A Neural Network Approach for Binary Hashing in Image Retrieval

Mohamed Moheeb Emara[1(✉)], Mohamed Waleed Fahkr[2], and M.B. Abdelhalim[1]

[1] Arab Academy for Science Technology and Maritime Transport, College of Computing and Information Technology, Cairo, Egypt
mohamed.moheeb90@gmail.com, mbakr@ieee.org

[2] Arab Academy for Science Technology and Maritime Transport, College of Engineering and Technology, Cairo, Egypt
waleedf@aast.edu

Abstract. Online and cloud storage has become an increasingly popular location to store personal data that led to raising the concerns about storage and retrieval. Similarity-preserving hashing techniques were used for fast storing and retrieval of data. In this paper, a new technique is proposed that uses both randomizing and hashing techniques in a joint structure. The proposed structure uses a Siamese-Twin architecture neural network that applies random projection on data before being used. Furthermore, Particle Swarm Optimization and Genetic Algorithms are used to fine-tune the Siamese-Twin neural network. The proposed technique produces a compact binary code with better retrieval performance than other hashing randomizing technique that varies from 2 % to 5 %.

Keywords: Neural network · Genetic algorithms · Similarity preserving hashing · Random projection

1 Introduction

In the past few years, the use of online and cloud storage has increased exponentially since it facilitates the storing of personal information such as face images and fingerprint. Consequently, two main concerns are raised; namely the ability to store and retrieve data efficiently.

One of the approaches that are used to efficiently store and retrieve data is the similarity preserving hashing function, in which each image feature vector is converted to a binary code where similar data has similar binary codes [1–3]. The binary code is very efficient in terms of storing the data since the space usage is very low and therefore the retrieval becomes fast. It is divided into different families like random hashing methods [4–6] and machine learning techniques [7–9]. The problem with randomizing hashing, it needs long codes to produce higher accuracy. A new direction was taken to produce shorter codes with higher accuracy using learning-based hashing methods. One of the most famous techniques of this direction is Binary Reconstructive Embedding (BRE) [2], Minimal Loss Hashing (MLH) [3] and Spectral Hashing (SP) [10].

© Springer International Publishing AG 2017

In [1] locality sensitive hashing (LSH) is one of the randomized hashing techniques family. It uses a hash function that map the similarity from original space to the same hashing buckets with high probability. It depends on a random hyperplane with a Gaussian independent distribution. The hyperplane is used to produce similar code for similar inputs. There are different similarity criteria used like cosine similarity and Jaccard similarity [11].

In [2] binary reconstructive embedding is introduced where the Euclidean distance between inputs in the input space is calculated, and then the hamming distance between binary codes in the hamming space is calculated, the loss function used as a hash function learning is based on minimizing the error between the difference of the two spaces.

In [3] minimal loss hashing is introduced where a structural SVMs was proposed that apply an online algorithm with latent variables. It uses a loss function that takes into consideration hamming distance and binary quantization.

In [10] spectral hashing is introduced. It was noticed that the problem of finding good binary codes clearly resembles the graph portioning problem. It aims to keep neighbors in input space as neighbors in the hamming space.

This paper proposes a binary hashing technique that creates discriminative binary codes that are compact. It is applied on a Context Based Image Retrieval system (CBIR) where the search is done according to the image content similarity.

Siamese-Twin Random Projection Neural Network (STRPNN) architecture is proposed which is composed of two identical random projections with nonlinear hard-threshold neurons with adjustable bias. The Random Projection Neural Network (RPNN) executes extensive random projection on the input feature space, and the hard threshold produces the binary code. Both Genetic algorithm and particle swarm optimization are used to both select an optimal sparse number of neurons to ensure a short binary code, as well as to adjust the thresholds and the weights of the selected neurons.

The paper organization is as follows: Sect. 2 is a detailed explanation of the STRPNN. Section 3 is a detailed explanation of the GA-PSO tuning of the STRPNN. Section 4 explains the supervised and unsupervised versions of the STRPNN followed by a comparison between normal STRPNN and GA-PSO tuned STRPNN as well as a comparison between GA-PSOSTRPNN and hashing techniques. Section 5 is a conclusion about the proposed technique.

2 Siamese Twin Random Projection Neural Network (STRPNN)

The original idea of a Siamese structure was introduced in [12]. The basic idea is to find optimal weight that produce small distance between the output if the inputs are from the same category and large distance if the inputs are not.

The proposed structure for the Siamese twin Random Projection Neural Network (STRPNN) is shown in Fig. 1. It is composed of two identical RPNN structures with hard-threshold neurons. The STRPNN can work in both supervised and unsupervised image retrieval systems, where the training consists of selecting the best neurons, and tuning the thresholds and weights of the neurons. It converts feature data into secure

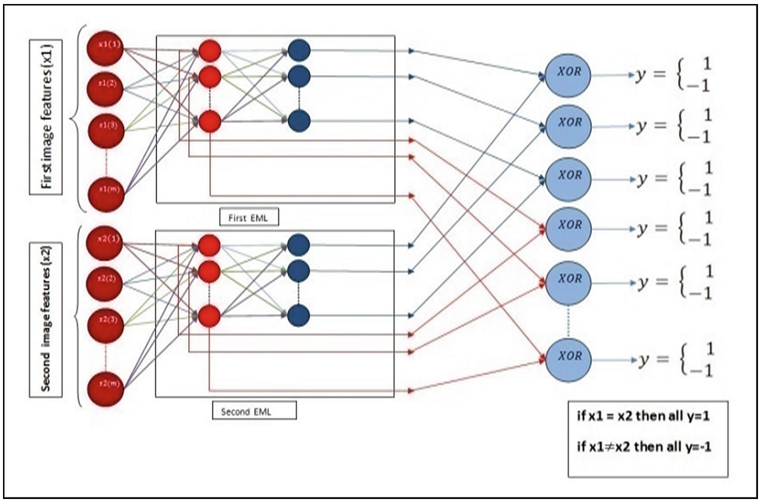

Fig. 1. Siamese Twin Random Projection Neural Network

binary codes where the objective is to produce similar codes for similar images and vice versa.

STRPNN goes through two main stages: the training stage and the binary code selection stage. In the first stage, the Euclidean distance is calculated between each feature vector (query) and the remaining training items, to select the closest (positive) and farthest (negative) match. In case of the supervised RPNN the positive class item has to be from the same class of the query while the negative class item has to be from different class than the query.

The query could have one or more positive and negative matches. The Target vector T is the vector that holds the values of the query compared to negative and positive matches. Target vector is composed of the $T_i i = 1 : N$ values where $T_i = 1$ if the 2 images are similar enough, and $T_i = -1$ if the 2 images are significantly different. RPNN consists of *J* neurons; the input for RPNN is the feature vector from the training data multiplied by random projection. The neurons outputs can be thought of as a vector Y which is given by for the twin neural networks:

$$Y_1 = f(\Phi * X_1) \tag{1}$$

$$Y_2 = f(\Phi * X_2) \tag{2}$$

Where X_1 and X_2 are the input feature vectors to the twin neural networks and *f* is the hard-thresholding neuron. Each neuron also has an adjustable threshold which is considered as an extra input feature with a value of 1. For the j_{th} neuron, the output is:

$$y_{ij} = f\left(\phi_j * X_k\right) \tag{3}$$

Where ϕ_j is the part of the random matrix Φ that is connected to the j_{th} neuron. Twin Random Projection Neural Network consists of two RPNN during the training phase; the first one is for the query image feature and the second RPNN is for checking the similarity with the query. To check for similarity the output of the first and second RPNN's are XORed: if they are similar the output (Z) is 1 and if they are not the output (Z) is −1:

$$if\, y_{1j} = y_{2j}\; Z_j = 1,\;\; and\ if\;\; y_{1j} \neq y_{2j}\; Z_j = -1 \tag{4}$$

The next stage is the binary code selection. The outputs of the neurons over the whole training data are used to determine which of the neurons are more correlated to the target. The top M neurons (M the size of binary code produced) that have the highest correlation with the targets over the whole training data are kept: For each neuron calculate the following:

$$S_j = \sum\nolimits_{i-1}^{N} y_{ij} * T_i \tag{5}$$

Thus, if y_{ij} (the output of the jth neuron for the ith training example) has the same sign as the target (T) then this adds to the score of this neuron, while if the neuron output is different than the target then this will discount from its score. Finally, the scores of all the neurons are sorted and the top M kept.

Even though this FAST technique is very quick and does not need solving of optimization problems, it suffers from an obvious drawback: Some of the remaining neurons may be identical or highly correlated which will result in redundancy and sub-optimal sparsity. This can be resolved by applying some enhancement to the algorithm by checking the remaining neurons correlation and removing the redundant ones. The GA solves this problem as explained in the next section.

3 GA and PSO Tuning of the STRPN

The Genetic Algorithm (GA) was introduced by John Holland [13]. It applies Darwinian evaluation theory in searching for optimal solution for a given problem. Particle Swarm Optimization (PSO) was introduced by Kennedy and Eberhart [14]. It was inspired by the behavior and movement of animals in nature like birds flock. It was simplified and applied also in searching for optimal solution for a given problem.

Particle Swarm Optimization and Genetic Algorithms are used with the training of the STRPNN to improve the distinction of the produced binary codes and therefore improve the performance. The PSO works on improving the hard-limiter neuron thresholds. As for the GA, it works on selecting the best neurons to create the best sparse binary code. The number of neurons selected is according to the binary code length required.

PSO depend on velocity and position of each particle, velocity and position are update at each iteration to find optimal solution, Eq. (6) is for updating velocity and

Eq. (7) is used to update position found in [14] where c_1, c_2 and ω are parameters set by the user and r_1 and r_2 are random number generated in the range [0 – 1].

$$V_{id}^{t+1} = w.V_{id}^{t} + c_1.r_1\left(P_{id}^{t} - X_{id}^{t}\right) + c_2.r_2\left(P_{gb}^{t} - X_{id}^{t}\right) \tag{6}$$

$$X_{id}^{t+1} = X_{id}^{t} + V_{id}^{t+1} \tag{7}$$

3.1 PSO Tuning of Thresholds

The PSO is used in tuning the hard-limiter neurons' thresholds. The function of hard-limiter neurons in RPNN is to converts input data to binary code. It depends on threshold: if the weighted sum is less than the threshold the output is −1 otherwise it is equal to 1. The PSO is used to find the best threshold for each neuron. The PSO consists of N particles; each particle consists of a random vector. Each random value in the vector represents a threshold for each neuron. The optimization is done on the random vector to find the best possible threshold suited for each neuron.

$$if\ \ y_j \geq P_j\ then\, f(Y) = 1\ else\, f(Y) = -1 \tag{8}$$

The fitness function starts by using the initialized particles as weights for hard limited binary neurons (Eq. (8)). The vector P is the particle of N values. The vector has the same size as number of neurons in the RPNN, if the value of y_j is less than P_j value the output is −1 otherwise it is 1. The P vectors value are improved after each iteration using PSO. The next step is XORing the output of twin RPNN (explained in Sect. 2) using Eq. (4). To select the top M neurons to produce the binary code Eq. (5) is used (explained in Sect. 2). The next step is to convert the validation data using trained RPNN with each particle as hard-limiter to binary code. The output of the top M neurons is used in producing the validation binary code. The next step is to compare the distinctive of validation data in the original space with the binary codes produced in binary space. For this task a class matrix that contains the class of validation data compared with training data is produced in the original space and another is produced for binary codes in the Hamming space and then they are matched together as shown in Eq. (9):

$$MA = ||V - TD|| \tag{9}$$

Where *MA* have rows equal to number of training data and columns equal to the number of validation data. The intersection between the columns represents the Euclidian distance between training item and validation item. V is the validation data and TD is the training data

$$\mathrm{L} = \mathrm{binarized\ (MA)\ according\ \ to\ TH} \tag{10}$$

Where *TH* is calculated by sorting the Euclidian distance between the training data followed by selecting top *C* values and averaging it. *L* is the binary matrix where the element is 1 when the training data and validation data are from the same class, and is 0 it is not from the same class.

Equations (9) and (10) are used one more time to produce a class matrix for the Hamming space using the same value of C. The next step is to match the two matrices together using XNOR, as shown in Eq. (11).

$$f(P_i) = \mathrm{sum}(\overline{\mathrm{MA} \oplus \mathrm{MB}}) \tag{11}$$

Where $f(P_i)$ represent the fitness function for each Particle P_i and is the sum of the matching pairs. *MA* is the class matrix for original data and *MB* is the class matrix for binary codes.

$$G = \sum\nolimits_{j=1}^{m} \max(f(P_1), f(P_2), \ldots, f(P_n)) \tag{12}$$

In Eq. (12) m is the maximum number of iteration done by PSO, $f(P_n)$ is the sum of matching pairs produced by each particle calculated in Eq. (11). The basic idea for matching the two matrices used with PSO or GA, this to ensure that the binary code produced has its equivalent pair in the feature data.

3.2 Genetic Algorithm Sparse Neurons Selection

The GA is used to select the best neurons to create binary code. The problem with the FAST technique is some of the unselected neurons maybe more significant than the selected neurons. This is due to the fact that selecting the most correlated neurons doesn't ensure that best performance, least correlated neurons when combined with other neurons can produce a better distinct code therefore increasing the performance. The FAST technique selects the top M neurons that have the highest correlation with the targets over the whole training data. The GA overcomes this problem by exploring the different combination of neurons to find the most significant neurons. Each chromosome in the GA has same size as number of neurons, the 1s in the chromosome represent the index of the selected neurons to produce binary code otherwise the 0s represent the unselected neurons.

The GA RPNN population is initialized by four chromosomes. Each chromosome is a binary vector where 1's represent the selected neurons and 0's represent the unselected neurons. The first chromosome is initialized by the output of the FAST technique in case it acquires the best performance, the other three are initialized randomly according to the required binary code (if the required code is M the number of 1's should be exactly equal to M).

The Genetic algorithm is used with RPNN to replace the FAST technique and it uses a similar fitness function like the one used with PSO, instead of using the top M ranked neurons a randomly selected M neurons are selected to produce the binary validation data then calculate class matrix for both features and binary training and validation data.

The sum of matching pairs is also calculated using Eq. (11). The difference here is that the matching pairs reflects the quality of each chromosome in the Population. The reason for using GA is to find the best set of neurons of size M that produce the best distinct binary code. In Eq. (12) m is the max number of iteration done by GA, $f(P_n)$ is the sum of matching pairs for each chromosome.

For the selection phase the *roulette wheel* is used. In the crossover, the order 1 crossover is used to prevent any increase or decrease in the required code size as shown in Fig. 2., it is done by creating random permuted vectors as shown in Fig. 2(a), the value in the permuted vector are converted into 1's and 0's in the equivalent binary chromosomes. So if the binary code required like in Fig. 2 are 4 bits, the numbers from one to four is converted to 1's otherwise the numbers are converted to 0's. This is to make sure that value in the two parents (permutated vector) are the same in the equivalent binary chromosome for example the value 1, 2, 3 and 4 in Fig. 2(a) in both parents have the same value in the equivalent binary chromosomes in Fig. 2(b) equal to 1, therefore the number of 1's for sure will not increase or decrease. The order one crossover [15] is then applied on those two vectors to produce the new two children. If a value is moved from the random permuted vector (the parent) to the new child. The index of this value is taken and the same move is applied for the equivalent binary chromosome shown in Fig. 2(b). In the mutation phase, a swap takes place between two randomly selected bits from the newly created children. The last step is the elimination of the least two fit chromosomes and replacing them with the new produced children.

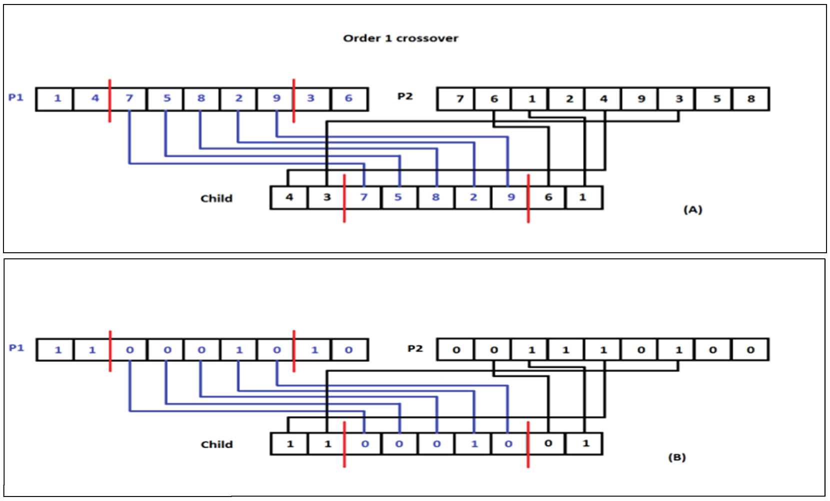

Fig. 2. The use of order 1 to maintain the size of binary code (4 bits binary code)

The technique used in pervious section is a smaller version of the GA-PSO RPNN, it doesn't take a lot of time in training and is suitable to use in application that has tight time constrain compared to GA-PSO RPNN. The two techniques only takes time in the training processes but takes average time in executing on testing data. The GA-PSO RPNN produce more secure and accurate binary codes since it has to extra layers of fine tuning and randomization using PSO and GA.

4 Experiments

4.1 Dataset Description

In following experiments, the COREL 1 k dataset is used. It has 10 categories and each category has 100 images. The COREL dataset is used in the first part of the experiments using STRPNN without GA-PSO. In the second part, the small datasets reported in [2] are used, namely the Labelme and MNIST. Each dataset consists of 5000 image feature vectors.

4.2 Feature Extraction

The features used on COREL database is a concatenation of indexed color histogram, Discrete Cosine Transform, Color Histogram, GIST and SURF-VLAD with vector lengths of 64, 64, 192, 512 and 1000 respectively. The formed concatenation feature is than reduced using PCA to a 25 dimension vector. Each image in the Labelme dataset is represented by a 512 GIST vector and each image in the MNIST dataset is represented by a 784 GIST vector [2].

4.3 Supervised versus Unsupervised RPNN

This experiment is a comparison between the supervised and unsupervised. This experiment is a comparison between the supervised and unsupervised STRPNN to find out which is has a better accuracy. The experiment is executed with the same random projection, number of neurons and number of neighbors and the accuracy performance

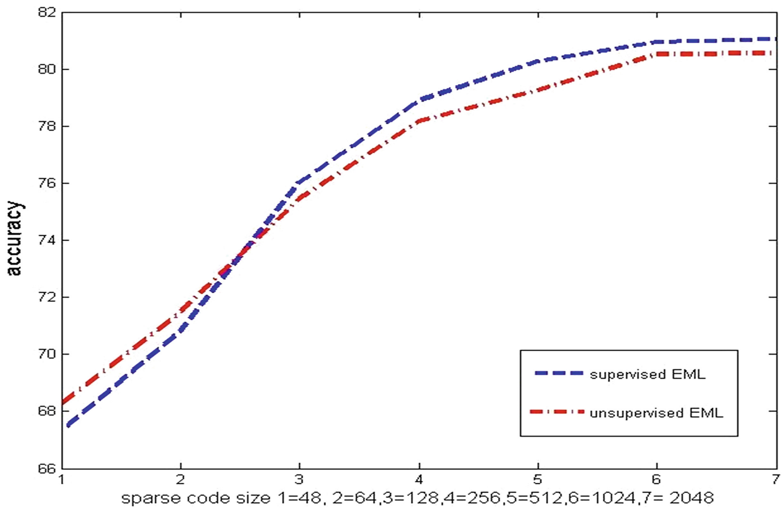

Fig. 3. Unsupervised vs. supervised for 8000 neurons

is shown in Fig. 3. The accuracy of unsupervised RPNN is better than supervised up to code length of 128 but the supervised overcomes unsupervised for longer codes. This experiment proves that the STRPNN can work on both supervised and unsupervised data with almost the same accuracy.

4.4 Comparing Hashing Techniques and GA-PSOSTRPNN

This subsection discusses a comparison between PSO and GA tuned STRPNN and other hashing techniques for compact binary codes using 2 dataset, namely MNIST and Labelme [5]. Each dataset is divided into 3000 images as training, 1000 for validation (used in the PSO and GA fitness function) and 1000 images as testing data randomly. The Euclidean distance is calculated for training data and itself, the 50 nearest data items are selected and averaged. The neighbors are the items that are less than average (ground truth neighbors), the PCA is applied on mean centered data and the top 40 PCA features are kept.

In this experiment, the RPNN used has 200 neurons with one hidden layer. PSO uses velocity update Eq. (5) that has ω (Inertia weight) equal to 0.9, c_1 equal to 1 and c_2 equal to 2 [14]. The GA algorithm has a mutation rate of 0.05 and crossover rate of 0.6 [13].

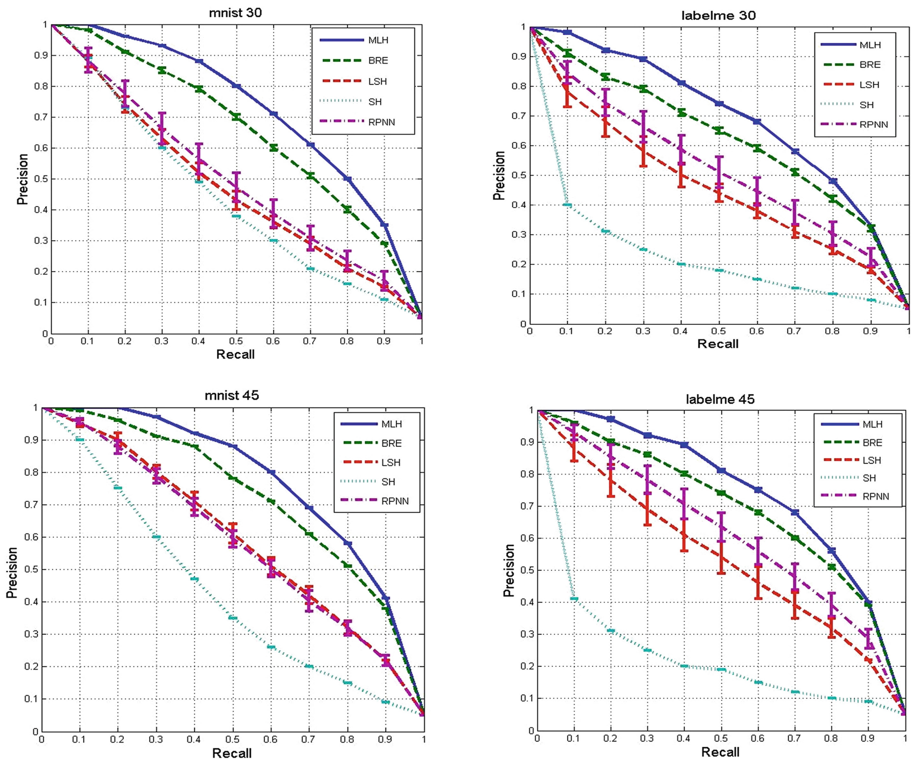

Fig. 4. Hashing techniques compared to RPNN [3]

The precision and recall curve found in [6] is used to compare between STRPNN and other state of the art hashing techniques. The precision recall curve is produced after varying the radius *R* from zero to "*q*", which is the radius value where the recall is equal to one. The experiments are executed 10 times and results shown represent the average with standard deviation as error bars, as shown in Fig. 4.

The results show that the RPNN is clearly better in precision than SH and LSH in almost the four precision and recall curves, but still need improvement to overcome the BRE and MLH.

4.5 Comparison Between Normal RPNN and GA-PSO RPNN

To conduct this experiment a precision and recall curve for 50 bits using COREL dataset concatenated features. The COREL dataset is divided into 500 images for training and validation and 500 images for testing. The same settings for normal RPNN and GA-PSO RPNN are used with the same random projection. The RPNN uses 200 neurons with one hidden layer and the settings for PSO and GA are the same as the previous subsection. In Fig. 5, the result shows that at recall 0.5 the precision of normal RPNN is improved by 15 % using PSOGA RPNN. The PSO and GA improved the average precision of normal RPNN by about 17.5 %.

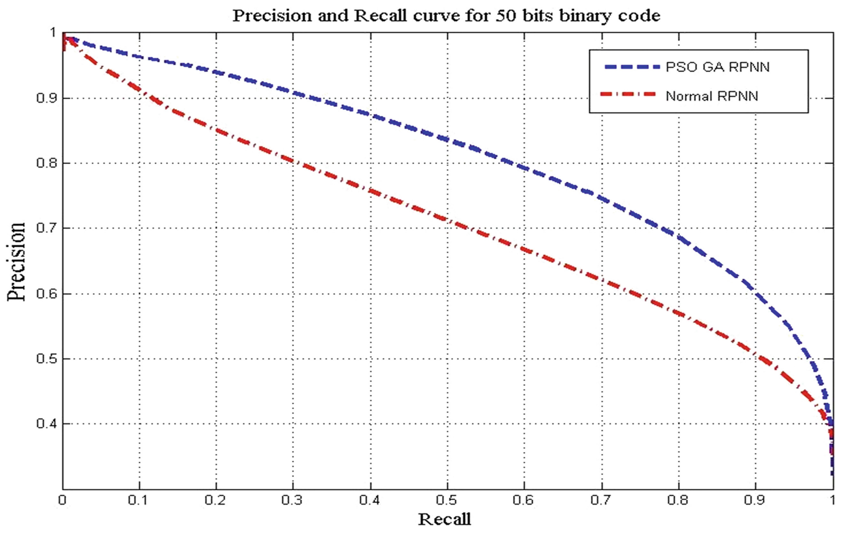

Fig. 5. Comparison between GA-PSO RPNN and normal RPNN

5 Conclusion

In this paper, a Siamese-Twin Random Projection Neural Network (STRPNN) is proposed for the fast retrieval and efficient storing of data, in which a combination of neurons are selected to produce the required binary code. A fine-tune PSO and GA are applied in the selecting process. STRPNN can also be used in both supervised and unsupervised data modes based on the availability of labeled data. When it is compared to other hashing techniques it overcomes SH with a precision that varies from 2 % to 20 % in average and LSH with a precision that varies from 2 % to 10 % in average, but it needs more improvement to overcome MLH and BRE [16] which will require the training of the random matrix weights for the selected neurons, and this is a future direction for research.

References

1. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. VLDB **99**(6), 518–529 (1999)
2. Kulis, B., Darrell, T.: Learning to hash with binary reconstructive embeddings. In: Advances in Neural Information Processing Systems, pp. 1042–1050 (2009)
3. Norouzi, M., Fleet, D.J.: Minimal loss hashing for compact binary codes. In: mij, 1, p. 2 (2011)
4. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe LSH: efficient indexing for high-dimensional similarity search. In: The 33rd International Conference on Very Large Data Bases, pp. 950–961. Endowment (2007)
5. Bawa, M., Condie, T., Ganesan, P.: LSH forest: self-tuning indexes for similarity search. In: The 14th International Conference on World Wide Web, pp. 651–660. ACM (2005)
6. Dong, W., Wang, Z., Josephson, W., Charikar, M., Li, K.: Modeling LSH for performance tuning. In: 17th ACM Conference on Information and Knowledge Management, pp. 669–678. ACM (2008)
7. Liu, W., Wang, J., Kumar, S., Chang, S.F.: Hashing with graphs. In: Proceedings of the International Conference on Machine Learning, Bellevue, WA, USA, pp. 1–8 (2011)
8. Gong, Y., Kumar, S., Verma, V., Lazebnik, S.: Angular quantization based binary codes for fast similarity search. In: Advances in Neural Information Processing Systems, Cambridge, MA, USA, vol. 25, pp. 1205–1213. MIT Press (2012)
9. Norouzi, M., Fleet, D.J., Salakhutdinov, R.R.: Hamming distance metric learning. In: Advances in Neural Information Processing Systems, Cambridge, MA, USA, pp. 1070–107. MIT Press (2012)
10. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: Advances in Neural Information Processing Systems, pp. 1753–1760 (2009)
11. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: The Thiry-Fourth Annual ACM Symposium on Theory of Computing, pp. 380–388. ACM (2002)
12. Chopra, S., Hadsell, R., LeCun, Y.: Learning a similarity metric discriminatively, with application to face verification. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005, vol. 1, pp. 539–546. IEEE (2005)

13. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University Michigan Press, Cambridge (1975)
14. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: The Sixth International Symposium on Micro Machine and Human Science, vol. 1, pp. 39–43 (1995)
15. Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D.: Genetic algorithms for the traveling salesman problem. In: The First International Conference on Genetic Algorithms and Their Applications, pp. 160–168. Lawrence Erlbaum, New Jersey (1985)
16. Wang, J., Liu, W., Kumar, S., Chang, S.F.: Learning to hash for indexing big data—a survey. IEEE **104**(1), 34–57 (2016)