

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ

**“ΒΑΘΙΑ ΜΑΘΗΣΗ ΚΑΙ ΕΦΑΡΜΟΓΗ ΤΗΣ
ΣΕ ΕΡΓΑΣΙΕΣ ΓΛΩΣΣΙΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ”**

Ιωάννης Κιβρακίδης

Π2015087

Επιβλέπουσα καθηγήτρια
Κάτια Λήδα Κερμανίδου

Μάιος 2019

Επιβλέπουσα

Ονοματεπώνυμο: Κάτια Λήδα Κερμανίδου

Βαθμίδα: Επίκουρη Καθηγήτρια

Ίδρυμα: Τμήμα Πληροφορικής Ιονίου Πανεπιστημίου

Τριμελής Επιτροπή

Ονοματεπώνυμο: Κάτια Λήδα Κερμανίδου

Βαθμίδα: Επίκουρη Καθηγήτρια

Ίδρυμα: Τμήμα Πληροφορικής Ιονίου Πανεπιστημίου

Ονοματεπώνυμο: Μιχαήλ Στεφανιδάκης

Βαθμίδα: Επίκουρος Καθηγητής

Ίδρυμα: Τμήμα Πληροφορικής Ιονίου Πανεπιστημίου

Ονοματεπώνυμο: Φοίβος Μυλωνάς

Βαθμίδα: Αναπληρωτής Καθηγητής

Ίδρυμα: Τμήμα Πληροφορικής Ιονίου Πανεπιστημίου

Περίληψη

Η βαθιά μάθηση είναι ένα νέο πεδίο της τεχνητής νοημοσύνης που βασίζεται στα τεχνικά νευρωνικά δίκτυα. Τα τεχνικά νευρωνικά δίκτυα έχουν μοναδικές ιδιότητες που επιτρέπουν στα μοντέλα βαθιάς μάθησης να λύνουν εργασίες που τα μοντέλα της μηχανικής μάθησης δεν θα μπορούσαν να λύσουν ποτέ. Στην παρούσα διπλωματική εργασία παρουσιάζονται αναλυτικά τα δομικά συστατικά της βαθιάς μάθησης μαζί με κάποιες από τις βασικότερες αρχιτεκτονικές της. Επιπλέον, κατασκευάζεται ένα πραγματικό μοντέλο βαθιάς μάθησης με χρήση του keras API και tensorflow frame work, το οποίο έχει φτιαχτεί για να αναγνωρίζει χειρόγραφους χαρακτήρες από εικόνες. Το πειραματικό κομμάτι επεκτείνεται και στο πεδίο των Γλωσσικών επιστημών, αφού γίνεται αναφορά στα μοντέλα μετάφρασης κειμένων και αξιολόγηση αυτών με τις μετρικές απόδοσης WER, METEOR, TER. Τέλος, παρουσιάζονται τα world embeddings και παρέχεται κώδικας που αντικαθιστά τον παραδοσιακό τρόπο υπολογισμού τους με χρήση Pre trained embeddings στα Ελληνικά.

Abstract

Deep Learning is a very young field of artificial intelligence based on artificial neural networks. Artificial neural networks have unique capabilities that enable Deep learning models to solve tasks that machine learning models could never solve. In this dissertation, are presented in great detail the structure elements of deep learning together with some of the most important architectures. In Addition, a deep learning model is created by using the Keras API and Tensor flow frame work, which is made to recognize hand written digits from pictures. The experimental part expands to the field of linguistic sciences, because of the reference to the text translation models and evaluation of them by the NLP metrics WER, METEOR, TER. Last but not least, there is a presentation of word embeddings together with code that replaces the traditional way of calculating them by using pre trained embeddings in Greek.

Πίνακας Περιεχομένων

Α΄ ΕΙΣΑΓΩΓΗ.....	6
Α΄.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ.....	6
Α΄.2 ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ.....	8
Α΄.3 ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ	10
Α΄.4 ΒΑΘΙΑ ΜΑΘΗΣΗ.....	11
Β΄ ΤΑ ΔΟΜΙΚΑ ΣΥΣΤΑΤΙΚΑ ΤΗΣ ΒΑΘΙΑΣ ΜΑΘΗΣΗΣ.....	14
Β΄.1 ΤΟ PERCEPTRON	14
Β΄.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ	19
Β΄.3 ΝΕΥΡΩΝΙΚΟ ΔΙΚΤΥΟ ΠΟΛΛΩΝ ΕΠΙΠΕΔΩΝ.....	21
Γ΄ ΔΙΑΔΙΚΑΣΙΑ ΜΑΘΗΣΗΣ ΚΑΙ ΕΚΠΑΙΔΕΥΣΗΣ.....	22
Γ΄.1 ΑΛΓΟΡΙΘΜΟΣ ΟΠΙΣΘΟΔΙΑΔΟΣΗΣ ΤΟΥ ΣΦΑΛΜΑΤΟΣ.....	22
Γ΄.1.1 ΕΙΣΑΓΩΓΗ.....	22
Γ΄.1.2 ΥΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ΛΑΘΟΥΣ	23
Γ΄.1.3 ΛΕΠΤΟΜΕΡΗΣ ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ.....	25
Δ΄ ΠΕΙΡΑΜΑΤΙΚΟ ΚΟΜΜΑΤΙ.....	28
Δ΄.1 ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ KERAS API ΚΑΙ ΤΟΥ FRAME WORK TENSOR FLOW	28
Δ΄.2 ΔΗΜΙΟΥΡΓΙΑ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ ΓΙΑ ΑΝΑΓΝΩΡΙΣΗ ΧΑΡΑΚΤΗΡΩΝ	30
Δ΄.2.1 ΕΙΣΑΓΩΓΗ.....	30
Δ΄.2.2 ΔΗΜΙΟΥΡΓΙΑ ΠΕΙΡΑΜΑΤΟΣ ΜΕ ΑΝΑΦΟΡΕΣ ΣΤΟΝ ΚΩΔΙΚΑ	30
Δ΄.3 ΥΠΟΛΟΓΙΣΜΟΣ ΜΕΤΡΙΚΩΝ ΑΠΟΔΟΣΗΣ ΜΟΝΤΕΛΩΝ ΜΕΤΑΦΡΑΣΗΣ ΚΕΙΜΕΝΩΝ.....	38
Δ΄.3.1 ΕΙΣΑΓΩΓΗ.....	38
Δ΄.3.2 TER.....	39
Δ΄.3.3 METEOR.....	41
Δ΄.3.4 WER	45
Δ΄.4 ΧΡΗΣΗ PRE-TRAINED EMBEDDINGS ΣΕ ΚΩΔΙΚΑ	48
Ε΄ ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ.....	54
Ε΄.1 ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΠΟ ΤΟΝ ΥΠΟΛΟΓΙΣΜΟ ΤΩΝ ΜΕΤΡΙΚΩΝ.....	54
Ε΄.2 ΜΕΛΛΟΝΤΙΚΕΣ ΒΕΛΤΙΩΣΕΙΣ ΤΟΥ ΜΟΝΤΕΛΟΥ ΑΝΑΓΝΩΡΙΣΗΣ ΧΑΡΑΚΤΗΡΩΝ	58

Η΄ ΠΑΡΑΡΤΗΜΑ ΓΙΑ ΚΩΔΙΚΕΣ PYTHON.....	60
Η΄.1 ΚΩΔΙΚΕΣ WER.....	60
Η΄.2 ΚΩΔΙΚΕΣ TER.....	61
Z΄ ΒΙΒΛΙΟΓΡΑΦΙΑ.....	62
Z΄.1 ΒΙΒΛΙΑ	62
Z΄.2 ΙΣΤΟΣΕΛΙΔΕΣ.....	62

Α΄ ΕΙΣΑΓΩΓΗ

Α΄.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Η τεχνητή Νοημοσύνη έχει τις ρίζες της σε δυο φιλοσοφικές ιδέες του σπουδαίου φιλόσοφου και μαθηματικού του δέκατου εβδόμου αιώνα Gottfried Leibniz , την *Characteristica Universalis* και την *Calculus ratiocinator*. Πιο αναλυτικά, η *Characteristica Universalis* είναι μια ιδανική γλώσσα, στην οποία θεωρητικά θα μπορούσε να μεταφραστεί ολόκληρη η επιστήμη. Θα είναι η γλώσσα στην οποία θα μεταφράζεται οποιαδήποτε φυσική γλώσσα (Ελληνικά, Αγγλικά κ.α), μετατρέποντάς την σε γλώσσα αγνής σημασίας, απαλλαγμένη από γλωσσολογικές τεχνικότητες. Αυτή η γλώσσα θα μπορέσει τότε να λειτουργήσει ως βάση για να εξηγήσει την λογική σκέψη, με ένα τρόπο τόσο ακριβή, που θα μπορούσε να φτιαχτεί μια μηχανή ικανή να την μιμηθεί. Το όνομα μιας τέτοιας μηχανής θα είναι *Calculus Ratiocinator*.

Υπάρχει μια αντιπαράθεση ανάμεσα στους ιστορικούς της φιλοσοφίας για το αν αυτό θα σήμαινε ότι θα έπρεπε να φτιαχτεί λογισμικό ή υλικό, αλλά αυτό είναι στην πραγματικότητα μια ασήμαντη ερώτηση αφού για να καταλάβουμε την διαφορά θα πρέπει να κατανοήσουμε την ιδέα μιας γενικής μηχανής που θα δέχεται διαφορετικές οδηγίες για διαφορετικά έργα. Αυτή ιδέα θα έρθει από τον Alan Turing το 1936, αλλά θα γίνει κατανοητή σε μια ευρύτερη επιστημονική κοινότητα μόνο στα τέλη του 1970 με την έλευση του προσωπικού υπολογιστή. Η Φιλοσοφία όμως δεν ήταν η μόνη που ασχολήθηκε με την τεχνητή νοημοσύνη. Επιπλέον επιστήμες όπως τα Μαθηματικά, τα Οικονομικά, οι Νευροεπιστήμες, η Ψυχολογία και η Γλωσσολογία ασχολήθηκαν και αυτές με το πώς λειτουργεί η Νοημοσύνη είτε σε φυσικό είτε σε θεωρητικό επίπεδο, με το πως πραγματοποιείται η λήψη μιας απόφασης ή η κατανόηση ενός ακουστικού μηνύματος και με την οπτική αναγνώριση ενός αντικειμένου. Πρώτος ο Έλληνας Φιλόσοφος Αριστοτέλης (384- 322 π.Χ.) διατύπωσε ένα ακριβές σύνολο νόμων που διέπουν το ορθολογικό μέρος της νόησης. Πιο συγκεκριμένα, ανέπτυξε ένα άτυπο σύστημα συλλογισμών ώστε να επιτευχθεί σωστή συλλογιστική, το οποίο θεωρητικά επέτρεπε να παράγει κανείς μηχανικά συμπεράσματα με δεδομένες ορισμένες αρχικές υποθέσεις. Ο Αριστοτέλης θεωρείται ο πατέρας του πρώτου αλγορίθμου που 2300 χρόνια αργότερα αναπτύχθηκε από τους Newell και Simon. Ο μαθηματικός George Boole (1815-1864), στο βιβλίο του με τίτλο *Laws of Thought* που δημοσιεύτηκε το 1854, παρουσίαζε συστηματικά την λογική ως ένα σύστημα από τυπικούς κανόνες που αποδείχτηκε ότι ήταν ένα μεγάλο βήμα στην αναδιαμόρφωση της λογικής ως επίσημη επιστήμη. Ο Alan Turing το 1950 στην εργασία του με τίτλο *Computing Machinery and Intelligence* θέτει τις βάσεις της τεχνητής νοημοσύνης με την παρουσίαση του περίφημου “turing test”, για να καθορίσει αν ένας υπολογιστής μπορεί να θεωρηθεί έξυπνος. Πιο συγκεκριμένα, το “test turing” είναι ένα τεστ σε φυσική γλώσσα που χορηγείται σε έναν άνθρωπο (ο οποίος λαμβάνει τον ρόλο ενός κριτή). Ο κριτής επικοινωνεί με έναν άνθρωπο και με έναν υπολογιστή για πέντε λεπτά, αν ο κριτής δεν μπορεί να διακρίνει τους δύο, τότε ο υπολογιστής περνάει το τεστ και μπορεί αυτομάτως να θεωρηθεί ως έξυπνος.

Για πρώτη φορά το 1943 ο νευροφυσιολόγος Warren McCulloch, και ο μαθηματικός Warren Pitts, στην εργασία τους με τίτλο “A Logical Calculus of the ideas immanent in nervous activity” (Ένας Λογικός Λογισμός των ιδεών που εμφανίζονται σε νευρική δραστηριότητα) αναπαρέστησαν, με βάση τις γνώσεις της φυσιολογίας των ανθρώπινων Νευρώνων, ένα μοντέλο τεχνητών Νευρώνων, όπου ο κάθε νευρώνας είναι ενεργός ή μη. Το 1949 ο Hebb παρουσίασε τον πρώτο κανόνα αυτο-οργανούμενης μάθησης, η οποία είναι γνωστή και ως μάθηση Hebb. Το 1951 δυο μεταπτυχιακοί φοιτητές του πανεπιστημίου Princeton, ο Marvin Minsky και ο Dean Edmonds κατασκεύασαν τον πρώτο υπολογιστή Νευρωνικού δικτύου SNARC ο οποίος χρησιμοποιούσε 3000 λυχνίες κενού και

ένα μηχανισμό αυτόματου πιλότου από βομβαρδιστικό B-24 για να προσομοιώσει ένα δίκτυο 40 Νευρώνων.

Το 1957 δημιουργείται από τον McCarthy η ονομασία Τεχνητή Νοημοσύνη σε μία συνάντηση επιστημόνων στο Dartmouth. Το ίδιο έτος γεννήθηκε και η Natural Language Processing (NLP) ή επεξεργασία φυσικής γλώσσας. Ένα χρόνο μετά τη γέννηση του όρου Τεχνητή νοημοσύνη το 1958 ο Frank Rosenblatt παρουσιάζει το δίκτυο Perceptron, ένα βελτιωμένο μοντέλο των McCulloch – Pitts. Παρόλα αυτά ο αλγόριθμος εκπαίδευσης Perceptron είχε περιορισμένες δυνατότητες που λύθηκαν με την εισαγωγή του αλγόριθμου Back Propagation. Το 1974 προτάθηκε από τον Paul Werbos στην εργασία του “P. J. Werbos. Beyond Regression: ‘New Tools for Prediction and Analysis in the Behavioral Sciences. PhD thesis, Harvard University”, ο αλγόριθμος Back Propagation στα πλαίσια της ανάλυσης μοντέλων οικονομικής και πολιτικής πρόβλεψης. Αν και δεν είχε σχεδιαστεί για τις ανάγκες νευρωνικών δικτύων, την δεκαετία του 80 έγινε αντιληπτός και μεταφέρθηκε αυτούσιος στην εκπαίδευση νευρωνικών δικτύων, όπου αποτελεί πλέον την πιο διαδεδομένη μέθοδο για την εκπαίδευση των νευρωνικών δικτύων. Τη δεκαετία του 1980 παρουσιάζεται το μοντέλο Perceptron πολλών στρωμάτων (Multi-Layer Perceptron ή MLP) σε συνδυασμό με τον αλγόριθμο εκπαίδευσης Back Propagation. Αυτό είχε ως αποτέλεσμα το δίκτυο Νευρώνων να αποκτήσει απεριόριστες δυνατότητες αναπαράστασης συναρτήσεων και διαχωρισμού κλάσεων. Ωστόσο, η απόδοση των ηλεκτρονικών υπολογιστών την περίοδο εκείνη καθιστούσε απαγορευτική τη δημιουργία και εκμετάλλευση τέτοιων μοντέλων, με αποτέλεσμα για 20 τουλάχιστον χρόνια να μην πάρει τις ερευνητικές και πρακτικές διαστάσεις που έχει σήμερα το συγκεκριμένο πεδίο. Το 1990 ο Ολλανδός Guido van Rossum δημιούργησε μια υψηλού επιπέδου γλώσσα προγραμματισμού, την Python, στόχος της οποίας είναι η δημιουργία ευανάγνωστου κώδικα και η ευκολία χρήσης του. Το συντακτικό της επιτρέπει στους προγραμματιστές να εκφράσουν έννοιες σε λιγότερες γραμμές κώδικα απ’ ότι θα ήταν δυνατόν σε γλώσσες όπως η C ή C++ ή Java. Η Python διακρίνεται για το λόγο του ότι διαθέτει πολλές βιβλιοθήκες που διευκολύνουν αρκετά συνηθισμένες εργασίες και για την ταχύτητα εκμάθησής της. Η γλώσσα Python θεωρείται πλέον ως η γλώσσα της τεχνητής Νοημοσύνης που τείνει να αντικαταστήσει την γλώσσα προγραμματισμού LISP η οποία είχε δημιουργηθεί μόνο για το σκοπό αυτό. Η απλότητά της καθώς και το γεγονός ότι δεν ανήκει καθαρά στις γλώσσες αντικειμενοστραφούς προγραμματισμού ή συναρτησιακής λογικής, αλλά μπορεί να υλοποιήσει τέτοιου είδους προγράμματα, την καθιστούν πολύ ισχυρή γλώσσα που σε συνδυασμό με το πλήθος των βιβλιοθηκών της συναγωνίζεται άλλες γλώσσες όπως η JAVA, C, C++. Το 2011 η Google Brain Team δημιούργησε το DistBelief ως ιδιόκτητο σύστημα μηχανικής μάθησης που βασίζεται σε νευρωνικά δίκτυα βαθιάς μηχανικής μάθησης. Η χρήση του αυξήθηκε ταχύτατα σε ποικίλες εταιρείες της Alphabet (Μητρική εταιρία της Google), τόσο σε ερευνητικές όσο και σε εμπορικές εφαρμογές. Η Google ανέθεσε σε πολλούς επιστήμονες υπολογιστών, συμπεριλαμβανομένου του Jeff Dean, να απλοποιήσουν και να επαναπροσδιορίσουν τον κώδικα βάσης του DistBelief σε μια ταχύτερη και πιο στιβαρή βιβλιοθήκη εφαρμογών, η οποία έγινε TensorFlow. Το 2015 δόθηκε στο κοινό υπό την άδεια ανοικτού κώδικα Apache 2.0 ως βιβλιοθήκη της Python. Τέλος, το 2016 παρουσιάστηκε η έκδοση TensorFlow V1.0, που χρησιμοποιείται για ένα πλήθος εφαρμογών όπως την δημιουργία μοντέλων αναγνώρισης χειρόγραφων χαρακτήρων και την δημιουργία μοντέλων με εφαρμογές στις Γλωσσικές τεχνολογίες. Αυτές οι εφαρμογές μαζί με την αξιολόγηση συστημάτων μηχανικής μετάφρασης θα αποτελέσουν τον ερευνητικό χώρο για αυτή την πτυχιακή.

Α'2 ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Ο όρος τεχνητή νοημοσύνη (Artificial Intelligence) σχετίζεται με τον κλάδο της πληροφορικής και πιο συγκεκριμένα με την σχεδίαση υπολογιστικών συστημάτων, τα οποία μιμούνται την ανθρώπινη συμπεριφορά, υποδηλώνοντας παράλληλα την παρουσία στοιχειώδους ευφυΐας, καθώς και την υλοποίησή τους. Παραδείγματα αποτελούν η εκμάθηση και προσαρμογή στο περιβάλλον, η εξαγωγή συμπερασμάτων και γενικότερα η επίλυση απλών ή και πολύπλοκων προβλημάτων. Ο Τζον Μακάρθι, μια από τις σημαντικότερες φυσιογνωμίες στον τομέα της θεωρητικής πληροφορικής, ορίζει τον συγκεκριμένο κλάδο ως «η επιστήμη και η μεθοδολογία της δημιουργίας νοούντων μηχανών». Ωστόσο, δεν υπάρχει ένας καθολικός ορισμός που να περιγράφει τον όρο «τεχνητή νοημοσύνη». Στον παρακάτω πίνακα (Πίνακας 1) παρατίθενται οχτώ ορισμοί.

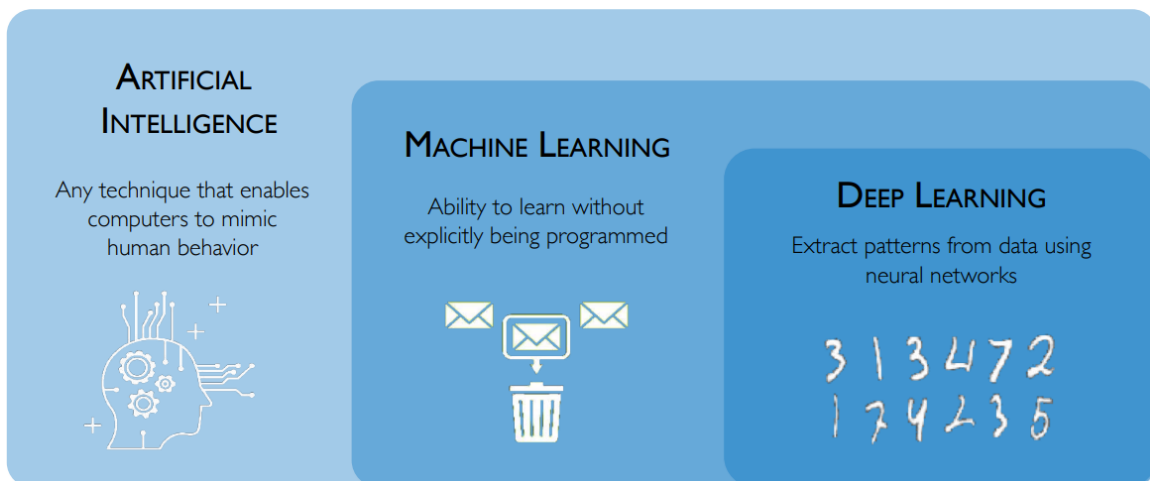
<u>Συστήματα που σκέπτονται σαν τον άνθρωπο.</u>	<u>Συστήματα που σκέπτονται ορθολογικά.</u>
« Η συναρπαστική νέα προσπάθεια για να κάνουμε τους υπολογιστές να σκέπτονται. » (Haugeland, 1985)	« Η μελέτη των νοητικών ικανοτήτων με τη χρήση υπολογιστικών μοντέλων. » (Charniak και McDermott, 1985)
« Η αυτοματοποίηση των δραστηριοτήτων που σχετίζονται με την ανθρώπινη σκέψη, όπως η λήψη αποφάσεων, η επίλυση προβλημάτων και η μάθηση. » (Bellman, 1978)	« Η μελέτη των υπολογιστικών εργασιών που μας δίνουν τη δυνατότητα να αντιλαμβανόμαστε και να ενεργούμε. » (Winston, 1992)
<u>Συστήματα που ενεργούν σαν τον άνθρωπο.</u>	<u>Συστήματα που ενεργούν ορθολογικά.</u>
« Η τέχνη της δημιουργίας μηχανών που πραγματοποιούν λειτουργίες που απαιτούν νοημοσύνη όταν πραγματοποιούνται από ανθρώπους. » (Kurzweil, 1990)	« Υπολογιστική νοημοσύνη είναι η μελέτη της σχεδίασης ευφυών πρακτόρων. » (Poole, 1998)
« Η μελέτη του πώς μπορούμε να κάνουμε τους υπολογιστές να κάνουν πράγματα, στα οποία, προς το παρόν, οι άνθρωποι είναι καλύτεροι. » (Rick και Knight, 1991)	« Η τεχνητή νοημοσύνη ασχολείται με την ευφυή συμπεριφορά των τεχνουργημάτων. » (Nilsson, 1998)

Πίνακας 1: Ορισμοί της τεχνητής νοημοσύνης(πηγή:users.sch.gr «εργαστήριο ρομποτικής»)

Οι ορισμοί αυτοί έχουν διαφορές σε δύο κύριες διαστάσεις. Οι επάνω ορισμοί εστιάζουν περισσότερο στις διαδικασίες σκέψης και στη συλλογιστική, ενώ οι κάτω σχετίζονται με τη συμπεριφορά. Οι ορισμοί στα αριστερά μετρούν την επιτυχία με βάση την εγγύτητα προς τις ανθρώπινες επιδόσεις, ενώ οι ορισμοί στα δεξιά τη μετρούν σε σχέση με μία ιδανική έννοια νοημοσύνης, η οποία ονομάζεται ορθολογικότητα. Ένα σύστημα είναι ορθολογικό αν κάνει «το σωστό», με δεδομένα όσα γνωρίζει.

Μεγάλος αριθμός επιστημών συνεισφέρει στην τεχνητή νοημοσύνη, όπως για παράδειγμα η επιστήμη της πληροφορικής, της ψυχολογίας και της φιλοσοφίας, η νευρολογία, η επιστήμη των μηχανών, ακόμη και η επιστήμη της γλωσσολογίας, προκειμένου να γίνει εφικτή η σύνθεση ευφυούς συμπεριφοράς, η εκμάθηση και η προσαρμογή στο εκάστοτε περιβάλλον μηχανών και υπολογιστών οι οποίοι προορίζονται για συγκεκριμένο συνήθως σκοπό. Η τεχνητή νοημοσύνη διαχωρίζεται σε δυο κομμάτια, την συμβολική και την υπο-συμβολική νοημοσύνη. Η πρώτη συνίσταται στην προσπάθεια εξομοίωσης της ανθρώπινης συμπεριφοράς με χρήση ειδικών αλγορίθμων, μέσα από ένα σύνολο συμβόλων και λογικών κανόνων υψηλού επιπέδου. Η δεύτερη στοχεύει στην προσέγγιση ή ακόμα και στην αναπαραγωγή της ανθρώπινης ευφυΐας, μέσα από στοιχειώδη αριθμητικά μοντέλα, τα οποία επαγωγικά συνθέτουν συμπεριφορές που υποδηλώνουν ευφυΐα, προσεγγίζοντας πραγματικές βιολογικές συμπεριφορές, όπως η διαδικασία της εξέλιξης και η λειτουργία του ανθρώπινου εγκεφάλου.

Η μεγαλύτερη υποκατηγορία της τεχνητής νοημοσύνης είναι η μηχανική μάθηση(Machine Learning) υποκατηγορία της οποίας είναι η βαθιά μάθηση(Deep Learning) όπως χαρακτηριστικά φαίνεται στον Πίνακα 2



Πίνακας 2:Κατηγορίες της ΤΝ (πηγή: MIT 6.S191)

Στην εποχή μας, η τεχνητή νοημοσύνη αποτελεί ένα από τα ταχύτερα εξελισσόμενα πεδία της επιστήμης, ενώ με την χρήση εργαλείων των εφαρμοσμένων μαθηματικών και επιστήμες μηχανικών έχει ξεφύγει από τα πλαίσια της θεωρητικής πληροφορικής. Τέλος, μελετάται από την ηλεκτρονική μηχανική, ενώ αποτελεί θεμέλιο του διεπιστημονικού πεδίου της γνωσιακής επιστήμης.

Α'3 ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

Η μηχανική μάθηση είναι μια υποκατηγορία της τεχνητής νοημοσύνης, η οποία αφορά αλγορίθμους και μεθόδους που επιτρέπουν τη “σταδιακή εκμάθησή” τους σε επιμέρους εργασίες. Με αυτόν τον τρόπο καθίσταται εφικτή η κατασκευή προσαρμόσιμων προγραμμάτων υπολογιστών τα οποία λειτουργούν με βάση την αυτοματοποιημένη ανάλυση συνόλων δεδομένων και όχι τη διαίσθηση ή τη γνώση των μηχανικών κατά τη στιγμή που τα προγραμματίσαν. Εξάλλου το 1959 ο σχεδιαστής παιχνιδιών Άρθουρ Σάμουελ (Arthur Samuel) όρισε ως μηχανική μάθηση *"Το πεδίο μελέτης όπου δίνει στους υπολογιστές την δυνατότητα να μαθαίνουν χωρίς να έχουν προγραμματιστεί"*. Η έννοια της Μηχανικής Μάθησης είναι στενά συνδεδεμένη και συνήθως επικαλύπτεται με το αντικείμενο της *Υπολογιστικής Στατιστικής* καθώς και οι δύο αυτοί κλάδοι καταπιάνονται με εφαρμογές όπως η ανάλυση και η “εξόρυξη” δεδομένων (*data mining*).

Οι εργασίες Μηχανικής Μάθησης χωρίζονται σε 3 βασικές κατηγορίες ανάλογα με τη φύση του ‘μαθησιακού σήματος’ ή/και με το εάν υπάρχει ανάδραση μέσω ενός μαθησιακού συστήματος, σε:

- Supervised Learning (Επιβλεπόμενη ή επιτηρούμενη Μάθηση): κατά την οποία είναι απαραίτητο να δίνονται κάποια δεδομένα εισόδου με τα αντίστοιχα επιθυμητά αποτελέσματα –δεδομένα εξόδου, στον Η/Υ. Με αυτό τον τρόπο ο υπολογιστής εκπαιδεύεται στο να μεταβαίνει από το κάθε νέο δεδομένο εισόδου στο αντίστοιχο αποτέλεσμα. Πιο συγκεκριμένα, η επιβλεπόμενη μάθηση βασίζεται στην υπόθεση ότι ένα σύνολο δεδομένων με ετικέτες (labels) είναι διαθέσιμο κατά την εκπαίδευση του νευρωνικού δικτύου.
- Unsupervised Learning (Μη Επιβλεπόμενη Μάθηση): κατά την οποία δεν παρέχονται δεδομένα εισόδου στο μαθησιακό σύστημα. Έτσι, ο υπολογιστής καλείται να μεταβεί μόνος του σε ένα ‘επιθυμητό αποτέλεσμα’ αναλύοντας ομοιότητες, ανομοιογένειες και πρότυπα από τα δεδομένα.
- Reinforcement Learning (Ενισχυμένη Μάθηση): κατά την οποία ένα υπολογιστικό πρόγραμμα επιδρά σε ένα δυναμικό περιβάλλον χωρίς να έχει καθοριστεί το πότε ή εάν βρίσκεται κοντά στον επιθυμητό στόχο.

Παραπάνω περιγράφονται οι γενικές κατηγορίες, ωστόσο υπάρχουν ενδιάμεσες και συνδυαστικές περιπτώσεις αλγορίθμων και εφαρμογών Μηχανικής Μάθησης (π.χ. Ημι – επιβλεπόμενη Μάθηση). Η βαθιά μάθηση είναι μια ειδική προσέγγιση της μηχανικής μάθησης που καλύπτει και τους τρεις παραπάνω κλάδους και επιδιώκει επίσης να τους επεκτείνει, με σκοπό την αντιμετώπιση των προβλημάτων της τεχνητής νοημοσύνης, τα οποία συνήθως δεν περιλαμβάνονται στη μηχανική μάθηση, όπως η συλλογιστική, ο σχεδιασμός κ.λπ.

Σήμερα, οι εφαρμογές αλγορίθμων Μηχανικής Μάθησης εφαρμόζονται σε πολλούς κλάδους της οικονομικής σφαίρας (οικονομικά, ιατρική, βιοπληροφορική, χρηματιστήριο, διαδικτυακές εφαρμογές κ.α.)

Α'.4 ΒΑΘΙΑ ΜΑΘΗΣΗ

Το deep learning είναι η νεότερη εξέλιξη στην οικογένεια αλγορίθμων που ονομάζονται “νευρωνικά δίκτυα”. Τα νευρωνικά δίκτυα ξεκίνησαν με τον αλγόριθμο Perceptron του Rosenblatt το 1957 και κατέληξαν σε σύγχρονες τεχνικές, ειδικά στον τομέα της οπτικής αναγνώρισης, στον οποίο, μεταξύ άλλων, πρωτοστάτησαν οι Geoffrey Hinton και Yann LeCun.

Οι αλγόριθμοι deep learning είναι από μαθηματική σκοπιά, προσεγγιστικές συναρτήσεις, οι οποίες προσπαθούν να ανακαλύψουν μια σχέση ανάμεσα σε αριθμητικές περιγραφές ορισμένων αντικειμένων, και των ετικετών τους. Έτσι, τα νευρωνικά δίκτυα αποτελούν μια ειδική περίπτωση του γενικότερου πλαισίου που είναι γνωστό ως “επιβλεπόμενη μάθηση” (Supervised learning) (Κεφάλαιο Α'.3). Για παράδειγμα, κάθε μια εικόνα σε μια ομάδα εικόνων μπορεί να αντιπροσωπεύεται αριθμητικά από έναν πίνακα pixels και μια ετικέτα, όπως “σκύλος” ή “γάτα”. Ενώ είναι πολύ εύκολο κάποιος να διαχωρίσει μια γάτα από ένα σκύλο, είναι πολύ δύσκολο να εκφράσει κανείς στη μορφή μίας λίστας από κανόνες (δηλαδή ενός αλγορίθμου) την διαδικασία που ακολουθεί ο εγκέφαλός του για να κάνει αυτήν την ταξινόμηση. Η μηχανική μάθηση λοιπόν η οποία βασίζεται στην “εκπαίδευση μέσω παραδείγματος” είναι μια επανάσταση στο χώρο του προγραμματισμού, καθώς αντικαθιστά τα παραδοσιακά “expert systems”, τα οποία βασίζονται σε προκατασκευασμένους κανόνες που ορίζονται από ειδήμονες ανθρώπους, κάτι μη εφικτό για δύσκολα προβλήματα ταξινόμησης. Ο στόχος της επιβλεπόμενης μάθησης επιτυγχάνεται μέσω της βελτιστοποίησης μιας πολυπαραμετρικής ευέλικτης συνάρτησης, η οποία θα πρέπει να μπορεί να αναπαράγει τις ετικέτες μιας σειράς ετικετοποιημένων δεδομένων (labelled dataset), γνωστή ως ομάδα δεδομένων εκπαίδευσης (training dataset). Ο αλγόριθμος ψάχνει μέσα στο χώρο όλων των πιθανών τιμών των παραμέτρων που είναι τα βάρη (Weights) και το κατώφλι (Threshold, Bias), με στόχο να προσδιορίσει τη μαθηματική συνάρτηση, δηλαδή μια συνάρτηση κόστους (Cost function), με μεγαλύτερη ακρίβεια ώστε να ταιριάζει με τα ετικετοποιημένα παραδείγματα και στη συνέχεια θα χρησιμοποιήσει αυτή τη συνάρτηση για να ταξινομήσει άγνωστες εικόνες. Τα νευρωνικά δίκτυα είναι μια μορφή επιβλεπόμενης μάθησης και έχουν κάποιες μοναδικές ιδιότητες. Είναι πολύ ευέλικτα στην καταρχήν διαδικασία εκπαίδευσης, όταν χρειάζεται να προσδιορίσουν, για παράδειγμα το σχήμα ενός σκύλου ως μια συνάρτηση από ενεργοποιήσεις (Activations) διαφορετικών pixels. Αυτή είναι και η ιδιότητα που δημιουργεί την ανάγκη των αλγορίθμων deep learning για ετικετοποιημένα δεδομένα, όπως θα δούμε και στη συνέχεια (Κεφάλαιο Δ). Πρακτικά, τα νευρωνικά δίκτυα είναι τόσο ευέλικτα, ώστε να μπορούν μαθηματικά να αποδώσουν την προσέγγιση οποιασδήποτε συνάρτησης και για να το πετύχουν αυτό χρησιμοποιούν μεγάλο αριθμό απλών συναρτήσεων που συνδέονται με ποικίλους τρόπους. Είναι δυνατό να συνδέονται σειριακά, ώστε το αποτέλεσμα ενός μετασχηματισμού να είναι εισερχόμενα δεδομένα για τον επόμενο μετασχηματισμό, ή παράλληλα, οπότε πολλοί μετασχηματισμοί εφαρμόζονται παράλληλα στα δεδομένα και το αποτέλεσμα προκύπτει συνδυαστικά.

Έτσι άλλωστε λειτουργεί και ο ανθρώπινος εγκέφαλος, ο οποίος επίσης κάνει πολύπλοκους υπολογισμούς, συνδυάζοντας απλές υπολογιστικές μονάδες, γνωστές ως νευρώνες. Ωστόσο, πρέπει να ξεκαθαρίσουμε ότι τα νευρωνικά δίκτυα δεν αποτελούν ένα μοντέλο του ανθρώπινου εγκεφάλου και οι σύγχρονες επιλογές αρχιτεκτονικών και συναρτήσεων έχουν σχεδιαστεί χωρίς καμία δέσμευση να αντικατοπτρίζουν τις πραγματικές δομές του ανθρώπινου εγκεφάλου.

Οι διαφορές των αλγορίθμων deep learning από παλαιότερες γενιές νευρωνικών δικτύων φαίνονται από τις συγκεκριμένες αρχιτεκτονικές που χρησιμοποιούν: τείνουν να εκτελούν πολλαπλά επίπεδα μετασχηματισμών. Για να θεωρηθεί ένα νευρωνικό δίκτυο “βαθύ” θα πρέπει να έχει τουλάχιστον δυο κρυφά επίπεδα και κάθε φορά να συνδυάζει τα δεδομένα εξόδου των προηγούμενων επιπέδων για να δημιουργήσουν ένα νέο επίπεδο. Αυτό τους επιτρέπει να εξάγουν περίπλοκες δομές με ιεραρχικό τρόπο. Για παράδειγμα μια εικόνα 28*28 pixel που έχει ως περιεχόμενο έναν χειρόγραφο αριθμό, είναι πιθανό αρχικά να διασπαστεί σε μια λίστα από καμπυλόγραμμα σχήματα. Αυτά στην

συνέχεια θα συνδυαστούν με άλλα επίπεδα για να δώσουν πιο πολύπλοκα μοτίβα, μέχρι να φτάσουν σε σχήματα όπως κύκλος, ημικύκλιο και ευθύγραμμο τμήματα που στο τέλος θα δημιουργήσουν έναν αριθμό από το μηδέν έως το εννιά (Κεφάλαιο Δ).

Τα ενδιάμεσα “επίπεδα” είναι γνωστά και ως κρυμμένα επίπεδα (hidden layers) διότι δεν χαίρουν επίβλεψης: δεν υπάρχουν ετικέτες που να επιβεβαιώνουν ή να διαψεύδουν τις προβλέψεις των ενδιάμεσων επιπέδων. Τα deep learning δίκτυα αναλαμβάνουν λοιπόν εξολοκλήρου μόνα τους να συνάγουν ποιες δομές είναι χρήσιμες στην συνολική ταξινόμηση των δεδομένων, και ποιες όχι, και να φροντίσουν να τις αναπαραστήσουν στα κρυμμένα επίπεδά τους. Με αυτόν τον τρόπο, κάνουν αυτόματα αυτό που οι Data Scientists χρειάζεται ακόμη και σήμερα να κάνουν χειροκίνητα: δηλαδή, το μετασχηματισμό raw data, σε δομές που μπορούν ευκολότερα να συσχετιστούν με μια ετικέτα από τον πραγματικό κόσμο, όπως “σκύλος” ή “γάτα”. Έτσι, ένα αδόμητο σύνολο ανθρωπιστικών δεδομένων, όπως για παράδειγμα μια εικόνα ή ένα κομμάτι κειμένου, ή ένα μουσικό κομμάτι μετατρέπεται σε μια λίστα από κύκλους που περιέχονται στην εικόνα ή επιστημονικούς όρους που περιέχονται στο κείμενο ή ήχους στο μουσικό κομμάτι αντίστοιχα. Αυτή η διαδικασία διαφοροποιεί τους αλγόριθμους deep learning από απλούς αλγόριθμους εκπαίδευσης, επιτρέποντας στους πρώτους να ξεκινούν από raw data χωρίς καμία επεξεργασία από άνθρωπο. Ίσως λοιπόν βρισκόμαστε στο πρώτο στάδιο της διαδρομής προς μια Γενική Τεχνητή Νοημοσύνη, η οποία θα έχει τη δυνατότητα να μαθαίνει μόνη της, χωρίς προηγουμένως να της έχουμε ορίσει κάποιο συγκεκριμένο πεδίο γνώσης.

Όπως προαναφέραμε, τα ετικετοποιημένα δεδομένα, είναι απαραίτητα για όλους τους αλγόριθμους εποπτικής μάθησης. Είναι ακριβώς το ίδιο, με τη διαδικασία που ένας ενήλικας δείχνει σκύλους και γάτες σε ένα παιδί μέχρι αυτό να αρχίσει να τα ξεχωρίζει μόνο του. Αυτός είναι και ο λόγος που τα δεδομένα αυτά είναι γνωστά ως “δεδομένα εκπαίδευσης”(Training Data).

Οι αλγόριθμοι deep learning καταναλώνουν τεράστιες ποσότητες ετικετοποιημένων δεδομένων διότι η ευελιξία τους συνεπάγεται ότι περιέχουν ένα τεράστιο αριθμό παραμέτρων που επιτρέπουν την εκμάθηση δύστροπων σχέσεων ετικέτας και αντικειμένου. Η συγκεκριμένη ευελιξία οδηγεί συχνά σε ένα φαινόμενο που είναι γνωστό ως “overfitting”, με αποτέλεσμα να ανακαλύπτονται μοτίβα που συμπτωματικά περιγράφουν σωστά κάποια παραδείγματα, αλλά αδυνατούν να δώσουν καλές προβλέψεις. Για παράδειγμα, εάν δώσουμε σε έναν αλγόριθμο deep learning algorithm δέκα παραδείγματα “γάτων και σκύλων”, ίσως αντιληφθεί ότι η παρουσία ενός κόκκινου pixel στην πάνω δεξιά γωνία μιας εικόνας, διαχωρίζει τη γάτα από το σκύλο, ενώ στην πράξη μπορεί απλά αυτές οι φωτογραφίες να έχουν αποτυπωθεί σε ένα κόκκινο φόντο. Ως γενικός κανόνας στη στατιστική μοντελοποίηση και στο machine learning, η ευελιξία ενός μοντέλου είναι άμεσα συσχετισμένη με έναν αριθμό ελεύθερων παραμέτρων, οι οποίες με τη σειρά τους είναι άμεσα συσχετισμένες με τον απαραίτητο αριθμό ετικετοποιημένων παραδειγμάτων που απαιτούνται για την εκπαίδευση.

Οι αλγόριθμοι deep learning είναι εξαιρετικά απαιτητικοί σε υπολογισμούς. Αυτός είναι και ο βασικός λόγος που ο τομέας πέρασε μια περίοδο, η οποία είναι γνωστή ως “AI winter”, στη διάρκεια του οποίου οι επιχειρήσεις απογοητεύτηκαν από τις πρακτικές εφαρμογές των νευρωνικών δικτύων. Αυτή η περίοδος τερματίστηκε με την εμφάνιση δύο νέων τεχνολογιών: των Big Data και του Internet που αύξησε μαζικά τις ετικετοποιημένες ομάδες δεδομένων. Η αποκορύφωση της εξέλιξης ήρθε μέσα από μια κατηγορία επεξεργασιών που είναι γνωστή ως Μονάδες Επεξεργασίας Γραφικών GPU και έχουν την ιδιότητα να επιταχύνουν με εκθετικό ρυθμό τη διαδικασία εκμάθησης, αξιοποιώντας τις ευκαιρίες παράλληλης επεξεργασίας που προσφέρει η αρχιτεκτονική των νευρωνικών δικτύων.

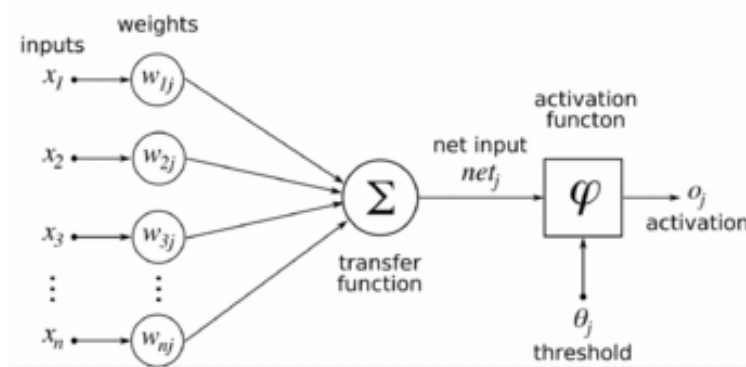
Οι νέες GPU ενσωματώνουν σε επίπεδο hardware πολλές από τις ρουτίνες υπολογισμών και έτσι μειώνεται η ανάγκη μεταφοράς δεδομένων μεταξύ διαφορετικών τμημάτων του υπολογιστή.

Πρακτικά, οι συγκεκριμένοι επεξεργαστές δε συμβάλουν στη βελτίωση της ακρίβειας των μοντέλων, απλά επιταχύνουν τη διαδικασία εκμάθησης. Ωστόσο αυτή είναι σημαντική βοήθεια για τους data scientists, γιατί τους επιτρέπει να δοκιμάζουν περισσότερες παραλλαγές του αλγορίθμου, καθώς και να βελτιστοποιούν τα νευρωνικά τους δίκτυα περισσότερο σε ένα δεδομένο χρονικό πλαίσιο, δυνατότητες οι οποίες τελικά βελτιώνουν και την ακρίβεια.

Β' ΤΑ ΔΟΜΙΚΑ ΣΥΣΤΑΤΙΚΑ ΤΗΣ ΒΑΘΙΑΣ ΜΑΘΗΣΗΣ

Β'.1 ΤΟ PERCEPTRON

Η εξέλιξη του τεχνητού νευρώνα (Artificial neuron) πέρασε από αρκετά στάδια μέχρι να πάρει την τελική του μορφή. Οι ρίζες του είναι σταθερά γειωμένες εντός της νευρολογικής δουλειάς που έγινε πρώτα από τους Santiago Ramon y Cajal και Sir Charles Scott. Ο Ramon y Cajal ήταν ένα διακεκριμένο πρόσωπο που ωφέλησε στην εξερεύνηση της δομής του νευρικού ιστού και έδειξε ότι παρά την ικανότητα τους να επικοινωνούν, οι νευρώνες είναι φυσικά διαχωρισμένοι. Έχοντας πλέον μεγαλύτερη κατανόηση των βασικών στοιχείων του εγκεφάλου, έγιναν προσπάθειες να περιγράψουν τον τρόπο που αυτοί οι βασικοί νευρώνες μπορούν να οδηγήσουν στις έκδηλες ανθρώπινες συμπεριφορές, στις οποίες ο William James συνεισέφερε σημαντικά. Δουλεύοντας από την αρχή της Νευροεπιστήμης, οι Warren McCulloch και Walter Pitts στην εργασία τους το 1943 που είχε τίτλο “A Logical Calculus of ideas Immanent in Nervous Activity,” υποστήριξαν ότι οι νευρώνες με δυαδική συνάρτηση ενεργοποίησης κατωφλιού ήταν ανάλογες με τις προτάσεις λογικής πρώτης τάξης. Ο βασικός νευρώνας των McCulloch και Pitts έμοιαζε με την παρακάτω Εικόνα(1):



Εικόνα 1:Ο νευρώνας McCulloch-Pitts (όλα τα βάρη στην πρώιμη αυτή μορφή είχαν τιμή 1)

(πηγή:Michigan State University)

Ο νευρώνας των McCulloch-Pitts λειτουργούσε με την εισαγωγή 1 ή 0 για κάθε μια από τις εισόδους (Inputs:χ), όπου το 1 αναπαρίστανε Αληθές και το 0 αναπαρίστανε Ψευδές. Ταυτόχρονα αν δινόταν στο κατώφλι (Threshold:θ) μια πραγματική τιμή, όπως για παράδειγμα το 2 , τότε θα έδινε μια έξοδο 1 (Activation:ο), αν το αλγεβρικό άθροισμα των τιμών των εισόδων(Transfer function : $\sum_1^n x_n$) ταυτιζόταν ή ξεπερνούσε αυτή την τιμή και μια έξοδο 0 στην αντίθετη περίπτωση. Έτσι, προκειμένου να αναπαρασταθεί η λογική σύζευξη “και”, θέτουμε: το κατώφλι με τον αριθμό 2 και τον αριθμό των εισόδων με 2 (x_1, x_2). Αν τα κάνουμε αυτά προκύπτει ο παρακάτω πίνακας αληθείας (Εικόνα 2)

Input x_1	Input x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Εικόνα 2:Πίνακας αληθείας για την σύζευξη

Αυτός ο πίνακας δείχνει το λογικό “και” έτσι ώστε, αν οι είσοδοι x_1 και x_2 είναι και οι δύο ψευδείς, τότε και η έξοδος από τον συνδυασμό αυτών θα είναι επίσης ψευδής. Παρομοίως ,αν το x_1 είναι αληθές δηλαδή ίσο με 1 και το x_2 είναι και αυτό αληθές δηλαδή ίσο με 1 ,τότε θα εξισωθεί με το κατώφλι που του είχαμε θέσει την τιμή 2, με αποτέλεσμα η έξοδος να είναι αληθής.

Αυτό ισχύει και για την λογική διάζευξη “ή” ,αν αλλάξουμε το κατώφλι από 2 σε 1 .Ο πίνακας αληθείας που προκύπτει είναι (Εικόνα 3)

Input x_1	Input x_2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Εικόνα 3:Πίνακας αληθείας για την διάζευξη

Αυτός ο τύπος τεχνητού νευρώνα θα μπορούσε να χρησιμοποιηθεί για να λύσει και την λογική άρνηση “όχι”, η οποία θα είχε μόνο μία είσοδο, όπως και για τους συνδυασμούς της “όχι και” και “όχι ή”. Ο νευρώνας των McCulloch-Pitts ήταν συνεπώς πολύ χρήσιμος για την εξελικτική πορεία του τεχνητού νευρώνα, αλλά είχε κάποιους σοβαρούς περιορισμούς. Πιο συγκεκριμένα, δεν θα μπορούσε να είχε λύσει καμία από τις συναρτήσεις “αποκλειστικό ή” (XOR) και “αποκλειστικό όχι ή”(XNOR). Ο νευρώνας των McCulloch-Pitts περιορισμένος σε δυαδικό κώδικα, δεν θα μπορούσε να λύσει τους παρακάτω πίνακες αληθείας χρησιμοποιώντας αυτή την πρώιμη μορφή του(Εικόνα 4 και Εικόνα 5).

Input x_1	Input x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Εικόνα 4:Πίνακας αληθείας για το “αποκλειστικό ή”

Input x_1	Input x_2	Output
0	0	1
0	1	0
1	0	0
1	1	1

Εικόνα 5 :Πίνακας αληθείας για το “αποκλειστικό όχι ή”

Το κυριότερο πρόβλημα με τον νευρώνα McCulloch-Pitts ήταν η μεγάλη απλότητα του. Πιο συγκεκριμένα, επέτρεπε μόνο δυαδικές εισόδους και εξόδους, χρησιμοποιούσε μόνο τη συνάρτηση ενεργοποίησης κατωφλίου (Threshold step activation function) και δεν συμπεριλάμβανε την εισαγωγή βαρών στις διάφορες εισόδους.

Το 1949, ο Donald Hebb συνέβαλλε στην αλλαγή του τρόπου με τον οποίον γινόταν η κατανόηση των νευρωνικών δικτύων. Στο βιβλίο του με τίτλο, “The Organization of Behavior”, πρότεινε τον μέχρι σήμερα ισχύοντα νόμο του Hebb, ο οποίος αναφέρει ότι όταν ένας άξονας από ένα κύτταρο A είναι κοντά στο να διεγείρει ένα κύτταρο B και επαναλαμβανόμενα ή επίμονα λαμβάνει μέρος στο να πυροδοτήσει, κάποια διαδικασία ανάπτυξης ή μεταβολική αλλαγή θα λάβει μέρος σε ένα ή και στα δύο κύτταρα, έτσι ώστε η αποτελεσματικότητα του A ,ως ένα από τα κύτταρα που πυροδοτούν το B, να αυξηθεί”(“When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased.). Ο Hebb δεν πρότεινε μόνο ότι όταν δύο νευρώνες πυροδοτούνται μαζί η σύνδεση μεταξύ τους ενδυναμώνεται, αλλά επίσης ότι αυτή η δραστηριότητα είναι μια από τις βασικές λειτουργίες που είναι απαραίτητες για την μάθηση και την μνήμη.

Για τον τεχνητό νευρώνα, αυτό σήμαινε ότι ο νευρώνας των McCulloch-Pitts θα έπρεπε να μετατραπεί για να μπορεί τουλάχιστον να επιτρέπει αυτή την νέα βιολογική πρόταση. Η μέθοδος που χρησιμοποιήθηκε ήταν να προστίθεται βάρος σε κάθε μια από τις εισόδους. Έτσι, μια είσοδος με 1 θα μπορούσε να παίρνει περισσότερο ή λιγότερο βάρος , σχετικά με το συνολικό άθροισμα του κατωφλίου.

Ο Frank Rosenblatt, χρησιμοποιώντας τον νευρώνα των McCulloch-Pitts και τα ευρήματα του Hebb, ανέπτυξε το πρώτο Perceptron. Αυτό το Perceptron, που μαθαίνει με την εισαγωγή βαρών στις εισόδους, όπως πρότεινε ο Hebb, ήταν καθοριστικό για την μετέπειτα διαμόρφωση των νευρωνικών δικτύων. Το 1962 ο Frank Rosenblatt αναφέρθηκε στο Perceptron στο βιβλίο του, με τίτλο, “Principles of Neurodynamics”, μια αναπαράσταση του οποίου θα μπορούσε να φανεί στην Εικόνα 1(Εκεί τα βάρη(Weights:W) μπορούν να πάρουν τιμές διάφορες του 1).

Έστω ότι το Perceptron της Εικόνας 1 έχει συνολικά 5 εισόδους (inputs: $x_1 \dots x_5$) με κάθε μια να έχει ένα αντίστοιχο βάρος (weight: $w_1 \dots w_5$). Τα γινόμενα των εισόδων με τα αντίστοιχα βάρη τους αθροίζονται στον κόμβο μέσω της αθροιστικής συνάρτησης(1). Εάν το αποτέλεσμα της συνάρτησης αυτής γίνει ίσο ή μεγαλύτερο από το ορισμένο κατώφλι τότε το Perceptron ενεργοποιείται και προκύπτει μια έξοδος. Είναι σημαντικό το γεγονός ότι σε κάθε μια από τις εισόδους μπορεί να μην τους έχει δοθεί το ίδιο βάρος. Το Perceptron “μαθαίνει”, με το να δίνει μεγαλύτερα βάρη στις εισόδους οι οποίες έχουν σημαντικότερη επίδραση στο να το πυροδοτούν και αντίστοιχα για τα μικρότερα βάρη.

Παρακάτω δίδεται ο αθροιστικός τύπος(Transfer function) με N εισόδους ($x_1 \dots x_N$) και τα αντίστοιχα βάρη τους ($w_1 \dots w_N$):

$$NetInput = \sum_{j=1}^N X_j * W_j + \theta \quad (1)$$

Η συνάρτηση ενεργοποίησης(Activation function:φ) τότε γίνεται:

$$Y = \varphi(NetInput)$$

Η συνάρτηση ενεργοποίησης που χρησιμοποιήθηκε από τους McCulloch και Pitt ήταν η Threshold step function. Ωστόσο, άλλες συναρτήσεις που μπορούν να χρησιμοποιηθούν είναι η Sigmoid, Piecewise Linear και η Gaussian συναρτήσεις ενεργοποίησης. Αυτές οι συναρτήσεις φαίνονται σχηματικά παρακάτω.



Παρά τις πολλές αλλαγές που έγιναν στον αρχικό νευρώνα των McCulloch – Pitt, το Perceptron περιοριζόταν στο να λύνει μόνο συγκεκριμένες συναρτήσεις. Πράγμα που φάνηκε στην εργασία που δημοσίευσε το 1969 ο Minsky σε συνεργασία με τον Seymour Papert που είχε τίτλο ,

“Perceptrons: An Introduction to Computational Geometry”. Σε αυτή την εργασία επισήμαναν τους περιορισμούς του Perceptron, με το να δείξουν ότι το Perceptron μπορούσε να λύσει μόνο γραμμικά διαχωριζόμενες συναρτήσεις. Ιδιαίτερο ενδιαφέρον είχε το γεγονός ότι το Perceptron εξακολουθούσε να μην μπορεί να λύσει τις συναρτήσεις XOR και NXOR, γεγονός που έκανε τους Minsky και Papert να δηλώσουν ότι το στυλ της έρευνας που έγινε στο Perceptron ήταν καταδικασμένο να αποτύχει εξαιτίας αυτών των περιορισμών. Αυτό ήταν φυσικά μια λανθασμένη παρατήρηση από τον Minsky, που είχε ως αποτέλεσμα να γίνει πάρα πολύ μικρή έρευνα στον χώρο μέχρι περίπου το 1980.

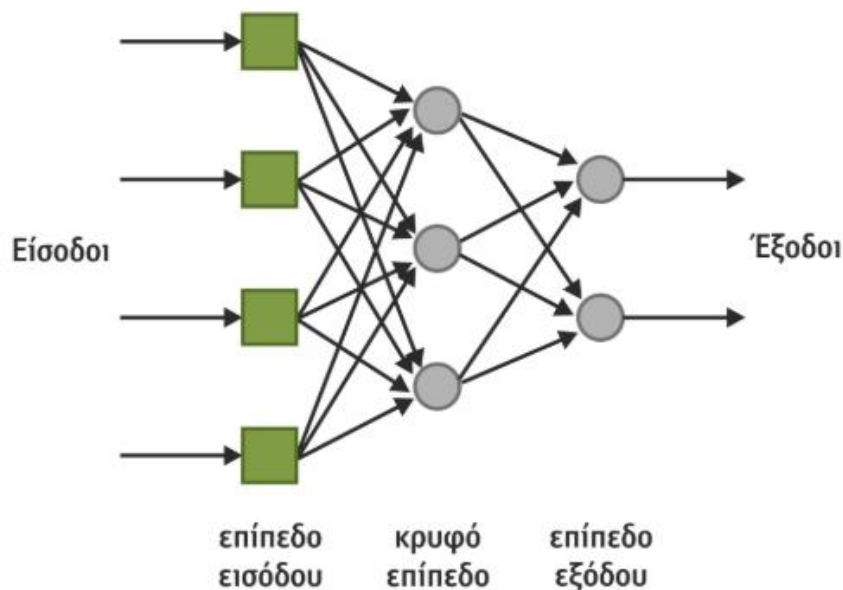
Αυτό που ήρθε για να επιλύσει πολλές από τις δυσκολίες ήταν η δημιουργία των νευρωνικών δικτύων. Αυτά τα δίκτυα συνδέουν τις εισόδους των τεχνικών νευρώνων με τις εξόδους άλλων τεχνικών νευρώνων με αποτέλεσμα να μεγαλώσουν σημαντικά σε πολυπλοκότητα και να είναι θέση να λύσουν πιο δύσκολα προβλήματα. Ωστόσο, πολλά από τα τεχνητά νευρωνικά δίκτυα που χρησιμοποιούνται σήμερα πηγάζουν από τις πρώιμες καταβολές του νευρώνα των McCulloch-Pitt και του Perceptron του Rosenblatt

Β.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ

Όσον αφορά το πώς είναι συνδεδεμένοι οι τεχνητοί νευρώνες μεταξύ τους, υπάρχουν δυο βασικές κατηγορίες Τεχνητών Νευρωνικών Δικτύων:

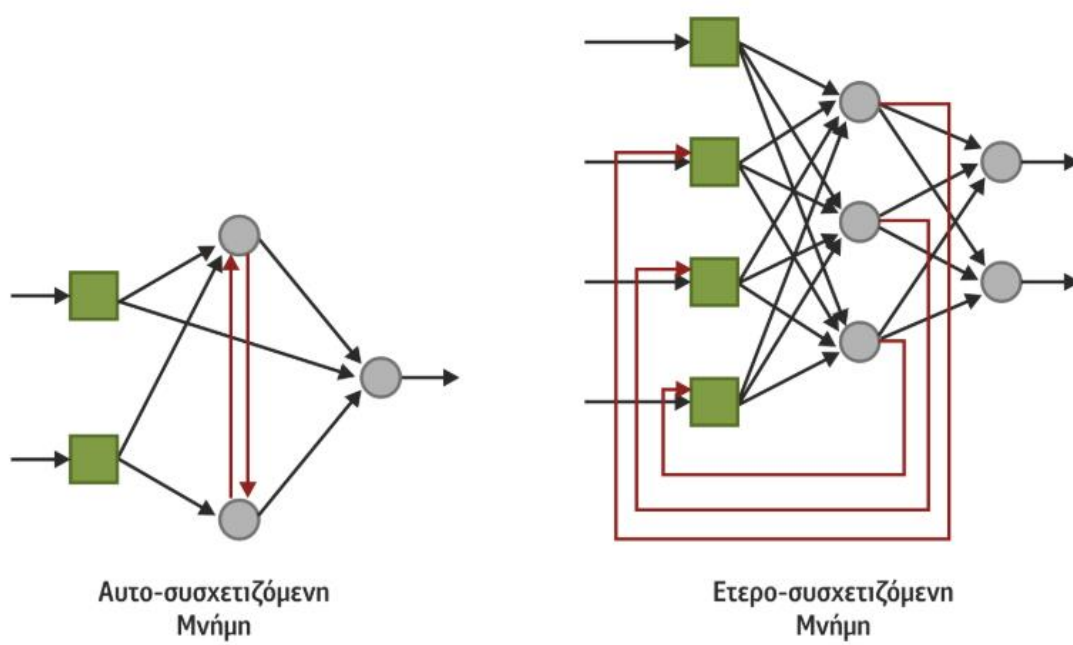
- **πρόσθιας τροφοδότησης** (feed forward)
- **οπίσθιας τροφοδότησης** (feed backward)

Στα νευρωνικά δίκτυα πρόσθιας τροφοδότησης, οι μονάδες είναι οργανωμένες σε διαφορετικά επίπεδα, ώστε οι μονάδες του ενός επιπέδου να τροφοδοτούν τις μονάδες του επόμενου επιπέδου, έως ότου τροφοδοτηθούν και οι μονάδες του τελευταίου επιπέδου (Εικόνα 1). Δηλαδή, δεν υπάρχει έξοδος μονάδας ενός επιπέδου που να αποτελεί είσοδο μονάδας του ίδιου ή προηγούμενων επιπέδων. Τέτοια ΤΝΔ είναι τα δίκτυα οπισθοδιάδοσης (backpropagation).



Εικόνα 1: Παράδειγμα ΤΝΔ πρόσθιας τροφοδότησης (πηγή: Κατερίνα Γεωργούλη)

Στα οπισθίως τροφοδοτούμενα δίκτυα, που καλούνται και **ανατροφοδοτούμενα ΤΝΔ** (recurrent ANN), επιτρέπεται στις μονάδες ενός επιπέδου να τροφοδοτούν και μονάδες του ίδιου επιπέδου ή και προηγούμενων επιπέδων. Αν η ανατροφοδότηση αφορά κόμβους στο ίδιο επίπεδο, τότε τα δίκτυα καλούνται **αυτοσυσχετιζόμενες μνήμες** (autoassociated memories) διαφορετικά καλούνται **ετεροσυσχετιζόμενες μνήμες** (heteroassociated memories) (βλέπε Εικόνα 2) .



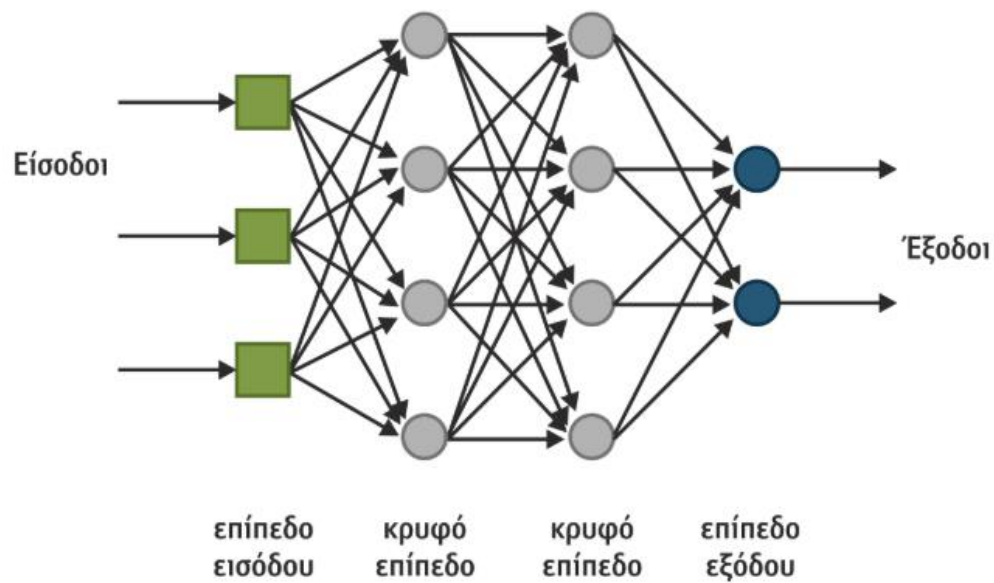
Εικόνα 2: Παράδειγμα ανατροφοδοτούμενων ΤΝΔ (πηγή: Κατερίνα Γεωργούλη)

Β.3 ΝΕΥΡΩΝΙΚΟ ΔΙΚΤΥΟ ΠΟΛΛΩΝ ΕΠΙΠΕΔΩΝ

Κοινό χαρακτηριστικό της δομής των πολυεπίπεδων ΤΝΔ είναι ότι διαθέτουν τουλάχιστον ένα κρυφό επίπεδο. Οι κόμβοι των διάφορων επιπέδων μπορεί να είναι πλήρως συνδεδεμένοι (fully connected), δηλαδή κάθε κόμβος του ενός επιπέδου συνδέεται με όλους τους κόμβους του επόμενου, όπως στην Εικόνα 3, ή μερικώς συνδεδεμένοι (partially connected),

Τα ΤΝΔ χαρακτηρίζονται, επιπλέον, με βάση τον τρόπο με τον οποίο είναι συνδεδεμένοι οι κόμβοι τους, όπως αναφέραμε στην προηγούμενη παράγραφο περί Αρχιτεκτονικής ΤΝΔ, δηλαδή αν είναι πρόσθιας τροφοδότησης (feed forward) ή οπίσθιας τροφοδότησης (feed backward ή recurrent).

Στην πλειοψηφία των εφαρμογών χρησιμοποιούνται δίκτυα πρόσθιας τροφοδότησης ενός κρυφού επιπέδου με πλήρως συνδεδεμένους κόμβους.



Εικόνα 3: Πολύ επίπεδο ΤΝΔ ενός κρυφού επιπέδου πρόσθιας τροφοδότησης 2 κρυμμένων επιπέδων με πλήρως συνδεδεμένους κόμβους (πηγή: Κατερίνα Γεωργούλη)

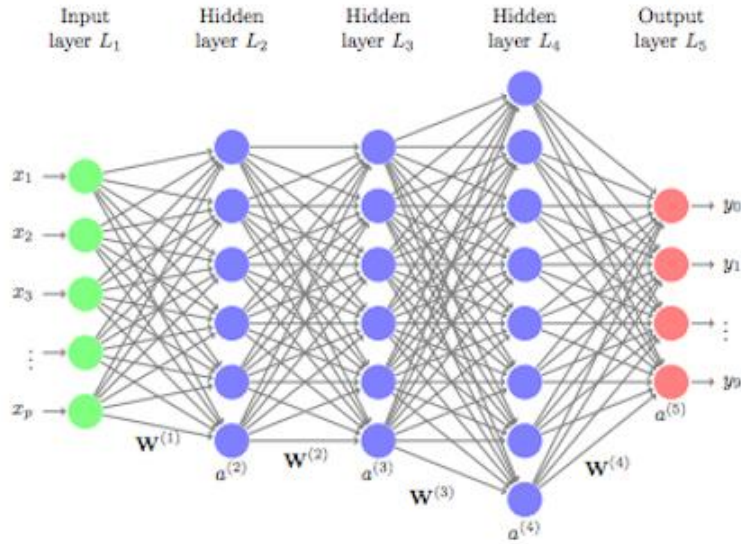
Γ' ΔΙΑΔΙΚΑΣΙΑ ΜΑΘΗΣΗΣ ΚΑΙ ΕΚΠΑΙΔΕΥΣΗΣ

Γ'.1 ΑΛΓΟΡΙΘΜΟΣ ΟΠΙΣΘΟΔΙΑΔΟΣΗΣ ΤΟΥ ΣΦΑΛΜΑΤΟΣ

Γ'.1.1 ΕΙΣΑΓΩΓΗ

Η μέθοδος οπισθοδιάδοσης του λάθους είναι η πιο δημοφιλής μέθοδος σήμερα για την εκπαίδευση ενός δικτύου που αποτελείται από πολλά επίπεδα, και έχει χρησιμοποιηθεί στις πιο πολλές εφαρμογές. Ιστορικά, πρώτα αναπτύχθηκαν δίκτυα ενός μόνο επιπέδου, όπως ο στοιχειώδης αισθητήρας (ΚΕΦ Β'.1), τα οποία όμως γρήγορα φάνηκε ότι έχουν μεγάλους περιορισμούς ως προς τις ικανότητες που είχαν, και έτσι σύντομα εγκαταλείφθηκαν. Έτσι φυσιολογικά ακολούθησαν τα δίκτυα πολλών επιπέδων που αναπτύχθηκαν αργότερα (ΚΕΦ Β'.3), και για τα οποία αρχικά δεν υπήρχαν θεωρητικοί τρόποι για την εκπαίδευσή τους, μέχρι που εμφανίστηκε η μέθοδος οπισθοδιάδοσης. Η μέθοδος αυτή αναπτύχθηκε ανεξάρτητα σε διάφορες παραλλαγές από τους Bryson και Ho (1969), P. Werbos (1974), D. Parker (1982), αλλά διαφημίστηκε πολύ και προωθήθηκε από το έργο "Parallel distributed processing" των D.E. Rumelhart και J.L. McClelland (1986), το οποίο άνοιξε πολλές εφαρμογές και νέα πεδία, και ανακίνησε μεγάλο ενδιαφέρον σε όλη την περιοχή των νευρωνικών δικτύων. Ως μέθοδος βασίζεται σε καθαρά μαθηματική θεώρηση με δικαιολογημένες αποδείξεις. Το νευρωνικό δίκτυο στο οποίο εφαρμόζεται είναι αρκετά πιο περίπλοκο από τον στοιχειώδη αισθητήρα. Είναι ένα δίκτυο πολλαπλών επιπέδων. Η κεντρική ιδέα είναι αρκετά απλή: το δίκτυο ξεκινά την διαδικασία μάθησης από τυχαίες τιμές των βαρών του. Εάν δώσει λάθος απάντηση (που είναι και το πιο πιθανό) τότε τα βάρη διορθώνονται έτσι ώστε το λάθος να γίνει μικρότερο. Η ίδια διαδικασία επαναλαμβάνεται πολλές φορές έτσι ώστε σταδιακά το λάθος ελαττώνεται μέχρις ότου γίνει πολύ μικρό και ανεκτό. Στο σημείο αυτό λέμε ότι το δίκτυο έχει μάθει τα παραδείγματα που του διδάξαμε με την ακρίβεια που θέλαμε να μάθει.

Γ.1.2 ΥΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ΛΑΘΟΥΣ



Εικόνα 1:Νευρωνικό δίκτυο με τρία κρυμμένα επίπεδα

Προκειμένου να γίνει πιο εύκολη η περιγραφή του υπολογισμού λάθους θεωρούμε το νευρωνικό δίκτυο της Εικόνας 1 το οποίο θα εκπαιδευτεί με 10 παραδείγματα μάθησης(για λόγους απλότητας) με κάθε παράδειγμα να συμβολίζεται E_j .Το δίκτυο αυτό δέχεται $x_1 \dots x_p$ εισόδους (Inputs) και έχει $y_0 \dots y_9$ πιθανές εξόδους(Outputs).

Αφού ορίσαμε το νευρωνικό δίκτυο είμαστε σε θέση να υπολογίσουμε το λάθος που προκύπτει για κάθε παράδειγμά μάθησης με την χρήση της συνάρτησης τετραγωνικού σφάλματος. Ο μαθηματικός τύπος που εκφράζει αυτή την συνάρτηση είναι

$$Error_{E_j} = \sum_{k=0}^9 (y_k - target_k)^2 \quad (1.0)$$

Με τον όρο y_k ($k:0..9$) ορίζονται οι τιμές των εξόδων που δίνει το δίκτυο ($0.0 \dots 1.0$) και με τον όρο $target_k$ ορίζονται οι τιμές που θέλουμε να πάρει ως έξοδο το δίκτυο σύμφωνα με το παράδειγμα E_k

Παράδειγμα:

Έστω ότι θέλουμε να υπολογίσουμε το λάθος για το παράδειγμα μάθησης E_1 για το οποίο έχουμε ως $target_3$ την έξοδο y_3 . Αφού βάλουμε ως είσοδο το παράδειγμα μάθησης E_1 το δίκτυο έβγαλε ως έξοδο τις τιμές y_k ($k:0...9$) από τις οποίες θα επιλέξει ως απάντηση αυτή με τον μεγαλύτερο αριθμό(0.0..1.0). Στο παράδειγμα μας η έξοδος που είχε τον μεγαλύτερο αριθμό και ως εκ τούτου είναι η επιλογή του δικτυού ήταν η y_4 . Στον παρακάτω πίνακα φαίνονται όλες οι τιμές των εξόδων και των στόχων που θέλαμε για το παράδειγμα μάθησης E_1 .

	y	target
0	0.24	0.00
1	0.40	0.00
2	0.30	0.00
3	0.66	1.00
4	0.90	0.00
5	0.33	0.00
6	0.44	0.00
7	0.55	0.00
8	0.64	0.00
9	0.12	0.00

Σύμφωνα με τον παραπάνω πίνακα και με την συνάρτηση τετραγωνικού σφάλματος προκύπτει ότι το σφάλμα για το παράδειγμα E_1 είναι:

$$Error_{E_1} = \sum_{k=0}^9 (y_k - target_k)^2 = (0.24 + 0.00)^2 + \dots + (0.12 + 0.00)^2 = 2.2622$$

Αν προσπαθούσαμε να εφαρμόσουμε τον αλγόριθμο οπισθοδιάδοσης του σφάλματος (στον οποίο θα αναφερθούμε αργότερα), για να ελαττώσουμε μόνο αυτό το σφάλμα ($Error_{E_1} = 2.2622$) ιδανικά στο 0.0 ή κάτι κοντά σε αυτό, θα προκύπτει το φαινόμενο υπερμοντελοποίησης (Overfitting). Δηλαδή το νευρωνικό δίκτυο του παραδείγματος θα είχε εκπαιδευτεί έτσι ώστε να ταξινομεί σωστά μονό παραδείγματα τύπου E_1 . Προκειμένου να μην συμβεί αυτό, θα πρέπει να βγει ο μέσος όρος όλων των σφαλμάτων από τα παραδείγματα που εκπαιδεύτηκε το δίκτυο και τότε να εφαρμοστεί ο αλγόριθμος οπισθοδιάδοσης. Ο μαθηματικός τύπος που υπολογίζει το μέσο σφάλμα είναι:

$$AverageTotalError = \frac{\sum_{x=0}^K Error_{E_x}}{K} \quad (1.2)$$

Όπου $Error_{E_x}$ τα σφάλματά που προέκυψαν από τα παραδείγματα μάθησης E_x :

Γ'.1.3 ΛΕΠΤΟΜΕΡΗΣ ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

Η μηχανική μάθηση (machine learning), στην περίπτωση νευρωνικών δικτύων σταθερής δομής, δηλαδή με συγκεκριμένο αριθμό νευρώνων, επιπέδων, συνδέσεων, συνίσταται στον υπολογισμό των ανεξάρτητων παραμέτρων – βαρών των συνδέσεων που αυτά περιέχουν, ώστε να εκτελούν μια συγκεκριμένη λειτουργία. Ο πλέον συνήθης αλγόριθμος εκπαίδευσης των πολυεπίπεδων νευρωνικών δικτύων εμπρόσθιας τροφοδότησης, είναι ο αλγόριθμος της οπισθοδιάδοσης του σφάλματος (Error Back-Propagation ή απλά back-propagation). Ο αλγόριθμος αυτός, προκύπτει εφαρμόζοντας τις τεχνικές της θεωρίας βελτιστοποίησης (optimisation theory) πάνω στην αρχιτεκτονική ενός πολυεπίπεδου δικτύου και αποτελεί επέκταση του αλγορίθμου μάθησης των perceptrons στην περίπτωση των πολυεπίπεδων αρχιτεκτονικών (multilayer perceptrons). Η βελτιστοποίηση -ελαχιστοποίηση μιας συνάρτησης κόστους που αναφέρθηκε στο προηγούμενο κεφάλαιο, επιτυγχάνεται στο στάδιο της εκπαίδευσης του δικτύου, οπότε αυτό μαθαίνει να προσεγγίζει, με τις εξόδους που παρέχει, ένα σύνολο από επιθυμητές τιμές (desired outputs) που του παρέχονται από έναν εξωτερικό δάσκαλο (supervised training) μαζί με το αντίστοιχο σύνολο από δεδομένα – πρότυπα εισόδου (training data set).

Έστω ότι έχουμε ένα σύνολο εκπαίδευσης που αποτελείται από N ζεύγη της μορφής $[\bar{x}, \bar{d}]$, όπου $\bar{x}$ συμβολίζει το διάνυσμα (πρότυπο) εισόδου και το $\bar{d}$ το διάνυσμα της επιθυμητής εξόδου, όταν εφαρμοσθεί ως είσοδος το πρότυπο $\bar{x}$ (σε αυτό το κεφάλαιο θα χρησιμοποιούνται διανύσματα για καλύτερη απεικόνιση των εξισώσεων). Ας αναφερθούμε σε έναν από τους νευρώνες του επιπέδου εξόδου του δικτύου. Το σφάλμα στην έξοδο του νευρώνα j όταν στην είσοδο του δικτύου παρουσιάζεται το πρότυπο $\bar{x}(n)$, όπου $n = 1, 2, \dots, N$ δίδεται από τον τύπο:

$$e_j(n) = d_j(n) - y_j(n) \quad (1.3)$$

όπου d_j και y_j η επιθυμητή και η πραγματική έξοδος αντίστοιχα του νευρώνα j για είσοδο την $\bar{x}(n)$.

Η στιγμιαία τιμή του τετραγωνικού σφάλματος με την χρήση των νέων συμβολισμών ορίζεται για τον νευρώνα j ως:

$$\frac{1}{2} e_j^2(n) \quad (1.4)$$

και το άθροισμα των τετραγωνικών σφαλμάτων όλων των νευρώνων εξόδου, στην n -οστή επανάληψη-παρουσίαση ενός προτύπου στην είσοδό του δίδεται από τη σχέση:

$$G(n) = \frac{1}{2} \sum_j e_j^2(n) \quad (1.5)$$

Η συνάρτηση $G(n)$ αποτελεί τη συνάρτηση κόστους, η οποία ελαχιστοποιείται, συσσωρευτικά για όλα τα n , μέσω του αλγόριθμου back propagation. Ειδικότερα, ο αλγόριθμος της ανάστροφης διάδοσης προβαίνει, σε κάθε επανάληψη, n , σε μια ανανέωση (διόρθωση) $\Delta w_{ji}(n)$ στο βάρος $w_{ji}(n)$ της σύνδεσης του νευρώνα j κάποιου επιπέδου (έστω m) του δικτύου και του νευρώνα i του προηγούμενου $(m-1)$ επιπέδου, $m = 1, \dots, N$. Η ανανέωση γίνεται πρώτα για τα βάρη των συνδέσεων στο επίπεδο εξόδου, και στη συνέχεια, ανάστροφα, για τα βάρη των κρυμμένων επιπέδων, μέχρι το επίπεδο εισόδου. Η ανανέωση κάθε βάρους είναι ανάλογη της κλίσης $\frac{\partial G(n)}{\partial w_{ji}(n)}$, δηλαδή της παραγώγου του σφάλματος ως προς το συγκεκριμένο βάρος σύνδεσης, κατά την επεξεργασία του n -οστού προτύπου, και προσδιορίζεται, σύμφωνα με τη μέθοδο της βαθύτατης καθόδου (steepest descent) ως εξής :

$$\Delta w_{ji}(n) = -\eta \frac{\partial G(n)}{\partial w_{ji}(n)} \quad (1.6)$$

Επειδή η μέθοδος της βαθύτατης καθόδου είναι μια προσέγγιση της μεθόδου του Newton, η οποία δείχνει μεν την κατεύθυνση της ανανέωσης των βαρών, δηλαδή την κλίση της συνάρτησης σφάλματος, αλλά όχι το ακριβές βήμα της ανανέωσης (που το δείχνει η δεύτερη παράγωγος – ο πίνακας Hessian της συνάρτησης), η κλίση της συνάρτησης πολλαπλασιάζεται με μια παράμετρο, η , η οποία καθορίζει την ταχύτητα σύγκλισης, δηλαδή τον ρυθμό μάθησης (learning rate) του δικτύου.

Αποδεικνύεται ότι η ανανέωση των βαρών γράφεται ως εξής:

$$\Delta w_{ji}(n) = -\eta \delta_j(n) y_i(n) \quad (1.7)$$

όπου είναι το σφάλμα στην έξοδο του νευρώνα j του επιπέδου - έστω m - και y_j είναι η έξοδος του νευρώνα i του επιπέδου $(m-1)$, ή το i -οστό στοιχείο του διανύσματος εισόδου, αν $m = 1$.

Οι τιμές δ_i υπολογίζονται με βάση τις τιμές των εξόδων των νευρώνων του δικτύου και τις τιμές των βαρών των συνδέσεων του κατά την επανάληψη n . Ειδικότερα, αν το m αναφέρεται στο επίπεδο της εξόδου, ισχύει (για βηματική σιγμοειδή συνάρτηση ενεργοποίησης των νευρώνων) για τον k -οστό νευρώνα:

$$\delta_k(n) = y_k(n)[1 - y_k(n)][d_k(n) - y_k(n)] \quad (1.8)$$

Ενώ, αν το m αναφέρεται σε κρυμμένα επίπεδα, τότε ισχύει:

$$\delta'_k(n) = y_k(n)[1 - y_k(n)] \sum_j \delta_j(n) w_{jk} \quad (1.9)$$

Στον υπολογισμό των σφαλμάτων δ'_k , στην περίπτωση αυτή, το άθροισμα του δεύτερου μέλους της εξίσωσης αφορά σε όλα τα σφάλματα δ_j του ανωτέρου $(m+1)$ επιπέδου.

Σε συνέχεια των ανωτέρω, η εφαρμογή του αλγορίθμου Back-Propagation κι αφού αρχικοποιηθούν τα βάρη των συνδέσεων του δικτύου σε μικρές τυχαίες τιμές, γίνεται, για κάθε πρότυπο εισόδου, σε δύο φάσεις, στην *ορθή φορά* (forward pass) και στην *ανάστροφη φορά* (reverse pass). Η πρώτη φάση ξεκινά από το επίπεδο εισόδου με την παρουσίαση του διανύσματος εισόδου στο δίκτυο, περιλαμβάνει υπολογισμό των εξόδων όλων των νευρώνων, με βάση τις εξισώσεις ενεργοποίησης του κεφαλαίου Β.1 , προχωρώντας στα επόμενα επίπεδα του δικτύου, ώσπου να φτάσει τελικά στο τελευταίο επίπεδο (εξόδου) και να υπολογίσει το διάνυσμα εξόδου του δικτύου, καθώς και το σφάλμα του κάθε νευρώνα αυτού του επιπέδου από τις εξισώσεις (1.8).

Η δεύτερη φάση, στη συνέχεια, ξεκινά από το επίπεδο εξόδου, διαδίδοντας τα σήματα σφάλματος, ακολουθιακά, προς τα προηγούμενα (κρυμμένα) επίπεδα - με βάση τις

εξισώσεις (1.9) - και υπολογίζοντας την ανανέωση του κάθε βάρους του δικτύου, ώσπου να φτάσει τελικά στο επίπεδο εισόδου του δικτύου.

Δ' ΠΕΙΡΑΜΑΤΙΚΟ ΚΟΜΜΑΤΙ

Δ'.1 ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ KERAS API ΚΑΙ ΤΟΥ FRAME WORK TENSOR FLOW

Σε αυτό το τεχνικό κομμάτι της εργασίας, θα χρειαστεί να γίνει compile και execute κώδικα Python, που κάνει χρήση του TensorFlow για βαθιά μάθηση και του Keras API .

Υπάρχουν δυο επιλογές για να γίνει αυτό

- Αυτή που επιλέχτηκε ήταν :Εγκατάσταση Python ,Tensorflow και κάποιου IDE(Integrated development environment) όπως το Jupyter Notebook
- Η εναλλακτική ήταν: Χρήση των online εργαστηρίων της IBM Cognitive class

Εγκατάσταση Python και Tensorflow

Το πρώτο βήμα είναι η εγκατάσταση της Python 3.7. Από τον Ιανουάριο του 2019, αυτή ήταν η τελευταία έκδοση της Python 3. Αυτό που χρησιμοποιήθηκε ήταν η Miniconda (Anaconda) κυκλοφορία της Python, η οποία περιέχει ήδη πολλά από τα επιστημονικά πακέτα που θα χρειαστούν για αυτό το πείραμα. Η Anaconda υποστηρίζεται απευθείας από τα εξής λειτουργικά συστήματα :Windows, Mac και Linux. Η Miniconda είναι ουσιαστικά το μινιμαλιστικό set χαρακτηριστικών από την πολύ μεγάλη Anaconda κυκλοφορία της Python.

Αντιμετώπιση ασυνδεσιμότητας του Tensor Flow 2.0 με την Python 3.7

Αφού ολοκληρωθεί η λήψη και εγκατάσταση της Miniconda, θα πρέπει να δημιουργηθεί ένα περιβάλλον για Python 3.6, γιατί πολλά από τα πακέτα του Tensorflow 2.0 δεν υποστηρίζουν Python 3.7.

Για να γίνει αυτό θα πρέπει να εκτελεστούν οι παρακάτω εντολές στο τερματικό παράθυρο της Anaconda:

```
conda create -y --name tensorflow python=3.6
```

Για να γίνει η είσοδος σε αυτό το περιβάλλον θα πρέπει να χρησιμοποιηθεί η εντολή που ακολουθεί:

```
activate tensorflow
```

Αυτή η εντολή πρέπει να γίνεται κάθε φορά που ανοίγει ένα νέο παράθυρο τερματικού Miniconda.

Εγκατάσταση του Jupiter notebook IDE

Το ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) του Jupiter note book περιέχει μια σουίτα λογισμικού που βοηθάει στην ανάπτυξη προγραμμάτων υπολογιστή .

Είναι εύκολο να εγκατασταθεί με την παρακάτω εντολή:

```
conda install -y jupyter
```

Αφού ολοκληρωθεί η εγκατάσταση του Jupyter notebook προχωράμε στη εγκατάσταση των θεμελιωδών πακέτων που θα χρειαστούν για το πείραμα.

```
conda install -y scipy
pip install --exists-action i --upgrade sklearn
pip install --exists-action i --upgrade pandas
pip install --exists-action i --upgrade pandas-datareader
pip install --exists-action i --upgrade matplotlib
pip install --exists-action i --upgrade pillow
pip install --exists-action i --upgrade tqdm
pip install --exists-action i --upgrade requests
pip install --exists-action i --upgrade h5py
pip install --exists-action i --upgrade pyyaml
pip install --exists-action i --upgrade tensorflow_hub
pip install --exists-action i --upgrade bayesian-optimization
pip install --exists-action i --upgrade spacy
pip install --exists-action i --upgrade genism
pip install --exists-action i --upgrade flask
pip install --exists-action i --upgrade boto3
pip install --exists-action i --upgrade gym
pip install --exists-action i --upgrade tf-nightly-2.0-preview
pip install --exists-action i --upgrade keras-rl2 --user conda
update -y -all
```

Μια τελευταία εντολή που χρειάζεται πριν ολοκληρωθεί η εγκατάσταση, είναι αυτή που θα συνδέει το νέο περιβάλλον tensorflow που δημιουργήσαμε με το Jupyter note book.

```
python -m ipykernel install --user --name tensorflow --
display-name "Python 3.6 (tensorflow)"
```

Για να ενεργοποιηθεί το Jupiter χρειάζεται την παρακάτω εντολή:

```
jupyter notebook
```

Δ'.2 ΔΗΜΙΟΥΡΓΙΑ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ ΓΙΑ ΑΝΑΓΝΩΡΙΣΗ ΧΑΡΑΚΤΗΡΩΝ

Δ'.2.1 ΕΙΣΑΓΩΓΗ

Αφού εισήχθησαν και εγκαταστάθηκαν όλα τα απαραίτητα πακέτα βαθιάς μάθησης (Κεφάλαιο :Δ'.1) , μπορούν να εισαχθούν δεδομένα στο πρόγραμμα, να δημιουργηθεί ένα μοντέλο βαθιάς μάθησης και να εκπαιδευτεί. Το πείραμα που επιλέχθηκε ήταν η δημιουργία ενός νευρωνικού δικτύου, το οποίο θα μπορεί να αναγνωρίζει αριθμούς (0 έως 9) γραμμένους στο χέρι. Ο λόγος που επιλέχθηκαν αριθμοί και όχι γράμματα, ήταν επειδή βρέθηκε ένα σετ δεδομένων το οποίο ήταν ήδη φορμαρισμένο στο να εισάγεται εύκολα στο πρόγραμμα χωρίς πολλές αλλαγές, ελαττώνοντας σημαντικά το χρόνο που αφιερώθηκε στη προ επεξεργασία των δεδομένων. Όμως με το κατάλληλο σετ δεδομένων και κάποιες μικρές αλλαγές στον κώδικα, η ίδια λογική θα μπορούσε να εφαρμοστεί και σε γράμματα της αλφαβήτου, γραμμένα στο χέρι. Το σετ δεδομένων που επιλέχθηκε, προέρχεται από την βάση δεδομένων MNIST, και προορίζεται για ανθρώπους που θέλουν να δοκιμάσουν τεχνικές βαθιάς μάθησης και μεθόδους αναγνώρισης προτύπων πάνω σε πραγματικά δεδομένα. Το σετ δεδομένων περιέχει εικόνες 28*28 pixel με ψηφία αριθμών από το μηδέν έως το εννιά, γραμμένα στο χέρι .

Δ'.2.2 ΔΗΜΙΟΥΡΓΙΑ ΠΕΙΡΑΜΑΤΟΣ ΜΕ ΑΝΑΦΟΡΕΣ ΣΤΟΝ ΚΩΔΙΚΑ

Το πρώτο κομμάτι για να ξεκινήσει το πείραμα είναι να εισαχθούν στο πρόγραμμα το keras και το tensorflow :

```
In [1]: import tensorflow.keras as keras
```

```
In [2]: import tensorflow as tf
```

Στην συνέχεια, μπορεί να εισαχθεί στο πρόγραμμα και το σετ δεδομένων που αναφέρθηκε προηγουμένως:

```
In [6]: mnist = tf.keras.datasets.mnist  
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

Τα δεδομένα x_train είναι οι εικόνες 28*28 pixel από αριθμούς γραμμένους στο χέρι. Το y_train είναι η ετικέτα(Label) για το τι αριθμοί είναι πραγματικά (0-9). Όπως είναι προφανές από το όνομα τους, τα δεδομένα αυτά θα χρησιμοποιηθούν για να εκπαιδεύσουν το νευρωνικό δίκτυο. Αντίθετα τα δεδομένα x_test ,y_test είναι νέα δεδομένα, ανεξάρτητα από τα προηγούμενα, τα οποία θα χρησιμοποιηθούν για να 'τεστάρουν' το νευρωνικό δίκτυο.

Οι ετικέτες `y_test` εμπεριέχουν και αυτές την πληροφορία για το τι αριθμοί είναι οι εικόνες `x_test` και θα συγκριθούν στην συνέχεια με τις προβλέψεις του δικτύου για να προσδιοριστεί η ακρίβεια του.

*Σημείωση: Τα νευρωνικά δίκτυα είναι πολύ καλά στο να προσαρμόζονται (fitting) στα δεδομένα τόσο πολύ, που πολλές φορές υπερμοντελοποιούνται (over-fit).Ο σκοπός που προσπαθεί να πετύχει ο παρακάτω κώδικας δεν είναι απλά να φτιάξει ένα νευρωνικό δίκτυο που να απομνημονεύει τα δεδομένα , αλλά να τα “γενικεύσει” και να μάθει το πρόβλημα και τα μοτίβα που συνδέονται σε αυτό. *

Αφού ολοκληρώθηκε η εισαγωγή των δεδομένων, μια καλή πρακτική θα ήταν να δούμε πως μοιάζει ένα από αυτά. Αυτό μπορεί να γίνει με την εκτέλεση της παρακάτω εντολής:

```
In [8]: print(x_train[1])
```

```

84 252 253 122 0 0 0 0 0 0 0 0 0 0 0
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 163 252 252 252 253 252 252
96 189 253 167 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 0 51 238 253 253 190 114 253 228
47 79 255 168 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 48 238 252 252 179 12 75 121 21
0 0 253 243 50 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 38 165 253 233 208 84 0 0 0 0
0 0 253 252 165 0 0 0 0]
[ 0 0 0 0 0 0 7 178 252 240 71 19 28 0 0 0 0
0 0 253 252 195 0 0 0 0]
[ 0 0 0 0 0 0 57 252 252 63 0 0 0 0 0 0 0
0 0 253 252 195 0 0 0 0]
[ 0 0 0 0 0 0 198 253 190 0 0 0 0 0 0 0 0
0 0 255 253 196 0 0 0 0]
[ 0 0 0 0 0 76 246 252 112 0 0 0 0 0 0 0 0
0 0 253 252 148 0 0 0 0]
[ 0 0 0 0 85 252 230 25 0 0 0 0 0 0 0 0
7 135 253 186 12 0 0 0 0]
[ 0 0 0 0 85 252 223 0 0 0 0 0 0 0 0 7

```

Κάθε τιμή αντιστοιχεί σε μια απόχρωση του μαύρου για κάθε pixel της εικόνας 28*28. Με το 253 να αντιστοιχεί στο μαύρο, το 100 να αντιστοιχεί στο γκρι και το μηδέν στο λευκό. Όλες οι ενδιάμεσες τιμές είναι αποχρώσεις αυτών των τριών βασικών χρωμάτων.

Στην πιο απλοποιημένη μορφή η παραπάνω αναπαράσταση θα ήταν η καλύτερη επιλογή που θα είχαμε για να αναπαραστήσουμε τα δεδομένα.

Αλλά αφού εγκαταστήσαμε στο προηγούμενο κεφάλαιο το πακέτο `matplotlib` μπορούμε να αναπαραστήσουμε τα δεδομένα με έναν ακόμα πιο ευανάγνωστο προς τον άνθρωπο τρόπο.

```
In [10]: import matplotlib.pyplot as plt
In [11]: plt.imshow(x_train[1], cmap=plt.cm.binary)
plt.show()
```



Φαίνεται έτσι λοιπόν με έναν καλύτερο τρόπο ότι το δεδομένο εκπαίδευσης `x_train[1]` είναι ο αριθμός μηδέν. Για να σιγουρευτούμε ότι η ετικέτα `y_train[1]` που αντιστοιχεί σε αυτόν τον αριθμό είναι επίσης σωστή, πρέπει να εμφανίσουμε το περιεχόμενο της

```
In [12]: print(y_train[1])  
0
```

Κάθε ένα pixel εικόνας 28×28 που θα χρησιμοποιηθεί είτε για εκπαίδευση, είτε για test, θα αποτελέσει ένα νευρώνα εισόδου (input) στο νευρωνικό δίκτυο. Αυτό σημαίνει ότι το δίκτυο θα έχει συνολικά $28 \times 28 = 784$ νευρώνες εισόδου. Σε προηγούμενο κεφάλαιο αναφέρθηκε ότι, το πόσο ‘ενεργοποιημένος’ είναι ένας νευρώνας, καθορίζεται από την τιμή που έχει ανάμεσα στο μηδέν και ένα, με το ένα να σημαίνει ότι ο νευρώνας είναι πλήρως ενεργοποιημένος και το μηδέν να σημαίνει ότι είναι ανενεργός. Πράγμα που δεν συμβαίνει με τις τιμές που έχουν τα pixel από τις εικόνες, οι οποίες κυμαίνονται από 0-253.

Για να διορθωθεί το παραπάνω πρόβλημα θα πρέπει να εκτελεστούν οι παρακάτω εντολές python. Αυτή η διαδικασία λέγεται “κανονικοποίηση των δεδομένων”.

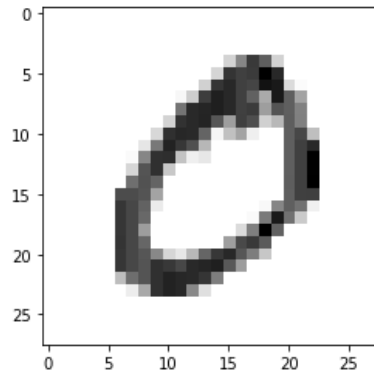
```
In [11]: x_train = tf.keras.utils.normalize(x_train, axis=1)  
x_test = tf.keras.utils.normalize(x_test, axis=1)
```

Αν εκτελέσουμε τώρα τις εντολές που εκτελέσαμε προηγουμένως για να δούμε πως μοιάζουν τα δεδομένα, θα παρατηρήσουμε ότι οι τιμές πράγματι άλλαξαν από 0-253 σε 0-1.

```
In [20]: print(x_train[1])  
0.30506548 0.08583808 0. 0. 0. 0.  
0. 0. 0. 0. ]  
[0. 0. 0. 0. 0. 0.  
0. 0. 0. 0. 0. 0.  
0. 0. 0.08087653 0.38341541 0.36240278 0.37133624  
0.48350001 0.4068725 0. 0. 0. 0.  
0. 0. 0. 0. ]  
[0. 0. 0. 0. 0. 0.  
0. 0. 0. 0. 0. 0.  
0. 0.08861609 0.3824786 0.40758025 0.36240278 0.35218  
0.44704564 0.43262392 0.06832372 0.00859123 0. 0.  
0. 0. 0. 0. ]  
[0. 0. 0. 0. 0. 0.  
0. 0. 0. 0. 0. 0.01621743  
0.095788 0.36759266 0.42460179 0.40758025 0.36240278 0.29765841  
0.16116667 0.43262392 0.30326141 0.17468832 0. 0.  
0. 0. 0. 0. ]  
[0. 0. 0. 0. 0. 0.  
0. 0. 0. 0. 0. 0.26434406  
0.4023096 0.41354174 0.42460179 0.40758025 0.36240278 0.37133624
```

Η εικόνα μπορεί να φαίνεται λίγο ξεθωριασμένη συγκριτικά με την προηγούμενη μορφή της αλλά αυτό δεν επηρεάζει καθόλου το πείραμα

```
In [22]: plt.imshow(x_train[1], cmap=plt.cm.binary)
plt.show()
```



Με αυτό το τελευταίο κομμάτι ολοκληρώνεται ουσιαστικά το κομμάτι της προ επεξεργασίας των δεδομένων. Πλέον είμαστε σε θέση να φτιάξουμε το νευρωνικό δίκτυο. Ξεκινάμε με το να φτιάξουμε ένα δίκτυο το οποίο αρχικά θα μεταφέρει πληροφορία προς τα εμπρός(Feed forward), όχι προς τα πίσω προς το παρόν.

```
In [23]: model = tf.keras.models.Sequential()
```

Αφού προσδιορίστηκε ο τύπος του δικτύου (Feed forward) ήρθε η ώρα να ξεκινήσουμε να προσθέτουμε επίπεδα (layers), ξεκινώντας με το επίπεδο των εισόδων. Όπως έχει ήδη αναφερθεί το επίπεδο εισόδων θα αποτελείται από 784 νευρώνες, καθένας από τους οποίους θα αναπαριστά ένα από τα Pixel των εικόνων 28*28. Όμως αυτές οι εικόνες αναπαρίστανται σε δισδιάστατο χώρο, σε αντίθεση με τα επίπεδα των νευρωνικών δικτύων τα οποία εμφανίζονται σε μονοδιάστατο (κάθετες γραμμές). Προκειμένου, λοιπόν, να κάνουμε την δισδιάστατη 28*28 εικόνα συμβατή με το μονοδιάστατο επίπεδο εισόδου 1*784, θα πρέπει να εκτελέσουμε την παρακάτω εντολή:

```
In [24]: model.add(tf.keras.layers.Flatten())
```

Αυτό θα λειτουργήσει ως το επίπεδο εισόδου. Θα παίρνει τις εικόνες που θα του δίνουμε και θα τις 'ισιώνει' μέσω της συνάρτησης Flatten() πριν τις βάλει στο δίκτυο. Στην συνέχεια θα πρέπει να προσθέσουμε τα κρυμμένα επίπεδα. Αφού το πείραμα είναι πείραμα βαθιάς μάθησης, το δίκτυο θα πρέπει να έχει δυο ή περισσότερα κρυμμένα επίπεδα. Στην περίπτωση αυτή το δίκτυο θα έχει δύο και θα είναι πλήρως συνεκτικό. Δηλαδή κάθε ένας νευρώνας θα συνδέεται με όλους τους νευρώνες του προηγούμενου επιπέδου, αυτό δηλώνεται στην λέξη Dense στον κώδικα. Ο αριθμός 128 στην παρακάτω εικόνα, δηλώνει τον αριθμό των νευρώνων σε αυτό το επίπεδο και επιλέχτηκε μετά από μια σειρά πειραματισμών που έδειξαν ότι είναι μια καλή επιλογή. Το όνομα relu αναφέρεται στην συνάρτηση ενεργοποίησης που θα χρησιμοποιηθεί στους νευρώνες αυτού του δικτύου. Υπάρχουν όμως και άλλες επιλογές για συναρτήσεις ενεργοποίησης όπως αναφέρεται στο κεφάλαιο B.1.1(sigmoid ,Gaussian κ.ά)

```
In [25]: model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
```

Στην συνέχεια θα προσθέσουμε και το δεύτερο κρυμμένο επίπεδο το οποίο θα έχει ίδια χαρακτηριστικά με το πρώτο.

```
In [26]: model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
```

Τέλος, ήρθε η ώρα να προστεθεί και το τελευταίο επίπεδο του νευρωνικού δικτύου, το επίπεδο εξόδου (output layer). Αυτό το επίπεδο θα αποτελείται από 10 νευρώνες, ένας για κάθε πιθανή πρόβλεψη αριθμού (0-9). Η συνάρτηση ενεργοποίησης για αυτό το επίπεδο θα είναι η softmax. Επειδή εδώ ψάχνουμε κάτι σαν διανομή πιθανοτήτων, σε όλες τις πιθανές προβλέψεις (αριθμοί 0-9) θα επιλεγεί ως τελική πρόβλεψη αυτή με την μεγαλύτερη πιθανότητα.

```
In [28]: model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

Μέχρι αυτό το σημείο το μοντέλο του νευρωνικού δικτύου έχει ολοκληρωθεί. Τώρα είναι το κομμάτι που περνάμε τις ρυθμίσεις για να το βελτιστοποιήσουμε – εκπαιδεύσουμε.

Ο βελτιστοποιητής που επιλέχτηκε ονομάζεται ‘adam’ και έχει παρόμοια δράση με τον αλγόριθμο “οπισθοδιάδοσης του λάθους” που περιγράφηκε στο κεφάλαιο Γ’.1.1. Αυτός προσπαθεί να ελαττώσει όσο το δυνατόν το κόστος που αναφέρθηκε στο κεφάλαιο Γ’.1.2.

Στον κώδικα η μετρική που υπολογίζει το κόστος έρχεται με το όνομα loss.

```
In [29]: model.compile(optimizer='adam',  
                        loss='sparse_categorical_crossentropy',  
                        metrics=['accuracy'])
```

Αφού προσδιορίστηκε ο βελτιστοποιητής, μπορούν να εισαχθούν τα δεδομένα εκπαίδευσης `x_train`, `y_train` (6000) για να εκπαιδευτεί το δίκτυο. Όπως φαίνεται στην παρακάτω εικόνα, μετά από κάθε πέρασμα (epoch) του σετ εκπαίδευσης (τρία περάσματα συνολικά), το δίκτυο μαθαίνει περισσότερο. Επειδή μειώνεται το κόστος και αυξάνεται η ακρίβεια, φτάνουμε σε ένα κόστος (0,0735), το οποίο μας δίνει να καταλάβουμε ότι το δίκτυο μπορεί ουσιαστικά να ταξινομήσει σχεδόν οποιαδήποτε εικόνα με χειρόγραφο αριθμό του δώσουμε.

```

model.fit(x_train, y_train, epochs=3)

WARNING: Logging before flag parsing goes to stderr.
W0809 19:08:45.220357 3888 deprecation.py:323] From C:\Users\Noula\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\ops\math_grad.py:1251: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
W0809 19:08:45.276076 3888 deprecation.py:323] From C:\Users\Noula\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\keras\optimizer_v2\optimizer_v2.py:460: BaseResourceVariable.constraint (from tensorflow.python.ops.resource_variable_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Apply a constraint manually following the optimizer update step.

Train on 60000 samples
Epoch 1/3
60000/60000 [=====] - 3s 52us/sample - loss: 0.2640 - accuracy: 0.9227
Epoch 2/3
60000/60000 [=====] - 3s 46us/sample - loss: 0.1079 - accuracy: 0.9661 -0.1561 +0.0434
Epoch 3/3
60000/60000 [=====] - 3s 48us/sample - loss: 0.0735 - accuracy: 0.9766 -0.0344 +0.0105
<tensorflow.python.keras.callbacks.History at 0x1699a4edb38>

```

Όπως αναφέρθηκε και προηγουμένως, τα νευρωνικά δίκτυα είναι πολύ εύκολο να προσαρμόζονται στα δεδομένα που τους δίνονται στο κομμάτι της εκπαίδευσης (over fit). Αν και στο παρόν πείραμα είναι κάτι πολύ δύσκολο να συμβεί εξαιτίας του μεγάλου αριθμού παραδειγμάτων (6000) που ‘αναγκάζουν’ το νευρωνικό δίκτυο να γενικευτεί και σε άλλα παραδείγματα πέρα από αυτά του σετ εκπαίδευσης. Εντούτοις, αξίζει το νευρωνικό δίκτυο να τεσταριστεί με τα παραδείγματα από το τεστ σετ (x_test,y_test), για να καθοριστεί αν η ακρίβεια είναι κοινή με το σετ εκπαίδευσης .

```

In [21]: val_loss, val_acc = model.evaluate(x_test, y_test)
          print(val_loss)
          print(val_acc)

10000/10000 [=====] - 0s 35us/sample - loss: 0.0951 - accuracy: 0.9704
0.09510906130848452                                     +0.0216 +0.0062
0.9704

```

Είναι αναμενόμενο το κόστος και η ακρίβεια να είναι καλύτερα στο σετ δεδομένων όπου εκπαιδεύτηκε το νευρωνικό δίκτυο από ότι στο σετ στο οποίο έχει τεσταριστεί. Αν οι διαφορές δεν είναι μεγάλες τότε δεν υπάρχει πρόβλημά over fit. Το γεγονός αυτό συμβαίνει και σε αυτή τη περίπτωση, όπου οι διαφορές στο κόστος και στην ακρίβεια από το τεστ σετ με το αντίστοιχο σετ εκπαίδευσης ήταν πολύ μικρές, συγκεκριμένα ήταν 216/10000 και 62/10000 αντίστοιχα. Με την συνθήκη αυτή ολοκληρώνεται πλέον το πείραμα. Προκειμένου να γίνουν μελλοντικές τροποποιήσεις ή βελτιώσεις στο μοντέλο που δημιουργήθηκε είναι απαραίτητο να εκτελεστεί ο παρακάτω κώδικας που το αποθηκεύει. Τέτοιου είδους παραινέσεις παρατίθενται στο σχετικό κεφάλαιο των συμπερασμάτων

```

In [22]: model.save('Ioannis_kivrakidis.model')

```

Για να γίνει η εκ νέου φόρτωση του μοντέλου που αποθηκεύτηκε για νέες τροποποιήσεις ή πειραματισμούς πρέπει να εκτελεστεί η παρακάτω εντολή:

```
In [23]: new_model = tf.keras.models.load_model('Ioannis_kivrakidis.model')
```

Κάτι που πρέπει να ελεγχθεί είναι ο τρόπος με τον οποίο εμφανίζονται οι προβλέψεις που έβγαλε το δίκτυο. Όπως συμβαίνει συχνά, έτσι και σε αυτή την περίπτωση η εμφάνιση δεν θα είναι πολύ φιλική για τον άνθρωπο, αυτό που ακολουθεί, είναι η διανομή πιθανοτήτων για κάθε παράδειγμα του τεστ σετ που έβγαλε η συνάρτηση SoftMax

```
In [24]: predictions = new_model.predict(x_test)
```

```
In [25]: print(predictions)
```

```
[[7.8469027e-09 2.5516901e-08 4.1843705e-06 ... 9.9992752e-01
 6.6414656e-09 1.6498943e-06]
 [6.6182054e-08 7.8510478e-05 9.9992049e-01 ... 3.7140402e-09
 1.2094900e-09 1.8968635e-12]
 [1.3081829e-07 9.9970078e-01 8.4057283e-05 ... 1.8882252e-04
 5.5966698e-06 7.4628878e-07]
 ...
 [2.4981970e-08 5.3210539e-07 1.9373235e-06 ... 7.9813304e-05
 7.8729045e-06 4.0495759e-04]
 [2.4445077e-08 1.7916973e-08 8.9349328e-09 ... 1.2911580e-07
 8.3900923e-06 1.8321957e-07]
 [1.0750278e-07 3.2790200e-07 2.1628895e-07 ... 3.6345107e-11
 3.2659901e-07 2.5472455e-09]]
```

Για να μπορέσουν να γίνουν για άλλη μια φορά τα αποτελέσματα πιο ευανάγνωστα θα πρέπει να αξιοποιήσουμε ένα ακόμα πακέτο εργαλείων που εγκαταστάθηκαν κατά την διάρκεια της εγκατάστασης. Το εργαλείο αυτό είναι το numpy.

```
In [26]: import numpy as np
```

Αν θέλαμε να δούμε τη πρόβλεψη του δικτύου για τον πρώτο αριθμό του test σετ θα έπρεπε να εκτελέσουμε την παρακάτω εντολή :

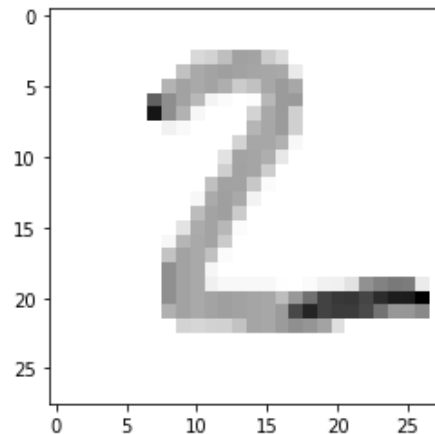
```
In [29]: print(np.argmax(predictions[1]))
```

```
2
```

Ταυτίζεται όμως με την αντίστοιχη εικόνα του πρώτου χειρόγραφου αριθμού του τεστ σετ;

Για να το μάθουμε αυτό, θα πρέπει να εκτελέσουμε την παρακάτω εντολή:

```
In [30]: plt.imshow(x_test[1], cmap=plt.cm.binary)
plt.show()
```

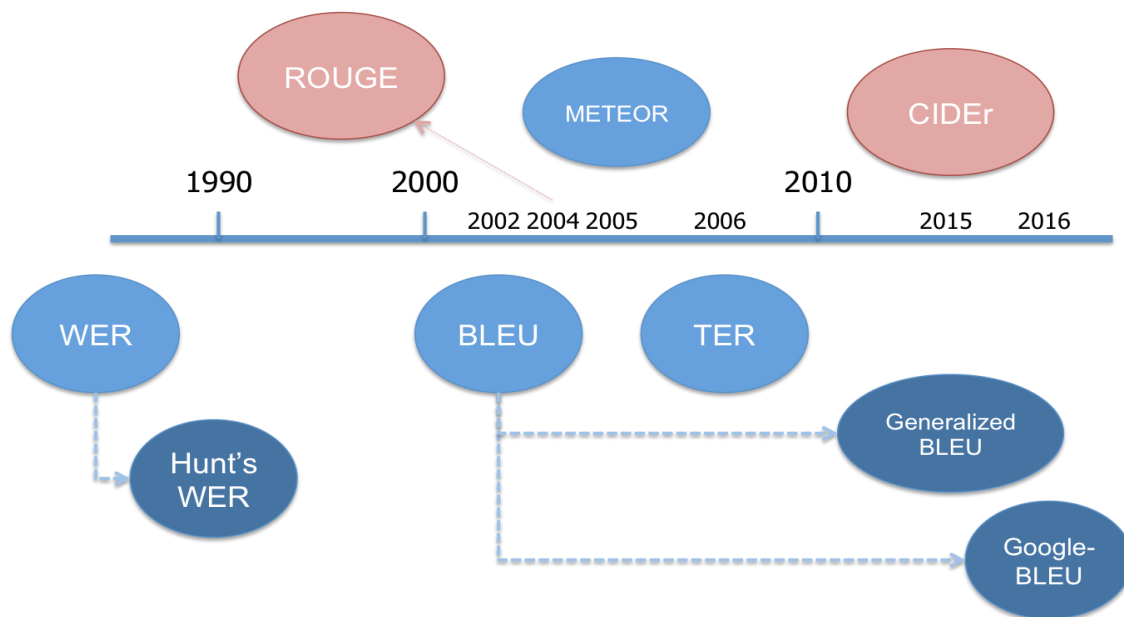


Πράγματι, η πρόβλεψη είναι σωστή. Με τον ίδιο τρόπο θα μπορούσαμε να ελέγξουμε και τις υπόλοιπες προβλέψεις, αλλά θα ήταν πολύ χρονοβόρο, για αυτό εμπιστευόμαστε τις μετρικές του κόστους και της ακριβείας που αναφέρθηκαν παραπάνω. Το σίγουρο είναι ότι ένα μικρό ποσοστό αριθμών θα μπερδέψουν το νευρωνικό δίκτυο και θα τους ταξινομήσει λάθος, εξαιτίας του πόσο κακογραμμένοι είναι. Αυτό όμως θα μπορούσε να παρθεί ως κάτι αποδεκτό, γιατί ούτε οι άνθρωποι μπορούν πάντοτε να αναγνωρίζουν όλους τους αριθμούς σωστά αν είναι πολύ κακογραμμένοι, κάνοντας το μοντέλο να φτάσει ακόμα πιο κοντά στο πραγματικό σκοπό του, να παρομοιάσει δηλαδή τον τρόπο λειτουργίας του ανθρωπίνου εγκεφάλου.

Δ'3 ΥΠΟΛΟΓΙΣΜΟΣ ΜΕΤΡΙΚΩΝ ΑΠΟΔΟΣΗΣ ΜΟΝΤΕΛΩΝ ΜΕΤΑΦΡΑΣΗΣ ΚΕΙΜΕΝΩΝ

Δ'3.1 ΕΙΣΑΓΩΓΗ

Τα τελευταία χρόνια η εξέλιξη της πληροφορικής έχει συνεισφέρει πολύ στην βελτίωση της ζωής των ανθρώπων σε διάφορους κλάδους. Στον κλάδο της επικοινωνίας πέτυχε να προσφέρει μοντέλα αυτόματης μετάφρασης κειμένων και μηνυμάτων. Όμως προκειμένου να γίνει εφικτή η σύγκριση αυτών των μοντέλων ώστε να προσδιοριστούν αυτά που έχουν την καλύτερη απόδοση, δημιουργήθηκαν κάποιες μετρικές απόδοσης μεταφράσεων. Τόσο τα ονόματα των μετρικών όσο και η χρονολογική σειρά δημιουργίας αυτών φαίνεται αναλυτικά στον παρακάτω πίνακα 1 .



Πίνακας 1: Ιστορική εξέλιξη Μετρικών απόδοσης (πηγή: Gwena Cunha NLP metrics)

Σκοπός αυτού του κεφαλαίου είναι ο υπολογισμός των μετρικών απόδοσης **WER**, **METEOR**, **TER** που θα χρησιμοποιηθούν για την σύγκριση δύο μοντέλων μετάφρασης, του στατιστικού και του νευρωνικού μοντέλου. Συγκεκριμένα, κάθε ένα από αυτά τα μοντέλα έχει μεταφράσει ένα κοινό κείμενο. Η μετάφραση του στατιστικού μοντέλου χαρακτηρίζεται με το όνομα **S1**, ενώ η μετάφραση του νευρωνικού μοντέλου χαρακτηρίζεται με το όνομα **S2**. Οι μεταφράσεις αυτές σε συνδυασμό με μια επιπλέον μετάφραση, την μετάφραση του μεταφραστή **Sr** (που θεωρείται ότι είναι η επιθυμητή), θα αποτελέσουν την είσοδο στους αλγορίθμους των μετρικών **WER**, **METEOR**, **TER**, οι οποίες με την σειρά τους θα βγάλουν τα ανάλογα αποτελέσματα.

Δ'.3.2 TER

Βιβλιογραφική ανασκόπηση της μετρικής TER:

Translation Error Rate(TER):Είναι μια μέθοδος που χρησιμοποιείται από τους ειδικούς της μηχανικής μετάφρασης για να καθορίσουν το ποσό της μετά επεξεργασίας που χρειάζεται για δουλειές μηχανικής μετάφρασης .Η αυτόματη μετρική μετράει τον αριθμό των ενεργειών που χρειάζονται για την επεξεργασία ενός μεταφρασμένου τμήματος κειμένου (Hypothesis) σε αντιστοιχία με ένα μεταφρασμένο κείμενο που λαμβάνεται ως αναφορά(Reference).Οι τιμές της μετρικής κυμαίνονται από 0.00 έως 1.00, με 1.00 να σημαίνει ότι το μεταφρασμένο τμήμα κειμένου χρειάζεται μεγάλο ποσό μετα επεξεργασίας ώστε να αντικατοπτρίζει το κείμενο αναφοράς. Γενικά μεταφράσεις με τιμές TER μικρότερες ή ίσες του 0.30 θεωρούνται αποδεκτές.

Μεθοδολογία που ακολουθήθηκε με αναφορές στον κώδικα:

Αρχικά το πρώτο πράγμα που έγινε ήταν η δημιουργία ενός νέου περιβάλλοντος για να γίνει εφικτή η εκτέλεση κώδικα, ο οποίος θα χρησιμοποιεί **python 2** .Ο λόγος που έγινε χρήση παλαιότερης έκδοσης Python ήταν επειδή, μια από τις θεμελιώδεις βιβλιοθήκες για τον υπολογισμό της μετρικής TER ήταν διαθέσιμη μόνο σε αυτή. Η δημιουργία του νέου περιβάλλοντος έγινε με την εντολή

```
conda create -y --name TER python=2
```

```
(base) C:\Users\ >conda create -y --name TER python=2
Collecting package metadata: done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.6.14
  latest version: 4.7.12

Please update conda by running

$ conda update -n base -c defaults conda
```

Μετά την είσοδο στο νέο περιβάλλον με την εντολή:

```
activate TER
```

έγινε η εγκατάσταση ενός απαραίτητου πακέτου, του pyter με την εντολή:

```
pip install pyter
```

Στην συνέχεια προκειμένου να γίνει εφικτή η παρουσίαση των αποτελεσμάτων σε πίνακα έγινε εγκατάσταση του πακέτου plotly με την εντολή:

```
pip install plotly==4.1.0
```

Τέλος έγινε η εκτέλεση του κώδικα TER.py, με την εντολή:

```
python TER.py
```

Ο οποίος δίνει σαν αποτέλεσμα των παρακάτω πίνακα:

TER(S1,Sr)	TER(S2,Sr)
0.750	0.000
0.600	0.400
0.417	0.333
0.529	0.559
0.667	0.625
1.000	1.000
0.545	0.364
0.261	0.304
1.500	1.500
0.857	0.143
0.375	0.000
0.492	0.458
0.353	0.235
1.000	0.667
0.750	0.750
0.300	0.200
0.000	0.000
0.250	0.167
0.591	0.636
0.455	0.591

Στην αριστερή στήλη είναι η τιμή της TER μετά από την σύγκριση του Sr με το S1 ανά γραμμή και αντίστοιχα δεξιά η σύγκριση του Sr με το S2.

Σημείωση: Στο αρχείο Sr όπου υπήρχαν ‘«’ ‘»’ αντικαταστάθηκαν με κενό ‘ ’ ’ γιατι δημιουργούσαν πρόβλημα τόσο σε αυτόν όσο και στον κώδικα του WER !

Δ'.3.3 METEOR

Βιβλιογραφική ανασκόπηση της μετρικής METEOR:

METEOR(Metric for Evaluation of Translation with Explicit Ordering) :Είναι μια μετρική για την εκτίμηση ενός αποτελέσματος μηχανικής μετάφρασης. Αυτή η μετρική είναι βασισμένη στον αρμονικό μέσο του precision και recall ενός μονογράμμου, με το recall να έχει μεγαλύτερο βάρος από το precision. Έχει ακόμα πολλά χαρακτηριστικά που δεν βρίσκονται σε άλλες μετρικές όπως είναι το stemming και η αντιστοίχιση συνωνυμίας, καθώς και αντιστοίχιση της ακριβής λέξης.

Μεθοδολογία που ακολουθήθηκε με αναφορές στον κώδικα:

Για να μπορέσει να γίνει ο υπολογισμός της μετρικής METEOR, έγινε λήψη κώδικα που είχε αναπτυχθεί το 2014 από τους Michael Denkowski, Alon Lavie στο ινστιτούτο γλωσσικών τεχνολογιών του CMU. Σύμφωνα με πολλές πηγές αυτός ο κώδικας αποτελεί τον πιο αποτελεσματικό και αξιόπιστο τρόπο υπολογισμού της μετρικής Meteor. Ο κώδικας είναι φορμαρισμένος σε μορφή JAR (Java ARchiVe) πράγμα που σημαίνει ότι η οποιαδήποτε τροποποίηση του είναι πολύ δύσκολη. Για τον λόγο αυτό η παρουσίαση των αποτελεσμάτων γίνεται μόνο με τον τρόπο που επέλεξαν οι δημιουργοί (όπως φαίνεται παρακάτω).

Η εκτέλεση κώδικα γίνεται με την παρακάτω διαδικασία:

Αρχικά, πρέπει γίνει η είσοδος στο φάκελο που περιέχονται τα αρχεία (meteor-1.5.jar,S1.txt,S2.txt,Sr.txt).το οποίο γίνεται με την παρακάτω εντολή τερματικού

```
cd C:\Users\..μονοπάτι..\Meteor
```

Στην συνέχεια μπορεί να γίνει η εκτέλεση του κώδικα με την παρακάτω εντολή

```
java -jar meteor-1.5.jar hyp ref -l other
```

Όπου:

hyp:H υπόθεση μετάφρασης(S1,S2)

ref:H αναφορά μεταφρασής(Sr)

-l:Είναι ο προσδιορισμός της γλώσσας, που σε αυτή την περίπτωση του δόθηκε η τιμή other, ώστε να μπορέσουν να χρησιμοποιηθούν τα Ελληνικά. Εξαιτίας του γεγονότος ότι τα Ελληνικά δεν είναι μια πλήρως υποστηριζόμενη γλώσσα από το πρόγραμμα, δηλαδή δεν είναι μια από αυτές en cz de es f rap ,δεν μπορούν να χρησιμοποιηθούν οι επιλογές προ επεξεργασίας που παρέχει. Ευτυχώς η μοναδική προ επεξεργασία που έπρεπε να

εφαρμόσει στα κείμενα ήταν η αντικατάσταση όλων των χαρακτήρων που ακολουθούν ("«,»») με κενό ,αυτό έγινε εύκολα μέσω της λειτουργίας Find Replace του Microsoft Word.

Με την αντικατάσταση των παραμέτρων hyr με S1.txt και ref με Sr.txt προκύπτουν οι τιμές Meteor του S1 ανά γραμμή. Δηλαδή με την εντολή:

```
java -jar meteor-1.5.jar S1.txt Sr.txt -l other
```

προκύπτει:

```
Eval ID:      meteor-1.5-wo-other-no_norm-0.75_1.4_0.7_0.5-ex-1.0
Language:     Other
Format:       plaintext
Task:         Language-Independent
Modules:      exact
Weights:      1.0
Parameters:   0.75 1.4 0.7 0.5

Segment 1 score: 0.47407026112738765
Segment 2 score: 0.12000000000000002
Segment 3 score: 0.46839709569732185
Segment 4 score: 0.20326790736999606
Segment 5 score: 0.262455548993297
Segment 6 score: 0.0
Segment 7 score: 0.5349386973968236
Segment 8 score: 0.6708848399735584
Segment 9 score: 0.06315789473684211
Segment 10 score: 0.08275862068965519
Segment 11 score: 0.5790357634229341
Segment 12 score: 0.3784180553860403
Segment 13 score: 0.5119187184503197
Segment 14 score: 0.06666666666666668
Segment 15 score: 0.17288225902604243
Segment 16 score: 0.6151795020063472
Segment 17 score: 1.0
Segment 18 score: 0.6860763177409968
Segment 19 score: 0.3668696056661761
Segment 20 score: 0.5737764677639324
Segment 21 score: 0.16666666666666669
Segment 22 score: 0.407385917308892
Segment 23 score: 0.11614722424795107
Segment 24 score: 0.3766952069629737
Segment 25 score: 0.7195911151502076
Segment 26 score: 0.5111301571204733
Segment 27 score: 0.5737764677639324
Segment 28 score: 0.16666666666666669
Segment 29 score: 0.28666990881836496
Segment 30 score: 0.11614722424795107
```

```

Segment 31 score: 0.3766952069629737
Segment 32 score: 0.13333333333333336
Segment 33 score: 0.5111301571204733
Segment 34 score: 0.5737764677639324
Segment 35 score: 0.16666666666666669
Segment 36 score: 0.28666990881836496
Segment 37 score: 0.11614722424795107
Segment 38 score: 0.3766952069629737
Segment 39 score: 0.33985655860212294
Segment 40 score: 0.5111301571204733
Segment 41 score: 0.40874224843218376
Segment 42 score: 0.3874677580201822
Segment 43 score: 0.39425949738601757
Segment 44 score: 0.7195911151502076
Segment 45 score: 0.5111301571204733
Segment 46 score: 0.40874224843218376
Segment 47 score: 0.3874677580201822
Segment 48 score: 0.39425949738601757
Segment 49 score: 0.2332347856587065
Segment 50 score: 0.44662386776751883

System level statistics:

Stage      Test Matches      Reference Matches
Content  Function  Total  Content  Function  Total
1         527      3      530      527      3      530
Total      527      3      530      527      3      530

Test words: 954
Reference words: 970
Chunks: 285
Precision: 0.5555555555555556
Recall: 0.5463917525773195
f1: 0.550935509355508
fMean: 0.5486542443064182
Fragmentation penalty: 0.2936936972804033
Final score: 0.3875179507674806

```

Αντίστοιχα για το S2:

```
java -jar meteor-1.5.jar S2.txt Sr.txt -l other
```

προκύπτει:

```

Eval ID: meteor-1.5-wo-other-no_norm-0.75_1.4_0.7_0.5-ex-1.0
Language: Other
Format: plaintext
Task: Language-Independent
Modules: exact
Weights: 1.0
Parameters: 0.75 1.4 0.7 0.5

Segment 1 score: 1.0
Segment 2 score: 0.3619207598831857
Segment 3 score: 0.5723036486568135
Segment 4 score: 0.39822064736085794
Segment 5 score: 0.3243899671153557
Segment 6 score: 0.0
Segment 7 score: 0.5003332613130482
Segment 8 score: 0.5723036486568135
Segment 9 score: 0.06315789473684211
Segment 10 score: 0.8485234510432377
Segment 11 score: 1.0
Segment 12 score: 0.41451809660697125
Segment 13 score: 0.5608267106892598
Segment 14 score: 0.19563284318010038
Segment 15 score: 0.21934591508071863
Segment 16 score: 0.6581491959553356
Segment 17 score: 1.0
Segment 18 score: 0.7252181844840363
Segment 19 score: 0.2822392756124888
Segment 20 score: 0.4434443758938333
Segment 21 score: 0.5666503038697234
Segment 22 score: 0.3706976347273368
Segment 23 score: 0.19223106771409035
Segment 24 score: 0.44927884689334957
Segment 25 score: 0.5877996806885443
Segment 26 score: 0.417872505368467
Segment 27 score: 0.3928705531216446
Segment 28 score: 0.5666503038697234
Segment 29 score: 0.2452433388593389
Segment 30 score: 0.19223106771409035
Segment 31 score: 0.44927884689334957
Segment 32 score: 0.6915290361041697
Segment 33 score: 0.417872505368467
Segment 34 score: 0.3928705531216446

Segment 35 score: 0.4378596542376504
Segment 36 score: 0.2452433388593389
Segment 37 score: 0.19223106771409035
Segment 38 score: 0.44927884689334957
Segment 39 score: 0.5977143364365771
Segment 40 score: 0.417872505368467
Segment 41 score: 0.44584331794272036
Segment 42 score: 0.4504915675794858
Segment 43 score: 0.2799408203877871
Segment 44 score: 0.5877996806885443
Segment 45 score: 0.417872505368467
Segment 46 score: 0.44584331794272036
Segment 47 score: 0.4504915675794858
Segment 48 score: 0.2799408203877871
Segment 49 score: 0.2398047796209236
Segment 50 score: 0.3494391031392718

System level statistics:

      Test Matches
Stage  Content  Function  Total
1      518      3      521
Total  518      3      521

      Reference Matches
Stage  Content  Function  Total
1      518      3      521
Total  518      3      521

Test words: 875
Reference words: 970
Chunks: 252
Precision: 0.5954285714285714
Recall: 0.5371134020618556
f1: 0.564769647696477
fMean: 0.5505944517833553
Fragmentation penalty: 0.253212991730229
Final score: 0.4111767834172266

```

Δ΄.3.4 WER

Βιβλιογραφική ανασκόπηση της μετρικής WER:

WER(Word Error Rate):Είναι μια συνήθης μετρική της απόδοσης ενός συστήματος μηχανικής μετάφρασης και αναγνώρισης φωνής.

Η γενική δυσκολία για την μέτρηση απόδοσης στηρίζεται στο γεγονός ότι η αναγνωρισμένη σειρά από λέξεις ή υπόθεση (hypothesis) μπορεί να έχει διαφορετικό μέγεθος από την αναφερομένη σειρά από λέξεις (Reference) που υποτίθεται ότι είναι η σωστή. Η WER προκύπτει από την απόσταση Levenshtein, δουλεύοντας στο επίπεδο λέξης αντί για το επίπεδο φωνήματος. Η WER είναι ένα πολύτιμο εργαλείο για σύγκριση διαφορετικών συστημάτων αλλά και για εκτίμηση βελτιώσεων σε ένα σύστημα. Αυτού του είδους η μετρική ωστόσο, δεν παρέχει καθόλου λεπτομέρειες για την φύση των λαθών της μετάφρασης και επομένως χρειάζεται περαιτέρω δουλειά για την αναγνώριση των κυρίων πηγών σφαλμάτων.

Αυτό το πρόβλημα λύνεται με το να ευθυγραμμιστούν αρχικά οι σειρές από τις λέξεις της υπόθεσης με αυτές της αναφοράς χρησιμοποιώντας δυναμική ευθυγράμμιση χαρακτήρων.

Τότε το Word Error Rate μπορεί να υπολογιστεί όπως ακολουθεί:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}$$

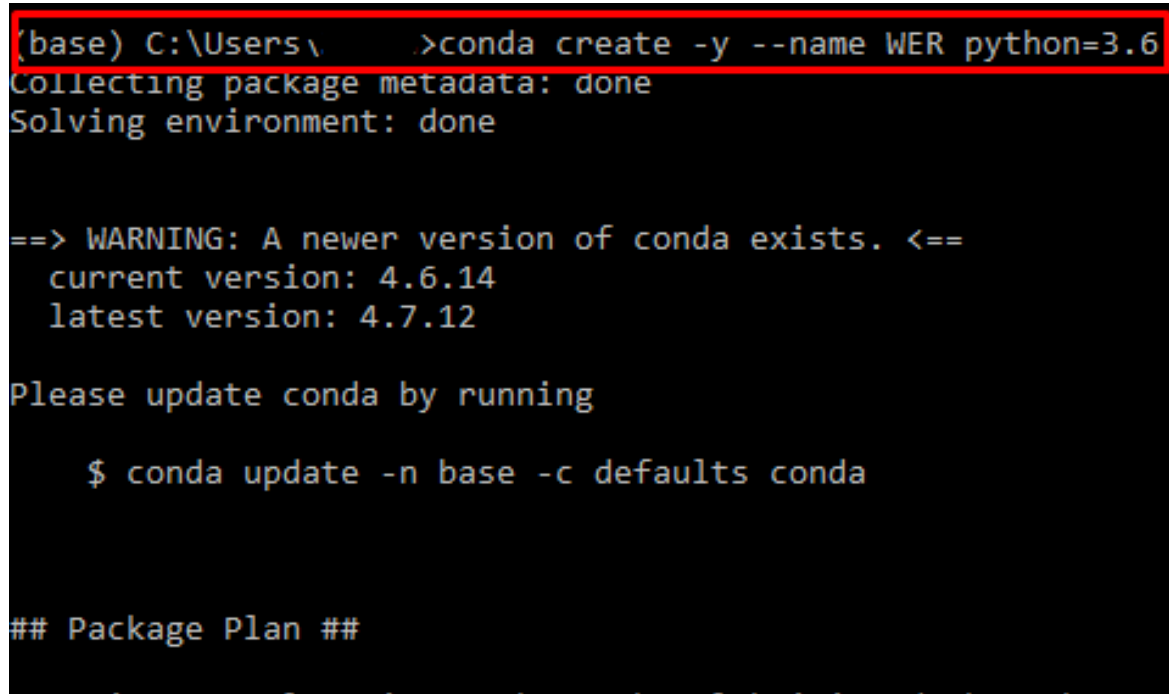
Όπου:

- S είναι ο αριθμός των αντικαταστάσεων
- D είναι ο αριθμός των διαγραφών
- I είναι ο αριθμός από τις προσθήκες
- C είναι ο αριθμός από τις σωστές λέξεις
- N είναι ο αριθμός των λέξεων στην αναφορά ($N=S+D+C$)

Μεθοδολογία που ακολουθήθηκε με αναφορές στον κώδικα:

Αρχικά το πρώτο πράμα που έγινε ήταν η δημιουργία ενός νέου περιβάλλοντος για να γίνει εφικτή η εκτέλεση κώδικα σε python 3, με την εντολή:

```
conda create -y --name WER python=3.6
```

A screenshot of a terminal window with a black background and white text. The first line shows the command prompt: (base) C:\Users\ >conda create -y --name WER python=3.6. This line is highlighted with a red rectangular border. The subsequent lines show the output of the command: Collecting package metadata: done, Solving environment: done, followed by a warning message about a newer version of conda existing (4.6.14 vs 4.7.12), and instructions to update conda. The terminal ends with the start of a package plan section: ## Package Plan ##.

```
(base) C:\Users\ >conda create -y --name WER python=3.6
Collecting package metadata: done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.6.14
  latest version: 4.7.12

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##
```

Μετά την είσοδο στο νέο περιβάλλον με την εντολή:

```
activate WER
```

έγινε η εγκατάσταση ενός απαραίτητου πακέτου, το numpy με την εντολή:

```
pip install numpy
```

Τέλος έγινε η εκτέλεση του κώδικα Main.py που έχω γράψει για την εξαγωγή των αποτελεσμάτων.

```

(WER) C:\Users\          \Desktop\wer_new>python Main.py
Line 1      The WER score of S1 is: 6      and S2 is: 0
Line 2      The WER score of S1 is: 3      and S2 is: 2
Line 3      The WER score of S1 is: 5      and S2 is: 4
Line 4      The WER score of S1 is: 18     and S2 is: 20
Line 5      The WER score of S1 is: 16     and S2 is: 15
Line 6      The WER score of S1 is: 1      and S2 is: 1
Line 7      The WER score of S1 is: 6      and S2 is: 4
Line 8      The WER score of S1 is: 6      and S2 is: 7
Line 9      The WER score of S1 is: 6      and S2 is: 6
Line 10     The WER score of S1 is: 6      and S2 is: 1
Line 11     The WER score of S1 is: 3      and S2 is: 0
Line 12     The WER score of S1 is: 31     and S2 is: 29
Line 13     The WER score of S1 is: 6      and S2 is: 4
Line 14     The WER score of S1 is: 9      and S2 is: 6
Line 15     The WER score of S1 is: 6      and S2 is: 6
Line 16     The WER score of S1 is: 3      and S2 is: 2
Line 17     The WER score of S1 is: 0      and S2 is: 0
Line 18     The WER score of S1 is: 3      and S2 is: 2
Line 19     The WER score of S1 is: 13     and S2 is: 14
Line 20     The WER score of S1 is: 15     and S2 is: 14
Line 21     The WER score of S1 is: 6      and S2 is: 3
Line 22     The WER score of S1 is: 14     and S2 is: 14
Line 23     The WER score of S1 is: 17     and S2 is: 14
Line 24     The WER score of S1 is: 20     and S2 is: 18
Line 25     The WER score of S1 is: 0      and S2 is: 1
Line 26     The WER score of S1 is: 8      and S2 is: 9
Line 27     The WER score of S1 is: 15     and S2 is: 14
Line 28     The WER score of S1 is: 6      and S2 is: 3
Line 29     The WER score of S1 is: 14     and S2 is: 14
Line 30     The WER score of S1 is: 17     and S2 is: 14
Line 31     The WER score of S1 is: 20     and S2 is: 18
Line 32     The WER score of S1 is: 3      and S2 is: 1
Line 33     The WER score of S1 is: 8      and S2 is: 9
Line 34     The WER score of S1 is: 15     and S2 is: 14
Line 35     The WER score of S1 is: 6      and S2 is: 4
Line 36     The WER score of S1 is: 14     and S2 is: 14
Line 37     The WER score of S1 is: 17     and S2 is: 14
Line 38     The WER score of S1 is: 20     and S2 is: 18
Line 39     The WER score of S1 is: 5      and S2 is: 5
Line 40     The WER score of S1 is: 8      and S2 is: 9
Line 41     The WER score of S1 is: 8      and S2 is: 8

Line 42     The WER score of S1 is: 17     and S2 is: 20
Line 43     The WER score of S1 is: 33     and S2 is: 37
Line 44     The WER score of S1 is: 0      and S2 is: 1
Line 45     The WER score of S1 is: 8      and S2 is: 9
Line 46     The WER score of S1 is: 8      and S2 is: 8
Line 47     The WER score of S1 is: 17     and S2 is: 20
Line 48     The WER score of S1 is: 33     and S2 is: 37
Line 49     The WER score of S1 is: 12     and S2 is: 12
Line 50     The WER score of S1 is: 9      and S2 is: 10

```

Δ'.4 ΧΡΗΣΗ PRE-TRAINED EMBEDDINGS ΣΕ ΚΩΔΙΚΑ

Βιβλιογραφική ανασκόπηση των “word embeddings” :

Ένα word embedding είναι μια κλάση από προσεγγίσεις για την αναπαράσταση λέξεων και εγγράφων χρησιμοποιώντας μια πυκνή αναπαράσταση από διανύσματα, όπου κάθε διάνυσμα αντιπροσωπεύει την προβολή μιας λέξης σε ένα συνεχές χώρο διανυσμάτων.

Η θέση της λέξης μέσα στο χώρο των διανυσμάτων μαθαίνεται από το κείμενο και βασίζεται στις γειτονικές λέξεις που περικυκλώνουν την λέξη όταν αυτή χρησιμοποιείται.

Η θέση της λέξης στον “μαθημένο” χώρο διανυσμάτων αναφέρεται ως το embedding της.

Ένας τρόπος που μπορεί να βρεθεί το embedding μιας λέξης είναι μέσω ενός μοντέλου βαθιάς μάθησης.

Το keras προσφέρει επίπεδο “layer” για embeddings που μπορεί να χρησιμοποιηθεί σε νευρωνικά δίκτυα πάνω σε δεδομένα κειμένου. Προϋποθέτει όμως, τα δεδομένα εισόδου να είναι κωδικοποιημένα σε ακραίους αριθμούς, έτσι ώστε κάθε λέξη να αναπαρίσταται από έναν μοναδικό ακέραιο. Αυτό το βήμα προ- επεξεργασίας των δεδομένων μπορεί να γίνει χρησιμοποιώντας το Tokenizer API (του οποίου ένα παράδειγμα εφαρμογής του φαίνεται παρακάτω στο κώδικα) που μαζί με το επίπεδο embedding παρέχεται από το keras.

Το επίπεδο embedding αρχικοποιείται με τυχαία βάρη και θα μάθει ένα embedding για όλες τις λέξεις.

Ένα παράδειγμα εξόδου για το embedding μιας λέξης θα ήταν αυτό που ακολουθεί:

Καλά -0.038194 -0.24487 0.72812..... -0.39961 0.083172 0.043953

Όπου στην αρχή δηλώνεται η λέξη που αντιστοιχεί το embedding (Καλά) και στην συνέχεια ακολουθούν κάποιοι αριθμοί που αναπαριστούν την θέση αυτής της λέξης σε κάθε διάσταση, ο αριθμός των διαστάσεων μπορεί να προσδιοριστεί από τον χρήστη.

Χρήση pre-trained word embeddings:

Το θέμα της παρούσας εργασίας είναι η χρήση pre trained word embeddings .Συγκεκριμένα ,το επίπεδο embedding μπορεί να χρησιμοποιήσει embedding που έχουν εξαχθεί από αλλού, έτσι ώστε η αρχικοποίηση των βαρών να μην γίνεται τυχαία. Σύμφωνα με την βιβλιογραφία, αυτό έχει πολλά οφέλη κυρίως στο ότι μπορεί να ελαττώσει τον χρόνο μάθησης αλλά και γενικά έχει καλύτερη απόδοση και ακρίβεια των αποτελεσμάτων.

Αναφορά στον κώδικα ανά στάδια με επισήμανση στα σημεία που έγιναν προσθήκες/αλλαγές κώδικα :

Όπως αναφέρθηκε και προηγουμένως σκοπός αυτής της εργασίας ήταν η χρήση pre trained word embeddings. Χρήση που θα γίνει πάνω σε κώδικα που υλοποιεί αρχικοποίηση τυχαίων βαρών στο επίπεδο embedding .

Ο κώδικας ξεκινάει με την εισαγωγή δυο επιπλέον βιβλιοθηκών:

```
In [57]: import pandas as pd
         from numpy import asarray
         import numpy as np
         from numpy import zeros
         from tensorflow.keras.preprocessing.text import one_hot
         from tensorflow.keras.preprocessing.sequence import pad_sequences
         from tensorflow.keras.preprocessing.text import Tokenizer
```

Έπειτα, πραγματοποιήθηκε η ανάγνωση των αρχείων που παρέμεινε ίδια με πριν.

```
dfS1 = pd.read_csv('data_gr_for_keras_S1.csv', delimiter = '\t', engine='python', quoting = 3)
dfS2 = pd.read_csv('data_gr_for_keras_S2.csv', delimiter = '\t', engine='python', quoting = 3)
dfR = pd.read_csv('data_gr_for_keras_R.csv', delimiter = '\t', engine='python', quoting = 3)
dfy = pd.read_csv('data_gr_for_keras_ANNOT1.csv', delimiter = ',')
```

```
docsS1 = dfS1['S1']
docsS2 = dfS2['S2']
docsR = dfR['Sr']
y = np.array(dfy['ANNOTATION'])
```

Στην συνέχεια έγινε η προετοιμασία του tokenizer API που αναφέρθηκε προηγουμένως για μετατροπή των λέξεων σε ακέραιους αριθμούς για κάθε ένα από τα κείμενα s1,s2,s0r:

```
In [6]: # prepare tokenizer
#s1
t_s1 = Tokenizer()
t_s1.fit_on_texts(docsS1)
vocab_size_s1 = len(t_s1.word_index) + 1
#s2
t_s2 = Tokenizer()
t_s2.fit_on_texts(docsS2)
vocab_size_s2 = len(t_s2.word_index) + 1
#sr
t_sr = Tokenizer()
t_sr.fit_on_texts(docsR)
vocab_size_sr = len(t_sr.word_index) + 1

<keras_preprocessing.text.Tokenizer object at 0x000002268CD7E748>
```

Αποθήκευση των πλέον κωδικοποιημένων κείμενων.

```
In [16]: # integer encode the documents
enc_S1 = t_s1.texts_to_sequences(docsS1)
enc_S2 = t_s2.texts_to_sequences(docsS2)
enc_R = t_sr.texts_to_sequences(docsR)
#print(enc_S1)
#print(enc_S2)
#print(enc_R)
```

Αν ήταν να εμφανίσουμε ένα από αυτά, το κείμενο(S1) θα είχε πάρει την παρακάτω μορφή:

```
In [15]: # integer encode the documents
enc_S1 = t_s1.texts_to_sequences(docsS1)
enc_S2 = t_s2.texts_to_sequences(docsS2)
enc_R = t_sr.texts_to_sequences(docsR)
print(enc_S1)
#print(enc_S2)
#print(enc_R)

[[40, 23, 41, 13, 24, 14, 15, 25, 42, 43], [44, 45, 46, 47, 4], [14, 15, 1, 25, 48, 49, 50, 5, 2, 6,
7], [51, 52, 53, 54, 55, 26, 56, 57, 8, 58, 59, 9, 13, 2, 60, 61, 1, 62, 63, 8, 27, 1, 64, 65, 66, 8,
27, 67, 1, 68, 69, 8, 28, 1, 16, 29], [16, 1, 10, 29, 70, 71, 72, 26, 30, 4, 73, 74, 75, 76, 77, 78,
2, 23, 79, 31, 10, 32, 3], [80], [81, 13, 24, 14, 15, 10, 17, 3, 1, 5, 2, 6, 7], [33, 82, 34, 83, 84,
18, 35, 10, 17, 3, 36, 85, 86, 19, 20, 30, 4, 87, 35, 9, 2, 11, 88], [89, 1, 12, 90, 91, 2, 16], [92,
2, 11, 93, 1, 9, 94, 95], [96, 97, 17, 3, 5, 2, 6, 7], [98, 1, 99, 100, 18, 101, 102, 103, 104, 105,
4, 106, 21, 22, 12, 107, 3, 108, 109, 110, 111, 112, 113, 11, 114, 2, 12, 115, 20, 37, 116, 117, 3, 1
9, 38, 36, 39, 118, 9, 119, 39, 120, 37, 11, 121, 122, 3, 19, 38, 123, 21, 124, 22, 125, 126, 31, 12
7, 128, 129], [33, 130, 131, 20, 1, 18, 1, 132, 4, 133, 21, 22, 28, 5, 2, 6, 7], [12, 134, 34, 135, 3
2, 3, 136, 137, 138]]
```

Επειδή το keras προτιμάει όλες οι σειρές να έχουν το ίδιο μέγεθος πρέπει τα κωδικοποιημένα κείμενα να περάσουν από την συνάρτηση `pad_sequences()` και θα έχουν μέγιστο μέγεθος 16

```
In [18]: # pad documents to a max length of N words
max_length= 16
padded_S1 = pad_sequences(enc_S1, maxlen=max_length, padding='post')
padded_S2 = pad_sequences(enc_S2, maxlen=max_length, padding='post')
padded_R = pad_sequences(enc_R, maxlen=max_length, padding='post')
#print(padded_S1)
#print(padded_S2)
#print(padded_R)
```

Η εμφάνιση του ιδίου κειμένου S1 παίρνει την μορφή:

```
In [19]: # pad documents to a max length of N words
max_length= 16
padded_S1 = pad_sequences(enc_S1, maxlen=max_length, padding='post')
padded_S2 = pad_sequences(enc_S2, maxlen=max_length, padding='post')
padded_R = pad_sequences(enc_R, maxlen=max_length, padding='post')
print(padded_S1)
#print(padded_S2)
#print(padded_R)
```

```
[[ 40 23 41 13 24 14 15 25 42 43  0  0  0  0  0  0]
 [ 44 45 46 47  4  0  0  0  0  0  0  0  0  0  0  0]
 [ 14 15  1 25 48 49 50  5  2  6  7  0  0  0  0  0]
 [ 27  1 64 65 66  8 27 67  1 68 69  8 28  1 16 29]
 [ 26 30  4 73 74 75 76 77 78  2 23 79 31 10 32  3]
 [ 80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 81 13 24 14 15 10 17  3  1  5  2  6  7  0  0  0]
 [ 10 17  3 36 85 86 19 20 30  4 87 35  9  2 11 88]
 [ 89  1 12 90 91  2 16  0  0  0  0  0  0  0  0  0]
 [ 92  2 11 93  1  9 94 95  0  0  0  0  0  0  0  0]
 [ 96 97 17  3  5  2  6  7  0  0  0  0  0  0  0  0]
 [ 11 121 122  3 19 38 123 21 124 22 125 126 31 127 128 129]
 [130 131 20  1 18  1 132  4 133 21 22 28  5  2  6  7]
 [ 12 134 34 135 32  3 136 137 138  0  0  0  0  0  0  0]]
```

Το επόμενο βήμα είναι η φόρτωση των pre trained word embedding αυτών καθαυτών από το αρχείο **grcorpus_def.vec.gz** το οποίο μετονομάστηκε σε **grcorpus_def_300d.txt** για δύο λόγους: Πρώτον 300d που δείχνει τον αριθμό των διαστάσεων κάθε embedding και .txt ώστε να είναι συμβατό με τον κώδικα.

```
In [20]: # load the whole embedding into memory
embeddings_index = dict()
f = open('grcorpus_def_300d.txt', encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

```
Loaded 1349088 word vectors.
```

Στην συνέχεια γίνεται η δημιουργία ενός πίνακα που αντιστοιχίζει την κάθε λέξη των κείμενων με το αντίστοιχο embedding και θα έχει ως περιεχόμενο το βάρος της κάθε λέξης ώστε να μπει έτοιμο στο επίπεδο embedding του keras :

```
In [63]: # create a weight matrix for words in training docs
embedding_matrix_s1 = zeros((vocab_size_s1, 300))
for word, i in t_s1.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_s1[i] = embedding_vector
print(embedding_matrix_s1)

embedding_matrix_s2 = zeros((vocab_size_s2, 300))
for word, i in t_s2.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_s2[i] = embedding_vector
#print(embedding_matrix_s2)

embedding_matrix_sr = zeros((vocab_size_sr, 300))
for word, i in t_sr.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_sr[i] = embedding_vector
#print(embedding_matrix_sr)
```

Το αποτέλεσμα για άλλη μια φορά για το κείμενο s1 θα είναι ο πίνακας λέξης-embedding όπου σε κάθε κελί θα έχει το βάρος:

```
In [63]: # create a weight matrix for words in training docs
embedding_matrix_s1 = zeros((vocab_size_s1, 300))
for word, i in t_s1.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_s1[i] = embedding_vector
print(embedding_matrix_s1)

embedding_matrix_s2 = zeros((vocab_size_s2, 300))
for word, i in t_s2.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_s2[i] = embedding_vector
#print(embedding_matrix_s2)

embedding_matrix_sr = zeros((vocab_size_sr, 300))
for word, i in t_sr.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_sr[i] = embedding_vector
#print(embedding_matrix_sr)
```

```
[[ 0.      0.      0.      ...  0.      0.
  0.      ]
 [ 0.014604 -0.01517 -0.1293  ...  0.10028 -0.13815001
 -0.018951 ]
 [ 0.094539 -0.12185 -0.099341 ...  0.32587001 -0.055404
  0.097727 ]
 ...
 [-0.21698  0.12515  0.0099077 ...  0.059531 -0.03948
  0.10476  ]
 [-0.15858001 -0.064845 -0.19183999 ...  0.24147999  0.028101
  0.33627  ]
 [ 0.0029136  0.028242 -0.099365 ...  0.11742  0.065964
  0.13755  ]]
```

Η τελευταία αλλαγή που έγινε στον κώδικα ήταν η αλλαγή των παραμέτρων του επιπέδου embedding για να είναι ενήμερες με τις αλλαγές αυτές:

```
# Create embeddings GLOBAL
#!INSERT_4!Replace old Embg with new one
#EmbG = Embedding(vocab_size, 16 , input_length=max_Length)
EmbG = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=max_length, trainable=False)
```

Ε΄ ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

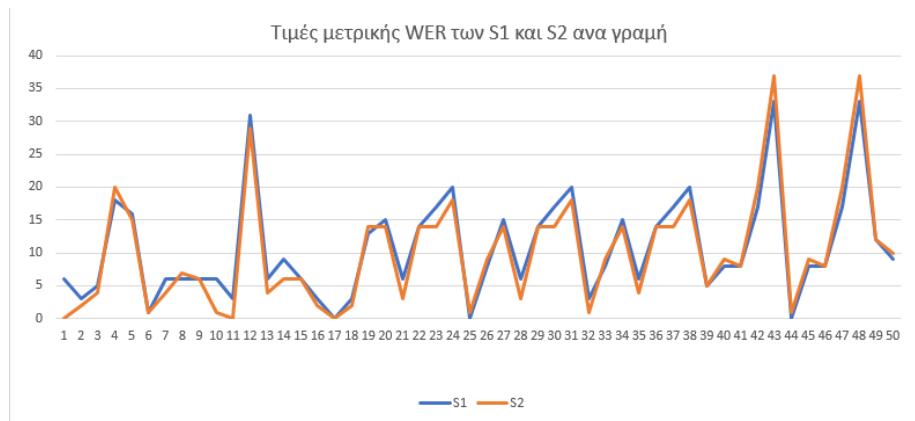
Ε΄.1 ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΠΟ ΤΟΝ ΥΠΟΛΟΓΙΣΜΟ ΤΩΝ ΜΕΤΡΙΚΩΝ

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, είναι κομβικής σημασίας για την ανάπτυξη μοντέλων μετάφρασης , πέρα από την δημιουργία τους να γίνει και ο υπολογισμός των αντιστοίχων μετρικών για υπολογισμό της απόδοσης τους .Αυτό θα έχει ως αποτέλεσμα όχι μόνο να προσδιοριστούν τυχόν λάθη σε αυτά και να εφαρμοστούν διορθώσεις βελτίωσης ,αλλά κυρίως θα διακριθούν εκείνα που είναι καλύτερα .

Στην παρούσα εργασία τα μοντέλα μετάφρασης που θα συγκριθούν είναι ,το στατιστικό και το νευρωνικό μοντέλο. Για να μπορέσει να γίνει η σύγκριση τους, και τα δυο μοντέλα έχουν δημιουργήσει το κάθε ένα μια μετάφραση ενός ίδιου κειμένου , η μετάφραση του στατιστικού συμβολίζεται με S1 ενώ του νευρωνικού με S2 .Αυτές οι μεταφράσεις που συχνά ονομάζονται και μεταφράσεις υπόθεσης θα συγκριθούν μέσω των μετρικών απόδοσης WER,METEOR και TER με μια ιδανική – επιθυμητή μετάφραση αναφοράς που συμβολίζεται με S_r .Το μοντέλο του οποίου η μετάφραση έχει καλύτερα αποτελέσματα στις μετρικές θα θεωρηθεί καλύτερο.

Η μετρική απόδοσης της οποίας τα αποτελέσματα θα εξεταστούν πρώτα είναι η WER (Word Error Rate). Αναλυτικότερα για κάθε πρόταση των μεταφράσεων υπόθεσης τόσο του στατιστικού όσο και του νευρωνικού μοντέλου θα προκύψει μια τιμή WER. Αυτή θα δείχνει την σύγκριση ανάμεσα σε αυτές τις προτάσεις ,με τις προτάσεις της μετάφρασης αναφοράς(ως προς την ύπαρξη επιπροσθέτων λέξεων στην πρόταση αναφοράς ,το σύνολο των αφαιρούμενων λέξεων στην πρόταση αναφοράς ,το σύνολο των αντικατεστημένων λέξεων στην πρόταση αναφοράς ,αλλά και το πλήθος των σωστών λέξεων που υπάρχουν στην πρόταση αναφοράς) .Αν η πρόταση υπόθεσης του ενός μοντέλου έχει τιμή wer μικρότερη από του αλλού, η πρόταση του είναι πιο κοντά σε αυτή της αναφοράς, άρα λέμε ότι το μοντέλο αυτό απέδωσε καλύτερη μετάφραση από το άλλο στην συγκεκριμένη πρόταση .Τέλος το μοντέλο με τις περισσότερες καλύτερες προτάσεις θα θεωρηθεί καλύτερο σύμφωνα με την μετρική απόδοσης WER.

Εξαιτίας του μεγάλου όγκου των κειμένου εξετάστηκαν οι πρώτες 50 γραμμές που θεωρούνται αντιπροσωπευτικό δείγμα από τα κείμενα S1 S2 και S_r,και αναλύθηκαν με τον τρόπο που παρουσιάστηκε παραπάνω. Τα αποτελέσματα από τις μετρήσεις φαίνονται αναλυτικά στον παρακάτω πίνακα (Πίνακας 1)



Πίνακας 1: Αποτελέσματα WER για στατιστικό και νευρωνικό μοντέλο

Στα σημεία όπου η πορτοκαλί γραμμή είναι κάτω από την μπλε σημαίνει ότι το νευρωνικό μοντέλο είχε καλύτερη τιμή WER από το στατιστικό, ομοίως η μπλε γραμμή όταν αυτή είναι κάτω από την πορτοκαλί σημαίνει ότι το στατιστικό είχε καλύτερη τιμή WER από το νευρωνικό. Η διαλογή των αποτελεσμάτων δείχνει ότι το νευρωνικό είχε σε 25 προτάσεις καλύτερες τιμές WER από ότι το στατιστικό. Σε 11 προτάσεις και τα δυο μοντέλα είχαν την ίδια τιμή και μόνο σε 14 το στατιστικό είχε καλύτερη τιμή από το νευρωνικό. Σύμφωνα με τα παραπάνω είναι ασφαλές να πούμε πως το νευρωνικό είναι καλύτερο από το στατιστικό μοντέλο σύμφωνα με την μετρική WER

Στην συνέχεια θα εξεταστούν τα αποτελέσματα της μετρικής απόδοσης μηχανικής μετάφρασης METEOR. Αυτή σε σύγκριση με άλλες μετρικές απόδοσης διαθέτει και άλλα χαρακτηριστικά όπως η αντιστοίχιση συνωνύμων άλλα και αντιστοίχιση παράγωγων λέξεων δηλαδή λέξεων που έχουν ίδια ρίζα αλλά διαφορετική κατάληξη. Σε αντίθεση με την WER το πρόγραμμα της METEOR που παρουσιάστηκε σε προηγούμενο κεφάλαιο βγάζει απευθείας το score από τις προτάσεις για κάθε μοντέλο. Δηλαδή βγάζει τους παρακάτω πίνακες (2 και 3)

Stage	Test Matches			Reference Matches		
	Content	Function	Total	Content	Function	Total
1	527	3	530	527	3	530
Total	527	3	530	527	3	530
Test words: 954						
Reference words: 970						
Chunks: 285						
Precision: 0.555555555555556						
Recall: 0.5463917525773195						
f1: 0.5509355509355508						
fMean: 0.5486542443064182						
Fragmentation penalty: 0.2936936972804033						
Final score: 0.3875179507674806						

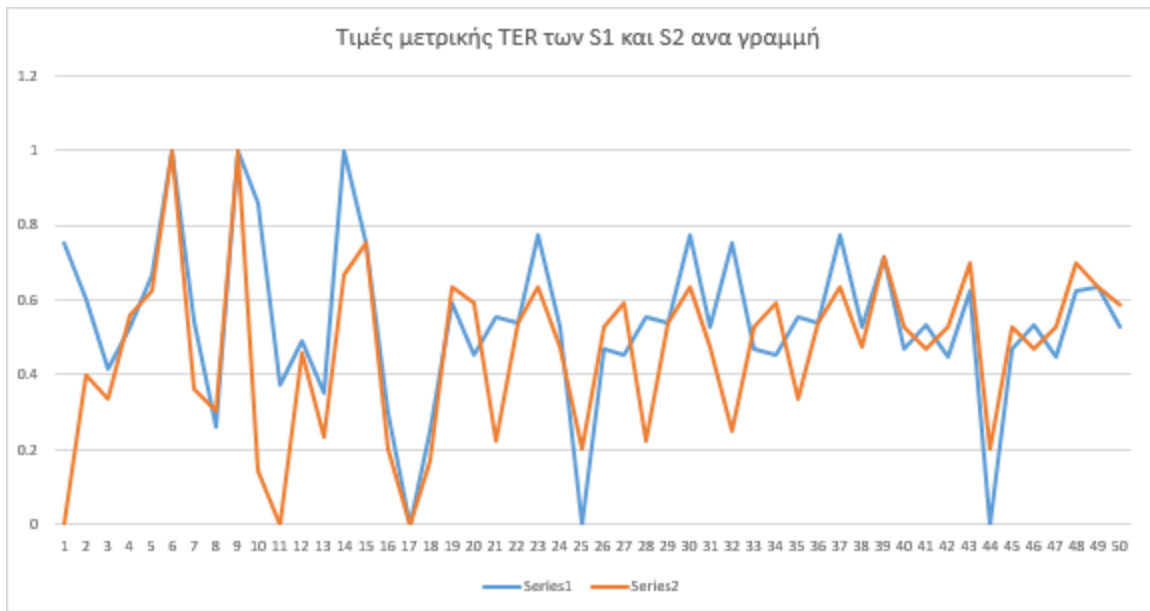
Πίνακας 2: Τελικό score METEOR για το στατιστικό μοντέλο

Stage	Test Matches			Reference Matches		
	Content	Function	Total	Content	Function	Total
1	518	3	521	518	3	521
Total	518	3	521	518	3	521
Test words: 875						
Reference words: 970						
Chunks: 252						
Precision: 0.5954285714285714						
Recall: 0.5371134020618556						
f1: 0.564769647696477						
fMean: 0.5505944517833553						
Fragmentation penalty: 0.253212991730229						
Final score: 0.4111767834172266						

Πίνακας 3:Τελικό score METEOR για το νευρωνικό μοντέλο

Και στην περίπτωση της μετρικής METEOR, τα αποτελέσματα δείχνουν πως το νευρωνικό μοντέλο είναι καλύτερο από το στατιστικό .Συγκεκριμένα το νευρωνικό σημείωσε τελικό score 0.411146 και το στατιστικό 0.387517

Η τελευταία μετρική που υπολογίστηκε ήταν η μετρική TER .Η TER αντικατοπτρίζει το ποσό της μετά επεξεργασίας που χρειάζεται το κείμενο αναφοράς ώστε να είναι ίδιο με το κείμενο υπόθεσης .Οι τιμές της κυμαίνονται από 0 έως 1 ,με 1 να συμβολίζει ότι η πρόταση χρειάζεται πολύ μετά επεξεργασία. Τα score των μοντέλων για την μετρική TER φαίνονται στον παρακάτω πίνακα 4 .



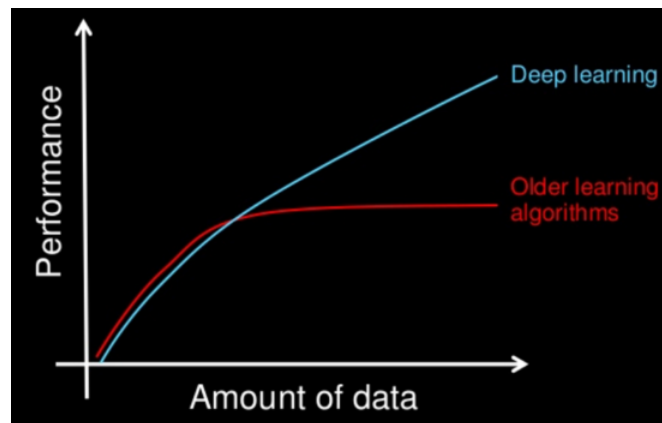
Πίνακας 4: Αποτελέσματά TER για στατιστικό και νευρωνικό μοντέλο

Στα σημεία όπου η πορτοκαλί γραμμή είναι κάτω από την μπλε ,σημαίνει ότι το νευρωνικό μοντέλο είχε καλύτερη τιμή TER από το στατιστικό , ομοίως για την μπλε γραμμή όταν αυτή είναι κάτω από την πορτοκαλί σημαίνει ότι το στατιστικό είχε καλύτερη τιμή TER από το νευρωνικό. Η διαλογή των αποτελεσμάτων δείχνει ότι το νευρωνικό είχε σε 24 προτάσεις καλύτερη τιμή TER από ότι το στατιστικό, σε 8 προτάσεις και τα δυο μοντέλα είχαν την ίδια τιμή και μόνο σε 18 το στατιστικό είχε καλύτερη τιμή από το νευρωνικό .Άρα και σε αυτή την περίπτωση εξάγεται το συμπέρασμα ότι το νευρωνικό είναι καλύτερο από το στατιστικό μοντέλο και σύμφωνα με την μετρική TER.

Ε΄.2 ΜΕΛΛΟΝΤΙΚΕΣ ΒΕΛΤΙΩΣΕΙΣ ΤΟΥ ΜΟΝΤΕΛΟΥ ΑΝΑΓΝΩΡΙΣΗΣ ΧΑΡΑΚΤΗΡΩΝ

Στο Κεφάλαιο Δ΄2 έγινε η δημιουργία ενός μοντέλου αναγνώρισης χειρόγραφων χαρακτήρων μαζί με την αξιολόγηση αυτού. Παρά το γεγονός ότι τόσο η μετρική της ακρίβειας (accuracy) όσο και του κόστους (Cost) σημείωσαν πολύ καλά αποτελέσματα, υπάρχει πάντα λίγο περιθώριο βελτιώσεων. Βελτιώσεις που περιγράφονται αναλυτικά παρακάτω

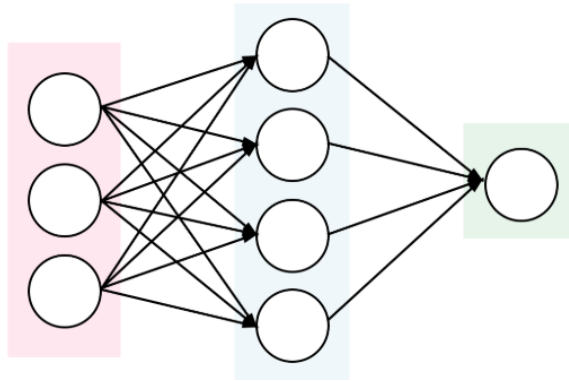
Η πρώτη και καλύτερη βελτίωση που μπορεί να εφαρμοστεί σε όλα τα μοντέλα βαθιάς μάθησης είναι η προσθήκη παραπάνω δεδομένων στο σετ εκπαίδευσης. Σε αντίθεση με τα μοντέλα μηχανικής μάθησης που η απόδοση τους σταθεροποιείται μετά από έναν συγκεκριμένο αριθμό παραδειγμάτων, τα μοντέλα βαθιάς μάθησης δεν έχουν όριο στα δεδομένα που μπορούν να πάρουν για να βελτιωθεί η απόδοση τους (Πίνακας 5)



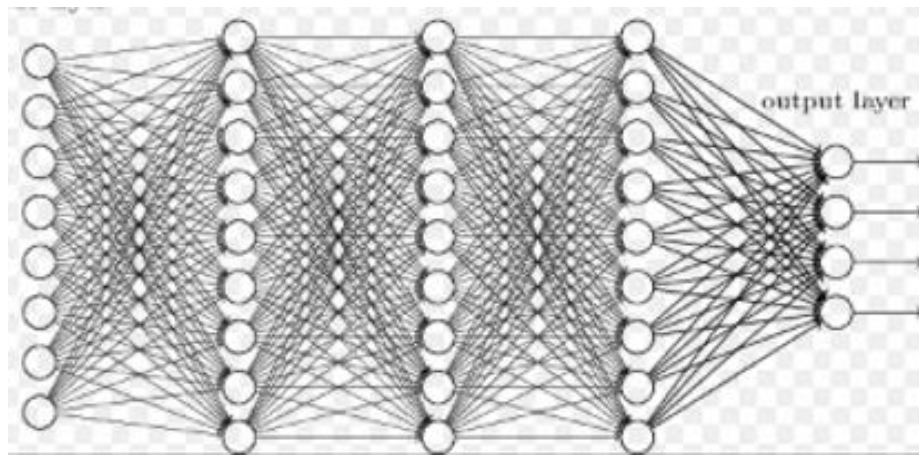
Πίνακας 5: Καμπύλη Απόδοσης – Μέγεθος δεδομένων (Πηγή:Machine Learning Mastery)

Η Δεύτερη τεχνική βελτίωσης που προτείνεται είναι η αναπροσαρμογή της τοπολογίας του νευρωνικού δικτύου. Δηλαδή, ή μεταβολή του αριθμού των νευρώνων σε κάθε κρυφό επίπεδο ή ακόμα και η αλλαγή του αριθμού των ίδιων των κρυφών επιπέδων. Δυστυχώς δεν υπάρχει κάποιος συγκεκριμένος κανόνας για το πώς θα γίνει η επιλογή των παραπάνω. Ο τρόπος με τον οποίο επιλέχτηκαν σε αυτή την εργασία ήταν με το να εξεταστούν αντίστοιχες εργασίες που προσπάθησαν να υλοποιήσουν παρόμοιο μοντέλο. Υπάρχουν όμως κάποιες τυποποιημένες αρχιτεκτονικές ή κάθε μια με κάποια ιδιαίτερα χαρακτηριστικά, οι πιο βασικές από αυτές αναγράφονται παρακάτω.

- Δίκτυο με ένα κρυφό επίπεδο αλλά πολλούς νευρώνες (Wide Neural Network)(Εικόνα 1)
- Δίκτυο με πολλά κρυφά επίπεδα αλλά λίγους νευρώνες σε κάθε ένα (Deep Neural Network)(Εικόνα 2)
- Συνδυασμός των παραπάνω



Εικόνα 1:Νευρωνικό δίκτυο με ένα κρυφό επίπεδο (Πηγή:Medium)



Εικόνα 2:Βαθύ Νευρωνικό δίκτυο με πολλά κρυφά επίπεδα (Πηγή:www. cse.iitb.ac.in)

Η Τρίτη και τελευταία τεχνική που προτείνεται είναι ο επαναπροσδιορισμός των epoch και Batch sizes. Το epoch συμβολίζει το πόσες φορές “πέρασε” το σετ εκπαίδευσης από το νευρωνικό δίκτυο και το Batch size καθορίζει την κλίση στην συνάρτηση κόστους (Κεφάλαιο Γ) δηλαδή το πόσο συχνά θα ενημερώνονται τα βάρη. Στην περίπτωση του δικτύου αυτής της εργασίας το Epoch ήταν 3 και το Batch size 6000, δηλαδή τα βάρη ενημερώνονταν με κάθε πέρασμα ενός παραδείγματος. Για άλλη μια φορά δεν υπάρχει κάποιος κανόνας για την επιλογή των παραπάνω, επιτρέποντας αρκετό χώρο για πειραματισμούς

Η΄ ΠΑΡΑΡΤΗΜΑ ΓΙΑ ΚΩΔΙΚΕΣ PYTHON

Η΄.1 ΚΩΔΙΚΕΣ WER

Main.py

```
import wer_algo as w
import re
import string

file_1=open("S1.txt",encoding="utf-8")
file_2=open("S2.txt",encoding="utf-8")
file_3=open("Sr.txt",encoding="utf-8")

for x in range(50):
    S1=file_1.readline()
    S1=re.sub('[!+string.punctuation+]', '', S1).split()
    S2=file_2.readline()
    S2=re.sub('[!+string.punctuation+]', '', S2).split()
    Sr=file_3.readline()
    Sr=re.sub('[!+string.punctuation+]', '', Sr).split()

    print("Line " ,str(x+1) , "\t The WER score of S1 is:" ,w.wer(Sr,S1) , "\t and S2 is:" ,w.wer(Sr,S2))
```

wer_algo.py

```
def wer(r, h):
    #initialisation
    import numpy
    d=numpy.zeros((len(r)+1)*(len(h)+1), dtype=numpy.uint8)
    d=d.reshape((len(r)+1, len(h)+1))
    for i in range(len(r)+1):
        for j in range(len(h)+1):
            if i==0:
                d[0][j]=j
            elif j==0:
                d[i][0]=i
    #computation
    for i in range(1, len(r)+1):
        for j in range(1, len(h)+1):
            if r[i-1]==h[j-1]:
                d[i][j]=d[i-1][j-1]
            else:
                substitution=d[i-1][j-1]+1
                insertion=d[i][j-1]+1
                deletion=d[i-1][j]+1
                d[i][j]=min(substitution, insertion, deletion)
    return d[len(r)][len(h)]

if __name__=="__main__":
    import doctest
    doctest.testmod()
```

Η.2 ΚΩΔΙΚΕΣ TER

TER.py

```
import io
import re
import string
import plotly.graph_objects as go

S1_re=[]
S2_re=[]

file_1=io.open("S1.txt","r",encoding='utf-8')
file_2=io.open("S2.txt","r",encoding='utf-8')
file_3=io.open("Sr.txt","r",encoding='utf-8')

for x in range(50):
    S1=file_1.readline()
    S1=re.sub('[\'+string.punctuation+']',' ',S1).split()
    S2=file_2.readline()
    S2=re.sub('[\'+string.punctuation+']',' ',S2).split()
    Sr=file_3.readline()
    Sr=re.sub('[\'+string.punctuation+']',' ',Sr).split()
    S1_re.append('%3f'%pyter.ter(S1,Sr))
    S2_re.append('%3f'%pyter.ter(S2,Sr))

fig=go.Figure(data=[go.Table(header=dict(values=['TER(S1,Sr)','TER(S2,Sr)']),
    cells=dict(values=[S1_re,S2_re]))])
fig.show()
```

Z' BIBΛΙΟΓΡΑΦΙΑ

Z'.1 BIBΛΙΑ

- Ian Goodfellow Yoshua Bengio and Aaron Courville - Deep Learning
- Josh Patterson and Adam Gibson - Deep Learning A Practitioner's Approach
- Yann LeCun, Yoshua Bengio & Geoffrey Hinton - Deep Learning
- Introduction to Deep Learning Sandro Shansi
- Natural Language Processing with Tensor Flow
- Yoshua Bengio - Learning Deep
- Stuart J. Russell and Peter Norvig - Artificial Intelligence
- Κοντούλης Ιωάννης -Μια ανασκόπηση της Βαθιάς Μάθησης
- Δημήτρης Κόλιας-Τεχνικές Βαθιάς Μηχανικής Μάθησης και Γνώσης
- Χριστόφορος Αναγνωστόπουλος -Βαθιά Μάθηση
- Skansi-S - Introduction to Deep-Learning

Z'.2 ΙΣΤΟΣΕΛΙΔΕΣ

MIT deep learning : <https://deeplearning.mit.edu/>

Pre trained word embeddings :https://keras.io/examples/pretrained_word_embeddings/

How to use pre trained embeddings : <https://sturzamihai.com/>