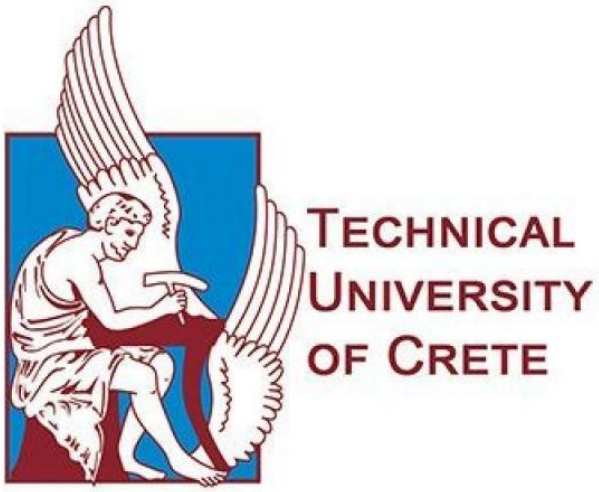


Πολυτεχνείο Κρήτης

Σχολή Μηχανικών Παραγωγής και Διοίκησης

Διπλωματική Εργασία

Τίτλος: Επίλυση ευθέων προβλημάτων στη μηχανική μέσω νευρωνικών δικτύων ενημερωμένων από τη φυσική - Solving direct problems in mechanics using physics informed neural networks (PINNs)

Από τον

Δημήτριο Κατσίκη

Επιβλέπων καθηγητής: Γεώργιος Ε. Σταυρουλάκης

1ο μέλος εξεταστικής επιτροπής: Ιωάννης Μαρινάκης

2ο μέλος εξεταστικής επιτροπής: Γεώργιος Ταϊρίδης

Χανιά, Οκτώβριος 2021

Ευχαριστίες

Από την θέση αυτή, θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Γεώργιο Ε. Σταυρουλάκη για την εμπιστοσύνη που μου έδειξε, αναθέτοντας μου το εν λόγω θέμα, καθώς επίσης τον ευχαριστώ για την συνεργασία μας καθ' όλη την διάρκεια της ενασχόλησης μου με την εργασία. Χάρη σε αυτήν την εργασία εισήλθα στον πανέμορφο κόσμο των Νευρωνικών Δικτύων. Θα ήθελα επίσης να ευχαριστήσω την κα Αλίκη Μουράντοβα για την ευγενική της παρουσία και την προθυμία της να διαθέσει χρόνο, συμμετέχοντας στις συζητήσεις με τον κ. Σταυρουλάκη, προσφέροντας τις γνώσεις και την εμπειρία της. Επίσης, πάνω στο τεχνικό κομμάτι, θα ήθελα να ευχαριστήσω την ξαδέρφη μου Κατερίνα Κώστα για την πολύτιμή βοήθειά της. Ιδιαίτερη θέση σε αυτή τη λίστα κατέχει η οικογένειά μου, οι γονείς μου Νικολέττα, Παναγιώτης και ο αδερφός μου Νίκος, τους οποίους ευχαριστώ για την στήριξή τους στην μέχρι τώρα πορεία μου. Η περίοδος των σπουδών μου αλλά και πιο συγκεκριμένα η περίοδος της εργασίας, σίγουρα δεν θα ήταν το ίδιο ευχάριστή και δημιουργική αν δεν είχα κοντά τους φίλους μου καθώς επίσης την Άρτεμις και τον Πέτρο, τους οποίους ευχαριστώ πολύ θερμά. Τέλος, μέσα από εδώ, θα ήθελα να πώ ένα ευχαριστώ σε έναν άνθρωπο που ήταν μαζί μου από την στιγμή που ανέλαβα το θέμα μέχρι και σήμερα, ο οποίος συνέβαλε ώστε η περίοδος αυτή να είναι ακόμα πιο ευχάριστη. Πρόκειται για την κοπέλα μου Τίνα, στην οποία αφιερώνω την εργασία.

Περίληψη

Στην παρούσα διπλωματική εργασία μελετάται μια σχετικά πρόσφατα αναπτυγμένη μέθοδος επίλυσης συνήθων και μερικών διαφορικών εξισώσεων με χρήση Τεχνητών Νευρωνικών Δικτύων ΤΝΔ (Artificial Neural Networks ANN). Η συγκεκριμένη μέθοδος ονομάζεται Physics Informed Neural Networks (PINNs) και η ανάπτυξή της προέκυψε από τον συνδυασμό αρκετών διαφορετικών επιστημονικών πεδίων όπως η νευρολογία, η πληροφορική, η επιστήμη υπολογιστών, τα εφαρμοσμένα μαθηματικά, η εφαρμοσμένη φυσική καθώς και η υπολογιστική μηχανική. Ουσιαστικά πρόκειται για μια αριθμητική μέθοδο επίλυσης γραμμικών και μη γραμμικών προβλημάτων που συναντώνται στην μετάδοση θερμότητας, στην ρευστομηχανική, στην δυναμική ή στην στατική μηχανική.

Σε όλες τις αριθμητικές μεθόδους, για την εφαρμογή τους σε προβλήματα μεγάλων διαστάσεων τα οποία επιφέρουν μια αξιόλογη προσέγγιση, απαιτείται η παράλληλη εξέλιξη της επιστήμης των υπολογιστών τόσο σε επίπεδο υλικού (Hardware) όσο και σε επίπεδο λογισμικού (Software). Το ίδιο ισχύει και για την εξεταζόμενη μέθοδο, της οποίας η θεωρητική υπόσταση είχε προταθεί σχεδόν πριν από 2 δεκαετίες για πρώτη φορά αλλά η εφαρμογή της συνοδευόμενη με εντυπωσιακά αποτελέσματα σε ανταγωνιστικό χρόνο εκτέλεσης πραγματοποιήθηκε μόλις τα τελευταία χρόνια. Συνέπεια του γεγονότος αυτού είναι ο μεγάλος αριθμός σχετικών δημοσιευμένων εργασιών σε παγκόσμιο επίπεδο από το 2017 και μετά.

Κύριος στόχος της παρούσας εργασίας είναι η ανάπτυξη αλγορίθμων σε γλώσσα προγραμματισμού Python, με σκοπό την εφαρμογή της μεθόδου σε προβλήματα στατικής μηχανικής. Συγκεκριμένα τα προβλήματα που επρόκειτο να λυθούν είναι το πρόβλημα εφελκυσμού σε μονόπακτη ράβδο και το πρόβλημα της δοκού της οποίας ασκούνται δυνάμεις κάμψης για διάφορες συνθήκες στήριξης όπως αμφιέρειστη δοκός, δοκός πρόβολος ή δοκός με πάκτωση στο ένα άκρο και κύλιση στο άλλο. Τα προβλήματα αυτά επιλύθηκαν με το χαμηλότερο σφάλμα που επετεύχθη να είναι της τάξης του 1e-11. Συγχρόνως θα παρουσιαστεί αναλυτικά ολόκληρη η συλλογιστική πορεία ανάπτυξης ή τροποποίησης ήδη υπάρχων αλγορίθμων που αναπαριστούν απλά τεχνητά νευρωνικά δίκτυα και PINNs. Η εργασία περιλαμβάνει αναλυτικά τον τρόπο λειτουργίας των τεχνητών νευρωνικών δικτύων τόσο σε θεωρητικό και μαθηματικό επίπεδο, όσο και σε επίπεδο προγραμματισμού μέσω απλών παραδειγμάτων αλλά και παράθεσης και επεξήγησης τμημάτων του κώδικα που επιλύουν τα προβλήματα της εργασίας.

Η δομή της εργασίας σχεδιάστηκε με τέτοιο τρόπο ώστε να εισάγει ήπια τον αναγνώστη αρχικά στα δομικά υλικά που συνθέτουν το εγχείρημα, να κατανοήσει τις λεπτομέρειες της υλοποίησης και στη συνέχεια να βρεθεί στη θέση να κατανοήσει όσο το δυνατόν πιο εύκολα την σύνδεση. Τα δομικά υλικά με αφηρημένη έννοια είναι: η μεγάλη εικόνα του τρόπου λειτουργίας των βιολογικών νευρωνικών δικτύων, η θεωρία της μηχανικής μάθησης (Machine Learning) καθώς και η θεωρία της στατικής μηχανικής από την οποία θα χρησιμοποιηθούν οι διαφορικές εξισώσεις προς επίλυση. Η σύνδεση προκύπτει από το εξής εύλογο ερώτημα: “ Θα μπορούσε να λυθεί μια διαφορική εξίσωση με χρήση νευρωνικών δικτύων; ”. Έως ότου όμως να απαντηθεί αυτό το ερώτημα αξίζει να γίνει μια ιστορική αναδρομή όλων των βημάτων που πραγματοποιήθηκαν από πολλούς επιστήμονες προκειμένου να φτάσουμε στο σημείο να βλέπουμε εντυπωσιακές προσεγγιστικές λύσεις με την μέθοδο PINNs σε δύσκολα προβλήματα που μέχρι και σήμερα λυνόντουσαν μόνο με τις κλασικές αριθμητικές μεθόδους. Τέλος, αξίζει να σημειωθεί ότι ένας θεμελιώδης στόχος της εργασίας είναι να προσφέρει μια στέρεα και ασφαλή βάση, μια έγκυρη πηγή πληροφόρησης και καθοδήγησης για τον επόμενο ενδιαφερόμενο που θα ασχοληθεί με το θέμα ώστε να λύσει σύντομα τα αρχικά προβλήματα κατανόησης του εγχειρήματος και να προχωρήσει ένα βήμα παρακάτω.

Περιεχόμενα

Κεφάλαιο 1 - Εισαγωγή

Στον φυσικό κόσμο, οι περισσότερες διεργασίες που παρατηρούνται, συνδέονται με τον ρυθμό μεταβολής ενός μεγέθους σε συνάρτηση μιας ή και περισσότερων μεταβλητών. Στις θετικές επιστήμες και στο πεδίο της μηχανικής, οι μεταβλητές αυτές είναι συνήθως οι 3 χωρικές διαστάσεις που αντιλαμβάνεται ο άνθρωπος, δηλαδή οι x, y, z όπως επίσης και ο χρόνος t. Οι διεργασίες αυτές πιθανώς να σχετίζονται με την εξέλιξη ενός τροπικού κυκλώνα, τον ρυθμό μεταβολής της θερμοκρασίας ενός αντικειμένου κατά την μετάδοση θερμότητας, την ροή ενός ρευστού μέσα από αγωγό ή την στατική και δυναμική ανάλυση ενός στερεού που υπόκειται σε δυνάμεις εφελκυσμού και κάμψης. Όλες οι παραπάνω διεργασίες εκφράζονται μαθηματικά μέσω των διαφορικών εξισώσεων (differential equations). Η έννοια και ο όρος διαφορική εξίσωση Δ.Ε. αποτελεί απόρροια της ανάπτυξης του Απειροστικού Λογισμού (Infinitesimal Calculus) ο οποίος είναι Μαθηματικός κλάδος που αναπτύχθηκε για πρώτη φορά μέσα από τα συγγράμματα "Nova Methodus pro Maximis et Minimis" (Leibniz, 1684) και “The Method of fluxions and fluents” (Newton, 1736). Γενικώς στην ιστορία των γεγονότων, υπάρχει σύγχυσή για το ποιος εκ των δύο επιστημόνων εγκαθίδρυσε τον κλάδο, ενώ η διαμάχη μεταξύ τους πάνω στο θέμα αναφέρεται στην βιβλιογραφία ως “Leibniz – Newton calculus controversy”. Παρόλα αυτά σήμερα, έχει επικρατήσει η άποψη ότι και οι δυο μαθηματικοί ανέπτυξαν ανεξάρτητα τον Απειροστικό λογισμό. Μια διαφορική εξίσωση η οποία περιλαμβάνει τον ρυθμό μεταβολής (παράγωγο) μιας συνάρτησης, μόνο ως προς μια μεταβλητή καλείται Συνήθης Διαφορική Εξίσωση Σ.Δ.Ε. (Ordinary Differential Equation O.D.E) ενώ αν περιλαμβάνει τις παράγωγους μιας συνάρτησης ως προς περισσότερες από μια μεταβλητές καλείται Μερική Διαφορική Εξίσωση Μ.Δ.Ε (Partial Differential Equation P.D.E). Μια Σ.Δ.Ε πρώτης τάξης αναφέρεται ως πρόβλημα αρχικών τιμών (Initial value problem) και εκφράζεται ως:

$\dot{y}(t) = f(t,y(t))$ με αρχική συνθήκη (initial or boundary condition) $y(t_0) = y_0$. (1)

Σε αρκετές εφαρμογές της φυσικής, των μαθηματικών και της μηχανικής, οι μελετητές καλούνται να επιλύσουν δύσκολα προβλήματα τα οποία απαιτούν την επίλυση Σ.Δ.Ε, Μ.Δ.Ε ή τις περισσότερες φορές συστημάτων Δ.Ε. Με τον όρο επίλυση αναφέρεται η εύρεση της συνάρτησης y(t) η οποία επαληθεύει την Δ.Ε. Η επίλυση μπορεί να προκύψει από την εύρεση της y συμβολικά - αναλυτικά ως συνάρτηση σταθερών παραμέτρων και γνωστών συναρτήσεων (sin, cos, tan, log κτλ) ή μπορεί να προκύψει αριθμητικά από την χρήση αριθμητικών μεθόδων (Numerical Methods) όπως αποκαλούνται (Butcher, 1999). Ο πρώτος τρόπος επίλυσης που αναφέρθηκε μπορεί να επιφέρει την αναλυτική λύση (exact solution) της Δ.Ε. δηλαδή βρίσκει την έκφραση της συνάρτησης y(t) η οποία ικανοποιεί την εξίσωση. Χαρακτηριστικά παραδείγματα υπολογισμού της αναλυτικής λύσης είναι η ολοκλήρωση των δυο μελών της Δ.Ε, η αλλαγή μεταβλητής (Εξισώσεις Bernoulli) ή η μέθοδος Lagrange. Αντιθέτως, οι αριθμητικές μέθοδοι επίλυσης, προσεγγίζουν της τιμές της y(t) με την συνάρτηση $\hat{y}(t)$, στα σημεία του πεδίου ορισμού, μέσω αλγορίθμων. Οι αλγόριθμοι των αριθμητικών μεθόδων αποτελούνται από βήματα τα οποία επαναλαμβάνονται έως ότου προκύψει σύγκλιση της μεθόδου δηλαδή μέχρι να επιτευχθεί η εύρεση των τιμών της $\tilde{y}(t)$ με το μικρότερο δυνατό σφάλμα απόκλισης από την πραγματική λύση. Χαρακτηριστικά παραδείγματα αριθμητικών μεθόδων είναι η μέθοδος Euler, η μέθοδος του Heun, η μέθοδος Runge – Kutta ή η μέθοδος πεπερασμένων στοιχείων (Finite Element Method FEM) η οποία χρησιμοποιείται για την επίλυση Μερικών Διαφορικών Εξισώσεων.

Για την καλύτερη κατανόηση της έννοιας «αναλυτική λύση» (exact solution) και «προσεγγιστική λύση» (approximate solution) ακολουθεί ένα παράδειγμα μιας απλούστατης διαφορικής εξίσωσης που συναντάται σε πολλά πληθυσμιακά μοντέλα σε πεδία όπως η βιολογία ή τα οικονομικά:

$\frac{dy(x)}{dx}$ = y(x), y(0) = 1 , x ∈ [0, ∞)

- **Αναλυτική λύση (Exact solution):**

  $\int \frac{1}{y(x)} \frac{dy(x)}{dx} dx = \int dx$ =>

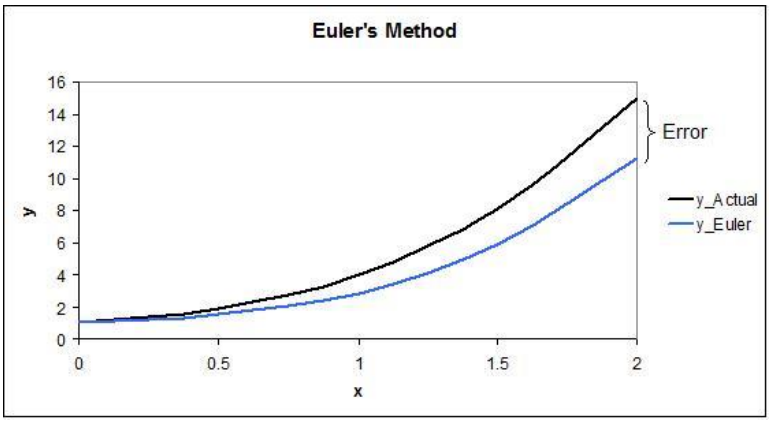
  $y(x) = e^x$ (2)

- **Προσεγγιστική λύση (Approximate Solution) - Χρήση της Αριθμητικής μεθόδου Euler:**

  $y_{n+1} = y_n + h * f(t_n, y_n)$ (3)

  Το h καλείται βήμα του αλγορίθμου.

  Η προσεγγιστική λύση παρατίθεται στο παρακάτω γράφημα:



Εικόνα 1.1 (Ποιοτικό γράφημα παρουσίασης της μεθόδου Euler. Πηγή: Engineering LibreTexts)

Στις περισσότερες εφαρμογές, η αναλυτική επίλυση δεν είναι εφικτή και για αυτόν ακριβώς τον λόγο οι μαθηματικοί που αναφέρθηκαν και πολλοί άλλοι, κατευθύνθηκαν στο να αναπτύξουν αριθμητικές μεθόδους των οποίων οι λύσεις στις περισσότερες περιπτώσεις είναι ιδιαίτερα ικανοποιητικές διότι διαφέρουν κατά ένα ελάχιστο σφάλμα από την πραγματική λύση.

Οι αλγόριθμοι που αναφέρθηκαν, εφαρμόζονταν στην έρευνα και στην παραγωγή, ακόμα και πριν την έλευση των ηλεκτρονικών υπολογιστών, από τους λεγόμενους ανθρώπους υπολογιστές (human computers) οι οποίοι ήταν συνήθως μαθηματικοί, χωρίζονταν σε τμήματα εργασίας και συνεργαζόντουσαν για την εκτέλεση υπολογισμών (Edwards & Harris, 2017). Η ιστορία τους μπορεί να θεωρηθεί ότι αρχίζει οργανωμένα, κατά την μελέτη της τροχιάς του κομήτη του Halley (Halley's Comet) και άλλων αστρονομικών μεγεθών και κορυφώνεται με την συστηματική εργασία τους κατά τους 2 παγκόσμιους πολέμους όπου οι «υπολογιστές» παρήγαγαν συστήματα πλοήγησης και χαρτογράφησης, υπολόγιζαν βαλλιστικές τροχιές βλημάτων, υπολόγιζαν μαθηματικούς πίνακες και εργάζονταν στον σχεδιασμό οπλικών συστημάτων.

Προς τα μέσα του Β Παγκοσμίου Πολέμου άρχισε να διεξάγεται το διάσημο Manhattan Project στο Los Alamos των ΗΠΑ κατά το οποίο πραγματοποιούνταν έρευνα για την κατασκευή των πρώτων πυρηνικών όπλων. Σε αυτό το εγχείρημα συμμετείχαν οι πιο επιτυχημένοι επιστήμονες της εποχής και για τους ατελείωτους υπολογισμούς εργάζονταν οι human computers με την βοήθεια των πρώτων ηλεκτρονικών υπολογιστών της IBM και του ENIAC. Σε αυτό το χρονικό σημείο γίνεται αντιληπτή η ισχύς των ηλεκτρονικών υπολογιστών έναντι των ανθρώπων, με αποτέλεσμα αρκετοί μαθηματικοί και φυσικοί της εποχής να κατευθυνθούν προς την περεταίρω έρευνα και την ανάπτυξη της επιστήμης των ηλεκτρονικών υπολογιστών (computer science). Έγινε εμφανές ότι ένας ορθά προγραμματισμένος ηλεκτρονικός υπολογιστής είναι σημαντικά πιο γρήγορος και ακριβής από οποιοδήποτε άνθρωπο ή ομάδα ανθρώπων στο να επιλύει ένα συγκεκριμένο πρόβλημα. Οι πρώτοι Η/Υ ήταν ογκώδης, κατανάλωναν μεγάλη ηλεκτρική ισχύ και παρουσίαζαν συχνά σφάλματα μετά από πολύωρη χρήση. Παρόλα αυτά τα θεμέλια της τρίτης βιομηχανικής επανάστασης είχαν είδη δομηθεί και μια νέα εποχή για την επιστήμη και για την κοινωνία έχει είδη ξεκινήσει.

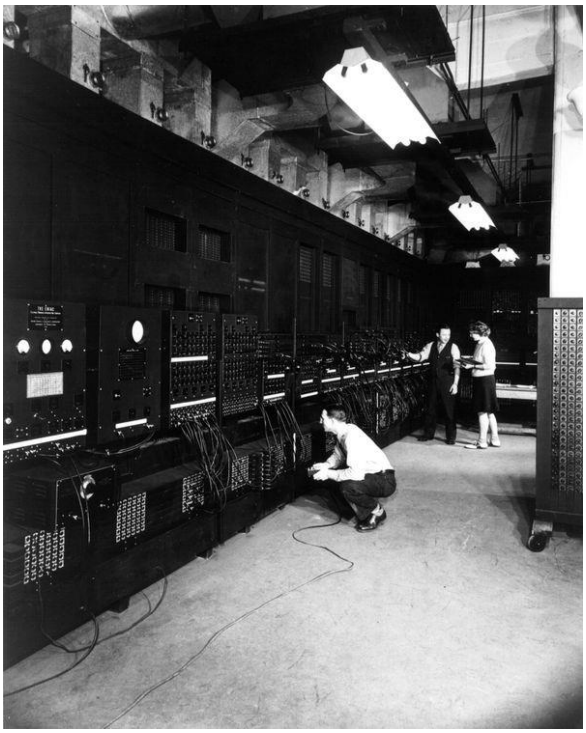
Εικόνα 1.2 (Ομάδα από ανθρώπους υπολογιστές σε εργαστήριο της NASA το έτος 1955 Πηγή: https://www.history.com/news/human-computers-women-at-nasa)

Μελετώντας κανείς τους παραπάνω σταθμούς της ιστορίας, σύντομα αντιλαμβάνεται ότι η πρόοδος σε όλα τα επιστημονικά πεδία είναι αλληλένδετη με την ανάπτυξη της υπολογιστικής ικανότητας του ανθρώπου και των υπολογιστικών εργαλείων που αυτός κατασκευάζει. Το ανθρώπινο είδος αρχικά χρειάστηκε μια βασική θεωρία για να εξηγήσει το σύμπαν γύρο του και έτσι οδηγήθηκε στην ανάπτυξη των μαθηματικών και της φυσικής. Εν συνεχεία, ανέπτυξε την θεωρία των απαραίτητων αριθμητικών μεθόδων που απαιτούνταν για να λύνει τα προβλήματα που προκύπτουν. Το επόμενο βήμα ήταν να κατευθυνθεί στο να επιχειρήσει την κατασκευή μηχανών που να επιταχύνουν και να βελτιώσουν τους υπολογισμούς. Για να συμβεί αυτό χρειάστηκε η κατασκευή Η/Υ και η ανάπτυξη λειτουργικών συστημάτων και γλωσσών προγραμματισμού. Οι μηχανές όμως αυτές παρόλο που ήταν ταχύτερες και πιο ακριβής από τους

ανθρώπους, μπορούν μόνο να εκτελούν μια προς μια συγκεκριμένες εντολές οι οποίες συντάσσονταν από τους προγραμματιστές αρχικά με τη μορφή διάτρητων καρτών και στην συνέχεια με την μορφή πηγαίου κώδικά. Ο ανθρώπινος εγκέφαλος δεν έχει ιδιαίτερα μεγάλη μνήμη, όμως μπορεί να πραγματοποιεί σύνθετες σκέψεις και να λύνει αφηρημένα προβλήματα σε αντίθεση με έναν Η/Υ ο οποίος μπορεί να αποθηκεύει μεγάλες ποσότητες πληροφορίας, να εκτελεί άρτια αριθμητικές πράξεις αλλά όχι κάτι παραπάνω.

Το επόμενο βήμα ήταν η αναζήτηση ενός τρόπου ώστε να συνδυαστεί ο αφηρημένος τρόπος σκέψης του ανθρώπου με την υπολογιστική ισχύ που μπορούν να προσφέρουν οι Η/Υ. Το αποτέλεσμα ήταν η γέννηση του επιστημονικού πεδίου της Τεχνητής Νοημοσύνης ΤΝ (Artificial Intelligence AI) και της Υπολογιστικής Νοημοσύνης ΥΝ (Computational Intelligence CI). Η ΤΝ αποτελεί ένα ολοκληρωμένο επιστημονικό πεδίο καθώς η μεθοδολογία εξέλιξής της υπακούει σαφώς την επιστημονική μέθοδο (Russel & Norvig, 2003). Μόλις το διαβάσει αυτό ένας ανήσυχος αναγνώστης, επόμενο είναι να αναρωτηθεί γιατί η ΤΝ δεν υπάγεται στον κλάδο των μαθηματικών ή άλλων σχετικών πεδίων όπως η θεωρία ελέγχου ή η επιχειρησιακή έρευνα ή η θεωρία αποφάσεων. Η απάντηση είναι ότι η ΤΝ από την γέννηση της, ενστερνίστηκε την ιδέα της αντιγραφής ανθρώπινων λειτουργειών όπως η δημιουργικότητα, η αυτοβελτίωση και η χρήση γλώσσας. Κανένα από τα προαναφερθέντα πεδία δεν ασχολείται με αυτά τα ζητήματα πόσο μάλλον δεν είναι ο αυτοσκοπός τους. Η Τεχνητή Νοημοσύνη είναι σαφώς ένας κλάδος της επιστήμης των υπολογιστών (computer science) και είναι το μόνο πεδίο που προσπαθεί να δημιουργήσει μηχανές – μοντέλα τα οποία να λειτουργούν αυτόνομα μέσα σε πολύπλοκα και μεταβαλλόμενα περιβάλλοντα (Russel & Norvig, 2003). Γίνεται λοιπόν εμφανές ένα μοτίβο όπου ο άνθρωπος συνεχώς αναζητεί νέους τρόπους και τεχνολογίες για να φτάσει ακόμα πιο μακριά στην μελέτη των φαινομένων της φύσης, με σκοπό την επιπλέον ακρίβεια στους υπολογισμούς, άλλα και για να βρει λύση σε μέχρι τώρα άλυτα προβλήματα. Απόρροια του παραπάνω μοτίβου αποτελεί η χρήση της τεχνητής νοημοσύνης και των ΤΝΔ για την επίλυση διαφορικών εξισώσεων.

Ως αναφορά την πιο αναλυτική δομή της εργασίας, στο κεφάλαιο 2 γίνεται μια σύντομη παράθεση των βασικών λειτουργιών του ανθρώπινου εγκεφάλου καθώς από εκεί εμπνεύστηκε η ιδέα των Τεχνητών Νευρωνικών Δικτύων ΤΝΔ, τα οποία είναι στο επίκεντρο της εργασίας. Στο κεφάλαιο 3 θα γίνει μια σύντομη ιστορική αναδρομή της Τεχνητής Νοημοσύνης και πιο συγκεκριμένα των νευρωνικών δικτύων. Το κεφάλαιο δίνει ιδιαίτερη έμφαση στην αναλυτική περιγραφή του τρόπου λειτουργίας των ΤΝΔ και του τρόπου που αυτά μαθαίνουν. Κρίνεται ιδιαίτερης σημασίας κεφάλαιο καθώς παραθέτει τον βασικότερο πυλώνα της μεθόδου Physics Informed Neural Networks (PINNs) που είναι και το κύριο θέμα της εργασίας. Σε αυτό το κεφάλαιο επεξηγείται η δομή των δικτύων και γίνεται μια αναλυτική παρουσίαση της διαδικασίας εκπαίδευσης συνοδευόμενη από ένα χαρακτηριστικό παράδειγμα. Στο κεφάλαιο 4 εισάγεται η μέθοδος PINNs. Αρχικά περιλαμβάνει μια σύντομη αναφορά της πορείας από τις πρώτες δημοσιεύσεις μέχρι και σήμερα, ενώ στην συνέχεια μελετάται ο τρόπος λειτουργίας της μεθόδου και τα εργαλεία που χρησιμοποιούνται για την υλοποίηση. Επιπλέον στο κεφάλαιο αυτό ορίζονται τα προβλήματα προς επίλυση, δηλαδή παρατίθενται οι διαφορικές εξισώσεις στα προβλήματα στατικής μηχανικής, οι οποίες επρόκειτο να λυθούν με την μέθοδο PINNs. Το κεφάλαιο 5 περιλαμβάνει τα αποτελέσματα της υλοποίησης της μεθόδου στα προβλήματα στατικής δηλαδή τα γραφήματα με τις καλύτερες επιδόσεις, συνοδευόμενα από πίνακες με τα συγκεντρωτικά αποτελέσματα. Τέλος στο κεφάλαιο 6 αναφέρονται οι παρατηρήσεις και τα συμπεράσματα που καταγράφηκαν καθ' όλη την διάρκεια εκτέλεσης της εργασίας. Εκεί αναφέρονται οι δυνατότητες της μεθόδου όπως παρατηρήθηκαν από την υλοποίηση και το κατά πόσο είναι εφικτή η ενασχόληση με το θέμα συγκριτικά με το γνωστικό υπόβαθρο του ενδιαφερόμενου. Επιπλέον στο κεφάλαιο αυτό αναφέρονται μελλοντικοί στόχοι για μελέτη και προτάσεις.



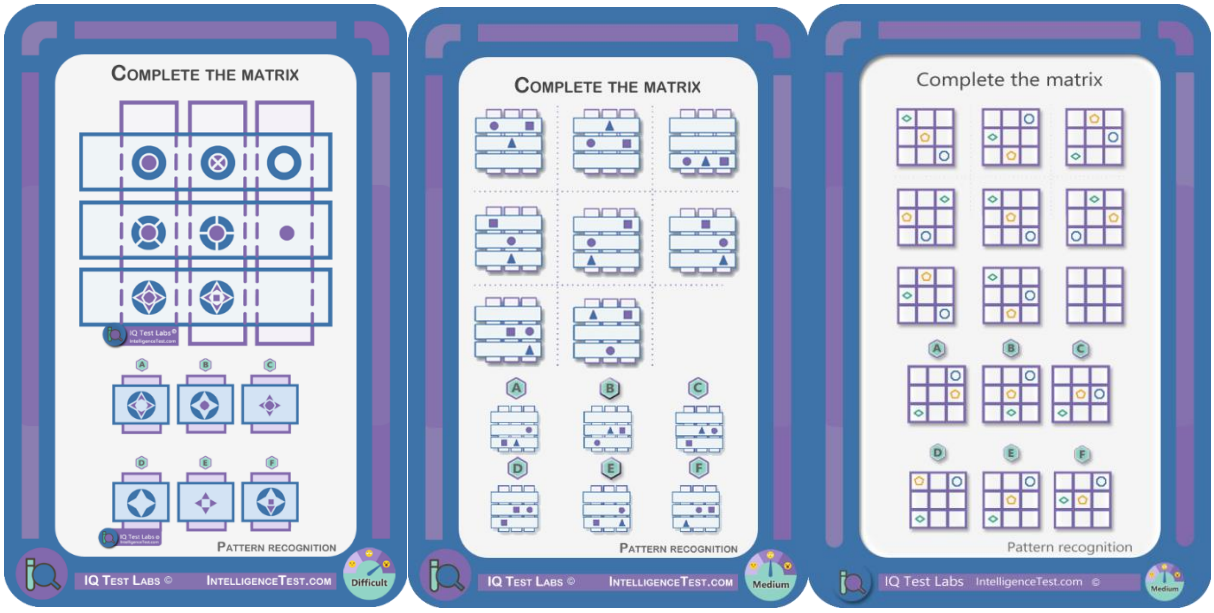
Εικόνα 1.3 (Ο υπολογιστής ENIAC 1946, Pennsylvania University , Πηγή: www.computerhistory.org)

Κεφάλαιο 2 - Βιολογικά Νευρωνικά Δίκτυα

2.1 Μια διαισθητική προσέγγιση

Από την στιγμή της γέννησης του, ένας άνθρωπος δέχεται μια σωρεία από ερεθίσματα όπως οπτικά, ακουστικά ή ερεθίσματα μέσω της αφής ή της όσφρησης. Από τα πρώτα χρόνια της ζωής του ανθρώπου, ο εγκέφαλος χρησιμοποιεί τα αισθητήρια όργανα και πραγματοποιείται μια διαδικασία μάθησης κατά την οποία το μωρό αρχίζει να αντιλαμβάνεται τον χώρο γύρο του, να αναγνωρίζει τα πρόσωπα και τις φωνές των ανθρώπων που βλέπει πιο συχνά, να μαθαίνει να μιλάει την μητρική του γλώσσα και να παρατηρεί τους βασικούς νόμους της φυσικής όπως η βαρύτητα.

Ένα συνηθισμένο πρόβλημα σε ένα κοινό test νοημοσύνης IQ είναι η αναγνώριση προτύπων (pattern recognition) με χαρακτηριστικά παραδείγματα τα ακόλουθα προβλήματα:



Ίσως ένα ακόμη πρόβλημα να είναι το εξής:

1 + 4 = 5

2 + 5 = 12

3 + 6 = 21

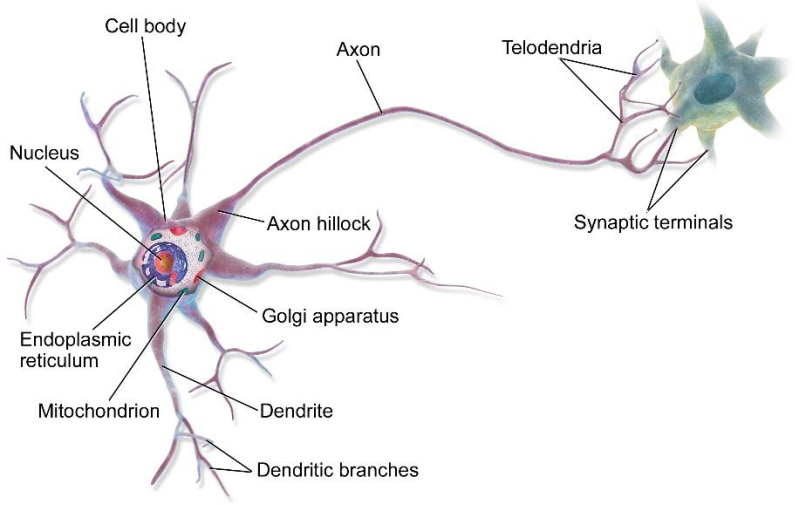
8 + 11 = ?

Με λίγη φαντασία κάποιος στο τελευταίο πρόβλημα μπορεί να εντοπίσει μια κρυμμένη σχέση στα 3 πρώτα παραδείγματα που περιλαμβάνουν το αποτέλεσμα. Μπορεί να ισχυριστεί ότι υπάρχει μια συνάρτηση $f(n) = n + (n+3) * n$ για $n \in [0, \infty)$ που ικανοποιεί τα δοσμένα παραδείγματα και κατά συνέπεια αποφασίζει να συμπληρώσει την τελευταία σχέση 8 +11 = 96. Όμως τι ακριβώς συμβαίνει στον εγκέφαλο του ανθρώπου για να βρει την παραπάνω συνάρτηση ή τι ακριβώς συμβαίνει στον εγκέφαλο για να συνδέσει την εικόνα

ενός αντικειμένου με το όνομά του και στην συνέχεια να κάνει γενίκευση ώστε να αναγνωρίζει όλα τα αντικείμενα που μοιάζουν με το συγκεκριμένο;

Το επόμενο ερώτημα είναι το εξής: Αν ο άνθρωπος κατανοούσε την αρχή λειτουργίας του πώς μαθαίνει ο ίδιος, θα μπορούσε στην συνέχεια να επινοήσει μια «μηχανή» η οποία να σκέφτεται όπως ο άνθρωπος; Αυτό το ερώτημα τέθηκε από τον μαθηματικό Alan Turing μέσα από την εργασία του "COMPUTING MACHINERY AND INTELLIGENCE" (Turing, 1950). Ο στόχος λοιπόν για την ανάπτυξη της μηχανικής μάθησης (machine learning) είχε είδη τεθεί και για να γίνει αντιληπτό το πώς επιτυγχάνεται αλγοριθμικά το εγχείρημα των Τεχνητών Νευρωνικών Δικτύων, πόσο μάλλον το εγχείρημα των Physics Informed NN (PINNs) κρίνεται απαραίτητο να γίνει μια σύντομη αναφορά στον τρόπο λειτουργείας των βιολογικών νευρωνικών δικτύων.

2.2 Αρχή λειτουργίας



Εικόνα 2.2.1 (Δομή ενός βιολογικού νευρώνα Πηγή: https://www.biologyonline.com/dictionary/axon-terminal)

Η αρχή λειτουργίας των βιολογικών νευρωνικών δικτύων των έμβιων οργανισμών κατέχει σημαντικό ρόλο στην ανάπτυξη των ΤΝΔ διότι μέσα από την κατανόηση τους προέκυψε η έμπνευση για την μηχανική μάθηση (machine learning) και την βαθιά μάθηση (deep learning). Ο ακριβής τρόπος με τον οποίο το μυαλό κάνει δυνατή τη σκέψη είναι ίσως από τα πιο δυσνόητα κομμάτια της επιστήμης, ωστόσο οι νευροεπιστήμη έχει σημειώσει ιδιαίτερα σημαντική πρόοδο στην μελέτη του νευρικού συστήματος. Στο παρόν κεφάλαιο γίνεται μια βασική περιγραφή του ανθρώπινου νευρικού συστήματος, των βασικών τμημάτων και λειτουργιών του.

Για την καλύτερη κατανόηση είναι χρήσιμό να αναφερθούν τα βασικά δομικά στοιχεία που αποτελούν το σύστημα. Στοιχειώδης δομική μονάδα του νευρικού συστήματος αποτελεί ένα κύτταρο που ονομάζεται νευρώνας (neuron). Ο ανθρώπινος εγκέφαλος εκτιμάται ότι περιλαμβάνει κατά προσέγγιση $86*10^9$ κύτταρα όπως αυτά. Κάθε νευρώνας αποτελείται από έναν κυτταρικό κορμό ή σώμα (cell body, soma) ο οποίος περιέχει έναν κυτταρικό πυρήνα (nucleus). Από τον κυτταρικό κορμό διακλαδίζονται μερικές ίνες οι οποίες ονομάζονται δενδρίτες (dendrites) και μια μοναδική μεγάλη ίνα που λέγεται άξονας (axon). Ο άξονας εκτείνεται σε μεγάλο μήκος το οποίο είναι κατά πολύ μεγαλύτερο από αυτό που φαίνεται στην κλίμακα της *Εικόνας 2.2.1.* Συνήθως οι άξονες έχουν μήκος 1 cm αλλά μπορούν να φτάσουν και το 1 m.

Ένας νευρώνας συνδέεται με 10 έως 100.000 άλλους νευρώνες μέσω σημείων σύνδεσης που λέγονται συνάψεις (synapces). Ένας νευρώνας έχει την δυνατότητα να στέλνει σήματα-παλμούς σε άλλους νευρώνες μέσω μιας περίπλοκης ηλεκτροχημικής αντίδρασης και είναι γνωστό ότι από αυτά τα σήματα ελέγχεται η εγκεφαλική δραστηριότητα βραχυπρόθεσμα, ενώ παράλληλα από αυτά επιτρέπονται οι μακροχρόνιες αλλαγές στην θέση και την συνδεσμολογία των νευρώνων (Russel & Norvig, 2003). Το μεγαλύτερο μέρος της επεξεργασίας των πληροφοριών πραγματοποιείται στον εγκεφαλικό φλοιό, δηλαδή το εξωτερικό στρώμα του εγκεφάλου.

Το νευρικό σύστημα χωρίζεται στο Κεντρικό Νευρικό Σύστημα ΚΝΣ και στο Περιφερειακό Νευρικό Σύστημα ΠΝΣ. Το ΚΝΣ αποτελείται από τον εγκέφαλο ο οποίος προστατεύεται από το κρανίο και τον νωτιαίο μυελό ο οποίος προστατεύεται από την σπονδυλική στήλη. Στα δύο αυτά σημεία πραγματοποιείται λήψη, επεξεργασία και αποστολή των πληροφοριών. Αντιθέτως το ΠΝΣ αποτελείται από κρανιακά νεύρα (cranial nerves), νωτιαία νεύρα (spinal nerves) και δισεκατομμύρια αισθητηριακούς (sensory) και κινητικούς νευρώνες (motor neurons). Η κύρια λειτουργεία του ΠΝΣ είναι να συνδέει το ΚΝΣ με το υπόλοιπο σώμα. Οι αισθητηριακοί νευρώνες στέλνουν την πληροφορία στο ΚΝΣ από τα εσωτερικά όργανα ή από εξωτερικά ερεθίσματα ενώ οι κινητικοί νευρώνες στέλνουν την πληροφορία από το ΚΝΣ προς τα εσωτερικά όργανα, στους μύες και σε αδένες.

Η επεξεργασία της πληροφορίας και ο τρόπος που οι βιολογικοί νευρώνες στέλνουν ηλεκτρικά σήματα είναι σημαντικά πολυπλοκότερη από τον τρόπο λειτουργίας των ΤΝΔ που θα αναλυθεί σε επόμενη παράγραφό. Παρόλα αυτά γίνεται εμφανές ότι ο νευρικό σύστημα του ανθρώπου αλλά και σε χαμηλότερο επίπεδο όλων των έμβιων όντων, αποτελείται από μονάδες εισόδου, κεντρική μονάδα επεξεργασίας πληροφοριών και σημάτων καθώς και μονάδες εξόδου. Το ενδιαφέρον συμπέρασμα από την μελέτη που έχει γίνει τα προηγούμενα χρόνια πάνω στο αντικείμενο, είναι ότι μια συλλογή μεμονωμένων κυττάρων μπορεί να οδηγήσει στη σκέψη, τη δράση και την συναίσθηση ή με άλλα λόγια, ότι ο εγκέφαλος δημιουργεί νόηση (Searle, 1992).

Κεφάλαιο 3 - Τεχνητά Νευρωνικά Δίκτυα ΤΝΔ

Το κεφάλαιο αυτό αποτελεί προθάλαμο για την κατανόηση της μεθόδου PINNs (Physics Informed Neural Networks) και βασικό πυλώνα ολόκληρης της εργασίας. Παρέχει το θεωρητικό υπόβαθρο, τόσο σε επίπεδο μαθηματικών όσο σε επίπεδο προγραμματισμού, για την κατανόηση του τι είναι ουσιαστικά ένα Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ), τι είναι η εκπαίδευση και πως πραγματοποιείται, με σκοπό την επίλυση προβλημάτων. Το κεφάλαιο αυτό συγγράφηκε με γνώμονα να απαντηθούν όλες οι ερωτήσεις που θα δημιουργηθούν αρχικά, σε όποιον επιχειρήσει να ασχοληθεί με την μέθοδο PINNs και δεν έχει κάποιο ιδιαίτερο υπόβαθρο στο συγκεκριμένο γνωστικό αντικείμενο. Παρακάτω αναλύονται η αρχιτεκτονική των ΤΝΔ, η διαδικασία της προς τα εμπρός (forward pass) και προς τα πίσω (backpropagation) διάδοσης του σήματος ενώ για την καλύτερη κατανόηση τους, ακολουθεί ένα παράδειγμα δικτύου το οποίο εκπαιδεύεται αρχικά με υπολογισμούς στο χαρτί και στην συνέχεια με ένα πρόγραμμα της Python το οποίο αναπτύχθηκε αποκλειστικά για την εργασία. Πριν όμως, ίσως είναι ενδιαφέρον να γίνει μια ανάγνωση των βασικότερων σταθμών του επιστημονικού κλάδου της Τεχνητής Νοημοσύνης ΤΝ (Artificial Intelligence AI) και των νευρωνικών δικτύων. Έτσι λοιπόν ακολουθεί μια σύντομη ιστορική αναδρομή από την γέννηση του πεδίου μέχρι και τα πιο πρόσφατα βήματα. Η ενασχόληση με την ιστορική πορεία της μεθόδου PINNs, η οποία είναι αρκετά πιο πρόσφατη, θα γίνει ξεχωριστά στο επόμενο κεφάλαιο.

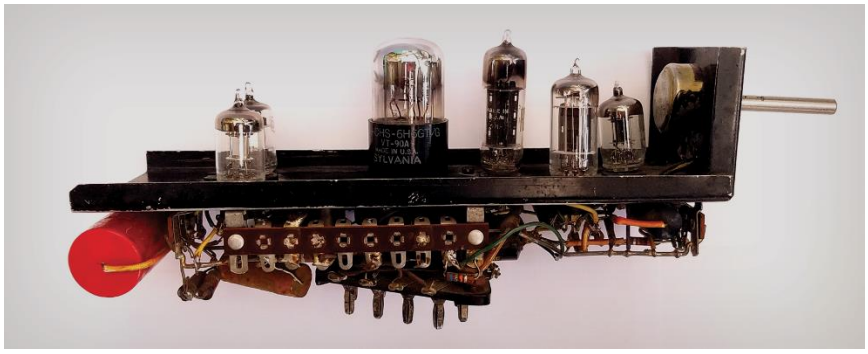
3.1 Ιστορική αναδρομή

Από την εισαγωγή, το νήμα της ιστορίας έμεινε στην αρχή της δεκαετίας του 1940 όπου τότε πραγματοποιούνταν τα πρώτα βήματα της επιστήμης των ηλεκτρονικών υπολογιστών. Ιδιαίτερο ενδιαφέρον προκαλεί το γεγονός ότι από αυτήν κιόλας την περίοδο φαίνεται ότι έγιναν τα πρώτα βήματα στο πεδίο της τεχνητής νοημοσύνης. Η πρώτη εργασία που αναγνωρίζεται ευρέως ως πρωταρχική στο πεδίο, έγινε από τον νευροφυσιολόγο Warren S. McCulloch και τον μαθηματικό Walter Pitts, με τίτλο 'A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY' (McCulloch & Pitts, 1943) (Russel & Norvig, 2003). Στην εργασία αυτή ουσιαστικά, βασίστηκαν στην γνώση της βασικής φυσιολογίας και λειτουργίας των νευρώνων του εγκεφάλου, σε μια τυπική ανάλυση της προτασιακής λογικής των Russell και Whitehead καθώς και στην θεωρία υπολογισμού του Alan Turing, με σκοπό να προτείνουν ένα μοντέλο τεχνητών νευρώνων (Russel & Norvig, 2003). Οι δυο αυτοί επιστήμονες εισήγαγαν την ιδέα ότι ένα κατάλληλα ορισμένο δίκτυο μπορεί να μαθαίνει. Ένας απλός κανόνας ενημέρωσης και τροποποίησης των συνάψεων μεταξύ των νευρώνων παρουσιάστηκε το έτος 1949 από τον Donald Hebb μέσα από το βιβλίο 'The Organization of Behavior' (Hebb, 2002) (επανέκδοση). Ο κανόνας αυτός σήμερα είναι ευρέως γνωστός ως μάθηση Hebb ή Hebbian leanring.

Μια επιπλέον γνωστή εργασία που αποτελεί ορόσημο για την ΤΝ και απασχόλησε αρκετά την επιστημονική κοινότητα, ήταν η εργασία με τίτλο "COMPUTING MACHINERY AND INTELLIGENCE" (Turing, 1950) η οποία αναφέρθηκε και στο κεφάλαιο 2 και ουσιαστικά μέσα από αυτήν ο Alan Turing θεσπίζει το ερώτημα του τι είναι πραγματικά η νοημοσύνη και πως μπορεί να αξιολογηθεί. Επόμενος σταθμός είναι το έτος 1951 όπου δύο μεταπτυχιακοί φοιτητές του μαθηματικού τμήματος του πανεπιστημίου του Princeton, ο Marvin Minsky και ο Dean Edmonds, κατασκεύασαν τον πρώτο ηλεκτρονικό υπολογιστή που προσομοιώνει ένα νευρωνικό δίκτυο. Ο υπολογιστής αυτός ονομαζόταν SNARC (Stochastic Neural Analog Reinforcement Calculator) ο οποίος χρησιμοποιούσε 3000 λυχνίες

κενού και ένα μηχανισμό αυτόματου πιλότου από βομβαρδιστικό B-24 για να προσομοιώνει ένα δίκτυο 40 νευρώνων.

Ο όρος τεχνητή νοημοσύνη φαίνεται να θεσπίστηκε από τον John McCarthy κατά το καλοκαίρι του έτους 1956 στο Dartmouth College, όπου εκεί πραγματοποιήθηκε μια δίμηνη συνάντηση εργασίας, με παρόντες μερικούς από τους σημαντικότερους ερευνητές του χώρου για αυτήν την περίοδο (Russel & Norvig, 2003).



Εικόνα 3.1.1 (Υπολογιστής SNAR, Πηγή: https://www.the-scientist.com/foundations/machine--learning--1951-65792)

Επόμενος σημαντικός σταθμός είναι η αναζήτηση ενός αλγόριθμού με τον οποίο τα δίκτυα πολλών στρωμάτων θα εκπαιδεύονται αποδοτικά. Η ιδέα της βελτίωσης των παραμέτρων του δικτύου με βάση το σφάλμα, χρησιμοποιώντας τις μερικές παραγώγους αναπτύχθηκε από αρκετούς ερευνητές ανεξάρτητα, κατά τις αρχές τις δεκαετίας του 1960. Πιο συγκεκριμένα ο Arthur Earl Bryson Jr. και ο Henry J. Kelley ήταν από τους πρώτους που ανέπτυξαν μεθόδους για την επίλυση προβλημάτων χρησιμοποιώντας τις παραγώγους μιας Loss function ως προς όλες τις παραμέτρους. Προς την ίδια κατεύθυνση κινήθηκε και ο Stuart E. Dreyfus (Dreyfus, 1990) καθώς και αρκετοί ακόμα. Κατά την δεκαετία αυτήν λοιπόν αναπτύσσεται μια πρώτη βάση για την ιδέα του αλγορίθμου της προς τα πίσω διάδοσης του σφάλματος (backpropagation) χρησιμοποιώντας τον κανόνα της αλυσιδωτής παραγώγισης (Rojas, 1996).

Το σημαντικότερο άλμα στην εξέλιξη του αλγορίθμου backpropagation σημειώθηκε στην δεκαετία του 1980 όπου τότε προτάθηκε ο αλγόριθμος στην μορφή που χρησιμοποιείται και σήμερα όπως θα παρουσιαστεί σε επόμενη παράγραφο. Χαρακτηριστικές εργασίες αυτής της δεκαετίας είναι οι εξής που ακολουθούν (Rumelhart, Hinton, & Williams, LEARNING INTERNAL REPRESENTATIONS By ERROR PROPAGATION, 1985) και (Rumelhart, Hinton, & Williams, Leanring representations by back - propagating errors, 1986). Ουσιαστικά από εκείνο το σημείο και μετά, το πεδίο των νευρωνικών δικτύων αρχίζει να παίρνει κατά κάποιο τρόπο, την μορφή που έχει σήμερα, καθώς ο βασικός τρόπος λειτουργίας του αλγορίθμου backpropagation δεν έχει αλλάξει σημαντικά από τότε. Ένας σημαντικός παράγοντας που επέτρεψε στο πεδίο των νευρωνικών δικτύων να ευδοκιμήσουν, ήταν η πρόταση ενός τρόπου για αποδοτικότερο υπολογισμό των απαραίτητων παραγώγων ώστε να πραγματοποιηθεί η διαδικασία του backpropagation. Πρόκειται για την πρόταση του αλγόριθμου Automatic differentiation από τον Φιλανδό μαθηματικό και computer scientist Seppo Ilmari Linnainmaa (Baydin, Pearlmutter , Radul, & Siskind, 2018). Ο αλγόριθμος αυτός ουσιαστικά υπολογίζει τις απαραίτητες παραγώγους με βέλτιστο τρόπο ελαχιστοποιώντας το υπολογιστικό κόστος και αποτελεί σημαντικό σκαλοπάτι για την πορεία των ΤΝΔ. Λεπτομέρειες για τον τρόπο λειτουργίας του αλγορίθμου υπάρχουν στο κεφάλαιο 4 όπου εκεί αναλύονται

οι 2 εκδοχές του και όλα τα βήματα τα οποία εξηγούνται παράλληλα μέσω ενός χαρακτηριστικού παραδείγματος.

Στην σημερινή εποχή, ως οι κύριοι ερευνητές οι οποίοι συνέβαλαν στην ανάπτυξη του αλγόριθμου backpropagation και γενικότερα της σημαντικής βελτίωσης του τρόπου λειτουργίας των ΤΝΔ θεωρούνται οι Yoshua Bengio, Geoffrey Hinton και Yann LeCun στους οποίους απονεμήθηκε το βραβείο Turing[1] το 2018 για την συνεισφορά τους αυτή. Φυσικά ανάμεσα στα γεγονότα που αναφέρθηκαν μεσολάβησαν αρκετοί ακόμα ερευνητές με πολύ σημαντικές εργασίες. Παράλληλα για να προκύψει η μεγάλη ακμή που βιώνει σήμερα το πεδίο των ΤΝΔ χρειάστηκε και η ανάπτυξη ισχυρού και οικονομικού υλικού υπολογιστών (hardware) ώστε να γίνεται εφικτή η υλοποίηση της εκπαίδευσης πυκνών δικτύων, διαδικασία που όπως θα φανεί παρακάτω είναι ιδιαίτερα επίπονη υπολογιστικά. Το υλικό αυτό, περιλαμβάνει ισχυρές κεντρικές μονάδες επεξεργασίας CPU, κάρτες γραφικών GPU ή TPU (Tensor Processing Unit), προσιτές σε μέγεθος και κόστος, προκειμένου να μπορεί να επεκταθεί ευκολότερα η εφαρμογή και η έρευνα πάνω στα νευρωνικά δίκτυα. Τα τελευταία χρόνια η εξέλιξη του πεδίου έχει φτάσει σε τέτοιο βαθμό που τα νευρωνικά δίκτυα εφαρμόζονται ευρέως και αποτελεσματικά σε προβλήματα όπως η αναγνώριση εικόνας και ήχου, η μετάφραση κειμένου, η εμφάνιση κατάλληλων προτεινόμενων αναζητήσεων, ή πρόβλεψη των τιμών ακινήτων, οι ιατρικές διαγνώσεις ή ακόμα και η επίλυση διαφορικών εξισώσεων.

Ως γενικός χάρτης του πεδίου της τεχνητής νοημοσύνης, υπάρχει μια ευρέως αποδεκτή αντίληψη. Θεωρείται ότι υπάρχουν δύο κοινά και παράλληλα εξελισσόμενα πεδία, όπου αυτά είναι η Τεχνητή Νοημοσύνη ΤΝ (Artificial Intelligence AI) και η Υπολογιστική Νοημοσύνη ΥΝ (Computational Intelligence CI). Η υπολογιστική νοημοσύνη αποτελεί υποσύνολο της τεχνητής νοημοσύνης. Η ΤΝ είναι το πεδίο που θεσπίζει τεχνικές με ισχυρό θεωρητικό υπόβαθρο και η ΥΝ φαίνεται να ασχολείται με την ανάπτυξη αλγορίθμων εμπνευσμένους από την φύση όπως για παράδειγμα οι εξελικτικοί αλγόριθμοι που ουσιαστικά μιμούνται την διαδικασία της φυσικής επιλογής ή η ασαφής λογική η οποία μιμείται την ομιλούμενη γλώσσα ή τα ΤΝΔ τα οποία μιμούνται την λειτουργία του εγκεφάλου. Υποσύνολο της ΥΝ συντελεί η μηχανική μάθηση (machine learning ML) της οποίας υποσύνολο αποτελεί η βαθιά μάθηση (deep learning DL) δηλαδή τα τεχνητά νευρωνικά δίκτυα (Mumford & Jain, 2009).

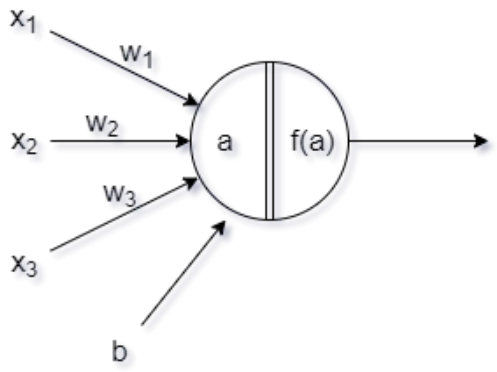
Κατά την ιστορική αναδρομή παραλήφθηκαν πολλά σημαντικά στοιχεία στην πορεία της τεχνητής νοημοσύνης καθώς κρίθηκε προτιμότερο να αναφερθούν οι κυριότεροι σταθμοί της εξέλιξης των ΤΝΔ τα οποία και απασχολούν εξ ολοκλήρου την παρούσα εργασία. Στο κεφάλαιο 4 θα παρατεθεί μια επιπλέον σύντομη ιστορική αναδρομή με επίκεντρο την ανάπτυξη της μεθόδου PINNs.

3.2 Αρχιτεκτονική ΤΝΔ (ANN)

3.2.1 Ο νευρώνας ως δομικό στοιχείο

Ένα ΤΝΔ είναι μια πολύπλοκη μη γραμμική συνάρτηση με πολλές παραμέτρους η οποία χρησιμοποιεί αρχιτεκτονική παρόμοια με αυτή των βιολογικών νευρωνικών δικτύων και προσομοιώνει την λειτουργία τους. Κατά αντιστοιχία με τα βιολογικά ΝΝ, στοιχειώδες δομικό στοιχείο των ΤΝΔ αποτελεί ένας νευρώνας (neuron) ο οποίος ουσιαστικά λαμβάνει τις πληροφορίες από τις εισόδους του δικτύου ή από άλλους νευρώνες, τις επεξεργάζεται και τις μεταβιβάζει στους επόμενους νευρώνες. Το σύνολο από αρκετούς νευρώνες σε στρώματα (layers) αποτελεί ένα ΤΝΔ. Για την μαθηματική περιγραφή της παραπάνω διαδικασίας θα χρειαστεί η απεικόνιση ενός νευρώνα όπως φαίνεται στην Εικόνα 3.2.1.1.

[1] https://www.cbc.ca/news/science/turing-award-ai-deep-learning-1.5070415



Εικόνα 3.2.1.1 (Δομή ενός απλού νευρώνα - Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net)

Οι τιμές x1, x2, x3 καλούνται είσοδοι (inputs) του νευρώνα και οι τιμές w1, w2, w3 καλούνται βάρη (weights) ενώ η τιμή b καλείται σταθερά πόλωσης (bias). Ο αθροιστής a ισούται με a = x1 *w1 + x2 * w2 + x3* w3 +b = $\sum_{i=1}^{n} xi * wi + b$ όπου n είναι ο αριθμός των εισόδων. Ο αθροιστής μπορεί να συναντηθεί στην βιβλιογραφία και με πολλούς άλλους συμβολισμούς πέρα του a όμως στην παρούσα εργασία θα χρησιμοποιηθεί ο συγκεκριμένος συμβολισμός κατά σύμβαση (Gurney, 1997). Για καλύτερη διευκόλυνση στις πράξεις και αργότερα στον προγραμματισμό είναι προτιμότερο να γραφούν οι παραπάνω πράξεις σε μορφή διανυσμάτων.

$$x = [x1 \quad x2 \quad x3], \qquad w = \begin{bmatrix} w1 \\ w2 \\ w3 \end{bmatrix} \tag{4}$$

Συνεπώς το a εκφράζεται ως το εσωτερικό γινόμενο των διανυσμάτων x και w ως **a = x * w + b.** Αυτός ο τρόπος γραφής θα φανεί πολύ χρήσιμος στην συνέχεια όταν θα κατασκευαστεί ένα δίκτυο από νευρώνες, καθώς για να κατασκευαστεί ένας αλγόριθμος που να προσομοιώνει ένα ΤΝΔ θα πρέπει οι τιμές να είναι εκφρασμένες με διανύσματα και πίνακες. Στην συνέχεια ο αθροιστής a εισέρχεται ως είσοδος σε μια συνάρτηση ενεργοποίησης (activation function) η οποία κατέχει σημαντικό ρόλο στην λειτουργία ενός νευρωνικού δικτύου καθώς το μετατρέπει σε μια μη γραμμική συνάρτηση με μεταβλητές τις τιμές των εισόδων και των βαρών. Οι συναρτήσεις που συναντώνται πιο συχνά είναι η συνάρτηση Heaviside, η σιγμοειδής (sigmoid) ή λογιστική, η υπερβολική εφαπτομένη (hyperbolic tangent) και η συνάρτηση ReLU.

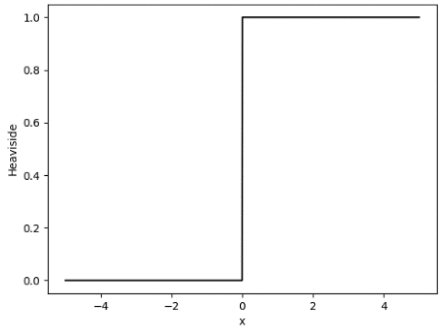
3.2.2 Συναρτήσεις ενεργοποίησης (Activation functions)

- <u>Συνάρτηση Heaviside</u>

$$f(x) = \begin{cases} 0, & x < T \\ 1, & x \geq T \end{cases} \tag{5}$$

Ως T αναφέρεται το κατώφλι στην ελληνική βιβλιογραφία όπου σε αυτήν την περίπτωση ο νευρώνας λειτουργεί δυαδικά ως διακόπτης. Αν ο αθροιστής έχει μεγαλύτερη τιμή από το κατώφλι ο νευρώνας ενεργοποιείται και μεταβιβάζει την τιμή 1 σαν έξοδο. Η συνάρτηση αυτή πλέον δεν

χρησιμοποιείται σχεδόν καθόλου στα σύγχρονα ΤΝΔ όμως στα πρώτα χρόνια χρησιμοποιούνταν για να προσομοιωθούν διάφορες λογικές πύλες όπως η πύλη AND, OR, NAND ή NOT.

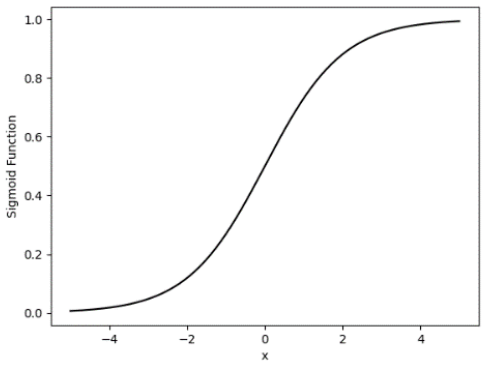


Εικόνα 3.2.2.1 (Heaviside Function - Σχεδιάστηκε στην Python)

- Σιγμοειδής Συνάρτηση (Sigmoid Function)

$$f(x) = \frac{1}{1+ e^{-x}} \quad (6)$$

Η συγκεκριμένη συνάρτηση είναι η πιο συχνά χρησιμοποιούμενη σε εφαρμογές καθώς το ΤΝΔ μέσω αυτής παρουσιάζει πολύ ομαλή συμπεριφορά.

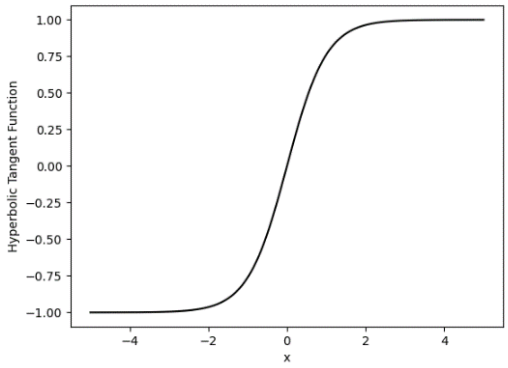


Εικόνα 3.2.2.2 (Sigmoid Function - Σχεδιάστηκε στην Python)

- Υπερβολική εφαπτομένη (Hyperbolic Tangent)

$$f(x) = \tanh(x) = \frac{e^{x}- e^{-x}}{e^{x}+ e^{-x}} \quad (7)$$

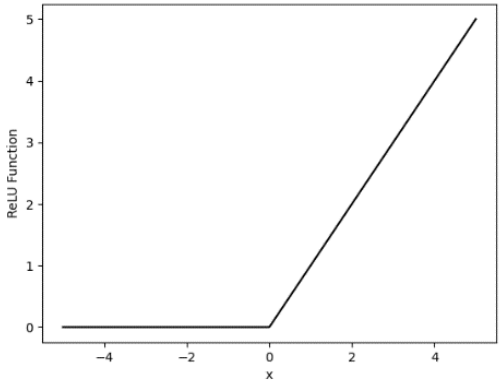
Πρόκειται επίσης για ακόμα μια πολύ συχνά χρησιμοποιούμενη συνάρτηση ενεργοποίησης η οποία θα χρησιμοποιηθεί στην εφαρμογή της μεθόδου PINNs σε επόμενο κεφάλαιο.



Εικόνα 3.2.2.3 (Hyperbolic Tangent Function – Σχεδιάστηκε στην Python)

- ReLU Function (Rectifier Linear Unit Activation Function)

$$f(x) = \max(0, x) \tag{8}$$

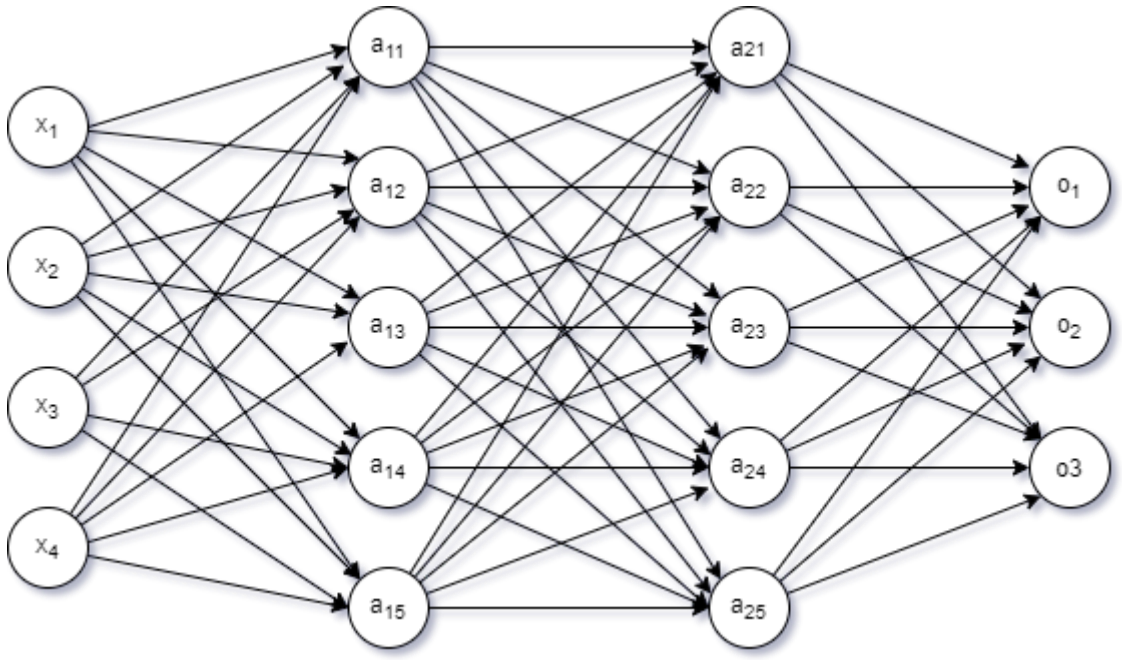


Εικόνα 3.2.2.4 (ReLU Function – Σχεδιάστηκε στην Pyhton)

Οι παραπάνω συναρτήσεις συχνά στην βιβλιογραφία συναντώνται και ως «squashing functions». Σε επόμενες παραγράφους θα γίνει κατανοητό γιατί θα πρέπει οι συναρτήσεις ενεργοποίησης να είναι παραγωγίσιμες.

3.2.3 Η δομή ενός δικτύου και η προς τα εμπρός τροφοδότηση (Forward – pass)

Ο συνδεσμολογία πολλών νευρώνων σαν αυτών που περιεγράφηκαν στην προηγούμενη παράγραφο, συντελούν ένα ΤΝΔ. Η προς τα εμπρός τροφοδότηση του δικτύου με την πληροφορία να διαδίδεται από τις εισόδους του δικτύου μέχρι τις εξόδους, διαπερνώντας όλα τα στρώματα νευρώνων, αναφέρεται ως forward - pass. Παρακάτω παρατίθεται ένα παράδειγμα νευρωνικού δικτύου, προκειμένου η επεξήγηση να μπορεί να γίνει πιο παραστατικά.



Εικόνα 3.2.3.1 (ΤΝΔ - 4 είσοδοι - 2 hidden layers – 3 έξοδοι – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net)

Στην Εικόνα 3.2.3.1 φαίνεται ένα ΤΝΔ που αποτελείται από 4 εισόδους και 3 εξόδους. Τα δυο ενδιάμεσα στρώματα αναφέρονται ως hidden layers και στην προκειμένη περίπτωση το παραπάνω δίκτυο αποτελείται από 2 hidden layers όπου το κάθε ένα αποτελείται από 5 νευρώνες. Για την επεξήγηση της προς τα εμπρός τροφοδότησης είναι χρήσιμο να οριστούν οι εξής πίνακες των βαρών. Ο πίνακας W^1 περιλαμβάνει τα βάρη μεταξύ των συνδέσεων των νευρώνων από το layer εισόδου (input layer) προς το 1ο hidden layer, ο πίνακας W^2 περιλαμβάνει τα βάρη από το 1ο hidden layer προς το 2ο hidden layer και ο πίνακας W^3 περιλαμβάνει τα βάρη από το 2ο hidden layer προς το layer εξόδου (output layer). Ως γενίκευση, ο πίνακας W^L περιλαμβάνει τα βάρη που συνδέουν τους νευρώνες του L-1 hidden layer με τους νευρώνες του L hidden layer. Πιο αναλυτικά:

$$W^1=\begin{bmatrix} w1,11 & w1,12 & w1,13 & w1,14 & w1,15 \\ w1,21 & w1,22 & w1,23 & w1,24 & w1,25 \\ w1,31 & w1,32 & w1,33 & w1,34 & w1,35 \\ w1,41 & w1,42 & w1,43 & w1,44 & w1,45 \end{bmatrix} \quad W^2=\begin{bmatrix} w2,11 & w2,12 & w2,13 & w2,14 & w2,15 \\ w2,21 & w2,22 & w2,23 & w2,24 & w2,25 \\ w2,31 & w2,32 & w2,33 & w2,34 & w2,35 \\ w2,41 & w2,42 & w2,43 & w2,44 & w2,45 \\ w2,51 & w2,52 & w2,53 & w2,54 & w2,55 \end{bmatrix}$$

$$W^3=\begin{bmatrix} w3,11 & w3,12 & w3,13 \\ w3,21 & w3,22 & w3,23 \\ w3,31 & w3,32 & w3,33 \\ w3,41 & w3,42 & w3,43 \\ w3,51 & w3,52 & w3,53 \end{bmatrix} \quad (9)$$

Κατά σύμβαση, τα layers αριθμούνται με L = 0, 1, 2, 3…n με το layer εισόδου να αριθμείται με τον αριθμό 0 και το layer εξόδου να αριθμείται με τον αριθμό n. Ως έξοδος από κάθε layer αποτελεί ένα διάνυσμα που συμβολίζεται με z και κατά συνέπεια το διάνυσμα των εισόδων μπορεί να γραφεί ως z^0. Επίσης όπως προαναφέρθηκε, σε κάθε νευρώνα προστίθεται ένα bias b συνεπώς για κάθε layer υπάρχει ένα επιπλέον διάνυσμα b το οποίο περιλαμβάνει τα bias των νευρώνων του layer. Προφανώς το layer

εισόδου δεν έχει bias, επομένως η αρίθμηση των διανυσμάτων ξεκινάει από b^1 το οποίο είναι το διάνυσμα που περιλαμβάνει τα biases του 1ου hidden layer.

Κατά σύμβαση[2], ο συμβολισμός ενός βάρους w (weight) ορίζεται ως wL,ij το οποίο περιέχεται μεταξύ του layer L-1 και του L και υπάρχει στην σύναψη μεταξύ του i - οστού νευρώνα του layer L-1 και του j – οστού νευρώνα του layer L. Τα διανύσματα των biases ορίζονται ως εξής:

$b^1 = [b11 \quad b12 \quad b13 \quad b14 \quad b15]$ (10)

$b^2 = [b21 \quad b22 \quad b23 \quad b24 \quad b25]$ (11)

$b^3 = [b31 \quad b32 \quad b33]$ (12)

Θεωρώντας ως συνάρτηση ενεργοποίησης την σιγμοειδή (sigmoid function) η οποία συμβολίζεται εδώ ως s(x), η προς τα εμπρός τροφοδότηση του δικτύου (forward – pass) πραγματοποιείται ως εξής:

Ως γενικός τύπος για την διαδικασία του forward – pass με activation function μια οποιαδήποτε σ(x), ορίζεται ο ακόλουθος:

$$\mathbf{z^L = \sigma(z^{L-1} * W^L + b^L)} \quad \mathbf{(13)}$$

$z^0 \equiv [x1 \quad x2 \quad x3 \quad x4]$ (14)

$z^1 = \sigma(z^0 * W^1 + b^1) =$

$$= \sigma([x1 \quad x2 \quad x3 \quad x4] * \begin{bmatrix} w1,11 & w1,12 & w1,13 & w1,14 & w1,15 \\ w1,21 & w1,22 & w1,23 & w1,24 & w1,25 \\ w1,31 & w1,32 & w1,33 & w1,34 & w1,35 \\ w1,41 & w1,42 & w1,43 & w1,44 & w1,45 \end{bmatrix} + [b11 \quad b12 \quad b13 \quad b14 \quad b15])$$

$$= \sigma(\begin{bmatrix} x1 * w1,11 + x2 * w1,21 + x3 * w1,31 + x4 * w1,41 \\ x1 * w1,12 + x2 * w1,22 + x3 * w1,32 + x4 * w1,42 \\ x1 * w1,13 + x2 * w1,23 + x3 * w1,33 + x4 * w1,43 \\ x1 * w1,14 + x2 * w1,24 + x3 * w1,34 + x4 * w1,44 \\ x1 * w1,15 + x2 * w1,25 + x3 * w1,35 + x4 * w1,45 \end{bmatrix}^T + \begin{bmatrix} b11 \\ b12 \\ b13 \\ b14 \\ b15 \end{bmatrix}^T) = [z11 \;\; z12 \;\; z13 \;\; z14 \;\; z15]$$

(15)

Ως zij ορίζεται η έξοδος του νευρώνα j που βρίσκεται στο layer i. Η αρίθμηση των νευρώνων σε κάθε layer αρχίζει από τον αριθμό 1 σε αντίθεση με την αρίθμηση των layers που αρχίζει από το 0.

$z^2 = \sigma(z^1 * W^2 + b^2) =$

[2] Ο συγκεκριμένος συμβολισμός των βαρών (wL,ij) δεν συναντήθηκε μέχρι στιγμής σε κάποιο άρθρο ή βιβλίο, όμως πολύ πιθανό να υπάρχει κάποιος παρόμοιος συμβολισμός. Τα περισσότερα συγγράμματα που αναγνώστηκαν, δεν ανέλυαν τόσο την διαδικασία του forward – pass, πόσο μάλλον δεν παρέθεταν πράξεις πινάκων όπως παραπάνω. Η κατανόηση του τρόπου λειτουργίας του forward – pass επήλθε μέσα από τον συνδυασμό μελέτης βιβλίων και άρθρων καθώς και πρακτικά μέσα από κατάλληλες εκτυπώσεις διανυσμάτων και πινάκων σε προγράμματα που αναπτύχθηκαν στην Python.

$$= \sigma([z11 \quad z12 \quad z13 \quad z14 \quad z15] * \begin{bmatrix} w2,11 & w2,12 & w2,13 & w2,14 & w2,15 \\ w2,21 & w2,22 & w2,23 & w2,24 & w2,25 \\ w2,31 & w2,32 & w2,33 & w2,34 & w2,35 \\ w2,41 & w2,42 & w2,43 & w2,44 & w2,45 \\ w2,51 & w2,52 & w2,53 & w2,54 & w2,55 \end{bmatrix} +$$

$$+ [b21 \quad b22 \quad b23 \quad b24 \quad b25]) =$$

$$= \sigma(\begin{bmatrix} z11 * w2,11 + z12 * w2,21 + z13 * w2,31 + z14 * w2,41 + z15 * w2,51 \\ z11 * w2,12 + z12 * w2,22 + z13 * w2,32 + z14 * w2,42 + z15 * w2,52 \\ z11 * w2,13 + z12 * w2,23 + z13 * w2,33 + z14 * w2,43 + z15 * w2,53 \\ z11 * w2,14 + z12 * w2,24 + z13 * w2,34 + z14 * w2,44 + z15 * w2,54 \\ z11 * w2,15 + z12 * w2,25 + z13 * w2,35 + z14 * w2,45 + z15 * w2,55 \end{bmatrix}^T + \begin{bmatrix} b21 \\ b22 \\ b23 \\ b24 \\ b25 \end{bmatrix}^T) =$$

$$= [z21 \;\; z22 \;\; z23 \;\; z24 \;\; z25] \qquad (16)$$

$z^3 = \sigma(z^2 * W^3 + b^3) =$

$$= \sigma([z21 \quad z22 \quad z23 \quad z24 \quad z25] * \begin{bmatrix} w3,11 & w3,12 & w3,13 \\ w3,21 & w3,22 & w3,23 \\ w3,31 & w3,32 & w3,33 \\ w3,41 & w3,42 & w3,43 \\ w3,51 & w3,52 & w3,53 \end{bmatrix} + [b31 \quad b32 \quad b33] =$$

$$= \sigma(\begin{bmatrix} z21 * w3,11 + z22 * w3,21 + z23 * w3,31 + z24 * w3,41 + z25 * w3,51 \\ z21 * w3,12 + z22 * w3,22 + z23 * w3,32 + z24 * w3,42 + z25 * w3,52 \\ z21 * w3,13 + z22 * w3,23 + z23 * w3,33 + z24 * w3,43 + z25 * w3,53 \end{bmatrix}^T + \begin{bmatrix} b31 \\ b32 \\ b33 \end{bmatrix}^T) =$$

$$= [z31 \;\; z32 \;\; z33] = [o1 \;\; o2 \;\; o3] \qquad (17)$$

Το παραπάνω διάνυσμα αποτελεί το διάνυσμα εξόδου του δικτύου και περιλαμβάνει τις τιμές που εξέρχονται από τους 3 νευρώνες εξόδου. Ουσιαστικά η έξοδος του κάθε νευρώνα εκ των 3^{ων} αποτελεί μια συνάρτηση $f_m(x_i)$ [3] των εισόδων x_i. Κατά την προς τα εμπρός διάδοση, μετά το πέρας της εκπαίδευσης, τα βάρη και τα biases αποτελούν σταθεροί παράμετροι της συνάρτησης. Το αντίθετο συμβαίνει στην εκπαίδευση, όπου οι είσοδοι x_i και έξοδοι o_i θεωρούνται σταθερές και τα βάρη με τα biases θεωρούνται μεταβλητές. Πάνω σε αυτό θα αναφερθούν περισσότερα σε επόμενη παράγραφο που θα αναλυθεί η διαδικασία της εκπαίδευσης και του backpropagation. Μια εικόνα της συνάρτησης που προκύπτει από την προς τα εμπρός διάδοση του σήματος μέσα στο δίκτυο για μια οποιαδήποτε έξοδό του $f_m(x_i)$ θα είναι της μορφής $f_m(x_i) = o_i$ = sigmoid(sigmoid(sigmoid(….) + …) + …). Αν στο παραπάνω δίκτυο που χρησιμοποιήθηκε ως παράδειγμα υπήρχαν επιπλέον 5 ή 6 layers με 50 ή 60 επιπλέον νευρώνες το κάθε ένα, γίνεται αντιληπτό το πόσο σύνθετη συνάρτηση θα είχε προκύψει.

Οι παραπάνω τύποι προέρχονται εν μέρει, από αρκετές εργασίες με θέμα την μέθοδο PINNs, όμως τροποποιήθηκαν ως έναν βαθμό ώστε να συμβαδίζουν αυστηρά με τις παραπάνω πράξεις πινάκων. Μια τέτοια σημαντική εργασία έχει τίτλο ‘A deep learning framework for solution and discovery in solid

[3] Ο συμβολισμός f_m παριστάνει την έξοδο του εκάστοτε νευρώνα εξόδου του ΤΝΔ. Εν προκειμένω m = 1, 2, 3. Παρακάτω στην παράγραφο 4.3.1 θα χρησιμοποιηθεί ο συμβολισμός f: $R^n \rightarrow R^m$ για τις συναρτήσεις εξόδου με n τον αριθμό των βαρών και biases του δικτύου (trainable parameters), ούτως ώστε να επεξηγηθεί το γιατί ο αλγόριθμος backpropagation πραγματοποιείται με την βοήθεια του Reverse Mode του αλγόριθμου Automatic Differentiation.

mechanics' (Haghighat, Raissi, Moure, Gomez, & Juanes, 2020), όπου σε αυτήν, η σχέση (13) που είναι γραμμένη παραπάνω, αναγράφεται ως:

$$\mathbf{z^l = \sigma(W^l z^{l-1} + b^l)\ \gamma\iota\alpha\ l = 1, 2, 3 .. L} \qquad \mathbf{(13)'}$$

Η σχέση αυτή είναι πολύ βοηθητική για να κατανοήσει κάποιος την λογική με την οποία πραγματοποιούνται οι πράξεις κατά την προς τα εμπρός τροφοδότηση (forward – pass). Όμως, παρατηρήθηκε ένα ασαφές κομμάτι ως αναφορά το πώς μεταφράζεται αυτή η σχέση σε πράξεις πινάκων. Για να γίνει η τεκμηρίωση, θα χρειαστεί να εφαρμοστεί η σχέση (13)' στο παράδειγμα της Εικόνας 3.2.3.1. Ας γίνει η αρχική υπόθεση ότι ο τύπος που εξετάζεται είναι σωστός και μπορεί να εφαρμοστεί. Θεωρείται αρχικά ως δεδομένο ότι $z^0 \equiv [x1 \quad x2 \quad x3 \quad x4]$ και $b^1 = [b11 \quad b12 \quad b13 \quad b14 \quad b15]$. Τότε:

$z^1 = \sigma(W^1 z^0 + b^1)$

Δηλαδή, σύμφωνα με τον ορισμό των πινάκων που περιλαμβάνουν τα βάρη από την σχέση (9), πρόκειται για πράξεις πινάκων με τις εξής διαστάσεις (m×n): (4×5) * (1×4) + (1×5) . Η συγκεκριμένη πράξη είναι αδύνατον να πραγματοποιηθεί. Η επόμενη υπόθεση, είναι ότι τα διανύσματα που περιλαμβάνουν τις τιμές των νευρώνων εισόδου - εξόδου, των νευρώνων των εσωτερικών στρωμάτων και των biases, δεν είναι διανύσματα γραμμής όπως ορίστηκαν στις σχέσεις (10), (11), (12) και (14) αλλά διανύσματα στήλης. Αυτό συνεπάγεται ότι $z^0 = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix}$ και $b^1 = \begin{bmatrix} b11 \\ b12 \\ b13 \\ b14 \\ b15 \end{bmatrix}$

Κατά συνέπεια, μιλώντας με διαστάσεις πινάκων, ισχύει η ακόλουθη σχέση:

$z^1 = \sigma(W^1 z^0 + b^1) = \sigma((4\times5) * (4\times1) + (5\times1))$

Η παραπάνω πράξη και αυτή τη φορά, δεν μπορεί να πραγματοποιηθεί, καθώς δεν συνάδουν οι διαστάσεις του πίνακα των βαρών και του διανύσματος των εισόδων, ώστε να πραγματοποιηθεί ο πολλαπλασιασμός. Ο μόνος τρόπος που θα μπορούσε να πραγματοποιηθεί αυτή η πράξη θα ήταν αν ίσχυε ότι ο πίνακας W^1 είχε διαστάσεις (5×4). Δηλαδή αν ήταν της εξής μορφής:

$$W^1 = \begin{bmatrix} w1,11 & w1,21 & w1,31 & w1,41 \\ w1,12 & w1,22 & w1,32 & w1,42 \\ w1,13 & w1,23 & w1,33 & w1,43 \\ w1,14 & w1,24 & w1,34 & w1,44 \\ w1,15 & w1,25 & w1,35 & w1,45 \end{bmatrix}$$

Αν συνέβαινε κάτι τέτοιο, τότε θα ίσχυε:

$$z^1 = \sigma(W^1 z^0 + b^1) = \sigma((5\times4) * (4\times1) + (5\times1)) = \sigma((5\times1)) = \begin{bmatrix} z11 \\ z12 \\ z13 \\ z14 \\ z15 \end{bmatrix}$$

Δηλαδή το διάνυσμα z^1, θα είχε προκύψει ως ένα διάνυσμα στήλης και θα επιβεβαίωνε την αρχική υπόθεση. Επίσης, με αυτόν τον τρόπο υπολογισμού προκύπτουν οι ίδιες ποσότητες z11, z12, z13, z14 και z15 που υπολογίστηκαν παραπάνω με την σχέση (13) και κατά συνέπεια και οι δύο τρόποι υπολογισμού είναι εφικτοί. Το πρόβλημα όμως έγκειται στο γεγονός ότι κατά την κατασκευή του παραπάνω μοντέλου στην Python με χρήση του Tensorflow και του Keras, γίνεται εμφανές ότι ο πίνακας W^1 δεν είναι διαστάσεων

(5×4) αλλά (4×5) όπως είχε γραφτεί εξ αρχής στην σχέση (9). Συνεπώς η υπόθεση ότι ο πίνακας W^1 έχει διαστάσεις (5×4) δεν ισχύει και κατά επέκταση, η σχέση $z^l = \sigma(W^l z^{l-1} + b^l)$ (13)' δεν μπορεί να χρησιμοποιηθεί. Η περίπτωση που μπορεί να χρησιμοποιηθεί η σχέση (13)' είναι εάν ισχύει ότι οι πίνακες W^l είναι πράγματι της μορφής των πινάκων που ορίστηκαν στην σχέση (9) αλλά χρησιμοποιούνται ανεστραμμένοι στην σχέση του z^l. Η παρατήρηση αυτή παρατίθεται με κάθε επιφύλαξη αλλά και με απόλυτο σεβασμό στους ερευνητές που χρησιμοποιούν την σχέση $z^l = \sigma(W^l z^{l-1} + b^l)$ (13)', καθώς εκείνοι είναι αυτοί που ανέπτυξαν και εξέλιξαν την μέθοδο, καθώς επίσης μέσω των εργασιών τους, δίνεται η ευκαιρία σε πολλούς ανθρώπους να γνωρίσουν και να ασχοληθούν με την μέθοδο PINNs. Παρακάτω παρατίθεται το μοντέλο της Εικόνας 3.2.3.1 κατασκευασμένο στην Python, συνοδευόμενο με την εκτύπωση των παραμέτρων του μοντέλου (trainable parameters) σε μορφή πινάκων, όπου επιβεβαιώνεται πως οι πίνακες W^L ορίζονται όπως φαίνεται στην σχέση (9).

```python
import tensorflow as tf
def build(num_inputs=4, layers=[5, 5], activation='sigmoid', num_outputs=3):
    inputs = tf.keras.layers.Input(shape=(num_inputs,))
    x = inputs
    for layer in layers:
        x = tf.keras.layers.Dense(layer, activation=activation,
                                  kernel_initializer='he_normal',
                                  bias_initializer='he_normal')(x)
    out = tf.keras.layers.Dense(num_outputs, activation=activation,
                                kernel_initializer='he_normal',
                                bias_initializer='he_normal')(x)
    model = tf.keras.models.Model(inputs=inputs, outputs=out)
    return model
example_model = build()
trainable_parameters = example_model.trainable_variables
print(trainable_parameters)
```

Εικόνα 3.2.3.2 (Κατασκευή ΤΝΔ με 4 νευρώνες εισόδου, 2 hidden layers με 5 νευρώνες το κάθε ένα και 3 νευρώνες εξόδου)

```
[<tf.Variable 'dense/kernel:0' shape=(4, 5) dtype=float32, numpy=
array([[ 0.15164064,  1.1182275 , -1.5341749 , -0.21658435, -1.1845505 ],
       [-0.62179685, -0.6855464 , -0.4903109 ,  1.5591513 , -0.4690039 ],
       [-0.5221696 , -0.44423673,  0.06754284, -0.391769  ,  0.126384  ],
       [-1.0291352 ,  0.72403395,  0.42360204, -0.2883727 , -0.8635492 ]],
      dtype=float32)>, <tf.Variable 'dense/bias:0' shape=(5,) dtype=float32, numpy=
array([-0.51648164,  0.4787659 ,  0.21227828, -0.4030173 ,  0.35185027],
      dtype=float32)>, <tf.Variable 'dense_1/kernel:0' shape=(5, 5) dtype=float32, numpy=
array([[ 0.56689876,  0.07109486,  0.443771  ,  0.19007613,  1.3605931 ],
       [-0.07733324, -0.26264924, -0.674861  ,  0.02046933,  0.10365798],
       [ 0.42575258,  0.47682813,  0.14388822, -0.62994605,  0.40968713],
       [-0.03236886, -0.10690074, -0.22802512,  1.2709543 ,  0.27768138],
       [-0.22632357,  1.0475844 ,  0.57953393,  0.9216974 ,  0.01501943]],
      dtype=float32)>, <tf.Variable 'dense_1/bias:0' shape=(5,) dtype=float32, numpy=
array([-1.114668  , -0.2553722 ,  0.30560616,  0.20904987,  0.85446006],
      dtype=float32)>, <tf.Variable 'dense_2/kernel:0' shape=(5, 3) dtype=float32, numpy=
array([[-0.70756567,  0.04642298,  0.6334062 ],
       [-0.55085635,  0.5427488 , -0.17444232],
       [ 0.24651338,  0.37886804,  0.9331477 ],
       [ 0.96172005,  0.6307613 ,  0.6451651 ],
       [-0.5789655 ,  0.9613481 , -0.01473559]], dtype=float32)>, <tf.Variable 'dense_2/bias:0' shape=(3,)

Process finished with exit code 0
```

Εικόνα 3.2.3.3 (Εκτύπωση των παραμέτρων του ΤΝΔ - πίνακες με βάρη, διανύσματα με biases)

Κλείνοντας το κομμάτι της προς τα εμπρός τροφοδότησης, καλό είναι να αναφερθεί ένα χαρακτηριστικό παράδειγμα εφαρμογής. Ίσως το αντίστοιχο εισαγωγικό πρόγραμμα "Hallo World!" για τα ΤΝΔ και την μηχανική μάθηση να είναι η αναγνώριση χειρόγραφων αριθμών. Συχνά το πρόβλημα αυτό συναντάται ως MNIST handwritten digit classification. MNIST (Modified National Institute of Standards and Technology) database είναι μια βάση δεδομένων με δεκάδες χιλιάδες ασπρόμαυρες εικόνες 28 × 28 pixel που απεικονίζουν χειρόγραφους αριθμούς από το 0 έως το 9. Το δίκτυο που θα κατασκευαστεί θα έχει 784 εισόδους, όσα δηλαδή και τα pixel της εικόνας, οι οποίες παίρνουν τιμές από 0 έως και 1. Ως έξοδο το δίκτυο θα έχει 10 classes που αντιπροσωπεύουν τα ψηφία από 0 έως 9, λαμβάνοντας τιμές από 0 έως 1. Αν ο i νευρώνας εξόδου λάβει τιμή κοντά στο 1 και οι υπόλοιποι νευρώνες εξόδου λάβουν τιμές κοντά στο 0 μετά την προς τα εμπρός διάδοση (forward pass), τότε η εικόνα που εισήλθε στο ΤΝΔ αναμένεται να αναπαριστά τον αριθμό i. Αυτή η λειτουργία αποτελεί στόχο της μηχανής – μοντέλου και αυτός είναι ο στόχος της εκπαίδευσης, η οποία πραγματοποιείται έχοντας ως δεδομένα χιλιάδες εικόνες χωρισμένες σε pixels συνοδευόμενες από τις τιμές στόχους 0 ή 1 των νευρώνων εξόδου του ΤΝΔ. Με κατάλληλη εκπαίδευση γίνεται εφικτή η ταξινόμηση των εισερχόμενων εικόνων στο αντίστοιχο ψηφίο που αναπαριστούν. Πάνω σε αυτή την αρχή λειτουργίας βασίζονται τα περισσότερα μοντέλα αναγνώρισης εικόνας.

3.3 Διαδικασία Εκπαίδευσης ΤΝΔ - Backpropagation

Η εκπαίδευση (training) ενός δικτύου αποτελεί ίσως το σημαντικότερο κομμάτι της μηχανικής μάθησης καθώς μέσω αυτής της διαδικασίας επιτυγχάνεται η ανάπτυξη μοντέλων τα οποία μπορούν να προσομοιώσουν οποιαδήποτε συνάρτηση. Σε επόμενη παράγραφο θα παρουσιαστεί θεώρημα το οποίο αποδεικνύει την ισχύ της τελευταίας φράσης. Σε αυτή την παράγραφό θα παρουσιαστεί ο τρόπος που πραγματοποιείται η εκπαίδευση μαθηματικά, θα παρουσιαστούν οι πιο συχνά εφαρμόσιμοι αλγόριθμοί και θα παρουσιαστεί η διαδικασία εκπαίδευσης σε ένα απλό ΤΝΔ για την καλύτερη κατανόηση. Στην συνέχεια, θα γίνει εμφανές πως η σωστή αντίληψη των αλγορίθμων εκπαίδευσης, είναι απαραίτητη για την εφαρμογή της μεθόδου PINNs, ούτως ώστε ο προγραμματιστής να είναι σε θέση να τροποποιήσει κατάλληλα τον κώδικά του σε περίπτωση ατελούς ή λανθασμένης εκπαίδευσης, φαινόμενο που είναι ιδιαίτερα συχνό. Είναι ιδιαίτερα σημαντικό να κατανοεί ποιες είναι οι πιθανές αιτίες μιας λανθασμένης εκπαίδευσης.

Ο όρος εκπαίδευση αναφέρεται στην κατάλληλη προσαρμογή των βαρών του ΤΝΔ ώστε μετά από κύκλους εκπαίδευσης (epochs) όπως ονομάζονται, το δίκτυο να προσεγγίζει όσο το δυνατόν καλύτερα την συμπεριφορά της συνάρτησης την οποία προσπαθεί να προσομοιώσει. Δείκτης για το αν το ΤΝΔ προσεγγίζει επιθυμητά την υπό μελέτη συνάρτηση, αποτελεί η συνάρτηση κόστους ή προτιμότερα η loss function. Στο πεδίο του machine υπάρχουν 3 κύρια είδη εκπαίδευσης τα οποία είναι η μάθηση με επίβλεψη (supervised learning), η μάθηση χωρίς επίβλεψη (unsupervised learning)[4] και η ενισχυτική μάθηση (reinforcement learning). Στην συγκεκριμένη εργασία μελετάται η μάθηση με επίβλεψη στα παραδείγματα αυτού του κεφαλαίου για κατανόηση, ως προθάλαμος για την εφαρμογή μιας παραλλαγής της η οποία ονομάζεται self-supervised learning και θα μελετηθεί εκτενώς στο κεφάλαιο που αφορά την μέθοδο PINNs. Κατά την μάθηση με επίβλεψη υπάρχουν γνωστά διανύσματα εισόδου με αντίστοιχα διανύσματα εξόδου - στόχου. Τα διανύσματα αυτά περιλαμβάνουν τιμές που εισέρχονται στους νευρώνες εισόδου του δικτύου και τιμές στόχους ως έξοδο του δικτύου για τις αντίστοιχες εισόδους. Συνήθως στα περισσότερα κλασικά μοντέλα ΤΝΔ από το set των δεδομένων κρατείται ένα μέρος για εκπαίδευση (training set) και ένα μέρος

[4] Η μάθηση χωρίς επίβλεψη εφαρμόζεται σε προβλήματα συσταδοποίησης (clustering), ενώ σε αρκετούς αλγόριθμους όπως ο K-means ο αριθμός των κλάσεων (class) που υπάρχουν στα δεδομένα είναι άγνωστος.

για τον έλεγχο (test set). Ένας κύκλος εκπαίδευσης (epoch) για να ολοκληρωθεί χρειάζεται να εισέλθουν στο ΤΝΔ όλα τα δεδομένα από το training set, είτε μεμονωμένα είτε χωρισμένα σε batches, πραγματοποιείται η προς τα εμπρός τροφοδότηση ώστε να υπολογιστούν οι τιμές εξόδου του δικτύου και στην συνέχεια υπολογίζεται η συνάρτηση κόστους (Loss function) η οποία εκφράζει έναν δείκτη για το πόσο διαφέρουν οι τιμές που εξέρχονται από το δίκτυο με τις τιμές στόχους που είναι γνωστές από τα δεδομένα. Ο δείκτης αυτός στις περισσότερες περιπτώσεις υπολογίζεται ως άθροισμα μέσων τετραγώνων των διαφορών της προσέγγισης και πραγματικής τιμής (mean square error). Στην συνέχεια τα βάρη μεταβάλλονται κατάλληλα προς την ελαχιστοποίηση της Loss function.

Η Loss function για το δίκτυο που χρησιμοποιήθηκε ως παράδειγμα στην ενότητα 3.2.3 είναι η εξής:

$$\text{Loss} = \frac{1}{3} * [(o1, prediction - o1, target)^2 + (o2, prediction - o2, target)^2 + (o2, prediction - o2, target)^2] \quad (18)$$

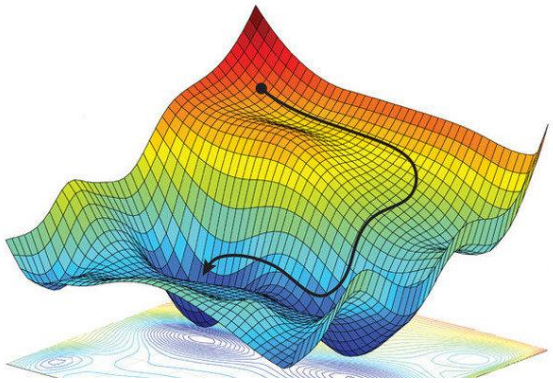
oi,prediction : Η τιμή που προκύπτει στον i νευρώνα εξόδου του δικτύου μετά την προς τα εμπρός τροφοδότηση.

oi,target = Η τιμή εξόδου – στόχου που έχει ο i νευρώνας εξόδου και είναι γνωστή από το set των δεδομένων για αντίστοιχη είσοδο.

Ως γενική μορφή του τύπου της Loss function είναι η ακόλουθη:

$$\text{Loss} = \frac{1}{m} \sum_{i=1}^{m} (oi, prediction - oi, target)^2 \quad (19)$$

Κατά την εκπαίδευση του δικτύου οι τιμές εισόδου και εξόδου λαμβάνονται ως δεδομένα, ενώ όλα τα βάρη και τα biases του δικτύου λαμβάνονται ως μεταβλητές οι οποίες αναφέρονται και ως trainable parameters. Προηγουμένως στην παράγραφο 3.2.3, υπολογίζοντας τις εξόδους του δικτύου με προς τα εμπρός διάδοση, προκύπτουν τα oi,prediction τα οποία στην έκφρασή τους περιλαμβάνουν όλα τα βάρη και biases. *Κατά συνέπεια η Loss function είναι μια συνάρτηση όλων των βαρών και των biases*. Η εκπαίδευση με μαθηματικούς όρους αποτελεί ένα πρόβλημα βελτιστοποίησης - ελαχιστοποίησης (optimization problem) της συνάρτησης κόστους Loss function.



Εικόνα 3.3.1 (Παράδειγμα προβλήματος βελτιστοποίησης - optimization)

Στην Εικόνα 3.3.1 φαίνεται ένα χαρακτηριστικό παράδειγμα προβλήματος ελαχιστοποίησης για μια μη κυρτή συνάρτηση 2 μεταβλητών που αναπαρίσταται σε ένα τρισδιάστατο γράφημα. Κατά την εκπαίδευση ενός ΤΝΔ το αντίστοιχο πρόβλημα έγκειται στην ελαχιστοποίηση μιας συνάρτησης χιλιάδων ή δεκάδων

χιλιάδων μεταβλητών (trainable parameters = weights and biases) και δεν μπορεί να αναπαρασταθεί σε γράφημα.

Κατά την ελαχιστοποίηση μιας συνάρτησης χρειάζεται ο υπολογισμός των μερικών παραγώγων της συνάρτησης ως προς όλες τις μεταβλητές της. Η πολυπλοκότητα των νευρωνικών δικτύων καθιστά ανέφικτο τον απευθείας υπολογισμό όλων των μερικών παραγώγων της Loss function ως προς τις trainable parameters αναλυτικά. Ο αλγόριθμος backpropagation ουσιαστικά υπολογίζει όλες τις απαραίτητες παραγώγους, με τον κανόνα της αλυσιδωτής παραγώγισης, ξεκινώντας από το τελευταίο layer και προχωρώντας προς τα πίσω μέχρι το input layer . Τονίζεται ότι ο αλγόριθμος backpropagation αφορά μόνο τον υπολογισμό των παραγώγων και είναι τμήμα της διαδικασίας εκπαίδευσης. Σε επόμενη παράγραφο ακολουθεί ένα απλό παράδειγμα για καλύτερη κατανόηση.

Τα γενικά βήματα της διαδικασίας εκπαίδευσης είναι τα ακόλουθα:

1. Forward Pass – υπολογισμός των τιμών που εξέρχονται από το ΤΝΔ o_i για ένα ζευγάρι εισόδων εξόδων.
2. Υπολογισμός Loss function.
3. Υπολογισμός των μερικών παραγώγων της Loss function ως προς όλες τις trainable parameters ξεκινώντας από το τελευταίο layer και συνεχίζοντας διαδοχικά προς τα πίσω (backpropagation).
4. Μεταβολή των trainable parameters προς την κατεύθυνση που ελαχιστοποιεί την Loss function.
5. Επανάληψη των βημάτων 1, 2, 3 για όλα τα δεδομένα (training set).
6. Επανάληψη των βημάτων 1, 2, 3, 4 για όσους κύκλους εκπαίδευσης (epochs) έχουν οριστεί.

Το γεγονός ότι για να ολοκληρωθεί ένας κύκλος εκπαίδευσης χρειάζεται ο υπολογισμός της loss function και η μεταβολή των παραμέτρων για κάθε ένα ξεχωριστά ζευγάρι εισόδων εξόδων, επιφέρει μεγάλο υπολογιστικό κόστος και προκαλεί μεγάλη καθυστέρηση στην σύγκλιση του αλγορίθμου. Για τον λόγο αυτό, συχνά σε εφαρμογές, πραγματοποιείται ένας διαχωρισμός του training set σε επιμέρους τμήματα που ονομάζονται batches. Κατά αυτόν τον τρόπο μειώνονται οι επαναλήψεις του βήματος 4 καθώς τα βήματα 1, 2, 3 δεν εκτελούνται για κάθε ένα ζευγάρι εισόδων εξόδων αλλά για κάθε batch. Στην συνέχεια ακολουθεί μια σύντομη περιγραφή των κυριότερων αλγορίθμων βελτιστοποίησης που χρησιμοποιούνται στην εκπαίδευση ΤΝΔ οι οποίοι είναι ο αλγόριθμος Gradient Decent, ο αλγόριθμος L-BFGS και ο αλγόριθμος Adam.

<u>Αλγόριθμος Gradient Decent</u>

Ο συγκεκριμένος αλγόριθμος προσφέρει μια μέθοδο ελαχιστοποίησης της συνάρτησης $L(w_i,b_i)$ με παραμέτρους τα βάρη w_i και τα biases b_i του δικτύου, τις οποίες μεταβάλει προς την αντίθετη κατεύθυνση της κλήσης δηλαδή της μερικής παραγώγου της συνάρτησης ως προς την εκάστοτε παράμετρο (Ruder, 2017). Νοητικά γίνεται πιο εύκολα κατανοητός αυτός ο αλγόριθμος αν σκεφτεί κανείς έναν άνθρωπο που βρίσκεται σε ένα λόφο ενώ δεν έχει καθόλου ορατότητα και προσπαθεί να κατέβει στο χαμηλότερο σημείο. Αυτό που μπορεί να κάνει είναι να ελέγξει γύρω του προς τα πού είναι η μεγαλύτερη ανοδική κλήση του εδάφους και να προχωρήσει ένα βήμα προς την αντίθετη κατεύθυνση. Στην συνέχεια το μόνο που έχει να κάνει, είναι να επαναλάβει την παραπάνω διαδικασία έως ότου φτάσει στο επιθυμητό ύψος.

Ακολουθεί η παρουσίαση των $3^{ων}$ παραλλαγών του αλγορίθμου. Ως θ ορίζεται το διάνυσμα που περιλαμβάνει όλες τις μεταβλητές του ΤΝΔ (weights, biases) και ως L(θ) ορίζεται η συνάρτηση κόστους (Loss function).

- Stochastic Gradient Decent

  Στον αλγόριθμο Stochastic Gradient Decent, σε έναν κύκλο εκπαίδευσης, εισέρχονται ένα - ένα τα δεδομένα εισόδου στο δίκτυο, υπολογίζεται η Loss function, υπολογίζονται οι μερικές παράγωγοι της ως προς όλες της trainable parameters και στην συνέχεια αυτές μεταβάλλονται με βάση τον ακόλουθο τύπο:

  $$\theta' = \theta - \eta * \nabla_\theta L(\theta; x_i; y_i) \quad (20)$$

  Ως 'η' ορίζεται το βήμα του αλγορίθμου ή ρυθμός μάθησης (learning rate) και ως $\nabla_\theta L(\theta)$ ορίζεται το εξής διάνυσμα (Ιακωμβιανή):

  $$\nabla_\theta L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \theta i} \\ \vdots \\ \frac{\partial L}{\partial \theta n} \end{bmatrix} \quad (21)$$

  Επομένως τα βήματα είναι τα εξής:

  1. Forward Pass – υπολογισμός των τιμών που εξέρχονται από το ΤΝΔ o_i για ένα ζευγάρι εισόδων εξόδων x_i, $o_{i,target}$.
  2. Υπολογισμός Loss function και $\nabla_\theta L(\theta; x_i; o_{i,target})$ (22)
  3. Μεταβολή των βαρών κατά $\theta' = \theta - \eta \nabla_\theta L(\theta; x_i; o_{i,target})$ (23)
  4. Επανάληψη των βημάτων 1, 2, 3 για όλα τα δεδομένα.
  5. Επανάληψη των βημάτων 1, 2, 3, 4 για όσους κύκλους εκπαίδευσης (epochs) έχουν οριστεί.

  Χαρακτηρίζεται από πολύ αργή σύγκλιση αλλά από ιδιαίτερα καλή τελική ακρίβεια.

- Batch Gradient Decent

  Στην συγκεκριμένη παραλλαγή, τα δεδομένα εκπαίδευσης λαμβάνονται υπόψη όλα μαζί και όχι ξεχωριστά όπως στην περίπτωση του Stochastic Gradient Decent, ενώ η μεταβολή των παραμέτρων γίνεται μια φορά σε κάθε κύκλο εκπαίδευσης και όχι σε κάθε training example. Ουσιαστικά κάθε παράμετρος θ_i μεταβάλλεται με βάση τον μέσο όρο των $\nabla_\theta L(\theta; x_i; o_{i,target})$ που υπολογίζονται για κάθε training example.

  Επομένως τα βήματα είναι τα εξής:

  1. Forward Pass – υπολογισμός των τιμών που εξέρχονται από το ΤΝΔ o_i για όλα ζευγάρια εισόδων εξόδων x_i, $o_{i,target}$.
  2. Υπολογισμός Loss function και $\nabla_\theta L(\theta; x_i; o_{i,target})$ (24)
  3. Υπολογισμός μέσου όρου και $\nabla_\theta L(\theta)$ όλων των ζευγαριών.
  4. Μεταβολή των βαρών κατά $\theta' = \theta - \eta \nabla_\theta L(\theta)$ (25)
  5. Επανάληψη των βημάτων 1, 2, 3, 4 για όσους κύκλους εκπαίδευσης (epochs) έχουν οριστεί.

  Χαρακτηρίζεται από πολύ μικρότερο χρόνο εκτέλεσης του κάθε κύκλου εκπαίδευσης σε σχέση με τον Stochastic Gradient Decent αλλά υστερεί στην ακρίβεια ανά κύκλο.

- Mini batch Gradient Decent
  Σε αυτή τη περίπτωση το training set χωρίζεται σε mini batches. Είναι ουσιαστικά μια ενδιάμεση λύση μεταξύ των δύο προηγούμενων.

  Στην προκειμένη περίπτωση τα βήματα έχουν ως εξής:
  1. Forward Pass – υπολογισμός των τιμών που εξέρχονται από το ΤΝΔ $o_{i,prediction}$ για όλα ζευγάρια εισόδων εξόδων x_i, $o_{i,target}$ σε ένα batch.
  2. Υπολογισμός Loss function και $\nabla_\theta L(\theta; x_i; o_{i,target})$ (26)
  3. Υπολογισμός μέσου όρου και $\nabla_\theta L(\theta)$ όλων των ζευγαριών του batch.
  4. Μεταβολή των βαρών κατά $\theta' = \theta - \eta \nabla_\theta L(\theta)$ (27)
  5. Επανάληψη βημάτων 1, 2, 3, 4 για όσα batch έχει χωριστεί το training set.
  6. Επανάληψη των βημάτων 1, 2, 3, 4, 5 για όσους κύκλους εκπαίδευσης (epochs) έχουν οριστεί.

  Πρόκειται για την βέλτιστη παραλλαγή εκ των $3^{ων}$ που αναφέρονται λόγω του ότι με κατάλληλη επιλογή batch size επιτυγχάνεται ο συνδυασμός των πλεονεκτημάτων των προηγούμενων αλγορίθμων. Επειδή το training set χωρίζεται σε batches πραγματοποιείται παραπάνω από μια φορά ενημέρωση (update) των παραμέτρων σε έναν κύκλο, ενώ λόγω του ότι σε κάθε batch περιέχονται παραπάνω από ένα training example, επιτυγχάνεται σχετικά ταχεία σύγκλιση του αλγορίθμου.

Αλγόριθμος L-BFGS

Ο L-BFGS αλγόριθμος ανοίκει στην οικογένεια των μεθόδων quasi – Newton ο οποίος αναπτύχθηκε από τους Broyden, Fletcher, Goldfard και Shanno. Η μέθοδος Newton, για προβλήματα ελαχιστοποίησης απαιτεί την ευρεση του Εσσιανού (Hessian) G πίνακα της αντικειμενικής συνάρτησης δηλαδή απαιτεί τον υπολογισμό δεύτερων παραγώγων για όλες τις παραμέτρους, το οποίο είναι υπολογιστικά απαγορευτικό σε προβλήματα deep learning. Αντιθέτως οι μέθοδοι quasi – Newton προσεγγίζουν τον αντίστροφο Εσσιανό πίνακα με έναν συμμετρικό, θετικά ορισμένο πίνακα H^k ο οποίος μεταβάλλεται σε κάθε επανάληψη. Ο αλγόριθμος L-BFGS (limited –memory BFGS algorithm) ουσιαστικά έχει ίδια λογική υπολογισμού με τον BFGS αλλά χρησιμοποιεί πιο περιορισμένη υπολογιστική μνήμη κατά την εκτέλεση. Τα βήματα του αλγορίθμου είναι τα ακόλουθα:

1. Αρχικοποίηση πίνακα H^0 (για παράδειγμα $H^0 = I$)
2. $p^k = - H^k * \nabla_\theta L(\theta^k)$ (28)
3. $\theta^{k+1} = \theta^k + a * p^k$ όπου το βέλτιστο βήμα $a < 0$ επιλέγεται από μια μέθοδο αναζήτησης επί γραμμής (Line search method) (Jiang, Byrd, Eskow, & Schnabel, 2004) (29)
4. $s^k = \theta^{k+1} - \theta^k$ (30)
5. $y^k = \nabla_\theta L(\theta^{k+1}) - \nabla_\theta L(\theta^k)$ (31)
6. $\rho = \frac{1}{y^{k\,T} * s^k}$ (32)
7. $V = I - \rho * y^k * s^T$ (33)
8. $H^{k+1} = V^T * H * V + \rho * s^k * s^{k\,T}$ (34)

Αλγόριθμος Adam (Adaptive moment estimation)

Η μέθοδος αυτή προτάθηκε από τους (Kingma & Lei Ba, 2015) και έχει το πλεονέκτημα ότι υπολογίζει ξεχωριστό ρυθμό μάθησης (learning rate), για κάθε επανάληψη. Ουσιαστικά συνδυάζει τα πλεονεκτήματα των μεθόδων AdaGrad (Adaptive Gradient Algorithm) και RMSProp (Root Mean Square Propagation). Ακολουθούν τα βήματα του αλγορίθμου σε μορφή ψευδοκώδικα.

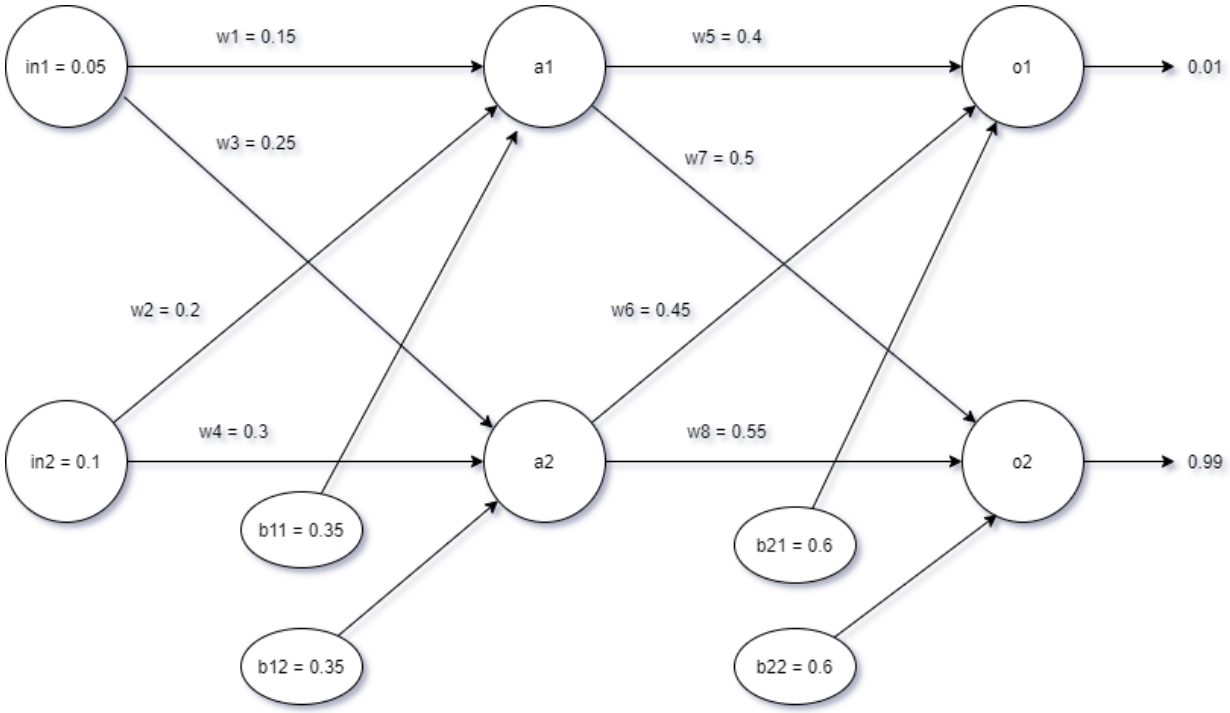
Αρχικοποίηση: a = 0.001 (initial step size), $\hat{\varepsilon} = 10^{-8}$
Ορισμός: β1 = 0.09, β2 = 0.999 (Exponential decay rates for the moment estimates)
Ορσμός: L(θ): Αντικειμενική συνάρτηση Loss function
Αρχικοποίηση βαρών: θ_0: Αρχικά βάρη με τυχαίους αριθμούς
m0 = 0 (Αρχικοποίηση 1st moment vector)
v0 = 0 (Αρχικοποίηση 2nd moment vector)
t = 0 (Αρχικοποίηση αρίθμησης επαναλήψεων)
while (ΣΥΝΘΗΚΗ ΣΥΓΚΛΙΣΗΣ) **do**
t = t + 1
$g_t = \nabla_\theta L(\theta^k)$ (Υπολογισμός μερικών παραγώγων της Loss function) (35)
$m_t = \beta1 \cdot m(t-1) + (1 - \beta1) \cdot g_t$ (Τροποποίηση του 1st moment vector) (36)
$u_t = \beta2 \cdot u(t-1) + (1 - \beta2) \cdot g_t^2$ (Τροποποίηση του 2nd raw moment vector) (37)
$a_t = a * \frac{\sqrt{1-\beta2^t}}{1-\beta1^t}$ (Τροποποίηση του βήματος a) (38)

$\theta t = \theta(t-1) - \frac{at * mt}{\sqrt{ut} + \hat{\varepsilon}}$ (Update parameters) (39)
end while
return θt (Επιστροφή των νέων παραμέτρων)

Οι αλγόριθμοι ελαχιστοποίησης L-BFGS και Adam θα χρησιμοποιηθούν για την εφαρμογή της μεθόδου PINNs σε προβλήματα στατικής τα οποία θα παρουσιαστούν αναλυτικά σε επόμενο κεφάλαιο.

3.4 Παράδειγμα Αλγορίθμου Εκπαίδευσης και backpropagation

Σε αυτήν την παράγραφο παρατίθεται ένα απλό παράδειγμα ΤΝΔ στο οποίο θα εφαρμοστεί ο αλγόριθμος Stochastic Gradient Decent για την εκπαίδευσή του. Η ενότητα αυτή έχει σκοπό την παράθεση των πράξεων που γίνονται κατά τον αλγόριθμο για την όσο το δυνατόν καλύτερη κατανόηση της διαδικασίας backpropagation και τροποποίησης των παραμέτρων. Ο αλγόριθμος Gradient Decent δεν χρησιμοποιήθηκε καθόλου για την μέθοδο PINNs, όμως μέσω ενός παραδείγματος θα βοηθήσει στην καλύτερη εξοικείωση για το πώς λειτουργούν τα ΤΝΔ στην πράξη, ούτως ώστε στην μέθοδο PINNs που είναι πιο απαιτητική, να υπάρχει το κατάλληλο υπόβαθρο. Παρακάτω θα εφαρμοστούν τα βήματα της τροποποίησης των βαρών τόσο στο χαρτί όσο και προγραμματιστικά.



Εικόνα 3.4.1 (Δίκτυο με 2 εισόδους, 1 hidden layer και 2 εξόδους – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net)

Στο παραπάνω δίκτυο υπάρχει μόνο ένα training example με x = [0.05 0.1] και έξοδο y_target = [0.01 0.99]. Χρησιμοποιείται η σιγμοειδής συνάρτηση ως συνάρτηση ενεργοποίησης.

Forward Pass

Οι ονομασίες των μεταβλητών που χρησιμοποιήθηκαν είναι οι εξής:

in1, in2: είσοδοι στο ΤΝΔ

neta1, neta2: τιμές που λαμβάνουν οι νευρώνες a1, a2 πριν την επιβολή της συνάρτηση ενεργοποίησης

outa1,outa2: τιμές που λαμβάνουν οι νευρώνες a1, a2 μετά την επιβολή της συνάρτησης ενεργοποίησης

Οι αντίστοιχες ερμηνείες δίνονται και στις μεταβλητές neto1, neto2 και outo1, outo2.

neta1 = in1 * w1 + in2 * w2 + b11 = 0.3775 (40)

outa1 = sigmoid(a1) = $\frac{1}{1+e^{-neta1}}$ = **0.593269992 = outa1** (41)

Κατά αντίστοιχο τρόπο για τον δεύτερο νευρώνα του hidden layer:

outa2 = 0.596884378

neto1 = outa1 * w5 + outa2 * w6 + b21 = 1.105905967 (42)

neto2 = outa1 *w7 + outa2 * w8 + b22 = 1.224921404 (43)

outo1 = $\frac{1}{1+e^{-neto1}}$ = 0.75136507, outo2 = $\frac{1}{1+e^{-neto2}}$ 0.772928465 (44)

Η παραπάνω διαδικασία μπορεί να γραφτεί στην γλώσσα Python σε μορφή πινάκων όπως φαίνεται στην παρακάτω εικόνα:

```python
class Neural_Network:
    def __init__(self, inputs, outputs, num_hidden):
        self.inputs = inputs
        self.outputs = outputs
        self.num_hidden = num_hidden         # num_hidden = number of hidden layers

        self.b1 = 0.35
        self.b2 = 0.6
        self.bias = [self.b1, self.b2]
        self.w1 = tf.constant([[0.15, 0.25], [0.2, 0.3]])
        self.w2 = tf.constant([[0.4, 0.5], [0.45, 0.55]])
        self.weights = [self.w1, self.w2]


    def forward_pass(self):
        activations = self.inputs
        i = 0
        for w in self.weights:
            net = tf.matmul(activations, w) + tf.constant([self.bias[i], self.bias[i]])
            activations = tf.sigmoid(net)
            i += 1
        return activations
```

Εικόνα 3.4.2 (Υλοποίηση αλγόριθμουForward pass σε Python)

Loss function = $\frac{1}{2}\sum_{i=1}^{2}(outoi - yi)^2 = \frac{1}{2}(\text{outo1} - \text{y1})^2 + \frac{1}{2}(\text{outo2} - \text{y2})^2$ =>

Loss function = 0.2983711

Για την εκτέλεση του backpropagation με πράξεις θα χρειαστεί ο κανόνας αλυσιδωτής παραγώγισης ο οποίος φαίνεται στην συνέχεια σε όλους τους υπολογισμούς. Όπως είχε αναφερθεί σε προηγούμενη παράγραφο, ο αλγόριθμος Gradient Decent μεταβάλλει την κάθε παράμετρο σύμφωνα με την παρακάτω σχέση. Χρησιμοποιείται learning rate = 0.5:

$$wi^{k+1} = wi^{k} - \eta * \frac{\partial L}{\partial wik} \quad (45)$$

Ο υπολογισμός των απαραίτητων παραγώγων για την χρήση της σχέσης (45), γίνεται από το τελευταίο layer και συνεχίζει προς τα πίσω ως εξής:

$$w5' = w5 - \eta * \frac{\partial L}{\partial w5} \quad (46)$$

$$\frac{\partial L}{\partial W5} = \frac{\partial L}{\partial outo1}\frac{\partial outo1}{\partial neto1}\frac{\partial neto1}{\partial w5} \quad (47)$$

- $\frac{\partial L}{\partial outo1} = \text{outo1} - \text{y1} = 0.74136507$ (48)
- $\frac{\partial outo1}{\partial neto1} = \frac{\partial}{\partial neto1}\left[\frac{1}{1+e^{-neto1}}\right] = \text{outo1} * (1 - \text{outo1}) = 0.186815602$ [5] (49)

[5] Η παράγωγος της σιγμοειδής συνάρτησης προκύπτει ως $\frac{ds(x)}{dx} = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = s(x) * (1 - s(x))$

- $\frac{\partial neto1}{\partial w5} = \frac{\partial}{\partial w5}$ (outa1 *w5 + outa2 * w6 + b21) = 0.593269992 (50)

Συνεπώς $\frac{\partial L}{\partial w5}$ = 0.082167041 και w5' = 0.4 – 0.5 * 0.082167041 = > **w5' = 0.35891648**

Με τον ίδιο ακριβώς τρόπο υπολογίζονται και οι νέες τιμές των υπόλοιπων βαρών του τελευταίου layer οπού προκύπτουν τα εξής αποτελέσματα:

w6' = 0.40866616, w7' = 0.5113013, w8' =0.56137013

Στην συνέχεια ο αλγόριθμος προχωράει στον υπολογισμό των νέων βαρών του ακριβώς προηγούμενου layer. Σε αυτό το σημείο αξίζει να σημειωθεί ότι κατά την μετάβαση σε προηγούμενο layer δεν ξανά υπολογίζεται η Loss function διότι οι υπολογισμοί πραγματοποιούνται με τα αρχικά βάρη ή γενικά με τα βάρη της προηγούμενης επανάληψης του αλγορίθμου. Τα βάρη ενημερώνονται πάντα όλα μαζί αφού πρώτα έχουν υπολογιστεί όλες οι μερικές παράγωγοι της Loss function προς τις trainable parameters. Κατά συνέπεια οι υπολογισμοί έχουν ως εξής:

$w1' = w1 - \eta * \frac{\partial L}{\partial w1}$ (51)

$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial outa1} \frac{\partial outa1}{\partial neta1} \frac{\partial neta1}{\partial w1}$ (52)

- $\frac{\partial L}{\partial outa1} = \frac{\partial L1}{\partial outa1} + \frac{\partial L2}{\partial outa1}$ [6] (53)

$\frac{\partial L1}{\partial outa1} = \frac{\partial L1}{\partial neto1} \frac{\partial neto1}{\partial outa1}$ (54)

$\frac{\partial L1}{\partial neto1} = \frac{\partial L1}{\partial outo1} \frac{\partial outo1}{\partial neto1}$ = 0.74136507 * 0.186815602 = 0.138498561 (55)

$\frac{\partial neto1}{\partial outa1} = \frac{\partial}{\partial outa1}$(outa1 *w5 + outa2 * w6 + b21) = 0.4 οπότε, (56)

$\frac{\partial L1}{\partial outa1}$ = 0.055399424

$\frac{\partial L2}{\partial outa1} = \frac{\partial L2}{\partial neto2} \frac{\partial neto2}{\partial outa1}$ (57)

$\frac{\partial L2}{\partial neto2} = \frac{\partial L2}{\partial outo2} \frac{\partial outo2}{\partial neto2}$ = - 0.038098236 (58)

$\frac{\partial neto2}{\partial outa1} = \frac{\partial}{\partial outa1}$(outa1 *w7 + outa2 * w8 + b22) = 0.5 (59)

Άρα $\frac{\partial L2}{\partial outa1}$ = -0.019049119 και κατά συνέπεια

$\frac{\partial L}{\partial outa1} = \frac{\partial L1}{\partial outa1} + \frac{\partial L2}{\partial outa1}$ = 0.036350306

[6] Η συνάρτηση Loss function L(wi) θεωρείται ως άθροισμα των 2 σφαλμάτων των 2 νευρώνων εξόδου. Άρα L = $\frac{1}{2}$ (outo1 – y1)^2 + $\frac{1}{2}$ (outo2 – y2)^2 = L1 +L2

- $\frac{\partial outa1}{\partial neta1} = \frac{\partial}{\partial neta1}\left(\frac{1}{1+e^{-neta1}}\right)$ = outa1 * (1 – outa1) = 0.241300709 (60)

- $\frac{\partial neta1}{\partial w1} = \frac{\partial}{\partial w1}$ (in1 * w1 + in2 * w2 + b11) = 0.5 (61)

Με βάση τους παραπάνω υπολογισμούς προκύπτει $\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial outa1}\frac{\partial outa1}{\partial neta1}\frac{\partial neta1}{\partial w1}$ = 0.036350306 * 0.241300709 * 0.5 = > $\frac{\partial L}{\partial w1}$ = 0.000438568

Επομένως επιστρέφοντας στην σχέση w1’ = w1 – η * $\frac{\partial L}{\partial w1}$ προκύπτει

w1’ = 0.14978072

Με την ίδια ακριβώς μεθοδολογία τα νέα βάρη λαμβάνουν τιμές **w2’ = 0.19956143, w3’ = 0.24975115, w4’ = 0.2995023.**

Το παραπάνω μικρό παράδειγμα έδωσε μια πρώτη εντύπωση για το πόσο υπολογιστικά κοστοβόρα διαδικασία είναι ο υπολογισμός των μερικών παραγώγων, ειδικά όταν το δίκτυο είναι κατά πολύ μεγαλύτερων διαστάσεων. Προγραμματιστικά θα ήταν πρακτικά ανέφικτη η εκτέλεση του αλγόριθμου backpropagation σε λογικά πλαίσια χρόνου, εάν δεν υπήρχε ένας τρόπος να υπολογίζονται οι παράγωγοι πιο αποδοτικά. Η λύση για αυτό το πρόβλημα προήλθε από την βιβλιοθήκη Tensorflow που περιέχεται στην Python, η οποία μεταξύ άλλων παρέχει την δυνατότητα ενός ιδιαίτερα ταχύτερου τρόπου υπολογισμού παραγώγων ο οποίος καλείται Automatic Differentiation. Ο τρόπος λειτουργίας του θα παρατεθεί αναλυτικά στο επόμενο κεφάλαιο που αφορά την μέθοδο PINNs, καθώς εκεί θα χρησιμοποιηθεί εκτενώς, όπως επίσης και άλλες συναρτήσεις της βιβλιοθήκης Tensorflow.

Στην συνέχεια ακολουθεί η παράθεση τμημάτων κώδικα που αναπτύχθηκε στην Python με σκοπό την καλύτερη κατανόηση του παραδείγματος και την επαλήθευση των παραπάνω αποτελεσμάτων. Στην Εικόνα 3.4.3 φαίνεται η συνάρτηση train μέσω της οποίας πραγματοποιούνται οι επαναλήψεις των κύκλων εκπαίδευσης (epochs). Με την εντολή grads = tape.jacobian(L, self.weight[i]) υπολογίζεται η Ιακωμβιανή μήτρα ή Jacobian matrix που περιλαμβάνει όλες τις μερικές παραγώγους της L (Loss function) ως προς κάθε στοιχείο του πίνακα weight[i]. Όπως τονίστηκε και πιο πάνω, τα βάρη δεν ενημερώνονται εάν πρώτα δεν υπολογιστούν όλες οι μερικές παράγωγοι της Loss function ως προς όλα τα βάρη. Για τον λόγο αυτό στην Εικόνα 3.4.3, η εντολή για την ενημέρωση των βαρών βρίσκεται εκτός της δομής επανάληψης for loop με την οποία υπολογίζονται οι παράγωγοι. Αξίζει να τονιστεί ότι όταν το δίκτυο διαθέτει biases, θεωρούνται και αυτά trainable parameters και κατά συνέπεια μεταβάλλονται και αυτά κατά την εκπαίδευση. Στο παραπάνω παράδειγμα έγινε παράληψη την ενημέρωσης των biases λόγω συντομίας. Τέλος, στην εικόνα 3.4.4 φαίνονται τα νέα βάρη που προέκυψαν μετά από τον 1ο κύκλο εκπαίδευσης τα οποία επαληθεύουν τους παραπάνω υπολογισμούς, ενώ στην εικόνα 3.4.5 γίνεται εμφανής η βελτίωση της Loss function σε σφάλμα 0.29102775 από 0.2983711 μετά από τον 1ο κύκλο.

```python
def train(self, epoch_number):
    print("--START TRAINING--")
    lr = 0.5

    for epoch in range(epoch_number):
        print("==========================================================================")
        print(f"EPOCH {epoch + 1}/{epoch_number}")
        grad_L_w = []
        for i in reversed(range(len(self.weights))):
            with tf.GradientTape() as tape:
                tape.watch(self.weights[i])
                aproximation = self.forward_pass()
                L = (1 / 2) * (self.outputs[0][0] - aproximation[0][0]) ** 2 + \
                    1 / 2 * (self.outputs[0][1] - aproximation[0][1]) ** 2
            grads = tape.jacobian(L, self.weights[i])
            grad_L_w.append(grads)

        # UPGRADE THE WEIGHTS
        for i in reversed(range(len(self.weights))):
            self.weights[i] -= lr * grad_L_w[len(self.weights)-1-i]
```

Εικόνα 3.4.3 (Ανάπτυξη αλγορίθμου Gradient Decent χωρίς την χρήση έτοιμης συνάρτησης εκπαίδευσης)

```
NN_FROM_SCRATCH_USING_CLASSES
C:\Users\user\miniconda3\envs\pythonProject2\python.exe C:/Users/user/PycharmProjects/NN_FROM_SCRATCH/
--START TRAINING--
==========================================================================================================
EPOCH 1/100
NEURAL NETWORK OUTPUT =
tf.Tensor([[0.75136507 0.7729285 ]], shape=(1, 2), dtype=float32)
Loss function =
tf.Tensor(0.2983711, shape=(), dtype=float32)
UPDATED WEIGHTS = [<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[0.14978072, 0.24975115],
       [0.19956143, 0.2995023 ]], dtype=float32)>, <tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[0.3589165 , 0.5113013 ],
       [0.40866616, 0.56137013]], dtype=float32)>]
==========================================================================================================
```

Εικόνα 3.4.4 (Αποτελέσματα αλγορίθμου Gradient Decent μετά από έναν κύκλο εκπαίδευσης)

```
EPOCH 2/100
NEURAL NETWORK OUTPUT =
tf.Tensor([[0.7420881 0.775285 ]], shape=(1, 2), dtype=float32)
Loss function =
tf.Tensor(0.29102775, shape=(), dtype=float32)
```

Εικόνα 3.4.5 (Μείωση της Loss function μετά τον πρώτο κύκλο εκπαίδευσης σε 0.29102775)

Κεφάλαιο 4 - Physics Informed Neural Networks (PINNs)

Μετά από την κατανόηση του βασικού τρόπου λειτουργίας των Τεχνητών Νευρωνικών Δικτύων, μπορεί να γίνει το επόμενο βήμα. Το κεφάλαιο αυτό αποτελεί τον πυρήνα της εργασίας καθώς σε αυτό, εξηγείται μαθηματικά και προγραμματιστικά ο τρόπος λειτουργίας της μεθόδου PINNs. Παράλληλα εξηγείται ο τρόπος λειτουργίας της παραγώγισης με Automatic differentiation και άλλων δυνατοτήτων της βιβλιοθήκης Tensorflow για deep learning καθώς επίσης θα οριστούν τα προβλήματα στατικής μηχανικής, δηλαδή οι διαφορικές εξισώσεις οι οποίες προορίζονται για επίλυση με την μέθοδο PINNs. Αρχικά όμως κρίνεται σημαντικό να αναφερθεί η ιστορία της μεθόδου από την έναρξή της ως πρότυπη ιδέα μέχρι την τωρινή κατάσταση, όπως επίσης είναι σημαντικό να αναφερθούν οι ερευνητές που έχουν ασχοληθεί με το θέμα τα προηγούμενα χρόνια.

4.1 Από τις πρώτες δημοσιεύσεις μέχρι και σήμερα

Στα σύγχρονα συγγράμματα ως η πρώτη ολοκληρωμένη πρόταση της χρήσης ΤΝΔ για την επίλυση διαφορικών εξισώσεων, συνοδευόμενη από υλοποίηση, αναγνωρίζεται η ως η εργασία με τίτλο ‘Artificial Neural Networks for Solving Ordinary and Partial Differential Equations’ (Lagaris, Likas, & Fotiadis, 1997), από το τμήμα Επιστήμης Ηλεκτρονικών Υπολογιστών του Πανεπιστημίου Ιωαννίνων. Βέβαια, είναι σημαντικό να αναφερθεί ότι ή αρχική ιδέα της προσπάθειας να χρησιμοποιηθούν τα νευρωνικά δίκτυα για την επίλυση διαφορικών εξισώσεων είχε προταθεί από τις αρχές τις δεκαετίας του 1990 μέσα από θεωρητικές εργασίες όπου ορισμένες από αυτές αναφέρονται στην βιβλιογραφία της παραπάνω. Ενδεικτικά ορισμένες από αυτές είναι η εργασία με τίτλο ‘Neural algorithms for solving differential equations’ (Lee & Kang, 1990) και ‘The numerical solution of Linear Ordinary Differential Equations by Feedforward Neural networks’ (Meade Jr & Fernadez, 1994). Οι παραπάνω ερευνητές που αναφέρθηκαν προέρχονταν από επιστημονικά πεδία κυρίως των εφαρμοσμένων μαθηματικών και της επιστήμης υπολογιστών. Η μέθοδος εισήχθη στο πεδίο της μηχανικής και συγκεκριμένα στο πεδίο της ανάλυσης κατασκευών μέσα από την εργασία με τίτλο ‘A neural network approach to the modelling, calculation and identification of semi-rigid connections in steel structures’ (Stavroulakis, Avdelas, Abdalla, & Panagiotopoulos, 1997).

Κατά την δεκαετία του 2000 δεν παρατηρήθηκε κάποια ιδιαίτερη εξέλιξη πάνω στο θέμα ούτε κάποια εφαρμογή. Μια χαρακτηριστική εργασία η οποία παρουσιάζει την μέθοδο PINNs με την μορφή που έχει σήμερα, αποτελεί η εργασία με τίτλο ‘Physics Informed Deep Learning (Part I): Data – driven Solutions of Nonlinear Partial Differential Equations’ (Raissi, Perdikaris, & Karniadakis, 2017) στην οποία ορίζονται η αρχές της μεθόδου, αναλύεται η προγραμματιστική υλοποίηση της και εφαρμόζεται στις μερικές διαφορικές εξισώσεις Burger’s Equation και Schrödinger Equation. Η εργασία αυτή αποτελεί έναν πολύ καλό οδηγό για κάποιον που επιθυμεί να ξεκινήσει την ενασχόλησή του με το θέμα. Οι παραπάνω συγγραφείς έχουν συμμετάσχει στη δημοσίευση αρκετών εργασιών τα τελευταία χρόνια, όπως επίσης φαίνεται ότι ασχολούνται συστηματικά με την μέθοδο και τις δυνατότητές της, ενώ δηλώνουν ιδιαίτερα αισιόδοξοι για την πορεία της. Πέρα όμως από τους ερευνητές που συναντάει κανείς πιο συχνά κατά την μελέτη της μεθόδου, παράλληλα έχουν δημοσιευτεί και ορισμένες επιπλέον εργασίες. Ενδεικτικά μια ακόμα εργασία πάνω στο θέμα έχει τίτλο ‘IDRLnet: A Physics-Informed Neural Network Library’ (Peng, και συν., 2021) στην οποία μπορεί ένας αναγνώστης να βρει πιο λεπτομερείς πληροφορίες για την υλοποίηση της μεθόδου.

Η μελέτη της μεθόδου φαίνεται να παρουσιάζει ιδιαίτερη ακμή από το 2017 και μετά, καθώς πέρα των χαρακτηριστικών εργασιών που αναφέρθηκαν παραπάνω, παράλληλα έχουν δημοσιευτεί αρκετές επιπλέον, από ερευνητές από όλο τον κόσμο. Ως πιο πρόσφατες εργασίες του τελευταίου χρόνου ενδεικτικά είναι η εργασία με τίτλο 'Physics-informed machine learning' (Karniadakis, Kevrekidis, Lu, & Perdikaris, 2021) και 'Physics-Informed Neural Networks for elastic plate problems' (Muradova & Stavroulakis, 2021) όπου στην εργασία αυτή, εφαρμόζεται η μέθοδος για την επίλυση της διαφορικής εξίσωσης που αφορά το πρόβλημα της πλάκας υπό κάμψη (Kirchhoff Plate Bending).

Όλες οι παραπάνω εργασίες και ορισμένες επιπλέον αποτέλεσαν πηγή πληροφόρησης για την κατανόηση της μεθόδου, την εφαρμογή της και για την συγγραφή της παρούσας εργασίας.

4.2 Τρόπος λειτουργίας

Όπως αναφέρθηκε και στην εισαγωγή της εργασίας τα περισσότερα προβλήματα μηχανικής εκφράζονται μέσω Διαφορικών Εξισώσεων. Ένας τρόπος επίλυσης είναι η εύρεση της αναλυτικής λύσης δηλαδή μιας έκφρασης της ζητούμενης συνάρτησης u(x) η οποία ικανοποιεί ακριβώς την Δ.Ε. και τις αρχικές συνθήκες για όλα τα σημεία του πεδίου ορισμού της συνάρτησης (symbolic solution). Ένας άλλος τρόπος επίλυσης είναι η χρήση Αριθμητικών Μεθόδων, όπως για παράδειγμα η μέθοδος Runge - Kutta ή η Μέθοδος Πεπερασμένων Στοιχείων (FEM), κατά τις οποίες η ζητούμενη συνάρτηση u(x) προσεγγίζεται σημείο προς σημείο από μια συνάρτηση $\hat{u}(x)$. Η μέθοδος Physics Informed Neural Networks ουσιαστικά είναι μια αριθμητική μέθοδος η οποία έχει την πρωτοτυπία να προσεγγίζει την ζητούμενη συνάρτηση u(x) μέσω ενός Τεχνητού Νευρωνικού Δικτύου (ΤΝΔ ή Artificial Neural Network). Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, ένα ΤΝΔ κατά την προς τα εμπρός τροφοδότηση (forward - pass) είναι μια σύνθετη συνάρτηση των μεταβλητών εισόδων. Επομένως $\mathbf{u(x) \approx \hat{u}(x) = NN(x).}$

Η φράση 'Physics Informed' στον τίτλο της μεθόδου αναφέρεται στο γεγονός ότι το δίκτυο εκπαιδεύεται με γνώμονα την διαφορική εξίσωση, τις αρχικές και τις οριακές συνθήκες που επιβάλλει η φυσική και τα μαθηματικά της εκάστοτε διεργασίας που μελετάται. Πρόκειται για ένα είδος self- supervised learning όπου η εκπαίδευση πραγματοποιείται με ζευγάρια εισόδων και επιθυμητών εξόδων τα οποία δεν εισάγονται από τον μελετητή από μια ξεχωριστή από το πρόγραμμα βάση, αλλά παράγονται από την ίδια την διαφορική εξίσωση μέσα στο ίδιο το πρόγραμμα. Οι είσοδοι του δικτύου είναι οι ανεξάρτητες μεταβλητές x, y, z, t της συνάρτησης αν πρόκειται για Μ.Δ.Ε (P.D.E) ή μόνο της μεταβλητής x ή t αν πρόκειται για Σ.Δ.Ε (O.D.E) με τις οποίες ασχολείται η εργασία. Οι είσοδοι προκύπτουν από το πεδίο ορισμού της u(x). Η αντίστοιχη έξοδος του δικτύου κατά την εμπρός τροφοδότηση, είναι η συνάρτηση $\hat{u}(x)$. Η ιδιαιτερότητα της μεθόδου και το σημείο που την διαφέρει από τα κοινά ΤΝΔ είναι το γεγονός ότι κατά την εκπαίδευση πέρα από τον υπολογισμό των παραγώγων της Loss function ως προς όλες τις trainable parameters του δικτύου, απαιτείται και ο υπολογισμός της παραγώγου του ίδιου του νευρωνικού δικτύου $\hat{u}(x)$ ως προς την είσοδο x. Αντίστοιχα αν πρόκειται για επίλυση Μ.Δ.Ε. απαιτούνται όλες οι μερικές παράγωγοι της $\hat{u}(x)$ ως προς τις ανεξάρτητες μεταβλητές x, y, z, t. Αυτό απαιτείται ώστε να σχηματιστούν οι παράγωγοι $\frac{d\hat{u}(x)}{dx}$, $\frac{d^2\hat{u}(x)}{dx^2}$, $\frac{d^3\hat{u}(x)}{dx^3}$ κ.τ.λ. με τις οποίες θα σχηματιστούν η διαφορική εξίσωση και οι αρχικές συνθήκες. Με αυτές τις σχέσεις θα κατασκευαστεί η Loss function προκειμένου να ελεγχθεί κατά πόσο απέχει η προσέγγιση $\hat{u}(x)$ από την u(x). Στην συνέχεια που θα παρουσιαστεί αναλυτικά η διαδικασία εκπαίδευσης και η διαδικασία backpropagation προγραμματιστικά, θα γίνει ακόμα πιο εμφανές ότι για την εκπαίδευση δεν απαιτούνται παραδείγματα με ζευγάρια εισόδων εξόδων τα οποία εισάγονται από κάποια εξωτερική βάση δεδομένων όπως συμβαίνει στα κοινά ΤΝΔ. Απεναντίας, στην προκειμένη περίπτωση της self – supervised learning, ουσιαστικά τα ζευγάρια αυτά παράγονται επιτόπου

κατά την έναρξη της εκτέλεσης του προγράμματος από τα δεδομένα της διαφορικής εξίσωσης και αποθηκεύονται σε κατάλληλα διανύσματα τα οποία θα παρουσιαστούν αναλυτικά σε κάθε πρόβλημα που θα λυθεί.

Ένα σημείο που αποτέλεσε μια έμπνευση για το εγχείρημα και συγχρόνως αποτέλεσε θετική ένδειξη πως η μέθοδος αυτή θα επιφέρει καλές προσεγγίσεις, είναι η δημοσίευση ενός θεωρήματος το οποίο αποδεικνύει ότι ένα ΤΝΔ με κατάλληλο αριθμό νευρώνων και Hidden layers μπορεί να προσεγγίσει οποιαδήποτε συνεχής συνάρτηση (Hornic, Stinchcombe, & White , 1989). Το θεώρημα αυτό καλείται ***'Universal Approximation Theorem'*** και συνέπειά του είναι ότι η οποιαδήποτε αποτυχία προσέγγισης της υπό μελέτης συνάρτησης, προκύπτει είτε από ατελής εκπαίδευση του δικτύου, είτε από ανεπαρκή αριθμό κρυφών νευρώνων, είτε από έλλειψη κάποιας αιτιοκρατικής σχέσης μεταξύ εισόδων και εξόδων.

4.2.1 Προς τα εμπρός τροφοδότηση (Forward – Pass)

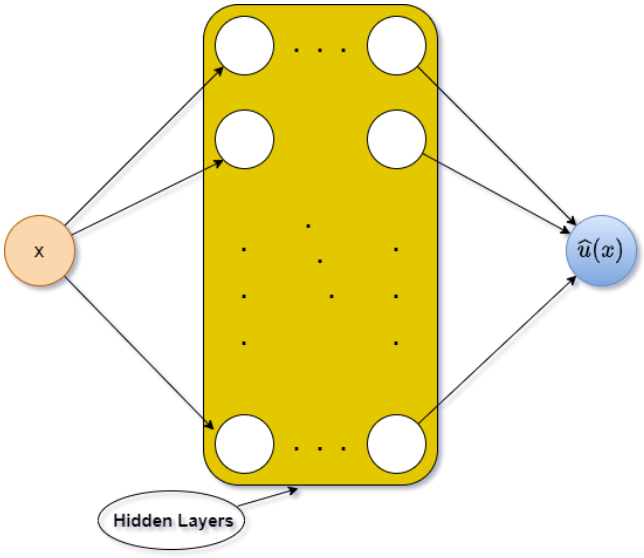
Στην συγκεκριμένη παράγραφο μελετάται η διαδικασία διάδοσης των τιμών εισόδου μέσα από το input layer προς τα hidden layers, μέχρι την έξοδο του νευρωνικού δικτύου. Η διαδικασία αυτή θα εξεταστεί μέσω ενός παραδείγματος διαφορικής εξίσωσης και συγκεκριμένα μιας Σ.Δ.Ε, το οποίο θα χρησιμοποιηθεί και για την επόμενη παράγραφο που θα εξεταστεί η προς τα πίσω διάδοση (backpropagation). Έστω ότι η Σ.Δ.Ε. η οποία προορίζεται για επίλυση είναι η εξής γραμμική δεύτερης τάξης εξίσωση:

$$\frac{d^2u(x)}{dx^2} + a * \frac{du(x)}{dx} = b \qquad \text{με a, b σταθεροί παράμετροι} \tag{62}$$

$$u(x_0) = u_0 \tag{63}$$

$$\frac{du(x1)}{dx} = u_1 \tag{64}$$

Για την εύρεση της λύσης της Δ.Ε. δηλαδή της συνάρτησης u(x) θα κατασκευαστεί το παρακάτω Τ.Ν.Δ της Εικόνας 4.2.1.1 του οποίου η έξοδος NN(x) = $\hat{u}(x)$ θα προκύψει από τους παρακάτω υπολογισμούς.



Εικόνα 4.2.1.1 (Προσέγγιση της συνάρτησης u(x) μέσω της συνάρτησης $\hat{u}(x)$ = NN(X) – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net/)

Στο κεφάλαιο 2 ορίστηκε ως L ο συμβολισμός αρίθμησης των layers με L = 0, 1, 2…n και ο υπολογισμός του διανύσματος εξόδου από κάθε layer γίνεται από τον τύπο της σχέσης (13) $\mathbf{z^L = \sigma(z^{L-1} * W^L + b^L).}$ Το σημείο που χρειάζεται προσοχή στην μέθοδο PINNs είναι ότι η έξοδος του δικτύου έχει γραμμική εξάρτηση με το τελευταίο hidden layer. Δηλαδή η έξοδος του τελευταίου hidden layer δεν εισέρχεται στην activation function όπως συμβαίνει συνήθως στα ΤΝΔ. Για καλύτερη κατανόηση ορίζονται ώς z^{n-1} ως το διάνυσμα που περιλαμβάνει τις εξόδους του τελευταίου hidden layer, W^n ως ο πίνακας που περιλαμβάνει τα βάρη μεταξύ του τελευταίου hidden layer και του νευρώνα εξόδου του δικτύου και b^n ως το διάνυσμα με τα biases το οποίο εφόσον στο συγκεκριμένο ΤΝΔ υπάρχει ένας μοναδικός νευρώνας εξόδου το b^n είναι ένας αριθμός. Αν υποθετικά, το τελευταίο hidden layer αποτελείται από 3 νευρώνες με εξόδους που περιέχονται στο εξής διάνυσμα:

$z^{n-1} = [z_{(n-1)1} \;\; z_{(n-1)2} \;\; z_{(n-1)3}]$ τότε: (64)

$$W^n = \begin{bmatrix} wn,11 \\ wn,21 \\ wn,31 \end{bmatrix}, \qquad b^n = [b_{n1}] = b_{n1} \tag{65}$$

$$z^n = \hat{u}(x) = z^{n-1} * W^n + b^n = [z_{(n-1)1} \;\; z_{(n-1)2} \;\; z_{(n-1)3}] * \begin{bmatrix} wn,11 \\ wn,21 \\ wn,31 \end{bmatrix} + b_{n1} =>$$

$$\mathbf{\hat{u}(x) = z_{(n-1)1} * w_{n,11} + z_{(n-1)2} * w_{n,21} + z_{(n-1)3} * w_{n,31} + b_{n1}}$$

Συνοψίζοντας τα παραπάνω, προκύπτουν 2 σχέσεις με τις οποίες εκτελείται η προς τα εμπρός διάδοση (forward – pass) στην μέθοδο PINNs.

$$\mathbf{z^L = \sigma(z^{L-1} * W^L + b^L) \qquad \text{για L = 0, 1, 2…n-1}} \tag{66}$$

$$\mathbf{z^n = \hat{u}(x) = z^{n-1} * W^n + b^n \qquad \text{για L = n}} \tag{67}$$

4.2.2 Εκπαίδευση & Backpropagation

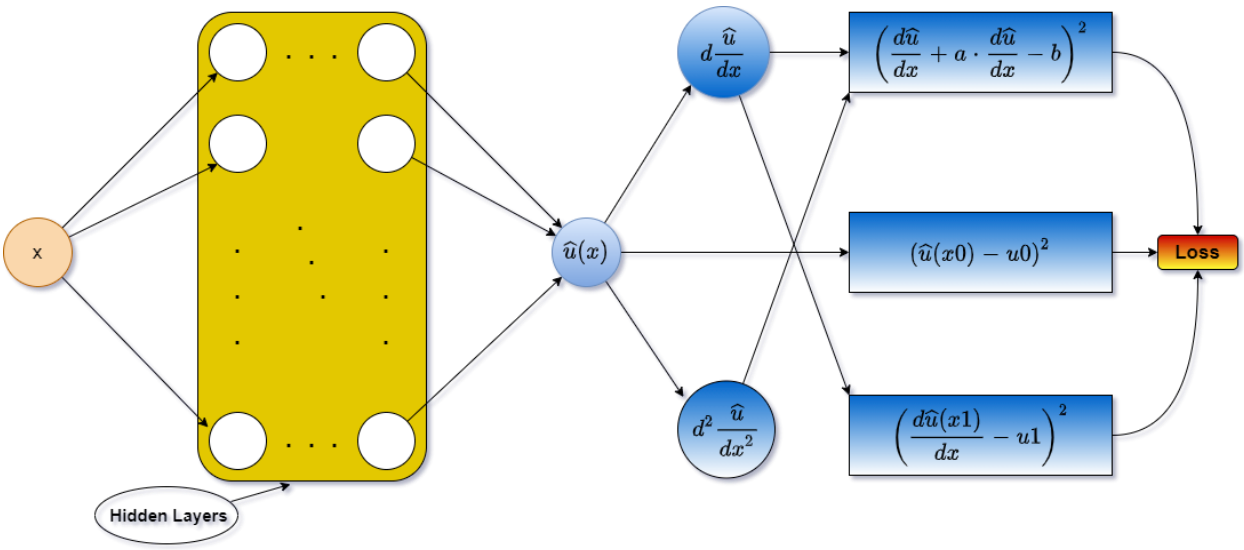
Η διαδικασία της εκπαίδευσης πραγματοποιείται τροποποιώντας τις trainable parameters (βάρη και biases) κατά τέτοιο τρόπο ώστε η συνάρτηση NN(x) = û(x) να ικανοποιεί την διαφορική εξίσωση και τις αρχικές συνθήκες με όσο το δυνατόν μικρότερο σφάλμα. Αυτό σημαίνει ότι θα πρέπει να σχηματιστεί μια custom Loss function, εφόσον στην προκειμένη περίπτωση δεν υπάρχουν τιμές στόχοι u(x) για κάθε x που εισέρχεται στο δίκτυο, καθώς η u(x) είναι άγνωστη. Ως εκ τούτου η συνάρτηση Loss function σχηματίζεται ως εξής:

$$\mathbf{Loss\ function} = \left(\frac{d^2\hat{u}(x)}{dx^2} + a * \frac{d\hat{u}(x)}{dx} - b\right)^2 + (\hat{u}(x_0) - u_0)^2 + \left(\frac{d\hat{u}(x1)}{dx} - u_1\right)^2 \tag{68}$$

Εφόσον υπάρχει μια σχέση που να περιγράφει την συνάρτηση Loss function, τότε ο αλγόριθμος backpropagation μπορεί να λειτουργήσει με τους γνωστούς αλγόριθμους βελτιστοποίησης (optimization algorithms) όπως ο Gradient Decent ή ο L-BFGS ή ο Adam. Κατά την εφαρμογή της μεθόδου PINNs στην συγκεκριμένη εργασία χρησιμοποιήθηκαν αρχικά ο L-BFGS και στην συνέχεια δοκιμάστηκε και ο Adam. Τα γενικά βήματα του αλγορίθμου εκπαίδευσης του δικτύου είναι τα ακόλουθα:

1. Forward Pass – υπολογισμός της û(x) που εξέρχεται από το ΤΝΔ για μια τιμή x_i.
2. Υπολογισμός απαραίτητων παραγώγων της û(x) ως προς x σύμφωνα με τη Δ.Ε.

3. Σχηματισμός και υπολογισμός του κατάλληλου Loss function.
4. Υπολογισμός των μερικών παραγώγων της Loss function ως προς όλες τις trainable parameters ξεκινώντας από το τελευταίο layer και συνεχίζοντας διαδοχικά προς τα πίσω (backpropagation).
5. Μεταβολή των trainable parameters προς την κατεύθυνση που ελαχιστοποιεί την Loss function με τον εκάστοτε αλγόριθμο ελαχιστοποίησης.
6. Επανάληψη των βημάτων 1, 2, 3, 4, 5 για όλα τα σημεία x_i (training set) ή για όλα τα batches.
7. Επανάληψη των βημάτων 1, 2, 3, 4, 5, 6 για όσους κύκλους εκπαίδευσης (epochs) έχουν οριστεί ή έως ότου επιτευχθεί μια συνθήκη τερματισμού.



Εικόνα 4.2.2.1 (Απεικόνιση του ΤΝΔ μαζί με τον σχηματισμό της Δ.Ε. και των αρχικών συνθηκών μέσω αυτού – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net/)

Παρά ταύτα, στην εφαρμογή της μεθόδου σε πραγματικά προβλήματα που παρατίθενται παρακάτω, το training set χωρίζεται σε batches. Επίσης αξίζει να τονιστεί ότι το μοντέλο με την custom Loss function που φαίνεται παραπάνω και αυτός ο τρόπος λειτουργίας έχει μεταφερθεί με την ίδια νοοτροπία και σε κώδικά της Python σε πολλές δημοσιευμένες εργασίες. Όμως, στην εφαρμογή της μεθόδου στην παρούσα εργασία χρησιμοποιήθηκε μια παραλλαγή του παραπάνω μοντέλου κατά την οποία χρησιμοποιούντα δύο συνδεδεμένα ΤΝΔ. Περισσότερες λεπτομέρειες πάνω σε αυτό θα συζητηθούν σε μια από τις επόμενες παραγράφους που αφορούν τον προγραμματισμό.

4.3 Automatic Differentiation

Πλέον, είναι δεδομένο ότι για την υλοποίηση των ΤΝΔ και ειδικά για την υλοποίηση της μεθόδου PINNs, χρειάζονται αλγόριθμοι ελαχιστοποίησης κατά την εκπαίδευση, οι οποίοι βασίζονται στον υπολογισμό μερικών παραγώγων για όλες τις trainable parameters, όπως επίσης χρειάζονται και οι παράγωγοι του δικτύου ως προς τις εισόδους του. Κατά συνέπεια χρειάζεται ένας αποδοτικός τρόπος υπολογισμού παραγώγων ούτως ώστε η διαδικασία να επιτυγχάνεται ταχύτερα και με όσο το δυνατό λιγότερες απαιτήσεις σε μνήμη. Σε αυτήν την παράγραφο θα γίνει μια συζήτηση για την διαδικασία Automatic

Differentiation καθώς και για την βιβλιοθήκη Tensorflow της Python η οποία παρέχει κατάλληλες συναρτήσεις που θα χρησιμοποιηθούν κατά κόρον.

Μια επιλογή για την παραγώγιση μιας συνάρτησης είναι με τους βασικούς κανόνες παραγώγισης όπου σε πολλές γλώσσες προγραμματισμού υπάρχουν αντίστοιχες συναρτήσεις οι οποίες κάνουν «symbolic differentiation» όπως αποκαλείται η διαδικασία. Σε εφαρμογές machine learning και deep learning όμως ουσιαστικά δεν χρειάζεται μια μαθηματική έκφραση της παραγώγου της Loss function αλλά μόνο η αριθμητική της τιμή στο ζητούμενο σημείο. Επιπρόσθετα οι symbolic functions είναι ιδιαίτερα χρονοβόρες και σε σύνθετες συναρτήσεις όπως τα ΤΝΔ, η χρήση τους είναι πρακτικά απαγορευτική. Ένας άλλος τρόπος είναι η αριθμητική παραγώγιση, με την οποία όμως το πρόβλημα είναι ότι όπως σε κάθε αριθμητική μέθοδο έτσι και σε αυτήν την περίπτωση υπάρχει κάποιο σφάλμα απόκλισης. Τα σφάλμα αυτό σε πολύπλοκες διεργασίες όπως η εκπαίδευση ενός multilayer ΤΝΔ θα επιφέρει σημαντικό θόρυβο, καθιστώντας την μέθοδο μη αποδοτική. Η επιλογή της Automatic differentiation συνδυάζει κυρίως τα θετικά των δύο παραπάνω επιλογών. Με τον κανόνα της αλυσίδας, ουσιαστικά προκύπτει τελική ακρίβεια ίδια με αυτή της «symbolic differentiation» και ταυτόχρονα επιθυμητή ταχύτητα υπολογισμού.

Οι πληροφορίες για τον τρόπο λειτουργίας αντλήθηκαν από το άρθρο με τίτλο ‘Automatic Differentiation in Machine Learning: a Survey’ (Baydin, Pearlmutter , Radul, & Siskind, 2018). Παρακάτω ακολουθεί ένα παράδειγμα το οποίο θα χρησιμοποιηθεί για την επεξήγηση της διαδικασίας υπολογισμού παραγώγων με Automatic Differentiation.

4.3.1 Παράδειγμα

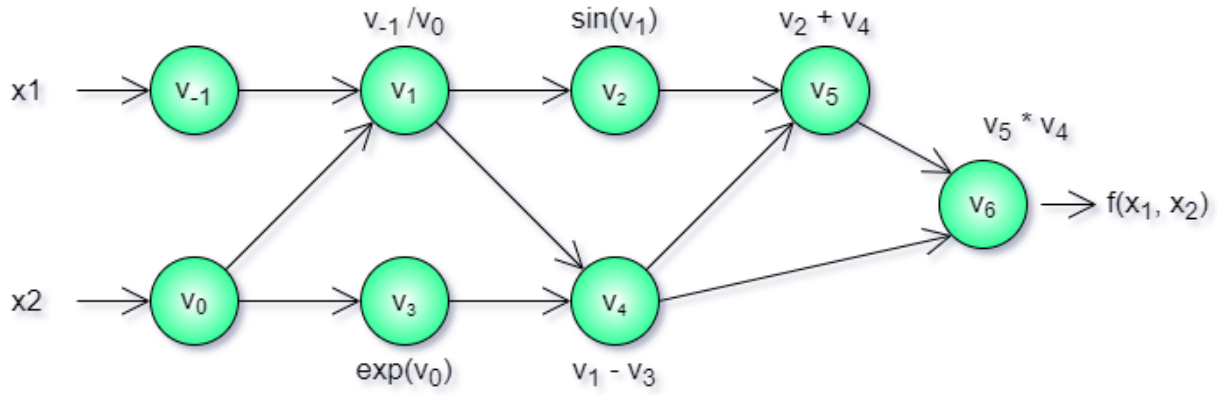
Έστω ότι η συνάρτηση προς παραγώγιση είναι η $f(x_1, x_2) = [\sin(\frac{x1}{x2}) + \frac{x1}{x2} - e^{x2}] * [\frac{x1}{x2} - e^{x2}]$ και η ζητούμενη παράγωγος είναι η μερική παράγωγος $\frac{df}{dx1}$ στο σημείο (1.5 , 0.5).

Η μέθοδος διαθέτει δύο εκδοχές, την Forward Mode και την Reverse Mode οι οποίες αναλύονται παρακάτω.

Forward Mode

Η συνάρτηση χωρίζεται σε επιμέρους τμήματα με απλές μαθηματικές εκφράσεις οι οποίες ονομάζονται Primals και συμβολίζονται με v_i. Στην συνέχεια υπολογίζονται οι παράγωγοι των επιμέρους αυτών τμημάτων ως προς την ενδιαφερόμενη μεταβλητή που στην προκειμένη περίπτωση είναι η x_1. Οι παράγωγοι καλούνται Tangents. Ορίζεται ως:

$$\dot{v}_i = \partial v_i / \partial x_1 \quad (69)$$



Εικόνα 4.3.1.1 (Automatic Differentiation – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net/)

Primals	Tangents
$v_{-1} = x_1 = 1.5$	$\dot{v}_{-1} = 1$
$v_0 = x_2 = 0.5$	$\dot{v}_0 = 0$
$v_1 = v_1 / v_0 = 3.0$	$\dot{v}_1 = (\dot{v}_{-1} * v_0 - \dot{v}_0 * v_{-1}) / v_0^2 = 2.0$
$v_2 = \sin(v_1) = 1.141$	$\dot{v}_2 = \cos(v_1) * \dot{v}_1 = -1.98$
$v_3 = \exp(v_0) = 1.649$	$\dot{v}_3 = \dot{v}_0 * v_3 = 0$
$v_4 = v_1 - v_3 = 1.351$	$\dot{v}_4 = \dot{v}_1 - \dot{v}_3 = 2.0$
$v_5 = v_2 + v_4 = 1.492$	$\dot{v}_5 = \dot{v}_2 + \dot{v}_4 = 0.02$
$v_6 = v_5 * v_4 = 2.017$	$\dot{v}_6 = \dot{v}_5 * v_4 + v_5 * \dot{v}_4 = 3.012$

Πίνακας 4.3.1.1 (Πίνακας υπολογισμού για Forward Mode)

Σύμφωνα με τους παραπάνω υπολογισμούς προκύπτει $\frac{\mathbf{df}}{\mathbf{dx1}} = \mathbf{3.012}$ στο σημείο (1.5 , 0.5).

Για τον υπολογισμό της $\frac{df}{dx2}$ θα χρειαζόταν ο υπολογισμός εκ νέου των παραγώγων της στήλης Tangents, γεγονός που προξενεί μια αδυναμία της Forward Mode.

Reverse Mode

Σε αυτή την εκδοχή πραγματοποιείται ο υπολογισμός των παραγώγων με ανάποδη σειρά από ότι στην Forward Mode. Η αριστερή στήλη υπολογίζεται με τον ίδιο τρόπο, όμως κάθε στοιχείο v_i που υπολογίζεται αποθηκεύεται στην μνήμη γιατί θα χρειαστεί στην συνέχεια για τον υπολογισμό των παραγώγων. Μόλις τελειώσουν οι υπολογισμοί των Primals τότε αρχίζει ο υπολογισμός των Adjoints όπως αποκαλούνται και συμβολίζονται ως $\bar{v}_i$. Ισχύει ότι:

$$\bar{v}_i = \frac{\partial f}{\partial vi} = \sum_{j:childs\ of\ i} \bar{v}j \frac{\partial vj}{\partial vi} \tag{70}$$

Ως j αναφέρονται οι κόμβοι στους οποίους δίνει την τιμή του ο κόμβος i.

Ενδεικτικά:

$$\bar{v}_6 = \frac{\partial f}{\partial v6} = \sum_{j:childs\ of\ i} \bar{v}j \frac{\partial vj}{\partial v6} = 1$$

$\bar{v}_5 = \frac{\partial f}{\partial v5} = \sum_{j:childs\ of\ i} \bar{v}j \frac{\partial vj}{\partial v5} = \bar{v}6 \frac{\partial v6}{\partial v5} = \bar{v}_6 * v_4$

$\bar{v}_4 = \frac{\partial f}{\partial v4} = \sum_{j:childs\ of\ i} \bar{v}j \frac{\partial vj}{\partial v4} = \bar{v}5 \frac{\partial v5}{\partial v4} + \bar{v}6 \frac{\partial v6}{\partial v4} = \bar{v}5 * 1 + \bar{v}6 * v_5$

Primals	Adjoints
$v_{-1} = x_1 = 1.5$	$\bar{v}_6 = \bar{f} = 1$
$v_0 = x_2 = 0.5$	$\bar{v}_5 = \bar{v}_6 * v_4 = 1.351$
$v_1 = v_1 / v_0 = 3.0$	$\bar{v}_4 = \bar{v}5 * 1 + \bar{v}6 * v_5 = 2.843$
$v_2 = \sin(v_1) = 1.141$	$\bar{v}_3 = -\bar{v}_4 = -2.843$
$v_3 = \exp(v_0) = 1.649$	$\bar{v}_2 = \bar{v}_5 = 1.351$
$v_4 = v_1 - v_3 = 1.351$	$\bar{v}_1 = \bar{v}_2 * \cos(v_1) + \bar{v}_4 = 1.506$
$v_5 = v_2 + v_4 = 1.492$	$\mathbf{\bar{v}_0 = \bar{v}_1 * (-\bar{v}_{-1})/ v_0^2 + \bar{v}_3 * e^{v0} = -13.723}$
$v_6 = v_5 * v_4 = 2.017$	$\mathbf{\bar{v}_{-1} = \bar{v}_1 * (1/ v_0) = 3.012}$

Πίνακας 4.3.1.2 (Πίνακας υπολογισμού για Reverse Mode)

Σύμφωνα με τους παραπάνω υπολογισμούς προκύπτει $\mathbf{\bar{v}_{-1} = \frac{\partial f}{\partial x1} = 3.012}$ **και** $\mathbf{\bar{v}_0 = \frac{\partial f}{\partial x2} = -13.723.}$ Αυτό σημαίνει ότι ο αλγόριθμος μπορεί να υπολογίσει μαζί όλες τις μερικές παραγώγους της συνάρτησης προς τις μεταβλητές της, δυνατότητα που δεν μπορεί να καλύψει η Forward Mode. Πάραυτα, εκεί που υστερεί η παραπάνω εκδοχή, είναι στο γεγονός ότι απαιτεί περισσότερη μνήμη από την Forward Mode καθώς πρέπει να αποθηκεύει όλες τις Primals τιμές, ενώ παράλληλα έχει και πιο σύνθετους υπολογισμούς. Γενικά αν η εν λόγω συνάρτηση είναι η f: $R^n \rightarrow R^m$ δηλαδή έχει n παραμέτρους και η f έχει εξόδους $f_1, f_2, f_3 \ldots f_m$ τότε στην περίπτωση που n << m προτιμάται η Forward Mode. Αντιθέτως αν m << n τότε προτιμάται η Reverse Mode. Ουσιαστικά η τελευταία περίπτωση φωτογραφίζει το πεδίο του deep learning όπου σε ένα ΤΝΔ οι παράμετροι του δικτύου (trainable parameters) είναι ασύγκριτα περισσότεροι από τους νευρώνες εξόδου του δικτύου.

4.3.2 Υλοποίηση σε Python code μέσω Tensorflow

Η υλοποίηση της παραγώγισης μέσω Automatic Differentiation στην Python πραγματοποιείται μέσω της Tensorflow η οποία περιλαμβάνει ένα πακέτο συναρτήσεων κατάλληλων για την υλοποίηση των ΤΝΔ. Tensorflow είναι μια ανοιχτού κώδικα end – to – end βιβλιοθήκη για εφαρμογές machine learning και deep learning η οποία αναπτύχθηκε από την Google Brain team το 2015 και έχει εφαρμογές σε μονάδες επεξεργασίας CPU, GPU, TPU .

Στην συνέχεια θα παρατεθεί ο τρόπος που συντάσσεται η συνάρτηση για τον υπολογισμό παραγώγων, συνοδευόμενος με τις αντίστοιχες γραμμές κώδικα της Python. Αρχικά πρέπει να εγκατασταθεί η βιβλιοθήκη, οπότε στο Terminal πρέπει να εκτελεστεί η εντολή 'pip install tensorflow'. Στην συνέχεια η βιβλιοθήκη μπορεί να συμπεριληφθεί στο πρόγραμμα γράφοντας 'import tensorflow as tf' και με τον τρόπο αυτό, μετά από το ακρωνύμιο 'tf.' υπάρχει πρόσβαση σε όλες τις συναρτήσεις της βιβλιοθήκης. Η tensorfow παρέχει την συνάρτηση tf.GradientTape() API (Application Programming Interface) η οποία είναι υπεύθυνη για τον υπολογισμό παραγώγων μέσω Automatic Differentiation Reverse Mode. Ενδεικτικό παράδειγμα αποτελεί ο υπολογισμός της παραγώγου της σιγμοειδής συνάρτησης σε ένα συγκεκριμένο σημείο x που πραγματοποιείται με τις ακόλουθες εντολές:

```
import tensorflow as tf
import numpy as np
```

```
x = tf.Variable(3.0)
with tf.GradientTape() as tape:
    y = tf.sigmoid(x)
dy_dx = tape.gradient(y, x)
print(f"dy_dx = {dy_dx}")
```

Η έξοδος είναι η εξής: ‘dy_dx = 0.04517665505409241’

Με την εντολή x = tf.Veriable(3.0) δηλώνεται η x ως μεταβλητή με αρχική τιμή x = 3.0

Αν αντί για x = tf.Veriable(3.0) γραφτεί η εντολή x = tf.constant(3.0), τότε η έξοδος του προγράμματος είναι ‘dy_dx = None’. Αυτό συμβαίνει διότι η εντολή tf.constant(3.0) δεν αντιστοιχεί σε μεταβλητή αλλά σε σταθερά. Στην προκειμένη περίπτωση χρειάζεται η εντολή tape.watch(x) η οποία «οδηγεί» την GradientTape στο να υπολογίσει τις παραγώγους ως προς την x. Ουσιαστικά η tape.watch(x) χρειάζεται πριν υλοποιηθεί η προς τα εμπρός διάδοση του σήματος (Forward Pass), ώστε η GradientTape να αποθηκεύει τις κατάλληλες Primals τιμές και τελικά κατά την προς τα πίσω διάδοση (Backpropagation) να είναι εφικτός ο υπολογισμός των Adjoints τιμών.

```
x = tf.constant(3.0)
with tf.GradientTape() as tape:
    tape.watch(x)
    y = tf.sigmoid(x)
dy_dx = tape.gradient(y, x)
print(f"dy_dx = {dy_dx}")
```

Το αποτέλεσμα είναι το εξής: ‘dy_dx = 0.04517665505409241’

Αν θεωρητικά υπάρχει ένα απλό δίκτυο χωρίς κανένα hidden layer, με 3 νευρώνες εισόδου και 2 νευρώνες εξόδου τότε με το παρακάτω πρόγραμμα υπολογίζεται η Loss function και ο Ιακωμβιανός πίνακας που περιλαμβάνει τις μερικές παραγώγους της Loss function ως προς τις trainable parameters.

```
w = tf.Variable([[0.5, 0.5], [0.5, 0.5], [0.5, 0.5]])
b = tf.Variable(tf.zeros(2, dtype=tf.float32), name='b')
x = [[1., 2., 3.]]

with tf.GradientTape(persistent=True) as tape:
  z = x @ w + b                  # z = [z1  z2]
  loss = tf.reduce_mean(z ** 2)  # loss = z1^2 + z2^2
  print("Loss = ")
  print(loss)
[dL_dw, dL_db] = tape.gradient(loss, [w, b])
print("dL_dw = ")
print(dL_dw)
print("dL_db")
print(dL_db)
```

Η έξοδος του παραπάνω προγράμματος είναι η ακόλουθη:

```
Loss =
tf.Tensor(9.0, shape=(), dtype=float32)
dL_dw =
tf.Tensor(
[[3. 3.]
 [6. 6.]
 [9. 9.]], shape=(3, 2), dtype=float32)
dL_db
tf.Tensor([3. 3.], shape=(2,), dtype=float32)

Process finished with exit code 0
```

Πίνακας 4.3.2.1 (Αποτελέσματα)

Η εντολή loss = tf.reduce_mean(z ** 2) δημιουργεί την εξής συνάρτηση:

$$\text{Loss} = \frac{1}{2}[(1.0 * w_{11} + 2.0 * w_{21} + 3.0 * w_{31} + b_1)^2 + (1.0 * w_{12} + 2.0 * w_{22} + 3.0 * w_{32} + b_2)^2]$$

Επίσης η εντολή [dL_dw, dL_db] = tape.gradient(loss, [w, b]) δημιουργεί τον εξής πίνακα και διάνυσμα:

$$\text{dL\_dw} = \nabla_w L = \begin{bmatrix} \frac{\partial L}{\partial w11} & \frac{\partial L}{\partial w12} \\ \frac{\partial L}{\partial w21} & \frac{\partial L}{\partial w22} \\ \frac{\partial L}{\partial w31} & \frac{\partial L}{\partial w32} \end{bmatrix}, \qquad \text{dL\_db} = \nabla_b L = \begin{bmatrix} \frac{\partial L}{\partial b1} & \frac{\partial L}{\partial b2} \end{bmatrix} \tag{71}$$

4.3.3 Κατασκευή μοντέλων ΤΝΔ μέσω Keras

Στο πρόγραμμα που κατασκευάστηκε για το παράδειγμα του Κεφαλαίου 3.4, δημιουργήθηκαν πίνακες και διανύσματα ώστε να σχηματιστούν τα layers του δικτύου. Όπως φαίνεται από τον τρόπο γραφής του προγράμματος, πρόκειται για low-level πρόγραμμα και αυτό έγινε για λόγους καλύτερης κατανόησης και επίδειξης του τρόπου που πραγματικά δημιουργούνται τα layers. Σε πραγματικές εφαρμογές όμως ένας τέτοιος τρόπος γραφής είναι μη αποδοτικός και για τον λόγο αυτό σε αρκετές περιπτώσεις χρησιμοποιείται το Keras. Πρόκειται για μια υψηλού επιπέδου διεπαφή της πλατφόρμας Tensorflow 2 (API) για την υλοποίηση μοντέλων machine learning και ειδικά deep learning[7].

Μια πιο πρακτική χρήση της Automatic differentiation φαίνεται στις παρακάτω γραμμές :

```
import tensorflow as tf
layer = tf.keras.layers.Dense(2, activation='relu')
x = tf.constant([[1., 2., 3.]])

with tf.GradientTape() as tape:
  # Forward pass
  y = layer(x)
  loss = tf.reduce_mean(y**2)
```

[7] Η σελίδα του Keras είναι η εξής https://keras.io/getting_started/ . Εκεί αναφέρεται τι ακριβώς είναι, και περιλαμβάνει αναλυτικό οδηγό για την χρήση των συναρτήσεων του.

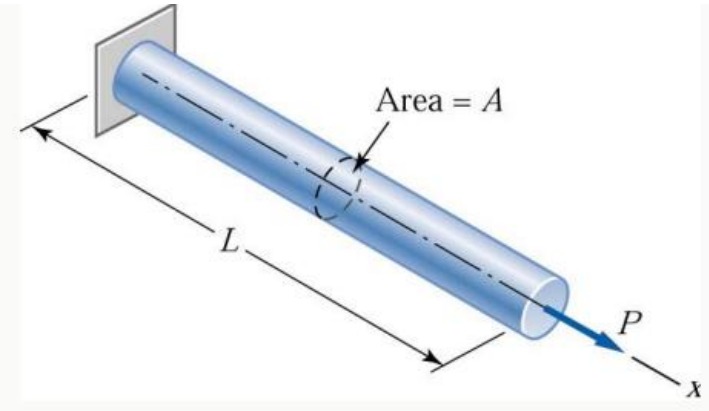
```
# Calculate gradients with respect to every trainable variable
grad = tape.gradient(loss, layer.trainable_variables)

# layer.trainable_variables is a list
print(layer.trainable_variables)
for var, g in zip(layer.trainable_variables, grad):
  print(f'{var.name}, shape: {g.shape}')
```

4.4 Ορισμός προβλημάτων προς επίλυση και προγραμματισμός

Στην παρούσα παράγραφο παρατίθενται οι διαφορικές εξισώσεις των προβλημάτων, με τις οποίες θα δοκιμαστεί η μέθοδος PINNs. Όπως έχει είδη αναφερθεί η εργασία ασχολείται με προβλήματα στατικής μηχανικής και συγκεκριμένα με την άσκηση εφελκυστικών δυνάμεων και δυνάμεων κάμψης σε στερεά σώματα όπως η ράβδος και η δοκός. Η διαφορική εξίσωση είναι ένα απαραίτητο εργαλείο για την μελέτη ενός φαινομένου στην μηχανική, καθώς μελετώντας ένα στερεό σώμα σε ένα απειροστά μικρό τμήμα του dx, γίνεται εφικτή η εξαγωγή μιας συνάρτησης που να περιγράφει την συμπεριφορά του σώματος για κάθε x σημείο του. Η συνάρτηση αυτή είναι αποτέλεσμα της επίλυσης της Δ.Ε. Ως αναφορά την δομή της παραγράφου, θα ακολουθήσουν 4 υποπαράγραφοι, μια για κάθε πρόβλημα εκ των τεσσάρων, όπου η κάθε μία θα συμπεριλαμβάνει τον τρόπο που γίνεται η προγραμματιστική υλοποίηση της μεθόδου.

4.4.1 Πρόβλημα 1: Άσκηση εφελκυστικής δύναμης σε ράβδο



Πίνακας 4.4.1.1 (Ράβδος πακτωμένη στο ένα άκρο της)

Εάν σε μια ράβδο ασκούνται κατανεμημένες δυνάμεις cx, τότε το φορτίο (load) σε ένα στοιχειώδες σημείο x είναι ισοδύναμο με load = $\int_x^L c\ x\ dx\ = \frac{c}{2}\ (L^2 - x^2)$ όπου L είναι το συνολικό μήκος της ράβδου. Η τάση (Stress) που ασκείται στη ράβδο ισούται με σ = $\frac{Load}{Area} = \frac{c\ (L^2 - x^2)}{2\ A}$. Σε μια ράβδο που υπόκειται σε αξονική φόρτιση, ορίζεται ως ορθή τροπή (Strain) το μέγεθος που εκφράζει την παραμόρφωση ανά μονάδα μήκους της ράβδου (Beer, Johnston, DeWolf, & Mazurek, 2016).

Ορίζεται ως ορθή τροπή σε ένα συγκεκριμένο σημείο της ράβδου το ακόλουθο μέγεθος.

$$\varepsilon = \lim_{\Delta x \to 0} \frac{\Delta \delta}{\Delta x} = \frac{du}{dx} \tag{72}$$

Η σύνδεση των παραπάνω μεγεθών γίνεται μέσω του μέτρου ελαστικότητας ή μέτρο ελαστικότητας Young (Modulus of Elasticity – Young's Modulus) το οποίο εκφράζεται ως $E = \frac{\sigma}{\varepsilon}$. Κατά συνέπεια ισχύει ότι:

$$E = \frac{\frac{c\,(L^2 - x^2)}{2\,A}}{\frac{du}{dx}} \Rightarrow E\frac{du}{dx} = \frac{c\,(L^2 - x^2)}{2\,A} \Rightarrow AE\frac{du}{dx} = \frac{c}{2}(L^2 - x^2) \quad (73)$$

Παραγωγίζοντας μια φορά την σχέση (73) προκύπτει η παρακάτω Σ.Δ.Ε

$$\mathbf{AE}\,\frac{\boldsymbol{d^2u}}{\boldsymbol{dx^2}} = \mathbf{-\,c}\,x \quad (74)$$

Ως οριακές συνθήκες θεωρούνται τα δεδομένα από την φύση του προβλήματος. Θεωρείται γνωστό ότι:

$u(0) = 0$ (75)

$\frac{du}{dx}\big|_{x = L} = 0$ (76)

Επιλύοντας την παραπάνω εξίσωση ως πρόβλημα αρχικών τιμών, εντοπίζεται η συνάρτηση u(x).

$$\mathbf{u(x) = \frac{c}{AE\,6}\,(-x^3 + 3\,L^2\,x)} \quad (77)$$

Η μέθοδος PINNs θα εφαρμοστεί σε μια παραλλαγή του παραπάνω προβλήματος όπου πέρα των εφελκυστικών δυνάμεων, εφαρμόζεται και μια επιπλέον δύναμη αντίθετης φοράς στο άκρο της ράβδου. Το πρόβλημα ορίζεται ως εξής:

$AE\frac{d^2u}{dx^2} + cx = 0 \qquad x \in [0,2L]$ (78)

$u(0) = 0 \qquad x = 0$ (79)

$E\frac{du}{dx}\big|_{x=2L}\,n + \frac{cL^2}{A} = 0 \qquad x = 2L$ (80)

u(x) : Μετατόπιση του σημείου x της ράβδου (m)

A : Εμβαδόν εγκάρσιας διατομής (cross sectional area) (m2)

E : Μέτρο ελαστικότητας (Pa – N/m2)

c : Εφελκυστική τάση (Pa – N/m2)

Για τον έλεγχο της αποτελεσματικότητας της μεθόδου θα χρειαστεί η αναλυτική λύση της παραπάνω Σ.Δ.Ε. ούτως ώστε να μπορεί να παρατηρηθεί αν η προσεγγιστική λύση είναι ικανοποιητικά κοντά με την πραγματική. Λύνοντας το πρόβλημα ως πρόβλημα αρχικών τιμών προέκυψε η παρακάτω λύση:

Αναλυτική Λύση (Exact Solution): $u(x) = \frac{c}{AE}(-\frac{x^3}{6} + L^2x)$ (81)

Η προγραμματιστική υλοποίηση πραγματοποιήθηκε σε γλώσσα Python 3.9 με χρήση Tensorflow 2.4.1 στο περιβάλλον Pycharm. Για την εφαρμογή της μεθόδου αναπτύχθηκε κώδικας ο οποίος στηρίχτηκε στον τρόπο γραφής του προγράμματος που βρίσκεται στον σύνδεσμο *https://github.com/okada39/pinn_burgers*. Το πρόγραμμα αυτό, εφαρμόζει την μέθοδο PINNs στο πρόβλημα της εξίσωσης Burgers (Burgers'

Equation), πρόβλημα το οποίο αρχικά μελετάται στην εργασία με τίτλο 'Physics informed Deep Learning (Part I) : Data - driven Solutions of Nonlinear Partial Differential Equations) (Raissi, Perdikaris, & Karniadakis, 2017). Ο συγκεκριμένος κώδικας που αναφέρθηκε, σχεδιάστηκε ώστε να λύνει μια Μερική Διαφορική Εξίσωση. Συνεπώς, χρειάστηκε να προσαρμοστούν κατάλληλα οι είσοδοι του ΤΝΔ, η διαδικασία εκπαίδευσης και πρόβλεψης καθώς επίσης τροποποιήθηκε κατάλληλα ο αλγόριθμος βελτιστοποίησης (optimization algorithm), προκειμένου να αναπτυχθεί ένας κώδικας ο οποίος να επιλύει τα προβλήματα στατικής μηχανικής που μελετώνται στην παρούσα εργασία.

Η προγραμματιστική υλοποίηση, παραδόξως, δεν είναι μια ακριβής μεταφορά του σχεδιαγράμματος της Εικόνας 4.2.2.1. Με λίγα λόγια, δεν χρησιμοποιείται ένα μοναδικό ΤΝΔ το οποίο χρησιμοποιείται για τον υπολογισμό των απαραίτητων παραγώγων ως προς την είσοδο, προκειμένου να κατασκευαστεί μια αυτοσχέδια custom Loss function, παρόλο που αυτό το μοντέλο συναντάται σε πολλές εργασίες. Εν προκειμένω, χρησιμοποιούνται 2 ΤΝΔ τα οποία αλληλεξαρτώνται και έχουν κοινές παραμέτρους δηλαδή βάρη και biases (trainable parameters). Το ένα δίκτυο στο πρόγραμμα ονομάζεται network και φτάνει μέχρι και τη έξοδο u(x). Το δίκτυο αυτό χρησιμοποιείται για τον υπολογισμό των παραγώγων που συντάσσουν την διαφορική εξίσωση, για τον υπολογισμό των $\hat{u}(x_i)$ που χρησιμοποιούνται για τις οριακές συνθήκες καθώς και για τον σχηματισμό της γραφικής παράστασης μετά την ολοκλήρωση της εκπαίδευσης. Το άλλο δίκτυο ονομάζεται pinn και συμπεριλαμβάνει το δίκτυο network καθώς και την επέκτασή του με τις παραγώγους του δικτύου network ως προς την είσοδο x. Ουσιαστικά αυτό το δίκτυο χρησιμοποιείται για την εκπαίδευση, δηλαδή την κατάλληλη τροποποίηση των κοινών παραμέτρων των 2 δικτύων. Όταν ολοκληρωθεί η εκπαίδευση του δικτύου pinn, τότε το δίκτυο network είναι έτοιμο να παράξει τις τελικές τιμές $\hat{u}(x_i)$ με τις οποίες θα κατασκευαστεί μια γραφική παράσταση. Παρόλο που η υλοποίηση δεν συνάδει επακριβώς με την Εικόνα 4.2.2.1, παρόμοιο σχεδιάγραμμα θα χρησιμοποιηθεί στην επεξήγηση των προβλημάτων που θα λυθούν, ώστε να γίνεται ξεκάθαρο ποιοι παράγωγοι χρειάζονται στο εκάστοτε πρόβλημα.

Ο κώδικας αποτελείται από την main και 5 modules τα οποία είναι το module network, το module layer, το module pinn, το module optimizer και το module tf_silent. Το module network περιλαμβάνει μια Class με όνομα Network η οποία μέσω μιας συνάρτησης με όνομα build κατασκευάζει το δίκτυο network και το επιστρέφει. Όπως αναφέρεται και στην παράγραφο 4.2.1, η έξοδος του τελευταίου hidden layer δεν εισέρχεται σε κάποια συνάρτηση ενεργοποίησης και κατά συνέπεια ισχύει η σχέση (67), $z^n = \hat{u}(x) = z^{n-1} * W^n + b^n$. Η λεπτομέρεια αυτή επιβεβαιώνεται και γίνεται ακόμα πιο εμφανής στην συνάρτηση build στην οποία για τον υπολογισμό της εξόδου δεν αναφέρεται κάποια συνάρτηση ενεργοποίησης σε αντίθεση με τα υπόλοιπα layers. Παρακάτω φαίνονται οι αντίστοιχες γραμμές κώδικα από το module network.

```
# input layer
inputs = tf.keras.layers.Input(shape=(num_inputs,))
# hidden layers
x = inputs
for layer in layers:
    x = tf.keras.layers.Dense(layer, activation=activation,
        kernel_initializer='he_normal')(x)
# output layer
outputs = tf.keras.layers.Dense(num_outputs,
    kernel_initializer='he_normal')(x)

return tf.keras.models.Model(inputs=inputs, outputs=outputs)
```

Το module layer έχει τον ρόλο του υπολογισμού των απαραίτητων παραγώγων του δικτύου network ως προς την είσοδο x που στην προκειμένη περίπτωση είναι οι παράγωγοι $\frac{d\hat{u}}{dx}$, $\frac{d^2\hat{u}}{dx^2}$. Το συγκεκριμένο module περιλαμβάνει μια Class με όνομα GradientLayer( ) η οποία θα κληθεί από το module pinn και λαμβάνει ως είσοδο το δίκτυο network. Τελικά επιστρέφονται οι αριθμητικές τιμές της πρώτης και της δεύτερης παραγώγου καθώς επίσης και η αριθμητική τιμή της $\hat{u}$(x) για κάθε τιμή x που εισέρχεται στο δίκτυο network. Ο υπολογισμός των παραγώγων γίνεται όπως ακριβώς επεξηγήθηκε στην παράγραφο 4.3.3 μέσω της συνάρτησης GradientTape() του Tensorflow. Παρακάτω αναγράφονται οι αντίστοιχες γραμμές του κώδικα από το module layer.

```
with tf.GradientTape() as g:
    g.watch(x)
    with tf.GradientTape() as gg:
        gg.watch(x)
        u = self.model(x)
    du_dx = gg.gradient(u, x)
d2u_dx2 = g.gradient(du_dx, x)
return u, du_dx, d2u_dx2
```

Το module pinn επικοινωνεί με το module layer καθώς λαμβάνει από αυτό τις παραγώγους ώστε να της χρησιμοποιήσει για την κατασκευή του δεύτερου νευρωνικού δικτύου το οποίο ονομάζεται pinn. Περιλαμβάνει μια Class με όνομα PINN στην οποία υπάρχει η συνάρτηση build οπού εκεί κατασκευάζεται το δίκτυο pinn και τελικά επιστρέφεται. Ουσιαστικά, αυτό που συμβαίνει είναι ότι χρησιμοποιούνται οι είσοδοι x που εισέρχονται στα δύο δίκτυα για να υπολογιστούν οι παράγωγοι, μέσω του module layer και στην συνέχεια κατασκευάζεται η διαφορική εξίσωση και οι αρχικές συνθήκες σαν ένα επιπλέον custom layer για να ολοκληρωθεί το δίκτυο pinn. Ο μόνος τρόπος να ελεγχθεί η πορεία της εκπαίδευσης είναι μέσω της φύσης της διεργασίας, δηλαδή με τον έλεγχο ότι ικανοποιούνται η διαφορική εξίσωση και οι οριακές συνθήκες. Για οποιαδήποτε x_1 ∈ [0,2L] υπολογίζεται η ποσότητα u_1 = AE$\frac{d^2\hat{u}(x_1)}{dx^2}$ + cx_1 , για x_2 = 0 υπολογίζεται ποσότητα u_2 = $\hat{u}$(x_2) και για x_3 = 2L υπολογίζεται η ποσότητα u_3 = E$\frac{d\hat{u}(x_3)}{dx}|_{x=2L}$ n + $\frac{cL^2}{A}$. Ως τιμές στόχοι για την έξοδο του δικτύου έχουν οριστεί οι u_1 = 0, u_2 = 0 και u_3 = 0. Στην συνέχεια, κατά την ανάλυση της main θα γίνει εμφανές ότι οι παραπάνω τιμές των x εισάγονται σε 3 διανύσματα εισόδου x_1, x_2, x_3. Παρακάτω παρατίθενται οι γραμμές κώδικά του module pinn στις οποίες γίνεται η κλήση της κλάσης (Class) GradientLayer από το module layer καθώς και ο υπολογισμός των u_1, u_2, u_3 αντίστοιχα.

```
self.grads = GradientLayer(self.network)


# compute gradients
u, du dx, d2u dx2 = self.grads(x 1)

# equation output being zero
u_1 = self.A * self.E * d2u_dx2 + self.c * x_1
# initial condition output
u 2 = self.network(x 2)
# boundary condition output
u, du_dx, d2u_dx2 = self.grads(x_3)
u_3 = self.E * du_dx * self.n + self.c * (self.L)**2 / self.A

# build the PINN model for BAR' equation
```

```
return tf.keras.models.Model(
    inputs=[x_1, x_2, x_3], outputs=[u_1, u_2, u_3])
```

Στο module optimizer πραγματοποιείται η διαδικασία της εκπαίδευσης του δικτύου pinn μέσω του αλγορίθμου βελτιστοποίησης L-BFGS ο οποίος επεξηγήθηκε συνοπτικά στην παράγραφο 3.3. Περιλαμβάνει μια Class με όνομα L_BFGS_B η οποία παίρνει ως παραμέτρους το νευρωνικό δίκτυο pinn, τις εισόδους x με τις αντίστοιχες τιμές στόχους u_1, u_2, u_3, καθώς και τις παραμέτρους factr, m, maxls, maxiter[8] . Αρχικά μέσω της συνάρτησης set_weights(self, flat_weights) γίνεται αρχικοποίηση των βαρών. Επίσης σε αυτό το module υπάρχει η συνάρτηση tf_evaluate(self, x, y) στην οποία υπολογίζονται οι παράγωγοι της Loss function ως προς τα κοινά βάρη και τα biases των δύο δικτύων. Στην συνάρτηση fit(self) πραγματοποιείται η τροποποίηση των παραμέτρων μέσω του αλγορίθμου L-BFGS. Παρακάτω υπάρχουν οι γραμμές κώδικα του module optimizer στις οποίες πραγματοποιείται η διαδικασία backpropagation και η διαδικασία της τροποποίησης των παραμέτρων αντίστοιχα.

```
with tf.GradientTape() as g:
    loss = tf.reduce_mean(tf.keras.losses.mse(self.model(x), y))
grads = g.gradient(loss, self.model.trainable_variables)
return loss, grads
```

```
scipy.optimize.fmin_l_bfgs_b(func=self.evaluate, x0=initial_weights,
    factr=self.factr, m=self.m, maxls=self.maxls, maxiter=self.maxiter)
```

Το module tf_silent έχει βοηθητικό χαρακτήρα και η μόνη του αποστολή είναι να απαλείφει τα περιττά μηνύματα που τυπώνονται κατά την εκκίνηση του προγράμματος λόγω της κλήσης της βιβλιοθήκης Tensorflow.

Στην main αρχικά πρέπει να συμπεριληφθούν οι βιβλιοθήκες που θα χρησιμοποιηθούν, όπως επίσης και οι απαραίτητες κλάσεις (Classes) από τα module που προαναφέρθηκαν. Συγκεκριμένα στην main θα χρησιμοποιηθεί η κλάση PINN από το module pinn, η κλάση Network από το module network και τέλος η κλάση L_BFGS_B από το module optimizer. Από την γραμμή που περιλαμβάνει την εντολή if __name__ == '__main__': και κάτω, αρχίζει η εκτέλεση του προγράμματος. Αρχικά ορίζονται οι τιμές των μεταβλητών num_train_samples και num_test_samples όπου στην συγκεκριμένη περίπτωση δόθηκαν οι τιμές 1000 και 100 αντίστοιχα. Ως num_train_samples ορίζεται ο αριθμός των σημείων x με τα οποία θα γίνει η εκπαίδευση (collocation points) ενώ ως num_test_samples ορίζεται ο αριθμός των σημείων x που μετά το πέρας της εκπαίδευσης, θα χρησιμοποιηθούν για τον έλεγχο και για την κατασκευή μιας γραφικής παράστασης της $\hat{u}(x)$. Παρακάτω ορίζονται οι σταθερές της διαφορικής εξίσωσης δηλαδή οι τιμές A, E, c, L και n και παρακάτω δίνονται οι ονομασίες network και pinn στα δύο ΤΝΔ που αναφέρθηκαν παραπάνω. Στην συνέχεια ακολουθεί το σημείο που ορίζονται τα διανύσματα εισόδου και εξόδου - στόχου x_1, x_2, x_3 και u_1, u_2, u_3 αντίστοιχα, οπού θα διαφοροποιούνται για κάθε πρόβλημα που θα ακολουθήσει. Το κάθε ένα από αυτά τα διανύσματα θα έχει διάσταση όσο και η τιμή num_train_samples. Τα διανύσματα αυτά αμέσως μετά εισάγονται σε δύο λίστες (lists) x_train = [x_1, x_2, x_3] και y_train = [u_1, u_2, u_3]. Έπειτα δημιουργείται ένα αντικείμενο με όνομα lbfgs, το οποίο ανοίκει στην κλάση L_BFGS_B() και μέσω της εντολής lbfgs.fit() καλείται η συνάρτηση fit() του module optimizer. Αφού

[8] Για την εκπαίδευση του δικτύου στο συγκεκριμένο πρόγραμμα χρησιμοποιείται η συνάρτηση optimize.fmin_l_bfgs_b της βιβλιοθήκης scipy. Στον σύνδεσμο https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html αναγράφεται αναλυτικά η σημασία των παραμέτρων της συνάρτησης.

ολοκληρωθεί η διαδικασία της εκπαίδευσης, ακολουθούν οι εντολές με τις οποίες κατασκευάζεται η γραφική παράσταση με τις εκτιμήσεις της u(x) οι οποίες προκύπτουν από το ΤΝΔ network. Συγκεκριμένα παράγεται ένα διάνυσμα x_flat με μέγεθος num_test_samples και τιμές από 0 έως 2L. Το διάνυσμα αυτό τροφοδοτεί το νευρωνικό δίκτυο network, μόνο για προς τα εμπρός τροφοδότηση (forward pass) και οι τιμές $\hat{u}(x)$ αποθηκεύονται, προκειμένου μέσω της συνάρτησης plot της βιβλιοθήκης matplotlib, να κατασκευαστεί η γραφική παράσταση η οποία προσεγγίζει την πραγματική συνάρτηση u(x). Για πιο περιγραφική απεικόνισή και πιο άμεση σύγκριση, κρίθηκε ωφέλιμο, στο ίδιο γράφημα να παρίσταται και η αναλυτική λύση της διαφορικής εξίσωσης. Παρακάτω παρατίθενται με την σειρά τα σημεία του κώδικα της main που ορίζονται τα δύο ΤΝΔ, τα διανύσματα εισόδου και εξόδου – στόχου και τέλος οι εντολές με τις οποίες αρχίζει η εκπαίδευση.

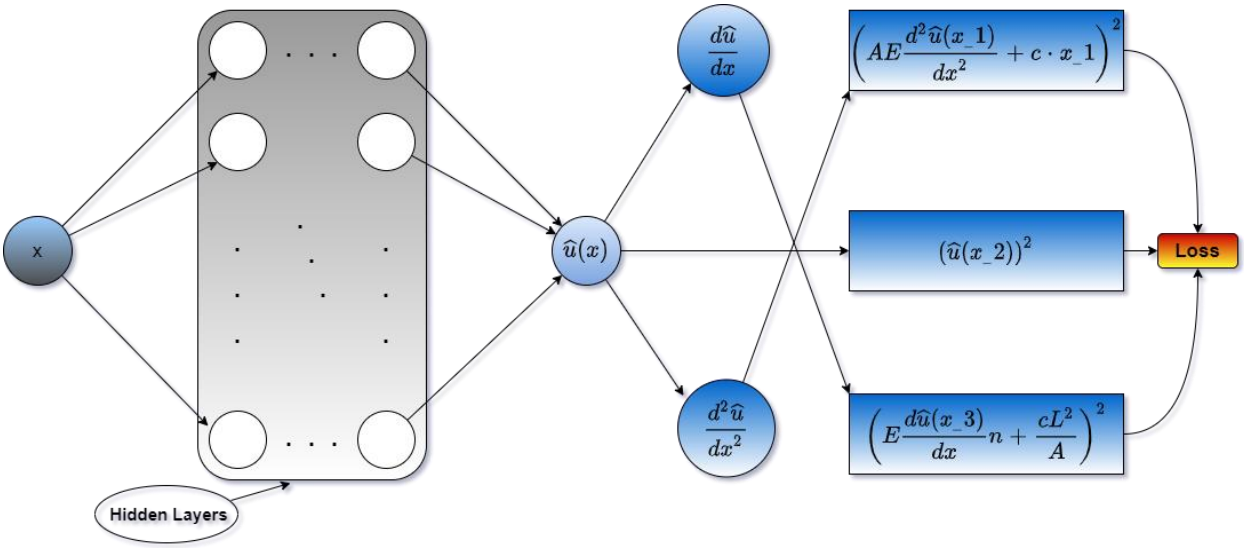
```
# build a core network model
network = Network.build()
# build a PINN model
pinn = PINN(network, E, A, c, n, L).build()


# create training input
x_1 = 2*L*np.random.rand(num_train_samples, 1)             # x_1 =  0 ~ 2*L
x_2 = np.zeros((num_train_samples, 1))                     # x_2 =  0
x_3 = 2*L*np.ones((num_train_samples, 1))                  # x_3 =  2*L
# create training output - target
u_1 = np.zeros((num_train_samples, 1))                     # u_1 = 0
u_2 = np.zeros((num_train_samples, 1))                     # u_2 = 0
u_3 = np.zeros((num_train_samples, 1))                     # u_3 = 0


lbfgs = L_BFGS_B(model=pinn, x_train=x_train, y_train=y_train)
lbfgs.fit()
```

Κατά την εκτέλεση του παραπάνω προγράμματος υπομονετικά πολλές φορές και για διαφορετικές παραμέτρους εκπαίδευσης όπως num_train_samples, αριθμό hidden layers και νευρώνων, επαναλήψεις ή παραμέτρους της συνάρτησης optimize.fmin_l_bfgs_b, το πρόβλημα λύθηκε με πολύ ικανοποιητικό σφάλμα και χρόνο περάτωσης ενώ το γράφημα που δημιουργήθηκε από την προσέγγιση, ήταν ιδιαίτερα ελπιδοφόρο για την πορεία της εργασίας. Το γράφημα αυτό παρατίθεται στο επόμενο κεφάλαιο το οποίο θα περιλαμβάνει συγκεντρωτικά τα αποτελέσματα από όλα τα προβλήματα που λύθηκαν. Εντούτοις, παρατηρήθηκε μια έλλειψη ευστάθειας ως αναφορά την επίδοση του προγράμματος στην προσέγγιση κατά το πέρας των εκτελέσεων. Λόγω της στοχαστικής φύσης της διαδικασίας εκπαίδευσης, ο αλγόριθμος συχνά αποτύγχανε να προσεγγίσει σωστά την συνάρτηση u(x) με ικανοποιητικό σφάλμα, παρόλο που η προσέγγιση είχε σχεδόν ίδια μορφή με την πραγματική λύση. Ένας παράγοντας που μπορεί να ευθύνεται για αυτό το γεγονός, ίσως είναι ότι ο αλγόριθμος L-BFGS συχνά εγκλωβιζόταν σε κάποιο τοπικό ελάχιστο με αποτέλεσμα η Loss function να διατηρεί σχεδόν την ίδια τιμή για αρκετές επαναλήψεις, γεγονός που προκαλούσε τον τερματισμό του αλγορίθμου. Για τον λόγο αυτό επιχειρήθηκε να αλλαχτεί ο αλγόριθμος βελτιστοποίησης L-BFGS με τον αλγόριθμο Adam του οποίου τα βήματα αναφέρονται στην παράγραφο 3.3. Το κριτήριο με το οποίο επιλέχτηκε ο συγκεκριμένος αλγόριθμος ήταν η ικανότητα του να μεταβάλει το βήμα του κατά την εκτέλεση των επαναλήψεών του. Από τα αποτελέσματα φάνηκε να λύθηκε το πρόβλημα της ευστάθειας καθώς μετά την αλλαγή, οι επιδόσεις του προγράμματος ως αναφορά την προσέγγιση ήταν πολύ κοντά η μια με την άλλη και ιδιαίτερα ικανοποιητικές όπως θα φανεί και από τα γραφήματα. Η παραπάνω αλλαγή θα εξηγηθεί προγραμματιστικά σε επόμενες υποπαραγράφους, στις

οποίες περιγράφονται τα προβλήματα της κάμψης σε δοκό καθώς σε αυτά τα προβλήματα εφαρμόστηκε εξ ολοκλήρου ο αλγόριθμος Adam για την εκπαίδευση.



Πίνακας 4.4.1.2 (Απεικόνιση των δύο δικτύων για την επίλυση του προβλήματος 1 – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net/)

4.4.2 Πρόβλημα 2: Δοκός σε κάμψη – Αμφιέρειστη Δοκός (Simply Supported Beam)

Πίνακας 4.4.2.1 (Αμφιέρειστη δοκός)

Η κάμψη είναι μια μείζονος σημασίας έννοια της Αντοχής των Υλικών, που χρησιμοποιείται στον σχεδιασμό πολλών δομικών στοιχείων και εξαρτημάτων μηχανών όπως είναι οι δοκοί οι οποίες θα συζητηθούν ιδιαίτερα στα επόμενα προβλήματα. Το ζητούμενο του προβλήματος 1 και των ακόλουθων που αφορούν την δοκό, είναι η εύρεση της συνάρτησης της ελαστικής γραμμής (elastic curve) y(x) η οποία περιγράφει την παραμόρφωση της δοκού σε κάθε σημείο x, κατά μήκος της. Ουσιαστικά η συνάρτηση y(x) παριστάνει την μορφή που αποκτά η δοκός λόγω της κάμψης.

Εάν σε μια δοκό ασκείται μια ροπή κάμψης (bending moment) M (Nm) όπου M = -P x με P (Newtons) ένα συγκεντρωμένο φορτίο (concentrated load), τότε από την θεωρία της στατικής μηχανικής είναι γνωστό ότι η διαφορική εξίσωση που περιγράφει το φαινόμενο της κάμψης είναι η εξής:

$$\frac{d^2y}{dx^2} = \frac{M(x)}{EI} \tag{82}$$

Ως E αναφέρεται το μέτρο ελαστικότητας και ως I αναφέρεται η ροπή αδράνειας της δοκού. Το γινόμενο EI είναι γνωστό ως ακαμψία (flexural rigidity).

Εάν όμως στην δοκό υπό μελέτη ασκείται ένα κατανεμημένο φορτίο (distributed load) w(x) (N/m) τότε η παραπάνω διαφορική εξίσωση θα πρέπει να διαμορφωθεί κατάλληλα. Για να συμβεί αυτό ορίζεται το μέγεθος V ως η τέμνουσα δύναμη (shear force) ενώ για αυτό το μέγεθος ισχύουν οι εξής σχέσεις (Beer, Johnston, DeWolf, & Mazurek, 2016).

$$\frac{dM}{dx} = \text{V} \tag{83}$$

$$\frac{dV}{dx} = \text{-w} \tag{84}$$

Κατά συνέπεια για τον σχηματισμό της διαφορικής εξίσωσης η οποία περιγράφει την μεταβολή της ελαστικής γραμμής μιας δοκού που υπόκειται σε κατανεμημένο φορτίο κάμψης, χρειάζεται να παραγωγιστεί 2 φορές ως προς x η εξίσωση (82).

$$\frac{d^3y}{dx^3} = \frac{d}{dx}\frac{M(x)}{EI} = \frac{V(x)}{EI} \tag{85}$$

$$\frac{d^4y}{dx^4} = \frac{d}{dx}\frac{V(x)}{EI} = -\frac{w(x)}{EI} \tag{86}$$

Ως εκ τούτου, η τελική διαφορική εξίσωση σε καθαρή μορφή που προκύπτει είναι η ακόλουθή:

$$\frac{d^4y}{dx^4} = -\frac{w(x)}{EI} \tag{87}$$

Στην συνέχεια πρέπει αν οριστούν οι οριακές συνθήκες ώστε να μπορεί να λυθεί η διαφορική εξίσωση. Εφόσον πρόκειται για αμφιέρειστη δοκό, τότε αν η δοκός έχει μήκος L, ισχύει ότι y(0) = 0, M(0) = 0, y(L) = 0 και M(L) = 0. Οι συνθήκες M(0) = 0 και M(L) = 0, μέσω της σχέσης (80) μεταφράζονται ως $\frac{d^2y(0)}{dx^2}$ = 0 και $\frac{d^2y(L)}{dx^2}$ = 0. Κατά συνέπεια το ολοκληρωμένο πρόβλημα προς επίλυση, δηλαδή η διαφορική εξίσωση και οι οριακές συνθήκες έχει την ακόλουθη μορφή.

$$\frac{d^4y}{dx^4} + \frac{w}{EI} = 0 \qquad x \in [0, L] \tag{88}$$

$$y(0) = 0 \tag{89}$$

$$y(L) = 0 \tag{90}$$

$$\frac{d^2y(0)}{dx^2} = 0 \tag{91}$$

$$\frac{d^2y(L)}{dx^2} = 0 \tag{92}$$

Στα προβλήματα που αφορούν την δοκό, έγινε προσπάθεια ώστε τα προβλήματα αυτά να ανταποκρίνονται στην πραγματικότητα για αυτό τον λόγο δόθηκαν αληθοφανής τιμές στις παραμέτρους w, E, I. Πιο

συγκεκριμένα, τα προβλήματα αυτά αφορούν W – Δοκό (Wide Flange Beam) με μέτρο ελαστικότητας E = 200 GPa (200 * 1e9 Pa), ροπή αδράνειας I = 0.000038929334 (m^4), μήκος L = 2.7 m και κατανεμημένο φορτίο w = 60 KN/m.

Για την αξιολόγηση της προσεγγιστικής λύσης που θα προέλθει από την μέθοδο PINNs, πέρα από την Loss function που αποτελεί έναν πολύ σημαντικό δείκτη, ένας επιπλέον δείκτης ο οποίος προσφέρει πολύ πιο άμεση σύγκριση, είναι η γραφική απεικόνιση της αναλυτικής λύσης στο ίδιο γράφημα με την προσεγγιστική λύση. Μετά από επίλυση του προβλήματος αρχικών τιμών της εξίσωσης, προκύπτει η αναλυτική λύση που αναγράφεται παρακάτω.

$$y(x) = \frac{w}{24\,EI}(-x^4 + 2\,L\,x^3 - L^3\,x) \tag{93}$$

Η προγραμματιστική υλοποίηση πραγματοποιήθηκε σε κώδικα παρόμοιας μορφής με τον κώδικα του προβλήματος 1, όμως όπως προαναφέρθηκε, αντικαταστάθηκε ο αλγόριθμος βελτιστοποίησης L-BFGS με τον αλγόριθμο Adam. Επιπρόσθετα έγιναν κάποιες απλοποιήσεις ώστε ο κώδικας να είναι πιο ευανάγνωστος, ενώ προστέθηκε μια Class με όνομα StopAtLossValue(Callback) στη οποία ορίζεται ως κριτήριο τερματισμού μια ελάχιστη τιμή της Loss function. Επιπλέον, προστέθηκε ένας μετρητής του χρόνου εκτέλεσης του προγράμματος ούτως ώστε να υπάρχει μια εικόνα του υπολογιστικού κόστους.

Πιο αναλυτικά, από το πρόγραμμα που επεξηγήθηκε παραπάνω, καταργήθηκαν τα modules network και optimizer. Η κατασκευή των 2 δικτύων πραγματοποιείται στο module pinn στο οποίο υπάρχει η Class PINN, όπου μέσω αυτής επιστρέφονται τα 2 δίκτυα. Το module layer παραμένει ίδιο, δηλαδή σε αυτό το module γίνεται ο υπολογισμός των παραγώγων του δικτύου network ως προς την είσοδο x, όπως και προηγούμενα. Για την καλύτερη ανάγνωση του κώδικα, τα δύο δίκτυα μετονομάστηκαν από network και pinn σε u_model και PINN_model αντίστοιχα. Τέλος στο νέο πρόγραμμα δεν υπάρχει το module optimizer καθώς για την εκπαίδευση χρησιμοποιείται η συνάρτηση compile και fit της βιβλιοθήκης Keras, οι οποίες εκτελούνται απευθείας στην main, χρησιμοποιώντας ως αλγόριθμο βελτιστοποίησης τον αλγόριθμο Adam. Παρακάτω παρατίθενται τα κύρια μέρη του κάθε module, συνοδευόμενα με τις αντίστοιχες γραμμές του κώδικα.

Στο module pinn μέσω της συνάρτησης build, κατασκευάζεται το δίκτυο u_model και το δίκτυο PINN_model. Για την κατασκευή του PINN_model θα χρειαστεί ο υπολογισμός των διανυσμάτων u_1, u_2, u_3 τα οποία αναπαριστούν τις σχέσεις που αποτελούν την διαφορική εξίσωση και τις οριακές συνθήκες. Τα διανύσματα αυτά είναι το u_1 = $\frac{d^4\hat{y}(x\_1)}{dx^4} + \frac{w}{EI}$ για x_1 ∈ [0, L], u_2 = $\hat{y}$(x_2) για x_2 = 0 ή x_2 = L και u_3 = $\frac{d^2\hat{y}(x\_3)}{dx^2}$ για x_3 = 0 ή x_3 = L. Στην συνέχεια τα διανύσματα x_i και u_i εισέρχονται σε 2 λίστες (lists) κατά τον ίδιο τρόπο όπως επεξηγήθηκε στο προηγούμενο πρόβλημα.

```
# compute gradients
u, d2u_dx2, d4u_dx4 = grads(x_1)
# equation output being zero
u_1 = d4u_dx4 + self.w/(self.E * self.I)
# initial condition output
u_2 = u_model(x_2)
u, d2u_dx2, d4u_dx4 = grads(x_3)
u_3 = d2u_dx2

# build the PINN model for beam's equation
return u_model, tf.keras.models.Model(
    inputs=[x_1, x_2, x_3], outputs=[u_1, u_2, u_3])
```

Το module layer έχει την ίδια μορφή όπως και στο προηγούμενο πρόβλημα με την διαφορά ότι σε αυτή την περίπτωση χρειάζεται ο υπολογισμός και της 4ης παραγώγου του δικτύου u_model. Η αντίστοιχες εντολές παρίστανται παρακάτω.

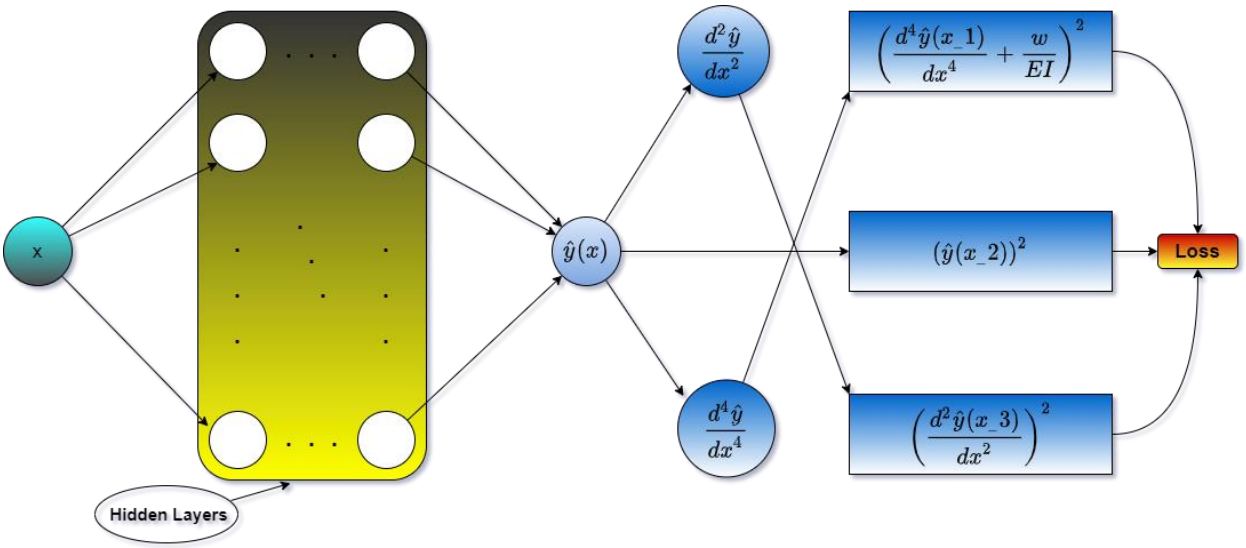
```
with tf.GradientTape() as g:
    g.watch(x)
    with tf.GradientTape() as gg:
        gg.watch(x)
        with tf.GradientTape() as ggg:
            ggg.watch(x)
            with tf.GradientTape() as gggg:
                gggg.watch(x)
                u = self.model(x)
            du_dx = gggg.gradient(u, x)
        d2u_dx2 = ggg.gradient(du_dx, x)
    d3u_dx3 = gg.gradient(d2u_dx2, x)
d4u_dx4 = g.gradient(d3u_dx3, x)
return u, d2u_dx2, d4u_dx4
```

Στην main αρχικά καλείται η συνάρτηση build από το module pinn, προκειμένου να οριστούν τα δύο δίκτυα u_model και PINN_model. Στην συνέχεια ορίζονται τα διανύσματα εισόδου και εξόδου – στόχου για το δίκτυο PINN_model του συγκεκριμένου προβλήματος, σύμφωνα με τα οποία θα πραγματοποιηθεί η εκπαίδευση. Τέλος καλούνται οι συναρτήσεις compile και fit με τις οποίες θα πραγματοποιηθεί η εκπαίδευση. Οι παραπάνω διεργασίες υλοποιούνται διαδοχικά, μέσω των εντολών που φαίνονται στις ακόλουθες γραμμές.

```
u_model, PINN_model = PINN(w, L, E, I).build()


# create training input
x_1 = L*np.random.rand(num_train_samples, 1)             # x_1 = 0 ~ L
x_2 = np.random.rand(num_train_samples, 1)
x_2 = L*np.round(x_2)                                    # x_2 = 0 or L
x_3 = np.random.rand(num_train_samples, 1)
x_3 = L*np.round(x_3)                                    # x_3 = 0 or L
# create training output
u_1 = np.zeros((num_train_samples, 1))                        # u_1 = 0
u_2 = np.zeros((num_train_samples, 1))                        # u_2 = 0
u_3 = np.zeros((num_train_samples, 1))                        # u_3 = 0
x_train = [x_1, x_2, x_3]
y_train_target = [u_1, u_2, u_3]


PINN_model.compile(optimizer='adam',
                   loss=tf.keras.losses.mse,
                   metrics=[tf.keras.metrics.mse],
                   )
model_history = PINN_model.fit(x_train, y_train_target, batch_size=64,
epochs=3000, callbacks=callbacks)
```



Πίνακας 4.4.2.2 (Απεικόνιση των δύο δικτύων για την επίλυση του προβλήματος 2 – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net/)

4.4.3 Πρόβλημα 3: Δοκός σε κάμψη – Δοκός Πρόβολος (Cantilever Beam)

Εικόνα 4.4.3.1 (Μοντέλο μιας δοκού προβόλου)

Η διαφορική εξίσωση και σε αυτό το πρόβλημα προφανώς είναι η ίδια με το προηγούμενο πρόβλημα εφόσον πρόκειται για δοκό σε κάμψη, όμως αλλάζουν οι οριακές συνθήκες. Για το πακτωμένο άκρο της δοκού ισχύει ότι y(0) = 0 και θ(0) [9] = 0 => $\frac{dy(0)}{dx}$ ενώ για το ελεύθερο άκρο της δοκού ισχύει ότι V(L) = 0 και M(L) = 0. Από τις σχέσεις (82), (83) και (85), οι δύο τελευταίες συνθήκες μπορούν να μεταφραστούν ως εξής:

$$M(L) = 0 \Rightarrow \frac{d^2y(L)}{dx^2} = 0 \quad (94)$$

$$V(L) = 0 \Rightarrow \frac{d^3y(L)}{dx^3} = 0 \quad (95)$$

Κατά συνέπεια η διαφορική εξίσωση και οι οριακές συνθήκες είναι οι εξής:

[9] Ως θ ορίζεται η γωνία που σχηματίζει η εφαπτομένη της ελαστικής γραμμής της δοκού σε ένα συγκεκριμένο σημείο με τον οριζόντιο άξονα. Επομένως θ = $\frac{dy}{dx}$.

$$\frac{d^4y}{dx^4} + \frac{w}{EI} = 0 \qquad x \in [0, L] \tag{96}$$

$$y(0) = 0 \tag{97}$$

$$\frac{dy(0)}{dx} = 0 \tag{98}$$

$$\frac{d^2y(L)}{dx^2} = 0 \tag{99}$$

$$\frac{d^3y(L)}{dx^3} = 0 \tag{100}$$

Το παραπάνω πρόβλημα λύθηκε αρχικά και ως πρόβλημα αρχικών τιμών για την εξαγωγή της αναλυτικής λύσης της y(x).

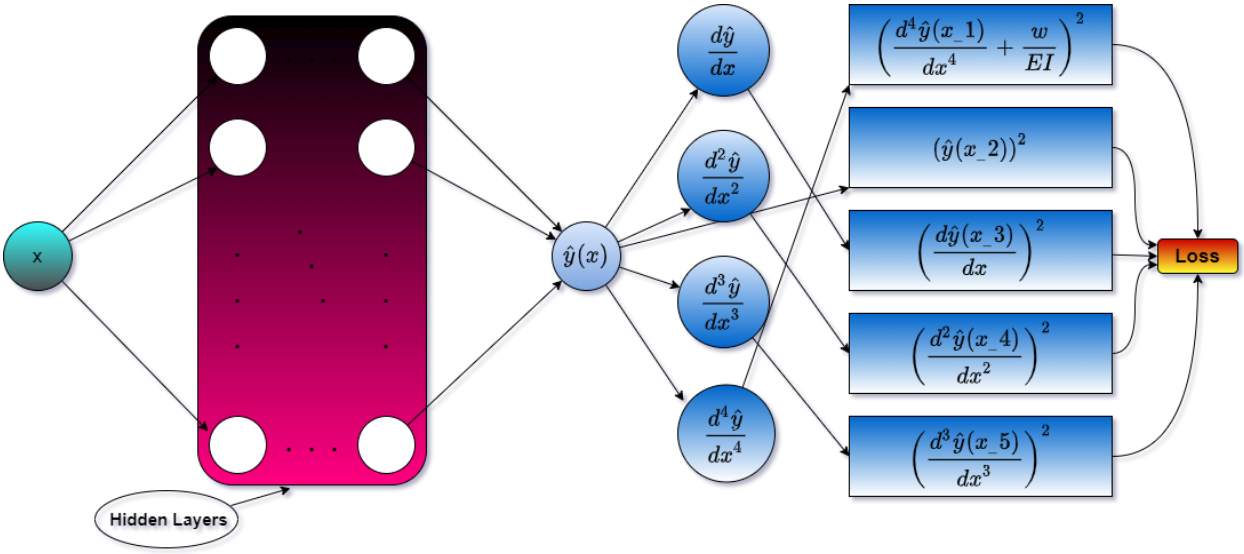
$$y(x) = \frac{w}{EI24}(- x^4 + 4L\, x^3 - 6L^2x^2) \tag{101}$$

Για την μεταφορά του παραπάνω προβλήματος στο πρόγραμμα της Python που αναπτύχθηκε στο πρόβλημα 2, χρειάζεται η αλλαγή των διανυσμάτων εισόδων εξόδων – στόχων στην main, ο επιπλέον υπολογισμός της $\frac{d^3\hat{y}}{dx^3}$ στο module layer καθώς και ο υπολογισμός των ανάλογων u_1, u_2, u_3, u_4, u_5 που αντιστοιχούν στις σχέσεις (96), (97), (98), (99), (100). Οι διεργασίες αυτές πραγματοποιούνται μέσω των εντολών των ακόλουθων γραμμών του κώδικα.

```
# create training input
x_1 = L*np.random.rand(num_train_samples, 1)            # x_1 = 0 ~ L
x_2 = np.zeros((num_train_samples, 1))                  # x_2 = 0
x_3 = np.zeros((num_train_samples, 1))                  # x_3 = 0
x_4 = L*np.ones((num_train_samples, 1))                 # x_4 = L
x_5 = L*np.ones((num_train_samples, 1))                 # x_5 = L
# create training output
u_1 = np.zeros((num_train_samples, 1))                  # u_1 = 0
u_2 = np.zeros((num_train_samples, 1))                  # u_2 = 0
u_3 = np.zeros((num_train_samples, 1))                  # u_3 = 0
u_4 = np.zeros((num_train_samples, 1))                  # u_4 = 0
u_5 = np.zeros((num_train_samples, 1))                  # u_5 = 0
```

```
with tf.GradientTape() as g:
    g.watch(x)
    with tf.GradientTape() as gg:
        gg.watch(x)
        with tf.GradientTape() as ggg:
            ggg.watch(x)
            with tf.GradientTape() as gggg:
                gggg.watch(x)
                u = self.model(x)
            du_dx = gggg.gradient(u, x)
        d2u_dx2 = ggg.gradient(du_dx, x)
    d3u_dx3 = gg.gradient(d2u_dx2, x)
d4u_dx4 = g.gradient(d3u_dx3, x)
return u, du_dx, d2u_dx2, d3u_dx3, d4u_dx4
```

```
# compute gradients
u, du_dx, d2u_dx2, d3u_dx3, d4u_dx4 = grads(x_1)
# equation output being zero
u_1 = d4u_dx4 + self.w/(self.E * self.I)
# initial condition output
u_2 = u_model(x_2)
u, du_dx, d2u_dx2, d3u_dx3, d4u_dx4 = grads(x_3)
u_3 = du_dx
u, du_dx, d2u_dx2, d3u_dx3, d4u_dx4 = grads(x_4)
u_4 = d2u_dx2
u_5 = d3u_dx3
```



Εικόνα 4.4.3.2 (Απεικόνιση των δύο δικτύων για την επίλυση του προβλήματος 3 – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net/)

4.4.4 Πρόβλημα 4: Δοκός σε κάμψη – Δοκός με πάκτωση στο ένα άκρο και κύλιση στο άλλο (Cantilever, Simply Supported Beam)

Πίνακας 4.4.4.1 (Μοντέλο μιας δοκού με πάκτωση στο αριστερό άκρο(x = 0) και κύλιση στο δεξί άκρο (x = L))

Σε αυτό το πρόβλημα οι οριακές συνθήκες που συνδέονται με τις στηρίξεις των 2 άκρων της δοκού είναι οι εξής. Στο αριστερό άκρο ισχύει ότι y(0) = 0 και θ(0) = 0 ενώ στο δεξί άκρο ισχύει ότι y(L) = 0 και M(L)

= 0. Σημαντικό είναι ότι στο δεξί άκρο δεν ισχύει η συνθήκη θ(L) = 0 καθώς στο δεξί άκρο δεν υπάρχει πάκτωση και κατά συνέπεια στο σημείο αυτό η δοκός μπορεί να έχει κλίση, όπως θα φανεί και από την ελαστική γραμμή στην γραφική παράσταση του επόμενου κεφαλαίου. Τονίζεται ότι σε αυτό το πρόβλημα το κατανεμημένο φορτίο ασκείται πάνω στην δοκό μόνο μεταξύ πάκτωσης και κύλισης. Η διαφορική εξίσωση και οι οριακές συνθήκες του παρόντος προβλήματος παρίστανται παρακάτω.

$$\frac{d^4y}{dx^4} + \frac{w}{EI} = 0 \quad (102)$$

$$y(0) = 0 \quad (103)$$

$$\frac{dy(0)}{dx} = 0 \quad (104)$$

$$y(L) = 0 \quad (105)$$

$$\frac{d^3y(L)}{dx^3} = 0 \quad (106)$$

Η αναλυτική λύση της διαφορικής εξίσωσης είναι η εξής:

$$y(x) = \frac{w}{24\,EI}\left(- x^4 + \frac{5}{2} L\, x^3 - \frac{3}{2} L^2\, x^2\right) \quad (107)$$

Όπως στο πρόβλημα 3 προηγουμένως, έτσι και στο συγκεκριμένο πρόβλημα, για την μεταφορά στο πρόγραμμα της Python το οποίο πραγματοποιεί την υλοποίηση της μεθόδου PINNs, χρειάζονται ορισμένες τροποποιήσεις στον κώδικα. Παρατηρώντας τις εξισώσεις (102), (103), (104), (105), (106) κρίνεται απαραίτητη η δημιουργία 5 διανυσμάτων εισόδου x_1, x_2, x_3, x_4, x_5 με μέγεθος όσα είναι και τα σημεία που θα χρειαστούν για την εκπαίδευση (num_train_samples). Παράλληλα πρέπει να δημιουργηθούν τα αντίστοιχα διανύσματα εξόδου – στόχου u_1, u_2, u_3, u_4, u_5 τα οποία όπως φαίνεται από τις παραπάνω εξισώσεις πρέπει να αποτελούνται από μηδενικά στοιχεία. Χρειάζεται προσοχή στο σημείο αυτό για κάτι που ισχύει σε όλα τα προβλήματα που επεξηγήθηκαν, καθώς δεν πρέπει να ταυτίζονται τα διανύσματα u_i της main με τα διανύσματα u_i του module pinn. Τα διανύσματα u_i της main αναφέρονται στις τιμές στόχου της εξόδου του νευρωνικού δικτύου PINN_model ενώ τα διανύσματα u_i του module pinn αναφέρονται στις τιμές που εξέρχονται από το νευρωνικό δίκτυο PINN_model κατά την διάρκεια της εκπαίδευσης. Σημαντικό επίσης είναι ότι στο συγκεκριμένο πρόβλημα χρειάζονται οι παράγωγοι $\frac{d\hat{y}}{dx}$, $\frac{d^2\hat{y}}{dx^2}$, $\frac{d^4\hat{y}}{dx^4}$, κατά συνέπεια η αντίστοιχη συνάρτηση που είναι υπεύθυνη για τον υπολογισμό των παραγώγων στο module layer θα πρέπει να επιστρέφει τις συγκεκριμένες παραγώγους στο module pinn. Οι παράγωγοι αυτοί θα χρειαστούν από το module pinn, για τον υπολογισμό των διανυσμάτων u_i τα οποία αναπαριστούν οι ακόλουθες ποσότητες:

$$u\_1 = \frac{d^4\hat{y}(x\_1)}{dx^4} + \frac{w}{EI} \quad (108)$$

$$u\_2 = \hat{y}(x\_2) \quad (109)$$

$$u\_3 = \frac{d\hat{y}(x\_3)}{dx} \quad (110)$$

$$u\_4 = \hat{y}(x\_4) \quad (111)$$

$$u\_5 = \frac{d^2\hat{y}(x\_5)}{dx^2} \quad (112)$$

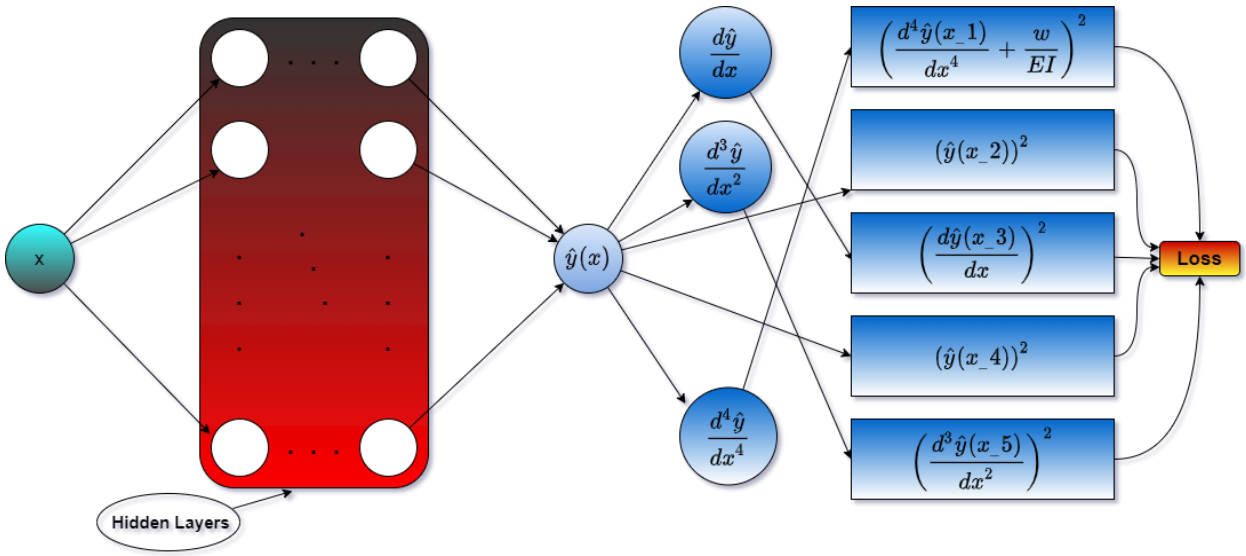
Οι παραπάνω διεργασίες πραγματοποιούνται στο πρόγραμμα μέσω των εντολών που παρατίθενται παρακάτω, με την σειρά που αναφέρθηκαν.

```
# create training input
x_1 = L*np.random.rand(num_train_samples, 1)             # x_1 = 0 ~ L
x_2 = np.zeros((num_train_samples, 1))                   # x_2 = 0
x_3 = np.zeros((num_train_samples, 1))                   # x_3 = 0
x_4 = L*np.ones((num_train_samples, 1))                  # x_4 = L
x_5 = L*np.ones((num_train_samples, 1))                  # x_5 = L
# create training output
u_1 = np.zeros((num_train_samples, 1))                   # u_1 = 0
u_2 = np.zeros((num_train_samples, 1))                   # u_2 = 0
u_3 = np.zeros((num_train_samples, 1))                   # u_3 = 0
u_4 = np.zeros((num_train_samples, 1))                   # u_4 = 0
u_5 = np.zeros((num_train_samples, 1))                   # u_5 = 0


return u, du_dx, d2u_dx2, d4u_dx4


grads = GradientLayer(u_model)


# compute gradients
u, du_dx, d2u_dx2, d4u_dx4 = grads(x_1)
# equation output being zero
u_1 = d4u_dx4 + self.w/(self.E * self.I)
# initial condition output
u_2 = u_model(x_2)
u, du_dx, d2u_dx2, d4u_dx4 = grads(x_3)
u_3 = du_dx
u, du_dx, d2u_dx2, d4u_dx4 = grads(x_4)
u_4 = u_model(x_4)
u_5 = d2u_dx2
```

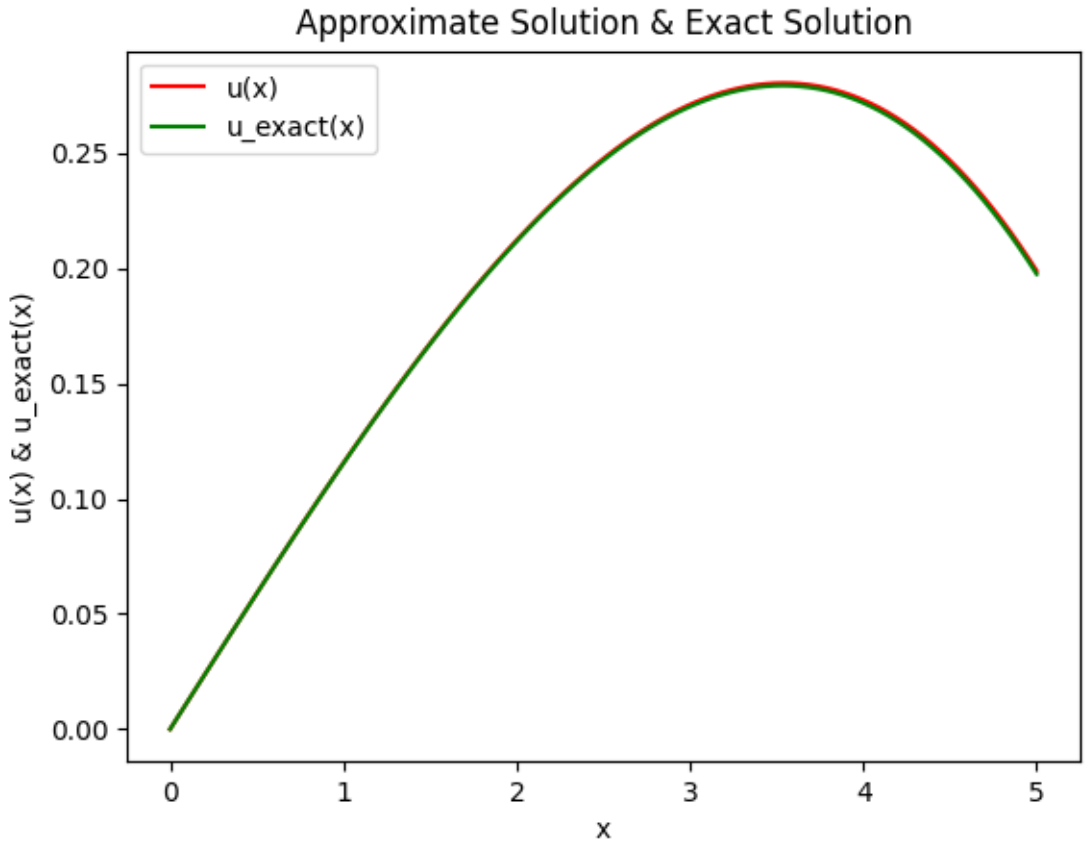


Πίνακας 4.4.4.2 (Απεικόνιση των δύο δικτύων για την επίλυση του προβλήματος 4 – Σχεδιάστηκε στο πρόγραμμα https://app.diagrams.net/)

Κεφάλαιο 5 – Αποτελέσματα

Στο κεφάλαιο αυτό, παρατίθενται συγκεντρωμένα τα αποτελέσματα από την επίλυση των τεσσάρων προβλημάτων τα οποία αναφέρθηκαν στην παράγραφο 4.4. Τα προβλήματα αυτά λύθηκαν αρκετές φορές με πολλές διαφορετικές παραμέτρους του προγράμματος όπως αριθμό hidden layers και νευρώνων, batch size, κύκλους εκπαίδευσης (epochs) όπως επίσης και μέγεθος δείγματος για εκπαίδευση (training sample – collocation points). Το πρόβλημα 1 λύθηκε με τον αλγόριθμο L-BFGS και με τον αλγόριθμο Adam δηλαδή λύθηκε και με το δεύτερο πρόγραμμα που επεξηγήθηκε στην παράγραφο 4.4, όχι μόνο με το αρχικό. Τα προβλήματα 2, 3 και 4 λύθηκαν μόνο με το πρόγραμμα που χρησιμοποιεί τον αλγόριθμο Adam. Σε όλες τις εκτελέσεις που θα αναφερθούν παρακάτω, ως συνάρτηση ενεργοποίησης (activation function) χρησιμοποιήθηκε η υπερβολική εφαπτομένη (hyperbolic tangent) tanh(x). Η δομή του κεφαλαίου θα περιλαμβάνει τέσσερεις παραγράφους, μια για το κάθε πρόβλημα, οπού η κάθε μια θα περιέχει ένα ενδεικτικό γράφημα επιτυχούς εκπαίδευσης με την αναλυτική και προσεγγιστική λύση μαζί, καθώς και έναν πίνακα με τα αποτελέσματα των εκτελέσεων του προγράμματος.

5.1 Πρόβλημα 1: Άσκηση εφελκυστικής δύναμης σε ράβδο



Εικόνα 5.1.1 (Γραφική παράσταση – Νο 8 από Πίνακα 5.1.1)

No	Layers	Training sample	Loss mse	Time(sec)
1	[15, 30, 60, 30, 15]	1000	0.009804	7167
2	[15, 30, 60, 30, 15]	1000	0.002842	4508
3	[15, 30, 60, 30, 15]	1000	0.001222	388
4	[15, 30, 60, 30, 15]	1000	0.004778	5150
5	[15, 30, 60, 30, 15]	1000	0.000596	5341
6	[15, 30, 60, 30, 15]	1000	0.002389	1323
7	[15, 30, 60, 30, 15]	1000	0.003372	3119
8	[20, 30, 60, 40, 20]	1000	0.000352	3307
9	[20, 30, 60, 40, 20]	1000	0.001479	3060

Πίνακας 5.1.1 (Πίνακας αποτελεσμάτων από την επίλυση του προβλήματος 1 με χρήση του αλγόριθμου L-BFGS)

No	Layers	Batch size	Epochs	Training sample	Loss mse	Time(sec)
1	[15, 30, 60, 30, 15]	16	1151	1000	0.008200	256
2	[30, 60, 30]	16	398	10000	0.008100	736
3	[30, 60, 30]	32	982	1000	0.007300	1015
4	[30, 60, 30]	32	1717	1000	0.009700	1798

Πίνακας 5.1.2 (Πίνακας αποτελεσμάτων από την επίλυση του προβλήματος 1 με χρήση του αλγορίθμου Adam)

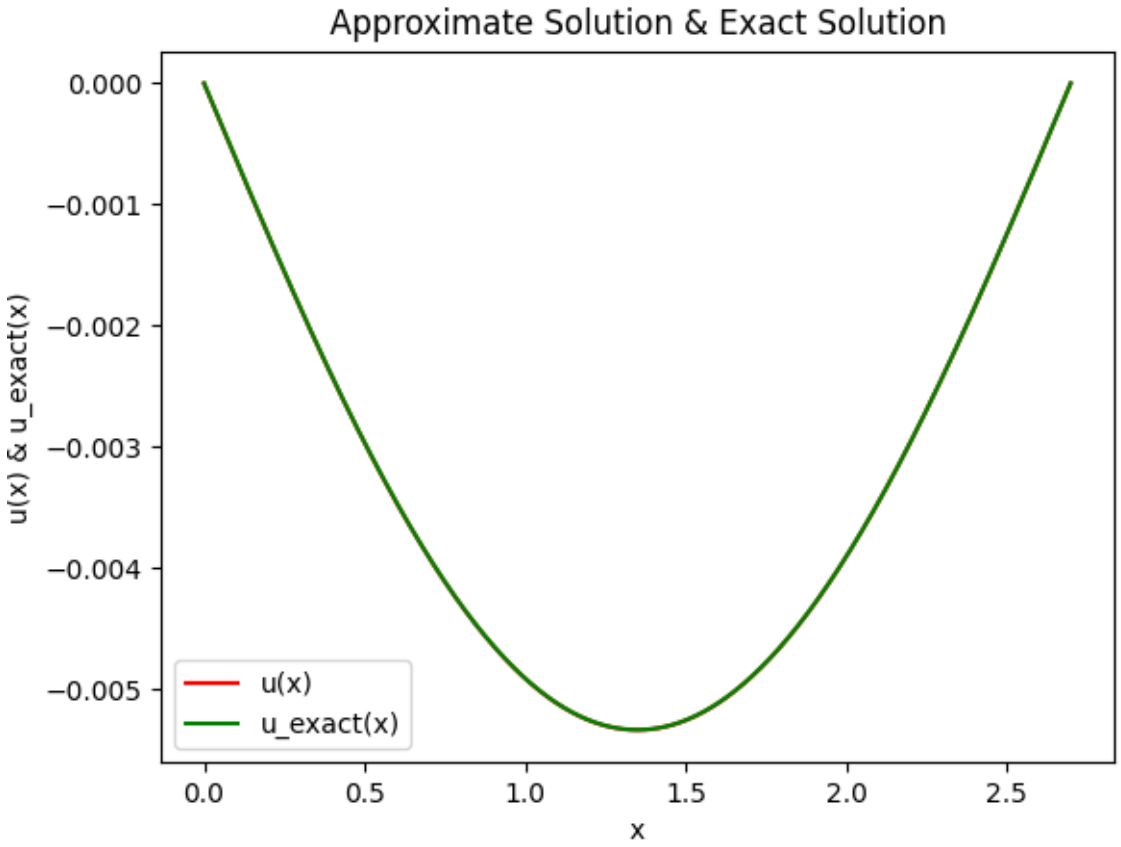
Παρατηρώντας τον πίνακα 5.1.1, αρχικά δίνεται η εντύπωση ότι το αρχικό πρόγραμμα με τον αλγόριθμο L-BFGS ως optimizer, επιφέρει πολύ ικανοποιητικές προσεγγίσεις και επιτυγχάνει τον σκοπό του. Εντούτοις η πραγματικότητα είναι αρκετά διαφορετική καθώς μεταξύ των επιτυχημένων εκτελέσεων του προγράμματος που παρατίθενται στον πίνακα 5.1.1, μεσολάβησαν αρκετές αποτυχημένες προσεγγίσεις με σφάλμα ανώτερο του 0.05 ή και αρκετά χειρότερο. Σε αυτές τις περιπτώσεις η γραφική παράσταση της προσεγγιστικής λύσης $\hat{u}(x)$ είχε αρκετά κοινή μορφή με την γραφική παράσταση της αναλυτικής λύσης $u(x)$, όμως φαινόταν αισθητά η διαφορά τους στο γράφημα καθώς δεν ταυτίζονταν οπτικά μεταξύ τους. Η έλλειψη ευστάθειας του συγκεκριμένου κώδικα έγινε αφορμή για την χρήση του αλγορίθμου Adam ως optimizer, όπως και για την κατάργηση των module network και module optimizer προκειμένου ο κώδικας να γίνει πιο ευανάγνωστος και πιο εύκολα διαχειρίσιμος.

Ο αλγόριθμος Adam φάνηκε να έχει πολύ καλύτερες και σταθερές επιδόσεις στην επίλυση της εξίσωσης καθώς κατά την εκτέλεση του νέου προγράμματος που αναπτύχθηκε, δεν σημειώθηκαν σχεδόν καθόλου μεγάλες διακυμάνσεις στο τελικό σφάλμα. Όπως μπορεί να παρατηρηθεί στον πίνακα 5.1.2, το σφάλμα δεν σημείωσε κάποια εντυπωσιακή βελτίωση σε σχέση με το προηγούμενο πρόγραμμα, όμως κατά την εκτέλεση παρατηρήθηκε ιδιαίτερη ευστάθεια, γεγονός που ήταν αρκετό ώστε να χρησιμοποιηθεί η ίδια μορφή του προγράμματος και στα επόμενα προβλήματα.

Ένας επιπλέον παράγοντας ο οποίος συνέβαλε στην ευστάθεια, στην ελαχιστοποίηση του χρόνου εκτέλεσης και παράλληλα στη μείωση του σφάλματος, ήταν η προσθήκη ενός κριτηρίου τερματισμού του αλγορίθμου. Το κριτήριο αυτό είναι η επίτευξή μιας ελάχιστης τιμής σφάλματος, η οποία όταν επιτευχθεί, ο αλγόριθμος της μεθόδου τερματίζεται και δεν εκτελούνται οι επόμενοι κύκλοι εκπαίδευσης. Συγκεκριμένα, ως κριτήριο τερματισμού τέθηκε η συνθήκη επίτευξής **loss function ≤ 1e-2** (0.01). Η προσθήκη αυτή έγινε με αφορμή την παρατήρηση ότι πολλές φορές, κατά την εκτέλεση του προγράμματος,

σε ενδιάμεσους κύκλους εκπαίδευσης επιτυγχάνονταν καλύτερο σφάλμα από ότι στον τελευταίο κύκλο εκπαίδευσης. Το γεγονός αυτό μπορεί να ερμηνευτεί με βάση την διαδικασία της εκπαίδευσης μέσω του αλγόριθμου Adam ή και οποιουδήποτε αλγόριθμου βελτιστοποίησης. Κατά την αναζήτηση του ολικού ελαχίστου της Loss function ο αλγόριθμος συχνά βρίσκεται σε τοπικά ελάχιστα. Όταν όμως δεν έχουν ολοκληρωθεί οι κύκλοι εκπαίδευσης ακόμα, τότε ο αλγόριθμος θα συνεχίσει την αναζήτηση. Στην πορεία, μπορεί να φτάσει ο τελευταίος κύκλος την στιγμή που αλγόριθμος βελτιστοποίησης βρίσκεται σε χειρότερο σημείο από όταν βρισκόταν στο τοπικό ελάχιστο που αναφέρθηκε προηγουμένως. Εάν το τοπικό ελάχιστο αυτό είναι επαρκές για το συγκεκριμένο πρόβλημα, τότε είναι πιο αποδοτικό να τερματίσει σε αυτό το σημείο ο αλγόριθμος παρά να συνεχίσει την αναζήτηση με αβέβαιο αποτέλεσμα και επιπλέον υπολογιστικό κόστος. Τα αποτελέσματα φαίνεται να επιβράβευσαν αυτήν την παρατήρηση εφόσον βελτιώθηκε επιπλέον η ευστάθεια του προγράμματος. Το κριτήριο τερματισμού εφαρμόστηκε σε όλες τις προσπάθειες του πίνακα 5.1.2.

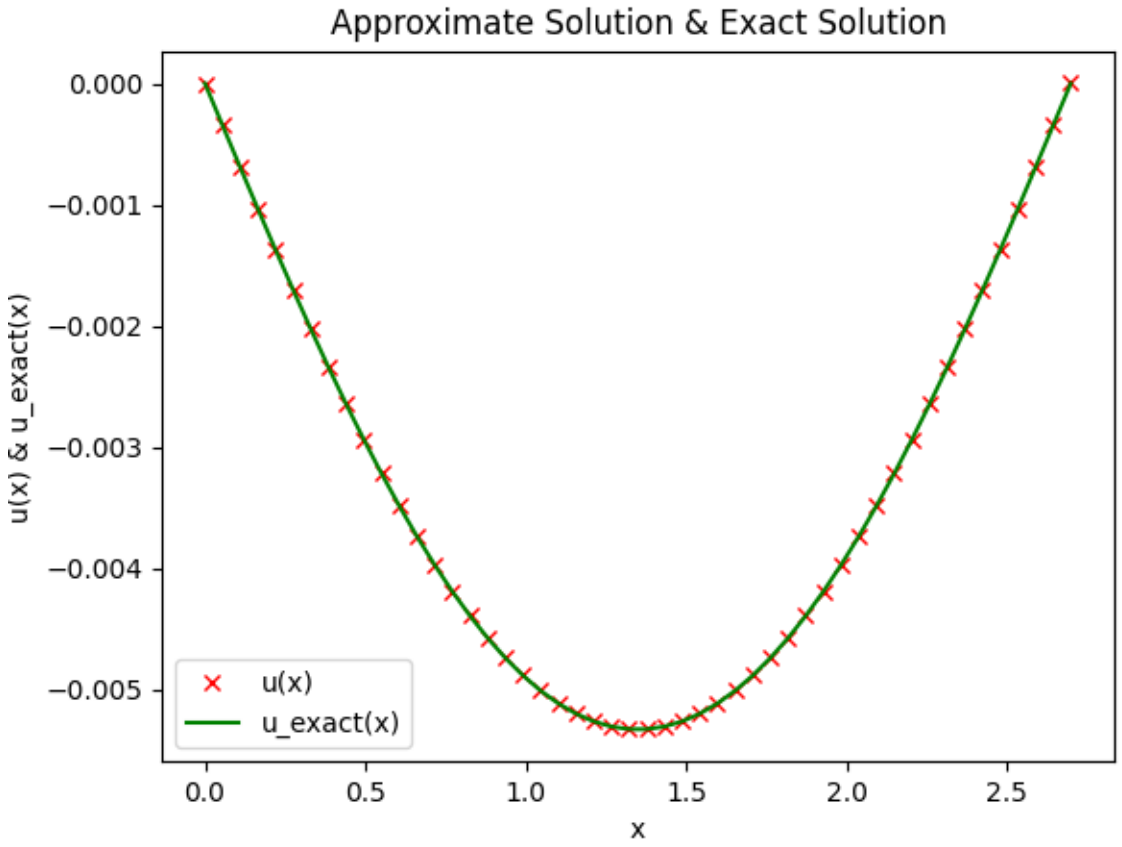
5.2 Πρόβλημα 2: Δοκός σε κάμψη – Αμφιέρειστη Δοκός (Simply Supported Beam)



Εικόνα 5.2.1 (Γραφική παράσταση – No 4 από Πίνακα 5.2.1)

No	Layers	Batch size	Epochs	Training sample	Loss mse	Time(sec)
1	[15, 30, 60]	128	3000	10000	1.84E-08	1763
2	[15, 30, 60]	128	870	10000	9.80E-11	491
3	[15, 30, 60, 30]	64	398	10000	9.96E-11	429
4	[15, 30, 60, 30, 15]	64	398	10000	6.74E-11	61
5	[15, 30, 60, 30, 15]	64	476	10000	9.67E-11	839
6	[15, 30, 60, 30, 15]	64	560	10000	9.97E-11	992
7	[15, 30, 15]	64	555	10000	8.52E-11	517
8	[15, 30, 15]	64	1456	10000	6.24E-11	1308
9	[15, 30, 15]	32	395	10000	7.75E-11	713
10	[40, 40]	32	1701	1000	9.25E-11	239

Πίνακας 5.2.1 (Πίνακας αποτελεσμάτων από την επίλυση του προβλήματος 2 με χρήση του αλγορίθμου Adam)



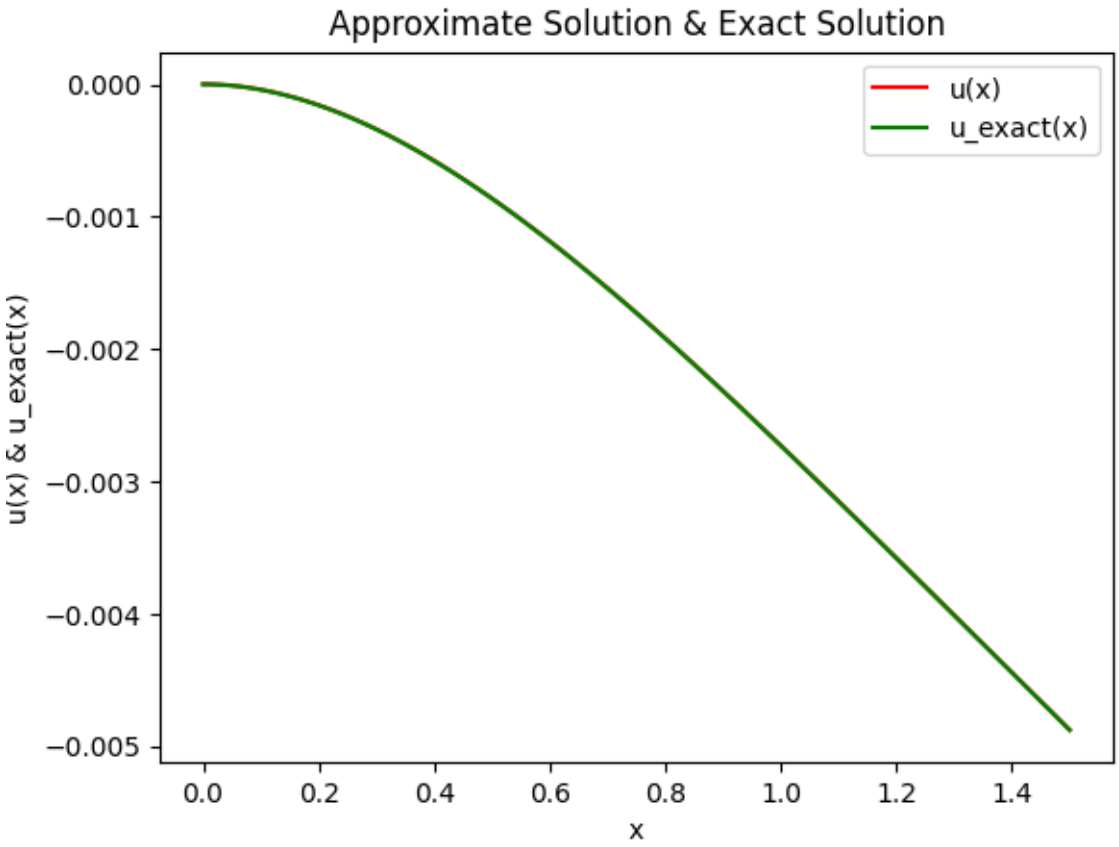
Εικόνα 5.2.2 (Ενδεικτικό γράφημα της αναλυτικής λύσης για την ελαστική γραμμή και πάνω της τα σημεία τα οποία προβλέπει το ΤΝΔ που εκπαιδεύτηκε με σφάλμα 9.9340e-11)

Παρατηρώντας τα αποτελέσματα, γίνεται πιο ευκρινές το κατά πόσο επηρεάζεται η τελική ακρίβεια και ο χρόνος εκτέλεσης του προγράμματος από παράγοντες όπως είναι η αρχιτεκτονική του δικτύου (hidden layers και αριθμός νευρώνων), ο αριθμός των σημείων για εκπαίδευση (training sample) καθώς και το μέγεθος των batches με τα οποία χωρίζεται το δείγμα. Συμβαίνει το αναμενόμενο, δηλαδή με την αύξηση των hidden layers και των νευρώνων, αυξάνεται ο χρόνος εκτέλεσης του κάθε κύκλου εκπαίδευσης λόγω των περισσότερων πράξεων. Παράλληλα αν παρατηρηθεί προσεκτικά η γραμμή 8 και 9 του πίνακα 5.2.1, διακρίνεται εύκολα πως με την μείωση του batch size για ίδια ακριβώς αρχιτεκτονική, αυξήθηκε κατακόρυφα ο χρόνος εκτέλεσης ανά κύκλο εκπαίδευσης. Το γεγονός αυτό συμβαίνει διότι με την μείωση του batch size, το δείγμα των δεδομένων για εκπαίδευση χωρίζεται σε περισσότερα batches και κατά συνέπεια σε έναν κύκλο εκπαίδευσης ο αλγόριθμος backpropagation εκτελείται περισσότερες φορές και οι παράμετροι ενημερώνονται πιο συχνά από ότι θα συνέβαινε στην περίπτωση με μεγαλύτερο batch size. Επιπροσθέτως, ιδιαίτερη επίδραση στον χρόνο εκτέλεσης ανά κύκλο, φαίνεται πως επιφέρει το μέγεθος του δείγματος για εκπαίδευση (training sample), όπου παρατηρήθηκε σημαντική αύξηση της ταχύτητας ολοκλήρωσης του κάθε κύκλου για μικρότερο δείγμα, ενώ συγχρόνως φάνηκε πως η αύξηση του μεγέθους δείγματος επιφέρει αύξηση της τελικής ακρίβειας (μικρότερο loss).

Τέλος, παρατηρήθηκε πως λειτούργησε ιδιαίτερα θετικά το κριτήριο τερματισμού, το οποίο σε όλα τα προβλήματα που αφορούν την δοκό τέθηκε ως **loss function ≤ 1e-10**. Στην προσπάθεια της πρώτης γραμμής του πίνακα 5.2.1, δεν εφαρμόζεται το κριτήριο με αποτέλεσμα να εκτελούνται όλοι οι κύκλοι

εκπαίδευσης που έχουν οριστεί (3000), ενώ σε όλες τις υπόλοιπές προσπάθειές εφαρμόζεται το κριτήριο με αισθητή βελτίωση της τελικής loss function και πιο σύντομη σύγκλιση του προγράμματος. Από τις προσπάθειες αυτές, διακρίνεται χαρακτηριστικά πως στη $10^{η}$, με hidden layers = [40, 40], χρειάστηκε να εκτελεστούν περισσότεροι κύκλοι εκπαίδευσης από τις υπόλοιπες προκειμένου να επέλθει τερματισμός λόγω του κριτηρίου. Αυτό οφείλεται στο γεγονός πως το συγκεκριμένο ΤΝΔ είχε λιγότερες παραμέτρους (trainable parameters) από τα ΤΝΔ των υπόλοιπων προσπάθειών.

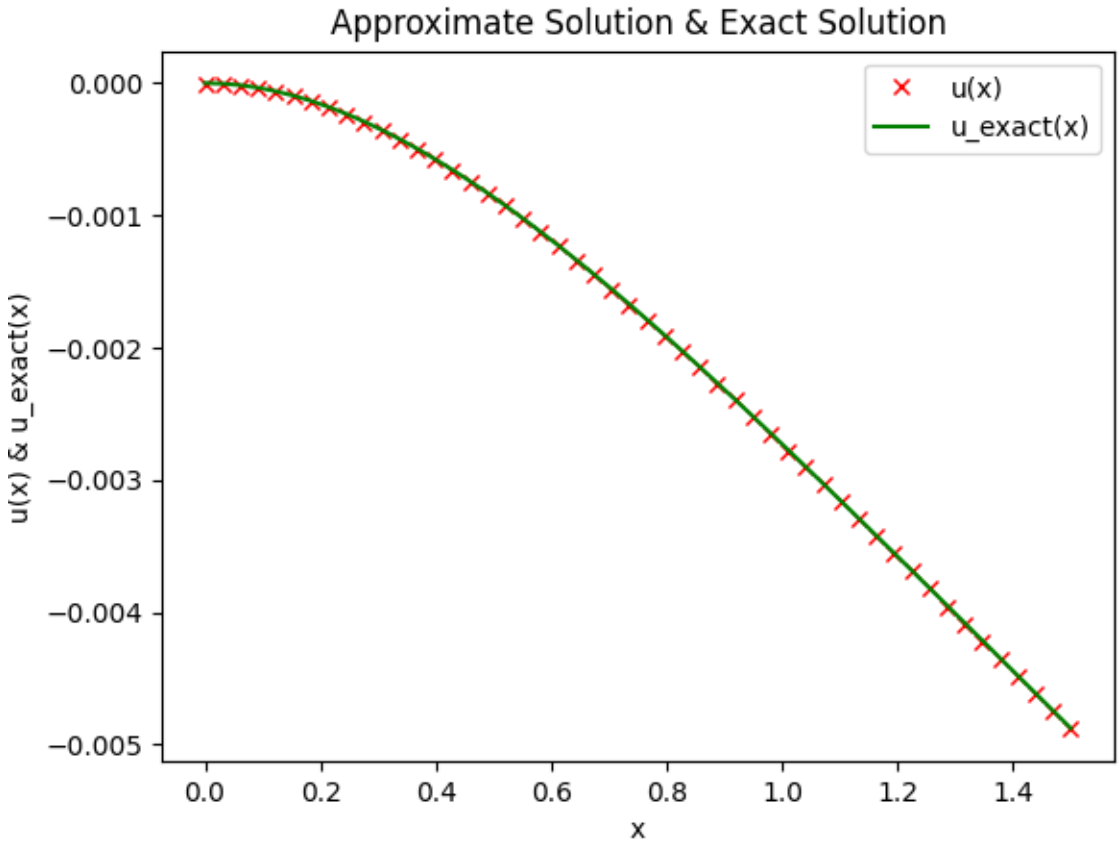
5.3 Πρόβλημα 3: Δοκός σε κάμψη – Δοκός Πρόβολος (Cantilever Beam)



Εικόνα 5.3.1 (Γραφική παράσταση – Νο 7 από Πίνακα 5.3.1)

No	Layers	Batch size	Epochs	Training sample	Loss mse	time(sec)
1	[15, 30, 60]	64	1000	1000	8.45E-10	364
2	[15, 30, 60]	32	1000	10000	6.66E-11	2011
3	[15, 30, 60]	32	1000	1000	2.47E-07	509
4	[15, 30, 60]	32	3000	1000	6.70E-10	720
5	[15, 30, 60]	32	4000	1000	6.24E-09	1020
6	[15, 30, 60]	32	1000	5000	4.15E-08	1085
7	[15, 30, 60]	32	1000	10000	1.60E-11	1860
8	[15, 30, 60]	32	900	10000	3.16E-08	1743
9	[15, 30, 60]	32	1000	10000	1.58E-10	1509
10	[15, 30, 60]	32	702	10000	7.44E-11	1206

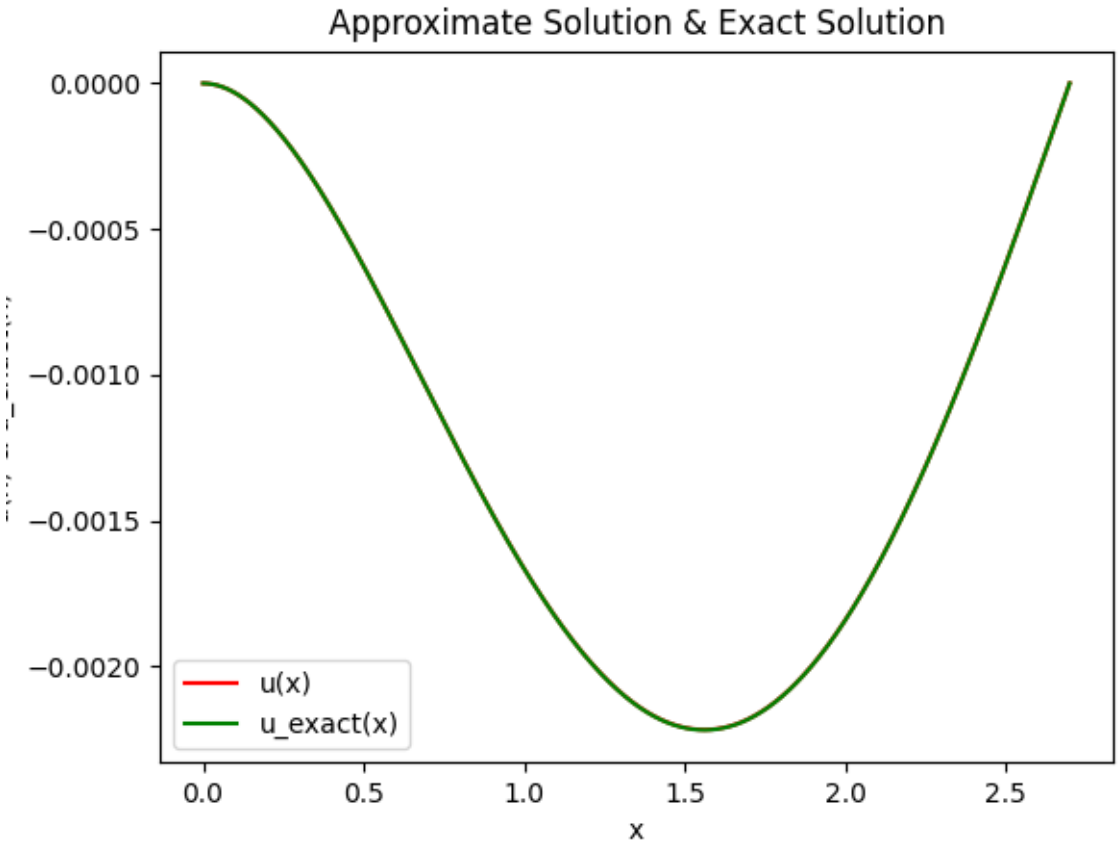
Πίνακας 5.3.1 (Πίνακας αποτελεσμάτων από την επίλυση του προβλήματος 3 με χρήση του αλγορίθμου Adam)



Εικόνα 5.3.2 (Ενδεικτικό γράφημα της αναλυτικής λύσης για την ελαστική γραμμή και πάνω της τα σημεία τα οποία προβλέπει το ΤΝΔ που εκπαιδεύτηκε με σφάλμα 8.7555e-11)

Στο παρόν πρόβλημα, το κριτήριο τερματισμού εφαρμόστηκε μόνο στην $10^{η}$ προσπάθεια που φαίνεται στον πίνακα 5.2.1 όπου σε αυτή την περίπτωση επετεύχθη «οικονομία» στους κύκλους εκπαίδευσης και υψηλή τελική ακρίβεια. Στις υπόλοιπες 9 προσπάθειες του πίνακα 5.2.1 δεν εφαρμόστηκε το κριτήριο τερματισμού και ως εκ τούτου εκτελέσθηκαν όλοι οι κύκλοι εκπαίδευσης που είχαν οριστεί, ενώ η τελική ακρίβεια διαφέρει τάξη μεγέθους από προσπάθεια σε προσπάθεια. Φαίνεται ότι στις πρώτες 9 προσπάθειες, επετεύχθη μερικές φορές η επιθυμητή ακρίβεια σχεδόν τυχαία. Μάλιστα αν γίνει μια προσεκτική παρατήρηση του πίνακα 5.2.1, φαίνεται πως στις προσπάθειες που δεν εφαρμόστηκε κανένα κριτήριο τερματισμού, οι μοναδικές 2 περιπτώσεις εξ αυτών, που προέκυψε η επιθυμητή ακρίβεια αφορούν την εκπαίδευση με 10000 σημεία.

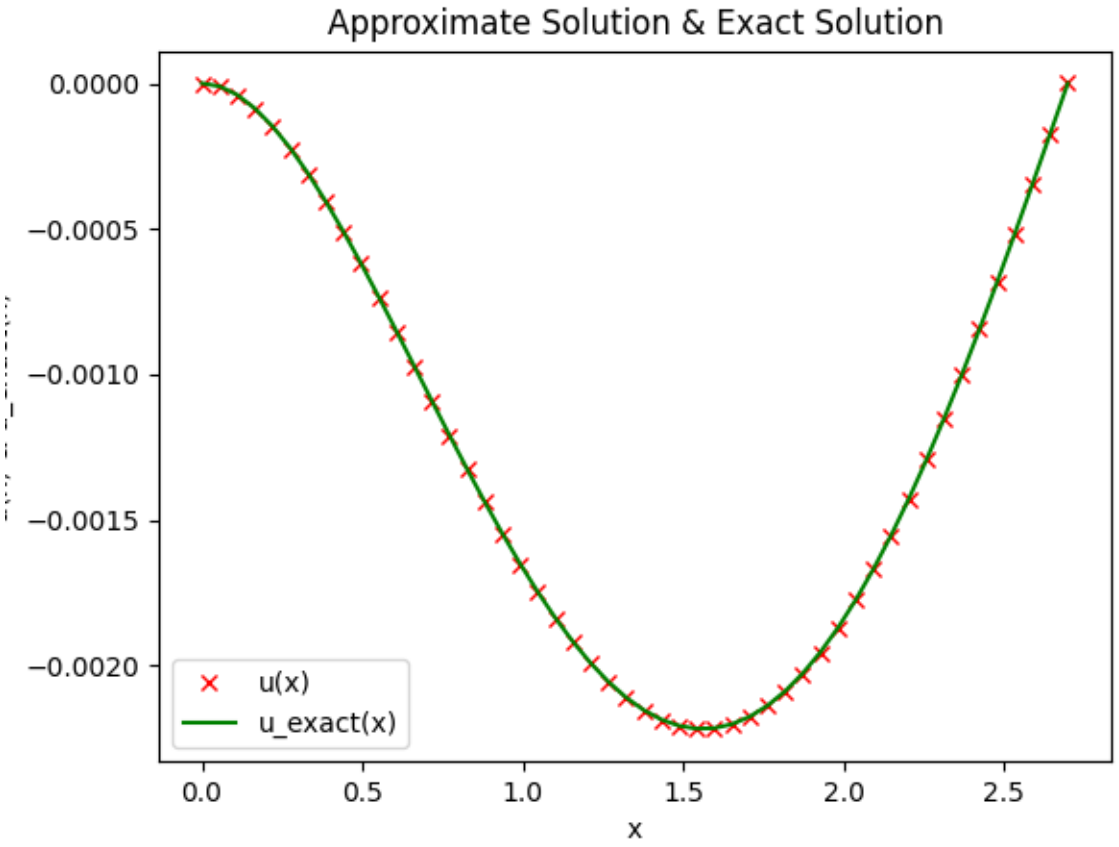
5.4 Πρόβλημα 4: Δοκός σε κάμψη – Δοκός με πάκτωση στο ένα άκρο και κύλιση στο άλλο (Cantilever, Simply Supported Beam)



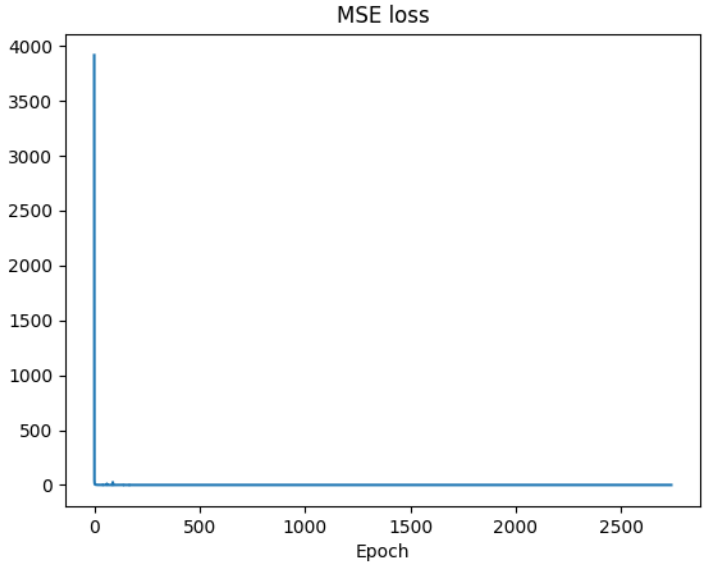
Εικόνα 5.4.1 (Γραφική παράσταση – Νο 1 από Πίνακα 5.4.1)

No	Layers	Batch size	Epochs	Training sample	Loss mse	Time(sec)
1	[15, 30, 60]	32	676	10000	7.70E-11	1504
2	[15, 30, 60]	32	3000	1000	1.93E-08	695
3	[15, 30, 60]	32	2446	1000	9.82E-11	608
4	[15, 30, 60]	16	2519	1000	9.99E-11	1036
5	[15, 30, 60]	64	3000	1000	1.11E-08	414
6	[15, 30, 60]	64	789	10000	7.98E-11	1002
7	[25, 50, 25]	64	900	10000	9.09E-11	1054
8	[25, 50, 25]	128	1021	10000	9.96E-11	1555
9	[25, 50, 25]	128	3000	1000	1.04E-07	215
10	[25, 50, 25]	64	3000	1000	6.84E-10	302
11	[25, 50, 50, 25]	16	2083	1000	9.26E-11	1053

Πίνακας 5.4.1 (Πίνακας αποτελεσμάτων από την επίλυση του προβλήματος 4 με χρήση του αλγορίθμου Adam)



Εικόνα 5.4.2 (Ενδεικτικό γράφημα της αναλυτικής λύσης για την ελαστική γραμμή και πάνω της τα σημεία τα οποία προβλέπει το ΤΝΔ που εκπαιδεύτηκε με σφάλμα 7.4316e-11)



Εικόνα 5.4.3 (Ενδεικτικό γράφημα της πορείας της Loss function για το πρόβλημα 4)

Στο 4ο πρόβλημα, το κριτήριο τερματισμού εφαρμόστηκε σε όλες τις προσπάθειες του πίνακα 5.4.1. Γίνεται πλέον εμφανές πως η εκπαίδευση πραγματοποιείται ορθότερα και πιο αποτελεσματικά στην περίπτωση που χρησιμοποιούνται πάνω από 5000 σημεία κατά μήκος της δοκού και ιδανικά 10000. Φαίνεται ότι ακόμα και με το κριτήριο τερματισμού, στην περίπτωση που χρησιμοποιούνται 1000 σημεία για εκπαίδευση, εκτελούνται όλοι οι κύκλοι εκπαίδευσης που έχουν οριστεί (3000) και δεν επιτυγχάνεται η επιθυμητή ακρίβεια. Χαρακτηριστικά η παραπάνω παρατήρηση επιβεβαιώνεται ανατρέχοντας στην 1η και 2η γραμμή του πίνακα 5.4.1 όπου στο ίδιο ΤΝΔ, για ίδιο μέγεθος batch, η εκπαίδευση με 10000 σημεία επιφέρει σύγκλιση σε 676 epochs με σφάλμα 7.70E-11, ενώ η εκπαίδευση με 1000 σημεία οδηγεί στην εξάντληση των ορισμένων epochs δίχως την επίτευξη της επιθυμητής ακρίβειας. Φυσικά η εκπαίδευση με 10000 σημεία στοιχίζει σε χρόνο εκτέλεσης όπως φαίνεται στις γραμμές 1 και 2 του πίνακα, με σχεδόν 14 λεπτά επιπλέον από την περίπτωση των 1000 σημείων.

Τέλος αξίζει να αναφερθεί μια επιπλέον παρατήρηση για την επίδραση του batch size στην ακρίβεια και στον χρόνο εκτέλεσης. Μελετώντας τις 3 τελευταίες προσπάθειες 9, 10 και 11 του πίνακα 5.4.1, επιβεβαιώνεται για άλλη μια φορά πως με την μείωση του batch size γίνεται μια πιο «λεπτομερής ανάγνωση» του δείγματος από το ΤΝΔ, γεγονός που φυσικά έχει αντίκτυπο στον χρόνο εκτέλεσης του προγράμματος το οποίο ολοκληρώνεται βραδύτερα. Εν προκειμένω, μιλώντας και στις 3 περιπτώσεις για δείγμα 1000 σημείων, στις περιπτώσεις των batch size = 128 και batch size = 64, εξαντλήθηκαν οι 3000 κύκλοι εκπαίδευσης χωρίς να επιτευχθεί η επιθυμητή ακρίβεια. Στη περίπτωση του batch size = 64 επιτυγχάνεται σαφώς καλύτερη ακρίβεια (loss mse = 6.84E-10) έναντι της περίπτωσης με batch size = 128 (loss mse = 1.04E-07). Στην 11η προσπάθεια, ακόμα και με 1000 σημεία για εκπαίδευση αλλά με batch size = 16, τελικά το πρόγραμμα τερματίζει πιο νωρίς από τους 3000 κύκλους, με σφάλμα loss mse = 9.26E-11. Ο χρόνος περάτωσης του προγράμματος στις 3 αυτές προσπάθειες φαίνεται να αυξάνεται καθώς μειώνεται το μέγεθος των batches όπως ήταν αναμενόμενο.

Κεφάλαιο 6 – Παρατηρήσεις και Συμπεράσματα

Με αυτή την διπλωματική εργασία, έγινε μια προσπάθεια κατανόησης της μεθόδου και παράθεσης του τρόπου λειτουργίας της με αναλυτικό τρόπο. Πρωταρχικός στόχος ήταν η εφαρμογή της μεθόδου σε βασικές εξισώσεις της στατικής μηχανικής, ώστε να παρατηρηθούν οι δυνατότητες της μεθόδου σε τέτοιου είδους προβλήματα αλλά και το κατά πόσο εύκολη είναι η πρόσβαση στην ενασχόληση με την μέθοδο από κάποιον που δεν έχει ισχυρό υπόβαθρο στα εφαρμοσμένα μαθηματικά ή στην επιστήμη υπολογιστών. Ένας επιπλέον σημαντικός στόχος ήταν, η εργασία αυτή να αποτελέσει προθάλαμο για πολλές επόμενες, αλλά συγχρόνως και ένα έμπιστο εγχειρίδιο στην ελληνική γλώσσα στο οποίο θα μπορεί να ανατρέξει με ασφάλεια κάποιος ο οποίος επιθυμεί να εισαχθεί στο θέμα των physics informed neural networks. Τέλος καίριο μέλημα ήταν επίσης η παρατήρηση τυχόν αστοχιών ή σημείων που χρειάζονται προσοχή και επιπρόσθετη μελέτη, προκειμένου αυτά να είναι γνωστά στον επόμενο που θα ασχοληθεί με το θέμα, να εξοικονομηθεί χρόνος και να γίνει πιο αποδοτική η μελέτη. Η εκτέλεση και η συγγραφή της εργασίας πραγματοποιήθηκε με γνώμονα την εκπλήρωση των παραπάνω στόχων με τον καλύτερο δυνατό τρόπο, με σεβασμό στον αναγνώστη και στους ερευνητές που έχουν ασχοληθεί με το θέμα.

Κατά την καταγραφή του τρόπου λειτουργίας των ΤΝΔ αλλά και της μεθόδου PINNs έγινε μια πιο ουσιαστική κατανόηση και αναγνώριση των σημείων που απαιτούν επιπλέον μελέτη. Η διαδικασία της επεξήγησης ενός φαινομένου ή μιας μεθόδου είναι ο καλύτερος τρόπος ώστε να κατανοήσει ο ίδιος ο αφηγητής αυτό που πάει να εξηγήσει. Είναι δεδομένο ότι ‘για να καταλάβεις κάτι σε βάθος, πρέπει να μπορείς να το εξηγήσεις σε κάποιον άλλον’. Παράλληλα, μέσω αυτής της διαδικασίας καλύπτονται κενά που υπήρχαν, τα οποία δεν είχαν αναγνωριστεί και με αυτόν τον τρόπο δίνεται η δυνατότητα βελτίωσης της εφαρμογής της μεθόδου, διορθώνοντας τον κώδικά ή προσθέτοντας επιπλέον κομμάτια. Κατά συνέπεια η συγγραφή της θεωρίας είναι καίριας σημασίας διαδικασία καθώς είναι και ένας δείκτης για το αν έχει γίνει σωστή κατανόηση.

Όσον αφορά την εφαρμογή της μεθόδου, τα αποτελέσματα δείχνουν ιδιαίτερα ικανοποιητικά και ελπιδοφόρα καθώς κρίνοντας από τους πίνακες και τα γραφήματα του κεφαλαίου 5, το σφάλμα της προσέγγισης είναι σημαντικά μικρό (1e-11). Αρχικά κατά την τροποποίηση του κώδικα προέκυψαν αρκετές δυσκολίες, ως εκ τούτου τα αποτελέσματα δεν ήταν καθόλου ικανοποιητικά καθώς το σφάλμα παρέμενε σημαντικά μεγάλο. Από την στιγμή όμως που λύθηκε το πρώτο πρόβλημα και μάλιστα με πολύ μικρό σφάλμα, η μεταφορά του επόμενου προβλήματος στο πρόγραμμα δεν είχε ιδιαίτερες δυσκολίες. Η κατάσταση βελτιώθηκε πολύ αισθητά με την αντικατάσταση του αλγόριθμου L-BFGS με τον αλγόριθμο Adam καθώς και με την απλοποίηση του κώδικα, χρησιμοποιώντας την συνάρτηση εκπαίδευσης του Keras. Από το σημείο αυτό και μετά, τροποποιώντας τις παραμέτρους εκπαίδευσης του προγράμματος (epoch, batch size κ.α) γίνεται δυνατή η επίτευξη σύγκλισης. Ως συμπέρασμα μετά το πέρας της εργασίας είναι πως η μέθοδος επιφέρει επαρκή ακρίβεια στα προβλήματα που λύθηκαν και ο χρόνος εκπαίδευσης δεν είναι απαγορευτικός.

Η επόμενη παρατήρηση είναι ότι οι περισσότεροι ερευνητές που ασχολούνται με το θέμα, προέρχονται από τα επιστημονικά πεδία των εφαρμοσμένων μαθηματικών και φυσικής καθώς και της επιστήμης υπολογιστών, εντούτοις φαίνεται ότι η ενασχόληση με το θέμα μπορεί να γίνει και από κάποιον που δεν προέρχεται από αυτά τα πεδία αποκλειστικά. Η μέθοδος αποτελεί μια προέκταση την τεχνητής νοημοσύνης και συγκεκριμένα των νευρωνικών δικτύων. Επιπροσθέτως, η μέθοδος για να αναπτυχθεί, απαιτούσε ισχυρές γνώσεις αριθμητικής ανάλυσης, θεωρίας βελτιστοποίησης και φυσικά προγραμματισμού. Όπως φαίνεται από την σύντομη ιστορική αναδρομή, χρειάστηκαν σχεδόν δύο δεκαετίες από τις πρώτες ολοκληρωμένες προτάσεις μέχρι την εγκαθίδρυση της μεθόδου και την αποτελεσματική εφαρμογή της.

Αυτό οφείλεται στο γεγονός ότι στο μεσοδιάστημα χρειάστηκε να γίνει έρευνα στα κομμάτια που υποστηρίζουν την εφαρμογή της μεθόδου και ιδιαίτερα την ανάπτυξή κατάλληλων προγραμματιστικών πακέτων και βιβλιοθηκών μαζί με το αντίστοιχο hardware που να τα υποστηρίζει. Αντιθέτως η εφαρμογή και δοκιμή της μεθόδου σε διαφόρων ειδών προβλήματα, ίσως μπορεί να γίνει και από κάποιον που δεν συμμετείχε στην ανάπτυξη της μεθόδου αλλά κατανοεί πλήρως την λειτουργία των ΤΝΔ, την Γραμμική Άλγεβρα και τον Απειροστικό Λογισμό που τα διέπουν. Παράλληλα κάποιος που θα ασχοληθεί με το θέμα είναι σημαντικό να έχει κάποια άνεση με τον αντικειμενοστραφή προγραμματισμό και να μπορεί να αναπτύξει κώδικα υλοποίησης απλών ΤΝΔ. Μπορεί λοιπόν να εξαχθεί το ελπιδοφόρο συμπέρασμα ότι μπορεί να ασχοληθεί κάποιος με το θέμα, χωρίς να είναι απόλυτα περιοριστικό το γνωστικό του υπόβαθρο, αρκεί φυσικά να πράξει τα απαραίτητα βήματα αρχικά πριν την ενασχόληση του με την μέθοδο PINNs, προκειμένου να κατανοήσει τα βασικά και να καλύψει τα κενά του.

Στο σημείο αυτό κρίνεται ωφέλιμο να αναφερθούν κάποιες επιπλέον παρατηρήσεις που έγιναν καθ' όλη την διάρκεια της εξέλιξης της εργασίας που πιθανώς θα βοηθήσουν κάποιον που θα ασχοληθεί μελλοντικά. Παρ' όλο που όπως αναφέρθηκε στην παραπάνω παράγραφο, δεν φάνηκε να είναι περιοριστικό το γνωστικό υπόβαθρο, είναι σίγουρο ότι αν δεν αναγνωριστούν οι αδυναμίες και δεν καλυφθούν, θα δημιουργηθεί πρόβλημα. Παρατηρήθηκε ότι είναι δύσκολο έως και ακατόρθωτο, να γίνει κατανόηση του τρόπου που λειτουργούν τα ΤΝΔ, αν δεν γίνει πειραματισμός στο χαρτί και ανάπτυξη προγράμματος που να τα υλοποιεί. Επιπροσθέτως, παρατηρήθηκε ότι η μεταφορά ενός αλγορίθμου σε πρόγραμμα είναι επίσης ένας πολύ καλός τρόπος για την κατανόηση των σημείων που δεν είναι ξεκάθαρο το πώς λειτουργούν. Εν κατακλείδι παρατηρήθηκε ότι οποιοδήποτε σημείο είτε από τα ΤΝΔ είτε από την μέθοδο PINNs παραμένει θολό, θα έρθει η στιγμή που θα εμφανιστεί και θα χρειαστεί άμεση επίλυση προκαλώντας σύγχυση. Συνεπώς είναι προτιμότερο οποιοδήποτε κενό να καλύπτεται την στιγμή που αναγνωρίζεται.

Κλείνοντας, θα μπορούσαν να παρατεθούν ορισμένες προτάσεις ή μελλοντικοί στόχοι όσον αφορά την εφαρμογή της μεθόδου. Ως ένας μελλοντικός στόχος έχει τεθεί η εφαρμογή της μεθόδου σε επιπλέον προβλήματα μηχανικής όπως σε εξισώσεις της Δυναμικής. Μπορεί να γίνει το επόμενο βήμα ώστε να εφαρμοστεί η μέθοδος σε Μερικές Διαφορικές Εξισώσεις που αφορούν την Στατική Μηχανική, την Μετάδοση Θερμότητας ή και άλλα πεδία. Παράλληλα, σύμφωνα με τις δημοσιευμένες εργασίες των τελευταίων ετών, το ενδιαφέρον πάνω στο θέμα κινείται προς την κατεύθυνση της εφαρμογής της μεθόδου για την επίλυση αντιστρόφων προβλημάτων ταυτοποίησης παραμέτρων στη μηχανική. Αρκετές φυσικές διεργασίες δεν έχουν μοντελοποιηθεί ικανοποιητικά μέχρι στιγμής, δηλαδή δεν υπάρχει μια σαφής διαφορική εξίσωση η οποία να τις περιγράφει. Σε αυτήν την περίπτωση, είναι γνωστά τα ζευγάρια εισόδου εξόδου και αναζητούνται οι παράμετροι που απαρτίζουν την διαφορική εξίσωση. Αισίως, η χρήση της μεθόδου για την ταυτόχρονη επίλυση ευθέων και αντίστροφων προβλημάτων φαίνεται να δίνει καλύτερα αποτελέσματα από τη χρήση κλασικών μεθόδων αντιστροφής.

Βιβλιογραφία

Baydin, A., Pearlmutter , B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: a Survey. *The Journal of Machine Learning Research*(18), σσ. 1-43. Ανάκτηση από https://arxiv.org/pdf/1502.05767.pdf

Beer, F., Johnston, E., DeWolf, J., & Mazurek, D. (2016). *Μηχανική των Υλικών.* Τζιόλα.

Butcher, J. (1999). Numerical methods for ordinary differential equations in the 20th century. (J. o. Mathematics, Επιμ.) *Jurnal of Computational and Applied Mathematics*(125), σ. 29. Ανάκτηση από https://reader.elsevier.com/reader/sd/pii/S0377042700004556?token=A71716407A8A44A8AA52BCD3BD5ABB1F3494ED19CD6E3720B968DE64349EA6630B4715D7FE0E10FAAB93E4DB40ABBD9D&originRegion=eu-west-1&originCreation=20210915173754

David Alan Grier - The George Washington University. (2001). *Wayback Machine*. Ανάκτηση από https://web.archive.org/web/20160308075109/http://www.philsoc.org/2001Spring/2132transcript.html

Dreyfus, S. E. (1990). Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure. *Journal of Guidance, Control, and Dynamics*.

Edwards, S. B., & Harris, D. (2017). *Hidden Human Computers: The Black Women of NASA.* Ανάκτηση από https://books.google.gr/books?hl=el&lr=&id=g4CqDQAAQBAJ&oi=fnd&pg=PP1&dq=human+computers+nasa&ots=9BQffcn4MO&sig=oSy8rDB6-Gt-6QwF8GAXR8kEDeM&redir_esc=y#v=onepage&q&f=false

Gurney, K. (1997). *An introduction to neural networks.* London and New York, University of Sheffield.

Haghighat, E., Raissi, M., Moure, A., Gomez, H., & Juanes, R. (2020). A deep learning framework for solution and discovery in solid mechanics. *arXiv preprint*. Ανάκτηση από https://arxiv.org/pdf/2003.02751.pdf

Hebb, D. O. (2002). *The organization of behavior.* New York: Psychology Press. Ανάκτηση από https://www.taylorfrancis.com/books/mono/10.4324/9781410612403/organization-behavior-hebb

Hornic, K., Stinchcombe, M., & White , H. (1989). Multilayer Feedforward Networks are universal Approximators. *Neural Networks, 2*(5). Ανάκτηση από https://www.cs.cmu.edu/~epxing/Class/10715/reading/Kornick_et_al.pdf

Jiang, L., Byrd, R., Eskow, E., & Schnabel, R. (2004, November 21). A Preconditioned L-BFGS Algorithm With Application to Molecular Energy Minimization. (U. o. Colorado, Επιμ.) σσ. 1-17. Ανάκτηση από https://apps.dtic.mil/sti/pdfs/ADA444850.pdf

Karniadakis, G. E., Kevrekidis, Y., Lu, L., & Perdikaris, P. (2021, May 24). Physics-informed machine learning. *Nature Reviews Physics volume*(3).

Kingma, D. P., & Lei Ba, J. (2015). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. *ICLR*, σσ. 1-15. Ανάκτηση από https://arxiv.org/pdf/1412.6980.pdf

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1997). Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Transactions on Neural Networks, 9*(5), σσ. 1-26. Ανάκτηση από https://arxiv.org/pdf/physics/9705023.pdf

Lee, H., & Kang, I. (1990). Neural algorithms for solving differential equations. *Journal of Computational Physics*(91).

Leibniz, G. W. (1684). *MATHEMATICAL ASSOCIATION OF AMERICA*. Ανάκτηση από https://www.maa.org/press/periodicals/convergence/mathematical-treasure-leibnizs-papers-on-calculus-differential-calculus

McCulloch, W. S., & Pitts, W. (1943). A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. *BULLETIN OF MATHEMATICAL BIOPHYSICS*(5). Ανάκτηση από https://homeweb.csulb.edu/~cwallis/382/readings/482/mccolloch.logical.calculus.ideas.1943.pdf

Meade Jr, A. J., & Fernadez, A. A. (1994). The numerical solution of Linear Ordinary Differential Equations by Feedforward Neural Networks. *Math. Comput. Modelling, 19*(12). Ανάκτηση από https://reader.elsevier.com/reader/sd/pii/0895717794900957?token=855B323B8C5B58F051E6A3FFCA6DD90B54878BCB6176039E6B50C0E1D77C497CC5D270E5904488174E859CA5F839E5C0&originRegion=eu-west-1&originCreation=20211010143603

Mumford, C. L., & Jain, L. C. (2009). *Synergy in Computational Intelligence.* Berlin. Ανάκτηση από https://link.springer.com/chapter/10.1007/978-3-642-01799-5_1

Muradova, A. D., & Stavroulakis, G. E. (2021, June). Physics-Informed Neural Networks for elastic plate problems'.

Newton, I. (1736). *Google*. Ανάκτηση από https://books.google.gr/books?id=WyQOAAAAQAAJ&printsec=frontcover&hl=el&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Peng, W., Zhang, J., Zhou, W., Zhao, X., Yao, W., & Chen, X. (2021). IDRLnet: A Physics-Informed Neural Network Library. Ανάκτηση από https://arxiv.org/pdf/2107.04320.pdf

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017, November). Physics Informed Deep Learning (Part I) : Data - driven Solutions of Nonlinear Partial Differential Equations. *arXiv*. Ανάκτηση από https://arxiv.org/pdf/1711.10561.pdf

Rojas, R. (1996). *Neural Networks A Systematic Introduction.*

Ruder, S. (2017, June 15). An overview of gradient descent optimization algorithms. σσ. 1-14. Ανάκτηση από https://arxiv.org/abs/1609.04747

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). LEARNING INTERNAL REPRESENTATIONS By ERROR PROPAGATION. Ανάκτηση από https://apps.dtic.mil/sti/pdfs/ADA164453.pdf

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Leanring representations by back - propagating errors. *Nature*(323). Ανάκτηση από http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986, October 9). Learning representations by back - propagating errors. *Nature*. Ανάκτηση από http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf

Russel, S., & Norvig, P. (2003). *Τεχνητή Νοημοσύνη - Μια σύγχρονη προσέγγιση.* ΚΛΕΙΔΑΡΙΘΜΟΣ.

Searle, J. R. (1992). *The Rediscovery of the Mind.* Cambridge, Massachusetts: MIT Press.

Stavroulakis, G. E., Avdelas, A., Abdalla, K. M., & Panagiotopoulos, P. D. (1997). A neural network approach to the modelling, calculation and identification of semi-rigid connections in steel structures. *Journal of Constructional Steel Research, 44*(1-2), σσ. 91-105. Ανάκτηση από https://reader.elsevier.com/reader/sd/pii/S0143974X97000394?token=3D1A9E3DE0886DF5BC1A2FE72ECEF3E9713A111EFD05B3CF18C480DF7D8BDFC3A27980FC79CE9E28909C5B721589800A&originRegion=eu-west-1&originCreation=20211010143934

Turing, A. (1950). Ανάκτηση από https://www.csee.umbc.edu/courses/471/papers/turing.pdf