Software Development for Algorithmic Problems
Project 3

Στοιχεία Φοιτητών

Λάζαρος Αυγερίδης - 1115201600013
Θέμης Βαρβέρης - 1115201600015
Github:
https://github.com/lazavgeridis/Comparing-Vector-Space-Representations-for-Images

Σύντομη Περιγραφή Προγραμμάτων

Μέρος Α

Στο πρώτο μέρος της εργασίας καλούμαστε να υλοποιήσουμε ένα νευρωνικό δίκτυο αυτοκωδικοποίησης εικόνων και πιο συγκεκριμένα του MNIST dataset. Το δίκτυο αυτό περιλαμβάνει στρώματα κωδικοποίησης και αποκωδικοποίησης καθώς και ένα “bottleneck” layer το οποίο είναι το latent vector των εικόνων. Σκοπός του προγράμματος είναι, αφού εκπαιδεύσουμε το δίκτυο με διαφορετικές υπερπαραμέτρους, να καταλήξουμε σε αυτό που έχει το μικρότερο loss κατά την εκπαίδευση του δικτύου. Έπειτα κάνουμε extract το latent vector, το μεσαίο layer, (default latent dimension = 10) και πλέον κάνουμε predict στον νέο διανυσματικό χώρο. Τέλος αποθηκεύουμε το dataset αλλά και το queryset σε δύο διαφορετικα αρχεία εξόδου, όμως πλέον οι εικόνες βρίσκονται σε διάσταση ίση με το latent dimension (το οποίο το δίνει ο χρήστης). Τα αρχεία εξόδου θα χρησιμοποιηθούν στο ερώτημα Β.

Μέρος Β

Σε αυτό το μέρος της εργασίας επεκτείνεται η λειτουργία της 1ης εργασίας, όπου πραγματοποιείται αναζήτηση πλησιέστερου γείτονα, όμως αυτή την φορά χρησιμοποιούνται τα αρχεία εξόδου που παρήχθησαν στο ερώτημα Α. Συνεπώς η αναζήτηση γίνεται στον νέο διανυσματικό χώρο (default 10d). Επίσης πραγματοποιείται και αναζήτηση προσεγγιστικού πλησιέστερου γείτονα με χρήση LSH, όπως στην 1η εργασία. Ολες οι παραπάνω αναζητήσεις γίνονται με την μετρική Manhattan. Τέλος συγκρίνουμε τα αποτελέσματα μας ως προς τον χρόνο αλλά και το approximation factor ως προς τον αρχικό χώρο διάστασης των εικόνων. Τα

Software Development for Algorithmic Problems
Project 3

αποτελέσματα μας τα γράφουμε σε αρχείο εξόδου, όπου ορίζει ο χρήστης στην γραμμή εντολών, σύμφωνα με το format που ζητείται.

Μέρος Γ

Σε αυτό το μέρος της εργασίας υλοποιήθηκε αναζήτηση πλησιέστερου γείτονα στον αρχικό διανυσματικό χώρο (784d) χρησιμοποιώντας 2 διαφορετικές μετρικές:

1. earth mover's distance
2. manhattan distance

Πιο συγκεκριμένα, για κάθε στοιχείο του query set εκτελούμε εξαντλητική αναζήτηση σε ολόκληρο το training set / input dataset και βρίσκουμε τους 10 πλησιέστερους γείτονές του.
Συγκρίνοντας τις 2 προσεγγίσεις, παρατηρούμε ότι η μετρική manhattan είναι συνολικά καλύτερη όσον αφορά την ορθότητα και το χρόνο εκτέλεσης της αναζήτησης, τουλάχιστον στο συγκεκριμένο dataset (MNIST)

Μέρος Δ

Στο κομμάτι αυτό, κάνουμε σύγκριση και αξιολόγηση των διαδικασιών:

1. clustering στο νέο διανυσματικό χώρο (10d) + μετάβαση στον αρχικό διανυσματικό χώρο (784d) και υπολογισμός silhouette και objective function
2. clustering στον αρχικό διανυσματικό χώρο (784d) + silhouette και objective function στον αρχικό διανυσματικό χώρο (784d)
3. χρήση του "clustering" που έχει προκύψει από τις προβλέψεις του νευρωνικού δικτύου (Project 2) + silhouette και objective function στον αρχικό διανυσματικό χώρο (784d)

Συγκρίνοντας τα 3 διαφορετικά clusterings, συμπεραίνουμε ότι το (1) δίνει πάντα τα "χειρότερα" αποτελέσματα, δηλαδή τη μικρότερη τιμή σιλουέτας και την μεγαλύτερη τιμή objective function. Προφανώς, λόγω της μειωμένης διάστασης των δεδομένων, το clustering διαρκεί το λιγότερο χρόνο σε αυτή την περίπτωση. Τα (2), (3) δίνουν παρόμοια αποτελέσματα κάθε φορά, πράγμα που υποδηλώνει ότι το clustering είναι παρόμοιας ποιότητας.

Κατάλογος Αρχείων Κώδικα

(Περιγράφουμε μόνο τα αρχεία που είχαν ουσιαστικές προσθήκες)

- **include:**
    - cluster: περιέχει τα αρχεία:

Software Development for Algorithmic Problems
Project 3

i. *cluster.conf* : περιέχει πλέον μόνο τον αριθμό των clusters
ii. *cluster.h* : εδώ υλοποιείται όλη η λειτουργία των αντικειμένων τύπου Cluster (k-medians++, init++, median update, objective function, silhouette κλπ)

- io_utils: προστέθηκε η λειτουργία ανάγνωσης των αρχείων που περιέχουν τα labels στο *io_utils.h*

- **src:**
  - cpp:
    i. cluster:
      1. *cluster_app.cpp*: εκτελεί τα 3 διαφορετικά clusterings και κάνει σύγκριση των αποτελεσμάτων στον αρχικό και στο μειωμένο διανυσματικό χώρο
      2. *cluster_utils.cpp*: προστέθηκε η λειτουργία ανάγνωσης του αρχείου που περιέχει τα predicted classes του νευρωνικού δικτύου
    ii. common:
      1. io_utils.cpp: προστέθηκαν ορισμένες λειτουργίες για το parsing των command line arguments και η συνάρτηση για το γράψιμο των αποτελεσμάτων, στην μορφή που ζητείται, στο αρχείο εξόδου.
    iii. emd:
      1. *search.cc*: για να καταφέρουμε να κάνουμε compile το πρόγραμμα με ενσωματωμένη τη βιβλιοθήκη γραμμικού προγραμματισμού της google, αναγκαστήκαμε να βάλουμε όλο το πηγαίο κώδικα σε ένα μόνο αρχείο. Στο αρχείο αυτό, υλοποιείται η αναζήτηση πλησιέστερου γείτονα χρησιμοποιώντας τις μετρικές emd και manhattan
    iv. search: Αυτός ο κατάλογος περιέχει 2 αρχεία τα οποία είναι τα εξής:
      1. Makefile: είναι το αρχείο για την μεταγλώττιση του προγράμματος
      2. search.cpp: το αρχείο αυτό αποτελεί την main λειτουργία του προγράμματος που μας ζητείται να υλοποιήσουμε στο ερώτημα Β.
  - python:
    i. *classification.py* : προστέθηκε επιλογή να γίνεται write των predicted classes του νευρωνικού δικτύου σε αρχείο output

Software Development for Algorithmic Problems
Project 3

ii. *cnn_model.py* : πλέον η αρχιτεκτονική του μοντέλου μας είναι “φιξαρισμένη”, καθώς εκτελέσαμε πολλαπλά πειράματα και καταλήξαμε σε αυτό που έβγαζε το χαμηλότερο loss.
iii. *reduce.py* : το αρχείο αυτό αποτελεί την βασική λειτουργία της εκτέλεσης του ερωτήματος Α.
iv. *utils.py* : προστέθηκε συνάρτηση που υλοποιεί την εγγραφή των predicted classes του νευρωνικού δικτύου σε αρχείο output

- output_files: στο directory αυτό έχουμε αποθηκεύσει τα outputs των προγραμμάτων μας (*reduce.py*, *search* - B, *search* - Γ, cluster)

Οδηγίες Χρήσης

Μέρος Α

Για την εκτέλεση του προγράμματος αφού βρισκόμαστε στο directory **src/python/** εκτελούμε την παρακάτω εντολή (παράδειγμα εκτέλεσης):

$ python3 reduce.py --dataset ../../datasets/train-images-idx3-ubyte --queryset ../../datasets/t10k-images-idx3-ubyte -od output_dataset_file -oq output_query_file

Αφού διαβαστούν τα αρχεία που δίνονται στο command line, το πρόγραμμα ζητάει από τον χρήστη να δώσει σαν παράμετρο το **latent dimension, epochs και batch_size,** έπειτα ρωτάει τον χρήστη εάν θέλει να φορτώσει κάποιο pre-trained μοντέλο και αν ναι, τότε του ζητάει να δώσει το path. Σε περίπτωση λάθους εμφανίζει αναγνωριστικό μήνυμα και το πρόγραμμα τερματίζει. Εάν ο χρήστης δεν επιθυμεί να φορτώσει κάποιο pre-trained μοντέλο τότε το πρόγραμμα δημιουργεί και εκπαιδεύει το μοντέλου που έχουμε ορίσει εμείς (μετά από πολλούς πειραματισμούς καταλήξαμε σε ένα μοντέλο, όπου έβγαζε τα καλύτερα αποτελέσματα σε σχέση με αυτά που είχαν προηγηθεί). Η διαδικασία είναι η ίδια με αυτή που έχουμε υλοποιήσει στο poject2. Τέλος το πρόγραμμα αποθηκεύει τα αποτελέσματα του στα αρχεία που είχαν δοθεί στην γραμμή εντολών κατα την έναρξη του προγράμματος. Για το γράψιμο των εικόνων, στο αρχείο εξόδου, στην νέα διάσταση πραγματοποιείται κανονικοποίηση των pixels από 0 έως 65535 (δηλάδη μεγέθους 2 bytes), ώστε να χάνεται όσο λιγότερη πληροφορία γίνεται. Οπότε τα pixels αποθηκεύονται σε 2 διαδοχικα bytes, αντί για 1 όπως ήταν στο project2, αντίστοιχα γίνεται και το διάβασμα των νέων αυτών αρχείων.

Μέρος Β

Software Development for Algorithmic Problems
Project 3

Για την μεταγλώττιση του προγράμματος αφού βρισκόμαστε στο directory **src/cpp/search** εκτελούμε την παρακάτω εντολή:

$ make

Για την εκτέλεση του προγράμματος αφού βρισκόμαστε στο directory **src/cpp/search/** εκτελούμε την παρακάτω εντολή (παράδειγμα εκτέλεσης):

$ ./search -d ../../../datasets/train-images-idx3-ubyte -i ../../../output_files/output_dataset_file -q ../../../datasets/t10k-images-idx3-ubyte -s ../../../output_files/output_query_file -k 4 -L 5 -o output

Για την διαγραφή του output file, των object files αλλά και του εκτελέσιμου εκτελούμε την παρακάτω εντολή:

$ make clean

Μέρος Γ

Όπως αναφέρθηκε παραπάνω, η βιβλιοθήκη γραμμικού προγραμματισμού που χρησιμοποιήθηκε είναι κομμάτι του ευρύτερου λογισμικού operations-research που παρέχει η google, οπότε για να εκτελεστεί το πρόγραμμα αυτού του μέρους, πρέπει πρώτα να το έχουμε κάνει install στο σύστημά μας (για c++ στην περίπτωσή μας): https://developers.google.com/optimization/install/cpp
Για να κάνουμε compile το πρόγραμμα πρέπει να βρισκόμαστε στο top-level directory όπου κάναμε install το or-tools (ή με άλλα λόγια στο directory στο οποίο βρίσκεται το Makefile τους) και τρέχουμε την εντολή:

$ make DEBUG='-Ofast' build SOURCE=relative/path/to/Comparing_Vector_Space_Represenations_for_Images/src/cpp/emd/search.cc

Για να εκτελέσουμε το πρόγραμμα, πρέπει από το directory στο οποίο βρισκόμασταν πριν, να μεταφερθούμε στο ./bin/, όπου εκεί βρίσκεται το εκτελέσιμο που παράχθηκε και στη συνέχεια τρέχουμε την εντολή:

$ ./search -d relative/path/to/Comparing_Vector_Space_Represenations_for_Images/datasets/trai

Software Development for Algorithmic Problems
Project 3

n-images-idx3-ubyte -q relative/path/to/Comparing_Vector_Space_Represenations_for_Images/datasets/t10k-images-idx3-ubyte -l1 relative/path/to/Comparing_Vector_Space_Represenations_for_Images/datasets/train-labels-idx1-ubyte -l2 relative/path/to/Comparing_Vector_Space_Represenations_for_Images/datasets/t10k-labels-idx1-ubyte -o relative/path/to/Comparing_Vector_Space_Represenations_for_Images/output_files/emd_results -EMD

Μετά την επιτυχή εκτέλεση του προγράμματος, εκτυπώνονται στο stdout οι χρόνοι εκτέλεσης των 2 αναζητήσεων, και τα ποσοστά ακρίβειας αποθηκεύονται στο αρχείο εξόδου που προσδιόρισε ο χρήστης στη γραμμή εντολών (αν το αρχείο υπήρχε από πριν, αλλιώς δημιουργείται ένα “output” αρχείο στο Comparing_Vector_Space_Represenations_for_Images/src/cpp/emd/ directory και τα αποτελέσματα αποθηκεύονται εκεί)

(**Συμπεράσματα - Παρατηρήσεις**): Τα παρακάτω συμπεράσματα προέκυψαν από εκτελέσεις του *search* για τα πρώτα 20 queries του “query” set χρησιμοποιώντας και τις 60000 εικόνες του training set. Βλέπουμε ότι, για το MNIST dataset, η μετρική manhattan δίνει καλύτερο accuracy (97%) και διαρκεί συνολικά πολύ λιγότερο (0.1 secs) από την μετρική emd.
Αντίθετα, η μετρική emd, στην περίπτωση όπου χρησιμοποιούμε για κάθε εικόνα 4x4 clusters και σε κάθε cluster έχουμε 7x7 pixels, δίνει accuracy 84%, αλλά διαρκεί πολύ περισσότερο, ~12 λεπτά. Όταν χρησιμοποιούμε για κάθε εικόνα 7x7 clusters και συνεπώς σε κάθε cluster έχουμε 4x4 pixels, το accuracy αυξάνεται σε 95,5%, όμως αυξάνεται ταυτόχρονα σημαντικά και ο χρόνος αναζήτησης σε περισσότερο από μία ώρα (~69 λεπτά). Ενδεχομένως, αν δοκιμάσουμε 14x14 clusters, και συνεπώς 2x2 pixels σε κάθε cluster, το accuracy να αυξηθεί ακόμη περισσότερο, αλλά ο χρόνος εκτέλεσης θα αυξηθεί και αυτός ανάλογα.
Ένας από τους λόγους που η αναζήτηση με emd διαρκεί σημαντικά περισσότερο από την αναζήτηση με manhattan, είναι ότι στην 1η περίπτωση, για να υπολογιστεί η “απόσταση” μεταξύ του query και μιας μόνο εικόνας του training set πρέπει να λυθεί ένα γραμμικό σύστημα n^2 μεταβλητών με $n^2 + 2n$ περιορισμούς, όπου n είναι ο αριθμός των clusters στα οποία χωρίζεται η κάθε εικόνα.

Μέρος Δ

Software Development for Algorithmic Problems
Project 3

Η εκτέλεση του προγράμματος γίνεται με την εξής εντολή:

$ make
$ ./cluster -d <input file original space> -i <input file new space>
-n <classes from NN as clusters file> -c <configuration file>
-o <output file>

Μετά την επιτυχή εκτέλεση του προγράμματος, τα αποτελέσματα για κάθε clustering αποθηκεύονται στο αρχείο εξόδου που προσδιόρισε ο χρήστης στη γραμμή εντολών (αν το αρχείο υπήρχε από πριν, αλλιώς δημιουργείται ένα “output” στο working directory και τα αποτελέσματα αποθηκεύονται εκεί).

(**Συμπεράσματα - Παρατηρήσεις**): Τα παρακάτω συμπεράσματα προέκυψαν μετά από 5 εκτελέσεις του *cluster*. Είναι φανερό ότι το clustering στο νέο χώρο οδηγεί στα χειρότερα αποτελέσματα. Αυτό είναι κάτι αναμενόμενο, καθώς τα ίδια δεδομένα αναπαρίστανται πλέον σ’ ένα χώρο πολύ μικρότερης διάστασης όπου η απώλεια πληροφορίας είναι δεδομένη. Επιπλέον, μην ξεχνάμε ότι τα δεδομένα αυτά είναι εσωτερικές αναπαραστάσεις ενός νευρωνικού δικτύου, οι οποίες εξαρτώνται σε μεγάλο βαθμό από το “βάθος” του δικτύου, καθώς και από το αν έχει εκπαιδευτεί επαρκώς η όχι. Τώρα, σχετικά με τα (2) και (3), τα αποτελέσματα είναι παρόμοια, αλλά το (2) παρουσίασε ελαφρώς καλύτερα αποτελέσματα. Πιο συγκεκριμένα, η συνολική τιμή της silhouette ήταν πάντα μεγαλύτερη και επίσης η τιμή του objective function ήταν πάντα μικρότερη. Ωστόσο, όπως φαίνεται και παρακάτω, η διαφορά είναι αρκετά μικρή (οι τιμές του clustering - nn είναι σταθερές καθώς χρησιμοποιούμε πάντα τα ίδια predicted “clusters”) :

Silhouette

	Εκτέλεση 1	Εκτέλεση 2	Εκτέλεση 3	Εκτέλεση 4	Εκτέλεση 5
clustering - new space	0.03	0.04	0.03	0.02	0.02
clustering - original space	0.138	0.143	0.139	0.15	0.138
clustering - nn	0.131	0.131	0.131	0.131	0.131

Software Development for Algorithmic Problems
Project 3

Objective Function

	Εκτέλεση 1	Εκτέλεση 2	Εκτέλεση 3	Εκτέλεση 4	Εκτέλεση 5
clustering - new space	1.254.467.366	1.266.712.621	1.261.915.658	1.260.455.343	1.251.495.234
clustering - original space	1.144.903.747	1.147.204.403	1.145.868.374	1.144.524.965	1.144.756.328
clustering - nn	1.174.635.776	1.174.635.776	1.174.635.776	1.174.635.776	1.174.635.776