ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ:

ΕΦΑΡΜΟΓΕΣ ΤΕΧΝΗΤΩΝ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ ΚΑΙ ΜΗΧΑΝΙΚΗΣ ΟΡΑΣΗΣ

Φοιτητής: Παπαφλωράτος Ανδρέας

Εισηγητής: Καθηγητής Αναστάσιος Ι. Ντούνης

ΑΘΗΝΑ 2020

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/~~Η~~ κάτωθι υπογεγραμμένος/η ...Παπαφλωράτος Ανδρέας...,του ...Γερασίμου..., φοιτητής του Τμήματος ...μηχανικών βιομηχανικής σχεδίασης και παραγωγής... του Πανεπιστημίου Δυτικής Αττικής, πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε, ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα, σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασή της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση Π.Ε με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε πρέπει να ολοκληρώσει εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσής της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18. παρ.5 του ισχύοντος Εσωτερικού Κανονισμού».

Ο Δηλών

Α Παπαφλωράτος

Ημερομηνία

02/06/2020

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου Δρ. Αναστάσιο Ντούνη για την βοήθεια, την υπομονή, την υποστήριξη και το ενδιαφέρον που έδειξε κατά την διάρκεια όλης τις πτυχιακής μου εργασίας.

Ένα ευχαριστώ θα ήθελα να πω επίσης και στην οικογένειά μου για την κατανόηση και την υποστήριξη στην διάρκεια της πτυχιακής μου εργασίας.

Περίληψη

Στην συγκεκριμένη πτυχιακή εργασία θα παρουσιαστούν τρόποι αξιοποίησης των θεωριών της Τεχνητής Νοημοσύνης και των τεχνητών νευρωνικών δικτύων. Σκοπός είναι με την χρήση διαφόρων λογισμικών προγραμμάτων και βιβλιοθηκών μέσω της γλώσσας προγραμματισμού Python, να μπορεί ο αναγνώστης να κατανοήσει σε γενικές γραμμές τον τρόπο με τον οποίο γίνεται η αντιστοίχιση θεωρίας με την πράξη. Στο πρώτο μέρος γίνεται αναφορά στο λογισμικό Keras και τον τρόπο να κατασκευάζουμε και να εκπαιδεύουμε τεχνητά νευρωνικά δίκτυα για την επίλυση προβλημάτων.

Στο δεύτερο μέρος θα κατασκευάσουμε ένα σύστημα για στόχευση. Στην βασική οθόνη του προγράμματος θα υπάρχει η εικόνα που μας δίνει η κάμερα στην οποία θα μπορούμε να θέσουμε τον επιθυμητό στόχο, σκοπός θα είναι τα μοτέρ να στραφούν στις σωστές μοίρες έτσι ώστε η δέσμη λέιζερ να συμπέσει με τον επιθυμητό στόχο στην οθόνη. Αυτά θα επιτευχθούν χρησιμοποιώντας Raspberry Pi 3 Model B+, 2 servo κινητήρες σε διάταξη δεξιά-αριστερά και πάνω-κάτω,1 δέσμη λέιζερ, 1 κάμερα, και έναν υπερηχητικό αισθητήρα για μέτρηση αποστάσεων. Όλα αυτά θα είναι σε μια συγκεκριμένη χωροταξία μεταξύ τους. Θα χρησιμοποιήσουμε ένα τεχνητό νευρωνικό δίκτυο με το οποίο θα καταφέρουμε να κάνουμε την γενίκευση, έτσι ώστε οπουδήποτε θέσουμε τον στόχο στην οθόνη να υπολογίζονται οι σωστές μοίρες καθώς και απαλοιφή των χωροταξικών παραμέτρων όπως απόσταση λέιζερ-κάμερας, λέιζερ-μεταλλική βάσης κτλ. που θα ήταν σημαντικοί παράμετροι σε ένα σύστημα υπολογισμού με εξισώσεις. Το τεχνητό νευρωνικό δίκτυο θα έχει ως εισόδους τον στόχο που θέσαμε στην οθόνη συγκεκριμένα η πρώτη είσοδος θα είναι η θέση του στόχου στο X άξονα, η δεύτερη θα είναι η θέση του στόχου στο Y άξονα και η τρίτη η απόσταση του στόχου. Οι έξοδοι θα είναι δύο (μια για το κάθε μοτέρ) όπου θα είναι ένα σήμα από το μηδέν έως το ένα που μεταφράζονται στις μοίρες να πάρει το κάθε μοτέρ αντίστοιχα. Θα γίνει χρήση ενός συστήματος ασαφούς λογικής το οποίο εξυπηρετεί δυο σκοπούς. Ο πρώτος είναι ο υπολογισμός των αληθινών τιμών που θα θέλαμε να υπολογίσει το νευρωνικό δίκτυο, έτσι ώστε να μπορεί να επιτεθεί η εκπαίδευση με εποπτεία. Ο δεύτερος είναι η γεφύρωση επικοινωνίας ανθρώπου με τη μηχανή με τους κανόνες γραμμένους σε ανθρώπινη γλώσσα, αλλά ταυτόχρονα υλοποιήσιμους από την μηχανή.

Summary

In this dissertation will be presented ways to utilize the theories of Artificial Intelligent and Artificial Neural Networks. The aim is to enable the reader to understand in general the way in which the practice theory and practice are combined by using various software programs and libraries with the Python programming language. In the first part we refer to Keras software and how to construct and train artificial neural networks to solve problems.

On the second part we will construct an aiming system. At the main screen of the program there will be the frames of the camera in which the user will be able to set the desired target, the purpose of the system will be to make the motor rotate to the correct degrees, so that the point of the laser match the desired target of the screen. These will be achieved using a Raspberry Pi 3 Model B+, 2 Servo motors in formation left-right and up-down, 1 laser, 1 web camera and one ultrasonic sensor for distance measurement. All these components will be at a specific setup. We will use an artificial neural network in order to achieve generalization, so that regardless were the target will be set the neural network will compute the correct motor positions. With the neural network we also eliminate the hardware setup parameters like distance of laser-camera, laser with metal base etc. which would have a dominating effect in a system that uses equations. Neural network will have as inputs the desired target we set on the screen, in details the first input will be the position of the target on X axis, the second will be the position of the target on Y axis and the third the distance of the target. The outputs will be two (one for each motor) a signal from zero to one indicating the degrees for each motor respectively. We will use a fuzzy system that serves two purposes, the first one is the calculation of the true values which is what we would like the neural network compute, so that we can achieve the supervised learning. The second is the bridging of human to machine communication with the fuzzy rules being written in human language, but at the same time implementable by the machines.

Πίνακας περιεχομένων

Ευχαριστίες 3
Περίληψη 4
Summary 5

Μέρος Πρώτο 9
ΕΙΣΑΓΩΓΗ ΣΤΑ ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΜΕΣΩ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ KERAS. 9
1.1 Εισαγωγή 10
1.2 Ο Τεχνητός Νευρώνας 10
1.2.1 Οι βασικές εξισώσεις του τεχνητού νευρώνα 11
1.2.2 Η συνάρτηση ενεργοποίησης 11
1.2.3 Εισαγωγή στο Keras 13
1.3 Η Εκπαίδευση του Νευρωνικού Δικτύου. 21
1.3.1 Η εκπαίδευση με οπισθοδρόμηση 21
1.3.2 Η συνάρτηση σφάλματος 21
1.3.3 Ο ρυθμός μάθησης 22
1.3.4 Η μέθοδος της οπισθοδρόμησης 22
1.3.5 Ο κανόνας της αλυσίδας 22
1.3.6 Παρτίδες και εποχές 23
1.4 Οπισθοδρόμηση στο Keras. 24
1.4.1 Βασικός κώδικας οπισθοδρόμησης. 25
1.4.2 Ανάλυση αποτελέσματος κώδικα 28
1.4.3 Ομαλοποίηση και κλιμάκωση. 32
1.5 Principal component analysis (PCA) και Autoencoder 32
1.5.1 Autoencoder με Keras, PCA με sklearn. 33
1.5.2 Μετρήσεις από εργοστάσιο χημικής επεξεργασίας 34
1.5.3 Αξιολογήσεις δοκιμών ρυζιού 42

Μέρος Δεύτερο 51
ΕΥΦΥΕΣ ΣΥΣΤΗΜΑ ΣΤΟΧΕΥΣΗΣ 51
2.1 Εισαγωγή 52
2.2 Η υλοποίηση της κατασκευής 53

2.2.1 Η συνδεσμολογία του υπερηχητικού μετρητή απόστασης 53
2.2.2 Η συνδεσμολογία των σέρβο κινητήρων 54
2.2.3 Η συνδεσμολογία του λέιζερ 55
2.3 Η υλοποίηση του λογισμικού 55
2.3.1 Το νευρωνικό δίκτυο 56
2.3.2 Ο έλεγχος των σέρβο κινητήρων 57
2.3.3 Ο έλεγχος του υπερηχητικού μετρητή απόστασης 57
2.3.4 Ο έλεγχος της κάμερας 58
2.3.5 Ο έλεγχος του λέιζερ 58
2.3.6 Το ασαφές σύστημα 58
2.3.7 Η εκπαίδευση του νευρωνικού 60
2.4 Συμπεράσματα. 61
Αναφορές 62
Παράρτημα 63

Λίστα Σχημάτων και Εικόνων

Εικόνα 1: Ο τεχνητός νευρώνας 10
Εικόνα 2: Η σιγμοειδής συνάρτηση κατασκευασμένη με numpy[5] και matplotlib[6]. 11
Εικόνα 3: Η υπερβολική εφαπτομένη κατασκευασμένη με numpy και matplotlib. 12
Εικόνα 4: Η ανορθωμένη γραμμική συνάρτηση κατασκευασμένη με numpy και matplotlib. 12
Εικόνα 5: Η γραμμική συνάρτηση κατασκευασμένη με numpy και matplotlib. 13
Εικόνα 6: Model summary από το Keras. 15
Εικόνα 7: Γραφικά ο νευρώνας της άσκησης. 18
Εικόνα 8: Η τυχαία αρχικοποίηση των βαρών. 20
Εικόνα 9: Γραφικά το νευρωνικό δίκτυο τα βάρη της εικόνας 8. 20
Εικόνα 10: Επιθυμητή συνάρτηση εισόδου - εξόδου 24
Εικόνα 11: Δομή νευρωνικού δικτύου για οπισθοδρόμηση. 25
Εικόνα 12: Αλλαγές στα βάρη από μια οπισθοδρόμηση. 28
Εικόνα 13: Εκπαίδευση με 10 εποχές. 31
Εικόνα 14: Αποτελέσματα νευρωνικού δικτύου σε σχέση με τα επιθυμητά αποτελέσματα. .. 31
Εικόνα 15:Γραφική παράσταση των δεδομένων από μετρήσεις από εργοστάσιο χημικών..... 34
Εικόνα 16:Αποτελέσματα δεδομένων PCA. Το αριστερό γράφημα είναι δεδομένα με ομαλοποίηση. Το δεξιό γράφημα είναι τα δεδομένα με αναστροφή της ομαλοποιήσεις. 38
Εικόνα 17: Εξοδος από PCA χρησιμοποιώντας δύο συνιστώσες και τα αρχικά δεδομένα. 39
Εικόνα 18: Διάγραμμα αυτοκωδικοποιητή. 39
Εικόνα 19: Αποτελέσματα απόδοσης PCA-AE για μετρήσεις από εργοστάσιο χημικών. 41
Εικόνα 20: Αποτελέσματα απόδοσης PCA-AE για αξιολογήσεις δοκιμών ρυζιού. 49
Εικόνα 21: Η κατασκευή. 53

Εικόνα 22: Κύκλωμα συνδεσμολογίας HC-SR04. 54
Εικόνα 23: Κύκλωμα συνδεσμολογίας σέρβο κινητήρων. 55
Εικόνα 24: Κύκλωμα συνδεσμολογίας λέιζερ. 55
Εικόνα 25: Το πρόγραμμα. 56
Εικόνα 26: Συναρτήσεις συμμετοχής του ασαφούς συστήματος. 59

Λίστα Πινάκων

Πινάκας 1: Δεδομένα από μετρήσεις από εργοστάσιο χημικών. 34
Πινάκας 2: Δεδομένα από αξιολογήσεις δοκιμών ρυζιού 41
Πίνακας 3: Πίνακας κανόνων ασαφούς συστήματος. 59

Μέρος Πρώτο

ΕΙΣΑΓΩΓΗ ΣΤΑ ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΜΕΣΩ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ KERAS

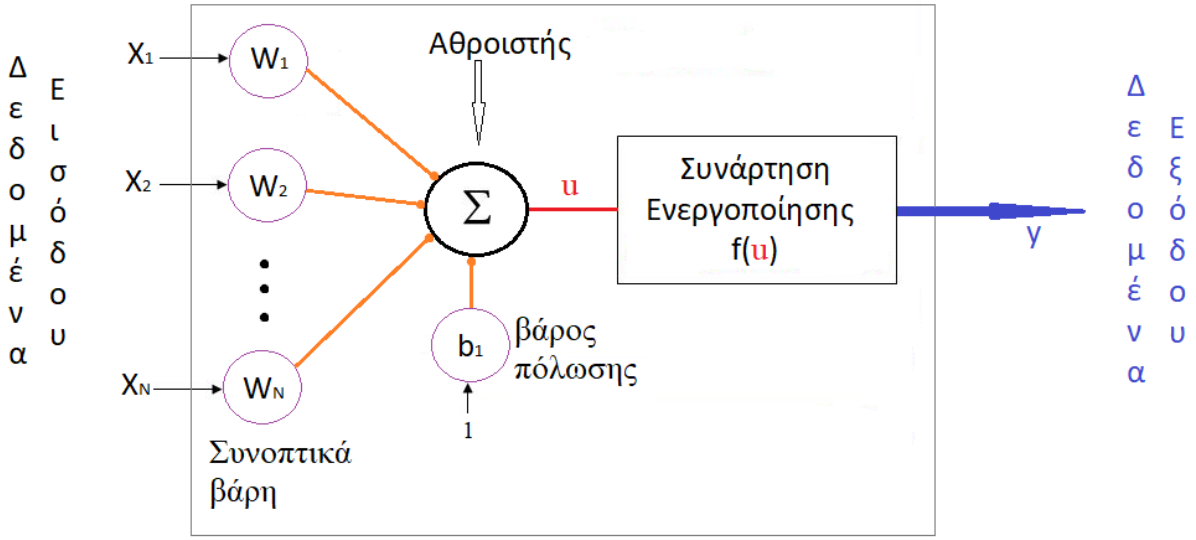
1.1 Εισαγωγή

Όπως και στην φύση ένας βιολογικός νευρώνας απαρτίζεται από κάποια δομικά χαρακτηριστικά έτσι και ο τεχνητός νευρώνας προσπαθεί σε ένα βαθμό να μιμηθεί τον βιολογικό. Σε αυτό το υποκεφάλαιο λοιπόν θα δούμε τα βασικά στοιχεία που απαρτίζουν έναν τεχνητό νευρώνα τόσο στην θεωρία όσο και στην πράξη με το Keras [1].

1.2 Ο Τεχνητός Νευρώνας

Ένας τεχνητός νευρώνας αποτελείται από τέσσερα βασικά στοιχεία:

1. Τα συνοπτικά βάρη, συμβολίζονται με w_{ij}.
2. Το βάρος πόλωσης, συμβολίζεται με b_i.
3. Έναν αθροιστή.
4. Την συνάρτηση ενεργοποίησης συμβολίζεται με f(u).



Εικόνα 1: Ο τεχνητός νευρώνας

Η βασική λειτουργία του νευρώνα είναι να λαμβάνει δεδομένα στην είσοδο όπως αριθμούς ή διανύσματα και να αποδίδει στην έξοδο y τα αποτελέσματα της συνάρτησης ενεργοποίησης που είναι πάλι αριθμοί ή διανύσματα [2].

1.2.1 Οι βασικές εξισώσεις του τεχνητού νευρώνα

Το αποτέλεσμα του αθροιστή στον νευρώνα το συμβολίζουμε ως u και ισχύει ότι:

$$u = \sum_{i=1}^{N} (W_i \cdot X_i) + b_1$$

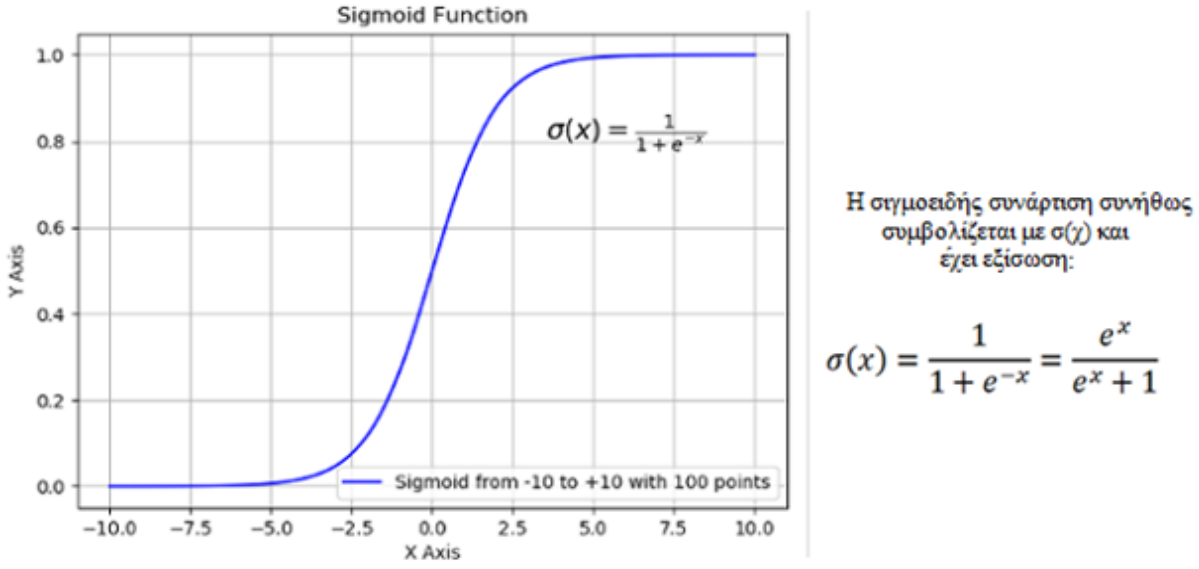
Συνεπώς η έξοδος $y = f(u)$ του τεχνητού νευρώνα χαρακτηρίζεται από την εξίσωση:

$$y = f\left(\sum_{i=1}^{N} (W_i \cdot X_i) + b_1\right)$$

1.2.2 Η συνάρτηση ενεργοποίησης

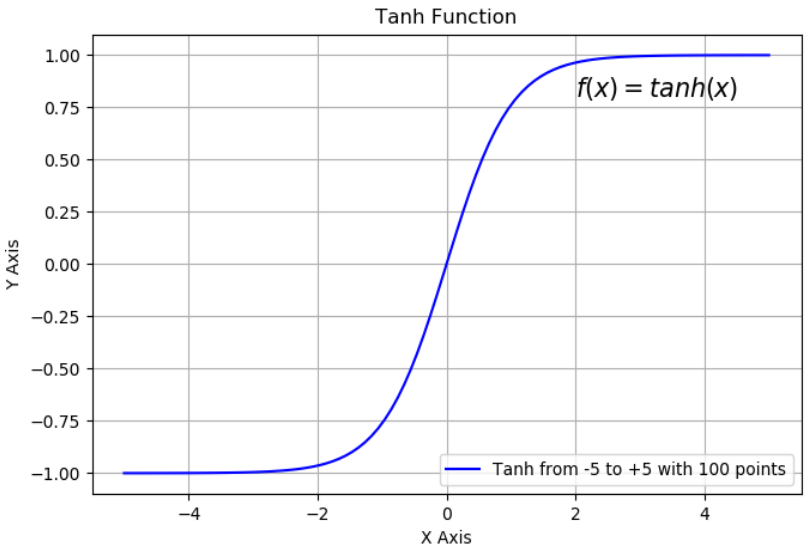
Τα συνοπτικά βάρη ενός τεχνητού νευρώνα είναι η μνήμη του, από την άλλη η συνάρτηση ενεργοποίησης είναι η καρδιά του αφού είναι αυτή που θα καθορίσει τελικά την έξοδο του νευρώνα. Υπάρχουν πολλών ειδών συναρτήσεις ενεργοποίησης f(.) κάποιες από τις πιο διαδεδομένες είναι οι ακόλουθες:

❖ <u>Σιγμοειδής συνάρτηση :</u>



Εικόνα 2: Η σιγμοειδής συνάρτηση κατασκευασμένη με numpy[5] και matplotlib[6].

❖ Υπερβολική εφαπτομένη:

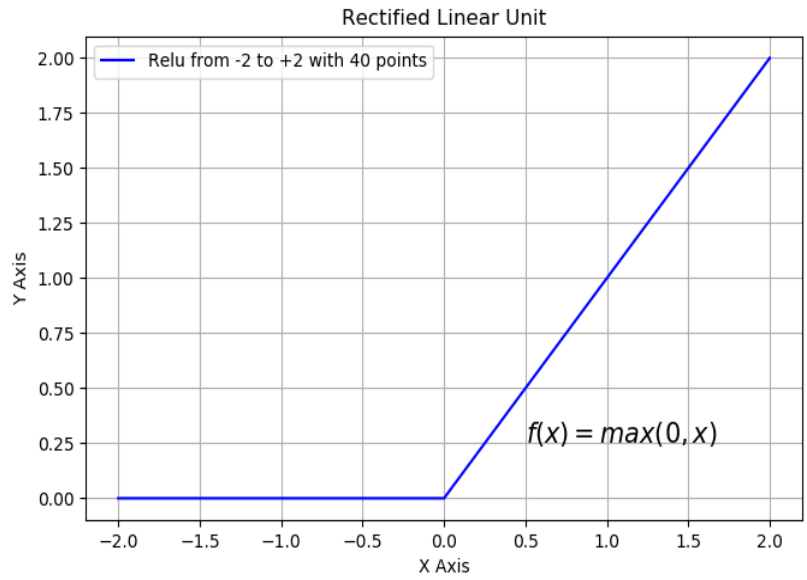


Η συνάρτηση υπερβολικής εφαπτομένης με εξίσωση:

$$f(x) = \tanh(x)$$

Εικόνα 3: Η υπερβολική εφαπτομένη κατασκευασμένη με numpy και matplotlib.

❖ Ανορθωμένη γραμμική συνάρτηση:



Η ανορθωμένη γραμμική συνάρτηση με εξίσωση:

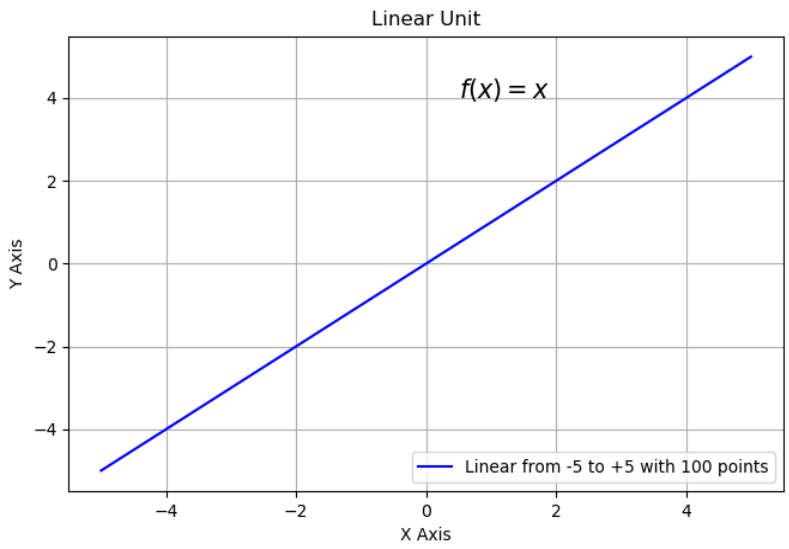
$$f(x) = \max(0, x)$$

ή

$$f(x) = \begin{cases} x \; if \; x > 0 \\ 0 \; if \; x \leq 0 \end{cases}$$

Εικόνα 4: Η ανορθωμένη γραμμική συνάρτηση κατασκευασμένη με numpy και matplotlib.

❖ Γραμμική συνάρτηση:



Η γραμμική συνάρτιση με εξίσωση:

$$f(x) = x$$

Εικόνα 5: Η γραμμική συνάρτηση κατασκευασμένη με numpy και matplotlib.

1.2.3 Εισαγωγή στο Keras

Στο λογισμικό Keras που στο παρασκήνιο θα τρέχει το Tensorflow[7] θα υλοποιήσουμε αρχικά τον νευρώνα. Σε αυτό το σημείο θα ήθελα να επισημάνω ότι το Tensorflow και κατ' επέκταση το Keras λειτουργεί με tensors (πολυδιάστατους πίνακες) που είναι λίγο πολύ παρόμοιοι με τους κλασικούς πινάκες που γνωρίζουμε. Συγκεκριμένα:

- Ένας μονοδιάστατος tensor είναι ένα διάνυσμα π.χ. $\Delta\iota\acute{\alpha}\nu\iota\sigma\mu\alpha = [10\ ,22\ ,34]$ (διάνυσμα μεγέθους 3)
- Ένας δισδιάστατος tensor αποτελείται από μονοδιάστατους tensors και δημιουργούν έναν πινάκα π.χ. $\Pi\acute{\iota}\nu\alpha\kappa\alpha\varsigma = \begin{bmatrix}[10\ ,22\ ,34] \\ [30\ ,99\ ,57]\end{bmatrix}$ (πινάκας μεγέθους 2x3)
- Ένας τρισδιάστατος tensor αποτελείται από δισδιάστατους tensors και δημιουργούν έναν κύβο π.χ. $K\acute{\upsilon}\beta o\varsigma = \left[\begin{bmatrix}[10\ ,22\ ,34] \\ [30\ ,99\ ,57]\end{bmatrix} \quad \begin{bmatrix}[5\ ,32\ ,51] \\ [31,72\ ,7]\end{bmatrix}\right]$ (κύβος μεγέθους 2x2x3)
- Ένας τετραδιάστατος tensor αποτελείται από τρισδιάστατους tensors και δημιουργούν ένα διάνυσμα από τρισδιάστατους tensors π.χ.

$$Διάνισμα\ 4d = \left[\begin{bmatrix} [10,22,34] \\ [30,99,57] \end{bmatrix} \begin{bmatrix} [5,32,51] \\ [31,72,7] \end{bmatrix} \right] \left[\begin{bmatrix} [220,26,84] \\ [350,4,555] \end{bmatrix} \begin{bmatrix} [52,32,561] \\ [331,742,7] \end{bmatrix} \right] \left[\begin{bmatrix} [20,21,35] \\ [70,93,47] \end{bmatrix} \begin{bmatrix} [6,32,541] \\ [61,782,1] \end{bmatrix} \right]$$

Διάνυσμα 4d μεγέθους 3x2x2x3.

Γνωρίζοντας αυτά μπορούμε τώρα να δημιουργήσουμε τον νευρώνα.

Ο κώδικας για την δημιουργία του νευρώνα είναι ο ακόλουθος:

```
#Αρχικά εισάγουμε κάποια στοιχεία από την βιβλιοθήκη του Keras

from keras.layers import Input, Dense

from keras.models import Model

#Εισάγουμε την βιβλιοθήκη numpy από την οποία θα δημιουργούμε τους πινάκες

import numpy as np

#Δημιουργούμε το επίπεδο εισόδου το οποίο θα έχει μορφή για ένα στοιχείο στοιχείο-μια
#διάσταση όπως φαίνεται παρακάτω: (το κόμμα στο (1,) είναι απαραίτητο για να δηλώσουμε
#ότι έχουμε μια διάσταση)

input = Input(shape=(1,))

#Τώρα θα δημιουργήσουμε τον νευρώνα

#Το Dense δηλώνει την πλήρη διασύνδεση (μιας και είναι μόνο ένας ο νευρώνας  και μόνο μια η
#είσοδος έχουμε μια μόνο σύνδεση). Το πρώτο όρισμα του  Dense (στην περίπτωση μας το 1) δηλώνει
#ότι θα έχουμε 1 νευρώνα και μετά το 2ο όρισμα (activation) δηλώνει την συνάρτηση ενεργοποίησης.
#Τέλος μέσα στις τελευταίες παρενθέσεις δηλώνουμε πια θα είναι η μορφή εισόδου(ΟΧΙ ΤΑ
#ΔΕΔΟΜΕΝΑ ΠΟΥ ΘΑ ΠΕΡΑΣΟΥΜΕ ΣΤΗΝ ΕΙΣΟΔΟ) στον συγκεκριμένο νευρώνα, εδώ έχουμε
#βάλει την είσοδο input αλλά θα μπορούσε να είναι και η έξοδος ενός άλλου νευρώνα.

neuronas = Dense(1, activation='linear')(input)

#Τέλος θα "πακετάρουμε" την είσοδο και την έξοδο του  νευρώνα σε ένα μοντέλο. Το Model θέλει ως
#inputs  (1ο όρισμα) την μορφή εισόδου και ως outputs (2ο όρισμα ) την μορφή εξόδου από το
#νευρωνικό μας δίκτυο.

model = Model(inputs=input, outputs=neuronas)
```

```
#Μια από τις πιο χρήσιμες μεθόδους είναι η model.summary() η οποία μας δείχνει αναλυτικά τα
#επίπεδα (layers) του μοντέλου μας. Σε κάθε περίπτωση αν δεν θυμόμαστε τα ορίσματα ή την χρήση
#σε κάποια μέθοδο ή συνάρτηση μπορούμε να χρησιμοποιήσουμε την help() π.χ.

help(model.summary)

#Να σημειωθεί, ότι εδώ χρησιμοποιούμε το model. γιατί αλλιώς δεν θα μπορούσε η Python να βρει
#για πια μέθοδο summary μιλάμε μιας και η συγκεκριμένη είναι του Keras και συγκεκριμένα της
#κλάσης  Model.

#Ας δούμε τώρα το αποτέλεσμα της  model.summary()

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

Εικόνα 6:Model summary από το Keras.

Στην εικόνα 6 αρχικά βλέπουμε ότι έχουμε 2 επίπεδα (Layers) το input_1 και το dense_1. Μετά παρατηρούμε ότι το input_1 έχει 0 παραμέτρους (param #) λογικό αφού είναι απλά μια μορφή. Από την άλλη, το dense_1 έχει 2 παραμέτρους που η μία είναι το συνοπτικό βάρος w_1 και το η άλλη είναι το βάρος πόλωσης b_1. Το Keras μας δίνει στο τέλος το άθροισμα των παραμέτρων από όλα τα επίπεδα.

Όλα φαίνονται σωστά εκτός από τις μορφές εξόδου (Output Shape). Όπως φαίνεται η μορφή εξόδου των tensor και στα δύο επίπεδα έχουν δισδιάστατη μορφή παρόλο που εμείς βάλαμε να έχουν μία διάσταση. Το Keras πάντα προσθέτει μια διάσταση στην τελική μας μορφή. Αυτή η διάσταση είναι το batch size. Το γεγονός ότι είναι None σηματοδοτεί ότι μπορεί να έχει οποιοδήποτε μέγεθος θελήσει ή οποιοδήποτε μέγεθος του ορίσουμε εμείς. Αν και το batch size βγάζει νόημα κυρίως στην εκπαίδευση του νευρωνικού δικτύου (στην συνέχεα θα δοθεί καλύτερος ορισμός), για την ώρα το batch size είναι το πλήθος των δεδομένων που θα περαστούν στην είσοδο. Δηλαδή στο συγκεκριμένο παράδειγμα αν πούμε

ότι έχουμε ως δεδομένα 3 μονοδιάστατους tensors μεγέθους 1 π.χ. $x_1 = [4]$, $x_2 = [6]$, $x_3 = [85]$ ένας τρόπος για να παραστήσουμε όλα αυτά τα δεδομένα με μορφή εισόδου 1 είναι να τα "πακετάρουμε" σε μια διάσταση πιο πάνω από τα δεδομένα μας (εδώ είναι 2d) οπότε θα έχουμε: $δεδομένα = \begin{bmatrix} [4] \\ [6] \\ [85] \end{bmatrix}$ οπότε θα έχουμε μορφή (γραμμές X στήλες) (3 , **1**) στην είσοδο και (3 , **1**) στην έξοδο. Αυτό που μας νοιάζει εδώ είναι να έχουμε ακριβώς το **1,** αφού και οποιοδήποτε άλλο μέγεθος θα έβγαζε σφάλμα ότι δεν του δίνουμε το σωστό μέγεθος πράγμα που είχαμε ορίσει στο επίπεδο εισόδου.

Ας δημιουργήσουμε τώρα αυτά τα δεδομένα με το numpy!

```
#Κάνουμε χρήση του numpy array για να δημιουργήσουμε τα δεδομένα μας .

data=np.array([[4],[6],[85]])

#Αν δεν είμαστε σίγουροι για το τι σχήμα έχει το data μπορούμε να χρησιμοποιήσουμε την
#ιδιότητα shape για να το δούμε.

data.shape

#Ας δούμε τώρα τις δύο παραμέτρους που έχει ο νευρώνας μας. Θα χρησιμοποιήσουμε την
#μέθοδο get_weights() στο 1ο επίπεδο (το επίπεδο 0 είναι η μορφή εισόδου και δεν έχει βάρη
#να πάρουμε από εκεί).

model.layers[1].get_weights()

#Σε αυτό το σημείο να πούμε ότι το συνοπτικό βάρος είναι τυχαίο κάθε φορά και το βάρος
#πόλωσης είναι 0 γιατί δεν έχουμε πει στο μοντέλο να το αρχικοποιήσει.

#Για ευκολία θα αρχικοποιήσουμε τα βάρη με δικιές μας τιμές. Αρχικά ας δούμε τι μορφή
#που έχουν. Μιας και εδώ έχουμε μια λίστα το index 0 είναι το συνοπτικό βάρος και το index
#1 είναι το βάρος πόλωσης οπότε:

model.layers[1].get_weights()[0].shape

# (1,1) μιλάμε για πίνακα

model.layers[1].get_weights()[1].shape
```

```
# (1,) μιλάμε για διάνυσμα

#Αυτό που πρέπει να περάσουμε για να ορίσουμε τα δικά μας βάρη ειναι να ορίσουμε μια
#λίστα με τις παραπάνω μορφές. Οπότε ξεκινάμε φτιάχνοντας μια άδεια λίστα:

new_weights=[]

#Γεμίζουμε την λίστα προσθέτοντας δεδομένα με την μέθοδο append()  η οποία προσθέτει
#στο τέλος της λίστας τα νέα αυτά δεδομένα.

new_weights.append(np.array([[2]], dtype='f')) #Συνοπτικό βάρος =2

new_weights.append(np.array([5], dtype='f')) #Βάρος πόλωσης =5

#Το dtype='f' είναι τύπος δεδομένων και είναι ο float32.

#Τέλος ας περάσουμε τα νέα βάρη στο μοντέλο μας με την μέθοδο set_weights()

model.layers[1].set_weights(new_weights)

#Βεβαιώνουμε ότι έχουν περαστεί σωστά με την get_weights()

model.layers[1].get_weights()


#Έφτασε η ώρα να κάνουμε εμπρός τροφοδότηση στον νευρώνα μας! Θα χρησιμοποιήσουμε
#την μέθοδο predict() όπου το πρώτο όρισμα θα είναι τα δεδομένα μας το δεύτερο όρισμα
#είναι προαιρετικό για να εμφανίσει γραφικά μια μπάρα φόρτωσης.

model.predict(data, verbose=1)
```

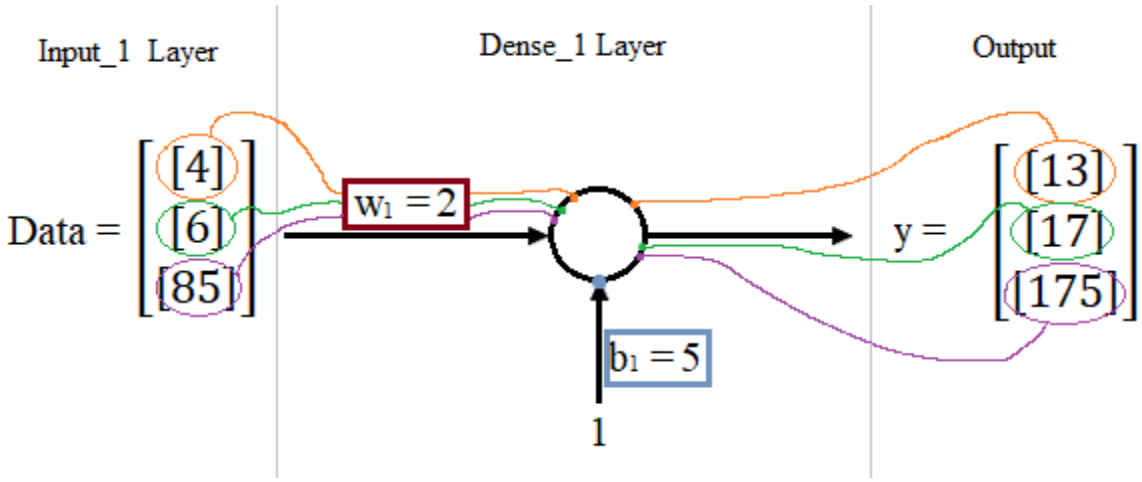
Ας δούμε τώρα την έξοδο και ας δούμε και τι λέει και η θεωρία. Έχουμε το αποτέλεσμα $\begin{bmatrix}[13]\\[17]\\[175]\end{bmatrix}$ με βάση την είσοδο $\begin{bmatrix}[4]\\[6]\\[85]\end{bmatrix}$ και συνάρτηση ενεργοποίησης την γραμμική $f(x) = x$ και αφού έχουμε 1 νευρώνα με διάνυσμα εισόδου μορφής 1 (που σημαίνει ότι θα περνά μια τιμή ανά φορά στο νευρώνα και θα δίνει την έξοδο) ο τύπος $y = f\left(\sum_{i=1}^{N}(W_i * X_i) + b_1\right)$ απλοποιείται σε $y = f(W_i * X_i + b_1)$.

Έχουμε: $y_{[0,0]} = f\left(\left(W_1 \cdot X_{[0,0]}\right) + b_1\right) \Rightarrow y_{[0,0]} = f\left((2 \cdot 4) + 5\right) \Rightarrow y_{[0,0]} = 13$

$$y_{[1,0]} = f\left(\left(W_1 \cdot X_{[1,0]}\right) + b_1\right) \Rightarrow y_{[1,0]} = f\left((2 \cdot 6) + 5\right) \Rightarrow y_{[1,0]} = 17$$

$$y_{[2,0]} = f\left(\left(W_1 \cdot X_{[2,0]}\right) + b_1\right) \Rightarrow y_{[2,0]} = f\left((2 \cdot 85) + 5\right) \Rightarrow y_{[2,0]} = 175$$

Παρατηρούμε ότι τα μεγέθη επαληθεύονται αλλά ας δούμε και γραφικά στην Εικόνα 7 παρακάτω τι κάναμε. Από εδώ και πέρα θα θεωρούμε ότι ο αθροιστής και η συνάρτηση ενεργοποίησης είναι μέσα στον σώμα για να είναι πιο απλά τα σχεδιαγράμματα.



Εικόνα 7: Γραφικά ο νευρώνας της άσκησης.

Ένας διαφορετικός τρόπος για να δημιουργήσουμε το νευρωνικό μας δίκτυο είναι με την συνάρτηση Sequential(). Μπορεί να φαίνεται πιο εύκολος αυτός ο τρόπος δημιουργίας, αλλά δεν προσφέρει την ίδια δυναμική με τον προηγούμενο τρόπο αφού εδώ θέτουμε από την αρχή ότι όλα τα επίπεδα θα είναι το ένα μετά το άλλο σε μια σειρά, ενώ με τον προηγούμενο τρόπο έχουμε την δυνατότητα να δημιουργήσουμε και διακλαδώσεις.

Σε ένα καθαρό Python script έχουμε:

```
#Αρχή κώδικα :

import numpy as np
```

```
from keras.layers import Dense

from keras.models import Model

from keras.models import Sequential


#Εδώ φτιάχνουμε ένα κενό μοντέλο με επίπεδα ακολουθίας.

model = Sequential()

'''Στην συνέχεια αρχίζουμε και προσθέτουμε επίπεδα όπως φαίνετε παρακάτω. Το πρώτο επίπεδο  πρέπει να έχει πάντα και την μορφή εισόδου. Εδώ έχουμε το πρώτο επίπεδο με 3 νευρώνες ,μορφή εισόδου διάνυσμα μεγέθους 2 και συνάρτηση ενεργοποίησης την Relu.'''

model.add(Dense(3, input_shape=(2,),activation='relu'))

''' Προσθέτουμε και ένα δεύτερο επίπεδο με ένα νευρώνα και συνάρτηση ενεργοποίησης την tanh (υπερβολική εφαπτομένη).'''

model.add(Dense(1,activation='tanh'))

#Τσεκάρουμε ότι το μοντέλο μας είναι σωστό με την summary:

model.summary()

#Δημιουργούμε τα δεδομένα εισόδου:

data=np.array([[3,5]])

'''Τσεκάρουμε τα τυχαία βάρη αν θέλουμε μετά να κάνουμε τους υπολογισμούς (για επιβεβαίωση της θεωρίας)'''

model.layers[0].get_weights()

model.layers[1].get_weights()

#Κάνουμε εμπρός τροφοδότηση το νευρωνικό μας δίκτυο.

model.predict(data, batch_size=None, verbose=1, steps=None)
```

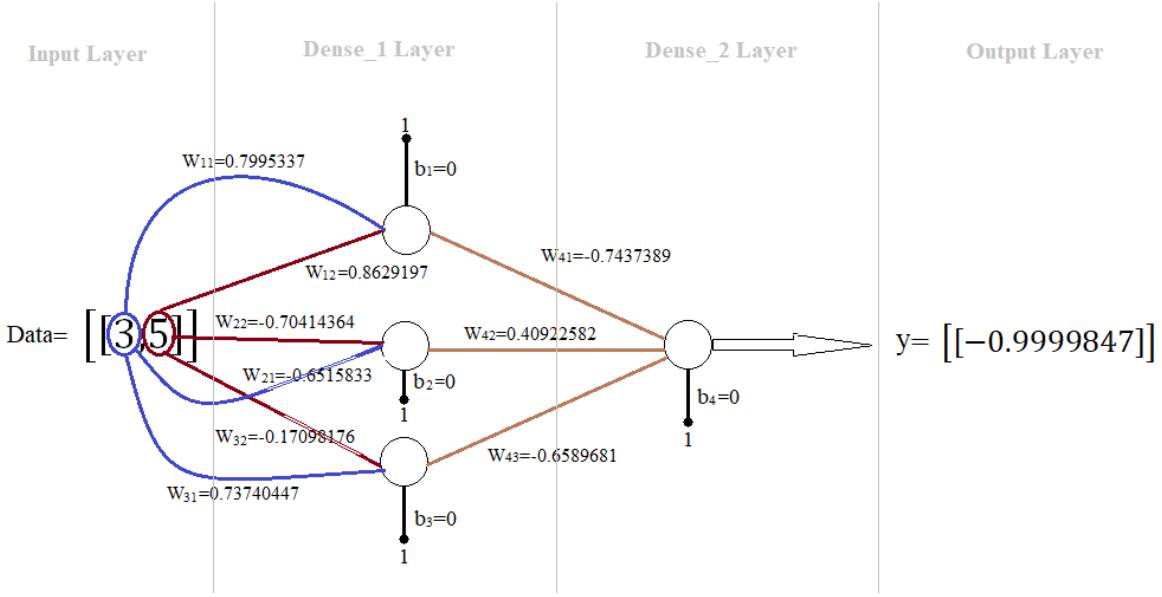
Στο δικό μου μοντέλο τα βάρη τόσο στο πρώτο επίπεδο όσο και στο δεύτερο αρχικοποιήθηκαν τυχαία όπως φαίνετε στην Εικόνα 8.

```
>>> model.layers[0].get_weights()
[array([[ 0.7995337 , -0.6515833 ,  0.73740447],
       [ 0.8629197 , -0.70414364, -0.17098176]], dtype=float32), array([0., 0., 0.], dtype=float32)]
>>> model.layers[1].get_weights()
[array([[-0.7437389 ],
       [ 0.40922582],
       [-0.6589681 ]], dtype=float32), array([0.], dtype=float32)]

>>> model.predict(data, batch_size=None, verbose=1, steps=None)
1/1 [==============================] - 0s 20ms/step
array([[-0.9999847]], dtype=float32)
```

Εικόνα 8: Η τυχαία αρχικοποίηση των βαρών.

Ας δούμε ωστόσο σε αυτό το πιο σύνθετο τεχνητό νευρωνικό δίκτυο τι κάναμε γραφικά στην Εικόνα 9:



Εικόνα 9: Γραφικά το νευρωνικό δίκτυο τα βάρη της εικόνας 8.

1.3 Η Εκπαίδευση του Νευρωνικού Δικτύου.

Στα νευρωνικά δίκτυα αν και ανά καιρούς δημοσιεύονται καινούριες τεχνικές για την εκπαίδευση τους έχοντας τα δικά τους πλεονεκτήματα και μειονεκτήματα, έχει κατά κύριο λόγο κυριαρχήσει η μάθηση με οπισθοδρόμηση (backpropagation) στα νευρωνικά δίκτυα με επίβλεψη (supervised learning).

1.3.1 Η εκπαίδευση με οπισθοδρόμηση

Για την εκπαίδευση του νευρωνικού δικτύου αρχικά κάνουμε την εμπρός τροφοδότηση του δικτύου όπως είδαμε στο προηγούμενο κεφάλαιο. Μετά με το αποτέλεσμα που προκύπτει από το επίπεδο εξόδου και των επιθυμητών αποτελεσμάτων (τα σωστά αποτελέσματα με βάση τις εισόδους μας) υπολογίζουμε με την συνάρτηση σφάλματος (error function / loss function) το ποσό απέχει το αποτέλεσμα του νευρωνικού δικτύου με την αληθινή τιμή. Με βάση το αποτέλεσμα της συνάρτηση σφάλματος, τον ρυθμό μάθησης και με τον κανόνα αλυσίδας υπολογίζονται και ανανεώνονται τα βάρη κάθε σύναψης.

1.3.2 Η συνάρτηση σφάλματος

Αυτό που προσπαθούμε να επιτύχουμε στην εκπαίδευση είναι να ελαχιστοποιήσουμε την συνάρτηση σφάλματος. Μια διαδεδομένη και απλή συνάρτηση σφάλματος είναι η τετραγωνική συνάρτηση σφάλματος. Αυτή έχει τύπο:

$$E = \frac{1}{2}(d - y)^2$$

Όπου:

- y : το αποτέλεσμα του νευρωνικού δικτύου.
- d : η πραγματική τιμή που θέλουμε να δώσει το νευρωνικό με βάση των δεδομένων εισόδων.

Οπότε όταν $d = y$ το σφάλμα E είναι 0 και είναι αυτό που θέλουμε να πετύχουμε.

1.3.3 Ο ρυθμός μάθησης

Ο ρυθμός μάθησης συνήθως συμβολίζεται με η και είναι μια σταθερά. Είναι από τις κυρίαρχες παραμέτρους του νευρωνικού δικτύου αφού είναι αυτός που καθορίζει μια εύκολη και πετυχημένη εκπαίδευση ή μια αποτυχημένη εκπαίδευση. Συγκεκριμένα αν ο ρυθμός μάθησης είναι πολύ μικρός τότε το νευρωνικό δίκτυο θα εκπαιδευτεί πολύ αργά (θα αργήσει να συγκλίνει) ενώ αν είναι πολύ μεγάλος το νευρωνικό δίκτυο δεν θα καταφέρει να συγκλίνει αφού θα δημιουργηθεί μια θετική ανατροφοδότηση εκτοξεύοντας το σφάλμα Ε σε τεράστια νούμερα. Το κάθε νευρωνικό δίκτυο ανάλογα και την δομή του, έχει διαφορετικό ρυθμό μάθησης στον οποίο μπορεί να αποδώσει και δεν υπάρχει κάποιος συγκεκριμένος τρόπος να βρεθεί ο καταλληλότερος αριθμός από την αρχή της εκπαίδευσης και συνήθως τον καθορίζουμε με την μέθοδο δοκιμής και σφάλματος (trial and error).

1.3.4 Η μέθοδος της οπισθοδρόμησης

Πρώτα έχουμε την εμπρός τροφοδότηση στην οποία υπολογίζουμε το αποτέλεσμα του νευρωνικού δικτύου (y), μετά με βάση το (y) και το επιθυμητό αποτέλεσμα (d) υπολογίζουμε το σφάλμα (E). Μετά με το σφάλμα υπολογίζουμε τον κανόνα αλυσίδας και με τις μερικές παραγώγους πόσο πρέπει να αλλάξει το βάρος του κάθε νευρώνα έτσι ώστε να ελαχιστοποιηθεί το σφάλμα. Αυτό αποτελεί την μέθοδο της οπισθοδρόμησης.

1.3.5 Ο κανόνας της αλυσίδας

Αν υποθέσουμε ότι έχουμε έναν νευρώνα στο επίπεδο εξόδου ενός νευρωνικού δικτύου με βάρος εισόδου w_k και ένα σφάλμα E και συνάρτηση ενεργοποίησης $f(x)$ τότε θέλουμε να γνωρίζουμε πόσο το βάρος w_k επηρεάζει το σφάλμα E. Δηλαδή θέλουμε την μερική παράγωγο του σφάλματος E ως προς το βάρος w_k ,δηλαδή $\frac{\partial E}{\partial w_k}$.

Για να υπολογίσουμε ωστόσο αυτή την μερική παράγωγο την σπάμε σε πιο άπλες μερικές παραγωγούς που μπορούμε να υπολογίσουμε το οποίο αποτελεί τον κανόνα της αλυσίδας. Συγκεκριμένα :

$$\frac{\partial E}{\partial w_k} = \frac{\partial E}{\partial f(x)} \cdot \frac{\partial f(x)}{\partial x} \cdot \frac{\partial x}{\partial w_k}$$

Μπορούμε να δούμε ότι οι παρονομαστές και οι αριθμητές σε κάποια από τα κλάσματα απαλείφονται επαληθεύοντας την ισότητα.[2]

Όποτε αρκεί να υπολογίσουμε αυτές τις τρεις μερικές παραγώγους που υπολογίζονται εύκολα για να βρούμε την αρχική μερική παράγωγο. Για να βρούμε την νέα τιμή που θα πάρει το βάρος αρκεί να αφαιρέσουμε την τωρινή τιμή μείον την μερική παράγωγο επί τον ρυθμό μάθησης. Δηλαδή :

$$w_{k+1} = w_k - n \cdot \frac{\partial E}{\partial w_k}$$

Με ανάλογο τρόπο υπολογίζονται και οι τιμές όλων των βαρών. Μόλις υπολογιστούν όλα τα βάρη τότε γίνεται η ανανέωση τους ολοκληρώνοντας την μέθοδο της οπισθοδρόμησης και ξεκινώντας έναν νέο κύκλο ξανά από την αρχή με εμπρός τροφοδότηση κτλ[2].

1.3.6 Παρτίδες και εποχές

Μια εποχή (epoch) είναι όταν κάνουμε εμπρός τροφοδότηση όλου του συνόλου δεδομένων και μετά οπισθοδρόμηση μόνο μια φορά [2]. Από την άλλη η παρτίδα (batch) είναι ένα μέρος από το σύνολο δεδομένων στο οποίο κάνουμε εμπρός τροφοδότηση και στην συνέχεια οπισθοδρόμηση . Αυτό σημαίνει ότι αν κάνουμε εκπαίδευση με παρτίδες για να πούμε ότι ολοκληρώσαμε εκπαίδευση μιας εποχής πρέπει να προσπελάσουμε τα δεδομένα μας χ φορές.

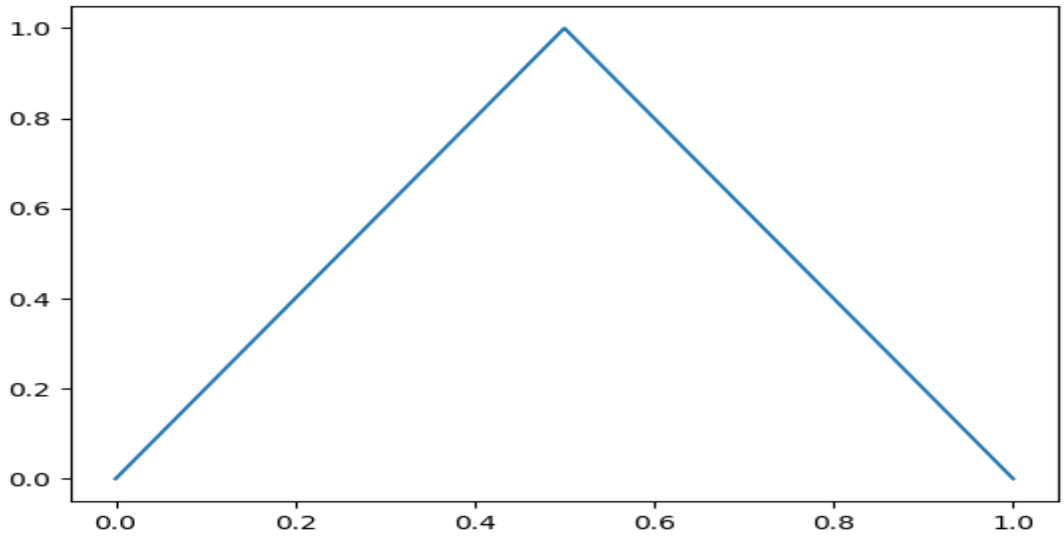
Αν λοιπόν πούμε ότι μια εποχή αποτελείται από 10000 δεδομένα εκπαίδευσης και ότι παίρνουμε παρτίδες των 1000, αυτό σημαίνει ότι θα πρέπει να προσπελάσουμε το νευρωνικό μας 10 φορές (με 1000 παρτίδες) για να ολοκληρώσουμε μια εποχή.

Συνήθως κάνουμε εκπαίδευση με παρτίδες γιατί σε ρεαλιστικά σενάρια έχουμε εκατομμύρια δεδομένα εκπαίδευσης που είναι αδύνατο να τα φορτώσει στην μνήμη τυχαίας προσπέλασης με την πρώτη ένας υπολογιστής. Τέλος συχνά χρειάζεται να προσπελάσουμε τα ίδια δεδομένα εκπαίδευσης πολλές φορές (για πολλές εποχές) μέχρι να ελαχιστοποιήσουμε το σφάλμα σε ένα αποδεκτό βαθμό.

1.4 Οπισθοδρόμηση στο Keras.

Σε αυτό το σημείο θα δούμε πως γίνεται η οπισθοδρόμηση στο Κέρας. Θα φτιάξουμε ένα νευρωνικό δίκτυο το οποίο θα κάνει πρόβλεψη σε μια δικιά μας απλή συνάρτηση.

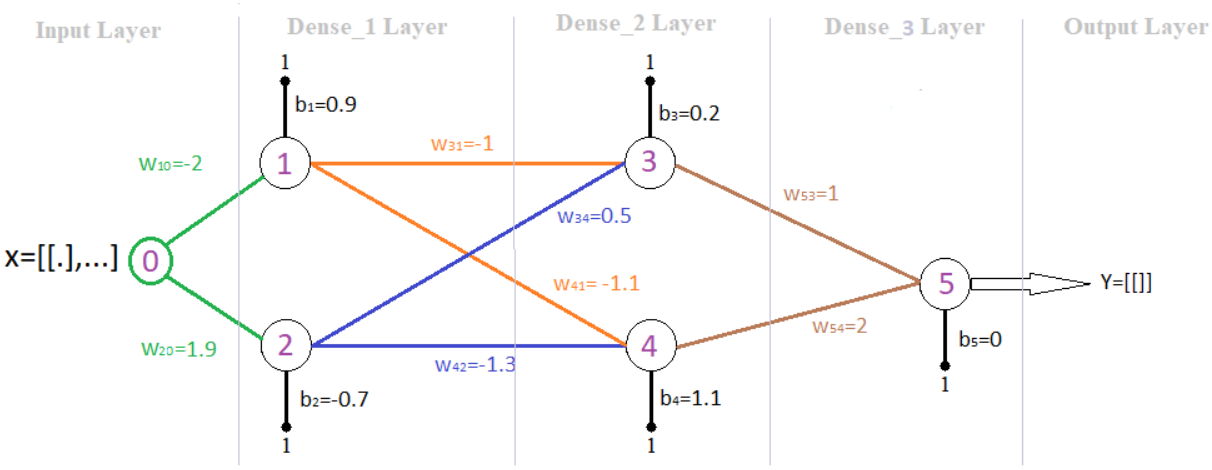
Η συνάρτηση αυτή στο καρτεσιανό επίπεδο έχει την εξής μορφή:



Εικόνα 10: Επιθυμητή συνάρτηση εισόδου - εξόδου

Από το γράφημα προκύπτει ότι για εισόδους $x = [0\ ,0.5\ ,1]$ έχουμε αντίστοιχες εξόδους $y = [0\ ,1\ ,0]$.

Το νευρωνικό μας δίκτυο θα έχει μια είσοδο τον πίνακα χ και μια έξοδο το y. Θα έχει 2 κρυφά επίπεδα από δυο νευρώνες το κάθε ένα και το επίπεδο εξόδου που θα αποτελείται από έναν νευρώνα. Όλοι οι νευρώνες θα είναι πλήρως διασυνδεδεμένοι και θα έχουν συνάρτηση ενεργοποίησης την ανορθωμένη γραμμική συνάρτηση (ReLu). Στο παράδειγμα θεωρούμε το εξής νευρωνικό δίκτυο.



Εικόνα 11: Δομή νευρωνικού δικτύου για οπισθοδρόμηση.

1.4.1 Βασικός κώδικας οπισθοδρόμησης.

Ο κώδικας για να επιτύχουμε την εκπαίδευση:

```
#Αρχή κώδικα  imports:

import numpy as np

import keras

from keras import optimizers

from keras.layers import Dense

from keras.models import Model

from keras.models import Sequential

import matplotlib.pylab as plt

#Δημιουργούμε το τεχνητό νευρωνικό δίκτυο.

model = Sequential()

model.add(Dense(2,input_shape=(1,),activation='relu'))

model.add(Dense(2,activation='relu'))

model.add(Dense(1,activation='relu'))
```

```
#Με την δημιουργία του νευρωνικού δικτύου ρυθμίζονται τυχαία τα βάρη των νευρώνων
#Για το συγκεκριμένο σενάριο θα ρυθμίσουμε σε κάθε επίπεδο τα δικά μας βάρη με τον
#παρακάτω κώδικα.
#set weights for layer 1
l=[]
x=np.array([[-2,1.9]]) #array of weights
y=np.array([0.9,-0.7]) #array of biases
l.append(x)
l.append(y)
model.layers[0].set_weights(l)
#set weights for layer 2
l=[]
x=np.array([[-1,-1.1] ,[0.5,-1.3]]) #array of weights
y=np.array([0.2,1.1])                #array of biases
l.append(x)
l.append(y)
model.layers[1].set_weights(l)

#set weights for layer 3
l=[]
x=np.array([[1],[2]]) #array of weights
y=np.array([0])        #array of biases
```

```
l.append(x)

l.append(y)

model.layers[2].set_weights(l)

#Μπορούμε να δούμε ότι ρυθμίστηκαν σωστά με :

model.layers[0].get_weights()

model.layers[1].get_weights()

model.layers[2].get_weights()


#Ρυθμίζουμε τον τρόπο βελτιστοποίησης σε Stochastic Gradient Descent(SGD)

#Με ρυθμό μάθησης lr=0.1 και τίποτε παραπάνω(decay, momentum, nesterov τα
#απενεργοποιούμε)

sgd = optimizers.SGD(lr=0.1, decay=.0, momentum=.0, nesterov=False)


#Τέλος κάνουμε compile το μοντέλο μας με τα παραπάνω και με συνάρτηση

#βελτιστοποίησης την mse (Mean squared error).

model.compile(loss='mse', optimizer=sgd ,weighted_metrics=None, metrics=['mse'])


#Η εκπαίδευση γίνεται με την  μέθοδο fit η οποία κάνει εμπρός πέρασμα και πίσω πέρασμα

#ανανεώνοντας τα βάρη. Ας δοκιμάσουμε να κάνουμε ένα απλό πέρασμα με είσοδο 0 και

#επιθυμητή έξοδο 0.

#single batch

train_single_batch=np.array([.0])

target_single_batch=np.array([.0])
```

model.fit(x=train_single_batch, y=target_single_batch, batch_size=1, steps_per_epoch=None, verbose=1, shuffle=False)

#Ας δούμε τώρα τα βάρη

model.layers[0].get_weights()

model.layers[1].get_weights()

model.layers[2].get_weights()

```
>>> model.layers[0].get_weights()
[array([[-2. ,  1.9]], dtype=float32), array([ 0.9968, -0.7   ], dtype=float32)]
>>> model.layers[1].get_weights()
[array([[-1.    , -1.1792],
       [ 0.5   , -1.3   ]], dtype=float32), array([0.2  , 1.012], dtype=float32)]
>>> model.layers[2].get_weights()
[array([[1.     ],
       [1.99516]], dtype=float32), array([-0.04400001], dtype=float32)]
```

Εικόνα 12: Αλλαγές στα βάρη από μια οπισθοδρόμηση.

1.4.2 Ανάλυση αποτελέσματος κώδικα

Αρχικά έχουμε την εμπρός τροφοδότηση ως εξής:

$$y_1 = f(u_1) = f(x \cdot w_{10} + b_1) = f(0 \cdot (-2) + 0.9) = f(0.9) = 0.9$$

$$y_2 = f(u_2) = f(x \cdot w_{20} + b_2) = f(0 \cdot 1.9 + (-0.7)) = f(-0.7) = 0$$

$$y_3 = f(u_3) = f(y_1 \cdot w_{31} + y_2 \cdot w_{32} + b_3) = f(0.9 \cdot (-1) + 0 \cdot 0.5 + 0.2) = f(-0.7) = 0$$

$$y_4 = f(u_4) = f(y_1 \cdot w_{41} + y_2 \cdot w_{42} + b_4) = f(0.9 \cdot (-1.1) + 0 \cdot (-1.3) + 1.1) = f(0.11) = 0.11$$

$$y_5 = f(u_5) = f(y_3 \cdot w_{53} + y_4 \cdot w_{54} + b_5) = f(0 \cdot 1 + 0.11 \cdot 2 + 0) = f(0.22) = 0.22$$

Τώρα με έξοδο 0.22 υπολογίζουμε το μέσο τετραγωνικό σφάλμα:

$$e_{mse} = (d - y_5)^2 = (0 - 0.22)^2 = 0.0484$$

Σημείωση: σε πολλά βιβλία μπορεί να δείτε τον τύπο $\frac{(d-y)^2}{2}$ η σταθερά $\frac{1}{2}$ είναι εκεί για βοηθητικούς σκοπούς (στην παραγώγιση απαλείφονται τα 2) και δεν έχει επίπτωση στο αποτέλεσμα αφού συνυπολογίζεται μαζί με το ρυθμό μάθησης. Το Keras δεν προσθέτει το $\frac{1}{2}$.

Η παράγωγος του σφάλματος ως προς το y_5 είναι:

$e'_{mse} = 2 \cdot (d - y_5) = 2 \cdot (0 - 0.22) = 0.44$

Με οπισθοδρόμηση υπολογίζουμε τα (δ_χ) του κανόνα δέλτα. Η παράγωγος της relu είναι:

$$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

$\delta_5 = e'_{mse} \cdot f'(u_5) = -0.44 \cdot 1 = -0.44$

$\delta_4 = f'(y_4) \cdot w_{54} \cdot \delta_5 = 1 \cdot 2 \cdot (-0.44) = -0.88$

$\delta_3 = f'(y_3) \cdot w_{53} \cdot \delta_5 = 0 \cdot 1 \cdot (-0.44) = 0$

$\delta_2 = f'(y_2) \cdot (w_{32} \cdot \delta_3 + w_{42} \cdot \delta_4) = 0 \cdot (-1.1 \cdot 0 + (-1.3) \cdot (-0.88)) = 0$

$\delta_1 = f'(y_1) \cdot (w_{31} \cdot \delta_3 + w_{41} \cdot \delta_4) = 1 \cdot (-1 \cdot 0 + (-1.1) \cdot (-0.88)) = 0.968$

Τέλος υπολογίζουμε τα νέα βάρη:

$w_{53}(new) = w_{53} + n \cdot \delta_5 \cdot y_3 = 1 + 0.1 \cdot (-0.44) \cdot 0 = 1$

$w_{54}(new) = w_{54} + n \cdot \delta_5 \cdot y_4 = 2 + 0.1 \cdot (-0.44) \cdot 0.11 = 1.99516$

$w_{41}(new) = w_{41} + n \cdot \delta_4 \cdot y_1 = -1.1 + 0.1 \cdot (-0.88) \cdot 0.9 = -1.1792$

$w_{42}(new) = w_{42} + n \cdot \delta_4 \cdot y_2 = -1.3 + 0.1 \cdot (-0.88) \cdot 0 = -1.3$

$w_{31}(new) = w_{31} + n \cdot \delta_3 \cdot y_1 = -1 + 0.1 \cdot 0 \cdot 0.9 = -1$

$w_{32}(new) = w_{32} + n \cdot \delta_3 \cdot y_2 = 0.5 + 0.1 \cdot 0 \cdot 0 = 0.5$

$w_{10}(new) = w_{10} + n \cdot \delta_1 \cdot x = -2 + 0.1 \cdot 0.968 \cdot 0 = -2$

$$w_{20}(new) = w_{20} + n \cdot \delta_2 \cdot x = 1.9 + 0.1 \cdot 0 \cdot 0 = 1.9$$

$$b_5(new) = b_5 + n \cdot \delta_5 \cdot 1 = 0 + 0.1 \cdot (-0.44) \cdot 1 = -0.044$$

$$b_4(new) = b_4 + n \cdot \delta_4 \cdot 1 = 1.1 + 0.1 \cdot (-0.88) \cdot 1 = 1.012$$

$$b_3(new) = b_3 + n \cdot \delta_3 \cdot 1 = 0.2 + 0.1 \cdot 0 \cdot 1 = 0.2$$

$$b_2(new) = b_2 + n \cdot \delta_2 \cdot 1 = -0.7 + 0.1 \cdot 0 \cdot 1 = -0.7$$

$$b_1(new) = b_1 + n \cdot \delta_1 \cdot 1 = 0.9 + 0.1 \cdot 0.968 \cdot 1 = 0.9968$$

Παρατηρούμε ότι όλα τα νούμερα επαληθεύονται εκτός από το $b_5(new)$ που το Keras έγραψε -0.04400001. Αυτό οφείλεται στον τρόπο με τον οποίο οι υπολογιστές αναπαριστούν και αποθηκεύουν στο δυαδικό σύστημα τους αριθμούς με υποδιαστολή. Πρακτικά ο τύπος μεταβλητής float32 (4 bytes) που χρησιμοποιεί το Tensorflow για τα βάρη δεν μπορεί να αναπαραστήσει με ακρίβεια τον αριθμό -0.044 στο δυαδικό σύστημα. Αντιθέτως τύποι όπως το Decimal μπορούν να αναπαραστήσουν τέτοιους αριθμούς γιατί έχουν ως βάση δυνάμεις του 10 και όχι του 2.

Τώρα το μόνο που μένει είναι να κάνουμε πολλαπλά περάσματα με οπισθοδρόμηση με τα δεδομένα εισόδου για να εκπαιδεύσουμε το νευρωνικό δίκτυο επιτυχώς. Θα κάνουμε εκπαίδευση για 10 εποχές.

```
target_set=np.array([[.0],[1],[.0]])

train_set=np.array([[.0],[.5],[1]])

model.fit(x= train_set, y= target_set, batch_size=None, epochs=10 , steps_per_epoch=None, verbose=1, shuffle=False)
```

```
>>> model.fit(x=trainSet, y=targetSet, batch_size=None, epochs=10 , steps_per_epoch=None, verbose=1, shuffle=False)
Epoch 1/10
3/3 [==============================] - 0s 3ms/step - loss: 0.3321 - mean_squared_error: 0.3321
Epoch 2/10
3/3 [==============================] - 0s 333us/step - loss: 0.0831 - mean_squared_error: 0.0831
Epoch 3/10
3/3 [==============================] - 0s 324us/step - loss: 0.0421 - mean_squared_error: 0.0421
Epoch 4/10
3/3 [==============================] - 0s 0us/step - loss: 0.0242 - mean_squared_error: 0.0242
Epoch 5/10
3/3 [==============================] - 0s 333us/step - loss: 0.0144 - mean_squared_error: 0.0144
Epoch 6/10
3/3 [==============================] - 0s 0us/step - loss: 0.0087 - mean_squared_error: 0.0087
Epoch 7/10
3/3 [==============================] - 0s 333us/step - loss: 0.0054 - mean_squared_error: 0.0054
Epoch 8/10
3/3 [==============================] - 0s 0us/step - loss: 0.0034 - mean_squared_error: 0.0034
Epoch 9/10
3/3 [==============================] - 0s 333us/step - loss: 0.0021 - mean_squared_error: 0.0021
Epoch 10/10
3/3 [==============================] - 0s 0us/step - loss: 0.0014 - mean_squared_error: 0.0014
```

Εικόνα 13: Εκπαίδευση με 10 εποχές.

Παρατηρούμε ότι το σφάλμα σε κάθε εποχή πέφτει, στην 10 εποχή έχει πέσει στο 0.0014 που είναι πολύ καλό αφού το νευρωνικό μας δίκτυο έμαθε να αφομοιώνει σε μεγάλο βαθμό τα δεδομένα.

Μπορούμε να δούμε τα δεδομένα μας γραφικά με τον παρακάτω κώδικα:

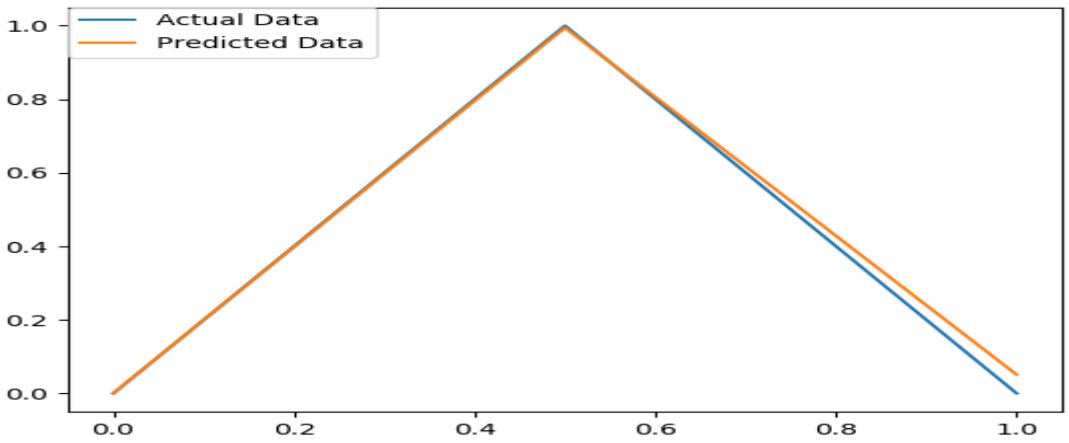
```
output=model.predict(train_set, verbose=1)

plt.plot(train_set, target_set ,label="Actual Data")

plt.plot(train_set, output , label="Predicted Data")

plt.legend(loc='upper left', borderaxespad=0.)

plt.show()
```



Εικόνα 14: Αποτελέσματα νευρωνικού δικτύου σε σχέση με τα επιθυμητά αποτελέσματα.

1.4.3 Ομαλοποίηση και κλιμάκωση.

Στο προηγούμενο παράδειγμα τα δεδομένα εισόδου και εξόδου διακυμαίνονταν ανάμεσα στο 0 και το 1. Σε πραγματικές περιπτώσεις το νευρωνικό δίκτυο θα έχει διάφορες εισόδους με διαφορετικές κλίμακες (πχ. η θερμοκρασία σε έναν κινητήρα να είναι 100 °C ενώ οι στροφές του 3000 rpm) αν αφήσουμε τα δεδομένα μας χωρίς ομαλοποίηση και κλιμάκωση θα έχει ως αποτέλεσμα το νευρωνικό να μαθαίνει πολύ αργά, να γίνεται επικάλυψη των δεδομένων εισόδων (πχ. οι στροφές στον κινητήρα θα έχουν μεγαλύτερη επιρροή στην έξοδο από την θερμοκρασία αφού έχουν φανερά μεγαλύτερες τιμές) ή και την αποτυχία της εκπαίδευσης του νευρωνικού δικτύου.

Τα βάρη στο Keras αρχικοποιούνται με τυχαίες τιμές ανάμεσα στο [-1,1] οπότε είναι λογικό οι είσοδοι/έξοδοι να έχουν μια ομοιόμορφη κλιμάκωση στην οποία θα κυμαίνονται. Για να διαμορφώσουμε τα δεδομένα μας σε μια κοινή κλίμακα [lower,upper] υπάρχει ο τύπος μεγίστου-ελαχίστου:

$$normalization = (\text{upper} - \text{lower}) \cdot \frac{x_{target} - \min(x)}{\max(x) - \min(x)} + \text{lower}$$

Για να αντιστρέψουμε την ομαλοποίηση έχουμε:

$$x_{target} = \frac{(normalization - \text{lower}) \cdot (\max(x) - \min(x))}{(\text{upper} - \text{lower})} + \min(x)$$

1.5 Principal component analysis (PCA) και Autoencoder

Το PCA είναι μια στατιστική διαδικασία που χρησιμοποιεί έναν ορθογώνιο μετασχηματισμό για να μετατρέψει ένα σύνολο παρατηρήσεων πιθανώς συσχετισμένων μεταβλητών σε ένα σύνολο τιμών γραμμικά μη συσχετισμένων μεταβλητών. Αυτός ο μετασχηματισμός ορίζεται με τέτοιο τρόπο ώστε το πρώτο κύριο συστατικό να έχει τη μεγαλύτερη δυνατόν μεταβλητότητα στα δεδομένα και κάθε επόμενο στοιχείο με τη σειρά του έχει την αμέσως μεγαλύτερη δυνατή διακύμανση υπό τον περιορισμό ότι είναι ορθογώνιο με το αμέσως προηγούμενο στοιχείο. Το PCA ορίζεται μαθηματικά ως ένας ορθογώνιος γραμμικός μετασχηματισμός που μετατρέπει τα δεδομένα σε ένα νέο σύστημα συντεταγμένων έτσι ώστε η μεγαλύτερη διακύμανση από κάποια κλιμακωτή προβολή των

δεδομένων να βρίσκεται στην πρώτη συντεταγμένη, η δεύτερη μεγαλύτερη διακύμανση στη δεύτερη συντεταγμένη και ούτω καθεξής [8].

Το Autoencoder είναι ένας τύπος τεχνητού νευρικού δικτύου που χρησιμοποιείται για την εκμάθηση αποτελεσματικών κωδικοποιήσεων των δεδομένων χωρίς επιτήρηση. Ο στόχος του είναι να μάθει να κωδικοποιεί ένα σύνολο δεδομένων, τυπικά για τη μείωση των διαστάσεων, εκπαιδεύοντας το δίκτυο έτσι ώστε να αγνοεί τον θόρυβο του σήματος. Μαζί με την κωδικοποίηση μαθαίνεται και η αποκωδικοποίηση ή αλλιώς η ανακατασκευή των δεδομένων, όπου προσπαθεί να παράγει από την μειωμένη σε διαστάσεις κωδικοποίηση μια αναπαράσταση όσο το δυνατόν πλησιέστερη στην αρχική της εισόδου [9]. Πρόκειται ουσιαστικά για μια συγκεκριμένη αρχιτεκτονική που ξεκινάει με τα δεδομένα , μέσα σε ένα από το κρυφά επίπεδα υπάρχει μια συμφόρηση (που σημαίνει οι νευρώνες στην συμφόρηση είναι λιγότεροι από τις διαστάσεις των δεδομένων στην είσοδο), και στην συνέχεια από αυτό το σημείο ο αποκωδικοποιητής ανακατασκευάζει τα δεδομένα και έχει στην έξοδο τις ίδιες διαστάσεις που είχαν στην αρχή (πριν γίνει η κωδικοποίηση) τα δεδομένα.

1.5.1 Autoencoder με Keras, PCA με sklearn.

Ο σχηματισμός ενός autoencoder (AE) με τεχνητά νευρωνικά δίκτυα αποτελείται από δύο βασικά μέλη έναν κωδικοποιητή που είναι το πρώτο σκέλος του νευρωνικού δικτύου σκοπός του οποίου είναι να συμπιέσει σε λιγότερες διαστάσεις τον αριθμό των δεδομένων εισόδου, το αποτέλεσμα αυτό στην συνέχεια το τροφοδοτούμε σε έναν αποκωδικοποιητή που ουσιαστικά αποσυμπιέσει τα δεδομένα του κωδικοποιητή και προσπαθεί να ανακατασκευάσει τα αρχικά δεδομένα. Σκοπός της εκπαίδευσης είναι να ανακατασκευάσουμε τα δεδομένα εισόδου.

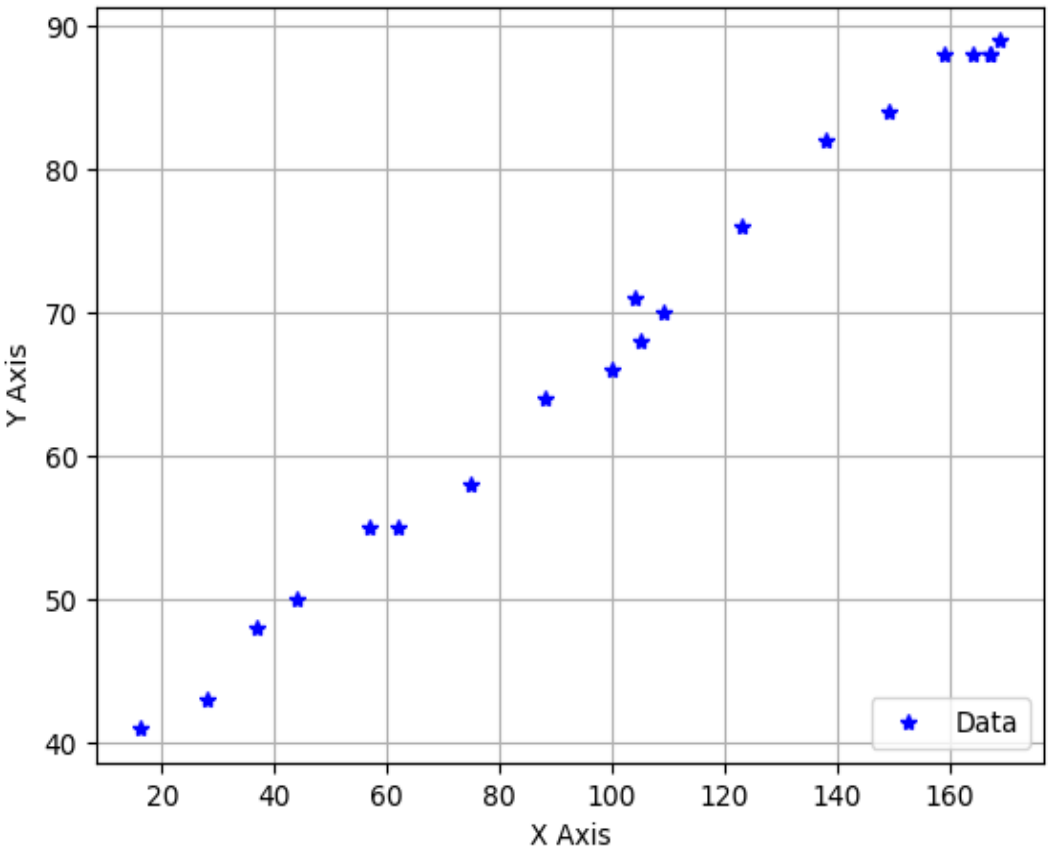
Για να κατασκευάσουμε το PCA θα χρησιμοποιήσουμε την βιβλιοθήκη scikit-learn[10] που μας δίνει έτοιμη την λειτουργικότητα ενός PCA.

Παρακάτω θα γίνει σύγκριση μεταξύ των δυο αυτών τεχνικών σε δύο διαφορετικά σύνολα δεδομένων, με ίδιο αριθμό διαστάσεων μεταξύ των αποκωδικοποιητών στον autoencoder και των αριθμό συνιστωσών στο PCA, έχοντας την ίδια ομαλοποίηση στα δεδομένα.

1.5.2 Μετρήσεις από εργοστάσιο χημικής επεξεργασίας

Η ανεξάρτητη μεταβλητή x είναι ο αριθμός οξέος ενός χημικού προϊόντος, καθοριζόμενο με ογκομέτρηση. Η εξαρτημένη μεταβλητή y είναι η περιεκτικότητα του προϊόντος σε οργανικό οξύ καθοριζόμενο μέσω εξαγωγής και ζύγισης [3]. Παρακάτω θα εφαρμόσουμε ανάλυση κύριων συνιστωσών PCA και ένα δίκτυο αποκωδικοποιητή (Autoencoder - AE) δύο στρωμάτων που υλοποιεί το μετασχηματισμό PCA.

x	y
123	76
109	70
62	55
104	71
57	55
37	48
44	50
100	66
16	41
28	43
138	82
105	68
159	88
75	58
88	64
164	88
169	89
167	88
149	84
167	88



Εικόνα 15:Γραφική παράσταση των δεδομένων από μετρήσεις από εργοστάσιο χημικών.

Πινάκας 1: Δεδομένα μετρήσεων από εργοστάσιο χημικών.

Αρχή κώδικα:

```
import numpy as np
import keras
from keras import optimizers
from keras.layers import Dense
from keras.models import Model
from keras.models import Sequential
import matplotlib.pyplot as plt
#[x,y]

data=np.array([[123,76],[109,70],[62,55],[104,71],[57,55],[37,48],[44,50],
[100,66],[16,41],[28,43],[138,82],[105,68],[159,88],[75,58],[88,64],[164,88],
[169,89],[167,88],[149,84],[167,88]])


norm_data =(data-data.min())/(data.max()-data.min())
model = Sequential()
#Encoder

model.add(Dense(2,input_shape=(2,),activation='linear'))
model.add(Dense(2,activation='linear'))
model.add(Dense(1,activation='linear'))
#Decoder

model.add(Dense(2,activation='linear'))
model.add(Dense(2,activation='linear'))


sgd = optimizers.SGD(lr=0.1, decay=.0, momentum=.0, nesterov=False)
model.compile(loss='mse', optimizer=sgd ,weighted_metrics=None)
model.fit(x=norm_data, y=norm_data,  epochs=200, shuffle=True)
nn_prediction=model.predict(norm_data)


#in order to see the data of hidden layer we create another nn and copy the weights
```

```
model2 = Sequential()
model2.add(Dense(2,input_shape=(2,),activation='linear',weights=model.layers[0].get_weights()))
model2.add(Dense(2,activation='linear',weights=model.layers[1].get_weights()))
model2.add(Dense(1,activation='linear',weights=model.layers[2].get_weights()))
model2.compile(loss='mse', optimizer=sgd ,weighted_metrics=None)
print(model2.predict(norm_data))

#PCA
from sklearn.decomposition import PCA
pca=PCA(n_components=1)
pca_data=pca.fit_transform(norm_data)
pca_predictions=pca.inverse_transform(pca_data)
print(pca.components_)
print(pca.explained_variance_)

s_norm_pca_data = np.sort(pca_predictions,axis=0)
plt.plot(s_norm_pca_data[:,0],s_norm_pca_data[:,1],'*b-',label='PCA output (normalized Data)')
plt.grid()
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.legend(loc='lower right')
plt.show()

rev_data = (pca_predictions-0)*(data.max()-data.min()) + data.min()
s_rev_norm_pca_data = np.sort(rev_data,axis=0)
plt.plot(s_rev_norm_pca_data[:,0],s_rev_norm_pca_data[:,1],'*b-',label='Data in original scale.')
plt.grid()
plt.xlabel('X Axis')
```

```
plt.ylabel('Y Axis')
plt.legend(loc='lower right')
plt.show()
```

Θα δημιουργηθεί μια συνάρτηση στην python που θα υπολογίζει το μέσο τετραγωνικό σφάλμα κάθε μιας εγγραφής από τον πίνακα 1 αλλά και το ολικό μέσο τετραγωνικό σφάλμα όλων των εγγραφών του. Με αυτόν τον τρόπο θα δούμε ποια από τις δύο τεχνικές έχει την καλύτερη απόδοση παρατηρώντας πια έχει το μικρότερο τελικό σφάλμα.

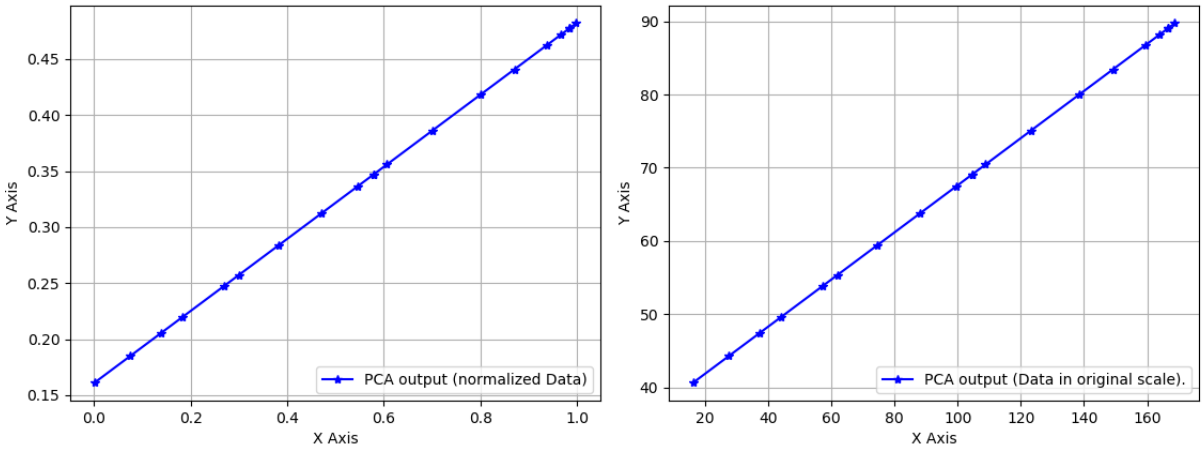
```
def mse(original_array,target_array):
  num_of_rows = target_array.shape[1]
  num_of_columns = target_array.shape[1]
  ret_array=np.array([])
  ret_avg=0
  for i,row in enumerate(target_array):
    error=0
    for j,col in enumerate(row):
      error+= math.pow((col - original_array[i,j]),2)
    ret_array=np.append(ret_array, error/num_of_columns)
  ret_avg = np.average(ret_array)
  return ret_array,ret_avg

pca_array,pac_avg=mse(norm_data,pca_predictions)
nn_array,nn_avg=mse(norm_data,nn_prediction)

pca_avg_arr=np.full((pca_array.shape[0],),pac_avg)
nn_avg_arr=np.full((nn_array.shape[0],),nn_avg)
xaxis = np.arange(0,nn_array.shape[0],1)
plt.plot(xaxis,nn_array,'b-' ,label='nn error')
plt.plot(xaxis,pca_array,'r-' ,label='pca error')
plt.plot(xaxis,nn_avg_arr,'g--',label='nn average error (' + str(nn_avg) + ')')
```

```
plt.plot(xaxis,pca_avg_arr,'m--' ,label='pca average error (' + str(pac_avg) + ')')
plt.legend(loc='upper right',fontsize='xx-large')
plt.xlabel('Data index')
plt.ylabel('MSE')
plt.show()
```

Στο PCA προέκυψαν τα ιδιοδιανύσματα [[-0.95193378 -0.30630389]] ενώ η ιδιοτιμή που προέκυψε είναι [0.12165153].



Εικόνα 16:Αποτελέσματα δεδομένων PCA. Το αριστερό γράφημα είναι δεδομένα με ομαλοποίηση. Το δεξιό γράφημα είναι τα δεδομένα με αναστροφή της ομαλοποιήσεις.

Εάν χρησιμοποιούσαμε δύο συνιστώσες στα αρχικά δεδομένα θα είχαμε ιδιοδιανύσματα:

$$\begin{bmatrix}[-0.95193378 & -0.30630389] \\ [-0.30630389 & 0.95193378]\end{bmatrix}$$

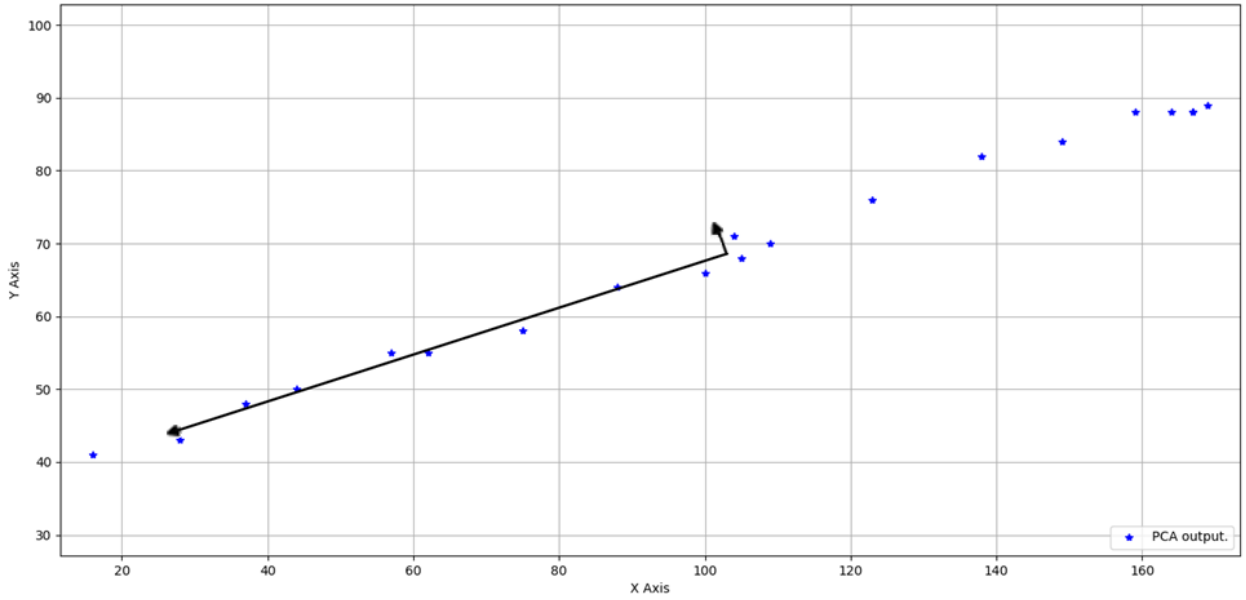
Με ιδιοτιμές :

$$\begin{bmatrix}[2847.74066\ ] \\ [1.29881756]\end{bmatrix}$$

Επειδή η πρώτη ιδιοτιμή (2847.74) είναι πολύ μεγαλύτερη από τη δεύτερη (1.2988), η δεύτερη ιδιοτιμή μπορεί να παραληφθεί και το σύνολο των δεδομένων μπορεί να μετασχηματιστεί σε σύνολο μιας διάστασης. Το πρώτο ιδιοδιάνυσμα $e_1 = [-0.9519, -0.3063]$, που αντιστοιχεί στη μεγαλύτερη ιδιοτιμή, αντιπροσωπεύει μια γραμμή βέλτιστης προσαρμογής. Σε αυτόν τον άξονα τα δεδομένα έχουν τη μεγαλύτερη δυνατή διακύμανση. Το δεύτερο ιδιοδιάνυσμα $e_2 = [-0.9519, -0.3063]$, που αντιστοιχεί

στην πολύ μικρή ιδιοτιμή, αντιπροσωπεύει τα σημεία που βρίσκονται ελαφρώς έξω από τη γραμμή βέλτιστης προσαρμογής e_1.

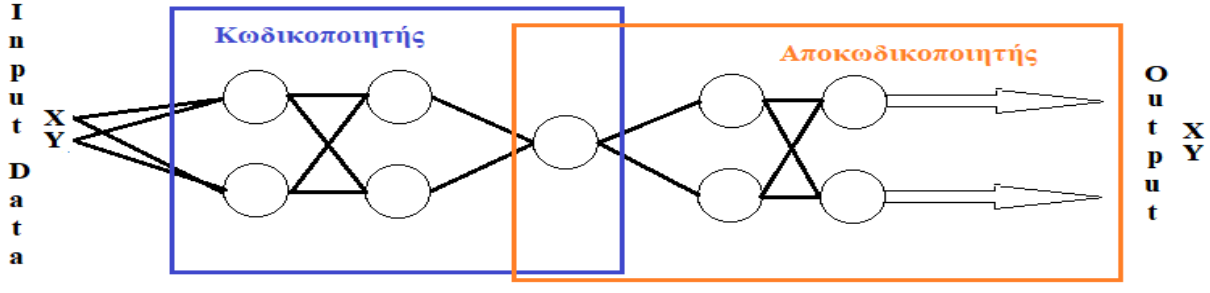
Το τελικό mse: 1.274799222837155e-28 το οποίο είναι παρά πολύ μικρό και σχεδόν αμελητέο αφού πρακτικά πήραμε πίσω τα αληθινά δεδομένα.



Εικόνα 17:Εξοδος από PCA χρησιμοποιώντας δύο συνιστώσες και τα αρχικά δεδομένα.

Στην εικόνα 17 φαίνεται ξεκάθαρα η επιρροή που έχει κάθε μια από τις συνιστώσες στα δεδομένα. Συγκεκριμένα η πρώτη συνιστώσα η οποία έχει και την μεγαλύτερη μεταβλητότητα είναι το μεγάλο βέλος ενώ αυτή που έχει σχεδόν μηδενική μεταβλητότητα είναι η δεύτερη συνιστώσα. Οι δυο συνιστώσες είναι κάθετες μεταξύ τους όπως ορίζεται στον ορισμό του PCA. Αξιοσημείωτο είναι να αναφερθεί ότι η αρχή των βελών είναι στον μέσο ορό των δεδομένων (PCA mean), ενώ το μήκος των βελών είναι η τετραγωνική ρίζα της ιδιοτιμής πολλαπλασιασμένη με το ιδιοδιανύσμα έτσι ώστε να χωρέσουν στο ίδιο διάγραμμα.

Αυτοκωδικοποιητής



Εικόνα 18: Διάγραμμα αυτοκωδικοποιητή.

Στον autoencoder το κρυφό επίπεδο είναι μονοδιάστατα και τα δεδομένα αυτού που προέκυψαν είναι μπορούν να βρεθούν στο παράρτημα.

Τα δεδομένα παραπάνω είναι τα αποτελέσματα του κρυφού επιπέδου του κωδικοποιητή των δύο διαφορετικών νευρωνικών που κατασκευάστηκαν (ένα με συνάρτηση ενεργοποίησης την γραμμική και ένα με την υπερβολική εφαπτομένη). Σε αυτό το σημείο του νευρωνικού δικτύου γίνεται ουσιαστικά η μείωση των διαστάσεων των δεδομένων. Αυτά τα δεδομένα είναι μονοδιάστατα αφού στο τέλος του κωδικοποιητή έχουμε μόνο έναν νευρώνα.

Στην εικόνα 19 παρακάτω φαίνονται τα αποτελέσματα από δύο διαφορετικά τρεξίματα του προγράμματος. Στην πάνω εικόνα έχουμε μέσο τετραγωνικό σφάλμα: PCA $2.63547 \cdot 10^{-5}$ ενώ ο AE $2.683300 \cdot 10^{-5}$. Στην κάτω εικόνα χρησιμοποιήσαμε την υπερβολική εφαπτομένη σε όλους τους νευρώνες, το PCA έχει το ίδιο σφάλμα αφού δεν αλλάζει, από την άλλη ο AE έχει $9.043521 \cdot 10^{-4}$ χειρότερο από όταν είχαμε γραμμικούς νευρώνες. Παρατηρούμε ότι και στις δύο περιπτώσεις το PCA έχει καλύτερη απόδοση από τον AE. Αξιόλογο είναι επίσης να παρατηρήσουμε ότι το σφάλμα που έχει η κάθε γραμμή στην πάνω εικόνα που είχαμε γραμμική πυροδότηση στους νευρώνες έχουν παρόμοια μορφή.

Εικόνα 19: Αποτελέσματα απόδοσης PCA-AE για μετρήσεις από εργοστάσιο χημικών.

1.5.3 Αξιολογήσεις δοκιμών ρυζιού.

Η γενική αξιολόγηση της γεύσης του ρυζιού προκύπτει μετά από αξιολογήσεις από πέντε παραμέτρους: άρωμα και γεύση (x_1), εμφάνιση (x_2), γεύση (x_3), κολλώδες (x_4), σκληρότητα (x_5). Παρακάτω θα εφαρμόσουμε έναν autoencoder με 105 δείγματα ρυζιού που θα αναδημιουργεί αυτά τα δεδομένα με εξίσωση της μορφής $\mathbf{y} = \boldsymbol{f(x_1, x_2, x_3, x_4, x_5)}$.[4]

Άρωμα και Γεύση (flavor) (x_1)	Εμφάνιση (appearance) (x_2)	Γεύση (taste) (x_3)	Κολλώδες (stickiness) (x_4)	Σκληρότητα (toughness) (x_5)	Γενική αξιολόγηση (overall evaluation) (y)
0,523	0,913	1,571	1,6	0	1,784
0,699	1,543	1,76	1,944	-0,875	1,706
-0,720	-2,0220	-1,733	-1,864	1,200	-2,217
-0,309	-0,377	-0,766	-0,96	0,326	-0,875
0,192	0,885	-0,166	-0,106	0,234	0,218
-0,163	-0,094	-0,315	0,083	0,034	-0,655
-0,274	0,044	-0,194	-0,401	1,57	-0,545
0,291	0,984	0,935	0,68	-0,652	1,033
-2,687	-2,853	-3,067	-1,75	2,392	-3,101
-0,704	-0,424	-0,518	-0,779	0,052	-0,652
1,094	1,987	1,613	1,514	-1,156	1,552
-0,247	0,334	0,122	-0,705	0,694	-0,416
-0,673	0,032	-0,838	-0,837	0,354	-0,784
-1,103	-1,046	-1,678	-1,335	-0,46	-1,71
-0,033	0,162	-0,541	-0,159	0,616	-0,529
-0,351	0,119	-0,786	-0,435	0,129	-0,635
0,594	0,266	0,885	1	-0,548	1,105
-0,156	0,173	0,272	0,092	0,163	0,145
-0,256	-0,593	-1,012	-1,403	1,165	-1,901
-0,183	-0,496	-0,77	-1,538	1,78	-1,188
-0,871	-1,27	-1,655	-1,307	1,281	-2,037

-0,060	-0,11	-0,95	-0,719	1,159	-0,788
-1,135	0,163	-0,47	0,285	-0,023	-0,911
-0,609	-0,215	-0,681	-0,534	0,963	-0,612
0,393	0,388	-0,232	0,154	-0,42	-0,127
-2,163	-0,911	-1,61	-1,328	0,145	-1,952
0,831	0,6	0,953	0,953	-0,163	1,141
0,081	1,712	0,946	0,678	-0,257	1,723
0,836	0,799	0,782	1,529	-0,769	1,038
0,214	0,544	0,679	0,314	-0,347	0,51
-0,827	-0,759	-1,012	-0,517	0,197	-0,788
0,221	0,708	0,737	0,923	-0,495	0,899
-0,558	-0,343	-0,331	0,266	-0,756	-0,157
-0,684	-0,33	-0,185	0,349	-0,545	-0,155
-1,816	-0,952	-1,409	-1,882	0,639	-1,783
0,013	1,01	0,927	1,135	-0,387	0,937
0,344	0,856	0,37	0,327	0,227	0,36
-0,323	-0,772	-0,895	-1,002	0,517	-1,286
-0,82	0,608	-0,288	-0,311	0,386	-0,837
0,24	-0,906	-0,715	-1,467	1,626	-1,646
-0,571	-0,151	-0,635	-0,093	0,956	-0,73
0,789	0,821	-0,769	0,428	-1,815	-0,508
-0,353	0,115	-0,394	0,722	-0,089	-0,731
-0,467	-0,622	-0,661	-0,302	0,698	-0,986
-0,491	0,021	-0,221	-0,563	0,385	-0,672
0,848	0,837	1,54	1,112	-0,475	1,113
0,099	0,12	0,387	1,252	-0,638	0,655
-0,738	0,989	0,415	0,574	-0,647	0,093
-0,82	-0,702	1,203	1,579	-1,212	0,483
-0,629	-1,049	-1,271	-1,116	1,637	-1,407
-0,593	-0,898	-0,883	-0,647	0,323	-1,235
0,14	0,181	0,06	0,136	-0,234	0,416
-1,022	-0,827	-1,294	-0,583	0,437	-1,357
-0,649	-0,288	0,257	0,327	-0,412	-0,653

-0,413	0,026	-0,108	0,393	-0,291	0,019
-0,358	-0,106	-1,275	-1,059	0,505	-1,168
-1,825	-2,018	-1,867	-1,517	0,114	-2,136
-0,002	-0,144	0,545	0,674	-0,111	0,641
0,158	0,163	0,03	0,359	-0,128	0,135
0,516	0,73	1,218	1,662	-0,366	1,44
-0,235	-0,566	-0,289	-0,763	0,048	-0,749
-1,085	-0,614	-1,24	-1,12	0,958	-0,65
-1,283	0,09	-0,651	-0,19	-0,131	-1,067
0,25	0,232	0,378	1,983	-1,231	0,519
0,288	0,205	0,147	0,399	0,365	0,352
0,567	1,502	0,824	0,53	-0,387	0,842
0,011	0,26	0,102	0,487	-0,402	0,067
-0,912	-0,911	-1,448	-1,666	1,375	-1,524
-0,588	0,792	1,023	0,995	-0,489	0,901
-0,686	-0,682	-0,514	-0,088	-0,43	-0,466
-2,871	0,59	-0,117	0,195	0,051	-1,084
-1,208	-0,257	-0,426	-0,237	0,24	-0,628
-1,819	-0,856	-1,758	-1,014	1,094	-1,925
0,713	1,172	0,648	1,065	-0,886	1,053
-0,248	0,185	0,204	-0,41	0,588	-0,186
-0,638	0,462	-0,149	-0,214	0,389	-0,23
0,255	0,98	0,837	1,64	0,028	1,747
-1,212	-0,074	-0,317	-0,144	0,18	-0,658
-0,093	0,712	0,594	1,27	-0,236	0,88
-0,315	-0,352	-0,741	-0,288	0,201	-0,964
0,021	0,805	0,367	0,19	0,33	0,056
0,158	0,265	0,053	0,373	-0,101	0,164
0,372	0,388	0,448	0,291	-0,402	0,489
-0,47	-0,301	-0,481	0,173	0,263	-0,546
-0,244	-0,044	-0,466	-0,464	0,469	-0,804
-0,049	0,401	-0,097	0,062	0,282	-0,223
-0,639	0,63	-0,089	0,166	-0,293	-0,394

0,039	0,524	0,207	1,082	-1,3030	0,5370
0,079	1,231	0,373	0,281	-1,273	0,218
0,104	-0,194	0,261	0,838	0,085	0,434
0,475	0,707	-0,172	0,097	-0,247	-0,246
0,468	0,333	0,511	1,188	-0,646	0,78
-0,465	-0,195	-0,491	-0,671	0,837	-0,614
0,125	0,127	-0,264	0,461	0,197	-0,046
-0,491	0,525	0,419	0,78	-0,512	0,291
-1,388	-0,114	-0,894	-0,168	0,028	-0,821
-0,955	-1,146	-1,107	-0,922	0,106	-1,367
-0,255	0,232	0,517	0,398	-0,247	0,424
-0,686	-0,252	-0,749	-0,428	0,048	-0,704
-0,016	0,674	-0,046	0,662	-0,694	0,326
-0,618	-0,425	0,098	0,471	0,158	0,104
0,57	1,542	1,302	1,842	-0,563	1,712
-1,134	-1,281	-1,031	-1,477	1,345	-1,624
0,124	0,588	0,053	-0,405	-0,322	-0,517
0,163	1,088	1,074	0,683	-0,135	0,913

Πινάκας 2: Δεδομένα από αξιολογήσεις δοκιμών ρυζιού.

Αρχή κώδικα:

```
import numpy as np
import keras
from keras import optimizers
from keras.layers import Dense
from keras.models import Model
from keras.models import Sequential

#[x1, x2, x3, x4, x5, y]

#data= . . .τα δεδομένα σε μορφή λίστας είναι στο παράρτημα.
```

```
norm_data =(data-data.min())/(data.max()-data.min())
model = Sequential()
#Encoder
model.add(Dense(6,input_shape=(6,),activation='linear'))
model.add(Dense(6,activation='linear'))
model.add(Dense(2,activation='linear'))
#Decoder
model.add(Dense(6,activation='linear'))
model.add(Dense(6,activation='linear'))

sgd = optimizers.SGD(lr=0.1, momentum=.5, nesterov=False)
model.compile(loss='mse', optimizer=sgd ,weighted_metrics=None)
model.fit(x=norm_data, y=norm_data,  epochs=40000, shuffle=True)
nn_prediction=model.predict(norm_data)

#PCA
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca_data=pca.fit_transform(norm_data)
pca_predictions=pca.inverse_transform(pca_data)
print(pca.components_)
print(pca.explained_variance_)


def mse(original_array,target_array):
  num_of_rows = target_array.shape[1]
  num_of_columns = target_array.shape[1]
  ret_array=np.array([])
  ret_avg=0
  for i,row in enumerate(target_array):
```

```
    error=0
    for j,col in enumerate(row):
      error+= math.pow((col - original_array[i,j]),2)
    ret_array=np.append(ret_array, error/num_of_columns)
  ret_avg = np.average(ret_array)
  return ret_array,ret_avg

pca_array,pac_avg=mse(norm_data,pca_predictions)
nn_array,nn_avg=mse(norm_data,nn_prediction)

pca_avg_arr=np.full((pca_array.shape[0],),pac_avg)
nn_avg_arr=np.full((nn_array.shape[0],),nn_avg)
xaxis = np.arange(0,nn_array.shape[0],1)
plt.plot(xaxis,nn_array,'b-' ,label='nn error')
plt.plot(xaxis,pca_array,'r-' ,label='pca error')
plt.plot(xaxis,nn_avg_arr,'g--',label='nn average error (' + str(nn_avg) + ')')
plt.plot(xaxis,pca_avg_arr,'m--' ,label='pca average error (' + str(pac_avg) + ')')
plt.legend(loc='upper right',fontsize='xx-large')
plt.xlabel('Data index')
plt.ylabel('MSE')
plt.show()
```

Στο PCA προέκυψαν τα ιδιοδιανύσματα:

$$\begin{bmatrix} [-0.2925261 & -0.37920516 & -0.44540011 & -0.46210002 & 0.28932128 & -0.52441151] \\ [-0.59663493 & -0.17185448 & -0.10635058 & 0.33668725 & -0.68792015 & -0.12880184] \end{bmatrix}$$

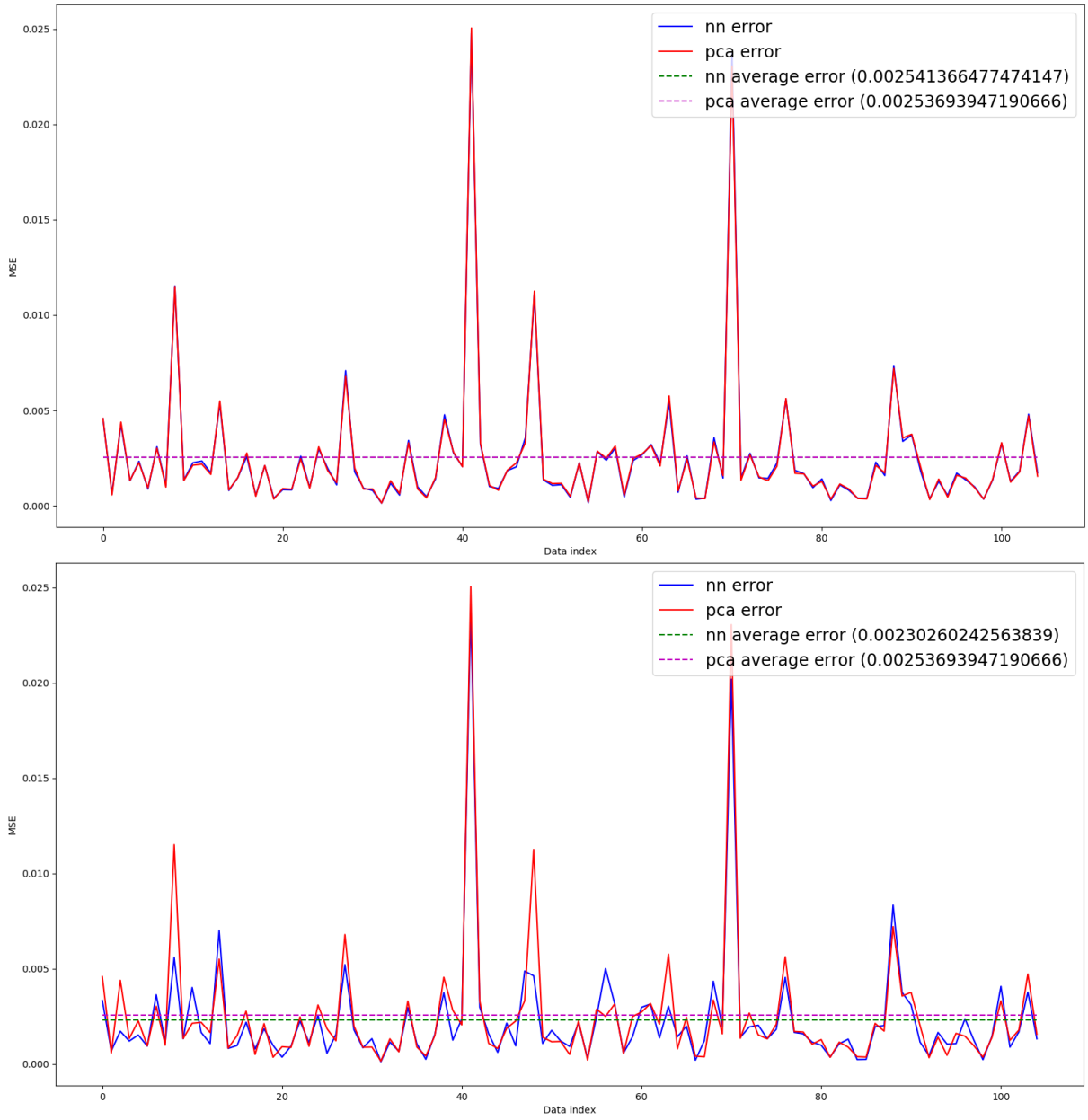
ενώ οι ιδιοτιμές που προέκυψε είναι [0.11806243 0.01090564].

Στον autoencoder το κρυφό επίπεδο έχει δύο διάσταση και τα δεδομένα αυτού που προέκυψαν μπορούν να βρεθούν στο παράρτημα.

Τα δεδομένα παραπάνω είναι τα αποτελέσματα του κρυφού επιπέδου του κωδικοποιητή των δύο διαφορετικών νευρωνικών που κατασκευάστηκαν (ένα με συνάρτηση ενεργοποίησης την γραμμική και ένα με την υπερβολική εφαπτομένη). Σε αυτό το σημείο του νευρωνικού δικτύου γίνεται ουσιαστικά η μείωση των διαστάσεων των δεδομένων. Αυτά τα δεδομένα έχουν δυο διαστάσεις αφού στο τέλος του κωδικοποιητή έχουμε δύο νευρώνες.

Έγιναν δύο δοκιμές. Στην εικόνα 20 φαίνονται τα αποτελέσματα. Αυτό που αλλάζει είναι το μέσο τετραγωνικό σφάλμα του νευρωνικού δικτυού με διαφορετικές παραμέτρους κάθε φορά, το σφάλμα του PCA είναι σταθερό και τις δυο φορές στα 0.00253693. Στην εικόνα 20 στο πάνω γράφημα έχουμε MSE 0.00254136, παρατηρούμε ότι είναι σαν να προσπαθεί το νευρωνικό να προσεγγίσει το σφάλμα του PCA. Στο κάτω γράφημα αλλάζοντας την συνάρτηση ενεργοποίησης όλων των νευρώνων από γραμμική σε υπερβολική εφαπτομένη έχουμε σφάλμα 0.00230260 ο AE έχει καλύτερη απόδοση από το PCA.

Μπορούμε να παρατηρήσουμε ότι στο πάνω γράφημα το MSE ανά γραμμή δεδομένων είναι σχεδόν ίδια, με το που αλλάζουμε την συνάρτηση ενεργοποίησης από γραμμική σε υπερβολική εφαπτομένη τότε αρχίζουν να αλλάζουν οι μορφές του σφάλματος μεταξύ τους. Αυτό συμβαίνει διότι το PCA μαθαίνει από γραμμικές συναρτήσεις οι οποίες μεταφέρουν τα δεδομένα σε ένα εναλλακτικό χώρο, αυτό καθιστά δύσκολο στο PCA να εφαρμόσει δεδομένα τα οποία έχουν καμπύλες δηλαδή συναρτήσεις βαθμού μεγαλυτέρου ή και ίσου του δύο. Από την άλλη ο AE έχει την δυνατότητα να μάθει αυτά τα μη γραμμικά δεδομένα εύκολα αφού μπορεί να ρυθμιστεί με πληθώρα από επίπεδα και συναρτήσεις ενεργοποίησης.



Εικόνα 20: Αποτελέσματα απόδοσης PCA-AE για αξιολογήσεις δοκιμών ρυζιού.

Έγιναν δοκιμές με όλα ιδία εκτός από τον αριθμό νευρώνων στο τέλος του κωδικοποιητή και τον αριθμό των συνιστωσών στο PCA. Συγκεκριμένα προέκυψαν τα εξής αποτελέσματα:

Ένας νευρώνας με μια συνιστώσα: Σφάλμα AE 0.0042 / Σφάλμα PCA 0.0043

Τρεις νευρώνες με τρεις συνιστώσες: Σφάλμα AE 0.0014 / Σφάλμα PCA 0.0015

Τέσσερις νευρώνες με τέσσερις συνιστώσες: Σφάλμα AE 0.0007 / Σφάλμα PCA 0.0006

Το πρώτο πράγμα που παρατηρείται είναι ότι όσο αυξάνουμε τους νευρώνες και τις συνιστώσες τόσο το σφάλμα μειώνεται, πράγμα το όποιο είναι λογικό αφού στον ΑΕ δίνουμε μεγαλύτερη υπολογιστική δυνατότητα στο επίπεδο που είναι σχεδιασμένο ώστε να κωδικοποιεί την πληροφορία. Από την άλλη στο PCA έχει περισσότερες συνιστώσες πράγμα που σημαίνει περισσότερες διαστάσεις που έχει ως αποτέλεσμα καλύτερη ακρίβεια μειώνοντας έτσι το σφάλμα. Μπορεί να παρατηρηθεί επίσης ότι ενώ στον έναν, δυο, τρεις νευρώνες αντίστοιχα το ΑΕ έχει ελαφρός καλύτερα αποτελέσματα στους τέσσερις νευρώνες έχει ελαφρός χειρότερη απόδοση από το PCA. Αυτό συμβαίνει διότι όσο μεγαλώνει το νευρωνικό δίκτυο τόσο πιο δύσκολη γίνεται και η εκπαίδευσή του. Συγκεκριμένα τα τέσσερα πειράματα έγιναν με 40000 εποχές το κάθε ένα. Στο πείραμα με τους τέσσερις νευρώνες το σφάλμα δεν κατάφερε να πέσει κάτω από αυτού του PCA επειδή χρειαζόταν περισσότερες εποχές.

Μέρος Δεύτερο

ΕΥΦΥΕΣ ΣΥΣΤΗΜΑ ΣΤΟΧΕΥΣΗΣ

2.1 Εισαγωγή

Η διαδικασία της στόχευσης σε μια μηχανή μπορεί να γίνει με διαφόρους τρόπους, ο τρόπος που θα ερευνηθεί εδώ είναι το αν μπορεί ένα τεχνητό νευρωνικό δίκτυο να στοχεύσει αποτελεσματικά.

Συγκεκριμένα έχουμε ένα σύστημα που μπορεί να περιστραφεί δεξιά-αριστερά και πάνω-κάτω, κάθε σετ κατευθύνσεων ελέγχεται από ένα σέρβο κινητήρα, στο πάνω μέρος της κατασκευής έχουμε μια δέσμη λέιζερ. Σκοπός του συστήματος είναι να μάθει να ελέγχει τα μοτέρ με τέτοιο τρόπο έτσι ώστε θέτοντας μια επιθυμητή τοποθεσία στην οθόνη του προγράμματος να μπορέσει να προσεγγίσει η δέσμη λέιζερ αυτό το επιθυμητό σημείο. Για την μάθηση του νευρωνικού δικτύου έχουμε τρεις εισόδους μια για την οριζόντια τοποθεσία του επιθυμητού στόχου σε pixel, μια για την κατακόρυφη σε pixel και μια για την απόσταση του στόχου από το σύστημα. Οι έξοδοι θα είναι δύο και θα είναι το σήμα από μηδέν έως ένα που συνδέονται σε κάθε μοτέρ αντίστοιχα. Σκοπός είναι να παρατηρήσουμε ποσό καλά μπορεί να γενικεύσει το νευρωνικό με λίγα δεδομένα και τον αν θα καταφέρει να απαλείψει παραμέτρους όπως ο τρόπος με τον οποίο είναι στημένος ο εξοπλισμός στον χώρο.

Αυτή η συσκευή μπορεί να χρησιμοποιηθεί σε συστήματα εικόνας όπου θέλουμε η κάμερα να ακολουθεί ένα συγκεκριμένο στόχο (έτσι ώστε να μην βγει εκτός εικόνας) ή και σε ένα σύστημα το οποίο θέλουμε πάντα ο στόχος να είναι σε ένα συγκεκριμένο σημείο έτσι ώστε να γίνει για παράδειγμα μια εκτύπωση σε ένα συγκεκριμένο σημείο σε ένα προϊόν στην γραμμή παραγωγής.

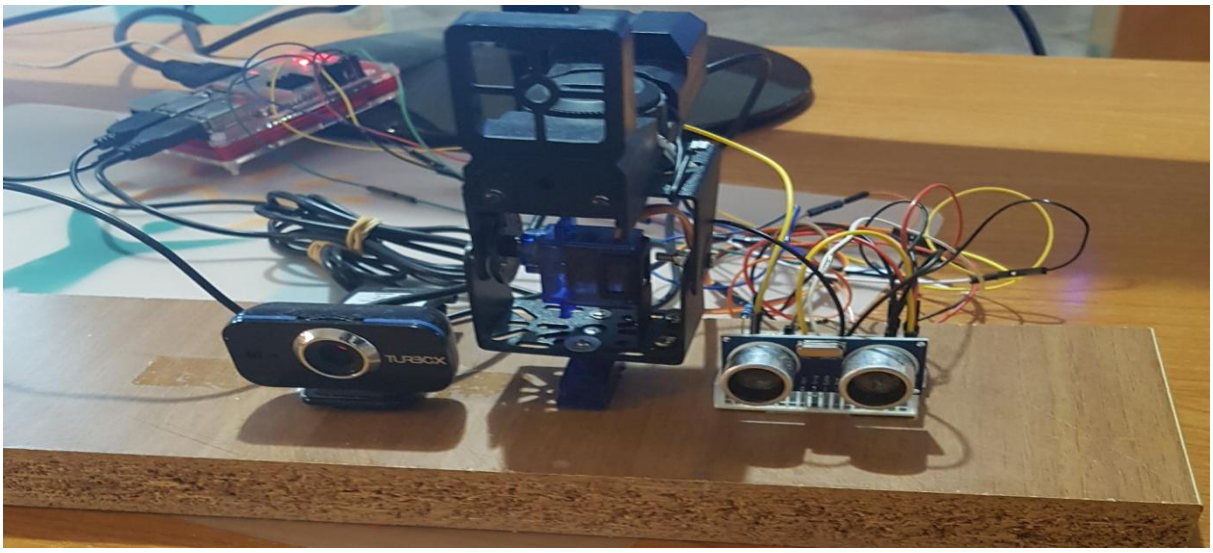
Αυτό το κομμάτι πρόκειται για μια απόδειξη της έννοιας, έτσι ώστε να δούμε αν θα δουλέψει. Στην συνέχεια παραλλαγές αυτού του συστήματος θα μπορούν να εφαρμοστούν στην πράξη όπως για παράδειγμα ο έλεγχος ρομποτικών βραχιόνων. Το γεγονός ότι γίνεται χρήση τεχνητού νευρωνικού δικτύου επιτρέπει την εύκολη προσαρμογή συμπλέγματα νευρωνικών δικτύων, όπου οι κινήσεις που θα κάνει το μοτέρ καθορίζεται από μια ανωτέρου επίπεδου λογική. Επίσης έχουμε την δυνατότητα το δίκτυο να μαθαίνει από αλλαγές και απώλειες του συστήματος (που μπορεί πχ. να προκύψει από φθορές του ρομποτικού βραχίονα) διορθώνοντας μόνο τα βάρη του νευρωνικού που έχουν να κάνουνε με τον έλεγχο των κινητήρων. Ένα παράδειγμα με το σύστημα στόχευσης θα ήταν να υπάρχει ένα συνελικτικό νευρωνικό δίκτυο το οποίο θα ήταν εκπαιδευμένο για την αναγνώριση ενός κατοικίδιου ζώου το οποίο θα είχε ως αποτέλεσμα το μέσο κέντρο στο οποίο το ζώο ανιχνεύτηκε. Μπορεί να συνδεθεί στην συνέχεια αυτή η έξοδο στο νευρωνικό δίκτυο το

οποίο ελέγχει τα μοτέρ έτσι ώστε να ακολουθεί η κάμερα που θα υπάρχει στο άνω μέρος τις πλατφόρμας του συστήματος την πορεία του ζώου.

Το κόστος του συστήματος είναι 44 ευρώ το Raspberry Pi 3 Model B+, και 20 ευρώ όλα τα άλλα εξαρτήματα μας κάνει σύνολο της τάξης των 64 ευρώ. Στην συνέχεια αυτού του μέρους θα περιγραφούν οι διαδικασίες με τις οποίες δημιουργήθηκε το ευφυές σύστημα στόχευσης. Συγκεκριμένα θα περιγραφούν τα φυσικά και κυκλωματικά μέρη της κατασκευής καθώς και το λογισμικό το οποίο δημιουργήθηκε.

2.2 Η υλοποίηση της κατασκευής

Η κατασκευή αποτελείται από το Raspberry Pi 3 Model B+, έναν υπερηχητικό μετρητή απόστασης (HC-SR04), μια απλή κάμερα διαδικτύου, ένα λέιζερ, δυο σέρβο κινητήρες καθώς και η μεταλλική κατασκευή στην οποία εφαρμόζονται πάνω οι κινητήρες ώστε να επιτύχουμε κίνηση δεξιά-αριστερά πάνω-κάτω.

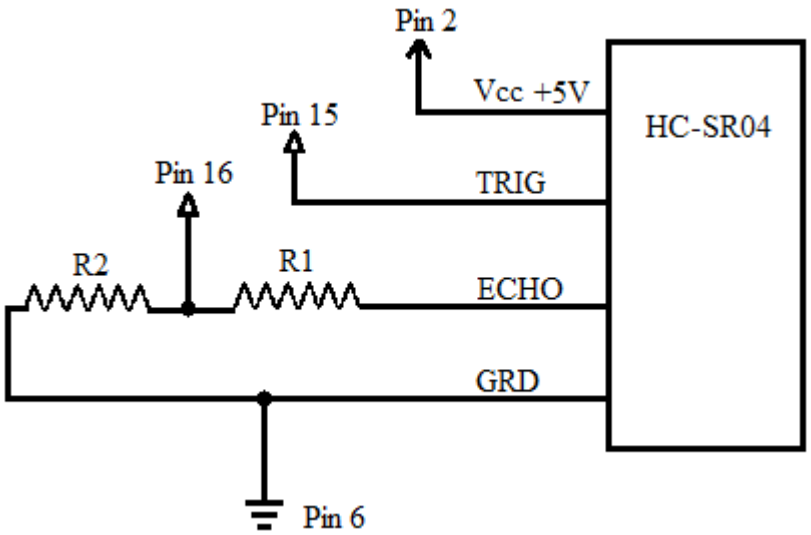


Εικόνα 21: Η κατασκευή.

2.2.1 Η συνδεσμολογία του υπερηχητικού μετρητή απόστασης

Πριν συνδέσουμε το κύκλωμα σημαντικό είναι να υλοποιήσουμε έναν διαιρέτη τάσης αφού το ECHO pin του HC-SR04 μας δίνει τάση 5V ενώ η μεγίστη τάση που επιτρέπεται στο Raspberry σε ακροδέκτη ρυθμισμένο ως είσοδο είναι 3.3 V. Στο κύκλωμα χρησιμοποιήθηκαν 10KΩhm αντιστάσεις (R1 ,R2) όποτε η τάση που παίρνουμε είναι στα 2.5V που είναι

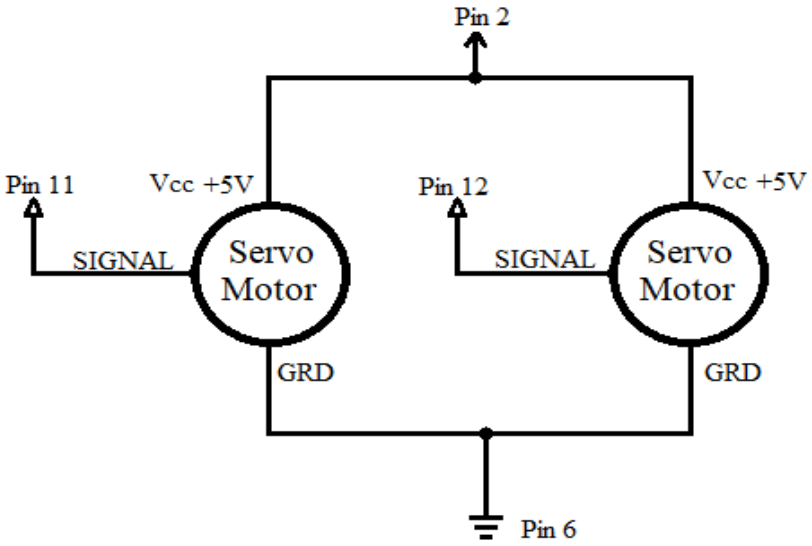
μικρότερη από 3.3 V αλλά αρκετά μεγαλύτερη από 1.3 V που είναι το κατώτερο όριο που καταλαβαίνει το Raspberry ως high. Παρακάτω φαίνεται η συνδεσμολογία με τους ακροδέκτες του Raspberry.



Εικόνα 22: Κύκλωμα συνδεσμολογίας HC-SR04.

2.2.2 Η συνδεσμολογία των σέρβο κινητήρων

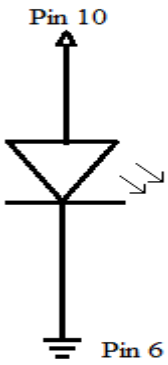
Στην συνδεσμολογία των σέρβο κινητήρων ο κινητήρας που συνδέεται στον ακροδέκτη 11 είναι υπεύθυνος για το δεξιά-αριστερά, ενώ ο κινητήρας του ακροδέκτη 12 είναι για το πάνω-κάτω.



Εικόνα 23: Κύκλωμα συνδεσμολογίας σέρβο κινητήρων.

2.2.3 Η συνδεσμολογία του λέιζερ

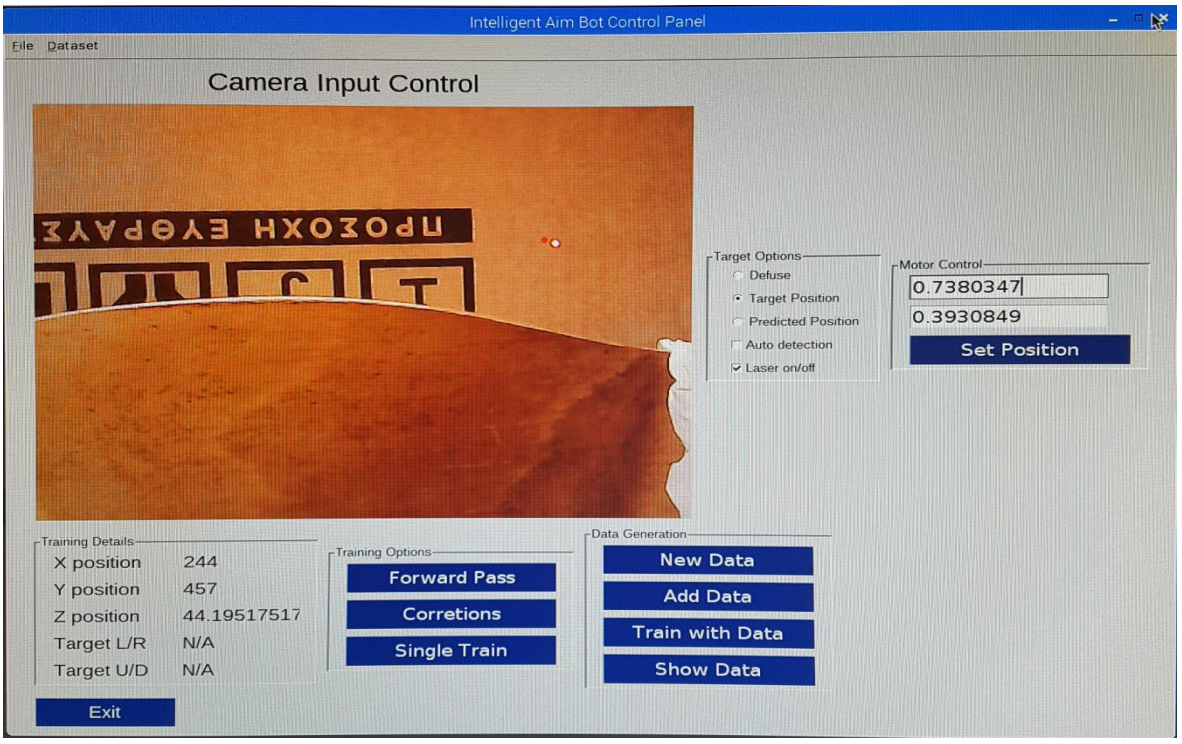
Το λέιζερ καταναλώνει περίπου 11mA οπότε είναι ασφαλής να το συνδέσω απευθείας σε ακροδέκτη του Raspberry αφού η μεγίστη κατανάλωση όλων των GPIO ακροδεκτών δεν πρέπει να ξεπερνά τα 16mA.



Εικόνα 24: Κύκλωμα συνδεσμολογίας λέιζερ.

2.3 Η υλοποίηση του λογισμικού

Το λογισμικό έχει δομηθεί σε κομμάτια και κάθε κομμάτι είναι υπεύθυνο για μια συγκεκριμένη λειτουργία. Για το γραφικό περιβάλλον του προγράμματος έχει χρησιμοποιηθεί η βιβλιοθήκη της python PySimpleGUI [11].



Εικόνα 25: Το πρόγραμμα.

2.3.1 Το νευρωνικό δίκτυο

Το νευρωνικό δίκτυο κατασκευάστηκε με Tensorflow. Πρόκειται για ένα μικρό και ελαφρύ δίκτυο αποτελούμενο από τρεις εισόδους (οριζόντια θέση στόχου στην οθόνη, κατακόρυφη θέση στόχου στην οθόνη, απόσταση αντικειμένου), δύο κρυφά επίπεδα πλήρως διασυνδεδεμένα από οκτώ νευρώνες το κάθε ένα με συναρτήσεις ενεργοποίησης την leaky Relu. Τέλος έχουμε δυο εξόδους πλήρως διασυνδεδεμένες με συναρτήσεις ενεργοποίησης την σιγμοειδή. Εφόσον χρησιμοποιούμε την σιγμοειδή οι έξοδοι παίρνουν τιμές από μηδέν έως και ένα. Με αυτό το σήμα από αυτές τις εξόδους θα οδηγήσουμε το κάθε ένα από τους σέρβο κινητήρες. Ο τρόπος σύγκλισης είναι ο SGD και ο ρυθμός μάθησης 0.5 .

Το πρόγραμμα έχει την δυνατότητα να αποθηκεύει τα βάρη του νευρωνικού όπως και να τα φορτώνει από αρχείο στο σκληρό δίσκο. Επίσης έχει την δυνατότητα να αποθηκεύει τα δεδομένα που δημιουργήθηκαν από τον χρήστη έτσι ώστε μετά αν χρειαστεί να τα ξαναφορτώσει.

2.3.2 Ο έλεγχος των σέρβο κινητήρων

Ο παλμός που ελέγχει τους σέρβο κινητήρες τους παράγει η βιβλιοθήκη της python GPIO με την μέθοδο PWM η οποία δίνει την δυνατότητα να παραχθούν παλμοί σε μια επιλεγμένη από τον χρήστη συχνότητα σε ένα επιλεγμένο ακροδέκτη στην οποία μετά μπορούμε να μεταβάλλουμε το duty cycle με το οποίο ρυθμίζεται η θέση του σερβοκινητήρα. Το duty cycle στους συγκεκριμένους κινητήρες στα 50Hz συχνότητα κυμαίνεται από 2%-12%. Οπότε αν είχαμε μοίρες ο τύπος υπολογισμού του duty cycle θα ήταν $duty = \frac{angle\ in\ degrees}{18} + 2$. Εμείς από το νευρωνικό δίκτυο έχουμε ένα σήμα από 0-1 το οποίο πρέπει να το μετατρέψουμε σε μοίρες. Για τον σέρβο κινητήρα που ελέγχει το δεξιά-αριστερά το 0 σημαίνει τέρμα αριστερά ενώ το 1 σημαίνει τέρμα δεξιά, αντίστοιχα για το πάνω-κάτω το 0 σημαίνει τέρμα πάνω ενώ το 1 τέρμα κάτω. Οι σέρβο κινητήρες έχουν την δυνατότητα περιστροφής 180 μοιρών, ωστόσο δεν χρειαζόμαστε όλες αυτές τις μοίρες αφού η κάμερα μπορεί να δει σε ένα εύρος 45 μοιρών, οπότε προγραμματιστικά μειώνουμε την μέγιστη και ελάχιστη μοίρα των μοτέρ από 45 μοίρες έως 135 μοίρες. Η βιβλιοθήκη με την οποία ελέγχουμε τους ακροδέκτες μας παρέχει έτοιμη μέθοδο για διαμόρφωση πλάτους του παλμού αλλά δεν έχει την ακρίβεια που θα είχε μια γεννήτρια συχνοτήτων οπότε σε κάποιες περιπτώσεις τα μοτέρ μπορεί να μετατοπίζονται συνεχόμενα μεταξύ δυο πολύ κοντινών μοιρών.

2.3.3 Ο έλεγχος του υπερηχητικού μετρητή απόστασης

Ο κατασκευαστής του υπερηχητικού μετρητή απόστασης έχει ορίσει ότι θέλει έναν παλμό 1μs στον ακροδέκτη trig και ο χρόνος που αρχίζει να μετράει από την στιγμή που ο echo ακροδέκτης πήγε στο 1 και τελειώνει να μετράει όταν πάει στο 0. Το κομμάτι του προγράμματος που ελέγχει τον αισθητήρα επιστρέφει την απόσταση σε cm με τον υπολογισμό:

$$απόσταση = (χρονος\ τελικός - χρόνος\ αρχικός) * 170 * 100.$$

Αυτό διότι γνωρίζουμε ότι ο ήχος κινείται με 340m/s η απόσταση που καλύφθηκε είναι διπλάσια (η απόσταση να πάει να αντανακλαστεί το ηχητικό κύμα και η απόσταση να επιστρέψει) οπότε είναι 340/2=170 , η μετατροπή των m σε cm είναι επί 100 οπότε με αυτόν τον τρόπο προκύπτει ο παραπάνω τύπος.

2.3.4 Ο έλεγχος της κάμερας

Για να διαβάσουμε τα πακέτα που μας στέλνει η κάμερα θα χρησιμοποιήσουμε το OpenCV [12] μια βιβλιοθήκη η οποία είναι και για μηχανική όραση αφού μας προσφέρει πληθώρα εργαλείων για να επεξεργαστούμε εικόνες. Η κάμερα δεν διαθέτει αυτόματη εστίαση οπότε ο χρήστης πρέπει να κάνει την ρύθμιση το οποίο μας επιτρέπει να δημιουργούμε εύκολα εικόνες θολές και με λανθασμένα χρώματα έτσι ώστε να μετρήσουμε την απόδοση του αυτόματου ανιχνευτή λέιζερ σε διάφορες συνθήκες.

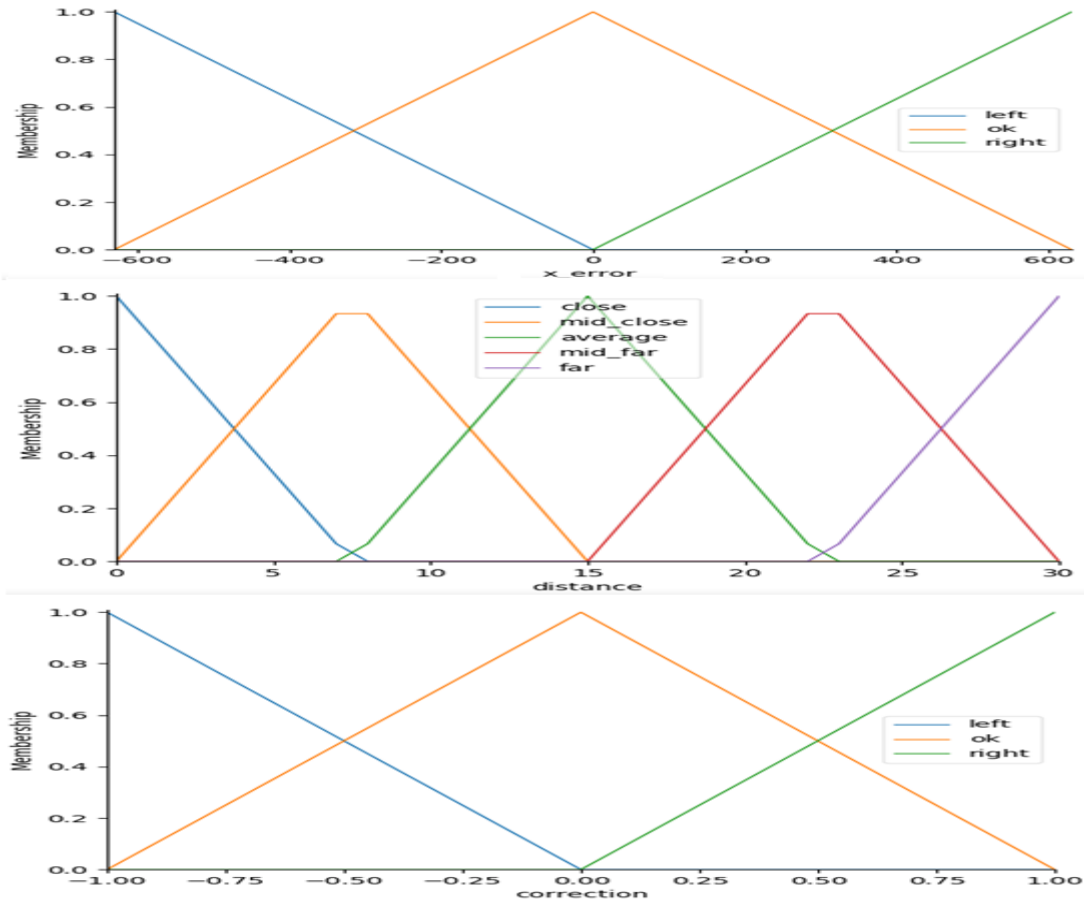
2.3.5 Ο έλεγχος του λέιζερ

Το λέιζερ μπορεί να ελεγχτεί από το πρόγραμμα για το πότε θα είναι ενεργό ή ανενεργό μέσο ενός checkbox πεδίου. Σε αυτό το κομμάτι του προγράμματος είναι και η αυτόματη ανίχνευση του μέσο του OpenCV.

2.3.6 Το ασαφές σύστημα

Θα χρησιμοποιήσουμε το ασαφές σύστημα για να βρεθεί η διόρθωση που πρέπει να γίνει έτσι ώστε να στοχεύσει το σύστημα σωστά. Το ασαφές σύστημα είναι κατάλληλο διότι έχουμε το σφάλμα και την απόσταση του στόχου και ψάχνουμε να βρούμε πια είναι η διόρθωση που πρέπει να προστεθεί έτσι ώστε να επιτύχουμε το βέλτιστο αποτέλεσμα.

Για την υλοποίηση του ασαφούς συστήματος θα χρησιμοποιήσουμε την βιβλιοθήκη skfuzzy[13]. Υπάρχουν τρία ασαφή σύνολα τα δυο πρώτα αφορούν τις εισόδους. Το πρώτο στην εικόνα 22 είναι το ασαφές σύνολο του σφάλματος που υπολογίζεται από την αφαίρεση της τοποθεσίας που δείχνει το λέιζερ και την επιθυμητή τοποθεσία που θέλουμε να δείχνει το λέιζερ. Προφανώς το εύρος αυτής κυμαίνετε από 0 έως μέγιστο αριθμό pixel που εδώ είναι 640.Οποτε ο χ άξονας αυτού το ασαφούς συνόλου είναι από -640 έως 640.Αυτο το ασαφές σύνολο έχει τρεις τριγωνικές συναρτήσεις συμμετοχής με βαθμό επικάλυψης 50%. Το δεύτερο ασαφές σύνολο έχει να κάνει με την απόσταση έχει εύρος από 0 έως 30cm απόσταση και έχει πέντε τριγωνικές συναρτήσεις συμμετοχής με βαθμό επικάλυψης 50%. Το τρίτο ασαφές σύνολο είναι αυτό τις εξόδου, και υπολογίζει πια θα είναι διόρθωση που θα πρέπει να προστεθεί στις εξόδους του νευρωνικού έτσι ώστε σταδιακά να έχουμε την σύγκλιση της επιθυμητής με την πραγματική θέση του λέιζερ. Αποτελείται από τρεις τριγωνικές συναρτήσεις συμμετοχής με βαθμό επικάλυψης 50% και εκτείνονται από -1 έως 1.



Εικόνα 26: Συναρτήσεις συμμετοχής του ασαφούς συστήματος.

Οι συνεπαγωγές είναι τύπου mamdani , και η αποσαφοποίηση γίνεται με centroid. Οι κανόνες για την πυροδότηση του ασαφούς συστήματος στην έξοδο ορίζονται από τον πινάκα 1.

	Σφάλμα			
		left	ok	right
Απόσταση	close	left	ok	right
	mid close	left	ok	right
	average	left	ok	right
	mid far	left	ok	right
	far	ok	ok	οκ

Πίνακας 3: Πίνακας κανόνων ασαφούς συστήματος.

2.3.7 Η εκπαίδευση του νευρωνικού

Η εκπαίδευση του νευρωνικού δικτύου γίνεται ως εξής:

Διαλέγουμε τον στόχο στην οθόνη (Target position) και πατάμε πάνω στην εικόνα που βλέπουμε από τη κάμερα. Μετά πατάμε να γίνει η εμπρός τροφοδότηση (forward pass). Μόλις γίνει η εμπρός τροφοδότηση τα δυο μοτέρ θα έχουν νέες θέσεις (τυχαίες στην αρχή αφού το δίκτυο είναι ανεκπαίδευτο). Στην συνέχεια διαλέγουμε το Predicted position και πατάμε στην οθόνη εκεί που δείχνει το λέιζερ και στην συνέχεια τρέχουμε το ασαφές σύστημα για να μας δώσει τις διορθώσεις (corrections).Μόλις γίνουν οι διορθώσεις κάνουμε μια οπισθοδρόμηση (Single Train) και επαναλαμβάνουμε από την αρχή την διαδικασία.

Η παραπάνω διαδικασία δεν έχει καλά αποτελέσματα αφού η ακρίβεια των σέρβο κινητήρων δεν είναι καλή και το γεγονός ότι έχουμε ένα ασαφές σύστημα το οποίο χρειάζεται άριστη ρύθμιση για να μην περνάει θόρυβος στα δεδομένα με τα οποία γίνεται η εκπαίδευση το καθιστά δύσκολο και χρονοβόρο στην τελική εκπαίδευση του νευρωνικού. Χρονοβόρο διότι μέχρι να φτάσει το λέιζερ σε μια προσεγγιστικά κοντινή απόσταση από το επιθυμητό σημείο πρέπει να γίνουν αρκετές διορθώσεις.

Οπότε η λύση που δόθηκε σε αυτό το προβλήματα είναι η βέλτιστη ως προς την ελαχιστοποίηση του θορύβου στα δεδομένα και την ταχύτητα συλλογής τους. Συγκεκριμένα η εκπαίδευση του νευρωνικού γίνεται με τον απλούστατο τρόπο: ο χρήστης αρχικά θέτει τις τιμές στα μοτέρ από 0-1, στην συνέχεια θέτει τον στόχο (Target position) εκεί που δείχνει το λέιζερ πατάμε το (New Data) για να πάρουμε και μέτρηση από τον μετρητή απόστασης και μόλις γίνει αυτό μπορούμε να το προσθέσουμε σε μια λίστα (Add data) επαναλαμβάνουμε την διαδικασία από την αρχή. Μόλις έχουμε αρκετά δεδομένα μπορούμε να αρχίσουμε την εκπαίδευση με τα δεδομένα πατώντας το Train with Data.

Είναι σημαντικό να επισημανθεί ότι τα δεδομένα πρέπει να έχουν ποικιλία, που σημαίνει ότι πρέπει να πορθούν τιμές από διάφορες τοποθεσίες των μοτέρ και διάφορες αποστάσεις και όχι μόνο από συγκεκριμένα σημεία (πχ. όλα τα σημεία στα αριστερά τις οθόνης), μόνο έτσι θα έχουμε τα βέλτιστα αποτελέσματα.

2.4 Συμπεράσματα.

Τα αποτελέσματα που είχαμε από την εκπαίδευση την δεύτερη φορά είναι αρκετά καλά αφού το σύστημα μετά από εκπαίδευση με 22 δεδομένα και μετά από 2000 εποχές έχουμε σφάλμα μικρότερο από 50 pixel. Το σύστημα κατάφερε να κάνει γενίκευση των δεδομένων έτσι ώστε να μην έχουμε το φαινόμενο της υπερμάθησης και κατάφερε να απαλείψει παράμερους που οφείλονται στην τοποθεσία του εξοπλισμού στον χώρο.

Αναφορές

[1] https://keras.io/

[2] Αναστάσιος Ντούνης , σημειώσεις υπολογιστικής νοημοσύνης ΤΝΔ 2016.

[3] M. Negnevitsky "Τεχνητή Νοημοσύνη"

[4] Ken Nozaki Hisao Ishibuchi, HideoTanaka, "A simple but powerful heuristic method for generating fuzzy rules from numerical data", Fuzzy Sets and Systems 86 (1997) 251-270

[5] https://numpy.org/

[6] https://matplotlib.org/

[7] https://www.tensorflow.org/

[8] https://en.wikipedia.org/wiki/Principal_component_analysis

[9] https://en.wikipedia.org/wiki/Autoencoder

[10] https://scikit-learn.org/stable/

[11] https://pysimplegui.readthedocs.io/en/latest/

[12] https://opencv.org/

[13] https://pythonhosted.org/scikit-fuzzy/

Παράρτημα

Δεδομένα από δοκιμές ρυζιού.

data=np.array([[0.523,0.913,1.571,1.6,0,1.784],[0.699,1.543,1.76,1.944,-0.875,1.706],[-0.720,-2.0220,-1.733,-1.864,1.200,-2.217],[-0.309,-0.377,-0.766,-0.96,0.326,-0.875],[0.192,0.885,-0.166,-0.106,0.234,0.218],[-0.163,-0.094,-0.315,0.083,0.034,-0.655],[-0.274,0.044,-0.194,-0.401,1.57,-0.545],[0.291,0.984,0.935,0.68,-0.652,1.033],[-2.687,-2.853,-3.067,-1.75,2.392,-3.101],[-0.704,-0.424,-0.518,-0.779,0.052,-0.652],[1.094,1.987,1.613,1.514,-1.156,1.552],[-0.247,0.334,0.122,-0.705,0.694,-0.416],[-0.673,0.032,-0.838,-0.837,0.354,-0.784],[-1.103,-1.046,-1.678,-1.335,-0.46,-1.71],[-0.033,0.162,-0.541,-0.159,0.616,-0.529],[-0.351,0.119,-0.786,-0.435,0.129,-0.635],[0.594,0.266,0.885,1,-0.548,1.105],[-0.156,0.173,0.272,0.092,0.163,0.145],[-0.256,-0.593,-1.012,-1.403,1.165,-1.901],[-0.183,-0.496,-0.77,-1.538,1.78,-1.188],[-0.871,-1.27,-1.655,-1.307,1.281,-2.037],[-0.060,-0.11,-0.95,-0.719,1.159,-0.788],[-1.135,0.163,-0.47,0.285,-0.023,-0.911],[-0.609,-0.215,-0.681,-0.534,0.963,-0.612],[0.393,0.388,-0.232,0.154,-0.42,-0.127],[-2.163,-0.911,-1.61,-1.328,0.145,-1.952],[0.831,0.6,0.953,0.953,-0.163,1.141],[0.081,1.712,0.946,0.678,-0.257,1.723],[0.836,0.799,0.782,1.529,-0.769,1.038],[0.214,0.544,0.679,0.314,-0.347,0.51],[-0.827,-0.759,-1.012,-0.517,0.197,-0.788],[0.221,0.708,0.737,0.923,-0.495,0.899],[-0.558,-0.343,-0.331,0.266,-0.756,-0.157],[-0.684,-0.33,-0.185,0.349,-0.545,-0.155],[-1.816,-0.952,-1.409,-1.882,0.639,-1.783],[0.013,1.01,0.927,1.135,-0.387,0.937],[0.344,0.856,0.37,0.327,0.227,0.36],[-0.323,-0.772,-0.895,-1.002,0.517,-1.286],[-0.82,0.608,-0.288,-0.311,0.386,-0.837],[0.24,-0.906,-0.715,-1.467,1.626,-1.646],[-0.571,-0.151,-0.635,-0.093,0.956,-0.73],[0.789,0.821,-0.769,0.428,-1.815,-0.508],[-0.353,0.115,-0.394,0.722,-0.089,-0.731],[-0.467,-0.622,-0.661,-0.302,0.698,-0.986],[-0.491,0.021,-0.221,-0.563,0.385,-0.672],[0.848,0.837,1.54,1.112,-0.475,1.113],[0.099,0.12,0.387,1.252,-0.638,0.655],[-0.738,0.989,0.415,0.574,-0.647,0.093],[-0.82,-0.702,1.203,1.579,-1.212,0.483],[-0.629,-1.049,-1.271,-1.116,1.637,-1.407],[-0.593,-0.898,-0.883,-0.647,0.323,-1.235],[0.14,0.181,0.06,0.136,-0.234,0.416],[-1.022,-0.827,-1.294,-0.583,0.437,-1.357],[-0.649,-0.288,0.257,0.327,-0.412,-0.653],[-0.413,0.026,-0.108,0.393,-0.291,0.019],[-0.358,-0.106,-1.275,-1.059,0.505,-1.168],[-1.825,-2.018,-1.867,-1.517,0.114,-2.136],[-0.002,-0.144,0.545,0.674,-0.111,0.641],[0.158,0.163,0.03,0.359,-0.128,0.135],[0.516,0.73,1.218,1.662,-0.366,1.44],[-0.235,-0.566,-0.289,-0.763,0.048,-0.749],[-1.085,-0.614,-1.24,-1.12,0.958,-0.65],[-1.283,0.09,-0.651,-0.19,-0.131,-1.067],[0.25,0.232,0.378,1.983,-

1.231,0.519],[0.288,0.205,0.147,0.399,0.365,0.352],[0.567,1.502,0.824,0.53,-0.387,0.842],[0.011,0.26,0.102,0.487,-0.402,0.067],[-0.912,-0.911,-1.448,-1.666,1.375,-1.524],[-0.588,0.792,1.023,0.995,-0.489,0.901],[-0.686,-0.682,-0.514,-0.088,-0.43,-0.466],[-2.871,0.59,-0.117,0.195,0.051,-1.084],[-1.208,-0.257,-0.426,-0.237,0.24,-0.628],[-1.819,-0.856,-1.758,-1.014,1.094,-1.925],[0.713,1.172,0.648,1.065,-0.886,1.053],[-0.248,0.185,0.204,-0.41,0.588,-0.186],[-0.638,0.462,-0.149,-0.214,0.389,-0.23],[0.255,0.98,0.837,1.64,0.028,1.747],[-1.212,-0.074,-0.317,-0.144,0.18,-0.658],[-0.093,0.712,0.594,1.27,-0.236,0.88],[-0.315,-0.352,-0.741,-0.288,0.201,-0.964],[0.021,0.805,0.367,0.19,0.33,0.056],[0.158,0.265,0.053,0.373,-0.101,0.164],[0.372,0.388,0.448,0.291,-0.402,0.489],[-0.47,-0.301,-0.481,0.173,0.263,-0.546],[-0.244,-0.044,-0.466,-0.464,0.469,-0.804],[-0.049,0.401,-0.097,0.062,0.282,-0.223],[-0.639,0.63,-0.089,0.166,-0.293,-0.394],[0.039,0.524,0.207,1.082,-1.3030,0.5370],[0.079,1.231,0.373,0.281,-1.273,0.218],[0.104,-0.194,0.261,0.838,0.085,0.434],[0.475,0.707,-0.172,0.097,-0.247,-0.246],[0.468,0.333,0.511,1.188,-0.646,0.78],[-0.465,-0.195,-0.491,-0.671,0.837,-0.614],[0.125,0.127,-0.264,0.461,0.197,-0.046],[-0.491,0.525,0.419,0.78,-0.512,0.291],[-1.388,-0.114,-0.894,-0.168,0.028,-0.821],[-0.955,-1.146,-1.107,-0.922,0.106,-1.367],[-0.255,0.232,0.517,0.398,-0.247,0.424],[-0.686,-0.252,-0.749,-0.428,0.048,-0.704],[-0.016,0.674,-0.046,0.662,-0.694,0.326],[-0.618,-0.425,0.098,0.471,0.158,0.104],[0.57,1.542,1.302,1.842,-0.563,1.712],[-1.134,-1.281,-1.031,-1.477,1.345,-1.624],[0.124,0.588,0.053,-0.405,-0.322,-0.517],[0.163,1.088,1.074,0.683,-0.135,0.913]])

Αποτελέσματα κρυφού επίπεδου από τις μετρήσεις χημικού εργαστηρίου.

Συνάρτηση ενεργοποίησης γραμμική	Συνάρτηση ενεργοποίησης υπερβολική εφαπτομένη
[[0.24986011]	[[-0.2821863]
[0.18043542]	[-0.4127887]
[-0.04252959]	[-0.68949616]
[0.1618154]	[-0.4284277]
[-0.06311428]	[-0.70235145]
[-0.15920523]	[-0.7716283]
[-0.1264575]	[-0.75161755]

[0.1355246]	[-0.48821187]
[-0.2594131]	[-0.8220801]
[-0.20608068]	[-0.80086106]
[0.32340184]	[-0.13159378]
[0.1600385]	[-0.44920668]
[0.42164505]	[0.06175293]
[0.01688438]	[-0.6364236]
[0.08219216]	[-0.5528927]
[0.44222984]	[0.09176587]
[0.46477905]	[0.13207144]
[0.45458058]	[0.1096247]
[0.37261727]	[-0.042856]
[0.45458058]]	[0.1096247]]

Αποτελέσματα κρυφού επίπεδου από δοκιμές ρυζιού.

Συνάρτηση ενεργοποίησης γραμμική	Συνάρτηση ενεργοποίησης υπερβολική εφαπτομένη
[[-0.5122299 -1.1046953]	[[3.87815893e-01 -1.84737965e-01]
[-0.6535093 -1.0905354]	[4.69966352e-01 -3.38295281e-01]
[0.28664777 -0.35890013]	[6.31060228e-02 7.03870118e-01]
[-0.02060623 -0.6037774]	[3.07434291e-01 4.90914524e-01]
[-0.19876556 -0.86712605]	[3.06324035e-01 2.88592637e-01]
[-0.17818081 -0.64784175]	[4.30171311e-01 3.85406405e-01]
[0.01740032 -0.8145539]	[1.65675893e-01 4.16375458e-01]
[-0.44883287 -0.9267339]	[4.40112084e-01 1.50071476e-02]
[0.45928317 0.01801991]	[1.61083311e-01 9.05016363e-01]
[-0.09896483 -0.5496679]	[4.21759665e-01 4.78241205e-01]
[-0.63056475 -1.1345475]	[4.10817325e-01 -3.07315826e-01]
[-0.07609981 -0.78907734]	[2.50418782e-01 3.84830058e-01]
[-0.05196832 -0.5851895]	[3.56577843e-01 4.85572189e-01]
[-0.0061003 -0.24917337]	[4.95150030e-01 6.53361857e-01]

[-0.09136167 -0.7391558]	[3.00980985e-01 4.02403265e-01]
[-0.11408502 -0.6239295]	[3.85275841e-01 4.39970106e-01]
[-0.4402532 -0.9260729]	[4.71802920e-01 5.09287883e-03]
[-0.23209153 -0.7811813]	[3.97628188e-01 2.86748379e-01]
[0.17654975 -0.5872615]	[6.97471425e-02 5.62873662e-01]
[0.2132811 -0.73755985]	[-1.25819659e-02 5.14716625e-01]
[0.20976017 -0.41020638]	[1.57419190e-01 6.62945569e-01]
[0.05351323 -0.7296877]	[1.74301133e-01 4.71564531e-01]
[-0.22226487 -0.4914044]	[5.64320683e-01 4.42968309e-01]
[-0.01879352 -0.65511525]	[2.99601495e-01 4.70822960e-01]
[-0.25658876 -0.7615426]	[4.19296503e-01 2.89003491e-01]
[0.03083372 -0.15448534]	[5.46780467e-01 7.41278529e-01]
[-0.3988567 -1.0282485]	[3.72882187e-01 6.80293515e-03]
[-0.462842 -1.0419974]	[3.41378421e-01 -5.27254827e-02]
[-0.51089084 -0.9567854]	[4.95129555e-01 -8.71986672e-02]
[-0.3389328 -0.86188024]	[4.18994516e-01 1.59461930e-01]
[-0.07636757 -0.47597474]	[4.57828104e-01 5.28196394e-01]
[-0.43489552 -0.8856496]	[4.70839202e-01 4.43119891e-02]
[-0.31057513 -0.5334953]	[6.03200853e-01 3.45481545e-01]
[-0.30527312 -0.5457917]	[5.97713113e-01 3.43938947e-01]
[0.13165492 -0.28238755]	[3.63004297e-01 7.03952789e-01]
[-0.46590602 -0.9008371]	[4.74255830e-01 -4.71587235e-04]
[-0.26137906 -0.92573196]	[3.20082039e-01 2.06899151e-01]
[0.03791204 -0.55233175]	[2.68835485e-01 5.35393298e-01]
[-0.13490446 -0.6329947]	[3.94641787e-01 4.22385603e-01]
[0.23289885 -0.7195389]	[-6.89796507e-02 5.29802084e-01]
[-0.05637832 -0.6517897]	[3.34198028e-01 4.55638707e-01]
[-0.39340496 -0.64346534]	[5.75429857e-01 2.28691414e-01]
[-0.25439423 -0.60388833]	[5.24333835e-01 3.53185654e-01]
[-0.03761087 -0.58916885]	[3.43347907e-01 4.91011441e-01]
[-0.09727828 -0.6581972]	[3.49164635e-01 4.29600805e-01]
[-0.47509152 -1.0512104]	[4.10147637e-01 -1.04973361e-01]
[-0.44437224 -0.7662477]	[5.71933627e-01 8.46964493e-02]

[-0.40712845 -0.6774477]	[5.61287522e-01 1.95524424e-01]
[-0.5707072 -0.55896384]	[7.76689947e-01 2.67151780e-02]
[0.18894872 -0.5660039]	[1.12693645e-01 5.88156402e-01]
[-0.02404375 -0.48647124]	[3.83497208e-01 5.41451633e-01]
[-0.27451962 -0.7913285]	[4.21498924e-01 2.54847974e-01]
[-0.01076043 -0.40510505]	[4.33455348e-01 5.93547046e-01]
[-0.283045 -0.5569458]	[5.77444851e-01 3.44958097e-01]
[-0.29278392 -0.65165704]	[5.23119450e-01 3.02488744e-01]
[0.03374185 -0.57864577]	[2.68442035e-01 5.23330927e-01]
[0.10194192 -0.09171146]	[4.91467983e-01 7.85433948e-01]
[-0.33630502 -0.80202466]	[4.79504108e-01 1.76028728e-01]
[-0.26783 -0.77573603]	[4.29724306e-01 2.64292419e-01]
[-0.5235163 -1.0042989]	[4.69129562e-01 -1.42713889e-01]
[-0.08689936 -0.61207086]	[3.63651186e-01 4.52512324e-01]
[0.05275925 -0.5284211]	[3.25580001e-01 5.58872342e-01]
[-0.17932612 -0.43774945]	[5.56162834e-01 4.96837646e-01]
[-0.56716156 -0.71614176]	[6.84676170e-01 -4.35188040e-02]
[-0.22905366 -0.8688076]	[3.48809451e-01 2.51519620e-01]
[-0.3968199 -1.0077739]	[3.42700750e-01 4.67649288e-02]
[-0.31638008 -0.73303455]	[4.88277972e-01 2.42064700e-01]
[0.21023566 -0.49786323]	[1.33557871e-01 6.18973553e-01]
[-0.48037887 -0.79681236]	[5.48733950e-01 3.59054208e-02]
[-0.21652405 -0.49204934]	[5.62740564e-01 4.35267329e-01]
[-0.27837312 -0.29674897]	[7.21265376e-01 5.59503913e-01]
[-0.15276863 -0.50936043]	[5.08845866e-01 4.76257831e-01]
[0.12073806 -0.2865339]	[3.88063937e-01 7.12428033e-01]
[-0.49011958 -0.953697]	[4.53645855e-01 -3.74936387e-02]
[-0.12488887 -0.7872224]	[3.01860571e-01 3.57195735e-01]
[-0.16780028 -0.70358354]	[3.89578283e-01 3.69989395e-01]
[-0.491979 -1.0230489]	[4.13060725e-01 -1.01666532e-01]
[-0.17546247 -0.5194392]	[5.17545521e-01 4.57549036e-01]
[-0.44160432 -0.85127884]	[4.99614000e-01 5.06075062e-02]
[-0.09309982 -0.57766634]	[3.94659758e-01 4.67460364e-01]

[-0.23240654 -0.8648818]	[3.33523840e-01 2.53888547e-01]
[-0.27095747 -0.78916854]	[4.21896905e-01 2.56619394e-01]
[-0.32369417 -0.8511346]	[4.19300020e-01 1.81733042e-01]
[-0.16437511 -0.6087373]	[4.54409480e-01 4.14371312e-01]
[-0.07155578 -0.6685161]	[3.17309737e-01 4.39647913e-01]
[-0.18662362 -0.7713501]	[3.59764457e-01 3.29701960e-01]
[-0.27616245 -0.6348876]	[5.02865970e-01 3.30546230e-01]
[-0.50117415 -0.70340246]	[6.19805515e-01 7.27533475e-02]
[-0.43313736 -0.75947493]	[5.14964640e-01 1.38529837e-01]
[-0.30404407 -0.7951841]	[4.65311587e-01 2.10302413e-01]
[-0.23459227 -0.80788904]	[3.66897345e-01 2.89950490e-01]
[-0.4436918 -0.85106355]	[5.18815577e-01 4.45905663e-02]
[-0.02354406 -0.6787315]	[2.80900955e-01 4.57600385e-01]
[-0.22139911 -0.76520586]	[4.03899640e-01 3.06249529e-01]
[-0.4009508 -0.70499676]	[5.58664382e-01 1.76239833e-01]
[-0.16392162 -0.42697877]	[5.62363625e-01 5.15835285e-01]
[-0.01315837 -0.3760622]	[4.42027599e-01 6.00235164e-01]
[-0.33237767 -0.7652351]	[4.81734872e-01 2.07008973e-01]
[-0.1233696 -0.53977734]	[4.50751334e-01 4.72002238e-01]
[-0.38169223 -0.7383438]	[5.16287863e-01 1.89991295e-01]
[-0.25814945 -0.65355146]	[5.16406775e-01 3.23029101e-01]
[-0.59607196 -1.0745647]	[4.31587189e-01 -2.48087093e-01]
[0.17984249 -0.4544453]	[1.84013367e-01 6.39397323e-01]
[-0.20181997 -0.743771]	[3.68518084e-01 3.36037248e-01]
[-0.39903897 -0.966041]]	[3.84777337e-01 4.39758226e-02]]

Κώδικας εισαγωγής στο πρόγραμμα:

```
import package.IntelligentControl as ic

import package.GUI as gui

import atexit

import os


is_raspberry = True

if is_raspberry:

    import RPi.GPIO as GPIO

    GPIO.setmode(GPIO.BOARD)


def main():

    net = ic.IntelligentControl()

    screen = gui.GUI(is_raspberry)

    while True:

        event, values = screen.window.Read(timeout = 10)

        screen.CheckRadioButton()

        screen.CheckEvent(event, values, net)

        screen.UpdateWindow()


def ClearAllGPIO():

    if is_raspberry:

        GPIO.cleanup()
```

```
    print("Now quiting . . .")


if __name__ == '__main__':
    atexit.register(ClearAllGPIO)
    main()
```

Κώδικας για το γραφικό περιβάλλων.

```
import PySimpleGUI as sg
import cv2 as cv
import numpy as np
#from PIL import Image
import io
from os import path
import package.fuzzyX as fuzzyX
import package.fuzzyY as fuzzyY

import package.LaserPosition as lp

import package.DataGenerator as dg

#import those two only with in raspberry env
```

```
import package.ServoActuator as sa
import package.DistanceMeasure as dm


class DataModel:
    def __init__(self):
        self.position_x=None
        self.position_y=None
        self.position_z=None
        self.target_lr=None
        self.target_ud=None


class GUI():
    def __init__(self, is_raspberry):
        self.is_raspberry=is_raspberry
        if is_raspberry:
            self.servo_left_right=sa.ServoActuator(11, "X Position")#pin 11
            self.servo_up_down=sa.ServoActuator(12, "Y Position")   #pin 12
            self.distance_measure=dm.Distance(15, 16) #pin 15=trig, pin 16 =echo
            self.lp=lp.LaserPosition(10)#pin 10


        self.image_resolution = (640,480)


        self.dgX=dg.DataGeneration()
```

```
        self.dgY=dg.DataGeneration()

        self.fuzzX=fuzzyX.FZX(self.image_resolution[0])
        self.fuzzY=fuzzyY.FZY(self.image_resolution[1])

        self.position_mode_target = None
        self.target_position = None
        self.predicted_position = None

        self.id_frame=None
        self.id_target=None
        self.id_predicted=None

        self.training_data=DataModel()

        menu_def = [['&File', ['&Load weights', '&Save weights', '&Exit']],
                    ['&Dataset', ['&Import data','&Export data']]]

        sg.ChangeLookAndFeel('LightBlue')
        self.layout = [[sg.Menu(menu_def, tearoff=True)],
                    [sg.Text('Camera Input Control', size=(40, 1), justification='center',
font='Helvetica 20')],
                    [sg.Graph(self.image_resolution, (0,self.image_resolution[1]),
(self.image_resolution[0],0), key='graph', enable_events=True, drag_submits=False),
```

```
            sg.Frame(layout=[[sg.Radio('Defuse', "radio_group_target" ,key='radio_defuse'
,default=True, size=(10,1))],

                        [sg.Radio('Target Position', "radio_group_target" ,key='radio_target')],

                        [sg.Radio('Predicted Position', "radio_group_target"
,key='radio_predicted')],

                        [sg.Checkbox('Auto detection',key='auto_detection')],

                        [sg.Checkbox('Laser on/off',key='laser_on_off')]],

            title='Target Options',title_color='black', relief=sg.RELIEF_SUNKEN),

             sg.Frame(layout=[[sg.InputText('', size=(15,1), font='Any
14',key='x_motor_position')],

                         [sg.InputText('', size=(15,1), font='Any 14',key='y_motor_position')],

                         [sg.Button('Set Position', size=(15,1), font='Any 14')]]

                    ,title='Motor Control',title_color='black', relief=sg.RELIEF_SUNKEN)

             ],

                         [sg.Frame(layout=[[sg.Text('X position', size=(10, 1),
justification='left', font='Helvetica 14')

            ,sg.Text('N/A',size=(10, 1), justification='left', font='Helvetica 14',
key="x_pos_text")],

            [sg.Text('Y position', size=(10, 1), justification='left', font='Helvetica 14')

            ,sg.Text('N/A', size=(10, 1), justification='left', font='Helvetica 14',
key="y_pos_text")],

            [sg.Text('Z position', size=(10, 1), justification='left', font='Helvetica 14')

            ,sg.Text('N/A', size=(10, 1), justification='left', font='Helvetica 14',
key="z_pos_text")],

            [sg.Text('Target L/R', size=(10, 1), justification='left', font='Helvetica 14')
```

```
                ,sg.Text('N/A', size=(10, 1), justification='left', font='Helvetica 14',
key="tagret_l_r_text")],

                [sg.Text('Target U/D', size=(10, 1), justification='left', font='Helvetica 14')

                ,sg.Text('N/A', size=(10, 1), justification='left', font='Helvetica 14',
key="tagret_u_d_text")]],

                title='Training Details',title_color='black', relief=sg.RELIEF_SUNKEN),

                 sg.Frame(layout=[[sg.Button('Forward Pass', size=(15,1), font='Any 14')],

                                  [sg.Button('Corretions', size=(15,1), font='Any 14')],

                                  [sg.Button('Single Train', size=(15,1), font='Any 14')]]

                             ,title='Training Options',title_color='black', relief=sg.RELIEF_SUNKEN),



                 sg.Frame(layout=[[sg.Button('New Data', size=(15,1), font='Any 14')],

                                  [sg.Button('Add Data', size=(15,1), font='Any 14')],

                                  [sg.Button('Train with Data', size=(15,1), font='Any 14')],

                                  [sg.Button('Show Data', size=(15,1), font='Any 14')]]

                             ,title='Data Generation',title_color='black', relief=sg.RELIEF_SUNKEN)

                 ],

 [sg.CButton('Exit', size=(10, 1), font='Helvetica 14')]]


        self.window = sg.Window('Intelligent Aim Bot Control Panel', location=(600,100))

        self.initWindow()

        self.graph_element = self.window.FindElement('graph')

        self.cap = cv.VideoCapture(0)
```

```
    def initWindow(self):
        self.window.Layout(self.layout).Finalize()
##############################################################################
    def CheckEvent(self, event, values ,network):
        if event is 'Exit' or values is None or event=='Exit':
            exit(0)
        elif event == 'Show Data':
            print(network.InputData)
            print("----------------------------")
            print(network.TargetData)
            '''sg.PopupNoWait('Made with PySimpleGUI',
                    'pop window',
                     keep_on_top=True)'''

        if event=='Load weights':
            filename = sg.PopupGetFile('Weights to load', no_window=True)
            if filename!='':
                checkpoint_dir = path.dirname(filename)
                network.LoadWeights(checkpoint_dir)
                #filename = filename[:-6]#remove .index (alternative way without the latest[see IC.py])
                #nn_model.LoadWeights(filename)
```

```
        if event=='Save weights':
            filename = sg.PopupGetFile('Save weihts to file', save_as=True, no_window=True)
            if filename!='':
                network.SaveWeight(filename)

        if event=='Import data':
            filename = sg.PopupGetFile('Data to load', no_window=True)
            network.LoadData(filename)

        if event=='Export data':
            filename = sg.PopupGetFile('Save data to csv', save_as=True, no_window=True)
            network.SaveData(filename)

        if event=='Forward Pass':
            if self.target_position==None:
                print("Please set target.")
            else:
                x_pos_norm=(1 - 0) * ((self.target_position[0] - 0)/(self.image_resolution[0] - 0)) +
0
                y_pos_norm=(1 - 0) * ((self.target_position[1] - 0)/(self.image_resolution[1] - 0)) +
0
                z_pos_norm=0
                dist=0
```

```
if self.is_raspberry:
    dist=self.distance_measure.Measure()
    z_pos_norm=(1 - 0) * ((dist-0)/(100 - 0)) + 0
else:
    z_pos_norm=10 # for testing

feature=np.array([[x_pos_norm, y_pos_norm, z_pos_norm]],dtype="float32")
nn_result=network.ModelPredict(feature,1)
print(nn_result)
self.training_data.position_x_corr=x_pos_norm #self.target_position[0]
self.training_data.position_y_corr=y_pos_norm#self.target_position[1]
'''
self.training_data.position_x=x_pos_norm #self.target_position[0]
self.training_data.position_y=y_pos_norm#self.target_position[1]
self.training_data.position_z=z_pos_norm
self.training_data.target_lr=None
self.training_data.target_ud=None

self.window.FindElement('x_pos_text').Update(self.target_position[0])
self.window.FindElement('y_pos_text').Update(self.target_position[1])
self.window.FindElement('z_pos_text').Update(dist)

self.window.FindElement('tagret_l_r_text').Update("N/A")
```

```
            self.window.FindElement('tagret_u_d_text').Update("N/A")


            if self.is_raspberry:
                self.servo_left_right.SetAngle(self.training_data.position_x)
                self.servo_up_down.SetAngle(self.training_data.position_y)
            '''
            if self.is_raspberry:
                self.training_data.target_lr=nn_result[0][0]
                self.training_data.target_ud=nn_result[0][1]

self.window.FindElement('x_motor_position').Update(self.training_data.target_lr)

self.window.FindElement('y_motor_position').Update(self.training_data.target_ud)
                self.servo_left_right.SetAngle(nn_result[0][0])
                self.servo_up_down.SetAngle(nn_result[0][1])


        if event=='Corretions':
            #proceed with target calculation fuzzy
            if self.predicted_position == None:
                print("Give actual position.")
                return

            self.training_data.position_z=(1 - 0) * ((self.distance_measure.Measure()-0)/(100 - 0))
+ 0
```

```
            self.training_data.target_lr= self.training_data.position_x_corr + self.fuzzX.calculate(
self.target_position[0] - self.predicted_position[0], self.training_data.position_z)

            self.training_data.target_ud= self.training_data.position_y_corr + self.fuzzY.calculate(
self.target_position[1] - self.predicted_position[1] , self.training_data.position_z)


            self.window.FindElement('tagret_l_r_text').Update(self.training_data.target_lr)

            self.window.FindElement('tagret_u_d_text').Update(self.training_data.target_ud)


        if event=='Single Train':

            if( self.training_data.target_lr==None or self.training_data.target_ud==None):

                print("Run corrections first.")

            else:


f=np.array([[self.training_data.position_x_corr,self.training_data.position_y_corr,self.training
_data.position_z]],dtype="float32")

                t=np.array([[self.training_data.target_lr
,self.training_data.target_ud]],dtype="float32")

                network.ModelTrain(f, t, 1)


        #-----------------------------------------------------#

        if event == 'New Data':

            if self.target_position == None:

                print("Please set target.")

                return
```

```
        self.dgX.ResetRate()
        self.dgX.ResetRate()

        x_pos_norm=0
        y_pos_norm=0
        z_pos_norm=0
        distance_cm=0
        if self.is_raspberry:
            distance_cm=self.distance_measure.Measure()
            z_pos_norm=(1 - 0) * ((self.distance_measure.Measure()-0)/(100 - 0)) + 0
            x_pos_norm=(1 - 0) * ((self.target_position[0] - 0)/(self.image_resolution[0] - 0)) +
0
            y_pos_norm=(1 - 0) * ((self.target_position[1] - 0)/(self.image_resolution[1] - 0)) +
0

        self.training_data.position_x=x_pos_norm
        self.training_data.position_y=y_pos_norm
        self.training_data.position_z=z_pos_norm
        #self.training_data.target_lr=0.5
        #self.training_data.target_ud=0.5

        self.window.FindElement('x_pos_text').Update(self.target_position[0])
        self.window.FindElement('y_pos_text').Update(self.target_position[1])
```

```
            self.window.FindElement('z_pos_text').Update(distance_cm)


            self.window.FindElement('tagret_l_r_text').Update("N/A")
            self.window.FindElement('tagret_u_d_text').Update("N/A")


        self.training_data.position_z
        if event == 'New Position':
            if self.target_position == None:
                print("Please set target.")
                return


            if self.predicted_position == None:
                print("Give actual position.")
                return


            self.training_data.target_lr = self.training_data.target_lr -
self.dgX.CalcNewPosition(self.predicted_position[0], self.target_position[0],
self.image_resolution[0])
            self.training_data.target_ud = self.training_data.target_ud -
self.dgY.CalcNewPosition(self.predicted_position[1], self.target_position[1],
self.image_resolution[1])


            self.window.FindElement('x_motor_position').Update(self.training_data.target_lr)
            self.window.FindElement('y_motor_position').Update(self.training_data.target_ud)
```

```
    if(self.training_data.target_lr > 1):
        print("X above 1")
        self.training_data.target_lr = 1

    if(self.training_data.target_ud > 1):
        print("Y above 1")
        self.training_data.target_ud = 1

    if(self.training_data.target_lr < 0):
        print("X below 0")
        self.training_data.target_lr = 0

    if(self.training_data.target_ud < 0):
        print("Y below 0")
        self.training_data.target_ud = 0


    if self.is_raspberry:
        self.servo_left_right.SetAngle(self.training_data.target_lr)
        self.servo_up_down.SetAngle(self.training_data.target_ud)

if event == 'Add Data':
```

```
network.InsertToDataSet(self.training_data.position_x,self.training_data.position_y,self.trainin
g_data.position_z, self.training_data.target_lr,self.training_data.target_ud)


    if event == 'Train with Data':
        network.ModelFullTrain(100)



    if event == 'Set Position':
        x=float(self.window.FindElement('x_motor_position').Get())
        y=float(self.window.FindElement('y_motor_position').Get())
        self.servo_left_right.SetAngle(x)
        self.servo_up_down.SetAngle(y)


        self.training_data.target_lr=x
        self.training_data.target_ud=y
    #-----------------------------------------------------#
    #update coresponding positon
    if event == 'graph':
        mouse = values['graph']
        if mouse != (None, None):
            if self.position_mode_target != None :
                if self.position_mode_target:
                    self.target_position = mouse
```

```
                else:
                    self.predicted_position = mouse
            print(event, mouse)


##########################################################################
    def CheckRadioButton(self):
        if(self.window.FindElement('radio_defuse').Get()):
            self.position_mode_target = None
        elif(self.window.FindElement('radio_target').Get()):
            self.position_mode_target = True
        else:
            self.position_mode_target = False
##########################################################################
    def UpdateWindow(self):
        ret, frame = self.cap.read()
        if frame is None:
            return
        #gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        #cv.imshow('frame',gray)
        #cv.waitKey(20)

        '''img = Image.fromarray(frame)# create PIL image from frame
```

```
        bio = io.BytesIO()          # a binary memory resident stream
        img.save(bio, format= 'PNG')# save image as png to it
        imgbytes = bio.getvalue()   # this can be used by OpenCV hopefully'''
        self.DeleteCurrentObjectsInGraphWindow()

        if self.window.FindElement('auto_detection').Get():
            self.predicted_position = self.lp.GetLaserPosition(frame)
        else:
            self.lp.ClearCvLaserWindows()

        if self.window.FindElement('laser_on_off').Get():
            self.lp.LaserOn()
        else:
            self.lp.LaserOff()

        data = cv.imencode( '.png', frame)[1].tostring()
        self.UpdateGraphWindow(ret,data)
    ##########################################################################
    def DeleteCurrentObjectsInGraphWindow(self):
        if(self.id_frame!=None):
            self.graph_element.DeleteFigure(self.id_frame)
        if(self.id_target!=None):
            self.graph_element.DeleteFigure(self.id_target)
```

```
        if(self.id_predicted!=None):
            self.graph_element.DeleteFigure(self.id_predicted)
##########################################################################
    def UpdateGraphWindow(self, has_frame, frame):
        if has_frame:
            self.id_frame=self.graph_element.DrawImage(data=frame,location=(0,0))
        else:
            print("Failed to capture frame.")
            exit(1)

        if self.target_position!=None:
            self.id_target = self.graph_element.DrawCircle(self.target_position, 3, fill_color='red', line_color="red")
        if self.predicted_position!=None:
            self.id_predicted = self.graph_element.DrawCircle(self.predicted_position, 3, fill_color='blue', line_color="blue")
##########################################################################
```

Κώδικας για το νευρωνικό.

```
import os
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
```

```
#tf.enable_eager_execution()
print("TensorFlow version: {}".format(tf.__version__))
print("Eager execution: {}".format(tf.executing_eagerly()))

class IntelligentControl():
  def __init__(self):
    self.model=self.CreateModel()
    self.BuildModel()
    self.InputData = None
    self.TargetData = None

  def CreateModel(self):
    return tf.keras.Sequential([
    tf.keras.layers.Dense(8, activation=tf.nn.leaky_relu, input_shape=(3,)),
    tf.keras.layers.Dense(8, activation=tf.nn.leaky_relu),
    tf.keras.layers.Dense(2, activation=tf.nn.sigmoid)
    ])

  def BuildModel(self):
    #tf.train.GradientDescentOptimizer(0.1)
    self.model.compile(loss='mse', optimizer=tf.keras.optimizers.SGD(lr=0.5), metrics=['mse'])
```

```
  def NpToTensor(self, data):
    return tf.convert_to_tensor(data)

  def ModelTrain(self, features, target ,step):
    print(features)
    print(target)
    self.model.fit(self.NpToTensor(features), self.NpToTensor(target),steps_per_epoch=step)

  def ModelFullTrain(self, step):
    self.model.fit(self.NpToTensor(self.InputData), self.NpToTensor(self.TargetData),
steps_per_epoch=step)

  def ModelSingleTrain(self, features, target ):
    self.model.train_on_batch(self.NpToTensor(features), self.NpToTensor(target))

  def ModelPredict(self, features, step):
    result=self.model.predict(self.NpToTensor(features), steps=step)
    print(result)
    return result
    #self.model(features)

  def SaveWeight(self,path):
    self.model.save_weights(path)
```

```
  def LoadWeights(self,path):
    last=tf.train.latest_checkpoint(path)
    self.model.load_weights(last)

  #all insert data considered normalized
  def InsertToDataSet(self, x, y, z, left_right_motor_pos, up_down_motor_pos):
    if self.InputData is None:
      self.InputData = np.array([[x, y, z]],dtype="float32")
      self.TargetData =np.array([[left_right_motor_pos,up_down_motor_pos]],dtype="float32")
    else:
      self.InputData = np.insert(self.InputData,0,[[x, y, z]],axis=0)
      self.TargetData
=np.insert(self.TargetData,0,[[left_right_motor_pos,up_down_motor_pos]],axis=0)

  def SaveData(self, file_name):
      result=np.concatenate((self.InputData, self.TargetData), axis=1)
      np.savetxt(file_name + ".csv", result, delimiter='?')

  def LoadData(self, file_name):
    with open(file_name) as f:
      num_lines = sum(1 for line in f)
      f.seek(0)
```

```
    #5 columns= 3 first columns=input ,2 last=target
    temp = np.genfromtxt(f, delimiter='?', dtype="float32").reshape(num_lines,5)
    self.InputData,self.TargetData= np.array_split(temp, [3], axis=1)
```

Κώδικας για μέτρηση απόστασης.

```
import RPi.GPIO as GPIO
import time

class Distance():
  def __init__(self, TRIG, ECHO):
    self.TRIG=TRIG
    self.ECHO=ECHO
    GPIO.setup(self.TRIG, GPIO.OUT)
    GPIO.output(self.TRIG, 0)
    GPIO.setup(self.ECHO, GPIO.IN)
    time.sleep(0.1)

  def Measure(self):
    print("HC-SR04 Starts measure. . .")
    GPIO.output(self.TRIG, 1)
    time.sleep(0.00001)#specs from datasheet
    GPIO.output(self.TRIG, 0)
```

```
        while GPIO.input(self.ECHO)==0:
            pass
        start=time.time()
        while GPIO.input(self.ECHO)==1:
            pass
        stop=time.time()


        elapsed_time=stop-start
        distance=elapsed_time*170*100#100=cm.distance |
(speed_of_sound[m/s])340=2*actual_distance/time_diff
        print("Time elapsed",elapsed_time)
        print("Distance:",distance)
        return distance
```

Κώδικας για έλεγχο μοτέρ.

```
import RPi.GPIO as GPIO
from time import sleep


class ServoActuator():
    def __init__(self, pwm_pin, name):
        self.name=name
        self.pwm_pin=pwm_pin
        #GPIO.setmode(GPIO.BOARD)
        GPIO.setup(pwm_pin, GPIO.OUT, initial=0)
```

```
def SetAngle(self, nn_angle):
    pwm=GPIO.PWM(self.pwm_pin, 50)
    GPIO.output(self.pwm_pin, True)
    sleep(0.5)#wait 0.5 sec for possition set
    pwm.start(self.AngleConversion(nn_angle))
    sleep(0.5)#wait 0.5 sec for possition stabilized
    GPIO.output(self.pwm_pin, False)
    pwm.stop()
    pwm.ChangeDutyCycle(0)

def AngleConversion(self, nn_angle):
    #nn_angle 0-1
    #desired angles 45-135
    nn_angle=1-nn_angle
    angles_deg = (135 - 45) * ((nn_angle - 0)/(1- 0)) + 45
    duty = angles_deg / 18 + 2 #bind to servo specs
    print("=================")
    print(self.name)
    print("angle:", angles_deg)
    print("duty:",duty)
    print("=================")
    return duty
```

Κώδικας για έλεγχο λέιζερ

```
#Detection of red laser

import cv2

import numpy as np

import time

import RPi.GPIO as GPIO


class LaserPosition():

    def __init__(self,laser_pin):

        self.laser_pin=laser_pin

        self.show_windows = False

        self.is_laser_on = False

        GPIO.setup(self.laser_pin, GPIO.OUT)

        GPIO.output(self.laser_pin, 0)


    def GetLaserPosition(self, frame):

        #hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        #lower = np.array([0,0,150])#([155,25,240])

        #upper = np.array([30,30,255])#([179,255,255])


        lower = np.array([0,0,150])#([155,25,240])

        upper = np.array([0,0,255])#([179,255,255])
```

```
    mask = cv2.inRange(frame, lower, upper)


    points = cv2.findNonZero(mask)
    cv2.imshow('hsv', frame)
    cv2.imshow('mask', mask)
    self.show_windows=True
    cv2.waitKey(10)
    if points is not None:
        avg = np.mean(points, axis=0)
        pointInScreen = (round(avg[0][0]), round(avg[0][1]))
        print("Laser position:", pointInScreen)
        return pointInScreen
    else:
        return None


def ClearCvLaserWindows(self):
    if self.show_windows:
        cv2.destroyWindow('hsv')
        cv2.destroyWindow('mask')
        self.show_windows=False


def LaserOn(self):
    if self.is_laser_on == False:
```

```
            GPIO.output(self.laser_pin, 1)
            self.is_laser_on=True


    def LaserOff(self):
        if self.is_laser_on:
            GPIO.output(self.laser_pin, 0)
            self.is_laser_on=False
```

.

Κώδικας για x axis fuzzy.

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl


class FZX():
    def __init__(self,max_error):
        x_error = ctrl.Antecedent(np.arange(-max_error, max_error+1, 1), 'x_error')
        distance = ctrl.Antecedent(np.arange(0, 31, 1), 'distance')
        correction = ctrl.Consequent(np.arange(-1, 2, 1), 'correction')



        #error should be  x_real - x_desired  (10-100) = -90
        x_error.automf(3,names=['left','ok','right'])
```

```
        distance.automf(3,names=['close','mid_close','average','mid_far','far'])
        correction.automf(3,names=['left','ok','right'])#note the reversal (if u sadd in ui the revese)

        rule1 = ctrl.Rule(x_error['right'] & distance['close'], correction['right'])
        rule2 = ctrl.Rule(x_error['right'] & distance['average'], correction['right'])
        rule3 = ctrl.Rule(x_error['right'] & distance['far'], correction['ok'])
        rule4 = ctrl.Rule(x_error['left'] & distance['close'], correction['left'])
        rule5 = ctrl.Rule(x_error['left'] & distance['average'], correction['left'])
        rule6 = ctrl.Rule(x_error['left'] & distance['far'], correction['ok'])
        ruleEx1=ctrl.Rule(x_error['right'] & distance['mid_far'], correction['right'])
        ruleEx2=ctrl.Rule(x_error['right'] & distance['mid_close'], correction['right'])
        ruleEx3=ctrl.Rule(x_error['left'] & distance['mid_far'], correction['left'])
        ruleEx4=ctrl.Rule(x_error['left'] & distance['mid_close'], correction['left'])
        rule7=ctrl.Rule(x_error['ok'], correction['ok'])

        correctionping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7,ruleEx1,ruleEx2,ruleEx3,ruleEx4])
        self.correctionping = ctrl.ControlSystemSimulation(correctionping_ctrl)


    def calculate(self,input_error,input_distance):
        self.correctionping.input['x_error'] = input_error
        self.correctionping.input['distance'] = input_distance
```

```
        self.correctionping.compute()
        print("x",self.correctionping.output['correction'])
        return self.correctionping.output['correction']
```

Κώδικας για y axis fuzzy.

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

class FZY():
    def __init__(self,may_error):
        y_error = ctrl.Antecedent(np.arange(-may_error, may_error+1, 1), 'y_error')
        distance = ctrl.Antecedent(np.arange(0, 31, 1), 'distance')
        correction = ctrl.Consequent(np.arange(-1, 2, 1), 'correction')

        #error should be  y_real - yy_desired  (10-100) = -90
        y_error.automf(3,names=['down','ok','up'])
        distance.automf(3,names=['close','mid_close','average','mid_far','far'])
        correction.automf(3,names=['down','ok','up'])#note the reversal

        rule1 = ctrl.Rule(y_error['up'] & distance['close'], correction['up'])
        rule2 = ctrl.Rule(y_error['up'] & distance['average'], correction['up'])
```

```
        rule3 = ctrl.Rule(y_error['up'] & distance['far'], correction['ok'])


        rule4 = ctrl.Rule(y_error['down'] & distance['close'], correction['down'])
        rule5 = ctrl.Rule(y_error['down'] & distance['average'], correction['down'])
        rule6 = ctrl.Rule(y_error['down'] & distance['far'], correction['ok'])


        ruleEx1=ctrl.Rule(y_error['up'] & distance['mid_far'], correction['up'])
        ruleEx2=ctrl.Rule(y_error['up'] & distance['mid_close'], correction['up'])
        ruleEx3=ctrl.Rule(y_error['down'] & distance['mid_far'], correction['down'])
        ruleEx4=ctrl.Rule(y_error['down'] & distance['mid_close'], correction['down'])


        rule7=ctrl.Rule(y_error['ok'], correction['ok'])


        correctionping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6,
rule7,ruleEx1,ruleEx2,ruleEx3,ruleEx4])
        self.correctionping = ctrl.ControlSystemSimulation(correctionping_ctrl)


    def calculate(self,input_error,input_distance):
        self.correctionping.input['y_error'] = input_error
        self.correctionping.input['distance'] = input_distance
        self.correctionping.compute()
        print("y",self.correctionping.output['correction'])
        return self.correctionping.output['correction']
```

Κώδικας για αναγέννηση δεδομένων (δοκιμαστικός).

```
class DataGeneration():
    def __init__(self):
        self.rate = 1
        self.previous_diff = 0

    def ResetRate(self):
        self.rate = 1

    def CalcNewPosition(self, predicted_position, actual_position, image_resolution):
        diff = predicted_position - actual_position # 0< ->left || 0> -> right


        if (self.previous_diff < 0 and diff > 0) or (self.previous_diff > 0 and diff < 0):
            self.rate = 0.1 * self.rate
```

```
self.previous_diff = diff


normlized_diff = ((1 - 0) * ((diff - 0)/(image_resolution - 0)) + 0) * self.rate

print("Calculated offset: ", normlized_diff)

return normlized_diff
```