Κ23γ: Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα

3η Προγραμματιστική Εργασία - Μείωση διάστασης μέσω Νευρωνικού Δικτύου και πειραματική μελέτη

Μαρία Σιακαβέλλα - 1115201400181

Τσάμης Αντώνιος - 1115201300183

Περιγραφή του προγράμματος

Για την 3η εργασία αναπτύξαμε τρία νευρωνικά δίκτυα αυτοκωδικοποίησης τα οποία χρησιμοποιήσαμε για την συμπίεση των δεδομένων σε 10 και 20 διαστάσεις. Επίσης τροποποιήσαμε τον κώδικα των δύο προηγούμενων εργασιών ώστε να χρησιμοποιεί τα συμπιεσμένα δεδομένα για την εκτέλεση των αλγορίθμων και τα μη συμπιεσμένα για την αξιολόγηση των αλγορίθμων.

Github link: mariasiak/project_2023 (github.com)

Κατάλογος των αρχείων κώδικα / επικεφαλίδων και περιγραφή τους

Στον φάκελο neural_networks θα βρείτε τον κώδικα σε αρχεία .py και .ipynb για τα τρία νευρωνικά δίκτυα που αναπτύξαμε με ονόματα nn1, nn2, nn3. Στο τέλος του παρόντος README έχουμε τοποθετήσει τις περιγραφές των νευρωνικών δικτύων. Και στις τρεις περιπτώσεις εκπαιδεύσαμε το νευρωνικό δίκτυο ολόκληρο (κωδικοποιητή και αποκωδικοποιητή) και στη συνέχεια χρησιμοποιήσαμε μόνο τον κωδικοποιητή για την συμπίεση των δεδομένων.

nn1
Νευρωνικό δίκτυο για τη συμπίεση των δεδομένων σε 10 διαστάσεις.

nn2
Νευρωνικό δίκτυο, με προσθήκες επιπλέον στρώσεων στο nn1 για τη συμπίεση των δεδομένων σε 10 διαστάσεις.

nn3
Νευρωνικό δίκτυο για τη συμπίεση των δεδομένων σε 20 διαστάσεις.

Οδηγίες μεταγλώττισης του προγράμματος

Η μεταγλώττιση του προγράμματος γίνεται με το εργαλείο make. Παρακάτω παραθέτουμε λίστα των εντολών και των λειτουργιών τους.

all: κάνει compile όλα τα εκτελέσιμα
lsh: compile για lsh
lsh_run: εκτέλεση lsh με συγκεκριμένα ορίσματα
cube: compile για hypercube
cube_run: εκτέλεση hypercube με συγκεκριμένα ορίσματα
cluster: compile για clustering
cluster_run: εκτέλεση clustering με συγκεκριμένα ορίσματα
graph_search: compile για graph search
gnns_run: εκτέλεση clustering με συγκεκριμένα ορίσματα
mrng_run: εκτέλεση clustering με συγκεκριμένα ορίσματα
clear: αφαίρεση εκτελέσιμων και αρχείων εξόδου (τελειώνουν σε .out)

Οδηγίες χρήσης του προγράμματος

Οι αλγόριθμοι των εργασιών 1 και 2 εκτελούνται όπως και στις προηγούμενες εργασίες με την προσθήκη όμως δύο καινούργιων παραμέτρων εισόδου -dl <compressed_input_path> και -ql <compressed_query_path>. Για παράδειγμα:

./lsh -d dataset/input.dat -q dataset/query.dat -dl dataset/input_nn1.csv -ql dataset/query_nn1.csv -k 5 -L 5 -o lsh.out -N 1 -R 0

Για την εκτέλεση των νευρωνικών δικτύων αρκεί να εκτελεστεί το αντίστοιχο .py ή .ipynb. Για την είσοδο των παραμέτρων στα νευρωνικά δίκτυα πρέπει να τροποποιηθεί το αρχείο reduce.config στον φάκελο neural_networks. Στο αρχείο αυτό υπάρχουν 4 γραμμές για τις παρακάτω παραμέτρους:

dataset: <original_input_dataset>
queryset: <original_input_queryset>
output_dataset_file: <compressed_output_dataset>
output_query_file: <compressed_output_queryset>

Τα νευρωνικά χρησιμοποιούν τα dataset και queryset για ονόματα αρχείων εισόδου και τα output_dataset_file και output_query_file για ονόματα αρχείων εξόδου. τα αρχεία εξόδου που δημιουργούνται είναι σε μορφή .csv.

ΣΗΜΑΝΤΙΚΟ Στο αρχείο source/common.h περιέχονται τα #define MAX_DATA_SIZE και MAX_QUERY_SIZE τα οποία δηλώνουν πόσα από τα στοιχεία του dataset θα χρησιμοποιηθούν για τον αλγόριθμο και πόσα από τα στοιχεία ελέγχου θα χρησιμοποιηθούν ως queries. Κατά την ανάπτυξη της εργασίας οι αριθμοί αυτοί ήταν σχετικά μικροί (~10000 για data size, ~5 για query size). Στο παραδοτέο τα έχουμε θέσει σε 60000 (για να χρησιμοποιείται όλο το dataset) και 100. Σε περίπτωση που το query file σας περιέχει περισσότερα από 100 στοιχεία παρακαλούμε αλλάξτε τον αριθμό αυτό. Ειδικά ο αριθμός query size ήταν αναγκαίος για την ανάπτυξη του κώδικα καθώς χρησιμοποιήσαμε το test dataset του mnist με τα 10000 σημεία.

Επίσης στο αρχείο source/common.h περιέχεται το #define DIMENSIONS το οποίο πρέπει να είναι ίσο με τον αριθμό των συμπιεσμένων διαστάσεων.

Σχολιασμός αποτελεσμάτων

			Original	NN1	NN2	NN3
LSH	(L,k) = 5,5	tAA (μs)	4625	237	283	334
		AAF	1,58	1,63	1,58	1,55
	(L,k) = 25,25	tAA (μs)	25072	1526	1537	2030
		AAF	1,35	1,46	1,40	1,36
	(L,k) = 100,100	tAA (μs)	135906	10024	10025	12528
		AAF	1,23	1,30	1,28	1,28
Hypercube	(k,probes) = 5,1	tAA (μs)	165692	8463	8366	12190
		AAF	1,08	1,28	1,26	1,20
	(k,probes) = 10,1	tAA (μs)	10081	588	644	831
		AAF	1,44	1,53	1,51	1,46
	(k,probes) = 15,4	tAA (μs)	60208	11721	11806	12789
		AAF	1,27	1,36	1,30	1,27
GNNS		tAA (μs)	9366	498	435	517
		AAF	1,34	1,87	1,92	1,84
MRNG		tAA (μs)	4419	494	468	711
		AAF	1,54	2,84	2,83	2,89
Clustering	(clusters) = 10	SScore	0,08	0,11	0,17	0,08
	tAverageTrue(μs)		875300			

Πιο πάνω παρουσιάζουμε στατιστικά που συλλέξαμε από εκτελέσεις των αλγορίθμων για διάφορες παραμέτρους στα ασυμπίεστα δεδομένα (Original) και στα συμπιεσμένα (nn1, nn2 με 10 διαστάσεις και nn3 με 20 διαστάσεις).

Στις δοκιμές μας χρησιμοποιήσαμε και τα 60000 train σημεία για την κατασκευή των δομών και τα 100 πρώτα test σημεία για queries. Όλοι οι χρόνοι είναι σε microseconds. Με AAF σημειώνουμε το κλάσμα Average Approximate Factor.Με tAA σημειώνουμε τον χρόνο tAverageApproximate. Παραθέτουμε για σύγκριση και τον χρόνο tAverageTrue που ήταν σε όλες τις εκτελέσεις περίπου ίδιος (περίπου 870 milliseconds ή 0,87 δευτερόλεπτα).

Παρακάτω παραθέτουμε τις παρατηρήσεις και τα συμπεράσματά μας από τις εκτελέσεις των αλγορίθμων.

Παρατηρούμε ότι οι χρόνοι tAA στα συμπιεσμένα δεδομένα είναι σημαντικά μικρότερη (τουλάχιστον υποδεκαπλάσιοι). Τα κλάσματα AAF είναι περίπου 10% μεγαλύτερα για τα συμπιεσμένα δεδομένα στους αλγορίθμους LSH και Hypercube και σημαντικά πιο μεγάλα για τους αλγορίθμους GNNS και MRNG. Το Silhouette Score είναι καλύτερο στα συμπιεσμένα δεδομένα σε σχέση με τα ασυμπίεστα για τα νευρωνικά nn1 και nn2. Η συμπίεση των δεδομένων φαίνεται να είναι σε γενικές γραμμές αποτελεσματική καθώς ο χρόνος εύρεσης απαντήσεων μειώνεται κατά πολύ και τα αποτελέσματα γίνονται ελαφρώς χειρότερα.

Neural Network 1

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 64)        640

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        18464

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 16)          4624

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 16)          0
 g2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 10)                2570

 dense_1 (Dense)             (None, 64)                704

 reshape (Reshape)           (None, 4, 4, 4)           0

 conv2d_3 (Conv2D)           (None, 4, 4, 16)          592

 up_sampling2d (UpSampling2  (None, 8, 8, 16)          0
 D)

 conv2d_4 (Conv2D)           (None, 8, 8, 32)          4640

 up_sampling2d_1 (UpSamplin  (None, 16, 16, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 64)        18496

 up_sampling2d_2 (UpSamplin  (None, 28, 28, 64)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 28, 28, 1)         577

=================================================================
Total params: 51307 (200.42 KB)
Trainable params: 51307 (200.42 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Neural Network 2

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 64)        640

 batch_normalization (Batch  (None, 28, 28, 64)        256
 Normalization)

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        18464

 batch_normalization_1 (Bat  (None, 14, 14, 32)        128
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 16)          4624

 batch_normalization_2 (Bat  (None, 7, 7, 16)          64
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 16)          0
 g2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 10)                2570

 batch_normalization_3 (Bat  (None, 10)                40
 chNormalization)

 dense_1 (Dense)             (None, 256)               2816

 reshape (Reshape)           (None, 4, 4, 16)          0

 conv2d_3 (Conv2D)           (None, 4, 4, 16)          2320

 up_sampling2d (UpSampling2  (None, 8, 8, 16)          0
 D)

 conv2d_4 (Conv2D)           (None, 8, 8, 32)          4640

 up_sampling2d_1 (UpSamplin  (None, 16, 16, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 64)        18496

 up_sampling2d_2 (UpSamplin  (None, 28, 28, 64)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 28, 28, 1)         577
=================================================================
Total params: 55635 (217.32 KB)
Trainable params: 55391 (216.37 KB)
Non-trainable params: 244 (976.00 Byte)
_________________________________________________________________
```

Neural Network 3

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 64)        640

 batch_normalization (Batch  (None, 28, 28, 64)        256
 Normalization)

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        18464

 batch_normalization_1 (Bat  (None, 14, 14, 32)        128
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 16)          4624

 batch_normalization_2 (Bat  (None, 7, 7, 16)          64
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 16)          0
 g2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 20)                5140

 batch_normalization_3 (Bat  (None, 20)                80
 chNormalization)

 dense_1 (Dense)             (None, 256)               5376

 reshape (Reshape)           (None, 4, 4, 16)          0

 conv2d_3 (Conv2D)           (None, 4, 4, 16)          2320

 up_sampling2d (UpSampling2  (None, 8, 8, 16)          0
 D)

 conv2d_4 (Conv2D)           (None, 8, 8, 32)          4640

 up_sampling2d_1 (UpSamplin  (None, 16, 16, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 64)        18496

 up_sampling2d_2 (UpSamplin  (None, 28, 28, 64)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 28, 28, 1)         577

=================================================================
Total params: 60805 (237.52 KB)
Trainable params: 60541 (236.49 KB)
Non-trainable params: 264 (1.03 KB)
_________________________________________________________________
```