Auton Robot (2017) 41:1–18
DOI 10.1007/s10514-015-9516-2

# Unsupervised learning to detect loops using deep neural networks for visual SLAM system

**Xiang Gao**[1] · **Tao Zhang**[1]

Received: 22 May 2015 / Accepted: 15 October 2015 / Published online: 11 December 2015
© Springer Science+Business Media New York 2015

**Abstract** This paper is concerned of the loop closure detection problem for visual simultaneous localization and mapping systems. We propose a novel approach based on the stacked denoising auto-encoder (SDA), a multi-layer neural network that autonomously learns an compressed representation from the raw input data in an unsupervised way. Different with the traditional bag-of-words based methods, the deep network has the ability to learn the complex inner structures in image data, while no longer needs to manually design the visual features. Our approach employs the characteristics of the SDA to solve the loop detection problem. The workflow of training the network, utilizing the features and computing the similarity score is presented. The performance of SDA is evaluated by a comparison study with Fab-map 2.0 using data from open datasets and physical robots. The results show that SDA is feasible for detecting loops at a satisfactory precision and can therefore provide an alternative way for visual SLAM systems.

**Keywords** Simultaneous localization and mapping (SLAM) · Loop closure detection · Stacked denoising auto-encoder · Deep neural network

## 1 Introduction

The simultaneous localization and mapping (SLAM) is regarded as one of the critical technologies in autonomous robot researches and has been extensively investigated in the past years (Boal et al. 2014; Puente and Rodriguez-Losada 2014). In a typical SLAM system, the robot will build a model of the environment around it and estimate the trajectory of itself simultaneously (Konolige and Agrawal 2008). The whole SLAM system relies on several key algorithms, like feature extraction, registration and loop closure detection. The loop detection algorithms aim at recognizing the previously visited places during the travel of the mobile robot (Ho and Newman 2007). If the loops are correctly detected, the robot will be able to re-localize itself, or help the mapping and registration algorithms obtain a more accurate and consistent result (Williams et al. 2011). Specifically, in visual SLAM systems based on pose graph, the loops will put extra constraints into the graph and reduce the optimization error (Endres et al. 2014). In filter-based systems like MonoSLAM (Davison et al. 2007), the loops are treated as the data association problem, which significantly affects the convergence of the filter.

In this paper we focus on the loop closure detection for visual SLAM systems. Many of the these systems are based on pose graph where the trajectory is built of a series of adjacent key-frames (Henry et al. 2012; Chen et al. 2007; Strasdat et al. 2012; Tian et al. 2013). The easiest way to detect loops is to pair-wisely match the visual features of all key-frames but this is often too time-consuming for real-time applications. Randomly checking a subset of the previous frames is an easy and practical way which has been applied in some applications (Endres et al. 2014; Stuckler and Behnke 2014). Once a loop is found, researchers will compute a spanning tree of the adjacent frames, and optimize the whole trajectory using pose graph tools like g2o (Kummerle et al. 2011). However, such a search strategy is aimless and the possibility of detecting true loops will decrease as the trajectory getting longer.

✉ Tao Zhang
taozhang@mail.tsinghua.edu.cn

Xiang Gao
gaoxiang12@mails.tsinghua.edu.cn

[1] Department of Automation, Tsinghua University, Beijing 100084, China

Other methods either take advantage of the odometry or the observation information. Odometry-based methods search the key-frames that is close to the current pose by assuming that the estimation of the past trajectory is accurate enough to find a loop (Hahnel et al. 2003). However, such an assumption is often too optimistic in real environments where the global odometry error is large due to the accumulating drift (Beeson et al. 2010). The second kind, which directly employs the observation data, has no relationship with the previously estimated trajectory. It compares the similarity between the observations, i.e., images in visual SLAM, then raises a loop hypothesis where the similarity is high enough. Hence, such a method is also known as the appearance-based SLAM (Cummins and Newman 2011).

The appearance-based methods have a long history since the beginning of this century (Dudek and Jugessur 2000; Ulrich and Nourbakhsh 2000). They have been successfully applied in practical SLAM systems (Labbe and Michaud 2013; Latif et al. 2013). Many of state-of-the-art appearance-based loop detection algorithms employ the bag-of-words (BoW) model (Filliat 2007; Kwon et al. 2013), which clusters the visual feature descriptors into a set called "dictionary". The dictionary contains the centers of the clusters which are considered to be independent with each other. When a new observation comes, the visual features, such as SIFT (Lowe 2004) or SURF (Bay et al. 2006), are extracted and the descriptors are approximated by the entries in the dictionary. If the image has a feature occurred in the dictionary, we say this image has such a "word". By this way, the original image can be represented by a vector indicating whether it has each word in the dictionary or not. The similarity of the observations is measured by the disparity of the corresponding vectors. These vectors can also be used to recognize places. Researchers have exploited various classifiers to solve the place recognition problem, such as support vector machine (SVM) (Kostavelis and Gasteratos 2013) and randomize tree (Lepetit and Fua 2006).

However, BoW-based approaches also have limitations. First, false positive loops (similar observation in different places) may seriously corrupt the estimated map and lead to a totally erroneous result. It is often observed in real environments that there are many scattered objects like chairs and desks in an office. The images captured in such a environment will have many similar words and can increase the possibility of encountering scene ambiguity (Salas-Moreno 2013). Second, the visual features, which are the actual content of the words, are usually manually designed by the researchers in computer vision area. There are numerous types of features and the quantity of them is still growing as the proposing of new features and improving of the traditional ones (Rosten and Drummond 2006; Rublee et al. 2011; Wang and Lin 2014). The problem that which one is better for visual SLAM or loop closure detection has been widely studied (Shi et al. 2013; Gil et al. 2010; Morell-Gimenez et al. 2014), but the conclusion is still indeterminate because each feature has its own characteristic. Some are fast but less distinctive and others are unique but more complex in computing. Furthermore, for new sensors like depth camera, designing an appropriate feature for them is also an ongoing topic (Ren et al. 2012). Third, the criteria of designing traditional visual features is the feeling of humans, i.e., the features are expected to be stable under the change of the camera view or distance. In other words, if we humans think two images are similar, the features are expected to be close in the feature space. However, this cannot be always guaranteed during the computing process. Also, we do not exactly know how to represent the complex structures in images. Perhaps there are good algorithms to detect corners or edges, but we are still unable to tell how to accurately describe a box or a chair.

The recent development of the deep learning technology (Bengio et al. 2013) have provided an alternative method for understanding the loop closure detection problem. The deep learning approaches try to learn a data representation directly from the raw sensor data, e.g., voices or images, via a multi-layer neural network. The features are learned during the training process and then used for classification and recognition. The application of deep learning covers a large area from speech recognition to large set image classification (Deng et al. 2013; Liou et al. 2014). However, in SLAM, deep learning is not fully applied except for a few work about object recognition (Bo et al. 2014).

In this paper, we propose a novel method that employs a modified stacked denoising auto-encoder (SDA) (Vincent et al. 2010), an unsupervised deep neural network, to solve the loop closure detection problem for visual SLAM systems. The key-point of our method is shown in Fig. 1 and is presented in detail in later sections. The modification is based on the consecutive motion assumption of the robot and can improve the result of learning. We present the way to train the network and to measure the similarity of key-frames. Furthermore, we discuss the effect of several important hyper-parameters like learning rate, sparse constraints and the dimension of the network by experiments on open datasets and real indoor environments. Finally, a comparison is provided about our approach and the FabMap 2.0 (Cummins and Newman 2011), a widely applied appearance-based algorithm used in many visual SLAM systems. By comparing the precision-recall curve and similarity matrix using data from the "New College' ', "City Centre" sequence (Cummins and Newman 2008) and the "r2_slam", "fr3_office" (Sturm et al. 2012), we can see SDA is able to effectively detect the loops in these experiments. We also provide a time cost analysis about our method.

The work presented here is improved from the work described in Gao and Zhang (2015) with more detailed experiments and discussion. The data collected in experiments

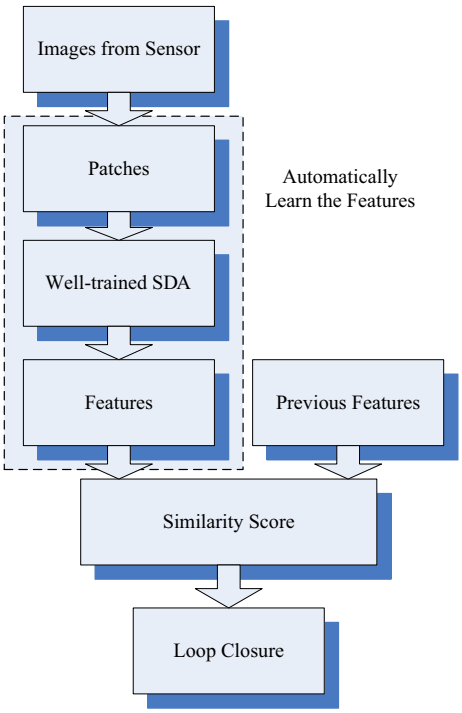


**Fig. 1** The diagram of the proposed method. The kernel of our method is to use a well-trained SDA (trained with similar data and proper parameter configurations) to automatically extract features

have also been used in our previous research (Gao and Zhang 2015). The paper is organized as follows: Sect. 2 introduces the basic theory of SDA and demonstrates how to train it for detecting loops. Section 3 presents the way to compute the similarity score based on the trained features. Then, we show the comparative results of experiments in Sect. 4. Section 5 discusses how to set the value of some important parameters. Finally, Sect. 6 gives a brief conclusion of this paper.

## 2 Learning features

Since SDA has not been widely applied in SLAM systems, we will briefly introduce the formulation of SDA at first, and then discuss our modifications. Readers can also find a more detailed introduction about SDA in (Bengio et al. 2013; Vincent et al. 2010).

### 2.1 Auto-encoders

SDA is a unsupervised neural network that learns an compressed representation of the input data. It contains several end-to-end layers and each layer is a Denoising Auto-encoder (DA). In SDA, the output of each DA is used as the input in the next layer. For a single DA, it is consisted of three layers: (1) input layer $\boldsymbol{x}$; (2) hidden layer $\boldsymbol{h}$; (3) recovery layer $\boldsymbol{y}$. Each layer contains many fully connected nodes which are the basic elements of the network. Each node computes a simple nonlinear function (usually sigmoid) from the connected input. Let $\boldsymbol{x}$ be its input and $a$ be the output, the function can be written as:

$$a = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{n} w_i x_i - b\right)} \tag{1}$$

where $\boldsymbol{w}, b$ are the weight and bias parameters contained in a single node. Notices that the $\boldsymbol{w}, b$ are goals of training. The well trained parameters can grab useful information from the input data.

In DA, the hidden layer is fully connected to the input layer, so the output of one hidden unit $h_j$ are:

$$h_j = f_{\theta_j}(\boldsymbol{x}) = \sigma\left(\boldsymbol{w}_j^T \boldsymbol{x} + b_j\right) \tag{2}$$

where $\theta_j$ denotes the parameters $\boldsymbol{w}_j, b_j$. Similarly, for the output layer $y_k$, we have:

$$y_k = g_{\theta_k}(\boldsymbol{h}) = \sigma\left(\boldsymbol{w}_k'^T \boldsymbol{h} + b_k'\right) \tag{3}$$

Put it into the matrix form, we have:

$$\begin{aligned} \boldsymbol{h} &= f_\theta(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\right) \\ \boldsymbol{y} &= g_\theta(\boldsymbol{h}) = \sigma\left(\boldsymbol{W}'^T \boldsymbol{h} + \boldsymbol{b}'\right) = g_\theta f_\theta(\boldsymbol{x}) \end{aligned} \tag{4}$$

Usually we set $\boldsymbol{W} = \boldsymbol{W}'^T$. The purpose of DA is to learn an equal representation of input data, i.e., to make $\boldsymbol{y} \approx \boldsymbol{x}$ and

$$g_\theta f_\theta(\cdot) = I(\cdot).$$

But $g_\theta$ and $h_\theta$ should not be identity functions $I$ because such a structure is useless. Finally, notice that in SDA, the hidden layer $\boldsymbol{h}$ is the real output fed to the latter layers, not $\boldsymbol{y}$.

During the training, $f_\theta$ is expected to grab abstract information from $\boldsymbol{x}$. Also, since the initial value of the parameters are randomly set and the dimension of hidden layer is also different with the input and output, the mapping function $f$ is irreversible and one cannot directly set $g_\theta$ to $f^{-1}$. The parameters $\boldsymbol{W}, b$ are trained by minimizing the error of construction, which is usually measured as the cross entropy if the input $\boldsymbol{x} \in [0, 1)$:

$$d = KL(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} x_i log\frac{y_i}{x_i} + (1 - x_i) log\frac{1 - x_i}{1 - y_i}. \tag{5}$$

The Auto-encoder employs the Stochastic Gradient Descent (SGD) to solve this optimization problem. SGD divides the optimizing (training) process into many small

epochs. In each epoch, the parameters are updated by a little step described by a learning rate $\eta$ along the descent gradient direction:

$$\theta^* = \theta - \eta \cdot \frac{\partial d}{\partial \theta} \tag{6}$$

After several epochs, the object function converges to a local minima and the algorithm is stopped. The recovery layer is then removed and the data in hidden layer is used as the output (or the features).

In real environments the input data captured from sensor is noisy and we do not want the noise be learned and represented. Therefore, the denoising auto-encoder (DA) is proposed (Vincent et al. 2008) which tries to reconstruct the data from a corrupted input $\widetilde{\boldsymbol{x}}$:

$$\min J = KL(\boldsymbol{x}, g_\theta f_\theta(\widetilde{\boldsymbol{x}})) \tag{7}$$

DA is a very useful extension of auto-encoder. The corruption is usually implemented by randomly masking certain percent of $\boldsymbol{x}$ into 0 or 1, so the DA will try to predict these missing values. It is observed in previous researches that such a corruption will help the auto-encoder get a more meaningful result (Poultney 2006), which will also be demonstrated in our experiments. DA has been applied in many pattern recognition tasks (Lu et al. 2013; Mesnil et al. 2012; Wang and Yeung 2013). Some important issues, including how to set the dimension of the hidden layer $\boldsymbol{h}$ and the initial value of the parameters, are also investigated and discussed. Generally speaking, if the dimension of $\boldsymbol{h}$ is smaller than the input $\boldsymbol{x}$, the result of auto-encoder is just like a nonlinear principal component analysis (Bourlard and Kamp 1988). On the other hand, if the dimension of $\boldsymbol{h}$ is larger, the training process is an over-complete one which will obtain better result (Ng 2011).

### 2.2 Overview of detecting loops

In order to train a structure for loop closure detection, we make a few modifications on traditional DAs. Unlike other pattern recognition tasks where the input is independent image, the data in loop closure detection is captured from the sensor, usually real-time video frames. Therefore, we can take advantage of the assumption that the observation of the sensor is continuous. To clarify how the modification is added, we first introduce the whole workflow of our approach.

The schematic overview of our approach is illustrated in Fig. 2. The raw observation data, i.e., gray-scale images, are first divided into small patches with size of $s \times s$. The patches are computed by sparse key-point detection algorithms like SIFT (Lowe 2004), FAST (Agrawal et al. 2008) or ORB (Rublee et al. 2011) and are then filtered to spread over the whole image. We sort the detected key-points in the descending order of the feature response, select the first $N$ key-points and then resize them into small image patches. The patches are vectorized and then fed to the neural network. Therefore, one input image will have $N$ patches which forms an input matrix $\boldsymbol{X}_{N \times s^2}$. Then, SDA corrupts the input and trains a structure to reconstruct it. The final hidden layer of SDA is used as feature output layer with the dimension of $N_F$. Therefore, once a new image comes, we can get a feature response $Z_{N \times N_F}$ from SDA, which is used for detecting loops.

The loops are detected if we find the similarity score of two frames exceed a given threshold. In the visualized similarity matrix image, a bright block indicates there may be a loop in this area. The way to compute similarity score is presented in Sect. 3. Therefore, when a new key-frame comes, what we need to do is put it into well-trained SDA, get the feature response, compute the similarity with previous key-frames, and then check if there is a loop. Regarding of this, our method is a online detection algorithm.

### 2.3 Modifications of DA

The SDA is trained greedily layer by layer. So the object function of SDA is same as DA. We modify the traditional DA by changing the form of object function:

$$J^* = c_d + \beta c_s + \gamma c_c \tag{8}$$

The joint cost function $J^*$ has three items: (1) The cross entropy of the input and the recovered output, which is denoted as

$$c_d = KL(\boldsymbol{x}, g_\theta f_\theta(\widetilde{\boldsymbol{x}}));$$

(2) The sparse constraint $c_s$; (3) the consecutive constraint $c_c$. The parameters $\beta$ and $\gamma$ are weight factors to balance the effect of each item.

The cross entropy item, i.e., the KL distance of the input and recovered output, has been discussed before. The sparse cost item $c_c$ is defined to prevent the algorithm to learn an identity function (Poultney 2006) and obtain a sparse result. Since the network is an over-complete one with large hidden layers, we do not want to get a full result where many hidden units are active (have non-zero response). Instead, we expect DA to find distinctive things in the trained data and discard the noise. The number of active units should be small and the response of them should be different according to input images. When an images comes, the units that describe a particular angle/corner/object should be active and others are silent.

The sparsity is achieved by penalizing the average output of hidden units:

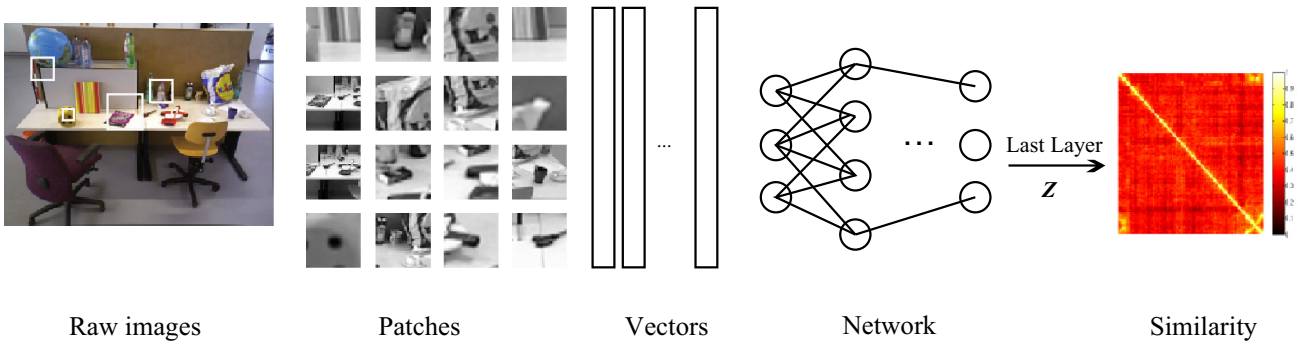


**Fig. 2** Overview of the approach. From *left* to *right*: Raw images obtained from the sensor; Patches of the key-points; Patches are vectorized into a set of vectors and then input to the network; The network is trained to recover the corrupted input; hidden variables are used as features to find loops

$$c_s(\boldsymbol{h}) = \frac{1}{N_F}\sum_{i=1}^{N_F}\|h_i - s_h\| \tag{9}$$

where $s_h$ is a threshold of sparsity. Assume that there are $N_b$ frames in a batch, the average sparse penalty will be:

$$c_s = \frac{1}{N_b}\sum_{i=1}^{N_b} c_s(\boldsymbol{h}_i). \tag{10}$$

An alternative way to do this is to penalize the $\mathcal{L}_1$ norm of the weight matrix $\boldsymbol{W}$. But restricting $\boldsymbol{h}$ seems to be more common (Bengio et al. 2013).

The consecutive cost $c_c$ keeps the response of adjacent frames to be similar. In SGD, we can set a relatively small batch size and assume that the data in a batch is close. The input of the network will be $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_b}$ and the hidden response are $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{N_b}$. We use $c_c$ to denote the average disparity of hidden outputs in one batch:

$$c_c = \frac{1}{N_b - 1}\sum_{i=1}^{N_b-1}\|\boldsymbol{h}_i - \boldsymbol{h}_{i+1}\|_2 \tag{11}$$

Put them all together, we have:

$$J^* = KL(\boldsymbol{x}, g_\theta f_\theta(\tilde{\boldsymbol{x}})) + \beta\frac{1}{N_b}\sum_{i=1}^{N_b} c_s(\boldsymbol{h}_i) + \gamma\frac{1}{N_b - 1}\sum_{i=1}^{N_b-1}\|\boldsymbol{h}_i - \boldsymbol{h}_{i+1}\|_2 \tag{12}$$

## 3 Defining the similarity

With the cost function $J^*$ and proper values of balance parameters $\beta$ and $\gamma$, we can obtain a set of hidden units $\boldsymbol{h}$ that grab useful, sparse, representative and denoising information from the input data. In ideal case, similar input (in the feeling of human beings) will have a similar hidden response in the network. Therefore, the obtained $\boldsymbol{h}$ can be used as distinctive features to measure the similarity of input images.

### 3.1 Computing the similarity

The purpose of loop closure detection is to find the same scenes in the robot's trajectory. Assume that there are two key-frames $F^{(1)}$, $F^{(2)}$, which contain $k_1, k_2$ features in total:

$$F^{(i)} = \{\boldsymbol{h}_1^{(i)}, \ldots, \boldsymbol{h}_{k_i}^{(i)}\}, i = 1, 2$$

where $\boldsymbol{h}$ is a the output of the last layer in SDA. We need to measure the similarity of these two frames. The detailed procedure is described in Algorithm 1.

**Algorithm 1** Computing the Similarity

**input:** Key-frames: $F^{(1)}$, $F^{(2)}$;
**output:** Similarity score: $S$;
1: Set $S = 0$;
2: Compute the average response $\bar{\boldsymbol{h}}$.
3: Compute the distinctive score of each hidden units:

$$\delta_i = \phi(h_i)$$

4: Match the features:

$$M = \left\{m_k | m_k = (\boldsymbol{h}_i^{(1)}, \boldsymbol{h}_j^{(2)}), 1 \le i \le k_1, 1 \le j \le k_2\right\}$$

5: **for** $m_k$ in $M$ **do**
6: Compute the weighted distance of features:

$$s_k = \left\|\delta^T\left(\boldsymbol{h}_i^{(1)} - \boldsymbol{h}_j^{(2)}\right)\right\|$$

7: Add to the similarity score:

$$S+ = \pi(s_k)$$

8: **end for**
9: **return** $S$

The algorithm has two parts. First, we compute the average response $\bar{\boldsymbol{h}}$ of each unit in SDA. If an unit is always activated (has a very high average response), it is likely to be an ordinary feature, such as a black block that may appear at floors, walls or the back of a chair. On the other hand, an unit that has very low response may have no useful information but only noise. Therefore, we prefer the units with medium response and regard them as distinctive features useful for recognition. The distinctive score is defined to measure such a preference, which is described by the function $\phi(\cdot)$. As the output of sigmoid function belongs to (0, 1), we choose the Gauss function to compute the score:

$$\delta_i = \phi(h_i) = exp\left(-\frac{(h_i - \mu)^2}{2\sigma^2}\right) \tag{13}$$

where $\mu$ and $\sigma$ are parameters to obtain a proper shape of Gauss function. The distinctive score $\delta$ is used as a weight vector, indicating that if the response of distinctive units are similar, the data is likely to be a loop.

Second, we match the features provided by SDA using existing algorithms. In the experiments we make use of the Brute-force match and the fast approximate nearest neighbor (FLANN) (Muja and Lowe 2009) implemented in OpenCV library (Bradski 2000). The match algorithm gives a list of possible matches denoted as $M$. For each match, the diversity of features can be measured by the weighted distance in the feature space:

$$s_k = \left\| \delta^T \left( \boldsymbol{h}_i^{(1)} - \boldsymbol{h}_j^{(2)} \right) \right\| \tag{14}$$

Finally, we compute an accumulating score using a similarity function $\pi$, whose purpose is to keep the score into a reasonable interval and to balance the effect of close and far matches. Because $s_k$ denotes the distance in feature space which is negatively correlated to similarity, we choose a log function:

$$\pi(s_k) = a + b \log(s_k) \tag{15}$$

with $b < 0$. The similarity score will be normalized to [0, 1] before loop closure detection.

### 3.2 Similarity matrix

The similarity score of each pair of key-frames forms a matrix $\boldsymbol{S}$ that describes the relationships of them. Figure 3a shows an example of a normalized similarity matrix, where the diagonal items are the most bright ones, and the bright part in top-right corner indicates that there is a possible loop. According to Ho and Newman (2007), a rank reduction process can help to reduce the ambiguity in detecting loops. So we also add such a step by discarding the largest $k$ eigenvalues.

$$\boldsymbol{S} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T, \quad \boldsymbol{\Lambda} = diag(\lambda_1, \ldots, \lambda_N), \lambda_1 > \ldots > \lambda_N$$

$$\boldsymbol{S_R} = \sum_{i=k+1}^{N} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T$$

Figure 3b shows the result of rank reduction with $k = 1$, i.e., removing the largest eigenvalue. It can be seen that the value of noise is reduced and the real loops are preserved.

## 4 Experiments

In this section, we demonstrate several offline experiments to evaluate the effect of our approach, compared with FabMap 2.0, a well-known BoW based loop closure detection algorithm. The input data is selected from tum open dataset (Sturm et al. 2012) and "New College", "City Centre" from Fabmap (Cummins and Newman 2008). We also show the results from a physical robot. Note that in order to check the loops correctly, we need a well-trained network before detecting the loops. So in these experiments we put all the images of key-frames into SDA and compute the similarities, which can be viewed as a off-line loop closure detection. However, if the robot is moving in the same environment, we can use the same network to check the loops on-line. After the experiments, we discuss how to choose the proper values of some important hyper-parameters by showing some failed training process, which we think will be helpful for future research.

### 4.1 The tum dataset

#### *4.1.1 Pre-process of the data*

The open dataset provided by Sturm et. al. (2012) has many RGB-D sequences with ground-truth trajectories. Figure 4 shows the trajectory, key-frames and sample images from the dataset "freiburg_office" where a Kinect is hold around a desk and moved to the original position at last. The trajectory of the Kinect is captured by an extrinsic tracking camera system with time-stamps. The odometry information comes at 100 Hz and the speed of video is 30 Hz.

The tum dataset is designed for verifying SLAM systems and do not provide ground truth loops, so we need to compute the true loops by ourselves. In a vSLAM system, key-frames are added after the registration algorithm finding the motion of camera has exceeded a certain threshold. Therefore, we divide the trajectory into a series of segments and record the ground-truth poses of key-frames: $\boldsymbol{T}_i, i = 1, \ldots, N$. The frames between these key-frames are discarded. Then, the relative distance between each pair of poses ($\boldsymbol{T}_i$ and $\boldsymbol{T}_j$) is calculated:

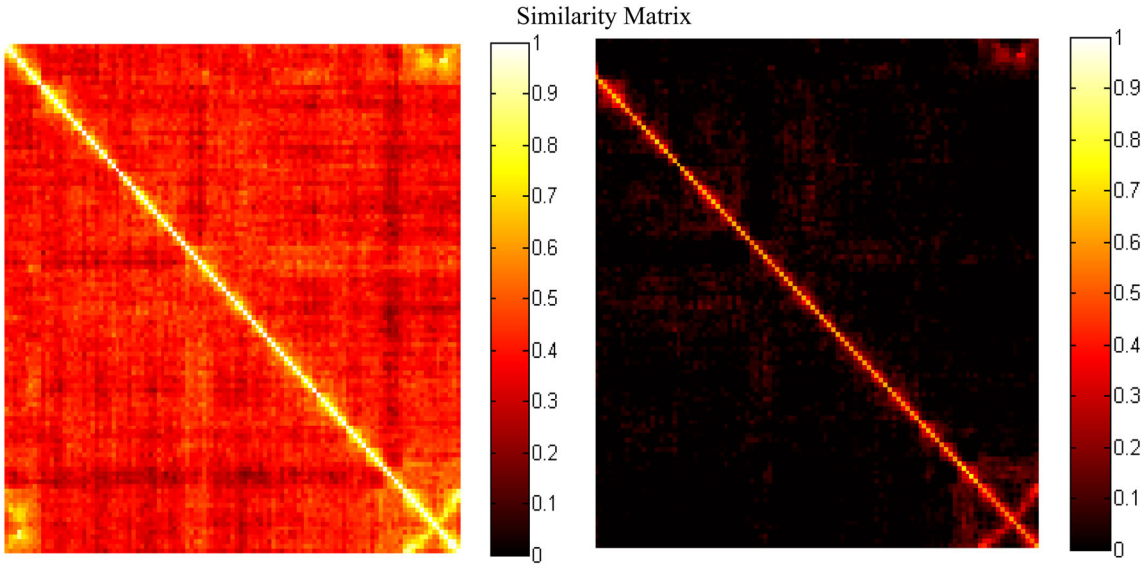


**Fig. 3** Similarity matrix of an example dataset. **a** Normalized similarity score. **b** Matrix after rank reduction

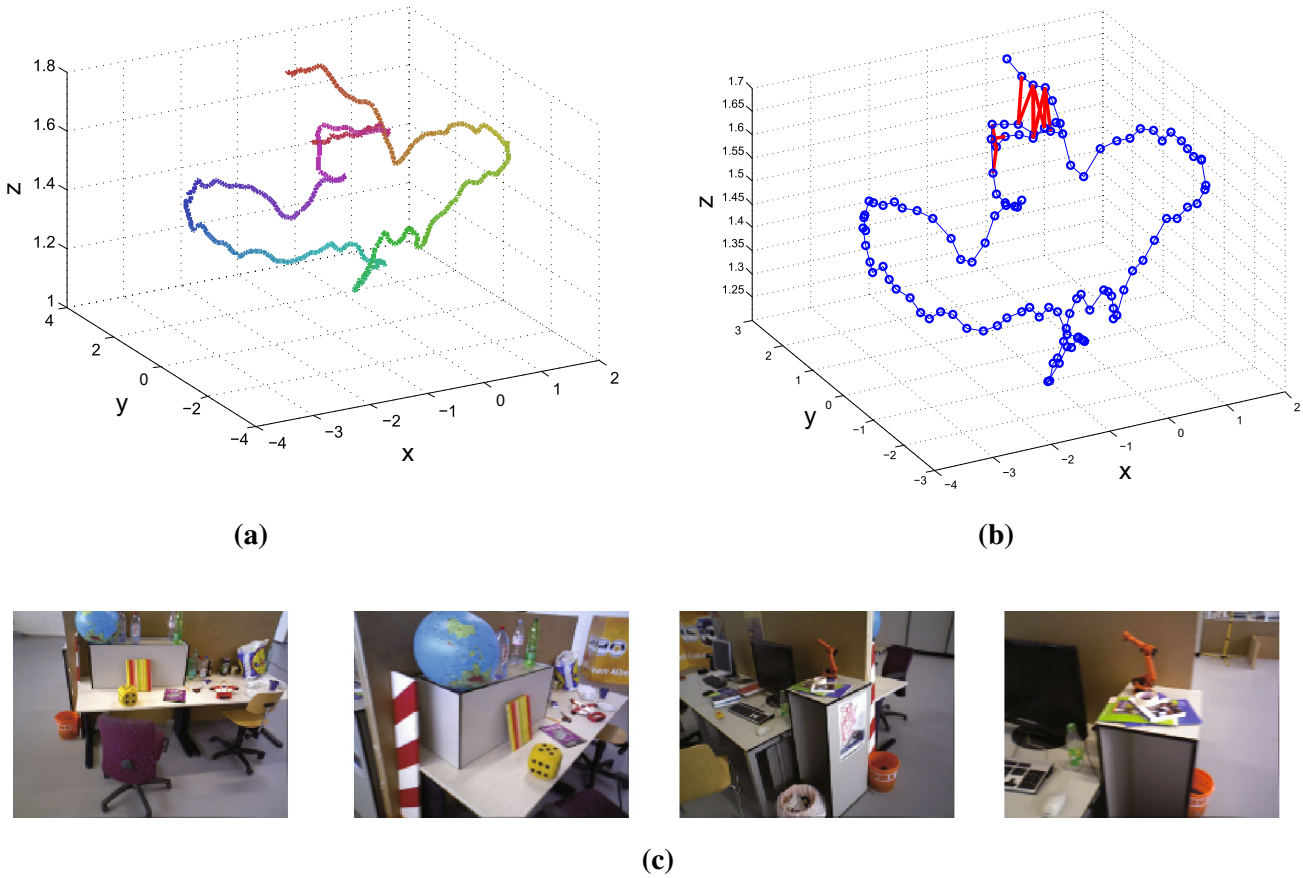


**Fig. 4** The trajectory and key-frames of the training data. **a** 3D view of the ground-truth trajectory. **b** The extracted 111 key-frames. The ground-truth loop closure is denoted as *red lines*; **c** Randomly selected sample images from the dataset

**Table 1** Hyper-parameters in SDA

| Parameter | Symbol | Default value |
|---|---|---|
| Learning rate | $\eta$ | 0.1 |
| Units in each layer | $N_h$ | 2500 |
| Corruption level | $c$ | 0.3 |
| Sparse level | $s_h$ | 0.05 |
| Sparse penalty | $\beta$ | 1.0 |
| Consecutive frames | $N_b$ | 5 |
| Consecutive penalty | $\gamma$ | 0.01 |
| Size of image patch | $s_I$ | $40 \times 40$ |
| Training epochs | $N_T$ | 100 |
| Distinctive weight function | $\mu$ | 0.5 |
| Distinctive weight function | $\sigma$ | 0.2 |
| Score function | a | 10 |
| Score function | b | $-10$ |

$$D_{i,j} = dis\left(\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\right) + angle\left(\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\right). \tag{16}$$

The function $dis(\cdot)$ and $angle(\cdot)$ denotes the translation and rotation parts of the transform matrix $\boldsymbol{T}_i^{-1}\boldsymbol{T}_j$. If $D_{i,j}$ is small enough, which means that the position and heading of the camera are close, this key-frame pair is marked as a ground-truth loop. The red lines in Fig. 4d show the ground-truth loops in this sequence. The trajectory of physical robot is treated in the same way. The ground-truth loops will be used to compute the precision-recall curve of algorithms.

### *4.1.2 Trained structure and loops*

The images from the open dataset are fed to SDA according to the schedule described in Fig. 2. The neural network is implemented using Theano library (Bastien et al. 2012; Bergstra et al. 2010). The default values of parameters are shown in Table 1.

In SDA, the $\boldsymbol{W}$ matrix of the first layer is often regarded as the feature detector because the hidden units compute a dot product of $\boldsymbol{W}$ and $\boldsymbol{x}$ (see Eq. 1). The column dimension of $\boldsymbol{W}$ is same as $\boldsymbol{x}$, so it can be conveniently visualized as images with same size as input data. Figure 5 shows part of representative detectors. Because of the sparse constraint, many of the hidden units have a low average response. Only part of them are detecting useful information (Fig. 6).

The response of hidden units forms a nonlinear description of the image data. In the visualized figure, there are hole, edge and corner detectors which are learned during the training. Their output is regarded as features which are used for measuring the similarity of the input data. Figure 7 shows an example of comparing three key-frames. The $F_1$ and $F_3$ are selected from true loops while $F_2$ is a different one. In this figure, (a) is the overall average response of all hidden units. The sparse constraint makes the average response close to 0.05, and the distinctive score function (13) makes the units that have medium response take a higher weight.

In (d), we project the feature vectors of the three key-frames into 2D plane using PCA. The matches are shown in this figure as lines between the matched features. The width of such lines indicates the similarity score of them. The thick

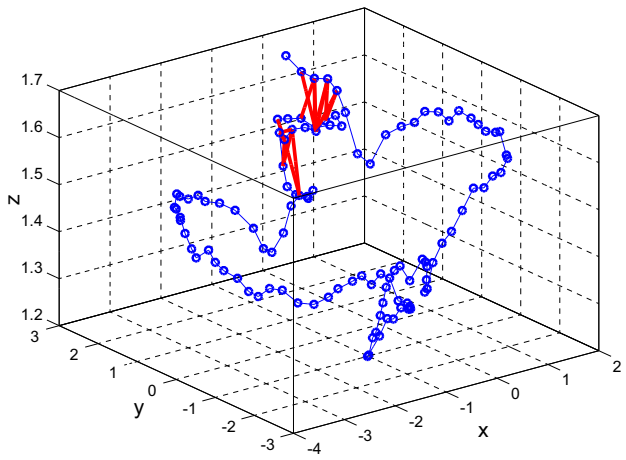


**Fig. 6** Loops detected by SDA with default parameter configuration

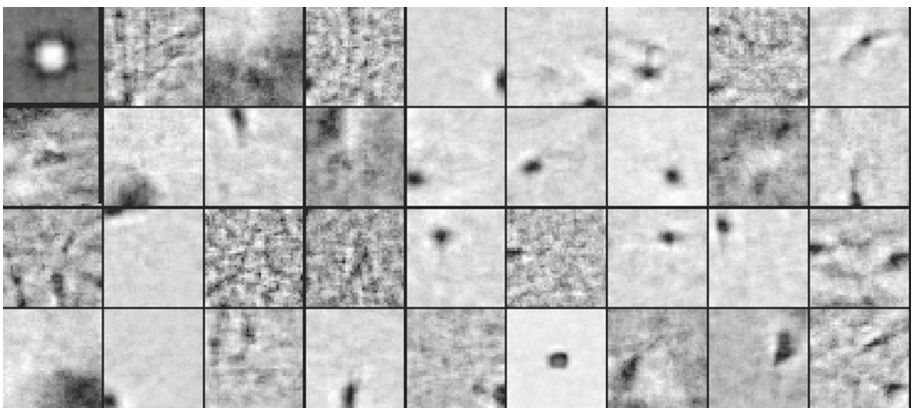

**Fig. 5** Representative feature detectors from SDA

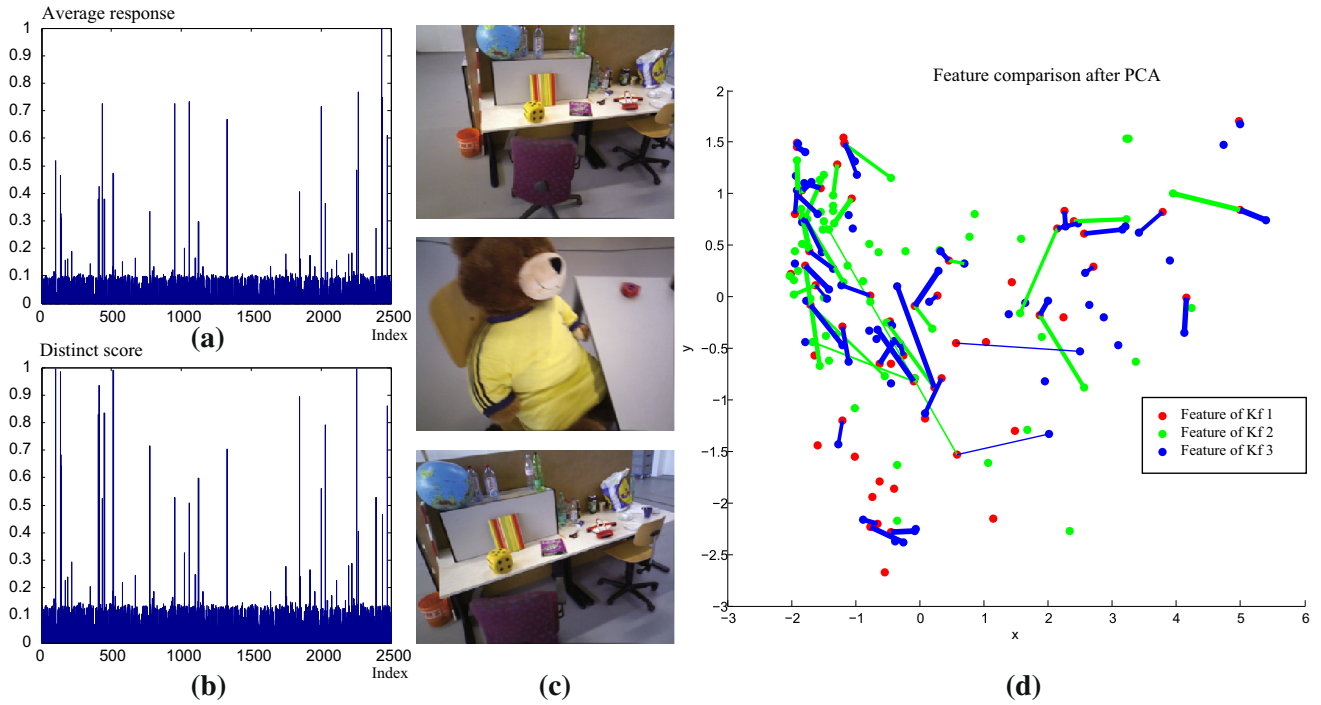


**Fig. 7** Comparing no loop key-frame pair $F_1$, $F_2$ and true loop key-frame pair $F_1$, $F_3$. **a** Average response of all 2500 units; **b** The distinctive score computed by Eq. 13; **c** three key-frames. The first one and the third one is selected from true loops, while the second one is observed from a different place; **d** the features of the three key-frames after PCA dimension reduction. The *lines* between features are match results, while the line width represent the similarity score of them. The total match of $F_1 - F_3$ and $F_1 - F_2$ are 43 and 20, respectively

lines show strong relationships of the features while the thin ones are weak. It can be seen that the number of thick blue lines are more than green lines, which means $F_1 - F_3$ are much more similar than $F_1 - F_2$.

The similarity matrix has been shown before in Fig. 3. The key-frames whose similarity exceeds a certain threshold are considered as possible loops. Figure 6 shows the detected loops in this experiment. Compared with Fig. 4d, the detected loops are almost same as the ground-truth loops. Hence, the feasibility of SDA-based loop closure detection is of satisfactory in this experiment.

To compare the results with Fab-map, we scan the similarity score threshold to obtain the precision-recall curve. Figures 8 and 9 shows the precision-recall curve of "fr3_office" and "fr2_slam" dataset, respectively. For Fab-map, we directly employ the Oxford surf 11k dictionary provided by FabMap V2 website. The output of Fabmap is the probability distribution about whether the current frame comes from the same place as the previous image. We scan the max possibility threshold while keep the other parameters as default values to obtain the precision-recall curve. From this figure we can see the SDA outperforms the traditional BoW model on the "fr3_office" dataset. For the other one, the situation is more complicated. The precision of Fab-map at low recall rate is higher than SDA, but when recall rate is larger than 0.5, the precision of SDA is better.

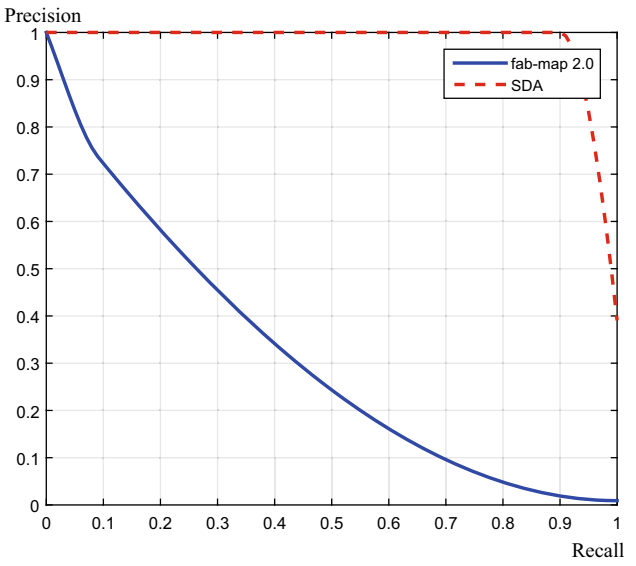


**Fig. 8** Precision-recall curve of SDA and Fab-map 2.0 (default parameter configurations) on dataset "fr3_office". The maximum recall rate at 100 % precision is 0.9 and 0.02

## 4.2 Fabmap dataset

To further evaluate our method, we perform another experiment using the "New College" and "City Centre" sequence provided by Fabmap. Each of these two datasets contains

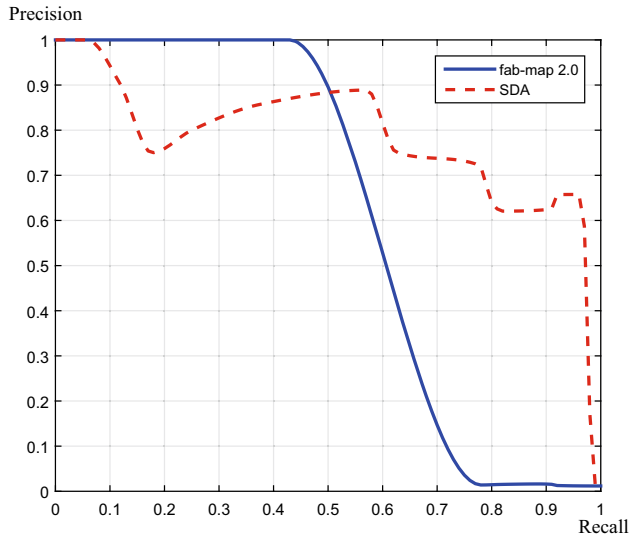


**Fig. 9** Precision-recall curve of SDA and Fab-map 2.0 (default parameter configurations) on dataset "fr2_slam". The maximum recall rate at 100 % precision is 0.44 and 0.07

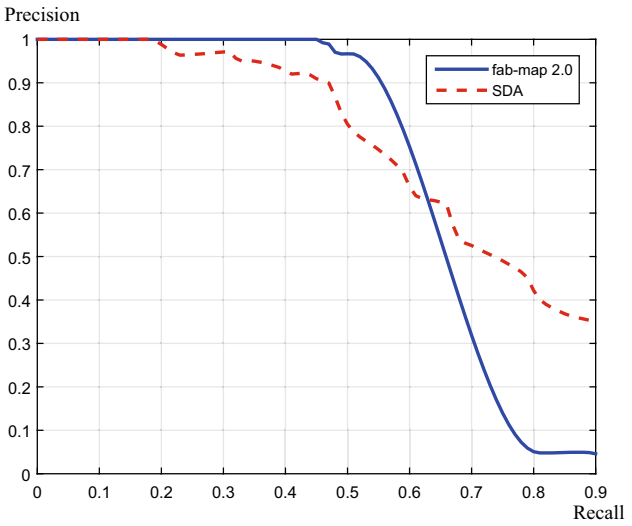


**Fig. 11** The precision-recall curve of Fabmap and SDA in "New College" sequence. The recall rate at 100 % precision is 0.45 and 0.2. The similarity score is 0.79 at 100 % precision

over 2000 images with ground-truth loops labelled manually. We uniformly select 240 key-frames among all frames. Since the pre-process and training step are similar with the previous experiment, we will directly give the results and discuss the advantages and disadvantages of SDA.

Figure 10 compares the visualized ground-truth loops with the estimated loop of fab-map and SDA. From the visualized images, we can see that both Fabmap and SDA can detect the most obvious loops. The difference is that Fabmap will give a near zero value if it considers that there is not a loop, so the figure contains many thin lines. Meanwhile, SDA will provides a bright block indicating that there may be a loop, so the nearby frames will also be considered as possible loops.

The precision-recall curves of these two datasets are shown in Figs. 11 and 12. Again, we have seen the SDA can achieve a higher precision at high recall rate, but in low recall rate area the Fab-map bas a better precision. This indicates that SDA will obtain more loops under the same condition, but it still needs a false rejection step if applied in real environments.

### 4.3 Physical experiment

We further test our approach on a physical mobile robot: Turtlebot 2.0. The robot is tele-operated along the corridor in our laboratory, capturing image data with the Microsoft Kinect equipped on its top board. The image sequence (7965 frames in total) is recorded at the video rate of 30 Hz, and then used for detecting loops via SDA. We set the key-frame distance threshold to 0.25 and obtain total 243 key-frames. The image is disturbed by humans and light conditions during

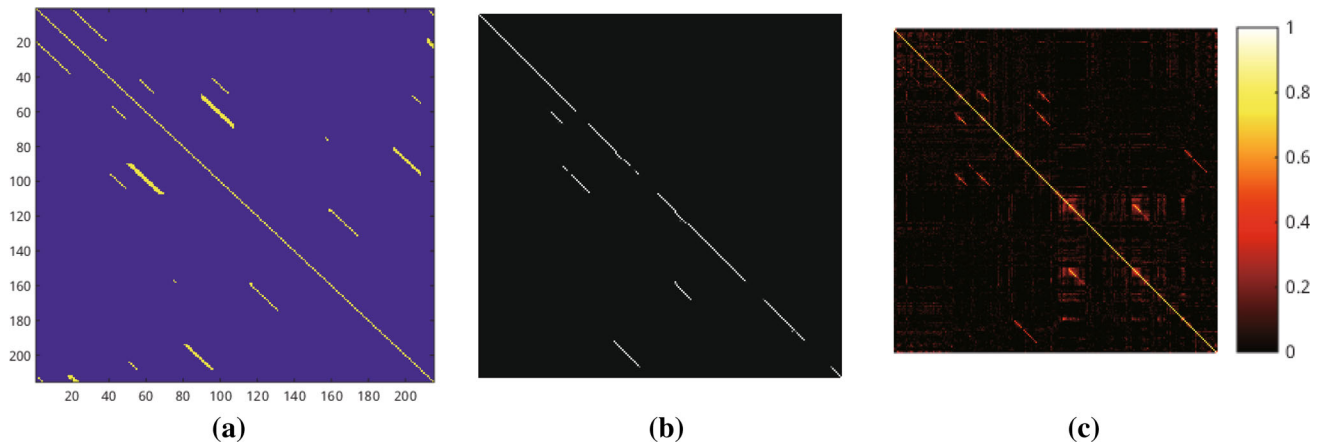


**Fig. 10** Similarity matrix. **a** The ground-truth loops of "New College"; **b** the probability distribution computed by Fab-map; **c** the rank-reduced similarity matrix of SDA

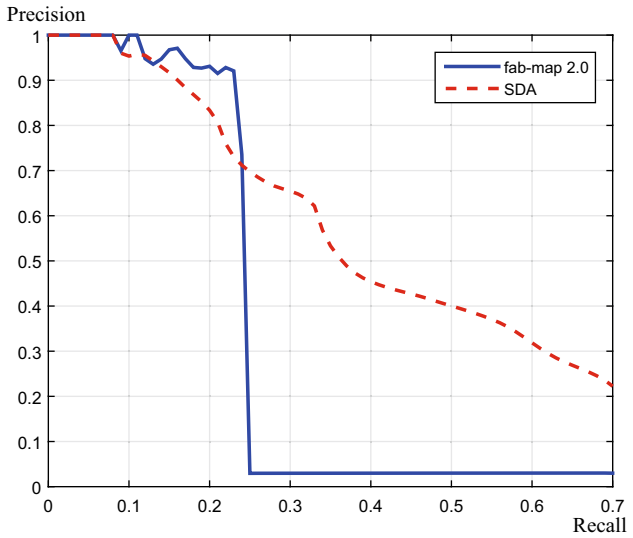


**Fig. 12** The precision-recall curve of Fabmap and SDA in "City Centre" sequence. The recall rate at 100 % precision is 0.07 and 0.06. The similarity score is 0.84 at 100 % precision

the travel. The data is also used in our previous work (Gao and Zhang 2015) to test an RGB-D SLAM system.

An overview of this experiment is shown in Fig. 13. The trajectory is recorded by the wheel encoder on the robot. The possible loops detected by SDA is shown in (b), where we can see that there are correct ones and false positives because the overall style of the lab is similar. There are many repetitive structures like long corridors and clapboard of each desk. Figure 13e is a true positive loop when the robot reaches the window at the second time (top left of (b)). Figure 13f is a false positive when the robot see the same white wall (long red lines in the right side of (b)). Finally, the precision-recall curve is shown in (c), where SDA has a better performance at high recall rates, or loose detection conditions. In practical SLAM systems, the possible loops provided by detection algorithms usually have post-processes like Conditional Random Fields (Latif et al. 2013; Cadena et al. 2012) to check their validity.

In a practical SLAM system, the loops will be periodically checked to refine the estimation of the current pose. In our system, we check the loops in a separate thread when a certain number of key-frames (usually 5) are added, or the robot find itself lost for a certain time.

## 5 Discussion

As many machine learning technology-based applications, the choice and values of hyper-parameters are of supreme importance to obtain a meaningful result. In this section, we will investigate the effect of some hyper-parameters listed in Table 1 by a series of comparison experiments, and also demonstrate the time cost of the training and detecting steps.

### 5.1 Learning rate and corruption

First, we study the effect of the learning rate $\eta$ and corruption level $c$, which controls the learning speed of SDA during the training and the mask rate of the input data. In this experiment, $\eta$ is set to $[0.01, 0.05, 0.1, 0.2, 0.3]$ and $c$ is set to $[0, 0.1, 0.3]$. We record the recovery error, feature detector and recovery images to see the difference.

Figure 14 shows the cost function $J^*$ in terms of training epochs with different learning rate $\eta$. A large $\eta$ will increase the step of the optimization, which means that decrease of cost in the first steps will be fast. But the result may oscillate because the optimization algorithm is unable to step into a more delicate structure of the cost function as the step of optimization is too large. Also, if $\eta$ is too small (like 0.01), the convergence may be too slow and algorithm need more epochs to train a good structure.

Figures 15 and 16 show the $\boldsymbol{W}$ matrix of the first layer and the recovery output with the same parameter configuration. The first row of Fig. 15 compares different corruption levels. The network learned from corrupted input seems to grab more useful and abstract information. The consequence that SDA can reduce the noise is also observed in this experiment. Besides, under the corruption level of 30%, the network is still able to reconstruct the original image, which is shown in the Fig. 16a–c.

On the hand, it also can be seen from these two figures that an inappropriate learning rate will make the training lost in failure. If $\eta$ is set to a very small value like 0.01, the network will not learn too much things in the 100 epochepochss training, so the $\boldsymbol{W}$ is almost gray. In contrast, if $\eta$ is too large, the parameters $\boldsymbol{W}$, $\boldsymbol{b}$ of the network will change a lot at the beginning of the training. So there will be many similar detectors, e.g., white dot in black background in Fig. 15e, and the data cannot be recovered well by these units. The results in Fig. 14 also show that the recovery error cannot converge when $\eta$ is too large.

### 5.2 The sparse penalty

In this experiment we investigate the effect the sparse constraint by setting the sparse factor to $\beta = 0.01, 0.1, 1, 10$, while other parameters are set to default values. The sparse cost (Eq. 9, average feature response, recovered images, feature detector are shown in Figs. 17, 18, 19, and 20, respectively.

First, consider about the sparse item:

$$J^* = c_d + \beta c_s + \gamma c_c$$

which is directly added to the cost function. A large $\beta$ value means that the sparse item occupies a more important position during the minimization. In Fig. 17, the black curve

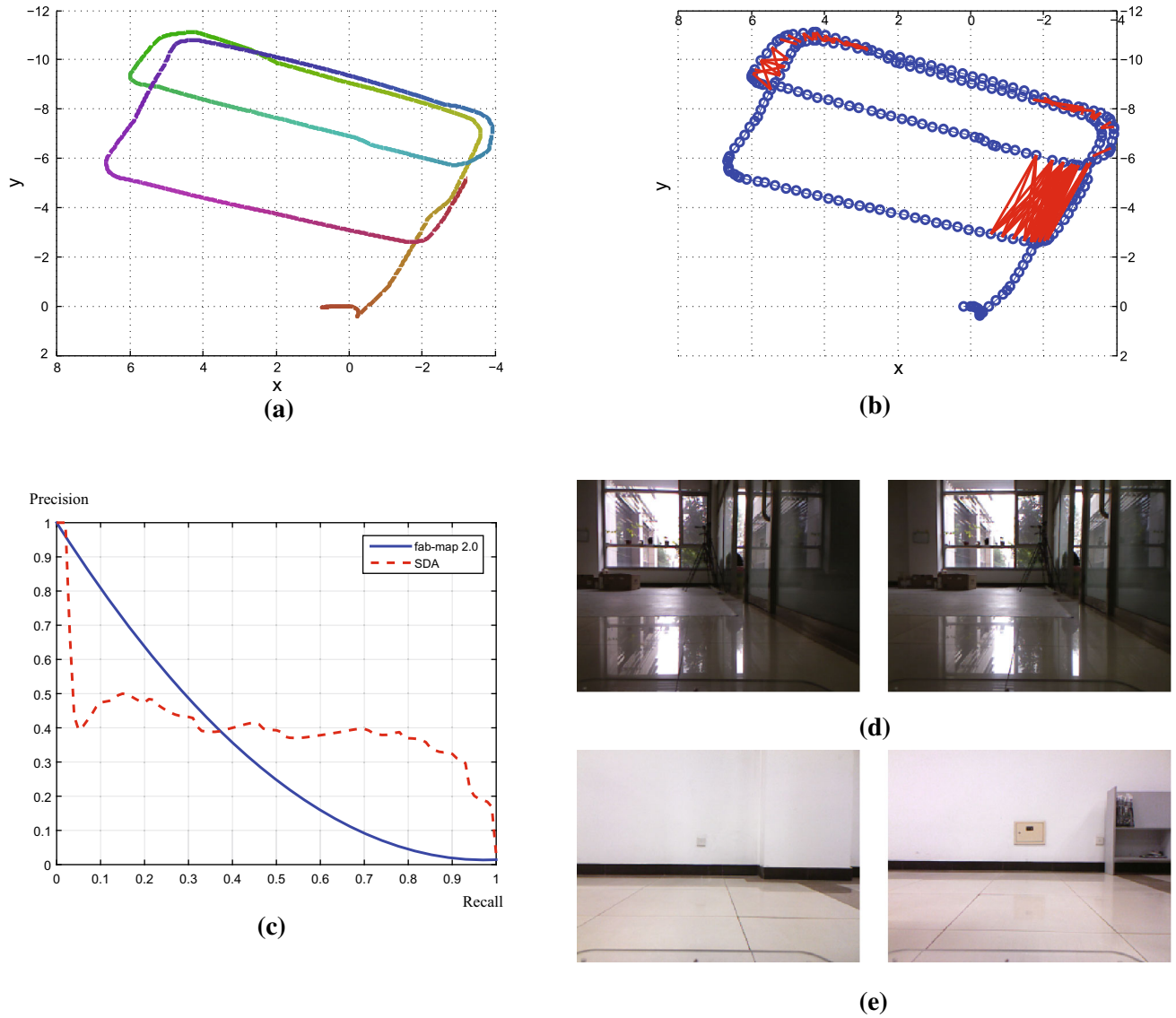


**Fig. 13** Result of the physical experiment. **a** Trajectory collected by the odometry. **b** Loops detected by SDA. **c** Precision-recall curve of Fab-map and SDA. **d** Correct loop. **e** False positive loop

converges quickly after few iterations, while the green curve ($\beta = 1$) converges at about 70 epochs. On the other hand, if $\beta$ is small (red and blue curves), the optimization algorithm will first minimize the recovery cost $c_d$ and then the sparse item $c_s$. This is the reason why they increase first and then decrease.

In Fig. 18, we record the response of all the input image patches (6660 patches in total) and calculate the average response of the hidden units in SDA. As $\beta$ grows larger, the feature response becomes more sparse. In Fig. 18c, only about 4.6 % (110 in 2500) units have an active response (larger than 0.1) and in (d) there are even fewer.

The sparse factor also affect the $\boldsymbol{W}$ matrix. Most unit are detecting gray noise with only a few of them contains useful information. In Fig. 20c, there are few similar patches, which means that the feature detectors are sparse and distinct at the same time. But in (d), the $\beta$ is so large that there are hardly any useful units. In this condition, the network cannot recover the input image at all.

The recovered images are shown in Fig. 19. In sub-figure (a–c), the recovered images remain almost same as in Fig. 16 because we train an over-complete structure (the dimension of hidden layer is larger than the input layer). A sparse structure is preferred because the hidden units are more distinct.

In conclusion, the experiment about sparse constraint shows that $\beta = 1$ may be a good value by comparing the $\boldsymbol{W}$ matrix and average response. A too large $\beta$ will lead to a failed result where the network is unable to reconstruct the input image.

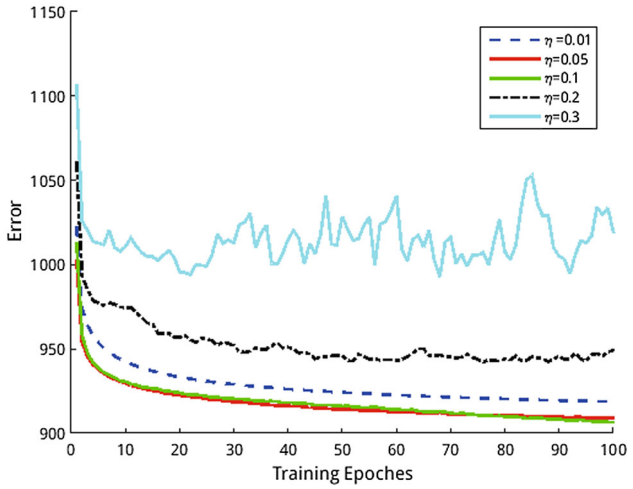


**Fig. 14** Recover error with different learning rate

### 5.3 Dimension of the hidden layer

In this experiment, we discuss how to set the size of the hidden layer. Researchers prefer to train an over-complete network in order to get sparse and meaningful results. It has been proved that if the dimension of hidden layer is smaller than input, the result of network is equal to an nonlinear Principle Component Analysis (PCA). In our approach, the dimension of input is 1600 (40 × 40 patches), so we set $N_h = 100, 500, 1000, 2500, 5000$ and then compare the cost and recovery output to see the difference.

Figures 21 and 22 show the result of the experiments described before. In Fig. 21, it can be seen that as the number of hidden units growing, the cost will also grow because the discrepancy between the input and hidden units is enlarged. Besides, the converge speed will decrease in large network because there are more units need to be trained. It is difficult to adjust all the parameters in one step in large networks. In the $N_h = 5000$ case, the cost is still decreasing after training 100 epochs. So in conclusion, consider the time cost and storage, training a large network will be more difficult than small ones. However, larger network has the ability to represent more complex structures which are needed in many applications.

Comparing the recovery image (Fig. 22) with original images (Fig. 16), we can see that the results does not change a lot when $N_h$ is larger than 500, because response of most units are nearly zero under the sparse constraint. In fact, the details of these images are still different if looking more care-

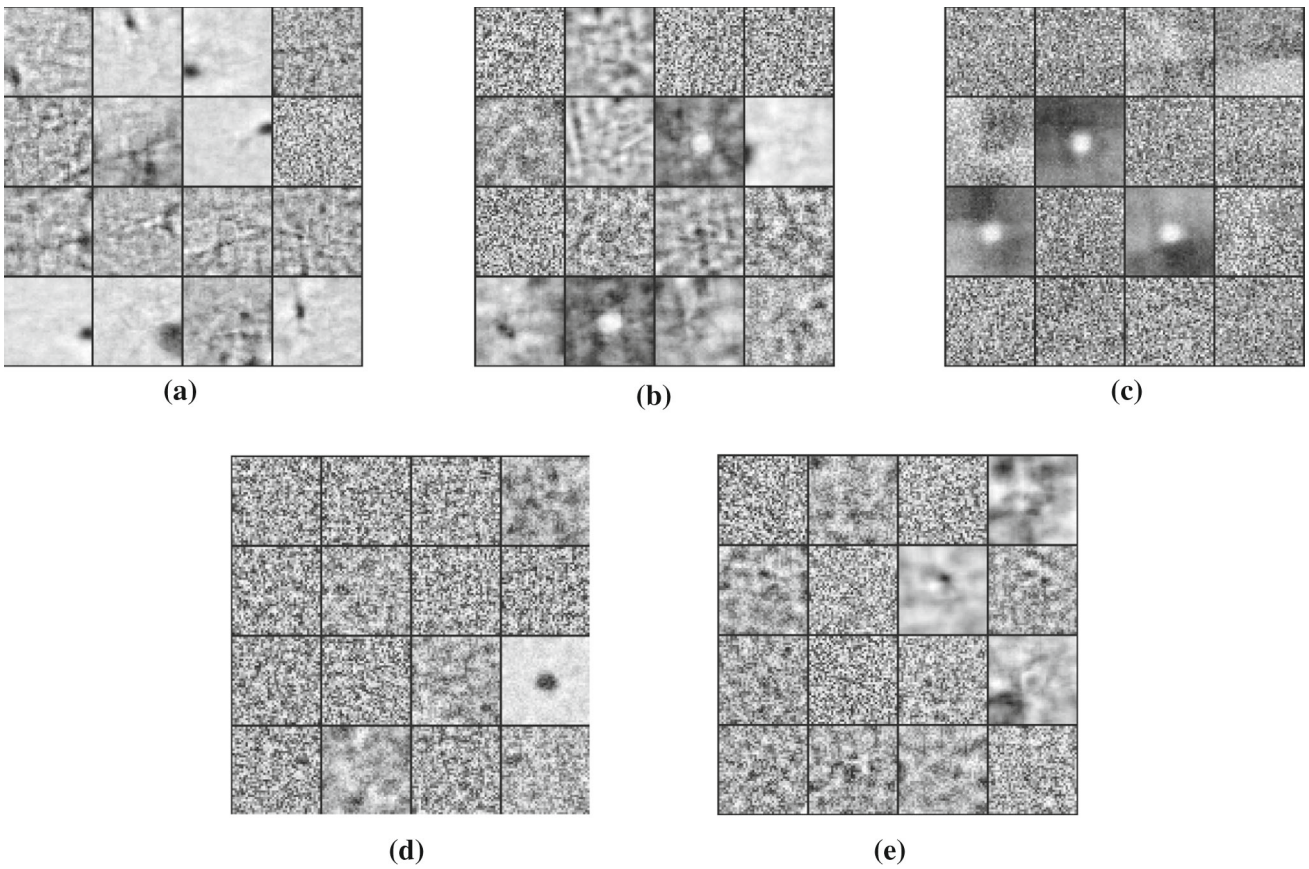


**Fig. 15** The $\boldsymbol{W}$ matrix of the first layer with different $c$, $\eta$ values. **a** $c = 0.3$, $\eta = 0.1$; **b** $c = 0.1$, $\eta = 0.1$; **c** $c = 0$, $\eta = 0.1$; **d** $c = 0.1$, $\eta = 0.01$; **e** $c = 0.1$, $\eta = 0.5$

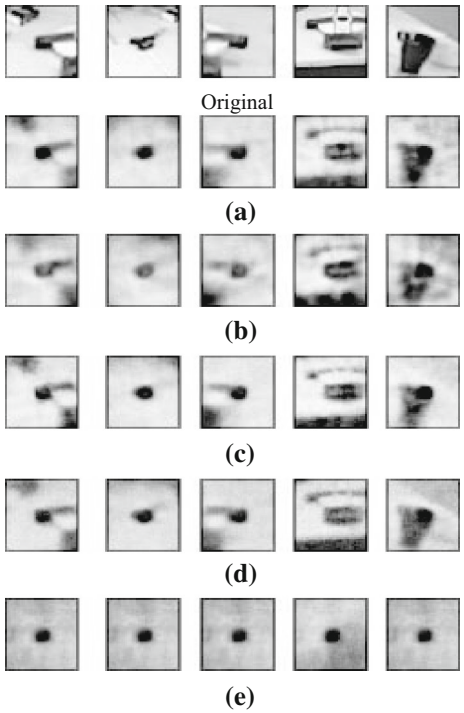


**Fig. 16** Recovered images. *First line* is the original image patches. **a** $c = 0.1, \eta = 0.1$; **b** $c = 0, \eta = 0.1$; **c** $c = 0.3, \eta = 0.1$; **d** $c = 0.1, \eta = 0.01$; **e** $c = 0.0, \eta = 0.5$

fully. For example, in the image of the fourth column where a plastic toy is lying on a desk, the 2500 nodes DA can recover the realistic small structure of the toy, but in 500 nodes case the center of image is almost black. Therefore, in this example, we can say that a larger network can represent more complex objects.

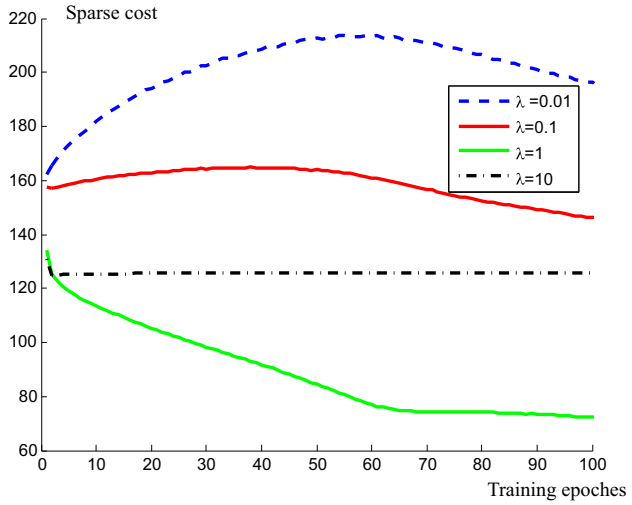


**Fig. 17** The change sparse cost item in terms of training epochs

### 5.4 Time cost

Finally, we record the training and detection time cost of our experiments. One of the drawbacks of deep neural network is the long training time. It sometimes take several hours to train a large network. In this application, the time cost is related to the size of input data, the hidden layer and the training epochs. As SDA is trained greedily layer by layer, the training time will be equal if the layer size is same. Therefore we just record

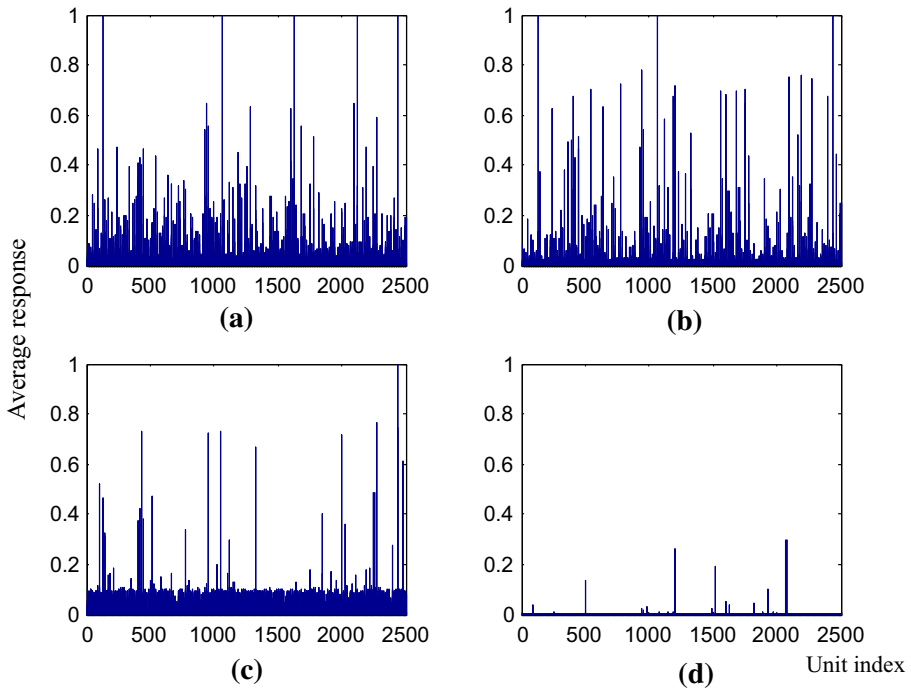


**Fig. 18** Average feature response with different penalty factors. **a** $\beta = 0.01$; **b** $\beta = 0.1$; **c** $\beta = 1$; **d** $\beta = 10$

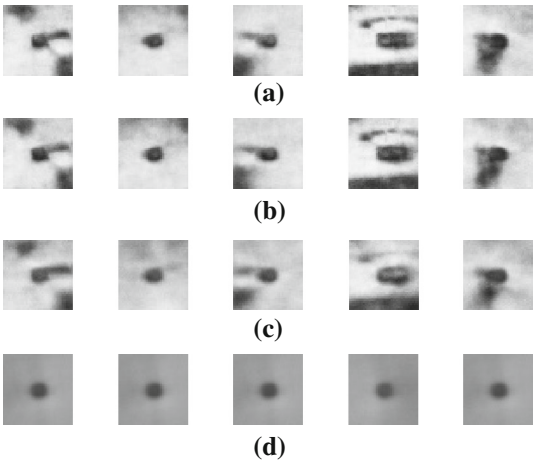


**Fig. 19** The recovered images. **a** $\beta = 0.01$; **b** $\beta = 0.1$; **c** $\beta = 1$; **d** $\beta = 10$. The original images are same as Fig. 16a

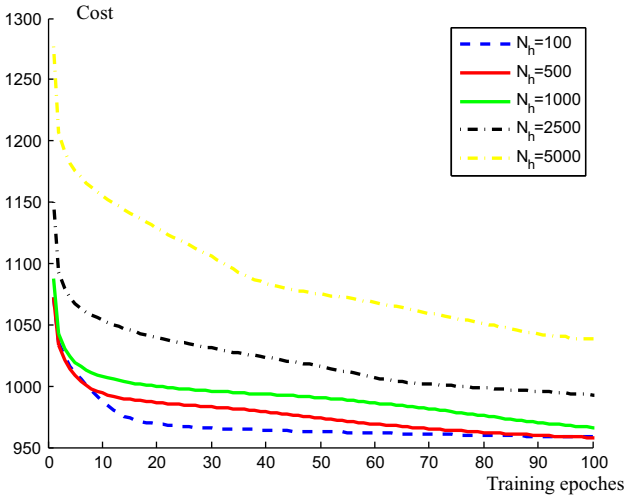


**Fig. 21** The cost curves with different hidden units

the time of training a single layer DA by 5 epochs (Table 2). The input samples are selected from "City Centre" dataset and are divided into 14,800 patches. Here the training time means the time used to adjust parameters, and the detection time means calculate the feature response of a input sample from a well-trained network. The calculation is done on a PC with a CPU of Intel core i7-4770 3.40G Hz, and accelerated by a GPU of GeForce GTX 750.

From Table 2, we can make a rough estimate about the training time of a network. Assume that we want to train 100 epochs on a five-layer network with 2500 nodes in each layer, and the input data contains 500 key-frames, each of which has 40 image patches, then it will take about 71.6 minutes to train the whole network, and the detection time of all frames will be about 2.2 seconds under the same hardware. Therefore, we will prefer to train the network offline and use the trained structure to detect the features.

## 6 Conclusions

This paper is concerned of the loop closure detection problem for visual SLAM systems. We propose a method that takes advantage of the stacked auto-encoder (SDA), a kind of well-studied deep neural network, to learn a nonlinear representation of the raw input data. The network is trained in the unsupervised way and represent the data by the response of hidden layers, which is used to compare the similarity of images. Finally, we use the similarity matrix to check the possible loops in the video sequence.

The feasibility of our method is tested on open datasets and a physical robot. The main merit of our approach is the ability of autonomously learning sparse feature representation, which no longer needs manually designed visual features. The learning-based approach can be applied to any sensors as long as the input data is a numeric vector. Therefore, our approach is not restricted by certain type of sensors. Besides, as deep network can obtain very complicated structures, the learned feature may reflect the inner patterns of the data. The traditional man-made feature descriptors may not be able to show that.

However, as mentioned before, the performance of deep network is sensitive to hyper-parameters like the learning rate, dimension of hidden layers, and other constraint factors.

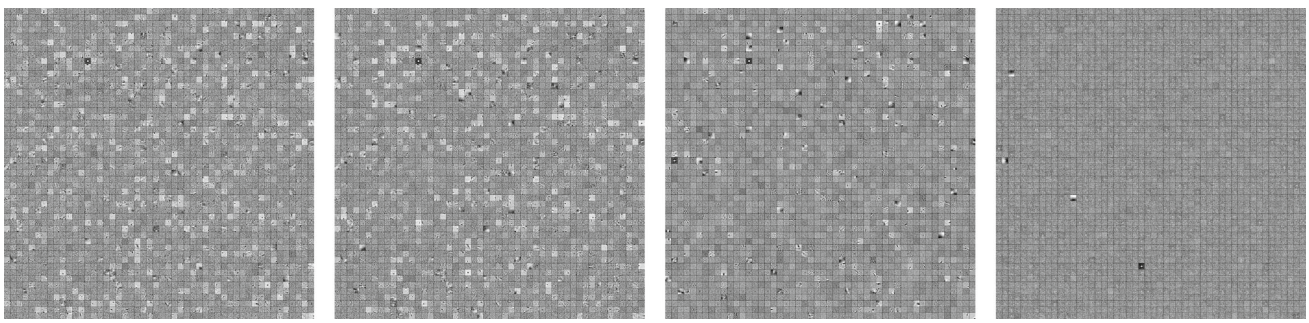


**Fig. 20** The $W$ matrix with sparse constraints. **a** $\beta = 0.01$; **b** $\beta = 0.1$; **c** $\beta = 1$; **d** $\beta = 10$

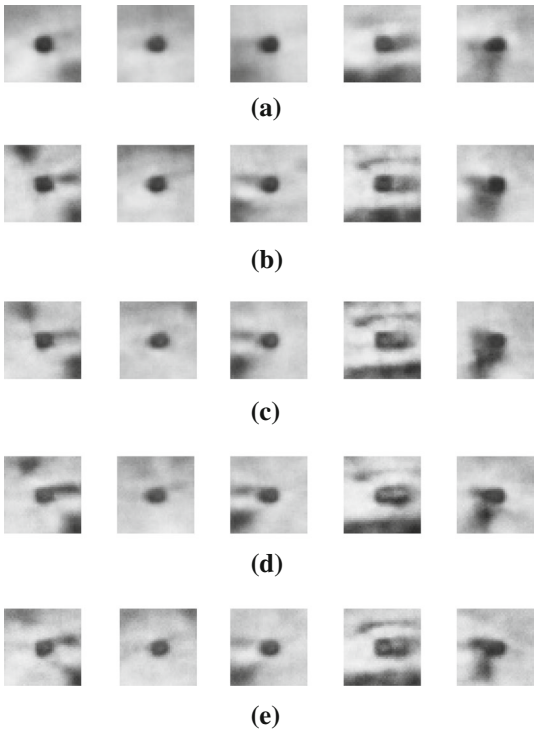


**Fig. 22** The recovered images. **a** $N_h = 100$; **b** $N_h = 500$; **c** $N_h = 1000$; **d** $N_h = 2500$; **e** $N_h = 5000$. The original images are same as Fig. 16a

**Table 2** The time cost of training and detection

| Nodes in hidden layer | Training time (s) | Detection time |
|---|---|---|
| 2500 | 31.77 | 0.327 |
| 1500 | 21.02 | 0.208 |
| 1000 | 16.12 | 0.174 |
| 500 | 10.9 | 0.16 |

Therefore we investigate various values of these parameters and even show some failed results during the training. The default values listed in Table 1, are used in the experiments, and can lead to satisfied results.

Finally, deep learning technologies has not been well understood nowadays. For example, we cannot clearly describe what the learned features are, how they change during the training, and how to select good parameters according to specific input data. It also takes a long time to train the whole network. Also, the proposed algorithm is an offline method because the SDA is trained on the same dataset in evaluation. We think SDA will obtain better results if trained on larger dataset and this will be remained as future work. There is still a long way to go to fully apply the deep networks into SLAM systems, and we wish to further investigate this idea in future research.

## References


Agrawal, M., Konolige, K., & Blas, M. (2008). Censure: Center surround extremas for realtime feature detection and matching. In D. Forsyth, P. Torr, & A. Zisserman (Eds.), *Computer vision–ECCV 2008. Lecture Notes in Computer Science* (Vol. 5305, pp. 102–115). Berlin Heidelberg: Springer.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., & Bengio, Y. (2012). Theano: New features and speed improvements, *arXiv preprintarXiv:1211.5590*.

Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In *Computer Vision–ECCV 2006* (pp. 404–417). New York: Springer.

Beeson, P., Modayil, J., & Kuipers, B. (2010). Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy. *International Journal of Robotics Research, 29*,(4), 428–459. Times Cited: 16 Beeson, Patrick Modayil, Joseph Kuipers, Benjamin 16.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., & Bengio, Y. (2010). Theano: A CPU and GPU math expression compiler. In *Proceedings of the python for scientific computing conference (SciPy)*, Oral Presentation.

Boal, J., Sánchez-Miralles, Á., & Arranz, Á. (2014). Topological simultaneous localization and mapping: A survey. *Robotica*, *32*, 803–821.

Bo, L., Ren, X., & Fox, D. (2014). Learning hierarchical sparse features for RGB-D object recognition. *International Journal of Robotics Research*, *33*(4), 581–599.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, *59*(4–5), 291–294.

Bradski, G. (2000). The opencv library. *Doctor Dobbs Journal*, *25*(11), 120–126.

Cadena, C., Galvez-Lopez, D., Tardos, J. D., & Neira, J. (2012). Robust place recognition with stereo sequences. *IEEE Transactions on Robotics*, *28*(4), 871–885.

Chen, Z., Samarabandu, J., & Rodrigo, R. (2007). Recent advances in simultaneous localization and map-building using computer vision. *Advanced Robotics*, *21*(3–4), 233–265.

Cummins, M., & Newman, P. (2008). Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, *27*(6), 647–665.

Cummins, M., & Newman, P. (2011). Appearance-only slam at large scale with fab-map 2.0. *International Journal of Robotics Research*, *30*(9), 1100–1123.

Davison, A., Reid, I., Molton, N., & Stasse, O. (2007). Monoslam: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(6), 1052–1067.

de la Puente, P., & Rodriguez-Losada, D. (2014). Feature based graph-slam in structured environments. *Autonomous Robots*, *37*(3), 243–260.

Deng, L., Li, J., Huang, J.-T., Yao, K., Yu, D., Seide, F., Seltzer, M., Zweig, G., He, X., & Williams, J. et al. (2013). Recent advances in deep learning for speech research at microsoft. In *2013 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 8604–8608), IEEE.

Dudek, G., & Jugessur, D. (2000). Robust place recognition using local appearance based methods. In *Proceedings of ICRA'00 IEEE international conference on robotics and automation* (Vol. 2, pp. 1030–1035), IEEE.

Endres, F., Hess, J., Sturm, J., Cremers, D., & Burgard, W. (2014). 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, *30*(1), 177–187.

Filliat, D. (2007). A visual bag of words method for interactive qualitative localization and mapping. In *2007 IEEE international conference on robotics and automation (ICRA)* (pp. 3921–3926), IEEE.

Gao, X., & Zhang, T. (2015). Loop closure detection for visual slam systems using deep neural networks. In *The 34th Chinese control conference, (Hangzhou, Zhejiang Province), technical committee on control theory (TCCT) of Chinese Association of Automation (CAA)*. Accepted July 2015.

Gao, X., & Zhang, T. (2015). Robust rgb-d simultaneous localization and mapping using planar point features. *Robotics and Autonomous Systems*, *72*, 1–14.

Gil, A., Mozos, O. M., Ballesta, M., & Reinoso, O. (2010). A comparative evaluation of interest point detectors and local descriptors for visual slam. *Machine Vision and Applications*, *21*(6), 905–920.

Hahnel, D., Burgard, W., Fox, D., & Thrun, S. (2003). An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Proceedings of 2003 IEEE/RSJ international conference on intelligent robots and systems, (IROS 2003)* (Vol. 1, pp. 206–211), IEEE.

Henry, P., Krainin, M., Herbst, E., Ren, X., & Fox, D. (2012). Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, *31*(5), 647–663.

Ho, K., & Newman, P. (2007). Detecting loop closure with scene sequences. *International Journal of Computer Vision*, *74*(3), 261–286.

Konolige, K., & Agrawal, M. (2008). Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, *24*(5), 1066–1077.

Kostavelis, I., & Gasteratos, A. (2013). Learning spatially semantic representations for cognitive robot navigation. *Robotics and Autonomous Systems*, *61*(12), 1460–1475.

Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K., & Burgard, W. (2011). G2o: A general framework for graph optimization. In *IEEE international conference on robotics and automation (ICRA)* (pp. 3607–3613), IEEE.

Kwon, H., Yousef, K. M. A., & Kak, A. C. (2013). Building 3d visual maps of interior space with a new hierarchical sensor fusion architecture. *Robotics and Autonomous Systems*, *61*(8), 749–767.

Labbe, M., & Michaud, F. (2013). Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, *29*(3), 734–745.

Latif, Y., Cadena, C., & Neira, J. (2013). Robust loop closing over time for pose graph slam. *The International Journal of Robotics Research*, *32*(14), 1611–1626.

Lepetit, V., & Fua, P. (2006). Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(9), 1465–1479.

Liou, C.-Y., Cheng, W.-C., Liou, J.-W., & Liou, D.-R. (2014). Autoencoder for words. *Neurocomputing*, *139*, 84–96.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110.

Lu, X., Tsao, Y., Matsuda, S., & Hori, C. (2013). Speech enhancement based on deep denoising autoencoder. In *INTERSPEECH* (pp. 436–440).

Mesnil, G., Dauphin, Y., Glorot, X., Rifai, S., Bengio, Y., Goodfellow, I. J., et al. (2012). Unsupervised and transfer learning challenge: A deep learning approach. *ICML Unsupervised and Transfer Learning*, *27*, 97–110.

Morell-Gimenez, V., Saval-Calvo, M., Azorin-Lopez, J., Garcia-Rodriguez, J., Cazorla, M., Orts-Escolano, S., et al. (2014). A comparative study of registration methods for RGB-D video of static scenes. *Sensors*, *14*(5), 8547–8576.

Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP*, *1*, 331–340.

Ng, A. (2011). Sparse autoencoder. *CS294A Lecture notes* (Vol. 72).

Poultney, C., Chopra, S., & Cun, Y. L. et al. (2006). Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems* (pp. 1137–1144).

Ren, X., Bo, L., & Fox, D. (2012). RGB-(D) scene labeling: Features and algorithms. In *2012 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 2759–2766), IEEE.

Rosten, E., & Drummond, T. (2006). Machine learning for high-speed corner detection. In *Computer vision–ECCV 2006* (pp. 430–443). New York: Springer.

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *2011 IEEE international conference on computer vision (ICCV)* (pp. 2564–2571), IEEE.

Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., & Davison, A. J. (2013). Slam++: Simultaneous localisation and mapping at the level of objects. *2013 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 1352–1359).

Shi, Z., Liu, Z., Wu, X., & Xu, W. (2013). Feature selection for reliable data association in visual slam. *Machine Vision and Applications*, *24*(4), 667–682.

Strasdat, H., Montiel, J. M., & Davison, A. J. (2012). Visual slam: Why filter? *Image and Vision Computing*, *30*(2), 65–77.

Stuckler, J., & Behnke, S. (2014). Multi-resolution surfel maps for efficient dense 3d modeling and tracking. *Journal of Visual Communication and Image Representation*, *25*(1), 137–147.

Sturm, J., Engelhard, N., Endres, F., Burgard, W., & Cremers, D. (2012). A benchmark for the evaluation of rgb-d SLAM systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 573–580), IEEE.

Tian, B., Shim, V. A., Yuan, M., Srinivasan, C., Tang, H., & Li, H. (2013). Rgb-d based cognitive map building and navigation. In *2013 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 1562–1567), IEEE.

Ulrich, I., & Nourbakhsh, I. (2000). Appearance-based place recognition for topological localization In *Proceedings of ICRA'00 IEEE international conference on robotics and automation* (Vol. 2, pp. 1023–1029), IEEE.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103), ACM.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, *11*, 3371–3408.

Wang, N., & Yeung, D.-Y. (2013). Learning a deep compact image representation for visual tracking. In *Advances in neural information processing systems* (pp. 809–817).

Wang, Y.-T., & Lin, G.-Y. (2014). Improvement of speeded-up robust features for robot visual simultaneous localization and mapping. *Robotica*, *32*, 533–549.

Williams, B., Klein, G., & Reid, I. (2011). Automatic relocalization and loop closing for real-time monocular SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(9), 1699–1712.

**Xiang Gao** received B.E. Degree in Automatic Control Science and Engineering from Tsinghua University, Beijing, China, in 2012 and is currently persuing the Ph.D. Degree in Control Science and Engineering from Tsinghua University. His research interests include computer vision, simultaneous localization and mapping, robotics.

**Tao Zhang** received the Ph.D. Degree in Control Science and Engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. Degree in Electrical Engineering from Saga University, Saga, Japan, in 2002. He became an Associate Professor with Saga University in 2002 and a Research Scientist in the National Institute of Informatics, tokyo, Japan, in 2003. In 2006, he became an Associate Professor in the Department of Automation, Tsinghua University. His current research interests include pattern recognition, nonlinear system control, robotics, control engineering and artificial intelligent.