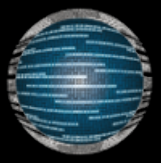


Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πανεπιστήμιο Πατρών
29 Ιανουαρίου 2021

Παράλληλος Προγραμματισμός
4ο *Εργαστήριο*

Ανδρέας Καρατζάς

Περιεχόμενα

3 Κεφαλαιο 1
Νευρωνικό Δίκτυο

1.1 Εισαγωγή 3
1.2 GitHub 3
1.3 Ανάλυση προσέγγισης 4
1.4 Σημαντικές κλάσεις 5
1.4.1 Η κλάση dataset 5
1.4.2 Η κλάση nn 5
1.5 Ανάλυση υλοποίησης 6
1.5.1 Driver.hpp και Driver.cpp 7
1.5.2 Neural.hpp 7
1.5.3 Activation.hpp και Activation.cpp 8
1.5.4 Interface.hpp και Interface.cpp 8
1.5.5 Parser.hpp και Parser.cpp 8
1.5.6 Common.hpp 8
1.6 Αποτελέσματα 9

11 Κεφαλαιο 2
Ο κώδικας

1

Νευρωνικό Δίκτυο

1.1 Εισαγωγή

Για τη 4η εργαστηριακή άσκηση υλοποιήθηκε απλό νευρωνικό δίκτυο πρόσθιας τροφοδότησης με σιγμοειδή συνάρτηση ενεργοποίησης. Τα στοιχεία του συστήματος στο οποίο έγιναν οι μετρήσεις παρουσιάζονται στον Πίνακα 1.

Πίνακας 1: Στοιχεία Συστήματος

Λειτουργικό Σύστημα	Windows 10 x64
Επεξεργαστής	Intel Core i7-9750H CPU @ 2.60 GHz
RAM frequency	2667 MHz
RAM size	16 GB
IDE	Visual Studio 2019
Compiler	Intel 19.1
Programming Language	C++ 17

1.2 GitHub

Η εργασία έχει ανέβει και στο GitHub. Υπάρχουν και οδηγίες εγκατάστασης, μαζί με το makefile που διευκολύνει τον εξεταστή. Συνοπτικά, η διαδικασία εγκατάστασης σε περιβάλλον Unix έχεις ως εξής:

- Άνοιγμα παραθύρου *Terminal*
- Περιήγηση σε ένα άδειο directory μέσω της εντολής `cd`, στο οποίο θα γίνει το cloning του repository
- Κλωνοποίηση του repository μέσω της εντολής
  `git clone https://github.com/andreasceid/neural-network.git`
- Περιήγηση στο main directory του repository με την εντολή
  `cd neural-network/`
- Τροποποίηση του project μέσω του αρχείου `Common.hpp`. Μια από τις αλλαγές που θα πρέπει να γίνουν στο συγκεκριμένο αρχείο, είναι το μονοπάτι των αρχείων εκπαίδευσης

- (Προαιρετικό) Τροποποίηση του makefile του project με σκοπό την εμφάνιση πληροφοριών βελτιστοποίησης του κώδικα
- Μεταγλώττιση του project εκτελώντας την εντολή `make`
- Περιήγηση στο directory που βρίσκεται το *build* του project με την εντολή `cd build/`
- Δημιουργία του directory *data* όπου θα τοποθετηθούν τα αρχεία CSV για την εκπαίδευση του νευρωνικού δικτύου εκτελώντας την εντολή `mkdir data`
- Εξαγωγή των ζητούμενων CSV στο directory *data*
- Εκτέλεση του build με την εντολή
  `./neural-network -i <int> -h <int> [-h <int>] -o <int>`

1.3 Ανάλυση προσέγγισης

Το νευρωνικό δίκτυο που υλοποιήθηκε στα πλαίσια του εργαστηρίου είναι ένα κλασσικό δίκτυο πρόσθιας τροφοδότησης. Η συνάρτηση ενεργοποίησης που χρησιμοποιείται στα πλαίσια της εργασίας είναι η σιγμοειδής (ή λογιστική) συνάρτηση. Υπάρχει και η υλοποίηση της συνάρτησης «ReLU». Αυτό έγινε για πειραματικό σκοπό. Ο κώδικας της εργασίας περιλαμβάνει πολλές επιπλέον δυνατότητες και χαρακτηριστικά από αυτά που διευκρινίζονται στην αντίστοιχη εκφώνηση. Αρχικά έχει υλοποιηθεί *progress bar* το οποίο ενημερώνει με λεπτομέρεια τον χρήστη για την πρόοδο της εκάστοτε εργασίας. Επίσης, υλοποιήθηκε μια έκδοση της σιγμοειδούς συνάρτησης χρησιμοποιώντας σειρές Taylor. Ωστόσο, ο μεγάλος αριθμός εποχών κατέστησε απαραίτητη μια ακριβέστερη υλοποίηση. Η συνάρτηση κόστους που υλοποιήθηκε είναι η *Mean Squared Error*. Ακόμα, μετά από την εκπαίδευσή του, το νευρωνικό δίκτυο αποθηκεύεται έτσι ώστε να επαναχρησιμοποιηθεί στο μέλλον. Η υπεύθυνη συνάρτηση εξάγει τα βάρη των συνάψεων των νευρώνων. Υπήρξε προσπάθεια χρησιμοποιώντας *ANSI escape codes* να ενημερώνεται ασύγχρονα κάποιο progress bar κατά την εκπαίδευση του νευρωνικού δικτύου και ταυτόχρονα να εκτυπώνονται και τα αποτελέσματα της κάθε εποχής. Τέλος, υλοποιήθηκε και συνάρτηση που αξιολογεί την ακρίβεια των προβλέψεων του νευρωνικού δικτύου. Με τον όρο «ακρίβεια» γίνεται αναφορά στον αριθμό των σωστών προβλέψεων του νευρωνικού δικτύου. Επομένως, αν για παράδειγμα η επιθυμητή έξοδος για μια είσοδος είναι $0.0, 1.0$ για τον $1^ο$ και $2^ο$ νευρώνα αντίστοιχα, και το νευρωνικό δίκτυο επιστρέψει $0.2, 0.8$, τότε το νευρωνικό δίκτυο θα κριθεί σωστό, και η συνάρτηση θ επιστρέψει 1. Ουσιαστικά, αν ο νευρώνας εξόδου που αντιστοιχεί στη σωστή κλάση έχει τη μέγιστη τιμή ανάμεσα σε όλους τους νευρώνες εξόδου, τότε η πρόβλεψη θεωρείται σωστή.

1.4 Σημαντικές κλάσεις

Οι κύριες κλάσεις της εργασίας είναι:

- Η κλάση *dataset*
- Η κλάση *nn*

1.4.1 Η κλάση dataset

Η κλάση *dataset* διαχειρίζεται τα δεδομένα που τροφοδοτούνται στο νευρωνικό δίκτυο. Διαθέτει τις παρακάτω λειτουργίες:

- Αρχικοποίηση των δεδομένων των αντικειμένων αυτής της κλάσεις από CSV αρχείο.
- Εκτύπωση του συνόλου δεδομένων

Κατά το φόρτωμα των δεδομένων σε ένα αντικείμενο τύπου dataset, γίνεται κανονικοποίηση της εισόδου με βάση μια μέγιστη τιμή που δίνεται από τον χρήστη. Αυτό το βήμα είναι απαραίτητο για την καλύτερη λειτουργία του νευρωνικού δικτύου. Σε περίπτωση εισαγωγής μη κανονικοποιημένων δεδομένων στον νευρωνικό δίκτυο, τότε υπάρχει μεγάλη πιθανότητα υπερχείλισης σε κάποιον νευρώνα, λόγω της σιγμοειδούς. Ακόμα και χωρίς να υπάρχει το φαινόμενο της υπερχείλισης όμως, το νευρωνικό δίκτυο δε θα εκπαιδευτεί σωστά, καθώς η δυσκολία ανίχνευσης των ζητούμενων μοτίβων στα δεδομένα εισόδου είναι μεγάλη. Για την αρχικοποίηση ενός αντικειμένου dataset, ο χρήστης θα πρέπει να γνωρίζει τον αριθμό των κλάσεων που υπάρχουν σε αυτό το dataset, καθώς και τον αριθμό των παραδειγμάτων. Τα δεδομένα ενός αντικειμένου dataset θα πρέπει να είναι ακέραιοι αριθμοί, όπως στην περίπτωση της άσκησης. Τέλος η πρώτη στήλη θα πρέπει να είναι η ετικέτα της κλάσης στην οποία ανήκει το εκάστοτε παράδειγμα, και η πρώτη σειρά του CSV θα πρέπει να είναι η λεκτική περιγραφή της εκάστοτε στήλης.

1.4.2 Η κλάση nn

Η κλάση *nn* είναι διαχειρίζεται το νευρωνικό δίκτυο το οποίο επιθυμεί να δημιουργήσει ο χρήστης. Παρόλο που η συνάρτηση ενεργοποίησης είναι για κάθε νευρώνα η σιγμοειδής, ο χρήστης μπορεί να αλλάξει τη δομή του νευρωνικού δικτύου όπως επιθυμεί. Ο χρήστης είναι υπεύθυνος να δώσει τον κατάλληλο αριθμό νευρώνων εισόδου κι εξόδου, ανάλογα με το σύνολο δεδομένων στο οποίο πρόκειται να εκπαιδεύσει το νευρωνικό δίκτυο. Επίσης, η δομή του νευρωνικού δικτύου είναι δυναμική. Συγκεκριμένα, ο χρήστης μπορεί να δώσει διαφορετικά μεγέθη στο κρυφό επίπεδο, αλλά και πολλαπλά κρυφά επίπεδα. Η δομή του νευρωνικού δικτύου καθορίζεται από τα ορίσματα που δίνει ο χρήστης στο εκτελέσιμο. Υπάρχουν 3 «σημαίες» που αναγνωρίζει το πρόγραμμα:

- `-i`: σημαία που δηλώνει τον αριθμό των νευρώνων στο επίπεδο εισόδου. Για παράδειγμα, `-i 784` σημαίνει 785 (784 + 1 bias) νευρώνες στο επίπεδο εισόδου[1].
- `-h`: σημαία που δηλώνει τον αριθμό των νευρώνων σε κάποιο κρυφό επίπεδο. Ο χρήστης μπορεί να δημιουργήσει πολλαπλά κρυφά επίπεδα. Για παράδειγμα, `-h 100 -h 50` σημαίνει τη δημιουργία πρώτου κρυφού επιπέδου 101 (100 + 1 bias) νευρώνων και δεύτερο κρυφό επίπεδο με 51 (50 + 1 bias) νευρώνες.
- `-o`: σημαία που δηλώνει τον αριθμό των νευρώνων στο επίπεδο εξόδου. Για παράδειγμα, `-o 10` σημαίνει 10 νευρώνες στο επίπεδο εξόδου. Αξίζει να σημειωθεί ότι στο επίπεδο εξόδου δεν υπάρχει κάποιος νευρώνας τύπου *bias*.

Όπως αναφέρθηκε παραπάνω, όλοι οι νευρώνες ενεργοποιούνται καλώντας τη σιγμοειδή συνάρτηση. Τα ορίσματα που αναφέρθηκαν παραπάνω πρέπει να δοθούν με αυτήν τη σειρά από το χρήστη κατά την κλήση του εκτελέσιμου. Για τις ανάγκες της εργασίας για παράδειγμα, ο χρήστης θα πρέπει να καλέσει το πρόγραμμα ως εξής:

- Σε περιβάλλον Windows: `.\neural-network.exe -i 784 -h 100 -o 10`
- Σε περιβάλλον Unix: `./neural-network -i 784 -h 100 -o 10`

Ένα ακόμα επιπλέον χαρακτηριστικό είναι και ο ρυθμός μάθησης του νευρωνικού δικτύου. Ο ρυθμός αυτός ορίζεται στην κεφαλίδα `Common.hpp` και είναι αρχικοποιημένος με 0.1. Το χαρακτηριστικό αυτό βοηθάει κατά την εκπαίδευση έτσι ώστε το νευρωνικό δίκτυο να προσπαθήσει να αποφύγει κάποιο τοπικό ελάχιστο.

1.5 Ανάλυση υλοποίησης

Σε αυτήν τη παράγραφο θα γίνει η ανάλυση των αρχείων της εργασίας. Για την εγκατάσταση των αρχείων και την εκτέλεση της εργασίας, υπάρχουν οδηγίες στην παράγραφο 1.2. Για να κατανοηθεί γρηγορότερα η αρχιτεκτονική του λογισμικού, γίνεται αντιστοίχιση των μεταβλητών που περιγράφονται στην εκφώνηση με τις μεταβλητές που υπάρχουν στο πρόγραμμα:

- Οι μεταβλητές `WL1[]` και `WL2[]`: Οι μεταβλητές αυτές υπάρχουν ως `weights` μέσα στην κλάση `nn`.
- Οι μεταβλητές `DL1[]` και `DL2[]`: Οι μεταβλητές αυτές υπάρχουν ως `z` μέσα στην κλάση `nn`.
- Οι μεταβλητές `OL1[]` και `OL2[]`: Οι μεταβλητές αυτές υπάρχουν ως `a` μέσα στην κλάση `nn`.

[1] Ο νευρώνας *bias* προστίθεται αυτόματα, και δε χρειάζεται να προσμετρηθεί από το χρήστη.

- Η ρουτίνα `activateNN`: Η ρουτίνα αυτή υπάρχει ως `forward` μέσα στην κλάση `nn`.
- Η ρουτίνα `trainNN`: Η ρουτίνα αυτή έχει χωριστεί σε 2 επιμέρους ρουτίνες, την `back_propagation` και την `optimize` μέσα στην κλάση `nn`.

1.5.1 Driver.hpp και Driver.cpp

Τα αρχεία αυτά είναι υπεύθυνα για την εκτέλεση του project. Όπως απορρέει κι από το όνομά τους αυτά τα αρχεία λειτουργούν ως drivers. Το αρχείο κεφαλίδας δηλώνει τις κεφαλίδες που θα χρειαστούν για την ικανοποίηση των ζητουμένων της εργασίας. Το αρχείο `CPP` αναλαμβάνει το φόρτωμα των δεδομένων εκπαίδευση καθώς και την εκπαίδευση του νευρωνικού δικτύου που δημιουργείται.

1.5.2 Neural.hpp

Στην κεφαλίδα `Neural.hpp` ορίζεται η κλάση του νευρωνικού δικτύου. Τα αντικείμενα της κλάσεις είναι τύπου `nn`. Όλες οι συναρτήσεις της κλάσης ορίζονται στα αρχεία:

- `Fit.cpp`. Στο αρχείο `Fit.cpp` ορίζονται οι συναρτήσεις υπεύθυνες για την εκπαίδευση και την αξιολόγηση του νευρωνικού δικτύου.
- `Forward.cpp`. Στο αρχείο `Forward.cpp` ορίζεται ο αλγόριθμος πρόσθιας διάδοσης της εισόδου που δίνεται στο νευρωνικό δίκτυο.
- `Optimize.cpp`. Στο αρχείο `Optimize.cpp` ορίζεται ο αλγόριθμος πίσω διάδοσης του σφάλματος του νευρωνικού δικτύου καθώς και η συνάρτηση διόρθωσης των βαρών των συνάψεων.
- `Loss.cpp`. Στο αρχείο `Loss.cpp` ορίζεται η συνάρτηση κόστους μιας πρόβλεψης του νευρωνικού.
- `Utilities.cpp`. Στο αρχείο `Utilities.cpp` ορίζονται οι συναρτήσεις που αρχικοποιούν τις παραμέτρους του νευρωνικού δικτύου. Επίσης, ορίζεται και η συνάρτηση `zero_grad`, η οποία αρχικοποιεί πριν κάθε πρόβλεψη τις μεταβλητές που χρησιμοποιεί το νευρωνικό δίκτυο. Αυτές οι μεταβλητές αλλάζουν μετά από κάθε κλήση του νευρωνικού δικτύου ανάλογα με τις τιμές στο επίπεδο εισόδου. Επίσης, ορίζεται και η συνάρτηση `predict` η οποία καλεί το νευρωνικό δίκτυο με δεδομένα που εισάγει ο χρήστης. Ωστόσο, ο χρήστης είναι υπεύθυνος για την κανονικοποίησή τους. Τέλος, υπάρχει και η συνάρτηση `summary` η οποία εκτυπώνει στο χρήστη τη δομή του νευρωνικού δικτύου μετά από τη μεταγλώττισή του.
- `Accuracy.cpp`. Στο αρχείο `Accuracy.cpp` ορίζεται η συνάρτηση που αξιολογεί την ακρίβεια μιας πρόβλεψης του νευρωνικού δικτύου.

- `Esport.cpp`. Στο αρχείο `Export.cpp` ορίζεται η συνάρτηση εξαγωγής των βαρών των συνάψεων του νευρωνικού δικτύου.

1.5.3 Activation.hpp και Activation.cpp

Στα αρχεία αυτά ορίζονται οι διαθέσιμες συναρτήσεις ενεργοποίησης των νευρώνων του μοντέλου της άσκησης. Υπάρχουν 2 υλοποιήσεις της σιγμοειδούς. Για λόγους ακρίβειας στα αποτελέσματα, προτιμήθηκε συνάρτηση που καλεί τη ρουτίνα `std::pow()` της βιβλιοθήκης `math.h` (ή `cmath`). Κατά την υλοποίηση, δοκιμάστηκε και η ρουτίνα `std::exp()` της ίδιας βιβλιοθήκης. Ωστόσο, τα αποτελέσματα δεν ήταν ικανοποιητικά για άγνωστο μέχρι στιγμής λόγο. Αυτό ήταν η αιτία δημιουργίας μιας *custom* υλοποίησης της συνάρτησης αυτής. Η προσαρμοσμένη υλοποίηση εκμεταλλεύεται προσεγγίζει το ζητούμενο αποτέλεσμα χρησιμοποιώντας τη σειρά Taylor. Για ερευνητικό σκοπό, έγινε υλοποίηση της συνάρτησης ενεργοποίησης *ReLU*. Ωστόσο, στο πρόγραμμα που δίνεται χρησιμοποιείται μόνο η ασφαλής έκδοση της σιγμοειδούς.

1.5.4 Interface.hpp και Interface.cpp

Στα αρχεία αυτά ορίζονται συναρτήσεις που διαχειρίζονται την εκτύπωση μηνυμάτων και αποτελεσμάτων στην οθόνη του χρήστη. Υπάρχουν πολλές επιπλέον συναρτήσεις, από τις οποίες οι περισσότερες έχουν να κάνουν με την υλοποίηση ενός είδους cross-platform CLI για τις ανάγκες του προγράμματος. Ωστόσο, οι συναρτήσεις αυτές λόγω bugs, δεν καλούνται στο πρόγραμμα[2]. Εδώ υλοποιείται και μια κλάση που εκτυπώνει *progress bar* στον χρήστη. Η συγκεκριμένη δυνατότητα είναι πολύ χρήσιμη για την παρακολούθηση χρονοβόρων διεργασιών, όπως το φόρτωμα των δεδομένων εκπαίδευσης.

1.5.5 Parser.hpp και Parser.cpp

Στα αρχεία αυτά ορίζονται συναρτήσεις υπεύθυνες για την προσπέλαση των ορισμάτων του χρήστη. Υπάρχουν μερικές δικλείδες ασφαλείας σε περίπτωση που ο χρήστης δεν εισάγει σωστά δεδομένα. Ωστόσο, ο χρήστης θα πρέπει να προσέξει τη σειρά με την οποία θα δώσει τα ορίσματα στο πρόγραμμα, όπως περιγράφηκε προηγουμένως (1.4.2).

1.5.6 Common.hpp

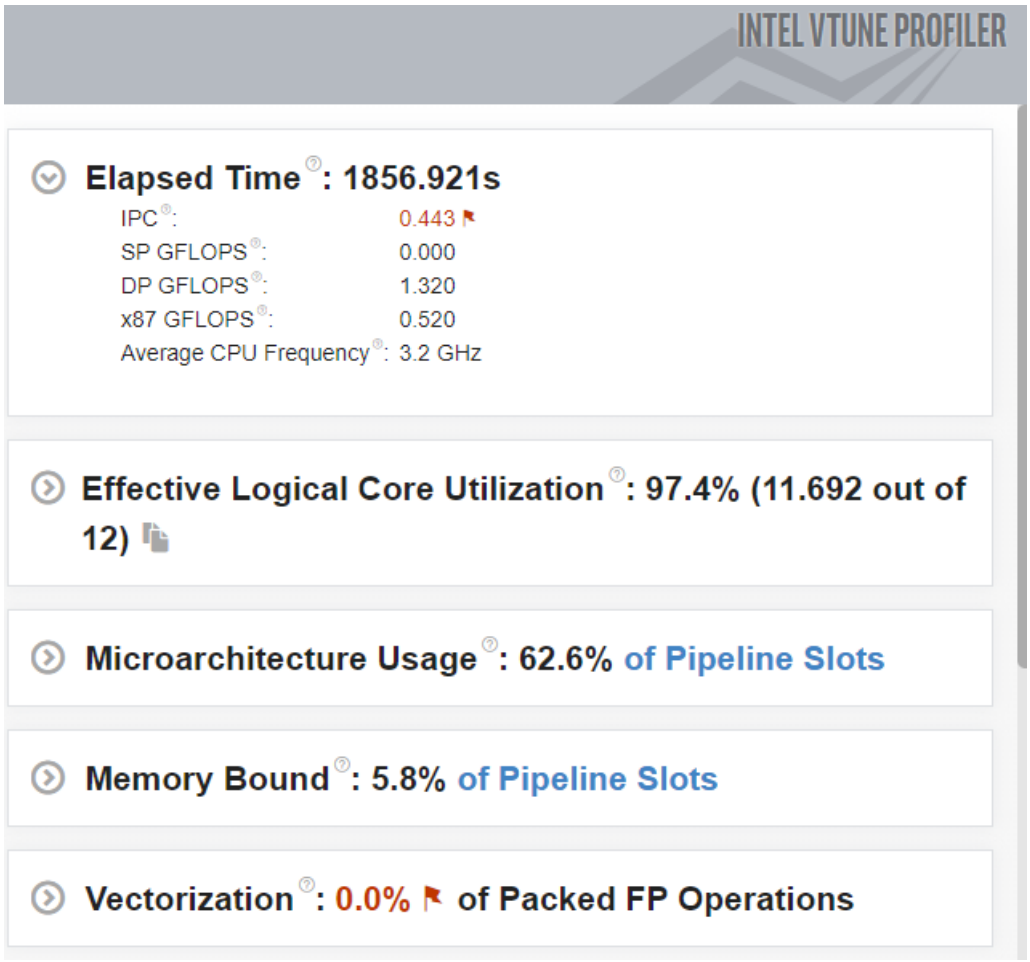
Σε αυτό το αρχείο κεφαλίδας, ορίζονται οι υπερπαράμετροι `EPOCHS` και `LEARNING_RATE`. Επίσης, ορίζεται το *filepath* του CSV αρχείου με τα δεδομένα (ή παραδείγματα) εκπαίδευσης και το *filepath* για τα δεδομένα του συνόλου επικύρωσης του μοντέλου. Επίσης, ορίζεται ο πληθάριθμος των παραδειγμάτων

[2] Ας θεωρηθεί future work.

του συνόλου εκπαίδευσης και του συνόλου επικύρωσης. Σημαντική επίσης είναι και η σταθερά που δηλώνει τον αριθμό των κλάσεων που υπάρχουν στο σύνολο δεδομένων «MNIST». Τέλος, δηλώνεται και ο αριθμός των threads που θα ζητήσει το πρόγραμμα από το σύστημα. Προτείνεται αυτός ο αριθμός να είναι ίσος με τον αριθμό των logical processors.

1.6 Αποτελέσματα

Χρησιμοποιώντας τον *Intel VTUNE Profiler*, φαίνεται πως το ποσοστό παραλληλοποίησης που πετυχαίνει η δεδομένη υλοποίηση είναι αρκετά υψηλό (σχήμα 1.1).



Σχήμα 1.1: Performance Snapshot από τον Profiler της Intel για το νευρωνικό δίκτυο

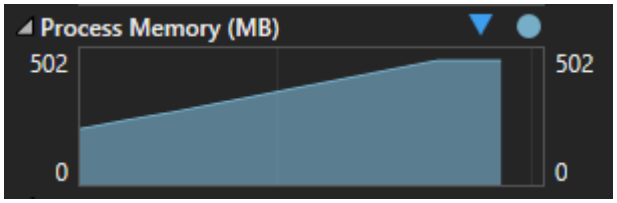
Η μέση διάρκεια εποχής εκπαίδευσης του νευρωνικού δικτύου χρησιμοποιώντας ως σύνολο εκπαίδευσης το *MNIST* είναι 19 δευτερόλεπτα, ενώ το νευρωνικό πετυχαίνει ποσοστό επικύρωσης 88.73% μετά από 100 εποχές με μόνο ένα κρυφό επίπεδο 100 νευρώνων. Συνολικά, η εκπαίδευση 100 εποχών αναμένεται να διαρ-

κέσει περίπου 2000 δευτερόλεπτα. Ενδεικτικά, το πρόγραμμα αναμένεται να έχει τα αποτελέσματα που φαίνονται στο σχήμα 1.2 για εκτέλεση 10 εποχών.

```
C:\Users\andreas\Downloads\neural-network\neural-network>Neural-Network.exe -i 784 -h 100 -o 10

Loading training dataset        |████████████████████████████████████████|  100%

Loading evaluation dataset      |████████████████████████████████████████|  100%

Neural Network Summary:         [f := Sigmoid]
----------------------------------------------------------
Layer [1]        785 neurons
Layer [2]        101 neurons
Layer [3]         10 neurons

[EPOCH    1] [LOSS 0.15241] [ACCURACY  47858 out of 60000] Work took   18 seconds
[EPOCH    2] [LOSS 0.11711] [ACCURACY  50809 out of 60000] Work took   22 seconds
[EPOCH    3] [LOSS 0.10638] [ACCURACY  51617 out of 60000] Work took   21 seconds
[EPOCH    4] [LOSS 0.09955] [ACCURACY  52205 out of 60000] Work took   23 seconds
[EPOCH    5] [LOSS 0.09848] [ACCURACY  52286 out of 60000] Work took   23 seconds
[EPOCH    6] [LOSS 0.09074] [ACCURACY  52949 out of 60000] Work took   22 seconds
[EPOCH    7] [LOSS 0.09002] [ACCURACY  53023 out of 60000] Work took   21 seconds
[EPOCH    8] [LOSS 0.08839] [ACCURACY  53111 out of 60000] Work took   20 seconds
[EPOCH    9] [LOSS 0.08486] [ACCURACY  53399 out of 60000] Work took   20 seconds
[EPOCH   10] [LOSS 0.08323] [ACCURACY  53643 out of 60000] Work took   22 seconds

[EVALUATION] [LOSS 0.09854] [ACCURACY   8693 out of 10000] Work took    2 seconds

Benchmark results: 249.52722 seconds

C:\Users\andreas\Downloads\neural-network\neural-network>
```

Σχήμα 1.2: Δείγμα εξόδου του προγράμματος για 10 εποχές εκπαίδευσης

Η απαιτούμενη μνήμη όπως φαίνεται από τον debugger του Visual Studio είναι περίπου 500 MB (σχήμα 1.3). Όπως φαίνεται και στο σχήμα 1.3, η μνήμη του προγράμματος σταθεροποιείται μετά από το φόρτωμα των δεδομένων.



Σχήμα 1.3: Process Memory του νευρωνικού δικτύου από το Visual Studio

2

Ο κώδικας

Κώδικας 2.0.1: Το αρχείο Accuracy.cpp

```cpp
#include "Neural.hpp"

/**
 * Computes model accuracy. This function pools the element with the maximum value from the
 * predictions vector and then uses the target vector to compute the accuracy of the given
 * prediction.
 *
 * @param[in, out] Y the vector with the desired values
 * @param[in] dim number of the vectors' elements
 *
 * @return 1 if the element with the maximum value from the predictions vector was accurate, else 0
 *
 * @note the given vectors must be of the same length
 *
 * @note Although passed by reference, the `Y` placeholder is not altered.
 */
int nn::accuracy(double* (&Y), int dim)
{
    double max_val = -2.0;
    int max_idx = 0;

    for (int i = 0; i < dim; i += 1)                        /// Iterate through the vector with the predictions
    {
        if (a[layers.size() - 1][i] > max_val)              /// Find the neuron with the maximum (filtered) value
        {
            max_val = a[layers.size() - 1][i];
            max_idx = i;
        }
    }

    return Y[max_idx] > 0.9 ? 1 : 0;                        /// Computes the accuracy of the given prediction based on the elite neuron found after the previous iteration
}
```

Κώδικας 2.0.2: Το αρχείο Activation.cpp

```cpp
#include "Activation.hpp"

/**
 * Efficient implementation to calculate e raise to the power x.
 *
 * @param[in] x the numerical base of the power
 * @param[in] y the numerical exponent of the power
 *
 * @return the double precision floating point result of the power operation.
 *
 * @note Returns approximate value of e^x using sum of first n terms of Taylor Series.
 *
 * @remark https://www.geeksforgeeks.org/program-to-efficiently-calculate-ex/
 */
double exp(double x)
{
    double sum = 1.0;
    int precision = 100;

    for (int i = precision - 1; i > 0; i -= 1)
    {
        sum = 1.0 + x * sum / (double)i;
    }

    return sum;
}

/**
 * Filters the input using the sigmoid activation function.
 * This implementation exploits Taylor series for fast computation.
 *
 * @param[in] x this is the double precision floating
 *              point variable to be filtered by the sigmoid
 *
 * @return the filtered value
 *
 * @note    The implementation does not utilize the std::exp routine to
 *          avoid overflow failures.
 */
double fast_sigmoid(double x)
{
    if (x > 13.0)
    {
        return 1.0;
    }
    else if (x < -13.0)
    {
        return 0.0;
    }
    else
    {
        return 1.0 / (1.0 + exp(-x));       /// Sigmoid formula
    }
}

/**
 * Safe implementation of sigmoid. This implementation gives a more precise result.
 *
 * @param[in] x this is the double precision floating
 *              point variable to be filtered by the sigmoid
 *
 * @return the filtered value
 *
 * @note    The std::pow() function is low on performance.
 */
double sigmoid(double x)
{
    if (x > 45.0)
    {
        return 1.0;
    }
    else if (x < -45.0)
    {
        return 0.0;
    }
    else
    {
        return 1.0 / (1.0 + std::pow(EXP, -x));
    }
}

/**
 * Computes the derivative of the sigmoid function.
 * This implementation uses already filtered values
 * by the sigmoid to speed up the computation process.
 *
 * @param[in] x this is the double precision floating
 *              point variable to be differentiated
 *
 * @return the derivative of x with respect to the sigmoid function
 */
double sig_derivative(double x)
{
    return (x * (1.0 - x));                 /// Sigmoid derivative formula
}

/**
 * Filters the input using the ReLU activation function.
 *
 * @param[in] x this is the double precision floating
 *              point variable to be filtered by the ReLU
 *
 * @return the filtered value
 */
double relu(double x)
{
    return (x > 0.0 ? x : 0.0);             /// ReLU formula
}

/**
 * Computes the derivative of the ReLU function.
 *
 * @param[in] x this is the double precision floating
 *              point variable to be differentiated
 *
 * @return the derivative of x with respect to the ReLU function
 */
double rel_derivative(double x)
{
    return (x < 0.0 ? 0.0 : 1.0);           /// ReLU derivative formula
}
```

Κώδικας 2.0.3: Η κεφαλίδα Activation.hpp

```cpp
/**
 * Activation.hpp
 *
 * In this header file, we define
 * all neuron activtion functions.
 * Specifically, there is an
 * implementation of the sigmoid
 * and the ReLU activation function.
 * There are also the corresponding
 * derivatives of those functions
 * to be used during neuron error
 * computation (back propagation
 * algorithm).
 */

#pragma once

#include "Common.hpp"

double exp(double x);
double sigmoid(double x);
double fast_sigmoid(double x);
double sig_derivative(double x);
double relu(double x);
double rel_derivative(double x);
```

Κώδικας 2.0.4: Η κεφαλίδα Common.hpp

```cpp
/**
 * Common.hpp
 *
 * In this header file, we define the constants
 * used throughout the project. We also
 * include all the header files necessary to
 * make the implementation work.
 */

#pragma once

#include <cmath>                            /// ceil()
#include <array>                            /// std::array
#include <cerrno>                           /// EINVAL
#include <random>                           /// std::random
#include <vector>                           /// std::vector
#include <limits>                           /// std::numeric_limits
#include <cstdio>                           /// printf()
#include <cstring>                          /// memmove()
#include <iomanip>                          /// std::setw
#include <fstream>                          /// std::ostream
#include <cassert>                          /// assert()
#include <cstdlib>                          /// system()
#include <iostream>                         /// std::cout
#include <stdexcept>                        /// std::runtime_error
#include <algorithm>                        /// std::fill_n
#include <inttypes.h>                       /// intptr_t

#include <omp.h>                            /// OpenMP Multiprocessing Programming Framework

#define array_sizeof(type) ((char *)(&type+1)-(char*)(&type))
                                           /// Macro that computes the size of an array
typedef intptr_t ssize_t;                   /// Declares `ssize_t` type that is used in `Preprocessing.h`

constexpr int EPOCHS = 100;                /// Declares the number of epochs for the model's training
constexpr int N_THREADS = 12;              /// Specifies the number of threads to request from the OS
constexpr int N_ACTIVATIONS = 2;           /// Declares the number of neuron activation functions declared in the project
constexpr int CLI_WINDOW_WIDTH = 50;       /// Defines the length of the progress bar for the project's CLI
constexpr int MNIST_CLASSES = 10;          /// Declares the number of classes found in the MNIST dataset
constexpr double LEARNING_RATE = 0.1;      /// Defines the learning rate for the neural network
constexpr double MNIST_TRAIN = 60000.0;    /// Declares the number of training examples found in the MNIST dataset
constexpr double MNIST_TEST = 10000.0;     /// Declares the number of evaluation examples found in the MNIST dataset
constexpr double EXP = 2.718282;           /// Defines the exponential constant `e`
constexpr double MNIST_MAX_VAL = 255.0;    /// Defines max value found in the input subset of theMNIST dataset
constexpr char TRAINING_DATA_FILEPATH[] = "C:/Users/andreas/Documents/workspace/source/repos/neural-network/neural-network/data/fashion-mnist_train.csv";
                                           /// Declares the filepath of the MNIST training CSV file
constexpr char EVALUATION_DATA_FILEPATH[] = "C:/Users/andreas/Documents/workspace/source/repos/neural-network/neural-network/data/fashion-mnist_test.csv";
                                           /// Declares the filepath of the MNIST evaluation CSV file
static int x = 1;                          /// Defines the starting row for the program's CLI
static int y = 0;                          /// Defines the starting column for the program's CLI
```

Κώδικας 2.0.5: Η κεφαλίδα Dataset.hpp

```cpp
/**
 * Preprocess.hpp
 *
 * In this header file, we define a
 * class that handles a dataset. The class
 * has a function that imports the dataset
 * directly from a CSV file. There is also
 * a function that prints out the input and
 * expected output of a dataset instance.
 * Finally, while parsing the CSV file, the
 * data is split into input examples and
 * the corresponding (expected) outputs.
 */

#pragma once

#include "Interface.hpp"

 /**
  * Implementation of a dataset class.
  *
  * Upon a dataset creation, the developer
  * has to call `read_csv` providing the
  * requested arguments to start using
  * the created dataset instance. The dataset
  * has got an attribute (variable) `classes`
  * which is initialized after the number of
  * classes in a dataset. For the project's
  * purposes, this attribute has been initialized
  * with 10 (ten), since there are 10 classes in the
  * MNIST fashion dataset.
  */
class dataset
{
public:
    int samples, dimensions, classes;
    double** X, ** Y;

    ssize_t getline(char** lineptr, size_t* n, FILE* stream);
    void read_csv(const char* filename, int dataset_flag, double x_max);
    int get_label(int sample);
    void print_dataset(void);

    dataset(int classes, int samples) :
        classes{ classes },
        samples{ samples }
    {
        dimensions = 0;
    }

    ~dataset()
    {
        for (int i = 0; i < samples; i += 1)
        {
            delete[] X[i];
            delete[] Y[i];
        }
        delete[] X;
        delete[] Y;
    }
};
```

Κώδικας 2.0.6: Το αρχείο Driver.cpp

```cpp
#include "Driver.hpp"

/**
 * Implements the driver for the Neural Network.
 *
 * @param[in] argc number of user arguments
 * @param[in] argv vector of user arguments
 *
 * @return 0, if the executable was terminated normally
 *
 * @note    For the driver to work properly, adjust the project settings found at the `Common.h` file.
 *          One such adjustment is to define the filepaths of the training and the evaluation subsets.
 *          Another stronlgy recommended change is the number of threads requested by the OS. This number
 *          is recommended to be equal to the number of the hosts's Logical Processors. This will very
 *          possibly optimize execution time and therefore increase performance.
 */
int main(int argc, char* argv[])
{
    int cli_rows, cli_cols, cursor_row, cursor_col;
    double start, end;
    std::vector<int> vec;

    nn fcn;                                                                         /// Declares the image of the neural network
    dataset TRAIN(MNIST_CLASSES, MNIST_TRAIN);                                      /// Declares training data subset
    dataset TEST(MNIST_CLASSES, MNIST_TEST);                                        /// Declares evaluation data subset

    parse_arguments(argc, argv, vec);                                               /// Parses user arguments
    start = omp_get_wtime();                                                        /// Initializes benchmark

    TRAIN.read_csv(TRAINING_DATA_FILEPATH, 0, MNIST_MAX_VAL);                       /// Initializes training data subset
    TEST.read_csv(EVALUATION_DATA_FILEPATH, 1, MNIST_MAX_VAL);                      /// Initializes evaluation data subset

    fcn.compile(vec, -1.0, 1.0);                                                    /// Initializes the neural network's image
    fcn.summary();                                                                  /// Prints model structure
    fcn.fit(TRAIN);                                                                 /// Trains the model
    fcn.evaluate(TEST);                                                             /// Evaluates the model
    fcn.export_weights("mnist-fcn");

    end = omp_get_wtime();                                                          /// Terminates the benchmark

    std::cout << "\n\nBenchmark results: " << end - start << " seconds\n";          /// Prints benchmark results

    return(0);
}
```

Κώδικας 2.0.7: Το αρχείο Dataset.cpp

```cpp
#include "Driver.hpp"

/**
 * Implements the driver for the Neural Network.
 *
 * @param[in] argc number of user arguments
 * @param[in] argv vector of user arguments
 *
 * @return 0, if the executable was terminated normally
 *
 * @note    For the driver to work properly, adjust the project settings found at the `Common.h` file.
 *          One such adjustment is to define the filepaths of the training and the evaluation subsets.
 *          Another stronlgy recommended change is the number of threads requested by the OS. This number
 *          is recommended to be equal to the number of the hosts's Logical Processors. This will very
 *          possibly optimize execution time and therefore increase performance.
 */
int main(int argc, char* argv[])
{
    int cli_rows, cli_cols, cursor_row, cursor_col;
    double start, end;
    std::vector<int> vec;

    nn fcn;                                                                         /// Declares the image of the neural network
    dataset TRAIN(MNIST_CLASSES, MNIST_TRAIN);                                      /// Declares training data subset
    dataset TEST(MNIST_CLASSES, MNIST_TEST);                                        /// Declares evaluation data subset

    parse_arguments(argc, argv, vec);                                               /// Parses user arguments
    start = omp_get_wtime();                                                        /// Initializes benchmark

    TRAIN.read_csv(TRAINING_DATA_FILEPATH, 0, MNIST_MAX_VAL);                       /// Initializes training data subset
    TEST.read_csv(EVALUATION_DATA_FILEPATH, 1, MNIST_MAX_VAL);                      /// Initializes evaluation data subset

    fcn.compile(vec, -1.0, 1.0);                                                    /// Initializes the neural network's image
    fcn.summary();                                                                  /// Prints model structure
    fcn.fit(TRAIN);                                                                 /// Trains the model
    fcn.evaluate(TEST);                                                             /// Evaluates the model
    fcn.export_weights("mnist-fcn");

    end = omp_get_wtime();                                                          /// Terminates the benchmark

    std::cout << "\n\nBenchmark results: " << end - start << " seconds\n";          /// Prints benchmark results

    return(0);
}
```

Κώδικας 2.0.8: Η κεφαλίδα Driver.hpp

```cpp
/**
 * Driver.hpp
 *
 * In this header file, we include all the
 * custom made header files used for the
 * model's implementation.
 */

#pragma once

#include "Parser.hpp"
#include "Neural.hpp"
#include "Interface.hpp"
```

Κώδικας 2.0.9: Το αρχείο Export.cpp

```cpp
#include "Neural.hpp"

/**
 * Exports weights of neural network instance into a CSV file.
 *
 * @param[in] filename the CSV name
 */
void nn::export_weights(std::string filename)
{
    std::ofstream export_stream;                                /// Defines an output file stream
    export_stream.open("./data/" + filename + ".csv");          /// Associates `export_stream` with a CSV file named after the `filename` variable
    for (int i = 1; i < layers.size() - 1; i += 1)              /// Loops through model's hidden layers
    {
        for (int j = 0; j < layers[i] - 1; j += 1)              /// Loops through layer's synapses
        {
            export_stream << "Neuron " << j << " Layer " << i << ",";
            for (int k = 0; k < layers[i - 1]; k += 1)
            {
                export_stream << weights[i - 1][j][k] << (j == layers[i] - 2 ? "" : ",");
            }                                                   /// Export element of that array to the `export_stream` file stream
            export_stream << std::endl;
        }
        export_stream << std::endl;
    }

    for (int j = 0; j < layers[layers.size() - 1]; j += 1)      /// Loops through model's output layers
    {
        export_stream << "Neuron " << j << " Layer " << layers.size() - 1 << ",";
        for (int k = 0; k < layers[layers.size() - 1]; k += 1)
        {
            export_stream << weights[layers.size() - 2][j][k] << (j == layers[layers.size()] - 1 ? "" : ",");
        }                                                       /// Export element of that array to the `export_stream` file stream
        export_stream << std::endl;
    }
    export_stream << std::endl;

    export_stream.close();                                      /// Closes file stream
}
```

Κώδικας 2.0.10: Το αρχείο Fit.cpp

```cpp
#include "Neural.hpp"
#include "Interface.hpp"

/**
 * Trains the given model. The model is a simple multi-
 * layer feed forward perceptron.
 *
 * @param[in] TRAIN the training dataset
 */
void nn::fit(dataset(&TRAIN))
{
    int shuffled_idx;                                                           /// Decalres dample "pointer"
    double start, end;                                                          /// Declares epoch benchmark checkpoints
    std::array<double, EPOCHS> loss;                                            /// Declares container for training loss
    std::array<int, EPOCHS> validity;                                           /// Declares container for training accuracy

    std::random_device rd;                                                      /// Initializes non-deterministic random generator
    std::mt19937 gen(rd());                                                     /// Seeds mersenne twister
    std::uniform_int_distribution<> dist(0, TRAIN.samples - 1);                 /// Distribute results between 0 and sample count exclusive
                                                                                /// Change this depending on the ammount of loaded datasets
    for (int epoch = 0; epoch < EPOCHS; epoch += 1)                             /// Trains model
    {
        loss[epoch] = 0.0;                                                      /// Initializes epoch's training loss
        validity[epoch] = 0;                                                    /// Initializes epoch's training accuracy

        start = omp_get_wtime();                                                /// Benchmarks epoch
        for (int sample = 0; sample < TRAIN.samples; sample += 1)               /// Iterates through all examples of the training dataset
        {
            shuffled_idx = dist(gen);                                           /// Selects a random example to avoid unshuffled dataset event
            zero_grad(TRAIN.X[shuffled_idx]);                                   /// Resets the neurons of the neural network
            forward();                                                          /// Feeds forward the selected input
            back_propagation(TRAIN.Y[shuffled_idx]);                            /// Computes the error for every neuron in the network
            optimize();                                                         /// Optimizes weights using pack propagation
            loss[epoch] += mse_loss(TRAIN.Y[shuffled_idx], TRAIN.classes);      /// Updates epoch's loss of the model
            validity[epoch] += accuracy(TRAIN.Y[shuffled_idx], TRAIN.classes);  /// Updates epoch's accuracy of the model
        }
        end = omp_get_wtime();                                                  /// Terminates epoch's benchmark

        loss[epoch] /= (TRAIN.samples + 0.0);                                   /// Averages epoch's loss of the model
        print_epoch_stats(epoch + 1, loss[epoch], validity[epoch], end - start);  /// Prints epoch's loss, accuracy and benchmark
    }
}

/**
 * Evaluates the given model.
 *
 * @param[in] TEST the evaluation dataset
 */

void nn::evaluate(dataset(&TEST))
{
    int validity = 0;
    double start, end, loss = 0.0;

    start = omp_get_wtime();                                                    /// Benchmarks model's evaluation
    for (int sample = 0; sample < TEST.samples; sample += 1)                    /// Iterates through all examples of the evaluation dataset
    {
        zero_grad(TEST.X[sample]);                                              /// Resets the neurons of the neural network
        forward();                                                              /// Feeds forward the evaluation sample
        loss += mse_loss(TEST.Y[sample], TEST.classes);                         /// Updates loss of the model based on the evaluation set
        validity += accuracy(TEST.Y[sample], TEST.classes);                     /// Updates accuracy of the model based on the evaluation set
    }
    end = omp_get_wtime();                                                      /// Terminates model's evaluation benchmark

    loss /= (TEST.samples + 0.0);
    print_epoch_stats(-1, loss, validity, end - start);                         /// Prints evaluation loss, accuracy, and benchmark
}
```

Κώδικας 2.0.11: Το αρχείο Forward.cpp

```cpp
#include "Neural.hpp"

/**
 * Feeds forward the given model a given input vector.
 *
 * @note    To exploit the full capabilities of the OpenMP framework, we use `collapse()` routine wherever possible.
 *          To use this routine, the given vector must be contiguous, therefore a 2D dynamic array cannot be collapsed.
 *          That's why there are temporary variables called `REGISTERS`, which are the 1D temporary image of those 2D
 *          vectors. Those registers are used during the parallel computations, and then we utilize the `memmove()`
 *          routine which has O(1) time complexity and transfers the computations back to the 2D vectors.
 */
void nn::forward(void)
{
    int dynamic_size;
    double* REGISTER;

    for (int layer = 1; layer < layers.size() - 1; layer += 1)
    {
        dynamic_size = layers[layer] - 1;                                                   /// fetches the number of neurons of the currently parsed hidden layer excluding the layer's bias

        REGISTER = (double*)calloc(dynamic_size, sizeof(double));                           /// Allocates temporary memory

        if (REGISTER == NULL)
        {
            perror("calloc() failed");                                                      /// Masks memory allocation fault
        }

#pragma omp parallel for collapse(2) reduction(+ : REGISTER[0: dynamic_size]) num_threads(N_THREADS) schedule(runtime)
        for (int neuron = 0; neuron < layers[layer] - 1; neuron += 1)                       /// Iterates through the hidden layer's neurons
        {
            for (int synapse = 0; synapse < layers[layer - 1]; synapse += 1)                /// Iterates throught the previous layer
            {
                REGISTER[neuron] += weights[layer - 1][neuron][synapse] * a[layer - 1][synapse];    /// Implements forward propagation for all hidden layers
            }
        }

        memmove(z[layer], REGISTER, dynamic_size * sizeof(double));                         /// Moves the results to the main data container

#pragma omp parallel for simd num_threads(N_THREADS) schedule(runtime)
        for (int neuron = 0; neuron < layers[layer] - 1; neuron += 1)
        {
            a[layer][neuron] = sigmoid(z[layer][neuron]);                                   /// Applies model's ativation function to computed results
        }

        free(REGISTER);                                                                     /// Deallocates the temporary container off the memory
    }

    dynamic_size = layers[layers.size() - 1];                                               /// Holds the number of neurons of the output layer

    REGISTER = (double*)calloc(dynamic_size, sizeof(double));                               /// Allocates temporary memory

    if (REGISTER == NULL)
    {
        perror("calloc() failed");                                                          /// Masks memory allocation fault
    }

#pragma omp parallel for collapse(2) reduction(+ : REGISTER[0: dynamic_size]) num_threads(N_THREADS) schedule(runtime)
    for (int neuron = 0; neuron < layers[layers.size() - 1]; neuron += 1)
    {
        for (int synapse = 0; synapse < layers[layers.size() - 2]; synapse += 1)
        {
            REGISTER[neuron] += weights[layers.size() - 2][neuron][synapse] * a[layers.size() - 2][synapse];    /// Implements forward propagation for the output layer
        }
    }

    memmove(z[layers.size() - 1], REGISTER, dynamic_size * sizeof(double));

#pragma omp parallel for simd num_threads(N_THREADS) schedule(runtime)
    for (int neuron = 0; neuron < layers[layers.size() - 1]; neuron += 1)
    {
        a[layers.size() - 1][neuron] = sigmoid(z[layers.size() - 1][neuron]);               /// Applies model's ativation function to computed results
    }

    free(REGISTER);                                                                         /// Deallocates the temporary container off the memory
}
```

Κώδικας 2.0.12: Η κεφαλίδα Interface.hpp

```cpp
/**
 * Interface.hpp
 *
 * In this header file, we implement a
 * basic User Interface using the Command
 * Line Window. The defined functions are
 * used to print information about the data
 * processed by the neural network, about
 * the dataset, about the neural network's
 * progress and help the user understand
 * how to use the project. there is also a
 * cross platform implementation of a progress
 * bar.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */

#pragma once

#include "Common.hpp"

#ifdef _WIN32
#define _CRT_SECURE_NO_WARNINGS 1
#include <windows.h>
#else
#include <sys/ioctl.h>
#include <termios.h>
#include <unistd.h>
#endif

void setupConsole(void);
void restoreConsole(void);
void getWindowSize(int(&rows), int(&columns));
void getCursorPosition(int* row, int* col);
void usage(char* filename);
void print_epoch_stats(int epoch, double epoch_loss, int epoch_accuracy, double benchmark);

void SetConsoleWindowSize(int x, int y);
void moveUp(int positions);
void moveDown(int positions);
void scrollUp(int positions);
void scrollDown(int positions);
void clearScreen(void);
void gotoxy(int x, int y);
void hideCursor(void);
void showCursor(void);
void saveCursorPosition(void);
void restoreCursorPosition(void);


/**
 * Implements a progress bar in CLI.
 *
 * Instances of this class are progress bars
 * that inform the user of a large task's
 * progress. There is also a short description
 * attached to each progress bar. The description
 * is recommended not to exceed the length of 35
 * characters.
 */
class progress_bar
{
public:
    std::string message;
    char* bar;
    char progress_token;
    int progress;
    int length;

    void indicate_progress(double checkpoint, int x, int y);

    progress_bar(std::string message, char progress_token, int length) :
        message{ message },
        progress_token{ progress_token },
        length{ length }
    {
        bar = new char[length + 1];
        for (int i = 0; i < length; i += 1)
        {
            bar[i] = ' ';
        }
        bar[length] = '\0';
        progress = 0;
        std::cout << "\n";
    }

    ~progress_bar()
    {
        delete[] bar;
    }
};
```

Κώδικας 2.0.13: Το αρχείο Loss.cpp

```cpp
#include "Neural.hpp"

/**
 * Computes the model's MSE loss.
 *
 * @param[in] Y the expected output. This is the ground truth given the same input
 * @param[in] dim the size of the output layer and therefore the size of the `Y` placeholder
 *
 * @return the total loss based on the model's predictions on a given sample and the corresponding (expected) output
 *
 * @note The given vectors must be of the same dimensions.
 *
 * @note Although passed by reference, the `Y` placeholder is not altered.
 */

double nn::mse_loss(double* (&Y), int dim)
{
    double l = 0.0;                                                     /// Initializes loss variable (accumulator)
#pragma omp parallel for num_threads(N_THREADS) reduction(+ : l) schedule(runtime)
    for (int i = 0; i < dim; i += 1)
    {
        l += (1.0 / 2.0) * (Y[i] - a[layers.size() - 1][i]) * (Y[i] - a[layers.size() - 1][i]);
    }

    return l;
}
```

Κώδικας 2.0.14: Το αρχείο Interface.cpp

```cpp
#include "Interface.hpp"

/**
 * Includes graphic libraries depending on the host's OS.
 * Sets up CLI environment and defines functions and modules
 * for the CLI.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
#ifdef _WIN32

#ifndef ENABLE_VIRTUAL_TERMINAL_PROCESSING
#define ENABLE_VIRTUAL_TERMINAL_PROCESSING 0x0004
#endif

static HANDLE stdoutHandle, stdinHandle;
static DWORD outModeInit, inModeInit;

/**
 * Sets up console (CLI) in Windows OS.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void setupConsole(void)
{
    DWORD outMode = 0, inMode = 0;
    stdoutHandle = GetStdHandle(STD_OUTPUT_HANDLE);
    stdinHandle = GetStdHandle(STD_INPUT_HANDLE);

    if (stdoutHandle == INVALID_HANDLE_VALUE || stdinHandle == INVALID_HANDLE_VALUE)
    {
        exit(GetLastError());
    }

    if (!GetConsoleMode(stdoutHandle, &outMode) || !GetConsoleMode(stdinHandle, &inMode))
    {
        exit(GetLastError());
    }

    outModeInit = outMode;
    inModeInit = inMode;
    outMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;                      /// Enable ANSI escape codes
    inMode &= ~(ENABLE_ECHO_INPUT | ENABLE_LINE_INPUT);                 /// Set stdin as no echo and unbuffered

    if (!SetConsoleMode(stdoutHandle, outMode) || !SetConsoleMode(stdinHandle, inMode))
    {
        exit(GetLastError());
    }
}

/**
 * Resets console (CLI) in Windows OS. During
 * the execution there might be some changes
 * in the CLI which need to be unmade upon
 * program's termination.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void restoreConsole(void)
{
    printf("\x1b[0m");                                                  /// Reset colors

    if (!SetConsoleMode(stdoutHandle, outModeInit) || !SetConsoleMode(stdinHandle, inModeInit))
    {                                                                   /// Reset console mode
        exit(GetLastError());
    }
}

/**
 * Evaluates CLI window size in Windows OS.
 *
 * @param[in, out] rows the number of rows of interface found
 * @param[in, out] columns the number of columns of interface found
 *
 * @remark https://stackoverflow.com/questions/23369503/get-size-of-terminal-window-rows-columns
 */
void getWindowSize(int (&rows), int (&columns))
{
    CONSOLE_SCREEN_BUFFER_INFO csbi;

    GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
    columns = csbi.srWindow.Right - csbi.srWindow.Left + 1;
    rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
}

/**
 * Sets CLI window size.
 *
 * @param[in] x the number of rows for the CLI
 * @param[in] y the number of columns for the CLI
 *
 * @remark https://www.cplusplus.com/forum/windows/121444/#msg661553
 */
void SetConsoleWindowSize(int x, int y)
{
    HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);

    if (h == INVALID_HANDLE_VALUE)
        throw std::runtime_error("Unable to get stdout handle.");

    // If either dimension is greater than the largest console window we can have,
    // there is no point in attempting the change.
    {
        COORD largestSize = GetLargestConsoleWindowSize(h);
        if (x > largestSize.X)
            throw std::invalid_argument("The x dimension is too large.");
        if (y > largestSize.Y)
            throw std::invalid_argument("The y dimension is too large.");
    }


    CONSOLE_SCREEN_BUFFER_INFO bufferInfo;
    if (!GetConsoleScreenBufferInfo(h, &bufferInfo))
        throw std::runtime_error("Unable to retrieve screen buffer info.");

    SMALL_RECT& winInfo = bufferInfo.srWindow;
    COORD windowSize = { winInfo.Right - winInfo.Left + 1, winInfo.Bottom - winInfo.Top + 1 };

    if (windowSize.X > x || windowSize.Y > y)
    {
        // window size needs to be adjusted before the buffer size can be reduced.
        SMALL_RECT info =
        {
            0,
            0,
            x < windowSize.X ? x - 1 : windowSize.X - 1,
            y < windowSize.Y ? y - 1 : windowSize.Y - 1
        };

        if (!SetConsoleWindowInfo(h, TRUE, &info))
            throw std::runtime_error("Unable to resize window before resizing buffer.");
    }

    COORD size = { x, y };
    if (!SetConsoleScreenBufferSize(h, size))
        throw std::runtime_error("Unable to resize screen buffer.");


    SMALL_RECT info = { 0, 0, x - 1, y - 1 };
    if (!SetConsoleWindowInfo(h, TRUE, &info))
        throw std::runtime_error("Unable to resize window after resizing buffer.");
}
#else

static struct termios orig_term;
static struct termios new_term;

/**
 * Sets up console (CLI) in Unix OS.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void setupConsole(void)
{
    tcgetattr(STDIN_FILENO, &orig_term);
    new_term = orig_term;

    new_term.c_lflag &= ~(ICANON | ECHO);

    tcsetattr(STDIN_FILENO, TCSANOW, &new_term);
}

/**
 * Resets console (CLI) in Unix OS. During
 * the execution there might be some changes
 * in the CLI which need to be unmade upon
 * program's termination.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void restoreConsole(void)
{
    printf("\x1b[0m");                                                  /// Reset colors
    tcsetattr(STDIN_FILENO, TCSANOW, &orig_term);                       /// Reset console mode
}

/**
 * Evaluates CLI window size in Unix OS.
 *
 * @param[in, out] rows the number of rows of interface found
 * @param[in, out] columns the number of columns of interface found
 *
 * @remark https://stackoverflow.com/questions/23369503/get-size-of-terminal-window-rows-columns
 */
void getWindowSize(int(&rows), int(&columns))
{
    struct winsize w;
    ioctl(STDOUT_FILENO, TIOCGWINSZ, &w);
    rows = w.ws_row;
    columns = w.ws_col;
}


/**
 * Sets CLI window size.
 *
 * @param[in] x the number of rows for the CLI
 * @param[in] y the number of columns for the CLI
 *
 * @remark https://apple.stackexchange.com/questions/33736/can-a-terminal-window-be-resized-with-a-terminal-command
 */
void SetConsoleWindowSize(int x, int y)
{
    printf("\x1b[8;%d;%dt", x, y);
}
#endif

/**
 * Fetches cursor position with respect to the CLI.
 *
 * @param[in, out] row the row of the cursor in the CLI
 * @param[in, out] col the column of the cursor in the CLI
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void getCursorPosition(int* row, int* col)
{
    printf("\x1b[6n");
    char buff[128];
    int indx = 0;
    for (;;)
    {
        int cc = getchar();
        buff[indx] = (char)cc;
        indx++;
        if (cc == 'R')
        {
            buff[indx + 1] = '\0';
            break;
        }
    }
    sscanf(buff, "\x1b[%d;%dR", row, col);
    fseek(stdin, 0, SEEK_END);
}

/**
 * Moves CLI cursor Up.
 *
 * @param[in] positions number of rows to move cursor up
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void moveUp(int positions)
{
    printf("\x1b[%dA", positions);
}

/**
 * Moves CLI cursor Down.
 *
 * @param[in] positions number of rows to move cursor down
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void moveDown(int positions)
{
    printf("\x1b[%dB", positions);
}

/**
 * Scrolls CLI window Up.
 *
 * @param[in] positions number of rows to scroll up
 *
 * @remark https://en.wikipedia.org/wiki/ANSI_escape_code
 */
void scrollUp(int positions)
{
    printf("\x1b[%dS", positions);
}

/**
 * Scrolls CLI window Down.
 *
 * @param[in] positions number of rows to scroll down
 *
 * @remark https://en.wikipedia.org/wiki/ANSI_escape_code
 */
void scrollDown(int positions)
{
    printf("\x1b[%dT", positions);
}

/**
 * Clears CLI screen.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void clearScreen(void)
{
    printf("\x1b[2J\x1b[H");
}

/**
 * Places CLI cursor at defined position in the CLI.
 *
 * @param[in] x the row to place the cursor
 * @param[in] y the column to place the cursor
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 *
 * @remark https://stackoverflow.com/questions/10401724/move-text-cursor-to-particular-screen-coordinate
 */
void gotoxy(int x, int y)
{
    printf("\x1b[%d;%df", x, y);
}

/**
 * Hides the cursor from the CLI.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void hideCursor(void)
{
    printf("\x1b[?25l");
}

/**
 * Makes CLI cursor visible.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void showCursor(void)
{
    printf("\x1b[?25h");
}

/**
 * Saves cursor's position in the CLI.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void saveCursorPosition(void)
{
    printf("\x1b%d", 7);
}

/**
 * Restores cursor's position in the CLI.
 *
 * @remark https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples
 */
void restoreCursorPosition(void)
{
    printf("\x1b%d", 8);
}

/**
 * Indicates progress of an operation.
 *
 * @param[in] percentage completion rate of the monitored task
 * @param[in] x the row of the CLI to print the progess bar
 * @param[in] y the column of the CLI to print the progess bar
 */
void progress_bar::indicate_progress(double checkpoint, int x, int y)
{
    std::cout << "\r" << message << "\t|";
    int ratio = (int)std::ceil(checkpoint * length);
    int completion_percentage = (int)std::ceil(checkpoint * 100);
    if (ratio > length)
    {
        ratio -= 1;
    }
    if (completion_percentage > 100)
    {
        completion_percentage -= 1;
    }
    for (int i = 0; i < ratio; i += 1)
    {
        bar[i] = progress_token;
    }
    std::cout << bar << "| " << std::setw(4) << completion_percentage << "%";
}


/**
 * Prints epoch stats. More specifically, it prints the epoch's number
 * along with the model's acuracy and loss. It also prints the epoch's benchmark.
 *
 * @param[in] epoch the epoch's number
 * @param[in] epoch_loss the model's loss during a certain epoch of training or evaluation
 * @param[in] epoch_accuracy the model's accuracy during a certain epoch of training or evaluation
 * @param[in] benchmark the epoch's benchmark
 * @param[in] des_x the row of the CLI to print the epoch stats message
 * @param[in] des_y the column of the CLI to print the epoch stats message
 */
void print_epoch_stats(int epoch, double epoch_loss, int epoch_accuracy, double benchmark)
{
    if (epoch == -1)
    {
        std::cout << "\n\n[EVALUATION] [LOSS " << std::fixed << std::setprecision(5) << epoch_loss << "] [ACCURACY " << std::setw(6) << epoch_accuracy << " out of " << (int)MNIST_TEST << "] Work took " << std::setw(4) << (int)benchmark << " seconds";
    }
    else
    {
        std::cout << "\n[EPOCH " << std::setw(4) << epoch << "] [LOSS " << std::fixed << std::setprecision(5) << epoch_loss << "] [ACCURACY " << std::setw(6) << epoch_accuracy << " out of " << (int)MNIST_TRAIN << "] Work took " << std::setw(4) << (int)benchmark << " seconds";
    }
}

/**
 * Prints information regarding the usage and the available options of the project.
 *
 * @param[in] filename the filepath of the executable
 *
 * @note Upon this function's execution, the program is terminated.
 */
void usage(char* filename)
{
    std::cout << "Usage of " << filename << ":\n";
    std::cout << "\t:option \'-i\': integer \t - \t The size of the input layer for the neural network.\n";
    std::cout << "\t:option \'-h\': integer \t - \t The size of a hidden layer for the neural network.\n\t\t\t\t\t There can be multiple hidden layers. For every hidden layer, use this option.\n";
    std::cout << "\t:option \'-o\': integer \t - \t The size of the output layer for the neural network.\n";
    exit(8);
}
```

Κώδικας 2.0.15: Η κεφαλίδα Neural.hpp

```cpp
/**
 * Neural.hpp
 *
 * In this header file, we define
 * a template for the custom neural
 * network.
 */

#pragma once

#include "Common.hpp"
#include "Dataset.hpp"
#include "Activation.hpp"

/**
 * Implements a Multi Layer Perceptron model.
 *
 * The neural network works with sigmoid
 * activation function and Mean Squared
 * Error loss function. However, there
 * is potential for support of various
 * activation and loss functions. The
 * performance of the neural network
 * class has been optimized with OpenMP
 * framework.
 */
class nn
{
public:
    double** z, ** a, ** delta, *** weights;

    std::vector<int> layers;

    void set_layers(const std::vector<int>& l);
    void set_z(const std::vector<int>& l);
    void set_a(const std::vector<int>& l);
    void set_delta(const std::vector<int>& l);
    void set_weights(const std::vector<int>& l, const double min, const double max);
    void compile(const std::vector<int>& l, const double min, const double max);
    void zero_grad(double* (&X));
    void forward(void);
    void back_propagation(double* (&Y));
    void optimize(void);
    int get_label(double* (&y_pred));
    int predict(double* (&X));
    double mse_loss(double* (&Y), int dim);
    int accuracy(double* (&Y), int dim);
    void fit(dataset(&TRAIN));
    void evaluate(dataset(&TEST));
    void export_weights(std::string filename);
    void summary(void);

    nn()
    {

    }

    ~nn()
    {
        for (int i = 0; i < layers.size(); i += 1)
        {
            delete[] z[i];
            delete[] a[i];
        }
        delete[] z;
        delete[] a;
        for (int i = 0; i < layers.size() - 1; i += 1)
        {
            delete[] delta[i];
        }
        delete[] delta;
        for (int i = 1; i < layers.size(); i += 1)
        {
            for (int j = 0; j < layers[i] - 1; j += 1)
            {
                delete[] weights[i - 1][j];
            }
            delete[] weights[i - 1];
        }
        delete[] weights;

        layers.clear();
        layers.shrink_to_fit();
    }
};
```

Κώδικας 2.0.16: Το αρχείο Optimize.cpp

```cpp
#include "Neural.hpp"

/**
 * Computes each neuron's error of a given neural network.

 * @param[in, out] Y the expected output of the model for a given input
 *
 * @note Although passed by reference, the `Y` placeholder is not altered.
 *
 * @note    Using the `REGISTER` variable, we can take advantage of the `reduction()` routine.
 *          The `delta` container is not contiguous and therefore cannot be 'reduced'.
 *
 * @note    Although there was no need for the purposes of the project to compute the error of more
 *          than 1 (one) hidden layers, there is a loop that does exactly that, for completeness.
 */
void nn::back_propagation(double* (&Y))
{
    int dynamic_size;
    double* REGISTER;                                                                                           /// Defines a temporary container that enables further optimizations

#pragma omp parallel for simd num_threads(N_THREADS) schedule(runtime)
    for (int neuron = 0; neuron < layers[layers.size() - 1]; neuron += 1)
    {
        delta[layers.size() - 2][neuron] = (a[layers.size() - 1][neuron] - Y[neuron]) * sig_derivative(a[layers.size() - 1][neuron]);    /// Computes the error of the neurons in the last layer
    }

    dynamic_size = layers[layers.size() - 2];                                                                  /// Declares ammount of memory to store `delta` corresponding to the neurons of the last *hidden* layer

    REGISTER = (double*)calloc(dynamic_size, sizeof(double));                                                  /// Allocates the ammount of memory computed above

    if (REGISTER == NULL)
    {
        perror("calloc() failed");                                                                             /// Masks memory allocation fault
    }

#pragma omp parallel for collapse(2) reduction(+ : REGISTER[0: dynamic_size]) num_threads(N_THREADS) schedule(runtime)
    for (int synapse = 0; synapse < layers[layers.size() - 2]; synapse += 1)
    {
        for (int neuron = 0; neuron < layers[layers.size() - 1]; neuron += 1)
        {
            REGISTER[synapse] += weights[layers.size() - 2][neuron][synapse] * delta[layers.size() - 2][neuron];   /// Computes the first factor of the error of neurons for the last *hidden* layer
        }
    }

    memmove(delta[layers.size() - 3], REGISTER, dynamic_size * sizeof(double));                                /// Moves the computed results from the temporary container to `delta`

#pragma omp parallel for simd num_threads(N_THREADS) schedule(runtime)
    for (int synapse = 0; synapse < layers[layers.size() - 2]; synapse += 1)
    {
        delta[layers.size() - 3][synapse] = delta[layers.size() - 3][synapse] * sig_derivative(a[layers.size() - 2][synapse]);   /// Computes the total neuron error for each neuron in the last *hidden* layer
    }

    free(REGISTER);                                                                                            /// Deallocates memory space requested for the `REGISTER` container

    for (int layer = 2; layer < layers.size() - 1; layer += 1)                                                 /// Computes the error for neurons in the remaining hidden layers using the same method
    {
        dynamic_size = layers[layers.size() - layer - 1];

        REGISTER = (double*)calloc(dynamic_size, sizeof(double));                                              /// Allocates the ammount of memory computed above

        if (REGISTER == NULL)
        {
            perror("calloc() failed");                                                                         /// Masks memory allocation fault
        }

#pragma omp parallel for collapse(2) reduction(+ : REGISTER[0: dynamic_size]) num_threads(N_THREADS) schedule(runtime)
        for (int synapse = 0; synapse < layers[layers.size() - layer - 1]; synapse += 1)
        {
            for (int neuron = 0; neuron < layers[layers.size() - layer] - 1; neuron += 1)                      /// There is no synapse between the bias at layer `l` and any neuron at layer `l - 1`
            {
                REGISTER[synapse] += weights[layers.size() - layer - 1][neuron][synapse] * delta[layers.size() - layer - 1][neuron];
            }
        }

        memmove(delta[layers.size() - layer - 2], REGISTER, dynamic_size * sizeof(double));                    /// Moves the computed results from the temporary container to `delta`

#pragma omp parallel for simd num_threads(N_THREADS) schedule(runtime)
        for (int synapse = 0; synapse < layers[layers.size() - layer - 1]; synapse += 1)
        {
            delta[layers.size() - layer - 2][synapse] = delta[layers.size() - layer - 2][synapse] * sig_derivative(a[layers.size() - layer - 1][synapse]);
        }

        free(REGISTER);                                                                                        /// Deallocates memory space requested for the `REGISTER` container
    }
}

/**
 * Optimizes weights by subtracting the precomputed error corresponding to each neuron pair (synapse).
 */
void nn::optimize(void)
{
#pragma omp parallel for collapse(2) num_threads(N_THREADS) schedule(runtime)
    for (int neuron = 0; neuron < layers[layers.size() - 1]; neuron += 1)                                      /// Loops through all neurons in the last layer
    {
        for (int synapse = 0; synapse < layers[layers.size() - 2]; synapse += 1)                               /// Loops through all neurons in the last *hidden* layer
        {
            weights[layers.size() - 2][neuron][synapse] -= LEARNING_RATE * delta[layers.size() - 2][neuron] * a[layers.size() - 2][synapse];      /// Optimizes weights between those synapses
        }
    }

    for (int layer = 2; layer < layers.size(); layer += 1)                                                     /// Loops through all the other layers
    {
#pragma omp parallel for collapse(2) num_threads(N_THREADS) schedule(runtime)
        for (int neuron = 0; neuron < layers[layers.size() - layer] - 1; neuron += 1)
        {
            for (int synapse = 0; synapse < layers[layers.size() - layer - 1]; synapse += 1)                   /// Uses the same method to optimize the rest of the model's synapses
            {
                weights[layers.size() - layer - 1][neuron][synapse] -= LEARNING_RATE * delta[layers.size() - layer - 1][neuron] * a[layers.size() - layer - 1][synapse];
            }
        }
    }
}
```

Κώδικας 2.0.17: Το αρχείο Parser.cpp

```cpp
#include "Parser.hpp"

/**
 * Converts a string argument to integer.
 *
 * @param[in] argv the string to convert
 *
 * @return the integer that corresponds to that string
 *
 * @note    Visual studio has marked the `sscanf()` as deprecated routine.
 *          However, this is just for Windows OS.
 */
int parse_integer(char* argv)
{
    int intvar;

    if (sscanf(argv, "%d", &intvar) != 1)
    {
        fprintf(stderr, "error - not an integer");
    }

    return intvar;
}

/**
 * Parses all user arguments and initializes all necessary project attributes, such as the model's hyperparameters.
 *
 * @param[in] argc the number of user arguments
 * @param[in] argv the vector of the user arguments
 * @param[in, out] vec the container to be given the neural network's structure
 */
void parse_arguments(int argc, char* argv[], std::vector<int>& vec)
{
    char* filename = argv[0];

    while ((argc > 1) && (argv[1][0] == '-'))                          /// Loops through all arguments
    {
        switch (argv[1][1])                                             /// Stops when there are no more arguments
        {
        case 'i':                                                       /// '-i' option: This is used to give an input size for the first layer of the model
            vec.push_back(parse_integer(&argv[2][0]) + 1);
            break;
        case 'h':                                                       /// '-h' option: This is used to give the size of a hidden layer of the model
            vec.push_back(parse_integer(&argv[2][0]) + 1);              /// There can be more than one hidden layers, and all have to be initialized using the '-h' option
            break;
        case 'o':                                                       /// '-o' option: This is used to give an output size for the last layer of the model
            vec.push_back(parse_integer(&argv[2][0]));
            break;
        default:
            usage(filename);                                            /// If given option is invalid, the program prints the usage and terminates execution
        }
        argv += 2;
        argc -= 2;
    }
}
```

Κώδικας 2.0.18: Η κεφαλίδα Parser.hpp

```cpp
/**
 * Parser.hpp
 *
 * In this header file, we define all
 * the functions that help parse the user's
 * arguments. Those arguments include the
 * sizes of the different layers to define
 * a neural network. The user can also choose
 * to review the programs usage.
 */

#pragma once

#include "Interface.hpp"

int parse_integer(char* argv);
void parse_arguments(int argc, char* argv[], std::vector<int>& vec);
```

Κώδικας 2.0.19: Το αρχείο Utilities.cpp

```cpp
#include "Neural.hpp"

/**
 * This function pools the element with the maximum value from the
 * predictions vector.
 *
 * @param[in, out] y_pred the predictions vector
 *
 * @return the index of the element with the maximum value
 *
 * @note Although passed by reference, `y_pred` is not altered.
 */
int nn::get_label(double* (&y_pred))
{
    int label;
    double max_val = -2.0;

    for (int i = 0; i < layers[layers.size() - 1]; i += 1)
    {
        if (y_pred[i] > max_val)
        {
            max_val = y_pred[i];
            label = i;
        }
    }

    return label;
}

/**
 * Uses model to make predictions using custom inputs.
 *
 * @param[in, out] X the input to be given to the model
 *
 * @return the predicted class with respect to the given input
 *
 * @note Although passed by reference, the `X` placeholder is not altered.
 */
int nn::predict(double* (&X))
{
    zero_grad(X);
    forward();
    return get_label(a[layers.size() - 1]);
}

/**
 * Initializes model structure.
 *
 * @param[in] l the vector containing the model's structure
 */
void nn::set_layers(const std::vector<int>& l)
{
    for (auto& elem : l)
    {
        layers.push_back(elem);
    }
}

/**
 * Allocates memory space for the dynamic matrix that contains the neurons' unfiltered value.
 *
 * @param[in, out] l the neural network layer structure vector
 *
 * @note    The `z` container for each neuron `i` in layer a `u` holds the sum given by the
 *          formula:
 *          \sum_j^L{synapse_{i, j} * value_{j}}, where synapse is the numerical weight
 *          of the synapse between neuron `i`, `j` and the value of `j` is the filtered
 *          output of that neuron and `L` is the number of neurons found in layer `u - 1`.
 *          The filter refers to the activation function used to 'activate' the neurons
 *          in the neural network.
 */
void nn::set_z(const std::vector<int>& l)
{
    z = new double* [l.size()];
    for (int i = 0; i < l.size(); i += 1)
    {
        z[i] = new double[l[i]];
    }
}

/**
 * Allocates memory space for the dynamic matrix that contains the neurons' filtered value.
 *
 * @param[in, out] l the neural network layer structure vector
 *
 * @note    The `a` container for each neuron `i` in layer `l` holds the sum given by the
 *          formula:
 *          f{(z_i)}, \forall i \in `l`, where f is the chosen activation function for every
 *          neuron i nthe model.
 */
void nn::set_a(const std::vector<int>& l)
{
    a = new double* [l.size()];
    for (int i = 0; i < l.size(); i += 1)
    {
        a[i] = new double[l[i]];
    }
}

/**
 * Allocates memory space for the dynamic matrix that contains the neurons' error.
 *
 * @param[in, out] l the neural network layer structure vector
 */
void nn::set_delta(const std::vector<int>& l)
{
    delta = new double* [l.size() - 1];
    for (int i = 1; i < l.size(); i += 1)
    {
        delta[i - 1] = new double[l[i]];
    }
}

/**
 * Sets model's weights of synapses.
 *
 * @param[in] l the neural network layer structure vector
 * @param[in] min the minimum weight of a synapse
 * @param[in] max the maximum weight of a synapse
 */
void nn::set_weights(const std::vector<int>& l, const double min, const double max)
{
    std::random_device rd;                                          /// Initializes non-deterministic random generator
    std::mt19937 gen(rd());                                         /// Seeds mersenne twister
    std::uniform_real_distribution<> dist(min, max);                /// Distribute results between `min` and `max` inclusive

    weights = new double** [l.size() - 1];                          /// Allocates memory for the weights container
    for (int i = 1; i < l.size() - 1; i += 1)
    {
        weights[i - 1] = new double* [l[i] - 1];                    /// Allocates memory for the weigts of a layer in a neural network
        for (int j = 0; j < l[i] - 1; j += 1)
        {
            weights[i - 1][j] = new double[l[i - 1]];               /// Allocates memory for the weigths of each neuron in a layer
            for (int k = 0; k < l[i - 1]; k += 1)
            {
                weights[i - 1][j][k] = dist(gen);                   /// Uses random generator to initialize synapse
            }
        }
    }
    weights[l.size() - 2] = new double* [l[l.size() - 1]];          /// Initializes weights in the output layer
    for (int j = 0; j < l[l.size() - 1]; j += 1)                    /// There is no bias in the output layer
    {
        weights[l.size() - 2][j] = new double[l[l.size() - 2]];     /// Allocates memory for the weigths of each neuron in the output layer
        for (int k = 0; k < l[l.size() - 2]; k += 1)
        {
            weights[l.size() - 2][j][k] = dist(gen);                /// Uses random generator to initialize synapse
        }
    }
}

void nn::compile(const std::vector<int>& l, const double min, const double max)
{
    set_layers(l);
    set_z(l);
    set_a(l);
    set_delta(l);
    set_weights(l, min, max);
}

/**
 * Clears the past values computed created during feed forward and/or back propagation processes.
 *
 * @param[in, out] X a vector that has been initialized with a random sample from the training data subset
 *
 * @note This function is called upon both training and evaluation. That's why it also clears past values of the `delta` container.
 */
void nn::zero_grad(double* (&X))
{
    for (int j = 0; j < layers[0] - 1; j += 1)                      /// Prepare - initialize input layer
    {
        z[0][j] = X[j];
        a[0][j] = X[j];
    }
    z[0][layers[0] - 1] = 1.0;
    a[0][layers[0] - 1] = 1.0;

    for (int i = 1; i < layers.size() - 1; i += 1)                  /// Prepare - initialize hidden layers
    {
        for (int j = 0; j < layers[i] - 1; j += 1)
        {
            z[i][j] = 0.0;
            a[i][j] = 0.0;
            delta[i - 1][j] = 0.0;
        }
        z[i][layers[i] - 1] = 1.0;
        a[i][layers[i] - 1] = 1.0;
        delta[i - 1][layers[i] - 1] = 0.0;
    }

    for (int j = 0; j < layers[layers.size() - 1]; j += 1)          /// Prepare - initialize output layer
    {
        z[layers.size() - 1][j] = 0.0;
        a[layers.size() - 1][j] = 0.0;
        delta[layers.size() - 2][j] = 0.0;
    }
}

/**
 * Prints Neural Network layer structure.
 */
void nn::summary(void)
{
    int l = 0;

    std::string s(CLI_WINDOW_WIDTH + 10, '-');

    std::cout << "\n\nNeural Network Summary:\t\t[f := Sigmoid]\n" << s << std::endl;

    for (auto& elem : layers)
    {
        std::cout << "Layer [" << ++l << "]\t" << std::setw(4) << elem << " neurons\n";
    }
}
```