NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Developing a Differentiable Programming System

Evangelos Stefanos D. Klitsas

Supervisors:
Panagiotis Rondogiannis, Professor
Konstantinos Chatzikokolakis, Associate Professor

ATHENS

JUNE 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη ενός Συστήματος Διαφορίσιμου Προγραμματισμού

Ευάγγελος Στέφανος Δ. Κλήτσας

Επιβλέποντες:
Παναγιώτης Ροντογιάννης, Καθηγητής
Κωνσταντίνος Χατζηκοκολάκης, Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2021

BSc THESIS

Developing a Differentiable Programming System

Evangelos Stefanos D. Klitsas
S.N.: 1115201500070

SUPERVISORS:
Panagiotis Rondogiannis, Professor
Konstantinos Chatzikokolakis, Associate Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη ενός Συστήματος Διαφορίσιμου Προγραμματισμού

Ευάγγελος Στέφανος Δ. Κλήτσας
Α.Μ.: 1115201500070

ΕΠΙΒΛΕΠΟΝΤΕΣ:
Παναγιώτης Ροντογιάννης, Καθηγητής
Κωνσταντίνος Χατζηκοκολάκης, Αναπληρωτής Καθηγητής

ABSTRACT

This thesis examines Differential Programming, a programming paradigm. This combines differentiation -as it is known from Calculus- with programming and coding. It is a relatively novel concept, which has been established within the last decade. Thus, our goal is puzzling out what Differentiable Programming is all about. We will mention furthermore the fundamental concepts that are associated to Differentiable Programming and focus on the algorithm of Backpropagation. Then we analyze Autograd thoroughly, a PyTorch's tool for differentiation, breaking down how it finds derivatives of a multi-variable function using Backpropagation. Finally,t we will describe how we developed our own differentiating tool equivalent to Autograd and make a solid comparison between both systems.

SUBJECT AREA: Differentiable Programming

KEYWORDS: Differentiable Programming, Autograd, Automatic Differentiation, Backpropagation, Reverse Accumulation

ΠΕΡΙΛΗΨΗ

Σε αυτή την πτυχιακή εργασία μελετάμε τον τομέα του "Διαφορίσιμου Προγραμματισμού" (Differentiable Programming). Είναι ένα είδος προγραμματισμού που ενσωματώνει την έννοια της κλασσικής παραγώγισης, όπως είναι γνωστή από τον Μαθηματικό Λογισμό, μέσα στον κώδικα. Εφόσον είναι ένας σχετικά νέος όρος που ξεκίνησε στα μέσα της προηγούμενης δεκαετίας, στόχος μας αρχικά είναι να οριοθετήσουμε με σαφήνεια το αντικείμενο που αυτός πραγματεύεται με βάση την συζήτηση στην σύγχρονη σχετική βιβλιογραφία. Στη συνέχεια αναφέρουμε τις βασικές έννοιες που σχετίζονται με το Differentiable Programming ενώ θα μιλήσουμε επίσης για ορισμένες τεχνικές παραγώγισης, εμβαθύνοντας λίγο περισσότερο στον αλγόριθμο του Backpropagation. Έπειτα παρουσιάζουμε αναλυτικά το Autograd, το οποίο είναι το εργαλείο που χρησιμοποιεί η PyTorch για την εύρεση παραγώγων. Αναλύουμε πώς αυτό, βασιζόμενο στο Backpropagation, επιτυγχάνει την παραγώγιση μιας συνάρτησης πολλών μεταβλητών. Τέλος, θα περιγράψουμε την υλοποίηση ενός δικού μας συστήματος αντίστοιχο με το Autograd. Θα επικεντρωθούμε στον τρόπο λειτουργίας του και θα κάνουμε μια σύγκριση μεταξύ των δύο αυτών εργαλείων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Differentiable Programming

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Differentiable Programming, Autograd, Automatic Differentiation, Backpropagation, Reverse Accumulation

ΠΕΡΙΕΧΟΜΕΝΑ

1 ΕΙΣΑΓΩΓΗ **10**

2 Differentiable Programming **11**

2.1 Εννοιολογικός Ορισμός 11

2.2 Automatic Differentiation 11

2.3 Forward ή Reverse Accumulation AD 12

2.4 Συσχέτιση με Deep Learning 13

3 Προαπαιτούμενες γνώσεις **14**

3.1 Κανόνας της αλυσίδας 14

3.2 Backpropagation 14

4 Autograd **16**

4.1 Παράδειγμα κώδικα 16

4.2 Πώς λειτουργεί 17

4.3 Δομές Δεδομένων, Επαναλήψεις, If-else 17

4.4 Περιορισμοί 18

5 Υλοποίηση Autograd **19**

5.1 Εισαγωγή 19

5.2 Υλοποίηση 19

5.2.1 Αναπαράσταση του Γράφου 19

5.2.2 Παραγώγιση 21

5.2.3 Παράδειγμα εκτέλεσης κώδικα 22

5.3 Περιορισμοί 25

5.4 Μελλοντικές Βελτιώσεις 26

6 Επίλογος **28**

7 Πίνακας Ορολογίας **29**

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

3.1 Εικόνα 1: Παράδειγμα Backpropagation 15

5.1 Εικόνα 2: Αναπαράσταση κώδικα με γράφο #1 23

5.2 Εικόνα 3: Αναπαράσταση κώδικα με γράφο #2 25

ΠΡΟΛΟΓΟΣ

Η ιδέα της ενασχόλησης με το Differentiable Programming ξεκίνησε από τους καθηγητές Παναγιώτη Ροντογιάννη και Κωνσταντίνο Χατζηκοκολάκη. Υπήρξε η κοινή διάθεση να ασχοληθούμε με αυτόν τον τομέα προγραμματισμού, οπότε, υπό την καθοδήγησή τους, ανέλαβα να κάνω μια πρώτη έρευνα για το θέμα. Ο σκοπός αρχικά ήταν να κατανοήσω το αντικείμενο και έπειτα να τους το παρουσιάσω μέσω μιας σειράς διαδικτυακών συναντήσεων, με στόχο να μελετήσουμε πιο αναλυτικά το κομμάτι του Differentiable Programming. Θα ήθελα, λοιπόν, να τους ευχαριστήσω για τον χρόνο και την υπομονή τους, μέσα σε μια περίεργη και δύσκολη χρονιά για όλους.

1. ΕΙΣΑΓΩΓΗ

Σε αυτή την πτυχιακή εργασία ασχοληθήκαμε με την έννοια του Differentiable Programming. Οι πρώτες αναφορές στον Διαφορίσιμο Προγραμματισμό συναντώνται από το 2014, όπου φαίνεται η πεποίθηση ότι η "νευρωνική μάθηση" (Deep Learning), βασίζεται σε κάτι άλλο, δηλαδή υπάρχει ένα δομικό στοιχείο αυτού. Παρόλο που το deep learning δεν είναι κάτι καινούργιο και συναντάται από την δεκαετία του ’70, πέρασαν περίπου σαράντα χρόνια μέχρι να γίνει μια αναφορά σε κάτι πιο θεμελιώδες. Αυτό παρουσιάζεται αρχικά σαν ένας συνδυασμός συναρτησιακού προγραμματισμού (functional programming) και βελτιστοποίησης μέσω παραγώγων, με τον όρο "Διαφορίσιμος Συναρτησιακός Προγραμματισμός" (Differentiable Functional Programming) *[1]*. Η ιδέα αυτή, αν και σε πρώιμο στάδιο, είναι πολύ κοντά στη λογική του Differentiable Programming όπως τον ξέρουμε σήμερα. Θεωρήσαμε αρκετά ενδιαφέρουσα την συσχέτιση με το deep learning, το οποίο ανήκει σε έναν από τους πιο δημοφιλείς κλάδους της πληροφορικής. Επίσης, το Differentiable Programming αποτελεί έναν αναπτυσσόμενο τομέα της πληροφορικής και μπορεί κανείς να συναντήσει πληθώρα ετερόκλητων απόψεων σχετικά με το αντικείμενο που πραγματεύεται. Για αυτούς τους λόγους, αποφασίσαμε σε αυτή την πτυχιακή εργασία να ασχοληθούμε με την έννοια του Differentiable Programming, ελπίζοντας να συμβάλουμε έστω και λίγο στην ανάπτυξή του.

Σύμφωνα με την πεποίθηση που επικρατεί στις μέρες μας, το Differentiable Programming είναι κάτι πιο γενικό από το Deep Learning. Βασίζεται στην "αυτόματη παραγώγιση" (Automatic Differentiation), δηλαδή τον υπολογισμό παραγώγων μέσω υπολογιστικών τεχνικών ή αλγορίθμων. Στη συνέχεια, χρησιμοποιεί τις παραγώγους αυτές, για να μπορέσει να βελτιώσει τα αποτελέσματα του προγράμματος. Φαίνεται λοιπόν η άμεση συσχέτιση με το Deep Learning. Προτού όμως αναλύσουμε περαιτέρω το Automatic Differentiation, θα δώσουμε έναν ακριβή εννοιολογικό ορισμό του Διαφορίσιμου Προγραμματισμού, έτσι ώστε να γίνει απόλυτα κατανοητός και να προχωρήσουμε στη συνέχεια σε πιο εξειδικευμένα στοιχεία, όπως το Autograd και το εργαλείο που δημιουργήσαμε εμείς.

2. DIFFERENTIABLE PROGRAMMING

2.1 Εννοιολογικός Ορισμός

Το Differentiable Programming είναι ένα είδος προγραμματισμού, στο οποίο εντάσσονται τα προγράμματα που δέχονται κάποιες αριθμητικές παραμέτρους και ανάλογα με αυτές στοχεύουν στην βελτιστοποίηση των αποτελεσμάτων τους, χρησιμοποιώντας παραγώγους. Αρχικά, υπολογίζει τις παραγώγους πρώτης τάξης των συναρτήσεων και στη συνέχεια πηγαίνει προς την κατεύθυνση που η παράγωγος ελαττώνεται/αυξάνεται μέχρι αυτή να μηδενιστεί. Το σημείο αυτό είναι τοπικό ελάχιστο/μέγιστο και ο αλγόριθμος ονομάζεται σύγκλιση με ελάττωση της παραγώγου ("gradient descent"). Φυσικά, το παραπάνω ισχύει μόνο για παραγωγίσιμες συναρτήσεις, όπως ορίζει ο Μαθηματικός Λογισμός. Έτσι, βρίσκει για ποιες τιμές της εισόδου ελαχιστοποιείται/μεγιστοποιείται η έξοδος και είναι σε θέση να τη βελτιστοποιήσει ανάλογα με το τι ζητάει ο χρήστης. Πολλές φορές το Differentiable Programming μπορεί να συναντηθεί και ως ∂P. Στη συγκεκριμένη γραφή, το σύμβολο της μερικής παραγώγου (∂) αναπαριστά το Differentiable και το *P* το Programming από το αρχικό γράμμα του.

Ένας ενδιαφέρων μαθηματικός τύπος που συναντήσαμε και κατά τη γνώμη μας αποτυπώνει ολόκληρη την φιλοσοφία του Differentiable Programming είναι ο εξής: $\frac{\partial Program}{\partial Param}$ *[2]*. Ακριβώς δηλαδή όπως αναλύθηκε παραπάνω, το Differentiable Programming είναι μια ομάδα προγραμμάτων που αξιολογούν πόσο συνεισφέρει η κάθε είσοδος στο πρόγραμμα και ανάλογα τροποποιείται δυναμικά για καλύτερη βελτιστοποίηση (optimization) της εξόδου του. Η επιστημονική κοινότητα φαίνεται να συμφωνεί στον παρακάτω πιο επίσημο ορισμό, που συμπεριλαμβάνει όλα τα προηγούμενα, καθώς παρουσιάζεται με παρεμφερείς μορφές σε αρκετές πηγές.

Το Differentiable Programming (∂P) είναι ένα πρότυπο προγραμματισμού, κατά το οποίο ένα πρόγραμμα παραγωγίζεται μέσω "αυτόματης παραγώγισης" (automatic differentiation) ως προς την είσοδό του και προσαρμόζεται σε αυτή, ούτως ώστε να επιτευχθεί η βελτιστοποίησή του, σύμφωνα με τις αριθμητικές τιμές των παραμέτρων του. Η μέθοδος που χρησιμοποιείται για αυτόν τον σκοπό είναι το gradient descent *[3]-[7]*.

2.2 Automatic Differentiation

Η "Αυτόματη Παραγώγιση" (Automatic Differentiation ή AD) είναι η πηγή του Differentiable Programming. Εδώ ανήκουν όλες οι τεχνικές και αλγόριθμοι παραγώγισης, οι οποίοι επιτυγχάνουν την παραγώγιση κάποιας συνάρτησης. Ο απλούστερος, αλλά και λιγότερο αποδοτικός τρόπος είναι η συμβολική παραγώγιση ("symbolic differentiation"). Χρησιμοποιεί την εφαρμογή όλων των κανόνων παραγώγισης και παράγει την αντίστοιχη έκφραση της παραγώγου ενώ στη συνέχεια υπολογίζει την τιμή αυτής για δοθείσες τιμές των μεταβλητών *[8],[9]*. Είναι η μοναδική τεχνική που παράγει ολόκληρη την έκφραση της παραγώγου, οπότε, εάν αυτό είναι το ζητούμενο, η χρήση του symbolic differentiation είναι μονόδρομος. Επίσης, δεν μπορεί να χειριστεί επαναλήψεις, δομές if-else και γενικά στοιχεία αμιγώς προγραμματιστικά, τα οποία δεν έχουν σχέση με τα μαθηματικά *[10]*. Βέβαια, τις περισσότερες φορές αυτό που ενδιαφέρει τον χρήστη δεν είναι η έκφραση της παραγώγου αλλά η αριθμητική τιμή της σε δεδομένο σημείο. Ως εκ τούτου, υπάρχουν πιο αποδοτικές μέθοδοι για αυτό: το *Forward Accumulation* AD και το

Reverse Accumulation AD.

Στο Forward Accumulation η παραγώγιση πραγματοποιείται με φορά από την είσοδο προς την έξοδο, δηλαδή βρίσκονται οι παράγωγοι ξεκινώντας από τις μεταβλητές εισόδου και καταλήγοντας στις μεταβλητές εξόδου, ανάλογα με τις συναρτήσεις που εφαρμόζονται κατά την εκτέλεση του κώδικα. Ως αποτέλεσμα, οι παράγωγοι υπολογίζονται παράλληλα με την εκτέλεση του κώδικα *[9]-[11]*.

Μία γνωστή μέθοδος που ανήκει στο *Forward Accumulation* είναι το Dual Numbers *[11],[12]*. Η κεντρική ιδέα εδώ μοιάζει σε μεγάλο βαθμό με τους μιγαδικούς αριθμούς. Όπως ένας αριθμός $a \in \mathbb{C}$ γράφεται σαν $a = x + yi$, με $x, y \in \mathbb{R}$ έτσι και στα Dual Numbers, ένας αριθμός Dual Number d έχει τη μορφή $d = x + y\varepsilon$, με $x, y \in \mathbb{R}$. Σε αυτή την περίπτωση το ε είναι μια ποσότητα αντίστοιχης λογικής με το i, τέτοια ώστε $\varepsilon \neq 0$ αλλά $\varepsilon^2 = 0$. Το y στον Dual Number d είναι το αποτέλεσμα της παραγώγου σε δεδομένο σημείο. Σημειώνεται ότι στην μεταβλητή ως προς την οποία ψάχνουμε την παράγωγο, το y αρχικοποιείται με 1 όπως ακριβώς η παράγωγος π.χ. $\frac{\partial a}{\partial a} = 1$. Με αυτόν τον τρόπο υπολογίζεται ταυτόγχρονα με την ροή εκτέλεσης του κώδικα η αντίστοιχη τιμή της παραγώγου. Για παράδειγμα, έστω ότι έχουμε στα αριστερά ένα μικρό κομμάτι κώδικα Python και στα δεξιά το ισοδύναμο με τη μέθοδο Dual Numbers:

1. $x = 2$	|	$x = 2 + 1\varepsilon$
2. $a = x * *2$	|	$a = (2 + 1\varepsilon)^2 = 4 + 4\varepsilon + \varepsilon^2 = 4 + 4\varepsilon$
3. $b = a + 3 * x$	|	$b = 4 + 4\varepsilon + 3 \cdot (2 + 1\varepsilon) = 10 + 7\varepsilon$
4. $c = b + 5$	|	$c = 10 + 7\varepsilon + 5 = 15 + 7\varepsilon$

Το παραπάνω κομμάτι κώδικα περιγράφει απλώς τη συνάρτηση $f(x) = x^2 + 3x + 5$, με $f'(x) = \frac{\partial f}{\partial x} = 2x + 3$. Στον κώδικά μας παρατηρούμε ότι η μεταβλητή x έχει αρχικοποιηθεί με την τιμή 2, οπότε $f'(2) = 7$. Πράγματι, κοιτώντας στα Dual Numbers τον συντελεστή του ε, βλέπουμε ότι είναι όντως 7, αναπαριστώντας την τιμή της παραγώγου της μεταβλητής c για $x = 2$.

Το *Reverse Accumulation* είναι η δεύτερη κατηγορία Automatic Differentiation *[7], [9]-[11], [13]*. Εδώ, η παραγώγιση γίνεται ακριβώς με την αντίθετη κατεύθυνση σε σχέση με το *Forward Accumulation*. Πρώτα εκτελείται μια φορά ο κώδικας και στην συνέχεια, αποθηκεύει σε μια κατάλληλη δομή δεδομένων την σειρά των συναρτήσεων, καθώς και πού αυτές έχουν εφαρμοστεί. Με αυτόν τον τρόπο, η παραγώγιση γίνεται ξεκινώντας από τις μεταβλητές εξόδου και ακολουθώντας το αποθηκευμένο ιστορικό των εφαρμοσμένων συναρτήσεων, καταλήγει στις μεταβλητές εισόδου. Η πιο διαδεδομένη μέθοδος *Reverse Accumulation* είναι το Backpropagation, στο οποίο θα αναφερθούμε εκτενώς στο 3.2

2.3 Forward ή Reverse Accumulation AD

Ένα βασικό ερώτημα είναι πότε θα ήταν ορθότερο να χρησιμοποιήσουμε μια τεχνική *Forward Accumulation* και πότε μια *Reverse Accumulation*. Οι δύο αυτές κατηγορίες παραγώγισης διαφοροποιούνται σε μεγάλο βαθμό στην πολυπλοκότητα, ανάλογα με την διάσταση της εισόδου και της εξόδου. Έστω ότι έχουμε ένα πρόγραμμα/συνάρτηση f που παίρνει είσοδο διάστασης n και επιστρέφει έξοδο διάστασης m, δηλαδή $f : \mathbb{R}^n \to \mathbb{R}^m$. Αν $n \ll m$, συμφέρει από άποψη πολυπλοκότητας να χρησιμοποιηθεί μία

μέθοδος Forward Accumulation, ενώ αν $m \ll n$, η βέλτιστη επιλογή είναι η μέθοδος Reverse Accumulation *[9], [11]*.

2.4 Συσχέτιση με Deep Learning

Σύμφωνα με τον παραπάνω ορισμό του ∂P, είναι εύκολο να καταλάβει κανείς γιατί το Differentiable Programming συγχέεται με το Deep Learning. Κατά τον βραβευμένο με Turing Award, Yann LeCun, "το Differentiable Programming είναι ένας επαναπροσδιορισμός του Deep Learning" *[14]*. Μαζί με τον Léon Bottou διατύπωσαν την ιδέα ότι "κάθε μοντέρνο σύστημα deep learning μπορεί να δημιουργηθεί μέσω ενός δικτύου, στο οποίο μπορεί να εφαρμοστεί το backpropagation (3.2), μία τεχνική automatic differentiation." *[15]*. Έτσι, γίνεται σαφές ότι το Differentiable Programming δεν είναι ταυτόσημη έννοια με το Deep Learning, αλλά το δεύτερο αποτελεί υποσύνολο του πρώτου, καθώς χρησιμοποιεί τις παραγώγους και την μέθοδο gradient descent, όπως περιγράφηκε στην αρχή, για να ελαχιστοποιήσει τη "συνάρτηση απόκλισης" (loss function). Αυτή δεν είναι παρά μια συνάρτηση που ταυτίζεται με την απόκλιση της απόφασης του νευρωνικού από τη βέλτιστη επιλογή. Με άλλα λόγια, αξιολογεί την επιλογή του νευρωνικού δικτύου, καταλαβαίνοντας εάν πλησιάζει προς την βέλτιστη λύση ή απομακρύνεται από αυτή, ανάλογα με το αποτέλεσμα της loss function. Άρα, όλη η ουσία είναι η εύρεση του ολικού ελαχίστου της συγκεκριμένης συνάρτησης, ώστε να ελέγχεται σε κάθε επανάληψη αν πλησιάζει προς αυτό το σημείο ή απομακρύνεται. Η βάση όλων των παραπάνω βρίσκεται ξανά στον υπολογισμό των παραγώγων, ώστε να βελτιστοποιηθεί το νευρωνικό ελαχιστοποιώντας την loss function, κάτι που είδαμε από την αρχή ότι είναι ο πυρήνας του Differentiable Programming *[16]*.

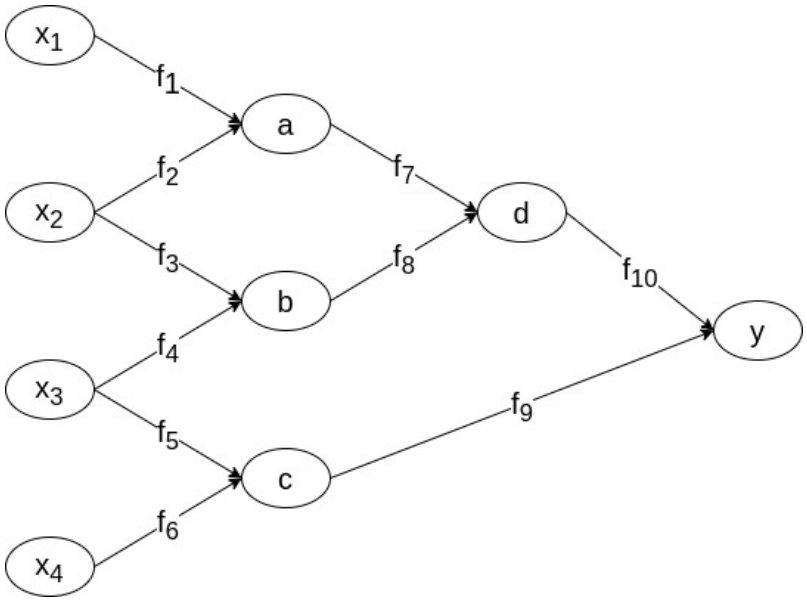
3. ΠΡΟΑΠΑΙΤΟΥΜΕΝΕΣ ΓΝΩΣΕΙΣ

3.1 Κανόνας της αλυσίδας

Προκειμένου να γίνει κατανοητό στον αναγνώστη πώς λειτουργεί το Autograd που θα αναλύσουμε στην συνέχεια, πρέπει να αναφερθούν αρχικά, κάποιες βασικές γνώσεις μαθηματικής ανάλυσης. Πιο συγκεκριμένα, θα εστιάσουμε στον κανόνα της αλυσίδας, ο οποίος ορίζει: $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial z} \cdot \frac{\partial z}{\partial y}$. Παρατηρούμε δηλαδή, ότι αν μία μεταβλητή x εξαρτάται από μια μεταβλητή y, η οποία με τη σειρά της εξαρτάται από μία άλλη μεταβλητή z, τότε η μερική παράγωγος της x ως προς y ισούται με το γινόμενο της μερικής παραγώγου της x ως προς z επί την μερική παράγωγο της z ως προς y.

3.2 Backpropagation

Μια έννοια που έχει ήδη αναφερθεί και θα χρησιμοποιηθεί πολύ στην συνέχεια είναι αυτή του Backpropagation. Είναι μια μέθοδος Reverse Accumulation, η οποία χρησιμοποιεί τον κανόνα της αλυσίδας για την εύρεση των παραγώγων. Το πρώτο χαρακτηριστικό του αλγορίθμου Backpropagation, που πρέπει να επισημανθεί, είναι το γεγονός ότι δεν υπολογίζει κάποιον τύπο της παραγώγου αλλά το αποτέλεσμα αυτής για δεδομένες τιμές των μεταβλητών εισόδου. Αρχικά, γίνεται μια πρώτη εκτέλεση του κώδικα, κατά την οποία αποθηκεύονται σε μια δομή δεδομένων οι συναρτήσεις που εφαρμόζονται σε κάθε μεταβλητή εισόδου. Στη συνέχεια, διατρέχοντας αυτή τη δομή, βλέπει την σειρά με την οποία έχουν εφαρμοστεί οι συναρτήσεις στις διάφορες μεταβλητές. Κατόπιν, βρίσκει το μονοπάτι από την μεταβλητή που παραγωγίζουμε, προς αυτήν ως προς την οποία γίνεται η παραγώγιση. Κάθε ακμή αναπαριστάται με την μερική παράγωγο της μιας μεταβλητής ως προς την άλλη, ανάλογα την κατεύθυνσή της. Έπειτα, ξεκινώντας ανάποδα από το τέλος του μονοπατιού, υπολογίζει το γινόμενο των μερικών παραγώγων που προκύπτουν εξαιτίας των αντίστοιχων ακμών από το ανάποδο μονοπάτι (αντιστρέφοντας αρχή με τέλος, καθώς και όλες τις εσωτερικές ακμές). Εάν το μονοπάτι δεν είναι μοναδικό, προσθέτουμε το γινόμενο των μερικών παραγώγων από όλα τα διαφορετικά μονοπάτια που ενώνουν την μεταβλητή που θέλουμε να παραγωγίσουμε, ως προς την οποία γίνεται η παραγώγιση *[7], [17], [18]*. Ας περιγράψουμε ένα απλό παράδειγμα, βασισμένο στην *Εικόνα 1* που ακολουθεί.



Εικόνα 1: Παράδειγμα Backpropagation

Έστω ότι x_i, με $i = 1, ..., 4$ είναι οι μεταβλητές εισόδου ενός προγράμματος, y η έξοδος και f_i, με $i = 1, ..., 10$ οι συναρτήσεις που εφαρμόζονται στις διάφορες μεταβλητές. Σκοπός του αλγόριθμου είναι να υπολογίσει την αλλαγή που επιφέρει στην έξοδο y μία αλλαγή στην τιμή οποιασδήποτε από τις μεταβλητές εισόδου, έστω της x_3. Αυτό δεν είναι κάτι διαφορετικό από το $\frac{\partial x_3}{\partial y}$. Ακολουθώντας την παραπάνω διαδικασία του Backpropagation πρώτα εντοπίζει όλα τα μονοπάτια $y \to x_3$, το $y \to d \to b \to x_3$ και το $y \to c \to x_3$. Στη συνέχεια, υπολογίζει τα $\frac{\partial y}{\partial x_3} = \frac{\partial y}{\partial d} \cdot \frac{\partial d}{\partial b} \cdot \frac{\partial b}{\partial x_3}$ και $\frac{\partial y}{\partial x_3} = \frac{\partial y}{\partial c} \cdot \frac{\partial c}{\partial x_3}$. Οπότε, αθροίζοντας τα δύο αυτά γινόμενα, υπολογίζεται η ζητούμενη παράγωγος: $\frac{\partial y}{\partial x_3} = \frac{\partial y}{\partial d} \cdot \frac{\partial d}{\partial b} \cdot \frac{\partial b}{\partial x_3} + \frac{\partial y}{\partial c} \cdot \frac{\partial c}{\partial x_3}$

4. AUTOGRAD

4.1 Παράδειγμα κώδικα

Το Autograd *[18]-[23]* είναι ένα εργαλείο της PyTorch για τον υπολογισμό παραγώγων και βασίζεται στον αλγόριθμο του Backpropagation. Ας δούμε όμως πρώτα ένα απλό παράδειγμα χρήσης του Autograd σε Python και πώς αυτό υπολογίζει τις παραγώγους πρώτης και δεύτερης τάξης ως προς x της συνάρτησης $f(x, y) = x^3 + x^2 \cdot y^3$, για τις συγκεκριμένες τιμές των x, y:

```
1.  from autograd import grad
2.
3.  def fun (x, y):
4.      return x***3 + x***2*y***3 + 1
5.
6.  fun_grad1 = grad (fun, 0)
7.  fun_grad2 = grad (fun_grad1, 0)
8.
9.  print (fun_1st_grad (3.0, 2.0))
10. print (fun_2nd_grad (5.0, 1.0))
```

Όπως αναφέραμε, στο παραπάνω παράδειγμα υπολογίζεται το $\frac{\partial f(x,y)}{\partial x} = 3 \cdot x^2 + 2 \cdot x \cdot y^3$ και $\frac{\partial f(x,y)}{\partial x^2} = 6 \cdot x + 2 \cdot y^3$ για $x = 3$, $y = 2$ και $x = 5$, $y = 1$ αντίστοιχα. Αρχικά δηλώνεται η συνάρτηση fun που αναπαριστά την $f(x, y)$. Στη συνέχεια, καλείται η συνάρτηση grad. Αυτή αποτελεί μέρος του package Autograd και κάνει όλη την παραγώγιση. Αν και μπορεί να κληθεί με αρκετά ορίσματα, ανάλογα του τι θέλει κάποιος να υπολογίσει, εμείς θα επικεντρωθούμε στην κλήση του παραδείγματος. Εδώ βλέπουμε ότι πρώτο όρισμα είναι το όνομα της συνάρτησης που θέλουμε να παραγωγίσουμε (fun εν προκειμένω). Το δεύτερο όρισμα είναι ένας ακέραιος που αναπαριστά το όρισμα της συνάρτησης που θέλουμε να παραγωγίσουμε. Αυτή θα είναι και η μεταβλητή ως προς την οποία θα γίνει η παραγώγιση. Το 0 είναι για το πρώτο όρισμα, το 1 για το δεύτερο κλπ. Το αποτέλεσμα της grad είναι η ζητούμενη παράγωγος της συνάρτησης. Καλώντας την grad διαδοχικά στη συνάρτηση fun και στην πρώτη της παράγωγο ως προς το x, την fun_1st_grad,

δημιουργούμε την παράγωγο δεύτερης τάξης της fun, την fun_2nd_grad. Εάν τρέξουμε αυτό το κομμάτι κώδικα, θα πάρουμε σαν αποτέλεσμα τους αριθμούς *75.0* και *32.0* που είναι και το ζητούμενο αποτέλεσμα

4.2 Πώς λειτουργεί

Αρχικά, κατά την εκτέλεση του κώδικα, δημιουργείται ένας κατευθυνόμενος άκυκλος γράφος, οι κορυφές του οποίου είναι μεταβλητές του προγράμματος και οι ακμές αναπαριστούν τις συναρτήσεις που εφαρμόζονται σε αυτές. Οι μεταβλητές εισόδου αναπαρίστανται στον γράφο ως κορυφές με εσωτερικό βαθμό ίσο με μηδέν, καθώς αρχικοποιούνται με κάποια συγκεκριμένη τιμή και δεν δημιουργούνται μέσω κάποιας πράξης ούτε εξαρτώνται από κάποια άλλη μεταβλητή/συνθήκη του προγράμματος. Αντίθετα, οι μεταβλητές εξόδου (η έξοδος του προγράμματος) είναι οι κορυφές του γράφου με εξωτερικό βαθμό ίσο με μηδέν, επειδή δεν εφαρμόζεται σε αυτές κάποια συνάρτηση. Προφανώς, αυτό συμβαίνει επειδή είναι το τελικό αποτέλεσμα του προγράμματος. Κάθε κορυφή περιέχει πληροφορίες σχετικά με το όνομά της, τα ονόματα των γονέων της, τη συνάρτηση με την οποία δημιουργήθηκε, καθώς και την αριθμητική τιμή της μεταβλητής. Ο γράφος αυτός δημιουργείται αυτόματα με την εκτέλεση του κώδικα.

Όταν ο χρήστης ολοκληρώσει τον κώδικά του, καλεί τη συνάρτηση grad με δύο ορίσματα. Το πρώτο είναι η μεταβλητή που θέλει να παραγωγίσει (έστω x) και το δεύτερο η μεταβλητή ως προς την οποία γίνεται η παραγώγιση (έστω y). Στη συνέχεια, μέσω κάποιου αλγόριθμου αναζήτησης, βρίσκονται όλα τα μονοπάτια $y \rightarrow x$ στο γράφο. Έπειτα, το Autograd χρησιμοποιεί τον αλγόριθμο Backpropagation, όπως ακριβώς περιγράφηκε στην ενότητα 3.2, με στόχο την εύρεση του $\frac{\partial x}{\partial y}$ που είναι και το ζητούμενο.

Εδώ πρέπει να σημειωθεί ότι το Autograd βασίζεται στο Backpropagation και έτσι δεν βρίσκει κάποιο τύπο συνάρτησης της παραγώγου. Ουσιαστικά, υπολογίζει την τιμή αυτής για δεδομένες τιμές στις μεταβλητές εισόδου. Αν το δούμε γραφικά, αυτό που συμβαίνει είναι ότι για ένα δεδομένο σημείο $P(x_1, ..., x_n)$ της γραφικής παράστασης της συνάρτησης f που δημιουργείται, υπολογίζεται η τιμή d της παραγώγου f' στο σημείο P, με $d = f'(x_1, ..., x_n)$.

4.3 Δομές Δεδομένων, Επαναλήψεις, If-else

Με μια πρώτη ματιά μπορεί να φαίνεται ότι ο αλγόριθμος Backpropagation ίσως μειώνει τις δυνατότητες του προγράμματος, καθώς δεν επιστρέφει κάποια συνάρτηση αλλά μόνο

ένα σημείο αυτής. Στην πραγματικότητα όμως προσθέτει επιπλέον λειτουργικότητα. Επειδή είναι αλγόριθμος που ανήκει στην κατηγορία Reverse Accumulation, πρώτα εκτελείται μια φορά ο κώδικας και μετά δημιουργείται αυτός ο γράφος. Οπότε, όλα τα for/while - loops και τα if - else υλοποιούνται κι έτσι εκ των υστέρων ακολουθεί τη ροή του κώδικα, βλέποντας ποιες συνθήκες είναι αληθείς, πόσες φορές έγινε ένα for/while κλπ. Επιτρέποντας, λοιπόν, την χρήση οποιασδήποτε μορφής επαναλήψεων και if - else, πέρα από αριθμητικές πράξεις και αναθέσεις τιμών, έχουμε ένα πλήρες πρόγραμμα προς παραγώγιση.

4.4 Περιορισμοί

Δυστυχώς, το Autograd υπόκειται σε ορισμένους περιορισμούς που έχουν να κάνουν με τον τρόπο γραφής του κώδικα. Αρχικά, καθώς είναι γραμμένο σε Python, οι βιβλιοθήκες που έχουν βάση την γλώσσα C (π.χ. math) δεν μπορούν να παραγωγιστούν αυτόματα. Για παράδειγμα, παρόλο που οι δύο εκφράσεις $math.sqrt(x)$ σε C και $x * *(1/2)$ σε Python ισούνται με $\sqrt{x}$ δεν είναι ισοδύναμες στον κώδικα, καθώς η πρώτη είναι C-based έκφραση, οπότε δεν παραγωγίζεται αυτόματα, ενώ η δεύτερη είναι έκφραση Python, οπότε είναι αποδεκτή. Ένας ακόμα περιορισμός που έχει παρατηρηθεί έχει να κάνει με την ανάθεση τιμής σε κάποιο κελί πίνακα. Δηλαδή, έστω ότι $A[n]$ είναι ένας μονοδιάστατος πίνακας αριθμητικών τιμών με n θέσεις. Η έκφραση $A[i] = x$ για $i = 0, ..., n-1$, όταν το x έχει αρχικοποιηθεί με κάποιον αριθμό, μπορεί να δημιουργήσει προβλήματα στον υπολογισμό των παραγώγων (ανεξαρτήτου διάστασης πίνακα). Τέλος, κάτι ακόμα που πρέπει να αποφεύγεται είναι τα in-place operations (π.χ. a+=b, x*=2 κλπ). Στην πράξη, αυτό που κάνει μια τέτοια εντολή είναι να σβήνει την τιμή που είχε η μεταβλητή και να την αντικαθιστά με τη νέα. Έτσι το Autograd μπορεί να επηρρεαστεί, καθώς απαιτεί την τιμή κάθε μεταβλητής τη δεδομένη στιγμή που βρίσκεται στο μονοπάτι κατά το Backpropagation και όχι μόνο την τελική.

5. ΥΛΟΠΟΙΗΣΗ AUTOGRAD

5.1 Εισαγωγή

Σε αυτή λοιπόν την πτυχιακή εργασία σκεφτήκαμε να αναπτύξουμε ένα σύστημα αντίστοιχο του Autograd. Καθώς αυτό δεν είναι πακέτο ελεύθερου κώδικα, δεν υπάρχει δημοσιευμένος ολόκληρος ο κώδικάς του. Ωστόσο υπάρχουν επαρκείς πληροφορίες για το πώς δουλεύει και τι χρησιμοποιεί. Βασιζόμενοι προφανώς στο πακέτο της PyTorch, αποφασίσαμε να δημιουργήσουμε την δική μας υλοποίηση του Autograd. Θα παρουσιάσουμε τις λεπτομέρειες υλοποίησης, κάποιες διαφοροποιήσεις σε σχέση με το πρωτότυπο και τι περιορισμούς έχει η δική μας υλοποίηση. Ολόκληρος ο κώδικας μαζί με μερικά πιο συγκεκριμένα παραδείγματα βρίσκεται στο Github, στον ακόλουθο σύνδεσμο: **GITHUB LINK**

5.2 Υλοποίηση

5.2.1 Αναπαράσταση του Γράφου

Ο κώδικας είναι χωρισμένος σε τρία αρχεία Python: `struct_and_math_functions.py`, `differentiation_rules.py` και `grad.py`. Η αρχή θα γίνει με το `struct_and_math_functions.py`, στο οποίο βρίσκεται η κλάση primitive, που αναπαριστά την κορυφή στον γράφο που δημιουργείται και αναλύθηκε στο 4.1. Περιέχει 3 πεδία:

- name: το όνομα της μεταβλητής
- value: η τιμή της μεταβλητής
- function: η συνάρτηση που εφαρμόστηκε και είχε ως αποτέλεσμα αυτή τη μεταβλητή
- parents: λίστα με τις μεταβλητές που χρησιμοποιήθηκαν στην προηγούμενη συνάρτηση, για να δημιουργηθεί η συγκεκριμένη μεταβλητή

Το name και το function είναι συμβολοσειρές, με τη συνάρτηση να κωδικοποιείται από τα πρώτα γράμματα του ονόματός της (δηλαδή add, sub, mult, div, exp, cos, sin κλπ). Το value είναι αριθμητική τιμή ενώ το parents μία λίστα με συμβολοσειρές.

Πρώτα από όλα, ο χρήστης αρχικοποιεί τις μεταβλητές που ορίζει σαν αρχικές μεταβλητές. Με τον όρο αρχικές μεταβλητές εννοούμε για παράδειγμα στη συνάρτηση $f(x, y, z) = x^2 + y \cdot t \cdot z + 3 \cdot t, t \in \mathbb{R}$ τις μεταβλητές x, y, z, από τις οι οποίες εξαρτάται η f. Η αρχικοποίησή τους γίνεται με τον constructor της κλάσης primitive, δίνοντας σαν ορίσματα κατά σειρά το

όνομα της μεταβλητής σε συμβολοσειρά, την τιμή της (input), "None" για τη συνάρτηση και την κενή λίστα για το parents. Τα δύο τελευταία συμπληρώνονται με αυτόν τον τρόπο, καθώς δεν είναι αποτέλεσμα συνάρτησης (εξού και ο όρος "αρχικές μεταβλητές"), οπότε τις συμβολίζουμε με "None", επειδή δεν έχουν μεταβλητές - γονείς. Μία μικρή μετατροπή που γίνεται στις αρχικές μεταβλητές είναι ότι το name συμπληρώνεται από το όνομα της μεταβλητής ανάμεσα σε '$'. Όταν αυτές δηλωθούν, δημιουργούν την αρχή του γράφου, ο οποίος σε αυτό το στάδιο είναι ένα n-ανεξάρτητο σύνολο, με n το πλήθος των δηλωμένων μεταβλητών. Όλα τα αντικείμενα της κλάσης primitive (μεταβλητές) που φτιάχτηκαν και θα φτιαχτούν στην πορεία, αποθηκεύονται στη λίστα `primitives_list`.

Στο `struct_and_math_functions.py` υπάρχουν επίσης εκτός της κλάσης και της λίστας, οι μαθηματικές συναρτήσεις που θα χρησιμοποιήσει ο χρήστης και καλούν τις αντίστοιχες της βιβλιοθήκης math. Επιπλέον, πέραν της αριθμητικής πράξης, δημιουργούν ένα νέο αντικείμενο primitive (κορυφή του γράφου). Έστω ότι στο ακόλουθο παράδειγμα έχουμε στα αριστερά τον κώδικα και στα δεξιά το αντίστοιχο μαθηματικό ισοδύναμο:

1. $x = primitive(\texttt{"}x\texttt{"}, 8, \texttt{"}None\texttt{"}, [\ ])$ | $x = 8$
2. $y = log(x, 2)$ | $y = \log_2 x$

Τα πεδία της μεταβλητής y συμπληρώνονται ως εξής:

- name: το όνομα της μεταβλητής συμπληρώνεται με τη συνάρτηση και μέσα σε παρένθεση τις μεταβλητές που χρησιμοποιούνται ($\log_2 x \rightarrow log(\$x\$)$)
- value: το αποτέλεσμα της συνάρτησης για δεδομένες τιμές των μεταβλητών ($\log_2 8 = 3$)
- function: η συνάρτηση που εφαρμόστηκε και είχε ως αποτέλεσμα αυτή τη μεταβλητή (log)
- parents: λίστα με τις μεταβλητές που χρησιμοποιήθηκαν στην προηγούμενη συνάρτηση, για να δημιουργηθεί η συγκεκριμένη μεταβλητή (στον παραπάνω παράδειγμα η x, άρα [$\$x\$$])

Το δυσκολότερο κομμάτι της υλοποίησης ήταν η δημιουργία των ακμών του γράφου που αναπαριστούν τις μαθηματικές πράξεις, όταν χρησιμοποιούνται για αυτές οι αριθμητικοί τελεστές της Pythons +, -, *, /, **. Η λύση ήταν να γίνει "υπερφόρτωση" (overloading) των τελεστών. Αρχικά, πρέπει να αποσαφηνιστεί τι ακριβώς αναπαριστούν οι τελεστές στην Python αλλά και σε κάθε άλλη σχεδόν γλώσσα, που δεν είναι κάτι διαφορετικό από μια συνάρτηση η οποία παίρνει δύο ορίσματα και επιστρέφει το αποτέλεσμα της αντίστοιχης αριθμητικής πράξης (π.χ. το $z = x + y$ είναι στην ουσία $z = +(x, y)$). Έτσι, προστέθηκαν ορισμένες επιπλέον λειτουργίες σε αυτούς, εκτός της προκαθορισμένης αριθμητικής πράξης. Κατά την κλήση κάποιου τελεστή, ελέγχεται αν τουλάχιστον ένα από τα δύο ορίσματα είναι μεταβλητή. Εάν είναι, τότε τα πεδία της μεταβλητής που γίνεται εκχώρηση του αποτελέσματος συμπληρώνονται όπως περιγράφηκε προηγουμένως με τις υπόλοιπες συναρτήσεις και η νέα μεταβλητή προστίθεται στην

λίστα `struct_and_math_functions.py`. Στην περίπτωση που και τα δύο ορίσματα είναι αριθμοί, καλείται ο προκαθορισμένος (default) τελεστής, οπότε γίνεται απλά η πράξη χωρίς τίποτα παραπάνω.

Στη συνέχεια ο χρήστης γράφει τον κώδικά του, που αποτελείται από αριθμητικές πράξεις, αναθέσεις τιμών (π.χ. $x = 2$), for/while - loops και if - else. Η μόνη προϋπόθεση για τις αριθμητικές πράξεις είναι να χρησιμοποιήσει τις συναρτήσεις του αρχείου `struct_and_math_functions.py` αντί της βιβλιοθήκης math. Παράλληλα, δημιουργείται ο γράφος που αναπαριστά τον κώδικα του χρήστη.

5.2.2 Παραγώγιση

Αφού περιγράφηκαν τα παραπάνω, ήρθε η ώρα να δούμε με ποιον τρόπο θα εφαρμοστεί ο αλγόριθμος του Backpropagation και πώς θα γίνει η παραγώγιση. Όλες οι σχετιζόμενες συναρτήσεις υπάρχουν στα αρχεία `differentiation.py` και `grad.py`. Η αρχή γίνεται όταν καλέσει ο χρήστης τη συνάρτηση grad με πρώτο όρισμα το όνομα της μεταβλητής που θέλει να παραγωγίσει (έστω x) και δεύτερο το όνομα της μεταβλητής ως προς την οποία γίνεται η παραγώγιση (έστω y). Αυτό προφανώς ισούται με το $\frac{\partial x}{\partial y}$. Εκεί γίνεται μια αναζήτηση στην primitives_list με βάση το όνομα για την εύρεση των μεταβλητών. Σύμφωνα με το Backpropagation, σειρά έχει η εύρεση των διαφορετικών μονοπατιών στον γράφο μεταξύ των δύο προαναφερθέντων μεταβλητών, το οποίο γίνεται με κάποιον αλγόριθμο αναζήτησης (π.χ. Dijkstra's). Στην δική μας υλοποίηση αυτό επιτυγχάνεται λίγο διαφορετικά, εκμεταλλευόμενοι το όνομα των μεταβλητών. Πιο συγκεκριμένα, από τη στιγμή που το όνομα μιας αρχικής μεταβλητής είναι το όνομα της μέσα σε '$' και οποιαδήποτε άλλης είναι η έκφραση που την έχει παράξει, αρκεί ένας έλεγχος για το αν η συμβολοσειρά '$$x$$' είναι υποσυμβολοσειρά του ονόματος της μεταβλητής y. Εάν είναι, τότε γνωρίζουμε ότι υπάρχει μονοπάτι $y \rightarrow x$. Για κάθε μαθηματική πράξη έχει υλοποιηθεί και μια συνάρτηση στον κώδικα, ώστε να πραγματοποιείται ο αντίστοιχος κανόνας παραγώγισης. Άρα, ανάλογα με τη συνάρτηση που έχει εφαρμοστεί, καλείται ο αντίστοιχος κανόνας παραγώγισης.

Στη συνέχεια χρησιμοποιεί τον κανόνα της αλυσίδας (3.1), υπολογίζοντας αναδρομικά το γινόμενο $\frac{\partial x}{\partial a_0} \cdot \frac{\partial a_0}{\partial a_1} \cdot ... \cdot \frac{\partial a_{n-1}}{\partial a_n} \cdot \frac{\partial a_n}{\partial y} = \frac{\partial x}{\partial y}$, όπου τα $a_i, i = 0, 1, ..., n$ είναι οι ενδιάμεσες κορυφές του μονοπατιού $y \rightarrow x$. Η πλοήγηση σε αυτό επιτυγχάνεται μέσω του πεδίου parent, στο οποίο υπάρχουν οι γονείς (κορυφές) της κάθε μεταβλητής. Άρα, κάθε μεταβλητή a_i του προηγούμενου γινομένου είναι γονέας της a_j αν $i \leq j$.

Η διαδικασία επαναλαμβάνεται αναδρομικά, ωσότου να είναι είτε κενή η λίστα parents ή να μην υπάρχει σαν υποσυμβολοσειρά στο εκάστοτε όνομα μεταβλητής η 'x'. Η κενή λίστα των γονέων σηματοδοτεί ότι το Backpropagation έφτασε σε αρχική μεταβλητή (δεν έχει γονέα), δηλαδή φύλλο του γράφου, οπότε έχει φτάσει σε άκρο του μονοπατιού. Αντίστοιχα, η δεύτερη περίπτωση ορίζει ότι η συγκεκριμένη μεταβλητή δεν σχετίζεται με την 'x', άρα δεν υπάρχει μονοπάτι $y \rightarrow x$ που να περιέχει την συγκεκριμένη κορυφή. Για τους παραπάνω λόγους και οι δύο περιπτώσεις δείχνουν το τέλος της αναδρομής.

Σε αυτό το σημείο πρέπει να τονιστεί η σημασία του χαρακτήρα '$', που λειτουργεί σαν "σημαία" (flag) και σηματοδοτεί την αρχή και το τέλος του ονόματος των αρχικών μεταβλητών. Έστω ότι δεν υπήρχαν τα '$' πριν και μετά ενώ είχαμε τις αρχικές μεταβλητές xx, x και ο κώδικας αποτελείται από τις ακόλουθες αριθμητικές πράξεις:

1. $x = primitive("x", 1, "None", [\ ])$
2. $xx = primitive("xx", 2, "None", [\ ])$
3. $y = 5 \cdot x + 10 \cdot xx$
4. $print(grad(y, x))$

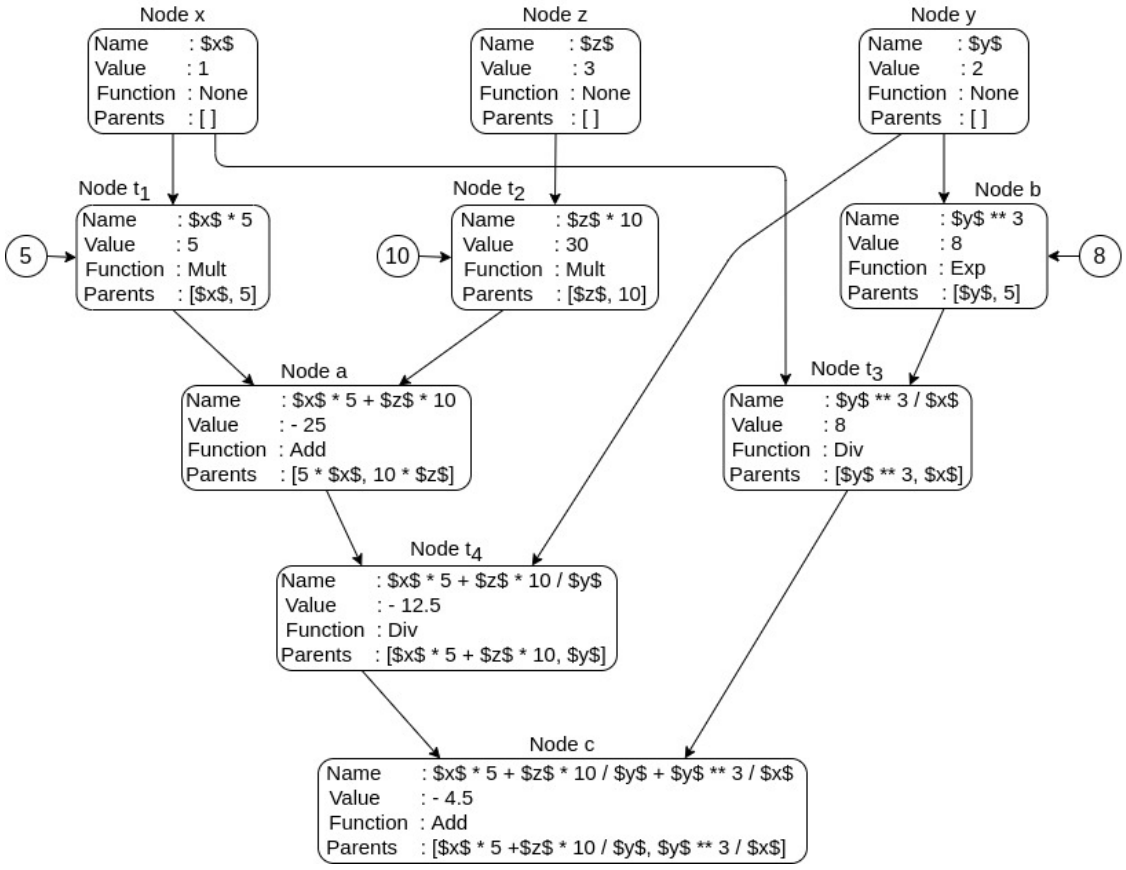
Προφανώς στον κώδικα οι δύο αυτές μεταβλητές είναι διαφορετικές και η έξοδος του προγράμματος ($\frac{\partial y}{\partial x}$) θα πρέπει να είναι η παράγωγος ως προς x της συνάρτησης $y(x, xx) = 5 \cdot x + 10 \cdot xx$ στο σημείο $(1, 2)$, δηλαδή $\frac{\partial y}{\partial x} = 5$. Όμως, όταν θα γίνει έλεγχος για την υποσυμβολοσειρά 'x', αυτή θα εμφανιστεί όχι μόνο στο '$5 \cdot x$' αλλά και στο '$10 \cdot xx$', καθώς το 'x' είναι υποσυμβολοσειρά του 'xx', με αποτέλεσμα να υπολογιστεί τελικά λανθασμένα η ζητούμενη παράγωγος.

5.2.3 Παράδειγμα εκτέλεσης κώδικα

Πιθανώς, όλα τα παραπάνω να φαίνονται πολύ μπερδεμένα, οπότε θα δείξουμε δυο παραδείγματα για να γίνουν όλα πιο ξεκάθαρα. Ας ακολουθήσουμε την εκτέλεση των παρακάτω κομματιών κώδικα, από τον γράφο που θα δημιουργηθεί μέχρι το αποτέλεσμα της εξόδου.

1. $x = primitive("x", 1, "None", [\ ])$
2. $y = primitive("y", 2, "None", [\ ])$
3. $y = primitive("z", 3, "None", [\ ])$
4. $a = 5 * x + 10 * z$
5. $b = y * *3$
6. $c = a/y - b/x + z$
7. $print(grad(c, y))$

Εάν γίνουν όλες οι αντικαταστάσεις, παρατηρούμε ότι η c είναι μια μεταβλητή, η τιμή της οποίας εξαρτάται από αυτή των x, y, z. Έτσι, καταλήγει κανείς ότι ο μαθηματικός τύπος της είναι $c(x, y, z) = \frac{x \cdot 5 + z \cdot 10}{y} - \frac{y^3}{x}$.



Εικόνα 2: Αναπαράσταση κώδικα με γράφο #1

Αρχικά, πρέπει να σημειωθεί ότι όλοι οι κόμβοι t_i, με $i = 1, 2, 3, 4$ αντιστοιχούν σε προσωρινές μεταβλητές, οι οποίες χρησιμοποιούνται αυτόματα από τον υπολογιστή, όταν γίνονται πολλές πράξεις σε μία γραμμή. Ο λόγος που υπάρχουν σαν κόμβοι στο παράδειγμα είναι για να γίνει ευκολότερη η κατανόησή του.

Στις πρώτες τρεις γραμμές καλείται ο constructor της κλάσης primitive, ώστε να δημιουργηθούν οι αρχικές μεταβλητές x, y, z και να συμπληρωθούν κατάλληλα τα πεδία τους. Αυτές αναπαρίστανται στον γράφο της *Εικόνας 2* με τους κόμβους (κορυφές) x, y, z. Στην τέταρτη γραμμή, η μεταβλητή a αποτελείται από δύο πολλαπλασιασμούς ("Mult") και μία αφαίρεση ("Sub"). Σύμφωνα με την προτεραιότητα των πράξεων γίνονται πρώτα οι δύο πολλαπλασιασμοί, οι οποίοι αποθηκεύονται στις μεταβλητές t_1, t_2, όπως έχουν ονομαστεί στο σχήμα και στη συνέχεια η διαφορά τους εκχωρείται στην μεταβλητή a. Τα πεδία της συμπληρώνονται αυτόματα με: name η έκφραση που την έχει παράξει, function το "Div", αφού είναι αποτέλεσμα μιας αφαίρεσης (τελικά), value ίσο με -25 (για $x = 1, z = 3$) ενώ η λίστα parents γεμίζει με $[5 * \$x\$, 10 * \$z\$]$ που είναι τα ονόματα των (προσωρινών) κορυφών t_1, t_2 από τις οποίες προήλθε η a. Με αντίστοιχο τρόπο δημιουργούνται και οι υπόλοιπες κορυφές μέχρι να συμπληρωθεί ο γράφος.

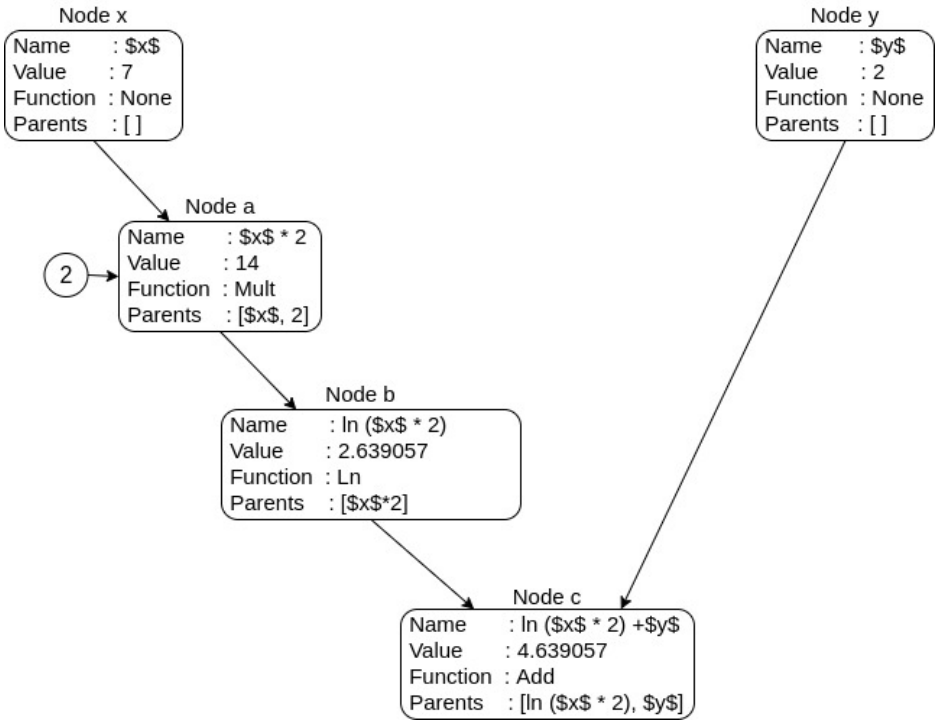
Έπειτα καλείται η συνάρτηση $grad(c, y)$, που επιστρέφει την τιμή της παραγώγου της

συνάρτησης $c(x,y,z) = \frac{x \cdot 5 + z \cdot 10}{y} - \frac{y^3}{x}$ ως προς y, δηλαδή $\frac{\partial y}{\partial x}$, για $x = 1, y = 2, z = 3$. Στην πρώτη κλήση της, η συνάρτηση grad βλέπει ότι η πράξη που εφαρμόστηκε τελευταία είναι η πρόσθεση. Άρα χρησιμοποιεί τον κανόνα της πρόσθεσης και επιστρέφει $\frac{\partial c}{\partial y} = \frac{\partial(\frac{x \cdot 5 + z \cdot 10}{y} - \frac{y^3}{x})}{\partial y} = \frac{\partial(\frac{x \cdot 5 + z \cdot 10}{y})}{\partial y} + \frac{\partial(\frac{y^3}{x})}{\partial y}$. Στην *Εικόνα 2* παρατηρούμε ότι υπάρχουν δύο μονοπάτια $c \to y$, το $c \to t_4 \to y$ και $c \to t_3 \to b \to y$. Βασιζόμενοι στο Backpropagation, θα πρέπει να τα προσθέσουμε σύμφωνα με τον κανόνα της πρόσθεσης. Προφανώς σε περίπτωση που έχουμε αφαίρεση δεν αλλάζει κάτι, καθώς οι δύο αυτές πράξεις είναι ισοδύναμες με διαφορετικό πρόσημο στην έκφραση μετά τον τελεστή. Έπειτα η grad καλείται αναδρομικά στις δύο μερικές παραγώγους, εφαρμόζοντας τον εκάστοτε κανόνα παραγώγισης κάθε φορά, για τον οποίο υπάρχει και η αντίστοιχη συνάρτηση στον κώδικα. Όταν φτάσει στον αρχικό κόμβο y, η αναδρομή θα σταματήσει επιστρέφοντας 1. Σε κάθε άλλη περίπτωση που θα βρεθεί εκτός των δύο προαναφερθέντων μονοπατιών θα επιστρέψει 0, αφού δεν θα υπάρχει στους parents η υποσυμβολοσειρά '$\$y\$$'. Κάνοντας τις πράξεις για επαλήθευση της εξόδου του προγράμματος, το αποτέλεσμα θα βγει ίσο με -20.75.

Ας δούμε άλλο ένα γρήγορο παράδειγμα εκτέλεσης κώδικα:

1. $x = primitive("x", 7, "None", [\ ])$
2. $y = primitive("y", 2, "None", [\ ])$
3. $a = x * 2$
4. $b = smf.ln(a)$
5. $c = b + y$
6. $print(grad(c, x))$

Κοιτάζοντας τον κώδικα, παρατηρούμε ότι σε κάθε ανάθεση τιμής στις γραμμές 2 έως και 5, γίνεται μια μόνο πράξη. Ο γράφος που δημιουργείται φαίνεται στην *Εικόνα 3* και ισχύουν ότι είδαμε στο προηγούμενο παράδειγμα.



Εικόνα 3: Αναπαράσταση κώδικα με γράφο #2

Εδώ θα αναφερθούμε μόνο στο Backpropagation και πώς συνδέεται το γινόμενο παραγώγων με το μονοπάτι του γράφου στον δικό μας κώδικα. Η έξοδος του προγράμματος είναι το $\frac{\partial c}{\partial x}$. Γνωρίζουμε ότι για την λύση θα πρέπει να υπολογίσουμε το ισοδύναμο γινόμενο $\frac{\partial c}{\partial x} = \frac{\partial c}{\partial b} \cdot \frac{\partial b}{\partial a} \cdot \frac{\partial a}{\partial x}$. Ας δούμε τώρα, πώς αυτό μεταφράζεται στο πρόγραμμά μας. Όταν καλείται η grad, ελέγχεται αν η μεταβλητή c έχει το '$\$x\$$' σαν υποσυμβολοσειρά κάποιου στοιχείου της λίστας parent $[log(\$x\$) * 3, \$y\$]$. Παρατηρούμε ότι για το πρώτο στοιχείο της λίστας, η παραπάνω συνθήκη είναι αληθής. Έτσι επιστρέφεται το γινόμενο $\frac{\partial c}{\partial b} \cdot \frac{\partial b}{\partial a} = \frac{\partial (b+y)}{\partial b} \cdot \frac{\partial b}{\partial a} = 1 \cdot \frac{\partial b}{\partial a}$. Η ίδια διαδικασία επαναλαμβάνεται αναδρομικά για τις μεταβλητές b και a, καλώντας την grad με πρώτο όρισμα την κατάλληλη μεταβλητή από την λίστα parent και δεύτερο πάντα την μεταβλητή x, ως προς την οποία γίνεται η παραγώγιση. Το τέλος της αναδρομής στο παράδειγμα θα έρθει, όταν καλεστεί η grad με τα δύο ορίσματα ίδια, δηλαδή $grad(x, x)$. Άρα, τελικά το γινόμενο θα μεγαλώσει και θα φτάσει στην ζητούμενη μορφή, περνώντας από τα ακόλουθα στάδια: $\frac{\partial (b+y)}{\partial b} \cdot \frac{\partial b}{\partial a} = 1 \cdot \frac{\partial b}{\partial a} \rightarrow 1 \cdot \frac{\partial (ln(a)}{\partial a} \cdot \frac{\partial a}{\partial x} = 1 \cdot \frac{1}{a} \cdot \frac{\partial 2 \cdot x}{\partial x} \rightarrow 1 \cdot \frac{1}{a} \cdot 2 = \frac{2}{14} = 0.142857$. Πράγματι, αν γίνουν οι πράξεις το αποτέλεσμα επαληθεύεται.

5.3 Περιορισμοί

Στην δική μας υλοποίηση του Autograd υπάρχουν κάποιοι περιορισμοί. Αρχικά, θα αναφερθούν ορισμένα τεχνικά ζητήματα που πρέπει να λάβει υπόψη του ο χρήστης. Το πρώτο πρόβλημα που εντοπίζεται και πρέπει να γνωρίζει ο χρήστης, είναι το γεγονός

ότι, αν θέλει να κάνει μια πράξη με κάποιο από τα operators της Python ($+, -, *, /, **$) ανάμεσα σε μεταβλητή τύπου primitive και κάποιον αριθμό (π.χ. int, double), πρέπει να γράφεται πρώτα η μεταβλητή και μετά ο αριθμός. Το operator δεν είναι κάτι διαφορετικό από μια συνάρτηση με ορίσματα. Για παράδειγμα το $x + y$ είναι στην πραγματικότητα η συνάρτηση `__add__(x, y)`, η οποία έχει σαν ορίσματα τα x, y. Εάν το πρώτο όρισμα είναι αριθμός τύπου int, double κλπ, τότε καλείται ο default operator της Python, με αποτέλεσμα να επιστρέφει την αντίστοιχη πράξη (πρόσθεση στη συγκεκριμένη περίπτωση) ανάμεσα στα ορίσματα. Όταν όμως το δεύτερο όρισμα είναι μεταβλητή τύπου primitive, η πράξη δεν γίνεται και ορθώς εμφανίζεται TypeError. Αντίθετα, αν το πρώτο όρισμα είναι μεταβλητή τύπου primitive, καλείται ο δικός μας operator, κάνοντας την διαδικασία που έχει περιγραφεί. Προφανώς αυτό δεν μειώνει την λειτουργικότητα, καθώς μπορεί κάποιος να φτιάξει ένα νέο αντικείμενο τύπου primitive και να του δώσει την αριθμητική τιμή που θέλει, οπότε να αντικαταστήσει τη σταθερά με αυτό (σαν να ορίζει αρχική μεταβλητή). Εννοείται ότι όπου ισχύει η αντιμεταθετική ιδιότητα ($+, *$), η πιο απλή λύση είναι η αντιστροφή των δύο ορισμάτων.

Μια άλλη επισήμανση που πρέπει να σημειωθεί είναι ότι δεν υπάρχει ειδική συνάρτηση για την αναπαράσταση της n-οστής ρίζας, $n \in \mathbb{N}$. Καθώς δεν είναι κάποιος διαφορετικός κανόνας από την ύψωση σε δύναμη, αναπαριστάται σαν ύψωση στην $\frac{1}{n}$.

Τέλος το σημαντικότερο στοιχείο που πρέπει να τονιστεί, είναι το γεγονός ότι, αν ζητηθεί να γίνει η παραγώγιση ως προς κάποια μεταβλητή, εκτός των αρχικοποιημένων με τον constructor της κλάσης primitive, δεν βγάζει πάντα σωστό αποτέλεσμα. Αυτό είναι απόλυτα λογικό, καθώς το όνομα των μη αρχικοποιημένων από τον constructor μεταβλητών είναι μια έκφραση με πράξεις μεταξύ αρχικοποιημένων. Επομένως, υπάρχει περίπτωση αυτή η ίδια έκφραση να έχει δημιουργηθεί και σε κάποια άλλη μεταβλητή, οπότε να λάβει υπόψη του το πρόγραμμα ότι κι αυτή η μεταβλητή πρέπει να πάρει μέρος στην παραγώγιση, παρόλο που είναι διαφορετική.

5.4 Μελλοντικές Βελτιώσεις

Δεν χρειάζεται να αναφέρουμε, ότι καθώς είναι μια πρώτη προσπάθεια υλοποίησης συστήματος αντίστοιχου του Autograd, σίγουρα υπάρχουν τρόποι για επιπλέον λειτουργικότητα. Όπως περιγράφηκε προηγουμένως, αν ζητηθεί να γίνει η παραγώγιση ως προς κάποια μεταβλητή πέρα των αρχικών, το αποτέλεσμα δεν είναι πάντα ορθό. Για παράδειγμα, αν έχουμε στον κώδικα ότι $b = x + 1$ και ζητηθεί η παράγωγος του b, τότε το πρόγραμμα θα ψάξει την παράγωγο ως προς $x + 1$. Έτσι, αν υπάρχει κι αλλού το $x + 1$, θα το χειριστεί σαν b, οπότε θα ληφθεί υπόψη στην παραγώγιση. Αυτή δεν είναι

αυτόματα λανθασμένη πορεία σκέψης, ωστόσο είναι πιθανότερο ο χρήστης να μπερδευτεί και να περιμένει διαφορετικό αποτέλεσμα. Άρα καλό θα ήταν να αποφεύγεται για την ώρα.

6. ΕΠΙΛΟΓΟΣ

Αρχικά ο στόχος μας ήταν να δούμε πώς το ∂P μπορεί να σταθεί σαν αντικείμενο προγραμματισμού και πώς θα μπορούσε ένα πρόγραμμα γραμμένο σε μια συνηθισμένη γλώσσα προγραμματισμού (π.χ. C, Python) να μετατραπεί σε κώδικα ∂P. Ουσιαστικά, να δούμε ποια είναι τα ισοδύναμα των συμβολοσειρών, δομών δεδομένων, if - else, while/for - loops κ.ο.κ., όπως τα γνωρίζουμε από τον Προστακτικό Προγραμματισμό (Imperative Programming), στον Διαφορίσιμο Προγραμματισμό. Μετά από αρκετή έρευνα παρατηρήσαμε ότι το Differentiable Programming σαν είδος προγραμματισμού, ήταν σε σχετικά πρώιμο στάδιο, με πληθώρα διαφορετικών απόψεων, στην πλειονότητά τους άμεσα ή έμμεσα συνδεδεμένες με το Deep Learning. Στη συνέχεια ανακαλύψαμε το Autograd και αφού το κατανοήσαμε, προσπαθήσαμε να δημιουργήσουμε κι ένα παρόμοιο εργαλείο παραγώγισης βασιζόμενοι σε αυτό.

Ελπίζουμε στο μέλλον να ασχοληθεί περισσότερος κόσμος με το αντικείμενο του Differentiable Programming, καθώς κατά τη γνώμη μας φαίνεται ότι έχει τις βάσεις να αποτελέσει σημαντική εξέλιξη στον προγραμματισμό και στην επιστήμη της πληροφορικής γενικότερα.

7. ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

1. Differentiable Programming: Διαφορίσιμος Προγραμματισμός
2. Autograd: Εργαλείο της PyTorch
3. PyTorch: βιβλιοθήκη μηχανικής εκμάθησης ανοιχτού κώδικα
4. Backpropagation: Οπισθοδιάδοση
5. Automatic Differentiation: αυτόματη παραγώγιση
6. Deep Learning: Νευρωνική Μάθηση
7. Functional Programming: Συναρτησιακός Προγραμματισμός
8. Differentiable Functional Programming: Διαφορίσιμος Συναρτησιακός Προγραμματισμός
9. Gradient Descent: αλγόριθμος σύγκλισης με ελάττωση της παραγώγου
10. optimization: βελτιστοποίηση
11. Symbolic Differentiation: Συμβολική παραγώγιση
12. Forward Accumulation: "Προς τα εμπρός" παραγώγιση
13. Reverse Accumulation: Ανάποδη παραγώγιση
14. Dual Numbers: Διπλοί Αριθμοί
15. Turing Award: Βραβείο Turing
16. Loss Function: Συνάρτηση απόκλισης
17. math: βιβλιοθήκη με μαθηματικές συναρτήσεις
18. C-based: Βασισμένο στη γλώσσα C
19. In-place operations: Πράξη ή λειτουργία στον κώδικα, γραμμένη σε μια γραμμή
20. Github: εταιρεία που παρέχει φιλοξενία για τον έλεγχο της έκδοσης λογισμικού ανάπτυξης χρησιμοποιώντας το Git
21. constructor: κατασκευαστής, συνάρτηση που αρχικοποιεί αυτόματα τα αντικείμενα μιας κλάσης

22. primitive: αρχικός
23. overloading function: δήλωση συνάρτησης με περισσότερους από έναν τύπους
24. flag: σημαία, χρησιμοποιείται σαν σήμα για κάποια λειτουργία του προγράμματος
25. int: ακέραιος αριθμός (τύπος δεδομένων)
26. double: πραγματικός αριθμός (τύπος δεδομένων)
27. default operator: προκαθορισμένος τελεστής με συγκεκριμένη λειτουργία από την γλώσσα
28. TypeError: Συγκεκριμένο μήνυμα λάθους από την Python κατά την εκτέλεση
29. Imperative Programming: Προστακτικός Προγραμματισμός

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] C. Olah. ”Neural Networks, Types, and Functional Programming”. github.com http://colah.github.io/posts/2015-09-NN-Types-FP/ (accessed June 22, 2021)

[2] G. Saupin. ”Differentiable Programming from Scratch”. towardsdatascience.com https://towardsdatascience.com/differentiable-programming-from-scratch-abba0ebebc1c (accessed June 22, 2021)

[3] H. J. Liao, J. G. Liu, L. Wang and T. Xiang, ”Differentiable Programming Tensor Networks”, in *Physical Review X, Vol. 9, Iss. 3, 031041*, Sept. 2019, doi:

[4] M. Hackenberg, M. Grodd, C. Kreutz, M. Fischer, J. Esins, L. Grabenhenrich, C. Karagiannidis and H. Binder, ”Using Differentiable Programming for Flexible Statistical Modeling”, 2020. [Online]. Available: https://arxiv.org/pdf/2012.05722.pdf

[5] A. Hernandez and J. M. Amigo, ”Differentiable programming and its applications to dynamical systems”, 2020. [Online]. Available: https://arxiv.org/pdf/1912.08168.pdf

[6] S. Scardapane. ”Deep learning from a programmer’s perspective (aka Differentiable Programming)”. towardsdatascience.com https://towardsdatascience.com/deep-learning-from-a-programmers-perspective-aka-differentiable-programming-ec6e8d1b7c60 (accessed June 22, 2021)

[7] F. Wang, J. Decker, X. Wu, G. Essertel and T. Rompf, ”Backpropagation with Continuation Callbacks: Foundations for Efficient and Expressive Differentiable Programming,” in *32nd Conference on Neural Information Processing Systems (NIPS)*, Montréal, Canada, 2018, pp. 10201-10212.

[8] C. Mak and L. Ong, ”A Differential-form Pullback Programming Language for Higher-order Reverse-mode Automatic Differentiation”, 2020: https://arxiv.org/pdf/2002.08241.pdf

[9] C. C. Margossian, ”A Review of Automatic Differentiation and its Efficient Implementation”, in *Wiley interdisciplinary reviews: data mining and knowledge discovery 9.4*, Jan. 2019, doi: 10.1002/WIDM.1305.

[10] L. Szirmay-Kalos, “Higher Order Automatic Differentiation with Dual Numbers”, *Period. Polytech. Elec. Eng. Comp. Sci.*, vol. 65, no. 1, pp. 1-10, Jan. 2021.

[11] M. Abadi and G. D. Plotkin, ”A Simple Differentiable Programming Language”, in *Proc. of the ACM on Programming Languages, Vol. 4*, Jan. 2020, pp. 1-28 doi: 10.1145/3371106.

[12] F. Penunuri, R. Peon-Escalante, C. Villanueva, C. A. Cruz-Villar, ”A Dual Number Approach for Numerical Calculation of derivatives and its use in the Spherical 4R Mechanism”, 2018: https://arxiv.org/pdf/1301.1409.pdf

[13] Innes Michael J, ”Don’t unroll adjoint: Differentiating SSA-Form programs”, 2019: https://arxiv.org/pdf/1810.07951.pdf

[14] Yann LeCun. ”OK, Deep Learning ... on this later....”. facebook.com https://www.facebook.com/yann.lecun/posts/10155003011462143 (accessed June 22, 2021)

[15] ”Fathers of the Deep Learning Revolution Receive ACM A.M. Turing Award”. awards.acm.org https://awards.acm.org/about/2018-turing (accessed June 22, 2021)

[16] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah and W. Tebbutt, ”∂P: A Differentiable Programming System to Bridge Machine Learning and Scientific Computing”, 2019: https://arxiv.org/pdf/1907.07587.pdf

[17] Nielsen Michael A., “How the backpropagation algorithm works,” in *Neural Networks and Deep Learning*. San Francisco, CA, USA: Determination Press, 2015, ch. 2, pp. 23-54.

[18] A. Kathuria. ”PyTorch 101, Part 1: Understanding Graphs, Automatic Differentiation and Autograd”. blog.paperspace.com https://blog.paperspace.com/pytorch-101-understanding-graphs-and-automatic-differentiation/ (accessed June 22, 2021)

[19] Johnson Matthew James, Montreal, Canada. *Automatic Differentiation*. (July 27, 2017). Accessed: June 21, 2021. [Online Video]. Available: http://videolectures.net/deeplearning2017_johnson_automatic_differentiation/

[20] R. Grosse. (2018). Automatic Differentiation [Beamer (LaTeX) slides]. Available: https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec10.pdf

[21] PyTorch Contributors. *Autograd Mechanics* (2019). Accessed: Jun 22, 2021. [Online]. Available: https://pytorch.org/docs/stable/notes/autograd.html

[22] P. Maeder-York, A. Nitido, D. Randle and S. Sebb. *Autograd Documentation* (2019). Accessed: Jun 22, 2021. [Online]. Available: `https://autograd.readthedocs.io/en/latest/index.html`

[23] A. Kak and C. Bouman. (2021). Autograd: for Automatic Differentiation and for Auto-Construction of Computational Graphs [Beamer (LaTeX) slides]. Available: `https://engineering.purdue.edu/DeepLearn/pdf-kak/week4.pdf`