

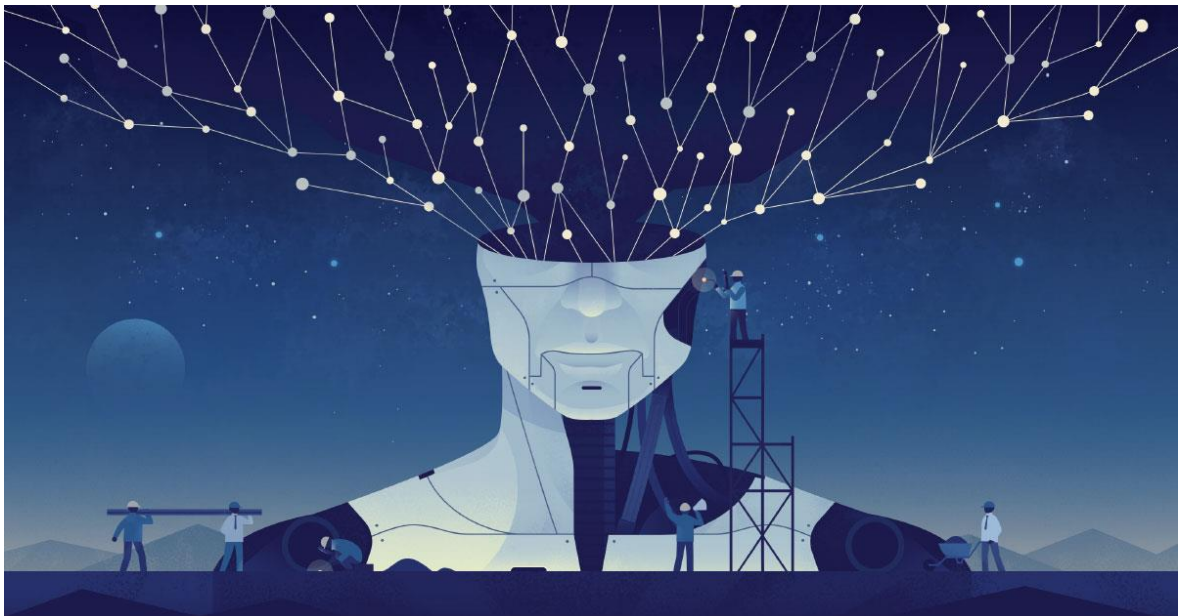


ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Πολιτικών Μηχανικών

Τομέας Υδατικών Πόρων & Περιβάλλοντος

**ΔΙΟΔΕΥΣΗ ΠΛΗΜΜΥΡΩΝ ΜΕ ΧΡΗΣΗ ΤΕΧΝΗΤΩΝ
ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ**



ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Τεπετίδης

Επιβλέπων: Δημήτρης Κουτσογιάννης, Καθηγητής ΕΜΠ

Αθήνα, Νοέμβριος 2020



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Πολιτικών Μηχανικών

Τομέας Υδατικών Πόρων & Περιβάλλοντος

**ΔΙΟΔΕΥΣΗ ΠΛΗΜΜΥΡΩΝ ΜΕ ΧΡΗΣΗ ΤΕΧΝΗΤΩΝ
ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Τεπετίδης

Επιβλέπων: Δημήτρης Κουτσογιάννης, Καθηγητής ΕΜΠ

Αθήνα, Νοέμβριος 2020

Εικόνα εξωφύλλου: Koma Zhang // Quanta Magazine

Cover Page Illustration: Koma Zhang // Quanta Magazine

Νικόλαος Α. Τεπετίδης

Διπλωματούχος Πολιτικός Μηχανικός Ε.Μ.Π.

Copyright © Νικόλαος Τεπετίδης, 2020

Με επιφύλαξη παντός δικαιώματος

Απαγορεύεται η αντιγραφή, αποθήκευση σε αρχείο πληροφοριών, διανομή, αναπαραγωγή, μετάφραση ή μετάδοση της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό, υπό οποιαδήποτε μορφή και με οποιοδήποτε μέσο επικοινωνίας, ηλεκτρονικό ή μηχανικό, χωρίς την προηγούμενη έγγραφη άδεια του συγγραφέα. Επιτρέπεται η αναπαραγωγή, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν στη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Η έγκριση της διπλωματικής εργασίας από τη Σχολή Πολιτικών Μηχανικών του Εθνικού Μετσοβίου Πολυτεχνείου δεν υποδηλώνει αποδοχή των απόψεων του συγγραφέα (Ν. 5343/1932, Άρθρο 202).

Copyright © Nikolaos Tepetidis 2020

All Rights Reserved

Neither the whole nor any part of this diploma thesis may be copied, stored in a retrieval system, distributed, reproduced, translated, or transmitted for commercial purposes, in any form or by any means now or hereafter known, electronic or mechanical, without the written permission from the author. Reproducing, storing and distributing this thesis for non-profitable, educational or research purposes is allowed, without prejudice to reference to its source and to inclusion of the present text. Any queries in relation to the use of the present thesis for commercial purposes must be addressed to its author.

Approval of this diploma thesis by the School of Civil Engineering of the National Technical University of Athens (NTUA) does not constitute in any way an acceptance of the views of the author contained herein by the said academic organization (L. 5343/1932, art. 202).



Νικόλαος Λ. Τεπετίδης

Διπλωματική Εργασία

Διόδευση πλημμυρών με χρήση τεχνητών νευρωνικών δικτύων

Σχολή Πολιτικών Μηχανικών, Εθνικό Μετσόβιο Πολυτεχνείο

Τομέας Υδατικών Πόρων και Περιβάλλοντος

Αθήνα 2020

Nikolaos L. Tepetidis

Diploma Thesis

Flood routing using artificial neural network models

School of Civil Engineering, National Technical University of Athens

Department of Water Resources and Environmental Engineering

Athens 2020

στην οικογένεια μου,

ΕΥΧΑΡΙΣΤΙΕΣ

Το πέρας της παρούσας διπλωματικής εργασίας σηματοδοτεί την ολοκλήρωση των προπτυχιακών σπουδών μου στην σχολή Πολιτικών Μηχανικών του Εθνικού Μετσόβιου Πολυτεχνείου. Θα ήθελα σε αυτό το σημείο να ευχαριστήσω τους ανθρώπους που συνέβαλλαν στην εκπόνηση της.

Αρχικά θα ήθελα να ευχαριστήσω ιδιαίτερος τον επιβλέποντα καθηγητή της εργασίας μου κ. Δημήτρη Κουτσογιάννη, ο οποίος με εμπιστεύτηκε από την πρώτη στιγμή, με ενθάρρυνε και με βοήθησε να ασχοληθώ με το θέμα της διπλωματικής. Με την καθοριστική καθοδήγησή του δημιουργήθηκε το πλαίσιο της εργασίας και με έφερε κοντά σε ανθρώπους, οι οποίοι με βοήθησαν να πετύχω τον σκοπό μου. Αισθάνομαι ιδιαίτερα τυχερός και ευγνώμων που συνεργάστηκα με τον κ. Κουτσογιάννη και θα ήταν μεγάλη μου χαρά να συνεργαστώ ξανά στο μέλλον.

Ένας άνθρωπος στον οποίο οφείλω ένα μεγάλο ευχαριστώ είναι ο Γιάννης Καλογεράς, Διδάκτορας της σχολής Π.Μ., που δίχως την πολύτιμη βοήθεια του δεν θα ήταν δυνατή η ολοκλήρωση της εργασίας. Από την αρχή με προέτρεψε να ασχοληθώ με τα τεχνητά νευρωνικά δίκτυα και με την συμβολή του με έκανε να τα κατανοήσω και να τα εφαρμόσω στην πράξη.

Ακόμη ευχαριστώ τον κ. Παναγιώτη Δημητριάδη, Διδάκτορα της σχολής Π.Μ. και την κα. Άννυ Ηλιοπούλου, Διδάκτορα της σχολής Π.Μ., για την σημαντική συνεισφορά τους στην εκπόνηση της διπλωματικής. Η συνεργασία μου με όλους ήταν άριστη, παρά την εξ αποστάσεως επίβλεψη της πορείας της εργασίας λόγω της ασθένειας COVID-19.

Επιπλέον θα ήθελα να ευχαριστήσω όλους τους καθηγητές της σχολής Π.Μ. για τις γνώσεις, τις αξίες και τις εμπειρίες που μου μετέφεραν, και κυρίως τον κ. Παναγιώτη Παπανικολάου, αναπληρωτή καθηγητή του ΕΜΠ, για τη βοήθειά του όποτε του ζητήθηκε.

Δεν θα μπορούσα να μην ευχαριστήσω τους φίλους μου, με τους οποίους πέρασα ευχάριστα αυτά τα πέντε χρόνια σπουδών και με τους οποίους συνεργάστηκα σε πολλές ομαδικές εργασίες.

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου, τους γονείς και τον αδελφό μου, για την συνεχή υποστήριξη που μου παρείχαν με κάθε τρόπο.

Νικόλαος Τεπετίδης

Αθήνα,
Νοέμβριος 2020

ΠΕΡΙΛΗΨΗ

Εξετάζεται η συνεισφορά των τεχνητών νευρωνικών δικτύων στην επίλυση προβλημάτων πολιτικού μηχανικού και πιο συγκεκριμένα στο πρόβλημα της διόδευσης πλημμύρας, δηλαδή το πρόβλημα της μαθηματικής αναπαράστασης της εξέλιξης ενός πλημμυρικού φαινομένου στο χώρο και στο χρόνο. Οι ροές που παρατηρούνται στη φύση είναι εν γένει μη μόνιμες, έτσι και τα πλημμυρικά φαινόμενα, τα οποία αποτελούν ένα πολύ σημαντικό επιστημονικό θέμα.

Αρχικά μελετάται η απόδοση ενός απλού πολυεπίπεδου τεχνητού νευρωνικού δικτύου για την εκτίμηση της διόδευσης στον ποταμό Πηνειό. Παράλληλα εισάγεται μια νέα μεθοδολογία επίλυσης σύνθετων προβλημάτων, τα οποία περιγράφονται από μη γραμμικές μερικές διαφορικές εξισώσεις. Ειδικότερα η μεθοδολογία αφορά τα τεχνητά νευρωνικά δίκτυα τα οποία υπακούν νόμους της φυσικής (γνωστά ως PINN). Ανάλογα με τη φύση του προβλήματος και τα διαθέσιμα δεδομένα, διακρίνονται δύο κύριες κατηγορίες προβλημάτων: η επίλυση της εξίσωσης που περιγράφει το πρόβλημα και η εύρεση των παραμέτρων της εξίσωσης για γνωστή τη λύση.

Όπως τα περισσότερα προβλήματα του πολιτικού μηχανικού έτσι και η διόδευση των πλημμυρών περιγράφεται ικανοποιητικά από μερικές διαφορικές εξισώσεις, γνωστές ως εξισώσεις Saint-Venant, οι οποίες δεν επιδέχονται αναλυτικής λύσης. Η αξιοπιστία της μεθόδου ελέγχεται επιλύοντας την εξίσωση Burgers, για την οποία η αναλυτική λύση είναι διαθέσιμη. Από τα τρία μοντέλα κυμάτων που προσφέρουν οι εξισώσεις Saint-Venant, αναλύεται το μοντέλο του κινηματικού κύματος μέσω μίας εφαρμογής, ενώ παρουσιάζεται και η λύση που προκύπτει από μία αριθμητική μέθοδο πεπερασμένων διαφορών. Παρατηρείται πως τα δίκτυα PINN μπορούν να εκτιμήσουν με μεγάλη ακρίβεια τόσο την λύση της διαφορικής εξίσωσης όσο και τις παραμέτρους αυτής (αντίστροφο πρόβλημα).

Λέξεις κλειδιά: τεχνητά νευρωνικά δίκτυα, διόδευση πλημμύρας, πολυεπίπεδο ΤΝΔ, τεχνητά νευρωνικά δίκτυα που υπακούν νόμους της φυσικής (PINN), μη γραμμικές διαφορικές εξισώσεις, εύρεση παραμέτρων, εξισώσεις Saint-Venant, μοντέλο κινηματικού κύματος.

ABSTRACT

We study the application of artificial neural networks for addressing civil engineering problems and more specifically flood routing which constitutes finding a mathematical representation of the spatio-temporal evolution of flood phenomena. In general, the flows observed in nature are unsteady and flood flows are no exception and thus generating a crucial scientific endeavor.

Firstly, we study the performance of a multilayer neural network in estimating routing for the river Pinios. Parallely, we introduce a new methodology for solving complex problems which are described by nonlinear partial differential equations. Namely, the methodology concerns the utilization of artificial neural networks that obey any given law of physics (known as Physics Informed Neural Networks or PINN). Based on the nature of the problem and the available data, we determine two classes of problems: those that aim at finding the solution of the equation that describes the physical problem and those that aim at estimating the parameters of the equation for a given known solution.

As in most of the civil engineering problems, flood routing is accurately described by the partial differential equations, known as Saint-Venant equations, which do not have an analytical solution. The robustness of our method is examined by solving the Burgers equation, for which an analytical solution is available. From the three wave models that are derived by the Saint-Venant equations, we analyze the model of the kinematic wave through an application, while we also compare with a solution obtained through a numerical finite differences method. We observe that PINNs can both estimate the solution of this differential equation and inversely estimate its parameters given the solution, with high accuracy.

Keywords: artificial neural networks (ANN), flood routing, multilayer artificial neural network, physics informed neural networks (PINN), non-linear differential equations, parameter estimation, Saint-Venant equations, kinematic wave model.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΧΑΡΙΣΤΙΕΣ.....	x
ΠΕΡΙΛΗΨΗ.....	xii
ABSTRACT.....	xiii
ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ.....	xvi
ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ.....	xviii
1 ΕΙΣΑΓΩΓΗ	1
1.1 Κοινοτική Οδηγία 2007/60 για τις πλημμύρες	1
1.2 Πλημμύρες	2
1.3 Διόδευση πλημμυρών.....	3
1.4 Αντικείμενο της εργασίας	5
1.5 Διάρθρωση εργασίας.....	5
2 ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ.....	7
2.1 Εισαγωγή στα νευρωνικά δίκτυα	7
2.2 Από τους βιολογικούς στους τεχνητούς νευρώνες.....	7
2.3 Συνάρτηση ενεργοποίησης (activation function).....	11
2.4 Πολυεπίπεδα τεχνητά νευρωνικά δίκτυα (Multilayer Perceptron).....	13
2.5 Εκπαίδευση τεχνητών νευρωνικών δικτύων (Training).....	14
2.6 Αλγόριθμοι Βελτιστοποίησης.....	14
2.6.1 Κατάβαση κλίσης (Gradient Descent).....	14
2.6.2 Οπισθοδιάδοση (backpropagation).....	16
2.6.3 Στοχαστική κατάβαση κλίσης (Stochastic gradient descent).....	18
2.6.4 Αλγόριθμος AdaGrad	19
2.6.5 Αλγόριθμος Adam.....	19
2.6.6 Αλγόριθμος L-BFGS.....	20
3 ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΠΟΥ ΥΠΑΚΟΥΝ ΝΟΜΟΥΣ ΤΗΣ ΦΥΣΙΚΗΣ (PHYSICS INFORMED NEURAL NETWORKS - PINN)	21
3.1 Εισαγωγή.....	21

3.2	Γενική περιγραφή και ορισμός του προβλήματος	21
3.3	Δεδομένα εισόδου και εκπαίδευση	22
3.4	Ταυτοποίηση παραμέτρων διαφορικής εξίσωσης (parameter identification)	24
4	ΓΕΝΙΚΕΣ ΕΞΙΣΩΣΕΙΣ ΜΗ ΜΟΝΙΜΗΣ ΡΟΗΣ ΜΕ ΕΛΕΥΘΕΡΗ ΕΠΙΦΑΝΕΙΑ.....	26
4.1	Μη μόνιμη ροή – Γενικά	26
4.2	Εξίσωση συνέχειας	27
4.3	Εξίσωση ορμής	28
4.4	Σχολιασμός των εξισώσεων Saint Venant	30
4.5	Τύποι κυμάτων μη μόνιμης ροής.....	31
4.6	Μοντέλο κινηματικού κύματος	32
5	ΕΦΑΡΜΟΓΗ MLP ΣΤΟΝ ΠΟΤΑΜΟ ΠΗΝΕΙΟ	35
5.1	Περιοχή μελέτης.....	35
5.2	Επεξεργασία δεδομένων	36
5.3	Αποτελέσματα.....	39
6	ΕΦΑΡΜΟΓΕΣ PINN.....	42
6.1	Εφαρμογή 1: Εξίσωση Burgers (Burgers' Equation).....	42
6.2	Εφαρμογή 2: Εξίσωση κινηματικού κύματος (Kinematic wave equation).....	51
7	ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΤΑΣΕΙΣ ΓΙΑ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ.....	58
7.1	Συμπεράσματα.....	58
7.2	Προτάσεις για μελλοντική έρευνα	59
8	ΑΝΑΦΟΡΕΣ.....	60
	ΠΑΡΑΡΤΗΜΑ	64
	Κώδικας MLP εφαρμογής Πηνειού.....	64
	Κώδικας PINN λύσης εξίσωσης Burgers	66
	Κώδικας PINN εύρεσης παραμέτρου εξίσωσης Burgers.....	72
	Κώδικας PINN λύσης εξίσωσης κινηματικού κύματος	76
	Κώδικας PINN εύρεσης παραμέτρου εξίσωσης κινηματικού κύματος	81

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

Σχήμα 1.1: Εικόνα από την καταστροφική πλημμύρα στην Μάνδρα (Μάνδρα, 2017).	3
Σχήμα 2.1 : Βιολογικός νευρώνας / Από User:Dhp1080, translated and modified by Badseed. - Originally Neuron.jpg taken from the US Federal (public domain) (Nerve Tissue, retrieved March 2007), redrawn by User:Dhp1080 in Illustrator. Source: "Anatomy and Physiology" by the US National Cancer Institute's Surveillance, Epidemiology and End Results (SEER) Program., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24139135	8
Σχήμα 2.2: Πολλαπλά επίπεδα σε ένα βιολογικό νευρωνικό δίκτυο.	9
Σχήμα 2.3: Μορφή τεχνητού νευρωνικού δικτύου Perceptron.	10
Σχήμα 2.4: Συναρτήσεις ενεργοποίησης και οι παράγωγοι τους (Πηγή: Aurélien Géron, 2019).	12
Σχήμα 2.5: Πολυεπίπεδο ΤΝΔ (Multi-layer Perceptron).	13
Σχήμα 2.6: Σχηματική απεικόνιση αλγορίθμου κατάβασης κλίσης (Gradient descent). ...	16
Σχήμα 2.7: Παράδειγμα επιλογής μεγάλου (αριστερά) και μικρού (δεξιά) ρυθμού μάθησης.	16
Σχήμα 3.1: Σχηματική απεικόνιση σημείων εκπαίδευσης ΤΝΔ που επιλύει διαφορική εξίσωση.....	24
Σχήμα 4.1: Επεξηγηματικό σχήμα για τα χαρακτηριστικά μεγέθη της ροής με ελεύθερη επιφάνεια για την εξίσωση συνέχειας.....	27
Σχήμα 4.2: Επεξηγηματικό σχήμα με τα χαρακτηριστικά μεγέθη για την εξίσωση ορμής (Πηγή: Παπανικολάου, 2017).....	28
Σχήμα 4.3: Μεταβολή της μορφής ενός πλημμυρικού κύματος.	34
Σχήμα 5.1: Υδατικό Διαμέρισμα Θεσσαλίας (Πηγή: www.ypethe.gr).	35
Σχήμα 5.2: Περιοχή μελέτης ποταμού Πηνειού (Πηγή: www.anavasi.gr).	36
Σχήμα 5.3: Ιστορική χρονοσειρά κοινών ημερών των δύο σταθμών του ποταμού Πηνειού.	37
Σχήμα 5.4: Χρονοσειρά εισροών και εκροών που χρησιμοποιήθηκε για την εκπαίδευση του ΤΝΔ.	39
Σχήμα 5.5: Απόδοση εκπαίδευσης ΤΝΔ αναλύοντας τη συνάρτηση κόστους στη διάρκεια των εποχών.	40
Σχήμα 5.6: Απεικόνιση μετρούμενης παροχής εισόδου, μετρούμενης παροχής εξόδου και εκτιμώμενης παροχής εξόδου στον ποταμό Πηνειό.	41
Σχήμα 5.7: Σύγκριση εκτιμώμενης παροχής εξόδου μοντέλου ΤΝΔ και μετρήσεων στον ποταμό Πηνειό.....	41
Σχήμα 6.1: Τρισδιάστατη απεικόνιση αναλυτικής λύσης εξίσωσης Burgers.	45
Σχήμα 6.2: Τρισδιάστατη απεικόνιση της λύσης που προέκυψε από το τεχνητό νευρωνικό δίκτυο για την εξίσωση Burgers.	45

Σχήμα 6.3: (α) Δισδιάστατη απεικόνιση αναλυτικής λύσης εξίσωσης Burgers, (β) Δισδιάστατη απεικόνιση εκτιμώμενης λύσης εξίσωσης Burgers.....	46
Σχήμα 6.4: Σύγκριση αναλυτικής και εκτιμώμενης λύση για $t=0.25$	47
Σχήμα 6.5: Σύγκριση αναλυτικής και εκτιμώμενης λύσης για $t=0.50$	47
Σχήμα 6.6: Σύγκριση αναλυτικής και εκτιμώμενης λύσης για $t=0.75$	48
Σχήμα 6.7: Αποτέλεσμα ΤΝΔ ταυτοποίησης της παραμέτρου κινηματικού ιξώδες ν της εξίσωσης Burgers.	50
Σχήμα 6.8: Διόδευση κινηματικού κύματος με χρήση γραμμικών πεπερασμένων διαφορών.	53
Σχήμα 6.9: Διόδευση κινηματικού κύματος με χρήση τεχνητού νευρωνικού δικτύου.....	55
Σχήμα 6.10: Αποτέλεσμα ΤΝΔ ταυτοποίησης της παραμέτρου α της εξίσωσης του κινηματικού κύματος.....	57

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

Πίνακας 2.1: Συναρτήσεις ενεργοποίησης (Πηγή: Sebastian Raschka, 2016).....	12
Πίνακας 5.1: Κοινές ημέρες χρονοσειρών.	37
Πίνακας 5.2: Κοινές ημέρες που χρησιμοποιήθηκαν για την εκπαίδευση του TND.	38
Πίνακας 5.3: Αρχιτεκτονική TND για την εφαρμογή στον ποταμό Πηνειό.	39
Πίνακας 6.1: Αρχιτεκτονική TND εφαρμογής εξίσωσης Burgers.	48
Πίνακας 6.2: Γεωμετρικά δεδομένα αγωγού εφαρμογής κινηματικού κύματος.....	52
Πίνακας 6.3: Υδρογράφημα εισόδου	52
Πίνακας 6.4: Αρχιτεκτονική TND εφαρμογής κινηματικού κύματος.....	55

1 ΕΙΣΑΓΩΓΗ

1.1 Κοινοτική Οδηγία 2007/60 για τις πλημμύρες

Τα πλημμυρικά φαινόμενα αποτελούν ένα μείζονος σημασίας πρόβλημα για την ανθρωπότητα με καταστροφικές συνέπειες για αυτήν. Λαμβάνοντας υπόψη το ανωτέρω, η Ευρωπαϊκή Ένωση μέσω του Ευρωπαϊκού Κοινοβουλίου εξέδωσε στις 23 Οκτωβρίου του 2007 την Οδηγία 2007/60/EK για την αξιολόγηση και τη διαχείριση των κινδύνων πλημμύρας (assessment and management of flood risk). Σύμφωνα με το άρθρο 1 (2007/60/EK) των γενικών διατάξεων, σκοπός της οδηγίας αυτής είναι η θέσπιση ενός πλαισίου για την αξιολόγηση και τη διαχείριση των κινδύνων πλημμύρας, με στόχο τη μείωση των αρνητικών συνεπειών στην ανθρώπινη υγεία, το περιβάλλον, την πολιτιστική κληρονομιά και τις οικονομικές δραστηριότητες που συνδέονται με τις πλημμύρες στην Κοινότητα.

Η Οδηγία 2007/60/EK διακρίνει τρία στάδια για την διαδικασία διαχείρισης του κινδύνου πλημμύρας. Αρχικά εφαρμόζεται η προκαταρκτική αξιολόγηση των κινδύνων πλημμύρας, στην οποία τα κράτη μέλη υποχρεούνται να προσδιορίσουν τις περιοχές εκείνες για τις οποίες συμπεραίνουν ότι υπάρχουν δυνητικοί σοβαροί κίνδυνοι πλημμύρας. Στη συνέχεια, εκπονούνται οι χάρτες επικινδυνότητας πλημμύρας και οι χάρτες κινδύνων πλημμύρας σε επίπεδο λεκάνης απορροής ποταμού. Οι χάρτες επικινδυνότητας πλημμύρας καλύπτουν τις γεωγραφικές περιοχές που θα μπορούσαν να πλημμυρίσουν σύμφωνα με τα ακόλουθα σενάρια:

- (α) πλημμύρες χαμηλής πιθανότητας ή σενάρια ακραίων φαινομένων.
- (β) πλημμύρες μέσης πιθανότητας (με περίοδο επαναληπτικότητας ≥ 100 χρόνια).
- (γ) πλημμύρες υψηλής πιθανότητας ανάλογα με την περίπτωση.

Για κάθε ένα από τα παραπάνω σενάρια παρατίθενται και η έκταση της πλημμύρας, το βάθος νερού και η ταχύτητα ροής των υδάτων. Οι χάρτες κινδύνου πλημμύρας περιγράφουν τις δυνητικές αρνητικές συνέπειες που συνδέονται με τις πλημμύρες και εκφράζονται μέσω του αριθμού του πληθυσμού που ενδέχεται να πληγεί, της οικονομικής δραστηριότητας στην περιοχή που ενδέχεται να πληγεί, των εγκαταστάσεων που ενδέχεται να προκαλέσουν τυχαία ρύπανση σε περίπτωση πλημμύρας και όποια άλλη πληροφορία το κράτος μέλος

θεωρεί χρήσιμη. Στο τρίτο και τελευταίο στάδιο, τα κράτη μέλη καταρτίζουν σχέδια διαχείρισης των κινδύνων πλημμύρας σε επίπεδο λεκάνης απορροής. Τα σχέδια αυτά περιλαμβάνουν μέτρα που εστιάζουν στη μείωση των δυσμενών συνεπειών που οι πλημμύρες έχουν για την ανθρώπινη υγεία, το περιβάλλον, την πολιτιστική κληρονομιά και την οικονομική δραστηριότητα, όπως επίσης και στη μείωση της πιθανότητας πλημμύρας.

Η Οδηγία 2007/60/EK δεν περιορίζεται μόνο στα τρία προαναφερθέντα στάδια, αλλά επισημαίνει την ανάγκη εναρμονισμού με παλιότερες νομοθετικές διατάξεις. Πιο συγκεκριμένα τα κράτη μέλη πρέπει να λάβουν μέτρα για να συντονίσουν την εφαρμογή της Οδηγίας 2007/60/EK και της οδηγίας 2000/60/EK, η οποία αποτελεί την Οδηγία Πλαίσιο για τα Νερά 2000/60/EK. Επιπλέον όλη η διαδικασία των τριών σταδίων, σύμφωνα με το άρθρο 10 (2007/60/EK) πρέπει να είναι διαθέσιμη στο κοινό.

1.2 Πλημμύρες

Ως υδρολογία πλημμυρών (flood hydrology) νοείται ο κλάδος της τεχνικής (engineering) υδρολογίας που εστιάζει στις διεργασίες που λαμβάνουν χώρα από την έναρξη μιας ισχυρής καταιγίδας μέχρι το πέρας της απορροής που οφείλεται στο συγκεκριμένο επεισόδιο βροχής. Η θεμελιώδης διαφορά σε σχέση με το ευρύτερο πεδίο έρευνας της υδρολογίας αφορά στη χρονική κλίμακα μελέτης, η οποία ταυτίζεται με τη διάρκεια εξέλιξης του επεισοδίου. Στη διαδικασία αυτή δίνεται έμφαση στην επιφανειακή απορροή η οποία έχει ως συνέπεια την αύξηση της διερχόμενης παροχής του ποταμού.

Σύμφωνα με το άρθρο 2 των γενικών διατάξεων της οδηγίας 2007/60/EK ως *πλημμύρα* ορίζεται «η προσωρινή κάλυψη από νερό εδάφους το οποίο, υπό φυσιολογικές συνθήκες, δεν καλύπτεται από νερό». Αυτό περιλαμβάνει πλημμύρες από ποτάμια, ορεινούς χείμαρρους, εφήμερα ρεύματα και πλημμύρες από την θάλασσα σε παράκτιες περιοχές.

Η εξέλιξη μιας πλημμύρας στο χώρο και το χρόνο εξαρτάται από τρεις παράγοντες:

- τη χωροχρονική εξέλιξη του επεισοδίου βροχής.
- τα φυσιογραφικά χαρακτηριστικά της λεκάνης απορροής.
- τα υδραυλικά χαρακτηριστικά του υδρογραφικού δικτύου.

Τα υδρολογικά και υδραυλικά μεγέθη που ενδιαφέρουν είναι:

- η παροχή αιχμής.

- ο χρόνος εμφάνισης της αιχμής.
- η συνολική απορροή που παράγεται κατά την πλημμύρα.
- η χρονική διάρκεια εξέλιξης του φαινομένου
- το μέγιστο βάθος και οι ταχύτητες ροής που αναπτύσσονται κατά μήκος και εγκάρσια του υδατορεύματος.
- η κατακλύσιμη έκταση και οι συναφείς επιπτώσεις

Οι συνέπειες των πλημμυρικών επεισοδίων είναι καταστροφικές για την ανθρώπινη ύπαρξη. Οι πλημμύρες μπορούν να προκαλέσουν θανάτους, μετακινήσεις πληθυσμών και ζημιές στο περιβάλλον, να θέσουν σε σοβαρό κίνδυνο την οικονομική ανάπτυξη και να υπονομεύσουν τις οικονομικές δραστηριότητες σε μια περιοχή. Στο Σχήμα 1 φαίνεται μια αεροφωτογραφία από την πρόσφατη καταστροφική πλημμύρα στην περιοχή της Μάνδρας Αττικής το 2017.



Σχήμα 1.1: Εικόνα από την καταστροφική πλημμύρα στην Μάνδρα (Μάνδρα, 2017).

1.3 Διόδευση πλημμυρών

Οι ροές που παρατηρούνται στη φύση είναι εν γένει μη μόνιμες. Ένα σημαντικό πρόβλημα μη μόνιμης ροής που καλείται να αντιμετωπίσει ένας μηχανικός είναι η κίνηση ενός πλημμυρικού κύματος κατά μήκος ενός ανοικτού αγωγού, συνήθως φυσικού ποταμού. Τα πλημμυρικά κύματα αποτελούν φαινόμενα μη μόνιμης ροής. Η διόδευση μιας πλημμύρας, δηλαδή το πρόβλημα της μαθηματικής αναπαράστασης της εξέλιξης ενός πλημμυρικού φαινομένου στο χώρο και στο χρόνο (Κουτσογιάννης, 2011), απαιτείται για την

αντιμετώπιση και την πρόβλεψη πλημμυρικών φαινομένων. Συνήθως στις μελέτες πρόβλεψης πλημμύρας, η διαδικασία που ζητείται είναι ο προσδιορισμός του υδρογραφήματος σε μία διατομή κατάντη του αγωγού, γνωρίζοντας το υδρογράφημα σε μία ανάντη διατομή.

Όπως τα περισσότερα προβλήματα τα οποία διαχειρίζεται ο πολιτικός μηχανικός, έτσι και το φαινόμενο της διόδευσης πλημμύρας, περιγράφεται από μερικές διαφορικές εξισώσεις. Οι μέθοδοι διόδευσης πλημμύρας διακρίνονται σε δύο μεγάλες κατηγορίες, στις υδρολογικές και τις υδραυλικές. Οι υδρολογικές μέθοδοι βασίζονται κυρίως στην εξίσωση συνέχειας (διατήρηση της μάζας) και σε προσεγγιστικές σχέσεις μεταξύ των παροχών που εισρέουν και εκρέουν σε ένα τμήμα του αγωγού και του αποθηκευμένου σε αυτό όγκου νερού (Νουτσόπουλος κ.ά., 2010). Από την άλλη, οι υδραυλικές μέθοδοι κάνουν χρήση των μη γραμμικών μερικών διαφορικών εξισώσεων μη μόνιμης ροής, γνωστών και ως εξισώσεων Saint-Venant. Η επίτευξη αναλυτικών λύσεων των εξισώσεων αυτών είναι σχεδόν αδύνατη, λόγω της πολυπλοκότητας και της μη γραμμικότητας που τις διέπει. Για το λόγο αυτό συνήθως χρησιμοποιούνται προσεγγιστικές λύσεις που προκύπτουν μέσω αριθμητικών σχημάτων.

Εκτός από τις παραπάνω μεθόδους, η συνεχής ανάπτυξη της υπολογιστικής τεχνητής νοημοσύνης δίνει την δυνατότητα δημιουργίας μοντέλων τα οποία μπορούν να εφαρμοστούν στην πρόβλεψη πλημμυρών. Τέτοια μοντέλα αποτελούν τα *τεχνητά νευρωνικά δίκτυα (TND)* με ευρεία εφαρμογή σε τομείς όπως η μηχανική, η χρηματοοικονομική, η βιολογία και πολλοί ακόμη. Η χρήση TND σε συστήματα διόδευσης πλημμυρών έχουν αποδείξει ότι αποτελούν μια έμπιστη και αποτελεσματική εναλλακτική των παραδοσιακών μεθόδων. Τα περισσότερα TND που ασχολήθηκαν με την διόδευση πλημμυρών χρησιμοποίησαν δεδομένα εισόδου και εξόδου από παλιότερες χρονολογικά μετρήσεις (π.χ. παροχών), για την εκπαίδευση των μοντέλων. Τα TND κατόρθωσαν να ξεπεράσουν την πολυπλοκότητα των εξισώσεων που διέπει το φαινόμενο και να καταλήξουν σε αξιόπιστα αποτελέσματα. Το μειονέκτημα της πλήρους εξάρτησης του μοντέλου από τα δεδομένα που δέχεται, έρχεται να απομακρύνει μια σχετικά καινούργια κατηγορία τεχνητών νευρωνικών δικτύων. Αυτά τα τεχνητά νευρωνικά δίκτυα, γνωστά και ως *PINN (Physics Informed Neural Networks)* δίνουν την δυνατότητα επίλυσης πολύπλοκων προβλημάτων, σεβόμενα οποιοδήποτε δοσμένο νόμο της φυσικής ο οποίος περιγράφεται από μια μη γραμμική μερική διαφορική εξίσωση. Πιο συγκεκριμένα οι εξισώσεις Saint-Venant μπορούν να επιλυθούν μέσω ενός *PINN*, αντί για την εφαρμογή των προσεγγιστικών θεωρήσεων των αριθμητικών

μεθόδων. Τα *PINN* συνθέτουν ένα σημαντικό εργαλείο στην αντιμετώπιση δύσκολων μαθηματικών εξισώσεων, καθώς σε αντίθεση με τις κλασικές αριθμητικές μεθόδους η επίλυση επιτυγχάνεται δίχως καμία διακριτοποίηση στο χώρο και στο χρόνο.

1.4 Αντικείμενο της εργασίας

Η παρούσα εργασία ασχολείται με την χρήση των τεχνητών νευρωνικών δικτύων στην επίλυση προβλημάτων διόδευσης πλημμυρών. Πιο συγκεκριμένα περιγράφεται η φιλοσοφία που διέπει τα ΤΝΔ και το πως αυτά μπορούν να εφαρμοστούν σε προβλήματα πολιτικού μηχανικού. Για τον έλεγχο τους εφαρμόζεται ένα πολυεπίπεδο ΤΝΔ στη διόδευση ενός κομματιού του Πηνειού ποταμού, για το οποίο υπάρχουν πραγματικές μετρήσεις.

Ακόμη εισάγεται μια σχετικά καινούργια κατηγορία τεχνητών νευρωνικά δικτύων, τα ονομαζόμενα *PINN*, τα οποία εκπαιδεύονται για να λύσουν οποιοδήποτε πρόβλημα υπακούει σε ένα νόμο της φυσικής μέσω μιας μη γραμμικής μερικής διαφορικής εξίσωσης. Τα δίκτυα αυτά εφαρμόζονται στην παρούσα εργασία για να λύσουν τις μη γραμμικές μερικές διαφορικές εξισώσεις που περιγράφουν το φαινόμενο της διόδευσης πλημμύρας, γνωστές και ως εξισώσεις Saint-Venant. Η εργασία αναλύει μόνο την εξίσωση του κινηματικού κύματος, ενώ παρουσιάζει την δυναμική των *PINN* μέσω της λύσης της εξίσωσης Burgers. Δίνεται έμφαση και στην δυνατότητα των *PINN* για την εύρεση των παραμέτρων της διαφορικής εξίσωσης εάν η λύση της είναι γνωστή με οποιοδήποτε τρόπο.

1.5 Διάρθρωση εργασίας

Εκτός του παρόντος κεφαλαίου, όπου εισάγεται η έννοια της διόδευσης των πλημμυρών αλλά και το γενικό πλαίσιο στο οποίο κινείται η διπλωματική, η εργασία αποτελείται από έξι επιπλέον κεφάλαια, καθώς και από το παράρτημα με τους κώδικες που αναπτύχθηκαν.

Στο Κεφάλαιο 2 περιγράφονται τα τεχνητά νευρωνικά δίκτυα, αναλύοντας τόσο τη δομή και τη λειτουργία τους όσο και τη προέλευση τους.

Στο Κεφάλαιο 3 παρουσιάζονται τα νευρωνικά δίκτυα τα οποία υπακούν νόμους της φυσικής (Physics Informed Neural Networks – *PINN*). Ορίζεται το γενικό πρόβλημα και η αντιμετώπιση του μέσω των *PINN*, καθώς και η δυνατότητα που προσφέρουν για την εύρεση παραμέτρων της εξίσωσης.

Στο Κεφάλαιο 4 περιγράφονται οι εξισώσεις μη μόνιμης ροής με ελεύθερη επιφάνεια και αναλύεται περαιτέρω η εξίσωση του κινηματικού κύματος.

Στο Κεφάλαιο 5 παρουσιάζεται η εφαρμογή ενός απλού πολυεπίπεδου τεχνητού νευρωνικού δικτύου (MLP), για την εκτίμηση της διόδευσης της πλημμύρας στον ποταμό Πηνειό και συγκρίνεται με τις πραγματικές μετρήσεις.

Στο Κεφάλαιο 6 παρουσιάζονται οι εφαρμογές των δικτύων PINN για την επίλυση της εξίσωσης Burgers και της εξίσωσης του κινηματικού κύματος. Σε συνέχεια των εφαρμογών επιλύεται και το αντίστροφο πρόβλημα για την εύρεση της παραμέτρου της εξίσωσης με γνωστή τη λύση.

Στο Κεφάλαιο 7 συνοψίζονται τα συμπεράσματα που προκύπτουν από την παρούσα εργασία και δίνονται ενδεικτικές προτάσεις για μελλοντική έρευνα.

Τέλος στο παράρτημα παρατίθενται οι κώδικες των τριών εφαρμογών.

2 ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

2.1 Εισαγωγή στα νευρωνικά δίκτυα

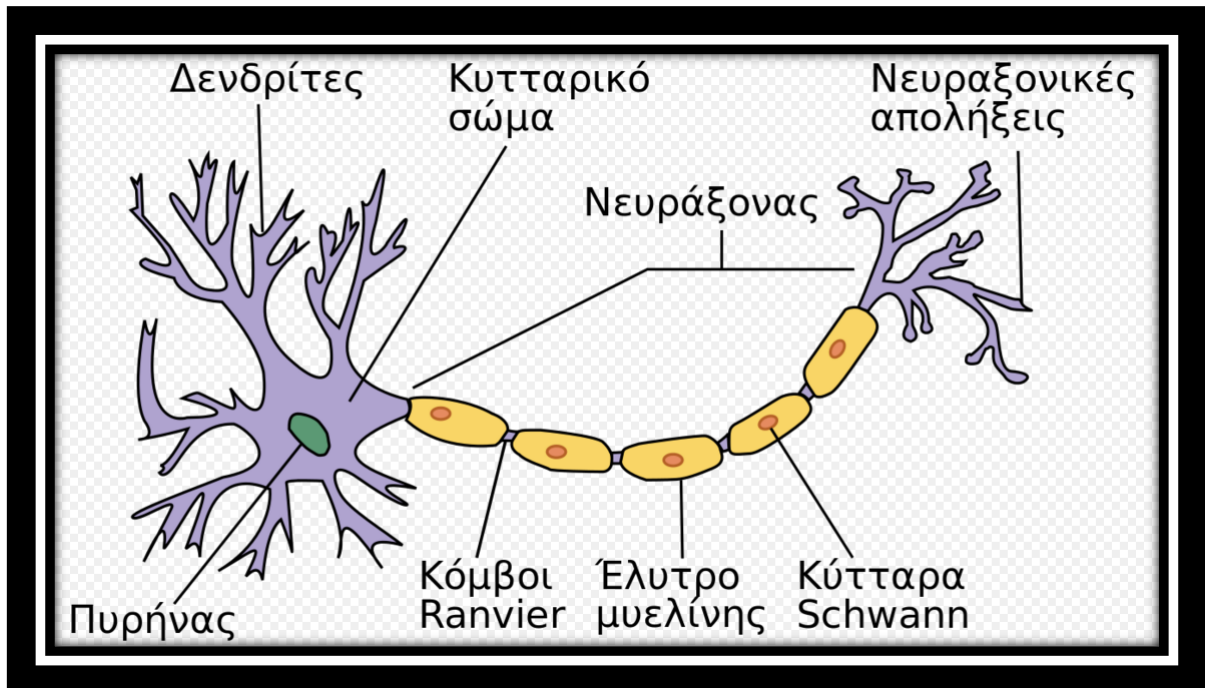
Νευρωνικό δίκτυο (Neural network) ονομάζεται ένα σύστημα διασυνδεδεμένων νευρώνων. Στην περίπτωση βιολογικών νευρώνων, πρόκειται για ένα τμήμα νευρικού ιστού. Στην περίπτωση τεχνητών νευρώνων, πρόκειται για ένα αφηρημένο αλγοριθμικό κατασκεύασμα αποτελούμενο από κόμβους το οποίο εμπίπτει στον τομέα της υπολογιστικής νοημοσύνης, όταν στόχος του νευρωνικού δικτύου είναι η επίλυση κάποιου υπολογιστικού προβλήματος.

Η κύρια ιδέα που κρύβεται πίσω από την ανάπτυξη των τεχνητών νευρωνικών δικτύων (ΤΝΔ) ή artificial neural networks (ANN), είναι η πιστή αντιγραφή της δομής με την οποία είναι φτιαγμένος ο ανθρώπινος εγκέφαλος. Η υλοποίηση ενός «εγκεφάλου» από το μηδέν, όπως ο ανθρώπινος, είναι ένα μεγάλο στοίχημα της ανθρωπότητας. Κάτι τέτοιο όμως είναι πολύ δύσκολο λόγω της πολυπλοκότητάς του. Εκτιμάται ότι ο ανθρώπινος εγκέφαλος αποτελείται από περισσότερα από 100 τρισεκατομμύρια (περίπου) νευρικά κύτταρα, νευρογλοιακά κύτταρα και νευρώνες ως επί το πλείστον. Η προσέγγιση λοιπόν αυτή, της αντιγραφής, θα πάρει αρκετό καιρό με εκτιμήσεις να δείχνουν πως χρειάζονται πάνω από 100 χρόνια για να αποκρυπτογραφηθεί η λειτουργία του ανθρώπινου εγκεφάλου. Είναι εφικτό ωστόσο η δημιουργία εναλλακτικών για την αντιμετώπιση του προβλήματος αυτού. Έτσι τα ΤΝΔ, αφού πήραν τα βασικά χαρακτηριστικά του βιολογικού διδύμου τους, σταδιακά διαφοροποιήθηκαν από αυτά.

2.2 Από τους βιολογικούς στους τεχνητούς νευρώνες

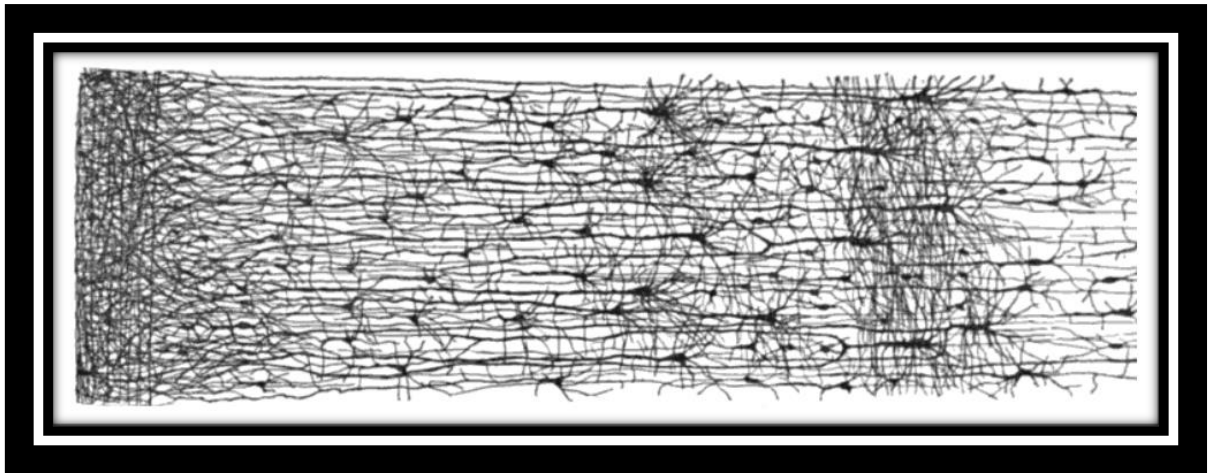
Ο βιολογικός νευρώνας είναι το κύτταρο που αποτελεί δομικό μέρος και λειτουργική μονάδα του νευρικού συστήματος. Κάθε νευρώνας, όπως φαίνεται στο παρακάτω σχήμα, αποτελείται από το κυτταρικό σώμα, τον άξονα, και τους δενδρίτες. Ο κυρίως κορμός του νευρώνα είναι το σώμα μέσα στο οποίο βρίσκεται όλο το γενετικό υλικό του οργανισμού. Ο άξονας είναι μια μεγάλη επέκταση, που το μέγεθος του μπορεί να είναι από μερικές έως και δεκάδες φορές μεγαλύτερο από το σώμα και εφάπτεται με άλλους νευρώνες. Οι άξονες σε μερικούς νευρώνες είναι καλυμμένοι με μια ουσία, που λέγεται μυελίνη, ενώ άλλοι είναι τελείως ακάλυπτοι. Κάθε νευρώνας έχει ένα μόνο άξονα, ο οποίος μεταδίδει σήματα σε άλλους νευρώνες. Υπάρχουν ακόμα οι λεπτές επεκτάσεις που μοιάζουν με διακλαδώσεις

δέντρου και ονομάζονται δενδρίτες. Οι δενδρίτες συλλέγουν τα σήματα που στέλνονται στο κύτταρο. Οι νευρώνες επικοινωνούν μεταξύ τους και με άλλους νευρώνες μέσω συνάψεων, όπου η άκρη του νευράξονα καταλήγει στους δενδρίτες, στο σώμα ή στον νευράξονα άλλων νευρώνων. Στα σημεία που εφάπτονται οι δενδρίτες δημιουργείται η σύναψη. Ο άξονας έχει συνήθως πάρα πολλές διακλαδώσεις και μπορεί να στέλνει σήματα σε διαφορετικά σημεία, πράγμα που τους επιτρέπει να επικοινωνούν ταυτόχρονα με δεκάδες χιλιάδες νευρικά κύτταρα. Το Σχήμα 2.1 συνοψίζει τα παραπάνω.



Σχήμα 2.1 : Βιολογικός νευρώνας / Από User:Dhp1080, translated and modified by Badseed. - Originally Neuron.jpg taken from the US Federal (public domain) (Nerve Tissue, retrieved March 2007), redrawn by User:Dhp1080 in Illustrator. Source: "Anatomy and Physiology" by the US National Cancer Institute's Surveillance, Epidemiology and End Results (SEER) Program., CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=24139135>.

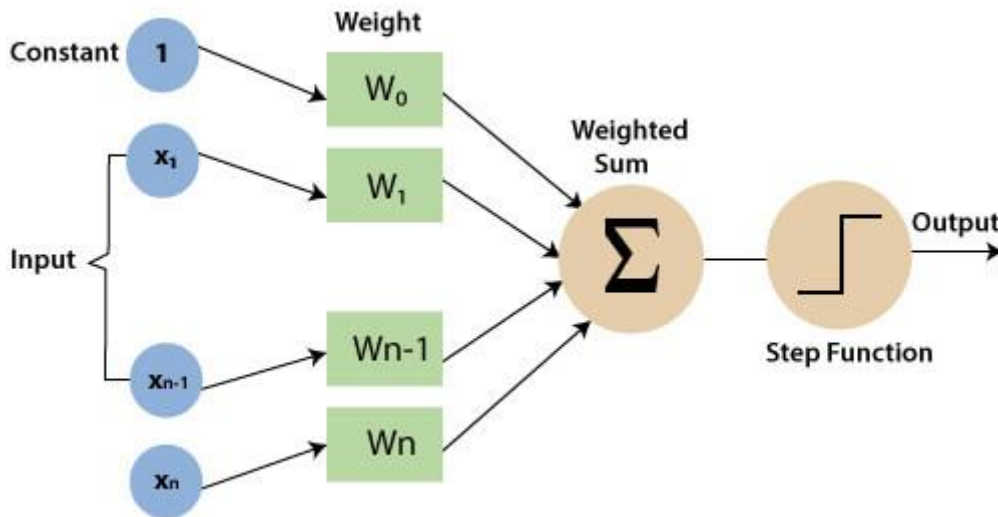
Οι βιολογικοί νευρώνες μεμονωμένα φαίνεται να συμπεριφέρονται με σχετικά απλό τρόπο, όμως είναι οργανωμένοι σε τεράστια δίκτυα αποτελούμενα από δισεκατομμύρια νευρώνες, με τον καθένα από αυτούς να συνδέεται με χιλιάδες άλλους. Είναι προφανές λοιπόν το πόσο περίπλοκοι μπορούν να γίνουν οι υπολογισμοί από ένα τέτοιο τεράστιο δίκτυο. Τα βιολογικά νευρωνικά δίκτυα είναι ένα αντικείμενο έρευνας αλλά κάποια κομμάτια έχουν χαρτογραφηθεί. Όπως μπορούμε να διακρίνουμε και στο Σχήμα 2.2, οι νευρώνες είναι οργανωμένοι σε διαδοχικά επίπεδα (layers).



Σχήμα 2.2: Πολλαπλά επίπεδα σε ένα βιολογικό νευρωνικό δίκτυο.

Το τεχνητό νευρωνικό δίκτυο όπως αναφέρθηκε φτιάχτηκε για να μιμηθεί το βιολογικό. Έτσι λοιπόν περιλαμβάνει πολλούς κόμβους (νευρώνες) διασυνδεδεμένους μεταξύ τους, που περνάνε πληροφορίες μέσα από τις συνδέσεις τους και ενεργοποιούνται όταν ξεπεράσουν ένα κατώφλι, βάρη και μια εξίσωση ενεργοποίησης (activation function). Οι συνδέσεις μεταξύ των κόμβων αντιπροσωπεύουν τον άξονα και τους δενδρίτες, τα βάρη αντιπροσωπεύουν τις συνάψεις ενώ η συνάρτηση ενεργοποίησης προσεγγίζει την δραστηριότητα στο σώμα.

Εάν απομονωθεί ένας από αυτούς τους κόμβους όπως απεικονίζεται στο Σχήμα 2.3 μπορούμε να δούμε την λειτουργία αυτού του απλού νευρωνικού δικτύου. Το μοντέλο αυτό ονομάζεται *Perceptron* και εφευρέθηκε από τον Frank Rosenblatt το 1957. Οι είσοδοι (inputs) είναι νούμερα όπως και η έξοδος και κάθε είσοδος συνδέεται με ένα βάρος (weight). Υπολογίζεται το σταθμισμένο άθροισμα ή *weighted sum* ($z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n = \mathbf{x}^T \mathbf{w}$), και έπειτα εφαρμόζεται σε αυτό μια συνάρτηση ενεργοποίησης (όπου στο σχήμα φαίνεται η βηματική συνάρτηση – step function). Τέλος η έξοδος από το νευρωνικό είναι το αποτέλεσμα $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(z)$, όπου $z = \mathbf{x}^T \mathbf{w}$. Όπως βλέπουμε στην Σχήμα 2.3 υπάρχει ένας νευρώνας με αρχική τιμή εισόδου $x_0 = 1$ και βάρος w_0 . Αυτό το βάρος καλείται *πόλωση ή κατώφλι* (bias, threshold) και μας δείχνει πόσο μεγάλο πρέπει να είναι το σταθμισμένο άθροισμα ώστε να αρχίζει ο νευρώνας να ενεργοποιείται. Εάν είναι μικρότερο από τη συγκεκριμένη τιμή τότε ο νευρώνας είναι ανενεργός. Αυτό επιτυγχάνεται με πρόσθεση του κατάλληλου αριθμού στο σταθμισμένο άθροισμα πριν περάσει από αυτήν η συνάρτηση ενεργοποίησης. Με αυτόν τον τρόπο προσομοιώνεται η λειτουργία των βιολογικών νευρικών κυττάρων, που κάποια μένουν ανενεργά ενώ άλλα ενεργοποιούνται.



Σχήμα 2.3: Μορφή τεχνητού νευρωνικού δικτύου Perceptron.

Το παραπάνω αποτελεί έναν μόνο κόμβο. Όπως αναφέρθηκε όμως κάθε κόμβος συνδέεται με άλλους και εκείνος με την σειρά του με άλλους φτιάχνοντας έτσι ένα δίκτυο, περιπλέκοντας ως ένα βαθμό την υπολογιστική διαδικασία. Χάρη στην γραμμική άλγεβρα είναι εφικτό να υπολογιστούν αποτελεσματικά οι εξοδοι ενός τέτοιου νευρωνικού δικτύου. Η εξίσωση που χρησιμοποιείται είναι η εξής:

$$h_{W,b}(X) = \varphi(XW + b) \quad (2.1)$$

, όπου:

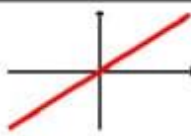

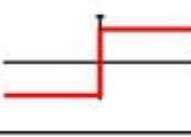




- Το X αντιπροσωπεύει τον πίνακα των εισόδων.
- Το W έχει όλα τα συνδεδεμένα βάρη εκτός από αυτά της πόλωσης.
- Το b που αποτελεί τον πίνακα με τους αριθμούς της πόλωσης.
- Το φ είναι η συνάρτηση ενεργοποίησης που θα αναλυθεί παρακάτω.

2.3 Συνάρτηση ενεργοποίησης (activation function)

Το αποτέλεσμα που προκύπτει από το σταθμισμένο άθροισμα μετά και την προσθήκη της πόλωσης ($z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$) μπορεί να είναι οποιοσδήποτε αριθμός στο διάστημα $(-\infty, +\infty)$. Ο νευρώνας δεν γνωρίζει τα όρια της μεταβλητής, οπότε θα πρέπει να αποφασιστεί πότε ο νευρώνας θα είναι ενεργός και πότε όχι, για να ενσωματωθεί η λειτουργία του ανθρώπινου εγκεφάλου. Αυτό το σκοπό έρχεται να καλύψει η *συνάρτηση ενεργοποίησης*. Ουσιαστικά συμπιέζει το σταθμισμένο άθροισμα σε ένα διάστημα ανάλογο με την συνάρτηση που θα επιλέξουμε, για παράδειγμα στο $[0, 1]$ για την βηματική συνάρτηση (step function) και δηλώνει ότι για την τιμή 0 ο νευρώνας είναι ανενεργός ενώ για την τιμή 1 ενεργοποιείται. Άλλες συναρτήσεις που δεν έχουν μόνο δύο εξόδους, όπως η βηματική, αλλά αριθμούς σε ένα διάστημα δηλώνουν το πόσο ενεργός είναι ένας νευρώνας ανάλογο με την τιμή που έχει. Για παράδειγμα ένας νευρώνας με τιμή 0.2 είναι προφανώς λιγότερο ενεργός σε σχέση με εκείνον με τιμή 0.8.

Ένας ακόμη πολύ σημαντικός λόγος για τον οποίο χρησιμοποιούνται οι συναρτήσεις ενεργοποίησης είναι η μη γραμμικότητα που θέλουμε να προσδώσουμε στο πρόβλημα. Εάν πραγματοποιηθεί μια ζεύξη γραμμικών μετασχηματισμών, όπως γίνεται στα διάφορα επίπεδα των νευρώνων, το μόνο που λαμβάνεται ως αποτέλεσμα είναι ένας γραμμικός μετασχηματισμός. Άρα εάν δεν υπάρξει κάτι για να προσδώσει μη γραμμικότητα ανάμεσα στα επίπεδα των νευρώνων, τότε ακόμα και οι πολλές στρώσεις νευρώνων θα είναι ισοδύναμες με μία. Κάτι τέτοιο καθιστά αδύνατο την επίλυση περίπλοκων προβλημάτων. Συνεπώς η χρήση της συνάρτησης ενεργοποίησης καθίσταται απαραίτητη. Στον Πίνακα 2.1 περιγράφονται συνοπτικά οι πιο διαδεδομένες συναρτήσεις ενεργοποίησης και στο Σχήμα 2.4 φαίνονται τα γραφήματα των δημοφιλέστερων συναρτήσεων (step, logistic, tanh, ReLU), όπως και τα γραφήματα των παραγώγων τους.

Πίνακας 2.1: Συναρτήσεις ενεργοποίησης (Πηγή: Sebastian Raschka, 2016).

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -1/2 \\ z + 1/2 & -1/2 \leq z \leq 1/2 \\ 1 & z \geq 1/2 \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

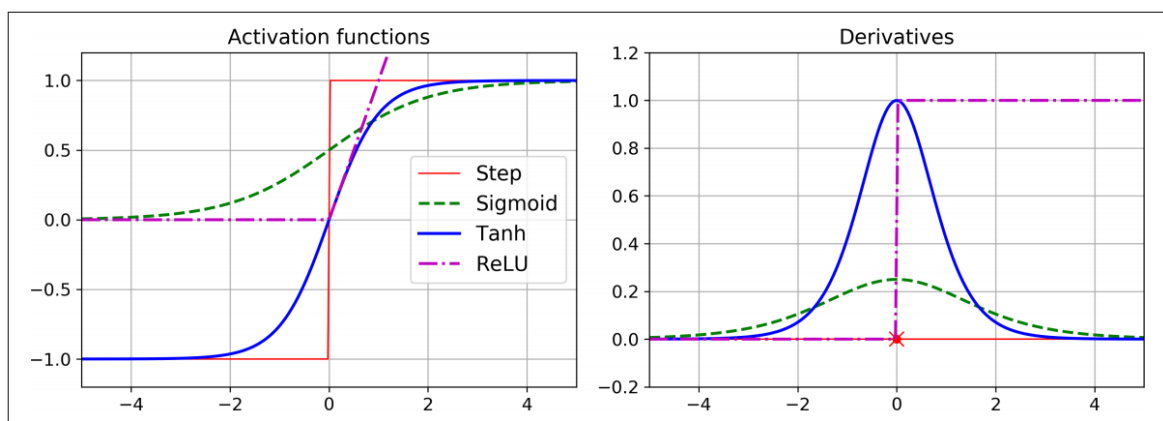
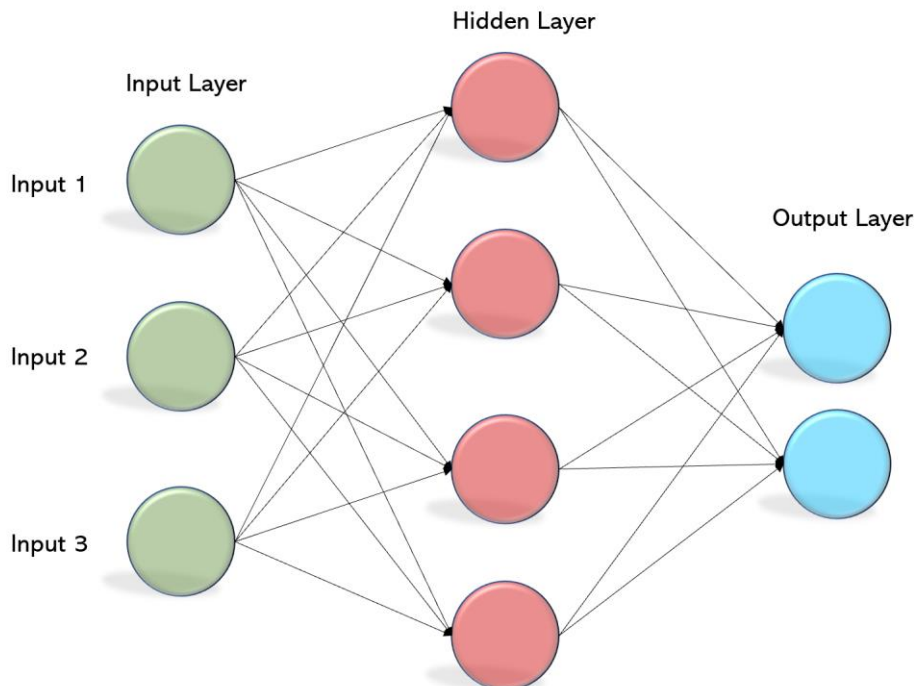


Figure 10-8. Activation functions and their derivatives

Σχήμα 2.4: Συναρτήσεις ενεργοποίησης και οι παράγωγοι τους (Πηγή: Aurélien Géron, 2019).

2.4 Πολυεπίπεδα τεχνητά νευρωνικά δίκτυα (Multilayer Perceptron)

Τα πολυεπίπεδα νευρωνικά δίκτυα, γνωστά με τον όρο MLP (Multilayer Perceptron), είναι το πιο διαδεδομένο είδος νευρωνικού δικτύου και αποτελεί συνέχεια του απλού νευρωνικού που αναλύθηκε προηγουμένως. Αυτό το είδος νευρωνικού περιέχει, εκτός από τα επίπεδα εισόδου (*input layer*) και εξόδου (*output layer*), ένα ή περισσότερα επίπεδα νευρώνων ακόμα, τα λεγόμενα κρυμμένα επίπεδα (*hidden layers*). Κάθε επίπεδο περιέχει ένα νευρώνα πόλωσης (*bias neuron*), εκτός από το τελευταίο, και είναι πλήρως συνδεδεμένο με το επόμενο, μέσω των βαρών (*weights*). Όταν ένα νευρωνικό δίκτυο περιέχει παραπάνω από δύο κρυμμένα επίπεδα τότε ονομάζεται βαθύ νευρωνικό δίκτυο (*deep neural network*). Όπως απεικονίζεται στο παρακάτω σχήμα, οι συνδέσεις μεταξύ των κόμβων δεν σχηματίζουν βρόχους και έτσι η πληροφορία κατευθύνεται προς μία κατεύθυνση από την είσοδο προς την έξοδο. Η μορφή αυτή των δικτύων συχνά αναφέρεται ως νευρωνικά δίκτυα τα οποία τροφοδοτούνται προς τα εμπρός (*feedforward neural networks*) και αποτελούν τη μόνη μορφή με την οποία θα ασχοληθεί η παρούσα εργασία.



Σχήμα 2.5: Πολυεπίπεδο ΤΝΔ (Multi-layer Perceptron).

2.5 Εκπαίδευση τεχνητών νευρωνικών δικτύων (Training)

Το κύριο χαρακτηριστικό των νευρωνικών δικτύων είναι η ικανότητα της μάθησης. Με τον όρο *μάθηση (learning)* στα ΤΝΔ, αναφερόμαστε στο να κατορθώσει ο υπολογιστής να βρει όλους αυτούς τους αριθμούς, τα βάρη και τις πολώσεις (weights and biases), ώστε τελικά να λύσουν το πρόβλημα που του δόθηκε. Κάτι τέτοιο επιτυγχάνεται με την εκπαίδευση του νευρωνικού, μια επαναληπτική διαδικασία σταδιακής προσαρμογής των παραμέτρων του δικτύου. Στη φάση της εκπαίδευσης στο νευρωνικό δίκτυο δίνεται ένα σετ δεδομένων, που αφορά την είσοδο και την έξοδο του συγκεκριμένου προβλήματος που θέλουμε να επιλυθεί, και εκείνο ψάχνει να βρει τις κατάλληλες παραμέτρους (βάρη και πολώσεις) ώστε με τις εισόδους που του δόθηκαν να επιστρέφει τις εξόδους για τις οποίες τροφοδοτήθηκε. Αφού εκπαιδευτεί το ΤΝΔ από εκεί και πέρα είναι σε λειτουργική κατάσταση για μελλοντικές εφαρμογές, με άλλα λόγια για καινούργια δεδομένα (εισόδους), με γνωστές πλέον παραμέτρους, θα παραγάγει τις εξόδους του προβλήματος.

Πως όμως το ΤΝΔ γνωρίζει ποιες παράμετροι είναι οι κατάλληλες; Θα πρέπει να καθοριστεί ένα μέτρο απόδοσης. Ένας τρόπος να γίνει αυτό είναι μέσω μιας *συνάρτησης κόστους* όπου η ελαχιστοποίηση της είναι ο επιθυμητός στόχος. Επομένως με τη μάθηση νοείται η εύρεση εκείνων των βαρών και των πολώσεων που ελαχιστοποιούν μια συγκεκριμένη συνάρτηση κόστους. Η συνηθέστερη συνάρτηση κόστους είναι η αυτή του μέσου τετραγωνικού σφάλματος (Mean Square Error – MSE). Η MSE ουσιαστικά παίρνει τα τετράγωνα των διαφορών ανάμεσα σε κάθε έξοδο του νευρωνικού και κάθε πραγματική τιμή (η έξοδος που θέλουμε από τα δεδομένα μας) και τελικά παίρνει τον μέσο όρο αυτών των όρων.

$$C(y, o) = \frac{1}{N} \sum_{i=1}^N (y_i - o_i)^2 \quad (2.2)$$

,όπου y_i είναι οι προβλέψεις από το τεχνητό νευρωνικό δίκτυο
και o_i είναι οι πραγματικές τιμές που θα έπρεπε να έχουμε.

2.6 Αλγόριθμοι Βελτιστοποίησης

2.6.1 Κατάβαση κλίσης (Gradient Descent)

Αρχικά ο προσδιορισμός και η κατανόηση του αλγορίθμου *κατάβασης κλίσης* αποτελεί κρίσιμο σημείο για τους αλγορίθμους βελτιστοποίησης των ΤΝΔ, καθώς οι περισσότεροι

μετέπειτα αλγόριθμοι βασίζονται σε αυτόν. Είναι ένας πολύ γενικός αλγόριθμος βελτιστοποίησης ικανός να βρει την βέλτιστη λύση σε ένα ευρύ φάσμα προβλημάτων. Η γενική ιδέα της κατάβασης κλίσης είναι να τροποποιήσει τις παραμέτρους της εξίσωσης προκειμένου να ελαχιστοποιήσει μια συνάρτηση κόστους. Ας υποθέσουμε ότι βρισκόμαστε στην κορυφή κάποιου βουνού (υψηλό κόστος) και θέλουμε να κατέβουμε στο κάτω μέρος της κοιλάδας (χαμηλό κόστος). Μια καλή ιδέα για να γίνει αυτό είναι να πάμε κατηφορικά προς την κατεύθυνση της απότομης πλαγιάς, με κάποιο μέγεθος βημάτων είτε μικρό είτε μεγάλο. Αυτό ακριβώς είναι που κάνει ο αλγόριθμος κατάβασης κλίσης, υπολογίζει την αρνητική μερική παράγωγο της συνάρτησης κόστους ως προς το διάνυσμα των παραμέτρων της θ και πηγαίνει προς την κατεύθυνση της φθίνουσας κλίσης. Έπειτα υπολογίζεται ξανά η αρνητική μερική παράγωγος στο σημείο με τις καινούργιες συντεταγμένες αυτή τη φορά. Η διαδικασία αυτή συνεχίζεται μέχρι να επιτευχθεί ένα ολικό ή τοπικό ελάχιστο της συνάρτησης, δηλαδή να μην είναι εφικτή η περαιτέρω κατάβαση προς τα κάτω.

Το μέγεθος αυτών των βημάτων ονομάζεται *ρυθμός μάθησης (learning rate)*. Εάν ο ρυθμός αυτός είναι μικρός θα πρέπει να κάνουμε περισσότερες επαναλήψεις για να συγκλίνει, κάτι που θα χρειαστεί αρκετό χρόνο. Από την άλλη εάν ο ρυθμός μάθησης είναι μεγάλος υπάρχει κίνδυνος να υπερπηδήσει το χαμηλότερο σημείο και να πάμε από την άλλη πλευρά της κοιλάδας αφού η κλίση αλλάζει συνεχώς. Αυτό μπορεί να οδηγήσει τον αλγόριθμο να αποκλίνει με αποτέλεσμα να αποτύχει να βρει μια καλή λύση. . Κάτι ανάλογο παρουσιάζεται στο Σχήμα 2.7, όπου συγκρίνονται οι δύο ακραίες τιμές του ρυθμού μάθησης.

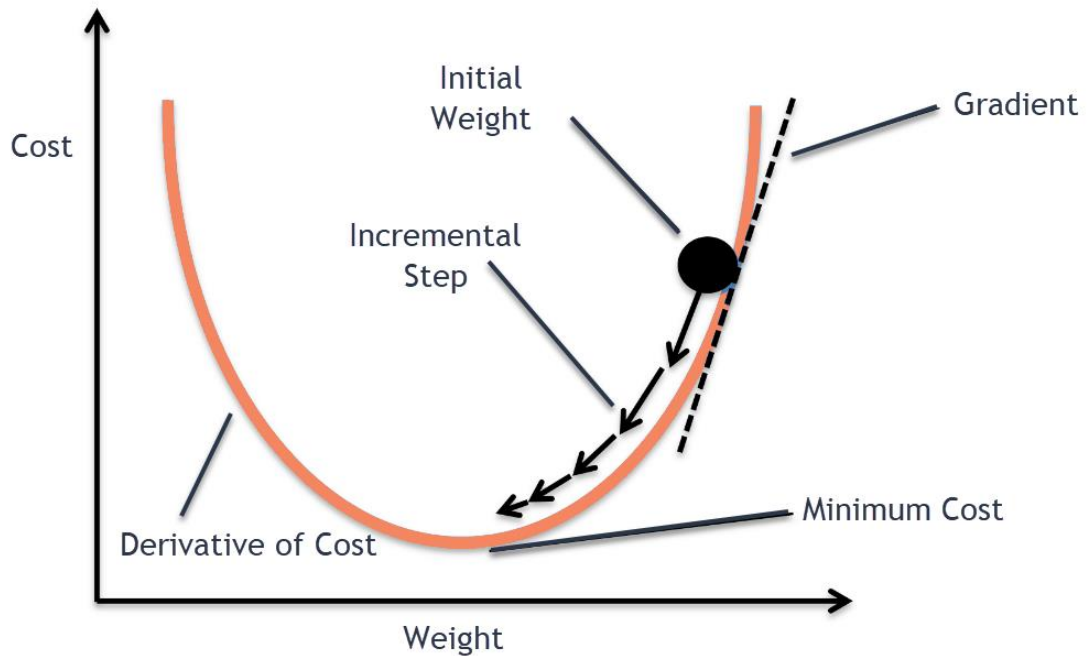
Ο αλγόριθμος περιγράφεται συνοπτικά ως εξής:

$$\mathbf{g} = \nabla_{\theta} \sum_i C(f(\mathbf{x}^{(i)}; \theta), y^{(i)}) \quad (2.3)$$

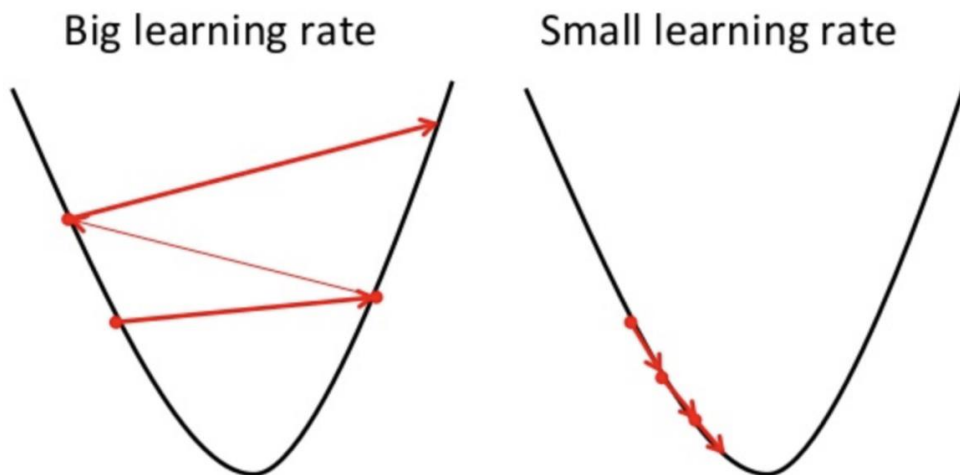
$$\theta = \theta - e_k \times \mathbf{g} \quad (2.4)$$

Αναζητείται η παράμετρος θ για την οποία ελαχιστοποιείται η συνάρτηση κόστους C , με τον ρυθμό μάθησης e_k να παίζει σημαντικό ρόλο.

Η ολοκλήρωση ενός πλήρη κύκλου με το σύνολο των δεδομένων εκπαίδευσης του τεχνητού νευρωνικού δικτύου καλείται *εποχή (epoch)*. Συνήθως για την εκπαίδευση των ΤΝΔ χρειάζονται παραπάνω από μία εποχές (epochs). Ο αριθμός των εποχών ποικίλει ανάλογα με το πρόβλημα που αντιμετωπίζουμε και μπορεί να φτάνει τις μερικές εκατοντάδες ή και χιλιάδες για πολύπλοκα προβλήματα.



Σχήμα 2.6: Σχηματική απεικόνιση αλγορίθμου κατάβασης κλίσης (Gradient descent).



Σχήμα 2.7: Παράδειγμα επιλογής μεγάλου (αριστερά) και μικρού (δεξιά) ρυθμού μάθησης.

2.6.2 Οπισθοδιάδοση (*backpropagation*)

Ο αλγόριθμος οπισθοδιάδοσης (*backpropagation*) είναι ένας ευρέως διαδεδομένος αλγόριθμος και καθίσταται πολύ σημαντικός για την εκπαίδευση νευρωνικών δικτύων, καθώς υπολογίζει τις μερικές παραγώγους με έναν αποτελεσματικό τρόπο. Εφόσον οι παράγωγοι υπολογιστούν γίνεται απλή και εφικτή η εφαρμογή αλγορίθμων βελτιστοποίησης όπως αυτός της κατάβασης κλίσης (gradient descent), κάνοντας χρήση αυτών των παραγώγων.

Για ένα δεδομένο τεχνητό νευρωνικό δίκτυο και για μια συνάρτηση κόστους, η μέθοδος της οπισθοδιάδοσης (backpropagation) υπολογίζει με μια αποτελεσματική τεχνική τις μερικές παραγώγους ως προς τα βάρη, από την συνάρτηση κόστους. Με άλλα λόγια μπορεί να βρει πως πρέπει να τροποποιηθεί το κάθε βάρος και η κάθε πόλωση ώστε να μειωθεί το σφάλμα. Η λέξη «όπισθεν» που εμπεριέχεται στην ονομασία της μεθόδου αναφέρεται στο γεγονός ότι οι παράγωγοι υπολογίζονται προς τα πίσω μέσα στο δίκτυο, με τις παραγώγους του τελευταίου επιπέδου νευρώνων να υπολογίζονται πρώτες και του πρώτου επιπέδου να υπολογίζονται τελευταίες. Οι υπολογισμένες μερικές παράγωγοι από το ένα επίπεδο (layer) ξαναχρησιμοποιούνται για τον υπολογισμό των παραγώγων του προηγούμενου επιπέδου, επηρεάζοντας έτσι η μία την άλλη. Αυτή η ροή προς τα πίσω της πληροφορίας για την συνάρτηση κόστους επιτρέπει ένα πιο αποτελεσματικό υπολογισμό των παραγώγων σε κάθε επίπεδο, έναντι της απλής προσέγγισης για τον υπολογισμό των παραγώγων σε κάθε επίπεδο ξεχωριστά.

Συνοψίζοντας λοιπόν, για κάθε εκπαίδευση που συμβαίνει ο αλγόριθμος οπισθοδιάδοσης πρώτα κάνει μια πρόβλεψη (προς τα εμπρός πέρασμα), μετράει το σφάλμα της συνάρτησης κόστους, έπειτα πηγαίνει σε κάθε επίπεδο αντίστροφα για να υπολογίσει την συνεισφορά που έχει κάθε σύνδεση νευρώνων στο σφάλμα και τελικά τροποποιεί ελαφρώς τα βάρη και τις πολώσεις για να ελαχιστοποιήσει το τελικό σφάλμα.

Στην παραπάνω διαδικασία κεντρικός είναι ο ρόλος του κανόνα της αλυσίδας (*Chain rule*) που έγινε γνωστός όταν οι Newton και Leibniz ανακάλυψαν τον απειροστικό λογισμό στα τέλη του 17^{ου} αιώνα. Αν μια μεταβλητή z εξαρτάται από μια y και εκείνη με την σειρά της εξαρτάται από μια άλλη μεταβλητή x τότε ο κανόνας της αλυσίδας διατυπώνεται ως εξής:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \quad (2.5)$$

Στο πλαίσιο των νευρωνικών δικτύων είναι σημαντικό να υπολογιστεί το πόσο επηρεάζει η κάθε παράμετρος την τελική συνάρτηση κόστους για να μπορέσει να την ελαχιστοποιήσει. Ξεκινώντας από το τέλος του νευρωνικού δικτύου, η συνάρτηση κόστους για ένα μόνο νευρώνα εξαρτάται από την τιμή που έχει σαν έξοδο και την πραγματική τιμή, η οποία είναι σταθερή. Η τιμή εξόδου y_i όμως εξαρτάται από τα βάρη, τις τιμές των προηγούμενων νευρώνων και των πολώσεων, όπου συνολικά μπορούμε να ορίζουμε αυτό το σταθμισμένο άθροισμα ως z . Στη συνέχεια εφαρμόζεται στο z μια συνάρτηση ενεργοποίησης για να προσδιοριστεί η τελική τιμή του νευρώνα. Για να υπολογιστεί λοιπόν η επιρροή που θα έχει

μια μικρή μεταβολή των βαρών στην συνάρτηση κόστους, θα πρέπει να βρεθεί η μερική παράγωγος της συνάρτησης κόστους ως προς το αντίστοιχο βάρος που μελετάται. Σε αυτό το σημείο είναι που εφαρμόζεται ο κανόνας της αλυσίδας για να υπολογίσει αυτή την παράγωγο. Συγκεκριμένα για τα νευρωνικά δίκτυα ο κανόνας της αλυσίδας διατυπώνεται:

$$\frac{\partial C}{\partial w_{ji}} = \frac{\partial z_i}{\partial w_{ji}} \frac{\partial y_i}{\partial z_i} \frac{\partial C}{\partial y_i} \quad (2.6)$$

με τις μερικές παραγώγους τους δεξιού μέλους να είναι υπολογίσιμες. Με τον ίδιο τρόπο υπολογίζεται και η συνεισφορά των υπόλοιπων παραμέτρων στην συνάρτηση κόστους. Η ίδια λογική μπορεί να εφαρμοστεί προς τα πίσω για τα βάρη και τις πολώσεις των προηγούμενων νευρώνων μέχρι την αρχή του δικτύου και έτσι τελικά υπολογίζεται το ανάδελτα ∇C της συνάρτησης κόστους.

2.6.3 Στοχαστική κατάβαση κλίσης (*Stochastic gradient descent*)

Το πρόβλημα με τον αλγόριθμο κατάβασης κλίσης (gradient descent) έγκειται στο κομμάτι των δεδομένων που χρησιμοποιεί. Πιο συγκεκριμένα κάνει χρήση όλων των δεδομένων εκπαίδευσης που διαθέτει για να υπολογίσει τις παραγώγους σε κάθε βήμα, πράγμα που καθιστά τον αλγόριθμο αρκετά αργό όταν το σύνολο των δεδομένων είναι πολύ μεγάλο. Από την άλλη, ο αλγόριθμος της *στοχαστικής κατάβασης κλίσης (Stochastic Gradient Descent ή SGD)* επιλέγει με τυχαίο τρόπο ένα κομμάτι από αυτά τα δεδομένα εκπαίδευσης και υπολογίζει τις παραγώγους μόνο σε αυτό το κομμάτι. Αυτή η παραλλαγή της κλασικής περίπτωσης κάνει τον αλγόριθμο πολύ πιο γρήγορο από την στιγμή που μειώνονται τα δεδομένα που διαχειρίζεται σε κάθε επανάληψη. Ωστόσο λόγω της στοχαστικότητας που διατρέχει το σύστημα θα υπάρξουν μικρές και μεγάλες μεταβολές στην συνάρτηση κόστους με το τελικό αποτέλεσμα των τιμών των παραμέτρων να είναι καλό αλλά όχι το βέλτιστο. Επιπρόσθετα η στοχαστικότητα βοηθάει τον αλγόριθμο να εξέλθει από τα τοπικά ελάχιστα δίνοντας έτσι περισσότερες πιθανότητες στον αλγόριθμο να καταλήξει σε ολικό ελάχιστο. Η περιγραφή του αλγορίθμου δίνεται από τις παρακάτω σχέσεις:

$$\mathbf{g} = \frac{1}{m} \nabla_{\theta} \sum_i C(f(\mathbf{x}^{(i)}; \theta), y^{(i)}) \quad (2.7)$$

$$\theta = \theta - e_k \times \mathbf{g} \quad (2.8)$$

Το τεχνητό νευρωνικό δίκτυο αναπαριστάται από το $f(\mathbf{x}^{(i)}; \boldsymbol{\theta})$, όπου το $x^{(i)}, y^{(i)}$ είναι το σετ των δεδομένων εκπαίδευσης, ενώ η παράγωγος της συνάρτησης κόστους \mathcal{C} υπολογίζεται ως προς τις παραμέτρους $\boldsymbol{\theta}$. Το m αντιπροσωπεύει το μέγεθος των δεδομένων που χρησιμοποιούνται ενώ το g δηλώνει την παράγωγο (gradient). Ο ρυθμός μάθησης (learning rate) e_k καθορίζει το μέγεθος των βημάτων που υλοποιείται από τον αλγόριθμο.

2.6.4 Αλγόριθμος AdaGrad

Πρόκειται για έναν προσαρμοστικό αλγόριθμο που ρυθμίζει το ρυθμό μάθησης (learning rate), κάτι που δεν συμβαίνει στους άλλους αλγορίθμους. Ουσιαστικά ο αλγόριθμος *AdaGrad* προσθέτει μια εξάρτηση του ποσοστού μάθησης με τις παραγώγους. Αρχικά συλλέγει τα τετράγωνα των παραγώγων, που έχουν υπάρξει μέχρι εκείνη την στιγμή, σε ένα διάνυσμα \mathbf{s} και έπειτα διαιρεί το ρυθμό μάθησης με την τετραγωνική ρίζα αυτού του αθροίσματος \mathbf{s} . Πιο συγκεκριμένα έχουμε:

$$\mathbf{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_t \mathcal{C}(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \quad (2.9)$$

$$\mathbf{s} = \mathbf{s} + \mathbf{g}^T \mathbf{g} \quad (2.10)$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - e_k \times \mathbf{g} / \sqrt{\mathbf{s} + \varepsilon} \quad (2.11)$$

Όπου ε είναι ένας πολύ μικρός αριθμός (συνήθως 10^{-10}) για να αποφευχθεί η διαίρεση με το μηδέν. Το αποτέλεσμα θα είναι οι παράμετροι που λαμβάνουν μεγάλες παραγώγους θα δουν το ρυθμό μάθησης να μικραίνει και το αντίθετο.

2.6.5 Αλγόριθμος Adam

Ο αλγόριθμος *Adam* (*Adaptive moments*) αποτελεί έναν συνδυασμό διαφόρων άλλων αλγορίθμων και λειτουργεί με τις ροπές πρώτης και δεύτερης τάξης. Η κύρια ιδέα πίσω από αυτόν τον αλγόριθμο είναι να μειωθεί η ταχύτητα πραγματοποίησης του αλγορίθμου για μια καλύτερη ανίχνευση του ολικού ελαχίστου. Ο αλγόριθμος *Adam* δημιουργεί έναν εκθετικά εξασθενημένο μέσο όρο των προηγούμενων παραγώγων και των τετραγώνων των παραγώγων. Αναλυτικά ο αλγόριθμος περιγράφεται:

$$\mathbf{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_t \mathcal{C}(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \quad (2.12)$$

$$\mathbf{m} = \beta_1 \mathbf{m} + (1 - \beta_1) \mathbf{g} \quad (2.13)$$

$$\mathbf{s} = \beta_2 \mathbf{s} + (1 - \beta_2) \mathbf{g}^T \mathbf{g} \quad (2.14)$$

$$\hat{\mathbf{m}} = \frac{\mathbf{m}}{1 - \beta_1^t} \quad (2.15)$$

$$\hat{\mathbf{s}} = \frac{\mathbf{s}}{1 - \beta_2^t} \quad (2.16)$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - e_k \times \hat{\mathbf{m}} / \sqrt{\hat{\mathbf{s}} + \varepsilon} \quad (2.17)$$

Όπου το t αντιπροσωπεύει τον αριθμό της επανάληψης (ξεκινώντας από 1). Οι τιμές που προτείνονται για τις παραμέτρους είναι:

$$\beta_1 = 0.9, \beta_2 = 0.999 \text{ και } \varepsilon = 10^{-8}.$$

2.6.6 Αλγόριθμος L-BFGS

Αποτελεί έναν αλγόριθμο βελτιστοποίησης που βασίζεται στη μέθοδο του Νεύτωνα (γνωστή και ως *Newton-Raphson*) για την εύρεση των ριζών μιας πραγματικής εξίσωσης. Είναι μια παραλλαγή του αλγορίθμου BFGS (*Broyden-Fletcher-Goldfarb-Shanno*) διότι χρησιμοποιεί περιορισμένη ποσότητα υπολογιστικής μνήμης (Limited-memory), όπως υποδηλώνει το αρχικό γράμμα του ονόματος. Και οι δύο αυτοί αλγόριθμοι κάνουν χρήση του αντίστροφου Εσσιανού (Hessian) πίνακα εκτίμησης για τον έλεγχο των μεταβλητών στον χώρο. Ο Εσσιανός (Hessian) πίνακας έχει σαν στοιχεία του τις δεύτερες μερικές παραγώγους της υπό ανάλυση συνάρτησης. Σε αντίθεση με τον BFGS που αποθηκεύει έναν $n \times n$ προσεγγιστικό Εσσιανό πίνακα (όπου n είναι ο αριθμός των παραμέτρων του προβλήματος), ο L-BFGS αποθηκεύει μόνο μερικά διανύσματα που αντιπροσωπεύουν την προσέγγιση. Αυτή η διαφορά είναι που ελευθερώνει μνήμη από τον υπολογιστή. Ο αλγόριθμος L-BFGS είναι κατάλληλος για προβλήματα βελτιστοποίησης με μεγάλο αριθμό μεταβλητών.

Περισσότερες πληροφορίες περί τεχνητών νευρωνικών δικτύων και μηχανικής μάθησης ο αναγνώστης παραπέμπεται στο βιβλίο του Aurélien Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2019.

3 ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΠΟΥ ΥΠΑΚΟΥΝ ΝΟΜΟΥΣ ΤΗΣ ΦΥΣΙΚΗΣ (PHYSICS INFORMED NEURAL NETWORKS - PINN)

3.1 Εισαγωγή

Στο παρόν κεφάλαιο θα αναλυθούν τα νευρωνικά δίκτυα τα οποία εκπαιδεύονται για να επιλύσουν προβλήματα τα οποία περιγράφονται από κάποιον νόμο της φυσικής, μέσω μη-γραμμικών μερικών διαφορικών εξισώσεων. Το πρόβλημα που προκύπτει στα τεχνητά νευρωνικά δίκτυα που καλούνται να εφαρμοστούν σε δύσκολα φυσικά συστήματα, είναι ότι συχνά δεν υπάρχουν δεδομένα για να τροφοδοτήσουν το ΤΝΔ ή καλύτερα για να αποκτηθούν χρειάζονται πόρους και μεγάλο κόστος, που μπορεί να είναι απαγορευτικό. Σε συνέχεια αυτού τα κλασικά τεχνητά νευρωνικά δίκτυα υστερούν σε ακρίβεια, γιατί πρέπει να βγάλουν κάποιο αποτέλεσμα με λίγα δεδομένα τα οποία μπορεί να μην είναι αντιπροσωπευτικά, και έτσι αποτυγχάνουν να αποδώσουν ένα αξιόπιστο αποτέλεσμα. Γι' αυτό προκύπτει η ανάγκη να δοθεί επιπλέον πληροφορία ώστε το ΤΝΔ να την αξιοποιήσει, έχοντας με αυτό τον τρόπο ένα καλύτερο αποτέλεσμα. Πιο συγκεκριμένα οι νόμοι της φυσικής είναι αυτοί που θα προσδώσουν την επιπλέον πληροφορία, όπως για παράδειγμα η διατήρηση της μάζας ή κάποιοι εμπειρικοί κανόνες. Αυτό σημαίνει ότι και με λίγα δεδομένα στο κομμάτι της εκπαίδευσης του νευρωνικού θα υπάρξει μια καλή λύση στο πρόβλημα, αφού θα έχει ενισχυθεί το ΤΝΔ με τη σχέση που συνδέει τις εισόδους και τις εξόδους του προβλήματος, η οποία θα έχει εκφραστεί μέσω μιας μη-γραμμικής διαφορικής εξίσωσης.

3.2 Γενική περιγραφή και ορισμός του προβλήματος

Αυτή η διαφορετική προσέγγιση των τεχνητών νευρωνικών δικτύων στοχεύει στην εύρεση αναλυτικής λύσης σε μη-γραμμικά προβλήματα χωρίς να καταφεύγουμε σε καινούργιες υποθέσεις. Μέχρι τώρα η επίλυση τέτοιου είδους εξισώσεων γινόταν μέσω γραμμικοποίησης ή αριθμητικών μεθόδων, όπως τα πεπερασμένα στοιχεία.

Ορίζεται η παραμετροποιημένη και μη γραμμική μερική διαφορική εξίσωση της γενικής μορφής (Karniadakis et al., 2017):

$$u_t + N[u] = 0, \quad x \in \Omega, t \in [0, T], \quad (3.1)$$

όπου το $u(t, x)$ συμβολίζει την λύση της εξίσωσης, $N[\cdot]$ είναι ένας μη γραμμικός διαφορικός τελεστής και Ω ένα υποσύνολο των πραγματικών αριθμών \mathbb{R}^D . Πως όμως θα τροποποιηθεί ο αλγόριθμος του ΤΝΔ για να καταφέρει να εκμεταλλευτεί αυτήν την επιπλέον πληροφορία;

Ορίζεται η συνάρτηση $f(t, x)$ η οποία θα δίνεται από το αριστερό μέλος της εξίσωσης (3.1), δηλαδή:

$$f := u_t + N[u] \quad (3.2)$$

Το τεχνητό νευρωνικό δίκτυο θα είναι σε θέση να προσεγγίσει την λύση της εξίσωσης με μόνο δύο ορίσματα:

- Τη μερική διαφορική εξίσωση που περιγράφει το φαινόμενο και
- Τις αρχικές και συνοριακές συνθήκες που ορίζουν αυτή τη διαφορική.

3.3 Δεδομένα εισόδου και εκπαίδευση

Το τεχνητό νευρωνικό δίκτυο δέχεται ως είσοδο δύο σύνολα σημείων για την εκπαίδευση του, τα N_u και N_f . Συγκεκριμένα τα N_u αποτελούν τα σημεία των αρχικών και συνοριακών συνθηκών, τα οποία είναι γνωστά, όπως και η λύση της $u(t, x)$ σε αυτά τα σημεία. Δίνεται για παράδειγμα μια οριακή συνθήκη, έστω $u(0, x) = 1$. Αυτό σημαίνει ότι για την χρονική στιγμή $t = 0$ και για όλα τα x η λύση της u είναι παντού ίση με τη μονάδα. Από την άλλη τα N_f καθορίζουν τα σημεία τα οποία βρίσκονται μέσα στο χωρίο το οποίο ορίζεται από τα διαστήματα του χρόνου και του χώρου. Αυτά τα σημεία λαμβάνονται με τυχαίο τρόπο και είναι δεδομένο, από τον προηγούμενο ορισμό, πως εκεί η f θα πρέπει να μηδενίζεται. Οι παράμετροι του ΤΝΔ θα βρεθούν από την ελαχιστοποίηση της συνάρτησης κόστους. Τώρα όμως θα πρέπει να υπάρχουν δύο όροι στην συνάρτηση αυτή, ο ένας θα αναφέρεται στα σημεία N_u και ο άλλος στα N_f . Συνεπώς η συνάρτηση κόστους παίρνει τη μορφή:

$$MSE = MSE_u + MSE_f \quad (3.3)$$

όπου

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2 \quad (3.4)$$

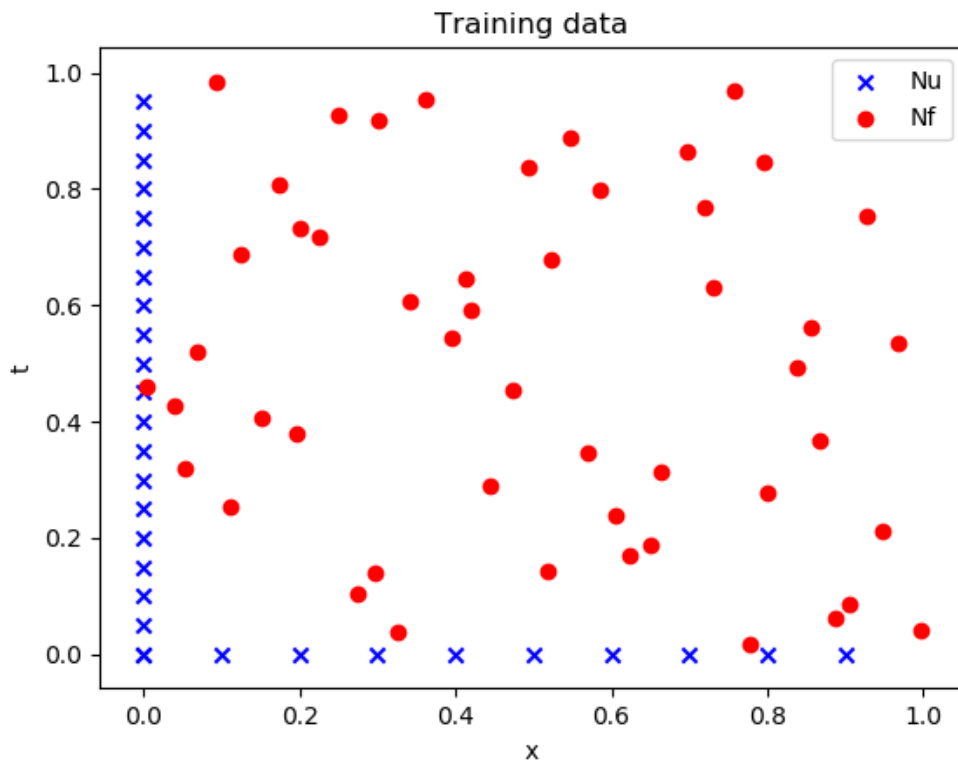
και

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2 \quad (3.5)$$

Ο όρος $u(t_u^i, x_u^i)$ δηλώνει το αποτέλεσμα του προκύπτει από το ΤΝΔ ενώ ο όρος u^i δηλώνει την πραγματική τιμή που θα έπρεπε να έχει στα συγκεκριμένα σημεία N_u , η $u(t, x)$. Αντίστοιχα για τα σημεία N_f το νευρωνικό δίνει τα αποτελέσματα $f(t_f^i, x_f^i)$, τα οποία θα πρέπει να συγκλίνουν στο μηδέν.

Στο Σχήμα 3.1 φαίνονται σχηματικά τα σημεία που επιλέχθηκαν για την εκπαίδευση με σκοπό τη λύση μιας μερικής διαφορικής εξίσωσης. Για την ακρίβεια διακρίνονται μία οριακή και μία συνοριακή συνθήκη. Ο χρόνος t και ο χώρος x παίρνουν τιμές στο διάστημα $[0,1]$. Από την οριακή συνθήκη ($t = 0$) τα σημεία που επιλέχθηκαν είναι 10, ενώ από την συνοριακή συνθήκη ($x = 0$) 20 σημεία, δηλαδή τα συνολικά N_u σημεία είναι 30. Στο εσωτερικό του χωρίου έγινε η επιλογή 50 τυχαίων κατανεμημένα σημείων, δηλαδή $N_f = 50$.

Αξίζει να αναφερθεί σε αυτό το σημείο ότι σε αντίθεση με τις κλασικές αριθμητικές μεθόδους η επίλυση της μερικής διαφορικής εξίσωσης με το ΤΝΔ έχει επιτευχθεί χωρίς καμία διακριτοποίηση στο χώρο και στο χρόνο, ενώ δεν χρειάζεται ειδική μεταχείριση το κάθε ξεχωριστό φαινόμενο που μελετάται όσον αφορά τον ορισμό της εξίσωσης. Το αποτέλεσμα που προκύπτει είναι πλήρως παραγωγίσιμο σε όλο το πεδίο ορισμού της. Η έννοια της παραγώγου που εισάγεται στο ΤΝΔ είναι ιδιαίτερα σημαντική πληροφορία για την ανάπτυξη λύσης στο εκάστοτε πρόβλημα. Αυτό σημαίνει πως η λύση που λαμβάνεται είναι σαφώς καλύτερη από εκείνη που προσφέρουν άλλες μέθοδοι, δεδομένου ότι είναι μια πολύ καλή προσέγγιση της αναλυτικής και ακριβής λύσης. Τα πλεονεκτήματα αυτά, μαζί με την ευκολία στην υλοποίηση, καθιστούν τα PINN ένα ιδιαίτερα ισχυρό εργαλείο στα χέρια των μηχανικών.



Σχήμα 3.1: Σχηματική απεικόνιση σημείων εκπαίδευσης ΤΝΔ που επιλύει διαφορική εξίσωση.

3.4 Ταυτοποίηση παραμέτρων διαφορικής εξίσωσης (parameter identification)

Ακόμη μία δυνατότητα αυτού του είδους των τεχνητών νευρωνικών δικτύων είναι η αξιοποίηση τους με στόχο την εύρεση των παραμέτρων του προβλήματος, το οποίο περιγράφεται από την διαφορική εξίσωση. Πιο συγκεκριμένα η προηγούμενη ανάλυση βασίστηκε στην υπόθεση πως η διαφορική που περιγράφει το πρόβλημα είναι γνωστή στην πλήρη μορφή της, δηλαδή με τις παραμέτρους δεδομένες, και ζητούμενη είναι η λύση σε όλο το πεδίο ορισμού της. Από την άλλη μεριά, αν με κάποιον τρόπο είναι δοσμένη η λύση της διαφορικής εξίσωσης, για παράδειγμα από μετρήσεις, αλλά αγνοούνται οι παράμετροι αυτής τότε καθίσταται δυνατός ο προσδιορισμός τους μέσω των ΤΝΔ. Αυτό το αντίστροφο μοντέλο καλείται και ως ταυτοποίηση παραμέτρων (parameter identification). Το τεχνητό νευρωνικό δίκτυο θα εκπαιδευτεί με στόχο οι τιμές εξόδου να είναι όσο το δυνατόν πιο κοντά στα δεδομένα. Τα δεδομένα αυτά μπορεί να είναι μετρήσεις πεδίου, όπως αναφέρθηκε, αλλά και συνθετικά δεδομένα προερχόμενα είτε από αναλυτικές επιλύσεις, αν είναι δυνατές, είτε από προσομοιώσεις του φυσικού φαινομένου μέσω αριθμητικών μεθόδων. Οι παράμετροι της διαφορικής εξίσωσης λ , αυτή τη φορά αντιμετωπίζονται σαν

μεταβλητές του δικτύου και για το λόγο αυτό συνεχώς αλλάζουν στην φάση της εκπαίδευσης του νευρωνικού δικτύου, σε αντίθεση με την περίπτωση αναζήτησης της λύσης $u(t, x)$ στην οποία θεωρούνται σταθεροί. Η συνάρτηση κόστους παραμένει ίδια, ενώ η διαφορά εμφανίζεται στις παραμέτρους λ οι οποίες θα πρέπει να οριστούν σαν μεταβλητές στον κώδικα και έτσι να αντιμετωπιστούν από το ΤΝΔ.

Υπάρχουν τριών ειδών μέθοδοι για την εκπαίδευση των δικτύων αυτών:

- (α) η παραγωγή μεγάλου αριθμού συνόλων δεδομένων, με οποιοδήποτε τρόπο αυτά δημιουργηθούν και η πραγματοποίηση μίας εποχής (epoch) πάνω σε αυτά τα δεδομένα.
- (β) εκπαίδευση με ένα σύνολο δεδομένων αλλά με πραγματοποίηση πολλών εποχών πάνω σε αυτά.
- (γ) ένας συνδυασμός των παραπάνω.

Στην πράξη η πρώτη μέθοδος είναι συνήθως αδύνατη να πραγματοποιηθεί καθώς είναι ιδιαίτερα δύσκολη η συλλογή των δεδομένων, ειδικά για μετρήσεις στο χώρο όπου οι σταθμοί είναι εγκατεστημένοι σε σταθερά και συγκεκριμένα σημεία. Στην περίπτωση που χρησιμοποιούνται συνθετικά δεδομένα όλες οι μέθοδοι μπορούν να εφαρμοστούν, διαφορετικά η δεύτερη μέθοδος συνιστάται.

4 ΓΕΝΙΚΕΣ ΕΞΙΣΩΣΕΙΣ ΜΗ ΜΟΝΙΜΗΣ ΡΟΗΣ ΜΕ ΕΛΕΥΘΕΡΗ ΕΠΙΦΑΝΕΙΑ

4.1 Μη μόνιμη ροή – Γενικά

Η ροή σε ανοιχτούς αγωγούς με ελεύθερη επιφάνεια χωρίζεται σε δύο κατηγορίες σύμφωνα με το κριτήριο του χρόνου, τη μόνιμη και τη μη μόνιμη ροή. Η μόνιμη ροή είναι εκείνη στην οποία το βάθος ροής, όπως και η μέση ταχύτητα ροής, δε μεταβάλλονται στο χρόνο. Από την άλλη στη μη μόνιμη ροή οι συνθήκες ροής μεταβάλλονται συναρτήσει του χρόνου. Οι ροές που παρατηρούνται στη φύση είναι εν γένει μη μόνιμες. Μερικά χαρακτηριστικά παραδείγματα μη μόνιμων ροών είναι οι πλημμυρικές ροές και τα κύματα.

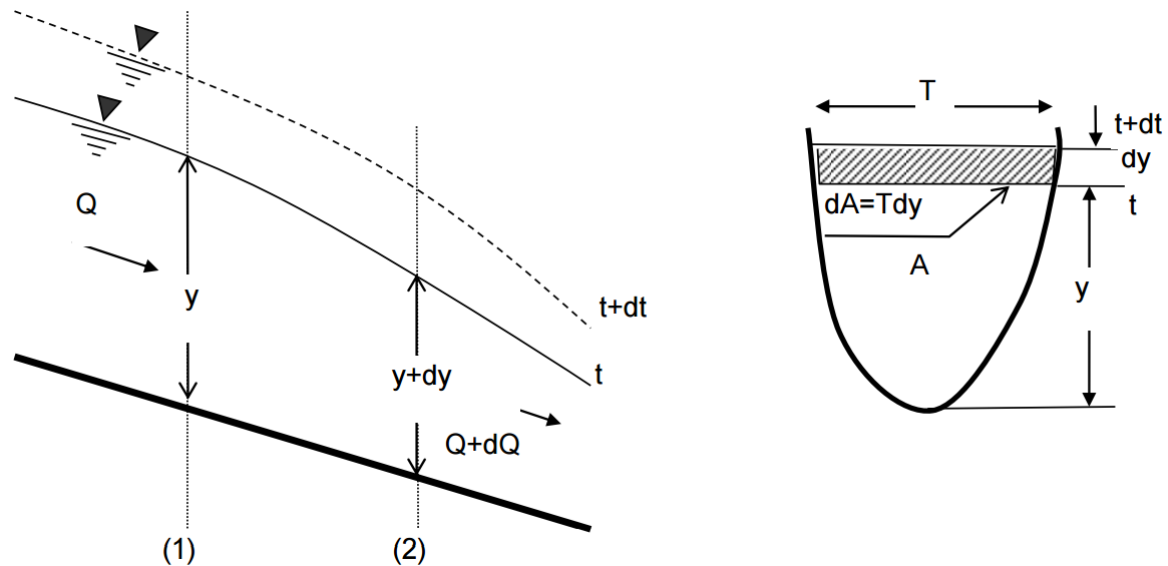
Το πρόβλημα της διόδευσης μιας πλημμύρας, δηλαδή το πρόβλημα της μαθηματικής αναπαράστασης της εξέλιξης ενός πλημμυρικού φαινομένου στο χώρο και στο χρόνο, στην πλειονότητα των περιπτώσεων περιγράφεται ικανοποιητικά από τις διαφορικές εξισώσεις μονοδιάστατης μη μόνιμης ροής βαθμιαίας μεταβολής σε ανοιχτούς αγωγούς (Κουτσογιάννης, 2011). Οι εξισώσεις αυτές καλούνται και εξισώσεις Saint Venant. Οι εξισώσεις Saint-Venant προκύπτουν από τις εξισώσεις Navier-Stokes ολοκληρώνοντας ως προς το βάθος, δηλαδή είναι κατάλληλες για μικρά βάθη ροής σε σχέση με το πλάτος (όπως για παράδειγμα τα ποτάμια). Αυτές οι εξισώσεις βασίζονται στην αρχή διατήρησης της συνέχειας και της ορμής. Η πολυπλοκότητα όμως που διέπει αυτές τις εξισώσεις οδήγησε στην επίλυση τους όχι με αναλυτικές αλλά με αριθμητικές μεθόδους.

Οι μέθοδοι που έχουν αναπτυχθεί για την επίλυση αυτών των εξισώσεων γενικά περιγράφονται με τον όρο υδραυλικές μέθοδοι και βασίζονται είτε σε αριθμητικά σχήματα πεπερασμένων διαφορών είτε σε απλοποιήσεις των εξισώσεων, ώστε να επιδέχονται αναλυτική επίλυση, είτε σε συνδυασμό αυτών των δύο μεθόδων. Μια άλλη μεγάλη κατηγορία μεθόδων που επιλύει το πρόβλημα της διόδευσης πλημμύρας, είναι οι υδρολογικές μέθοδοι. Οι υδρολογικές μέθοδοι χρησιμοποιούν την εξίσωση συνέχειας και αντί της εξίσωσης ορμής χρησιμοποιούν προσεγγιστικές σχέσεις μεταξύ των παροχών που εισρέουν και εκρέουν σε ένα τμήμα του ποταμού και του αποθηκευτικού όγκου νερού. Τέτοιες μέθοδοι είναι η μέθοδος Muskingum, η μέθοδος Muskingum-Cunge (παραλλαγή

της προηγούμενης), η μέθοδος Υστέρησης-Διόδευσης (Lag and Route) και άλλες. Στη συνέχεια ακολουθεί η περιγραφή των εξισώσεων μη μόνιμης ροής.

4.2 Εξίσωση συνέχειας

Η πρώτη εξίσωση που χρησιμοποιείται για την ανάλυση της μη μόνιμης ροής σε ανοιχτούς αγωγούς για μονοδιάστατη ανάλυση είναι η εξίσωση συνέχειας.



Σχήμα 4.1: Επεξηγηματικό σχήμα για τα χαρακτηριστικά μεγέθη της ροής με ελεύθερη επιφάνεια για την εξίσωση συνέχειας.

Η εξίσωση συνέχειας σε ένα κομμάτι μεταξύ δύο διατομών του αγωγού που απέχουν μεταξύ τους Δx , όπως στο Σχήμα 4.1 έχει τη μορφή

$$\frac{\partial S}{\partial t} = Q - \left(Q + \frac{\partial Q}{\partial x} \Delta x \right) + q \Delta x \quad (4.1)$$

όπου S ο όγκος που περιλαμβάνεται μεταξύ των δύο διατομών. Ο όγκος αυτός δίνεται από τη σχέση $S = A \Delta x$. Εάν αντικατασταθεί στην παραπάνω εξίσωση και πραγματοποιηθούν οι απαιτούμενες πράξεις η εξίσωση συνέχειας έχει την εξής τελική μορφή

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = q \quad (4.2)$$

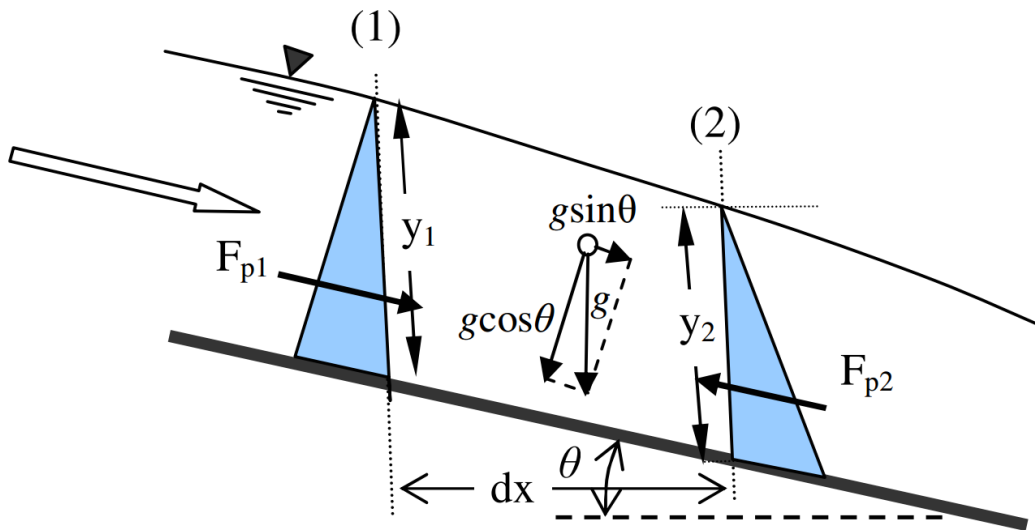
όπου:

- x ο παράλληλος με την κύρια διεύθυνση της ροής άξονας συντεταγμένων
- t ο χρόνος

- A το εμβαδόν της υγρής διατομής
- Q η παροχή
- q η πλευρική εισροή (παροχή ανά μονάδα μήκους)

4.3 Εξίσωση ορμής

Η δεύτερη εξίσωση που χρησιμοποιείται για την περιγραφή της μη μόνιμης ροής είναι αυτή της ορμής ή ποσότητας κίνησης όπως αλλιώς ονομάζεται.



Σχήμα 4.2: Επεξηγηματικό σχήμα με τα χαρακτηριστικά μεγέθη για την εξίσωση ορμής (Πηγή: Παπανικολάου, 2017).

Η εξίσωση ορμής στην πλήρη μορφή της δίνεται από την εξίσωση:

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x} (\beta QV) + gA \frac{\partial y}{\partial x} + gAS_f - gAS_0 = 0 \quad (4.3)$$

όπου:

- β ο συντελεστής συνόρθωσης ορμής
- V η μέση ταχύτητα ροής κατά τη διεύθυνση x
- g η επιτάχυνση της βαρύτητας
- y το βάθος ροής
- S_f η κλίση τριβών
- S_0 η κατά μήκος κλίση του πυθμένα

Εάν γίνει η υπόθεση ότι ο συντελεστής β είναι ίσος με τη μονάδα ($\beta = 1$) για έναν πρισματικό αγωγό και λαμβάνοντας υπόψη πως η μέση ταχύτητα ροής είναι ίση με $V = Q/A$, τότε η (4.3) παίρνει τη μορφή:

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x} \left(\frac{Q^2}{A} \right) + gA \frac{\partial y}{\partial x} + gAS_f - gAS_0 = 0 \quad (4.4)$$

Οι όροι της παραπάνω εξίσωσης μπορεί να θεωρηθεί ότι αντιπροσωπεύουν τα ακόλουθα μεγέθη:

- $\partial Q/\partial t$, όρος τοπικής επιτάχυνσης
- $\partial(Q^2/A)/\partial x$, όρος μεταθετικής επιτάχυνσης
- $gA \partial y/\partial x$, όρος δυνάμεων πίεσης
- gAS_f , όρος δυνάμεων τριβής
- gAS_0 , όρος δυνάμεων βαρύτητας

Στην εξίσωση (4.4) θεωρείται ότι η πλευρική εισροή q εισέρχεται στον αγωγό με κατεύθυνση κάθετη στην κύρια διεύθυνση της ροής και έτσι δεν επηρεάζει την ορμή κατά τον άξονα x . Στην περίπτωση που η εισροή πραγματοποιείται με τυχαία διεύθυνση με συνιστώσα παράλληλη προς την κύρια διεύθυνση της ροής v_x , τότε η εξίσωση ορμής τροποποιείται προσθέτοντας έναν ακόμη όρο και γίνεται:

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x} \left(\frac{Q^2}{A} \right) + gA \frac{\partial y}{\partial x} + gAS_f - gAS_0 + qv_x = 0 \quad (4.5)$$

Το σύστημα λοιπόν των εξισώσεων (4.2) και (4.4) περιγράφουν τη μονοδιάστατη μη μόνιμη ροή σε πρισματικούς ανοιχτούς αγωγούς και αποτελούν τις εξισώσεις Saint Venant, στον οποίο και αποδίδονται. Οι κύριες παραδοχές που χρησιμοποιούνται στις παραπάνω εξισώσεις είναι οι εξής:

- (α) Η κατανομή των πιέσεων είναι υδροστατική.
- (β) Η ταχύτητα είναι ομοιόμορφα κατανεμημένη κατά μήκος της διατομής και συνεπώς ο συντελεστής διόρθωσης β ισούται με τη μονάδα.
- (γ) Η κατά μήκος κλίση του πυθμένα του αγωγού είναι μικρή και επομένως το κατακόρυφο βάθος ροής y ταυτίζεται με το εγκάρσιο βάθος ροής t .
- (δ) Το ρευστό είναι ομογενές και ασυμπίεστο.

Η κλίση των τριβών S_f μπορεί να προσδιοριστεί εμπειρικά από σχέσεις όπως αυτές των Chezy ή Manning. Η πιο διαδεδομένη είναι αυτή του Manning, με την κλίση των τριβών να δίνεται από τον τύπο:

$$S_f = \frac{V^2 n^2}{R^{4/3}} = \frac{Q^2 n^2}{A^2 R^{4/3}} = \frac{Q^2 P^{4/3} n^2}{A^{10/3}} \quad (4.6)$$

όπου:

n ο συντελεστής τραχύτητας Manning

A το εμβαδόν της υγρής διατομής

P η βρεχόμενη περίμετρος

R η υδραυλική ακτίνα ($= A/P$)

Η πιο πάνω σχέση μπορεί να εμφανιστεί και με τη μορφή:

$$S_f = \frac{V|V|n^2}{R^{4/3}} = \frac{Q|Q|n^2}{A^2 R^{4/3}} = \frac{Q|Q|P^{4/3}n^2}{A^{10/3}} \quad (4.7)$$

Γράφοντας τη σχέση της κλίσης των τριβών με τη μορφή της εξίσωσης (4.7) λαμβάνεται υπόψη και η περίπτωση αναστροφής της ροής.

4.4 Σχολιασμός των εξισώσεων Saint Venant

Οι εξισώσεις Saint Venant κατατάσσονται σε σχεδόν γραμμικές, πρώτης τάξης, μερικές διαφορικές εξισώσεις υπερβολικού τύπου. Δεν υπάρχουν αναλυτικές λύσεις κλειστού τύπου για αυτές τις εξισώσεις και κατά συνέπεια χρησιμοποιούνται αριθμητικές μέθοδοι, ως επί το πλείστον, για την επίλυση τους. Όπως είναι λογικό στη μη μόνιμη ροή όλα τα γεωμετρικά και υδραυλικά μεγέθη είναι συναρτήσεις του χρόνου και του χώρου. Οι εξαρτημένες μεταβλητές των εξισώσεων είναι η παροχή Q και το βάθος ροής y . Οι υπόλοιπες μεταβλητές όπως το εμβαδόν A και η κλίση των τριβών S_f μπορούν να εκφραστούν σε όρους των Q και y . Τις ανεξάρτητες μεταβλητές αποτελούν η διαμήκης απόσταση x και ο χρόνος t . Επομένως οι δύο αυτοί άγνωστοι μπορούν να υπολογιστούν από το σύστημα εξισώσεων, της εξίσωσης συνέχειας και της εξίσωσης ορμής. Για να επιλυθούν οι διαφορικές εξισώσεις χρειάζονται μια αρχική συνθήκη και δύο συνοριακές. Η αρχική συνθήκη πρέπει να ορίζει τις τιμές των εξαρτημένων μεταβλητών Q , y που επικρατούν κατά μήκος του αγωγού τη χρονική στιγμή μηδέν ($t = 0$). Για παράδειγμα, η ροή στο κανάλι μπορεί να είναι ομοιόμορφη, δηλαδή για το χρόνο $t = 0$ η παροχή είναι σταθερή και ίση με μία τιμή κατά

μήκος του ποταμού. Εάν η ροή είναι υποκρίσιμη, μία συνοριακή συνθήκη στην είσοδο (ανάντη άκρο) και μια στην έξοδο (κατάντη άκρο) του καναλιού είναι απαραίτητες. Εάν η ροή είναι υπερκρίσιμη τότε και οι δύο συνοριακές συνθήκες πρέπει να δοθούν στο ανάντη άκρο. Η συνοριακή συνθήκη μπορεί να δοθεί με δύο τρόπους, είτε με μια καθορισμένη σχέση μεταξύ των άγνωστων μεταβλητών σε ένα από τα άκρα, είτε παρουσιάζοντας τη μεταβολή μιας εξαρτημένης μεταβλητής στο χρόνο σε ένα άκρο. Η πιο συχνή μορφή είναι η δεύτερη, στην οποία δίδεται το υδρογράφημα εισόδου (ένα γράφημα που απεικονίζει την εξέλιξη της παροχής Q με το χρόνο t), ενώ στην έξοδο θεωρείται ομοιόμορφη ροή.

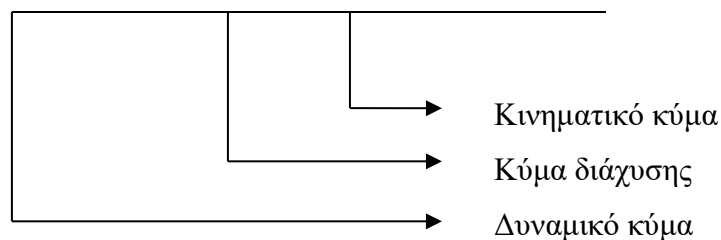
4.5 Τύποι κυμάτων μη μόνιμης ροής

Ανάλογα με την επιθυμητή ακρίβεια με την οποία προσεγγίζεται το εκάστοτε φαινόμενο, παρέχονται διαφορετικά μοντέλα διάδοσης πλημμυρών. Η διαφορά του κάθε μοντέλου έγκειται στους όρους που αυτό χρησιμοποιεί για την περιγραφή της εξίσωσης ορμής. Άρα λοιπόν αξιοποιώντας την πλήρη εξίσωση συνέχειας και απαλείφοντας κάποιους από τους όρους της εξίσωσης ορμής δημιουργούνται τριών ειδών πλημμυρικά κύματα. Πιο συγκεκριμένα διακρίνονται τα εξής:

- το κινηματικό κύμα
- το κύμα διάχυσης
- το δυναμικό κύμα

Οι όροι της εξίσωσης ορμής που αντιστοιχούν σε κάθε ένα από τα τρία αυτά μοντέλα φαίνονται παρακάτω.

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x} \left(\frac{Q^2}{A} \right) + gA \frac{\partial y}{\partial x} + gAS_f - gAS_0 = 0 \quad (4.8)$$



Συμπερασματικά λοιπόν, το πλήρες δυναμικό κύμα αποτελείται από ολόκληρη την εξίσωση ορμής, ενώ για το κύμα διάχυσης απαλείφονται οι όροι $\partial Q/\partial t$ και $\partial(Q^2/A)/\partial x$ με αποτέλεσμα να δημιουργηθεί η εξίσωση $gA \partial y/\partial x + gAS_f - gAS_0 = 0$ ή ισοδύναμα

$\partial y / \partial x + S_f - S_0 = 0$. Αν παραλειφθεί ο όρος $\partial Q / \partial t$ τότε προκύπτει η διαφορική εξίσωση της ανομοιόμορφης μόνιμης ροής. Τέλος στο κινηματικό κύμα αφαιρείται και ο όρος των δυνάμεων πίεσης $gA \partial y / \partial x$ οδηγώντας την εξίσωση ορμής στην πολύ απλή μορφή $gAS_f - gAS_0 = 0$ ή $S_f = S_0$. Στο επόμενο κεφάλαιο θα αναλυθεί η πιο απλή μορφή κύματος, αυτή του κινηματικού κύματος.

4.6 Μοντέλο κινηματικού κύματος

Όπως αναφέρθηκε παραπάνω, για τη μέθοδο του κινηματικού κύματος η εξίσωση ορμής μειώνεται στην απλούστατη μορφή:

$$S_f = S_0 = \frac{Q^2 P^{4/3} n^2}{A^{10/3}} \quad (4.9)$$

το οποίο ερμηνεύεται στο ότι η γραμμή ενέργειας είναι παράλληλη στον πυθμένα του καναλιού. Η εξίσωση συνέχειας, για μηδενική πλευρική εισροή, μαζί με την εξίσωση (4.9) μπορούν να συνδυαστούν και να δημιουργήσουν μία εξίσωση της μορφής:

$$\frac{\partial Q}{\partial t} + c \frac{\partial Q}{\partial x} = 0 \quad (4.10)$$

με το c να εκφράζει την ταχύτητα διάδοσης του κινηματικού κύματος και δίνεται από την σχέση:

$$c = \frac{\partial Q}{\partial A} \quad (4.11)$$

Στην περίπτωση μηδενικής πλευρικής εισροής ($q = 0$), για έναν παρατηρητή που μετακινείται προς τα κατάντη με ταχύτητα c , η μορφή του κύματος δε μεταβάλλεται. Η παραπάνω διαφορική έχει αναλυτική λύση μόνο όταν η πλευρική εισροή είναι μηδέν και για σταθερή ταχύτητα κύματος.

Ας θεωρήσουμε ότι η παροχή δίνεται από την σχέση του Manning, δηλαδή αναμορφώνοντας τη σχέση (4.9) προκύπτει:

$$Q = \frac{1}{n} AR^{2/3} S^{1/2} \quad (4.12)$$

Η παραπάνω σχέση χρησιμοποιείται όταν οι μονάδες είναι εκφρασμένες στο διεθνές σύστημα μονάδων SI. Εάν χρησιμοποιηθεί άλλο σύστημα μονάδων όπως το Βρετανικό και

το Αμερικάνικο, όπου η μονάδα μήκους είναι το πόδι (feet – ft) τότε θα πρέπει να προστεθεί ένας συντελεστής μετατροπής με τιμή 1.49.

Λύνοντας την (4.12) ως προς το εμβαδόν της υγρής διατομής A προκύπτει,

$$A = \left(\frac{nP^{2/3}}{S^{1/2}} \right)^{3/5} Q^{3/5} \quad (4.13)$$

δηλαδή προκύπτει μια σχέση μεταξύ της παροχής Q και του εμβαδού της υγρής διατομής A , της μορφής

$$A = \alpha Q^\beta \quad (4.14)$$

όπου τα α, β αποτελούν παραμέτρους που σχετίζονται με τα γεωμετρικά χαρακτηριστικά του καναλιού και δίνονται από:

$$\alpha = \left(\frac{nP^{2/3}}{S^{1/2}} \right)^{3/5} \quad \text{και} \quad \beta = 3/5 \quad (4.15)$$

Αυτή λοιπόν η εξίσωση (4.14) μαζί με την εξίσωση συνέχειας (4.2) αποτελούν τη μέθοδο του κινηματικού κύματος. Εάν εφαρμοστεί ο κανόνας της αλυσίδας στην εξίσωση συνέχειας, εκείνη μπορεί να γραφτεί ως ακολούθως:

$$\frac{\partial Q}{\partial x} + \frac{\partial A}{\partial Q} \frac{\partial Q}{\partial t} = q \quad (4.16)$$

όμως,

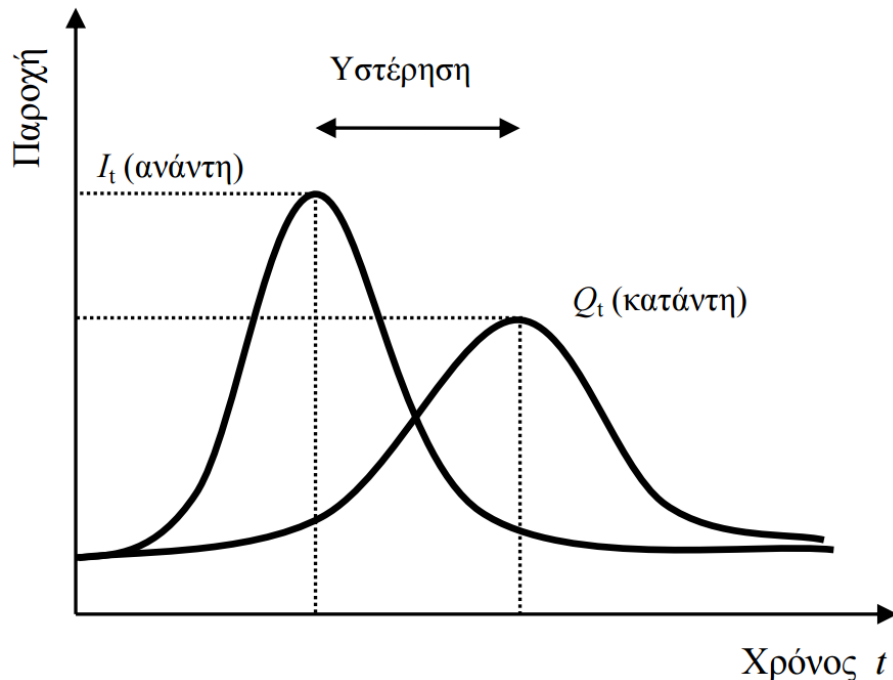
$$\frac{\partial A}{\partial t} = \frac{\partial A}{\partial Q} \frac{\partial Q}{\partial t} = \alpha \beta Q^{\beta-1} \left(\frac{\partial Q}{\partial t} \right) \quad (4.17)$$

και άρα προκύπτει η μερική διαφορική εξίσωση που περιγράφει το κινηματικό κύμα:

$$\frac{\partial Q}{\partial x} + \alpha \beta Q^{\beta-1} \left(\frac{\partial Q}{\partial t} \right) = q \quad (4.18)$$

Για ένα ορθογωνικό κανάλι μεγάλου πλάτους, δηλαδή για πλάτος αγωγού πολύ μεγαλύτερο του βάθους ροής ($b \gg y$) η βρεχόμενη περίμετρος $P = b + 2y$ θα ισούται μόνο με το πλάτος, δηλαδή $P = b$. Έτσι μοναδικός άγνωστος στην παραπάνω διαφορική είναι η παροχή Q , με τις παραμέτρους α, β να είναι σταθερές.

Με τη διόδευση ενός πλημμυρικού κύματος είναι δυνατός ο υπολογισμός του πλημμυρικού υδρογραφήματος σε μία κατάντη θέση του ποταμού, για γνωστό υδρογράφημα σε μία ανάντη θέση. Από την φύση είναι γνωστό ότι, όταν δεν πραγματοποιούνται πλευρικές εισροές, το πλημμυρικό κύμα εμφανίζεται κατάντη με χρονική υστέρηση, μειωμένη αιχμή και μεγαλύτερη διασπορά, όπως φαίνεται χαρακτηριστικά στο Σχήμα 4.3.



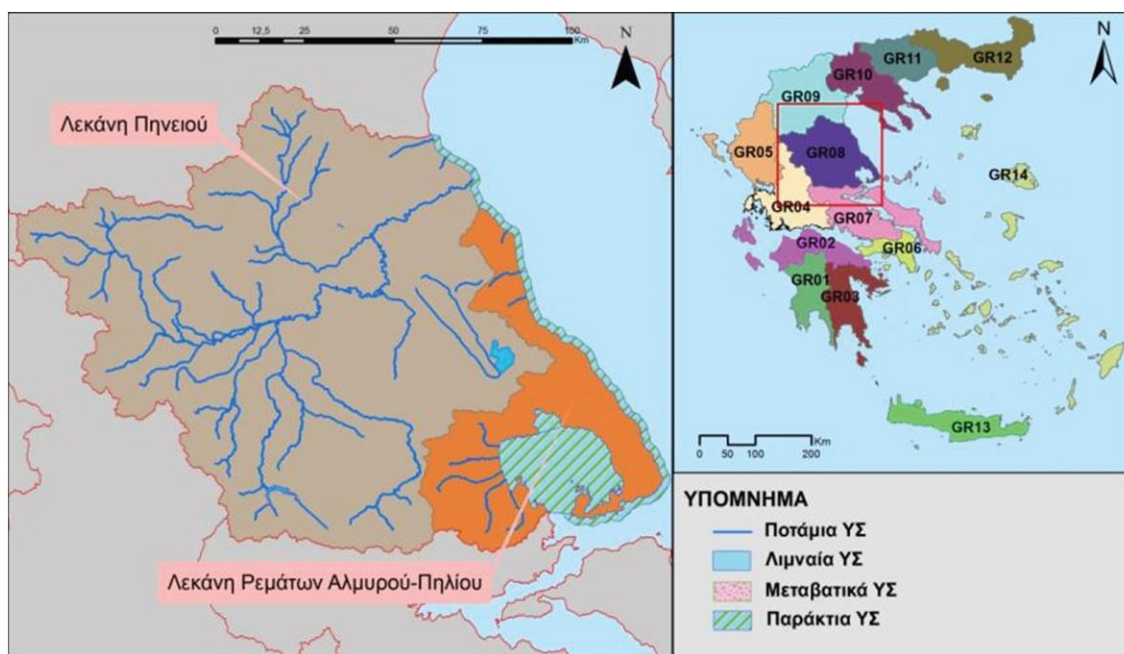
Σχήμα 4.3: Μεταβολή της μορφής ενός πλημμυρικού κύματος.

Σε αντίθεση με τη φυσική πραγματικότητα στο κινηματικό κύμα, όπως αυτό περιγράφεται από τη διαφορική του εξίσωση στην περίπτωση μηδενικής πλευρικής εισροής, η παροχή αιχμής δεν απομειώνεται κατά την πορεία του προς τα κατάντη. Η επίλυση λοιπόν της διαφορικής εξίσωσης του κινηματικού κύματος δεν επιφέρει την απομείωση της αιχμής. Για το λόγο αυτό το κινηματικό κύμα δεν είναι κατάλληλο κύμα για να περιγράψει αυτού του είδους την συνθήκη, ενώ μπορούν να χρησιμοποιηθούν κάποιο από τα άλλα δύο κύματα. Η χρήση όμως αριθμητικών μεθόδων εισάγουν διάχυση μέσω της προσεγγιστικής λύσης που δίνουν και αναιρούν το θεωρητικό αυτό μειονέκτημα.

5 ΕΦΑΡΜΟΓΗ MLP ΣΤΟΝ ΠΟΤΑΜΟ ΠΗΝΕΙΟ

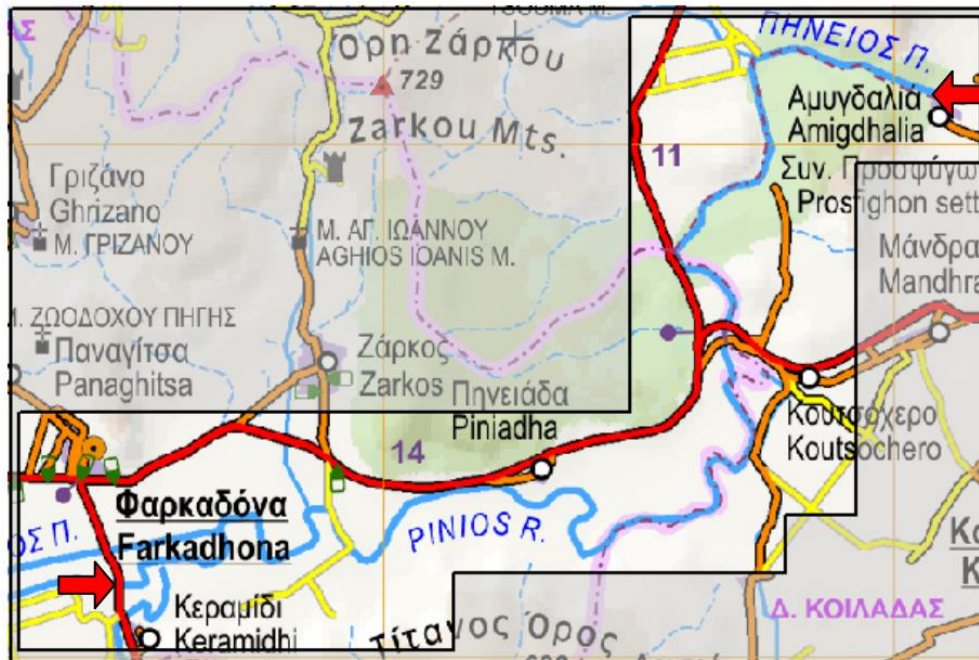
5.1 Περιοχή μελέτης

Ο Πηνειός αποτελεί τον τρίτο μεγαλύτερο ποταμό της Ελλάδας και εκτείνεται εντός του Υδατικού Διαμερίσματος της Θεσσαλίας. Η Λεκάνη Απορροής του Πηνειού αποτελεί την κύρια υδρολογική λεκάνη του Υδατικού Διαμερίσματος της Θεσσαλίας με έκταση της τάξης των 9.500 km² και διατρέχεται από τον ποταμό Πηνειό και τους παραποτάμους του.



Σχήμα 5.1: Υδατικό Διαμέρισμα Θεσσαλίας (Πηγή: www.ypethe.gr).

Η περιοχή μελέτης εντάσσεται στη Δυτική Λεκάνη του Υδατικού Διαμερίσματος της Θεσσαλίας. Αφορά το πλέον κατάντη τμήμα του ποταμού Πηνειού. Το ανάντη όριο βρίσκεται στη θέση Αλή Εφέντη ενώ εκτείνεται μέχρι τη θέση Αμυγδαλιά, που αποτελεί το κατάντη όριο. Το κομμάτι αυτό του ποταμού έχει μήκος περίπου 37 km, με πολύ μικρές κλίσεις ιδιαίτερα στα πρώτα χιλιόμετρα. Στην περιοχή αυτή έχουν κατασκευαστεί κάποια αναχώματα για την προστασία της πεδιάδας από επικείμενες πλημμύρες. Κύρια χρήση της περιοχής είναι η καλλιέργεια της γης. Η γεωμορφολογία της περιοχής είναι σχετικά απλή, με την διατομή του ποταμού να είναι μεταβλητή.



Σχήμα 5.2: Περιοχή μελέτης ποταμού Πηνειού (Πηγή: www.anavasi.gr).

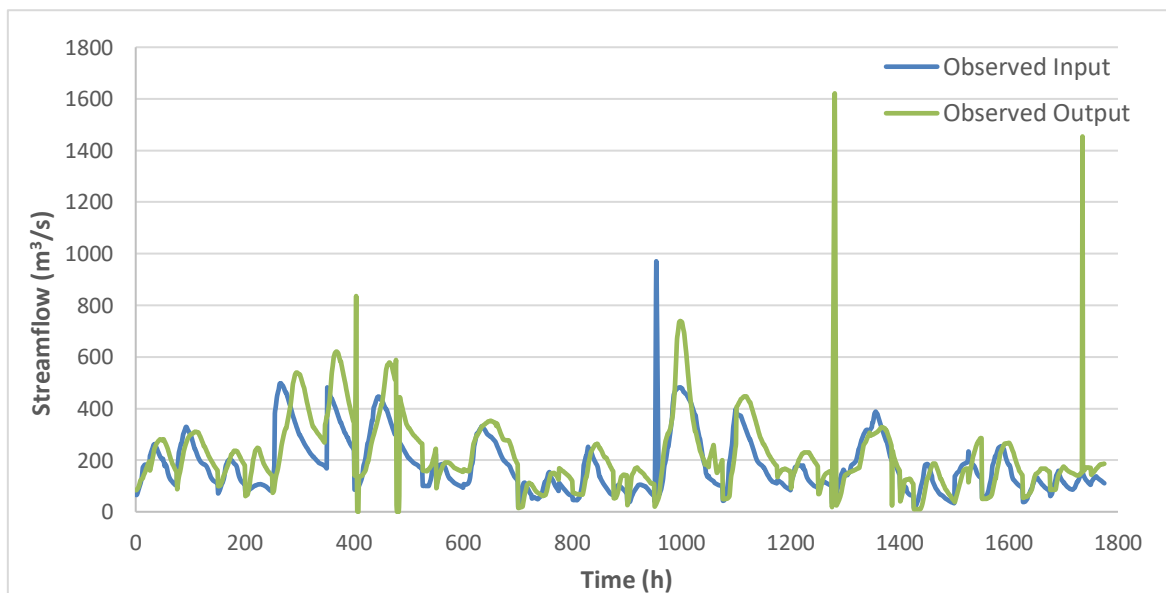
Η περιοχή μελέτης αποτελεί τμήμα ζώνης Δυνητικά Υψηλού Κινδύνου σύμφωνα με την προκαταρκτική αξιολόγηση κινδύνων πλημμύρας, όπως αυτή συντάχθηκε στο πλαίσιο της οδηγίας για την αξιολόγηση και διαχείριση των κινδύνων πλημμύρας (2007/60/ΕΚ).

5.2 Επεξεργασία δεδομένων

Οι δύο σταθμοί που είναι τοποθετημένοι στα δύο όρια, ο ένας στη θέση Αλή Εφέντη και ο δεύτερος στη θέση Αμυγδαλιά, παρέχουν δεδομένα για την παροχή του ποταμού. Τα δεδομένα είναι σε μορφή ωριαίας χρονοσειράς στη διάρκεια της μέρας. Οι μετρήσεις αφορούν τη χρονική περίοδο 11/12/1973 έως 15/02/1979. Να επισημανθεί πως τα δεδομένα των ιστορικών χρονοσειρών προέρχονται από το αρχείο: Υδρολογική διερεύνηση υδατικού διαμερίσματος Θεσσαλίας, Παράρτημα Δ2 (ΕΜΠ, Οκτώβριος 1988). Σε πρώτη ανάλυση συγκεντρώθηκαν οι κοινές ημέρες για τις οποίες υπάρχουν μετρήσεις και στους δύο σταθμούς, όπως φαίνεται στον Πίνακα 5.1. Στο Σχήμα 5.3 παρουσιάζεται η ιστορική χρονοσειρά στους δύο σταθμούς. Στον οριζόντιο άξονα εμφανίζεται ο αύξων αριθμός των ωρών ξεκινώντας από την πρώτη κοινή ημέρα μετρήσεων, δηλαδή 11/12/1973.

Πίνακας 5.1: Κοινές ημέρες χρονοσειρών.

Common Periods		Common Periods	
Start Day	End Day	Start Day	End Day
11/12/1973	11/12/1973	3/2/1976	6/2/1976
15/12/1973	15/12/1973	8/2/1976	8/2/1976
17/12/1973	17/12/1973	17/2/1976	17/2/1976
3/1/1974	5/1/1974	19/2/1976	21/2/1976
11/1/1974	12/1/1974	19/3/1976	19/3/1976
8/2/1974	9/2/1974	12/4/1976	13/4/1976
16/2/1974	19/2/1974	21/4/1976	21/4/1976
21/2/1974	22/2/1974	3/12/1976	7/12/1976
5/3/1974	9/3/1974	14/1/1977	14/1/1977
14/3/1974	14/3/1974	11/12/1977	13/12/1977
12/4/1974	13/4/1974	19/1/1978	19/1/1978
17/4/1974	20/4/1974	22/1/1978	22/1/1978
8/11/1974	10/11/1974	14/2/1978	16/2/1978
19/2/1975	19/2/1975	4/4/1978	5/4/1978
21/2/1975	23/2/1975	30/1/1979	1/2/1979
25/3/1975	25/3/1975	9/2/1979	9/2/1979
19/12/1975	20/12/1975		



Σχήμα 5.3: Ιστορική χρονοσειρά κοινών ημερών των δύο σταθμών του ποταμού Πηνειού.

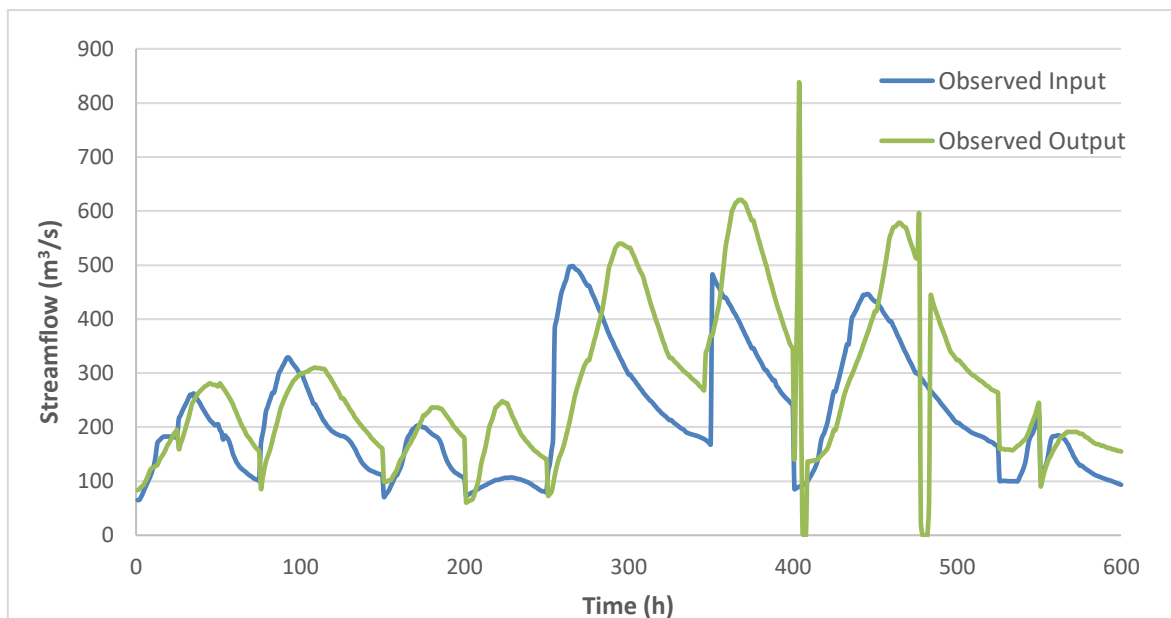
Στόχος της εφαρμογής αποτελεί η εκτίμηση της χρονοσειράς παροχής του σταθμού στη θέση Αμυγδαλιά για γνωστή αυτή του σταθμού στη θέση Αλή Εφέντη. Ο στόχος αυτός επιτεύχθηκε αναπτύσσοντας ένα πολυεπίπεδο τεχνητό νευρωνικό δίκτυο (Multilayer Perceptron-MLP). Για την εκπαίδευση του πολυεπίπεδου τεχνητού νευρωνικού δικτύου

επιλέχθηκαν να χρησιμοποιηθούν οι πρώτες 24 ημέρες, ενώ για την υπόλοιπη χρονοσειρά χρησιμοποιήθηκε το πολυεπίπεδο ΤΝΔ ώστε να εκτιμήσει τη διόδευση του πλημμυρικού κύματος από τον πρώτο στον δεύτερο σταθμό. Για τον σκοπό αυτό έχοντας εκπαιδεύσει το ΤΝΔ, δηλαδή έχοντας προσδιορίσει τα βάρη που συσχετίζουν τις εισόδους με τις εξόδους, δίνονται στο μοντέλο οι καινούργιες τιμές του πρώτου σταθμού (για τις υπόλοιπες ημέρες) και το μοντέλο παράγει τις εξόδους.

Στο Σχήμα 5.4 εμφανίζεται η χρονοσειρά των δύο σταθμών για την οποία το ΤΝΔ εκπαιδεύτηκε και στον Πίνακα 5.2 οι 24 κοινές ημέρες πλημμυρικών δεδομένων για την εκπαίδευση. Από αυτά τα δεδομένα ένα κομμάτι, εδώ το 30 %, αξιοποιήθηκε για την επιβεβαίωση του μοντέλου. Πριν την εφαρμογή του μοντέλου όλα τα δεδομένα κανονικοποιήθηκαν στο διάστημα $[0,1]$ για την καλύτερη απόδοση.

Πίνακας 5.2: Κοινές ημέρες που χρησιμοποιήθηκαν για την εκπαίδευση του ΤΝΔ.

Common Periods	
Start Day	End Day
11/12/1973	11/12/1973
15/12/1973	15/12/1973
17/12/1973	17/12/1973
3/1/1974	5/1/1974
11/1/1974	12/1/1974
8/2/1974	9/2/1974
16/2/1974	19/2/1974
21/2/1974	22/2/1974
5/3/1974	9/3/1974
14/3/1974	14/3/1974
12/4/1974	13/4/1974



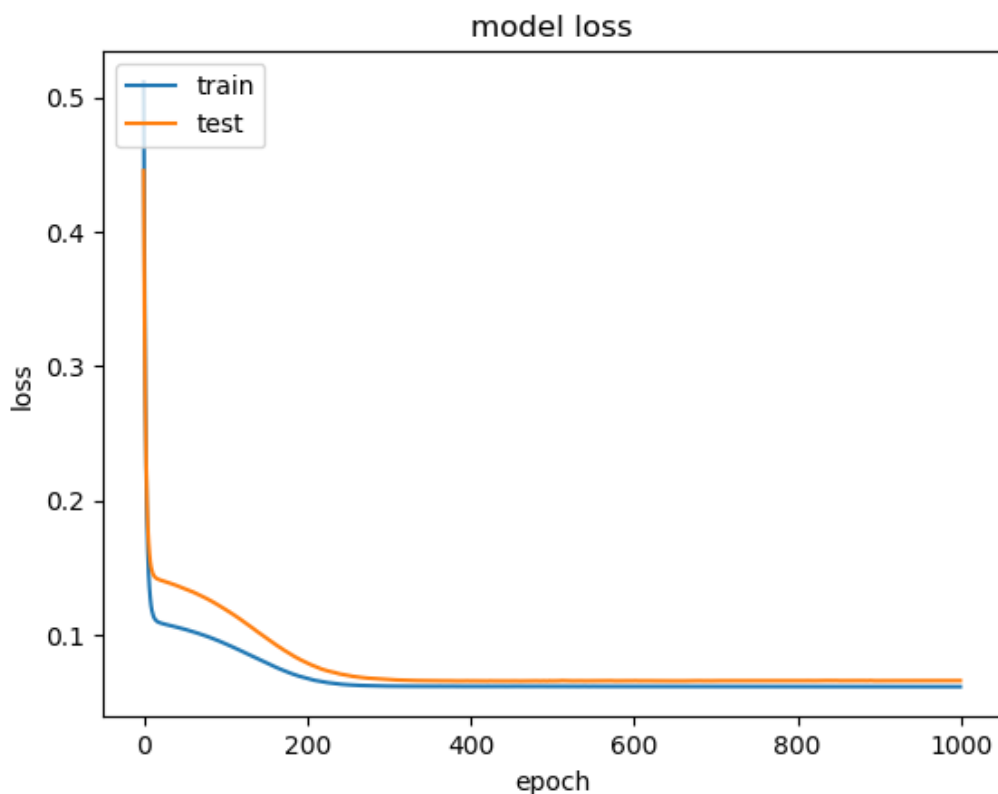
Σχήμα 5.4: Χρονοσειρά εισροών και εκροών που χρησιμοποιήθηκε για την εκπαίδευση του ΤΝΔ.

5.3 Αποτελέσματα

Έπειτα από επανειλημμένες δομικές με την αρχιτεκτονική του δικτύου, παρουσιάζεται στον Πίνακα 5.3 η τελική μορφή του δικτύου που φαίνεται να προσεγγίζει καλύτερα τα δεδομένα. Πιο συγκεκριμένα το πολυεπίπεδο τεχνητό νευρωνικό δίκτυο MLP αποτελείται από 28 επίπεδα χρησιμοποιώντας για την εκπαίδευση την συνάρτηση κόστους του μέσου τετραγωνικού σφάλματος. Σε κάθε κρυμμένο επίπεδο περιλαμβάνονται 20 νευρώνες με τη μη γραμμικότητα να την προσφέρει η συνάρτηση ενεργοποίησης tanh. Η βελτιστοποίηση πραγματοποιήθηκε μέσω του αλγορίθμου SGD (Stochastic Gradient Descent). Η δημιουργία του κώδικα έγινε στη γλώσσα προγραμματισμού Python αξιοποιώντας την βιβλιοθήκη Keras, μία ελεύθερη βιβλιοθήκη κατάλληλη για τη δημιουργία κλασικών τεχνητών νευρωνικών δικτύων.

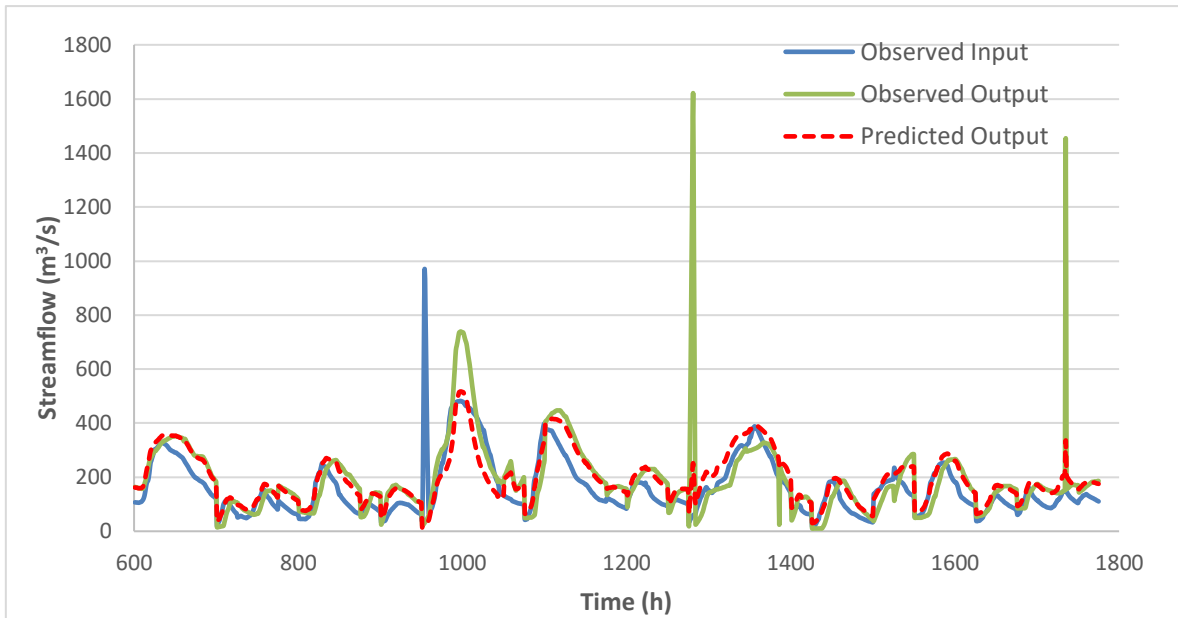
Πίνακας 5.3: Αρχιτεκτονική ΤΝΔ για την εφαρμογή στον ποταμό Πηνειό.

Επίπεδα	28
Νευρώνες	20
Συνάρτηση κόστους	MSE
Συνάρτηση Ενεργοποίησης	Tanh
Αλγόριθμος Βελτιστοποίησης	SGD

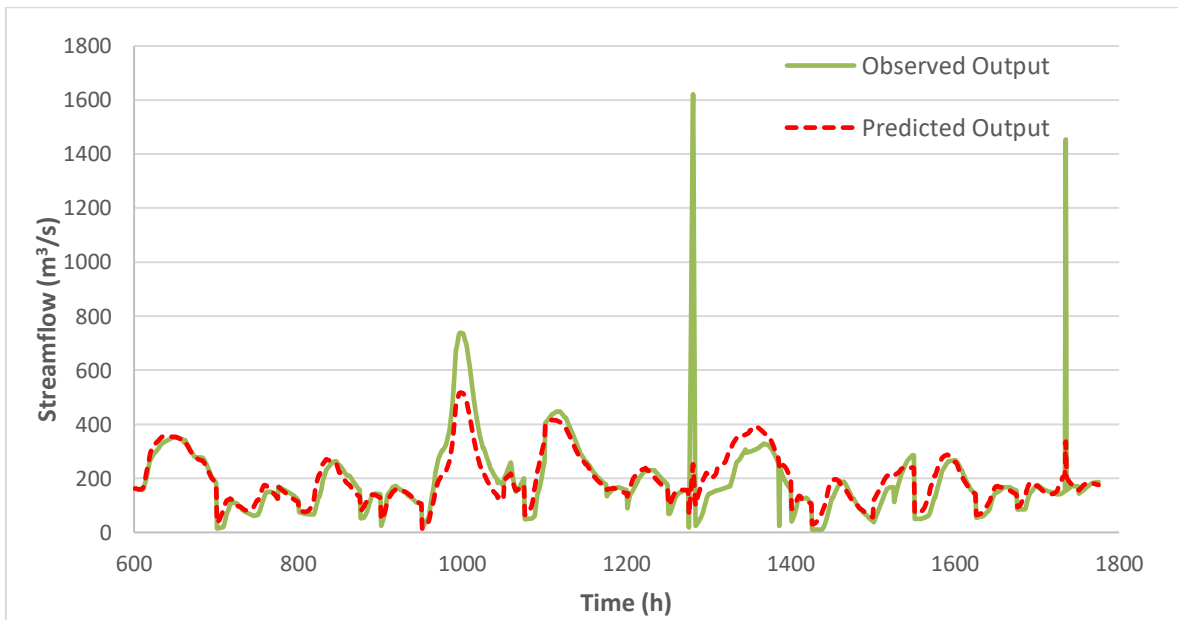


Σχήμα 5.5: Απόδοση εκπαίδευσης ΤΝΔ αναλύοντας τη συνάρτηση κόστους στη διάρκεια των εποχών.

Η απόδοση που είχε το ΤΝΔ κατά την εκπαίδευση φαίνεται στο Σχήμα 5.5, όπου διακρίνεται η συνάρτηση κόστους να μειώνεται στη διάρκεια των εποχών. Στο Σχήμα 5.6 παρουσιάζονται τα αποτελέσματα που προέκυψαν για την εφαρμογή του ΤΝΔ, ξεκινώντας από την 17/4/1974. Φαίνονται τόσο οι μετρούμενες παροχές στην είσοδο και την έξοδο, όσο και η εκτίμηση για την παροχή εξόδου από το μοντέλο. Για την σύγκριση των δύο χρονοσειρών απομονώθηκαν στο Σχήμα 5.7 οι καμπύλες που αφορούν την έξοδο του ποταμού. Παρατηρείται πως η προσέγγιση στις πραγματικές τιμές είναι ικανοποιητική στο μεγαλύτερο κομμάτι της χρονοσειράς. Διακρίνεται ακόμη πως στα κομμάτια μεγάλων τιμών παροχής το ΤΝΔ δεν συμπεριφέρεται με ακρίβεια. Ένας λόγος στον οποίο οφείλεται αυτή η αδυναμία προσέγγισης τόσο υψηλών τιμών, είναι τα δεδομένα που χρησιμοποιήθηκαν για την εκπαίδευση. Το μοντέλο αδυνατεί να εκτιμήσει τιμές εκτός του εύρους των τιμών εκπαίδευσης του, το οποίο και αποτελεί ένα σημαντικό μειονέκτημα.



Σχήμα 5.6: Απεικόνιση μετρούμενης παροχής εισόδου, μετρούμενης παροχής εξόδου και εκτιμώμενης παροχής εξόδου στον ποταμό Πηνειό.



Σχήμα 5.7: Σύγκριση εκτιμώμενης παροχής εξόδου μοντέλου ΤΝΔ και μετρήσεων στον ποταμό Πηνειό.

6 ΕΦΑΡΜΟΓΕΣ PINN

6.1 Εφαρμογή 1: Εξίσωση Burgers (Burgers' Equation)

Μια πρώτη εφαρμογή για την αξιολόγηση αυτής της μεθόδου των τεχνητών νευρωνικών δικτύων θα αποτελέσει η εξίσωση Burgers. Η εξίσωση Burgers αποτελεί μια θεμελιώδη μερική διαφορική εξίσωση με πεδίο εφαρμογής σε πολλούς τομείς των εφαρμοσμένων μαθηματικών. Σε αυτούς συμπεριλαμβάνονται η μηχανική των ρευστών, η μη γραμμική ακουστική, η θερμοδυναμική, η συμπιεστή ροή και η θεωρία πιθανοτήτων. Η εξίσωση αυτή μπορεί να προκύψει από τις εξισώσεις Navier-Stokes εάν δεν λάβουμε υπόψη τον όρο της πίεσης όπως και έκανε ο Ολλανδός επιστήμονας J.M.Burgers. Για ένα δοσμένο πεδίο $u(x, t)$ και γνωστό κινηματικό ιξώδες ν η γενική μορφή της εξίσωσης Burgers γράφεται:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \quad (6.1)$$

Για μικρές τιμές του κινηματικού ιξώδες η εξίσωση Burgers οδηγείται σε μορφή που είναι δύσκολη η επίλυση της με τις κλασικές αριθμητικές μεθόδους. Συγκεκριμένα για μία χωρική διάσταση, η εξίσωση Burgers μαζί με τις οριακές συνθήκες του Dirichlet (Dirichlet boundary conditions), μετατρέπεται σε:

$$u_t + uu_x - \left(\frac{0.01}{\pi}\right) u_{xx} = 0, \quad x \in [-1,1], \quad t \in [0,1] \quad (6.2)$$

$$u(0, x) = -\sin(\pi x),$$

$$u(t, -1) = u(t, 1) = 0$$

Εδώ οι δείκτες t, x και xx δηλώνουν τη μερική παράγωγο ως προς το χρόνο, ως προς το χώρο και την δεύτερη μερική παράγωγο ως προς τον χώρο αντίστοιχα. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο ορίζεται η συνάρτηση $f(t, x)$ με τον ακόλουθο τρόπο:

$$f := u_t + uu_x - \left(\frac{0.01}{\pi}\right) u_{xx} \quad (6.3)$$

Το βαθύ τεχνητό νευρωνικό δίκτυο (deep neural network) προσεγγίζει την λύση $u(x, t)$ της μερικής διαφορικής εξίσωσης. Παρακάτω παραθέτεται το απόσπασμα του κώδικα στο οποίο ορίζεται η μερική διαφορική εξίσωση, για να τονιστεί η απλότητα με την οποία μπορεί να

υλοποιηθεί η κύρια ιδέα της νευρωνικών δικτύων τα οποία υπακούν νόμους της φυσικής. Ο κώδικας γράφτηκε στην γλώσσα προγραμματισμού Python, που αποτελεί την πιο διαδεδομένη γλώσσα για την ανάπτυξη τεχνητών νευρωνικών δικτύων. Επιπλέον χρησιμοποιήθηκε το PyTorch, το οποίο μαζί με το Tensorflow αποτελούν τις δημοφιλέστερες και πιο εμπειριστατωμένες ελεύθερες βιβλιοθήκες για τον χειρισμό προβλημάτων μηχανικής μάθησης.

```
#Create the function to compute the differential equation loss
def dif_loss(model, T_collocation, X_collocation):
    T = T_collocation.clone().detach().requires_grad_(True)
    X = X_collocation.clone().detach().requires_grad_(True)
    u = model(T,X)
    du_dt = grad(u, T, torch.ones_like(T), create_graph =
                 True, retain_graph=True)[0]
    du_dx = grad(u, X, torch.ones_like(X), create_graph =
                 True, retain_graph=True)[0]
    du2_dx2 = grad(du_dx, X, torch.ones_like(X),
                  create_graph=True, retain_graph=True)[0]
    pi = np.pi
    dif_equation = du_dt + u*du_dx-(0.01/pi)*du2_dx2
    return dif_equation
```

Οι παράμετροι του προβλήματος των δύο τεχνητών νευρωνικών δικτύων $u(x, t)$ και $f(t, x)$ προσδιορίζονται από την ελαχιστοποίηση της συνάρτησης του μέσου τετραγωνικού σφάλματος όπως ορίστηκε στο Κεφάλαιο 3.3:

$$MSE = MSE_u + MSE_f \quad (6.4)$$

Ο όρος MSE_u αντιστοιχεί στα αρχικά και συνοριακά σημεία ενώ ο όρος MSE_f προσδίδει την δομή της εξίσωσης (6.2) σε ένα πεπερασμένο αριθμό εσωτερικών σημείων. Οι επιμέρους όροι δίνονται από τις σχέσεις (3.4) και (3.5) αντίστοιχα.

Η επίτευξη του επιδιωκόμενου αποτελέσματος μπορεί να πραγματοποιηθεί με αρκετά μικρό αριθμό δεδομένων εκπαίδευσης N_u , από μερικές εκατοντάδες μέχρι μερικές χιλιάδες. Δεδομένου ότι δεν υπάρχει κάποιο θεωρητικό υπόβαθρο ότι η διαδικασία αυτή οδηγεί σε ολικό ελάχιστο, τα εμπειρικά στοιχεία δείχνουν πως εάν η δοσμένη μερική διαφορική εξίσωση είναι καλά ορισμένη και η λύση της είναι μοναδική, η μέθοδος αυτή είναι ικανή να επιτύχει μια καλή και ακριβή πρόβλεψη, μέσω ενός τεχνητού νευρωνικού δικτύου με επαρκώς ορισμένη αρχιτεκτονική και επαρκή αριθμό εσωτερικών σημείων N_f (Karniadakis et al., 2017). Με τον όρο της αρχιτεκτονικής του δικτύου εννοείται η διαμόρφωση του

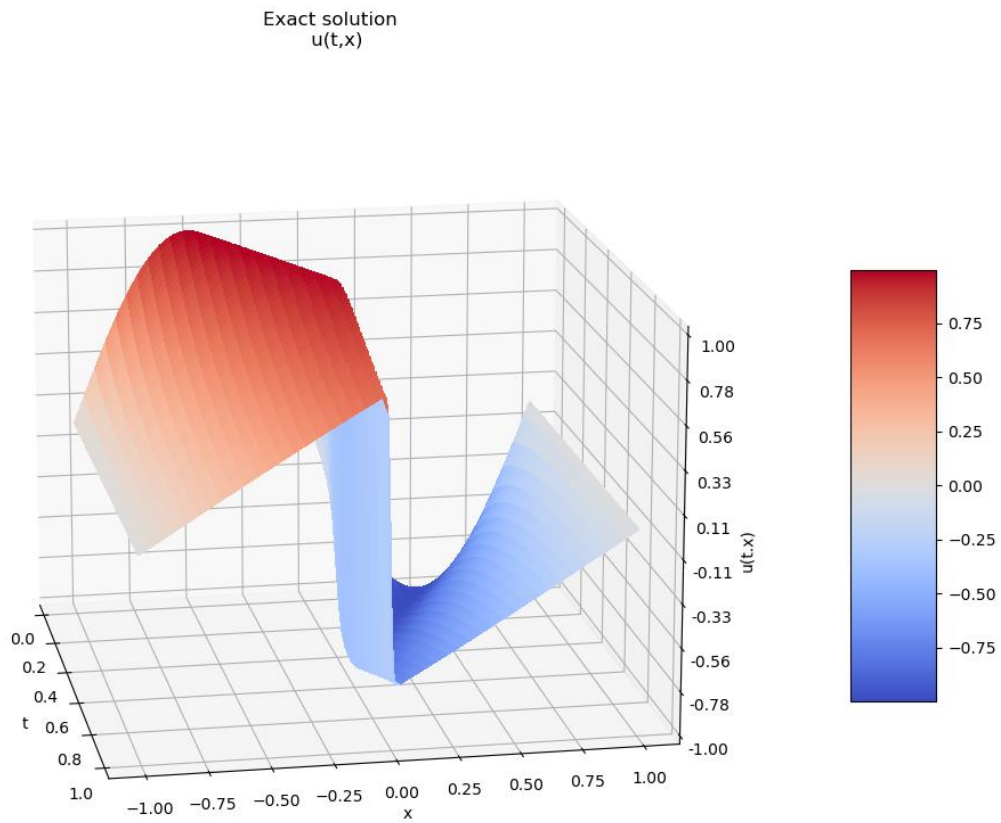
αριθμού των επιπέδων (layers) του νευρωνικού που θα χρησιμοποιηθούν, όπως επίσης και του αριθμού των νευρώνων που θα έχει το κάθε κρυμμένο επίπεδο (hidden layer). Η καλύτερη πρακτική για την εύρεση του ολικού ελαχίστου είναι η πρακτική της δοκιμής και του σφάλματος (trial and error), είτε αυτό σημαίνει αλλαγή του αλγορίθμου βελτιστοποίησης είτε αλλαγή της αρχιτεκτονικής του νευρωνικού δικτύου.

Η αναλυτική λύση της εξίσωσης Burgers είναι διαθέσιμη (C. Basdevant et al., 1986) και απεικονίζεται στο Σχήμα 6.1. Στο Σχήμα 6.2 φαίνεται η λύση που προέκυψε από την επίλυση του τεχνητού νευρωνικού δικτύου, με το ακρωνύμιο PINN. Συγκεκριμένα χρησιμοποιήθηκε ένα ΤΝΔ με 8 επίπεδα, το οποίο περιείχε 20 νευρώνες σε κάθε κρυμμένο επίπεδο. Όσον αφορά την εκπαίδευση του δόθηκαν 200 ζεύγη δεδομένων από τις αρχικές και συνοριακές συνθήκες, δηλαδή ορίστηκαν $N_u = 200$ σημεία ενώ τα εσωτερικά σημεία τέθηκαν $N_f = 20\ 000$. Για κάθε νευρώνα η συνάρτηση ενεργοποίησης που επιλέχθηκε να τοποθετηθεί ήταν αυτή της υπερβολικής εφαπτομένης tanh. Η προσέγγιση της λύσης $u(x, t)$ επιτεύχθηκε αξιοποιώντας την σχέση του μέσου τετραγωνικού σφάλματος (6.4). Για την ακρίβεια η συνάρτηση κόστους βελτιστοποιήθηκε κάνοντας χρήση του αλγορίθμου βελτιστοποίησης της στοχαστικής κατάβασης κλίσης (stochastic gradient descent).

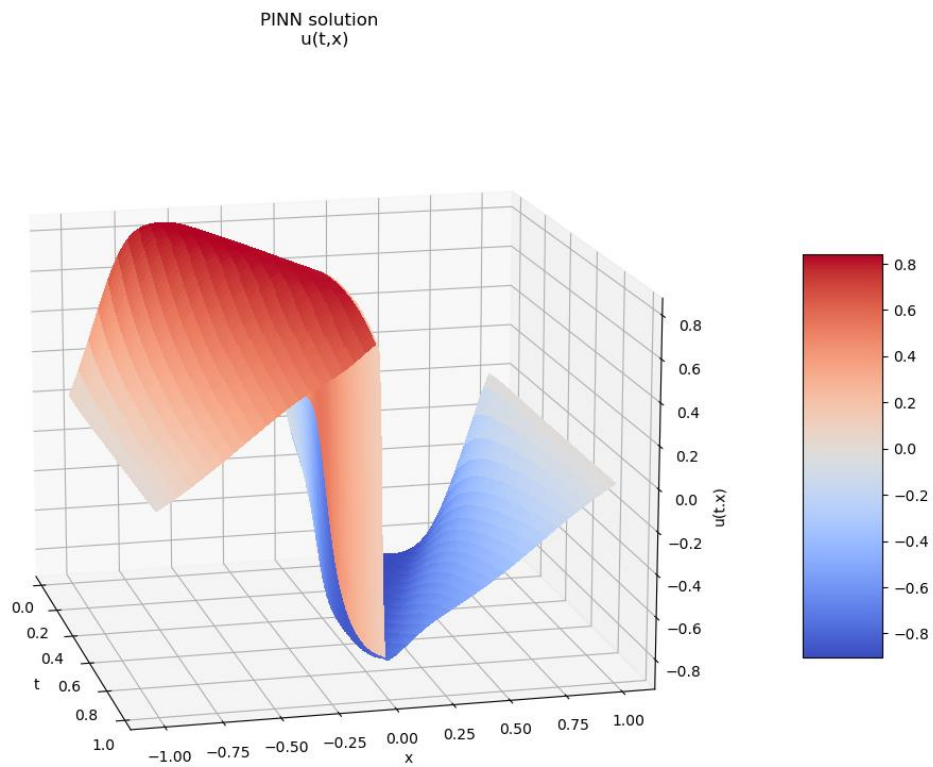
Η απόκλιση της μεθόδου η οποία υπολογίστηκε μέσω της σχετικής L_2 ή Ευκλείδειας νόρμας, όπως αλλιώς λέγεται, μετρήθηκε στην τιμή $8.0 \times 10^{-4} \%$ και αποτελεί μια πολύ καλή προσέγγιση. Η L_2 νόρμα για ένα διάνυσμα $\mathbf{x} = (x_1, x_2, x_3)$ δίδεται από την σχέση $\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + x_3^2}$. Η σχετική L_2 νόρμα είναι ένας τρόπος για τη μέτρηση σφαλμάτων. Η εξίσωση που την περιγράφει είναι η εξής:

$$L_2 \text{ error} = \frac{\sqrt{\sum_{i=1}^N (u_{exact}(i) - u_{PINN}(i))^2}}{\sqrt{\sum_{i=1}^N u_{exact}(i)^2}} \quad (6.5)$$

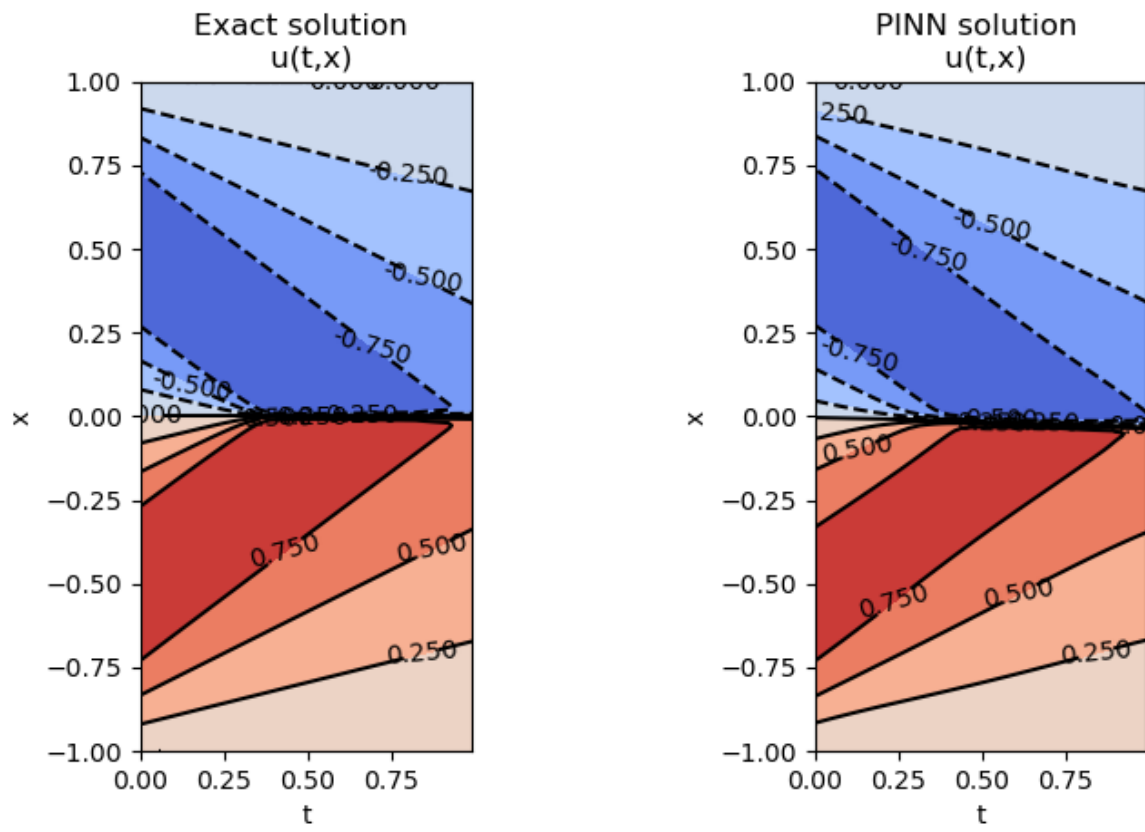
όπου στον αριθμητή λαμβάνεται η νόρμα της διαφοράς της εκτιμώμενης $u_{PINN}(i)$ από την αναλυτική $u_{exact}(i)$ λύση ενώ αυτή διαιρείται με την νόρμα την αναλυτικής λύσης.



Σχήμα 6.1: Τρισδιάστατη απεικόνιση αναλυτικής λύσης εξίσωσης Burgers.



Σχήμα 6.2: Τρισδιάστατη απεικόνιση της λύσης που προέκυψε από το τεχνητό νευρωνικό δίκτυο για την εξίσωση Burgers.

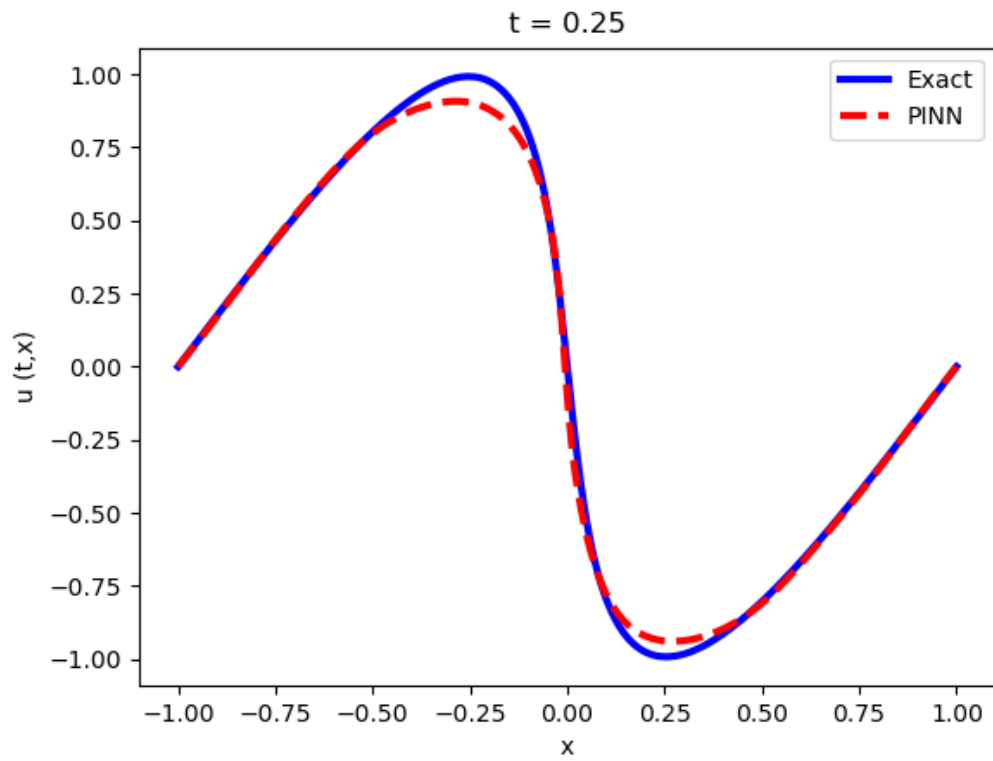


(α)

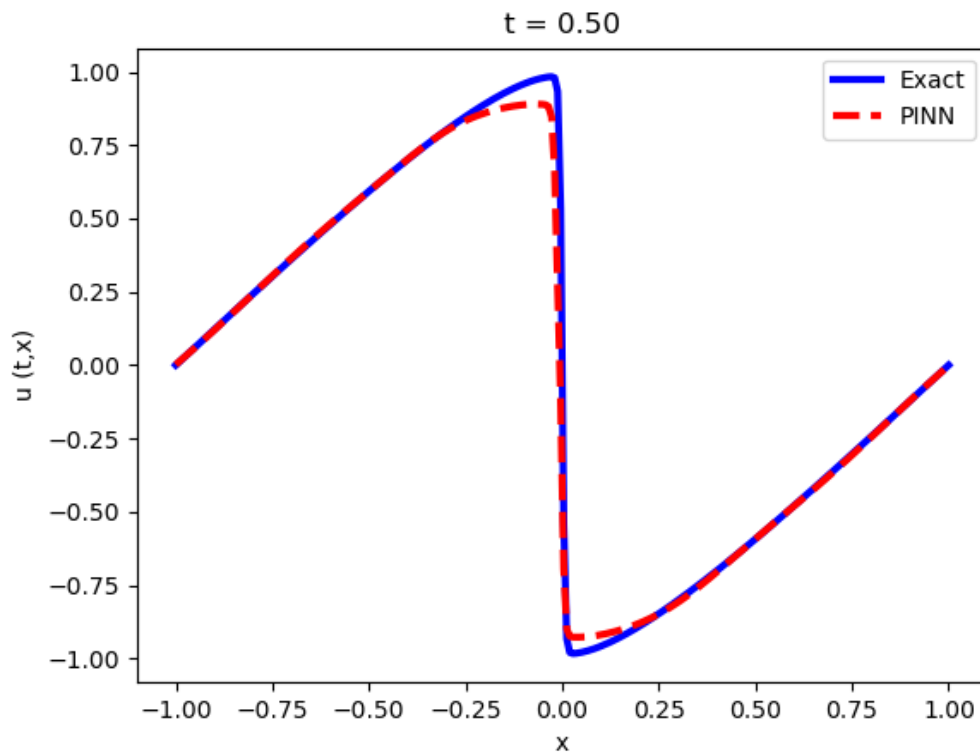
(β)

Σχήμα 6.3: (α) Δισδιάστατη απεικόνιση αναλυτικής λύσης εξίσωσης Burgers, (β) Δισδιάστατη απεικόνιση εκτιμώμενης λύσης εξίσωσης Burgers

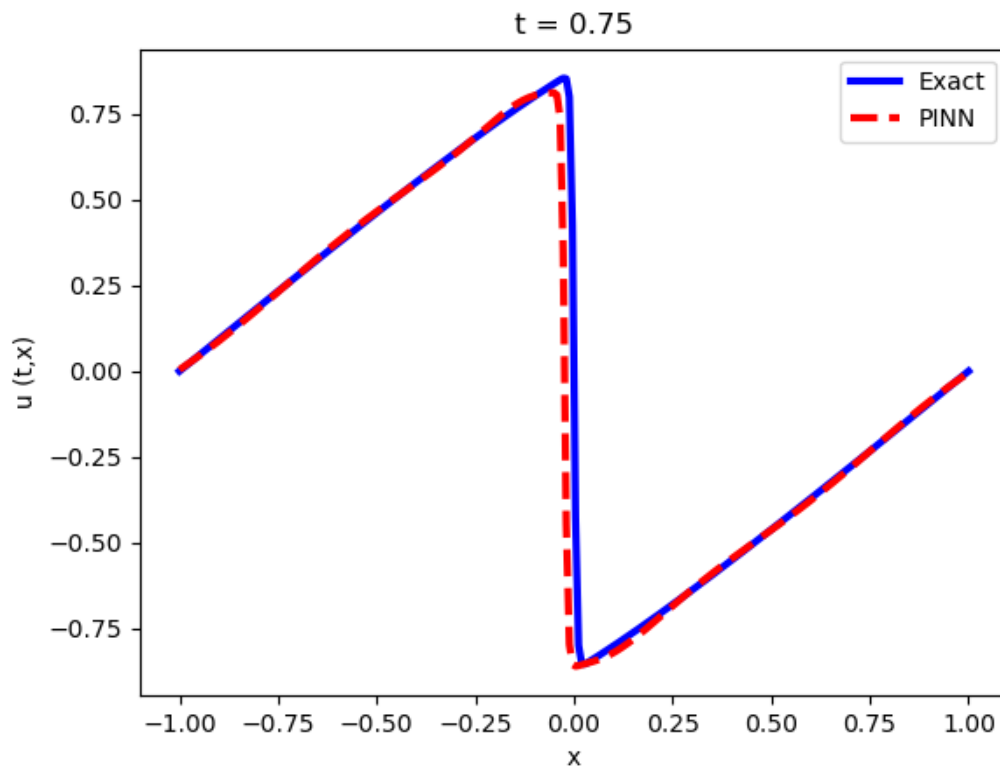
Για την περαιτέρω κατανόηση των αποτελεσμάτων, στα γραφήματα των Σχημάτων 6.4, 6.5, 6.6 αποτυπώνονται η γραφική παράσταση της αναλυτικής λύσης της μερικής διαφορικής εξίσωσης και η προσέγγιση που έγινε από το ΤΝΔ στο χώρο, για συγκεκριμένες χρονικές στιγμές $t = 0.25, 0.50, 0.75$. Αξιοποιώντας μόνο τις αρχικές και συνοριακές συνθήκες, το τεχνητό νευρωνικό δίκτυο κατάφερε να προσεγγίσει με μεγάλη ακρίβεια την πολυπλοκότητα και κυρίως τη μη γραμμικότητα η οποία διέπει την συγκεκριμένη μερική διαφορική εξίσωση. Κάτι ανάλογο είναι σαφώς πιο δύσκολο να επιτευχθεί από κλασικές αριθμητικές μεθόδους, στις οποίες θα απαιτηθεί μια επίμονη και σχολαστική διακριτοποίηση στο χώρο και στο χρόνο. Η καλύτερη προσέγγιση των απότομων αλλαγών (κορυφών) που εμφανίζονται, μπορεί να υλοποιηθεί εάν τροποποιηθούν οι παράμετροι του προβλήματος. Μια σημαντική τέτοια παράμετρο αποτελεί ο ρυθμός μάθησης (learning rate).



Σχήμα 6.4: Σύγκριση αναλυτικής και εκτιμώμενης λύση για $t=0.25$.



Σχήμα 6.5: Σύγκριση αναλυτικής και εκτιμώμενης λύσης για $t=0.50$.



Σχήμα 6.6: Σύγκριση αναλυτικής και εκτιμώμενης λύσης για $t=0.75$.

Συγκεντρωτικά στον Πίνακα 6.1 φαίνεται η τελική αρχιτεκτονική του ΤΝΔ για την εφαρμογή της εξίσωσης Burgers.

Πίνακας 6.1: Αρχιτεκτονική ΤΝΔ εφαρμογής εξίσωσης Burgers.

Επίπεδα	8
Νευρώνες	20
Σημεία N_u	200
Σημεία N_f	20 000
Συνάρτηση Ενεργοποίησης	tanh
Αλγόριθμος Βελτιστοποίησης	SGD
Σχετική L_2 νόρμα	8.0×10^{-4}

Ο τρόπος με τον οποίο στήνεται η αρχιτεκτονική του τεχνητού νευρωνικού δικτύου όπως και ο αριθμός των σημείων τα οποία επιλέγονται για την επίτευξη μεγαλύτερης ακρίβειας αποτελεί πεδίο έρευνας. Η γενικότερη κατεύθυνση που προέρχεται κυρίως από την εμπειρία εκπαίδευσης τέτοιων αλγορίθμων, δείχνει πως η ικανότητα πρόβλεψης των νευρωνικών

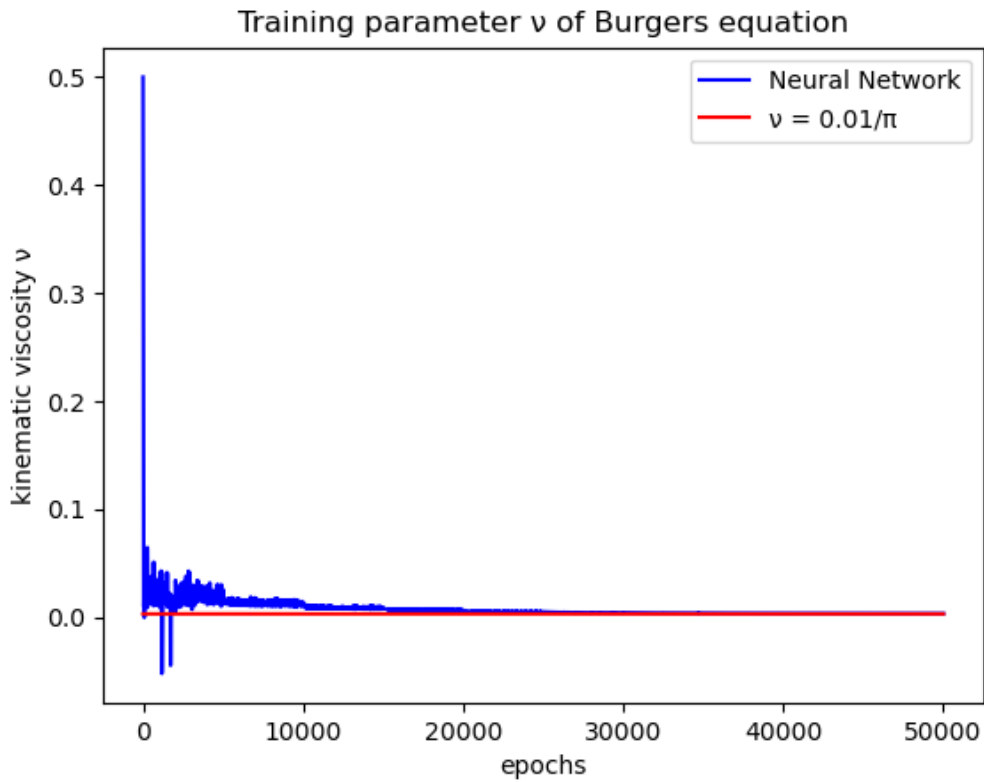
δικτύων αυξάνεται με την αύξηση των δεδομένων εισόδου N_u και N_f . Σχετικά με την αρχιτεκτονική αυτών των δικτύων, πάλι από ερευνητικά στοιχεία τροποποιώντας τον αριθμό των κρυμμένων επιπέδων και των νευρώνων που τα αποτελούν, προκύπτει ότι η αύξηση τους δίνει μεγαλύτερη προγνωστική ακρίβεια.

Επιθυμώντας να αξιοποιήσουμε την ταυτοποίηση των παραμέτρων, την οποία προσφέρουν τα ΤΝΔ δημιουργήθηκε ένα βαθύ τεχνητό νευρωνικό δίκτυο για την πραγματοποίηση της. Στα δεδομένα εισόδου αυτή την φορά ενσωματώνονται τα σημεία της αναλυτικής λύσης για την επίλυση με δεδομένη την παράμετρο της εξίσωσης, δηλαδή το κινηματικό ιξώδες $\nu = 0.01/\pi$. Η αρχιτεκτονική του νευρωνικού δικτύου παρέμεινε ίδια. Το κινηματικό ιξώδες ν αντιμετωπίζεται σαν παράμετρος του προβλήματος και έτσι ορίζεται στον κώδικα. Παρακάτω δίνεται το σημείο του κώδικα στο οποίο ορίζεται το κινηματικό ιξώδες.

```
#Define kinematic viscosity  $\nu$  as Variable  
self.Viscosity = torch.nn.Parameter(torch.tensor(0.5,  
                                             requires_grad=True))
```

Η παράμετρος ν αρχικά πήρε την τιμή 0.5 και μέσω βελτιστοποίησης η τιμή αυτή άλλαζε συνεχώς κατά την διάρκεια των εποχών (epochs) για να επιτευχθεί η παράμετρος που περιγράφει κατάλληλα τα δεδομένα. Χρησιμοποιήθηκε ο αλγόριθμος βελτιστοποίησης Adam, ο οποίος συμπεριφέρθηκε καλύτερα συγκριτικά με τις άλλες επιλογές, όπως αυτή του SGD (Stochastic Gradient Descent).

Όπως αναφέρθηκε στο κεφάλαιο των αλγορίθμων βελτιστοποίησης συνηθίζεται η χρήση πολλών εποχών (epochs), για την βελτίωση της λύσης του προβλήματος και την εύρεση των παραμέτρων. Κάθε εποχή είναι ένας γύρος εκπαίδευσης πάνω στα δεδομένα. Η μεταβολή της παραμέτρου του κινηματικού ιξώδες ν κατά την διάρκεια των εποχών και η σύγκλιση της φαίνεται στο Σχήμα 6.7. Ο κατακόρυφος άξονας περιγράφει τη μεταβολή της παραμέτρου, ενώ ο οριζόντιος αριθμεί τις εποχές. Επίσης διακρίνεται και η πραγματική τιμή της παραμέτρου ν σαν σταθερή ευθεία. Εξετάζεται με αυτόν τον τρόπο η επιρροή του αριθμού των εποχών στην ταυτοποίηση των παραμέτρων και η ταχύτητα με την οποία το μοντέλο κατορθώνει να προσεγγίσει την επιθυμητή τιμή. Ο ρυθμός μάθησης (learning rate) αρχικά τέθηκε στην τιμή $l_r = 0.02$, ενώ όσο το τεχνητό νευρωνικό δίκτυο πλησίαζε τη λύση ο αριθμός αυτός υποδιπλασιαζόταν κατορθώνοντας έτσι την εύρεση του ολικού ελαχίστου.



Σχήμα 6.7: Αποτέλεσμα ΤΝΔ ταυτοποίησης της παραμέτρου κινηματικού ιξώδες ν της εξίσωσης Burgers.

Ξεκινώντας από την τιμή 0.5, η οποία τέθηκε ως αρχική τιμή, το βαθύ τεχνητό νευρωνικό δίκτυο μόλις από τις πρώτες εποχές κυμαίνεται σε ένα διάστημα μεταξύ 0.8 και -0.8, ενώ παρατηρούμε πως περίπου στις 25 000 εποχές συγκλίνει στην πραγματική τιμή. Η τελική τιμή του κινηματικού ιξώδες ν μετά από τις 50 000 εποχές παίρνει την τιμή $\nu = 0.0034$. Σε σύγκριση με την πραγματική τιμή $\nu = 0.01/\pi = 0.0032$ οδηγούμαστε στο συμπέρασμα ότι το ΤΝΔ προσεγγίζει τη λύση με μεγάλη ακρίβεια.

6.2 Εφαρμογή 2: Εξίσωση κινηματικού κύματος (Kinematic wave equation)

Για την σύγκριση της μεθόδου των ΤΝΔ παρουσιάζεται η λύση και με αριθμητική μέθοδο, όπως άλλωστε συνηθίζεται σε τέτοιου είδους προβλήματα. Η λύση της εξίσωσης (4.18) πραγματοποιήθηκε με αριθμητική ολοκλήρωση μέσω γραμμικών πεπερασμένων διαφορών. Η εξίσωση γραμμικοποιήθηκε αντικαθιστώντας τον μέσο όρο των γνωστών λύσεων στον συντελεστή Q που αποτελεί τον συντελεστή του μη γραμμικού όρου. Πιο συγκεκριμένα οι όροι της εξίσωσης αντικαθιστώνται από τα εξής μεγέθη:

$$\frac{\partial Q}{\partial x} = \frac{Q_{i+1}^{j+1} - Q_i^{j+1}}{\Delta x} \quad (6.6)$$

$$Q = \left(\frac{Q_{i+1}^j + Q_i^{j+1}}{2} \right) \quad (6.7)$$

$$\frac{\partial Q}{\partial t} = \left(\frac{Q_{i+1}^{j+1} - Q_{i+1}^j}{\Delta t} \right) \quad (6.8)$$

$$q = \left(\frac{q_{i+1}^{j+1} + q_{i+1}^j}{2} \right) \quad (6.9)$$

Έτσι η εξίσωση οδηγείται στην ακόλουθη μορφή:

$$\frac{Q_{i+1}^{j+1} - Q_i^{j+1}}{\Delta x} + \alpha\beta \left(\frac{Q_{i+1}^j + Q_i^{j+1}}{2} \right)^{\beta-1} \left(\frac{Q_{i+1}^{j+1} - Q_{i+1}^j}{\Delta t} \right) = \left(\frac{q_{i+1}^{j+1} + q_{i+1}^j}{2} \right) \quad (6.10)$$

ενώ αν λυθεί ως προς τον όρο Q_{i+1}^{j+1} προκύπτει

$$Q_{i+1}^{j+1} = \frac{\left[\frac{\Delta t}{\Delta x} Q_i^{j+1} + \alpha\beta \left(\frac{Q_{i+1}^j + Q_i^{j+1}}{2} \right)^{\beta-1} Q_{i+1}^j + \Delta t \left(\frac{q_{i+1}^{j+1} + q_{i+1}^j}{2} \right) \right]}{\left[\frac{\Delta t}{\Delta x} + \alpha\beta \left(\frac{Q_{i+1}^j + Q_i^{j+1}}{2} \right)^{\beta-1} \right]} \quad (6.11)$$

Στους όρους αυτής της λύσης οι δείκτες αναφέρονται στον χώρο ενώ οι εκθέτες αναφέρονται στο χρόνο. Η διαδικασία ολοκλήρωσης προχωράει αυξάνοντας το i μέχρι το τέλος, για δεδομένο j , ενώ στη συνέχεια επαναφέρεται στην αρχική τιμή το i και αυξάνεται το j .

Για την ανάλυση της διόδευσης μιας πλημμύρας μέσω του μοντέλου του κινηματικού κύματος, γίνεται η εφαρμογή σε ένα κανάλι ορθογωνικής διατομής με τα ακόλουθα χαρακτηριστικά:

- πλάτος αγωγού $b = 200 \text{ ft} = 60.96 \text{ m}$
- μήκος αγωγού $L = 24\,000 \text{ ft} = 7\,315.2 \text{ m}$
- κλίση πυθμένα $S_0 = 0.01$
- συντελεστής τραχύτητας Manning $n = 0.035$

Συγκεντρωτικά στον Πίνακα 6.2 δίνονται τα γεωμετρικά δεδομένα της εφαρμογής.

Πίνακας 6.2: Γεωμετρικά δεδομένα αγωγού εφαρμογής κινηματικού κύματος.

Πλάτος αγωγού, b (m)	60.96
Μήκος αγωγού, L (m)	7 315.2
Κλίση πυθμένα, S_0	0.01
Συντελεστής τραχύτητας, n	0.035

Η συνοριακή συνθήκη δίνεται από το υδρογράφημα εισόδου, το οποίο φαίνεται στον Πίνακα 6.3, ενώ για την αρχική συνθήκη θεωρείται ομοιόμορφη παροχή κατά μήκος του καναλιού $Q_0 = 2\,000 \text{ cfs} = 56.63 \text{ m}^3/\text{s}$.

Πίνακας 6.3: Υδρογράφημα εισόδου

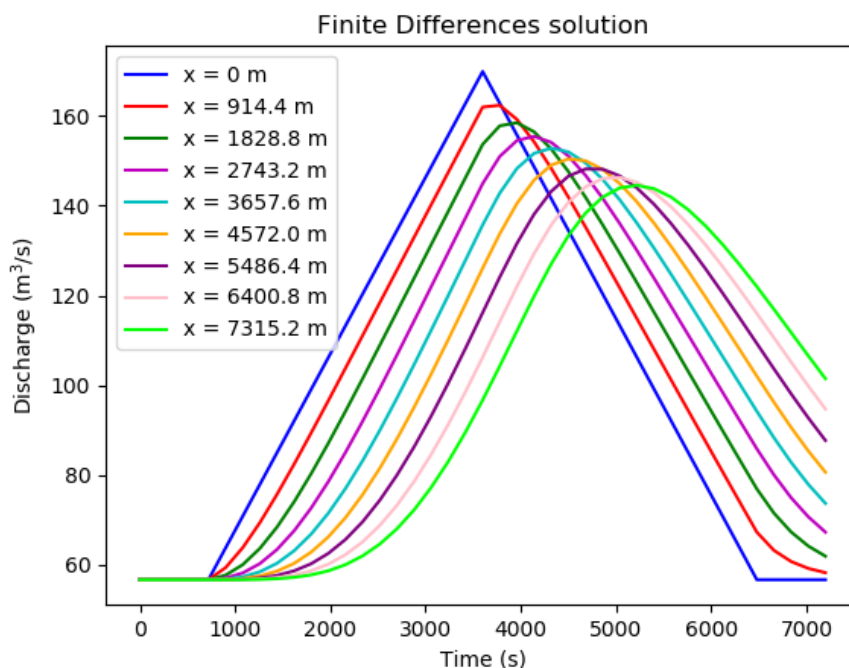
Χρόνος (min)	Παροχή εισόδου (cfs)	Παροχή εισόδου (m^3/s)
0	2 000	56.63
12	2 000	56.63
24	3 000	84.95
36	4 000	113.27
48	5 000	141.58
60	6 000	169.90
72	5 000	141.58
84	4 000	113.27
96	3 000	84.95
108	2 000	56.63
120	2 000	56.63

Ο λόγος για τον οποίο οι μονάδες μήκους παρουσιάζονται και σε πόδια (feet – ft) είναι διότι με αυτόν τον τρόπο αναγράφονται στην βιβλιογραφία (Example-Kinematic Wave, CIVE322 Basic Hydrology), από όπου λήφθηκαν τα δεδομένα της εφαρμογής. Το συγκεκριμένο υδρογράφημα αποτελεί ένα τριγωνικό διάγραμμα με μέγιστη παροχή αυτή των $169.90 \text{ m}^3/\text{s}$ και παροχή βάσης $56.63 \text{ m}^3/\text{s}$. Επιπλέον προσδιορίζοντας το βάθος ροής y από την εξίσωση (4.12) για τη μέγιστη παροχή $Q = 169.90 \text{ m}^3/\text{s}$ προκύπτει ένα βάθος ροής $y = 1.0 \text{ m}$, το οποίο είναι πολύ μικρότερο του πλάτους b ($b \gg y$) και άρα ο αγωγός μπορεί να θεωρηθεί ορθογωνικός μεγάλου πλάτους. Επομένως οι συντελεστές α, β παίρνουν τις αντίστοιχες τιμές:

$$\alpha = \left(\frac{nP^{2/3}}{S^{1/2}} \right)^{3/5} = \left(\frac{0.035 * 60.96^{2/3}}{0.01^{1/2}} \right)^{3/5} = 2.76 \quad \text{και} \quad \beta = 3/5 \quad (6.12)$$

Η επίλυση μέσω της αριθμητικής μεθόδου των πεπερασμένων διαφορών πραγματοποιήθηκε για χρονική και χωρική διακριτοποίηση, $\Delta t = 3 \text{ min} = 180 \text{ s}$ και $\Delta x = 3 \text{ 000 ft} = 914.40 \text{ m}$ αντίστοιχα.

Παρακάτω στο Σχήμα 6.8 φαίνεται η διόδευση του κύματος όπως αυτό προέκυψε από την εξίσωση (6.11). Στο σχήμα παρουσιάζεται η εξέλιξη της παροχής στο χρόνο για διάφορες κατάντη θέσεις του καναλιού μέχρι το τέλος του.



Σχήμα 6.8: Διόδευση κινηματικού κύματος με χρήση γραμμικών πεπερασμένων διαφορών.

Είναι εμφανής η μείωση της παροχής αιχμής οδεύοντας από τα ανάντη προς τα κατόντη του καναλιού, όπως επίσης και η μεγάλη διασπορά, γεγονός που οφείλεται στην αριθμητική μέθοδο που χρησιμοποιήθηκε.

Το τεχνητό νευρωνικό δίκτυο, το οποίο χρησιμοποιήθηκε για την επίλυση της μερικής διαφορικής εξίσωσης του κινηματικού κύματος (4.18), δημιουργήθηκε στο προγραμματιστικό περιβάλλον της Python. Ο κώδικας κατασκευάστηκε κάνοντας χρήση της βιβλιοθήκης PyTorch.

Δεδομένου πως η πλευρική εισροή q στην συγκεκριμένη εφαρμογή θεωρείται μηδενική, η εξίσωση (4.18) γράφεται:

$$\frac{\partial Q}{\partial x} + \alpha\beta Q^{\beta-1} \left(\frac{\partial Q}{\partial t} \right) = 0 \quad (6.13)$$

Ορίζεται η συνάρτηση $f(t, x)$ ως το αριστερό μέλος της εξίσωσης () ως εξής:

$$f := \frac{\partial Q}{\partial x} + \alpha\beta Q^{\beta-1} \left(\frac{\partial Q}{\partial t} \right) \quad (6.14)$$

Τα δεδομένα εισόδου του ΤΝΔ έχουν κανονικοποιηθεί για να διευκολυνθεί το νευρωνικό δίκτυο στην επίλυση. Έχει συμπεριληφθεί το κομμάτι του κώδικα στο οποίο ορίζεται η υπό μελέτη μερική διαφορική εξίσωση, όπως αυτή προκύπτει μετά την κανονικοποίηση που επιβλήθηκε. Στον παρακάτω κώδικα το u συμβολίζει την παροχή Q .

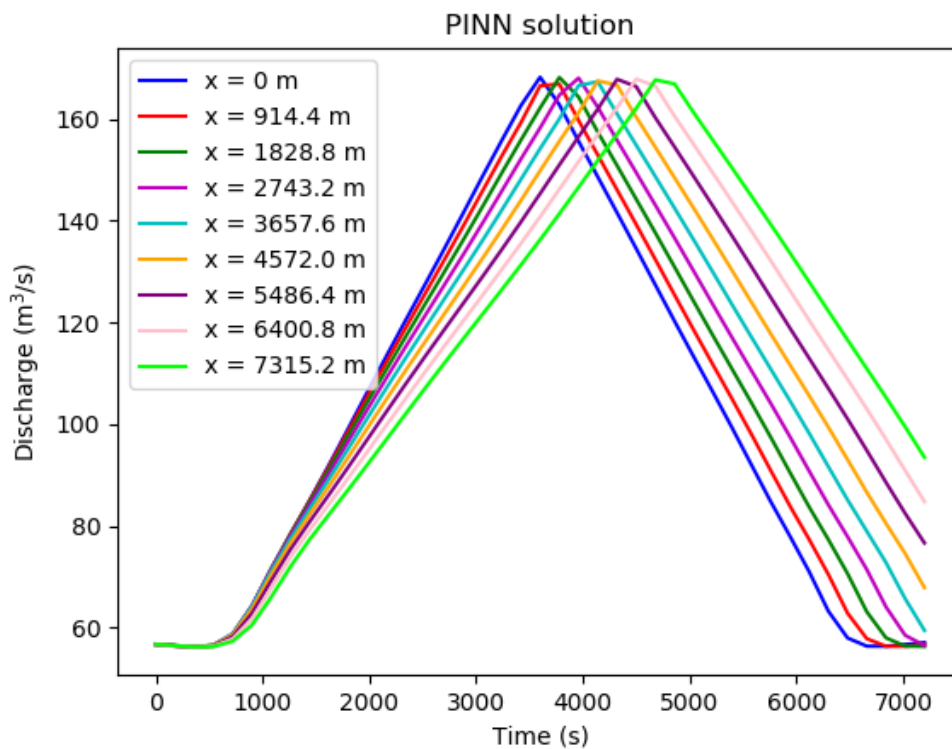
```
#Create the function to compute the differential equation loss
def dif_loss(model, X_collocation, T_collocation):
    X = X_collocation.clone().detach().requires_grad_(True)
    T = T_collocation.clone().detach().requires_grad_(True)
    u = model(X, T)
    du_dx = grad(u, X, torch.ones_like(X), create_graph=True,
                 retain_graph=True)[0]
    du_dt = grad(u, T, torch.ones_like(T), create_graph=True,
                 retain_graph=True)[0]

    dif_equation=du_dx+1.016*a*b*abs(113.27*u+56.63)**(b-1)*du_dt
    return dif_equation
```

Το σύνολο των σημείων που αφορά τις συνοριακές συνθήκες επιλέχθηκε να είναι $N_u = 200$ ενώ τα εσωτερικά τέθηκαν σε $N_f = 5\,000$ τυχαία καταναμημένα σημεία στο πεδίο ορισμού της εξίσωσης. Το ΤΝΔ αποτελείται από 8 επίπεδα νευρώνων, με κάθε επίπεδο να περιέχει 20 νευρώνες και την υπερβολική εφαπτομένη \tanh να έχει επιλεγθεί σαν συνάρτηση

ενεργοποίησης. Η εκπαίδευση έγινε χρησιμοποιώντας την συνάρτηση του μέσου τετραγωνικού σφάλματος. Όσον αφορά τον αλγόριθμο βελτιστοποίησης, ο αλγόριθμος Adam χρησιμοποιήθηκε, καθώς βρέθηκε να συμπεριφέρεται καλύτερα σε σχέση με τους υπόλοιπους.

Στη συνέχεια στο Σχήμα 6.9 παρουσιάζεται το υδρογράφημα που προκύπτει από το βαθύ τεχνητό νευρωνικό δίκτυο, ενώ στον Πίνακα 6.4 συνοψίζεται η αρχιτεκτονική του δικτύου αυτού.



Σχήμα 6.9: Διόδευση κινηματικού κύματος με χρήση τεχνητού νευρωνικού δικτύου.

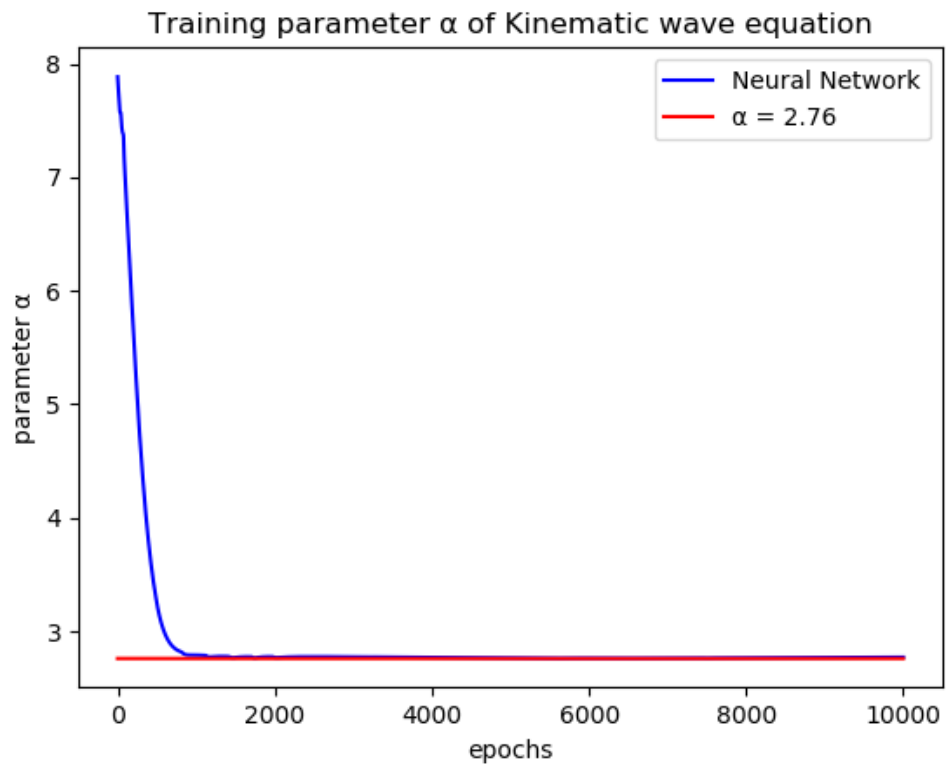
Πίνακας 6.4: Αρχιτεκτονική ΤΝΔ εφαρμογής κινηματικού κύματος.

Επίπεδα	8
Νευρώνες	20
Σημεία N_u	200
Σημεία N_f	5 000
Συνάρτηση Ενεργοποίησης	tanh
Αλγόριθμος Βελτιστοποίησης	Adam

Πολλές φορές η γνώση των παραμέτρων του προβλήματος να μην είναι δεδομένη. Πιο συγκεκριμένα στο πρόβλημα της διόδευσης των πλημμυρών η διατομή τις περισσότερες φορές είναι γνωστή, αλλά το μέγεθος της τραχύτητας n συχνά επιλέγεται μέσω προσεγγιστικών θεωρήσεων. Η εφαρμογή των ΤΝΔ τα οποία υπακούν νόμους της φυσικής (PINN) δίνει την δυνατότητα της εύρεσης των παραμέτρων της εξίσωσης με αρκετά ακριβή τρόπο. Η συλλογή της αναλυτικής λύσης μέσω εργαστηριακών δοκιμών ή από μετρήσεις πεδίου είναι ιδιαίτερα δύσκολη και κοστοβόρα. Στα πλαίσια της συγκεκριμένης εφαρμογής η αναλυτική λύση λήφθηκε από την λύση που προσέφερε το προηγούμενο ΤΝΔ με γνωστές παραμέτρους. Όπως είναι φανερό η άγνωστη παράμετρος θα είναι η παράμετρος α της εξίσωσης (6.13). Για τον σκοπό αυτό, δημιουργήθηκε ένα βαθύ τεχνητό νευρωνικό δίκτυο διατηρώντας την αρχιτεκτονική του ίδια με πριν. Η παράμετρος α ορίζεται ως μεταβλητή στην γλώσσα προγραμματισμού Python, με την απλή διαδικασία που φαίνεται παρακάτω.

```
#Define Parameter  $\alpha$  as Variable  
self.ParameterA = torch.nn.Parameter(torch.tensor(8.0,  
                                             requires_grad=True))
```

Στην παράμετρο α δόθηκε αρχικά η τιμή 8, η οποία βελτιστοποιήθηκε με σκοπό την καλύτερη περιγραφή των δεδομένων εισόδου. Για την ακρίβεια έγινε χρήση του αλγορίθμου Adam. Η μεταβολή της παραμέτρου α κατά την διάρκεια των εποχών και η σύγκλιση της φαίνεται στο Σχήμα 6.10. Η τελική τιμή στην οποία καταλήγει το ΤΝΔ είναι $\alpha = 2.77$ ενώ ο στόχος είναι η τιμή 2.76. Σε σύνολο 10 000 εποχών διακρίνουμε την ταχύτητα προσέγγισης αλλά και την ακρίβεια του μοντέλου.



Σχήμα 6.10: Αποτέλεσμα ΤΝΔ ταυτοποίησης της παραμέτρου α της εξίσωσης του κινηματικού κύματος.

7 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΤΑΣΕΙΣ ΓΙΑ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ

7.1 Συμπεράσματα

Η παρούσα εργασία είχε ως στόχο τη μελέτη της συνεισφοράς των τεχνητών νευρωνικών δικτύων στα προβλήματα διόδευσης πλημμυρών και τη παρουσίαση μίας σχετικά νέας μεθόδου επίλυσης μερικών διαφορικών εξισώσεων για την λύση των εξισώσεων μη μόνιμης ροής με έμφαση στο κινηματικό κύμα. Δεν είναι σκοπός της εργασίας η αντικατάσταση των αριθμητικών που ήδη υπάρχουν (π.χ. πεπερασμένα στοιχεία) με ακριβή αποτελέσματα σε πολλές περιπτώσεις. Οι δύο μεθοδολογίες μπορούν να συνυπάρξουν στους αλγόριθμους πρόβλεψης και επίλυσης διαφορικών εξισώσεων. Παρ' όλα αυτά μετά την επεξεργασία των εφαρμογών οδηγούμαστε σε κάποια συμπεράσματα για τα τεχνητά νευρωνικά δίκτυα, τα οποία συνοψίζονται ακολούθως:

- (α) Τα απλά πολυεπίπεδα τεχνητά νευρωνικά δίκτυα τα οποία δέχονται εισόδους και εξόδους από πραγματικά δεδομένα προσδιορίζουν το φαινόμενο με αρκετά μεγάλη ακρίβεια, αλλά έχουν το μειονέκτημα του ότι το αποτέλεσμα εξαρτάται από την ποιότητα και την ποσότητα των διαθέσιμων δεδομένων για εκπαίδευση.
- (β) Η πρόβλεψη της διόδευσης της πλημμύρας από τον πρώτο σταθμό στο δεύτερο στην περίπτωση του ποταμού Πηνειού με χρήση του MLP ήταν ικανοποιητική με αποκλίσεις να υπάρχουν κυρίως σε κομμάτια δεδομένων με υψηλές τιμές, οι οποίες ήταν εκτός του εύρους των δεδομένων εκπαίδευσης.
- (γ) Τα PINN προσεγγίζουν την αναλυτική λύση με μεγάλη ακρίβεια παίρνοντας πληροφορία από τις παραγώγους καθώς επίσης δεν διακριτοποιούν το πεδίο του χώρου και του χρόνου, κάτι που δεν συμβαίνει στις αριθμητικές μεθόδους.
- (δ) Η ευκολία στην υλοποίηση και η μη ανάγκη διαφορετικής μεταχείρισης του εκάστοτε προβλήματος, όπως συμβαίνει με τις αριθμητικές μεθόδους, αποτελούν ένα μεγάλο πλεονέκτημα των δικτύων αυτών.
- (ε) Από την εφαρμογή της εξίσωσης Burgers προέκυψε η ακρίβεια με την οποία τα PINN προσεγγίζουν την αναλυτική λύση από πολύπλοκες εξισώσεις όπως είναι η Burgers.

- (στ) Η εξίσωση του κινηματικού κύματος λύθηκε για έναν ορθογωνικό αγωγό μεγάλου πλάτους χωρίς πλευρική εισροή. Η λύση ήταν ικανοποιητική με εμφανή τη μη μείωση της παροχής αιχμής, σε αντίθεση με τη μέθοδο των πεπερασμένων διαφορών.
- (ζ) Και στις δύο εφαρμογές των δικτύων PINN προσδιορίστηκε με μεγάλη ακρίβεια το αντίστροφο πρόβλημα της εύρεσης παραμέτρου με γνωστή τη λύση της διαφορικής, αποδίδοντας στα PINN ένα ακόμη πλεονέκτημα.

Τα συμπεράσματα που προέκυψαν από την παρούσα εργασία είναι ιδιαίτερα ενθαρρυντικά τόσο για την προσφορά των τεχνητών νευρωνικών δικτύων στην επίλυση προβλημάτων μηχανικού, όπως αυτό της διόδευσης, αλλά και για τη μελλοντική τους ανάπτυξη.

7.2 Προτάσεις για μελλοντική έρευνα

Μεγαλύτερη αξία σε αυτή τη διπλωματική εργασία έχει η εισαγωγή μιας καινούργιας μεθόδου επίλυσης των εξισώσεων Saint Venant, αυτή των τεχνητών νευρωνικών δικτύων. Η παρούσα εργασία περιορίστηκε στην επίλυση της εξίσωσης του κινηματικού κύματος, χωρίς αυτό να σημαίνει πως δεν είναι εφικτή η επίλυση των εξισώσεων Saint Venant στην πλήρη μορφή τους. Το σύστημα αυτό των εξισώσεων που καλείται να διαχειριστεί ένας μελλοντικός ερευνητής, στο πλαίσιο των τεχνητών νευρωνικών δικτύων, είναι επιλύσιμο. Ο κώδικας δεν θα αλλάξει σημαντικά όσον αφορά τα κύρια σημεία του, αλλά είναι σαφές ότι θα διαφοροποιηθεί. Δεδομένη είναι η προσθήκη μίας ακόμη εξίσωσης όπως επίσης και ένας ακόμη όρος στη συνάρτηση κόστους, ο οποίος θα αφορά τη δεύτερη εξίσωση του συστήματος. Οι προτάσεις για μελλοντική έρευνα συνοψίζονται στις εξής:

- (α) Εκμετάλλευση των τεχνητών νευρωνικών δικτύων στην πρόγνωση πλημμυρών.
- (β) Επίλυση των εξισώσεων μη μόνιμης ροής στην πλήρη μορφή τους μέσω τεχνητών νευρωνικών δικτύων και πειραματική επαλήθευση των αποτελεσμάτων.
- (γ) Σύγκριση αριθμητικών μεθόδων επίλυσης των εξισώσεων Saint Venant με μεθόδους τεχνητών νευρωνικών δικτύων.
- (δ) Αξιοποίηση της δυνατότητας των ΤΝΔ για την εύρεση παραμέτρων εξίσωσης, σε περιοχές όπου η ύπαρξη των δεδομένων το επιτρέπει.
- (ε) Βελτιστοποίηση της αρχιτεκτονικής των ΤΝΔ που ασχολούνται με προβλήματα διόδευσης πλημμυρών.

8 ΑΝΑΦΟΡΕΣ

- Δασκαλάκη, Ε., *Υδραυλική και υδρολογική ανάλυση μη μόνιμης ροής σε πρισματικό αγωγού*, Μεταπτυχιακή εργασία, ΔΠΜΣ «Επιστήμη και Τεχνολογία Υδατικών Πόρων», Αθήνα, 2017.
- Ε.Κ., 2007, *Οδηγία 2007/60/ΕΚ του Ευρωπαϊκού Κοινοβουλίου και του Συμβουλίου της 23ης Οκτωβρίου 2007 για την αξιολόγηση και τη διαχείριση των κινδύνων πλημμύρας*, Επίσημη Εφημερίδα της Ευρωπαϊκής Ένωσης της 6.11.2007: L 288: 27 - 34.
- Κουτσογιάννης, Δ., *Σχεδιασμός Αστικών Δικτύων Αποχέτευσης*, Έκδοση 4, Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, 2011.
- Νουτσόπουλος, Γ., Γ. Χριστοδούλου και Τ. Παπαθανασιάδης, *Υδραυλική Ανοικτών Αγωγών*, Έκδοση 2, Fountas, Αθήνα, 2010.
- Οικονόμου, Α., *Διερεύνηση λειτουργίας λογισμικών υδραυλικής προσομοίωσης στην εξέλιξη πλημμυρικής κατάκλυσης. Εφαρμογή στην πεδιάδα της Θεσσαλίας*, Μεταπτυχιακή εργασία, ΔΠΜΣ «Επιστήμη και Τεχνολογία Υδατικών Πόρων», Αθήνα, 2013.
- Παπανικολάου, Π. Ν., *Στοιχεία Μόνιμης Ροής σε Αγωγούς με Ελεύθερη Επιφάνεια*, Έκδοση 4, Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, 2016.
- Παπανικολάου, Π., Κουτσογιάννης, Δ., και Στάμου, Α., *Οδηγίες για την παρουσίαση πανεπιστημιακών εργασιών στον τομέα Υδατικών Πόρων και Περιβάλλοντος*, Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, 2012.
- Πέτσιου, Α., *Ανάλυση αβεβαιότητας των παραμέτρων πλημμυρικής διόδευσης*, Εφαρμογή στον ποταμό Πηνειό, Διπλωματική εργασία, Σχολή Πολιτικών Μηχανικών ΕΜΠ, Αθήνα, 2017.
- Akan, O., *Open Channel Hydraulics*, 1st Edition, Elsevier Butterworth-Heinemann, pp. 384 2006.
- Aurélien Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, O'Reilly Media, United States of America, pp. 510, 2019.

- Basdevant, C., Deville, M., Haldenwang, P., Lacroix, J., Ouazzani, J., Peyret, R., Orlandi, P., Patera, A., *Spectral and finite difference solutions of the Burgers equation*, *Computers & fluids* 14, 1986 (23–41).
- Chow, V. T., Maidment, D. R., and Mays, L. W., *Applied Hydrology*, McGraw Hill, New York, 1988.
- Chua, L. H. C., Wong, T. S.W., and Sriramula, L.K., Comparison between kinematic wave and artificial neural network models in event-based runoff simulation for an overland plane, *ScienceDirect*, pp.12, 2008.
- DeVries, J. J., MacArthur, R.C., *Introduction and application of kinematic wave routing techniques*, The Hydrologic Engineering Center, U.S. Army Corps of Engineers, California, Document No. 10, 1979.
- Doshi, S., Various Optimization Algorithms for Training Neural Network, *Towards Data Science*, 2019, (<https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>).
- Example-Kinematic Wave, CIVE322 Basic Hydrology, Colorado State University, (https://www.engr.colostate.edu/~ramirez/ce_old/classes/cive322-Ramirez/CE322_Web/ExampleKinematicWave.pdf).
- Fenton, J. D., Flood routing methods, *Journal of Hydrology*, pp. 14, 2019.
- Haghighat, E., Raissi, M., Moure, A., Gomez, H., and Juanes, R., A deep learning framework for solution and discovery in solid mechanics, arXiv:2003.02751v2, pp.24, 2020.
- Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1999.
- Kingma, D. P., Ba, J., Adam: A method for stochastic optimization, arXiv:1412.6980v9, pp.15, 2017.
- Krzaczyński, R., Limited-Memory-Broyden-Fletcher-Goldfarb-Shanno Algorithm, *Towards Data Science*, 2020, (<https://towardsdatascience.com/limited-memory-broyden-fletcher-goldfarb-shanno-algorithm-in-ml-net-118dec066ba>).
- Landajuela, M., *Burgers Equation*, BCAM Internship, 2011.

- McGonagle, J., Shalkouski, G., Williams, C., Backpropagation, *Brilliant*, (<https://brilliant.org/wiki/backpropagation/>).
- Mujumdar, P. P., Flood Wave Propagation: The Saint Venant Equations, *Resonance*, pp.8, 2001.
- Peters, R., Schmitz, G., and Cullmann, J., Flood routing modelling with Artificial Neural Network, *Advances in Geosciences*, pp. 6, 2006.
- Purves, D., Augustine, G. J., Fitzpatrick, D., Katz, L. C., Lamantia, A. S., McNamara, J. O., *Neuroscience*, Sinauer Associates, Sutherland, Mass, 1997.
- Pytlak, Radoslaw, *Limited Memory Quasi-Newton Algorithms*, Conjugate Gradient Algorithms in Nonconvex Optimization. Springer, pp. 159–190, 2009.
- Raissi, M., Perdikaris, P., and Karniadakis, G. Em., Physics Informed Deep Learning (Part1): Data-driven Solutions of Nonlinear Partial Differential Equations, arXiv:1711.10561v1 , pp. 22, 2017.
- Razavi, S., Karamouz, M., Adaptive Neural Networks for Flood Routing in River Systems, *Water International*, 32:3, pp. 360-375, 2007.
- Reggiani, P., Todini, E., and Meißner, D., Analytical solution of kinematic wave approximation for channel routing, *Hydrology Research*, pp. 15, 2014.
- Retsinis, E., Daskalaki, E., and Papanicolaou, P., Dynamim flood wave routing in prismatic channels with hydraulic and hydrologic methods, *Journal of Water Supply*, pp.12, 2019.
- Rosenblatt, F., The perceptron: a probabilistic model for information storage and organization in the brain., in *Psychological review* vol. 65, pp. 6, 1958.
- Smolyakov, V., Neural Network Optimization Algorithms, *Towards Data Science*, 2018. (<https://towardsdatascience.com/neural-network-optimization-algorithms-1a44c282f61d>).
- Wilson, A., Simple Neural Networks in Python, *Towards Data Science*, 2019, (<https://towardsdatascience.com/inroduction-to-neural-networks-in-python-7e0b422e6c24>).

Zahidul Islam, Md., Flood Routing by Kinematic Wave Model, *Research Gate*, pp. 164, 2002.


```

min_u = np.float64(0.00)
#max_u1 = np.amax(Input)
#min_u1 = np.min(Input)
Train_Input = (Input - min_u) / (max_u - min_u) # normalize the input
so that it falls in the range between [0,1]
#max_u2 = np.amax(Output)
#min_u2 = np.min(Output)
Train_Output = (Output - min_u) / (max_u - min_u) # normalize the
output so that it falls in the range between [0,1]

history = model.fit(Train_Input,Train_Output,epochs=1000,
validation_split=0.3)

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# predict for new inputs
New_Inputs = data['NewInput'].flatten()[:, None]
#max_u3 = np.amax(New_Inputs)
#min_u3 = np.min(New_Inputs)
New_Inputs1 = (New_Inputs - min_u) / (max_u - min_u) # normalize the
new inputs so that it falls in the range between [0,1]
Predicts = model.predict([New_Inputs1])

Real_pred = Predicts*(max_u-min_u)+min_u

```

Κώδικας PINN λύσης εξίσωσης Burgers

```
import torch
import numpy as np
from pyDOE import lhs
from torch.autograd import grad
cuda = torch.device('cuda')
from scipy.interpolate import griddata
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

np.random.seed(1234)
torch.manual_seed(1234)
# Problem parameters
h = 2
t = 1
lb = np.array([0.0, -h/2])
ub = np.array([t, h/2])

#create the points
N_dirichlet_points = 200
N_collocation_points = 20000

T_dirichlet_down =
torch.from_numpy(np.expand_dims(np.linspace(0,t,N_dirichlet_points),
axis=1)).float()
X_dirichlet_down = torch.from_numpy(np.zeros_like(T_dirichlet_down)-
1).float()
u_dirichlet_down=
torch.from_numpy(np.zeros_like(T_dirichlet_down)).float()

T_dirichlet_top =
torch.from_numpy(np.expand_dims(np.linspace(0,t,N_dirichlet_points),
axis=1)).float()
X_dirichlet_top =
torch.from_numpy(np.zeros_like(T_dirichlet_top)+1).float()
u_dirichlet_top=
torch.from_numpy(np.zeros_like(T_dirichlet_top)).float()

X_dirichlet_left = torch.from_numpy(np.expand_dims(np.linspace(-1,-
1+h,N_dirichlet_points), axis=1)).float()
T_dirichlet_left =
torch.from_numpy(np.zeros_like(X_dirichlet_left)).float()
temp=-np.sin(np.pi*X_dirichlet_left)
u_dirichlet_left = temp

T_nodal = np.concatenate((T_dirichlet_down,
T_dirichlet_top,T_dirichlet_left), 0)
X_nodal = np.concatenate((X_dirichlet_down,
X_dirichlet_top,X_dirichlet_left), 0)
TX_nodal = np.concatenate((T_nodal,X_nodal), 1)
```

```

u_nodal = np.concatenate((u_dirichlet_down,
u_dirichlet_top,u_dirichlet_left), 0)

# initial conditions
T_dirichlet = torch.from_numpy(TX_nodal[:,0:1]).float()
X_dirichlet = torch.from_numpy(TX_nodal[:,1:2]).float()
u_dirichlet = torch.from_numpy(u_nodal).float()

# these are my collocation points
collocation = lb + (ub - lb) * lhs(2, N_collocation_points)
T_collocation = torch.from_numpy(collocation[:,0:1]).float()
X_collocation = torch.from_numpy(collocation[:,1:2]).float()

#create the NN class
class MyNetwork(torch.nn.Module):
    def __init__(self):
        # call constructor from superclass
        super().__init__()

        # define network layers
        self.fc1 = torch.nn.Linear(2, 20)
        self.fc2 = torch.nn.Linear(20, 20)
        self.fc3 = torch.nn.Linear(20, 20)
        self.fc4 = torch.nn.Linear(20, 20)
        self.fc5 = torch.nn.Linear(20, 20)
        self.fc6 = torch.nn.Linear(20, 20)
        self.fc7 = torch.nn.Linear(20, 20)
        self.fc8 = torch.nn.Linear(20, 1)

    def forward(self, t, x):
        # define forward pass
        t = t
        x = x
        u = torch.tanh(self.fc1(torch.cat([t,x], dim=1)))
        u = torch.tanh(self.fc2(u))
        u = torch.tanh(self.fc3(u))
        u = torch.tanh(self.fc4(u))
        u = torch.tanh(self.fc5(u))
        u = torch.tanh(self.fc6(u))
        u = torch.tanh(self.fc7(u))
        u = torch.tanh(self.fc8(u))
        return u

#Instantiate the model
model = MyNetwork()

def dirichlet_boundary_conditions_loss(model, T_dirichlet,
X_dirichlet):
    U_dirichlet_pred = model(T_dirichlet,X_dirichlet)

    return U_dirichlet_pred

#Create the function to compute the differential equation loss
def dif_loss(model, T_collocation, X_collocation):
    T = T_collocation.clone().detach().requires_grad_(True)

```

```

X = X_collocation.clone().detach().requires_grad_(True)
u = model(T,X)
du_dt = grad(u, T, torch.ones_like(T), create_graph = True,
retain_graph=True)[0]
du_dx = grad(u, X, torch.ones_like(X), create_graph=True,
retain_graph=True)[0]
du2_dx2 = grad(du_dx, X, torch.ones_like(X), create_graph=True,
retain_graph=True)[0]
pi = np.pi
dif_equation = du_dt + u*du_dx-(0.01/pi)*du2_dx2
return dif_equation

#Create the optimizer and the loss function
def loss_fn(model, u_dirichlet_pred, T_collocation, X_collocation):
f = dif_loss(model, T_collocation, X_collocation)
error_diffEq = torch.nn.functional.mse_loss(f, 0 * f)
errorDirichlet = torch.nn.functional.mse_loss(u_dirichlet_pred,
u_dirichlet)
return errorDirichlet + error_diffEq

#optimizer = torch.optim.Adam(model.parameters(), lr = 0.02)
#optimizer=torch.optim.Adadelta(model.parameters(), lr=0.05, rho=0.9,
eps=1e-06, weight_decay=0)
#optimizer=torch.optim.Adagrad(model.parameters(), lr=0.05, lr_decay=0,
weight_decay=0, initial_accumulator_value=0, eps=1e-10)
optimizer = torch.optim.SGD(model.parameters(), lr=0.2, momentum=0.5)

#train the model
epochs = 15000
for epoch in range(epochs):
# Forward pass: compute predicted u_pred by passing X_u to the
model.
U_dirichlet_pred =
dirichlet_boundary_conditions_loss(model,T_dirichlet, X_dirichlet)
loss = loss_fn(model, U_dirichlet_pred, T_collocation,
X_collocation)
if epoch % 100 == 99:
print('epoch', epoch + 1, ", loss: ", loss.item())

if epoch % 3000 == 2999:
for param_group in optimizer.param_groups:
param_group['lr'] = param_group['lr']/2

# Before the backward pass, use the optimizer object to zero all of
the
# gradients for the variables it will update (which are the
learnable
# weights of the model). This is because by default, gradients are
# accumulated in buffers( i.e, not overwritten) whenever
.backward()
# is called. Checkout docs of torch.autograd.backward for more
details.
optimizer.zero_grad()

# Backward pass: compute gradient of the loss with respect to model

```

```

# parameters
loss.backward()

# Calling the step function on an Optimizer makes an update to its
# parameters
optimizer.step()

#Results
import scipy.io
data = scipy.io.loadmat("burgers_shock.mat")

t = data['t'].flatten()[:, None]
x = data['x'].flatten()[:, None]
Exact = np.real(data['usol']).T

X, T = np.meshgrid(x, t)

X_star = np.hstack((T.flatten()[:, None], X.flatten()[:, None]))
u_star = Exact.flatten()[:, None]

u_exact = u_star
u_PINN = model(torch.from_numpy(X_star[:, 0:1]).float(),
torch.from_numpy(X_star[:, 1:2]).float())
error = torch.norm(torch.from_numpy(u_star).float() - u_PINN) /
torch.norm(torch.from_numpy(u_star).float()) / len(u_star) #L2 error
relative error
print('L2 norm relative error overall: ', error.detach().numpy()*100,
'%')

##### PLOTTING part#####
## Exact solution

fig1=plt.figure(1)
ax1 = fig1.gca(projection='3d')
surf1 = ax1.plot_surface(T, X, Exact, cmap=cm.coolwarm, linewidth=1,
antialiased=False)
ax1.zaxis.set_major_locator(LinearLocator(10))
ax1.zaxis.set_major_formatter(FormatStrFormatter('%0.02f'))
ax1.set_title('Exact solution \n u(t,x)')
ax1.set_xlabel('t')
ax1.set_ylabel('x')
ax1.set_zlabel('u(t,x)')
# Add a color bar which maps values to colors.
fig1.colorbar(surf1, shrink=0.5, aspect=5)

fig2 = plt.figure(2)
ax2 = fig2.gca()
ax2.set_xlim(t.min(), t.max())
ax2.set_ylim(x.min(), x.max())
cfset = ax2.contourf(T, X, Exact, cmap='coolwarm')
ax2.imshow(np.rot90(Exact), cmap='coolwarm', extent=[t.min(), t.max(),
x.min(), x.max()])
cset2 = ax2.contour(T, X, Exact, colors='k')
ax2.clabel(cset2, inline=1, fontsize=10)

```



```

ax2.set_xlabel('t')
ax2.set_ylabel('x')
plt.title('Exact solution \n u(t,x)')

## PINN solution
u_PINN=u_PINN.detach().numpy()
U_val = griddata(X_star, u_PINN, (T, X), method='cubic')
U_val=U_val[:, :, 0]
fig3=plt.figure(3)
ax3 = fig3.gca(projection='3d')
surf3 = ax3.plot_surface(T, X, U_val, cmap=cm.coolwarm, linewidth=1,
antialiased=False)
ax3.set_xlabel('t')
ax3.set_ylabel('x')
ax3.set_zlabel('u(t,x)')
plt.title('PINN solution \n u(t,x)')

# Add a color bar which maps values to colors.
fig3.colorbar(surf3, shrink=0.5, aspect=5)

fig4 = plt.figure(4)
ax4 = fig4.gca()
ax4.set_xlim(t.min(), t.max())
ax4.set_ylim(x.min(), x.max())
cfset4 = ax4.contourf(T, X, U_val, cmap='coolwarm')
ax4.imshow(np.rot90(U_val), cmap='coolwarm', extent=[t.min(), t.max(),
x.min(), x.max()])
cset4 = ax4.contour(T, X, U_val, colors='k')
ax4.clabel(cset4, inline=1, fontsize=10)
ax4.set_xlabel('t')
ax4.set_ylabel('x')
plt.title('PINN solution \n u(t,x)')

# compare exact and PINN in constant time
# t=0.25,0.50,0.75
X_t_025 = X_star[6400:6656, 1:2]
X_t_050 = X_star[12800:13056, 1:2]
X_t_075 = X_star[19200:19456, 1:2]
u_exact_t_025 = u_exact[6400:6656, 0:1]
u_exact_t_050 = u_exact[12800:13056, 0:1]
u_exact_t_075 = u_exact[19200:19456, 0:1]
u_PINN_t_025 = u_PINN[6400:6656, 0:1]
u_PINN_t_050 = u_PINN[12800:13056, 0:1]
u_PINN_t_075 = u_PINN[19200:19456, 0:1]
fig5 = plt.figure(5)
fig5, ax5 = plt.subplots()
ax5.plot(X_t_025, u_exact_t_025, 'b', label='Exact', linewidth=3.0)
ax5.plot(X_t_025, u_PINN_t_025, 'r', linestyle='--', label='PINN',
linewidth=3.0)
ax5.clabel(cset2, inline=1, fontsize=10)
ax5.set_xlabel('x')
ax5.set_ylabel('u (t,x)')
plt.title('t = 0.25')

```

```

ax5.legend()

fig6 = plt.figure(6)
fig6, ax6 = plt.subplots()
ax6.plot(X_t_050, u_exact_t_050, 'b', label='Exact', linewidth=3.0)
ax6.plot(X_t_050, u_PINN_t_050, 'r', linestyle='--', label='PINN',
linewidth=3.0)
ax6.clabel(cset2, inline=1, fontsize=10)
ax6.set_xlabel('x')
ax6.set_ylabel('u (t,x)')
plt.title('t = 0.50')
ax6.legend()

fig7 = plt.figure(7)
fig7, ax7 = plt.subplots()
ax7.plot(X_t_075, u_exact_t_075, 'b', label='Exact', linewidth=3.0)
ax7.plot(X_t_075, u_PINN_t_075, 'r', linestyle='--', label='PINN',
linewidth=3.0)
ax7.clabel(cset2, inline=1, fontsize=10)
ax7.set_xlabel('x')
ax7.set_ylabel('u (t,x)')
plt.title('t = 0.75')
ax7.legend()
plt.show()

```

Κώδικας PINN εύρεσης παραμέτρου εξίσωσης Burgers

```
import torch
import numpy as np
from pyDOE import lhs
from torch.autograd import grad
cuda = torch.device('cuda')
from scipy.interpolate import griddata
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

np.random.seed(1234)
torch.manual_seed(1234)
# Problem parameters
h = 2
t = 1
lb = np.array([0.0, -h/2])
ub = np.array([t, h/2])

#create the points
N_dirichlet_points = 2
N_collocation_points = 5000
IntegerForPointDivision = 10

T_dirichlet_down =
torch.from_numpy(np.expand_dims(np.linspace(0,t,N_dirichlet_points),
axis=1)).float()
X_dirichlet_down = torch.from_numpy(np.zeros_like(T_dirichlet_down)-
1).float()
u_dirichlet_down=
torch.from_numpy(np.zeros_like(T_dirichlet_down)).float()

T_dirichlet_top =
torch.from_numpy(np.expand_dims(np.linspace(0,t,N_dirichlet_points),
axis=1)).float()
X_dirichlet_top =
torch.from_numpy(np.zeros_like(T_dirichlet_top)+1).float()
u_dirichlet_top=
torch.from_numpy(np.zeros_like(T_dirichlet_top)).float()

X_dirichlet_left = torch.from_numpy(np.expand_dims(np.linspace(-1,-
1+h,N_dirichlet_points), axis=1)).float()
T_dirichlet_left =
torch.from_numpy(np.zeros_like(X_dirichlet_left)).float()
temp=-np.sin(np.pi*X_dirichlet_left)
u_dirichlet_left = temp

import scipy.io
data = scipy.io.loadmat("burgers_shock.mat")
t = data['t'].flatten()[:, None]
x = data['x'].flatten()[:, None]
Exact = np.real(data['usol']).T
```

```

X, T = np.meshgrid(x, t)
u_star = Exact.flatten() [1:25600:IntegerForPointDivision, None] #[:,
None]
knownTimes=T.flatten() [1:25600:IntegerForPointDivision, None]
knownLocations=X.flatten() [1:25600:IntegerForPointDivision, None]
T_nodal = np.concatenate((T_dirichlet_down,
T_dirichlet_top,T_dirichlet_left,knownTimes), 0)
X_nodal = np.concatenate((X_dirichlet_down,
X_dirichlet_top,X_dirichlet_left,knownLocations), 0)
TX_nodal = np.concatenate((T_nodal,X_nodal), 1)
u_nodal = np.concatenate((u_dirichlet_down,
u_dirichlet_top,u_dirichlet_left,u_star), 0)

# initial conditions
T_dirichlet = torch.from_numpy(TX_nodal[:,0:1]).float()
X_dirichlet = torch.from_numpy(TX_nodal[:,1:2]).float()
u_dirichlet = torch.from_numpy(u_nodal).float()

# these are my collocation points
collocation = lb + (ub - lb) * lhs(2, N_collocation_points)
T_collocation = torch.from_numpy(collocation[:,0:1]).float()
##T_dirichlet
X_collocation = torch.from_numpy(collocation[:,1:2]).float() ##
X_dirichlet

#create the NN class
class MyNetwork(torch.nn.Module):
    def __init__(self):
        # call constructor from superclass
        super().__init__()
        self.Viscosity = torch.nn.Parameter(torch.tensor(0.5,
requires_grad=True))
        # define network layers
        self.fc1 = torch.nn.Linear(2, 20)
        self.fc2 = torch.nn.Linear(20, 20)
        self.fc3 = torch.nn.Linear(20, 20)
        self.fc4 = torch.nn.Linear(20, 20)
        self.fc5 = torch.nn.Linear(20, 20)
        self.fc6 = torch.nn.Linear(20, 20)
        self.fc7 = torch.nn.Linear(20, 20)
        self.fc8 = torch.nn.Linear(20, 1)

    def forward(self, t, x):
        # define forward pass
        t = t
        x = x
        u = torch.tanh(self.fc1(torch.cat([t,x], dim=1)))
        u = torch.tanh(self.fc2(u))
        u = torch.tanh(self.fc3(u))
        u = torch.tanh(self.fc4(u))
        u = torch.tanh(self.fc5(u))
        u = torch.tanh(self.fc6(u))
        u = torch.tanh(self.fc7(u))
        u = torch.tanh(self.fc8(u))
        return u

```

```

#Instantiate the model
model = MyNetwork()

def dirichlet_boundary_conditions_loss(model, T_dirichlet,
X_dirichlet):
    U_dirichlet_pred = model(T_dirichlet,X_dirichlet)

    return U_dirichlet_pred

#Create the function to compute the differential equation loss
def dif_loss(model, T_collocation, X_collocation):
    T = T_collocation.clone().detach().requires_grad_(True)
    X = X_collocation.clone().detach().requires_grad_(True)
    u = model(T,X)
    du_dt = grad(u, T, torch.ones_like(T), create_graph = True,
retain_graph=True)[0]
    du_dx = grad(u, X, torch.ones_like(X), create_graph=True,
retain_graph=True)[0]
    du2_dx2 = grad(du_dx, X, torch.ones_like(X), create_graph=True,
retain_graph=True)[0]
    pi = np.pi
    dif_equation = du_dt + u*du_dx-model.Viscosity *du2_dx2
    return dif_equation

#Create the optimizer and the loss function
def loss_fn(model, u_dirichlet_pred, T_collocation, X_collocation):
    f = dif_loss(model, T_collocation, X_collocation)
    error_diffEq = torch.nn.functional.mse_loss(f, 0 * f)
    errorDirichlet = torch.nn.functional.mse_loss(u_dirichlet_pred,
u_dirichlet)
    return errorDirichlet + error_diffEq

#optimizer = torch.optim.SGD(model.parameters(), lr=0.02, momentum=0.5)
optimizer = torch.optim.Adam(model.parameters(), lr = 0.02)

#train the model
epochs = 50000
param_array = np.empty([epochs, 0])
epochs_array = np.empty([epochs, 0])
for epoch in range(epochs):
    param_array = np.append(param_array, [ model.Viscosity.item()])
    epochs_array = np.append(epochs_array, [epoch])
    # Forward pass: compute predicted u_pred by passing X_u to the
model.
    U_dirichlet_pred =
dirichlet_boundary_conditions_loss(model,T_dirichlet, X_dirichlet)
    loss = loss_fn(model, U_dirichlet_pred, T_collocation,
X_collocation)
    if epoch % 100 == 99:
        print('epoch', epoch + 1, ", loss: ", loss.item())

    if epoch % 5000 == 4999:
        for param_group in optimizer.param_groups:
            param_group['lr'] = param_group['lr']/2
            print('parameter', model.Viscosity)

```

```
optimizer.zero_grad()

loss.backward()

optimizer.step()

# Create graph of how model parameter changes during epochs
param_true_value = np.full((epochs, 1), [0.01/np.pi])
fig1 = plt.figure(1)
fig1, ax1 = plt.subplots()
ax1.plot(epochs_array, param_array, 'b', label='Neural Network')
ax1.plot(epochs_array, param_true_value, 'r', label='v = 0.01/π')
ax1.set_xlabel('epochs')
ax1.set_ylabel('kinematic viscosity v')
plt.title('Training parameter v of Burgers equation')
ax1.legend()
plt.show()
```

Κώδικας PINN λύσης εξίσωσης κινηματικού κύματος

```
import numpy as np
import torch
from matplotlib import pyplot as plt
from pyDOE import lhs
from torch.autograd import grad
import os
import scipy.io
from scipy.interpolate import griddata
from scipy.interpolate import interp1d
cuda = torch.device('cuda')

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

np.random.seed(1234)
torch.manual_seed(1234)
# Problem parameters
a = 3.50
b = 0.60

#create the points
N_dirichlet_points = 200
N_collocation_points = 5000
data = scipy.io.loadmat("Data_seconds.mat")

data_t = data['tt'].flatten()[:, None]
data_x = data['x'].flatten()[:, None]
data_t = data_t / 7200 # normalize the input so that it falls in the
range between [0,1]
data_x = data_x / 24000 # normalize the input so that it falls in the
range between [0,1]
Exact = data['Exact']
Exact_u = np.real(Exact)
max_u = np.amax(Exact_u)
min_u = np.min(Exact_u)
Exact_u = (Exact_u - min_u) / (max_u - min_u) # normalize the output
so that it falls in the range between [0,1]

XX, TT = np.meshgrid(data_x, data_t)

# X_star, u_star are only required for the final error check wrt the FD
solution
X_star = np.hstack((XX.flatten()[:, None], TT.flatten()[:, None]))
u_star = Exact_u.flatten()[:, None]

lb = np.array([0.0, 0.0])
ub = np.array([1.0, 1.0])

# these are my known nodal values corresponding to the ICs (use some
interpolation to take more points)
NoOfPoints = N_dirichlet_points
```

```

x0_interp = np.linspace(data_x.min(), data_x.max(), NoOfPoints)
x0_interp = np.reshape(x0_interp, (NoOfPoints, 1))
x0 = np.concatenate((x0_interp, 0 * x0_interp), 1)
f = interp1d(np.ravel(data_x), Exact_u[0:1, :])
u0 = f(x0_interp[:, 0:1])
u0 = u0[0, :, 0:1]

# these are my known nodal values corresponding to the boundary (use
some interpolation to take more points)
NoOfPoints2 = N_dirichlet_points
x_b_interp = np.linspace(data_t.min(), data_t.max(), NoOfPoints2)
x_b_interp = np.reshape(x_b_interp, (NoOfPoints2, 1))
x_b = np.concatenate((0 * x_b_interp, x_b_interp), 1)
f2 = interp1d(np.ravel(data_t), Exact_u[:, 0:1].T)
u_b = f2(x_b[:, 1:2])
u_b = u_b[0, :, 0:1]

# these are the internal known values (we will use the X_star-u_star
data pairs)
xt_known = X_star
u_known = u_star

# let's put together all our nodal values
# case with interior points
#xt_nodal = np.concatenate((x0, x_b, xt_known), 0)
#u_nodal = np.concatenate((u0, u_b, u_known), 0)
# case without interior points
xt_nodal = np.concatenate((x0, x_b), 0)
u_nodal = np.concatenate((u0, u_b), 0)

# initial conditions
X_dirichlet = torch.from_numpy(xt_nodal[:, 0:1]).float()
T_dirichlet = torch.from_numpy(xt_nodal[:, 1:2]).float()
u_dirichlet = torch.from_numpy(u_nodal).float()

# these are my collocation points
collocation = lb + (ub - lb) * lhs(2, N_collocation_points)
X_collocation = torch.from_numpy(collocation[:, 0:1]).float()
T_collocation = torch.from_numpy(collocation[:, 1:2]).float()

#Results in 2000 collocation points
collocation_2 = lb + (ub - lb) * lhs(2, 2000)

#create the NN class
class MyNetwork(torch.nn.Module):
    def __init__(self):
        # call constructor from superclass
        super().__init__()

        # define network layers
        self.fc1 = torch.nn.Linear(2, 20)
        self.fc2 = torch.nn.Linear(20, 20)
        self.fc3 = torch.nn.Linear(20, 20)
        self.fc4 = torch.nn.Linear(20, 20)
        self.fc5 = torch.nn.Linear(20, 20)
        self.fc6 = torch.nn.Linear(20, 20)
        self.fc7 = torch.nn.Linear(20, 20)

```



```

        self.fc8 = torch.nn.Linear(20, 1)

    def forward(self, x, t):
        # define forward pass
        x = x
        t = t
        u = torch.tanh(self.fc1(torch.cat([x,t], dim=1)))
        u = torch.tanh(self.fc2(u))
        u = torch.tanh(self.fc3(u))
        u = torch.tanh(self.fc4(u))
        u = torch.tanh(self.fc5(u))
        u = torch.tanh(self.fc6(u))
        u = torch.tanh(self.fc7(u))
        u = torch.tanh(self.fc8(u))
        return u

#Instantiate the model
model = MyNetwork()

def dirichlet_boundary_conditions_loss(model, X_dirichlet,
T_dirichlet):
    U_dirichlet_pred = model(X_dirichlet,T_dirichlet)

    return U_dirichlet_pred

#Create the function to compute the differential equation loss
def dif_loss(model, X_collocation, T_collocation):
    X = X_collocation.clone().detach().requires_grad_(True)
    T = T_collocation.clone().detach().requires_grad_(True)
    u = model(X,T)
    du_dx = grad(u, X, torch.ones_like(X), create_graph=True,
retain_graph=True)[0]
    du_dt = grad(u, T, torch.ones_like(T), create_graph=True,
retain_graph=True)[0]

    dif_equation = du_dx + 3.333*a*b*abs(4000*u+2000)**(b-1)*du_dt
    return dif_equation

#Create the optimizer and the loss function
def loss_fn(model, u_dirichlet_pred, X_collocation, T_collocation):
    f = dif_loss(model, X_collocation, T_collocation)
    errorDirichlet =
torch.nn.functional.mse_loss(u_dirichlet_pred,u_dirichlet)
    error_diffEq = torch.nn.functional.mse_loss(f,0*f)
    return errorDirichlet + error_diffEq

optimizer = torch.optim.Adam(model.parameters(), lr = 5e-4)
#optimizer=torch.optim.Adadelta(model.parameters(), lr=1.0, rho=0.9,
eps=1e-06, weight_decay=0)
#torch.optim.Adagrad(params, lr=0.01, lr_decay=0, weight_decay=0,
initial_accumulator_value=0, eps=1e-10)
#optimizer = torch.optim.SGD(model.parameters(), lr=0.001,
momentum=0.5)

#train the model

```

```

epochs = 15000
for epoch in range(epochs):
    # Forward pass: compute predicted u_pred by passing X_u to the
    model.
    U_dirichlet_pred =
dirichlet_boundary_conditions_loss(model,X_dirichlet, T_dirichlet)
    loss = loss_fn(model, U_dirichlet_pred, X_collocation,
T_collocation)
    if epoch % 100 == 99:
        print('epoch', epoch + 1, ", loss: ", loss.item())

    if epoch % 5000 == 4999:
        for param_group in optimizer.param_groups:
            param_group['lr'] = param_group['lr']/2
        # Before the backward pass, use the optimizer object to zero all of
the
        # gradients for the variables it will update (which are the
learnable
        # weights of the model). This is because by default, gradients are
        # accumulated in buffers( i.e, not overwritten) whenever
.backward()
        # is called. Checkout docs of torch.autograd.backward for more
details.
        optimizer.zero_grad()

        # Backward pass: compute gradient of the loss with respect to model
# parameters
        loss.backward()

        # Calling the step function on an Optimizer makes an update to its
# parameters
        optimizer.step()

#Results
u_exact = u_star
u_PINN = model(torch.from_numpy(X_star[:,0:1]).float(),
torch.from_numpy(X_star[:,1:2]).float())
u_PINN_2 = model(torch.from_numpy(collocation_2[:,0:1]).float(),
torch.from_numpy(collocation_2[:,1:2]).float())
u_PINN_2 =u_PINN_2.detach().numpy();
error = torch.norm(torch.from_numpy(u_star).float() - u_PINN) /
torch.norm(torch.from_numpy(u_star).float()) #L2 error relative
error
print('L2 norm relative error overall: ', error.detach().numpy()*100,
'%')

# %%PLOTTING part
X_axis=np.reshape(X_star[:,0:1], (41,9))
X_axis=X_axis.T
X_axis=X_axis*24000 #scale the space dimension back to the physical
domain
Y_axis=np.reshape(X_star[:,1:2], (41,9))
Y_axis=Y_axis.T
Y_axis=Y_axis*7200 #scale the time dimension back to the physical
domain

```

```
u_PINN=u_PINN.detach().numpy();
U_val=np.reshape(u_PINN, (41, 9))
U_val=U_val.T
U_val=U_val*(max_u-min_u)+min_u
fig=plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X_axis, Y_axis, U_val, cmap=cm.coolwarm,
linewidth=1, antialiased=False)
ax.set_zlim(1500, 6000)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```

Κώδικας PINN εύρεσης παραμέτρου εξίσωσης κινηματικού κύματος

```
import numpy as np
import torch
from matplotlib import pyplot as plt
from pyDOE import lhs
from torch.autograd import grad
import os
import scipy.io
from scipy.interpolate import griddata
from scipy.interpolate import interp1d
cuda = torch.device('cuda')

import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

np.random.seed(1234)
torch.manual_seed(1234)
# Problem parameters
# We want to find parameter a
b = 0.60

#create the points
N_dirichlet_points = 2
N_collocation_points = 5000
data = scipy.io.loadmat("Data_ParamIdent.mat")

data_t = data['tt'].flatten()[:, None]
data_x = data['x'].flatten()[:, None]
data_t = data_t / 7200 # normalize the input so that it falls in the
range between [0,1]
data_x = data_x / 24000 # normalize the input so that it falls in the
range between [0,1]
Exact = data['Exact']
Exact_u = np.real(Exact)
max_u = np.amax(Exact_u)
min_u = np.min(Exact_u)
Exact_u = (Exact_u - min_u) / (max_u - min_u) # normalize the output
so that it falls in the range between [0,1]

XX, TT = np.meshgrid(data_x, data_t)

# X_star, u_star are only required for the final error check wrt the FD
solution
X_star = np.hstack((XX.flatten()[:, None], TT.flatten()[:, None]))
u_star = Exact_u.flatten()[:, None]

lb = np.array([0.0, 0.0])
ub = np.array([1.0, 1.0])

# these are my known nodal values corresponding to the ICs (use some
interpolation to take more points)
```

```

NoOfPoints = N_dirichlet_points
x0_interp = np.linspace(data_x.min(), data_x.max(), NoOfPoints)
x0_interp = np.reshape(x0_interp, (NoOfPoints, 1))
x0 = np.concatenate((x0_interp, 0 * x0_interp), 1)
f = interp1d(np.ravel(data_x), Exact_u[0:1, :])
u0 = f(x0_interp[:, 0:1])
u0 = u0[0, :, 0:1]

# these are my known nodal values corresponding to the boundary (use
some interpolation to take more points)
NoOfPoints2 = N_dirichlet_points
x_b_interp = np.linspace(data_t.min(), data_t.max(), NoOfPoints2)
x_b_interp = np.reshape(x_b_interp, (NoOfPoints2, 1))
x_b = np.concatenate((0 * x_b_interp, x_b_interp), 1)
f2 = interp1d(np.ravel(data_t), Exact_u[:, 0:1].T)
u_b = f2(x_b[:, 1:2])
u_b = u_b[0, :, 0:1]

# these are the internal known values (we will use the X_star-u_star
data pairs)
xt_known = X_star
u_known = u_star

# let's put together all our nodal values
# case with interior points
#xt_nodal = np.concatenate((x0, x_b, xt_known), 0)
#u_nodal = np.concatenate((u0, u_b, u_known), 0)
# case without interior points
xt_nodal = np.concatenate((x0, x_b, xt_known), 0)
u_nodal = np.concatenate((u0, u_b, u_known), 0)

# initial conditions
X_dirichlet = torch.from_numpy(xt_nodal[:, 0:1]).float()
T_dirichlet = torch.from_numpy(xt_nodal[:, 1:2]).float()
u_dirichlet = torch.from_numpy(u_nodal).float()

# these are my collocation points
collocation = lb + (ub - lb) * lhs(2, N_collocation_points)
X_collocation = torch.from_numpy(collocation[:, 0:1]).float()
T_collocation = torch.from_numpy(collocation[:, 1:2]).float()

#create the NN class
class MyNetwork(torch.nn.Module):
    def __init__(self):
        # call constructor from superclass
        super().__init__()
        self.ParameterA = torch.nn.Parameter(torch.tensor(10.0,
requires_grad=True))

        # define network layers
        self.fc1 = torch.nn.Linear(2, 20)
        self.fc2 = torch.nn.Linear(20, 20)
        self.fc3 = torch.nn.Linear(20, 20)
        self.fc4 = torch.nn.Linear(20, 20)
        self.fc5 = torch.nn.Linear(20, 20)
        self.fc6 = torch.nn.Linear(20, 20)

```

```

self.fc7 = torch.nn.Linear(20, 20)
self.fc8 = torch.nn.Linear(20, 1)

def forward(self, x, t):
    # define forward pass
    x = x
    t = t
    u = torch.tanh(self.fc1(torch.cat([x,t], dim=1)))
    u = torch.tanh(self.fc2(u))
    u = torch.tanh(self.fc3(u))
    u = torch.tanh(self.fc4(u))
    u = torch.tanh(self.fc5(u))
    u = torch.tanh(self.fc6(u))
    u = torch.tanh(self.fc7(u))
    u = torch.tanh(self.fc8(u))
    return u

#Instantiate the model
model = MyNetwork()

def dirichlet_boundary_conditions_loss(model, X_dirichlet,
T_dirichlet):
    U_dirichlet_pred = model(X_dirichlet,T_dirichlet)

    return U_dirichlet_pred

#Create the function to compute the differential equation loss
def dif_loss(model, X_collocation, T_collocation):
    X = X_collocation.clone().detach().requires_grad_(True)
    T = T_collocation.clone().detach().requires_grad_(True)
    u = model(X,T)
    du_dx = grad(u, X, torch.ones_like(X), create_graph=True,
retain_graph=True)[0]
    du_dt = grad(u, T, torch.ones_like(T), create_graph=True,
retain_graph=True)[0]

    dif_equation = du_dx +
3.333*model.ParameterA*b*abs(4000*u+2000)**(b-1)*du_dt
    return dif_equation

#Create the optimizer and the loss function
def loss_fn(model, u_dirichlet_pred, X_collocation, T_collocation):
    f = dif_loss(model, X_collocation, T_collocation)
    errorDirichlet =
torch.nn.functional.mse_loss(u_dirichlet_pred,u_dirichlet)
    error_diffEq = torch.nn.functional.mse_loss(f,0*f)
    return errorDirichlet + error_diffEq

#optimizer = torch.optim.SGD(model.parameters(), lr=0.02, momentum=0.5)
optimizer = torch.optim.Adam(model.parameters(), lr = 0.02)

#train the model
epochs = 50000
param_array = np.empty([epochs, 0])

```

```

epochs_array = np.empty([epochs, 0])
for epoch in range(epochs):
    param_array = np.append(param_array, [model.ParameterA.item()])
    epochs_array = np.append(epochs_array, [epoch])
    # Forward pass: compute predicted u_pred by passing X_u to the
    model.
    U_dirichlet_pred =
dirichlet_boundary_conditions_loss(model, T_dirichlet, X_dirichlet)
    loss = loss_fn(model, U_dirichlet_pred, T_collocation,
X_collocation)
    if epoch % 100 == 99:
        print('epoch', epoch + 1, ", loss: ", loss.item())

    if epoch % 5000 == 4999:
        for param_group in optimizer.param_groups:
            param_group['lr'] = param_group['lr']/2
            print('parameter', model.ParameterA)

optimizer.zero_grad()

loss.backward()

optimizer.step()

# Create graph of how model parameter changes during epochs
param_true_value = np.full((epochs, 1), [3.50])
fig1 = plt.figure(1)
fig1, ax1 = plt.subplots()
ax1.plot(epochs_array, param_array, 'b', label='Neural Network')
ax1.plot(epochs_array, param_true_value, 'r', label='α = 3.50')
ax1.set_xlabel('epochs')
ax1.set_ylabel('parameter α')
ax1.legend()
plt.show()

```