Χούκλης Αθανάσιος // ΑΕΜ: 10396
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

1Η ΥΠΟΧΡΕΩΤΙΚΗ ΕΡΓΑΣΙΑ
ΣΤΟ ΜΑΘΗΜΑ «ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ – ΒΑΘΙΑ ΜΑΘΗΣΗ»

Ενδιάμεση Εργασία

Για την πραγματοποίηση της εργασίας χρησιμοποιήθηκε η βιβλιοθήκη PyTorch, μία από τις προτεινόμενες deep learning αρχιτεκτονικές.

Αποτελεί γενικά δυνατό και εύχρηστο εργαλείο για μηχανική μάθηση με ευρεία χρήση στην έρευνα και την βιομηχανία, επιλέχθηκε όμως συγκεκριμένα διότι είναι ένα από τα λίγα εργαλεία που βρέθηκε να παρέχει native υποστήριξη για το AMD equivalent του CUDA, το ROCm, χρησιμοποιώντας βέβαια αναγκαστικά λειτουργικό Linux. Έτσι το πρόγραμμα ήταν δυνατό να τρέξει τοπικά σε σύστημα που έχει AMD GPU και παρόλο που για την συγκεκριμένη εργασία δεν παρατηρήθηκε μεγάλη διαφορά σε χρόνο εκτέλεσης, η δυνατότητα αυτή αναμένεται να βοηθήσει περισσότερο στις επόμενες εργασίες.

Να σημειωθεί ότι τα semantics είναι ίδια, δηλαδή τα keywords εξακολουθούν να αναφέρονται σε ‘cuda’ παρόλο που στη πραγματικότητα δε χρησιμοποιείται, όπως έχει ορίσει στο documentation η PyTorch.

Λόγω της χρήσης GPU που δεν έχει επίσημη υποστήριξη από την AMD για τη συγκεκριμένη χρήση, ήταν αναγκαίο να τρέξει το command ‘export HSA_OVERRIDE_GFX_VERSION=10.3.0’ στην κονσόλα ώστε να τρέξει το πρόγραμμα στην GPU. Λόγω της εμφάνισης ορισμένων warnings που αφορούν την “unofficial” χρήση επιλέχθηκε να φιλτραριστούν στην αρχή του προγράμματος, εφόσον δεν φαίνεται να επηρεάζουν αρνητικά την εκτέλεσή του.

Το dataset που επιλέχθηκε ήταν το Cifar-10, ένα από τα (πολλά) προτεινόμενα. Ένα πλεονέκτημά του είναι ότι παρόλο που είναι image database, έχει διαχειρήσιμο μέγεθός σε σχέση με αρκετά άλλα.

Κατά την εκτέλεση του προγράμματος καταγράφεται ο χρόνος εκτέλεσης των:

- Initialization and loading phase
- 1-NN Classifier
- 3-NN Classifier
- Nearest Centroid Classifier
- Program as a whole, from start to finish

μέσω της time.perf_counter() (σημειώνοντας σε variables τα start και end times των παραπάνω φάσεων) ώστε να μπορούν να συγκριθούν οι αποδόσεις των classifiers όχι μόνο σε accuracy αλλά και σε χρόνο, καθώς και οι διαφορές μεταξύ εκτέλεσης σε CPU και σε GPU.

Αρχικά ορίζεται η μέθοδος φόρτωσης του database χρησιμοποιώντας την cPickle, σύμφωνα με τις οδηγίες που δίνονται στην ιστοσελίδα του Cifar-10:

```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

Στη συνέχεια αφού γίνει έλεγχος για τη διαθεσιμότητα δυνατότητας GPU compute (‘torch.cuda.is_available()’), ώστε να επιλεχθεί το κατάλληλο device, τα δεδομένα φορτώνονται στο πρόγραμμα: το test_batch (data και labels) φορτώνεται μονομιάς, όμως το training data προσφέρεται σε 5 ξεχωριστά αρχεία-batches, τα οποία φορτώνονται με χρήση for loop, συνδυάζοντας τα τελικά σε ένα μεγάλο batch το οποίο είναι και αυτό στο οποίο γίνεται η χρήση των classifiers.

Το data μετατρέπεται σε μορφή torch.tensor και αποθηκεύεται με datatype ‘float’, απαραίτητο για τις πράξεις που θα ακολουθήσουν.

Για την υλοποίηση του 1-NN Classifier:

Χρησιμοποιείται η έτοιμη συνάρτηση torch.cdist με p=2 για να υπολογισθεί η Ευκλείδεια απόσταση κάθε training sample από κάθε test sample. Με την επίσης έτοιμη συνάρτηση torch.argmin βρίσκεται το index της ελάχιστης απόστασης για κάθε training sample και έτσι εκμεταλλεύοντας το structure του dataset γίνεται πρόβλεψη για το label κάθε sample σε μία λίστα “predicted_labels”. Το accuracy του classifier ελέγχεται με τη σύγκριση αυτής της λίστας με τα πραγματικά labels του test_data.

Για την υλοποίηση του 3-NN Classifier:

Όμοια με πριν, χρησιμοποιείται η torch.dist με p=2 για να υπολογισθεί η Ευκλείδεια απόσταση κάθε training sample από κάθε test sample. Και πάλι με χρήση έτοιμης συνάρτησης, αυτή τη φορά της torch.topk, βρίσκονται οι 3 μικρότερες αποστάσεις για κάθε sample, και άρα με την torch.argmin βρίσκονται και τα indices τους. Μέσω αυτών γίνεται μετάβαση όπως και πριν σε labels, από τα οποία πρέπει να επιλεγεί αυτό που εμφανίζεται πιο συχνά. Η torch.unique, που επιστρέφει μοναδικά στοιχεία τένσορα, σε συνδυασμό με την προαιρετική παράμετρο return_counts σεταρισμένη σε True, που επιστρέφει τον αριθμό εμφανίσεων κάθε μοναδικού στοιχείου, δίνουν την πληροφορία που χρειάζεται ώστε με την torch.argmax να βρούμε το πιο συχνά εμφανιζόμενο label, το οποίο θέτεται και ως η πρόβλεψη του classifier. Όμοιως με παραπάνω υπολογίζεται το accuracy μέσω σύγκρισης με των πραγματικών labels.

Για την υλοποίηση του Κατηγοριοποιητή Πλησιέστερου Κέντρου:

Βάσει των labels τους, υπολογίζεται το κέντρο (δηλ. ο μέσος όρος / το mean) των τιμών κάθε κλάσης. Τα δεδομένα αυτά συγκεντρώνονται σε έναν ενιαίο τένσορα ‘centroid_tensor’ με τη χρήση ξανά μεθόδου της PyTorch, συγκεκριμένα της torch.stack. Η torch.cdist χρησιμοποιείται αυτή τη φορά σε σχέση με τον τένσορα centroid, και η argmin παίζει ακριβώς τον ίδιο ρόλο όσον αφορά τον τελικό ορισμό των προβλεπόμενων labels και την ακόλουθη σύγκριση με το test_data.

Συγκρίσεις:

Με χρήση GPU (ROCm, RX 6700 XT):

```
Time taken for initialization and data loading: 2.06 seconds
Accuracy of 1-NN classifier: 35.39%
Time taken for 1-NN: 0.42 seconds
Accuracy of 3-NN classifier: 33.02%
Time taken for 3-NN: 1.38 seconds
Accuracy of Nearest Centroid classifier: 27.74%
Time taken for Nearest Centroid: 0.03 seconds
Time taken for program execution: 3.90 seconds
```

Με χρήση CPU (Ryzen 7 5800X3D, 32 GB DDR4 RAM @ 3200 MHz):

```
Time taken for initialization and data loading: 1.89 seconds
Accuracy of 1-NN classifier: 35.40%
Time taken for 1-NN: 4.06 seconds
Accuracy of 3-NN classifier: 33.02%
Time taken for 3-NN: 0.26 seconds
Accuracy of Nearest Centroid classifier: 27.74%
Time taken for Nearest Centroid: 0.28 seconds
Time taken for program execution: 6.49 seconds
```

Παρατηρείται ότι οι αποδόσεις όσον αφορά το accuracy είναι πρακτικά ίδιες, με μία πολύ μικρή διαφορά 0.01% στην περίπτωση του 1-NN classifier η οποία ενδεχομένως να εξηγείται από μικρές διαφορές στο rounding (εφαρμόζεται στα 2 δεκαδικά ψηφία) ανά περίπτωση λόγω ελάχιστα διαφορετικών τιμών.

Η αρχική φόρτωση των δεδομένων είναι πιο γρήγορη στην περίπτωση του CPU, πράγμα που εξηγείται από το γεγονός ότι δε χρειάζεται να περασθούν τα δεδομένα στην VRAM της GPU, η οποία έχει μεγαλύτερο “κόστος” χρόνου στο initial loading απ’ ότι έχει το να μείνουν τα δεδομένα απλά στην RAM του συστήματος ή/και η cache του CPU.

Ο 1-NN Classifier τρέχει σχεδόν 10 φορές πιο γρήγορα στην GPU, λόγω των πολλών παράλληλων πράξεων. Δύσκολο στην εξήγηση φαίνεται όμως το γεγονός ότι ο 3-NN Classifier τρέχει πάνω από 5 φορές πιο γρήγορα στον CPU απ’ ότι στην GPU. Ίσως να περιέχεται κάποιο data loading στο παρασκήνιο το οποίο δεν έχω καταφέρει να εντοπίσω, ή δεν έχω καταλάβει σωστά το scaling του συγκεκριμένου classifier με την τιμή του k.

Ο Nearest Centroid τρέχει πιο γρήγορα στην GPU με την ίδια υπόθεση που έγινε για τον 1-NN.

Βλέπουμε ότι παρά τα όποια penalties στο data loading, ο ολικός χρόνος εκτέλεσης του προγράμματος εξακολουθεί να είναι υπέρ της GPU, ακόμα και με έναν σχετικά γρήγορο επεξεργαστικά CPU χρησιμοποιώντας ένα αρκετά μικρό database.

Ο 1-NN έχει την καλύτερη επίδοση σε accuracy, μετά ο 3-NN και μετά ο centroid. Και οι τρεις είναι καλύτεροι από τυχαία επιλογή αλλά όχι ιδιαίτερα ακριβείς.

Σύνδεσμοι:

https://gist.githubusercontent.com/damico/484f7b0a148a0c5f707054cf9c0a0533/raw/43c317bfbde626d9112d44462d815613194988e0/test-rocm.py

https://rocm.docs.amd.com/projects/install-on-linux/en/develop/install/3rd-party/pytorch-install.html

https://rocm.docs.amd.com/projects/install-on-linux/en/latest/install/native-install/ubuntu.html

https://github.com/ROCm/ROCm/issues/3882

https://pytorch.org/

https://www.cs.toronto.edu/~kriz/cifar.html

Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009. (https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf)

Χρησιμοποιήθηκε VSCode:

https://code.visualstudio.com/