναι πλέον μέρος της βιβλιοθήκης TensorFlow. Χρησιμοποιούμαι την βιβλιοθήκη keras για τους παρακάτω λόγους:

Πρώτον, το Keras είναι ένα περικάλυμμα που μας επιτρέπει να χρησιμοποιήσουμε είτε το Theano είτε το TensorFlow backend. Αυτό σημαίνει ότι μπορούμε εύκολα να κάνουμε εναλλαγή μεταξύ των δύο, ανάλογα με την εφαρμογή σας.

Δεύτερον, έχει όμορφες κατευθυντήριες αρχές: αρθρωτότητα, μινιμαλισμός, επεκτασιμότητα και ικανότητα Python. Στην πράξη, αυτό καθιστά την εργασία στο Keras απλή και ευχάριστη.

Τέλος, η Keras έχει υλοποιήσεις κοινών δομών δικτύου. Είναι γρήγορο και εύκολο να δημιουργηθεί ένα συνελικτικό νευρωνικό δίκτυο και να λειτουργεί.

Προς το παρόν, το μεγαλύτερο μειονέκτημα του Keras είναι ότι δεν υποστηρίζει περιβάλλοντα πολλαπλών GPU για παράλληλη εκπαίδευση.

```
from keras.models import Sequential
from keras.layers import Dense, Activation

# Simple feed-forward architecture
model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))

# Optimize with SGD
model.compile(loss='categorical_crossentropy',
              optimizer='sgd', metrics=['accuracy'])

# Fit model in batches
model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)

# Evaluate model
loss_and_metrics = model.evaluate(X_test, Y_test, batch_size=32)
```

Εικόνα 7 Example Simple sequential model in Keras

3.4 TENSORFLOW

Το TensorFlow είναι μια δωρεάν βιβλιοθήκη λογισμικού ανοιχτού κώδικα για μηχανική μάθηση. Μπορεί να χρησιμοποιηθεί σε ένα ευρύ φάσμα εργασιών, αλλά δίνει ιδιαίτερη έμφαση στην εκπαίδευση και την εξαγωγή συμπερασμάτων βαθιών νευρωνικών δικτύων. Το Tensorflow είναι μια συμβολική βιβλιοθήκη μαθηματικών που έχει βασιστεί στη ροή δεδομένων και στον διαφοροποιήσιμο προγραμματισμό και έχει αναπτυχθεί από την Google.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Εικόνα 8 Example Simple sequential model in Tensorflow

Είναι σημαντικό να καταλάβουμε ότι από τώρα, η Keras έχει ενταχθεί πλήρως στην TensorFlow. Η αυτόνομη έκδοση του Keras δεν ενημερώνεται πλέον ή συντηρείται από την ομάδα Keras. Έτσι, όταν μιλάμε για την Keras τώρα, μιλάμε για αυτό ως API ενσωματωμένο στο TensorFlow, όχι ως ξεχωριστή αυτόνομη βιβλιοθήκη.

Το Keras ενσωματώνεται βαθιά με τη λειτουργικότητα TensorFlow χαμηλού επιπέδου, μπορούμε πραγματικά να χρησιμοποιήσουμε τη λειτουργικότητα υψηλού επιπέδου του Keras για να κάνουμε πολλά πράγματα χωρίς να απαιτείται να κάνουμε χρήση του κώδικα TensorFlow χαμηλότερου επιπέδου.

Ο κωδικός TensorFlow, συμπεριλαμβανομένου του Keras, θα εκτελείται διαφανώς σε μία μόνο GPU χωρίς να απαιτείται ρητή διαμόρφωση κώδικα. Η υποστήριξη Gens TensorFlow είναι προς το παρόν διαθέσιμη για συστήματα Ubuntu και Windows με κάρτες με δυνατότητα CUDA.

Όσον αφορά τον τρόπο με τον οποίο ο κώδικας TensorFlow θα εκτελείται στην GPU, λαμβάνουμε υπόψη ότι οι λειτουργίες που είναι ικανές να εκτελούνται σε GPU είναι πλέον προεπιλεγμένες. Επομένως, εάν το TensorFlow εντοπίσει CPU και GPU, τότε ο κώδικας με δυνατότητα GPU θα εκτελείται από προεπιλογή στην GPU.

Εάν δεν θέλουμε αυτό να συμβεί για οποιονδήποτε λόγο, μπορούμε να αλλάξουμε ρητά τη συσκευή στην οποία θέλουμε να εκτελείται ο κώδικάς μας. Η μόνη απαίτηση υλικού είναι η κάρτα NVIDIA GPU με δυνατότητα υπολογισμού CUDA. Βέβαια, υπάρχουν διαφορετικές οδηγίες ανάλογα με το αν εκτελείτε ο κώδικά μας από περιβάλλον Windows ή περιβάλλον Linux.

Για την απλοποίηση της εγκατάστασης και την αποφυγή διενέξεων βιβλιοθηκών, η TensorFlow συνιστά τη χρήση μιας εικόνας TensorFlow Docker με υποστήριξη GPU, καθώς αυτή η εγκατάσταση απαιτεί μόνο την εγκατάσταση προγραμμάτων οδήγησης NVIDIA GPU. Το TensorFlow διαθέτει έναν οδηγό (https://www.tensorflow.org/install/docker) με όλα τα αντίστοιχα βήματα.

Η εγκατάσταση της TensorFlow σε περιβάλλον windows είναι τόσο απλή όσο η εκτέλεση της εντολής εγκατάστασης tensorflow. Επίσης δεν πρέπει να ξεχνάμε την ανάγκη ελέγχου για να βεβαιωθούμε ότι το σύστημα μας πληροί τις απαιτήσεις του συστήματος TensorFlow (https://github.com/tensorflow/tensorflow/releases/tag/v2.1.0)

Μία από αυτές τις απαιτήσεις, είναι εάν έχει εγκατασταθεί η κατάλληλη έκδοση του Microsoft Visual C ++ με δυνατότητα αναδιανομής.

```
C:\Development\Python\Python37\lib\imp.py, line 342, in
load_dynamic
return _load(spec)
```

Χωρίς την εγκατάσταση της κατάλληλης έκδοσης Microsoft Visual C ++ , όταν θα προσπαθήσουμε να εισαγάγουμε την TensorFlow θα εμφανιστεί το παρακάτω σφάλμα, οπότε θα πρέπει να βεβαιωθούμε ότι έχουμε προβεί στην παραπάνω εγκατάσταση.

```
ImportError: DLL load failed: The specified module could not
be found.
```

Εάν το πρόγραμμα που θα χρησιμοποιήσουμε δεν θα έχει ιδιαίτερες υπολογιστικές απαιτήσεις τότε μπορούμε από αυτό εδώ το σημείο να ξεκινήσουμε να εργαζόμαστε με τη βιβλιοθήκη. Εάν όμως χρειαζόμαστε μεγαλύτερη υπολογιστική ισχύ τότε είναι αναγκαίο να προβούμε στις παρακάτω εγκαταστάσεις.

Τώρα θα πρέπει να εγκαταστήσουμε τα προγράμματα οδήγησης Nvidia. Πλοηγούμαστε στον ιστότοπο της Nvidia’s για να ξεκινήσει η λήψη. Θα πρέπει να γνωρίζουμε τις προδιαγραφές της GPU μας, ώστε να μπορούμε να κάνουμε λήψη τα

κατάλληλα προγράμματα οδήγησης. Εάν δεν γνωρίζουμε αυτές τις προδιαγραφές, μπορούμε να μεταβούμε στους «προσαρμογής οθόνης» στη «Διαχείριση συσκευών» και να λάβουμε τις πληροφορίες που χρειαζόμαστε.

Τώρα πρέπει να εγκαταστήσουμε το CUDA Toolkit. Πλοηγούμαστε στον ιστότοπο της Nvidia's για να επιλέξουμε την έκδοση που θέλουμε να κατεβάσουμε. Επίσης πρέπει να βεβαιωθούμε ότι έχουμε επιλέξει την έκδοση CUDA Toolkit που υποστηρίζει το TensorFlow. Μπορούμε να βρούμε αυτές τις πληροφορίες στον ιστότοπο της TensorFlow's site. Σημείωση, εάν δεν έχουμε εγκατεστημένο το Microsoft Visual Studio στον υπολογιστή μας, τότε κατά την εγκατάσταση, ενδέχεται να λάβουμε το παρακάτω μήνυμα διότι για να εγκατασταθεί το CUDA Toolki είναι απαραίτητο το Microsoft Visual Studio:

```
No supported version of Visual Studio was found.
Some components of CUDA Toolkit will not work properly.
Please install Visual Studio first to get the full functionality
```

Αντ' αυτού, μεταβαίνουμε στον ιστότοπο της Microsoft για λήψη και εγκατάσταση της έκδοσης community Visual Studio, χρειαζόμαστε μόνο το βασικό πακέτο.

Τώρα πρέπει να εγκαταστήσουμε το cuDNN SDK. Πλοηγούμαστε ξανά στον ιστότοπο της Nvidia's. Για να αποκτήσουμε πρόσβαση στη λήψη, πρέπει πρώτα να δημιουργήσουμε έναν δωρεάν λογαριασμό και να περάσουμε από μια γρήγορη επιβεβαίωση μέσω email. Στη συνέχεια, επιλέγουμε να κάνουμε λήψη της έκδοσης του cuDNN που αντιστοιχεί στην έκδοση που υποστηρίζεται από το TensorFlow του CUDA Toolkit που κατεβάσαμε προηγουμένως.

Για να ελέγξουμε ότι όλα εγκαταστάθηκαν σωστά τότε ανοίγουμε ένα σημειωματάριο Jupyter ή οποιοδήποτε IDE της επιλογής μας και εκτελούμε τη παρακάτω γραμμή κώδικα για να ελέγξουμε ότι η TensorFlow έχει εντοπίσει μια GPU στον υπολογιστή μας.

```
import tensorflow as tf
print("Num GPUs Available: ",
len(tf.config.experimental.list_physical_devices('GPU')))

> Num GPUs Available: 1
```

Εάν η έξοδος είναι 1, τότε το TensorFlow έχει αναγνωρίσει με επιτυχία την GPU μας. Εάν η έξοδος είναι 0, και περιέχονται σφάλματα τότε επανεκκινούμε το μηχάνημα

μας και προσπαθούμε ξανά, εάν δεν υπάρχουν σφάλματα και η κάρτα γραφικών μας δεν είναι NVIDIA τότε απλά δεν υποστηρίζει η κάρτα μας την εκτέλεση του προγράμματος από την GPU παρά μόνο από την CPU. Τέλος εάν όλα πήγαν καλά τότε εξορισμού ο μελλοντικός κώδικας TensorFlow θα τρέχει από την GPU.

4. ΚΑΤΑΝΕΜΗΜΕΝΟ ΣΧΗΜΑ ΑΝΑΘΕΣΗΣ ΕΡΓΑΣΙΩΝ

Η συγκεκριμένη εργασία κάνει προσομοίωση ενός δικτύου υπολογιστών στις παρυφές του (edge computing). Υπάρχει ένα πλήθος κόμβων οι οποίοι είναι αυτόνομοι. Κάθε κόμβος έχει κάποια χαρακτηριστικά που σχετίζονται με τα resources που έχει, όπως μνήμη, δίσκο, κ.λπ.. Κάθε κόμβος σε κάθε χρονική στιγμή έχει ένα φόρτο που δείχνει πόσες εργασίες (tasks) αναμένουν για εκτέλεση. Έχουμε στη διάθεσή μας ένα σύνολο από tasks τα οποία στέλνονται σε ένα κόμβο που πρέπει να τα εκτελέσει και για το κάθε task παίρνουμε τη ζήτηση που έχει σαν ένα αριθμό στο [0,1]. Αποθηκεύουμε τη ζήτηση για το κάθε task. Πάνω σε αυτή τη ζήτηση θέλουμε να βγάλουμε μια τιμή που να μας δείχνει αν είναι σωστό να κρατήσουμε και να εκτελέσουμε το task τοπικά στον κόμβο ή πρέπει να το στείλουμε σε κάποιο άλλο κόμβο. Γενικά, θέλουμε να κρατάμε δημοφιλή tasks τοπικά ώστε να κάνουμε επαναχρησιμοποίηση των πόρων που καταναλώνουμε για να τα εκτελέσουμε. Επίσης, το κάθε task προκαλεί κάποιο φόρτο στον κόμβο, ο οποίος εκτελεί και άλλα tasks καθώς και άλλα προγράμματα. Για κάθε task μπορούμε να υπολογίσουμε το όφελος της εκτέλεσης τοπικά και το όφελος της εκτέλεσης σε κάποιο άλλο κόμβο. Η εκτέλεση με το μεγαλύτερο όφελος θα ενισχύσει την απόφαση μας να κρατηθεί και να εκτελεστεί τοπικά ένα task.

Το όφελος υπολογίζεται ως εξής: όσο μεγαλύτερη είναι η ζήτηση για ένα task τόσο περισσότερο ωφελούμαστε από την εκτέλεση του, παρόλα αυτά η εκτέλεση του μας κοστίζει ένα ποσό, ίσο με το φόρτο που θα προκαλέσει η εκτέλεση του task, έτσι κατασκευάζουμε μία συνάρτηση:

f(DEMAND, LOAD)

Ορίζουμε ως DEMAND την παράμετρο η οποία αφορά στην συχνότητα εκτέλεσης ενός task, με άλλα λόγια πόσο δημοφιλές είναι ένα task. Παράγουμε τις τιμές για το DEMAND βασιζόμενοι στον αλγόριθμο που αναφέρεται στη στατιστική του **Trigg**. Οπώς θα αναλύσουμε παρακάτω, ο δείκτης παρακολούθησης του Trigg **(T)** θα μας δώσει την τιμή για το DEMAND που θα εισάγετε στην παραπάνω συνάρτηση.

Η προσέγγιση ανίχνευσης τάσεων του **Trigg** είναι μια στατιστική μέθοδος που παρακολουθεί τη διακύμανση ενός σήματος σε πραγματικό χρόνο (ή ψεύδο - πραγματικό χρόνο) χρησιμοποιώντας μια μεταβλητή παρακολούθησης **(T)**. Η

μεταβλητή παρακολούθησης του Trigg είναι ένας δείκτης ανίχνευσης σήματος που αποδίδει μια τιμή μεταξύ **-1** και **+ 1** στην πιθανότητα εμφάνισης τάσης. Στο **T = + 1** υπάρχει 100% βεβαιότητα ότι η μεταβλητή αυξάνεται και στο **-1** υπάρχει 100% βεβαιότητα ότι η μεταβλητή μειώνετε. Η μεταβλητή **T** υπολογίζεται χρησιμοποιώντας τη διαφορά μεταξύ της τρέχουσας τιμής της μεταβλητής και του χρονικού σταθμισμένου κινούμενου μέσου όρου των προηγούμενων τιμών

Η αρχικοποίηση του αλγορίθμου απαιτεί τιμή του σήματος στην αρχή του παραθύρου παρακολούθησης **φ**. Ο σταθμισμένος μέσος όρος της πρώτης περιόδου δειγματοληψίας είναι:

$$u_{t(0)}=\varphi$$

και το σφάλμα πρόβλεψης των δειγμάτων στο χρονικό διάστημα παρακολούθησης μπορεί να οριστεί ως:

$$S_{t(0)}=\varphi/100$$

Επίσης, η αρχική μέση απόλυτη απόκλιση στο χρονικό διάστημα παρακολούθησης μπορεί να προσεγγιστεί κατά:

$$M_{t(0)}=\varphi/10$$

Επομένως, καθώς αξιολογούμε την τάση σε ένα σύνολο δειγμάτων ενός σήματος σε ένα παράθυρο παρακολούθησης, η πρόβλεψη για ένα επερχόμενο δείγμα μπορεί να οριστεί ως:

$$U_t=\theta d_t+(1-\theta)u_{t-1}$$

Όπου d_t είναι η τρέχουσα τιμή του σήματος, το u_{t-1} είναι η πρόβλεψη στο προηγούμενο δείγμα χρόνου και **θ** είναι μια παράμετρος σχεδιασμού μεταξύ 0 και 1, η οποία καθορίζει τη σταθερά χρόνου της εκθετικής στάθμισης (συνήθως 0,1-0,22) . Επομένως, το σφάλμα στην πρόβλεψη της τρέχουσας τιμής είναι:

$$e_t=d_t-u_{t-1}$$

Αυτό το σήμα σφάλματος μπορεί να επαναπροσδιοριστεί χρησιμοποιώντας τα εξής:

$$s_t=\theta e_t-(1-\theta)s_{t-1}$$

Στη συνέχεια, η μέση απόλυτη απόκλιση για την τρέχουσα στιγμή ορίζεται ως:

$$M_t=\theta|e_t|+(1-\theta)M_{t-1}$$

Ο δείκτης παρακολούθησης του Trigg **T** ο οποίος θα μας δώσει την τιμή για το DEMAND υπολογίζεται ως εξής :

$$T=s_t/M_t$$

Τέλος, οι στατιστικές μεταβλητές ενημερώνονται ως εξής:

$$u_{t-1}= u_t$$

$$S_{t-1}=S_t$$

$$M_{t-1}=M_t$$

Αφού έχουμε υπολογίσει το trend της ζήτησης για το task, το LOAD το παράγουμε με χρήση τυχαίων αριθμών. Η παραπάνω συνάρτηση είναι στην ουσία ένα νευρωνικό δίκτυο που έχει δύο (2) εισόδους και μια (1) έξοδο: είσοδοι DEMAND, LOAD, έξοδος: εκτελείται τοπικά (1) ή εκτελείται εκτός κόμβου (0) (σε άλλο κόμβο ή στο cloud). Για κάθε task λοιπόν δίνουμε το ζεύγος τιμών στο νευρωνικό δίκτυο και αυτό θα αποφασίζει ποιο ζεύγος θα σημανθεί με 1 ή 0. Όσα task έχουν σημανθεί με 0 θα εκτελούνται εκτός κόμβου ενώ θα κρατούνται τα task τα οποία θα έχουν σημανθεί με 1 και τα οποία θα εκτελούνται τοπικά.

Όσα έχουν πάρει 1 θα πρέπει να τα κρατήσουμε τοπικά, όμως μπορεί όλα αυτά μαζί να προκαλέσουν πολύ μεγάλο φόρτο που θα δυσκολέψει την εκτέλεσή τους. Έτσι παίρνουμε όλα τα task με το 1 και κάνουμε ένα τελευταίο βήμα ορίζοντας ένα συντελεστή οφέλους η τιμή του οποίου δίδεται από τον παρακάτω τύπο:

$$\exp(DEMAND/LOAD - a)$$

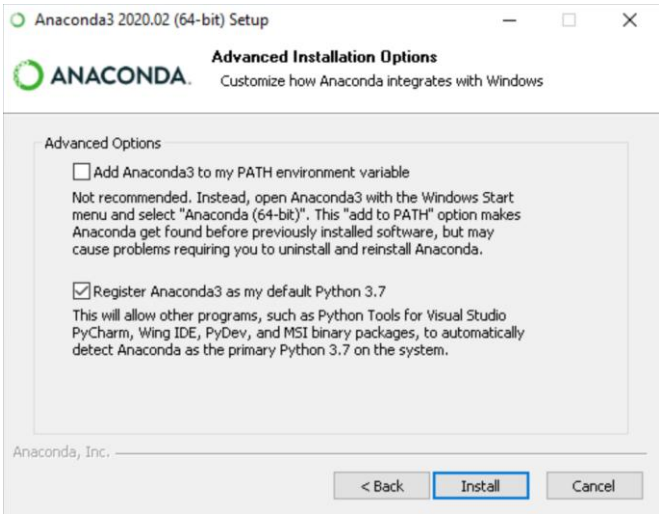
Όπου a είναι μια σταθερά. Αν το DEMAND είναι πολύ μεγαλύτερο από το LOAD τότε ο συντελεστής βγαίνει και αυτός αρκετά μεγάλος. Αν το LOAD είναι αρκετά μεγαλύτερο από το DEMAND τότε ο συντελεστής βγαίνει αρκετά μικρός. Έτσι δεν μας συμφέρει να κρατήσουμε το task που έχει μικρό συντελεστή, γιατί έχει μικρή ζήτηση σε σχέση με το φόρτο που θα προκαλέσει. Ταξινομούμε όλα τα tasks σε φθίνουσα σειρά του συντελεστή και για δεύτερη φορά επιλέγουμε τα last-k τα οποία διώχνουμε σε δεύτερη φάση.

4.1 Εργαλεία που χρησιμοποιήθηκαν

Για να υλοποιηθεί ο παραπάνω κώδικας σε python χρησιμοποιήθηκε το notebook Jupyter. Το Jupyter Notebook είναι μια διαδικτυακή εφαρμογή ανοιχτού κώδικα που μας επιτρέπει να δημιουργήσουμε και να μοιραζόμαστε έγγραφα που περιέχουν ζωντανό κώδικα, εξισώσεις, απεικονίσεις και αφηγηματικό κείμενο. Οι χρήσεις περιλαμβάνουν καθαρισμό και μετατροπή δεδομένων, αριθμητική προσομοίωση, στατιστική μοντελοποίηση, οπτικοποίηση δεδομένων, μηχανική μάθηση και πολλά άλλα.

Το παραπάνω notebook εκτελούταν μέσω της πλατφόρμας anaconda. Το Anaconda είναι μια διανομή των γλωσσών προγραμματισμού Python και R για επιστημονικούς υπολογιστές, που στοχεύει στην απλούστευση της διαχείρισης και ανάπτυξης πακέτων. Η διανομή περιλαμβάνει πακέτα επιστήμης δεδομένων κατάλληλα για Windows, Linux και macOS.

Από τον επίσημο ιστοχώρο κατεβάζουμε το εκτελέσιμο αρχείο και το εγκαθιστούμε στο μηχάνημά μας. Επιλέγουμε αν θα προσθέσουμε το Anaconda στη μεταβλητή περιβάλλοντος PATH. Συνιστούμε να μην προσθέσουμε το Anaconda στη μεταβλητή περιβάλλοντος PATH, καθώς αυτό μπορεί να επηρεάσει άλλο λογισμικό. Αντ 'αυτού, χρησιμοποιούμε το λογισμικό Anaconda ανοίγοντας το Anaconda Navigator ή το Anaconda Prompt από το μενού Έναρξη.

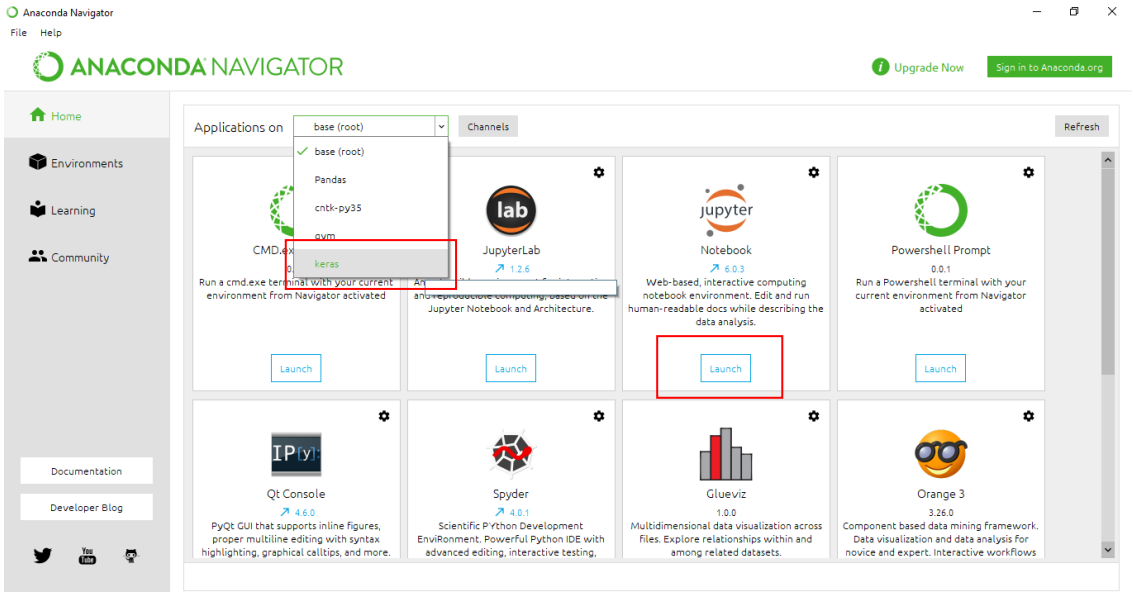


Εικόνα 9 Δεν προσθέτουμε το anaconda στο PATH

Χρησιμοποιήσαμε την βιβλιοθήκη Keras για να υλοποιήσουμε το νευρωνικό μας δίκτυο. Η παραπάνω βιβλιοθήκη πρέπει να εγκατασταθεί στη πλατφόρμα anaconda. Εγκαθιστούμε τα παρακάτω πακέτα με τις παρακάτω εντολές

```
conda install -c conda-forge keras
conda install -c conda-forge/label/broken keras
conda install -c conda-forge/label/cf201901 keras
conda install -c conda-forge/label/cf202003 keras
```

Έτσι δημιουργούμε ένα καινούργιο περιβάλλον το οποίο ονομάζουμε Keras. Για να δημιουργήσουμε ένα καινούργιο πρόγραμμα το οποίο απαιτεί την χρήση ενός νευρωνικού δικτύου μέσω της βιβλιοθήκης Keras, επιλέγουμε το παραπάνω περιβάλλον και μέσω αυτού εκτελούμαι το notebook jupyter όπως φαίνετε και στην παρακάτω εικόνα.



Εικόνα 10 Anaconda - Keras

4.2 Κώδικας Προγράμματος

Στον παρακάτω κώδικα θα κάνουμε μια προσομοίωση ενός κόμβου ο όποιος βρίσκεται στις παρυφές του δικτύου μαζί με άλλους κόμβους. Κάθε κόμβος έχει κάποια χαρακτηριστικά που σχετίζονται με τους πόρους του, μνήμη, δίσκο, επεξεργαστική ισχύ κ.λπ.. Σε αυτό τον κόμβο έρχονται εργασίες τις οποίες θα πρέπει να τις εξυπηρετήσει. Όσες εργασίες μπορεί, τις εξυπηρετεί ο ίδιος, ειδάλλως θα πρέπει να τις στείλει σε άλλο κόμβο ή στο cloud. Το πρώτο κριτήριο για την τοπική εκτέλεση είναι κατά πόσο η συγκεκριμένη εργασία είναι δημοφιλής, δηλαδή εκτελείται πιο συχνά στα υπολογιστικά

συστήματα ***(DEMAND)***. Δεύτερο κριτήριο είναι το μέγεθος του φορτίου που θα εκτελεστεί τοπικά, προτιμάμε τις εργασίες με μικρό φορτίο ***(LOAD)***. Αφού επιλεγούν οι εργασίες (tasks) που πληρούν ταυτόχρονα τις παραπάνω προϋποθέσεις γίνεται ένας τελευταίος έλεγχος για την αποφυγή υπερβολικού υπολογιστικού φόρτου στο κόμβο. Ο έλεγχος αυτός διενεργείται με το κριτήριο του συντελεστού οφέλους. Ο παραπάνω συντελεστής προκύπτει από τον παρακάτω τύπο όπου (a) μία σταθερά :

Συντελεστής Οφέλους =exp(DEMAND/LOAD - a)

Έτσι όσο μεγαλύτερος είναι ο συντελεστής τόσο περισσότερο μας συμφέρει να κρατήσουμε την εργασία για τοπική εκτέλεση. Τέλος με αυτό το κριτήριο διώχνουμε επιπλέον εργασίες που θα μας δημιουργούσαν πρόβλημα φόρτου.

Η απόφαση για το που θα εκτελεστεί μία εργασία θα παρθεί από ένα νευρωνικό δίκτυο. Δημιουργούμε ένα (sequential) διαδοχικό μοντέλο. Το Keras ορίζει ένα διαδοχικό μοντέλο ως μια διαδοχική στοίβα γραμμικών επιπέδων. Αυτό το διαδοχικό μοντέλο είναι η υλοποίηση ενός τεχνητού νευρικού δικτύου από τον Keras. Το συγκεκριμένο νευρωνικό δίκτυο αποτελείται από τέσσερα επίπεδα ή στρώματα. Το πρώτο επίπεδο, input layer έχει δύο εισόδους, δηλαδή δύο νευρώνες. Το δεύτερο επίπεδο, το οποίο είναι το πρώτο κρυφό επίπεδο, hidden layer έχει (16) δεκαέξι νευρώνες, και το επόμενο κρυφό επίπεδο έχει 32 νευρώνες. Το τελευταίο επίπεδο, το επίπεδο εξόδου έχει ένα νευρώνα. Τροφοδοτούμε την είσοδο του παραπάνω δικτύου με το DEMAND και το LOAD και παίρνουμε έξοδο (1) ή (0). Το (1) μας δείχνει ότι η εργασία θα εκτελεστεί τοπικά στον κόμβο και το (0) ότι θα πρέπει η εργασία να αποσταλεί στο cloud.

Ο παρακάτω κώδικας χωρίζεται σε τρία μέρη. Στο πρώτο μέρος εκπαιδεύουμε το νευρωνικό μας δίκτυο με γνωστά δεδομένα. Στο δεύτερο μέρος τροφοδοτούμε το δίκτυο με άγνωστα στοιχεία και το καλούμε να πάρει τη σωστή απόφαση. Στο τρίτο μέρος ελέγχουμε εάν έχει αποφασίσει σωστά και εξάγουμε στατιστικά δεδομένα

Πριν εισάγουμε τις απαραίτητες βιβλιοθήκες εγκαθιστούμε τα προαπαιτούμενα για διαχείριση των παραγόμενων δεδομένων μας μέσω του excel καθώς επίσης και μία βιβλιοθήκη μηχανικής μάθησης για python την “scikit-learn”

pip install openpyxl

pip install xlsxwriter

pip install -U scikit-learn

Εισάγουμε τις απαραίτητες βιβλιοθήκες

```
from random import randint
from sklearn.utils import shuffle
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from functools import reduce
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense
from keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import datetime
import statistics
import math
import keras
import numpy as np
import random
import operator
import pandas as pd
import xlsxwriter
from keras.callbacks import CSVLogger
```

4.2.1 ΠΡΩΤΟ ΜΕΡΟΣ - ΚΩΔΙΚΑΣ ΕΚΠΑΙΔΕΥΣΗΣ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ

Ορίζουμε τις παραμέτρους του αλγόριθμου Trigg από όπου θα παραχθεί η τιμή της μεταβλητής DEMAND. Ορίζουμε την μεταβλητή Values που αφορά στο πλήθος των τιμών που θα παράγουμε. Επίσης ορίζουμε τους πίνακες οι οποίοι θα φιλοξενήσουν τις τιμές των μεταβλητών DEMAND, LOAD αλλά και των ετικετών των δεδομένων (labels).

```
#Trigg parameters
U=[]
e=[]
s=[]
M=[]
T=[]
P=[]
L=[]
Theta=0.2

#parameters
Values = 1000
n_features =2

train_labels = []
train_demand = []
train_Load = []
```

Παράγουμε τυχαίες τιμές από το 0 έως το 100, οι οποίες προσομοιάζουν την δημοφιλία (DEMAND).

```
random_Demand = np.random.normal(0,100,Values)
```

Τροποποιούμε τους παραπάνω τυχαίους αριθμούς που παράχθηκαν, όσον αφορά το DEMAND, και με την βοήθεια του αλγορίθμου Trigg υπολογίζουμε την νέα τιμή της μεταβλητής DEMAND. Επίσης υπολογίζουμε τυχαίες τιμές για τη μεταβλητή Load από 0 έως 10000.

```
U=np.average(random_Demand)
s=U/100
M=U/10
print(U,"----",s,"-----",M)

for i in range(Values):
    random_Load = randint(0,10000)

    U = ((Theta*(random_Demand[i])) +(1-Theta)*(U))
    e=(random_Demand[i]-U)
    s=Theta*e-(1-Theta)*s
    M=(Theta*abs(e))+(1-Theta)*M
    T=s/M
    L= random_Load
    train_demand.append(T)
    train_Load.append(L)
```

Τοποθετούμε ετικέτες στα δεδομένα μας (1) και (0) ανάλογα με το εάν πληρούν τα κριτήρια που αναφέρθηκαν παραπάνω. Σύμφωνα με τον αλγόριθμο Trigg, εάν το (T) τείνει προς το (1) τότε έχουμε δημοφιλία ενώ όταν τείνει στο (-1) δεν έχουμε. Με αλλά λόγια, τοποθετούμε με ετικέτα (1) τις εργασίες που έχουν (T>0) και φορτίο (L=<5000). Παράγουμε μεγάλες τιμές για το Load γιατί θέλουμε ποικιλομορφία στην εκπαιδευση του μοντέλου μας. Έχουμε συμπεράνει ότι όσο μεγαλύτερη ποικιλομορφία υπάρχει στις τιμές που εκπαιδεύεται ένα νευρωνικό δίκτυο τόσο πιο ορθά αποφασίζει. Τέλος δημιουργούμε τους πίνακες που περιέχουν τις τιμές με τις όποιες θα εκπαιδευτεί το μοντέλο μας.

```
    if T > 0  and L<=5000:
        train_labels.append(1)
    else:
        train_labels.append(0)

#make lists arrayes
train_labels = np.array(train_labels)
train_demand = np.array(train_demand)
train_Load=np.array(train_Load)
```

Θα χρησιμοποιήσουμε την κλάση MinMaxScaler της scikit-learn για να μειώσουμε όλα τα δεδομένα από μια κλίμακα που κυμαίνεται από 0 έως 100 και από 0 έως 10000 σε κλίμακα από 0 έως 1. Αναδιαμορφώνουμε τα δεδομένα ως τεχνική απαίτηση, αφού η συνάρτηση fit_transform () δεν δέχεται δεδομένα μίας διάστασης (1D) από προεπιλογή.

```
74 #normalize (0,1)
75 scaler = MinMaxScaler(feature_range=(0,1))
76 scaled_train_demand = scaler.fit_transform(train_demand.reshape(-1,1))
77 scaled_train_Load = scaler.fit_transform(train_Load.reshape(-1,1))
```

Ενοποιούμε τους δύο πίνακες που αφορούν το DEMAND και το LOAD σε έναν πίνακα τον *InDNN* ώστε να τον εισάγουμε στην είσοδο του νευρωνικού δικτύου που απαιτεί πίνακα δύο στηλών.

```
79 #concatenate arrays demand and load
80 InDNN = np.concatenate((scaled_train_demand,scaled_train_Load), axis=1)
```

Στη συνέχεια, δημιουργούμε μια μεταβλητή που ονομάζεται (model) μοντέλο και την ορίζουμε ως ένα Sequential αντικειμένο, έτσι δημιουργούμε το νευρωνικό μας δίκτυο το οποίο αποτελείται από τέσσερα επίπεδα. Στον κατασκευαστή, περνάμε μια σειρά από Dense αντικείμενα. Κάθε ένα από αυτά τα αντικείμενα που ονομάζεται Dense είναι στην πραγματικότητα ένα επίπεδο (layer). Η λέξη Dense δείχνει ότι αυτά τα επίπεδα είναι τύπου Dense ή διαφορετικά πλήρως συνδεδεμένα. Το Dense είναι ένας συγκεκριμένος τύπος layer, αλλά υπάρχουν πολλοί άλλοι τύποι όπως:

- ✓ Convolutional layers
- ✓ Pooling layers
- ✓ Recurrent layers
- ✓ Normalization layers

Για παράδειγμα, ένα Convolutional layer, συνελικτικό επίπεδο χρησιμοποιείται συνήθως σε μοντέλα που δουλεύουν με δεδομένα εικόνας. Τα Recurrent layers επαναλαμβανόμενα επίπεδα χρησιμοποιούνται σε μοντέλα που λειτουργούν με δεδομένα χρονοσειρών και Dense πλήρως συνδεδεμένα επίπεδα, όπως υποδηλώνει το όνομα, συνδέει πλήρως κάθε είσοδο σε κάθε έξοδο εντός του επιπέδου της. Για τους παραπάνω λόγους χρησιμοποιούμε Dense επίπεδα, έτσι το πρώτο επίπεδο είναι η είσοδος η οποία έχει δύο νευρώνες. Τα επόμενα δύο κρυφά επίπεδα αποτελούνται από 16 νευρώνες και 32 νευρώνες αντίστοιχα. Το τετάρτο και τελευταίο επίπεδο είναι η έξοδος με ένα νευρώνα. Η επιλογή των νευρώνων σε κάθε επίπεδο είναι αυθαίρετη και εξαρτάται από του πειραματισμούς που κάνουμε ώστε να πετύχουμε τη βέλτιστη έξοδο.

```
85 model = Sequential([
86     Dense(units=16, input_shape=(n_features,), activation='relu'),
87     Dense(units=32, activation='relu'),
88     Dense(units=1, activation='sigmoid')
```

Κάθε σύνδεση μεταξύ δύο κόμβων έχει ένα σχετικό βάρος, που είναι απλώς ένας αριθμός. Κάθε βάρος αντιπροσωπεύει τη δύναμη της σύνδεσης μεταξύ των δύο κόμβων. Όταν το δίκτυο λαμβάνει μια είσοδο σε έναν δεδομένο κόμβο στο επίπεδο εισόδου, αυτή η είσοδος μεταφέρεται στον επόμενο κόμβο μέσω μιας σύνδεσης και η είσοδος πολλαπλασιάζεται επί το βάρος που έχει αντιστοιχιστεί σε αυτήν τη σύνδεση.

Για κάθε κόμβο στο δεύτερο επίπεδο, στη συνέχεια υπολογίζεται ένα σταθμισμένο άθροισμα με καθεμία από τις εισερχόμενες συνδέσεις. Αυτό το άθροισμα μεταφέρεται στη συνέχεια σε μια συνάρτηση ενεργοποίησης, η οποία εκτελεί κάποιο είδος μετασχηματισμού στο δεδομένο άθροισμα. Για παράδειγμα, μια συνάρτηση ενεργοποίησης μπορεί να μετατρέψει το άθροισμα σε έναν αριθμό μεταξύ μηδέν και ένα. Ο πραγματικός μετασχηματισμός θα ποικίλει ανάλογα με το ποια λειτουργία ενεργοποίησης χρησιμοποιείται.

έξοδος κόμβου = ενεργοποίηση (σταθμισμένο άθροισμα εισόδων)

Οι συναρτήσεις ενεργοποίησης που χρησιμοποιούμαι είναι “relu” και “sigmoid”. Σε ένα τεχνητό νευρικό δίκτυο, μια συνάρτηση ενεργοποίησης είναι μια συνάρτηση που χαρτογραφεί τις εισόδους ενός κόμβου στην αντίστοιχη έξοδο του. Η συνάρτηση ενεργοποίησης κάνει κάποιο είδος λειτουργίας για να μετατρέψει το άθροισμα σε έναν αριθμό που είναι συχνά ,μεταξύ κάποιου κατώτερου ορίου και κάποιου ανώτερου ορίου. Αυτός ο μετασχηματισμός είναι συχνά ένας μη γραμμικός μετασχηματισμός.

$$relu(x) = max(0, x)$$

$$sigmoid(x) = \frac{e^x}{e^x + 1}$$

Εικόνα 11 Συναρτήσεις ενεργοποίησης

Οι περισσότερες συναρτήσεις ενεργοποίησης είναι μη γραμμικές και επιλέγονται με αυτόν τον τρόπο σκόπιμα. Η ύπαρξη λειτουργιών μη γραμμικής ενεργοποίησης επιτρέπει στα νευρικά μας δίκτυα να υπολογίζουν αυθαίρετα πολύπλοκες λειτουργίες.

Εκτελώντας την εντολή «model.summary()» στην παρακάτω εικόνα παρουσιάζουμε το πλήθος των παραμέτρων εκπαίδευσης του νευρωνικού μας δικτύου.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                48
_________________________________________________________________
dense_2 (Dense)              (None, 32)                544
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 625
Trainable params: 625
Non-trainable params: 0
```

Εικόνα 12 Πλήθος παραμέτρων εκπαίδευσης του νευρωνικού μοντέλου.

Στο επίπεδο εισόδου δεν αναφέρετε κάποια παράμετρος εκπαίδευσης, διότι στην είσοδο δεν γίνεται κάποιου είδους εκπαίδευσης, απλά εισάγουμε τα δεδομένα μας ώστε να διοχετευτούν στα κρυφά επίπεδα και να ξεκινήσει η διαδικασία της εκπαίδευσης. Η εκπαίδευση ξεκινά από το πρώτο κρυφό επίπεδο. Όταν εκπαιδεύουμε ένα μοντέλο, βασικά προσπαθούμε να λύσουμε ένα πρόβλημα βελτιστοποίησης. Προσπαθούμε να βελτιστοποιήσουμε τα βάρη του μοντέλου. Ο στόχος μας είναι να βρούμε τα βάρη που αντιστοιχούν με ακρίβεια τα δεδομένα εισόδου μας στη σωστή κλάση εξόδου. Αυτή η χαρτογράφηση είναι αυτό που πρέπει να μάθει το δίκτυο. Κάθε σύνδεση μεταξύ κόμβων έχει ένα αυθαίρετο βάρος. Κατά τη διάρκεια της εκπαίδευσης, αυτά τα βάρη ενημερώνονται επαναληπτικά και μεταφέρονται προς τις βέλτιστες τιμές τους.

Προετοιμάζουμε το μοντέλο για εκπαίδευση και καλούμε τη συνάρτηση compile(). Αυτή η λειτουργία διαμορφώνει το μοντέλο για εκπαίδευση και αναμένει έναν αριθμό παραμέτρων. Πρώτον, καθορίζουμε τον βελτιστοποιητή *Adam*. Ο *Adam* αποδέχεται μια προαιρετική παράμετρο *learning_rate*, την οποία μπορούμε να ορίσουμε από 0,1 εώς 0,0001. Η εξ ορισμού η τιμή είναι 0,001, εδώ αφήνουμε τη συγκεκριμένη τιμή. Το ποσοστό μάθησης (*learning_rate)* μας λέει πόσο μεγάλο είναι ένα βήμα που πρέπει να κάνουμε προς την κατεύθυνση της ελάχιστης τιμής. Τα βάρη βελτιστοποιούνται χρησιμοποιώντας αυτό που αποκαλούμε αλγόριθμο βελτιστοποίησης. Η διαδικασία βελτιστοποίησης εξαρτάται από τον επιλεγμένο αλγόριθμο βελτιστοποίησης. Χρησιμοποιούμε επίσης τον όρο βελτιστοποίησης για να αναφερθούμε στον επιλεγμένο αλγόριθμο. Το πιο ευρέως γνωστό εργαλείο βελτιστοποίησης ονομάζεται στοχαστική κατάβαση κλίσης, ή πιο απλά, stochastic

gradient descent ,SGD .Ο ADAM είναι μια παραλαγή του SGD που μπορεί να χρησιμοποιηθεί αντί της διαδικασίας κλασικής στοχαστικής κατάβασης κλίσης για την ενημέρωση επαναληπτικών βαρών δικτύου με βάση τα δεδομένα εκπαίδευσης..

Ο στόχος του SGD είναι να ελαχιστοποιήσει κάποια δεδομένη συνάρτηση που ονομάζουμε συνάρτηση απώλειας. Έτσι, η SGD ενημερώνει τα βάρη του μοντέλου με τέτοιο τρόπο ώστε αυτή η απώλεια να λειτουργεί όσο το δυνατόν πλησιέστερα στην ελάχιστη τιμή.

Η επόμενη παράμετρος που καθορίζουμε είναι η loss (απώλεια). Η απώλεια είναι το σφάλμα ή η διαφορά μεταξύ του τι προβλέπει το δίκτυο έναντι της πραγματικότητας και η SGD θα προσπαθήσει να ελαχιστοποιήσει αυτό το σφάλμα για να κάνει το μοντέλο μας όσο το δυνατόν ακριβέστερο στις προβλέψεις του. Θα χρησιμοποιούμε την “MSE”. Το μέσο σφάλμα τετραγώνου MSE, είναι η προεπιλεγμένη απώλεια που χρησιμοποιείται για προβλήματα παλινδρόμησης. Το μέσο τετραγωνικό σφάλμα υπολογίζεται ως ο μέσος όρος των τετραγωνικών διαφορών μεταξύ των προβλεπόμενων και των πραγματικών τιμών. Το αποτέλεσμα είναι πάντα θετικό ανεξάρτητα από τις προβλεπόμενες και πραγματικές τιμές και μια τέλεια τιμή είναι 0,0. Το τετράγωνο σημαίνει ότι μεγαλύτερα λάθη οδηγού σε περισσότερα λάθη από μικρότερα λάθη, πράγμα που σημαίνει ότι το μοντέλο τιμωρείται για τα μεγαλύτερα λάθη. Το KERAS παρέχει ένα πλήθος συναρτήσεων απώλειας όπως:

- ✓ mean_squared_error
- ✓ mean_absolute_error
- ✓ mean_absolute_percentage_error
- ✓ mean_squared_logarithmic_error
- ✓ squared_hinge
- ✓ hinge
- ✓ categorical_hinge
- ✓ logcosh
- ✓ categorical_crossentropy
- ✓ sparse_categorical_crossentropy
- ✓ binary_crossentropy
- ✓ kullback_leibler_divergence
- ✓ poisson
- ✓ cosine_proximity

Προχωρώντας, η τελευταία παράμετρος που καθορίζουμε στη *compile()* είναι η *metrics*. Αυτή η παράμετρος αναμένει μια λίστα μετρήσεων που θα θέλαμε να αξιολογηθεί από το μοντέλο κατά τη διάρκεια της εκπαιδευση και των δοκιμών. Θα το ορίσουμε σε μια λίστα που περιέχει τη συμβολοσειρά 'accuracy' «ακρίβεια».

```
92 model.compile(
93     loss='mse',
94     optimizer='adam',
95     metrics=['accuracy']
96 )
```

Αρχικοποιούμε το χρόνο ώστε να μετρήσουμε ποσο διαρκεί η εκπαιδευση του μοντέλου.

```
96 start = datetime.datetime.now()
```

Δημιουργούμε το αρχείο οπου θα αποθηκευτούν οι μετρήσεις verbose.

```
101 csv_logger = CSVLogger('Verbose_Train.csv', append=True, separator=';')
```

Τώρα που έχει δημιουργηθεί το μοντέλο, μπορούμε να το εκπαιδεύσουμε χρησιμοποιώντας τη συνάρτηση fit (). Τροφοδοτούμε με τον πίνακα *InDNN* που δημιουργήσαμε πιο πάνω. Εισάγουμε τον πίνακα με τις ετικέτες *train_labels* που δημιουργηθηκε από το κριτηριο (T>1) και (L=<5000). Με την τιμή δέκα (10) στην μεταβλητή batch ορίζουμε ότι τα δεδομένα θα εισάγονται ανά δεκάδες στο νευρωνικό μας δίκτυο. Το epochs αναφέρεται σε ένα μόνο πέρασμα ολόκληρου του συνόλου δεδομένων στο δίκτυο κατά τη διάρκεια της εκπαίδευσης, με άλλα λόγια μας δείχνει ποσες φορες θα περάσει το σύνολο των δεδομένων από το δίκτυο, εδώ ορίζουμε το πέρασμα του συνόλου των δεδομενων να γίνει πενήντα (50) φορές. Αφού περάσουμε όλα τα δεδομένα μας μέσω του μοντέλου μας, θα συνεχίσουμε να διαβάζουμε τα ίδια δεδομένα ξανά και ξανά. Αυτή η διαδικασία της επαναλαμβανόμενης αποστολής των ίδιων δεδομένων μέσω του δικτύου θεωρείται εκπαίδευση. Κατά τη διάρκεια αυτής της εκπαιδευτικής διαδικασίας το μοντέλο θα μάθει πραγματικά. Έτσι, μέσω αυτής της διαδικασίας που συμβαίνει επαναληπτικά με το SGD, το μοντέλο μπορεί να μάθει από τα δεδομένα. Αυτό το πέρασμα μέσω του δικτύου από είσοδο σε έξοδο ονομάζεται forward pass. Τέλος, καθορίζουμε verbose = 0. Αυτό καθορίζει απλώς πόση έξοδο θέλουμε να εκτύπωνεται σε κάθε epoch εκπαίδευσης (Εικόνα 13). Οι τιμές κυμαίνονται από 0 έως 2. Με την παράμετρο callbacks αποθηκεύουμε τις μετρήσεις verbose σε αρχείο excel που έχουμε δημιουργήσει πιο πάνω.

```
102 model.fit(
103     InDNN,
104     train_labels,
105     batch_size=10,
106     epochs=50,
107     shuffle=True,
108     verbose=0,
109     callbacks=[csv_logger]
110 )
```

```
 - 0s - loss: 0.2238 - accuracy: 0.6710
Epoch 3/30
 - 0s - loss: 0.2200 - accuracy: 0.6710
Epoch 4/30
 - 0s - loss: 0.2167 - accuracy: 0.6710
Epoch 5/30
 - 0s - loss: 0.2137 - accuracy: 0.6710
Epoch 6/30
 - 0s - loss: 0.2114 - accuracy: 0.6710
Epoch 7/30
 - 0s - loss: 0.2092 - accuracy: 0.6710
Epoch 8/30
 - 0s - loss: 0.2070 - accuracy: 0.6710
Epoch 9/30
 - 0s - loss: 0.2044 - accuracy: 0.6710
Epoch 10/30
```

Εικόνα 13 Έξοδος Verbose τιμή 2

Δημιουργούμε το αρχείο όπου θα αποθηκευτούν οι μετρήσεις του σετ επικύρωσης.

```
114 csv_logger = CSVLogger('Verbose_Valid.csv', append=True, separator=';')
```

Στον προγραμματισμό νευρωνικών δικτύων, η ενημέρωση των βαρών είναι ουσιαστικά αυτό που εννοούμε όταν λέμε ότι το μοντέλο μαθαίνει. Αφού υπολογιστεί η απώλεια, υπολογίζεται η κλίση αυτής της λειτουργίας απώλειας σε σχέση με κάθε ένα από τα βάρη εντός του δικτύου. Αυτός ο υπολογισμός γίνεται χρησιμοποιώντας μια τεχνική που ονομάζεται backpropagation. Η διαδικασία backpropagation χρησιμοποιεί λογισμούς. Το backpropagation προσπαθεί να υπολογίσει το παράγωγο της απώλειας σε σχέση με κάθε βάρος. Για να γίνει αυτός ο υπολογισμός, το backpropagation χρησιμοποιεί τον κανόνα αλυσίδας για να υπολογίσει την κλίση της συνάρτησης απώλειας. Ο κανόνα της αλυσίδας στον λογισμό χρησιμοποιείται ως μέθοδος υπολογισμού του παραγώγου της σύνθεσης δύο ή περισσότερων συναρτήσεων.

Μόλις έχουμε την τιμή για τη διαβάθμιση της λειτουργίας απώλειας, μπορούμε να χρησιμοποιήσουμε αυτήν την τιμή για να ενημερώσουμε το βάρος του μοντέλου. Η κλίση μας λέει ποια κατεύθυνση θα μετακινήσει την απώλεια στο ελάχιστο, και το καθήκον μας είναι να κινηθούμε προς μια κατεύθυνση που μειώνει την απώλεια και

πλησιάζει αυτήν την ελάχιστη τιμή. Στη συνέχεια πολλαπλασιάζουμε την τιμή διαβάθμισης με το ποσοστό εκμάθησης. Συνεχίζουμε πολλαπλασιάζοντας την κλίση με το ρυθμό εκμάθησης και αφαιρούμε αυτό το γινόμενο από το βάρος, το οποίο θα μας δώσει τη νέα ενημερωμένη τιμή για αυτό το βάρος.

new weight = old weight - (learning rate * gradient)

Η παραπάνω ίδια διαδικασία θα συμβεί με κάθε ένα από τα βάρη του μοντέλου κάθε φορά που τα δεδομένα περνούν από αυτό. Η μόνη διαφορά είναι ότι όταν υπολογίζεται η κλίση της συνάρτησης απώλειας, η τιμή για την κλίση θα είναι διαφορετική για κάθε βάρος, επειδή η κλίση υπολογίζεται σε σχέση με κάθε βάρος. Όλα αυτά τα βάρη ενημερώνονται επαναληπτικά με κάθε epoch. Τα βάρη σταδιακά πλησιάζουν και πλησιάζουν στις βελτιστοποιημένες τιμές τους, ενώ το SGD λειτουργεί για να ελαχιστοποιήσει τη λειτουργία απώλειας. Παρακάτω συνοψίζουμε τα βήματα που λαμβάνουν χώρα σε ένα κύκλο εκπαίδευσης :

- ✓ Υπολογισμός απώλειας.
- ✓ Υπολογισμός της κλίσης της απώλειας σε σχέση με κάθε ένα από τα βάρη.
- ✓ Πολλαπλασιασμός των κλίσεων με το ποσοστό εκμάθησης.
- ✓ Υπολογισμός των νέων τιμών των βαρών.
- ✓ Κατάργηση των προηγούμενων βάρών που είχαν οριστεί σε κάθε σύνδεση και ενημέρωση με τις νέες τιμές.

Αφού τελειώσει η εκπαίδευση είναι απαραίτητο να επικυρώσουμε ότι η παραπάνω εκπαίδευση είναι αποτελεσματική. Γιαυτόν τον λόγο δημιουργούμε ένα σετ επικύρωσης δεδομένων ξεχωριστό από το εκπαιδευτικό σύνολο και στην ουσία αξιολογούμε εάν η προηγούμενη εκπαίδευση θα έχει αποτέλεσμα σε γενικευμένα δεδομένα. Αυτή η διαδικασία επικύρωσης βοηθά στην παροχή πληροφοριών που μπορεί να μας βοηθήσουν στην προσαρμογή των παραμέτρων μας. Επίσης, κατά τη διάρκεια της εκπαίδευσης, το μοντέλο θα προβλέπει για κάθε είσοδο και από το σετ επικύρωσης. Θα κάνει αυτήν την πρόβλεψη βασισμένη μόνο σε ό, τι έχει μάθει για τα δεδομένα στα οποία εκπαιδεύεται στο εκπαιδευτικό σύνολο. Τα βάρη δεν θα ενημερωθούν στο μοντέλο με βάση την απώλεια που υπολογίζεται από τα δεδομένα επικύρωσης.

Ένας από τους κύριους λόγους που χρειαζόμαστε ένα σετ επικύρωσης είναι να διασφαλίσουμε ότι στο μοντέλο μας δεν θα γίνεται overfitting ή underfitting με τα

δεδομένα στο εκπαιδευτικό σύνολο. Κατά τη διάρκεια της εκπαίδευσης, εάν επικυρώνουμε επίσης το μοντέλο στο σύνολο επικύρωσης και βλέπουμε ότι τα αποτελέσματα που δίνει για τα δεδομένα επικύρωσης είναι εξίσου καλά με τα αποτελέσματα που δίνει για τα δεδομένα εκπαίδευσης, τότε μπορούμε να είμαστε πιο σίγουροι ότι το μοντέλο μας δεν θα έχει πρόβλημα overfitting. Το πρόβλημα του overfitting παρουσιάζετε όταν τα αποτελέσματα στα δεδομένα εκπαίδευσης είναι πραγματικά καλά, αλλά τα αποτελέσματα στα δεδομένα επικύρωσης υστερούν. Ενώ εάν ένα μοντέλο δεν είναι σε θέση να προβλέψει ορθά στα δεδομένα στα οποία εκπαιδεύτηκε λέγεται ότι είναι σε κατάσταση Underfitting. Για να αποφύγουμε τα παραπάνω προβλήματα μπορούμε να προβούμε σε αλλάγες των παραμέτρων του μοντέλου μας, όπως και θα κάνουμε στο πειραματικό μέρος της εργασίας.

Προχωρούμε στην δημιουργία δεδομένων επικύρωσης εισάγοντας τους πίνακες InDNN και train_labels. Ορίζω μία τιμή για την παράμετρο validation_split που πρέπει να κυμαίνεται ένας δεκαδικός αριθμός από το 0 εως το 1. Με αυτή την παράμετρο, το Keras θα διαχωρίσει ένα κλάσμα (10% σε αυτό το παράδειγμα) των δεδομένων εκπαίδευσης που θα χρησιμοποιηθούν ως δεδομένα επικύρωσης. Το μοντέλο θα διαχωρίσει αυτό το κλάσμα των δεδομένων εκπαίδευσης, δεν θα το εκπαιδεύσει και θα αξιολογήσει την απώλεια και τυχόν μετρήσεις μοντέλου σε αυτά τα δεδομένα στο τέλος κάθε epoch. Η συνάρτηση fit () πριν από κάθε epoch, από προεπιλογή, αλλάζει την σειρά των δεδομένων ώστε να μην υπάρχει ο κίνδυνος εμφάνισης επαναλαμβανόμενης όμοιας σειράς. Αυτό θα οδηγούσε σε εσφαλμένα συμπεράσματα, διότι το μοντέλο θα έλεγχε το ίδιο ζεύγος τιμών, δεν θα μάθαινε σωστά και δεν θα γενίκευε ορθά σε νέα άγνωστα δεδομένα. . Με την παράμετρο callbacks αποθηκεύουμε τις μετρήσεις σε αρχείο excel που έχουμε δημιουργήσει πιο πάνω.

```
116 model.fit(
117       InDNN,
118       train_labels,
119       validation_split=0.1,
120       batch_size=10,
121       epochs=50,
122       verbose=0,
123       callbacks=[csv_logger]
124 )
```

Αφού επιλέξαμε verbose=2 θα εμφανιστούν τα αποτελεσματα της μορφής που φαίνονται στην εικόνα 14.

```
Train on 900 samples, validate on 100 samples
Epoch 1/30
 - 0s - loss: 0.1215 - accuracy: 0.8644 - val_loss: 0.1236 - val_accuracy: 0.8400
Epoch 2/30
 - 0s - loss: 0.1195 - accuracy: 0.8644 - val_loss: 0.1220 - val_accuracy: 0.8400
Epoch 3/30
 - 0s - loss: 0.1178 - accuracy: 0.8600 - val_loss: 0.1192 - val_accuracy: 0.8500
Epoch 4/30
 - 0s - loss: 0.1160 - accuracy: 0.8633 - val_loss: 0.1177 - val_accuracy: 0.8500
Epoch 5/30
 - 0s - loss: 0.1144 - accuracy: 0.8667 - val_loss: 0.1154 - val_accuracy: 0.8500
Epoch 6/30
 - 0s - loss: 0.1130 - accuracy: 0.8644 - val_loss: 0.1139 - val_accuracy: 0.8500
Epoch 7/30
 - 0s - loss: 0.1115 - accuracy: 0.8656 - val_loss: 0.1121 - val_accuracy: 0.8600
Epoch 8/30
```

Εικόνα 14 Verbose Validation test

Μετρούμε τον χρόνο που διήρκεσε η εκπαίδευση και τον εκτυπωνουμε στο τέλος του προγράμματος.

```
117 end = datetime.datetime.now()
118 totalTime = end - start
119 print("### Training time : ", totalTime, " seconds")
```

4.2.2 ΔΕΥΤΕΡΟ ΜΕΡΟΣ – ΚΩΔΙΚΑΣ ΔΟΚΙΜΑΣΙΑΣ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ

Ορίζουμε τις νέες παραμέτρους ώστε να ξεκινήσει η δοκιμασία και η εξαγωγή των συμπερασμάτων.

```
139 Experiments=10
140 loadThreshold = 0.5
141 demandthreshold = 0.5
142 ThresholdBenefitRate = 1
143
144 Tasks=1000
145 k = 10
146 a = 5
147
148 final = np.zeros(Tasks)
149 final_bad= np.zeros(Tasks)
150 load = np.zeros(Tasks)
151
152 # variables for metrics
153 sumTime = 0.0
154 sumTimes = []
155
156 percentageCorrectBenefit_Rate=[]
157 percentageCorrectDemand_Load=[]
158 percentageCorrectDemand=[]
159 percentageCorrectLoad=[]
160 SumAveragePopoularLoad=[]
161 SumAverageRareLoad=[]
```

Αρχίζουμε τον κύκλο τον πειραμάτων και ορίζουμε τους νέους πίνακες που θα αποτελέσουν τους υποδοχείς των νέων τιμών.

```
151 for e in range(Experiments):
152     test_labels =  []
153     test_demand = []
154     test_Load = []
155 
156     est1=[]
157     est2=[]
158     Final_Demand=[]
159     Final_Load=[]
160     Final_Bad_demand=[]
161     Final_Bad_Load =[]
162     Tfinal = []
163     Final_Benefit = []
```

Παράγουμε τις τυχαίες τιμές των εργασιών για τους πίνακες δημοφιλίας test_demand και του φορτίου των εργασιών test_load και αλλάζουμε την σειρά των παραγόμενων τιμών ώστε να αποφύγουμε τον κίνδυνο εμφάνισης όμοιου επαναλαμβανόμενου ζεύγους τιμών. Όπως αναφέρθηκε και παραπάνω αυτό θα μας οδηγούσε σε λανθασμένα συμπεράσματα .

```
165     for i in range(Tasks):
166         # produce Demand
167         random_test = randint(1,100)
168         test_demand.append(random_test)
169         # produce Load
170         random_load_test = randint(1,100)
171         test_Load.append(random_load_test)
172 
173     test_demand = np.array(test_demand)
174     test_Load=np.array(test_Load)
175     test_demand, test_Load = shuffle(test_demand, test_Load)
176 
```

Θα χρησιμοποιήσουμε την κλάση MinMaxScaler της scikit-learn για να μειώσουμε όλα τα δεδομένα από μια κλίμακα που κυμαίνεται από 1 έως 100 σε κλίμακα από 0 έως 1. Αναδιαμορφώνουμε τα δεδομένα ως τεχνική απαίτηση, αφού η συνάρτηση fit_transform() δεν δέχεται δεδομένα μίας διάστασης (1D) από προεπιλογή.

```
177     scaler = MinMaxScaler(feature_range=(0,1))
178     scaled_test_demand = scaler.fit_transform(test_demand.reshape(-1,1))
179     scaled_test_Load = scaler.fit_transform(test_Load.reshape(-1,1))
```

Αρχίζουμε να μετράμε την διάρκεια της πρόβλεψης. Ενοποιούμε τους δύο πίνακες που αφορούν το DEMAND και το LOAD σε έναν πίνακα τον *InDNN* ώστε να τον εισάγουμε στην είσοδο του νευρωνικού δικτύου που απαιτεί πίνακα δύο στηλών.

```
181     sum = 0.0
182     count = 0.0
183
184     start = datetime.datetime.now()
185
186
187     InDNN_Tasks = np.concatenate((scaled_test_demand,scaled_test_Load), axis=1)
188
189
```

Το σετ δοκιμών είναι ένα σύνολο δεδομένων που χρησιμοποιούνται για τη δοκιμή του μοντέλου αφού το μοντέλο έχει ήδη εκπαιδευτεί. Το συγκεκριμένο σύνολο τιμών είναι ξεχωριστό τόσο από το σετ προπόνησης όσο και από το σετ επικύρωσης. Αφού εκπαιδευτεί και επικυρωθεί το μοντέλο μας χρησιμοποιώντας τα σύνολα εκπαίδευσης και επικύρωσης, θα χρησιμοποιήσουμε το μοντέλο μας για να προβλέψουμε την έξοδο των δεδομένων χωρίς ετικέτα στο σύνολο δοκιμών.

Μία σημαντική διαφορά μεταξύ του σετ δοκιμής και των δύο άλλων συνόλων είναι ότι το σετ δοκιμής δεν πρέπει να φέρει ετικέτα. Το σετ εκπαίδευσης και το σετ επικύρωσης πρέπει να φέρουν ετικέτα έτσι ώστε να μπορούμε να δούμε τις μετρήσεις που δίνονται κατά τη διάρκεια της εκπαίδευσης, όπως η απώλεια και η ακρίβεια από κάθε epoch. Για να λάβουμε προβλέψεις από το μοντέλο για το σύνολο δοκιμών, καλούμε το model.predict_classes(). Χρησιμοποιούμε το “model.predict_classes()” για να έχουμε πρόβλεψη δύο τιμών (0) και (1), ειδάλως θα χρησιμοποιούσαμε το “model.predict()” όπου μας παρέχει μία πιο λεπτομερέστερη έξοδο τιμών. Σε αυτή τη συνάρτηση, περνάμε τα δοκιμαστικά δείγματα δηλαδή τον πίνακα InDNN και καθορίζουμε το batch_size. Επίσης ορίζουμε τι θέλουμε να εκτυπώνεται από τα μηνύματα καταγραφής κατά τη δημιουργία προβλέψεων, εδώ ορίζουμε verbose = 0 χωρίς έξοδο. Όπως αναφέραμε σε αντίθεση με τα σετ εκπαίδευσης και επικύρωσης, δεν περνάμε τις ετικέτες του σετ δοκιμής στο μοντέλο κατά τη διάρκεια του σταδίου συμπερασμάτων, περιμένουμε το μοντέλο μετά την εκπαίδευση που έλαβε παραπάνω, να κάνει σωστή πρόβλεψη.

```
200     predictions = model.predict_classes(
201           InDNN_Tasks
202         , batch_size=10
203         , verbose=0
204     )
```

Αφού το μοντέλο έχει αποφασίσει για το ποιο ζευγος του πίνακα InDNN παίρνει την τιμή (1) και ποιο παίρνει την τιμή (0), δημιουργούμε δύο καινούργιους πίνακες. Ο πρώτος πίνακας ο «optimalTask» περιλαμβάνει τις τιμές που έχουν υψηλή δημοφηλία και μικρό βάρος εργασίας, άρα το μοντέλο έδωσε την τιμή (1) και ο δεύτερος πίνακας ο «badTask» χαμηλή δημοφιλία και υψηλά βάρη εργασίας άρα το ζεύγος των τιμών πήρε την τιμή (0), σύμφωνα πάντα με την πρόβλεψη του νευρωνικού δικτύου.

```
212     optimalTask=np.where(predictions==1,InDNN_Tasks,0)
213     # create a new array from InDNN if prediction is 0
214     badTask=np.where(predictions == 0,InDNN_Tasks,0)
215
216     #remove rows with zero
217     optimalTask = optimalTask[~np.any(optimalTask == 0, axis=1)]
218     badTask=badTask[~np.any(badTask == 0, axis=1)]
```

Ανακτούμε τις διαστάσεις των νέων πινάκων. Αυτό θα μας βοηθήσει αργότερα ώστε να εξάγουμε στατιστικά στοιχεία για τον έλεγχο της ορθής λειτουργίας του νευρωνικού μας δικτύου. Εξάλλου γνωρίζοντας τις διαστάσεις των πινάκων ξέρουμε και το πλήθος των εργασιών που έχουν επιλεγεί για να εκτελεστούν στον κόμβο μας.

```
221     shapeOptimal=optimalTask.shape
222     shapebad=badTask.shape
223     # set j equal shape array for Optimal and set b equal for bad
224     j=shapeOptimal[0]
225     b=shapebad[0]
```

Διαχωρίζουμε τους παραπάνω πίνακες σε πίνακα δημοφιλίας και βαρών διαστάσεων μίας στήλης. Αυτό το κάνουμε για να μπορέσουμε να υπολογίσουμε το συντελεστή οφέλους.

```
230     est1, est2 =np.hsplit(optimalTask, 2)
231     badDemand, badload = np.hsplit(badTask, 2)
```

Κατά την παραγωγή των τυχαίων αριθμών όπως είναι φυσικό παράγετε και η τιμή μηδέν (0) και παρεισφρέει στις τιμές των πινάκων για το DEMAND και το LOAD. Για τον υπολογισμό του συντελεστή οφέλους σύμφωνα με το παρακάτω τύπο απαιτείται η διαίρεση των τιμών του πίνακα DEMAND (est1) από τις τιμές του πίνακα LOAD (est2).

Συντελεστής Οφέλους =exp(DEMAND/LOAD - a)

Σύμφωνα με τα παραπάνω υπάρχει ο κίνδυνος να έχουμε διαίρεση με το μηδέν κάτι το οποίο προκαλεί σφάλμα στην εκτέλεση του προγράμματος . Για να μην απορίψουμε καμία παραγομενη τιμή και για μεγαλύτερη ακρίβεια, γιαυτόν τον λόγο αντικαθιστούμε το μηδέν με ένα πολύ μικρό αριθμό τον 0,0001 ώστε να προσομοιάζει με το μηδέν και να είναι δυνατή η διάιρεση .

```
234     est22= np.where(est2 == 0,0.0001,est2)
235     badload1=np.where(badload == 0,0.0001,badload)
```

Υπολογίζουμε το συντελεστή οφέλους .

```
237     #calcuate the Benefit value
238     final= np.exp(((est1/est22)-a))
239     final_bad=np.exp(((badDemand/badload1)-a))
```

Ενοποιούμε τους πίνακες, συντελεστή οφέλους , Demand και Load σε έναν πίνακα τριών στηλών και ταξινομούμαι το νέο πίνακα κατά φθίνουσα σειρά με βαση την πρώτη στήλη δηλαδή την στήλη του συντελεστή οφέλους. Με την φθίνουσα ταξινόμηση θα επιτύχουμε να τοποθετήσουμε στις τελευταίες θέσεις, τις εργασίες με τον μικρότερο συντελεστή οφέλους ,ώστε να αφαιρέσουμε επιπλέον εργασίες που δεν πληρούν την προυπόθεση της δημοφιλίας και του βάρους. Κατά αύξουσα σειρά ταξινομούμε τον πίνακα που περιλαμβάνει τις εργασίες που έχουν απορριφθεί και θα αποσταλλούν στο cloud.

Διαχωρίζουμε τον πίνακα των τριών στηλών σε τρεις πίνακες της μίας στήλης που περιλαμβάνει ένα πίνακα με τη δημοφιλία Demand, ένα πίνακα φορτίου εργασίων Load ώστε να κάνουμε τους απαραίτητους ελέγχους και να ελέγξουμε το ποσοστό των σωστών αποφάσεων του νευρωνικού δικτύου.

```
251     final, Final_Demand, Final_Load=np.hsplit(FFF,3)
252     final_bad, Final_Bad_demand, Final_Bad_Load=np.hsplit(BBB,3)
```

Κρατάμε τις τελικές τιμές αφού αφαιρέσουμε τις τελευταίες K τιμές.

```
254     Final_Bf_Rt=[]
255     Final_Bf_demand=[]
256     Final_Bf_load =[]
257
258     # dismiss tasks low benefit
259     for i in range(j-k):
260         Final_Bf_Rt.append(final[i])
261         Final_Bf_demand.append(Final_Demand[i])
262         Final_Bf_load.append(Final_Load[i])
263
```

Μετρούμε το χρόνο που διήρκησε η διαδικασία του τεστ.

```
270     end = datetime.datetime.now()
271     totalTime = end - start
272     sum += totalTime.total_seconds()
273     count += 1
```

4.2.3 ΤΡΙΤΟ ΜΕΡΟΣ – ΚΩΔΙΚΑΣ ΕΛΕΓΧΟΥ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ

Βρίσκουμε το μέσο όρο των εργασιών που επιλέχθηκαν ως δημοφιλείς με μικρό φορτίο εργασιών και θα εκτελεστούν τοπικά στον κόμβο μας και τον μέσο όρο των εργασιών που θα αποσταλούν στο cloud ή σε άλλο κόμβο λόγω μη δημοφιλίας και υψηλού φόρτου.

```
264     AveragePopoularLoad=((j-k)/Tasks)
265     AverageRareLoad=((b+k)/Tasks)
266
267     SumAveragePopoularLoad.append(AveragePopoularLoad)
268     SumAverageRareLoad.append(AverageRareLoad)
```

Αποθηκεύουμε τους τελικούς πίνακες σε αρχείο excel.

```
285     df_InDNN_Tasks = pd.DataFrame(InDNN_Tasks)
286     df_predictions = pd.DataFrame(predictions)
287     df_est2 = pd.DataFrame(est22)
288     df_est1 = pd.DataFrame(est1)
289     df_FFF=pd.DataFrame(FFF)
290     df_BBB=pd.DataFrame(BBB)
291     df_Final_Demand=pd.DataFrame(Final_Bf_demand) #Final_Demand
292     df_Final_Load=pd.DataFrame(Final_Bf_load) #Final_Load
293     df_Final_Rt=pd.DataFrame(Final_Bf_Rt)
294     df_final = pd.DataFrame(final)
295     df_Final_All=pd.DataFrame(Final_All)
296
297
298     writer = pd.ExcelWriter('my_excel_file.xlsx')
299     df_InDNN_Tasks.to_excel(writer,'InDNN',float_format='%.5f')
300     df_predictions.to_excel(writer,'predictions',float_format='%.5f')
301     df_final.to_excel(writer,'final_and_K',float_format='%.5f')
302     df_est1.to_excel(writer,'Demand_and_K',float_format='%.5f')
303     df_est2.to_excel(writer,'Load_and_K',float_format='%.5f')
304     df_FFF.to_excel(writer,'Final_All_and_K',float_format='%.5f')
305     df_BBB.to_excel(writer,'Final_BAD_All',float_format='%.5f')
306     df_Final_Demand.to_excel(writer,'Final_Demand',float_format='%.5f')
307     df_Final_Load.to_excel(writer,'Final_Load',float_format='%.5f')
308     df_Final_Rt.to_excel(writer,'Final_Bf',float_format='%.5f')
309     df_Final_All.to_excel(writer,'Final_All',float_format='%.5f')
310     writer.save()
```

Δημιουργούμε τους ελέγχους για να ελέγξουμε κατά πόσο επί τις εκατό το νευρωνικό μας δίκτυο αποφασίζει σωστά.

```
314     counterDemand_Load_Corect=0.0
315     counterbenefit_rate=0.0
316     Rightdemand1 = 0.0
317     counterLoadPopular = 0.0
318
319     for i in range (j-k):
320         if Final_Bf_demand[i] > demandthreshold and Final_Bf_load[i]<= loadThreshold:
321           counterDemand_Load_Corect +=1
322         elif Final_Bf_Rt[i]>=ThresholdBenefitRate:
323           counterDemand_Load_Corect +=1
324     for i in range (j-k):
325         if Final_Bf_Rt[i]>=ThresholdBenefitRate:
326           counterbenefit_rate +=1
327     for i in range (j-k):
328         if Final_Bf_demand[i] > demandthreshold:
329            Rightdemand1 +=1
330     for i in range (j-k):
331         if Final_Bf_load[i]<= loadThreshold:
332            counterLoadPopular +=1
```

Βρίσκουμε τους μέσους όρους των παραπάνω ελέγχων.

```
    percentageCorrectDemand_Load.append(counterDemand_Load_Corect/(j-k))
    percentageCorrectBenefit_Rate.append(counterbenefit_rate/(j-k))
    percentageCorrectDemand.append((Rightdemand1)/(j-k))
    percentageCorrectLoad.append((counterLoadPopular)/(j-k))
```

Βρίσκουμε το μέσο όρο του χρόνου που χρειάστηκε για να γίνει η δοκιμασία του νευρωνικού μας δικτύου και εκτυπώνουμε αυτόν το χρόνο.

```
sumTime += (sum / count)
sumTimes.append(sum / count)

averageTime = statistics.mean(sumTimes)
print("Average time for getting result (tau): ", averageTime)
```

Βρίσκουμε το μέσο όρο των παραπάνω μετρήσεων καθώς επίσης και της τυπικής απόκλισης αυτών.

```
averagePercentageCorrect_Demand_Load= statistics.mean(percentageCorrectDemand_Load)
DeviationaveragePercentageCorrect_Demand_Load= statistics.stdev(percentageCorrectDemand_Load)

averagepercentageCorrectBenefit_Rate=statistics.mean(percentageCorrectBenefit_Rate)
DeviationaveragepercentageCorrectBenefit_Rate=statistics.stdev(percentageCorrectBenefit_Rate)


averagePercentageRight = statistics.mean(percentageCorrectDemand)
DeviationaveragePercentageRight = statistics.stdev(percentageCorrectDemand)


averagepercentageCorrectLoad = statistics.mean(percentageCorrectLoad)
DeviationaveragepercentageCorrectLoad = statistics.stdev(percentageCorrectLoad)


PercentageSumAveragePopoularLoad=statistics.mean(SumAveragePopoularLoad)
DeviationPercentageSumAveragePopoularLoad=statistics.stdev(SumAveragePopoularLoad)

PercentageSumAverageRareLoad=statistics.mean(SumAverageRareLoad)
DeviationPercentageSumAverageRareLoad=statistics.stdev(SumAverageRareLoad)
```

Αποθηκεύουμε τις παραπάνω τιμές σε αρχείο excel.

```
workbook = xlsxwriter.Workbook('Total_data.xlsx')
worksheet = workbook.add_worksheet()

worksheet.write(0, 1, averagePercentageCorrect_Demand_Load)
worksheet.write(1, 1, DeviationaveragePercentageCorrect_Demand_Load)
worksheet.write(2, 1, averagepercentageCorrectBenefit_Rate)
worksheet.write(3, 1, DeviationaveragepercentageCorrectBenefit_Rate)
worksheet.write(4, 1, averagePercentageRight)
worksheet.write(5, 1, DeviationaveragePercentageRight)
worksheet.write(6, 1, averagepercentageCorrectLoad)
worksheet.write(7, 1, DeviationaveragepercentageCorrectLoad)
worksheet.write(8, 1, PercentageSumAveragePopoularLoad)
worksheet.write(9, 1, DeviationPercentageSumAveragePopoularLoad)
worksheet.write(10, 1, PercentageSumAverageRareLoad)
worksheet.write(11, 1, DeviationPercentageSumAverageRareLoad)

workbook.close()
```

Εκτυπώνουμε τις παραπάνω τιμές.

```
379 print("------------------------")
380 print("Average percentage CorrectDemand_and_load",averagePercentageCorrect_Demand_Load)
381 print("Deviation Average percentage CorrectDemand_and_load",DeviationaveragePercentageCorrect_Demand_Load)
382 print("------------------------")
383 print("Average percentage Correct Benefit_Rate",averagepercentageCorrectBenefit_Rate)
384 print("Deviation Average percentage Correct Benefit_Rate",DeviationaveragepercentageCorrectBenefit_Rate)
385 print("------------------------")
386 print("Average percentage CorrectDemand",averagePercentageRight)
387 print("Deviation Average percentage CorrectDemand",DeviationaveragePercentageRight)
388 print("------------------------")
389 print("Average percentage Correct Load ",averagepercentageCorrectLoad)
390 print("Deviation Average percentage Correct Load ",DeviationaveragepercentageCorrectLoad)
391 print("------------------------")
392 print("Pergentage Popoular Tasks",PercentageSumAveragePopoularLoad)
393 print("Deviation Pergentage Popoular Tasks",DeviationPercentageSumAveragePopoularLoad)
394 print("------------------------")
395 print("Pergentage Rare Tasks",PercentageSumAverageRareLoad)
396 print("Deviation Pergentage Rare tasks",DeviationPercentageSumAverageRareLoad)
397 print("------------------------")
```

4.3 Πειραματικό Μέρος

Για να καταλήξουμε στην κατάλληλη παραμετροποίηση του κώδικα του νευρωνικου δικτύου, απαιτήθηκε η διεξαγωγή τριών χιλιάδων εκατόν ενενήντα (3190) πειραμάτων στο μέρος εκπαίδευσης του νευρωνικού μας δικτύου. Σε κάθε κύκλο λειτουργίας του προγράμματος εκτελούνταν από δέκα έως τριάντα πειράματα ταυτόχρονα. Για την διεξαγωγή του πειραματικού μέρους της εργασίας χρησιμοποιήθηκε ηλεκτρονικός υπολογιστής μεσαίων επιδόσεων. Συγκεκριμένα: Επεξεργαστική ισχύ: Intel Celeron N2940 1.83GHz, Μνήμη : 4GB, Λειτουργικό Σύστημα :Windows Pro 64 Bit.

Στόχος ήταν η αναζήτηση των βέλτιστων παραμέτρων εκπαίδευσης ώστε να καταλήξουμε σε ένα νευρωνικό δίκτυο το οποίο θα εκπαιδεύεται σωστά και θα γενικεύει αποδοτικά σε νέα άγνωστα δεδομένα. Όταν λέμε να γενικεύει σωστά, εννοούμε ότι το νευρωνικό δίκτυο δεν θα απομνημονεύει απλά δεδομένα στο στάδιο της εκπαίδευσης και θα χρησιμοποιεί αυτήν την απομνημόνευση στο στάδιο της δοκιμασίας, αλλά μαθαίνει να διαχειρίζεται δεδομένα άγνωστα και τοποθετείται με ορθές αποφάσεις.

Βέβαια το μέρος της εκπαίδευσης είναι καθοριστικότερο στην ορθή λειτουργία ενός νευρωνικού δικτύου αλλά δεν πρέπει να παραγνωρίσουμε και την διαδικασίας της δοκιμασίας (Testing). Στην διαδικασία της δοκιμασίας διεξαγάγαμε δύο χιλιάδες τετρακόσια (2400) πειράματα.

Επιπλέον, τα δεδομένα με τα οποία εκπαιδεύονται τα νευρωνικά δίκτυα διαδραματίζουν πρωτεύοντα ρόλο στην αποτελεσματικότητα του. Με άλλα λόγια, το

πλήθος και η ποικιλία των δεδομένων που τροφοδοτούμε στους νευρώνες κατά τη διάρκεια της εκπαίδευσης, σε συνδυασμό με τις άλλες παραμέτρους δημιουργούν ένα μηχανισμό αποτελεσματικό που αποφασίζει ορθά με μικρό ποσοστό λαθών.

Τα προβλήματα που αντιμετωπίσαμε και θα αναλυθούν στο επόμενο μέρος ήταν πρωτίστως το “Overfitting”, όπου το μοντέλο μας φαινόταν ότι μαθαίνει καλά, αλλά σε δεδομένα άγνωστα δεν είχε την απόδοση που περιμέναμε. Με την ποικιλομορφία των δεδομένων εκπαίδευσης επιλύσαμε το συγκεκριμένο πρόβλημα.

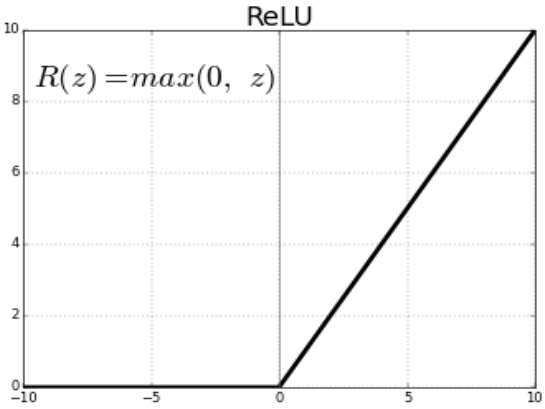
Εξάλλου, το μέγεθος αυτών των βημάτων που κάνουμε για να επιτύχουμε την ελαχιστοποιημένη απώλεια θα εξαρτηθεί από το ποσοστό εκμάθησης. Το ποσοστό εκμάθησης είναι ένας αριθμός που κυμαίνεται από 0,1 έως 0,0001. Εάν αυτός ο αριθμός είναι μεγάλος υπάρχει περίπτωση να έχουμε πρόβλημα υπέρβασης, δηλαδή το μοντέλο μας δεν μπορέσει να προβλέψει και πάλι ορθά. Αυτό συμβαίνει όταν κάνουμε ένα βήμα που είναι πολύ μεγάλο προς την κατεύθυνση της λειτουργίας ελαχιστοποιημένης απώλειας και ξεπερνάμε αυτό το ελάχιστο και το χάνουμε. Εάν ο αριθμός γίνει πολύ μικρός υπάρχει περίπτωση να χρειαζόμαστε περισσότερο χρόνο για να φτάσουμε στην ελάχιστη απώλεια άρα στο βέλτιστο αποτέλεσμα. Το πρόβλημα εδώ βρίσκεται στο γεγονός ότι ένα μικρό ποσοστό εκμάθησης καθιστά επίσης το δίκτυο ευαίσθητο στο να εγκλωβιστεί στο τοπικό ελάχιστο, δηλαδή το δίκτυο θα συγκλίνει σε τοπικά ελάχιστα και δεν μπορεί να βγει από αυτό λόγω του μικρού ποσοστού εκμάθησης.

Η αρχιτεκτονική του δικτύου συνάδει στη σωστή λειτουργία των νευρώνων και κατ’ επέκταση στα ορθά αποτελέσματα. Έτσι ο αριθμός των κρυφών επιπέδων καθώς επίσης και ο αριθμός των κόμβων μπορεί να βελτιστοποιήσει ή να καταρρεύσει ένα δίκτυο . Δεν υπάρχει τυπική αρχιτεκτονική που να μας δίνει υψηλή ακρίβεια σε όλες τις περιπτώσεις δοκιμών.

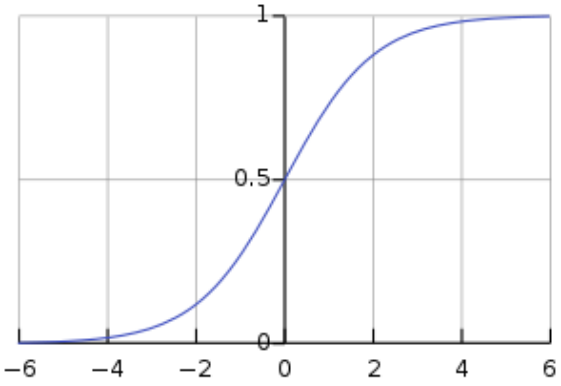
Η λειτουργία βελτιστοποίησης και απώλειας χρησιμοποιεί συναρτήσεις κατηγοριοποιημένης εγκάρσιας εντροπίας (RMSprop, Stochastic Gradient Descent και Adam) που και αυτές επηρεάζουν την αποδοτικότητα του παραπάνω μηχανισμού.

Στο μέγεθος παρτίδας (batch) και στο epoch δεν υπάρχει τυπική τιμή και αυτά με τη σειρά τους ενισχύουν ή μειώνουν την απόδοση της μηχανικής μάθησης. Ο αριθμός των epoch εξαρτάται από την προτίμηση του προγραμματιστή και την υπολογιστική ισχύ που διαθέτει.

Σε ένα τεχνητό νευρικό δίκτυο, μια συνάρτηση ενεργοποίησης είναι μια συνάρτηση που χαρτογραφεί τις εισόδους ενός κόμβου στην αντίστοιχη έξοδο του. Οι περισσότερες συναρτήσεις ενεργοποίησης είναι μη γραμμικές και επιλέγονται με αυτόν τον τρόπο σκόπιμα. Η ύπαρξη λειτουργιών μη γραμμικής ενεργοποίησης επιτρέπει στα νευρικά μας δίκτυα να υπολογίζουν αυθαίρετα πολύπλοκες λειτουργίες. Οι λειτουργίες ενεργοποίησης είναι πολύ σημαντικές και η επιλογή της σωστής λειτουργίας ενεργοποίησης βοηθά το μοντέλο μας να μάθει καλύτερα. Σήμερα, το Rectified Linear Unit (ReLU) (εικόνα 15) είναι η πιο ευρέως χρησιμοποιούμενη λειτουργία ενεργοποίησης, καθώς επιλύει το πρόβλημα της εξαφάνισης των διαβαθμίσεων. Νωρίτερα το Sigmoid (εικόνα 16) και το Tanh ήταν η ευρύτερα χρησιμοποιούμενη λειτουργία ενεργοποίησης. Όμως υπέφεραν από το πρόβλημα της εξαφάνισης των βαθμίδων, δηλαδή κατά τη διάρκεια του backpropagation, οι κλίσεις μειώνονται σε αξία όταν πλησίαζαν στα αρχικά επίπεδα. Αυτό σταματούσε το νευρικό δίκτυο να κλιμακώσει σε μεγαλύτερα μεγέθη με περισσότερα επίπεδα. Το ReLU κατάφερε να ξεπεράσει αυτό το πρόβλημα και ως εκ τούτου επέτρεψε στα νευρωνικά δίκτυα να έχουν μεγάλα μεγέθη.



Εικόνα 16 ReLU



Εικόνα 15 Sigmoid

Τέλος Θα πρέπει να επισημάνουμε τη σημαντικότητα της συνάρτησης απώλειας (loss Function) που συνήθως χρησιμοποιείται το μέσο τετραγωνικό σφάλμα MSE. Το μέσο τετραγωνικό σφάλμα υπολογίζεται ως ο μέσος όρος των τετραγωνικών διαφορών μεταξύ των προβλεπόμενων και των πραγματικών τιμών. Το αποτέλεσμα είναι πάντα θετικό ανεξάρτητα από τις προβλεπόμενες και πραγματικές τιμές και μια τέλεια τιμή είναι 0,0. Το τετράγωνο σημαίνει ότι μεγαλύτερα λάθη οδηγούν σε περισσότερα λάθη από μικρότερα λάθη, πράγμα που σημαίνει ότι το μοντέλο τιμωρείται για τα μεγαλύτερα

λάθη. Στα πειράματα μας χρησιμοποιήσαμε και τη συνάρτηση Cross-entropy loss στην οποία κάθε προβλεπόμενη πιθανότητα συγκρίνεται με την πραγματική τιμή εξόδου κλάσης (0 ή 1) και υπολογίζεται ένα σκορ που τιμωρεί την πιθανότητα με βάση την απόσταση από την αναμενόμενη τιμή. Το πέναλτι είναι λογαριθμικό, προσφέροντας ένα μικρό σκορ για μικρές διαφορές (0,1 ή 0,2) και τεράστιο σκορ για μεγάλη διαφορά (0,9 ή 1,0).

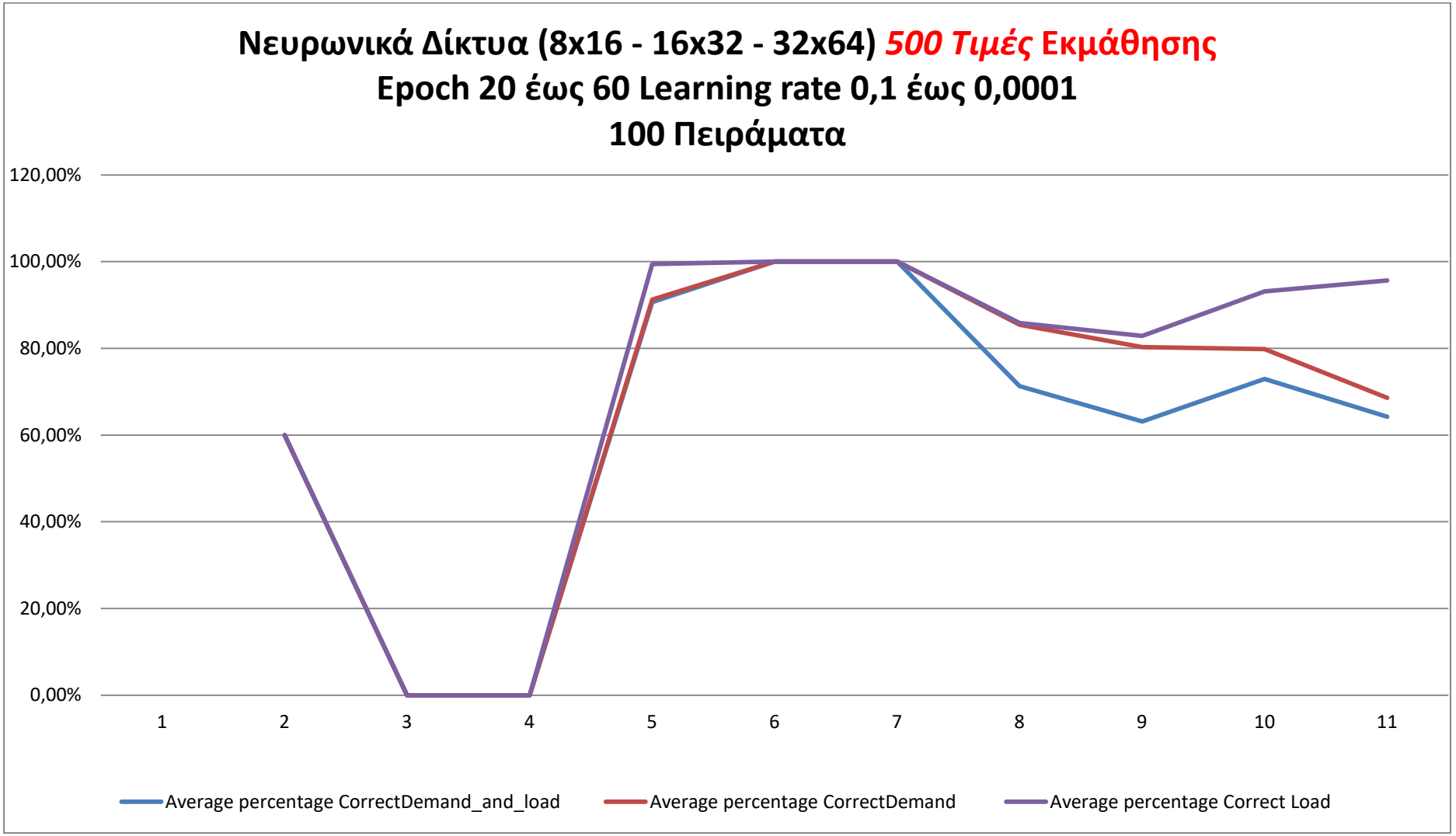
4.3.1. Εκπαίδευση Νευρωνικού Δικτύου

Στόχος του νευρωνικού μας δικτύου είναι η επιλογή εργασιών οι οποίες θα πρέπει να είναι δημοφιλείς (Demand) και να έχουν έναν μικρό φόρτο (Load) εργασίας. Θα πρέπει δηλαδή να τροφοδοτήσουμε το μοντέλο μας με ανάλογα δεδομένα. Στη διαδικασία εκπαίδευσης διεξαγάγαμε τρείς χιλιάδες εκατόν ενενήντα (3190) πειράματα.

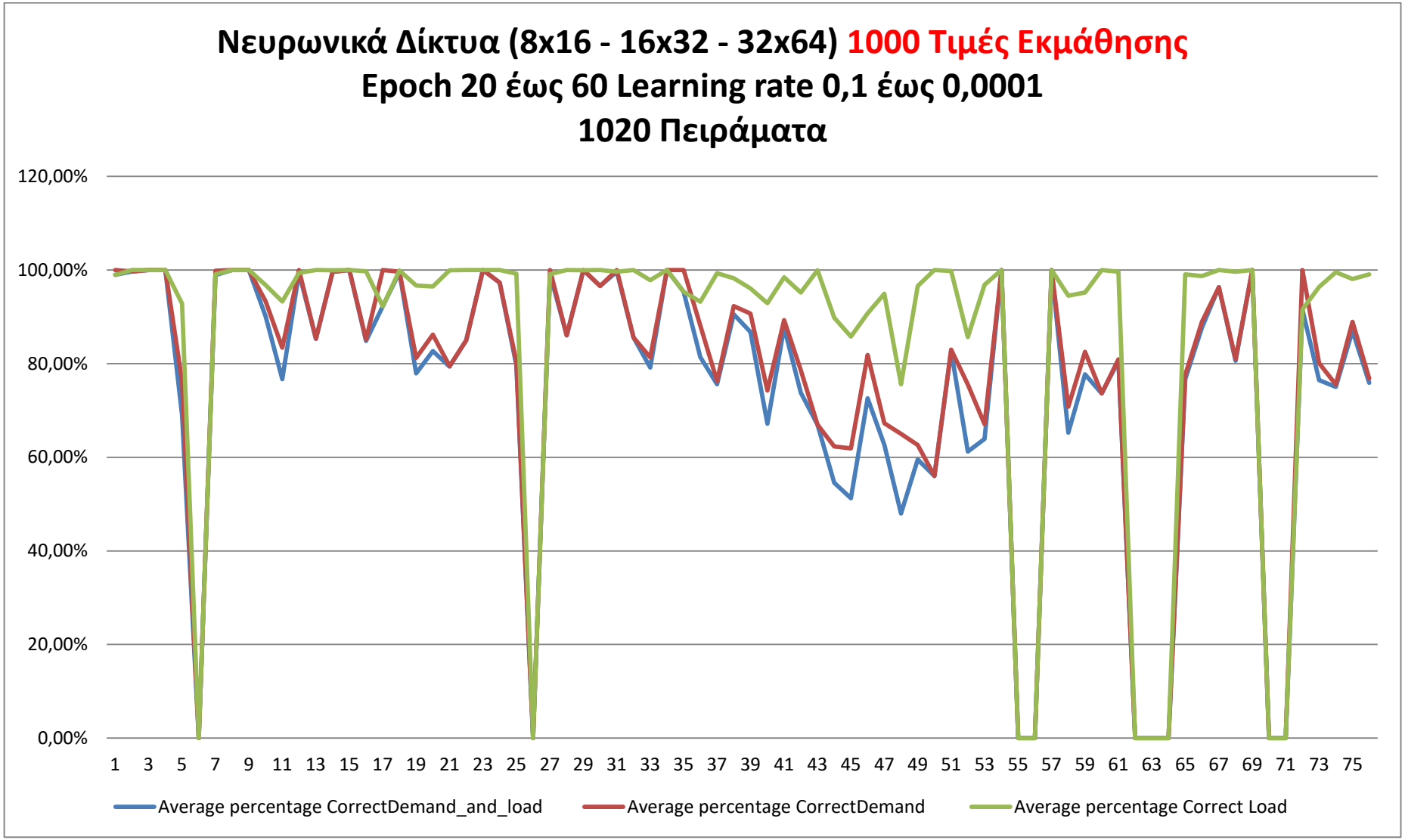
1ο μέρος

Στα πρώτα πειράματα μας χρησιμοποιούμε για τη δημοφιλία ένα τυχαίο πλήθος τιμών μεταξύ 1 και 100, οι οποίες τιμές διοχετεύονται στον αλγόριθμο Trigg, που αφορά τη δημοφιλία (Trend) και παράγονται νέες τιμές. Αυτές οι τιμές κανονικοποιούνται μέσα σε ένα εύρος από 0 έως 1. Οι νέες τιμές διακρινόταν για την μεγάλη ποικιλομορφία αλλά και την μεγάλη έκταση, δηλαδή δεκαδικοί αριθμοί με 8 δεκαδικά ψηφία. Όπως θα αποδειχθεί και παρακάτω αυτό βοηθά στην εκπαίδευση του νευρωνικού μας δικτύου διότι το μοντέλο μας μαθαίνει να διαχειρίζεται τιμές που έχουν μεγάλη λεπτομέρεια. Παράδειγμα, είναι καλύτερη η εκπαίδευση σε ένα πλήθος τιμών που περιέχει αριθμούς που είναι τις μορφής 0,85965365, δηλαδή αποτελούνται από πολλά ψηφία, από ότι σε ένα ίδιο πλήθος τιμών με την μορφή 0,85965 με λιγότερα ψηφία. Με την πρώτη περίπτωση όταν το μοντέλο μας θέλει να αποφασίσει για μία τιμή η οποία είναι οριακή στο ορθό κριτήριο απόφασης, τότε το νευρωνικό μας μοντέλο αποφασίζει σωστά. Εάν έχουμε την ίδια κατάσταση με προηγουμένως και οι τιμές έχουν την δεύτερη μορφή, τότε υπάρχει μεγάλη πιθανότητα σε μία οριακή κατάσταση να μην έχουμε ορθή απόφαση. Για το βάρος του φορτίου εργασιών παράγουμε ένα τυχαίο πλήθος τιμών μεταξύ 1 και 100 και κανονικοποιούμε αυτές τις τιμές σε ένα εύρος από 0 έως 1. Εξαιτίας του γεγονότος ότι πριν την κανονικοποίηση οι τιμές που παραγόταν είχαν τρία ψηφία, αυτό είχε σαν αποτέλεσμα να παραχθούν τιμές με λιγότερα ψηφία για το Load από ότι για το Demand. Σε κάθε δυάδα Demand και Load τοποθετούσαμε την ετικέτα (1) εάν πληρούσαν το ακόλουθο κριτήριο “Demand >0 **και** Load < 50”, σε κάθε άλλη περίπτωση στην εκάστοτε δυάδα (Demand και Load) τοποθετούσαμε την ετικέτα (0). Δημιουργήσαμε τρία νευρωνικά δίκτυα δύο εισόδων και μίας εξόδου που αποτελούταν από δύο κρυφά επίπεδα που είχαν αντίστοιχα 8x16, 16x32 και 32x64 εσωτερικούς νευρώνες. Επίσης χρησιμοποιήσαμε Learning Rate από 0,01 έως 0,0001 και epochs από 20 έως 400 και μέγεθος παρτίδας από 2 έως 20 και επιλέξαμε loss function το μέσο τετραγωνικό σφάλμα “MSE” . Επίσης οι τιμές που

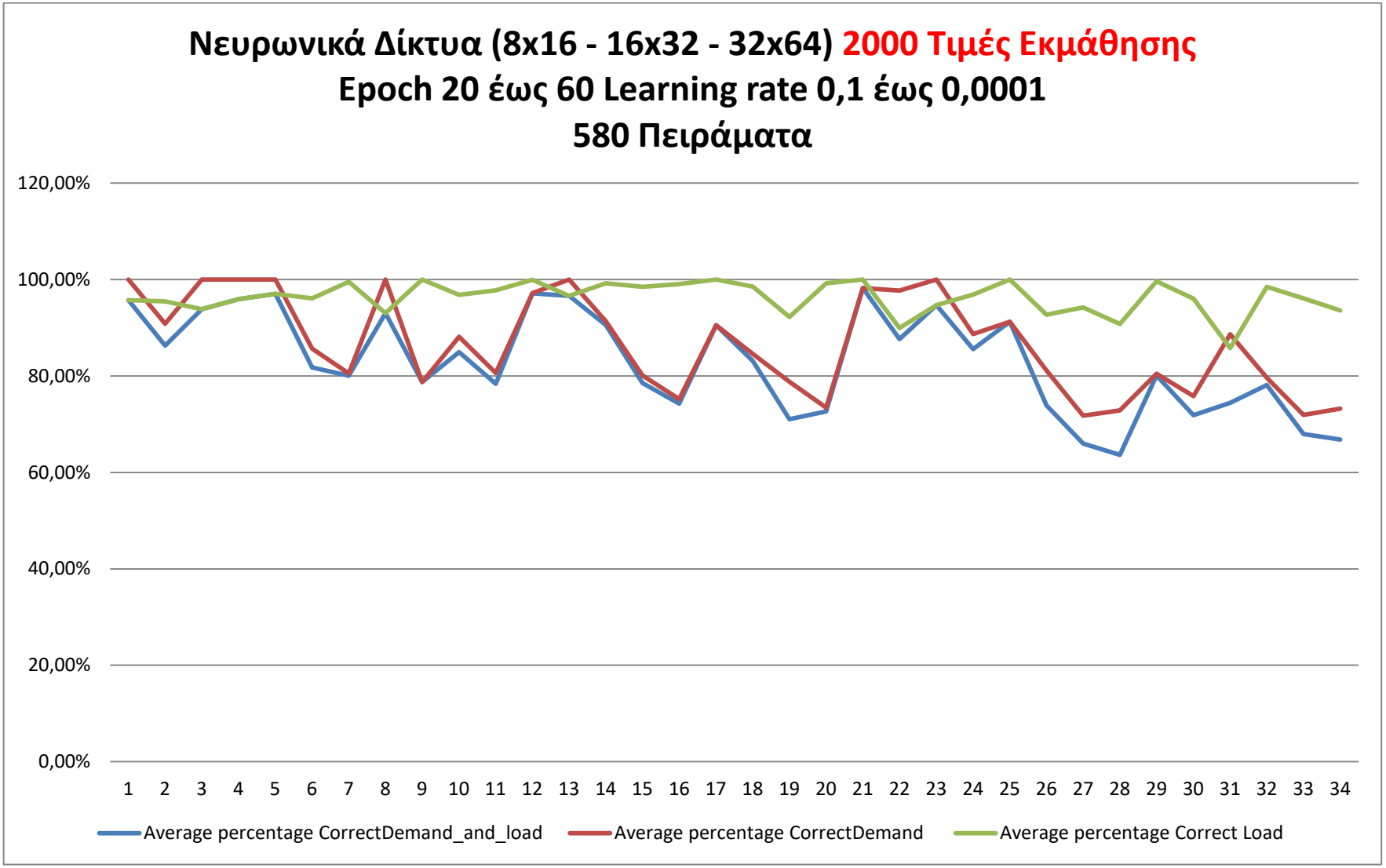
δεχόμασταν ως σωστές, για να δούμε εάν το μοντέλο δουλεύει όπως εμείς θέλουμε, ήταν για το demand μεγαλύτερες από (0,6) και για το φορτίο μικρότερες από (0,5). Αυτό που διαπιστώσαμε είναι ότι όταν το μοντέλο εκπαιδευόταν σε λίγα δεδομένα 500 (εικόνα 17) δεν μπορούσε να γενικεύσει σωστά, όσο ανέβαινε ο αριθμός των δεδομένων εκπαίδευσης 1000 και 2000 τότε το μοντέλο είχε μία αρκετά καλή λειτουργία όπως φαίνεται και στις εικόνες 18 και 19 και σταθεροποιούταν στα 3000 δεδομένα εκπαίδευσης όπως φαίνεται στην εικόνα 20.



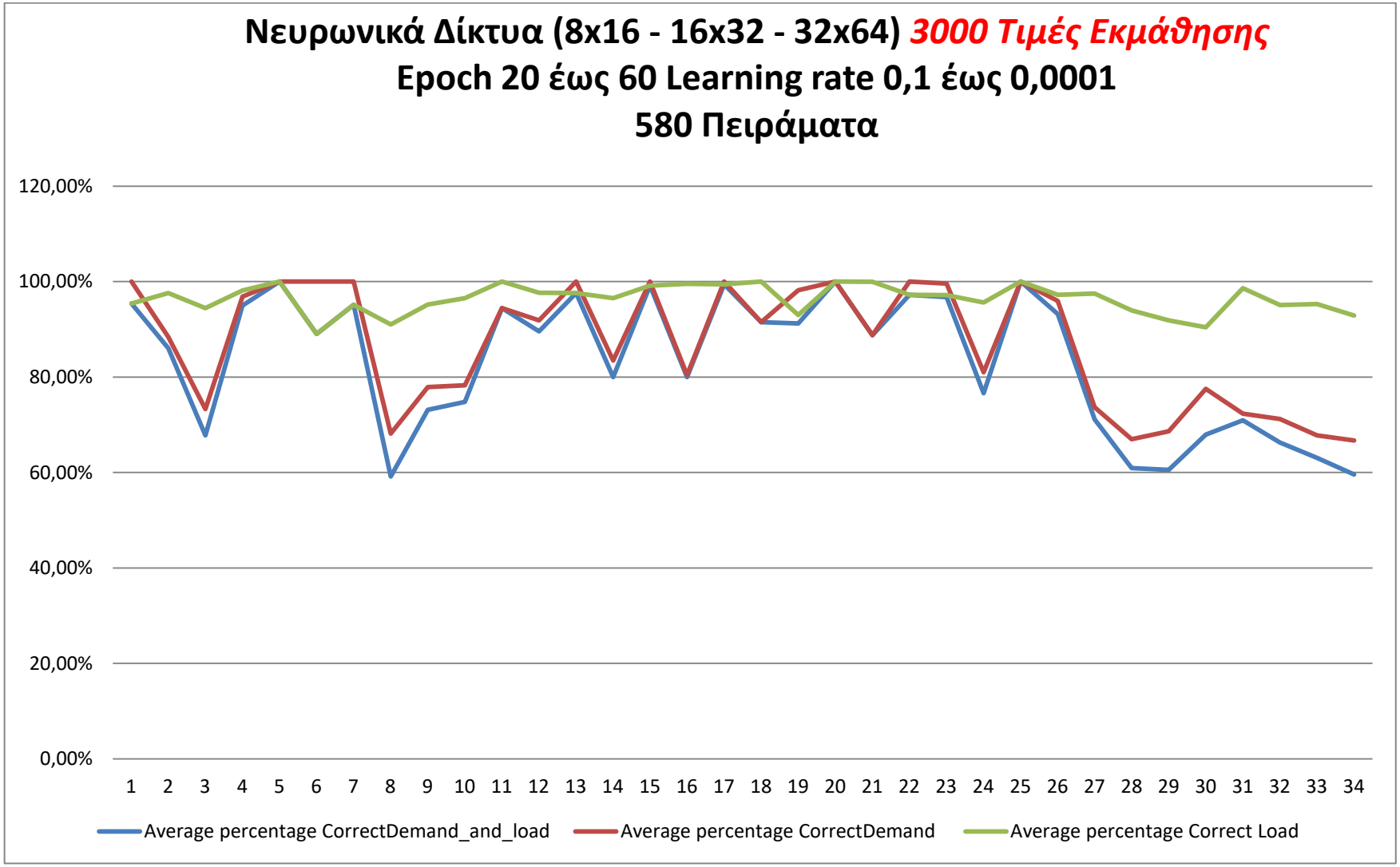
Εικόνα 17 Σωστές αποφάσεις συνολικά και ξεχωριστά για Demand και Load Τιμές



Εικόνα 18 Σωστές αποφάσεις Συνολικά και ξεχωριστά για Demand και Load

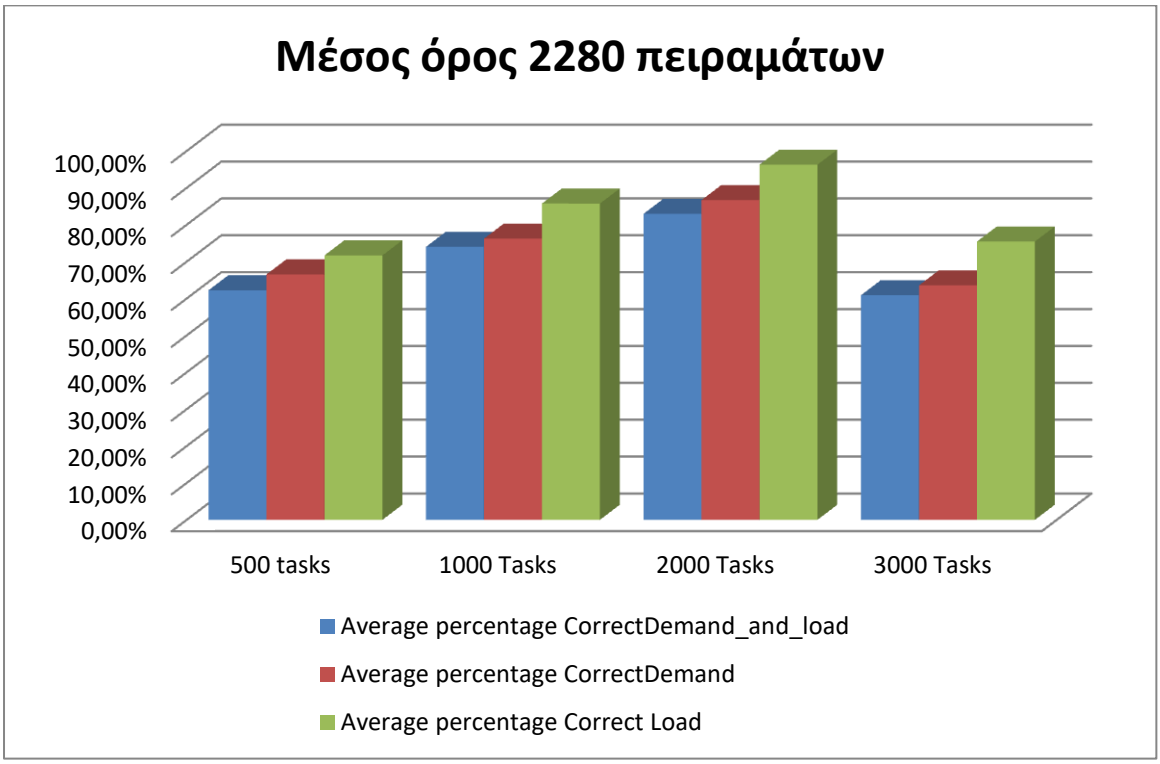


Εικόνα 19 Σωστές αποφάσεις Συνολικά και ξεχωριστά για Demand και Load



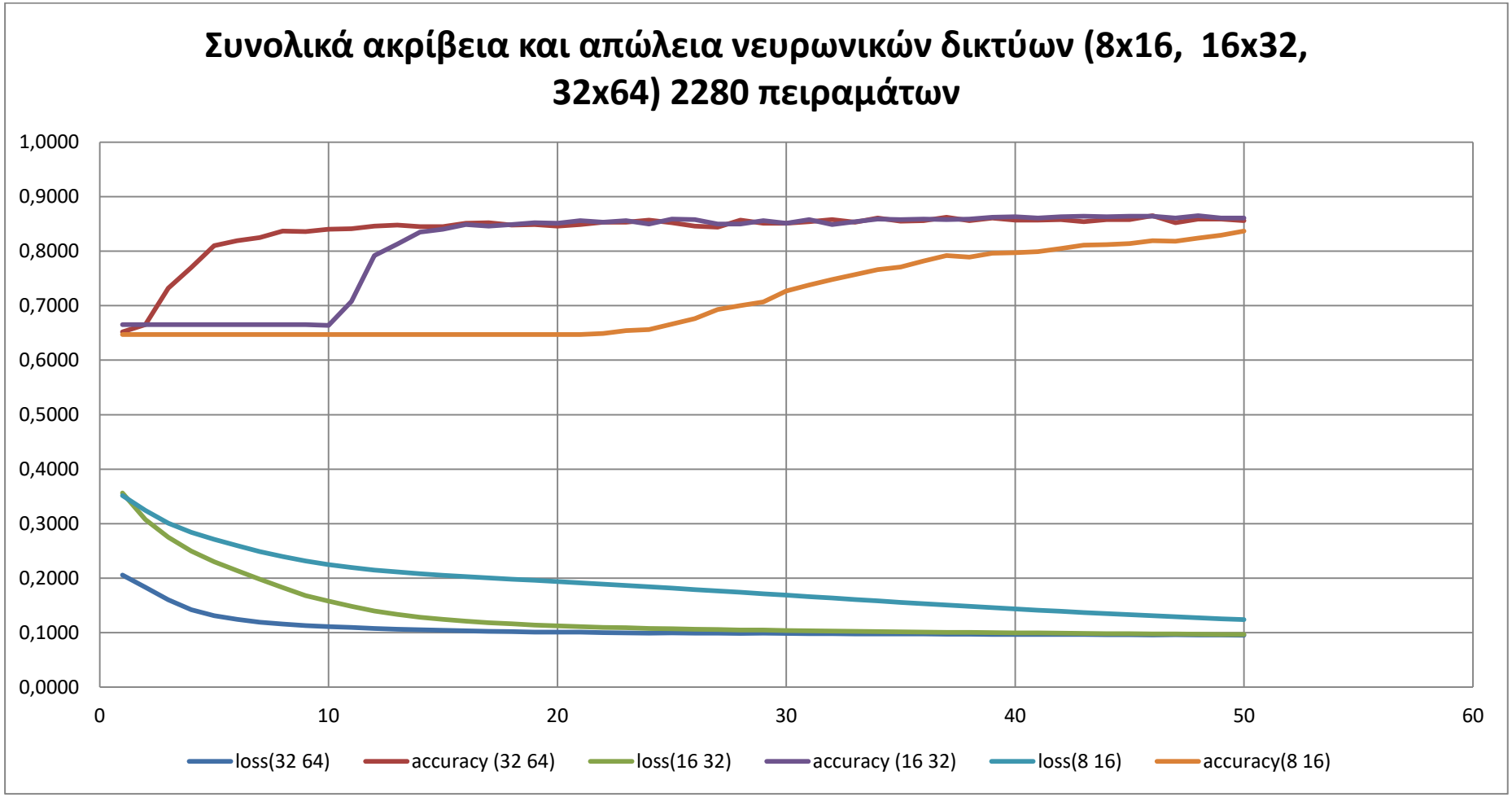
Εικόνα 20 Σωστές αποφάσεις Συνολικά και ξεχωριστά για Demand και Load

Παρά τη βελτιωμένη λειτουργικότητα του μοντέλου, αφού αυξήσαμε το πλήθος των δεδομένων εκπαίδευσης, δεν ικανοποιηθήκαμε από το μέσο όρο των σωστών αποφάσεων. Όπως φαίνεται και στην εικόνα 21 το μέγιστο ποσοστό που πετύχαμε ήταν από 82,94% έως 96% για 2000 τιμές εκπαίδευσης. Εκτός αυτού, παρατηρώντας τις επιμέρους τιμές των πειραμάτων βλέπουμε μεγάλη διακύμανση από 48,01% έως 100%. Αυτό φανερώνει ότι το νευρωνικό μας δίκτυο δεν είναι αξιόπιστο.



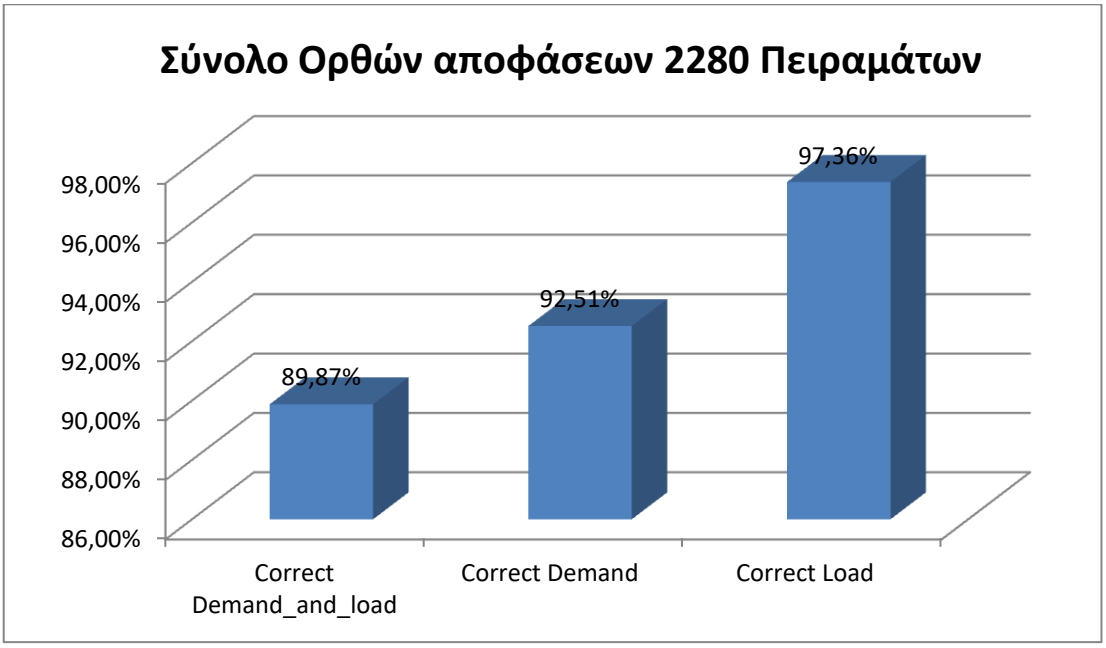
Εικόνα 21 Μέσος όρος 2280 πειραμάτων

Τα παραπάνω αποδεικνύονται στην εικόνα 22 όπου φαίνεται η συνολική ακρίβεια και η συνολική απώλεια της διαδικασίας της εκπαίδευσης. Παρατηρούμε ότι το νευρωνικό δίκτυο με 32x64 νευρώνες στα κρυφά επίπεδα συμπεριφέρεται καλύτερα και πλησιάζει τις βέλτιστες τιμές ενώ χειρότερο είναι το μοντέλο με 8x16 νευρώνες. Επίσης διαπιστώνουμε ότι το καλύτερο μοντέλο επουδενί δεν πλησιάζει τις μέγιστες τιμές ακρίβειας και απώλειας. Να σημειωθεί ότι χρησιμοποιήσαμε παραλλαγές στο epoch, στις παρτίδες δεδομένων, στο βήμα μάθησης στην δομή του νευρωνικού δικτύου. Η απόκλιση από τις μέγιστες τιμές ακρίβειας και απώλειας, έρχεται να συνηγορήσει στην μεγάλη διακύμανση των σωστών αποφάσεων του μοντέλου μας. Επίσης αυτό που παρατηρούμε είναι ότι η απόφαση για το Load είναι ποσοστιαία καλύτερη από το Demand.



Εικόνα 22 Ακρίβεια - Απώλεια Εκπαίδευσης

Από τα παραπάνω κατανοούμε ότι το μοντέλο μας δεν είναι αξιόπιστο για αυτό θα προβούμε σε κάποιες αλλαγές. Αυτό που διαπιστώνουμε από τα παραπάνω πειράματα είναι ότι η απόφαση για το Load είναι πιο αποτελεσματική από ότι για το Demand (εικόνα 23)

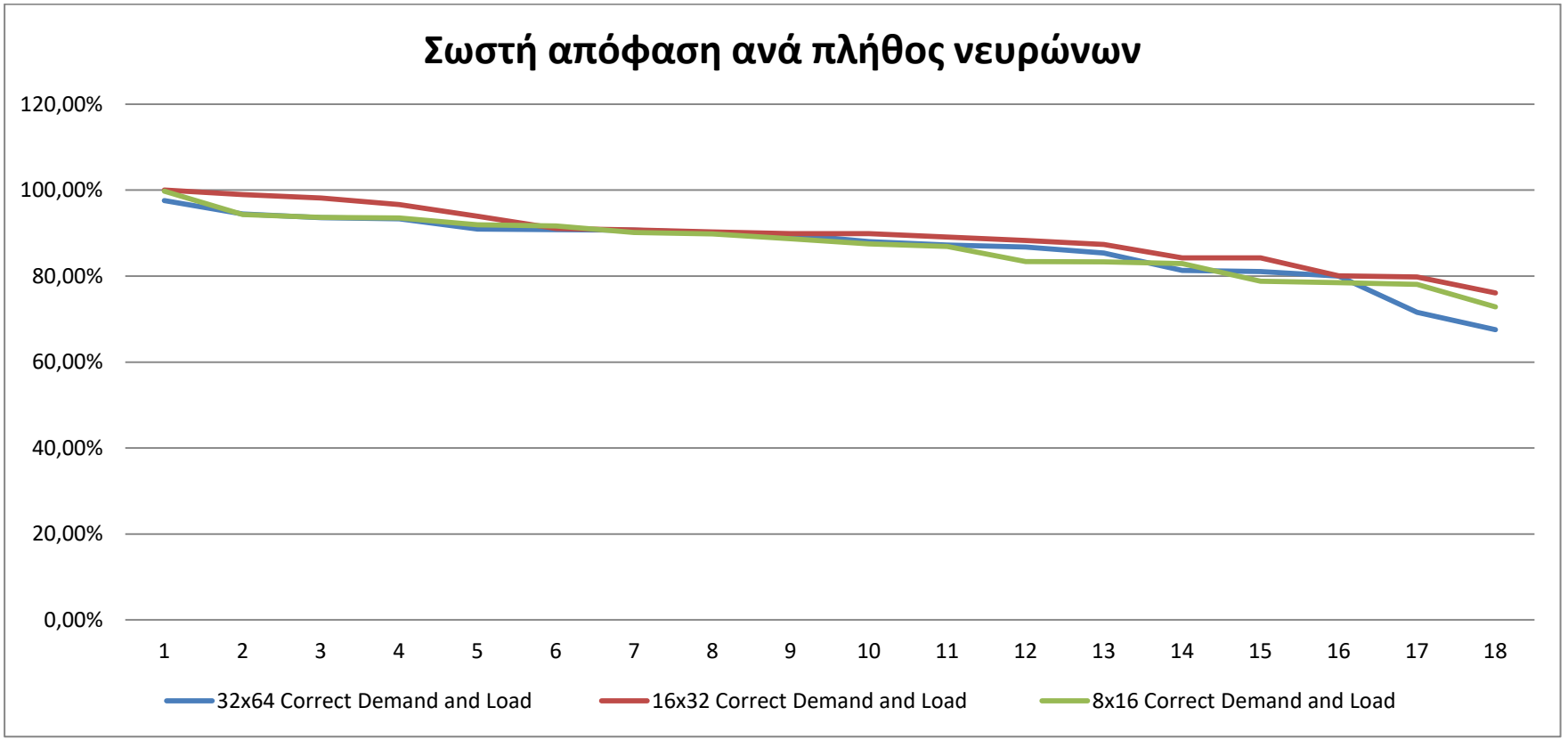


Εικόνα 23 Σύνολο Ορθών αποφάσεων 2280 Πειραμάτων

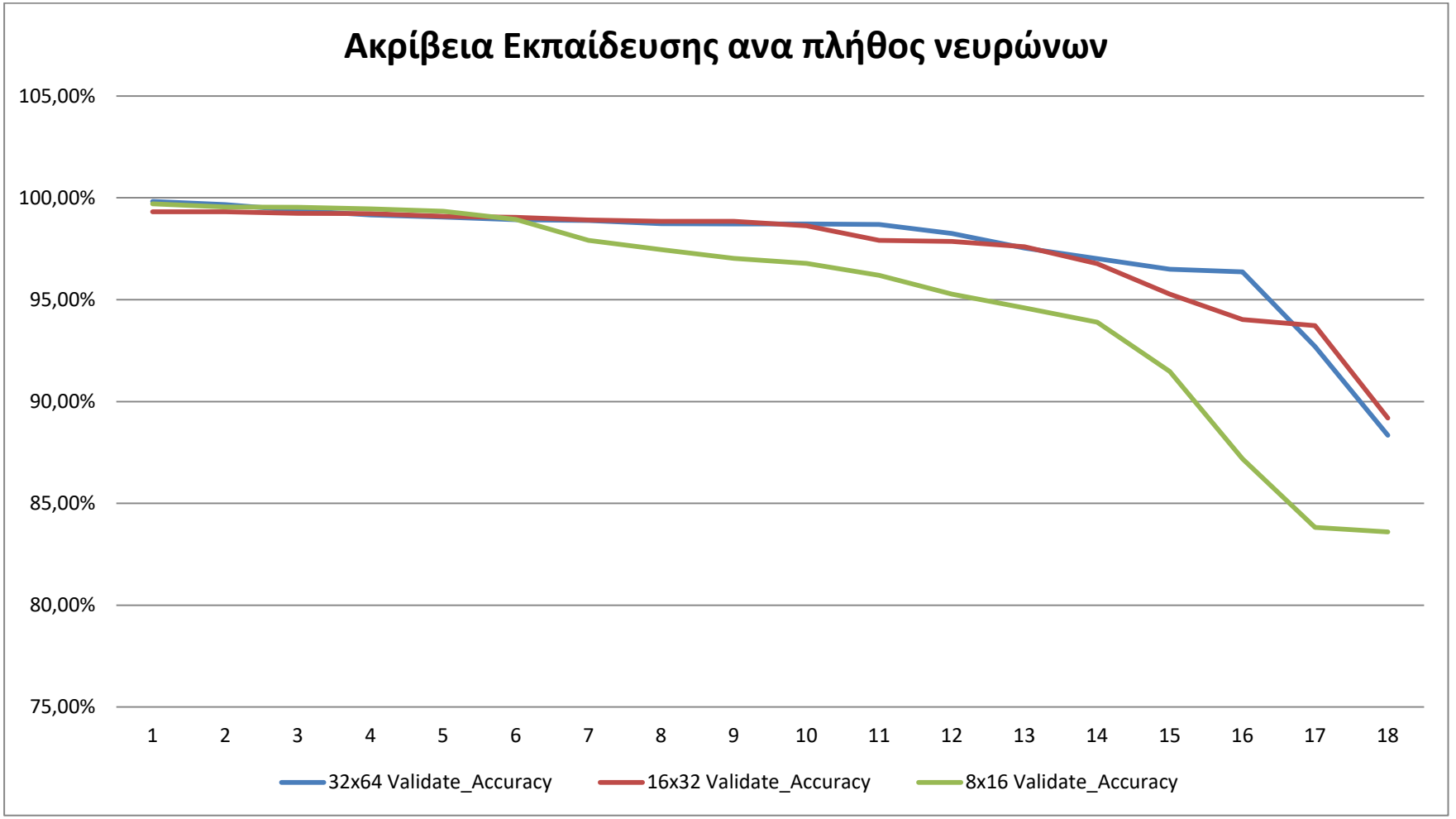
2ο μέρος

Αυτό που παρατηρούμε είναι ότι εκπαιδεύουμε το μέγεθος Demand για μία κλίμακα αριθμών μεταξύ -1 έως 1 και εκπαιδεύουμε το μοντέλο να δέχεται ως ορθή απόφαση τιμές που είναι μεγαλύτερες από το μηδέν. Δηλαδή ζητάμε ως ορθές τιμές στο μέσο μιας κλίμακας αριθμών που στην περίπτωση μας, το μέσο της κλίμακας -1 έως 1 είναι το μηδέν. Έπειτα, εμείς δοκιμάζουμε το μοντέλο σε τιμές μιας κλίμακας από 0 έως 1 για το Demand και ζητάμε ορθή τιμή η οποία να είναι μεγαλύτερη από το 0,6. Το 0,6 όμως δεν είναι το μέσο της κλίμακας 0 έως 1, το μέσο της παραπάνω κλίμακας είναι το 0,5. Αυτό είχε σαν αποτέλεσμα να έχουμε λιγότερη ακρίβεια στην πρόβλεψη του μοντέλου μας. Επίσης χρησιμοποιούσαμε loss function την MSE. Διαπιστώσαμε όμως ότι για προβλήματα που θέλουμε απάντηση 0 ή 1 λειτουργεί καλύτερα η Binary crossentropy. Οπότε τα επόμενα πειράματα έγιναν με loss fuction, τη Binary crossentropy και δεχόμαστε ως ορθή απόφαση για το Demand τιμές οι οποίες είναι μεγαλύτερες του 0,5. Τελικά με τις παραπάνω αλλαγές και έπειτα από 550 πειράματα οδηγηθήκαμε στα παρακάτω συμπεράσματα και γραφικές.

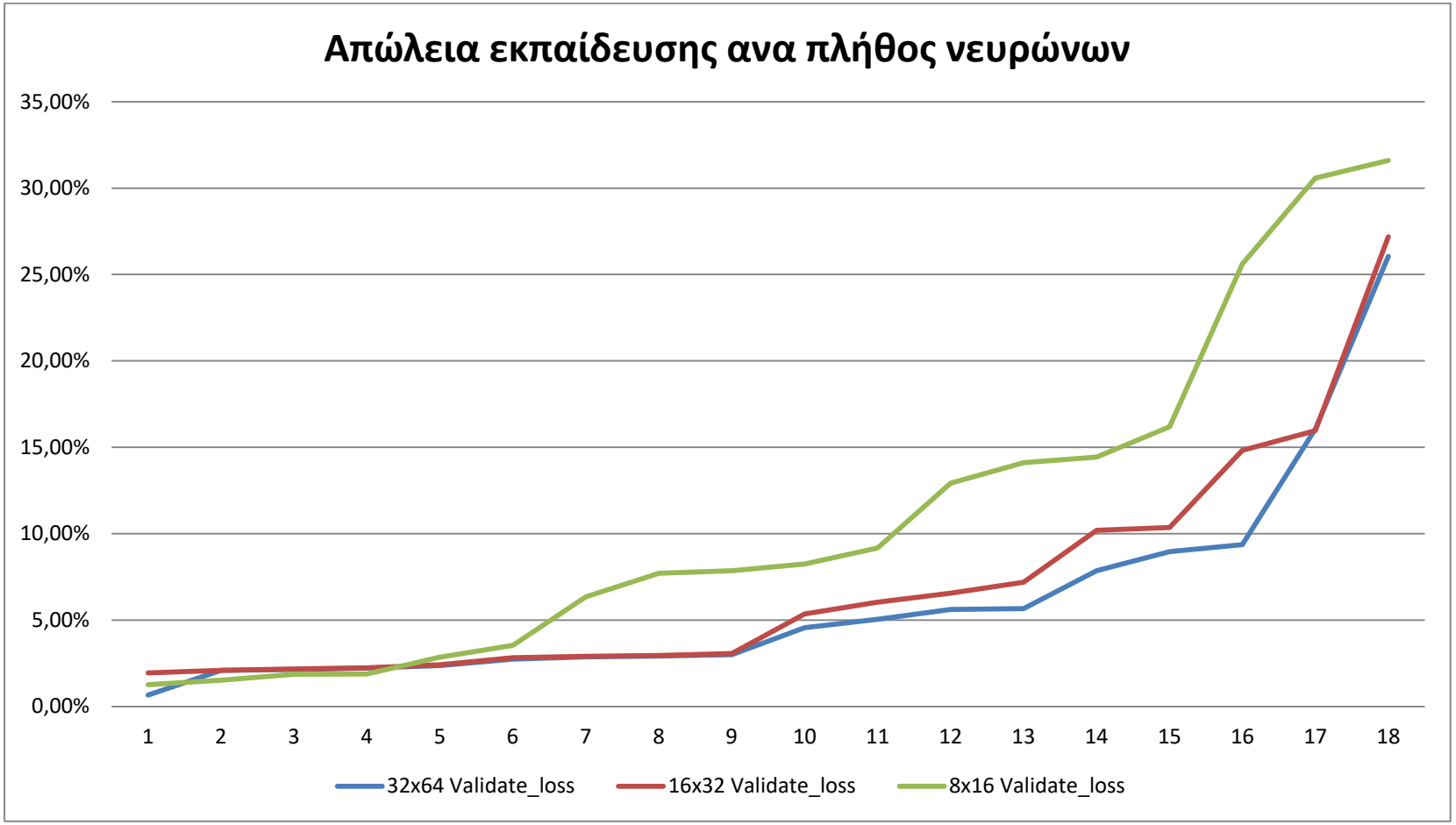
Ας δούμε την απόδοση τριών νευρωνικών δικτύων δύο κρυφών επιπέδων και εσωτερικά 8x16,16x32 και 32x64 νευρώνες. Χρησιμοποιήσαμε epoch από 60 έως 300, βήμα μάθησης από 0,1 έως 0,0001, βελτιστοποίηση ADAM, τιμές εκπαίδευσης από 1000 έως 2000.



Εικόνα 24 Σωστή απόφαση ανά πλήθος νευρώνων



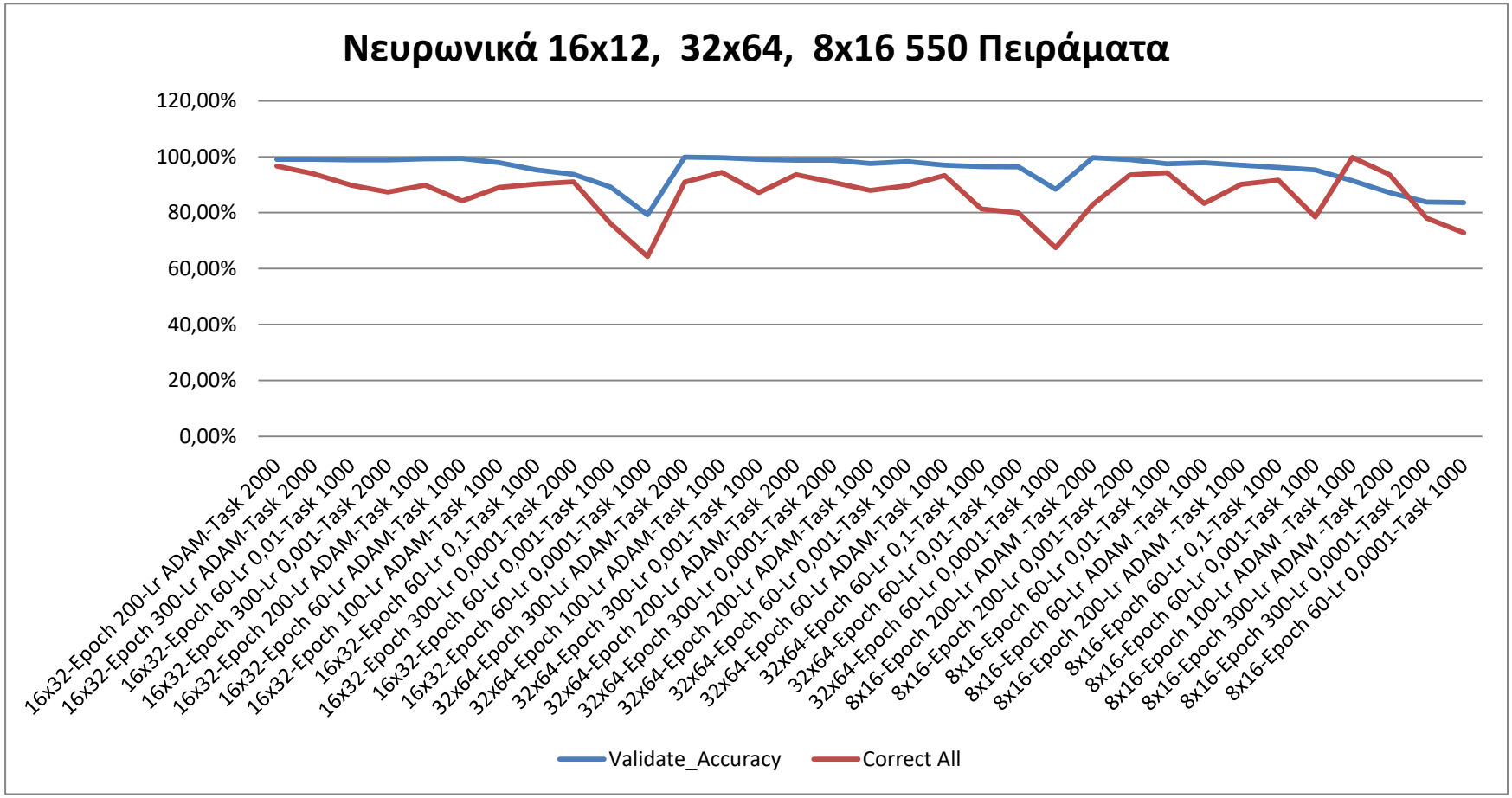
Εικόνα 25 Ακρίβεια Εκπαίδευσης ανα πλήθος νευρώνων



Εικόνα 26 Απώλεια εκπαίδευσης ανα πλήθος νευρώνων

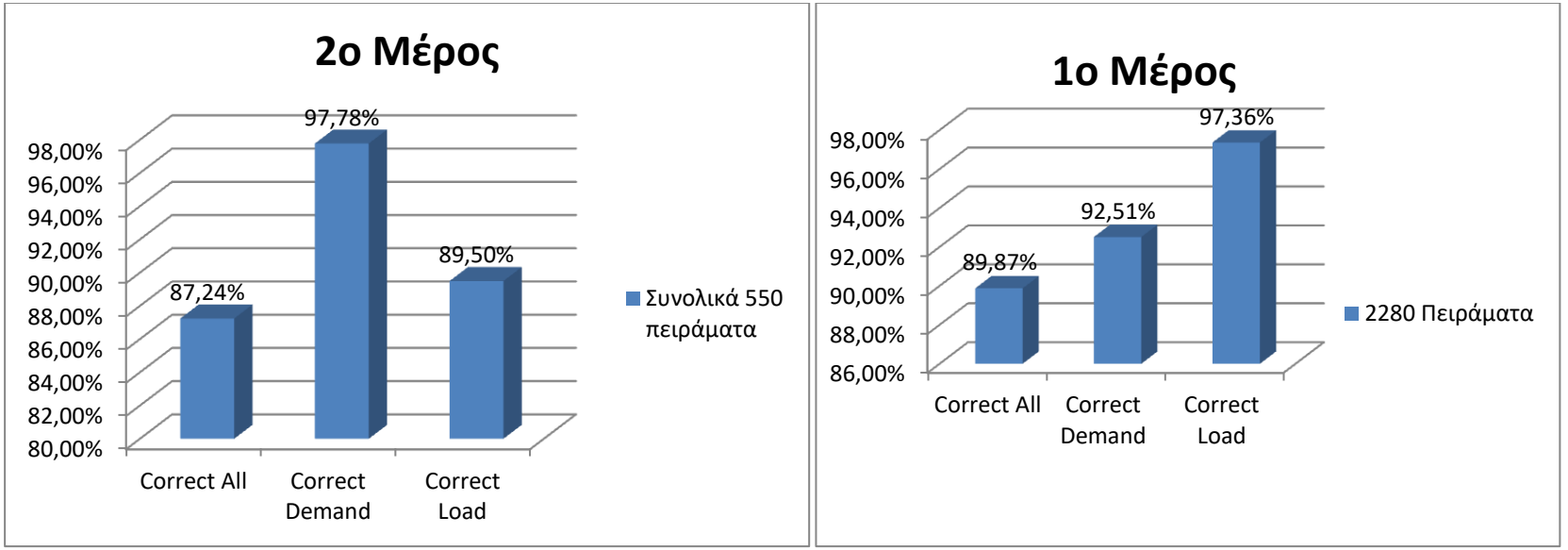
Από την εικόνα 24 εξάγουμε το συμπέρασμα ότι το καλύτερο νευρωνικό είναι αυτό που έχει 16x32 νευρώνες αφού το πλήθος των σωστών αποφάσεων πλησιάζει στο 100%. Παρατηρώντας όμως την ακρίβεια εκπαίδευσης και την απώλεια από τις εικόνες 25 και 26, βλέπουμε ότι το νευρωνικό με 32x64 νευρώνες μαθαίνει καλύτερα. Αυτό το παράδοξο μας δείχνει σημάδια

overfitting. Δηλαδή το μοντέλο μας απομνημονεύει τις τιμές και δεν είναι καλό σε νέα άγνωστα δεδομένα. Αυτό θα κληθούμε στα επόμενα πειράματα να το διορθώσουμε. επίσης, παρατηρούμε ότι όσο μειώνετε η ακρίβεια της εκπαίδευσης και αυξάνετε η απώλεια (εικόνα 25 και 26) μειώνεται και το πλήθος τω σωστών αποφάσεων (εικόνα 24) κάτι το αναμενόμενο και λογικό.



Εικόνα 27 Νευρωνικά 16x12, 32x64, 8x16

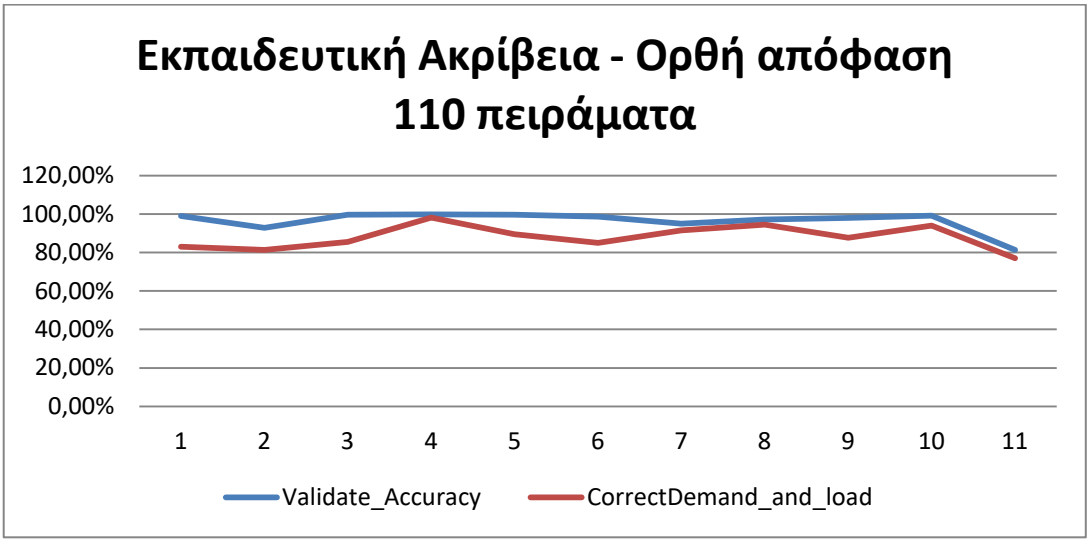
Συνολικά στην εικόνα 27 παρατηρούμε την ακρίβεια της εκπαίδευσης να ακολουθεί το πλήθος τω σωστών αποφάσεων, εκτός από το τέλος της γραφικής παράστασης οπού εκεί η ακρίβεια εκπαίδευσης μειώνεται και θα περίμενε κάνεις να έχουμε μείωση και στο πλήθος των ορθών αποφάσεων. Παρόλα αυτά βλέπουμε μία αύξηση στο πλήθος των σωστών αποφάσεων, το οποίο συνηγορεί στο ότι πρέπει να βελτιώσουμε το μοντέλο μας. Εξάλλου, ενώ στο πρώτο πειραματικό μέρος η ορθή απόφαση που αφορά την μεταβλητή Load ήταν καλύτερη, αφού αλλάξαμε τον έλεγχο της μεταβλητής Demand από το 0,6 στο 0,5, διαπιστώνουμε από τα γραφήματα της εικόνας 28, ότι η ορθή απόφαση της μεταβλητής Demand έχει αυξήσει τα ποσοστά της.



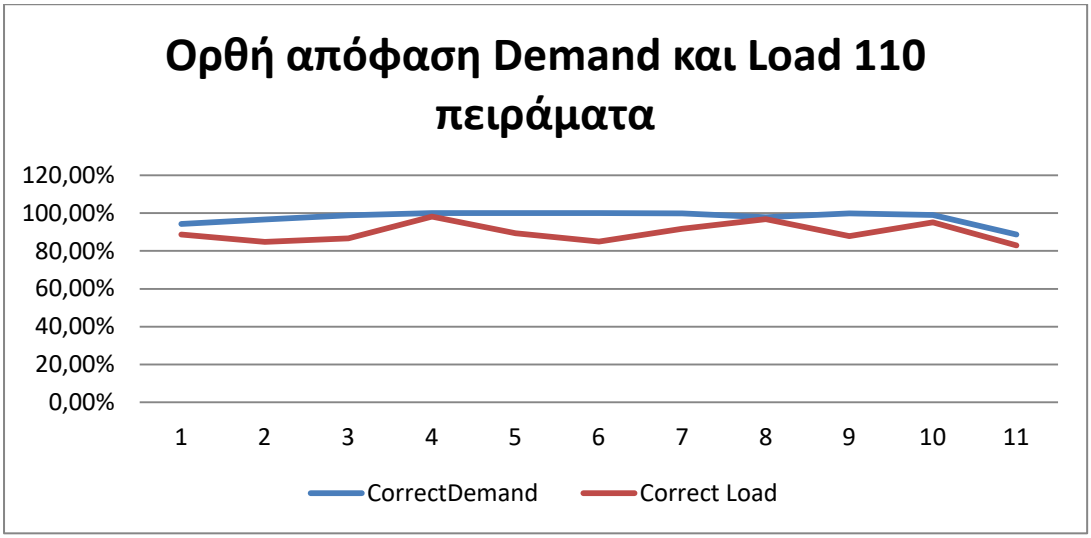
Εικόνα 28 Σύγκριση Σωστών αποφάσεων ανά πειραματικό μέρος

3ο μέρος

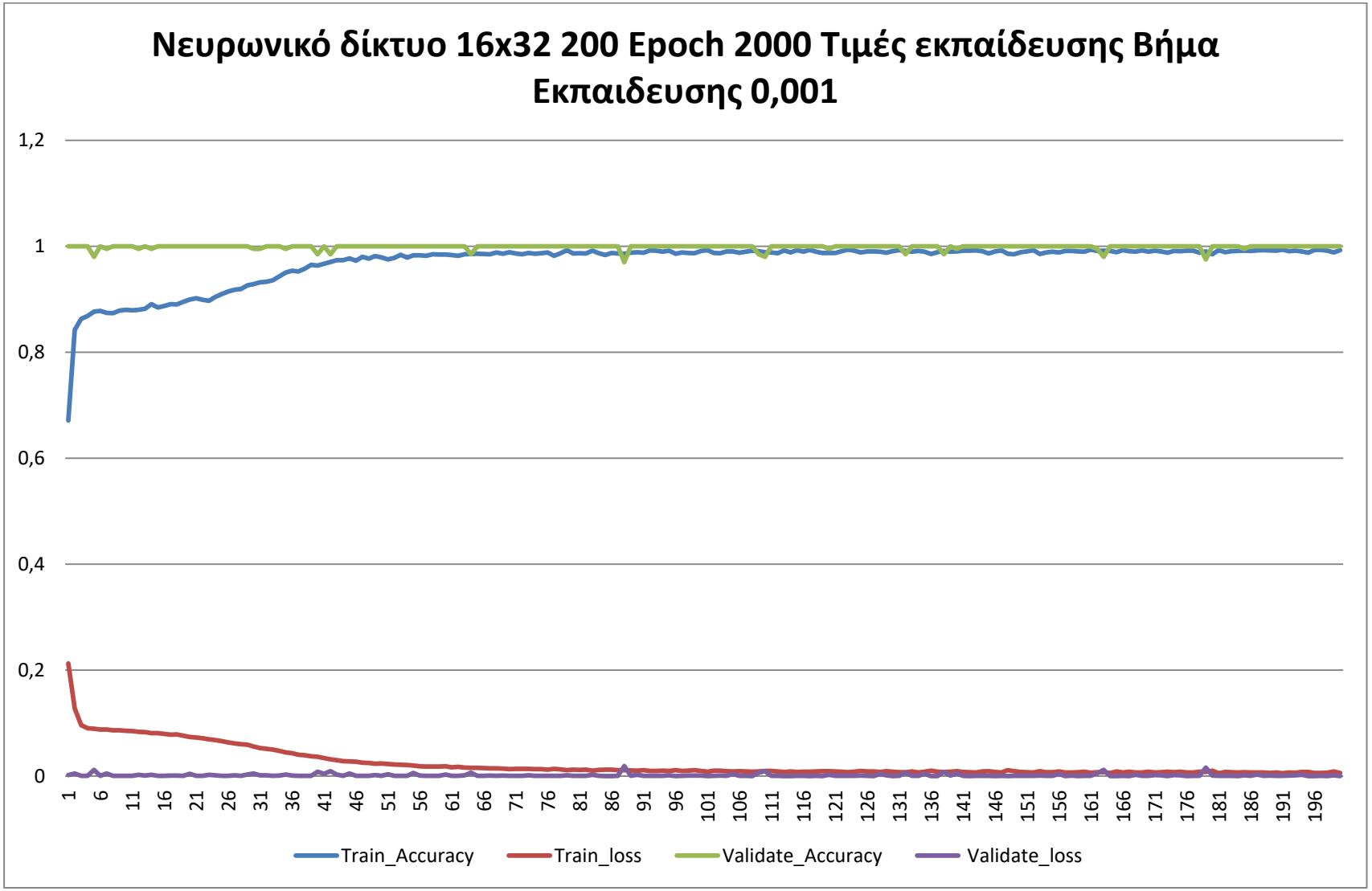
Για να βελτιώσουμε την απόδοση του δικτύου μας πειραματιζόμαστε σε ένα νευρωνικό δίκτυο δύο εισόδων και δύο κρυφών επιπέδων με 16x32 εσωτερικούς νευρώνες σε κάθε επίπεδο αντίστοιχα και μία τελική έξοδο. Αλλάζουμε την loss function από Binary crossentropy στη συνάρτηση του μέσου τετραγωνικού σφάλματος την MSE. Διεξαγάγουμε 110 πειράματα με epoch που κυμαίνεται από 60 έως 300, βελτιστοποίηση με ADAM και βήμα μάθησης από 0,1 έως 0,0001. Το μοντέλο μας εκπαιδεύτηκε από 1000 έως 2000 τιμές. Στην εικόνα 29 φαίνεται η ακρίβεια του σε συνδυασμό με την ορθή απόφαση. Ενώ στην εικόνα 30 παρατηρούμε ότι πάλι το Demand έχει καλύτερη απόδοση από το Load.



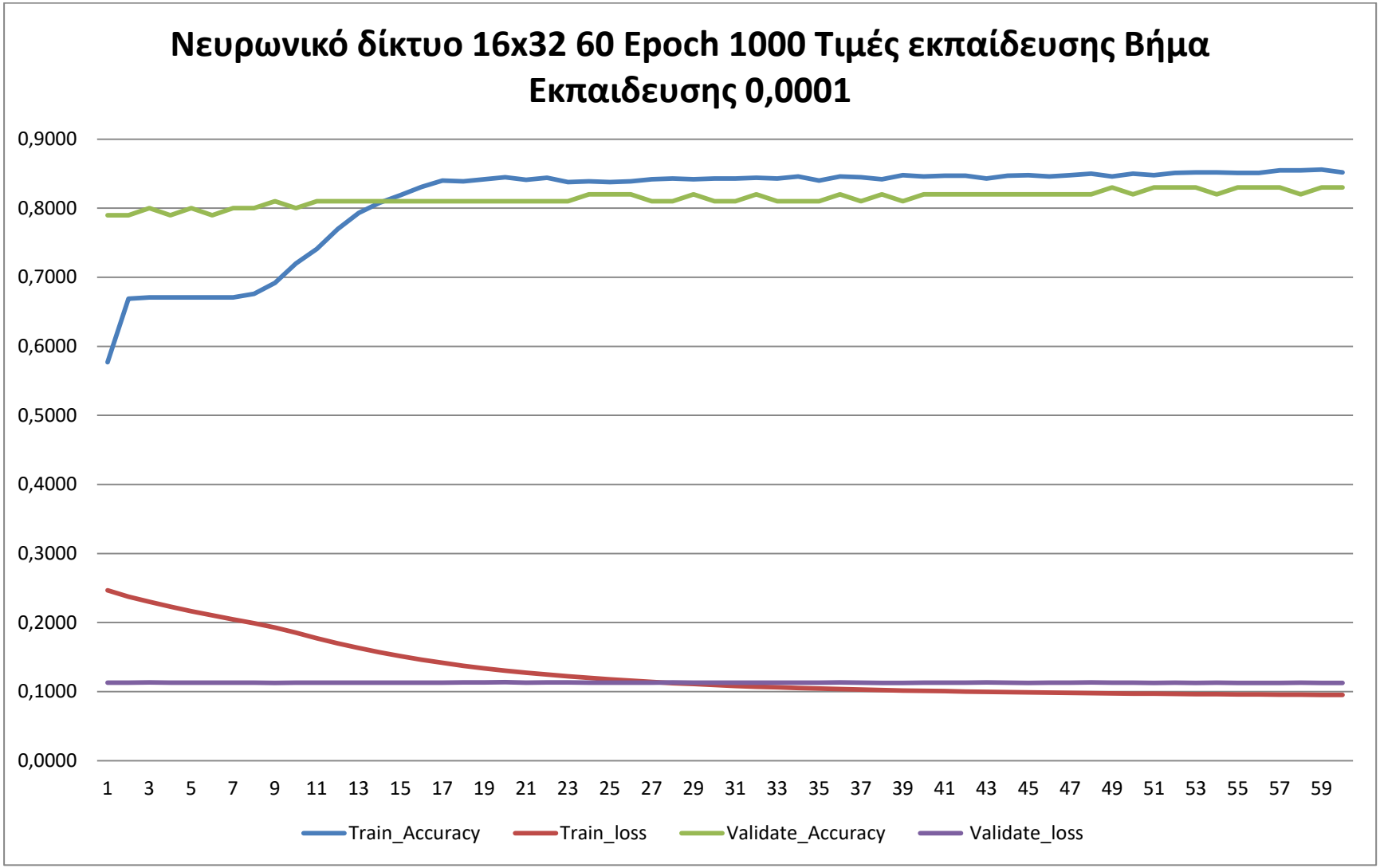
Εικόνα 29 Εκπαιδευτική Ακρίβεια



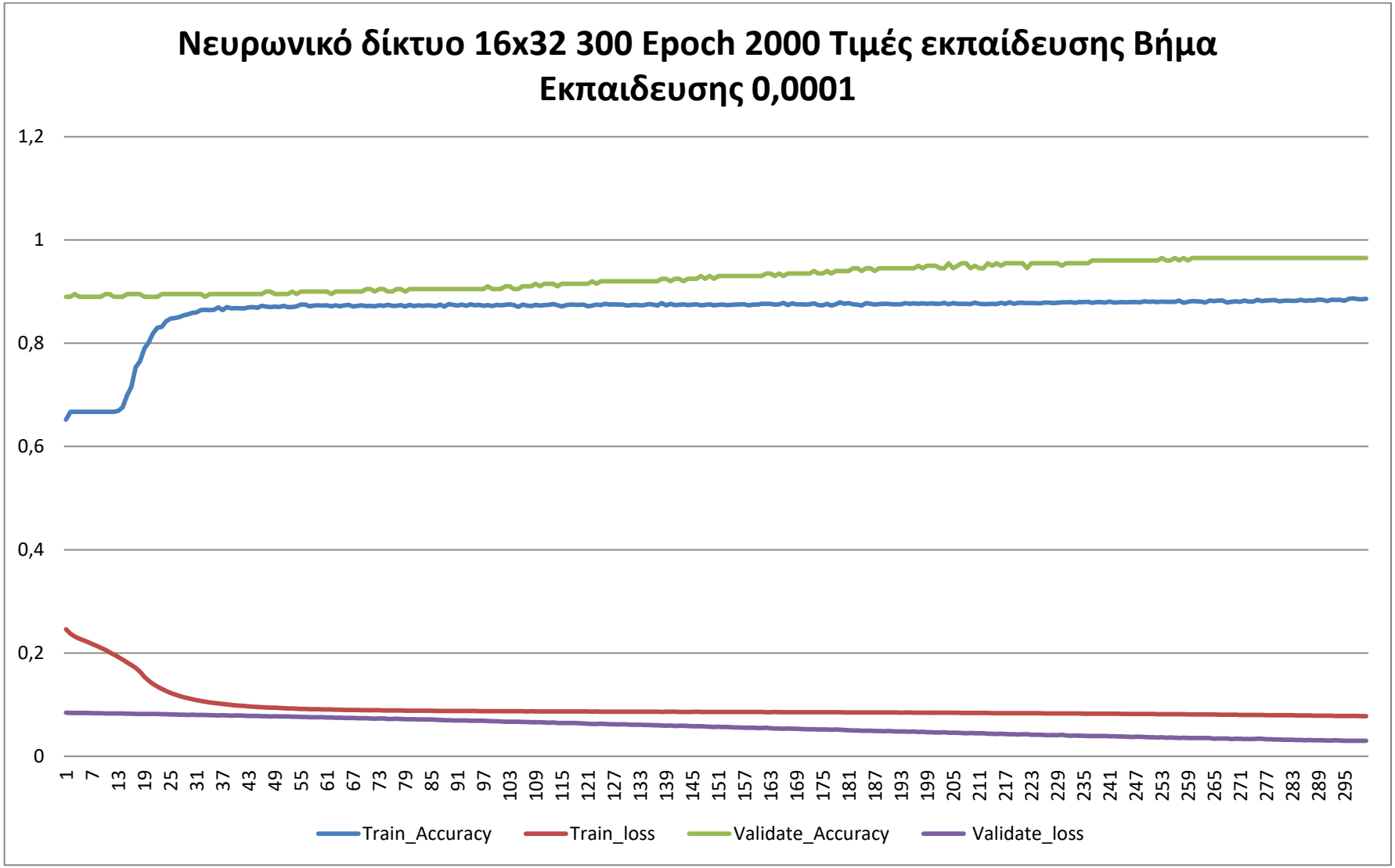
Εικόνα 30 Ορθή απόφαση Demand και Load



Εικόνα 31 Νευρωνικό δίκτυο 16x32 Epoch 2000 Τιμές

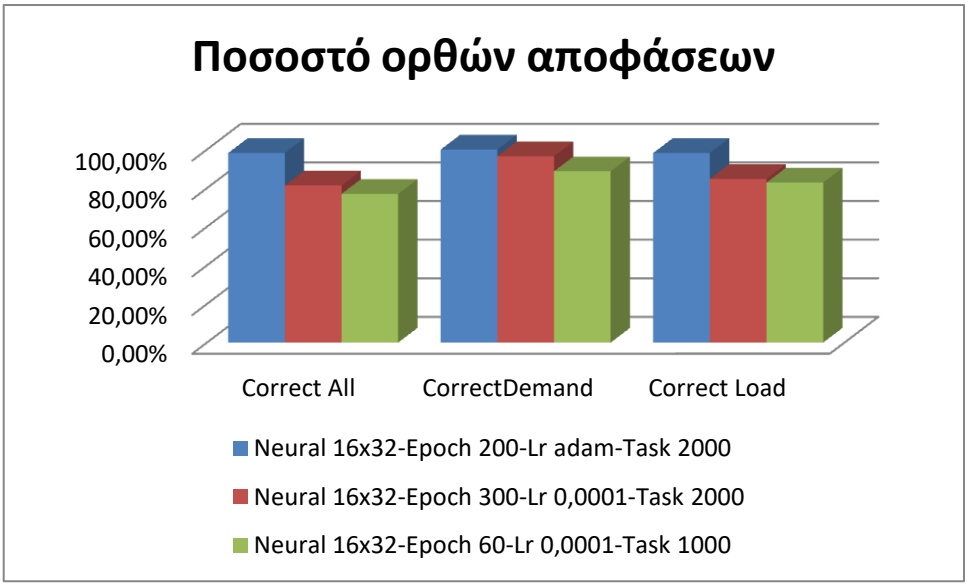


Εικόνα 32 Νευρωνικό δίκτυο 16x32 60 Epoch 1000Τιμές



Εικόνα 33 Νευρωνικό δίκτυο 16x32 300 Epoch 2000 Τιμές

Οι εικόνες 31 και 32 περιέχουν το καλύτερο νευρωνικό δίκτυο και το χειρότερο αντίστοιχα. Ενώ στο γράφημα της εικόνας 33 περιγράφεται ένα μοντέλο με epoch 300. Στο γράφημα της εικόνας 31 παρατηρούμε ότι η ακρίβεια εκπαίδευσης και η απώλεια, καθώς επίσης και η επικύρωση φτάνουν στα μέγιστα βέλτιστα όρια. Ενώ στο γράφημα της εικόνας 32 παρατηρούμε ότι το μοντέλο αποκλίνει αρκετά από τις βέλτιστες τιμές. Τα δεδομένα του γραφήματος 33 προσιδιάζουν με αυτά του χειρότερου σε απόδοση μοντέλου παρόλο που έχουμε ένα πολύ μεγάλο epoch. Τα παραπάνω αποτυπώνονται και στα ποσοστά ορθής απόφασης όπως είναι ορατά στην εικόνα 34 για τα τρία μοντέλα.



Εικόνα 34 – Ποσοστό ορθών αποφάσεων 110 πειράματα

Συμπεραίνουμε ότι περισσότερα epochs και περισσότερες τιμές εκπαίδευσης μας δίνουν αποτελέσματα πιο κοντά στις επιθυμητές τιμές. Παρόλα αυτά στο γράφημα της εικόνας 33 ενώ έχουμε epoch 300 δεν έχουμε τα επιθυμητά αποτελέσματα όπως αυτά εμφανίζονται στην εικόνα 34. Αυτό το περιμέναμε διότι όπως βλέπουμε στο γράφημα της εικόνας 33 η εκπαίδευση και η απώλεια δεν πλησιάζουν τις μέγιστες τιμές. Η προηγούμενη παρατήρηση μας οδηγεί στο συμπέρασμα ότι το μοντέλο μας δεν λειτουργεί αξιόπιστα. Με άλλα λόγια θα συνεχίσουμε στο επόμενο μέρος να εργαζόμαστε για τη δημιουργία ενός πιο αξιόπιστου συστήματος.

4ο μέρος

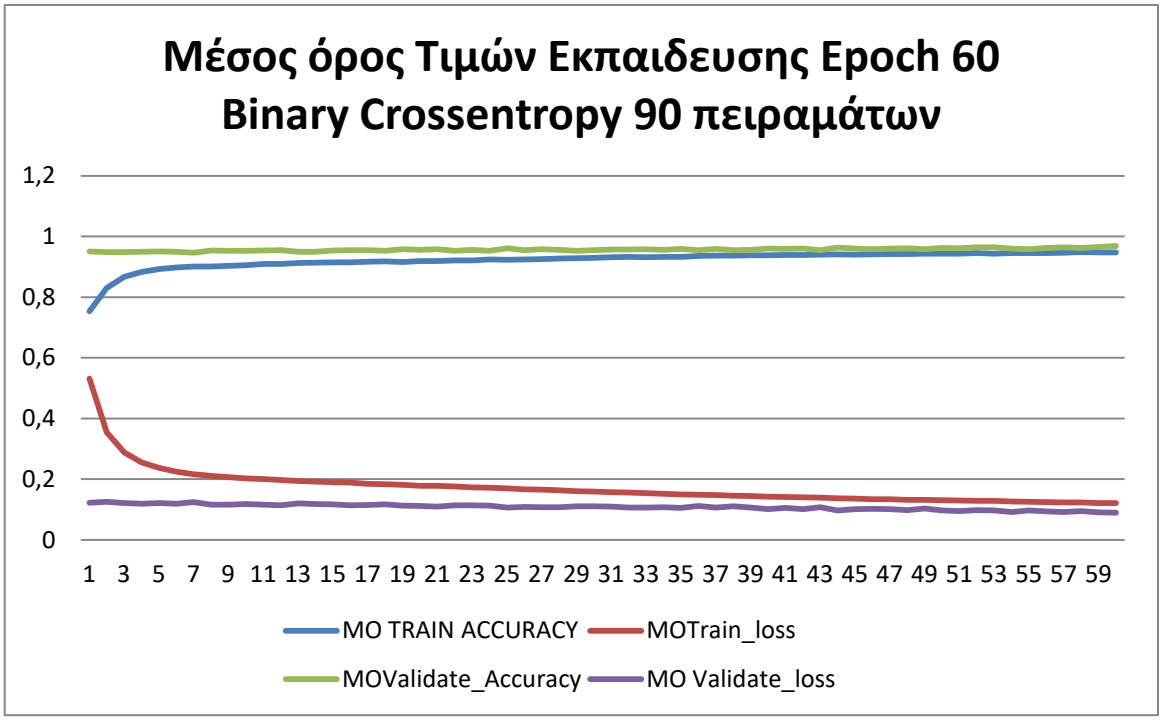
Ενώ έχουμε βελτιώσει αρκετά καλά την απόδοση του δικτύου μας, προσπαθούμε να πετύχουμε μία πιο σταθερή διακύμανση σωστών αποφάσεων προς το 100%. Θέλουμε οι τιμές των ορθών αποφάσεων να μην πέφτουν κάτω από το 90%. Αυτό που παρατηρούμε είναι ότι ενώ στο πρώτο μέρος οι τιμές για το Demand ήταν χειρότερες από ότι του Load, από το δεύτερο μέρος και έπειτα η κατάσταση αντιστράφηκε με κάποιες παρεμβάσεις που κάναμε. Για να βελτιώσουμε το παραπάνω πρόβλημα αλλάζουμε την κλίμακα των αριθμών με την οποία εκπαιδεύεται το νευρωνικό μας δίκτυο. Ενώ είχαμε επιλέξει να εκπαιδεύεται το Demand και το Load μεταξύ των τιμών 1 – 100, πρωτίστως επιλέγουμε το Demand να εκπαιδεύεται μεταξύ 0-100 και το Load μεταξύ 0-10000. Δεν αλλάζουμε πολύ την κλίμακα με την οποία εκπαιδεύεται το Demand, διότι βλέπουμε ότι έχουμε αρκετά ικανοποιητικά αποτελέσματα όπως φαίνεται και στον πίνακα 1. Το δεύτερο βήμα που κάνουμε είναι να αλλάξουμε την εκπαίδευση του Load ριζικά, έτσι τοποθετούμε την ετικέτα (1) όταν έχουμε Demand>0.5 and Load<5000. Στην ουσία η φιλοσοφία, του πιο θα θεωρεί το μοντέλο μας ορθή απόφαση (Demand>0.5 and Load<5000) δεν έχει αλλάξει, διότι και εδώ θέλουμε το μέσο της κλίμακας, απλώς αυτό που κερδίσαμε στην προκειμένη περίπτωση είναι η ποικιλομορφία των δεδομένων και επιπλέον αυτήν η ποικιλομορφία επιτείνεται από το γεγονός ότι οι κλίμακες εκπαίδευσης ξεκινούν από το (0) και όχι από το (1), όπως ξεκινούσαν στα προηγούμενα μέρη, με αποτέλεσμα αυτό να προσδώσει μια επιπλέον ακρίβεια στο μοντέλο μας.

CorrectDemand	Correct Load
88,72%	82,87%
96,56%	84,83%
100,00%	85,01%
98,82%	**86,70%**
99,89%	87,84%
94,29%	88,64%
100,00%	89,43%
99,76%	91,75%
98,98%	95,02%
97,79%	96,75%
100,00%	**98,22%**

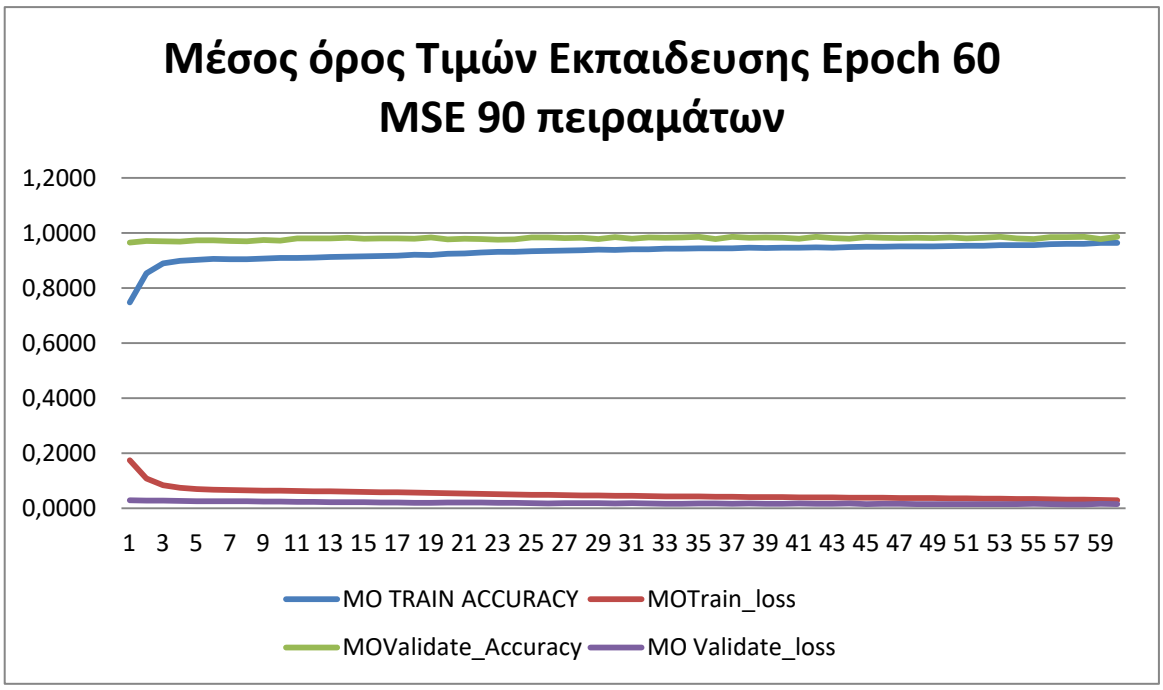
Πίνακας 1 Ποσοστά Ορθών αποφάσεων Demand και Load

Στα πειράματα που θα ακολουθήσουν χρησιμοποιούμε loss function την “MSE” και την “Binary crossentropy”, επίσης εφαρμόσαμε epoch από 60 έως 100, τιμές

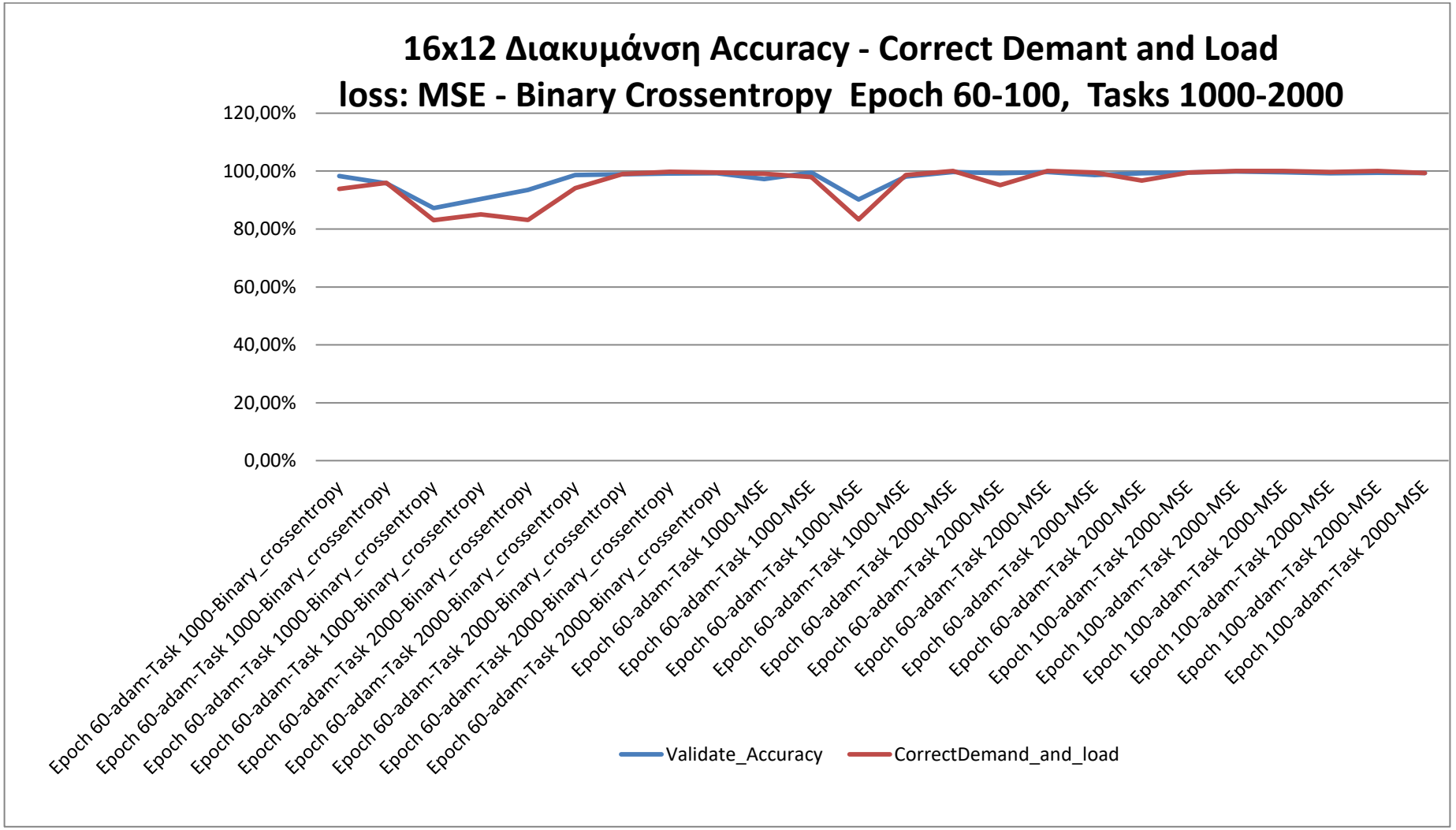
εκπαίδευσης από 1000 έως 2000 και βελτιστοποίηση Adam με βήμα εκπαίδευσης η εξορισμού τιμή του Adam δηλαδή 0,001. Στα παρακάτω πειράματα θα διαχωρίσουμε τις γραφικές ανά loss function για να επιλέξουμε την καλύτερη λύση. Διεξήχθησαν 250 πειράματα.



Εικόνα 35 - Μέσος όρος Τιμών Εκπαίδευσης Binary Crossentropy

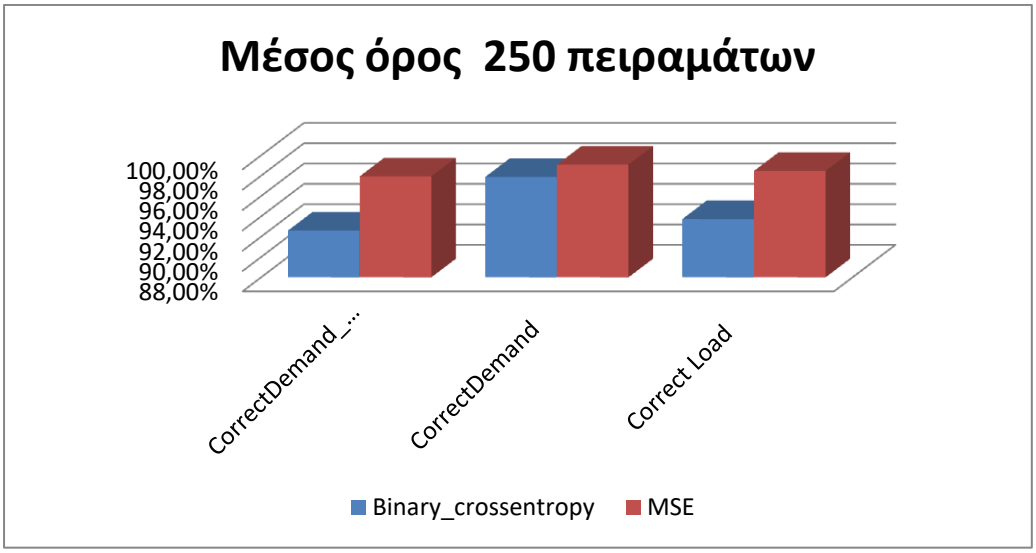


Εικόνα 36 - Μέσος όρος Τιμών Εκπαίδευσης MSE



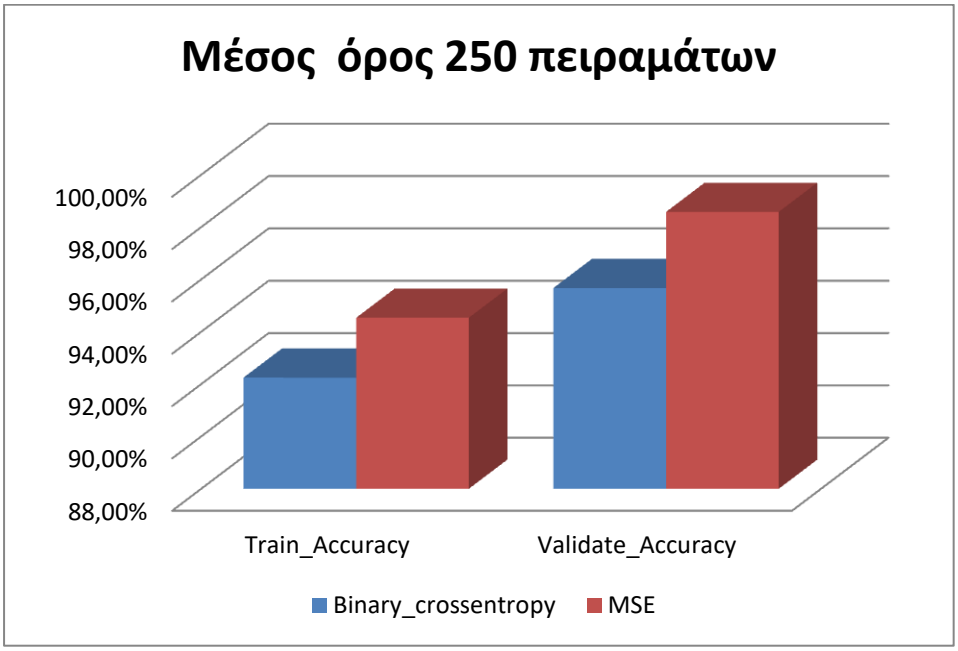
Εικόνα 37- Συνολικά αποτελέσματα 250 πειραμάτων

Από τα γραφήματα των εικόνων 35 και 36 παρατηρούμε ότι τα πειράματα με loss function MSE έχουν καλύτερη απόδοση. Αυτό διακρίνεται ξεκάθαρα στο γράφημα της εικόνας 37, όπου η ευθεία των ορθών αποφάσεων και της ακρίβειας αγγίζουν το βέλτιστο.



Εικόνα 38 Σύνολο Ορθών αποφάσεων ανά Loss Function

Επίσης οι παραπάνω τιμές συνάδουν και με τα γραφήματα των εικόνων 35,36,37 όπου και εκεί είναι φανερό ότι το μοντέλο μας εκπαιδεύεται πολύ καλυτέρα με την μεταβλητή loss function **MSE** από ότι με την **Binary Crossentropy.**



Εικόνα 39- Μέσος όρος 250 πειραμάτων ανά συνάρτηση απώλειας

Επιπλέον, αυτό που θέλουμε να επισημάνουμε είναι ότι, εάν συγκρίνουμε τις τιμές των ορθών αποφάσεων από όλα τα πειράματα που έχουμε διεξαγάγει, θα διαπιστώσουμε ότι από τη στιγμή που αλλάξαμε την εκπαιδευτική κλίμακα των τιμών τόσο στο Demand όσο και στο Load και σε συνδυασμό με την loss function MSE, στο $4^{ο}$ μέρος, έχουμε αποτελέσματα που δεν πέφτουν κάτω από το 94,66% για το Demand and Load, αλλά και ξεχωριστά όπως φαίνεται στον πίνακα 2.

Μοντέλο Νευρωνικού Δικτύου	Correct Demand_and_load	Correct Demand	Correct Load
Epoch 60-adam-Task 1000-MSE	95,17%	94,66%	99,29%
Epoch 60-adam-Task 1000-MSE	96,71%	100,00%	96,71%
Epoch 60-adam-Task 1000-MSE	97,95%	100,00%	97,95%
Epoch 60-adam-Task 2000-MSE	98,65%	99,43%	99,22%
Epoch 60-adam-Task 2000-MSE	99,10%	99,10%	100,00%
Epoch 60-adam-Task 2000-MSE	99,385%	99,863%	99,385%
Epoch 60-adam-Task 2000-MSE	99,50%	99,34%	99,62%
Epoch 60-adam-Task 2000-MSE	99,539%	100,000%	99,539%
Epoch 100-adam-Task 2000-MSE	99,735%	100,000%	99,735%
Epoch 100-adam-Task 2000-MSE	100,00%	100,00%	100,00%
Epoch 100-adam-Task 2000-MSE	100,00%	100,00%	100,00%
Epoch 100-adam-Task 2000-MSE	100,000%	100,000%	100,000%
Epoch 100-adam-Task 2000-MSE	100,000%	100,000%	100,000%
Epoch 100-adam-Task 2000-MSE	100,000%	100,000%	100,000%

Πίνακας 2 Συνολικά ποσοστά ορθών αποφάσεων - MSE

Άρα έπειτα και από τα τελευταία πειράματα καταλήγουμε στο συμπέρασμα ότι το μοντέλο που θα καταλήξουμε θα είναι ένα νευρωνικό δίκτυο δύο εισόδων, δύο κρυφών επιπέδων, όπου το κάθε κρυφό επίπεδο θα έχει 16 και 32 νευρώνες αντίστοιχα και τέλος θα έχει μία έξοδο. Θα χρησιμοποιήσουμε loss function MSE, βελτιστοποίηση Adam με βήμα μάθησης την εξορισμού τιμή 0,001, epoch 100 και πλήθος εκπαιδευτικών τιμών 2000 οι οποίες θα κυμαίνονται από το 0 έως το 100 για το Demand και για το Load από το 0 έως το 10000.

Έπειτα από τρείς χιλιάδες εκατόν ενενήντα (3190) πειράματα διαπιστώνουμε από το εκπαιδευτικό μέρος ότι όλες οι παράμετροι παίζουν πολύ σημαντικό ρόλο και κάθε πρόβλημα αντιμετωπίζετε με τον δικό του τρόπο, δεν υπάρχει μία λύση για όλα τα θέματα. Κάθε εργασία έχει τις δίκες της ιδιαιτερότητες και σαν τέτοια πρέπει να αντιμετωπίζετε. Παράδειγμα, αναζητώντας στο διαδίκτυο, μας προτεινόταν η **Binary Crossentropy** ως loss function για θέματα απόφασης που απαιτούσαν έξοδο 0 και 1, αυτή η λύση δεν δούλεψε τόσο καλά όσο με η **MSE**. Αφού καταλήξαμε στο

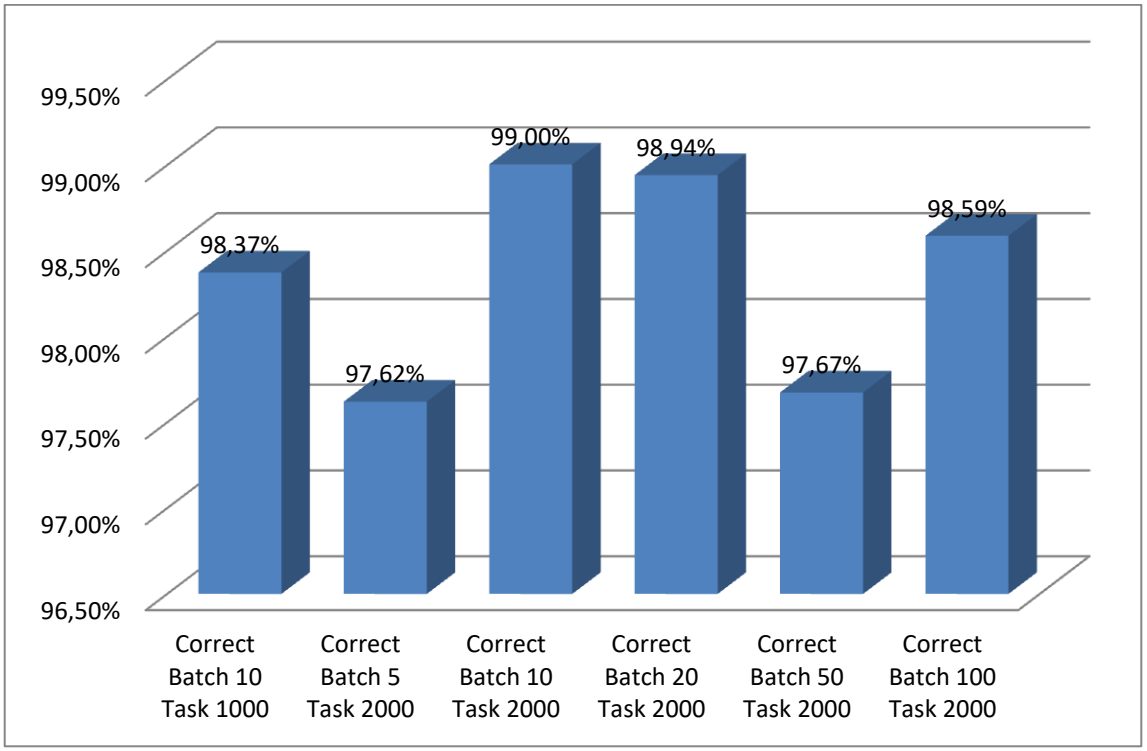
εκπαιδευτικό μοντέλο θα προχωρήσουμε στο επόμενο στάδιο στην διαδικασία της δοκιμής για την εξαγωγή περαιτέρω συμπερασμάτων.

4.3.2. Δοκιμασία Νευρωνικού Δικτύου

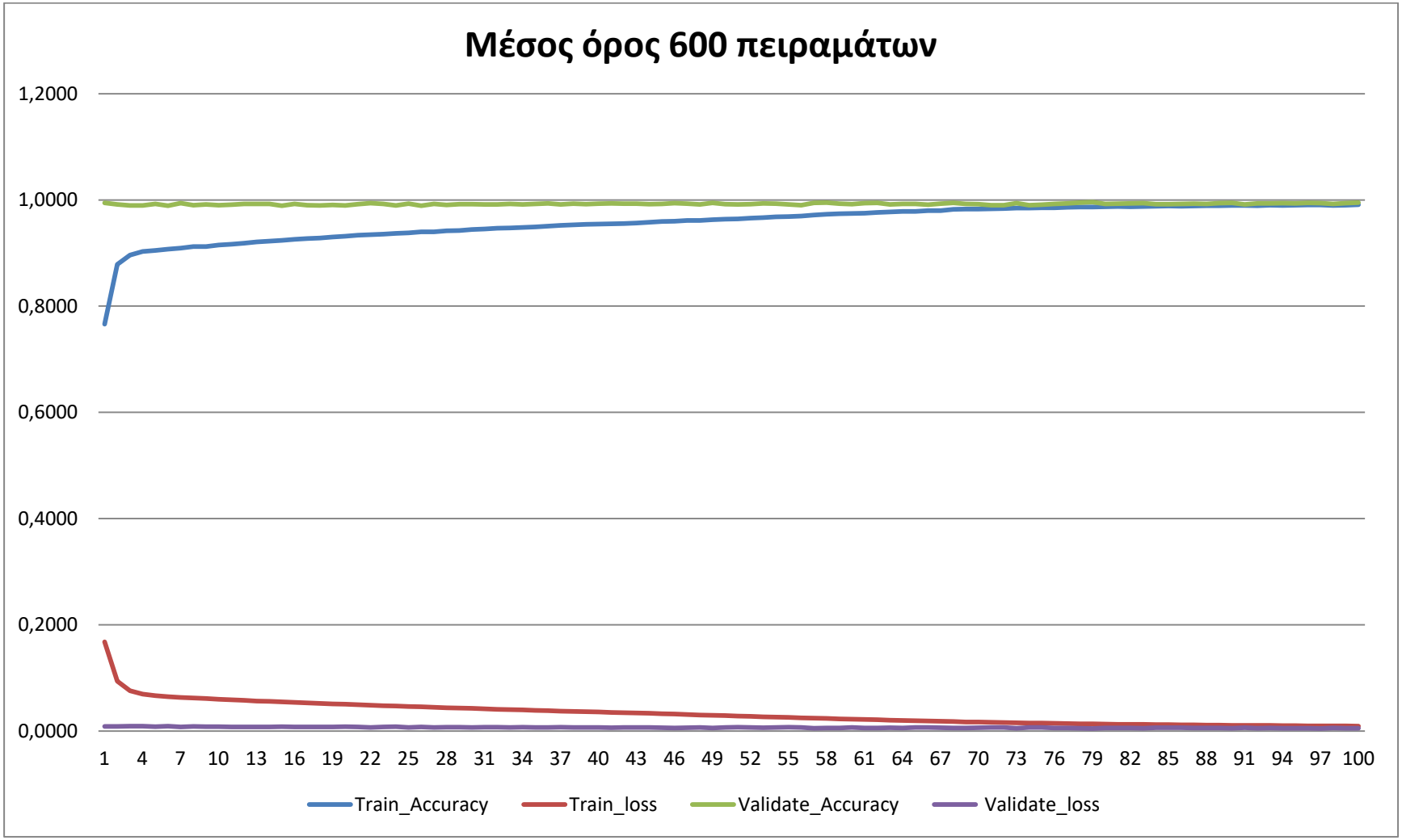
Έπειτα από την δημιουργία του πιο αποτελεσματικού νευρωνικού δικτύου θα εκτελέσουμε δοκιμές στο μοντέλο μας. Θα διεξαγάγουμε πειράματα με διάφορες παραλλαγές στις παραμέτρους που αφορούν στο μέρος της δοκιμασίας του προγράμματος.

1ο μέρος- Batches

Στο παρακάτω γράφημα της εικόνας 40 βλέπουμε το ποσοστό ορθών αποφάσεων του μοντέλου μας ανά παρτίδες εργασιών (batch). Παρατηρούμε ότι οι αποδόσεις είναι εξαιρετικές και αυτό επιβεβαιώνεται και από το γράφημα της εικόνας 41 που αφορά τον μέσο όρο 600 πειραμάτων σχετικά με το πώς εκπαιδεύτηκε το νευρωνικό μας μοντέλο. Τέλος συνάγεται το συμπέρασμα ότι όταν έχουμε παρτίδες ανά 10 (batch 10) έχουμε καλύτερη απόδοση.

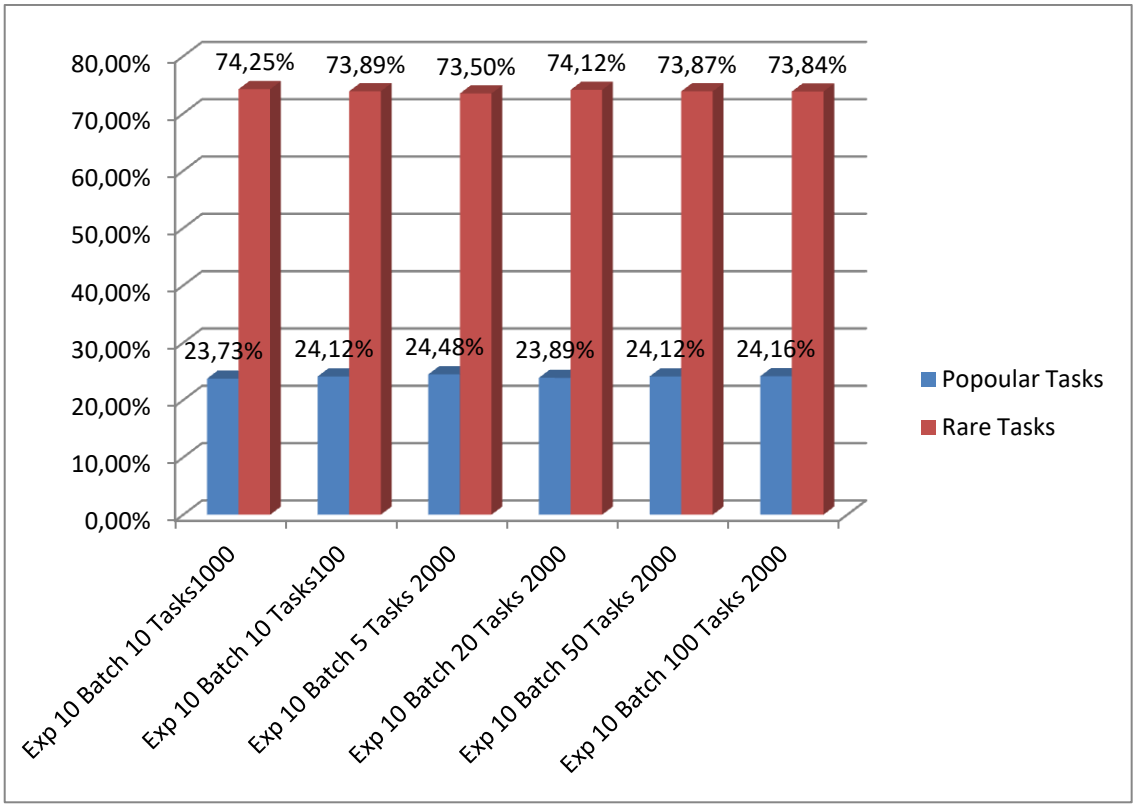


Εικόνα 40 – Ποσοστό ορθών αποφάσεων ανά Batches σε 600 πειράματα

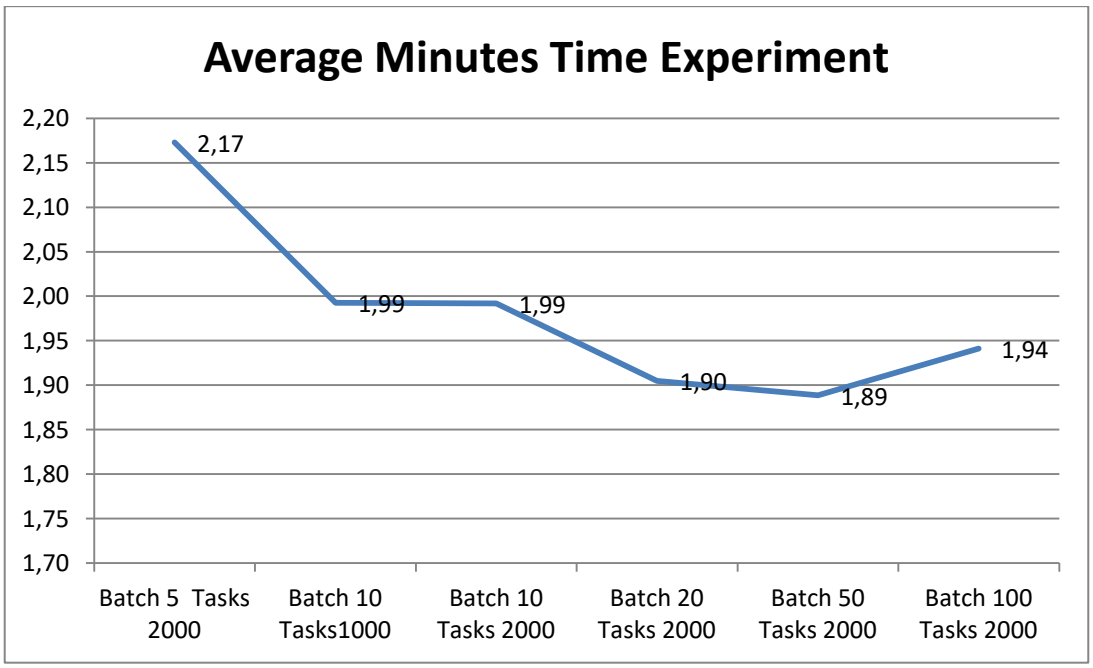


Εικόνα 41 Μέσος όρος -Batch από 5-100 -Tasks 1000-2000

Στο γράφημα της εικόνας 42 βλέπουμε τον μέσο όρο επιλογής των ζευγών με υψηλό demand και χαμηλό Load (Demand>0,5 and Load<0.5) σε 600 πειράματα ως (popular) δημοφιλή task ενώ όλα τα υπόλοιπα με την μορφή σπανίων (rare) tasks δεν επιλέγονται.



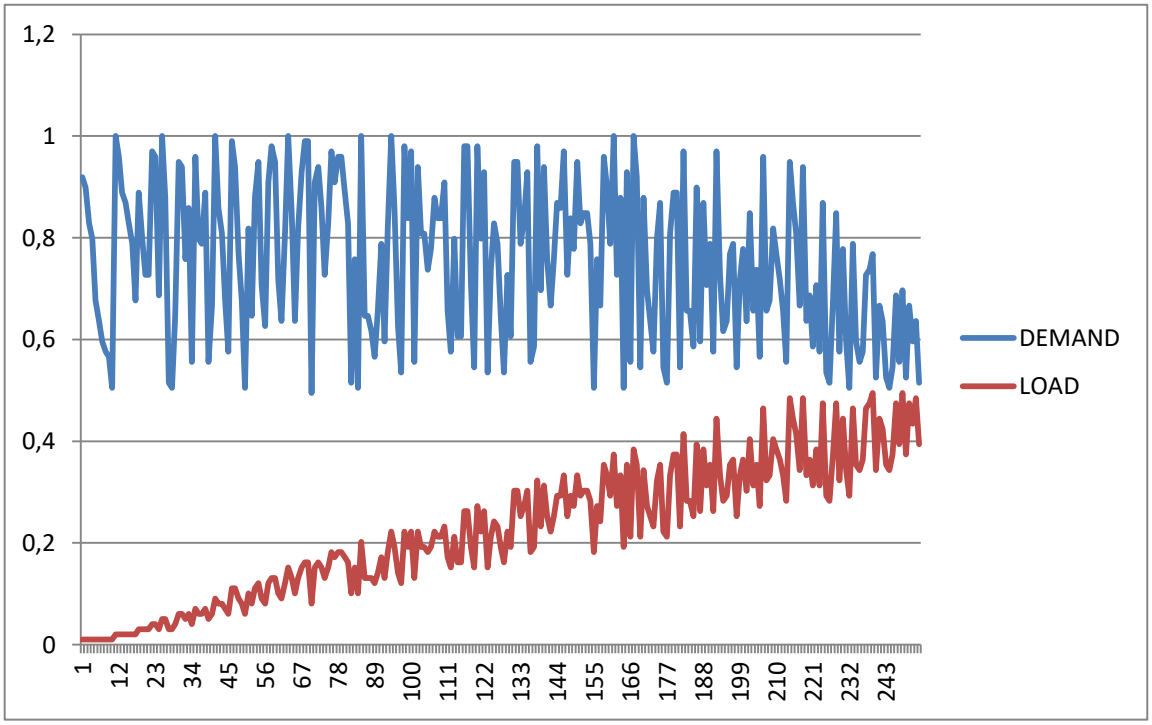
Εικόνα 42 – Μέσος όρος επιλογής Δημοφιλών – Σπανίων tasks σε 600 πειράματα



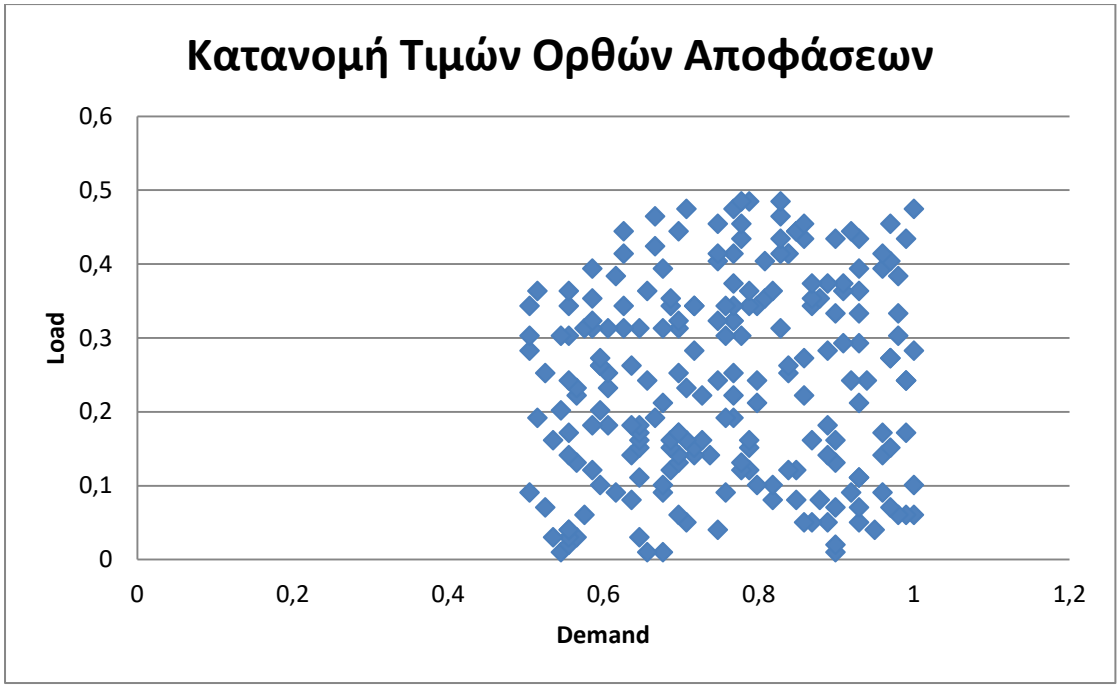
Εικόνα 43- Μέσος όρος σε λεπτά εκτέλεσης πειραμάτων

Στην εικόνα 43 βλέπουμε το μέσο όρο του χρόνου που απαιτήθηκε για να γίνει η κάθε δεκάδα πειραμάτων σε κάθε κύκλο του προγράμματος. Όπως είναι φανερό όταν οι παρτίδες των εργασιών είναι μικρότερες απαιτείται περισσότερος χρόνος αλλά χρειαζόμαστε λιγότερη υπολογιστική ισχύ, ενώ όταν έχουμε μεγάλες παρτίδες εργασιών χρειαζόμαστε λιγότερο χρόνο αλλά μεγαλύτερη υπολογιστική ισχύ. Ενώ παρατηρούμε ότι για 5, 10, 20, 50 batch ο χρόνος μειώνεται, για 100 batch ο χρόνος αυξάνεται, ενώ εμείς θα περιμέναμε ο χρόνος να συνεχίσει να μειώνεται. Αυτό συμβαίνει διότι ναι μεν έχουμε μεγάλη παρτίδα εργασιών αυτό που καθυστερεί το πείραμα είναι η υπολογιστική ισχύ. Άρα δεν είναι πανάκεια το μεγάλο μέγεθος της παρτίδας εργασιών, θα πρέπει να υπάρχει ισορροπία μεγέθους παρτίδας και υπολογιστικής ισχύς.

Στα γραφήματα των εικόνων 44 και 45 εμφανίζεται η κατανομή των τιμών που έχουν επιλεχθεί ως ορθές αποφάσεις. Παρατηρούμε ότι οι τιμές που επιλέγονται από το νευρωνικό μας δίκτυο πληρούν τις προϋποθέσεις ορθότητας του πειράματος μας. Με άλλα λόγια το νευρωνικό μας δίκτυο έχει επιλέξει τιμές μεταξύ 0 και 0,5 για το Load και 0,5 έως 1 για το Demand.

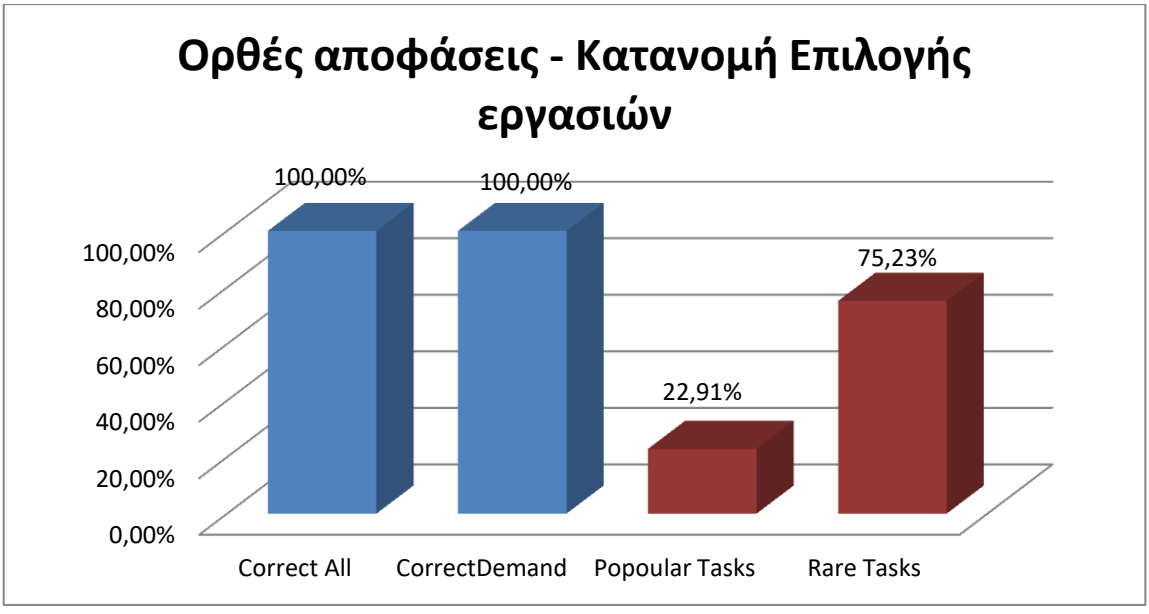


Εικόνα 44 Εύρος ζευγών τιμών που επιλέχθηκαν για το Demand και το Load , ορθών αποφάσεων



Εικόνα 45 – Περιοχή τιμών ζευγών ορθών αποφάσεων Demand και Load

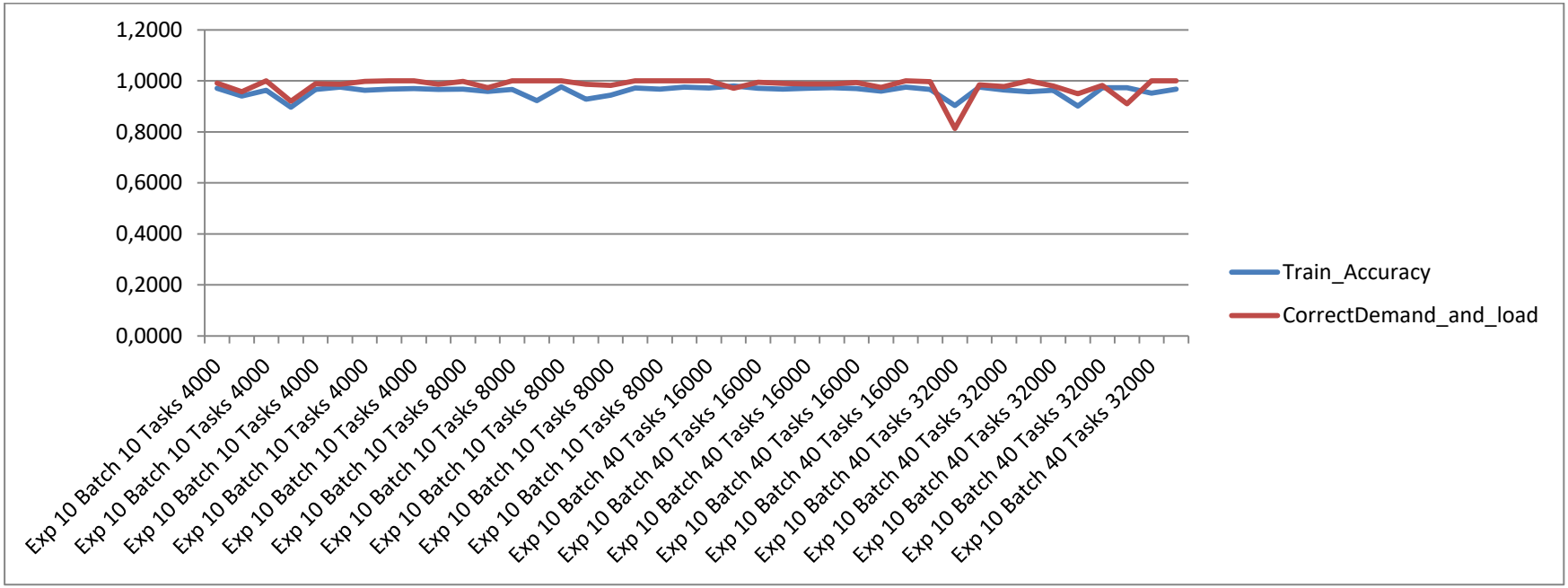
Τα παραπάνω γραφήματα των εικόνων 44 και 45 περιέχουν τις τιμές που παρήχθησαν από το νευρωνικό μας δίκτυο, το οποίο πήρε τις αποφάσεις του γραφήματος της εικόνας 46.



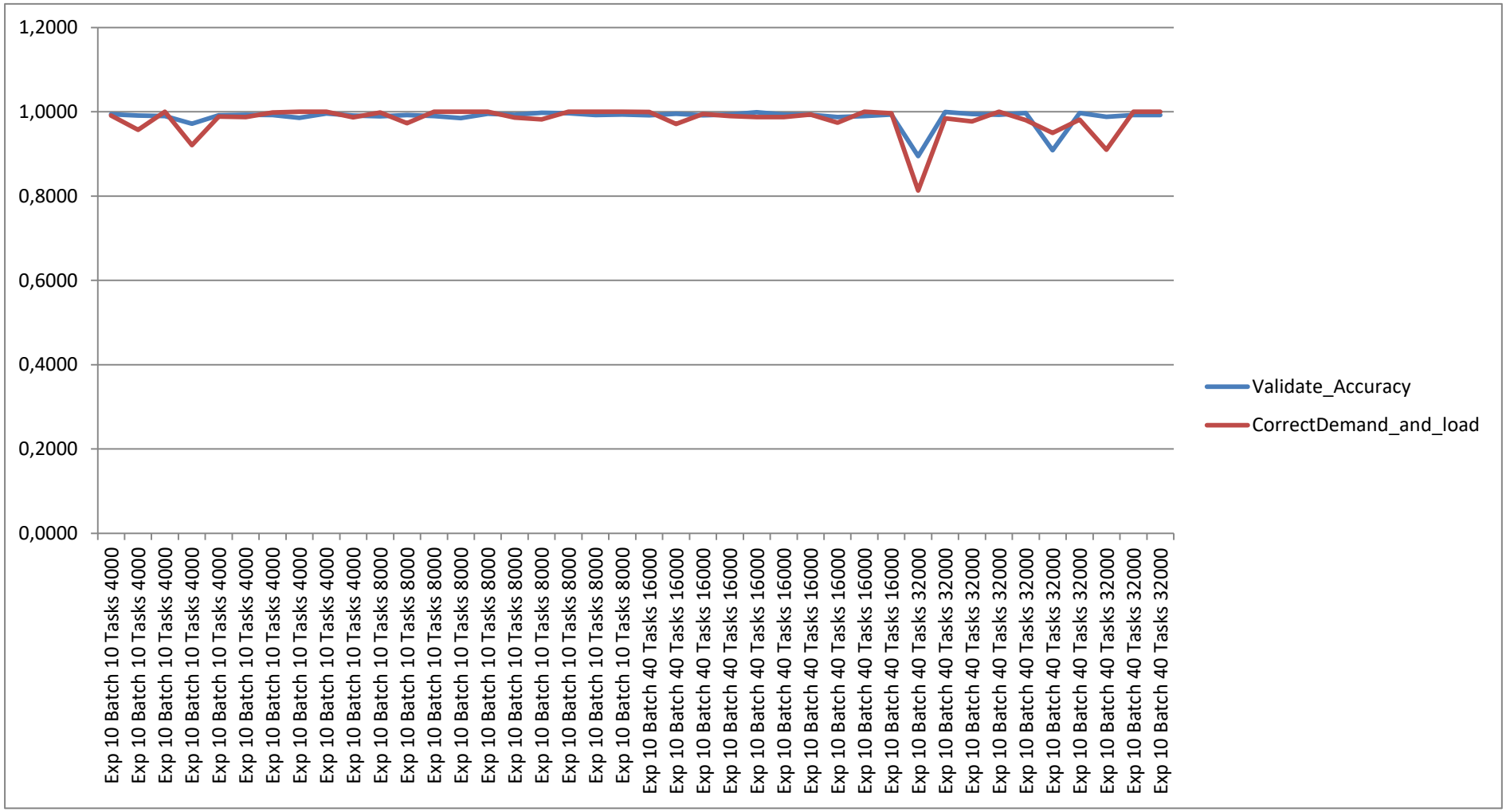
Εικόνα 46 Ποσοστό ορθών αποφάσεων – Ποσοστό Δημοφιλών και σπανίων εργασιών

2ο μέρος- Tasks

Σε αυτό το μέρος τροφοδοτούμε το μοντέλο μας με περισσότερες εργασίες και ανεβάζουμε και τις παρτίδες των εργασιών ώστε να αξιολογήσουμε τη συμπεριφορά του δικτύου μας. Εισάγουμε στους νευρώνες από 4000 εργασίες έως 32000 χιλιάδες εργασίες και τις τροφοδοτούμε ανά 10,40 και 80 κάθε φορά. Στο γράφημα της εικόνας 47 και της εικόνας 48, παρατηρούμε σε πλήθος 400 πειραμάτων την εκπαίδευση και την ακρίβεια εκπαίδευσης αντίστοιχα σε κάθε εικόνα, σε συνδυασμό με το πλήθος των σωστών αποφάσεων.

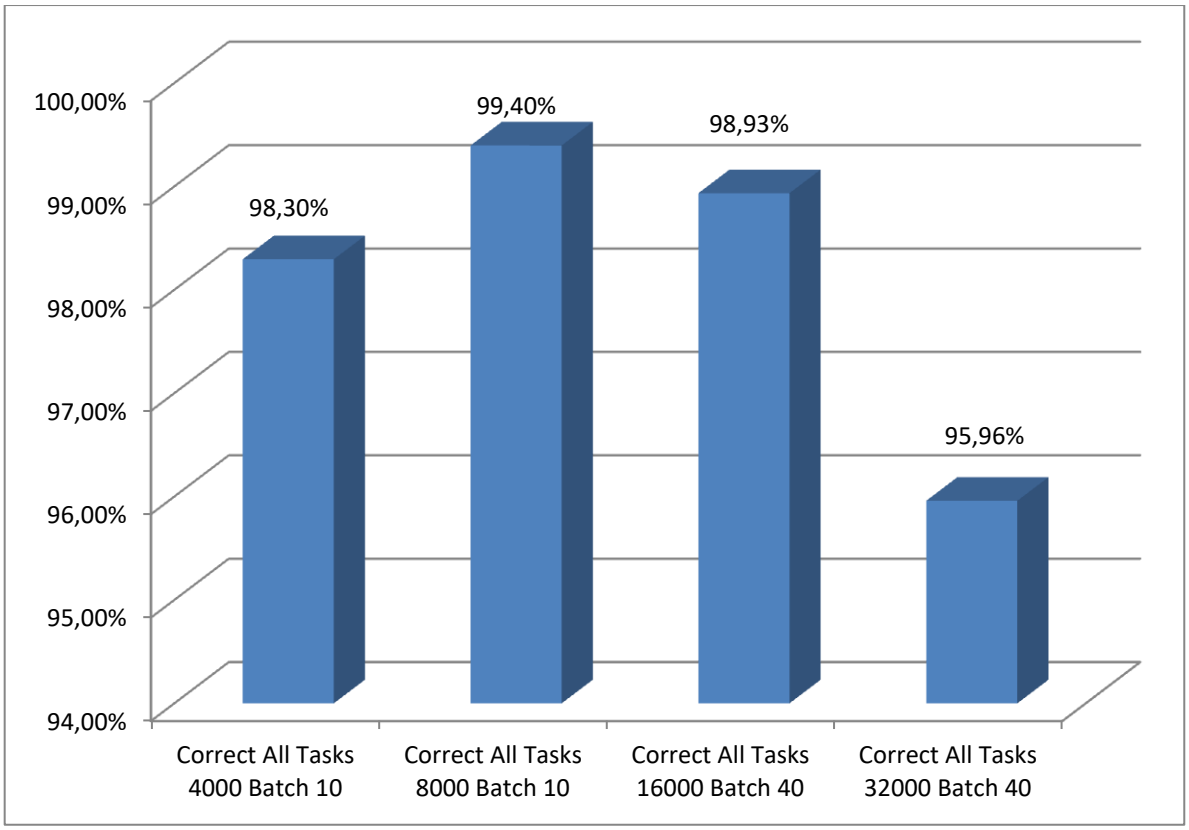


Εικόνα 47 Συνολική εκπαίδευση και ποσοστό ορθών αποφάσεων σε πλήθος 400 πειραμάτων



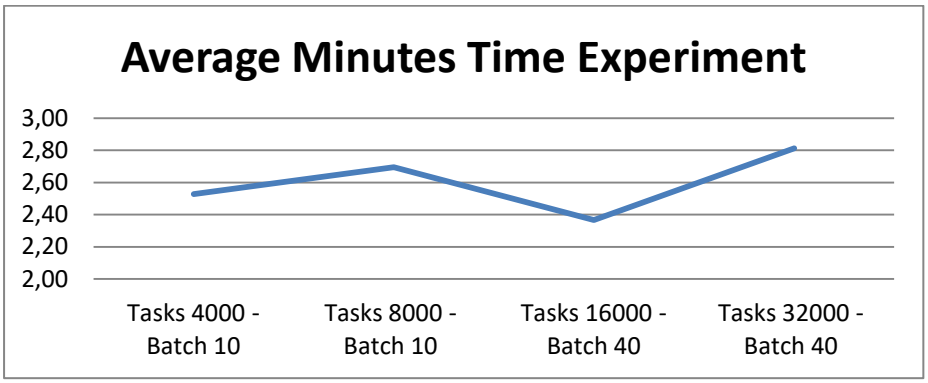
Εικόνα 48 Συνολική ακρίβεια εκπαίδευσης και ποσοστό ορθών αποφάσεων σε πλήθος 400 πειραμάτων

Από τα παραπάνω γραφήματα συμπεραίνουμε ότι η απόδοση της εκπαίδευσης του νευρωνικού μας δικτύου αντικατοπτρίζεται και στην ορθότητα της απόφασης του μοντέλου μας. Αυτό το διαπιστώνουμε και στο παρακάτω γράφημα της εικόνας 49.



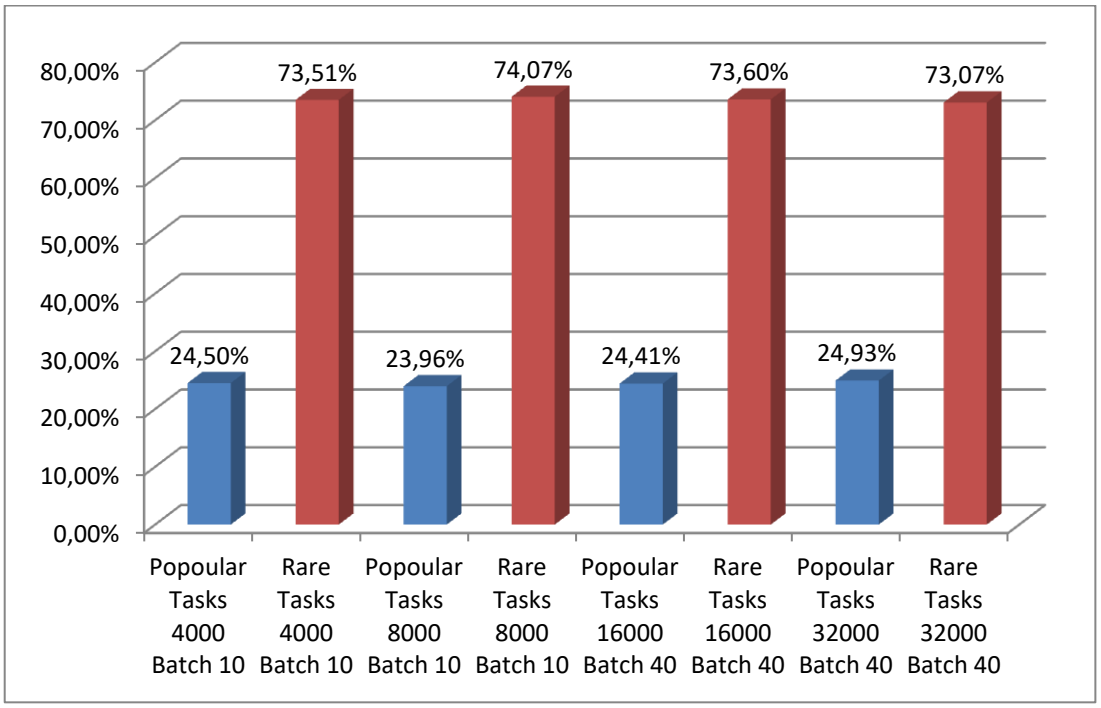
Εικόνα 49 –Πλήθος σωστών αποφάσεων 400 πειραμάτων

Παρατηρούμε ότι το μοντέλο μας αποδίδει καλύτερα στις 8000 εργασίες με παρτίδα εργασιών 10. Ενώ μειώνεται η ικανότητα πρόβλεψης του μοντέλου μας όταν οι εργασίες τετραπλασιάζονται, παρόλο που τετραπλασιάζεται και η παρτίδα των εργασιών. Στο γράφημα της εικόνας 50 παρουσιάζεται ο μέσος χρόνος διάρκειας σε λεπτά του εκάστοτε πειράματος.



Εικόνα 50 Μέσος όρος εκπαίδευσης σε λεπτά εκτέλεσης πειραμάτων

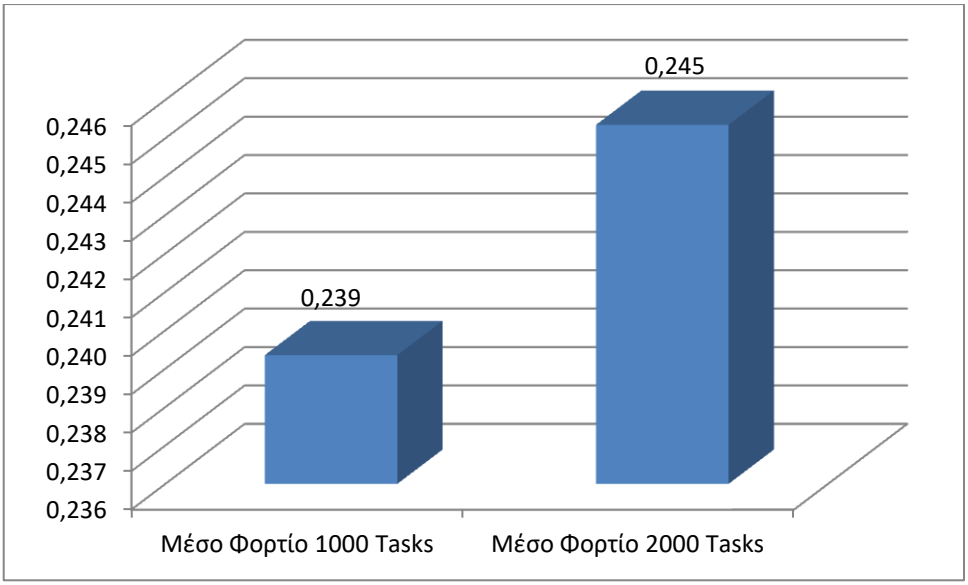
Συμπεραίνουμε από το γράφημα της εικόνας 50 ότι αυτή η χρονική υστέρηση του πειράματος με τις 32000 εργασίες έχει αντίκτυπο και στην απόδοσή του. Εικάζουμε ότι η καθυστέρηση αυτήν οφείλεται στην υπολογιστική ισχύ. Άρα, εάν θέλουμε να εφαρμόσουμε το παραπάνω μοντέλο θα πρέπει να λάβουμε υπόψη μας σε ποιο μηχάνημα θα εκτελεστεί ο κώδικας του προγράμματος.



Εικόνα 51 Μέσος όρος επιλογής δημοφιλών και σπανίων tasks σε 400 πειράματα

Στο γράφημα της εικόνας 51 βλέπουμε το μέσο όρο επιλογής των ζευγών με υψηλό demand και χαμηλό Load (Demand>0,5 and Load<0.5) σε 400 πειράματα ως δημοφιλή (popoular) task ενώ όλα τα υπόλοιπα με την μορφή σπανίν (rare) tasks δεν επιλέγονται. Αυτό που παρατηρούμε στο μέρος της δοκιμασίας του νευρωνικού μας δικτύου και σε μία κλίμακα 1000 πειραμάτων η χαμηλότερη απόδοση ήταν 95,96% που σημαίνει μια λειτουργία εξαιρετική. επίσης, διαπιστώνουμε ότι κατά μέσο όρο επιλέγονται από τους κόμβους για τοπική εκτέλεση περίπου το 24% των εργασιών ενώ το υπόλοιπο αποστέλλεται στο cloud ή σε άλλο κόμβο.

Επίσης, ο μέσος όρος των βαρών που επιλέγεται για να εκτελέσει ο κάθε κόμβος αποτυπώνεται στο γράφημα της εικόνας 52. Όπως παρατηρούμε, ο μέσος όρος κυμαίνεται γύρω από το 0,24.



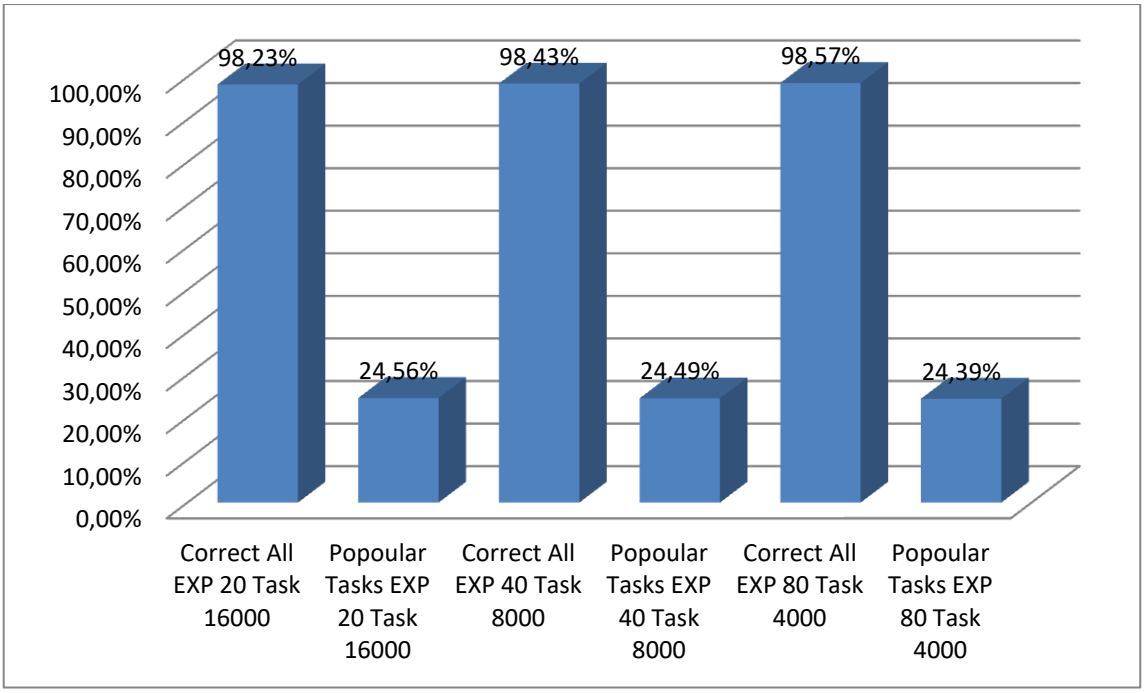
Εικόνα 52 – Μέσος όρος φορτίου δημοφιλών tasks σε 200 πειράματα

3ο μέρος- Πειράματα

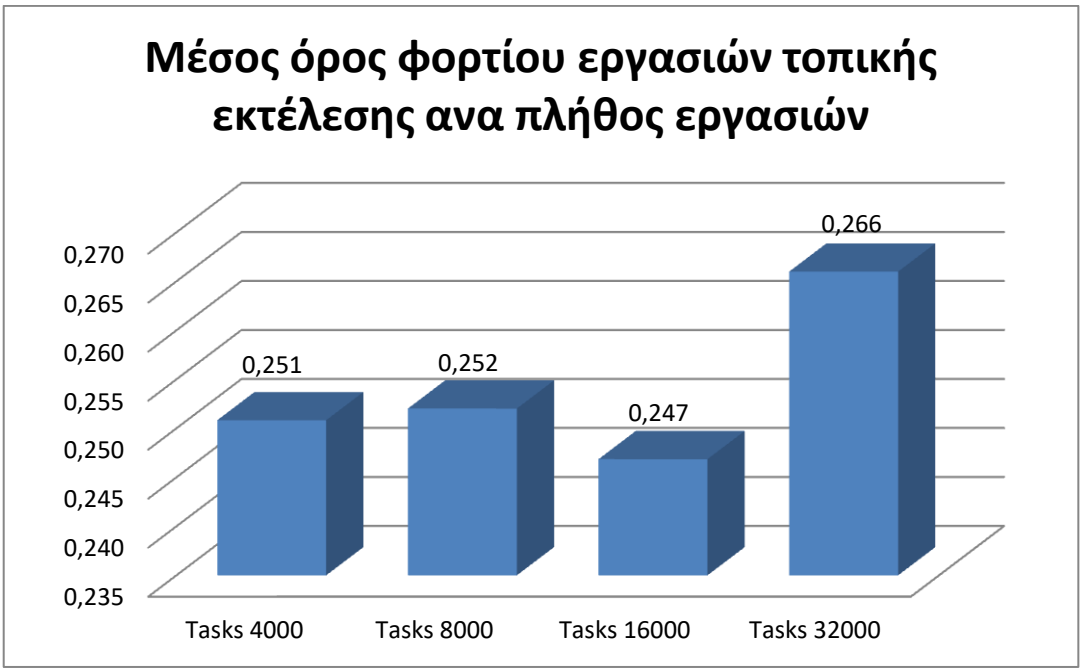
Σε αυτό το μέρος αλλάζουμε το πλήθος των πειραμάτων που εκτελούνται σε κάθε κύκλο του προγράμματός μας. Κάθε φορά που εκτελείται το πρόγραμμα μας, το μοντέλο μας εκπαιδεύεται και έπειτα ακολουθεί ο κύκλος της πρόβλεψης του νευρωνικού μας δικτύου και αυτός ο κύκλος επαναλαμβάνεται τόσες φορές όσες φορές έχουμε εμείς ορίσει τον αριθμό των πειραμάτων. Στα προηγούμενα μέρη, ο αριθμός των πειραμάτων που εκτελούνταν σε κάθε κύκλο του προγράμματος ή καλύτερα σε κάθε εκπαίδευση ήταν δέκα (10). Εδώ τα πειράματα μας σε κάθε κύκλο εκπαίδευσης θα κυμανθούν από 20 έως 80. Με αυτές τις αλλαγές θέλουμε να δούμε πως συμπεριφέρεται το μοντέλο διότι θα απαιτηθεί μεγαλύτερη υπολογιστική ισχύ, περισσότερος χρόνος.

Στο γράφημα της εικόνας 53 παρατηρούμε τα αποτελέσματα χιλίων τετρακόσιων (1400) πειραμάτων. Εκτελέστηκαν 200 πειράματα 16000 εργασιών, 400 πειράματα 8000 εργασιών και 800 πειράματα 4000 εργασιών. Όπως συμπεραίνουμε οι διαφορές δεν είναι μεγάλες.

Στην εικόνα 54 βλέπουμε το μέσο φορτίο με το οποίο επιβαρύνεται ο κάθε κόμβος για την εκτέλεση των επιλεγμένων εργασιών. Είναι ορατό ότι το μέσο φορτίο είναι περίπου 0,25 .

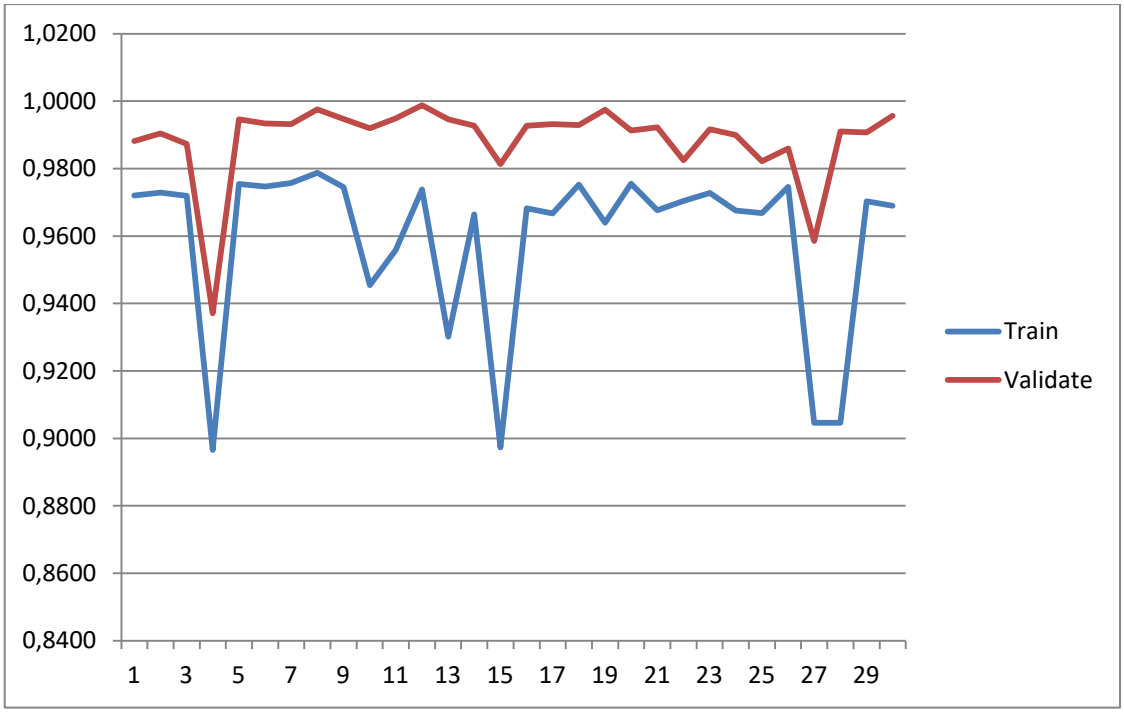


Εικόνα 53 – Πλήθος σωστών αποφάσεων σε συνδυασμό με το πλήθος δημοφιλών εργασιών για τοπική εκτέλεση ανά πλήθος πειραμάτων.

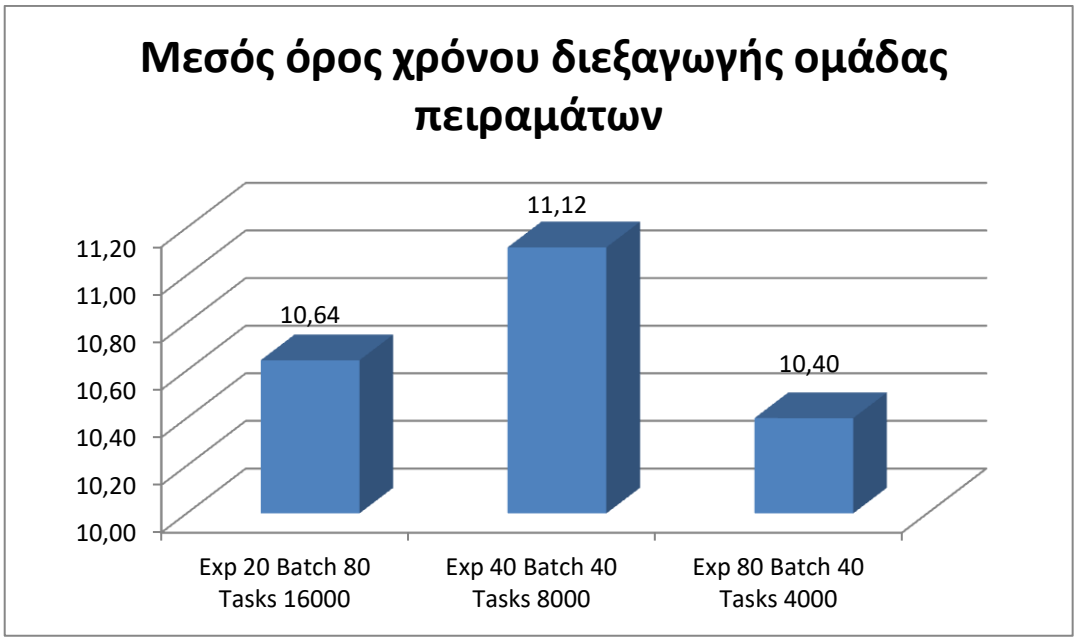


Εικόνα 54 Μέσος όρος φορτίου δημοφιλών tasks

Όπως φαίνεται και στην εικόνα 55 η εκπαίδευση του νευρωνικού δικτύου συνεχίζει να βρίσκεται σε άριστα επίπεδα, για αυτό άλλωστε ο μέσος όρος των αποτελεσμάτων μας δεν πέφτει κάτω από το 98,23%



Εικόνα 55 – Πορεία Εκπαίδευσης νευρωνικού δικτύου 1400 πειραμάτων



Εικόνα 56 - Μεσός όρος χρόνου διεξαγωγής ομάδας πειραμάτων

Στην εικόνα 56 εμφανίζεται ο χρόνος σε λεπτά που απαιτούνταν για την διεξαγωγή ενός κύκλου εκπαίδευσης και εξαγωγή πρόβλεψης του νευρωνικού δικτύου. Με άλλα λόγια για να διεξαχθεί ένας κύκλος προγράμματος, στον οποίο κύκλο εκτελούνται 20 πειράματα και ελέγχονται 16000 εργασίες απαιτείται χρόνος 10,64 λεπτών. Τα παραπάνω πειράματα απαιτούσαν χρόνο, εξαιτίας του μεγάλου αριθμού των πειραμάτων αλλά και υπολογιστική ισχύ εξαιτίας του πλήθους των εργασιών. Ήταν

μία δοκιμασία στην οποία ανταπεξήλθε το μοντέλο μας με σχεδόν άριστα αποτελέσματα.

Το παραπάνω νευρωνικό δίκτυο μπορούμε εάν θέλουμε να το αποθηκεύσουμε στην τρέχουσα κατάσταση του, όπως είναι εκπαιδευμένο και να το φορτώσουμε αργότερα σε άλλο πρόβλημα. Μπορούμε να καλέσουμε την λειτουργία save(). Για save(), περνάμε τη διαδρομή αρχείου και το όνομα του αρχείου στο οποίο θέλουμε να αποθηκεύσουμε το μοντέλο με επέκταση h5.

```
model.save('models/experiment_model.h5')
```

Αυτή η μέθοδος θα αποθηκεύσει για το μοντέλο - την αρχιτεκτονική, τα βάρη, τη βελτιστοποίηση, την κατάσταση του βελτιστοποιητή, το ποσοστό εκμάθησης, την απώλεια κ.λπ. Τώρα που έχουμε αποθηκεύσει αυτό το μοντέλο, μπορούμε να χρησιμοποιήσουμε το μοντέλο αργότερα σε ένα άλλο πρόβλημα. Για να γίνει αυτό, εισάγουμε πρώτα τη συνάρτηση *load_model ().* Στη συνέχεια, μπορούμε να καλέσουμε τη λειτουργία για φόρτωση του μοντέλου, δείχνοντας το αποθηκευμένο μοντέλο στο δίσκο.

```
from tensorflow.keras.models import load_model
new_model = load_model('models/experiment_model.h5')
```

ΣΥΜΠΕΡΑΣΜΑΤΑ

Νέες ιδέες και τεχνολογίες έρχονται και εφαρμόζονται στη σημερινή εποχή. Οι ιδέες αυτές δείχνουν καινοτόμες, παρόλα αυτά έρχονται από το παρελθόν και εφαρμόζονται στο μέλλον. Τώρα δημιουργήθηκαν οι κατάλληλες συνθήκες και αυτά που ειπώθηκαν πριν πολλά χρόνια έρχονται να εφαρμοστούν σήμερα. Με την διάχυση του τεράστιου όγκου της πληροφορίας και την αύξηση της υπολογιστικής ισχύος ο κάθε ένας από εμάς έχει γίνει κοινωνός όλον αυτών των υπέροχων τεχνολογιών. Η τεχνητή νοημοσύνη, η ενισχυτική μάθηση, τα αποκεντρωμένα δίκτυα είναι πλέον προσβάσιμα για τον απλό ερευνητή, φοιτητή που θέλει να διευρύνει τις γνώσεις του ή να εκπονήσει την εργασία του. Ερευνώντας, συνεχώς διαπιστώνουμε ότι δεν υπάρχει όριο για το που θα πρέπει να σταματήσει μία εργασία που πραγματεύεται όλες αυτές τις προαναφερόμενες φανταστικές νέες εφαρμογές. Συνεχώς ανακαλύπταμε και κάτι πιο ενδιαφέρον και θεωρούσαμε την εργασία μας ελλιπής. Εξάλλου στον κόσμο της πληροφορικής τα πράγματα τρέχουν με ιλιγγιώδης ρυθμούς.

Το παραπάνω πόνημα για το συγγραφέα αποτέλεσε ένα παράθυρο για έναν νέο κόσμο τον οποίο άκουγε αλλά δεν είχε αγγίξει ποτέ. Ξεκινώντας έπρεπε να ξεκαθαριστούν έννοιες, όπως τεχνητή νοημοσύνη, βαθιά μάθηση, νευρωνικά δίκτυα, keras, tensorflow, βελτιστοποιητές, βήμα μάθησης, πίνακες αποφάσεων, συναρτήσεις ενεργοποίησης. Διαπιστώσαμε ότι το θεωρητικό υπόβαθρο είναι το σημαντικότερο στην δόμηση μίας εργασίας σαν αυτή που παρουσιάζεται εδώ σε αυτές τις γραμμές. Αφού όλα έγιναν κατανοητά και τοποθετήθηκαν στις κατάλληλες θέσεις τότε και μόνο τότε ήμασταν σε θέση να προχωρήσουμε και στο μέρος τις δημιουργίας του κώδικα του προγράμματος.

Μέσα από την εκπόνηση του κώδικα γινόμασταν σοφότεροι και ταυτόχρονα αισθανόμασταν μικροί απέναντι στους εμπνευστές και στου εφευρέτες όλων αυτών των τεχνολογιών. Διαπιστώναμε συνεχώς το εύρος αυτής της εξειδίκευσης και επιβεβαιώναμε αυτό που διαχεόταν στο διαδίκτυο, ότι η ανθρωπότητα έχει ένα όπλο που μπορεί να το χρησιμοποιήσει κατά το δοκούν. Σε κάθε βήμα της συγγραφής εφαρμόζαμε, δοκιμάζαμε απορρίπταμε και καταλήξαμε σε έναν κώδικα ο οποίος τελειοποιήθηκε όταν φτάσαμε στο στάδιο των πειραματισμών.

Διεξαγάγαμε πέντε χιλιάδες εκατόν ενενήντα πειράματα (5190) και συνεχώς συμπεραίναμε και μαθαίναμε πόσες παράμετροι ρυθμίζουν την απόδοση του μοντέλου μας. Ξεκινήσαμε με αστάθειες αλλά με την παρατήρηση και την επανάληψη καταλήξαμε σε ένα νευρωνικό δίκτυο αρκετά ικανοποιητικό. Ανατρέχαμε στην θεωρία και ανακαλύπταμε τις λύσεις. Αυτό που διαπιστώσαμε είναι ότι στο διαδίκτυο υπάρχει πληθώρα πληροφοριών που μπορεί να σε βοηθήσει αλλά και να σε πλανέψει. Η επίλυση ενός προβλήματος μπορεί να έχει περισσότερες από μία λύσεις καθώς επίσης και μια λύση που προτείνετε για παρόμοιο πρόβλημα μπορεί και να μην αποδώσει τα αναμενόμενα. Αυτό το διαπιστώσαμε με την loss function, στο μέρος της εκπαίδευσης, όπου μας προτεινόταν η **Binary Crossentropy** και εμείς χρησιμοποιήσαμε το μέσο τετραγωνικό σφάλμα **MSE.**

Στο μέρος της δοκιμασίας του νευρωνικού μας δικτύου οδηγηθήκαμε σε ορθά αποτελέσματα χάρη στην κατάλληλη εκπαίδευση που λάμβανε το μοντέλο μας. Έτσι, οποιοδήποτε παράμετρο και εάν αλλάξαμε στο συγκεκριμένο μέρος, οι αποδόσεις δεν έπεφταν κάτω από το 94% και η ακρίβεια της εκπαίδευσης κυμαινόταν στο 99%. Είχαμε μια απόδοση σταθερή, που αυτό άλλωστε είναι το ζητούμενο, ώστε να έχουμε ομαλοποιημένα αποτελέσματα, διότι αυτά θα οδηγηθούν σε έναν μηχανισμό κόμβων εκτέλεσης εργασιών και εάν τα συγκεκριμένα δεδομένα δεν είναι σταθερά υπάρχει κίνδυνος κατάρρευσης του μηχανισμού.

Το συγκεκριμένο εγχείρημα αποτελεί τα πρώτα βήματα του συγγραφέα σε αυτόν τον θαυμαστό κόσμος της βαθιάς μάθησης και ελπίζει να τη συνεχίσει στο μέλλον, διότι σίγουρα η παρούσα εργασία επιδέχεται βελτίωσης αλλά και επέκτασης. Θα δανειστούμε μια εικόνα σχετικά με το **R**einforcement **L**earning. Ένα τέρας που ανά πάσα στιγμή μπορεί να μας κατασπαράξει και πάντα θα νιώθουμε μικροί μπροστά του.

Εικόνα 57- deeplizard.com

ΒΙΒΛΙΟΓΡΑΦΙΑ

- ✓ Reinforcement Learning An introduction – Richard S. Sutton and Andrew G Barto second edition
- ✓ Multi-agent Deep Reinforcement Learning for Task Allocation in Dynamic Environment (Dhouha Ben Noureddine, Atef Gharbi and Samir Ben Ahmed)
- ✓ Artificial Intelligence and Intrusion DetectionCurrent and Future Directions (Jeremy Frank, Division of Computer Science)
- ✓ Comparison of Trend Detection Algorithms in the Analysis of Physiological Time-Series Data (William W. Melek*, Member, IEEE, Ziren Lu, Alex Kapps, and William D. Fraser)
- ✓ A Byte of Python - python.swaroopch.com
- ✓ innovationatwork.ieee.org
- ✓ deeplizard.com
- ✓ www.analyticsvidhya.com
- ✓ www.tensorflow.org
- ✓ machinelearningmastery.com
- ✓ keras.io

ΠΑΡΑΡΤΗΜΑ – ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
from random import randint
from sklearn.utils import shuffle
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from functools import reduce
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense
from keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import datetime
import statistics
import math
import keras
import numpy as np
import random
import operator
import pandas as pd
import xlsxwriter
from keras.callbacks import CSVLogger
```

```
#Trigg parameters
U=[]
e=[]
s=[]
M=[]
T=[]
P=[]
L=[]
Theta=0.2
#parameters
Values = 2000
n_features =2
train_labels = []
train_demand = []
train_Load = []
#Produce values
random_Demand = np.random.normal(0,100,Values)

#calculation Demand
U=np.average(random_Demand)
s=U/100
M=U/10

for i in range(Values):
    random_Load = randint(0,10000)
```

```
    U = ((Theta*(random_Demand[i])) +(1-Theta)*(U))
    e=(random_Demand[i]-U)
    s=Theta*e-(1-Theta)*s
    M=(Theta*abs(e))+(1-Theta)*M
    T=s/M
    L= random_Load
    train_demand.append(T)
    train_Load.append(L)

#Production list label
    if T > 0  and L<=5000:
        train_labels.append(1)
    else:
        train_labels.append(0)

#make lists arrayes
train_labels = np.array(train_labels)
train_demand = np.array(train_demand)
train_Load=np.array(train_Load)

#normalize (0,1)
scaler = MinMaxScaler(feature_range=(0,1))
scaled_train_demand = scaler.fit_transform(train_demand.reshape(-1,1))
scaled_train_Load = scaler.fit_transform(train_Load.reshape(-1,1))

#concatenate arrays demand and load
```

```
InDNN = np.concatenate((scaled_train_demand,scaled_train_Load), axis=1)
#Train the model

model = Sequential([
    Dense(units=16, input_shape=(n_features,), activation='relu'),
    Dense(units=32, activation='relu'),
    Dense(units=1, activation='sigmoid')
])
model.compile(
    loss='mse',
    optimizer='adam',
    metrics=['accuracy']
)
#loss='binary_crossentropy',
#sigmoid
#Adam(learning_rate=0.0001)
#'adam'
#fit model DNN
start = datetime.datetime.now()

csv_logger = CSVLogger('Verbose_Train.csv', append=True, separator=';')
model.fit(
    InDNN,
    train_labels,
    batch_size=10,
    epochs=100,
```

```
    shuffle=True,
    verbose=0,
    callbacks=[csv_logger]
)
csv_logger = CSVLogger('Verbose_Valid.csv', append=True, separator=';')
#validate the model
model.fit(
     InDNN,
     train_labels,
     validation_split=0.1,
     batch_size=10,
     epochs=100,
     verbose=0,
     callbacks=[csv_logger]
)

end = datetime.datetime.now()
totalTime = end - start
print("### Training time : ", totalTime, " seconds")

#--------TESTING----

# parameters
Experiments=80
loadThreshold = 0.5
demandthreshold = 0.5
```

```
ThresholdBenefitRate = 1
Tasks=4000
k = 10
a = 5
final = np.zeros(Tasks)
final_bad= np.zeros(Tasks)
load = np.zeros(Tasks)

# variables for metrics
sumTime = 0.0
sumTimes = []

percentageCorrectBenefit_Rate=[]
percentageCorrectDemand_Load=[]
percentageCorrectDemand=[]
percentageCorrectLoad=[]
SumAveragePopoularLoad=[]
SumAverageRareLoad=[]
for e in range(Experiments):
    test_labels =  []
    test_demand = []
    test_Load = []

    est1=[]
    est2=[]
    Final_Demand=[]
```

```
Final_Load=[]
Final_Bad_demand=[]
Final_Bad_Load =[]
Tfinal = []
Final_Benefit = []
    for i in range(Tasks):
    # produce Demand
    random_test = randint(1,100)
    test_demand.append(random_test)
    # produce Load
    random_load_test = randint(1,100)
    test_Load.append(random_load_test)

test_demand = np.array(test_demand)
test_Load=np.array(test_Load)
test_demand, test_Load = shuffle(test_demand, test_Load)

scaler = MinMaxScaler(feature_range=(0,1))
scaled_test_demand = scaler.fit_transform(test_demand.reshape(-1,1))
scaled_test_Load = scaler.fit_transform(test_Load.reshape(-1,1))

sum = 0.0
count = 0.0

start = datetime.datetime.now()
```

```
InDNN_Tasks = np.concatenate((scaled_test_demand,scaled_test_Load), axis=1)
predictions = model.predict_classes(
      InDNN_Tasks
   ,  batch_size=80
   ,  verbose=0
)



#normalize predictions 0 and 1
#P = np.where(predictions >= 0.90, 1, 0)



# create a new array from InDNN if prediction is 1
optimalTask=np.where(predictions==1,InDNN_Tasks,0)
# create a new array from InDNN if prediction is 0
badTask=np.where(predictions == 0,InDNN_Tasks,0)

#remove rows with zero
optimalTask = optimalTask[~np.any(optimalTask == 0, axis=1)]
badTask=badTask[~np.any(badTask == 0, axis=1)]

# found shape in new array after remove zeroes
shapeOptimal=optimalTask.shape
shapebad=badTask.shape
# set j equal shape array for Optimal and set b equal for bad
```

```
    j=shapeOptimal[0]
    b=shapebad[0]

    # split the arrays into two new arrays

    est1, est2 =np.hsplit(optimalTask, 2)
    badDemand, badload = np.hsplit(badTask, 2)

    # change value 0 to 0.0001
    est22= np.where(est2 == 0,0.0001,est2)
    badload1=np.where(badload == 0,0.0001,badload)

    #calcuate the Benefit value
    final= np.exp(((est1/est22)-a))
    final_bad=np.exp(((badDemand/badload1)-a))

    # concatenate the arrayes
    Final_Demand_Load = np.concatenate((final,est1,est22), axis=1)
    Bad_Final_Demand_load=np.concatenate((final_bad,badDemand,badload1),
axis=1)

    #sort the new array descending
    FFF=Final_Demand_Load[np.lexsort(([1,-1]*Final_Demand_Load[:,[0,0]]).T)]
    #sort the new array increasing

BBB=Bad_Final_Demand_load[np.lexsort(([1,1]*Bad_Final_Demand_load[:,[0,0]]).T)]
```

```
#split Final arrayes
final, Final_Demand, Final_Load=np.hsplit(FFF,3)
final_bad, Final_Bad_demand, Final_Bad_Load=np.hsplit(BBB,3)

Final_Bf_Rt=[]
Final_Bf_demand=[]
Final_Bf_load =[]
Final_All=[]

# dismiss tasks low benefit
for i in range(j-k):
   Final_Bf_Rt.append(final[i])
   Final_Bf_demand.append(Final_Demand[i])
   Final_Bf_load.append(Final_Load[i])
   Final_All.append(FFF[i])

end = datetime.datetime.now()
totalTime = end - start
sum += totalTime.total_seconds()
count += 1
#print("### Test time : ", totalTime, " seconds")
#print("*** ", totalTime.total_seconds())


AveragePopoularLoad=((j-k)/Tasks)
AverageRareLoad=((b+k)/Tasks)
```

```
SumAveragePopoularLoad.append(AveragePopoularLoad)
SumAverageRareLoad.append(AverageRareLoad)

df_InDNN_Tasks = pd.DataFrame(InDNN_Tasks)
df_predictions = pd.DataFrame(predictions)
df_est2 = pd.DataFrame(est22)
df_est1 = pd.DataFrame(est1)
df_FFF=pd.DataFrame(FFF)
df_BBB=pd.DataFrame(BBB)
df_Final_Demand=pd.DataFrame(Final_Bf_demand) #Final_Demand
df_Final_Load=pd.DataFrame(Final_Bf_load) #Final_Load
df_Final_Rt=pd.DataFrame(Final_Bf_Rt)
df_final = pd.DataFrame(final)
df_Final_All=pd.DataFrame(Final_All)

writer = pd.ExcelWriter('my_excel_file.xlsx')
df_InDNN_Tasks.to_excel(writer,'InDNN',float_format='%.5f')
df_predictions.to_excel(writer,'predictions',float_format='%.5f')
df_final.to_excel(writer,'final_and_K',float_format='%.5f')
df_est1.to_excel(writer,'Demand_and_K',float_format='%.5f')
df_est2.to_excel(writer,'Load_and_K',float_format='%.5f')
df_FFF.to_excel(writer,'Final_All_and_K',float_format='%.5f')
df_BBB.to_excel(writer,'Final_BAD_All',float_format='%.5f')
df_Final_Demand.to_excel(writer,'Final_Demand',float_format='%.5f')
df_Final_Load.to_excel(writer,'Final_Load',float_format='%.5f')
```

```
df_Final_Rt.to_excel(writer,'Final_Bf',float_format='%.5f')
df_Final_All.to_excel(writer,'Final_All',float_format='%.5f')
writer.save()


counterDemand_Load_Corect=0.0
counterbenefit_rate=0.0
Rightdemand1 = 0.0
counterLoadPopular = 0.0

for i in range (j-k):
    if Final_Bf_demand[i] > demandthreshold and Final_Bf_load[i]<= loadThreshold:
      counterDemand_Load_Corect +=1
    elif Final_Bf_Rt[i]>=ThresholdBenefitRate:
      counterDemand_Load_Corect +=1
for i in range (j-k):
    if Final_Bf_Rt[i]>=ThresholdBenefitRate:
      counterbenefit_rate +=1
for i in range (j-k):
    if Final_Bf_demand[i] > demandthreshold:
       Rightdemand1 +=1
for i in range (j-k):
    if Final_Bf_load[i]<= loadThreshold:
       counterLoadPopular +=1

percentageCorrectDemand_Load.append(counterDemand_Load_Corect/(j-k))
```

```
    percentageCorrectBenefit_Rate.append(counterbenefit_rate/(j-k))
    percentageCorrectDemand.append((Rightdemand1)/(j-k))
    percentageCorrectLoad.append((counterLoadPopular)/(j-k))



sumTime += (sum / count)
sumTimes.append(sum / count)

averageTime = statistics.mean(sumTimes)
print("Average time for getting result (tau): ", averageTime)

#print("Deviation time for getting result (tau): ", averageTimeX)
#averageTimeX = statistics.stdev(sumTimes) #sumTime / Experiments
 #sumTime / Experiments
averagePercentageCorrect_Demand_Load=
statistics.mean(percentageCorrectDemand_Load)
DeviationaveragePercentageCorrect_Demand_Load=
statistics.stdev(percentageCorrectDemand_Load)

averagepercentageCorrectBenefit_Rate=statistics.mean(percentageCorrectBenefit_Ra
te)
DeviationaveragepercentageCorrectBenefit_Rate=statistics.stdev(percentageCorrectB
enefit_Rate)

averagePercentageRight = statistics.mean(percentageCorrectDemand)
DeviationaveragePercentageRight = statistics.stdev(percentageCorrectDemand)
```

```
averagepercentageCorrectLoad = statistics.mean(percentageCorrectLoad)
DeviationaveragepercentageCorrectLoad = statistics.stdev(percentageCorrectLoad)
PercentageSumAveragePopoularLoad=statistics.mean(SumAveragePopoularLoad)
DeviationPercentageSumAveragePopoularLoad=statistics.stdev(SumAveragePopoularLoad)
PercentageSumAverageRareLoad=statistics.mean(SumAverageRareLoad)
DeviationPercentageSumAverageRareLoad=statistics.stdev(SumAverageRareLoad)
workbook = xlsxwriter.Workbook('Total_data.xlsx')
worksheet = workbook.add_worksheet()
worksheet.write(0, 1, averagePercentageCorrect_Demand_Load)
worksheet.write(1, 1, DeviationaveragePercentageCorrect_Demand_Load)
worksheet.write(2, 1, averagepercentageCorrectBenefit_Rate)
worksheet.write(3, 1, DeviationaveragepercentageCorrectBenefit_Rate)
worksheet.write(4, 1, averagePercentageRight)
worksheet.write(5, 1, DeviationaveragePercentageRight)
worksheet.write(6, 1, averagepercentageCorrectLoad)
worksheet.write(7, 1, DeviationaveragepercentageCorrectLoad)
worksheet.write(8, 1, PercentageSumAveragePopoularLoad)
worksheet.write(9, 1, DeviationPercentageSumAveragePopoularLoad)
worksheet.write(10, 1, PercentageSumAverageRareLoad)
worksheet.write(11, 1, DeviationPercentageSumAverageRareLoad)

workbook.close()
```

```
print("------------------------")
print("Average percentage CorrectDemand_and_load",averagePercentageCorrect_Demand_Load)
print("Deviation Average percentage CorrectDemand_and_load",DeviationaveragePercentageCorrect_Demand_Load)
print("------------------------")
print("Average percentage Correct Benefit_Rate",averagepercentageCorrectBenefit_Rate)
print("Deviation Average percentage Correct Benefit_Rate",DeviationaveragepercentageCorrectBenefit_Rate)
print("------------------------")
print("Average percentage CorrectDemand",averagePercentageRight)
print("Deviation Average percentage CorrectDemand",DeviationaveragePercentageRight)
print("------------------------")
print("Average percentage Correct Load ",averagepercentageCorrectLoad)
print("Deviation Average percentage Correct Load ",DeviationaveragepercentageCorrectLoad)
print("------------------------")
print("Pergentage Popoular Tasks",PercentageSumAveragePopoularLoad)
print("Deviation Pergentage Popoular Tasks",DeviationPercentageSumAveragePopoularLoad)
print("------------------------")
print("Pergentage Rare Tasks",PercentageSumAverageRareLoad)
print("Deviation Pergentage Rare tasks",DeviationPercentageSumAverageRareLoad)
print("------------------------")
```