NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

COMPUTER SCIENCE DIVISION
IMAGE, VIDEO AND MULTIMEDIA SYSTEMS LABORATORY

Automatic Detection of Opinion Polarity from Twitter

DIPLOMA THESIS

of

ELENI MANDILARA

Supervisor: Stefanos Kollias
Professor NTUA

Athens, September 2015

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
COMPUTER SCIENCE DIVISION
IMAGE, VIDEO AND MULTIMEDIA SYSTEMS LABORATORY

Automatic Detection of Opinion Polarity from Twitter

DIPLOMA THESIS

of

ELENI MANDILARA

Supervisor: Stefanos Kollias
Professor NTUA

Approved by the three-member committee on the 18th of September 2015.

....	....	....
Stefanos Kollias	Kostas Karpouzis	Giorgos Stamou
Professor NTUA	Research Director of ICCS of NTUA	Assistant Professor NTUA

Athens, September 2015

....

Eleni E. Mandilara
Graduate Electrical and Computer Engineer of NTUA

Περίληψη

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η αυτόματη ανίχνευση της πολικότητας – ή, αλλιώς, η συναισθηματική ανάλυση – της γνώμης που εκφράζεται από χρήστες σε διαδικτυακές πηγές. Διαδικτυακή πηγή του ενδιαφέροντός μας αποτέλεσε το Twitter, το οποίο είναι μια online υπηρεσία κοινωνικής δικτύωσης που προσφέρει τη δυνατότητα σε χρήστες να δημοσιεύουν και να διαβάζουν σύντομα μηνύματα γνωστά ως *tweets*. Στόχος της συναισθηματικής ανάλυσης είναι η ταξινόμηση δειγμάτων κειμένου με βάση τη συνολική πολικότητα της γνώμης που εκφράζουν (θετική, αρνητική, ουδέτερη).

Με στόχο, λοιπόν, τη συναισθηματική ανάλυση των tweets, υλοποιήθηκε ένα σύστημα επιβλεπόμενης online μάθησης, βασισμένο στα τεχνητά νευρωνικά δίκτυα, το οποίο αποτελείται από δύο κύρια μέρη: το πρώτο ασχολείται με την προ-επεξεργασία των tweets, ενώ το δεύτερο υλοποιεί τη διαδικασία μάθησης μέσω του νευρωνικού δικτύου. Το πρώτο μέρος έχει ως στόχο τον εξωραϊσμό και καθαρισμό των tweets από θόρυβο ή περιττή πληροφορία, καθώς και τη μεταφορά τους σε μια μορφή κατανοητή από ένα υπολογιστικό σύστημα. Αυτό επιτυγχάνεται μέσω της εξαγωγής συγκεκριμένων χαρακτηριστικών και της κατασκευής των αντίστοιχων διανυσμάτων. Το δεύτερο μέρος, αυτό του νευρωνικού δικτύου, υλοποιείται από ένα πολυεπίπεδο perceptron, το οποίο εκπαιδεύεται μέσω του αλγορίθμου οπίσθιας διάδοσης σφάλματος, και αποβλέπει στην ορθή ταξινόμηση των tweets σε δύο ή τρεις διακριτές κλάσεις, οι οποίες αντιστοιχούν στην πολικότητα της γνώμης που εκφέρεται ανά tweet (θετική, αρνητική ή θετική, αρνητική, ουδέτερη).

Λέξεις – κλειδιά: ανάλυση συναισθήματος, μηχανική μάθηση, Twitter,
τεχνητά νευρωνικά δίκτυα, αλγόριθμος οπίσθιας διάδοσης σφάλματος, εξόρυξη δεδομένων.

Εκτεταμένη Περίληψη

Η παρούσα διπλωματική εργασία έχει ως θέμα την αυτόματη ανίχνευση της πολικότητας της γνώμης στο Twitter, ή αλλιώς τη συναισθηματική ανάλυση (sentiment analysis) στο Twitter. Στο πρώτο κεφάλαιο θα δούμε μια εισαγωγή για την ανάλυση συναισθήματος στο Twitter· στη συνέχεια (δεύτερο κεφάλαιο) θα μιλήσουμε για τα τεχνητά νευρωνικά δίκτυα πάνω στα οποία βασίστηκε το μοντέλο που υλοποιήθηκε. Κατόπιν, στο τρίτο κεφάλαιο θα ασχοληθούμε αναλυτικά με το συγκεκριμένο μοντέλο, και στο τέταρτο κεφάλαιο δούμε τα πειράματα που διεξάχθηκαν και τα αποτελέσματά τους. Θα μιλήσουμε επίσης για τα συμπεράσματα που προκύπτουν από αυτά και θα δώσουμε κάποιες ιδέες για τη βελτίωση του μοντέλου.

Ένα σύστημα ανάλυσης συναισθήματος στο Twitter ουσιαστικά λαμβάνει εισόδους από το Twitter και παράγει ως έξοδο την πολικότητα της κάθε εισόδου, που στην περίπτωσή μας μπορεί να ανήκει σε μία από 3 κλάσεις: θετική, αρνητική, ουδέτερη.

Το Twitter είναι μια online υπηρεσία κοινωνικής δικτύωσης που δίνει τη δυνατότητα σε χρήστες να στέλνουν και να διαβάζουν σύντομα μηνύματα γνωστά ως tweets. Η ανάλυση συναισθήματος στο Twitter έχει ως αντικείμενο την ανίχνευση της διάθεσης που είναι παρούσα σε ένα tweet που έχει δημοσιευθεί από κάποιον χρήστη του Twitter. Η διάθεση αυτή μπορεί να είναι μία γνώμη ή κριτική, η συναισθηματική κατάσταση του συγγραφέα ή η συναισθηματική επίδραση που στοχεύει ο συγγραφέας του να έχει στον αναγνώστη.

Τα κίνητρα για τη συγκεκριμένη εργασία είχαν να κάνουν αρχικά με το γεγονός ότι ο τομέας της ανάλυσης συναισθήματος έχει πολλές ιδιαίτερα ενδιαφέρουσες και χρήσιμες εφαρμογές όπως την ταξινόμηση συναισθήματος, την ανάκτηση και σύνοψη γνώμης, την αναγνώριση του κατόχου μιας γνώμης, την παρακολούθηση του θέματος/συναισθήματος, την ανίχνευση spam γνώμης, την πρόβλεψη ανθρωπίνων συμπεριφορών, αγοραστικών τάσεων, αποτελεσμάτων εκλογών κλπ. Επιπλέον, το Twitter συγκεντρώνει ορισμένα χαρακτηριστικά που το καθιστούν εξαιρετική πηγή για ανάλυση συναισθήματος από μικρά κείμενα (microtexts): έχει ανώτατο όριο 140 χαρακτήρων, συνηθίζεται η χρήση hashtags που συχνά ομαδοποιούν τα tweets ανά θέμα/συναίσθημα, η πρόσβαση στο Twitter είναι ιδιαίτερα εύκολη και μέσω πολλών μέσων (της ιστοσελίδας, της εφαρμογής για κινητές συσκευές, ή SMS), και εκατομμύρια tweets μοιράζονται καθημερινά σε πραγματικό χρόνο. Το Twitter μπορεί να αποτελεί μια αφετηρία, αλλά η αναγνώριση του συναισθήματος από γραπτή πληροφορία γενικότερα από μία μηχανή είναι ένα βήμα προς την ανάπτυξη της τεχνητής νοημοσύνης.

Παρά τα παραπάνω χαρακτηριστικά που καθιστούν το Twitter πολύ καλή πηγή για την εφαρμογή της συναισθηματικής ανάλυσης, ορισμένα χαρακτηριστικά των tweets κάνουν τη συγκεκριμένη εργασία μια πολύ πολύπλοκη και δύσκολη πρόκληση. Αυτά είναι τα εξής: (α) το μικρό μήκος δεν επιτρέπει γλωσσολογική ανάλυση, (β) συχνά χρησιμοποιούνται μη κανονικές λέξεις ή λέξεις σε μη συνεπή γραπτή μορφή, (γ) γίνεται χρήση ιδιωματισμών καθημερινής γλώσσας (slang), (δ) συχνά γίνονται ορθογραφικά

λάθη, (ε) χρησιμοποιούνται τοπικές λέξεις ή εκφράσεις, (στ) χρησιμοποιούνται emoticons, ακρωνύμια και hashtags.

Υπάρχουν διάφορες τεχνικές για τη συναισθηματική ανάλυση, όπως ο αλγόριθμος Naive Bayes, ο ταξινομητής Maximum Entropy, οι μηχανές διανυσμάτων υποστήριξης, τα τεχνητά νευρωνικά δίκτυα κ.ά. Στην παρούσα εργασία το μοντέλο που υλοποιήθηκε βασίζεται στα τεχνητά νευρωνικά δίκτυα.

Ένα νευρωνικό δίκτυο είναι ένας τεράστιος, παράλληλος επεξεργαστής, κατανεμημένης αρχιτεκτονικής που αποτελείται από απλές μονάδες επεξεργασίας (νευρώνες), που έχουν τη δυνατότητα να αποθηκεύουν εμπειρική γνώση και να την καθιστούν διαθέσιμη για χρήση. Η γνώση αυτή λαμβάνεται από το περιβάλλον του δικτύου και αποθηκεύεται μέσω των συναπτικών βαρών των συνδέσεων μεταξύ των νευρώνων.

Το perceptron του Rosenblatt αποτελεί το πρώτο νευρωνικό δίκτυο. Βασίζεται σε έναν μη γραμμικό νευρώνα στη βάση ενός μη γραμμικού συνδυαστή ακολουθούμενου από έναν απότομο περιοριστή (συνάρτηση προσήμου). Το perceptron έχει τη δυνατότητα να ταξινομεί δείγματα σε γραμμικά διαχωρίσιμες κλάσεις. Η εξέλιξή του, το πολυεπίπεδο perceptron διαθέτει 1 ή περισσότερα κρυφά επίπεδα νευρώνων μεταξύ εισόδου και εξόδου, πλήρως συνδεδεμένα μεταξύ τους, και έχει τη δυνατότητα να ταξινομεί δείγματα σε μη γραμμικά διαχωρίσιμες κλάσεις. Κάθε νευρώνας του πολυεπίπεδου perceptron υπολογίζει 2 σήματα: ένα λειτουργικό σήμα, και μια εκτίμηση του διανύσματος κλίσης (των κλίσεων της επιφάνειας σφάλματος σε σχέση με τα βάρη που είναι συνδεδεμένα στις εισόδους ενός νευρώνα).

Για την εκπαίδευση του πολυεπίπεδου perceptron χρησιμοποιείται ο αλγόριθμος οπίσθιας διάδοσης σφάλματος (error back propagation algorithm), ο οποίος χωρίζεται σε δύο φάσεις. Στην πρώτη φάση γίνεται το πέρασμα προς τα εμπρός, κατά το οποίο τα συναπτικά βάρη παραμένουν αμετάβλητα σε όλο το δίκτυο και υπολογίζονται τα λειτουργικά σήματα από νευρώνα σε νευρώνα, ως

$$y_j(n)=\varphi(\upsilon_j(n))$$

όπου φ η συνάρτηση ενεργοποίησης, y_j το λειτουργικό σήματα στην έξοδο του νευρώνα j και υ_j το τοπικό πεδίο του νευρώνα j:

$$\upsilon_j(n)=\sum_{i=0}^{m} w_{ji}(n)\, y_i(n)$$

Μόλις τερματίσει η πρώτη φάση, δηλαδή μόλις υπολογιστούν τα λειτουργικά σήματα ως και το επίπεδο εξόδου, εκκινεί η δεύτερη φάση, το πέρασμα προς τα πίσω. Σε αυτή τη φάση, στέλνονται σήματα σφάλματος από δεξιά προς τα αριστερά σε όλα τα επίπεδα του δικτύου, επίπεδο προς επίπεδο, υπολογίζοντας αναδρομικά την τοπική κλίση δ για κάθε νευρώνα, και μεταβάλλοντας τα συναπτικά βάρη όλων των συνδέσεων σύμφωνα με τον κανόνα Δέλτα:

$$\begin{pmatrix} \text{διόρθωση βάρους} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{παράμετρος ρυθμού μάθησης} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{τοπική κλίση} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \text{σήμα εισόδου του νευρώνα } j \\ y_j(n) \end{pmatrix}$$

Η μέθοδος cross-validation χρησιμοποιείται για τον έλεγχο του μοντέλου με βάση ένα υποσύνολο δεδομένων διαφορετικό από αυτό που χρησιμοποιήθηκε για την εκτίμηση των παραμέτρων του δικτύου. Για το λόγο αυτό, το δείγμα εκπαίδευσης χωρίζεται σε δύο ξένα μεταξύ τους υποσύνολα: ένα υποσύνολο εκτίμησης για την επιλογή του μοντέλου και ένα υποσύνολο επικύρωσης για τον έλεγχο του μοντέλου. Η K-fold cross-validation είναι μια παραλλαγή αυτής της μεθόδου, κατά την οποία το διαθέσιμο δείγμα μεγέθους N διαιρείται σε K υποσύνολα, με $1<K<N$. Για K δοκιμές, το μοντέλο εκπαιδεύεται με όλα τα υποσύνολα εκτός από ένα, διαφορετικό κάθε φορά.

Το μοντέλο της παρούσας εργασίας υλοποιείται από ένα σύστημα επιβλεπόμενης online μάθησης, που αποτελείται από 4 μέρη: (1) απόκτηση δεδομένων, (2) προ-επεξεργασία δεδομένων, (3) εξαγωγή χαρακτηριστικών και κατασκευή των αντίστοιχων διανυσμάτων χαρακτηριστικών, (4) νευρωνικό δίκτυο.

Τα δεδομένα είναι απαραίτητα για την εκπαίδευση και τη δοκιμή του μοντέλου μας. Για το λόγο αυτό, αποκτήθηκαν δεδομένα από το Twitter, στα οποία προστέθηκαν ετικέτες ("θετικό", "αρνητικό", "ουδέτερο") ανά tweet και τελικά δημιουργήθηκαν τα σύνολα δεδομένων. Για την απόκτηση αυτών των δεδομένων, δημιουργήθηκε μια εφαρμογή στο Twitter API και έγινε χρήση του OAuth ώστε να δοθεί έγκριση σε αυτή να επικοινωνεί με το Twitter μέσω του λογαριασμού μας.

Τα tweets που αναζητήθηκαν και αποθηκεύθηκαν είχαν κοινό θέμα το Δημοψήφισμα που έλαβε χώρα στην Ελλάδα την 5η Ιουλίου 2015. Ένα δημοψήφισμα από τη φύση του έχει την τάση να πολώνει τις γνώμες και θεωρήθηκε εξαιρετική ευκαιρία για εύρεση tweets με έντονη πολικότητα γνώμης. Αρχικά συλλέχθηκαν κατά μέσο όρο 2,191 tweets ανά ημέρα, από την 1η Ιουλίου έως και την 7η Ιουλίου 2015. Λόγω του μεγάλου όγκου δεδομένων, πριν από το στάδιο της προσθήκης ετικετών, τα tweets υποβλήθηκαν σε κάποια βήματα προ-επεξεργασίας, ώστε να εξαλειφθούν τα διπλότυπα και η περιττή πληροφορία. Μετά από το παραπάνω στάδιο, έμειναν 1,095 tweets ανά ημέρα, δηλαδή επιτεύχθηκε μία μείωση του όγκου (χάρη στην εξάλειψη της περιττής πληροφορίας) κατά 50%.

Η προσθήκη των ετικετών στα παραπάνω δεδομένα έγινε "χειροκίνητα". Υπάρχουν τεχνικές αυτόματης προσθήκης ετικετών με βάση hashtags ή emoticons, ωστόσο θεωρήθηκαν ακατάλληλες σε αυτή την περίπτωση καθώς το θέμα των tweets είναι πολιτικό, και αφενός δε συνηθίζεται η έντονη χρήση emoticons, αφετέρου συχνά μπορεί να συνδέεται με ειρωνική διάθεση. Λόγω πολύ μεγάλου όγκου δεδομένων, η προσθήκη ετικετών έγινε μόνο σε ένα μέρος των δεδομένων· σε 500 tweets ανά ημέρα, δηλαδή $500 \cdot 7 = 3{,}500$ tweets συνολικά. Συχνά παρατηρήθηκε μεγάλη δυσκολία στην εξαγωγή συμπερασμάτων σχετικά με την ακριβή γνώμη ενός χρήστη, καθώς η αλληλουχία των λέξεων εξέφραζε συχνά ασαφή ή διφορούμενη διάθεση και ο αριθμός των κλάσεων της πολικότητας ήταν πολύ μικρός (3:

θετική, αρνητική, ουδέτερη) ώστε να γίνει μια ακριβής ταξινόμηση της γνώμης. Η προσθήκη των ετικετών έγινε από τον ίδιο εκπαιδευτή για κάθε σύνολο συλλεγμένων δεδομένων ανά ημέρα, επομένως τυχούσες παρεκκλίσεις κατά την ταξινόμησή τους θεωρούνται ομοιογενείς και αμελητέες. Γίνεται απόλυτα σαφές από τα παραπάνω ότι αποτελεί πολύ μεγάλη και πολύπλοκη πρόκληση για ένα αυτόματο σύστημα η ανίχνευση της πολικότητας της γνώμης, όταν ακόμα και ένας άνθρωπος δεν μπορεί να εξάγει βέβαια και ακριβή συμπεράσματα πάντα.

Για την εκπαίδευση και τη δοκιμή του μοντέλου χρησιμοποιήθηκαν 2 σύνολα δεδομένων:

- ***balanced_referendum_ds***: αποτελείται από 2,100 tweets με κοινό θέμα το Δημοψήφισμα της 5ης Ιουλίου 2015, που αποθηκεύθηκαν και σχολιάστηκαν ως θετικά, αρνητικά ή ουδέτερα: 700 θετικά, 700 αρνητικά και 700 ουδέτερα tweets.
- ***various_contents_ds***: αποτελείται από 2,000 ήδη σχολιασμένα tweets, διαφόρων περιεχομένων και θεμάτων: 1,000 θετικά και 1,000 αρνητικά tweets.

Λόγω διαφόρων χαρακτηριστικών που αναφέρθηκαν παραπάνω, είναι φανερό ότι η γλώσσα του Twitter χρίζει ιδιαίτερης μεταχείρισης. Συνεπώς, αναπτύχθηκαν βήματα προ-επεξεργασίας των tweets με σκοπό τον καθαρισμό τους από θόρυβο και περιττή πληροφορία και τη μεταφορά τους σε μια μορφή κατανοητή από ένα αυτόματο σύστημα. Αυτά τα βήματα είναι τα εξής:

1) Μετατροπή τους σε πεζούς χαρακτήρες
2) Μετατροπή των υπερσυνδέσμων (συμβολοσειρών που ξεκινούν από “www.”, “http://” και “https://”) στη γενική συμβολοσειρά “url”
3) Μετατροπή των αναφορών σε ονόματα (“@username”) στη γενική συμβολοσειρά “at_user”
4) Απαλοιφή περισσοτέρων του ενός κενών
5) Αντικατάσταση των hashtags (“#hashtag”) με τη λέξη που περιέχουν χωρίς το σύμβολο της δίεσης “#” (“hashtag”)
6) Απαλοιφή σημείων στίξης
7) Απαλοιφή συμβολοσειρών “url”, “at_user”, “rt”
8) Απαλοιφή διπλότυπων
9) Αντικατάσταση emoticons με το συναίσθημα που εκφράζουν
10) Αντικατάσταση ακρωνυμίων με την πλήρη φράση
11) Απαλοιφή stop words (συνήθεις λέξεις της γλώσσας που δεν προσφέρουν ουσιώδη γνώση στο σύστημα)
12) Απαλοιφή εναπομεινάντων ASCII χαρακτήρων (π.χ 香)

Το επόμενο στάδιο του μοντέλου που υλοποιήθηκε έχει να κάνει με την εξαγωγή των χαρακτηριστικών που κρίθηκαν σημαντικά, και την κατασκευή των αντίστοιχων διανυσμάτων χαρακτηριστικών. Τα χαρακτηριστικά της κάθε λέξης που ενδιέφεραν την παρούσα υλοποίησή είναι: το λήμμα (lemma), το στέλεχος (stem) και το μέρος του λόγου της (part-of-speech, POS) της κάθε λέξης. Η διαδικασία της εξαγωγής των χαρακτηριστικών των λέξεων εξάγει τα παραπάνω χαρακτηριστικά για κάθε λέξη ανά tweet, και, στη συνέχεια, με τον κατάλληλο συνδυασμό τους κατασκευάζει τα διανύσματα χαρακτηριστικών, τα οποία στην ουσία είναι πλειάδες (tuples) που έχουν συντεθεί από έναν συνδυασμό της αρχικής λέξης και των χαρακτηριστικών της.

Πριν από το στάδιο του νευρωνικού δικτύου, έγιναν κάποια βήματα για την προετοιμασία των εισόδων του έτσι ώστε να είναι σε μορφή κατανοητή από αυτό. Χρησιμοποιήθηκε μια συνάρτηση hashing για την κωδικοποίηση του κάθε διανύσματος χαρακτηριστικών σε έναν μοναδικό ακέραιο αριθμό. Κάθε tweet αντιπροσωπεύεται από μια λίστα ακεραίων και έτσι δίνεται ως είσοδος στο δίκτυο. Επίσης, έγινε "γέμισμα" (padding) της κάθε λίστας ακεραίων με μηδενικά (τόσα όση η διαφορά της καθεμίας με τη λίστα του μεγίστου μήκους), έτσι ώστε να έχουν όλες κοινό μήκος.

Έγινε δοκιμή της τεχνικής των n-grams, για n=1 (μονογράμματα, unigrams) και n=2 (διγράμματα, bigrams). Τα n-grams είναι μία συνεχής ακολουθία από n λέξεις ενός tweet. Μια λίστα διανυσμάτων που έχει προκύψει από ένα tweet, με την τεχνική των μονογραμμάτων, αναπαρίσταται από μια όμοια λίστα, ενώ με την τεχνική των διγραμμάτων, αναπαρίσταται από μια λίστα αποτελούμενη από πλειάδες 2 διανυσμάτων: από το κάθε διάνυσμα συνδυασμένο με το επόμενό του. Χρησιμοποιήθηκε ξανά η παραπάνω συνάρτηση hashing για την κωδικοποίηση των διγραμμάτων ως ακεραίους.

Το νευρωνικό δίκτυο υλοποιείται από ένα πολυεπίπεδο perceptron 3 επιπέδων και εκπαιδεύεται από τον αλγόριθμο BK. Το επίπεδο εισόδου υλοποιείται από ένα γραμμικό επίπεδο, που εφαρμόζει έναν απλό πολλαπλασιασμό μεταξύ εισόδων και συναπτικών βαρών και προσθέτει την πόλωση. Στη συνέχεια, το κρυφό επίπεδο εφαρμόζει τη συνάρτηση υπερβολικής εφαπτομένης (tanh) στις εξόδους του προηγούμενου επιπέδου. Τέλος, το επίπεδο εξόδου είναι ένα γραμμικό επίπεδο. Παρακάτω φαίνεται ο συμβολισμός που χρησιμοποιήθηκε:

$$\begin{aligned}
d_{input} &= \textit{διάσταση επιπέδου εισόδου}\\
d_{hidden} &= \textit{διάσταση κρυφού επιπέδου}\\
d_{output} &= \textit{διάσταση επιπέδου εξόδου}\\
w_{12} &= \textit{διάνυσμα βαρών από το επίπεδο εισόδου στο κρυφό επίπεδο}\\
w_{23} &= \textit{διάνυσμα βαρών από το κρυφό επίπεδο στο επίπεδο εξόδου}
\end{aligned}$$

Οι διάφορες αρχιτεκτονικές που εξετάσθηκαν κρατούν σταθερές τις διαστάσεις των επιπέδων εισόδου και εξόδου και δοκιμάζουν διαφορετικές διαστάσεις στο κρυφό επίπεδο, όπως φαίνεται στον παρακάτω πίνακα:

Αρχιτεκτονική	**Διαστάσεις Επιπέδων**		
	Επίπεδο Εισόδου	**Κρυφό Επίπεδο**	**Επίπεδο Εξόδου**
1	d_{input}	d_{input}	d_{output}
2	d_{input}	d_{input} *div 2*	d_{output}
3	d_{input}	d_{input} *div 4*	d_{output}
4	d_{input}	$d_{input} \cdot 2$	d_{output}
5	d_{input}	$d_{input} \cdot 2$ *div 3*	d_{output}
6	d_{input}	$d_{input} \cdot 4$	d_{output}
7	d_{input}	$(d_{input} + d_{output}) \cdot 3$ *div 2*	d_{output}
8	d_{input}	$(d_{input} + d_{output}) \cdot 2$ *div 3*	d_{output}

Οι αρχιτεκτονικές εξετάστηκαν σε δύο πειράματα: στο 1ο τα διανύσματα χαρακτηριστικών συντίθεται από την αρχική λέξη, το λήμμα λέξης και το μέρος του λόγου, ενώ στο 2ο συντίθενται από την αρχική λέξη, το στέλεχός της και το μέρος του λόγου. Ακόμα εξετάστηκαν 2 τεχνικές, αυτή των μονογραμμάτων και αυτή των διγραμμάτων. Μέτρο σύγκλισης του δικτύου αποτέλεσε το μέσο τετραγωνικό σφάλμα (ΜΤΣ).

Για το 1ο σύνολο δεδομένων, εξετάστηκαν όλα τα αρχιτεκτονικά σχήματα, για καθένα από τα 2 πειράματα με την τεχνική των μονογραμμάτων. Η διάσταση του επιπέδου εισόδου είναι 26 ενώ η διάσταση του επιπέδου εξόδου 3 (για την ταξινόμηση των tweets σε 3 κλάσεις). Το 75% των tweets του συνόλου χρησιμοποιήθηκε για την εκπαίδευση του δικτύου, ενώ το 25% για τη δοκιμή του. Χρησιμοποιήθηκε η μέθοδος K-fold cross-validation, για K=5. Παρατηρήθηκε ότι κάθε αρχιτεκτονική συνέκλινε σε ένα περίπου ίδιο ΜΤΣ, κατά μέσο όρο ίσο με 0.147. Μετά τη σύγκλιση του δικτύου, ξεκίνησε η φάση της δοκιμής του. Η καλύτερη απόδοση επιτεύχθηκε και για τα 2 πειράματα για την 3η αρχιτεκτονική, με ποσοστό επιτυχίας στις προβλέψεις ίσο με 70%.

Για το δεύτερο σύνολο δεδομένων εξετάστηκαν ξανά όλες οι αρχιτεκτονικές, για το 1ο πείραμα, με τις δύο τεχνικές (μονογράμματα και διγράμματα). Η διάσταση του επιπέδου εισόδου είναι 27 ενώ η διάσταση του επιπέδου εξόδου 2 (για την ταξινόμηση των tweets σε 2 κλάσεις πολικότητας). Και εδώ, το 75% των tweets του συνόλου χρησιμοποιήθηκε για την εκπαίδευση του δικτύου, ενώ το 25% για τη δοκιμή του. Επίσης, χρησιμοποιήθηκε η 5-fold cross-validation. Κάθε αρχιτεκτονική παρατηρήθηκε ότι συνέκλινε για το 1ο πείραμα σε ένα μέσο ΜΤΣ ίσο με 0.166, ενώ για το 2ο πείραμα σε ένα μέσο ΜΤΣ ίσο με 0.176. Μετά τη σύγκλιση του δικτύου, ξεκίνησε η φάση της δοκιμής του. Το συγκεκριμένο σύνολο δεδομένων δεν έδωσε τόσο ικανοποιητικά αποτελέσματα όσο το προηγούμενο: Η καλύτερη απόδοση επιτεύχθηκε για την 1η τεχνική με την 1η αρχιτεκτονική, με ποσοστό επιτυχίας στις προβλέψεις ίσο με 63.84%. Για τη 2η τεχνική, η 5η αρχιτεκτονική έδωσε την καλύτερη απόδοση με ποσοστό επιτυχίας 58.35%.

Είναι φανερό από τα παραπάνω αποτελέσματα ότι το κοινό λεξιλόγιο λόγω του κοινού θέματος των tweets που είναι παρόν στο πρώτο σύνολο δεδομένων είναι το χαρακτηριστικό εκείνο που οδήγησε στην καλύτερη απόδοσή του. Το δεύτερο σύνολο δεδομένων είναι σχετικά μικρό σε μέγεθος, το οποίο σε συνδυασμό με την ποικιλία των θεμάτων και περιεχομένων των tweets που το συνιστούν, οδηγεί σε απουσία επανάληψης κοινών λέξεων ανά tweet και συνεπώς σε αδυναμία του νευρωνικού δικτύου να κάνει τους κατάλληλους συσχετισμούς. Αυτό επιδεινώνεται ακόμα περισσότερο με την τεχνική των διγραμμάτων. Η τεχνική αυτή συνήθως δίνει καλύτερες αποδόσεις από αυτή των μονογραμμάτων, ωστόσο στην περίπτωσή μας συμβαίνει το αντίθετο, εξαιτίας του μικρού μεγέθους και της απουσίας κοινού θέματος στο δεύτερο σύνολο, το οποίο όχι μόνο δε βοηθάει στη συσχέτιση των διγραμμάτων (εκ των οποίων κοινά διγράμματα σπάνια επαναλαμβάνονται) αλλά χάνει και πληροφορία που θα μπορούσε να έχει αποκτήσει το δίκτυο από το συσχετισμό των μεμονωμένων λέξεων.

Μια ιδέα για βελτίωση είναι, κατ' αρχάς, η χρήση μεγαλύτερων συνόλων δεδομένων ή συνόλων με tweets κοινού θέματος. Στη συνέχεια, μπορεί να δοκιμαστούν τροποποιήσεις στη δομή του δικτύου: ένα συνελικτικό δίκτυο είναι ιδιαίτερα κατάλληλο για την ταξινόμηση προτύπων. Ακόμα μπορεί να γίνει θεώρηση άλλων χαρακτηριστικών που ενδεχομένως να οδηγήσουν σε καλύτερη απόδοση, όπως η συχνότητα εμφάνισης των λέξεων ή η θέση τους που ενδεχομένως να ακολουθεί κάποια δομή μέσα στο tweet (π.χ. κύριο σώμα tweet – hashtag που εκφράζει συναίσθημα). Τέλος, η υλοποίηση της μεθόδου του user profiling δηλαδή η δημιουργία ενός προφίλ ανά χρήστη ενδεχομένως να βοηθήσει στη βελτίωση της απόδοσης καθώς έτσι θα λαμβάνονται επιπλέον υπόψιν στατιστικά του χρήστη και η πιθανότητα να εκφραστεί θετικά/αρνητικά/ουδέτερα ως προς ένα θέμα.

Abstract

The subject of this diploma thesis is the automatic detection of the polarity – also known as sentiment analysis – of the opinion which is expressed by users on web. Twitter consisted the web source of our interest. Twitter is an online social networking service that enables users to publish and read short messages known as *tweets*. Sentiment analysis aims to classify correctly text samples according the overall polarity of the opinion they express (positive, negative, neutral).

In order to achieve the sentiment analysis of tweets, we implemented a supervised online learning system, based on artificial neural networks. This system consists of two main parts: the first one pre-processes the tweets, while the second one implements the learning procedure based on the neural network. The first part aims to the refinement and cleaning of the tweets from noise and useless information, as well as to their transmutation into a form which is comprehensible by a machine. This is achieved through the extraction of specific features and the construction of the corresponding vectors. The second part, the part of the neural network, is implemented by a multi-layer perceptron, which is trained by the error back propagation algorithm, and aims to the correct classification of the tweets into two or three discrete classes, each one of them corresponds to the opinion polarity expressed in each tweet (positive, negative or positive, negative, neutral).

Keywords: *sentiment analysis, user opinion mining, machine learning,*

artificial neural networks, back propagation algorithm, Twitter, data mining.

Acknowledgements

This diploma thesis was conducted in the Image, Video and Multimedia Systems Laboratory under the supervision of Professor Stefanos Kollias, whom I would like to thank for the opportunity he offered me to get involved with a very inspiring topic of computer science, that combines areas of my great interest - machine learning and linguistics - and that motivated me to explore them in depth.

Furthermore, I would specially like to express my appreciation and gratitude to the Research Director of ICCS of NTUA, Kostas Karpouzis, for his guidance and precious support during the whole process of this thesis, his inspiring ideas and valuable feedback.

I would also like to thank deeply my family, my parents Sofia and Manolis, my brothers Thodoris and Giorgos, and my grandmothers Kaiti and Eleni, who have been on my side on the bad and the good times, and have always supported my choices.

Finally, a big thank you goes out to all the friends who have accompanied me so far, and especially Marina, Efigianna, Ioanna, Anna, Antonis, Giannis, Giorgos and Kostas.

Instead of a Preface

The sixth member of the crew cared for none of these things, for it was not human. It was the highly advanced HAL 9000 computer, the brain and nervous system of the ship.

Hal (for Heuristically programmed ALgorithmic computer, no less) was a masterwork of the third computer breakthrough. These seemed to occur at intervals of twenty years, and the thought that another one was now imminent already worried a great many people.

The first had been in the 1940s, when the long-obsolete vacuum tube had made possible such clumsy, high-speed morons as ENIAC and its successors. Then, in the 1960s, solid-state microelectronics had been perfected. With its advent, it was clear that artificial intelligences at least as powerful as Man's need be no larger than office desks - if one only knew how to construct them.

Probably no one would ever know this; it did not matter. In the 1980s, Minsky and Good had shown how neural networks could be generated automatically – self replicated – in accordance with any arbitrary learning program. Artificial brains could be grown by a process strikingly analogous to the development of a human brain. In any given case, the precise details would never be known, and even if they were, they would be millions of times too complex for human understanding. Whatever way it worked, the final result was a machine intelligence that could reproduce - some philosophers still preferred to use the word "mimic" - most of the activities of the human brain - and with far greater speed and reliability. It was extremely expensive, and only a few units of the HAL9000 series had yet been built; but the old jest that it would always be easier to make organic brains by unskilled labor was beginning to sound a little hollow.

Hal had been trained for this mission as thoroughly as his human colleagues - and at many times their rate of input, for in addition to his intrinsic speed, he never slept. His prime task was to monitor the life-support systems, continually checking oxygen pressure, temperature, hull leakage, radiation, and all the other interlocking factors upon which the lives of the fragile human cargo depended. He could carry out the intricate navigational corrections, and execute the necessary flight maneuvers when it was time to change course. And he could watch over the hibernators, making any necessary adjustments to their environment and doling out the minute quantities of intravenous fluids that kept them alive. The first generations of computers had received their inputs through glorified typewriter keyboards, and had replied through high-speed printers and visual displays. Hal could do this when necessary, but most of his communication with his shipmates was by means of the spoken word. Poole and Bowman could talk to Hal as if he were a human being and he would reply in the perfect idiomatic English he had learned during the fleeting weeks of his electronic childhood.

Whether Hal could actually think was a question which had been settled by the British mathematician Alan Turing back in the 1940s. Turing had pointed out that, if one could carry out a prolonged conversation with a machine - whether by typewriter or microphone was immaterial - without *being able to distinguish between its replies and those that a man might give, then the machine was thinking, by any sensible definition of the word. Hal could pass the Turing test with ease.*

The time might even come when Hal would take command of the ship. In an emergency, if no one answered his signals, he would attempt to wake the sleeping members of the crew, by electrical and chemical stimulation. If they did not respond, he would radio Earth for further orders.

And then, if there was no reply from Earth, he would take what measures he deemed necessary to safeguard the ship and to continue the mission - whose real purpose he alone knew, and which his human colleagues could never have guessed.

Poole and Bowman had often humorously referred to themselves as caretakers or janitors aboard a ship that could really run itself. They would have been astonished, and more than a little indignant, to discover how much truth that jest contained.

Excerpt from the book “2001: A Space Odyssey”

Arthur C. Clarke, 1968 [33]

Contents

List of Figures

List of Tables

1 *Sentiment Analysis and Twitter*

Sentiment analysis is a subfield of text mining which aims to the identification of the user's sentiment with respect to a specific subject. The subjective information is extracted from texts by using a combination of machine learning and natural language processing techniques [1].

In this chapter, we are going to speak about machine learning and describe its approaches. Afterwards, we will speak about sentiment analysis, the historical background as well as the learning approaches that have been used for this task. Finally, we will speak about microblogs and, especially, Twitter. We are going to describe the special characteristics that Twitter gathers and make it an ideal source for the task of user opinion mining, which motivated us for this work.

1.1 Machine Learning

1.1.1 Definition

Machine learning is an approach to the development of algorithms of artificial intelligence that produce predictions by exploiting known properties learned from a certain dataset [2]. It is a subfield of computer science that explores the construction and study of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions [3].

Machine learning evolved from the study of pattern recognition and computational learning theory in Artificial Intelligence.

1.1.2 Theory

A core objective of a learner is to generalize from its experience. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples or tasks after having experienced a learning dataset. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a general model about this space that enables it to produce sufficiently accurate predictions in new cases.

Computational learning theory is the computational analysis of machine learning algorithms and their performance. Training sets are finite and the future is uncertain, hence learning theory usually does not guarantee the performance of algorithms. Instead, probabilistic bounds on the performance are quite common. The bias–variance decomposition is one way to quantify generalization error.

In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time [4].

1.1.3 Approaches

Below we describe the approaches of machine learning [3].

- **Decision tree learning**

A decision tree is used as a predictive model, mapping observations corresponding to an item to conclusions about the item's target value.

- **Association rule learning**

Association rule learning is a method for discovering interesting relations between variables in large databases.

- **Artificial neural networks**

An artificial neural network (ANN) learning algorithm, usually called “neural network” (NN), is a learning algorithm that is inspired by the structure and functional aspects of biological neural networks (nervous systems). Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables. ANN is the technique that we are going to use in our application and they are further described in chapter 2.

- **Inductive logic programming**

Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs.

- **Support vector machines**

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

- **Clustering**

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some predesignated criterion or criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated for example by internal compactness (similarity between members of the same cluster) and separation between different clusters. Other methods are based on estimated density and graph connectivity. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

- **Bayesian networks**

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independences via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

- **Reinforcement learning**

Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

- **Representation learning**

Several learning algorithms, mostly unsupervised learning algorithms, aim to discovering better representations of the inputs provided during training. Classical examples include principal components analysis and cluster analysis. Representation learning algorithms often attempt to preserve the information in their input but transform it in a way that makes it useful, often as a pre-processing step before performing classification or predictions, allowing to reconstruct the inputs coming from the unknown data generating distribution, while not being necessarily faithful for configurations that are implausible under that distribution.

Manifold learning algorithms attempt to do so under the constraint that the learned representation is low-dimensional. Sparse coding algorithms attempt to do so under the constraint that the learned representation is sparse (has many zeros). Multi-linear subspace learning algorithms aim to learn low-dimensional representations directly from tensor representations for multidimensional data, without reshaping them into (high-dimensional) vectors. Deep learning algorithms discover multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an intelligent machine is one that learns a representation that disentangles the underlying factors of variation that explain the observed data.

- **Similarity and metric learning**

In this problem, the learning machine is given pairs of examples that are considered similar and pairs of less similar objects. It then needs to learn a similarity function (or a distance metric function) that can predict if new objects are similar. It is sometimes used in Recommendation systems.

- **Sparse dictionary learning**

In this method, a datum is represented as a linear combination of basis functions, and the coefficients are assumed to be sparse. Let x be a d-dimensional datum, D be a d by n matrix, where each column of D

represents a basis function. *r* is the coefficient to represent *x* using *D*. Mathematically, sparse dictionary learning means the following $x \approx Dr$ where *r* is sparse. Generally speaking, *n* is assumed to be larger than *d* to allow the freedom for a sparse representation.

Learning a dictionary along with sparse representations is strongly NP-hard and also difficult to solve approximately. A popular heuristic method for sparse dictionary learning is K-SVD.

Sparse dictionary learning has been applied in several contexts. In classification, the problem is to determine which classes a previously unseen datum belongs to. Suppose a dictionary for each class has already been built. Then a new datum is associated with the class such that it's best sparsely represented by the corresponding dictionary. Sparse dictionary learning has also been applied in image de-noising. The key idea is that a clean image patch can be sparsely represented by an image dictionary, but the noise cannot.

- **Genetic algorithms**

A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection, and uses methods such as mutation and crossover to generate new genotype in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms found some uses in the 1980s and 1990s. Vice versa, machine learning techniques have been used to improve the performance of genetic and evolutionary algorithms.

1.2 Sentiment Analysis

1.2.1 Definition

Sentiment analysis (also known as user sentiment and opinion analysis, opinion mining) is a subfield of text mining which aims to the identification of the user's sentiment with respect to a specific subject. The subjective information is extracted from texts by using a combination of machine learning and natural language processing techniques [1]. Subjective information expressed in texts vary from opinion attitudes to feelings expressed. The tasks of sentiment analysis include, though they are not limited to, the following [5]:

- Sentiment classification, which classifies a given piece of text as positive, negative or neutral.
- Opinion retrieval, which retrieves opinions in relevance to a specific topic or query.
- Opinion summarization, which summarizes opinions over multiple text sources towards a certain

topic.

- Opinion holder identification, which identifies who express a specific opinion.
- Topic/sentiment dynamics tracking, which aims to track sentiment and topic changes over time.
- Opinion spam detection, which identifies fake/untruthful opinions.
- Prediction, which predicts people's behaviors, market trends, political election outcomes etc., based on opinions or sentiments expressed in online contents.

1.2.2 Historical Background and Learning Approaches

In the past, text information processing focused mainly on mining and retrieving factual information, such as classifying documents according to their subject matter. Later, in recent years, there has been a rapid growth of research interests in natural language processing that seeks to better understand sentiment and opinion expressed in text. There are several reasons that explain this growth. One of them is that with the rise of various types of social media, communicating on the web has become increasingly popular, and millions of people are able to broadcast their thoughts and opinions on a wide variety of topics, such as feedbacks on products and services, opinions on political development and events etc. For that reason, new computational tools are needed for the organization, summarization and comprehension of this huge amount of information. In addition, a big number of ideas for useful applications is triggered by the discovery of opinions reflecting people's attitudes towards various topics, and this is an additional motivation for sentiment analysis. We are going to describe here some learning approaches and their historical background [5].

1.2.2.1 Supervised Learning

Sentiment analysis can be considered as computational treatments of subjective information such as opinions and emotions expressed in text. In a primitive manner, sentiment analysis aims to the automatic identification of the positive or negative opinion expressed in a given piece of text. The first approaches of Pang et al. (2002) and Matsumoto et al. (2005) view sentiment classification as a text classification problem, where a corpus with sentiment orientation annotated is required for the classifiers training. This is the basic idea of **supervised** sentiment classification approaches. These approaches usually perform well when there is a large enough training set. The state-of-art approach (Matsumoto et al. 2005) can achieve an accuracy greater than 90% on the movie review data. [5] Nevertheless, this approach has several not negligible issues: the supervised classifier trained on a specific domain, it is very possible to fail to produce satisfactory performance when tested on other domains, while online content varies widely in domains and evolves rapidly over time, making corpora annotation for each domain unrealistic (domain transfer and labeling cost problems).

1.2.2.2 Semi-Supervised Learning

In response to the above problems, there has been a rising interest in exploring hybrid or **semi-supervised** approaches, leveraging a large amount of unlabeled data and a small amount of labeled data for classifier training. Aue and Gamon (2005) explored various strategies for training SVM classifiers for the target domain lacking sufficient labeled data. Blitzer et al. (2007) proposed structural correspondence learning (SCL), addressing to the domain transfer problem with [5].

1.2.2.3 Unsupervised or Weakly Supervised Learning

Unsupervised or weakly supervised approaches are mostly lexicon based, which do not require labeled document for training. Their main idea is that the sentiment orientation of a document is an averaged sum of the sentiment orientation of its words and phrases. Supervised or weakly supervised methods for sentiment classification are very challenging, having taken into account the difficulties of supervised and semi-supervised sentiment analysis. However, solutions to unsupervised or weakly supervised sentiment classification are of practical significance owing to its domain-independent nature.

The pioneer work is the point-wise mutual information (PMI) approach proposed by Turney (2002). Turney calculated the sentiment orientations of phrases in documents as its PMI with a positive prototype “excellent” minus the PMI with a negative prototype “poor”. This approach achieved 84% accuracy for automobile reviews and 66% accuracy for movie reviews. The work of Read and Carroll (2009) is also a good example of a lexical-based approach.

Weakly supervised sentiment classification approaches are similar to unsupervised approaches as they do not require labeled documents for training. What they do is that they typically incorporate supervision information either from sentiment lexicons containing a list of words marked as positive or negative (usually much larger in size than the sentiment seed words used in unsupervised approaches) or from users feedback. Lin and He (2009) proposed a joint sentiment-topic (JST) model to detect document-level sentiment and extract sentiment bearing topics simultaneously from text. The JST model incorporated a small set of domain-independent sentiment words as prior knowledge for model learning, and it achieved comparable performance to semi-supervised approaches with a percentage of 40% of labeled data [5].

1.3 Twitter Sentiment Analysis

1.3.1 Microblogging

Microblogging is a web service that allows the subscriber to broadcast short messages to other subscribers of the service. A subscriber can publish microblog posts on a website and/or distribute them to a group of subscribers. Depending on the privacy settings of the microblog, unsubscribed users might be able to read microblog posts but not post new ones or share/comment the posts of subscribers. Subscribers can read microblog posts online. They can also request updates to be delivered in real time to their desktop as an instant message or sent to a mobile device as an SMS text message [6].

Micropost is a term that is sometimes used to describe a microblog post. Microtext is another term, that describes a type of written text document that has the following three characteristics [7]:

- it is very short (typically one or two sentences)
- it is written in an informal manner and unedited for quality, and thus it may use loose grammar, a conversational tone, vocabulary errors, and uncommon abbreviations and acronyms
- it is semi-structured in the NLP sense, in that it includes some metadata such as a time stamp, an author or the name of a field it was entered into.

Microtexts have become omnipresent in today's world: they are notably found in online chat discussions, online forum posts, user comments posted on online material such as videos, pictures and news stories, Facebook newsfeeds and Twitter updates, Internet search queries, and SMS.

The content of a microtext/micropost may vary from short sentences to individual images, video or other links. This may be the main reason for the popularity of microblogs. The major difference between a microblog and a traditional blog is that the content of a microblog is typically smaller in both actual and aggregated file size. The first microblogs were known as tumblelogs. Jason Kottke described tumblelogs on October 19, 2005: “A tumblelog is a quick and dirty stream of consciousness, a bit like a remaindered links style linklog but with more than just links. They remind me of an older style of blogging, back when people did sites by hand, before Movable Type made post titles all but mandatory, blog entries turned into short magazine articles, and posts belonged to a conversation distributed throughout the entire blogosphere. Robot Wisdom and Bifurcated Rivets are two older style weblogs that feel very much like these tumblelogs with minimal commentary, little cross-blog chatter, the barest whiff of a finished published work, almost pure editing...really just a way to quickly publish the "stuff" that you run across every day on the web.” [8].

Microblogging is both immediate and portable, and this is why it's massively attractive. Posts are brief (typically 140 - 200 characters) and can be written or received with a variety of computing devices, including cell phones. Although most microblog broadcasts are posted as text, some microblogging services allow video or audio posts. Microblogging is slowly moving into the mainstream. For example, in USA, presidential candidate Barack Obama microblogged from the campaign trail using Twitter. Traditional media organizations, such as newspapers, television channels have started to share headlines and links in microposts [9]. Furthermore, other potential applications of microblogging include traffic and sports updates and emergency broadcast systems.

Microbloggers post about topics ranging from very simple everyday issues to the topical issues, as they have the possibility to use hashtags and share their opinion on a certain topic. Commercial microblogs also exist to promote websites, services and products, and to promote collaboration within an organization.

Microblogs often offer features such as privacy settings, which allow users to control who can read their microblogs, or alternative ways of publishing entries besides the web-based interface. These may include text messaging, instant messaging, e-mail, digital audio or digital video [8].

Several popular microblogging services are Google+, Tumblr, Facebook and Twitter. Besides, Twitter and Facebook are the two most popular social networks today. We are going to focus on Twitter for this thesis and we explain the reasons of this choice in the next chapter.

1.3.2 Twitter Sentiment Analysis

Twitter [10] is an online social networking service that enables users to send and read short 140-character messages. These messages are called “tweets”. A tweet is an expression of a moment or idea. It can contain text, photos and videos. Millions of tweets are shared in real time, every day. Registered users of Twitter can read and post tweets but unregistered users can only read them. Users can access Twitter through the website interface twitter.com, SMS or mobile device app.

Twitter Inc. is based in San Francisco and has more than 25 offices around the world. The service was created in March 2006 by Jack Dorsey, Evan Williams, Biz Stone and Noah Glass and launched by July 2006. Twitter rapidly gained worldwide popularity with more than 100,000,000 users who in 2012 posted 340,000,000 tweets per day. It also handled 1.6 billion search queries per day. Twitter was one of the ten most-visited websites in 2013, and due to its short size, it has been described as “the SMS of the Internet”. As of May 2015, Twitter has more than 500,000,000 users, out of which 302,000,000 are active users [9].

It is obvious that Twitter can be seen as a large source of short texts (tweets), often consisted of user opinions, most of which are appropriate for sentiment analysis tasks by identifying user attitudes and opinions toward a particular topic or product. Besides, the fact that it is quite common to use hashtags in order to indicate the topic about which a user expresses an opinion, as well as the fact that Twitter limits tweet length to 140 characters makes tweets more possible to contain pure opinions on a topic without much chatter (noise). From all the above, it is quite clear that Twitter can be considered as an excellent source of short texts containing user opinions for the task of user opinion mining [7].

Twitter microblog sentiment analysis aims to identify and detect the sentiments or emotions that are present in a microblog post (tweet). The techniques developed for microblog sentiment analysis can also be applied to classify social media data in a real-time manner [7].

Twitter has become a quite attractive source of data for opinion analysis due to the large amount of information contained in tweets. However, Twitter also consists a much harder challenge than sentiment analysis on conventional text. The main reasons for this are some particular characteristics of Twitter: the short length of tweets (tweets are too short to be linguistically analyzed), the frequent use of informal and irregular words, slang, misspellings and colloquialisms as well as the rapid evolution of language [7]. In addition, we have to deal with human subjectivity, while even humans often disagree on the categorization of the positive or negative sentiment that is supposed to be expressed on a given text. All these special characteristics make the task of Twitter sentiment analysis a hard challenge.

Annotated tweets data are impractical to obtain. There are several ideas for the task of annotation of the tweets. The works of Go et al. (2009), Pak and Paroubek (2010), Barbosa and Feng (2010) proposed noisy labels or distant supervision, for example by taking emoticons or hashtags as the indication of tweet sentiment to train supervised classifiers [1]. Other works (Agarwal et al. 2011, Kouloumpis et al. 2011) explore feature engineering in combination of machine learning methods to improve sentiment classification accuracy on tweets.

There have been many approaches for Twitter sentiment analysis. The most prominent of them are Naive Bayes algorithm, Maximum Entropy, Support Vector Machines and Artificial Neural Networks. For the present work, we explore the approach that is based on Artificial Neural Networks.

2 *Artificial Neural Networks*

Artificial neural networks consist the machine learning technique that was used for the implementation of the learning procedure of our model. A neural network is a huge parallel processor with distributed architecture, comprised of simple processing units, called neurons, and natively having the ability to store empirical knowledge and make it available for use. It resembles the human brain in two points:

- The network receives the knowledge from its environment through a learning procedure.
- The strength of the connections among neurons, which is called synaptic weight, is used for the storing of the acquired knowledge.

The learning procedure through which a neural network achieves learning is called learning algorithm and its operation is modify the synaptic weights of the network in an appropriate way for the achievement of the desirable goal.

In this chapter, firstly we are going to describe the human nervous system, from which the artificial neural networks were inspired. Afterwards, we will speak about the neurons as processing units and the neural networks architectures. Furthermore, we will see the ways knowledge can be represented and the learning procedures. We are also going to describe the Rosenblatt's perceptron and the multi-layer perceptron, as well as the back propagation algorithm that aims to train a multi-layer perceptron. Finally, we are going to see the method of cross-validation.

2.1 Human Nervous System

2.1.1 Basic Structure

The human nervous system is comprised of two major subdivisions: the central nervous system (CNS) and the peripheral nervous system (PNS). The CNS includes the brain and spinal cord. It is consisted of the nerve cells, known as neurons, and the supporting cells, known as glial cells. The brain is the body's control center [11].

Human nervous system can be treated as a three stages system (figure 1), as Arbid proposed in 1987. Brain can be represented by a neural network that always receives information, processes it and makes the appropriate decisions based on it. In fig. 1, the arrows are of two directions; the ones that are directed from left to right indicate the forward transmission of information signals, the ones that are directed from right to left mark the presence of feedback in the system. The receptors convert the stimuli or stimulations coming in from the human body or the external environment through the human sensors to electrical signals (known as impulses) that transport information to the neural network (the brain). The effector cells convert these electrical signals produced by the neural network into aesthetic responses, which are the system outputs.

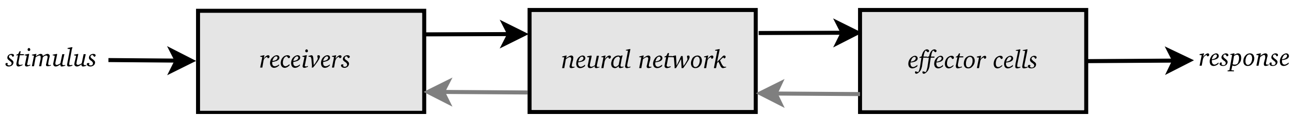


Figure 1: Schematic representation of human nervous system.

Ramon y Cajal (1911) [12] firstly inducted the idea of the neuron as the structural component of the brain. Typically, neurons are 5 or 6 orders of magnitude slower than silicon based logic gates; in an integrated circuit, events take place in the scale of nanoseconds, while in human neural network they take place in the scale of milliseconds. However, the human brain compensates this relatively slow operating speed of a neuron with the presence of an immense crowd of neurons and a respectively huge crowd of interconnections among them. It is estimated that there are 10 billions of neurons in the human cerebral cortex and 60 trillions connections (Shepherd and Koch, 1990). All these make the brain an amazingly efficient structure. The energy efficiency of the human brain is around 10^{-16} Joule, while the corresponding value for the best computers is greater by many orders of magnitude. There are three main types of neurons:

- The sensory neurons: These neurons are connected to receptors that are specialized to detect and respond to various stimulations from the internal or external environment. The receptors that are sensitive to changes of light, sound, mechanical or chemical stimulations serve the senses of sight, hearing, touch, smell and taste. When thermal or chemical stimulations on the skin exceed a certain intensity level, it is possible to cause tissue disaster, which will activate a specific category of receptors, the nociceptors (receptors of stimulations of pain). These receptors activate the protective reflexes as well as the pain sensation.

- The motor neurons (or motoneurons): These neurons control the muscles' activity and are involved in all kinds of behavior, including speech.
- The interneurons: These neurons are interposed between the sensory and the motor neurons. These nerve cells consist the majority of cells in human brain. The interneurons intermediate simply reflexively, but they also participate in the brain's superior functions.

The glial cells were believed to contribute only in a supportive way; however today it is known that they contribute in a very significant way to the development of the nervous system and to the operation of an adult brain. Even though the glial cells are more in number than the neurons, they do not transmit the information in the same way with them.

The architectural structure of neurons (fig. 2) is consisted of the cell body, which is known as the soma, and two additional parts, called neurite. The neurite is consisted of the axons and the dendrites. Axons' task is the information transmission from a neuron to the ones it is connected to. Dendrites' task is to receive the transmitted information from the axons of other neurons. Both the axons and the dendrites participate in the formation of special points of contact, which are called synapses. The neurons are organized in complicated chains and networks, that consist the paths through which the information is transmitted in the nervous system [11].

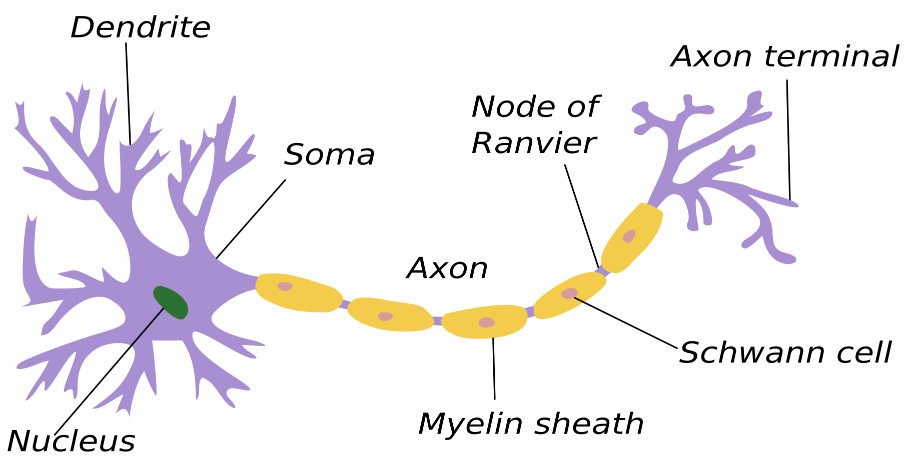


Figure 2: Structure of a typical neuron [34].

The connections or nerve endings (synapses) are the elementary structural and functional units that intermediate during the interactions among the neurons. The most common type of a synapse is the chemical synapse. Its operation is the following: a presynaptic procedure releases a chemical substance, known as neurotransmitter, which is diffused in the synaptic junction between neurons and then acts on a postsynaptic process. Thus, a synapse converts a presynaptic electrical signal to a chemical signal, and then it converts again the chemical signal into a postsynaptic electrical signal (Shepherd and Koch, 1990).

In terms of electronics, such a component consists a double-door network. In traditional descriptions of the nervous system organization, a synapse is considered a simple connection that can cause stimulation or suspension, but not both of them to the neuron-receiver.

The brain and the spinal cord are connected with aesthetic receptors and muscles through a large length of axons that consist the peripheral nerves. The spinal cord has two main operations: it consists the setting of simple and more complicated reflexes, and it forms a path of quick information transmission from the body to the brain and vice versa.

The above basic structures of the nervous system are the same in all the vertebrate. The attribute that makes the human brain different is its big size compared with its body size. This is caused by the huge growth of the number of interneurons during the evolution of the species, and this fact provides the human with countless choices of reactions to stimulations received from the environment.

2.1.2 Anatomy and Physiology

The human brain is consisted of the brainstem and the cerebral hemispheres. The brainstem is divided to the hindbrain, the midbrain and to an interbrain, called diencephalon. It contains neural networks

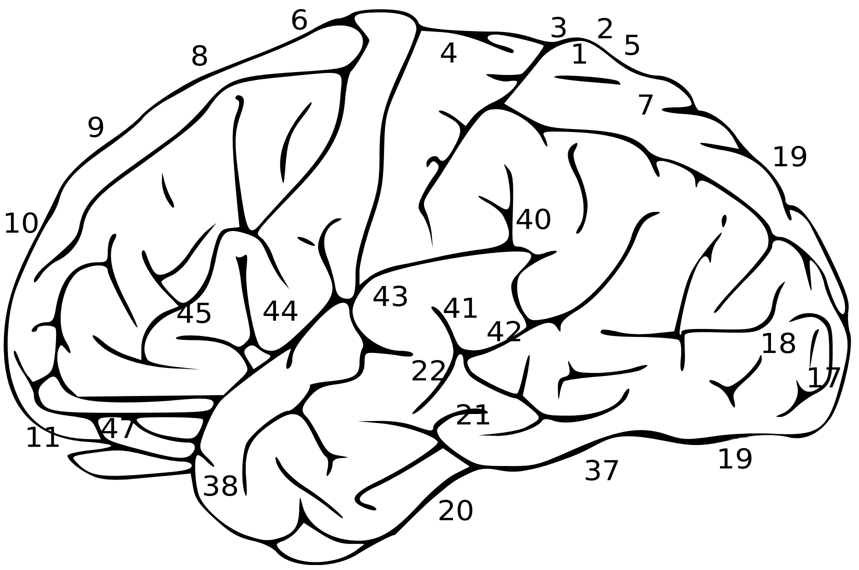


Figure 3: Architecture of the cerebral cortex. Some of the main sensory areas are the following: Motor cortex: areas 4, 6 and 8; Somatosensory cortex: areas 1, 2 and 3; Visual cortex: areas 17, 18 and 19; Auditory cortex: areas 41 and 42 [35].

that consist control centers for vital operations, such as breathing and the arterial pressure. From the top of the hindbrain, the cerebellum arises, playing a quite significant role in the movements control and coordination.

The midbrain contains the groups of neurons. Each one of them uses a specific type of a chemical neurotransmitter, but all of them project to the cerebral hemispheres. It is believed that these neurons are able to regulate the neurons' activity to the superior brain centers that control operations such as sleep, attention and reward.

The diencephalon is divided into two very different areas, which are the thalamus and the hypothalamus. Thalamus transfers impulses from all the sensory systems to the cerebral cortex, which, subsequently, sends messages back to thalamus. This forward-backward way of this assembly is quite interesting; the information is not transferred only to one direction. Hypothalamus controls operations such as eating and drinking. In addition, it controls the release of hormones regarding sexual functions.

The cerebral hemispheres are consisted of a nucleus, the basic ganglia and an extended but thin casing of neurons, which consists the gray matter of cerebral cortex. The basic ganglia play a central role to the inception and control of movements. The cerebral cortex, pressured in the limited space of cranium, is shaped by aspects that are wrapped inwards and outwards, enlarging in this way the surface of the casing. The cortex is the most developed structure of human brain; it is 4 times bigger than the one of gorillas. It is divided into a big number of discrete areas. Each of these areas is distinguished depending on layers and connections. The operations of many of these areas are known, for example the visual, auditory and olfactory areas, the somatosensory areas receiving information from the skin and several motor areas. The paths of the sensory receptors to the cortex and from the cortex to the muscles are intersected on the one side and the other. Thus, the movements of the right side of body are controlled by the left side of the cortex, and vice versa. Likewise, the left part of the body sends sensory signals to the right hemisphere. However, the two parts of the brain don't work isolated; the left and right cerebral cortex are connected through a long bundle of neural fibers, called corpus callosum. The cerebral [13] cortex is indispensable for the voluntary movements of tongue and speech, as well as superior operations like thinking and memory. Many of these operations take place in both sides of the brain, but some of them are located mainly in one of the two hemispheres [11].

2.2 Neurons

2.2.1 Neuron Models

A neuron is a generic computational unit that takes a number of inputs and produces a single output [13]. It processes information which is fundamental for the operation of a neural network, such as the human nervous system or an artificial neural network (ANN). The model of a neuron on which is based the design of big neural networks family is presented in the schematic diagram of figure 4 [14].

The basic components of this model are described here [14]:

- A set of synapses or interconnections. Each one of them is characterized by its own weight or strength. Specifically, a signal x_j in the input of a synapse j that is connected to the neuron k, is multiplied by the synaptic weight w_{kj}. It is important here to explain the manner of indicating the symbol of the synaptic weight w_{kj}: The first indicator corresponds to the certain neuron, k, and the second indicator corresponds to the input edge of the synapse to which the weight refers. Unlike the synaptic weights of human brain, a synaptic weight of an artificial neuron can take

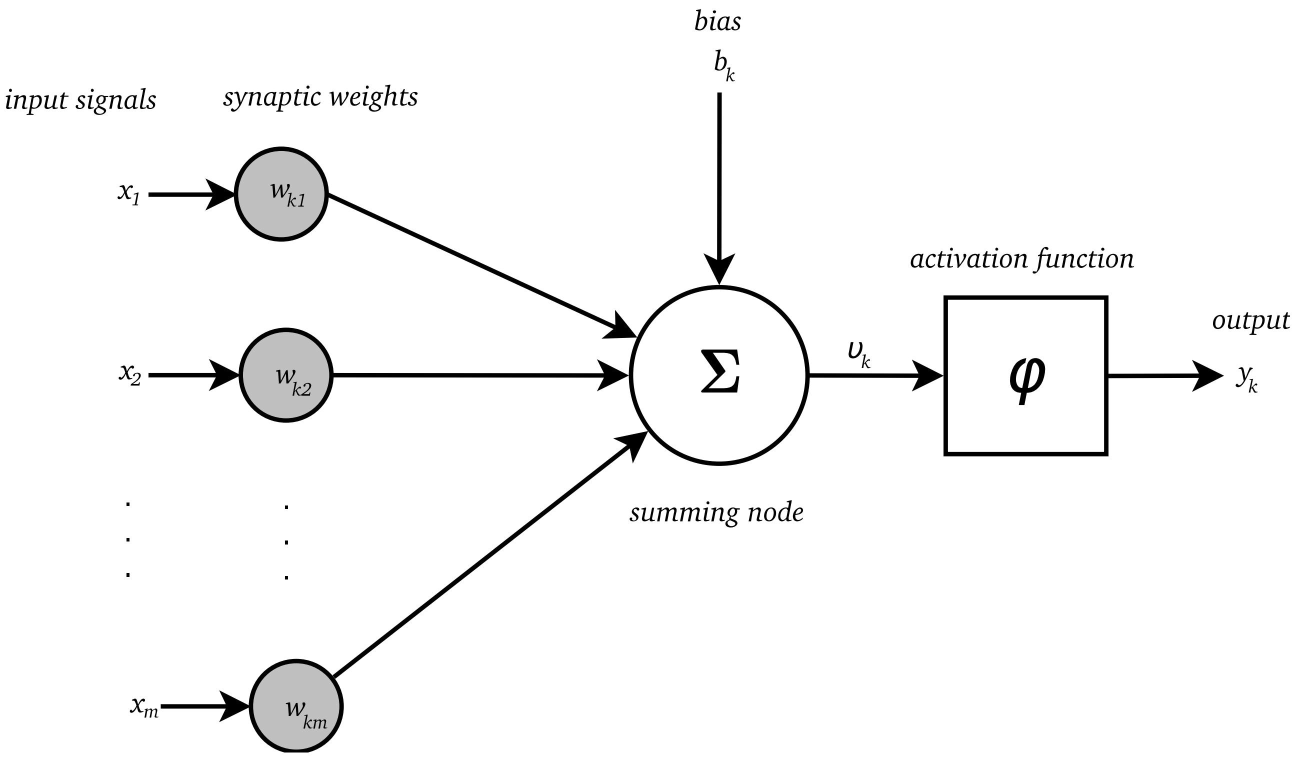


Figure 4: Non-linear neuron model

both negative or positive values.

- An adder. This component adds the input signals, weighted by the respective synaptic weights of the neuron; these operations consist a linear combiner.
- An activation function. The activation function is necessary for the amplitude limitation of the neuron's output signal within a finite range. This is why it is also mentioned as squashing function. Typically, the normalized range of a neuron is written as a unit closed interval, in the form of [0, 1]

or [-1, 1].

Furthermore, the neuron's model of fig. 4 includes a bias that is externally applied and is symbolized by the symbol b_k. This bias b_k is used to increase or decrease the network stimulation of the activation function, depending on whether the latter is, respectively, positive or negative.

Neuron k of fig. 4 can be described in mathematical terms with the following equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1}$$

and

$$y_k = \varphi(u_k + b_k) \tag{2}$$

where $x_1, x_2, \ldots, x_m$ are the input signals, $w_{k1}, w_{k2}, \ldots, w_{km}$ are the corresponding synaptic weights of neuron k, u_k is the output of the linear combiner caused by the input signals, b_k is the bias, $\varphi(\cdot)$ is the activation function, and y_k is the neuron's output signal. The use of bias b_k causes the application of an affine transformation on the output u_k of the linear combiner in the model of fig. 4, as it is obvious in the equation:

$$\upsilon_k = u_k + b_k \tag{3}$$

Specifically, depending on the bias, if it is negative or positive, the relation between the local field or activation potential υ_k of the neuron k and the u_k is modified in the way that is presented in fig. 5. These two terms are going to be used as synonymous. Because of this affine transformation, the curve υ_k-u_k doesn't pass through the origin (0, 0) [14].

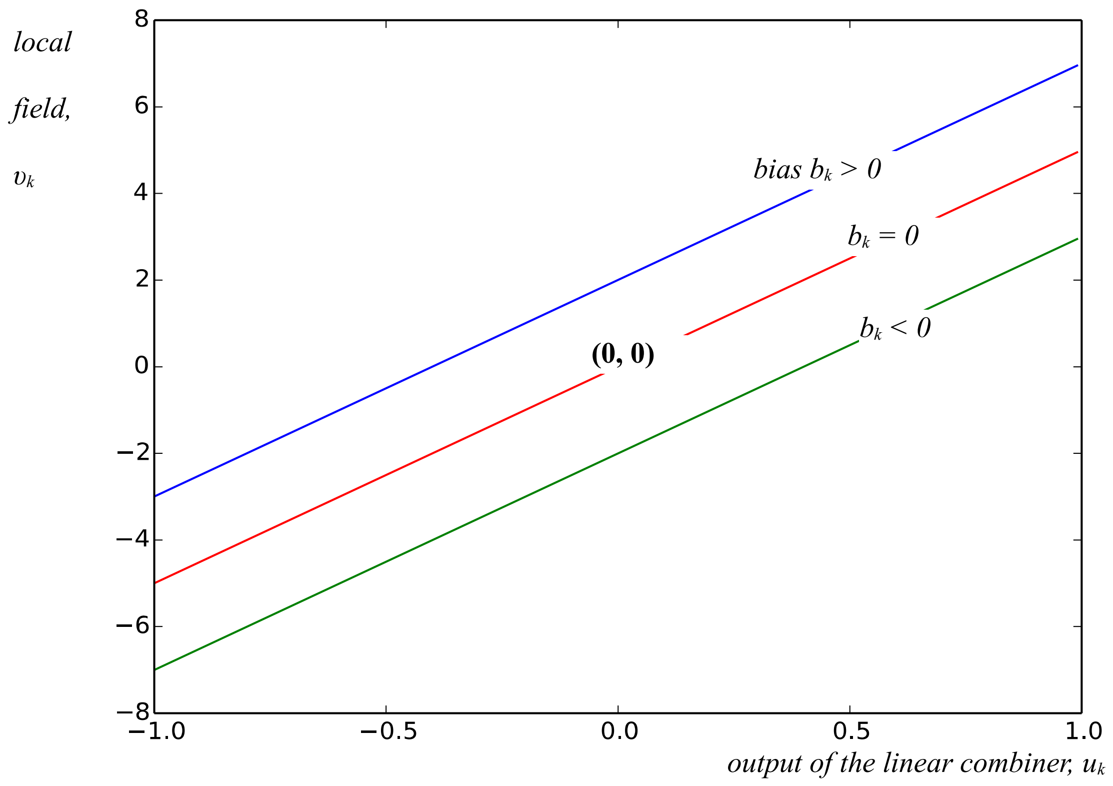


Figure 5: Affine transformation due to the presence of bias. It is $\upsilon_k = b_k$ at $u_k = 0$.

Bias b_k is an external parameter of neuron k and can be counted in through the equation (2). Equivalently, we can combine the equations (1) and (2) into the following:

$$\upsilon_k = \sum_{j=0}^{m} w_{kj} x_j \tag{4}$$

And

$$y_k = \varphi(\upsilon_k) \tag{5}$$

We added a new synapse in equation (4) and now its input and its weight are, respectively:

$$x_0 = +1 \tag{6}$$

and

$$w_{k0} = b_k \tag{7}$$

The model of a neuron *k* can now be rephrased, as seen in fig. 6. Here, we count in the effect of the bias in two steps: firstly we add a new constant input signal equal to +1, and afterwards we add a new synaptic weight equal to the bias b_k. Even though they look different, models of figures 4 and 6 are mathematically equivalent.

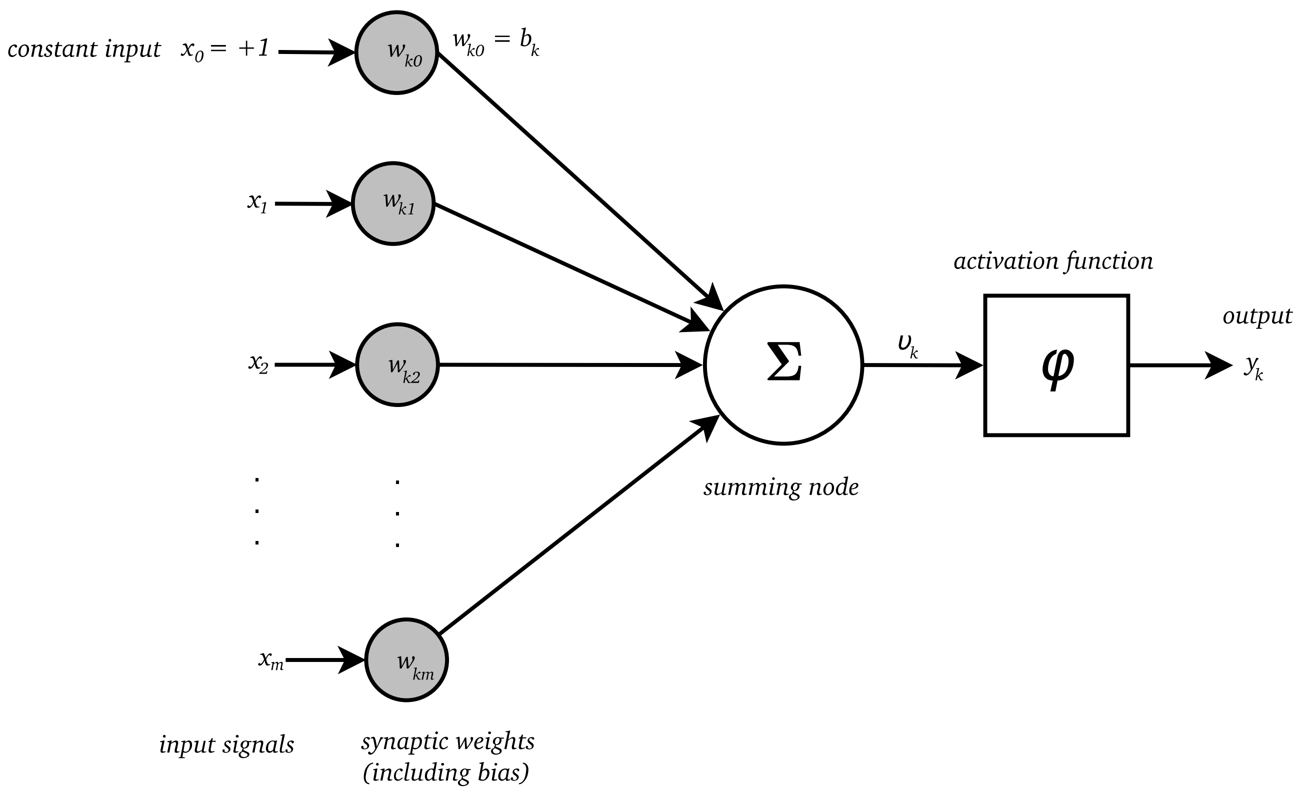


Figure 6: Non linear neuron model: the weight w_{k0} corresponds to the bias b_k.

2.2.2 Activation Function

The neuron's output is defined by the activation function, *φ(υ)*, where *υ* is the symbol of the local field [15]. The activation function can belong to one of two basic categories: the threshold function or the sigmoid function.

The threshold function (fig. 7), which is known as Heaviside function in Mechanics, is described by the following equation:

$$\varphi(\upsilon)=\begin{cases}1, & for\,\upsilon\geq 0\\ 0, & otherwise\end{cases} \tag{8}$$

The output of neuron *k* is:

$$y(k)=\begin{cases}1, & for\,\upsilon_k\geq 0\\ 0, & otherwise\end{cases} \tag{9}$$

where υ_k is the local field of the neuron, which is expressed by

$$\upsilon_k=\sum_{j=1}^{m} w_{kj}x_j+b_k \tag{10}$$

This is the McCulloh-Pitts model, within which the neuron's output is equal to 1 if the local field of the particular neuron isn't negative, and 0 otherwise.

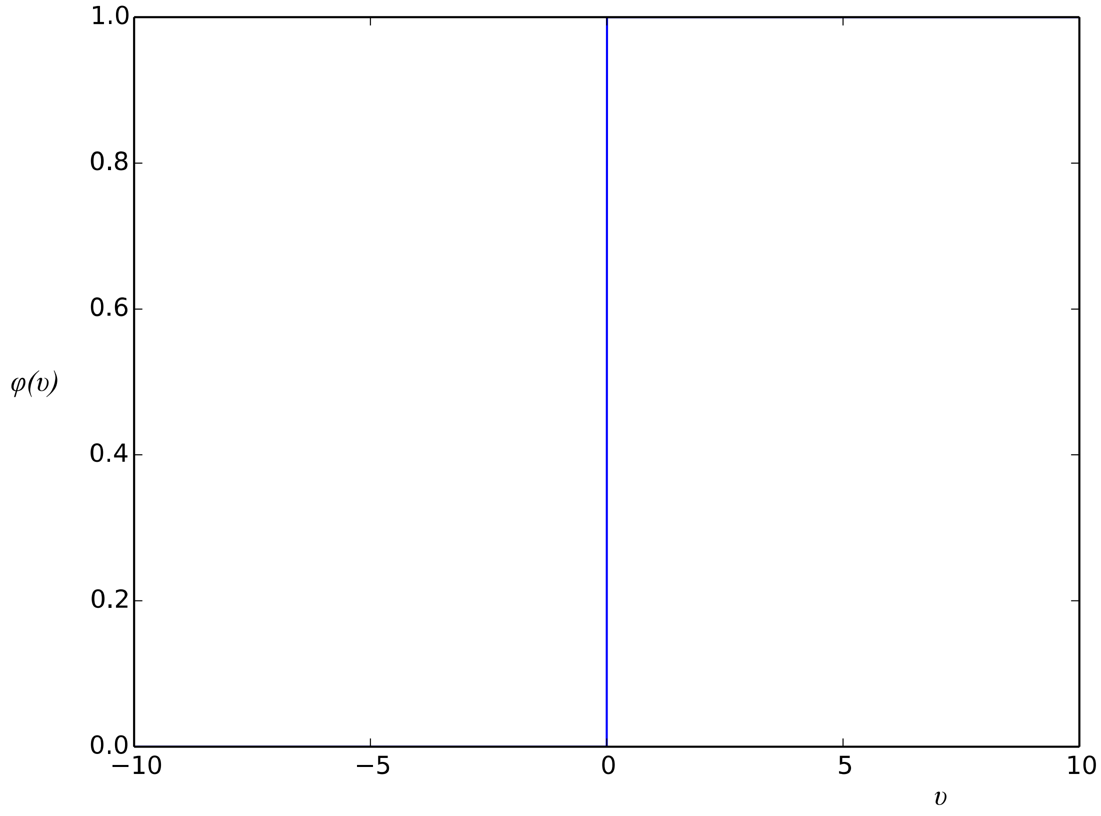


Figure 7: Plot of the threshold function.

The sigmoid function (fig. 8) is the most common activation function for the implementation of neural networks. It is defined as a strictly increasing function, displaying elegant balance between linear and non-linear behavior. Logistic function is an example of sigmoid function and is defined as:

$$\varphi(\upsilon)=\frac{1}{1+\exp(-\alpha\upsilon)} \tag{11}$$

where α is the gradient parameter of the sigmoid function. By varying the parameter α, we take sigmoid functions of various gradients. Actually, the gradient on the origin is equal to $\alpha/4$. While the gradient parameter approaches infinity, the sigmoid function turns into a threshold function. A threshold function can take only the discrete values of 1 and 0. A sigmoid function can take values of a constant range, from 0 to 1. In addition, the sigmoid function is differentiable while the threshold function is not.

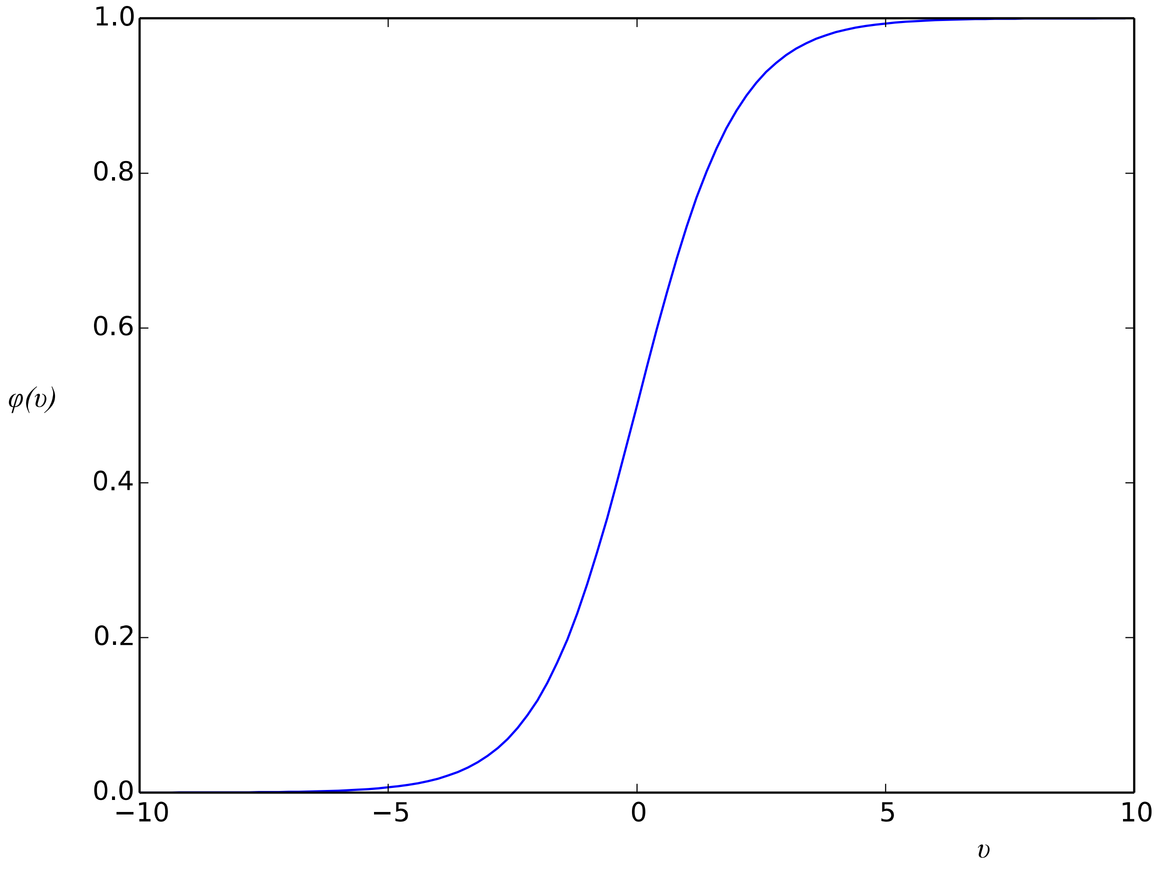


Figure 8: Plot of the sigmoid function.

Both the activation functions (8) and (11) have a range from 0 to +1. Sometimes it is important to have a range from -1 to +1; then the activation function is an odd function of local field. Specifically, the threshold function would be now defined as the widely known sign function:

$$\varphi(v)=\begin{cases} 1, & for\, v>0 \\ 0, & for\, v=0 \\ -1, & for\, v<0 \end{cases} \tag{12}$$

For the sigmoid function, we could use in this case the hyperbolic tangent function, which can get negative values and allows us have practical benefits instead of a logistic function. The sigmoid function would be then defined as:

$$\varphi(v)=\tanh(v) \tag{13}$$

2.2.3 Stochastic Neuron Model

The neuron model of fig. 6 is deterministic, as its behavior is exactly predefined for every input. For certain NN applications, the analysis is desired to be based on a stochastic neuron model. The activation function of McCulloh-Pitts can have a probabilistic interpretation; a neuron can be only in one of two states, e.g. +1 or -1. The decision of the neuron's activation is probabilistic. If *x* is the neuron's state and *P(υ)* is the activation probability, where *υ* is the local field of the neuron, then we can write:

$$x=\begin{cases} +1, & with\, a\, possibility\, of\; P(v) \\ -1, & with\, a\, possibility\, of\; 1-P(v) \end{cases} \tag{14}$$

A typical choice for *P(υ)* is the sigmoid form function:

$$P(v)=\frac{1}{1+\exp(-v/T)} \tag{15}$$

where $\mathcal{T}$ is a pseudotemperature (borrowed by thermodynamics), which is used for the level control of the noise, thus the uncertainty regarding to the activation (Little, 1974). However, it is important to make clear here that $\mathcal{T}$ is not the natural temperature of a NN, even if it is a biological NN or an ANN. $\mathcal{T}$ is just a parameter that controls the thermal fluctuation that represents the affect of synaptic noise. When $\mathcal{T} \rightarrow 0$, the stochastic neuron that is described by the equations (14) and (15), is transformed into a noise-free (deterministic) form, and specifically into the McCulloh-Pitts model [14].

2.2.4 Definition of Neural Network

Let's now express the definition of a neural network [14]:

A neural network is a huge parallel processor with distributed architecture, comprised of simple processing units, called neurons, and natively having the ability to store empirical knowledge and make it available for use. It resembles the human brain in two points:

- The network receives the knowledge from its environment through a learning procedure.
- The strength of the connections among neurons, which is called synaptic weight, is used for the storing of the acquired knowledge.

The learning procedure through which a neural network achieves learning is called learning algorithm and its operation is modify the synaptic weights of the network in an appropriate way for the achievement of the desirable goal.

2.3 Neural Networks as Directed Graphs

We are going to handle the neural networks that will be described here as directed graphs which will be presented with signal flowcharts [14]. A signal flowchart is a network of directive connections (branches) that are interconnected at certain points which are named nodes. Node *j* corresponds to the node signal x_j. There are three basic rules, which should be respected by the flow of the signals:

- <u>First rule</u>: A signal flows along a connection only towards the direction that is defined by the arrow of the connection.

- <u>Second rule</u>: A node signal is equal to the algebraic sum of all the incoming to this node signals through the incoming connections.

- <u>Third rule</u>: A node signal is transmitted to every outgoing connection starting from this node, and its transmission is totally independent of the transfer functions of the outgoing connections.

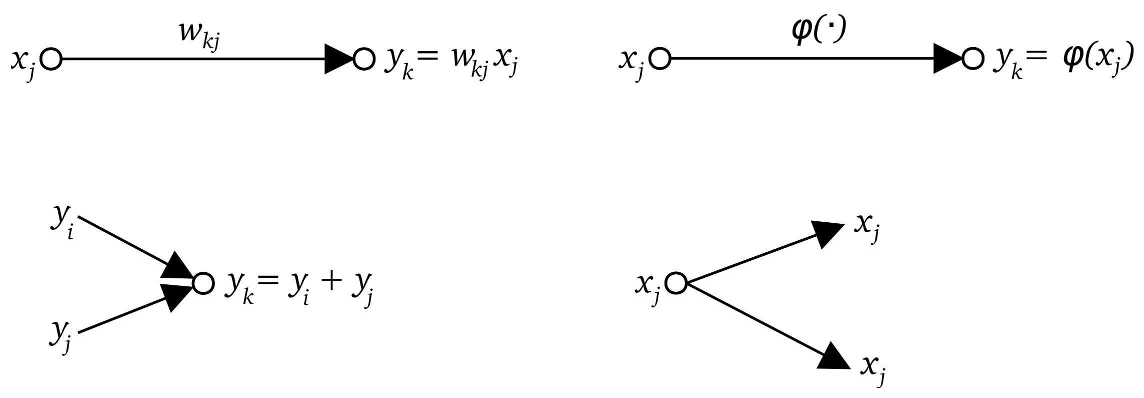


Figure 9: Basic rules for the design of signal flowcharts.

Taking into account the above rules, a neural network can be mathematically defined as a directed graph through the following definition:

A neural network is a directed graph, which is consisted of nodes with interconnected synaptic connections and activation connections, and is characterized by the four following properties:

1. Each neuron is represented by a set of linear synaptic connections, an externally applied bias and one possibly nonlinear activation function. The bias is represented by a synaptic connection that is connected to an input of constant value equal to +1.
2. The synaptic connections of a neuron weighs the corresponding input signals.
3. The weighted sum of the input signals defines the local field of the particular neuron.
4. The activation connection limits the local field of the neuron in order to produce an output.

A directed graph which is defined respecting the above rules is considered full, on the sense that it describes both the signal's flow neuron by neuron, and the signal's flow within each neuron. However, we will use a simplified graph here, which will not take into account the signal flow within a neuron. Such a graph is considered partially full. It is described by the source nodes, which provide the graph with input signals, the computational nodes, each one of which represents a neuron, and the communication connections that interconnect the source and the computational nodes of the network. The communication connections are not weighted; they just show the direction of the signal flow in the graph. This type of graphs actually describes the topology of the graph and because of this, it is named architectural graph.

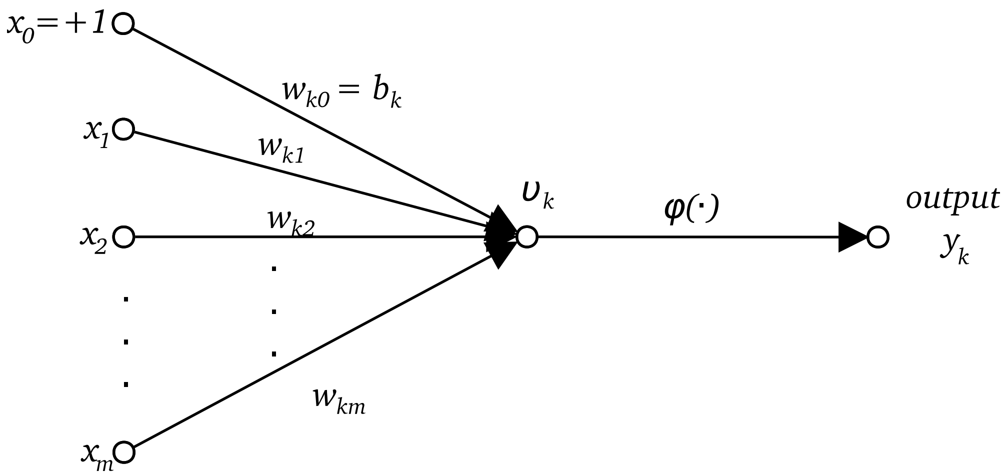


Figure 10: Signal flowchart of a neuron.

To summarize, there are three representations of a neural network:

- the schematic diagram, that describes the operation of the network
- the architectural graph, that describes the topology of the network
- the signal flowchart, that describes the signal flow in the network

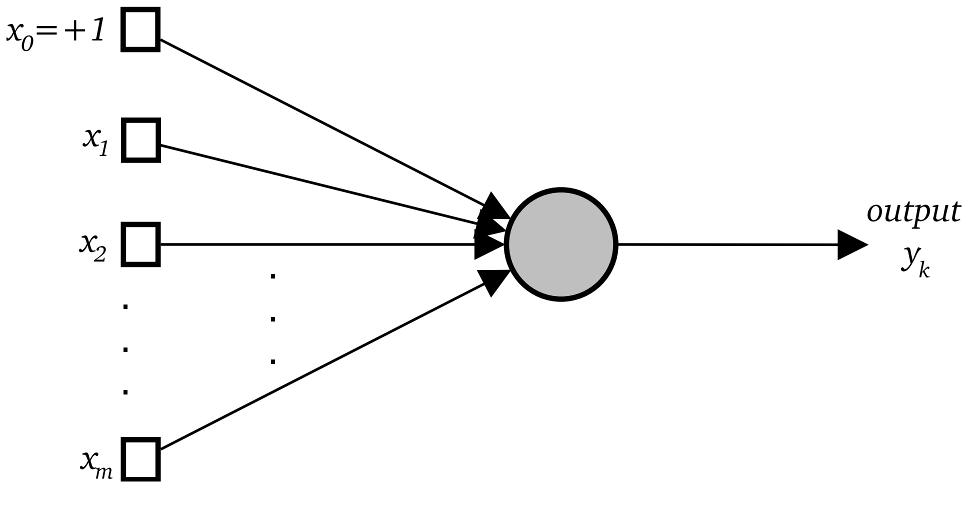


Figure 11: Architectural graph of a neuron.

2.4 Feedback

Feedback [14] exists in a dynamic system whenever the output of a component of the system affects partially the input that is applied to the certain component, causing the appearance of one or more closed paths for the signal transmission in the system.

As a matter of fact, feedback takes place in almost all the parts of the neural network (nervous system) of animals. Feedback is an important factor in the study of a particular category of neural networks, known as recurrent networks.

Figure 12 presents the signal flowchart of a system with single loop feedback. The input signal $x_j(n)$, the internal signal $x'_j(n)$ and the output signal $y_k(n)$ are discrete time functions of the variable *n*.

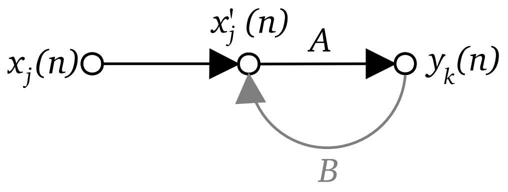


Figure 12: Signal flowchart of a system with single-loop feedback

The system is assumed to be linear, consisted of a feed-forward path and a feed-backwards path, characterized by the operators ***A*** and ***B*** respectively. The relationship between input and output is defined by the following equations:

$$y_k(n) = \boldsymbol{A}[x'_j(n)] \tag{16}$$

$$x'_j(n) = x_j(n) + \boldsymbol{B}[y_k(n)] \tag{17}$$

$$y_k(n) = \frac{\boldsymbol{A}}{1 - \boldsymbol{AB}}[x_j(n)] \tag{18}$$

The factor ***A/(1-AB)*** is called closed-loop operator and ***AB*** is called open-loop operator. In general, ***AB≠BA***.

The analysis of the dynamic behavior of neural networks containing feedback gets complicated by the fact that the processing units that construct the network are usually non-linear.

2.5 Neural Networks Architectures

The learning algorithm that is used for the training of the network is closely related to the way that the neurons of the network are structured. Hence, the learning algorithms or rules that are used in neural networks design should be treated as structured. There are three fundamental ways of structure or architectures of neural networks and they are described below [14].

2.5.1 Single-layer Feedforward Network

The neurons are organized in layers in a neural network. The simplest neural network has an input layer that is consisted of source nodes and is directly connected to an output neurons layer, which act as computational nodes, but not vice versa. This is a feedforward network, as seen in fig. 13. It is a single layer network; the input layer is not admeasured as there is no computation taking place there.

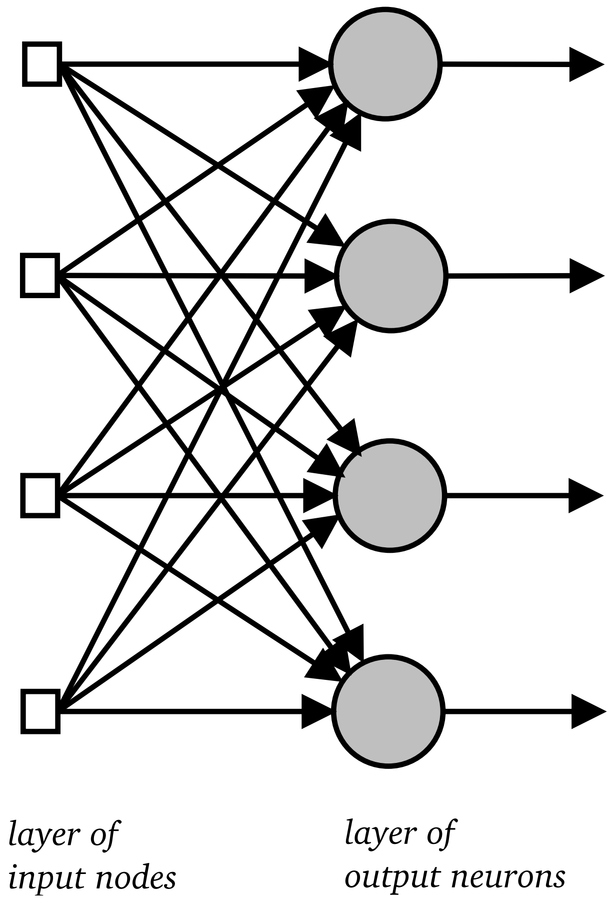


Figure 13: Feedforward network with a single layer of neurons

2.5.2 Multi-layer Feedforward Network

This architecture consists of one or more hidden layers. The computational nodes of each layer are called hidden neurons or hidden units; the term “hidden” refers to the fact that this part of the neural network is not directly seen from the input or the output of the network. The hidden neurons intervene between the externally provided input and the output of the network in a useful manner. The source nodes at input layer provide the first hidden layer nodes with the input vector. The output signals of the first hidden layer are provided as inputs to the second hidden layer, which after some computation, provides them to the next hidden layer. Typically, the neurons of each layer take as inputs the output signals of the previous layer. This procedure continues for all the hidden layers and finishes when they reach the output layer. The set of the output signals consist the total response of the network in the activation pattern that is provided by the source nodes at the input layer.

Figure 14 presents the architecture of such a network. The network of this figure is fully connected, as each node of each layer in the network is connected to all the nodes of the next layer. If some of the synaptic connections were absent, the network would be partially connected.

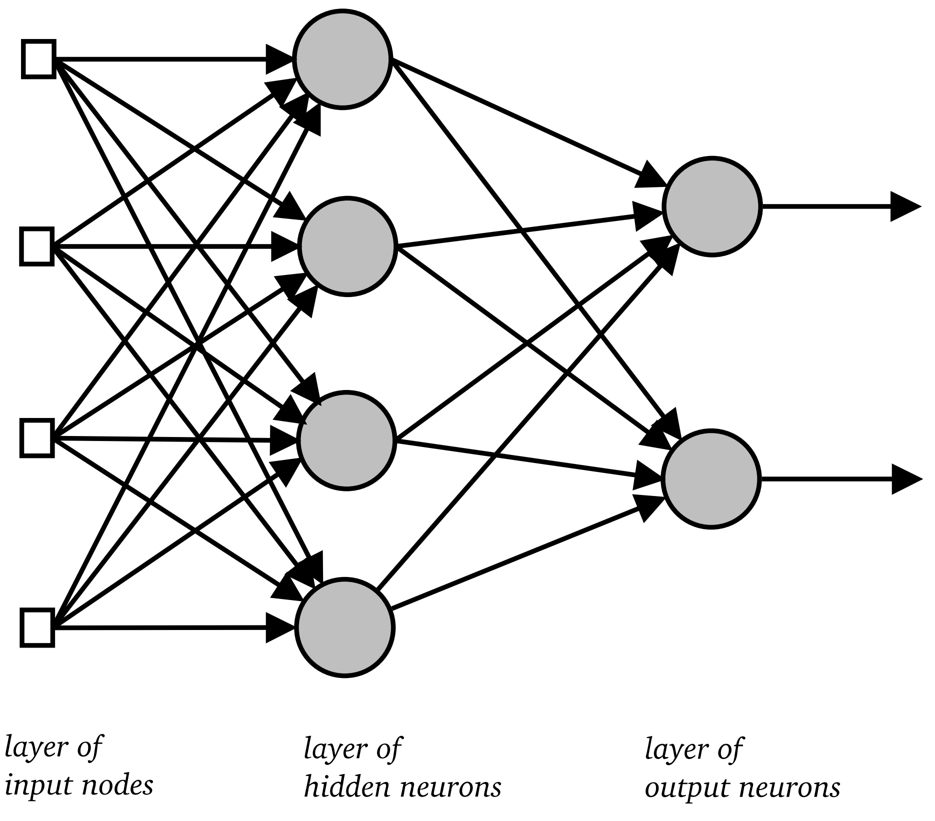


Figure 14: Fully connected feedforward network with a hidden layer and an output layer

2.5.3 Recurrent Neural Network

A recurrent neural network has at least one feedback loop. Figure 15 presents a recurrent neural network, which has a single-layer of neurons, with each neuron feeding its output signal back to the inputs of all the other neurons. In the architecture of this figure, there is no auto-feedback in the network; there is auto-feedback in a network when the output of a neuron feeds the input of the same neuron.

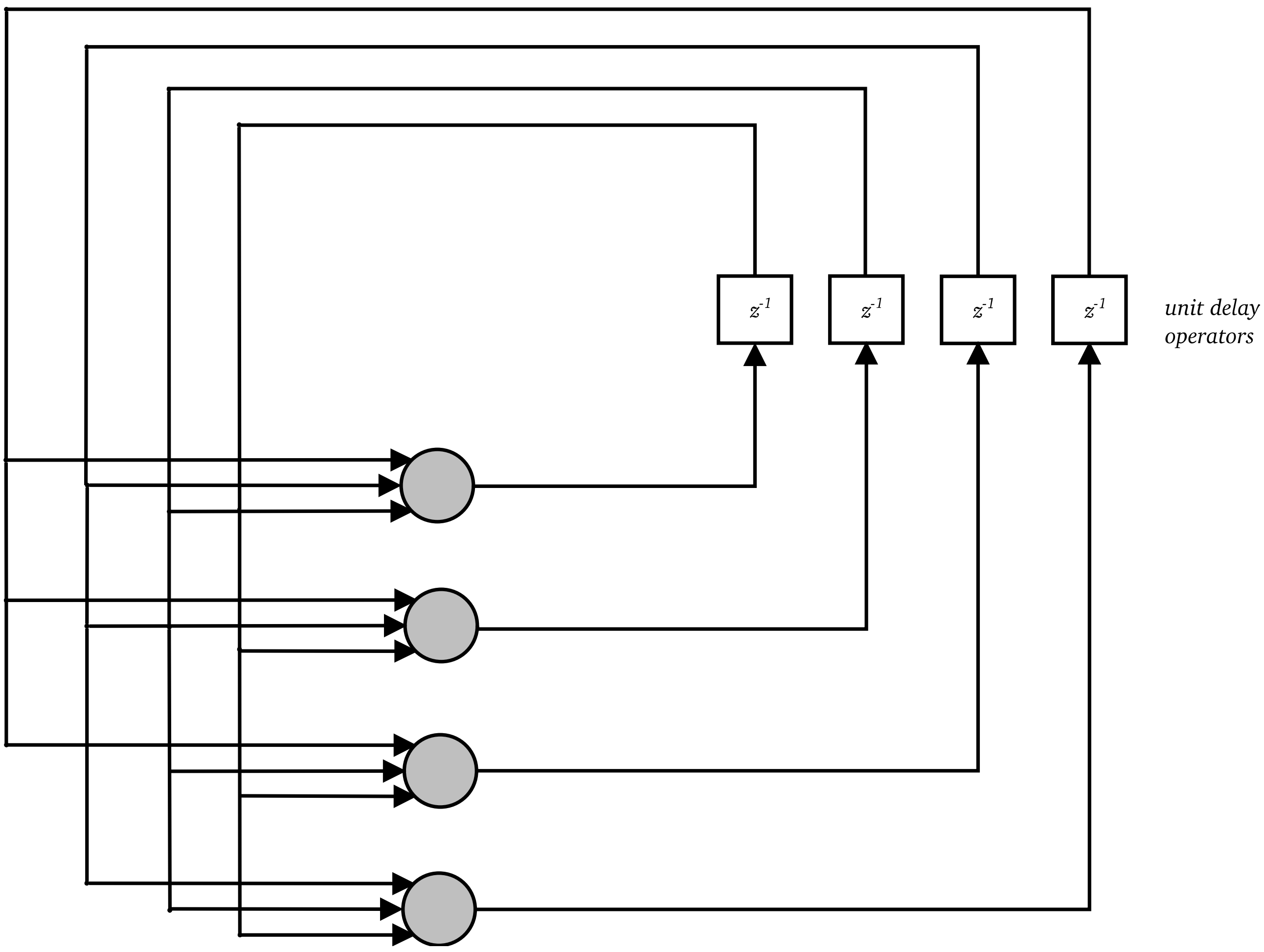


Figure 15: Recurrent network without auto-feedback loops and hidden neurons.

Figure 16 presents a different category of recurrent networks with hidden neurons. The feedback connections of this figure come from both the hidden and the output neurons.

Feedback loops are very essential and have a great effect on the learning ability of the network, in both forms of figures 15 and 16. Furthermore, the feedback loops require the use of certain branches, consisted of elements of unit time delay (symbolized as z^{-1}), which result to the non-linear behavior of the network, having supposed that it includes non-linear units.

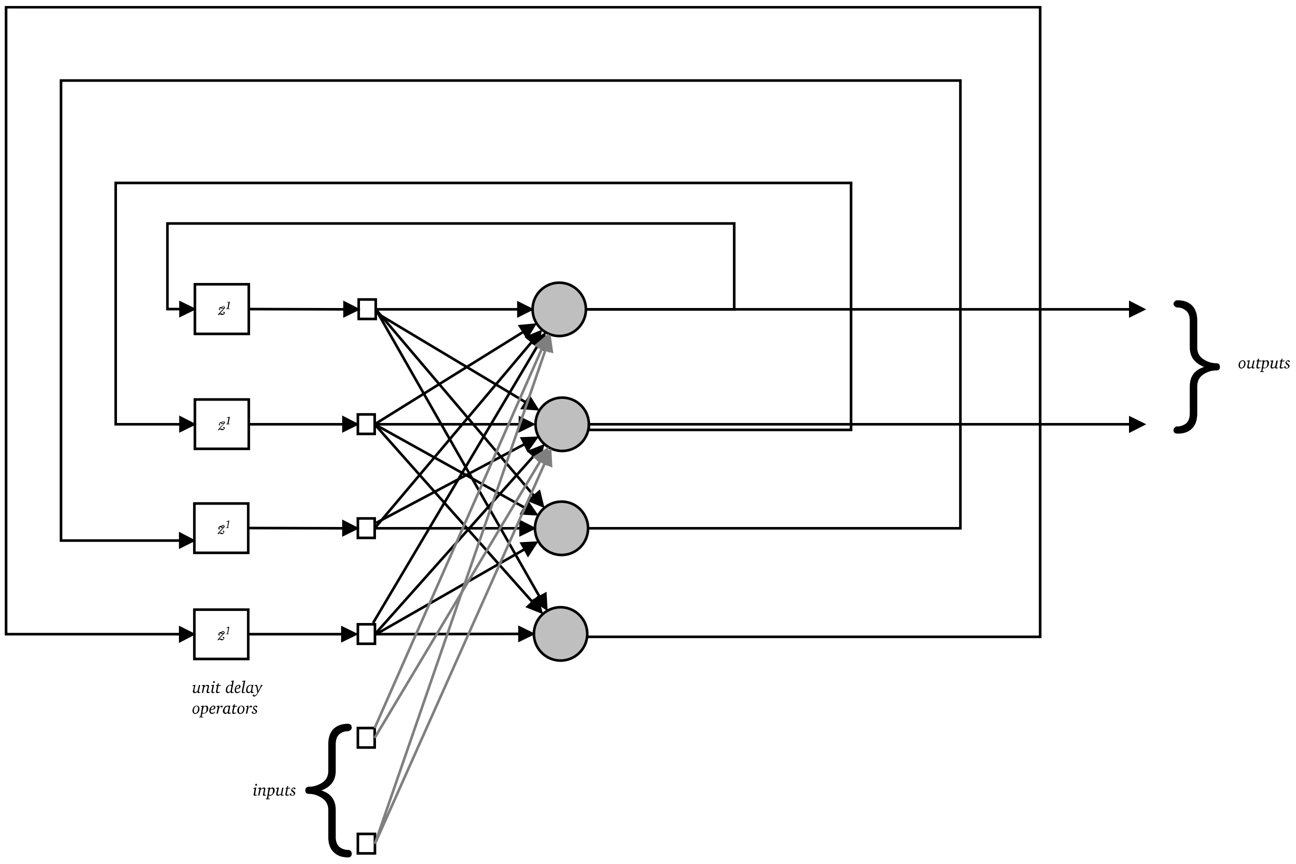


Figure 16: Recurrent network with hidden neurons.

2.6 Knowledge Representation

Fischler and Firschein, in 1987, expressed the following definition of knowledge:

Knowledge is the stored information or the models that are used by an individual or a machine in order to interpret, predict and respond appropriately to the external world [16].

Intelligence and problem solving depend upon the use of stored knowledge and information about objects, processes, goals, causality, time and action. The study of knowledge, memory, symbols and mental representations are areas of high interest for cognitive scientists and psychologists as well as Artificial Intelligence (AI) researchers. Cognitive science researchers have studied the representations used by people in the process of solving different types of problems.

A problem in AI is translated into a system for representing the types of knowledge and reasoning. Knowledge representation may consist of hierarchical, sequential and spatial relationships with names, facts, procedures and constraints. Models are language representations for describing the world. Logic and mathematical systems are also representations of formal languages for representing the world. For example, a natural language is a representational system which obviously has a language and a structure or syntax for both verbal and written communication [17].

The main characteristics of knowledge representation are the following ones: what information is really represented in explicit form and how the information is encoded for future use. The possible forms of knowledge representation, from the inputs to the internal parameters of a network, may vary significantly, and this makes the development of a satisfactory solution through a neural network a great challenge.

An essential task for a neural network is to learn a model of the world (its environment), to which it belongs, and maintain this model sufficiently consistent to the real world in order to be able to achieve the goals for which it was initially designed. The knowledge of the world is consisted of two kinds of information:

- The known state of the world, which is represented by facts and hints of what has already been known; this kind of information is mentioned as past information.
- The observations (and measurements) on the world, which are received by sensors that were designed in order to monitor the environment in which the neural network operates. Usually, these observations contain noise, due to the noise collected by the sensors and the system's imperfections. In all cases, the observations that are received in this way consist an information pool; the examples that are used for the neural network's training are pulled from it.

The examples can be labeled (annotated) or unlabeled (unannotated). In the labeled examples, every example representing an input signal corresponds to a desirable response. On the other hand, the unlabeled examples are consisted of various implementations of the input signal only. In any case, a set of examples represents the knowledge about the environment, which a neural network can learn through the training. We should note here that the labeled examples may be expensive to acquire as they require the availability of a trainer who will provide a desirable and accurate response for every input example. In opposition, unlabeled examples are in abundance and do not require such a supervision from a trainer.

The fundamental difference between the design of a neural network and the design of traditional information processing network is in the mechanism that executes the pattern classification. In a traditional network implementation, usually we move on based on a mathematical model of environmental observations, verifying its correctness with real data, and then basing the network's design on this model. In opposition, the design of a neural network is based directly on the real world's data; the dataset can “speak” and reveal information on its behalf. Hence, the neural network not only provides the implied model of the environment, but it also executes the desirable information processing.

The examples set that is used for the training of a neural network can contain positive and negative examples. In a neural network of a certain architecture, the representation of the knowledge of the environment is defined by the values of the free parameters (which are the synaptic weights and the bias) of the network. The form of this knowledge representation consists the design of the network and, consequently, is the key for its performance [14].

2.7 Learning Procedures

There are various ways in which a neural network is able to learn [14]. The ways that were described in section 1.2.2, for learning approaches of sentiment analysis tasks, are inspired by the approaches we are going to describe here. These approaches were inspired by human learning and try to simulate it. In a broad sense, the categories of learning procedures are the following: learning with a trainer (or supervised learning) and learning without trainer. Learning without trainer can be divided into the subcategories of unsupervised learning and reinforcement learning.

2.7.1 Learning with a Trainer or Supervised Learning

The trainer knows the environment and this knowledge is represented by a set of input-output examples. However, the environment is unknown to the neural network. Let's assume that the trainer and the neural network are exhibited to a training vector, which is an example alleged from the environment. Due to its native knowledge, the trainer is able to provide the neural network with a desired response for the certain training vector. The desired response represents the optimal action that the neural network should perform. The network parameters are fitted by the combination of the training vector and the error signal. The error signal is defined as the difference between the desired response and the real response of the network. The network parameters are fitted in a repetitive way, step by step, and its goal is to bring the neural network in a state where it can simulate the trainer's behavior. This simulation is considered optimal under a statistical sense.

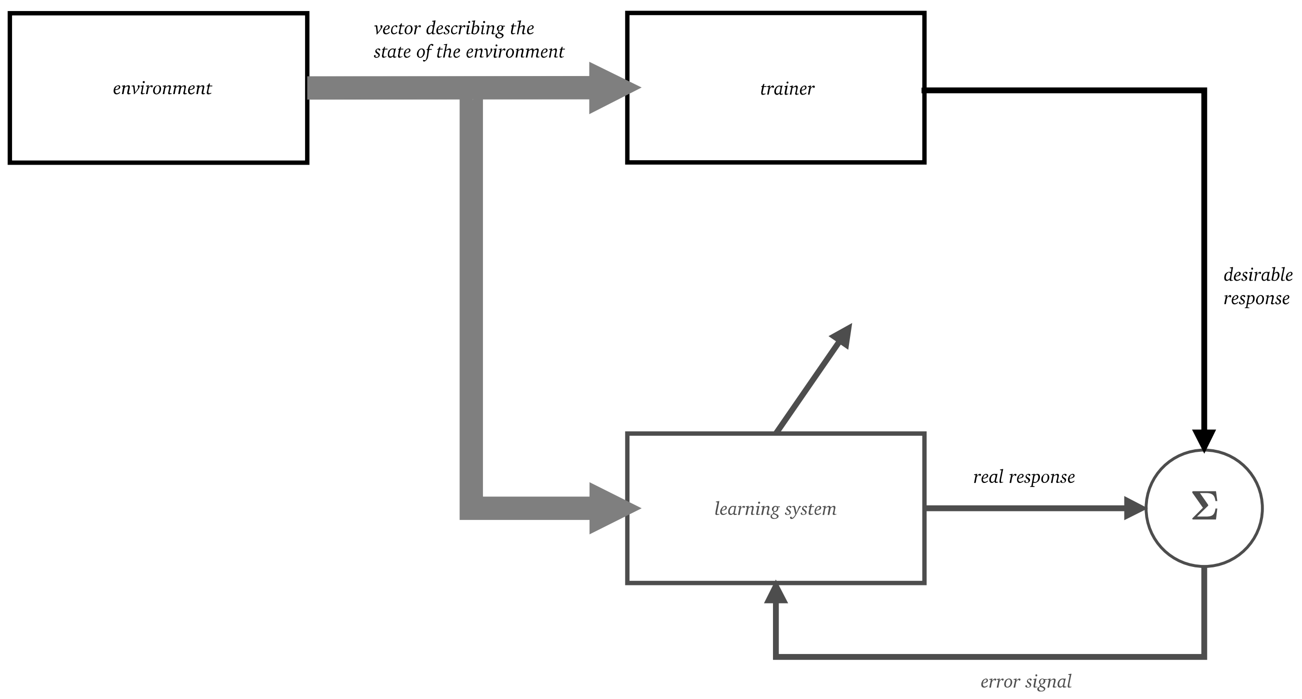


Figure 17: Schematic diagram of supervised learning; the gray part of the diagram consists a feedback loop.

The above consist the base of learning through error correction. In fig. 17 the gray part of the diagram consists a closed loop feedback system. The unknown environment is out of this loop. The mean squared error (MSE) or the error sum of squares (SSE) on the training sample, defined as function of the free parameters (which are the synaptic weights), can be used as a performance metric of the system. This operation can be seen as a multidimensional surface of error-performance, or simply an error surface, with the free parameters acting as coordinates. The real error surface is calculated by the average of all possible input-output examples. Every specific operation of the system, under the supervision of the trainer, is represented by a point on the error surface. This point must be moved successively downwards, towards a minimum point of the error surface, in order to improve the system's performance over time and make the system learn by the trainer. The minimum point can be a local or a global minimum. A supervised learning system is able to achieve this by using the available useful information regarding the gradient of the error surface corresponding to the current system's behavior. The gradient of the error surface at a point is represented by a vector towards the direction of the steepest descent. In fact, in the case of supervised learning through examples, the system can use an instantaneous evaluation of the gradient vector, with time indicators for the examples. The use of such an evaluation has as a result the movement of the operating point on the error surface, which typically has the form of a random path. However, given an algorithm that is specialized to minimize the cost function, a sufficient example of inputs-outputs and sufficient time for the training, a supervised learning system is usually able to approach an unknown, reasonably fine, pairing of inputs-outputs.

2.7.2 Learning without Trainer

The learning procedure takes place without the guidance of a trainer. This actually means that there are no labeled examples of the operation that the network aims to learn. There are two subcategories of this procedure, that are described below.

2.7.2.1 Reinforcement Learning

In reinforcement learning, an input-output mapping is learned through the constant interaction between the network and its environment in order to minimize a scalar performance indicator.

Fig. 18 presents the schematic diagram of the procedure of reinforcement learning. The base of this procedure is a mechanism that functions as a “judge”. The judge converts the main reinforcement signal, that is received from the environment, into a higher-quality reinforcement signal, which is named heuristic reinforcement signal. Both these signals are scalar inputs (Barto et al. 1983). The system is designed in such a way that it is able to learn through *delayed reinforcement*, which actually means that the system observes a time sequence of stimuli, received from the environment, which end up to the production of the heuristic reinforcement signal.

Reinforcement learning aims to minimize a current error function, which is defined as the prediction of the summed cost of actions that are executed in a step sequence. Several of the executed procedures of this step sequence are possibly the best determinants of the total system behavior. The learning system function is to discover these actions and to feed them back, to the environment.

Delayed Reinforcement learning is difficult to be executed due to the following factors: There is no trainer to provide a desired response at every step of the learning procedure, and the delay with which every reinforcement signal is produced suggests that learning machine has to solve a time problem of trust assignment. This means that the learning machine must be able to determine individually the success degree individually for each action of the time step sequence that drove to the final result, while the main reinforcement mechanism may evaluate only the final result.

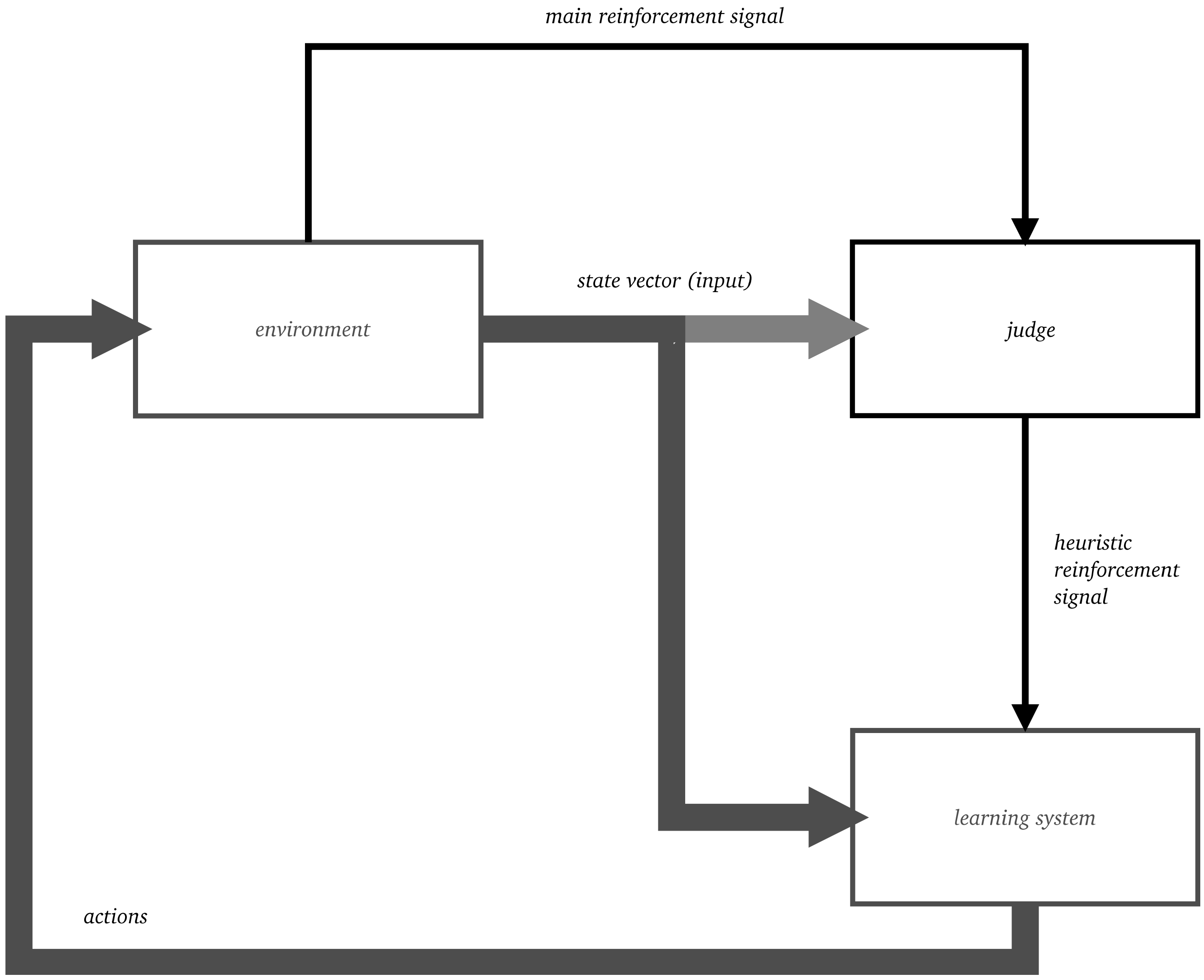


Figure 18: Schematic diagram of reinforcement learning; both the learning system and the environment are in the feedback loop.

2.7.2.2 Unsupervised Learning

In unsupervised or self organized learning there is no external trainer or judge supervising the learning procedure. Instead, an independent of the task metric of the quality of the representation is used, and this is assigned to train the network. The network's free parameters are optimized compared to this metric. When the network is coordinated with the statistic regularities of the input data, it develops the ability to form internal representations for the encoding of the input features, and, through them, to automatically create new classes (Becker, 1991).

For the execution of unsupervised learning, we can use a rule of competitive learning. A very simple example of this would be a neural network consisted of two layers: an input layer and a competitive layer. The input layer receives the available data. The competitive layer is consisted of neurons, of which each one competes the others, according a learning rule, for the chance to respond to features that are contained in the input data. In its simplest possible form, the network functions under the strategy “the winner takes everything”. Based on such a strategy, the neuron with the highest total input “wins” in the competition and is activated; all the other network neurons are deactivated.

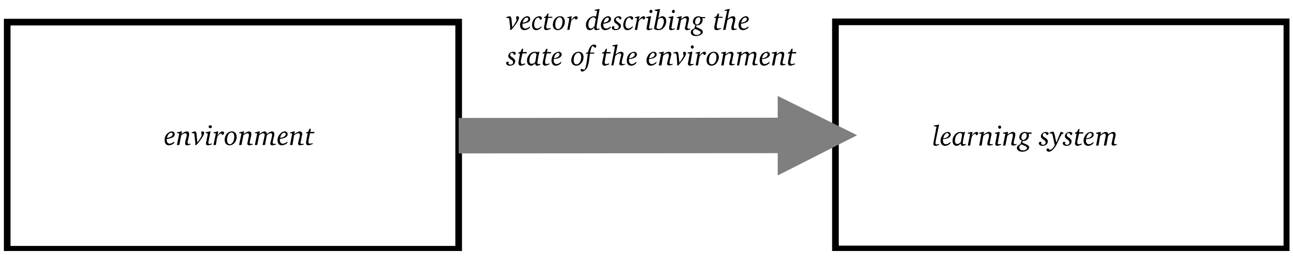


Figure 19: Schematic diagram of unsupervised learning.

2.8 Rosenblatt's Perceptron

2.8.1 The Perceptron

Perceptron (“sensor”) was the first neural network that was able to be described with an algorithm. It was invented by Rosenblatt in 1958 and is based on a non-linear neuron (the neuron model of McCulloch-Pitts) [14]. This model is consisted of a linear combiner followed by a hard limiter. The latter executes the sign function, as presented in figure 20. The summing node of the neural model calculates a linear combination of the inputs that are applied to its synapses, and, additionally, it incorporates an externally applied bias (or predisposition). The resulting sum or induced local field is applied to a hard limiter. In response, the neuron produces an output equal to +1 if the input of the hard limiter is positive, and -1 if the input is negative.

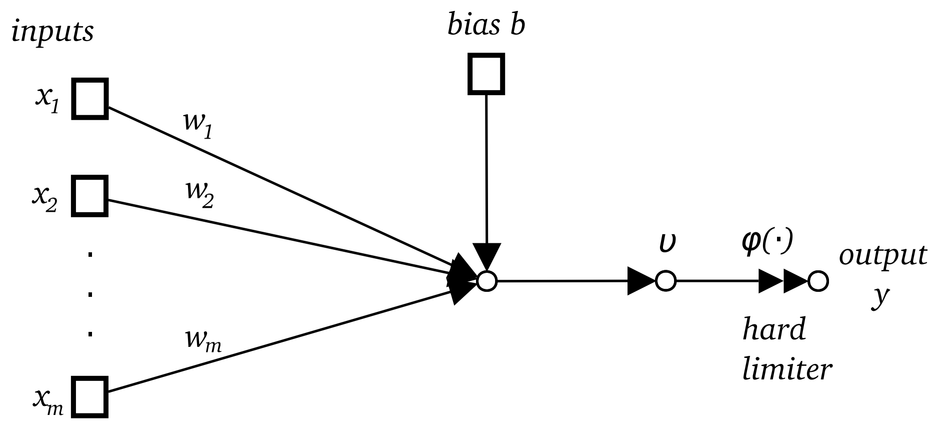


Figure 20: Signal flowchart of perceptron.

In the flowchart of fig. 20, the perceptron's synaptic weights are symbolized by $w_1, w_2, ..., w_m$. The perceptron's inputs are respectively symbolized by $x_1, x_2, ..., x_m$, and the externally applied bias is symbolized by b. The input of the hard limiter (or the neuron's local field) is equal to:

$$\upsilon = \sum_{i=1}^{m} w_i x_i + b \tag{19}$$

Perceptron's goal is the correct classification of a set of externally applied stimulations $x_1, x_2, ..., x_m$ into one of two classes, C_1 or C_2. The decision rule for the classification dictates the mapping of an item represented by the inputs $x_1, x_2, ..., x_m$ to the class C_1 if the perceptron's output y equals to +1, or to the class C_2 if y equals -1.

It is common to represent a map of the decision areas in the m-dimensional space of signals, due to the m input variables. In the simplest possible form of a perceptron, there are two decision areas, separated by a hyperplane, defined by the following equation:

$$\sum_{i=1}^{m} w_i x_i + b = 0 \tag{20}$$

Fig. 21 presents the case of two input variables, x_1 and x_2. In this case, the decision bound is a straight line. An item (x_1, x_2) that is above the boundary line is classified to the class C_1, while an item (x_1, x_2) that is below the boundary line is classified to the class C_2. Bias b affects on the displacement of the decision limit from the origin.

The synaptic weights w_1, w_2, ..., w_m can be fitted through a repetitive procedure and an error-correction rule, which is known as perceptron's convergence algorithm.

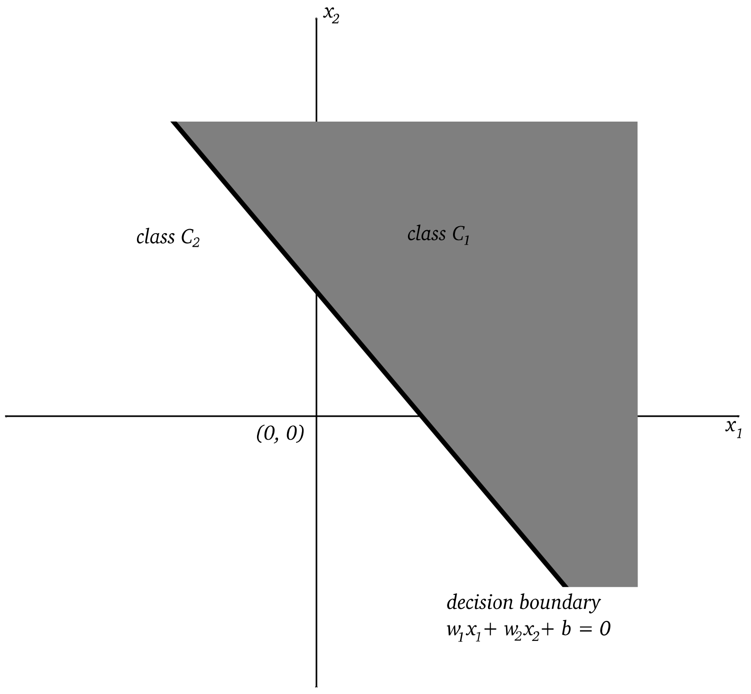


Figure 21: Hyperplane as decision boundary for a binary classification task.

2.8.2 Perceptron's Convergence Theorem

In order to express perceptron's error-correction learning algorithm, it is more convenient to use a modified model, which is presented in the flowchart of fig. 22, and is equivalent to the model presented in the flowchart of fig. 20. In this model, the bias *b(n)* is treated as a synaptic weight driven from a stable input which is equal to +1. Hence, the inputs vector *(m+1)x1* is defined as

$$\boldsymbol{x}(n)=[+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$$

where *n* symbolizes the time step of the application of the algorithm. The weights vector is defined as:

$$\boldsymbol{w}(n)=[b, w_1(n), w_2(n), \ldots, w_m(n)]^T$$

Consequently, the output of the linear combiner can be written as

$$\upsilon(n) = \sum_{i=0}^{m} w_i(n) x_i(n) = \mathbf{w}^T(n)\mathbf{x}(n) \quad (21)$$

$w_0(n)$ is the synaptic weight for *$i = 0$* and actually represents the bias *b*.

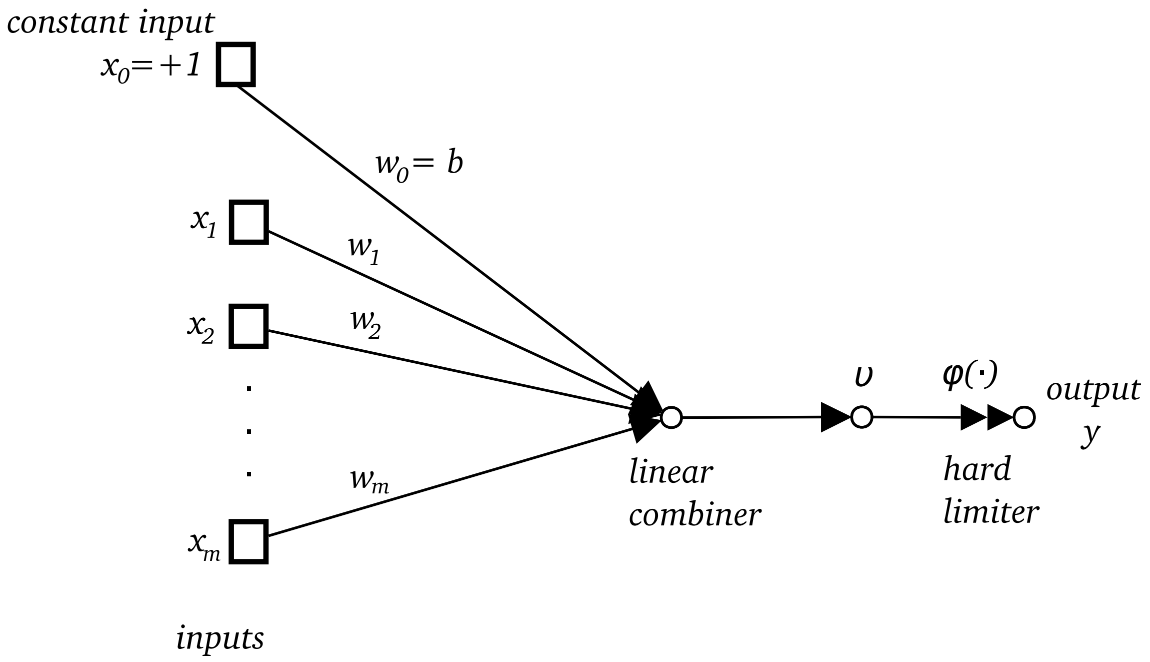


Figure 22: Equivalent signal flowchart of perceptron

If *n* is constant and b preset, if we map the equation $\mathbf{w}^T\mathbf{x} = 0$ in a m-dimensional space with coordinates $x_1, x_2, ..., x_m$, a hyperplane is defined, as the boundary decision surface between two separate classes of inputs.

Classes C_1 and C_2 must be linearly separable so that the perceptron can work properly. This means that the patterns for classification should be sufficiently separable so that the decision surface is consisted of one hyperplane. This requirement is presented in fig. 23(a), for the case of a perceptron of 2 dimensions. In this figure, C_1 and C_2 are sufficiently separable and we can draw a hyperplane (a line) as the decision boundary. However in fig. 23(b), the classes C_1 and C_2 are very close, and hence they are non-linearly separable; this is a state out of the perceptron's computational ability.

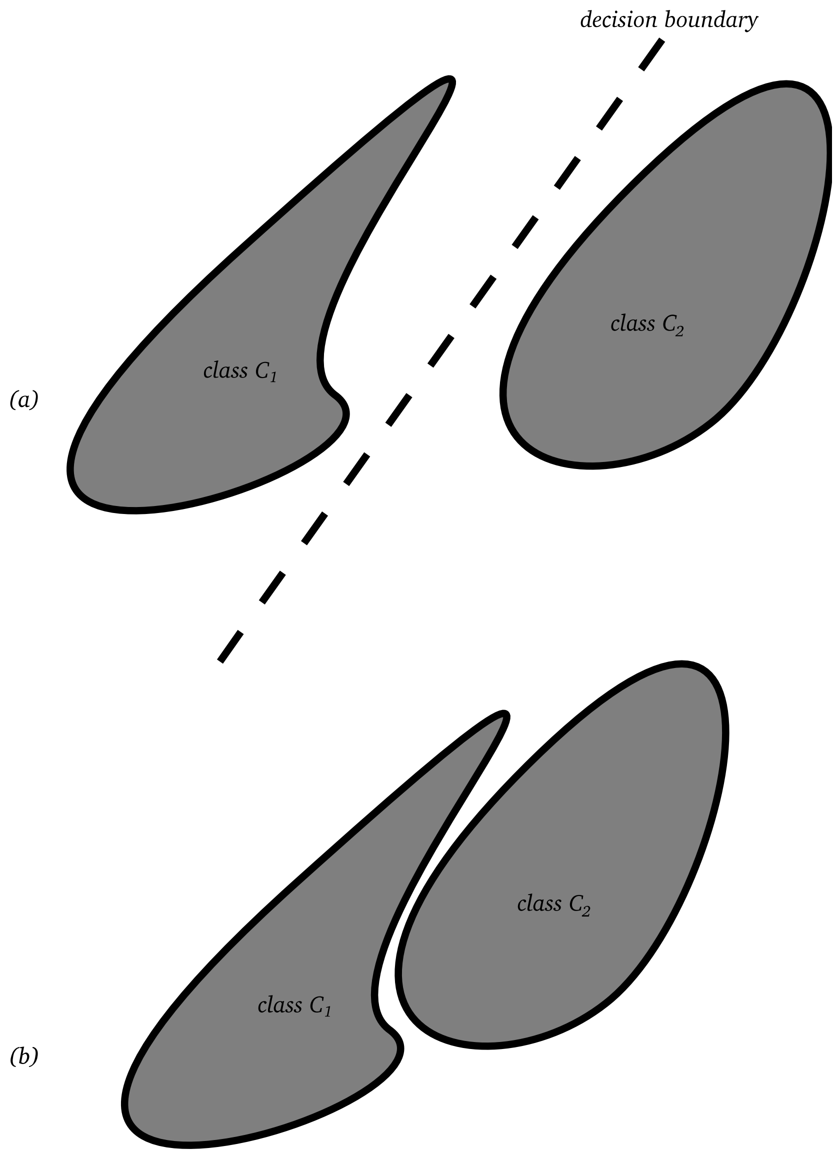


Figure 23: (a) Pair of linearly separable patterns. (b) Pair of non-linearly separable patterns.

Assuming that the perceptron's input variables come from two linearly separable classes, let H_1 and H_2 be the subspaces of the following training vectors, respectively,

$$\boldsymbol{x}_1(1), \boldsymbol{x}_1(2), \ldots \in C_1$$
$$\boldsymbol{x}_2(1), \boldsymbol{x}_2(2), \ldots \in C_2$$

$$H = H_1 \cup H_2$$

H symbolizes the full space. Given the vectors H_1 and H_2 for the classifier's training, the training procedure requires the weights vector's fitting, so that C_1 and C_2 are linearly separable. This means that there is a weights vector w such that we can denote that:

$$\begin{aligned} \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}>0 \; \textit{for every inputs } \boldsymbol{x} \textit{ vector} \in C_1 \\ \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}\leq 0 \; \textit{for every inputs } \boldsymbol{x} \textit{ vector} \in C_1 \end{aligned} \tag{22}$$

For (22) we have arbitrarily decided that the inputs $\boldsymbol{x}$ vector belongs to C_2 if $\boldsymbol{w}^T\boldsymbol{x} = 0$. Given the training vectors subsets, H_1 and H_2, the perceptron's training problem consists on finding a weights vector $\boldsymbol{w}$, such that the inequalities of formula (22) are satisfied.

The algorithm for the perceptron's weights vector fitting can be formulated as following:

1. If the *n*-th member of a training subset, $\boldsymbol{x(n)}$, is correctly classified from the weights vector $\boldsymbol{w(n)}$ calculated in the *n*-th iteration of the algorithm, no correction takes place for the perceptron's weights vector, according the following rule:

$$\begin{aligned} \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}>0 \; \textit{for every inputs } \boldsymbol{x} \textit{ vector} \in C_1 \\ \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}\leq 0 \; \textit{for every inputs } \boldsymbol{x} \textit{ vector} \in C_1 \end{aligned} \tag{23}$$

2. Otherwise, the perceptron's weights vector $\boldsymbol{w(n)}$ is updated according the following rule:

$$\begin{aligned} \boldsymbol{w}(n+1)=\boldsymbol{w}(n)-\eta(n)\boldsymbol{x}(n) \textit{if } \boldsymbol{w}^{\boldsymbol{T}}(n)\boldsymbol{x}(n)>0 \wedge \boldsymbol{x}(n)\in C_2 \\ \boldsymbol{w}(n+1)=\boldsymbol{w}(n)-\eta(n)\boldsymbol{x}(n) \textit{if } \boldsymbol{w}^{\boldsymbol{T}}(n)\boldsymbol{x}(n)\leq 0 \wedge \boldsymbol{x}(n)\in C_1 \end{aligned} \tag{24}$$

where the learning rate parameter $\eta(n)$ controls the fitting applied to the weights vector in the *n*-th iteration.

If $\eta(n) = \eta > 0$, where η is a constant independent of the iteration number *n*, then we get a fitting rule through constant increment for the perceptron, which can be formulated in the following theorem:

Perceptron's Convergence Theorem with constant increment:

If two subsets of the training vectors H_1 and H_2 are linearly separable, and the inputs that are applied to the perceptron belong to one of these subsets, then the perceptron converges after n_0 iterations, in the sense that

$$\boldsymbol{w}(n_0)=\boldsymbol{w}(n_0+1)=\boldsymbol{w}(n_0+2)=\ldots \tag{25}$$

is a solutions vector for $n_0=n_{max}$.

Let's now examine the absolute error correction procedure for the single-layer perceptron's fitting, for which $\eta(n)$ is variable. Specifically, we assume that $\eta(n)$ is the minimum integer for which it is

$$\eta(n)\boldsymbol{x}^{\boldsymbol{T}}(n)\boldsymbol{x}(n)>|\boldsymbol{w}^{\boldsymbol{T}}(n)\boldsymbol{x}(n)|$$

With this procedure, we find that if the dot product $\boldsymbol{w}^T(n)\boldsymbol{x}(n)$ in the n-th iteration has a wrong sign, then $\boldsymbol{w}^T(n+1)\boldsymbol{x}(n)$ in $(n+1)$-th iteration will have the correct sign. This shows that if $\boldsymbol{w}^T(n)\boldsymbol{x}(n)$ has a wrong sign in iteration n, then we can notify the training procedure for the $(n+1)$ iteration by setting $\boldsymbol{x}(n+1) = \boldsymbol{x}(n)$. In other words, every pattern is presented repeatedly in the perceptron until it is classified correctly.

The following algorithm is the perceptron's convergence algorithm. The symbol $sgn(\cdot)$ represents the sign function:

$$sgn(v)=\begin{cases}+1, & for\, v>0\\ -1, & for\, v<0\end{cases} \tag{26}$$

The perceptron's quantized response $y(n)$ can be expressed in the following compact form:

$$y(n)=sgn[\boldsymbol{w}^{\boldsymbol{T}}(n)\boldsymbol{x}(n)] \tag{27}$$

2.8.3 Summary of Perceptron's Convergence Algorithm

The summary of the steps of perceptron's convergence algorithm is described below [14].

Variables and Parameters:

x(n) = inputs vector (m+1)-by-1 = $[+1, x_1(n), x_2(n), ..., x_m(n)]^T$

w(n) = weights vector (m+1)-by-1 = $[b, w_1(n), w_2(n), ..., w_m(n)]^T$

b = bias

y(n) = real response (quantized)

d(n) = desirable response

η = learning rate parameter, positive constant less than 1

1. Initialization: Set ***w(0)*** = ***0***. Then execute the following computations for a time step *n* = 1, 2, … .

2. Activation: In time step *n*, activate the perceptron applying the (constant values) inputs vector ***x(n)*** and the desirable response *d(n)*.

3. Computation of real response: Compute the real response of the perceptron *y(n)* = sgn[$\boldsymbol{w}^T(n)\boldsymbol{x}(n)$], where sgn(·) is the sign function.

4. Fitting the weights vector: Update the perceptron's weights vector in order to end up to:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta[d(n) - y(n)]\boldsymbol{x}(n)$$
$$where\, d(n) = \begin{cases} +1, & if\ \boldsymbol{x}(n) \in C1 \\ -1, & if\ \boldsymbol{x}(n) \in C2 \end{cases}$$

5. Continuation: Increase the time step *n* by 1 and go to step 2.

The weights vector ***w(n)*** is a vector *(m+1)-by-1*, the first element of which equals to the bias b. Another important note for the perceptron's convergence algorithm is that we have introduced a quantized desirable response *d(n)*, which is defined as:

$$d(n) = \begin{cases} +1, & if\ \boldsymbol{x}(n) \in C1 \\ -1, & if\ \boldsymbol{x}(n) \in C2 \end{cases} \tag{28}$$

Thus, the fitting of the weights vector w(n) is summed up in the form of the learning rule with error correction:

$$\boldsymbol{w}(n+1)=\boldsymbol{w}(n)+\eta[d(n)-y(n)]\boldsymbol{x}(n) \tag{29}$$

where η is the learning rate parameter. and the difference *d(n)-y(n)* acts as an error signal. The learning rate parameter is a positive constant that is limited within the range $0<\eta\leq 1$. When it is assigned to a value from this range, we have to have in mind two conflicting requirements (Lippman, 1987):

- The computation of the average past inputs for the provision of constant evaluations for the weights, requires a small η.
- The quick fitting regarding the real changes in the underlying distributions of the procedure that is responsible for the production of the inputs vector $\boldsymbol{x}$ requires a large η.

2.9 Multi-layer Perceptron

2.9.1 Definition

A multi-layer perceptron (MLP) [14] is a feedforward artificial neural network model that maps sets of input data to a set of appropriate outputs. A multi-layer perceptron consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one (which means that each neuron on any layer is connected to all the neurons of the previous layer). Except for the input nodes, each node is a neuron with a non-linear activation function.

Figure 24 presents the architectural graph of a multi-layer perceptron with two hidden layers and an output layer. The signal flows forward, from left to right and from a layer to its next layer. There are operating signals and error signals, as seen in fig. 25. An operating signal is a stimulation (an input signal) that arrives at the input of the network, is transmitted forward (neuron by neuron) in the whole network and, eventually, arrives at the output of the network as an output signal. An error signal comes from an output neuron and is transmitted backwards (layer by layer) through the network.

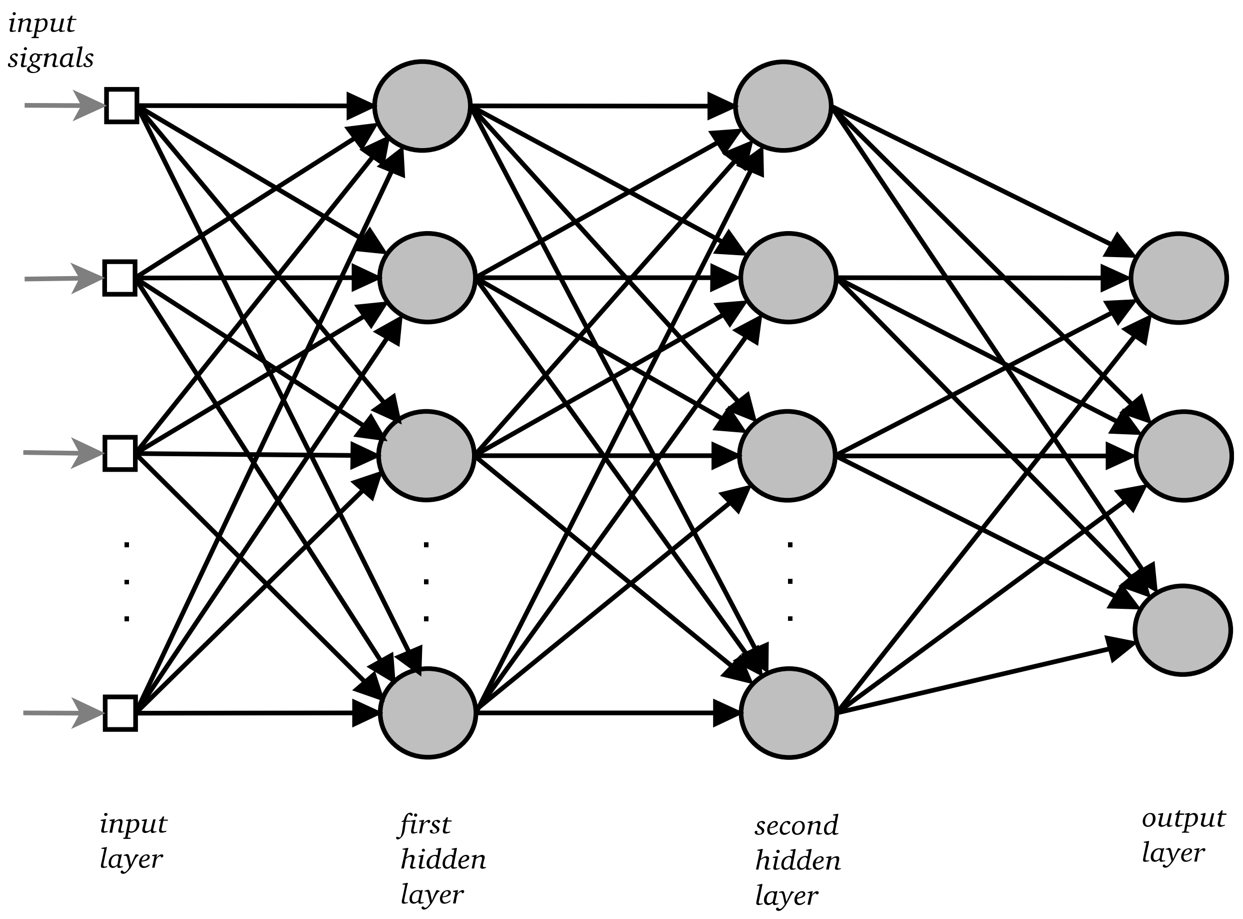


Figure 24: Architectural graph of a multi-layer perceptron with two hidden layers.

The neurons of the output consist the output layer of the network. The rest of the neurons consist the hidden layers of the network. It is clear that the hidden layers are neither part of the input nor of the output of the network. The first hidden layer is fed by the input layer, which is consisted of sensor units (source nodes). The outputs of the first hidden layer feeds as inputs the second hidden layer, and so on for the rest of the network.

Every hidden neuron and every output neuron of a multi-layer perceptron is designed to perform two kinds of computations. The first one is the computation of the operating signal showing up at the output of each neuron, and which is expressed as a constant non-linear function of the input signal and the synaptic weights that are related to the neuron. The second computational task of the neuron is the computation of an estimation of the gradient vector, which is necessary for the phase of the backwards development of the network. The gradient vector is actually the gradients of the error surface compared with the weights that are connected to the inputs of a neuron.

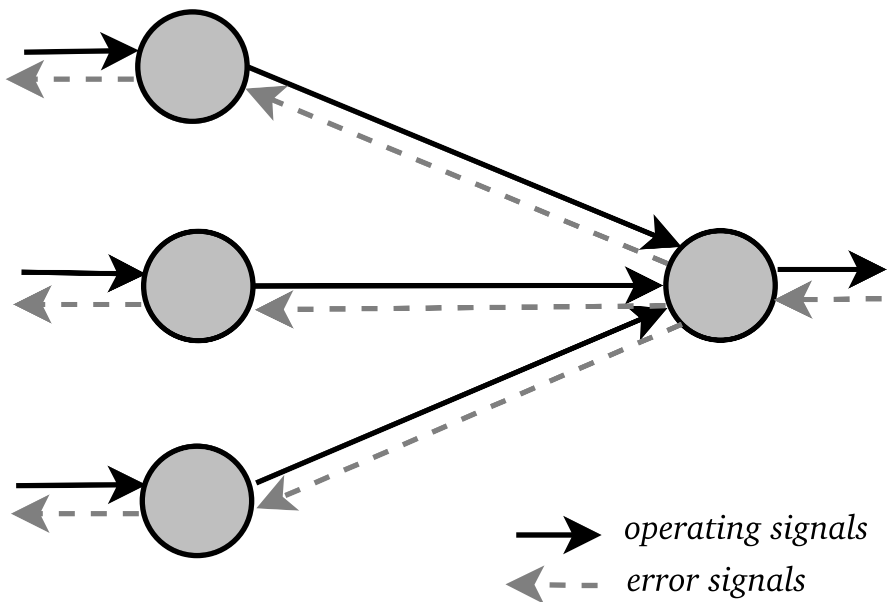


Figure 25: The flow directions of the two basic signals in a multi-layer perceptron: forward propagation of operating signals and backwards propagation of error signals

The hidden neurons act as features detectors and their contribution is critical for the operation of a multi-layer perceptron. While the learning procedure develops, the hidden neurons start gradually to discover the outgoing features. These features are the ones that describe the training data. In order to do this, the hidden neurons perform a non-linear transformation on the input data, in the *feature space*. In the feature space, the classes of the features, that a classification task is interested in, are able to be separated from everything else existing in the initial input data space. The formation of this features space is the difference between the multi-layer perceptron and Rosenblatt's perceptron.

2.9.2 Batch and Online Learning

Let's consider a multi-layer perceptron, with an input layer, one or more hidden layers, and an output layer consisted of one or more neurons as the one of fig. 24. The training sample. which is used for the training of the network in a supervised way, is symbolized by

$$T=\{\boldsymbol{x}(n),\boldsymbol{d}(n)\}_{n=1}^{N} \tag{30}$$

The operating signal that is produced at the output of neuron *j* by the stimulation **x**(n) is symbolized by yj(n).

The error signal is symbolized by

$$e_j(n)=d_j(n)-y_j(n) \tag{31}$$

where *d_j(n)* is the *i*-th item of the desirable responses vector ***d(n)***. The instantaneous error energy (cost function) of neuron *j* is defined as:

$$\mathscr{E}(n)=\frac{1}{2}e_j^2(n) \tag{32}$$

If we sum the energy-error distributions of all neurons of the output layer, we can express the total instantaneous error energy of the total network as:

$$\mathscr{E}(n)=\sum_{j\in C}\mathscr{E}(n)=\frac{1}{2}\sum_{j\in C}e_j^2(n) \tag{33}$$

where *C* contains all the output layer neurons. The training sample is consisted of *N* examples. The average error energy for all the training sample, which is also known as experienced risk, is defined as:

$$\mathscr{E}_{av}(N)=\frac{1}{N}\sum_{n=1}^{N}\mathscr{E}(n)=\frac{1}{2\mathrm{N}}\sum_{n=1}^{N}\sum_{j\in C}e_j^2(n) \tag{34}$$

Apparently, the instantaneous error energy as well as the average error energy are both functions of all the adaptable synaptic weights (which are the free parameters) of the multi-layer perceptron. This functional dependency has not been included in formulas (33) and (34) in order to have a simpler terminology.

Depending on the way of the supervised learning of a multi-layer perceptron, we can recognize two different methods: the batch learning and the on-line learning [14].

2.9.2.1 Batch Learning

In this learning method, the fitting of the multi-layer perceptron's synaptic weights takes place after the presentation of the set of N examples of the training set $\mathcal{T}$, which consist a training epoch. In other words, the cost function for batch learning is defined by an average error energy, $\mathcal{E}_{av}$. The fitting of the multi-layer perceptron's synaptic weights takes place *epoch by epoch*. As a consequence, we can produce a learning curve implementation, presenting $\mathcal{E}_{av}$ by the number of epochs, where, for each epoch, the examples of the training sample $\mathcal{T}$ are presented in a random order. The learning curve is produced by computing statistical medians for a set of a sufficiently great number of such implementations, where each implementation is executed for a different set of initial, randomly chosen, conditions.

With the method of gradient descent, that is used for the training, batch learning shows the following advantages:

- precise estimation of the gradient vector (which is the derivative of the cost function $\mathcal{E}_{av}$ to the weights vector $\boldsymbol{w}$, which guarantees, under simple conditions, the convergence of the steepest descent method to a local minimum.
- parallel implementation of the learning procedure.

However, the significant drawback of batch learning is its large storing requirements.

In a statistical framework, batch learning can be treated as a form of statistical implication. For this reason, it is appropriate for solving problems of non-linear regression.

2.9.2.2 Online Learning

In this learning method, the fitting of the multi-layer perceptron's synaptic weights is taking place example by example. The cost function that has to be minimized is the total instantaneous error energy $\mathcal{E}(n)$.

Let's consider a given epoch that contains N training examples, organized in the following order: $\{\mathbf{x}(1), \mathbf{d}(1)\}, \{\mathbf{x}(2), \mathbf{d}(2)\}, ..., \{\mathbf{x}(N), \mathbf{d}(N)\}$. The first example pair $\{\mathbf{x}(1), \mathbf{d}(1)\}$ of this epoch is presented to the network and the weights' fitting takes place using the gradient descent method. Afterwards, the second example $\{\mathbf{x}(2), \mathbf{d}(2)\}$ is presented to the network and leads to further fitting on the synaptic weights. This procedure continues until the last example, $\{\mathbf{x}(N), \mathbf{d}(N)\}$. Unfortunately, this kind of procedure doesn't allow a parallel implementation.

For a given set of initial conditions, we can take a single implementation of the learning curve, presenting the final value $\mathcal{E}(N)$ by the number of epochs that were used during the training session, where the examples arrive in a random order at the network's input after every epoch. The learning curve for on-line learning is made by computing statistical medians for a set of such implementations for a sufficiently great number of initial, randomly chosen, conditions. It's obvious the fact that, for a given network structure, the learning curve achieved by on-line learning is much different from the one achieved by batch learning.

Given the fact that the training examples are presented to the network in a random way, the use of on-line learning makes the search in multidimensional weights space natively stochastic; this is why sometimes on-line learning is also mentioned as a stochastic method. This helps to decrease the possibility of the learning procedure being trapped in a local minimum, which consists a certain advantage of on-line learning in comparison to batch learning. Another advantage is that on-line learning requires much less storing space than batch learning does. On-line learning has the additional useful property of observing small changes of the training data, especially when the environment that is responsible for the production of these data is non-static.

To sum up, on-line learning is extremely popular for the solution of pattern classification tasks, as it can be implemented very easily and can provide efficient solutions to classification problems of large scale and increased difficulty.

2.9.3 Back Propagation

2.9.3.1 The Algorithm

The Back Propagation (BK) algorithm was first proposed by Paul Werbos in the 1970's. However, it became widely used when it was rediscovered in 1986 by Rumelhart and McClelland. BK algorithm has helped to improve even more the performance of on-line learning for the supervised training of a multi-layer perceptron [14].

Figure 26 presents an output neuron *j*, being fed by a set of operating signals produced at a neurons' layer on the left. The local field of $\upsilon_j(n)$, produced at the input of the activation function regarding neuron *j*, is described by the equation:

$$\upsilon_j(n)=\sum_{i=0}^{m} w_{ji}(n)\, y_i(n) \tag{35}$$

where m is the total number of inputs (apart from the bias) applied to neuron j. The synaptic weight w_{j0} (corresponding to the constant input $y_0=+1$) equals to the bias b_j applied to neuron *j*. Hence, the operating signal $y_j(n)$, appearing in the output of neuron *j* in the *n*-th iteration is:

$$y_j(n)=\varphi_j(\upsilon_j(n)) \tag{36}$$

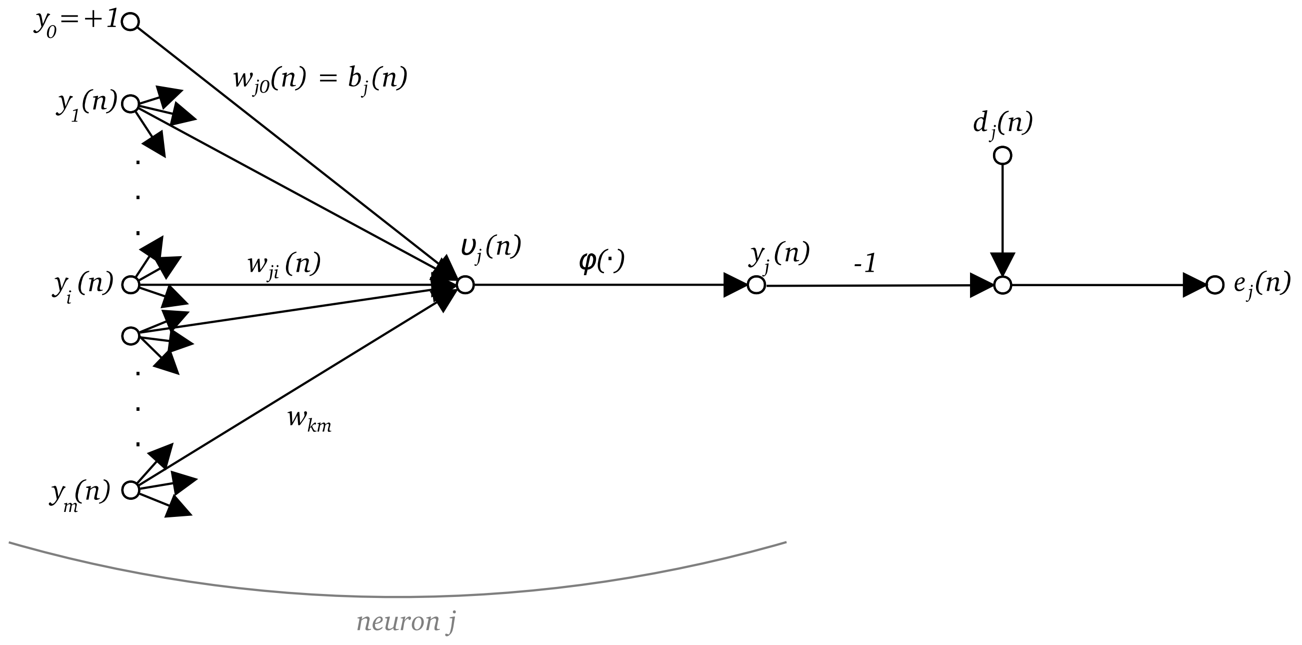


Figure 26: Signal flowchart describing the details of output neuron j.

BK algorithm applies a correction *$\Delta w_{ji}(n)$* to the synaptic weight *$w_{ji}(n)$*, which depends on the partial derivative *$\partial\mathscr{E}(n)/\partial w_{ji}(n)$*. According the chain rule of calculus, this gradient can be expressed as:

$$\frac{\partial\mathscr{E}(n)}{\partial w_{ji}(n)}=\frac{\partial\mathscr{E}(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial \upsilon_j(n)}\frac{\partial \upsilon_j(n)}{\partial w_{ji}(n)} \tag{37}$$

The partial derivative *$\partial\mathscr{E}(n)/\partial w_{ji}(n)$* represents a sensitivity factor, which determines the search direction in the weights space for the synaptic weight *w_{ji}*.

By differentiating the equations (31) and (33) we receive:

$$\frac{\partial\mathscr{E}(n)}{\partial e_j(n)}=e_j(n) \tag{38}$$

$$\frac{\partial e_j(n)}{\partial y_j(n)}=-1 \tag{39}$$

And then, after differentiating the (35) and (36),

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'_j(v_j(n)) \quad (40)$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_j(n) \quad (41)$$

Combining the equations (38) and (40), we get:

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = e_j(n)\varphi'_j(v_j(n))y_i(n) \quad (42)$$

The correction *$\Delta w_{ji}(n)$* applied to *$w_{ji}(n)$* is defined by the delta rule. The delta rule is a gradient descent learning rule for updating the weights of the inputs to neurons in single-layer neural network. For neuron *j*, the delta rule is given by:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} \quad (43)$$

where *η* is the learning rate parameter of BK algorithm. The negative sign refers to the gradient descent in the weights space. It is:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \quad (44)$$

where the local gradient *$\delta_j(n)$* is defined by

$$\delta_j(n) = \frac{\partial \mathscr{E}(n)}{\partial v_j(n)} = \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n)) \quad (45)$$

As seen in the above equations, the error signal *$e_j(n)$* at the output of neuron *j* is an important factor for the calculation of the weight fitting *$\Delta w_{ji}(n)$*. We can recognize two separate cases, depending on the place where neuron *j* is located in the network:

1. Neuron *j* is an output node: In this case, neuron *j* is fed by its own desirable response. The equation (31) can be used for the calculation of the error signal *$e_j(n)$* corresponding to this neuron (fig. 26).

Then, it is a simple assumption to calculate the local gradient $\delta_j(n)$, using the equation (45).

2. Neuron *j* is a hidden node: In this case, there is no predefined desirable response for *j*. Consequently, the error signal for a hidden neuron should be defined recursively, working backwards, based on the error signals of all the neurons *j* is directly connected. In fig. 27, neuron *j* is a node of a hidden layer of the network. According to the equation (45) the local gradient of neuron *j*, $\delta_j(n)$, can now be written as:

$$\delta_j(n) = -\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \upsilon_j(n)} = -\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \varphi'_j(\upsilon_j(n)) \tag{46}$$

In fig. 27, it is also obvious that

$$\mathscr{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n) \tag{47}$$

where neuron *k* is an output node.

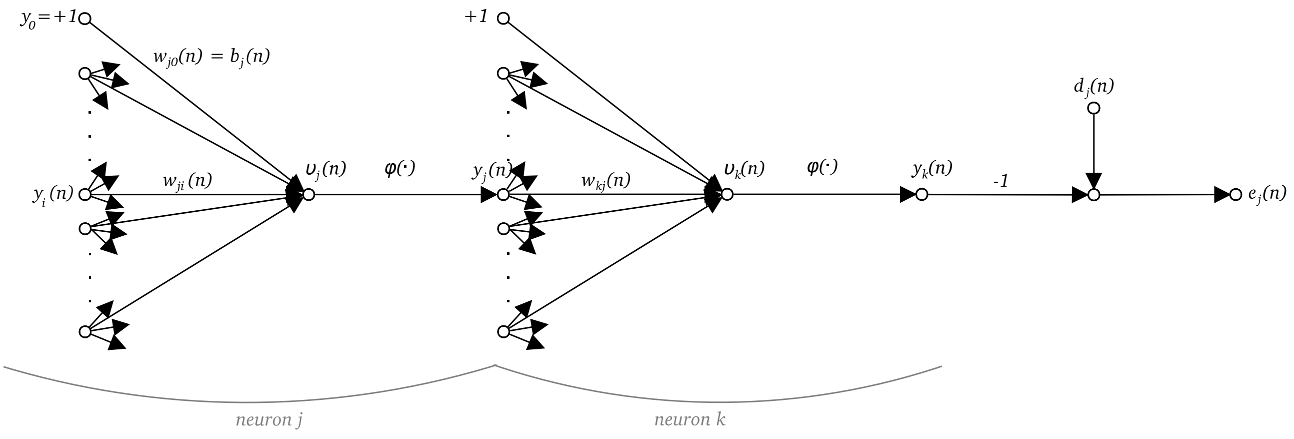


Figure 27: Signal flowchart describing the details of output neuron k, which is connected to hidden neuron j .

By differentiating to the operating signal $y_j(n)$, it is:

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} \tag{48}$$

Using the chain rule for the partial derivative *∂e$_k$(n)/∂y$_j$(n)*, formula (48) is equivalent to

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \tag{49}$$

However, in fig. 27, it is clear that

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n)) \tag{50}$$

and hence,

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k(v_k(n)) \tag{51}$$

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) \tag{52}$$

where *m* is the total number of inputs (apart from the bias) applied to neuron *k*. The synaptic weight *w$_0$(0)* is equal to the bias *b$_k$(n)* applied to neuron *k* and the corresponding input is equal to +1 constantly. After differentiating the equation (52), we get:

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \tag{53}$$

And then, using the formulas (49), (51) and (53), we can write:

$$\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi'_k(v_k(n)) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n) \tag{54}$$

Finally, using the equations (46) and (54), we get the back propagation formula for the local gradient *δ$_j$(n)*, which is described by:

$$\delta_j(n)=\varphi'_j(v_j(n))\sum_k \delta_k(n)w_{kj}(n) \tag{55}$$

where neuron *j* is hidden.

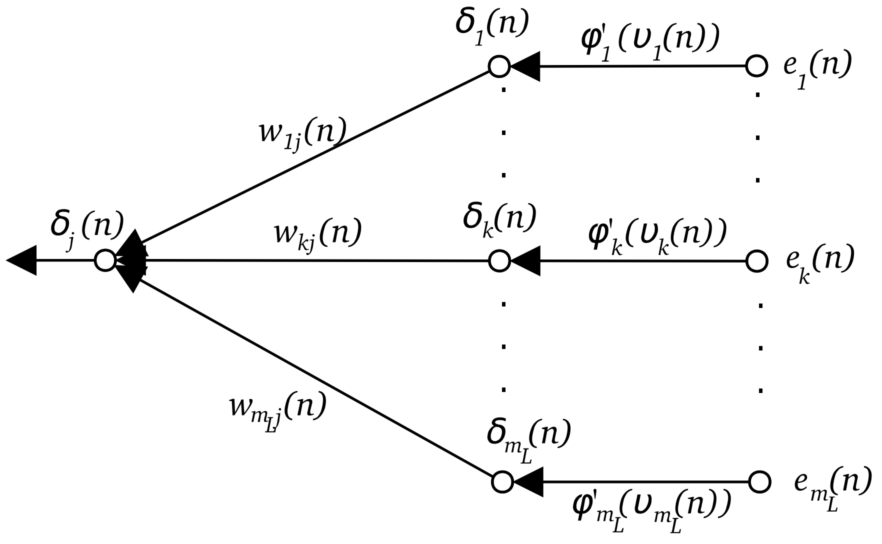


Figure 28: Signal flowchart of part of the conjugate system that executes the back propagation of error signals.

Fig. 28 represents the equation (55) as a signal flowchart, assuming that the output layer is consisted of m_L neurons.

For the calculation of the local gradient $\delta_j(n)$, two factors are used. The first one is the external factor $\varphi'_j(\upsilon_j(n))$ is dependent exclusively on the activation function that corresponds to neuron *j*. The second factor is the sum for all *k* and depends on two sets: the first terms set, $\delta_k(n)$, requires knowledge for the error signals $e_k(n)$ for all the neurons of the next (to the right) layer of the hidden neuron *j* and connected directly to it; the second terms set, $w_{kj}(n)$, is consisted of all the synaptic weights regarding these connections.

Let's now summarize the formulas about BK algorithm that were described before. Firstly, the correction $\Delta w_{ji}(n)$ that is applied to the synaptic weight connecting neuron i to neuron j, is defined by the delta rule:

$$\Delta w_{ji}(n)=\eta\times\delta_j(n)\times y_j(n) \tag{56}$$

Secondly, the local gradient $\delta_j(n)$ depends on whether the neuron *j* is an output or a hidden node, as described analytically before.

2.9.3.2 Phases of BK Algorithm

When BK algorithm takes place, there are two different phases of computations; the feed-forward phase and the feed-backward phase. We need to underline here that the input vector remains the same during both phases [14].

- **Feed-Forward Phase**

In this phase, the synaptic weights remain invariable in all network and the operating signals of the network are calculated neuron by neuron. The operating signal at the output of neuron *j* is calculated by

$$y_j(n)=\varphi(\upsilon_j(n)) \tag{57}$$

where $\upsilon_j(n)$ is the neuron's *j* local field, defined as:

$$\upsilon_j(n)=\sum_{i=0}^{m} w_{ji}(n)\, y_i(n) \tag{58}$$

where *m* is the total number of inputs (apart from the bias) applied to neuron *j*; $w_{ji}(n)$ is the synaptic weight connecting neuron *i* with neuron *j*; $y_j(n)$ is the input signal for neuron *j*, or, equivalently, the operating signal appearing in the output of neuron *i*. If neuron *j* is in the first hidden layer of the network, then $m=m_0$ and *i* refers to the *i*-th terminal input node of the network, for which it is:

$$y_i(n)=x_i(n) \tag{59}$$

where $x_i(n)$ is the *i*-th element of the input vector (pattern). On the other side, if neuron *j* is in the network's output layer, then $m = m_L$ and the indicator *j* refers to the *j*-th terminal output node of the network, for which it is

$$y_j(n)=o_j(n) \tag{60}$$

where $o_j(n)$ is the *j*-th element of the output vector of the multi-layer perceptron. This output is compared with the desirable response $d_j(n)$ and then we get the error signal $e_j(n)$ for the *j*-th output neuron.

Thus, the feed-forward phase starts at the first hidden layer, presenting to this layer the inputs vector and terminates at the output layer, computing the error signal for every neuron of this layer.

- **Feed-Backward Phase**

In this phase, the computations pass backwards. It starts from the output layer, sending error signals to the left, in all layers of the network, layer by layer, and by computing δ recursively. This recursive procedure allows the synaptic weights of the network to be changed according delta rule. For a neuron of the output layer, δ is simply equal to the error signal of this neuron, multiplied by the first derivative of its non-linearity. Consequently, we use the equation (56) in order to calculate the changes in the weights of all connections that feed the output layer. Given the δ for the output layer neurons, we use the equation (55) in order to compute the δ for the neurons of the semifinal layer, and, consequently, the changes of the weights of all connections feeding it. This recursive computation continues, layer by layer, propagating the changes of all the synaptic weights in the network.

2.9.3.3 Activation Function

As described before in section 2.2.2, a neuron's output is defined by the activation function, $\varphi(\upsilon)$, where υ is the symbol of the local field. For the calculation of δ for each neuron of a multi-layer perceptron, it is required to know the derivative of its activation function. This means that $\varphi(\cdot)$ has to be differentiable. A constantly differentiable non-linear activation function that is widely used in multi-layer perceptrons is a function with sigmoid non-linearity. Two forms of this functions are described here [14].

- **Logistic Function**

This form of sigmoid non-linearity is defined, in its generic form, as:

$$\varphi_j(\upsilon_j(n))=\frac{1}{1+\exp(-\alpha\upsilon_j(n))},\ \alpha>0 \tag{61}$$

where $\upsilon_j(n)$ is the local field of neuron j and α is a fitting positive parameter. According to this non-linearity, the output's amplitude is within the range $0\leq y_j\leq 1$. After differentiating (61), we get:

$$\varphi'_j(v_j(n))=\frac{\alpha\exp(-\alpha v_j(n))}{[1+\exp(-\alpha v_j(n))]^2} \tag{62}$$

With $y_j(n)=\varphi_j(v_j(n))$, we can express $\varphi'_j(v_j(n))$ as

$$\varphi'_j(v_j(n))=\alpha y_j(n)[1-y_j(n)] \tag{63}$$

For a neuron j of the output layer, it is $y_j(n)=o_j(n)$, where $o_j(n)$ is the operating signal at the output of neuron j. Hence, the local gradient for neuron j is

$$\delta_j(n)=e_j(n)\varphi'_j(v_j(n))=\alpha[d_j(n)-o_j(n)]o_j(n)[1-o_j(n)] \tag{64}$$

where $d_j(n)$ is the desirable response for neuron j.

For a hidden neuron j, the local gradient can be expressed as

$$\delta_j(n)=\varphi'_j(v_j(n))\sum_k \delta_k(n)w_{kj}(n)=\alpha y_j(n)[1-y_j(n)]\sum_k \delta_k(n)w_{kj}(n) \tag{65}$$

- **Hyperbolic Tangent Function**

  Hyperbolic Tangent Function is another widely used form of sigmoid non-linearity. In its generic form, it is defined as

$$\varphi_j(v_j(n))=\alpha\tanh(bv_j(n)) \tag{66}$$

where α and b are positive constants. In fact, the hyperbolic tangent function is a scaled version of the logistic function where bias has been applied.

$$\varphi_j(v_j(n))=\alpha b\,sech^2(bv_j(n))=\alpha b(1-\tanh^2(bv_j(n)))=\frac{b}{\alpha}[\alpha-y_j(n)][\alpha+y_j(n)] \tag{67}$$

For a neuron j of the output layer, the local gradient is:

$$\delta_j(n)=e_j(n)\varphi'_j(v_j(n))=\frac{b}{\alpha}[d_j(n)-o_j(n)][\alpha-o_j(n)][\alpha+o_j(n)] \tag{68}$$

For a hidden neuron *j*, the local gradient is:

$$\delta_j(n)=\varphi'_j(v_j(n))\sum_k \delta_k(n)w_{kj}(n)=\frac{b}{\alpha}[\alpha-y_j(n)][\alpha+y_j(n)]\sum_k \delta_k(n)w_{kj}(n) \tag{69}$$

2.9.3.4 Learning Rate

BK algorithm is actually an approach on the weights space, as computed by the method of steepest descent. The smallest the learning parameter η is, the smallest the changes in the synaptic weights of the network from the one iteration to the next are. Nevertheless, there is a trade-off on the learning rate, which is decreased by this improvement. On the other hand, if we increase extremely the learning rate parameter in order to accelerate a learning rate, the occurring (big) changes in the synaptic weights take such a form that the network may become unstable and start to oscillate.

A simple method to increase the learning rate with concurrent avoiding of the instability danger is to modify the delta rule, by including a momentum term:

$$\Delta w_{ji}(n)=\alpha\,\Delta w_{ji}(n-1)+\eta\delta_j(n)y_i(n) \tag{70}$$

where α is the momentum constant and usually is a positive number. This constant controls the feedback loop around $\Delta w_{ji}(n)$, as presented in fig. 29, where z^{-1} is the unit delay operator. The equation (70) is called generalized delta rule.

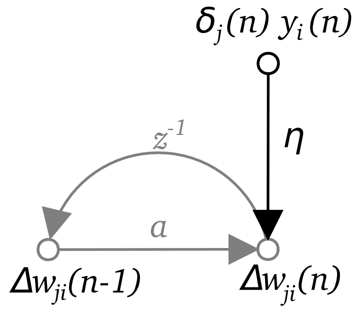


Figure 29: Signal flowchart presenting the effect of momentum constant α (in the feedback loop).

The equation (70) can be written as an time sequence with *t* as indicator. The indicator *t* starts from the initial moment 0 until the current moment *n*. Equation (70) can be seen as an equation of differences of first order for the weights correction $\Delta w_{ji}(n)$.

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t)\, y_i(t) \tag{71}$$

Equation (71) represents a time sequence with (*n*+1) length and is equivalent to:

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial \mathscr{E}(t)}{\partial w_{ji}(t)} \tag{72}$$

Based on the equation (72) we observe that:

- The current fitting $\Delta w_{ji}(n)$ represents the sum of an exponentially weighted time series. In order this time series to converge, the momentum constant has to be limited in the range $0 \leq |\alpha| < 1$. When α equals to 0, the BK algorithms functions without momentum. Besides, α can be positive or negative, even though it is rather impossible to be negative in practice.
- When the partial derivative $\partial\mathscr{E}(t)/\partial w_{ji}(t)$ has the same algebraic sign in following iterations, the exponentially weighted sum $\Delta w_{ji}(n)$ is increased in value, and consequently the weight $w_{ji}(n)$ is fitted at a great quantity. The inclusion of the momentum in the BK algorithm tends to accelerate the descent.
- When the partial derivative $\partial\mathscr{E}(t)/\partial w_{ji}(t)$ has a positive sign in successive iterations, the exponentially weighted sum $\Delta w_{ji}(n)$ is decreased in value, and consequently a small quantity of the weight $w_{ji}(n)$

is fitted. The inclusion of momentum in the BK algorithm has a stabilizing affect in the directions of which the sign oscillates.

The embedding of momentum in BK algorithm represents a modification of small significance in the weights update; however, it may have specific beneficial affects in the learning behavior of the algorithm. The momentum term may also have the advantage that it prevents the learning procedure from finishing in a shallow local minimum of the error surface.

We have assumed that the learning rate parameter is a constant symbolized by η. However, in fact it should be defined as η_{ji}; this means that it should depend on the connection between neurons i and j.

It is also important to mention here that in the application of the BK algorithm, we are able to decide whether all the synaptic weights of the network are adaptable or a certain set of synaptic weights remains constant during the fitting procedure. In the second case, the errors propagate backwards through the network in the usual way; however the constant synaptic weights remain invariable. This is feasible by setting the learning rate parameter η_{ji} for the synaptic weight w_{ji} equal to 0 [14].

2.9.3.5 Termination Criteria

In general, it cannot be proved that BK algorithm converges and there are no well-defined criteria for its termination. Instead, there are some logical criteria, each one of them having its own practical value, which could be used for the termination of the weights fitting procedure. In order to define such a criterion, it is reasonable to use the unique properties of a local or global minimum of the error surface. Let the weights vector $\boldsymbol{w}^*$ symbolize a minimum, local or global. A necessary condition for the $\boldsymbol{w}^*$ to be a minimum is the gradient vector $\boldsymbol{g(w)}$ of the error surface depending on the weights vector $\boldsymbol{w}$, to be equal to 0 at $\boldsymbol{w} = \boldsymbol{w}^*$. Consequently, a convergence criterion for the learning rate with error back propagation is the one of Kramer and Sangiovanni-Vincentelli (1989) :

BK algorithm is considered to converge when the Euclid norm of the gradient vector reaches a sufficiently small gradient threshold.

However, this criterion may have a long learning time for successful testings and it requires the calculation of $\boldsymbol{g(w)}$.

Another unique property of a minimum is the fact that the cost function $\mathscr{E}_{av}(\boldsymbol{w})$ is static at $\boldsymbol{w} = \boldsymbol{w}^*$. Then we could suggest the following convergence criterion:

BK algorithm is considered to converge when the absolute learning rate of the MSE per period is sufficiently small.

Typically, the rate of change of the MSE is considered sufficiently small if it is within the range from 0.1 to 1% per epoch. Sometimes, much smaller values, e.g. 0.01% per epoch, are used. Unfortunately, this criterion may have as a result the early termination of the learning procedure.

Another useful and theoretically supported criterion is to control the network for its performance about the achievable generalization after each iteration of the learning procedure. The learning procedure is finished when the performance about generalization is judged as sufficient or when it is obvious that its performance about generalization has come to a maximum [14].

2.9.3.6 Summary

Fig. 24 presents the architectural topology of a multi-layer perceptron. The signal flowchart of fig. 30 presents the learning procedure with error backpropagation, which embeds both phases (feed-forward and feed-backward) of the involving computations, for the case where $L=2,\ m_0=m_1=m_2=3$. The top of the flowchart refers to the feed-forward phase, while the bottom refers to the error backpropagation and is referred as a flowchart of sensitivity for the computation of the local gradients of BK algorithm (Narendra and Parthasarathy, 1990) [14].

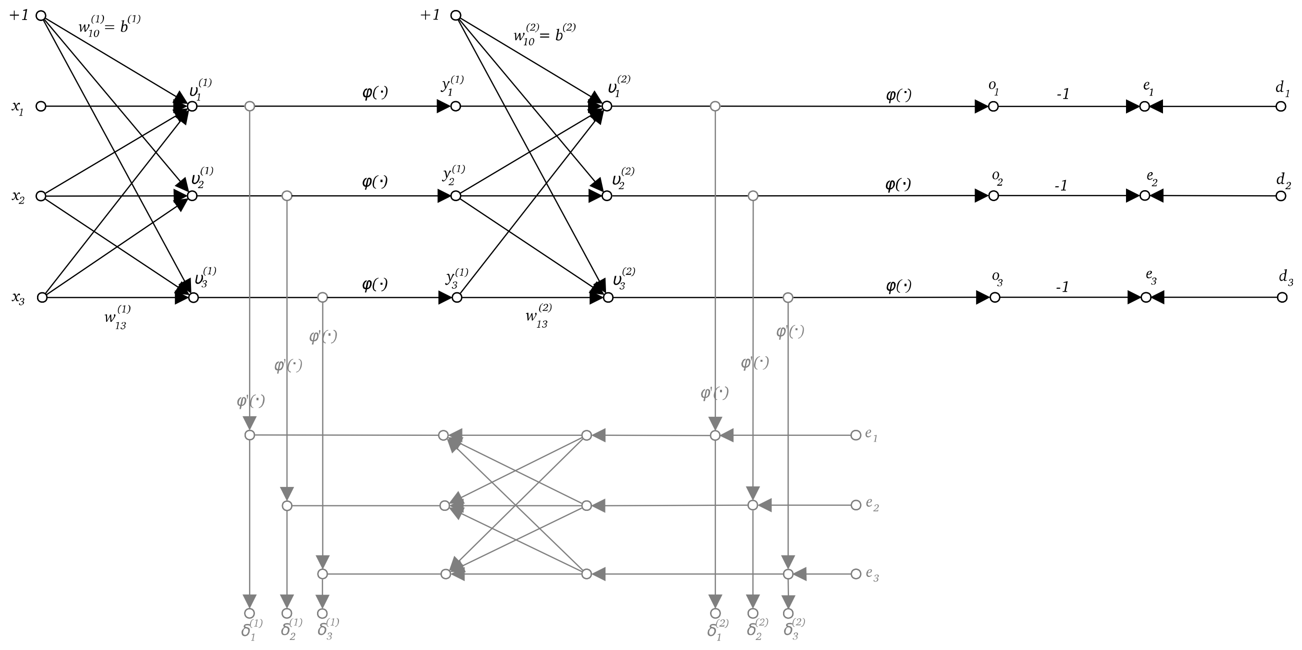


Figure 30: Summary plot of the learning procedure with error back propagation. Top: the feedforward phase. Bottom: the error back propagation phase.

For the implementation of BK algorithm, it is considered preferable to update successively the synaptic weights. For this state of operation, BK algorithm runs the training sample in the following steps [14]:

1. Initialization: Assuming that no past information is available, choose the synaptic weights and the thresholds from a uniform distribution, of which the median is 0 and the variance is chosen in such a way that it sets the standard deviation of neurons' local fields within the limit between the linear and the constant part of the sigmoid activation function.

2. Presentations of training examples: Present an epoch of training examples to the network. For each example included in the sample, execute the (forward and backwards) computations sequence that are described in steps 3 and 4 (respectively).

3. Forward Computations: Let a training example in the current epoch be symbolized by ***(x(n), d(n))***. The inputs vector ***x(n)*** is applied to the inputs layer of the sensory nodes and the desirable responses vector ***d(n)*** is presented to the output layer of the computational nodes. Calculate the local fields and the operating signals of the network moving on forward, in all the extent of the network, layer by layer. The local field $\upsilon_j^{(l)}(n)$ for neuron *j* of layer *l* is:

$$v_j^{(l)}(n)=\sum_i w_{ji}^{(l)}(n)\, y_i^{(l-1)}(n) \tag{73}$$

where $y_i^{(l-1)}(n)$ is the output signal (operating signal) of neuron i of the previous layer l-1, in the n-th iteration, and $w_{ji}^{(l)}(n)$ is the synaptic weight of neuron j of layer l, fed by neuron i of layer l-1. For $i = 0$, it is $y_0^{(l-1)}(n) = +1$ and $w_{j0}^{(l)}(n) = b_j^{(l)}(n)$ is the bias applied to neuron j of level l. Assuming that a sigmoid function is used, the output signal of neuron j of level l is

$$y_j^{(l)}=\varphi_j(v_j(n))$$

If neuron j belongs to the first hidden layer, set

$$y_j^{(0)}=x_j(n)$$

where $x_j(n)$ is the j-th element of the input vector $x(n)$.

If neuron j belongs to the output layer (and then $l = L$, where L indicates the network's depth), set

$$y_j^{(L)}=o_j(n)$$

Compute the error signal as

$$e_j(n)=d_j(n)-o_j(n) \tag{74}$$

where $d_j(n)$ is the j-th element of the desirable responses vector $\boldsymbol{d(n)}$.

4. <u>Backwards computations</u>: Compute the local gradients δ of the network, which are defined as:

$$\delta_j^{(l)}(n)=\begin{cases} e_j^{(L)}\varphi'_j(v_j^{(L)}(n)), & \textit{for neuron } j \in \textit{output layer } L \\ \varphi'_j(v_j^{(l)}(n))\sum_k \delta_k^{(l+1)}(n)\, w_{kj}^{(l+1)}(n), & \textit{for neuron } j \in \textit{hidden layer } l \end{cases} \tag{75}$$

Fit the network's synaptic weights of layer l according the generalized delta rule:

$$w_{ji}^{(l)}(n+1)=w_{ji}^{(l)}(n)+\alpha[w_{ji}^{(l)}(n-1)]+\eta\delta_j^{(l)}(n)\,y_i^{(l-1)}(n) \tag{76}$$

where η is the learning rate parameter and α is a constant of momentum.

5. Repetition: Repeat the computations described in steps 3 and 4, presenting new epochs of training examples to the network, until the selected termination criterion is satisfied.

2.9.4 Cross-Validation

2.9.4.1 Basic Method

The substance of learning with back propagation of the error signal is to encode an input-output mapping (represented by a set of annotated samples) to the synaptic weights and thresholds of a multi-layer perceptron. Hence, we expect that the network will end to be well trained, in order to be able to generalize and make decisions for the future, after having learned a lot from the past experience. Seen from this angle, a learning procedure is equivalent to the choice of a network configuration for a dataset. Specifically, we can see the problem of choosing a network as the problem of choosing the "optimal" configuration according to a particular criterion. A standard tool of statistics for this work is the cross-validation method, which provides an attractive directive principle (Stone 1974, 1978) [14].

In the first stage of cross-validation method, the available dataset is partitioned randomly in a training sample and a control set. The training sample is then partitioned in two disjoint subsets: an evaluation subset and a validation subset. The evaluation subset is used for the choice of the model. The validation subset is used for the control (validation) of the model.

The idea of the method of cross-validation is the validation of the model to take place based on a different dataset from the one that was used for the evaluation of the parameters. In this way, the training sample can be used for the validation of the performance of several candidate models, hence the optimal among them can be chosen. However, it's very possible the model chosen in this way to end up to being overfitted. In order to avoid this case, the performance of generalization of the chosen model is evaluated by a control set which is different from the validation subset. The use of this validation method is attractive especially if we need to design a large neural network for reliable generalization.

In order to describe the way of thinking for the model choice with cross-validation, we assume an inlaid structure of categories of Boolean functions:

$$F_1 \subset F_2 \subset \ldots \subset F_n$$
$$F_k = \{\boldsymbol{F_k}\} = \{\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{w}); \boldsymbol{w} \in W_k\} \tag{77}$$
$$where\, k = 1, 2, \ldots, n$$

The *k*-th category of functions F_k encloses a family of multi-layer perceptrons with similar architecture and vectors of weights ***w***, pulled from a multidimensional space of weights, W_k. A member of this category, which is characterized by the function or hypothesis $F_k = F(\mathbf{x}, \mathbf{w})$, where $\boldsymbol{w} \in W_k$, maps the vector of inputs ***x*** to {0, 1}. The vector of inputs ***x*** is pulled from a space of inputs *H* with some unknown probability *P*. Every multi-layer perceptron of this structure is educated through the BK algorithm, which is responsible for the education of the parameters of the perceptron. The choice of a model is a problem of a choice of a certain multi-layer perceptron that gives the optimal value for w, the number of free parameters. Taking into account that the scalar desirable response for a vector of inputs ***x*** is d={0, 1}, we can define the generalization error as the following probability:

$$\boldsymbol{\varepsilon}_g(F) = P(F(\boldsymbol{x}) \neq d),\ for\ \boldsymbol{x} \in H \tag{78}$$

A training sample consisted of annotated examples is described by

$$T = \{(\boldsymbol{x}_i, d_i)\}_{i=1}^{N} \tag{79}$$

Our goal is to choose a particular hypothesis $F(\mathbf{x}, \mathbf{w})$ that minimizes the generalization error $\varepsilon_g(F)$, occurring when the network is fed by inputs from the control set.

We assume that the structure described by equation (77) has the following property: for every sample size *N*, there is always a multi-layer perceptron with a sufficiently great number of free parameters $W_{max}(N)$, which is called fitting number, such that the fitting of the training sample is sufficient. The fitting number is very significant, as a logical procedure for the choice of a model would choose a hypothesis $F(\mathbf{x}, \mathbf{w})$ which requires $W \leq W_{max}(N)$; otherwise, the complexity of the network would be increased.

Let *r* be an independent parameter, with value range from 0 to 1 specifies the partitioning of the training sample $\mathcal{T}$, between the evaluation subset $\mathcal{T}'$ and the validation subset $\mathcal{T}''$. The evaluation subset is used for the training of an inlaid sequence of multi-layer perceptrons, resulting to hypotheses $F_1, F_2, \ldots, F_n$ of increasing complexity. $\mathcal{T}'$ is consisted of $(1-r)N$ examples. We examine the values of W that are less than or equal to the corresponding fitting number $Wmax((1-r)N)$.

Using the cross-validation method, we end up to the choice:

$$F_{cv} = \min_{k=1,2,\dots,v} \{e''_t(F_k)\} \tag{80}$$

where v corresponds to $W_v < W_{max}((1-r)N)$, and $e_t''(F_k)$ is the classification error that is produced by the hypothesis F_k when is controlled on the validation subset $\mathcal{T}''$, which is consisted of rN examples.

The way to determine the parameter r is quite important. There are several quality properties of the optimal parameter r, as described in a study of Kearns (1996), such as the following:

- When the target function complexity, which actually defines the desirable response d depending on the vector of inputs $\boldsymbol{x}$, is relatively small compared to the sample size N, the performance of the cross-validation method, shows a relative "unconsciousness" on the choice of r.
- While the target function gets more complex depending on the sample size N, the choice of an optimal r has a bigger effect on the performance of cross-validation and the value of the same target function is decreased.
- A single constant value of r works almost optimally for a wide complexity range of the target function.

Based on the results of the above study (Kearns, 1996), a constant value of the parameter r, equal to 0.2, seems to be a reasonable choice, which means that 80% of the training sample $\mathcal{T}$ consists a partition for the evaluation subset, and the rest 20% consists a partition for the validation subset.

2.9.4.2 Early Stopping Method

A multi-layer perceptron trained by BK algorithm learns in stages: it starts with the implementation of simple mapping functions and continues with more complicated ones while its training session develops. In a typical case, the training session develops in many epochs, and while the number of epochs increases, the mean squared error (MSE) is decreased; usually, in the beginning of the training the MSE has a large value but after a while it decreases rapidly and then it continues decreasing but more smoothly when the network is reaching a local minimum on the error surface. When a good generalization is the target, it is quite difficult to identify when is the appropriate moment to stop the training of the network, taking into account only the learning curve. The network is likely to result in over-fitting if the training session doesn't stop on the correct moment.

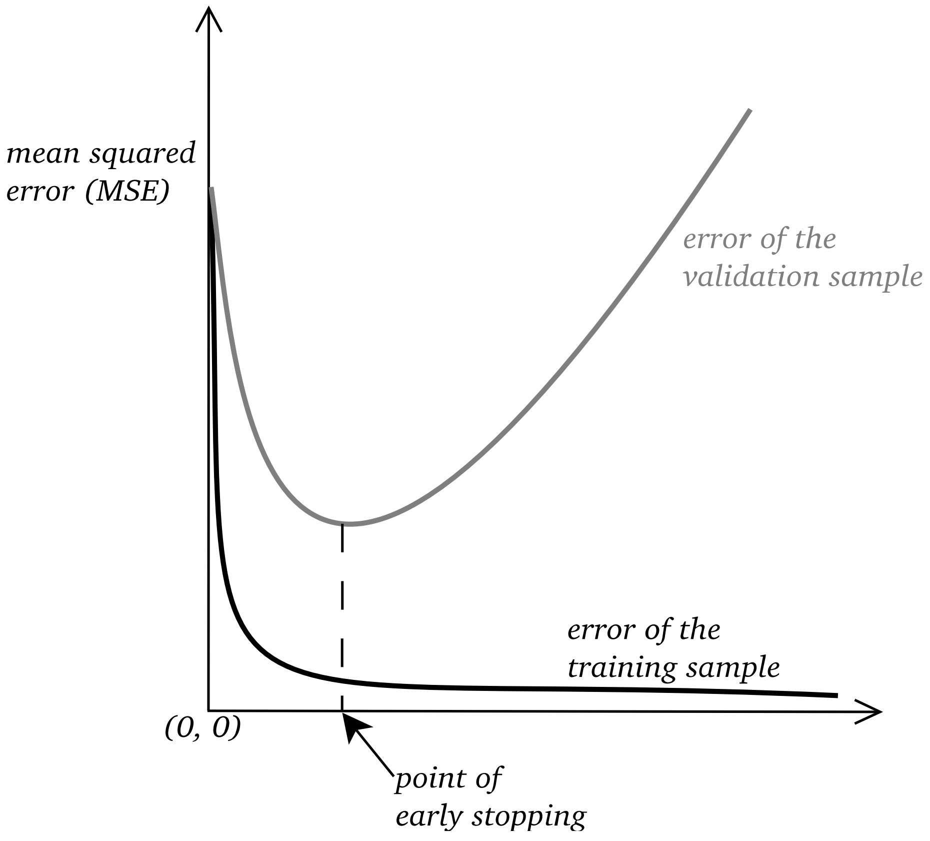


Figure 31: Plot of the early stopping rule based on cross-validation.

The point at which the overfitting starts can be recognized through the cross-validation method, with which, as described before, the training data is divided into an evaluation subset and a validation subset. The evaluation subset can be used for the network training with the usual way with a small difference. The training session is interrupted periodically every some epochs. After every training period, the network is controlled on the base of the validation subset. This procedure is called early stopping of the training and is widely used. It is performed in the following two phases:

- After an evaluation period (which is the training period), every some epochs, all the synaptic weights and the biases of the multi-layer perceptron are constant and the network functions in a forward state. The validation error is calculated for each example in the validation subset.
- When the above phase is complete, the evaluation (training) continues for a next period and this procedure is repeated.

Fig. 31 presents two ideal learning curves; the one corresponds to the evaluation subset and the other one corresponds to the validation subset. Typically, this model has a better performance in the evaluation subset (which was the base for the design of the curves) rather than the validation subset. The learning curve for the evaluation subset is decreased monotonically with the increment of the number of epochs. In opposition, the learning curve for the validation subset is decreased monotonically until a minimum and then it starts increasing while the training goes on. After the minimum point in the learning curve for the validation subset, the network only learns the noise contained in the training data. This heuristic direction suggests that the minimum point of the learning curve for the validation subset can be used as a logical criterion for the termination of the training session.

Nevertheless, in fact the error for the validation subset doesn't evolve over time so smoothly as the ideal curve of fig. 31. Instead, the error for the validation subset can reach some local minima before starting to increase with the increment of the number of epochs. In such cases, a termination criterion should be chosen in a systematic way. Prechelt (1998) conducted an empirical research and proved experimentally that in fact there is a trade-off on the training time compared to the network performance, regarding the generalization as target. Based on the experimental results of this work, on a set of 1,296 training sessions, 12 different problems and 24 different network architectures, it is found that, when there are two or more local minima, the choice of a slower termination criterion, which will finish the procedure later than the other criteria would do, allows the achievement of a small improvement in the generalization performance, of 4% on average, with a trade-off on the training time (which is quadruple on average).

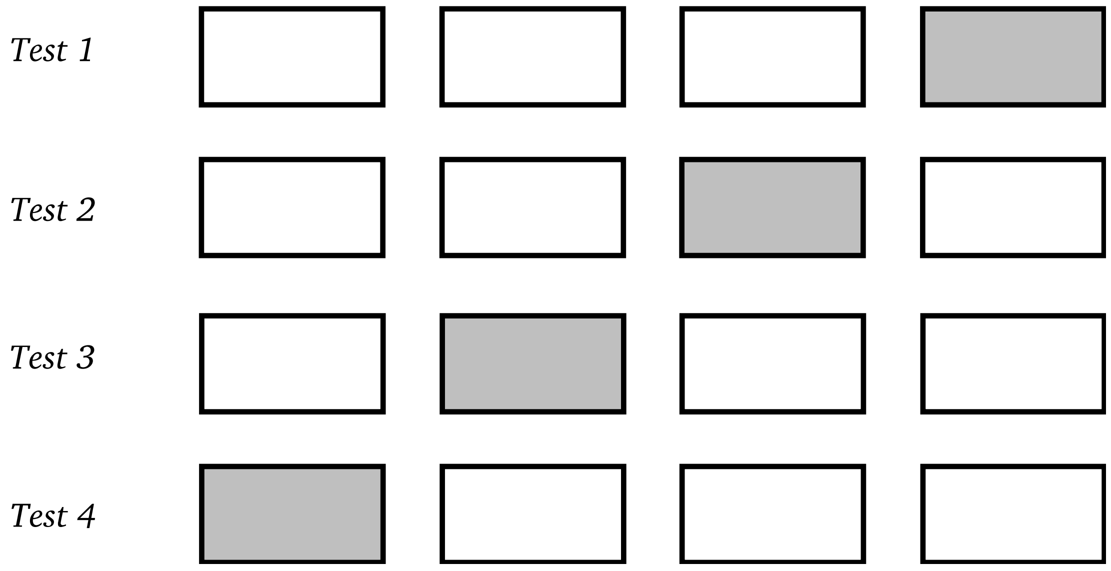


Figure 32: Plot of multiple cross-validation method. For a given test, the colored data subset is used for the model's validation. The model is trained by the rest of the data.

Apart from the procedure that was described above, there are variations of it that are applied in practice, mainly in cases of insufficient annotated examples. In such a case, we can use the multiple cross-validation method, by dividing the available set of *N* examples into *K* subsets, where *K > 1*. This procedure assumes that *K* is divisible by *N*. The model is trained with all subsets except from one, and the validation error is calculated by controlling the validation subset based on the specific subset with which the network didn't train before, during the training period. This procedure is repeated for a set of *K* tests and each time it uses a different subset for the validation, as presented in fig. 32 for $K = 4$. The performance of this model is evaluated based on the MSE of the validation over the the testing set. The drawback of the multiple cross-validation is that it might require an excessive volume of computations, as the model needs to be trained K times, and $1<K\leq N$.

If the available number of annotated examples *N* is drastically reduced, an extreme form of multiple cross-validation can be used. This method is known as leave-one-out method. In this case, *N*-1 examples are used for the training of the model and the validation is taking place based on the example that was left out. The experiment is repeated *N* times in total and every time a different example is left out and left for the validation phase. Afterwards, the validation MSE is calculated over the testing set, consisted of *N* tests [14].

3 *Model's Implementation*

Our model implements a supervised on-line learning system for Twitter sentiment analysis. It consists of the following stages:

- Data acquisition: collection of raw tweets, construction and annotation of the datasets.
- Data pre-processing: a series of steps that aim to reduce as much as possible the noise of our datasets and maintain the meaningful information.
- Feature Extraction: a series of steps for the extraction of the most important features we considered in our model, out of the pre-processed tweets, and the construction of the feature vectors.
- Multi-layer Perceptron: the feature vectors feed the inputs of an artificial neural network trained by the BK algorithm, which aims to classify correctly its inputs into two or three classes ({positive, negative} or {positive, negative, neutral}).

Each of these stages is going to be described in depth in this chapter. A schematic representation of the implemented model can be seen in the following figure (fig. 33).

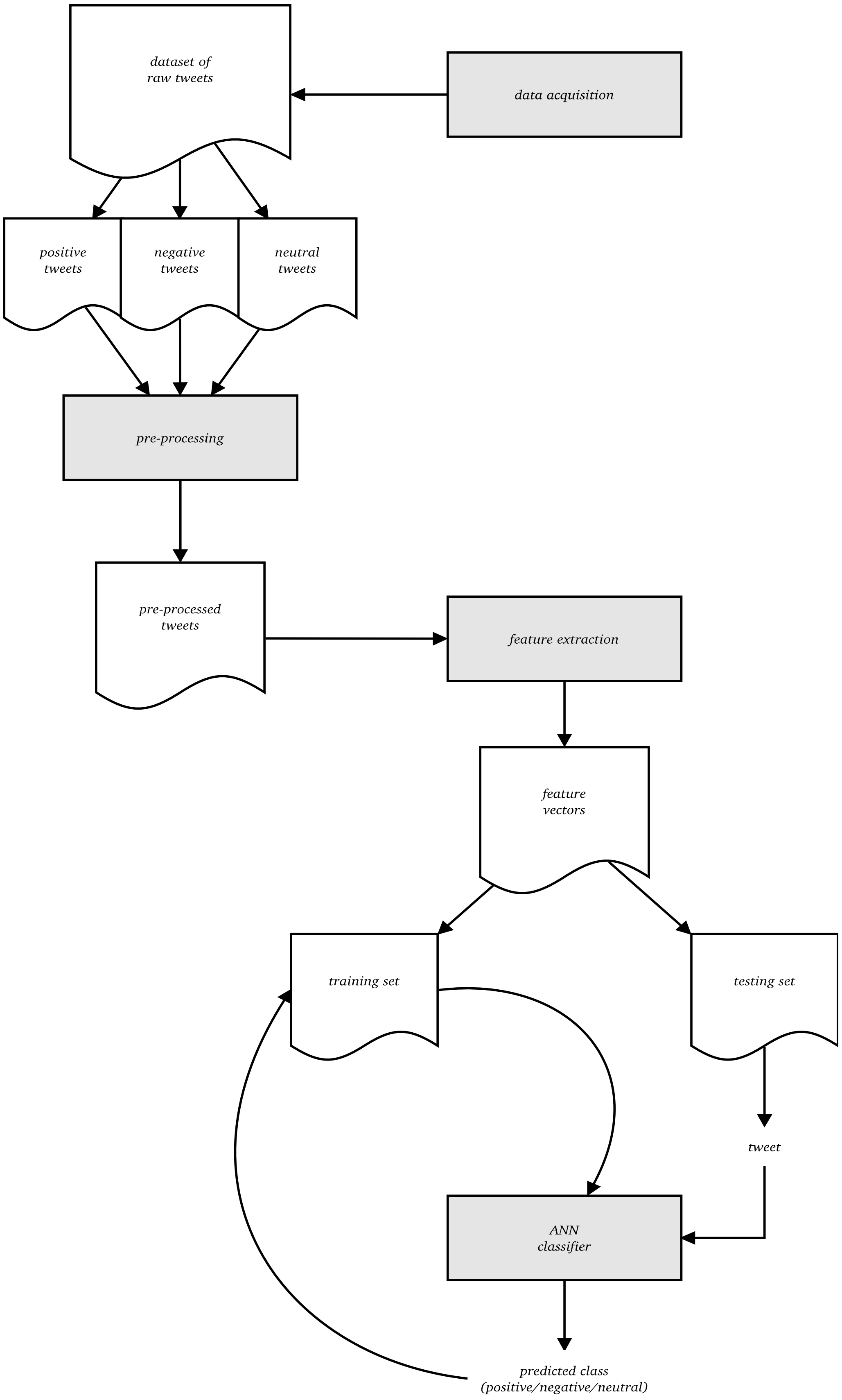


Figure 33: Schematic representation of the implemented model.

3.1 Data Acquisition

Our system is going to automatically detect the sentiment polarity of tweets. Hence, as a matter of fact, our first goal is to collect data of raw tweets. For this cause, we created a Twitter account and a Twitter application under the name “sentiment_analysis_gr” in the Twitter API [18]. A consumer key and a consumer token as well as an access key and an access token were generated for this application, which allow us to connect to Twitter and have access to the data the users we follow share (fig. 34).

In order to work with the Twitter API, we have to use OAuth to authorize our application for making requests on our behalf. Twitter supports OAuth, which is an open standard for authorization. OAuth provides client applications a secure delegated access to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. It is designed specifically to work with Hypertext Transfer Protocol (HTTP); OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner, or end-user. The client then uses the access token to access the protected resources hosted by the resource server. OAuth is commonly used in Twitter as well as other third party websites, as a way for users to log into accounts without exposing their password.

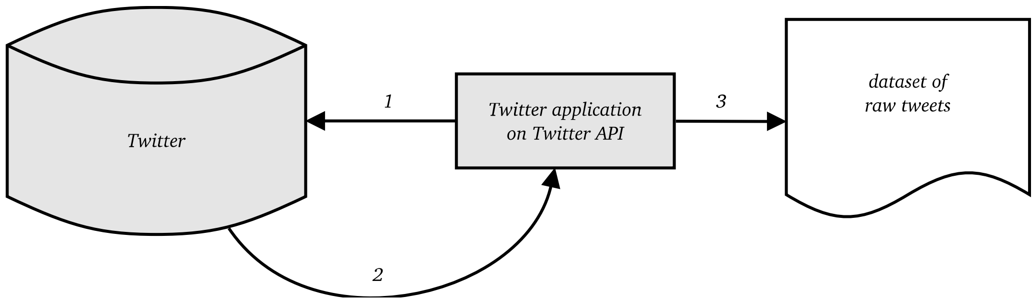


Figure 34: Schematic representation of the sequence of actions taking place for the acquisition of data from Twitter.

We trained and tested our model using a corpus that was divided in two datasets: the first one contains tweets expressing opinions of three discrete classes (positive, negative, neutral), while the second one contains tweets expressing opinions of two discrete classes (positive, negative).

The second dataset consists of raw tweets of various contents, already annotated as positive or negative. Apart from this already prepared annotated dataset, we also collected our own data according to a certain topic, which consisted our first dataset. The topic that was chosen is the referendum that took place in

Greece on July 5th, 2015. Inherently, a referendum is a binary problem that tends to polarize and divide people's opinions, therefore it was considered an excellent source of data for our cause. We collected data in a period of time from 1/7/2015 until 7/7/2015.

The annotation was performed manually. In chapter 1, we described some techniques for the annotation of tweets in an automated way, such as the annotation by taking into account the emoticons used [7]; for example the smiling smiley "*:-)*" is more possible to be part of a generally positive tweet rather than a negative one. However, as the topic about which the tweets were collected is a political one, so the use of emoticons was not so common, as well as it is very often to use them in an ironic way, we performed the task of the annotation manually in order to have an accurate and, as much as possible, error-free annotation.

In order to make this task easier for the trainer that performed the annotation, we implemented a series of pre-processing tasks (described in the subsequent section) that reduced the volume of the available data. As it will be seen further along, we achieved an average volume reduction of the datasets by **50.07%** per day (or per data subset), which proved to be a very good and helpful result for the trainer's job. However, even the fact that the original datasets containing from 2,000 to 2,670 tweets were reduced to ones containing from 1,042 to 1,139 tweets, due to time constraints, we performed the annotation on a set of 500 tweets per day. The reason for this is that the task of manual annotation is a quite demanding and time consuming process. Nevertheless, these data were collected with view to be used in a common experiment. In addition, they were used in combination and not being isolated, so our system could take into account and evaluate the common dictionary being used in all of them in the framework of the certain political topic. The total number of the all these data subsets is 7·500 = 3,500 tweets. This number can be considered quite sufficient in order to train our system.

During the process of manual annotation, it was apparent how difficult it is even for a human to identify the exact polarity of the sentiment of a written opinion, especially when the polarity should be quantized in such a small number of classes (3 classes; positive, negative or neutral) and cannot take into account other types of sentiments, e.g. the irony. Many times we met the difficulty of recognizing the polarity of the sentiment of an opinion as well as the framework within which it was expressed.

For example, let's check out a tweet of the tweets' collection of July 3rd, 2015 (from the dataset referendum_ds{3}):

"RT @EJDionne: A hope post-#greekreferendum: "The risk of catastrophe will concentrate minds..lead to positive surprises" @LHSummers"

The above tweet was finally considered to express a positive sentiment. However, it is commonly accepted that its meaning can be ambiguous due to the use of both words that express extremely negative sentiments (*"risk"*, *"catastrophe"*) and words that express positive sentiments (*"positive"*); thus, it took us some time to process and consider the meaning within the total context of the sentence and finally decide about its class. Therefore, it gets quite clear now that the task of sentiment extraction and classification of microtexts is a very hard and complicated challenge for an automated system like the one we are implementing here.

Given the fact that the annotation was performed by the same trainer for all datasets, we can assume that there is a kind of uniformity in the way that the samples were classified as positive, negative or neutral; hence, even though a flawless annotation of our samples was, as a matter of fact, impossible, there is a uniformity in the possibly existing deviations (between the trainer's annotation and the actual intentions of sentiment from the users expressing the collected opinions) of all datasets, and we therefore consider these deviations negligible regarding the conclusions extracted from our system.

- **referendum_ds{1}**: a dataset of initially 2,000 unannotated raw tweets, posted on July 1st, 2015 and regarding the Greek Referendum of July 5th, 2015. After a series of pre-processing tasks, a subset of the initial dataset was created, consisting of 1,119 positive, negative and neutral tweets.
- **referendum_ds{2}**: a dataset of initially 2,670 unannotated raw tweets, posted on July 2nd, 2015 and regarding the Greek Referendum of July 5th, 2015. After a series of pre-processing tasks, a subset of the initial dataset was created, consisting of 1,095 positive, negative and neutral tweets.
- **referendum_ds{3}**: a dataset of initially 2,670 unannotated raw tweets, posted on July 3rd, 2015 and regarding the Greek Referendum of July 5th, 2015. After a series of pre-processing tasks, a subset of the initial dataset was created, consisting of 1,042 positive, negative and neutral tweets.
- **referendum_ds{4}**: a dataset of initially 2,000 unannotated raw tweets, posted on July 4th, 2015 and regarding the Greek Referendum of July 5th, 2015. After a series of pre-processing tasks, a subset of the initial dataset was created, consisting of 1,043 positive, negative and neutral tweets.
- **referendum_ds{5}**: a dataset of initially 2,000 unannotated raw tweets, posted on July 5th, 2015 and regarding the Greek Referendum that took place on this day. After a series of pre-processing tasks, a subset of the initial dataset was created, consisting of 1,108 positive, negative and neutral tweets.
- **referendum_ds{6}**: a dataset of initially 2,000 unannotated raw tweets, posted on July 6th, 2015 and regarding the Greek Referendum of July 5th, 2015. After a series of pre-processing tasks, a subset of the initial dataset was created, consisting of 1,139 positive, negative and neutral tweets.
- **referendum_ds{7}**: a dataset of initially 2,000 unannotated raw tweets, posted on July 7th, 2015 and regarding the Greek Referendum of July 5th, 2015. After a series of pre-processing tasks, a subset of the initial dataset was created, consisting of 1,121 positive, negative and neutral tweets.

Table 1 summarizes all datasets of our experiments. As one can notice in this Table, the original subsets of positive, negative and neutral tweets for the datasets concerning the referendum (*referendum_ds{day}*) are not balanced. However, for the examination of our implemented architecture, we kept a set of 300 tweets per day (*balanced_referendum_ds{day}*); 100 positive, 100 negative and 100 neutral. The sum of these subsets produces a greater dataset, the *balanced_referendum_ds*, of 300·7 = 2,100 tweets, out of them 700 are positive, 700 negative and 700 neutral.

To sum up, the datasets that were used in order to train and test our model are the following ones:

- **balanced_referendum_ds**: a dataset of 2,100 annotated raw tweets, regarding the Greek Referendum of July 5th, 2015, and consisting of 700 positive, 700 negative and 700 neutral tweets.
- **various_contents_ds**: a dataset of 2,000 already annotated raw tweets of various contents, consisting of 1,000 positive and 1,000 negative tweets.

Datasets	Total number of tweets	Polarity	Positive class	Negative class	Neutral class
referendum_ds{1}	500	Positive, negative, neutral	114	218	168
referendum_ds{2}	500	Positive, negative, neutral	154	182	164
referendum_ds{3}	500	Positive, negative, neutral	131	127	242
referendum_ds{4}	500	Positive, negative, neutral	163	162	175
referendum_ds{5}	500	Positive, negative, neutral	155	152	193
referendum_ds{6}	500	Positive, negative, neutral	121	178	201
referendum_ds{7}	500	Positive, negative, neutral	121	262	117
Sum of all referendum_ds{1:7}	3,500	Positive, negative, neutral	959	1281	1260
balanced_referendum_ds{day}	300	Positive, negative, neutral	100	100	100
Sum of all balanced_referendum_ds{1:7}	2,100	Positive, negative, neutral	700	700	700
various_contents_ds	2,000	Positive, negative	1000	1000	0

Table 1: Overview of the datasets.

3.2 Pre-processing and Feature Extraction

3.2.1 Features of our Model

The features we decided to take into account for the model we implemented are extracted from the original words that consist a tweet, and are the following ones:

- lemmas
- stems
- parts-of-speech (POS)

A **lemma** (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words (headword). For example, in English, the words *"run"*, *"runs"*, *"ran"* and *"running"* are forms of the same lexeme, and their lemma is the word *"run"*. Lexeme, in this context, refers to the set of all the forms that have the same meaning, and lemma refers to the particular form that is chosen by convention to represent the lexeme. In lexicography, this unit is usually also the citation form or headword by which it is indexed.

The process of determining the lemma for a given word is called **lemmatization**. The lemma can be viewed as the chief of the principal parts, although lemmatization is at least partly arbitrary [19].

A **stem** is a part of a word. The term is used with slightly different meanings. In one usage, a stem is a form to which affixes can be attached. Thus, in this usage, the English word *"friendships"* contains the stem *"friend"*, to which the derivational suffix *"-ship"* is attached to form a new stem *"friendship"*, to which the inflectional suffix *"-s"* is attached. In a variant of this usage, the root of the word (in the example, *"friend"*) is not counted as a stem.

In a slightly different usage, a word has a single stem, namely the part of the word that is common to all its inflected variants. Thus, in this usage, all derivational affixes are part of the stem. For example, the stem of *"friendships"* is *"friendship"*, to which the inflectional suffix *"-s"* is attached. This is the usage we adopted in our model. The stem needs not to be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root.

Stemming is the term used to describe the process for reducing inflected (or sometimes derived) words to their word stem, base or root form. Algorithms for stemming (also known as stemmers) have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion [19].

The difference between stems and lemmas is that a stem is the part of the word that never changes even when morphologically inflected, whilst a lemma is the base form of the word. For example, from *"produced"*, the lemma is *"produce"*, but the stem is *"produc"*. This is because there are words such as *"production"*. In linguistic analysis, the stem is defined more generally as the analyzed base form from which all inflected forms can be formed. Some lexemes have several stems but only one lemma. For example, the lemma *"go"* has the stems *"go"* and *"went"*.

A **part-of-speech (POS)** is a category of words (or, more generally, of lexical items) which have similar grammatical properties. Words that are assigned to the same part of speech generally display similar behavior in terms of syntax -they play similar roles within the grammatical structure of sentences- and sometimes in terms of morphology, in that they undergo inflection for similar properties. Commonly listed English parts of speech are noun, verb, adjective, adverb, pronoun, preposition, conjunction, interjection, and sometimes numeral, article or determiner. Table 2 presents all the parts-of-speech we considered in our model [20].

A part of speech (particularly in more modern classifications, which often make more precise distinctions than the traditional scheme does) may also be called a word class, lexical class, or lexical category, although the term lexical category refers in some contexts to a particular type of syntactic category, and may thus exclude parts of speech that are considered to be functional, such as pronouns. The term form class is also used, although this has various conflicting definitions. Word classes may be classified as open or closed: open classes (like nouns, verbs and adjectives) acquire new members constantly, while closed classes (such as pronouns and conjunctions) acquire new members infrequently, if at all.

Part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context.

Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS tagging algorithms fall into two distinctive groups: rule-based and stochastic [19].

By POS tagging, we take into account the parts of speech within a document that indicate emotion. In most cases these are adjective-noun combinations such as *"devastating loss"*.

CC	Conjunction, coordinating
	(e.g. &, 'n, and, both, et, for, times, vs., yet)
CD	Numeral, cardinal
	(e.g. mid-1890, fifteen, 271.124, '60s, dozen)
DT	Determiner
	(e.g. all, an, another, any, this)
EX	Existentiel there
	(there)
FW	Foreign word (non-english word)
	(e.g. je, objets, fiche, hund, Herr)
IN	Preposition or conjunction, subordinating
	(e.g. astride, among, upon, below, within, behind)
JJ	Adjective or numeral, ordinal
	(e.g. third, pre-war, multilingual)
JJR	Adjective comparative
	(e.g. calmer)
JJS	Adjective, superlative
	(e.g. calmest)
LS	List item marker
	(e.g. A, A., first, SP-44002)
MD	Modal auxiliary
	(can, cannot, couldn't, might)
NN	Noun, common singular or mass
	(e.g. wind, hyena, override)
NNP	Noun, proper, singular
	(e.g. Oceanside, Escobar, Christos)
NNPS	Noun, proper, plural
	(e.g Americans, Amharas, Syndicalists)
NNS	Noun, common, plural

	(e.g. undergraduates, bric-a-brac, products)
PDT	Pre-determiner
	(e.g. all, both, half)
POS	Genitive marker
	(e.g. George's)
PRP	Pronoun, personal
	(e.g. hers, herself, me)
PRP$	Pronoun, possessive
	(e.g. her, his, mine)
RB	Adverb
	(e.g. occasionally, technologically, fiscally)
RBR	Adverb, comparative
	(e.g. further, grander, gloomier, louder)
RBS	Adverb, superlative
	(e.g. best, biggest, earliest)
RP	Particle
	(e.g. aboard, about, across, along, back)
SYM	Symbol
	(e.g. %, &, *, +, <, =, >, @)
TO	"to" as preposition or infinitive marker
	(to)
UH	Interjection
	(e.g. Goodbye, Wow, Oops, amen, honey, anyways)
VB	Verb, base form
	(e.g. ask, assess, assign, bake)
VBD	Verb, past tense
	(e.g. dipped, pleaded, halted)
VBG	Verb, present participle or gerund

	(e.g. telegraphing, focusing, alleging, encrypting)
VBN	Verb, past participle (e.g. desired, used, experimented, imitated)
VBP	Verb, present tense, not 3rd person singular (e.g. resort, sue, cure, appear, tend)
VBZ	Verb, present tense, 3rd person singular (e.g. bases, reconstructs, marks, mixes)
WDT	WH-determiner (that, what, whatever, which, whichever)
WP	WH-pronoun (that, what, whatever, whatsoever, which, who, whom, whosoever)
WP$	WH-pronoun, possessive (whose)
WRB	WH-adverb (how, however, whence, whenever, where, whereby, wherever, wherein, whereof, why)

Table 2: The part-of-speech (POS) tags considered by our model.

Therefore, after the pre-processing procedure of the raw datasets of tweets that is subsequently described, we expect to have kept a refined form of the tweet, which constructs a feature vector consisting of a combination of the features we described here: the original words, lemmas, stems and POS tags. For example, the tweet

“the children are playing in the yard ☀ #sunday”

is expected to take the following form, which makes use of a combination of the original words, their lemmas and POS tags:

[('children', u'child', 'NNS'), ('are', u'be', 'VBP'), ('playing', u'play', 'VBG'),
('yard', 'yard', 'NN'), ('sun', 'sun', 'NN'), ('sunday', 'sunday', 'NN')]

3.2.2 Pre-processing

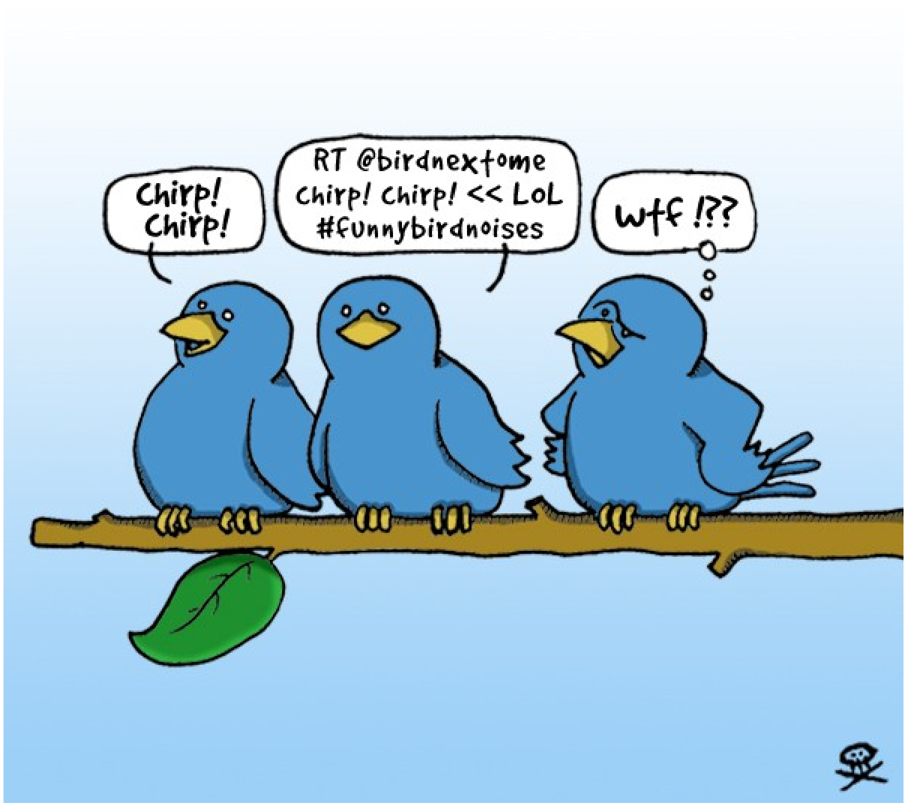


Figure 35: "Twitter bird in real life". Humorous portrayal of the language used in Twitter by Scott Hampson [36].

Given the fact that the language of microblogging requires a special treatment (as the use of informal and irregular words as well as the use of slang, misspellings, emoticons and acronyms are very frequent, which is aptly presented in the humorous fig. 35), we implemented a series of pre-processing tasks in order to correct and normalize the tweets for the feature extraction and construction of the feature vectors [21].

The pre-processing tasks we considered are the following:

1) Conversion of tweet to lower-case.
2) Conversion of hyperlinks (strings starting by *"www."*, *"http://"* and *"https://"*) to the string *"url"*.
3) Conversion of name mentions (*"@username"*) to the string *"at_user"*.
4) Elimination of additional white spaces.
5) Replacement of hashtags by converting hashtags of the generic form *"#foo"* into *"foo"* (removing the hash sign *"#"*).

6) Elimination of punctuation

7) Elimination of strings *“at_user”*, *“url”* and *“rt”* strings.

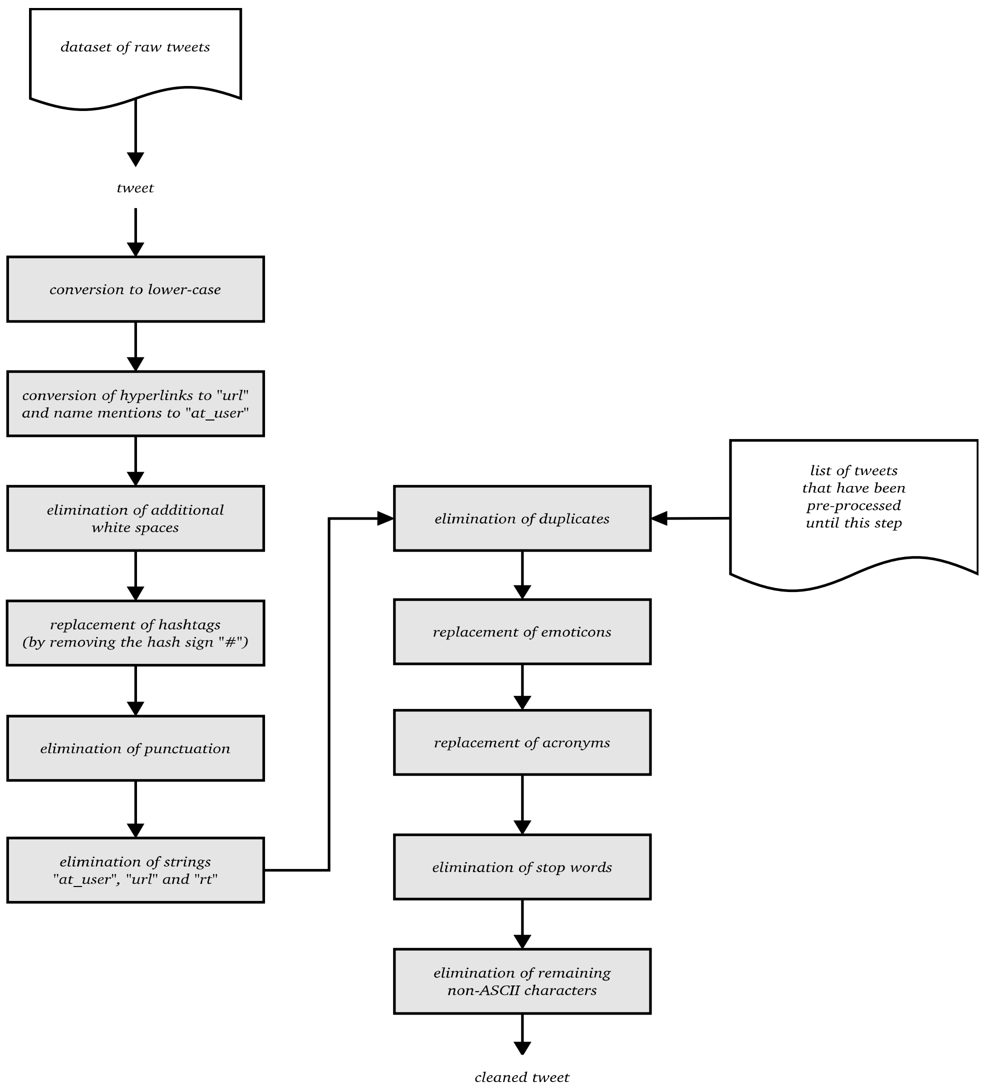
8) Elimination of duplicates.

9) Replacement of emoticons.

10) Replacement of acronyms.

11) Elimination of stop words.

12) Elimination of remaining non-ASCII characters.



dataset of raw tweets
tweet
conversion to lower-case
conversion of hyperlinks to "url" and name mentions to "at_user"
elimination of additional white spaces
replacement of hashtags (by removing the hash sign "#")
elimination of punctuation
elimination of strings "at_user", "url" and "rt"
elimination of duplicates
list of tweets that have been pre-processed until this step
replacement of emoticons
replacement of acronyms
elimination of stop words
elimination of remaining non-ASCII characters
cleaned tweet

Figure 36: Flowchart of the pre-processing procedure. The output of this procedure (the pre-processed tweet) is a cleaned version of the original tweet, in the sense that noise has tried to be eliminated while the meaningful information has tried to be maintained.

Let's now describe further the above steps:

- Step 1: In the first step we convert all tweets to lower-case. This is because our implementation is case-sensitive; if this step was absent, the system would treat different forms of the same word as different words. Thus, it would miss a lot of significant information about the correlation among words.

- Step 2: In this step, hyperlinks (all strings starting by *"www."*, *"http://"*, *"https://"*) are converted into the generic string *"url"*. The processing of hyperlinks would be extremely hard and quite unlikely to provide us with meaningful information, due to the fact that hyperlinks are very possible to contain meaningless sequences of letters or numbers. The most possible scenario would be that such a task would add noise to our data.

- Step 3: This step converts all mentions of usernames (of the form *"@username"*) into the generic string *"at_user"*. Name mentions don't provide any useful information for our system.

- Step 4: In this step, more than one white spaces are replaced.

- Step 5: In opposition to hyperlinks and name mentions, hashtags are possible to carry very useful information. A hashtag is a type of label or meta-data tag used on social network and microblogging services which makes it easier for users to find messages with a specific theme or content, e.g. *"#elections"*. Users create and use hashtags by placing the hash sign *"#"* in front of a word or unspaced phrase, either in the main text of a message or at the end of it. Searching for that hashtag will then present each message that has been tagged with it. It is also very common for Twitter users to use hashtags in order to describe their emotion or attitude, e.g. *"#cool"*. For this reason, we want to make use of the information that hashtags contain for our feature vectors. In step 5, a hashtag of the generic form *"#foo"* is replaced by the string *"foo"*, which is actually the original string of the hashtag without the *"#"* sign. As mentioned before, it is common a hashtag to contain an unspaced phrase, which makes the task of natural language processing much more complex. However, we have not taken any precautions in order to recognize such phrases and divide them into the original words; this kind of phrases may be considered as noise, but they are possible to additionally provide information if they are repeated and express a certain attitude regarding a topic.

- Step 6: In this step, all punctuation symbols are removed. Even though they are used to emphasize the presence of a sentiment in a microtext (e.g. *"I am very angry!"*), they just amplify the intensity of this sentiment and not add a new one; hence we didn't consider them important for the extraction of significant information that would lead our system to further acquisition of knowledge.

- Step 7: As by now we have grouped all hyperlinks and name mentions under the generic strings *"url"* and *"at_user"* respectively, our system filters them out. In addition, in this step it filters out the indicative string of retweets *"rt"* as neither does this type of meta-data provide any useful information to our model; actually the existence of retweets confirms the existence of duplicates, which do not offer more knowledge to our system but repeating the already existent knowledge. For this reason, we need to eliminate the strings of *"rt"* from all tweets in order to identify and eliminate the useless duplicates from our dataset (as seen in the subsequent step).

- Step 8: In this step, all duplicate tweets from the whole dataset are eliminated, and only one instance of every tweet is kept, as more instances of the same sample do not provide further knowledge to our system.

- Step 9: This step replaces emoticons by the word of the emotion or attitude they are considered to express. Emoticons (or emoticon icons) are meta-communicative pictorial representations of facial expressions that, in the absence of body language and prosody, serve to draw a receiver's attention to the tenor or temper of a sender's nominal non-verbal communication, changing and improving its interpretation. It expresses -usually by means of punctuation marks (though it can include numbers and letters)- a person's feelings or mood, though as emoticons have become more popular, some devices have provided stylized pictures that do not use punctuation. As social media has become widespread, emoticons have played a significant role in communication through technology. They offer another range of "tone" and feeling through texting that portrays specific emotions through facial gestures while in the midst of text-based cyber communication. For these reasons, in order to make use of their potentially very important value, in this step of the pre-processing procedure we replace emoticons by the word of the emotion they are supposed to represent. A dictionary of the 100 most common emoticons used in Twitter was made and can be accessed in the Appendix.

- Step 10: Acronyms are abbreviations formed from the initial components in a phrase or a word. They dominate in social networks and Twitter; apart from the fact that acronyms consist a quick, easy and popular way to write a phrase, they also compress a long phrase into several letters, which is very useful for expressing a greater piece of information while respecting the 140-characters limitation of Twitter. For this reason we couldn't ignore the information contained in acronyms. For our work, we created a dictionary of the 664 most frequently used acronyms of Twitter. In this step of the pre-processing procedure, acronyms are "decompressed" and replaced by the full phrase they express, according to the dictionary of the collected acronyms. The first 50 acronyms of this dictionary are given in the Appendix.

- Step 11: Stop words are words which are usually filtered out before or after processing of natural language data. Though stop words usually refer to the most common words in a language, there is no single universal list of stop words used by all processing of natural language tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search. In this step, all words that are members of a list of stop words we chose are eliminated from the tweets dataset. The list contains 320 english stop words and can be seen in Appendix. This step takes place after having replaced the emoticons and acronyms, as some words used in them (in the emoticons and especially in the acronyms) may consist stop words and therefore they should be eliminated.

- Step 12: In this step, the remaining non-ASCII characters (that were not replaced in previous steps, e.g. 香) are eliminated as our system wouldn't be able to understand and use them and, as a matter of fact, they would consist pure noise.

Let's see an example of how the pre-processing procedure develops. The following tweet (from *referendum_ds{6}*):

"RT @ariannahuff: Time for Europe to embrace #Greece http://www.demanjo.com/news/world/..."

after steps 1 to 6, would be converted into:

"rt at_user time for europe to embrace greece url"

and after step 7 it would be converted into

"time for europe to embrace greece"

It is very common a user to share a tweet of another user, using the option of retweet and then the string of *"RT"* would appear in the beginning of the shared tweet. In addition, a user might retweet an already existing tweet and mention another user or add a hyperlink or hashtag. For instance, a user different from the one that reposted the above tweet, had earlier written:

"Time for Europe to embrace #Greece"

These two tweets carry the same piece of information, which is extracted in the string:

"time for europe to embrace greece "

This example makes obvious the fact that the first 1-8 steps are very useful in order to identify the available information within a tweet and eliminate the meta-data of mentioning another user, retweeting or adding a hyperlink, which are not offering any meaningful information but noise to our system. By doing this, we are able to identify the duplicates in our dataset and eliminate them; they wouldn't give any additional knowledge to the system, and they would increase the required storing space and the processing time; hence they would burden the performance of the system. In addition, they would make the task of manual annotation much harder: As we described in section 3.1 for the data acquisition, our first datasets consist of already annotated tweets; nevertheless the datasets concerning the topic of the referendum (*referendum_ds{day}*, where day={1,2, …, 9}) were not annotated. The annotation was performed manually by us. The task of annotation requires plenty of time and concentration in order to offer careful and accurate results regarding the correct classification of tweets in the three classes of interest (positive, negative and neutral). For this additional reason, steps 1-8 were very essential for making the task of annotation the simplest possible for the human trainer that was assigned to it. Hence, for the datasets concerning the referendum, the pre-processing procedure was interrupted after step 8 in order to perform the manual annotation. After the annotation of tweets, the pre-processing procedure continued from step 9 to step 12. For datasets DS1 and DS2, all steps of the pre-processing procedure, from 1 to 12, were performed in order, without any interruption.

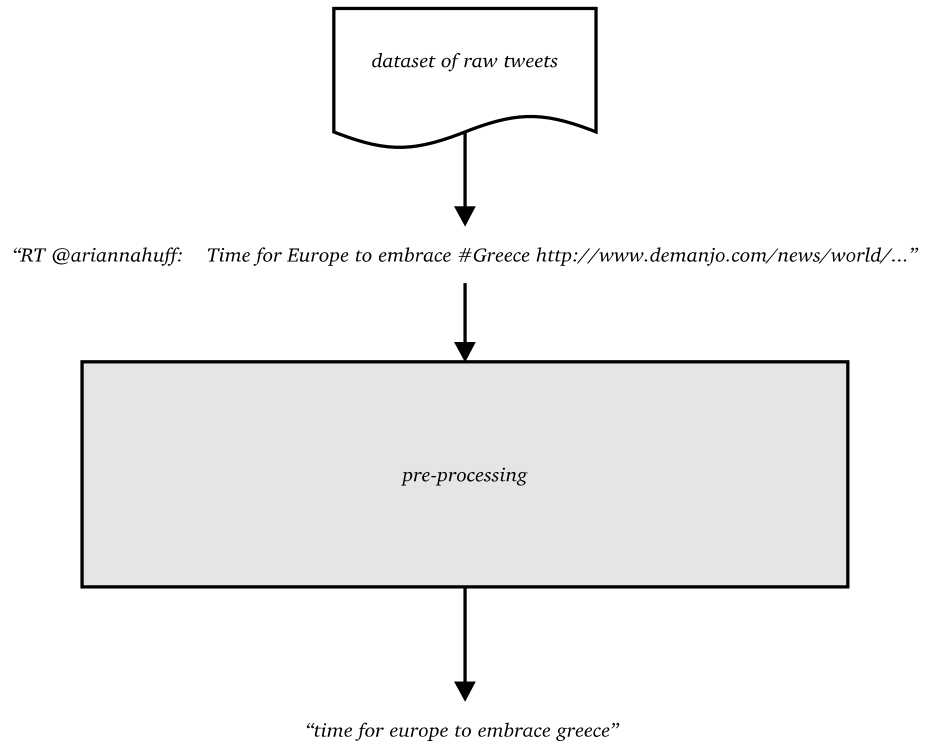

dataset of raw tweets
“RT @ariannahuff: Time for Europe to embrace #Greece http://www.demanjo.com/news/world/...”
pre-processing
“time for europe to embrace greece”

Figure 37: Schematic representation of the pre-processing procedure (as a black box) of a random raw tweet from our dataset. The output of this procedure is a cleaned tweet containing the meaningful information of the original tweet.

Let's now see how a tweet is transmuted after each pre-processing step, in another example. The tweet:

"RT @ledzeppelin: FYI Now Playing on #radio: Led Zeppelin - Kashmir https://youtu.be/sfR_HWMzgyc 😄♬♬"

after steps 1-6, would take the following form:

"rt at_user fyi now playing on radio led zeppelin kashmir url 😄 ♬♬"

After step 7, "at_user", "url" and "rt" strings are eliminated, and it would be converted into:

"fyi now playing on radio led zeppelin kashmir 😄 ♬♬"

If this cleaned tweet was repeated again in the dataset, the repetitions would be removed at step 8. Then, after step 9, the tweet would take the form:

"fyi now playing on radio led zeppelin kashmir happy ♬♬"

The subsequent step 10 would convert the tweet into:

"for your information now playing on radio led zeppelin kashmir happy ♬♬"

Afterwards, at step 11, stop words would be eliminated:

"information playing on radio led zeppelin kashmir happy ♬♬"

and finally, step 12 would convert the tweet into the final pre-processed tweet:

"information playing on radio led zeppelin kashmir happy"

Fig. 36 summarizes all the steps of the pre-processing procedure of our model.

3.2.3 Feature Extraction

After having performed the pre-processing steps described above onto our datasets, we have reduced their volume and have tried to eliminate the noise from them. In addition we have tried to make the text of tweets more comprehensible for a machine (for example a machine wouldn't be able to make a correlation between an emoticon and its corresponding word; for instance, *"happy"* and *"😄"* are referring to the same or similar sentiment, and our algorithm should be able to make use of this property). Hence, at this point, each tweet is represented by a much more clear sequence of words.

The features we chose for our system are the stems, lemmas and POS tags. Thus the next steps have to do with extracting or constructing these features from our set of tweets, which actually now consist lists of words; each tweet is now represented by a list of words. After extracting these features, by combining them we construct the feature vectors, as seen in fig. 38.

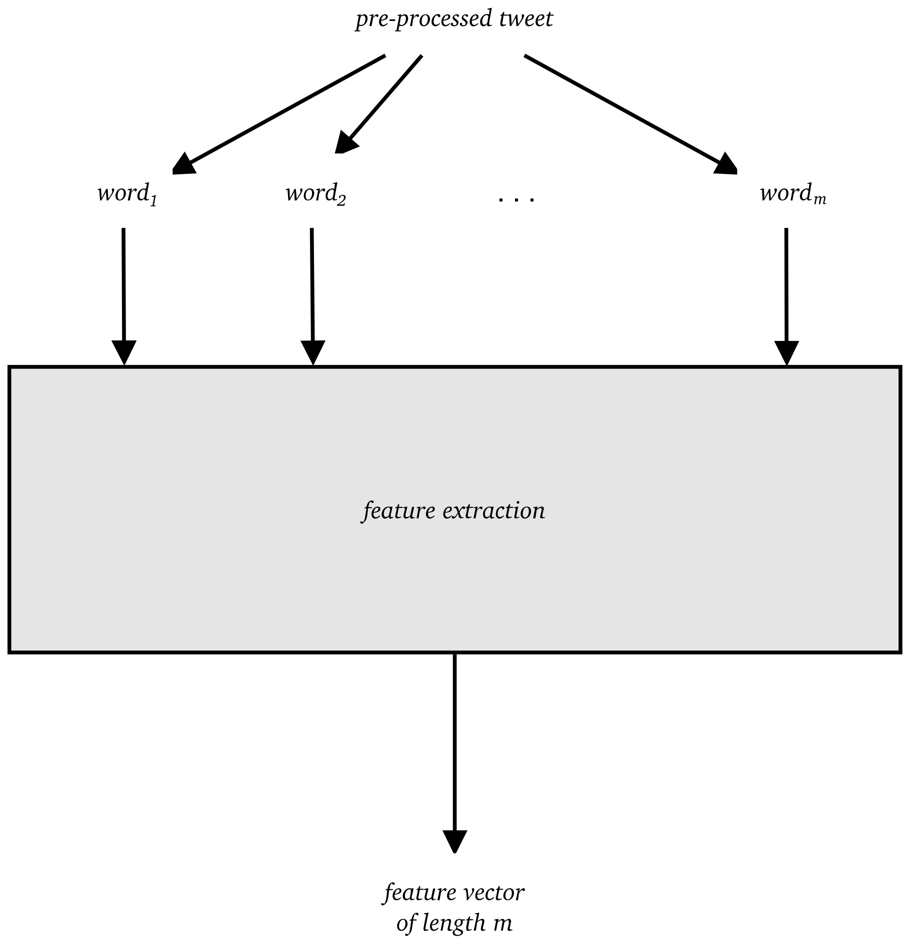


Figure 38: Schematic representation of the feature extraction process (as a black box). After extracting the features of each word consisting a tweet, the latter is transformed into a feature vector.

For the extraction of features, we have defined three functions: a function for the stemming, a function for the lemmatization and a function for the POS tagging of the tweets.

For the POS tagging, the Natural Language Toolkit (NLTK) platform provides a tool which assigns each word to its corresponding POS tag (one of the available POS tags we presented in Table 2). For example, *"loved"* would be assigned to the POS tag "VBD", which corresponds to a verb of past tense, according to Table 2.

For the stemming we have used the Lancaster Stemmer from NLTK. Every word out of a tweet is assigned to its stem. For example, the word *"loved"* would be assigned by the Lancaster Stemmer to the stem *"lov"*.

Finally, for the lemmatization we have used the Wordnet Lemmatizer from NLTK, which also takes into account the POS tag of words. For example, the word *"loved"* would be assigned to its lemma *"love"*.

After having assigned each word to its POS tag, stem and lemma, we can make various combinations of these features. For example, we can construct feature vectors consisted of the original word appearing in the raw tweet, with its stem and its POS tag; or we can construct feature vectors consisted of the original word with its lemma and its POS tag. Let's see an example: For the original tweet coming from the dataset *referendum_ds{2}*:

"RT @tsipras_eu: The #referendum gives an entire people the chance to affect the negotiation process. #Greece #Greferendum"

after the pre-processing procedure we would end up with a list of words of this shape:

['referendum', 'gives', 'entire', 'people', 'chance', 'affect', 'negotiation', 'process', 'greece', 'greferendum']

which can be assigned to a feature vector which consists of tuples of the form (original word, stem, POS tag):

[('referendum', 'referend', 'NN'), ('gives', 'giv', 'VBZ'), ('entire', 'entir', 'JJ'), ('people', 'peopl', 'NNS'), ('chance', u'chant', 'NN'), ('affect', 'affect', 'NN'), ('negotiation', u'negoty', 'NN'), ('process', 'process', 'NN'), ('greece', 'greec', 'NN'), ('greferendum', 'greferend', 'NN')]

or it could be presented as a feature vector of tuples of the form (original word, lemma, POS tag):

[('referendum', 'referendum', 'NN'), ('gives', u'give', 'VBZ'), ('entire', 'entire', 'JJ'), ('people', 'people', 'NNS'), ('chance', 'chance', 'NN'), ('affect', 'affect', 'NN'), ('negotiation', 'negotiation', 'NN'), ('process', 'process', 'NN'), ('greece', 'greece', 'NN'), ('greferendum', 'greferendum', 'NN')]

or even as a vector that combines both stems and lemmas, of the form (word, stem, lemma):

[('referendum', 'referend', 'referendum'), ('gives', 'giv', u'give'), ('entire', 'entir', 'entire'), ('people', 'peopl', 'people'), ('chance', u'chant', 'chance'), ('affect', 'affect', 'affect'), ('negotiation', u'negoty', 'negotiation'), ('process', 'process', 'process'), ('greece', 'greec', 'greece'), ('greferendum', 'greferend', 'greferendum')]

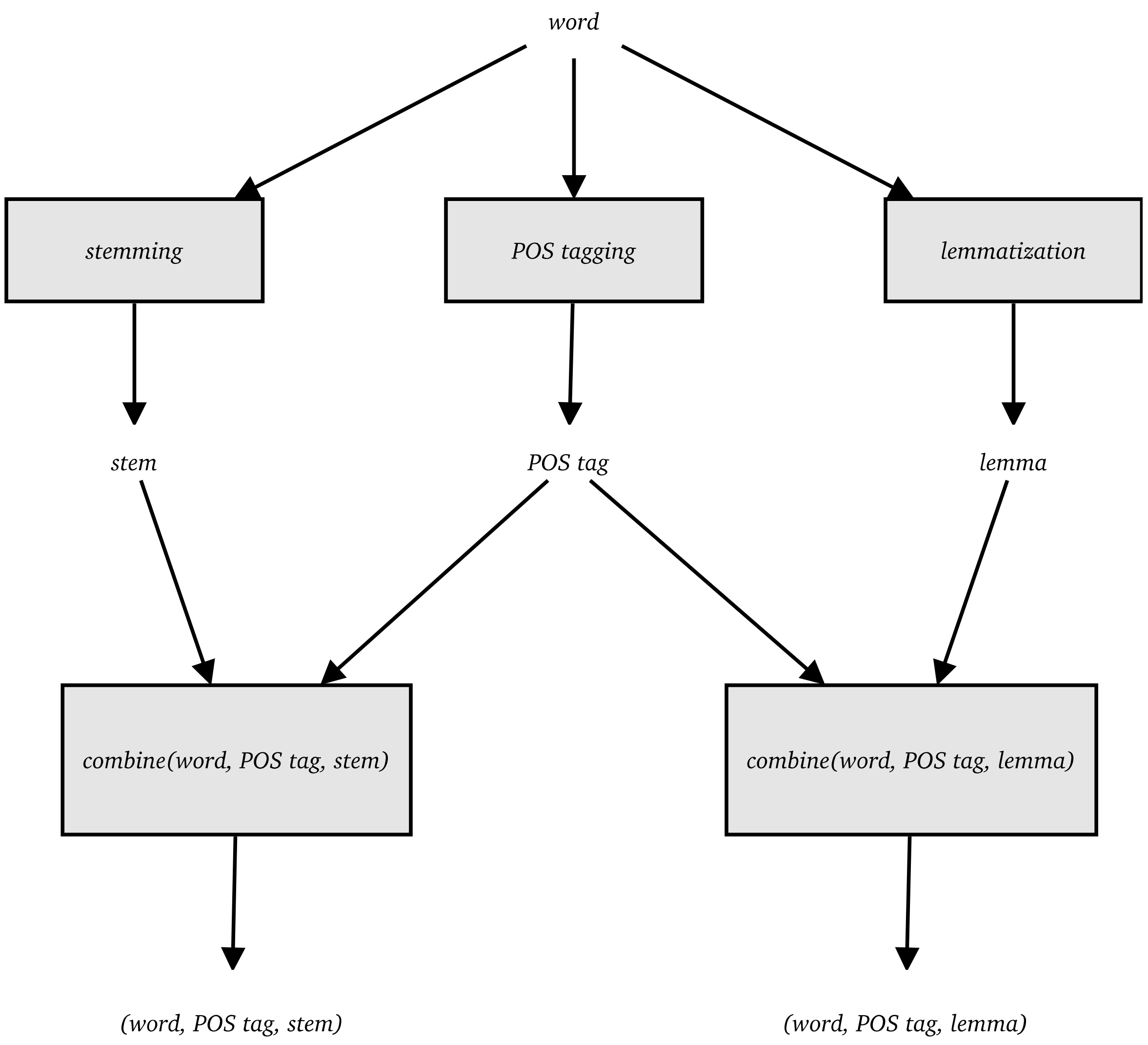


Figure 39: Procedure of feature extraction. The stems, lemmas and POS tags of a word are extracted and combined in order to construct the feature vector of a tweet.

This variety of combinations was used in our experiments (described in Chapter 4) in order to examine which type of feature vectors achieves the optimal performance for our case.

3.3 Architecture

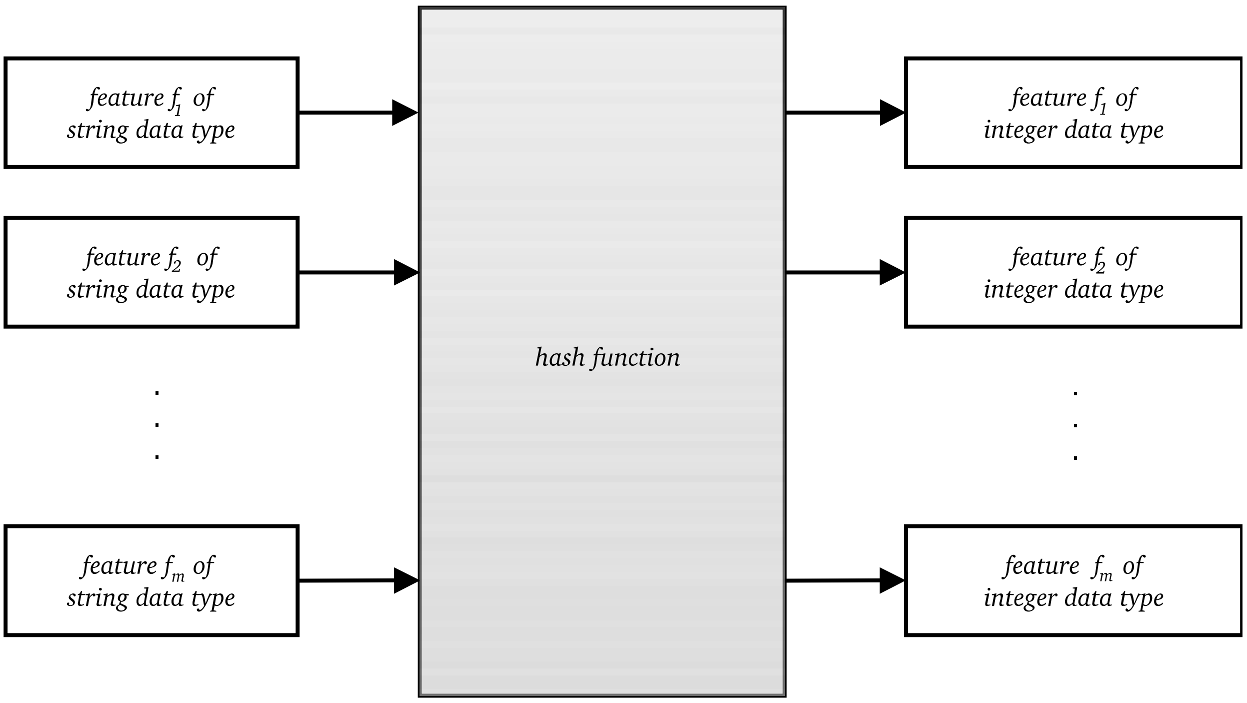
3.3.1 Input Preparation

As described in section 3.2.3, our feature vectors are consisted of a combination of the original words of the preprocessed tweets, their stems, lemmas and POS tags. In our implementation, these vectors are initially represented by lists of tuples; each tuple contains a number of string values, e.g. *('people', 'peopl', 'NNS')*.

After the stages of pre-processing and feature extraction, we need to start training our model, and the way we chose to do this is by an artificial neural network. The feature vectors feed the inputs of this ANN, being also assigned to a target value, which corresponds to the desirable output of the network (+1 if the tweet expresses a positive sentiment, 0 if it is neutral, and -1 if it expresses a negative sentiment). Based on these feature vectors and their assigned targets, our network is being trained by the BK algorithm (described in section 2.9).

Typically, a neural network is able to receive as inputs numerical values, in order to calculate the outputs of the activation functions. For this reason, we need a way to transform the inputs of our network from a list of tuple into numerical values. In order to achieve this, we made use of a hash function that assigns each of the tuples contained in the feature vector to a unique integer.

In general, a hash function is any function that can be used to map digital data of arbitrary size to digital data of fixed size. The values returned by this hash function are called hash values, hash codes, hash sums, or simply hashes. For our model, the hash function converts every different tuple of string values into a unique numerical value of integer data type.



feature f_1 of string data type
feature f_2 of string data type
feature f_m of string data type
hash function
feature f_1 of integer data type
feature f_2 of integer data type
feature f_m of integer data type

Figure 40: Transformation of the feature vectors' data type through a hash function.

The length of a feature vector, *m*, is the number of its component tuples. Hence, the length of the vector of the example of section 3.2.3 is 10. Our constructed set of feature vectors, which will be used for the training and testing of our network, contains vectors of various lengths. The number of the inputs of the network is judged by the length of the feature vectors; for this reason, all feature vectors of a given set need to have a common, consistent, length. In order to achieve this, we pad the vectors that are shorter than the longest one with as many padding tags as needed in order to make them consistent to the longest vector of our set. The padding tags we used are zeros (0), as the padding tags need to be of the same data type as the rest of the components of the vector, and, furthermore, zeros will not affect the network: the padding tags are treated as another regular word by the network while it learns its weights; so, as they will be multiplied by the synaptic weights, a result of 0 will be produced. The only reason we pad our vectors is to have a common length for all tweets and these padding tags should not play any other role within the network; hence, zeros serve this cause in an excellent way. Padding could have been avoided if the network was implemented with a max-over-time module, which chooses the most significant features from variable-length sentences to get a fixed size feature vector.

All the above processing steps took place for every experimental combination and system architecture we examined. After performing the above steps, we examined two different techniques of treating the input vectors. These techniques are two variations of the idea of n-grams.

An n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs. The n-grams typically are collected from a text or speech corpus. A “unigram” is an n-gram of size 1 and a “bigram” is an n-gram of size 2. Bigram features are not that commonly used in text classification tasks. However, bigrams seem to improve the performance [22], and this is why we decided to try our system on both bigrams and unigrams.

For our application, the n sequential items are words (even if they are represented by numerical values); we consider the feature vectors as vectors consisted of n-grams of the features we have extracted for each tweet. For example, for the sentence

“time for europe to embrace greece”

a vector consisting of unigrams would be

[“time”, “for”, “europe”, “to”, “embrace”, “greece”]

while a vector of bigrams would be

[(“time”, “for”), (“for”, “europe”), (“europe”, “to”), (“to”, “embrace”),
(“embrace”, “greece”)]

Therefore, we examined the performance of our system being fed by the following types of feature vectors:

- a feature vector of unigrams of our processed tweets, which actually is equivalent to the feature vector produced after the step of padding.
- a feature vector of bigrams of our processed tweets.

The vector of n-grams that is going to feed the neural network, in the second case (vector of bigrams), is consisted of a number of tuples. For the reason we described earlier, we need again to apply a hash function onto this vector in order to convert the tuples into numerical values. For the case of unigrams, the feature vector remains the same.

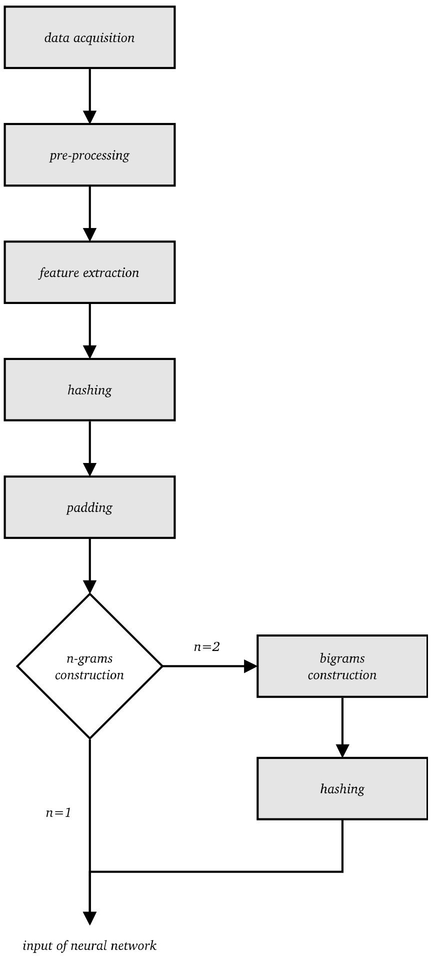


Figure 41: Summary of the sequence of stages of tweets' processing procedure before feeding the input of the neural network.

3.3.2 Multi-layer Perceptron

By now, we have prepared the feature vectors in order to be in acceptable forms to feed the inputs of our network. Let's now describe the architecture of the implemented neural network.

Neural networks initialized using weights derived from linear models have been shown to present a good performance on a variety of classification tasks [23]. Our neural network for the task of sentiment analysis consists of 3 layers (including the input layer); an input layer, a hidden one and an output one, as seen in fig. 37. The architectures we examined depend on the dimension of the hidden layer and are presented later, in Table 3. Let's, firstly, define the symbols we are going to use:

d_{input} =	dimension of input layer
d_{hidden} =	dimension of hidden layer
d_{output} =	dimension of output layer
$\boldsymbol{w_{12}}$ =	weights vector from input layer to hidden layer
$\boldsymbol{w_{23}}$ =	weights vector from hidden layer to output layer

The layers of our network are common in all the architectural variations we examined and are described below.

1. Input Layer

The input layer is a linear layer, which performs a simple multiplication and an additive bias layer. The synaptic weights vector $\boldsymbol{w_{12}}$ is multiplied with the input matrix $\boldsymbol{x}$, and a bias b is added to each element. The synaptic weights and the bias are learned using the BK algorithm (described in section 2.9). The dimension of the input layer is equal to the length of the padded feature vectors, m_p:

$$d_{input} = m_p$$

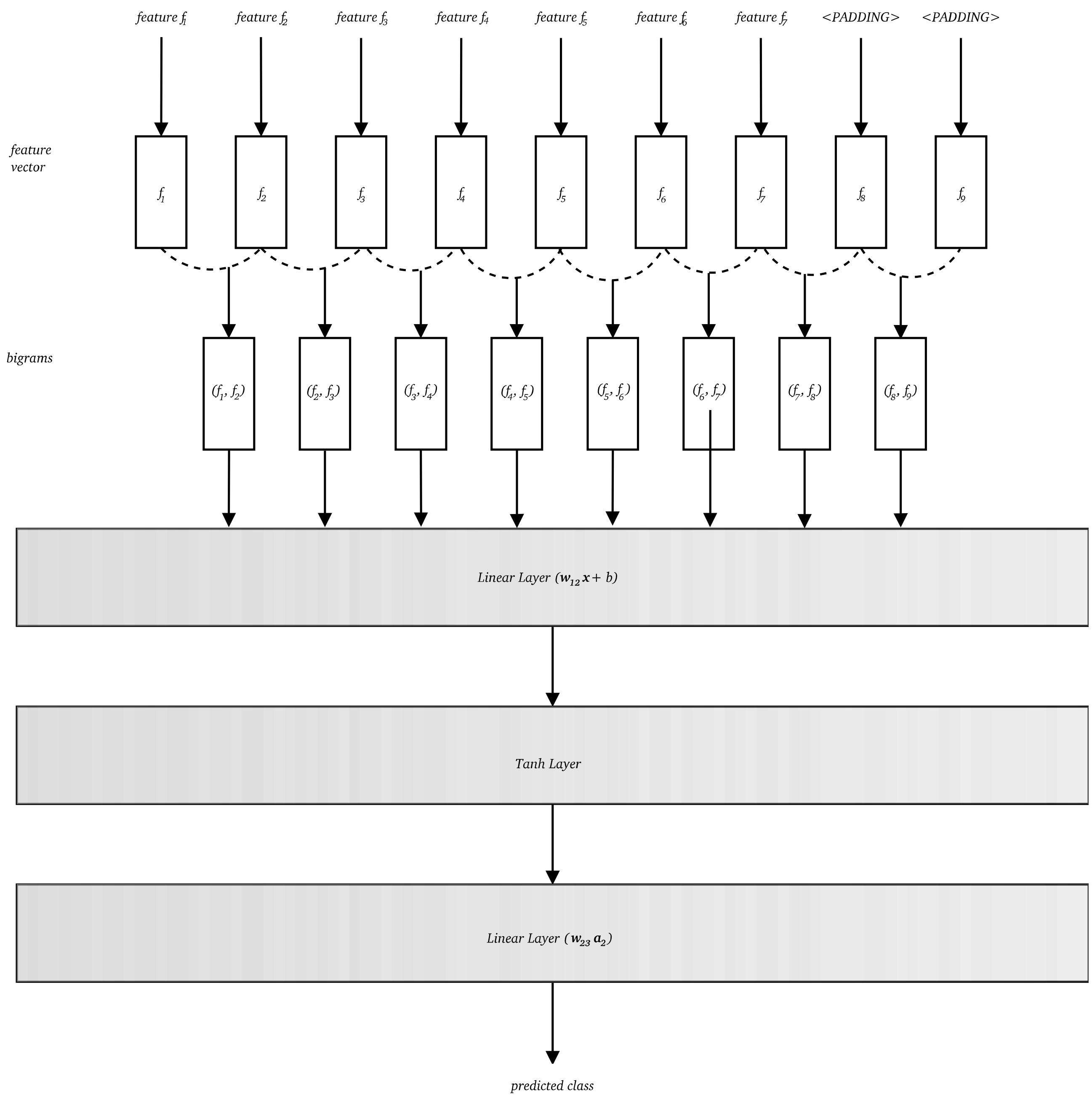


Figure 42: Architecture of the network. In this example, the inputs of the network are fed by bigrams of the feature vector.

2. Hidden Layer

The hidden layer is a hyperbolic tangent (tanh) layer, in the sense that it applies the tanh function to each element in the matrix $\boldsymbol{a_1}$ which is consisted of the outputs of the input layer. This introduces a sigmoid non-linearity which make the architecture different from regular linear classifiers. Tanh is a constantly differentiable non-linear activation function (it is a function with sigmoid non-linearity). It is defined as

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$
$$\textit{where } x \textit{ is a continuous variable}$$

We examined the system's response depending on different architectures; the difference among them lays on the dimension of this layer. Table 3 presents the dimension of the hidden layer for every architecture we examined.

3. Output Layer

The output layer is again a linear layer. As for the input, the linear output layer performs a multiplication and an additive bias layer. The synaptic weights vector $\boldsymbol{w_{23}}$ is multiplied by the matrix $\boldsymbol{a_2}$ of the outputs of the hidden layer (i.e. after the application of the sigmoid activation function on them). The synaptic weights and the bias are learned using the BK algorithm. The dimension of the output layer may be 3 in the case of 3 classes of interest (positive, negative, neutral), or 2 in the case of only two classes of interest (positive, negative).

$$d_{output} = \textit{number of classes of interest} = \begin{cases} 2, & \textit{if 2 classes of interest} \{\textit{positive}, \textit{negative}\} \\ 3, & \textit{if 3 classes of interest} \{\textit{positive}, \textit{negative}, \textit{neutral}\} \end{cases}$$

	Size of Layers		
	Input Layer	**Hidden Layer**	**Output Layer**
Architecture 1	d_{input}	d_{input}	d_{output}
Architecture 2	d_{input}	d_{input} div 2	d_{output}
Architecture 3	d_{input}	d_{input} div 4	d_{output}
Architecture 4	d_{input}	$d_{input} \cdot 2$	d_{output}
Architecture 5	d_{input}	$d_{input} \cdot 2$ div 3	d_{output}
Architecture 6	d_{input}	$d_{input} \cdot 4$	d_{output}
Architecture 7	d_{input}	$(d_{input}+d_{output}) \cdot 3$ div 2	d_{output}
Architecture 8	d_{input}	$(d_{input}+d_{output}) \cdot 2$ div 3	d_{output}

Table 3: Overview of the examined architectures.

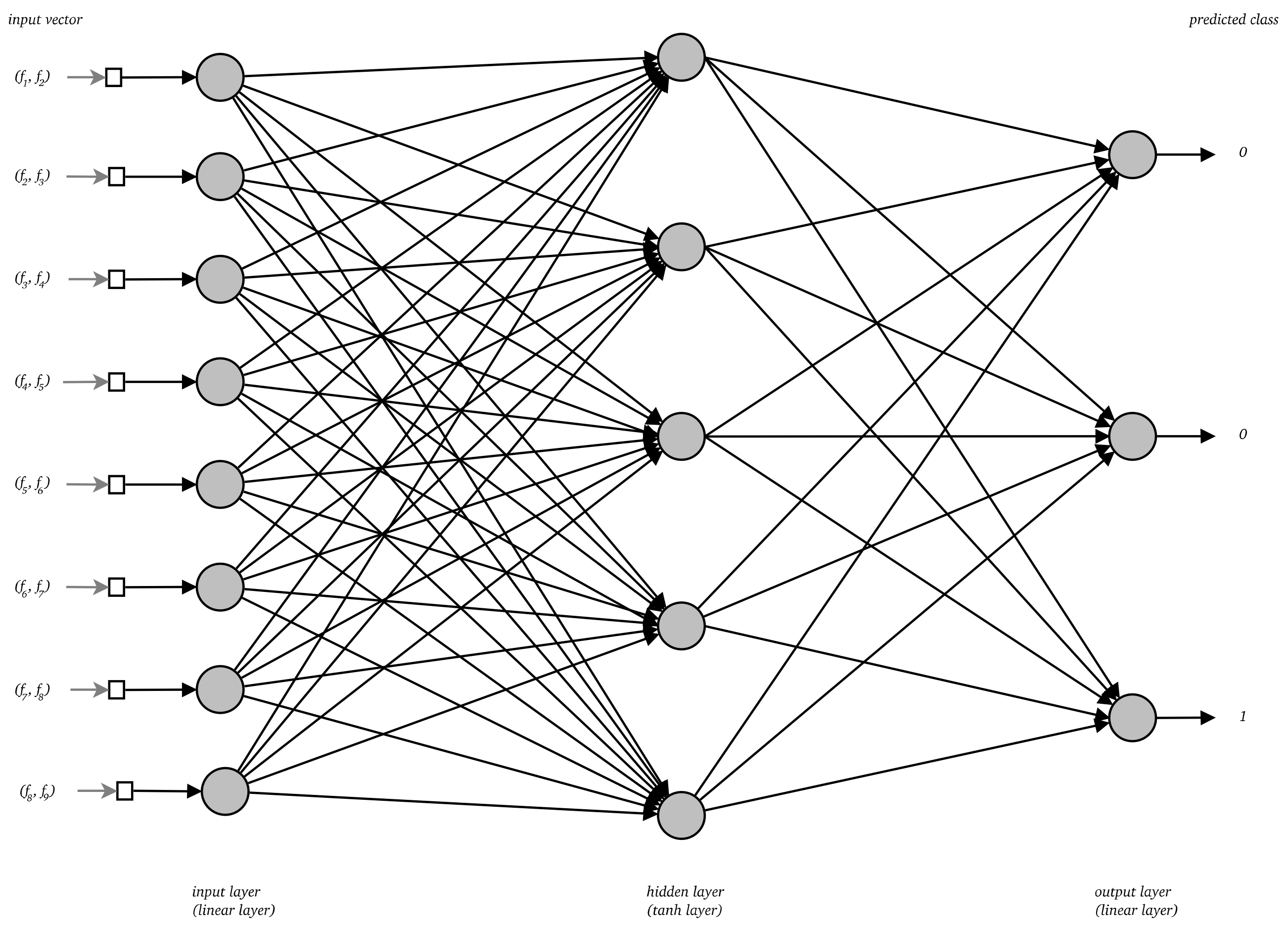


Figure 43: Example of network with the architecture 5. The input vector is consisted of bigrams of the feature vector [f1, f2, ..., f9]. The dimension of the input layer is 8; the dimension of the hidden layer is (8·2 div 3) = 5; and the dimension of the output layer is 3, as the network is able to classify the input vector into one out of three classes (positive, negative or neutral).

3.4 Computational Complexity

As described earlier, our model consists of four main stages:

- the pre-processing stage
- the feature extraction stage
- the input preparation stage
- the multi-layer perceptron stage

The computational complexity of the total model is derived from the partial orders of complexity of these stages. Before computing the complexity of each stage, let's define the symbols we are going to use for the computation of the order of complexity of our model.

$n_{initial}$ =	number of tweets that consist the original dataset
$m_{initial}$ =	initial maximum number of words per tweet
$n_{final} = n$ =	number of remaining tweets in dataset after step 7 of pre-processing, $\mathrm{n} \leq \mathrm{n}_{\mathrm{initial}}$
m_{temp1} =	number of words per tweet after step 7 of pre-processing, $\mathrm{m}_{\mathrm{temp1}} \leq \mathrm{m}_{\mathrm{initial}}$
m_{temp2} =	number of words per tweet after steps 9-10 of pre-processing, $\mathrm{m}_{\mathrm{temp2}} \geq \mathrm{m}_{\mathrm{temp1}}$
$m_{final} = m$ =	number of remaining words per tweet after step 11 of pre-processing, $\mathrm{m} \leq \mathrm{m}_{\mathrm{temp2}}$
$d_{input} = m$	dimension of input layer of MLP
d_{hidden} =	dimension of hidden layer of MLP
d_{output} =	dimension of output layer of MLP
l =	number of training epochs

Let's note here that the symbol *n* that will be used here as the (final) number of remaining tweets after step 7 of pre-processing, is independent of the symbol *n* that was used in the case of n-grams earlier.

Let's now explain further these symbols. The initial number of tweets per dataset is symbolized by $n_{initial}$. In step 8 of pre-processing, possible duplicate tweets are eliminated, hence the number of tweets per dataset changes and now is equal to n_{final} which will be referred as *n* for reasons of simplicity. It is $n_{final} \leq n_{initial}$.

The initial maximum number of words per tweet is symbolized by $m_{initial}$; at the worst case scenario, all tweets are consisted of m words. In step 7, strings *"at_user"*, *"url"* and *"rt"* are eliminated, so the new number of words per tweet is symbolized now by m_{temp1}, $m_{temp1} \leq m_{initial}$. In steps 9 and 10, emoticons and acronyms are replaced by their corresponding words; thus the total number of words per tweet is possible to increase, and now it is equal to $m_{temp2} \geq m_{temp1}$. Nevertheless, in the next step of pre-processing (step 11), stop words are filtered out, hence the number of words per tweets is possible to differ from the previous one. We symbolize this final number of words per tweet with m_{final} or *m*, for reasons of simplicity. It is $m_{final} \leq m_{temp2}$.

In the stage of input preparation our system pads the tweets in order to set them all in a consistent length; this length is equal to m as well as to the number of the inputs of the perceptron: $d_{input} = m$.

In the stage of pre-processing, steps 1-6 are performed in a single dataset scan, which means that the system needs to access $n_{initial}$ tweets of length $m_{initial}$ each (at worst case scenario); step 7 is performed similarly; step 8 accesses each tweet ($n_{initial}$ tweets) without trying to access the words consisting it; steps 9-10 are performed in a single scan and need to access $n_{final} = n$ tweets of length m_{temp1} each; step 11 needs to access n tweets of the new length m_{temp2}; step 12 needs to access n tweets of the new length (after step 11) $m_{final} = m$. The partial orders of complexity can be seen in Table 4.

In the stage of feature extraction, each one of the tasks of lemmatization, stemming and POS tagging need to access each word of the pre-processed tweets. The number of pre-processed tweets is n and the maximum number of words in them is m. This stage produces a tuple out of each word; the length of each feature vector produced of each tweet remains equal to m.

In the stage of input preparation, each of the tasks of padding, hashing and construction of the bigrams needs to access each tuple per feature vector; this means they need to access n feature vectors consisting of m tuples.

The orders of complexity for the stages of feature extraction and input preparation can be also seen in Table 4.

Let's now compute the complexity of the multi-layer perceptron. When training a multi-layer perceptron, its computational complexity is determined by several factors. Our multi-layer perceptron consists of a single hidden layer network. The network is fully connected and the factor of the momentum is standard.

The dominating factor in training the MLP is the number of synaptic weights. As our network is fully connected, there are $d_{input} \cdot d_{hidden} = m \cdot d_{hidden}$ weights from the input layer to the hidden layer, and $d_{hidden} \cdot d_{output}$ weights from the hidden layer to the output layer. This gives a total of $d_{hidden} \cdot (m+d_{output})$ weights. The number of inputs and outputs for a given dataset is fixed, hence the only variable term is d_{hidden}.

Let's now describe in terms of complexity the back propagation error phase. The computation of error at the output nodes uses the same back propagation error term, δ_j :

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \tag{81}$$

where t_j is the target for output node j and o_j is the actual value for output node j.

This error term is $\mathcal{O}(1)$ for each output node and that there are $d_{output} \cdot d_{hidden}$ output weights, which yields $\mathcal{O}(d_{output} \cdot d_{hidden})$ for the output layer. After computing the error at the output nodes, the error at each hidden node is computed. Using the same assumptions as for equation 81,

$$\delta_h = o_h (1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k \tag{82}$$

where w_{kh} represents the weight from node h in the hidden layer to node k in the output layer.

This computation is $\mathcal{O}(d_{output})$ for each hidden node and there are $d_{hidden}\cdot m$ weights, yielding $\mathcal{O}(m\cdot d_{output}\cdot d_{hidden})$ for the entire hidden layer.

The total order of complexity is $\mathcal{O}(m\cdot d_{output}\cdot d_{hidden}+d_{hidden}\cdot d_{output})$ or $\mathcal{O}(d_{hidden}\cdot d_{output}\cdot(m+1))$ for training a single epoch. For training l epochs, the total order of complexity is $\mathcal{O}(d_{hidden}\cdot d_{output}\cdot l\cdot(m+1))$. For the training of the whole (pre-processed and prepared) dataset that contains n tweets, the total order of complexity is $\mathcal{O}(d_{hidden}\cdot d_{output}\cdot l\cdot n\cdot(m+1))$. As we use standard momentum, this order of complexity is unchanged and fewer epochs are required for convergence that without momentum. Standard momentum requires additional storage for each weight. Storage requirements are directly proportional to the number of weights; with $d_{hidden}\cdot(m+d_{output})$ weights, there are $2\cdot d_{hidden}\cdot(m+d_{output})$ values to store.

The total order of complexity of our model is $\mathcal{O}(d_{hidden}\cdot d_{output}\cdot l\cdot n\cdot(m+1))$. Table 4 summarizes the orders of complexity for every stage and individual task of our model.

	Task	**Computational Complexity**
Pre-processing	Steps 1-6	$\mathcal{O}(n_{initial}\cdot m_{initial})$
	Step 7	$\mathcal{O}(n_{initial}\cdot m_{initial})$
	Step 8	$\mathcal{O}(n_{initial})$
	Steps 9-10	$\mathcal{O}(n\cdot m_{temp1})$
	Step 11	$\mathcal{O}(n\cdot m_{temp2})$
	Step 12	$\mathcal{O}(n\cdot m)$
Feature extraction	Lemmatization	$\mathcal{O}(n\cdot m)$
	Stemming	$\mathcal{O}(n\cdot m)$
	POS tagging	$\mathcal{O}(n\cdot m)$
Input preparation	Padding	$\mathcal{O}(n\cdot m)$
	Hashing	$\mathcal{O}(n\cdot m)$
	Bigrams construction	$\mathcal{O}(n\cdot m)$
Multi-layer perceptron		$\mathcal{O}(d_{hidden}\cdot d_{output}\cdot l\cdot n\cdot(m+1))$

Table 4: Summary of computational complexity of all individual tasks of the model.

3.5 Performance Evaluation

For the performance evaluation of the implemented system we used the method of k-fold cross-validation. As described in section 2.9.4, in this method, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining *k−1* subsamples are used as training data. The cross-validation process is then repeated *k* times, with each of the *k* subsamples used exactly once as the validation data. The *k* results from the folds can then be averaged (or otherwise combined) to produce a single estimation. By using this method over repeated random subsampling, all observations are used for both training and validation, and each observation is used for validation exactly once. In our implementation, *k* was set equal to 5 (**5-fold cross-validation**).

The performance of the our system is evaluated by the metric of the mean squared error (MSE), defined as a function of the free parameters of our system, which are the synaptic weights of the neural network. The MSE of an estimator measures the average (mean magnitude) of the squares of the errors, that is the difference between the model's estimation of the test values and the actual (corresponding) test values. Squaring is used to covert the errors to an absolute value. The physical interpretation of the MSE metric is how close, on average, the hyperplane drawn by our network gets to the actual cloud of data in the validation set.

3.6 *Implementation*

We implemented our system using Python [24]. Python provides some very useful tools for our application. We mainly used the modules of Tweepy, NLTK and PyBrain. All of these modules are open-sourced and hosted on GitHub [37].

Tweepy is Twitter API library for Python [25]. It enables Python to communicate with Twitter platform and use its API.

NLTK (Natural Language Toolkit) is a leading platform for building Python programs to work with human language data [20]. It provides easy-to-use interfaces to corpora and lexical resources, along with a suite of text processing libraries for classification, tokenization, stemming, lemmatization, tagging, parsing, and semantic reasoning.

PyBrain (Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library) [26] is a modular Machine Learning Library for Python. It offers powerful algorithms for machine learning tasks and a variety of environments for testing and comparison. It contains algorithms for neural networks, reinforcement learning (and the combination of the two), unsupervised learning, and evolution.

4 Experiments and Results

In this chapter we are going to describe the different experiments, techniques and architectures of the neural network that were examined, and present their results. Before that, we are going to describe the system characteristics of the computer where the experiments took place, the parameters that we set for the training and the testing of the artificial neural network, as well as the data corpus that was used. In the end, we will discuss the results and suggest ideas for improvement.

4.1 Computer System Characteristics

The characteristics of the computer system where the experiments took place are presented in the following Table.

Memory	7.7 GiB
Processor	Intel® Core™ i7-4500U CPU @ 1.80GHz x 4
OS type	64-bit
OS	Ubuntu 14.04.2 LTS
Disk	448.9 GB

Table 5: Computer system characteristics.

4.2 Training and Testing Corpus

Let's remind here the corpus of our experiments. The initial corpus is pre-processed and cleaned of duplicates. The sections of the corpus that are used for the training and the testing phases of our system consist of the 75% and 25%, respectively, of the datasets after the pre-processing steps.

Datasets	**Polarity**	**Number of tweets**						
		Initial corpus	**Positive class of initial corpus**	**Negative class of initial corpus**	**Neutral class of initial corpus**	**Pre-processed corpus**	**Training corpus**	**Testing corpus**
balanced_referendum_ds	Positive, negative, neutral	2,100	700	700	700	1,830	1,373	457
various_contents_ds	Positive, negative	2,000	1,000	1,000	0	1,751	1,314	437

Table 6: Overview of the training and testing corpus.

Let's also underline that the two datasets of our corpus differ in two points:

- The dataset *balanced_referendum_ds* contains tweets of a certain topic (the Referendum of July 5th, Greece) while the dataset *various_contents_ds* contains tweets of various topics and contents.
- The dataset *balanced_referendum_ds* contains tweets with opinions that have been annotated as positive, negative or neutral (three classes of polarity), while the dataset *various_contents_ds* contains tweets of opinions that have been annotated as positive or negative (two classes of polarity).

4.3 Training and Testing parameters

The proportion of the training and the testing data to the total corpus was set at 75% and 25% respectively.

The term of the learning rate is a training parameter that controls the size of weight and bias changes in learning of the training algorithm [14]. Its real domain is [0, 1]. We decided that the value that leads our system to a better performance is 0.01.

The momentum is a term that adds a fraction m of the previous weight update to the current one. The momentum parameter is used to prevent the system from converging to a local minimum or saddle point. A high momentum parameter can also help to increase the speed of convergence of the system. However, setting the momentum parameter too high can create a risk of overshooting the minimum, which can cause the system to become unstable. A momentum coefficient that is too low cannot reliably avoid local minima, and can also slow down the training of the system. Its real domain is [0, 1] [27]. We set the momentum parameter equal to 0.5

The parameter of weight decay adds a penalty term to the error function. The penalty used is the sum of squared weights times a decay constant. The weight decay penalty term causes the weights to converge to smaller absolute values than they otherwise would. Large weights can hurt generalization in two different ways. Excessively large weights leading to hidden units can cause the output function to be too rough, possibly with near discontinuities [27]. Excessively large weights leading to output units can cause wild outputs far beyond the range of the data if the output activation function is not bounded to the same range as the data. In other words, large weights can cause excessive variance of the output (Geman, Bienenstock, and Doursat 1992).

The number of epochs was set at 1,000. This number determines when training will stop once the number of iterations exceeds epochs or when the network converges to a minimum error.

Table 5 summarizes the training parameters of the neural network.

Proportion of the training dataset to the total corpus	0.75
Proportion of the testing dataset to the total corpus	0.25
Learning rate	0.01
Momentum	0.5
Weight decay	0.01
Number of epochs	1,000
Cross-validation	5-fold
Performance metric	Mean Squared Error (MSE)

Table 7: Overview of the training parameters

4.4 Experiments and Techniques

For the dataset *balanced_referendum_ds*, we examined the performance of the implemented system in two different experiments, depending the features that constructed the feature vectors:

- Experiment 1: The features that were selected are the original word, its stem and POS tag.
- Experiment 2: The features that were selected are the original word, its lemma and POS tag.

For the second dataset, *various_contents_ds*, we examined the performance of the system with the techniques of unigrams and bigrams, taking into account the features of the experiment 1.

4.5 Examined Architectures

Apart from the different techniques and experiments that were tried and are related to the inputs of the multi-layer perceptron, various architectural schemata of the perceptron were also examined, as described in section 3.3.2. We rewrite here the Table that summarizes all the architectures that we examined (keeping the symbolization of the previous chapter).

	Size of Layers		
	Input Layer	Hidden Layer	Output Layer
Architecture 1	d_{input}	d_{input}	d_{output}
Architecture 2	d_{input}	d_{input} div 2	d_{output}
Architecture 3	d_{input}	d_{input} div 4	d_{output}
Architecture 4	d_{input}	$d_{input}\cdot 2$	d_{output}
Architecture 5	d_{input}	$d_{input}\cdot 2$ div 3	d_{output}
Architecture 6	d_{input}	$d_{input}\cdot 4$	d_{output}
Architecture 7	d_{input}	$(d_{input}+d_{output})\cdot 3$ div 2	d_{output}
Architecture 8	d_{input}	$(d_{input}+d_{output})\cdot 2$ div 3	d_{output}

Table 8: Overview of the examined architectures

4.6 Results

4.6.1 Results for the Dataset balanced_referendum_ds

4.6.1.1 Examined Architectures

The following Table summarizes the dimensions of the layers of the multi-layer perceptron that was created for the dataset *balanced_referendum_ds*.

	Size of Layers		
	Input Layer	**Hidden Layer**	**Output Layer**
Architecture 1	26	26	3
Architecture 2	26	13	3
Architecture 3	26	6	3
Architecture 4	26	52	3
Architecture 5	26	17	3
Architecture 6	26	104	3
Architecture 7	26	43	3
Architecture 8	26	19	3

Table 9: Overview of the examined architectures of the multi-layer perceptron for the dataset balanced_referendum_ds.

4.6.1.2 Summarized Results

For the dataset *balanced_referendum_ds*, we tested our implementation under all the architectural schemata (1, 2, 3, 4, 5, 6, 7 and 8) and for both experiments, with the technique of unigrams. After 10 repetitions, we gathered the mean values of the MSE for every architectural schema. These results are presented in the following Table.

After the pre-processing steps, the final number of tweets in our dataset is 1,830, out of them 1,373 are used for the training and 457 are used for the testing of our system (75% and 25%, respectively, of the final corpus).

Architecture	MSE	MSE (@ number of epoch)	
		Unigrams	
		Experiment	
		1	2
1	Minimum	0.1485 @ 708	0.1468 @ 694
1	Average	0.1905	0.1909
1	Maximum	0.5138 @ 98	0.5684 @ 126
2	Minimum	0.1455 @ 692	0.1485 @ 641
2	Average	0.1657	0.1695
2	Maximum	0.3858 @ 103	0.3585 @ 233
3	Minimum	0.1474 @ 264	0.1488 @ 242
3	Average	0.1572	0.1621
3	Maximum	0.2488 @ 89	0.2844 @ 122
4	Minimum	0.1474 @ 788	0.1470 @ 697
4	Average	0.2391	0.2485
4	Maximum	0.7964 @ 336	0.8472 @ 336
5	Minimum	0.1486 @ 691	0.1450 @ 845
5	Average	0.1765	0.1716
5	Maximum	0.4237 @ 132	0.4538 @ 147
6	Minimum	0.1457 @ 643	0.1478 @ 509
6	Average	0.4513	0.4557
6	Maximum	3.9719 @ 11	4.5260 @ 32
7	Minimum	0.1477 @ 748	0.1472 @ 592
7	Average	0.2210	0.2209
7	Maximum	0.7313 @ 117	0.6551 @ 92
8	Minimum	0.1476 @ 887	0.1451 @ 795
8	Average	0.1758	0.1741
8	Maximum	0.4825 @ 149	0.4483 @ 104

Table 10: Summarized results of dataset balanced_referendum_ds, for experiments 1 and 2, with the technique of unigrams.

The indicative running times for architecture 6 (which is the most complex architecture) for the first and second experiments are 201m and 203m respectively.

We noticed that the neural network converges to a minimum MSE with a mean value of 0.147 for all architectures. Thus, we started the testing phase of the system after this criterion was satisfied; its convergence to the optimal MSE value for each architecture. After 10 repetitions for the tests of each architectural schema and each experiment, we concluded to the following results (Table 6).

		Unigrams					
		Experiment 1			**Experiment 2**		
Architecture	**Total number of predictions**	**Number of correct predictions**	**Number of wrong predictions**	**Success rate**	**Number of correct predictions**	**Number of wrong predictions**	**Success rate**
1	457	302	155	66.08%	310	147	67.83%
2	457	314	143	68.71%	303	154	66.30%
3	457	320	137	**70.02%**	322	135	**70.46%**
4	457	295	162	64.55%	307	150	67.18%
5	457	313	144	68.49%	306	151	66.96%
6	457	311	146	68.05%	303	154	66.30%
7	457	300	157	65.65%	305	152	66.74%
8	457	301	156	65.86%	306	151	66.96%

Table 11: Summarized results of the predictions of the neural network for the dataset balanced_referendum_ds.

4.6.2 Results for the Dataset various_contents_ds

4.6.2.1 Examined Architectures

The following Table summarizes the dimensions of the layers of the multi-layer perceptron that was created for the dataset *various_contents_ds*.

	Size of Layers		
	Input Layer	**Hidden Layer**	**Output Layer**
Architecture 1	27	27	2
Architecture 2	27	13	2
Architecture 3	27	6	2
Architecture 4	27	54	2
Architecture 5	27	18	2
Architecture 6	27	108	2
Architecture 7	27	43	2
Architecture 8	27	19	2

Table 12: Overview of the examined architectures of the multi-layer perceptron for the dataset various_contents_ds.

4.6.2.2 Summarized Results

For the dataset *various_contents_ds*, we tested our implementation under the architectural schemata 1, 2, 3, 4, 5, 6, 7 and 8, for the first experiment and both the techniques of unigrams and bigrams. Again we tested each architecture for 10 times in order to get the mean value of the MSE. We present the results in the following Table (Table 13).

After the steps of pre-processing, the initial corpus of 2,000 tweets was decreased and the final corpus contains 1,751 tweets. A section of 75% of this corpus, which consists of 1,314 tweets, was used in order to train the system, while a section of 25% of it, which consists of 437 tweets, was used for testing it.

Architecture	MSE	MSE (@ number of epoch)	
		Experiment 1	
		Unigrams	Bigrams
1	Minimum	0.1629 @ 567	0.1789 @ 671
1	Average	0.1739	0.2012
1	Maximum	0.4081 @ 122	0.4532 @ 301
2	Minimum	0.1670 @ 696	0.1751@ 584
2	Average	0.1895	0.1960
2	Maximum	0.3604 @ 459	0.3641 @ 108
3	Minimum	0.1665 @ 578	0.1749 @ 479
3	Average	0.1767	0.1848
3	Maximum	0.2554 @ 306	0.3063 @ 265
4	Minimum	0.1676 @ 763	0.1751 @ 203
4	Average	0.2783	0.2808
4	Maximum	0.7224 @ 63	0.6993 @ 70
5	Minimum	0.1662 @ 74	0.1758 @ 912
5	Average	0.1971	0.2024
5	Maximum	0.3914 @ 676	0.3704 @ 102
6	Minimum	0.1662 @ 320	0.1769 @ 574
6	Average	0.9730	0.5902
6	Maximum	40.1450 @ 13	8.2925 @ 1
7	Minimum	0.1664 @ 998	0.1739 @ 807
7	Average	0.2512	0.2546
7	Maximum	0.6239 @ 855	0.6132 @ 68
8	Minimum	0.1661 @ 303	0.1754 @ 622
8	Average	0.2003	0.2062
8	Maximum	0.4236 @ 639	0.3916 @ 540

Table 13: Summarized results of the dataset various_contents_ds, for experiment 1, with the techniques of unigrams and bigrams.

The indicative running time for architecture 6 (which is the most complex architecture) for unigrams is 116m and for bigrams is 117m.

For the dataset *various_contents_ds* and the technique of unigrams, we notice that the network converges for all architectures to a minimum MSE with a mean value of 0.166. For the technique of bigrams, the corresponding minimum MSE has a mean value of 0.176. Consequently, we started the testing phase of the system after the convergence criterion was satisfied, and the network converged to the optimal MSE for each architecture and technique. After 10 repetitions of the tests of each architectural schema and each technique, we concluded to the mean values of the success rates of the predictions of the network, which are presented in the following Table (Table 14).

		Experiment 1					
		Unigrams			**Bigrams**		
architecture	**Total number of predictions**	**Number of correct predictions**	**Number of wrong predictions**	**Success rate**	**Number of correct predictions**	**Number of wrong predictions**	**Success rate**
1	437	279	178	**63.84%**	222	215	50.80%
2	437	234	203	53.55%	225	212	51.49%
3	437	255	182	58.35%	238	199	54.46%
4	437	237	200	54.23%	235	202	53.78%
5	437	239	198	54.69%	255	182	**58.35%**
6	437	231	206	52.86%	244	193	55.84%
7	437	236	201	54.00%	235	202	53.78%
8	437	226	211	51.72%	229	208	52.40%

Table 14: Summarized results of the predictions of the neural network for the dataset various_contents_ds.

4.7 Discussion

Let's, firstly, discuss about the outcome of our tests for the two separate datasets. We notice that the first dataset (*balanced_referendum_ds*) achieves an overall very satisfactory performance, with a maximum success rate of 70%. On the other hand, the second dataset (*various_contents_ds*) doesn't give such good results; the maximum success rate it achieves is 63.84%. We consider that this great difference between the results of the experiments on our datasets is produced due to an essential characteristic that distinguishes them: the fact that the first dataset contains tweets of a common topic. As a matter of fact, this dataset contains a common dictionary which helps the perceptron correlate words and phrases in a context in a more accurate and massive way, and learn more from its environment. It gets very clear here that the contents of a dataset play a very significant role in the final results. Another factor that may have caused the unsatisfactory performance of the network for the second dataset is its small size. We assume that a dataset that contains opinions of various contents should be quite large in order to give the network the chance to gain knowledge about the way the various dictionary is used and the information it carries.

Let's now explore the results of the experiments for the first dataset: ***balanced_referendum_ds***. We notice that all the architectural schemata for both the experiments converge to a common optimum, for a common minimum MSE, below which our model doesn't seem to be able to walk. The mean value of this minimum MSE is 0.147. However, each architecture that was examined has a different behavior over time, starts with different configurations and reaches its minimum more smoothly or abruptly. The best performance of the experiments that used this dataset is achieved with the 3rd architectural schema, with a success rate of 70% for both experiments. The difference between the success rates of this architecture between the two experiments is very slight, hence it is considered negligible. The simplest probabilistic model would classify tweets of three discrete classes with a success rate of 33.33%. As a matter of fact, our system achieves a very satisfactory improvement to this performance, with a success rate of 70%. The worst performance for experiment 1 is achieved with the 4th architecture, with a success rate of 64.55%, while for the experiment 2 it is achieved with the 2nd and 6th architecture, with a success rate of 66.30% for both. In general, we can say that the architectures for the 2nd experiment present success rates that follow a more smooth distribution than the ones of the 1st experiment. However, four of the architectures that were tried for the 1st experiment (architectures 2, 5 and 6) achieve a better success rate than the corresponding ones of the 2nd experiment. The mean value of the success rates of the 1st experiment is 67.18% while the corresponding mean value of the 2nd experiment is 67.34%; the difference between them is very small. The following plot (fig. 44) presents the distributions of the success rates for the two experiments, for the technique of unigrams, using the dataset *balanced_referendum_ds*.

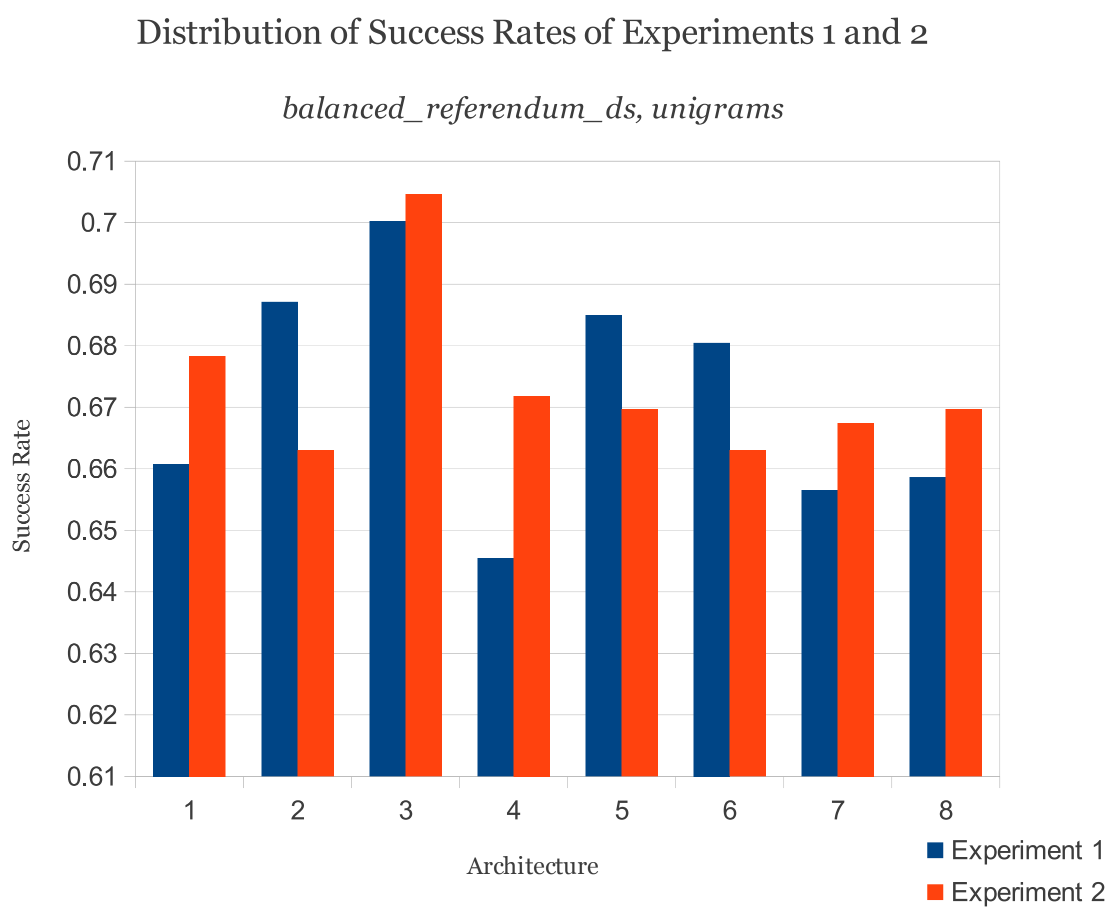


Figure 44: Distribution of success rates of experiments 1 and 2, for the technique of unigrams and the dataset balanced_referendum_ds.

Let's now see the results that were achieved by experiment 1 for the dataset ***various_contents_ds,*** for each of the techniques of unigrams and bigrams, and all the architectures. First of all, as we mentioned before and can be seen in Table 9, all the architectural schemata for each technique converge to an approximately common optimum, a common minimum MSE, the mean value of which is 0.167 for unigrams and 0.176 for bigrams. After the convergence of every configuration of the networks to this minimum value, we started testing the network on the testing dataset. The mean success rate for the technique of unigrams is 55.40%, while the corresponding value for the technique of bigrams is 53.86%. The technique of unigrams achieves an overall better performance than the technique of bigrams. We can assume that this happens because of the fact that the dictionary that is used in the certain dataset is various and is not associated to a common content or topic; hence, as a matter of fact, the phrases and the word sequences that are used might be very different from tweet to tweet and, consequently, the bigrams that are formed are not frequently repeated in order to help the network learn. The best performance of the network for the dataset *various_contents_ds* is achieved for the 1st architecture and the technique of unigrams, with a success rate of

63.84%. Architecture 3 presents a performance that, for the case of unigrams, is the next most satisfactory performance. Regarding the technique of bigrams, the maximum success rate achieved is 58.35% with the 5th architectural schema. The following plot (fig. 45) presents the distribution of the success rates of the experiment 1, for the techniques of unigrams and bigrams and every architecture, for the *dataset various_contents_ds*.

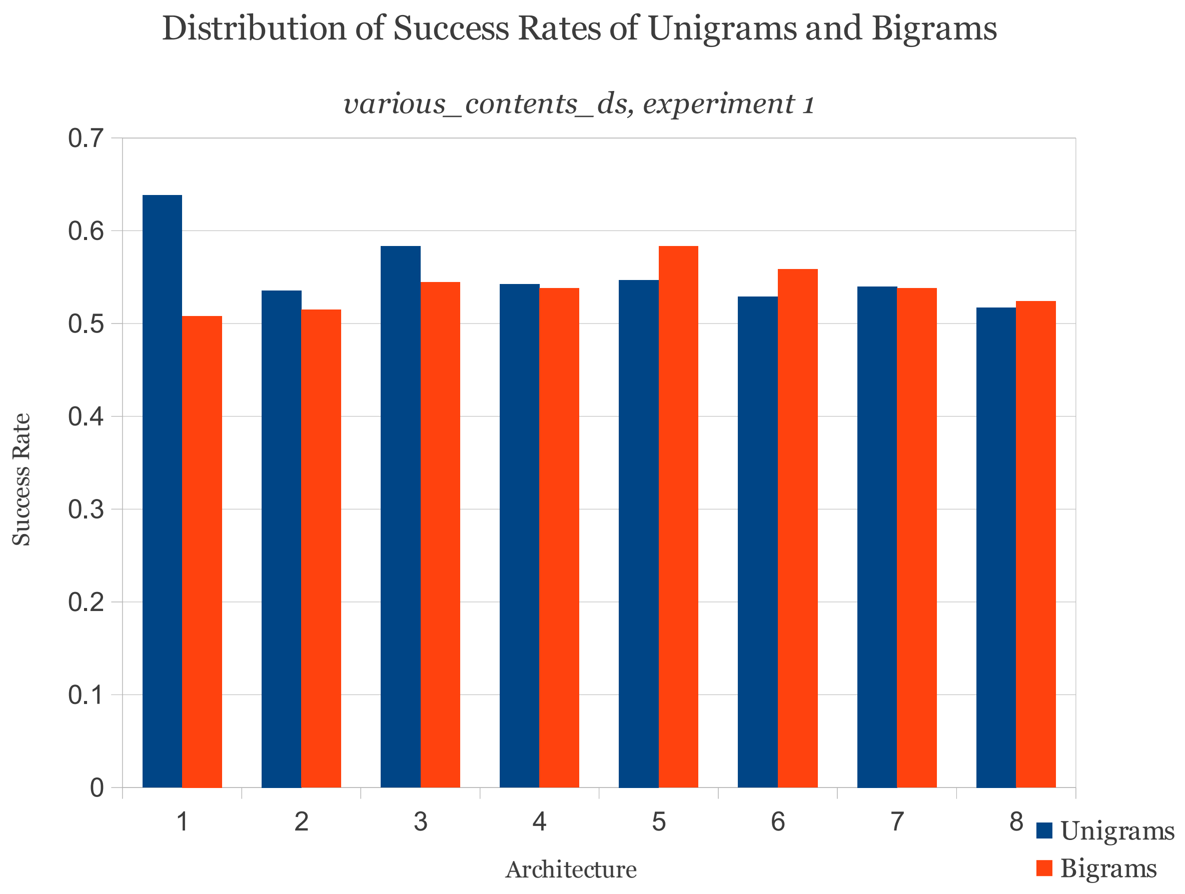


Figure 45: Distribution of success rates of experiment 1, for the techniques of unigrams and bigrams, and the dataset various_contents_ds.

4.8 Ideas for Improvement and Future Directions

In the approaches we examined, we noticed that, out of the two datasets, the network performs much better on the first one, *balanced_referendum_ds*. This dataset differs from the second one, *various_contents_ds,* on the point that it is composed of tweets and opinions about a common topic and content; this particular property of this dataset makes it contain a basic common dictionary, the words of which may be repeated many times from tweet to tweet. This is a big assistance for the neural network, and it helps it find the correlation between words and the sentiment they probably or usually carry, in a more accurate and massive way. It is apparent that the results of our experiments depend very much on the contents of a dataset. Thus, as a matter of fact, we consider that datasets containing opinions of the same topic help the network learn better from its environment and improve itself. We suggest the use of such datasets for the future experiments, or much larger (than the one we used) datasets of various contents.

Another idea for improvement of our model is a modification in the structure of the network. Our model implements a multi-layer perceptron with one hidden layer that applies the hyperbolic tangent function. A modification to the model of the multi-layer perceptron that can appear very promising is its extension to a convolutional network [28] [29], by adding a layer of one of the following three types: convolutional, max-pooling or fully-connected [30].

- Convolutional: Such layers consist of a rectangular grid of neurons. It requires the previous layer also to be a rectangular grid of neurons. Each neuron takes inputs from a rectangular section of the previous layer of neurons; the weights of this rectangular section are the same for each neuron in the convolutional layer. Thus, the convolutional layer is just an image convolution of the previous layer, where the weights specify the convolution filter.

- Max-Pooling: After each convolutional layer, there may be a pooling layer. The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block. There are several ways to do this pooling, such as taking the average or the maximum, or a learned linear combination of the neurons in the block. Our pooling layers will always be max-pooling layers; that is, they take the maximum of the block they are pooling.

- Fully-Connected: Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. A fully connected layer takes all neurons in the previous layer (be it fully connected, pooling, or convolutional) and connects it to every single neuron it has. Fully connected layers are not spatially located anymore (you can visualize them as one-dimensional), so there can be no convolutional layers after a fully connected layer.

Apart from modifying the structure of the network, various forms of inputs can also be examined. The input representations we tried are combinations of the lemmas, stems and POS tags of words, as well as the n-grams. Other representations can also be tried. A simplifying representation that is widely used in natural language processing and information retrieval is the bag-of-words model [31]. In this model, a text (or sentence) is represented as a multiset (bag) of words appearing in the text, disregarding grammar and even word order but keeping multiciplitly.

Furthermore, new features that are considered important can be extracted and taken into account. For instance, the frequency of occurrences of a word in a tweet or in the total corpus, or the position of a word [32] might make a difference (tweets might often have some kind of structure: it's common to begin with a periphrastic expression of an opinion and then finish the tweet by an overall sentiment statement usually part of a hashtag, e.g. *"#angry"*).

Finally, another idea we suggest for the improvement of our model is the implementation of a user profiling system. Such a system will keep information about every user or every user model. This information can be very essential for the task of the sentiment extraction of a tweet. For example, let's imagine a user that usually posts positive tweets about a certain topic. If once he posts an apparently negative tweet, a traditional system that performs sentiment analysis, like ours, would assume that this tweet should be classified as negative, even though it might be a humorous or sarcastic tweet with a positive attitude. Nevertheless, a system that keeps information about the users' profiles will be able to take into account the fact that this particular user usually posts positive tweets about the certain topic, and after a procedure that takes into account the user's statistics and the theory of probabilities, it might finally classify the certain tweet into the positive class.

4.9 Conclusion

In this work we examined the performance of a model for Twitter sentiment analysis, composed of two parts: a part that performs pre-processing tasks for the text refinement and noise cleaning and a part that implements a multi-layer perceptron with one hidden layer. We tried our system onto a corpus of tweets that was divided into two datasets: a dataset of tweets referring to a common topic and expressing sentiments of three discrete classes (positive, negative, neutral), and a dataset of tweets expressing various content of subjective opinions of two classes (positive, negative).

We performed two different experiments; each one of them was based on different combinations of the features that consist the feature vectors: composed of the original words, stems and POS tags, or the original words, lemmas and POS tags. We also examined what difference the performance of unigrams and bigrams could have. In addition, we examined several different architectural schemata for the perceptron.

We noticed that the experiments that used the dataset of opinions of a common topic present a much better performance, with a maximum success rate of 70%, compared to the experiments that used the dataset of various contents, which succeeded a maximum success rate of 63.84%. The architectural schema that achieved the most satisfactory performance for the first dataset and both experiments is the architecture 3, with a success rate of 70% (there is a slight difference between the success rates of architecture 3 of the two experiments but it is so small that is considered negligible). For the second dataset, the architecture 1 achieves the best performance for the technique of unigrams, with a success rate of 63.84%, while the architecture 5 achieves the best performance for the technique of bigrams, with a success rate of 58.35%. In our approaches, the technique of unigrams presents an overall better performance; we consider that the reason for this performing difference is the fact that there is no common dictionary in the dataset *various_contents_ds*, hence each phrase or word sequence is not present in many tweets. Consequently, no more knowledge is offered to the network; actually the network might lose knowledge that it could have gained by the repetition of single words (as in the case of unigrams).

The results of our approaches for the first dataset are very satisfactory. However, we recognize that the results of the second dataset are not good enough and the overall outcome is not optimal. There are other approaches, like support vector machines [32] [22], that seem to be highly competitive on such tasks and are able to provide very good results. Nevertheless, our system based on artificial neural networks can potentially present an equally competitive performance with light modifications or extensions. Our implementation of a strong and self-sufficient model can consist a powerful base, to which a few extensions, such as a convolutional layer in the multi-layer perceptron as mentioned earlier, can make the difference and provide a very satisfactory improvement.

Bibliography

[1] L. Batista, “User Sentiment and Opinion Analysis”, *Encyclopedia of Social Network Analysis and Mining,* Prof. Reda Alhajj, Prof. Jon Rokne, 2014.

[2] G. Lugano, “Extracting Individual and Group Behavior from Mobility Data”, *Encyclopedia of Social Network Analysis and Mining,* Prof. Reda Alhajj, Prof. Jon Rokne, 2014.

[3] “Machine learning”, URL https://en.wikipedia.org/wiki/Machine_learning.

[4] C. Bishop, *Pattern Recognition and Machine Learning,* Springer, 2006.

[5] C. Lin and Y. He, “Sentiment Analysis in Social Media”, *Encyclopedia of Social Network Analysis and Mining,* Prof. Reda Alhajj, Prof. Jon Rokne, 2014.

[6] “Enterprise Mobile Computing news and information.” URL http://searchmobilecomputing.techtarget.com/.

[7] G. Li and K. Chang, “Twitter Microblog Sentiment Analysis”, *Encyclopedia of Social Network Analysis and Mining,* Prof. Reda Alhajj, Prof. Jon Rokne, 2014.

[8] “Microblogging”, 2015, URL https://en.wikipedia.org/wiki/Microblogging.

[9] “Twitter,” 2015, URL https://en.wikipedia.org/wiki/Twitter.

[10] “Twitter, Inc.”, URL https://twitter.com/.

[11] W. Kahle and M. Frotscher, *Color Atlas of Human Anatomy, Vol. 3: Nervous System and Sensory Organs*, 2nd ed. Paschalidis, 2010.

[12] S. Ramon y Cajal, “Cajal’s Degeneration and Regeneration of the Nervous System”, Oxford

University Press, New York, 1991.

[13] R. Mundra and R. Socher, “Deep Learning for Natural Language Processing”, Stanford University, 2015.

[14] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Pearson Education, Inc, 2009.

[15] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice Hall, 1998.

[16] M. A. Fischler and O. Firschein, *Intelligence: The Eye, the Brain and the Computer*. Addison-Wesley, 1987.

[17] T. D. McFarland and R. Parker, *Expert Systems in Education and Training*. Educational Technology, 1990.

[18] *Twitter Application Manager,* URL https://apps.twitter.com/.

[19] L. Bauer, *Introducing Linguistic Morphology*, 2nd ed. Georgetown University Press, 2003.

[20] *Natural Language Toolkit for Python*, URL http://www.nltk.org/.

[21] P. Gamallo and Garcia, Marcos, “Citius: A Naive-Bayes Strategy for Sentiment Analysis on English Tweets”.

[22] S. Wang and C. D. Manning, “Baselines and Bigrams: Simple, Good Sentiment and Topic Classification”.

[23] K. Sheng Tai, “Sentiment Analysis of Tweets: Baselines and Neural Network Models”, 2013.

[24] *Python*, URL https://www.python.org/.

[25] *Twitter API library for Python*, URL http://www.tweepy.org/.

[26] "Pybrain, Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library", URL http://pybrain.org/.

[27] "Artificial Neural Networks/Neural Network Basics", 2015. URL https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Neural_Network_Basics.

[28] C. Nogueira dos Santos and M. Gatti, "Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts".

[29] Y. Kim, "Convolutional Neural Networks for Sentence Classification", New York University.

[30] A. Gibiansky, "Convolutional Neural Networks", URL http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks.

[31] Y. Yoshikawa, T. Iwata, and H. Sawada, "Latent Support Measure Machines for Bag-of-Words Data Classification".

[32] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment Classification using Machine Learning Techniques", presented at the EMNLP 2002, 2002.

[33] A. C. Clarke, *2001: A Space Odyssey,* Hutchinson, 1968.

[34] Q. Jarosz, "Structure of a typical neuron", URL https://en.wikipedia.org/wiki/Axon.

[35] "Brain clip art", www.openclipart.org.

[36] "'Twitter bird in real life' by Scott Hampson", URL https://www.flickr.com/photos/toonz/3677263997.

[37] "GitHub: Where software is built", URL https://github.com/.

Appendix

A. Table of the 100 Most Common Emoticons in Twitter Considered by our Model

:-)	happy	\xF0\x9F\x98\x83	happy
:-}	happy	\xF0\x9F\x98\x86	laugh
;-)	happy	\xF0\x9F\x98\x8A	happy
;-P	ironic	\xF0\x9F\x98\x8D	love
O:-)	angel	\xF0\x9F\x98\x93	tired
\xf0\x9f\x98\x8a	happy	\xF0\x9F\x98\x98	kiss
;-<	sad	\xF0\x9F\x98\x9D	ironic
:)	happy	\xF0\x9F\x98\xA1	angry
;-}	happy	\xF0\x9F\x98\xA4	triumph
:-P	ironic	\xF0\x9F\x98\xA9	weary
;P	ironic	\xF0\x9F\x98\xAD	cry
3:-)	evil	\xF0\x9F\x98\xB2	astonished
\xF0\x9F\x98\x81	happy	\xF0\x9F\x98\xB7	mute
|-{	sad	\xF0\x9F\x98\xBA	happy
:o)	happy	\xF0\x9F\x98\xBD	kiss
:->	happy	\xF0\x9F\x98\x80	scream
:P	ironic	\xE2\x9C\x96	no
<3	love	\xF0\x9F\x92\x94	love
:X	mute	\xF0\x9F\x92\x97	love
:-(	sad	\xF0\x9F\x92\x9A	love
:(	sad	\xF0\x9F\x92\x9D	love
:-|	sad	\xe2\x99\xa5\xe2\x99\xa5	much love
:|	sad	\xE2\x9D\xA4\xE2\x9D\xA4\xE2\x9D\xA4	very much love
|-:	sad	\xF0\x9F\x91\x8D\xF0\x9F\x91\x8D	like
|:	sad	\xE2\x98\x81	cloud
;-(	sad	\xF0\x9F\x98\x84	happy
:-X	mute	\xE2\x9D\xA4	love
\xF0\x9F\x98\x82	laugh	\xF0\x9F\x98\x8B	ironic
\xF0\x9F\x98\x85	laugh	\xF0\x9F\x98\x8F	smirk
\xF0\x9F\x98\x89	happy	\xF0\x9F\x98\x94	sad
\xF0\x9F\x98\x8C	relieved	\xF0\x9F\x98\x9A	kiss
\xF0\x9F\x98\x92	unhappy	\xF0\x9F\x98\x9E	disappointed
\xF0\x9F\x98\x96	confused	\xF0\x9F\x98\xA2	sad
\xF0\x9F\x98\x9C	ironic	\xF0\x9F\x98\xA5	disappointed
\xF0\x9F\x98\xA0	angry	\xF0\x9F\x98\xAA	sleepy
\xF0\x9F\x98\xA3	persevere	\xF0\x9F\x98\xB0	sad
\xF0\x9F\x98\xA8	afraid	\xF0\x9F\x98\xB3	astonished
\xF0\x9F\x98\xAB	tired	\xF0\x9F\x98\xB8	laugh
\xF0\x9F\x98\xB1	scream	\xF0\x9F\x98\xBB	love
\xF0\x9F\x98\xB5	dizzy	\xF0\x9F\x98\xBE	unhappy
\xF0\x9F\x98\xB9	laugh	\xE2\x9C\x8C	victory
\xF0\x9F\x98\xBC	wry	\xF0\x9F\x92\xAA	power
\xF0\x9F\x98\xBF	cry	\xF0\x9F\x92\x95	love
\xE2\x9C\x94	yes	\xF0\x9F\x92\x98	love
\xF0\x9F\x92\x93	love	\xF0\x9F\x92\x9B	love

Table 15: Table of the 100 most common emoticons in Twitter that were considered by our model.

B. Table of 50 out of the 664 Acronyms Considered by our Model

AAMOF	as a matter of fact
ABFL	a big fat lady
ABT	about
ADN	any day now
AFAIC	as far as I’m concerned
AFAICT	as far as I can tell
AFAICS	as far as I can see
AFAIK	as far as I know
AFAYC	as far as you’re concerned
AFK	away from keyboard
AH	asshole
AISI	as I see it
AIUI	as I understand it
AKA	also known as
AML	all my love
ANFSCD	and now for something completely different
ASAP	as soon as possible
ASL	assistant section leader
ASL	age sex location
ASLP	age sex location picture
A/S/L	age/sex/location
ASOP	assistant system operator
ATM	at this moment
AWA	as well as
AWHFY	are we having fun yet?
AWGTHTGTTA	are we going to have to go trough this again?
AWOL	absent without leave
AWOL	away without leave
AYOR	at your own risk
AYPI?	and your point is?
B4	before
B4N	bye for now
BAC	back at computer
BAG	busting a gut
BAK	back at the keyboard
BBIAB	be back in a bit
BBL	be back later
BBLBNTSBO	be back later but not to soon because of
BBR	burnt beyond repair
BBS	be back soon
BBS	bulletin board system
BC	be cool
B/C	because
BCnU	be seeing you
BEG	big evil grin
BF	boyfriend
B/F	boyfriend
BFN	bye for now
BG	big grin
BION	believe it or not

Table 16: Table of 50 out of the 664 acronyms that were considered by our model.

C. Table of the 320 English Stop Words Considered by our Model

a	behind	everything	i	nobody	side	thus	why
about	being	everywhere	ie	none	since	to	will
above	below	except	if	noone	sincere	together	with
across	beside	few	in	nor	six	too	within
after	besides	fifteen	inc	not	sixty	top	without
afterwards	between	fify	indeed	nothing	so	toward	would
again	beyond	fill	interest	now	some	towards	yet
against	bill	find	into	nowhere	somehow	twelve	you
all	both	fire	is	of	someone	twenty	your
almost	bottom	first	it	off	something	two	yours
alone	but	five	its	often	sometime	un	yourself
along	by	for	itself	on	sometimes	under	yourselves
already	call	former	keep	once	somewhere	until	
also	can	formerly	last	one	still	up	
although	cannot	forty	latter	only	such	upon	
always	cant	found	latterly	onto	system	url	
am	co	four	least	or	take	us	
among	computer	from	less	other	ten	very	
amongst	con	front	ltd	others	than	via	
amoungst	could	full	made	otherwise	that	was	
amount	couldnt	further	many	our	the	we	
an	cry	get	may	ours	their	well	
and	de	give	me	ourselves	them	were	
another	describe	go	meanwhile	out	themselves	what	
any	detail	had	might	over	then	whatever	
anyhow	do	has	mill	own	thence	when	
anyone	done	hasnt	mine	part	there	whence	
anything	down	have	more	per	thereafter	whenever	
anyway	due	he	moreover	perhaps	thereby	where	
anywhere	during	hence	most	please	therefore	whereafter	
are	each	her	mostly	put	therein	whereas	
around	eg	here	move	rather	thereupon	whereby	
as	eight	hereafter	much	re	these	wherein	
at	either	hereby	must	same	they	whereupon	
back	eleven	herein	my	see	thick	wherever	
be	else	hereupon	myself	seem	thin	whether	
became	elsewhere	hers	name	seemed	third	which	
because	empty	herself	namely	seeming	this	while	
become	enough	him	neither	seems	those	whither	
becomes	etc	himself	never	serious	though	who	
becoming	even	his	nevertheless	several	three	whoever	
been	ever	how	next	she	through	whole	
before	every	however	nine	should	throughout	whom	
beforehand	everyone	hundred	no	show	thru	whose	

Table 17: Table of the 320 english stop words that were considered by our model.