Proceedings of the 34th Chinese Control Conference
July 28-30, 2015, Hangzhou, China

Loop Closure Detection for Visual SLAM Systems Using Deep Neural Networks

Xiang Gao [1], Tao Zhang [1]
1. Department of Automation, Tsinghua University, Beijing, 100084, China
E-mail: gaoxiang12@mails.tsinghua.edu.cn
E-mail: taozhang@mails.tsinghua.edu.cn

Abstract: The detection of loop closure is of essential importance in visual simultaneous localization and mapping systems. It can reduce the accumulating drift of localization algorithms if the loops are checked correctly. Traditional loop closure detection approaches take advantage of Bag-of-Words model, which clusters the feature descriptors as words and measures the similarity between the observations in the word space. However, the features are usually designed artificially and may not be suitable for data from new-coming sensors. In this paper a novel loop closure detection approach is proposed that learns features from raw data using deep neural networks instead of common visual features. We discuss the details of the method of training neural networks. Experiments on an open dataset are also demonstrated to evaluate the performance of the proposed method. It can be seen that the neural network is feasible to solve this problem.

Key Words: Simultaneous Localization and Mapping, Loop Closure Detection, Deep Neural Networks, Denoising Auto-encoder.

1 Introduction

In this paper we focus on the problem of robustly detecting loop closure for vision-based simultaneous localization and mapping (SLAM) systems. The detection of loop closure is a crucial part in for mobile robots based on various sensors [1, 2]. The ability of recognizing previously visited places can benefit a lot for SLAM algorithms [3]. It can significantly reduce the accumulating drift and make the filters or pose graph more stable [4]. For instance, the Extended Kalman Filters (EKFs) rely on correct data associations, which can be seen as a kind of loops constructed on landmarks [5]. For pose graph, the constraints brought by loop closure detection algorithms, if correct, can also help the optimization algorithm find a better result [6].

The keypoint of detecting loops is to define the similarity between the observations. For visual systems where the observations are images, most of state-of-the-art algorithms take advantage of the Bag-of-Words (BoW) model [7–9]. The BoW model clusters the visual feature descriptors in images, builds the dictionary, and then finds the corresponding words of each image. The visual features like SIFT [10], Surf [11] are commonly used in these systems and have achieved great success in the past years.

However, what remains difficult in detecting loops is the fact that the incorrectly detected loops may greatly corrupt the estimated map and lead to an totally erroneous result. The "incorrect loops" fall into two categories: (1) Perceptual aliasing, or false positive, which means that different scenes look similar and are treated as loops. (2) Perceptual variability, or false negative, which means that the same scene looks different and is not detected as loops [12]. A good loop closure detection algorithm should distinguish those situations and provide most true positive results. Generally speaking, algorithms based on BoW have successfully treated the perceptual aliasing problem, because the visual features designed by experts are usually different in different scenes. On the other hand, the perceptual variability problem has not been handled well since images from the same scene may change according to the light conditions or the movements of the objects, therefore having totally different features.

The recent development of the deep learning technology [13], whose purpose is to learn a representation from the original data that can be used for classification, brings a new way for the classic loop closure detection problem. In the area of SLAM research, the deep learning has not been fully understood and applied except for few recognization works [14, 15]. In most applications of deep learning like [16], the features are trained from a multi-layer neural network and then used to predict the labels of data. By the same way, if we carefully train a structure from the data of robot sensors, it can tell whether a new-coming observation is from the same or different scenes. Since internal assistance like odometry is not used, we do not have *a prior* knowledge about whether the loops are likely to occur, which means that the structure should be trained in an unsupervised way without extra information of labels.

A method that employs deep neural networks to detect loops is proposed in this paper, which can learn features from raw observation data, i.e., the pixels of images, instead of using artificially designed features. The learned features are used to construct a representation of the scenes. We define the similarity score based on the data representation and introduce the algorithm of checking loops. Finally, a series of experiments are demonstrated, including an evaluation of our approach on an open dataset [17].

2 Proposed Method

The proposed method contains two parts: (1) The feature learning process, in which we train a stacked auto-encoder [18] to learn a feature representation. (2) The loop detecting algorithm, in which we find loops in a similarity/difference matrix. These two parts will be discussed in this section. We firstly give a brief introduction of the auto-encoder, which is an unsupervised training method of machine learning.

2.1 Stacked auto-encoder

The auto-encoder is a kind of neural network, which contains three layers of units (see Fig. 1): (1)The first layer is the

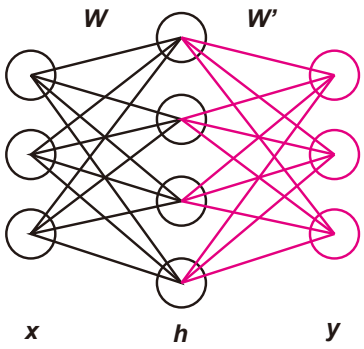

Fig. 1: The auto-encoder.

input layer that directly connects to the sensor data, which is denoted as $\boldsymbol{x}$. (2) The second hidden layer $\boldsymbol{h}$. (3) The output layer $\boldsymbol{y}$. Each unit in the network calculates a sigmoid response a of the front layer with a weight vector $\boldsymbol{w}$ and a bias b:

$$a = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b) = \frac{1}{1 + \exp(-\sum_{j=1}^{n} w_j x_j - b)}. \quad (1)$$

By this way we construct a relationship between the layers:

$$\begin{array}{rcl} \boldsymbol{h} & = & f_\theta(\boldsymbol{x}) \\ \boldsymbol{y} & = & g_\theta(\boldsymbol{h}) = g_\theta f_\theta(\boldsymbol{x}) \end{array} \quad (2)$$

where θ is the parameters of the network, i.e. $\boldsymbol{w}$ and b. $f_\theta = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$ is a mapping function from the input layer to hidden layer, and $g_\theta = \sigma(\boldsymbol{w'}^T\boldsymbol{h} + b')$ is a function from the hidden layer to the output. The purpose of auto-encoder is to train the parameters in the function f, g so that they can recover the input data from the hidden layer:

$$\boldsymbol{y} = g_\theta f_\theta(\boldsymbol{x}) \approx \boldsymbol{x} \quad (3)$$

Since the dimension of $\boldsymbol{h}$ is different with $\boldsymbol{x}$ and the parameters are randomly initialized, we will not get an identical function like $g_\theta f_\theta = I$. The training process is modeled as an optimization problem whose object function is the distance of $\boldsymbol{x}$ and $\boldsymbol{y}$, which is usually defined as the cross entropy if the input $\boldsymbol{x} \in [0, 1)$:

$$d = KL(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} x_i log\frac{y_i}{x_i} + (1 - x_i) log\frac{1 - x_i}{1 - y_i} \quad (4)$$

The weight and bias are parameters to be trained in order to get such a function:

$$(\boldsymbol{W^*}, b^*) = (\boldsymbol{W}, b) - \eta(\frac{\partial d}{\partial \boldsymbol{W}}, \frac{\partial d}{\partial b}) \quad (5)$$

where η is the learning rate. The training process is done by applying stochastic gradient descent (SGD), which divide the training data into small batches and update the $\boldsymbol{W}, \boldsymbol{b}$ in each batch. After some training epoches, the parameters will converge to a local minima. The responses of hidden units are regarded as features. We then discard the decoder part of this layer, i.e., the red part in Fig. 1 and only keep the encoders. If several auto-encoders are stacked together, we train them layer-wise and call them the stacked auto-encoder.

2.2 Training visual features

In order to learn meaningful features for loop closure detection we put the images into a stacked auto-encoder. The overview of the training process is shown in Fig. 2. Since the data is acquired by depth camera (Microsoft Kinect) , the input of network is raw pixel values of RGB and depth. The images are divided into small patches with size of 10×10, and then vectorized so that the input of the neural network is a set of vectors: $\{\boldsymbol{x} | \boldsymbol{x} \in R^{100}\}$.

We modify the object function of the auto-encoder in order to get better results. The modification comes from three aspects:

Denoising: The real input data is usually affected by unwanted noise. So we intentionally add a corruption into the input data $\boldsymbol{x}$, which randomly mask a certain percent of input to zero. The original input $\boldsymbol{x}$ is corrupted into $\widetilde{\boldsymbol{x}}$. By setting the object function of comparing the original input and the corrupted one, we wish the optimization algorithm find the best way to recover the data from the corrupted input:

$$d = KL(\boldsymbol{x}, g_\theta f_\theta(\widetilde{\boldsymbol{x}})) \quad (6)$$

Sparsity: The network is usually trained in an over-complete way in applications of deep learning, which means the dimension of hidden variable $\boldsymbol{h}$ is larger than the input $\boldsymbol{x}$. Otherwise, the network will be only a dimensionality reduction method like the Principle Component Analysis (PCA) in the linear case. Therefore, we train a over-complete and sparse structure in our approach by penalizing the hidden units:

$$c_s = \sum_{i=1}^{H} \|h_i - s_h\|_1 \quad (7)$$

where s_h is a threshold of sparsity. Since the $\mathcal{L}_0$ norm is not derivable, the constraint is relaxed into $\mathcal{L}_1$ norm, which means that the mean response of hidden variables will be converge to s_h during the training process.

Continuity: The denoising and sparsity parts are likely to occur at many applications of auto-encoder, but another assumption can be made in a SLAM system. It is the continuity in the movement of the robot. Consider there are k frames in a batch of SGD, like $\{\boldsymbol{x}_i, i = 1, \ldots, k\}$. If k is not too large, we can assume that the movement in adjacent frames is not large so that the observations and features, which are represented by $\boldsymbol{h}_i, i = 1, \ldots, k$, are similar. We add a roll operation on $\boldsymbol{h}_i$:

$$\begin{array}{rcl} \boldsymbol{H} & = & \{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_{k-1}, \boldsymbol{h}_k\} \\ \widetilde{\boldsymbol{H}} & = & \{\boldsymbol{h}_2, \boldsymbol{h}_3, \ldots, \boldsymbol{h}_k, \boldsymbol{h}_1\} \end{array} \quad (8)$$

and define the average continuity cost:

$$c_c = \frac{1}{k} \sum \left\| H - \widetilde{H} \right\|_2. \quad (9)$$

In summary, we add three additional penalty items into the original auto-encoder. Now the object function is:

$$\begin{array}{rcl} J & = & d(\boldsymbol{x}, g_\theta f_\theta(\widetilde{\boldsymbol{x}})) + \lambda c_s + \gamma c_c \\ & = & \sum KL(\boldsymbol{x}, g_\theta f_\theta(\widetilde{\boldsymbol{x}})) + \lambda \sum_{i=1}^{n_h} \|h_i - s_h\|_1 \\ & & + \gamma \frac{1}{k} \sum \left\| H - \widetilde{H} \right\|_2 \end{array} \quad (10)$$

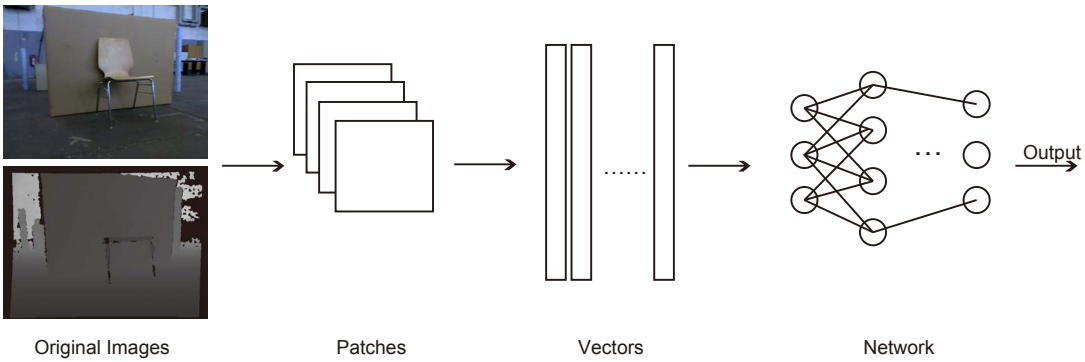

Fig. 2: Overview of the training process

where λ, γ are penalty weights defined to balance the effect of the items.

2.3 Detecting Loops

After training the network, we get the learned features which are represented as the response of hidden units. Since similar input will lead to the similar features, the difference of two arbitrary scenes (m, n) can be represented by their hidden response:

$$D(m,n) = \|\boldsymbol{h}_m - \boldsymbol{h}_n\|_1 \tag{11}$$

Inspired by the BoW model exploits a reverse-indexing mechanism, we add a weight $\boldsymbol{z}$ to each hidden unit. Because units that have high average response are not useful for distinguishing different scenes (like there is a large floor in each image), the weight is defined to reduce the effect of them:

$$z_i = \log \frac{h}{h_i}, h = \sum_{j=1}^{n_h} h_j \tag{12}$$

So

$$D(m,n) = \left\|\boldsymbol{z}^T(\boldsymbol{h}_m - \boldsymbol{h}_n)\right\|. \tag{13}$$

After comparing scene m with all other scenes, we need to normalize the measurement:

$$\bar{D}(m,\cdot) = \frac{D(m,\cdot)}{\max D(m,\cdot)} \tag{14}$$

By this definition, if an item like $\bar{D}(m,n)$ is large, we can say the two scene looks different and vise versa. An example of this difference matrix is illustrated in Fig.3. The diagonal part of the matrix contains a series of black boxes because the frames are taken from similar places. There are also two off-diagonal deep color lines indicating that the robot is moving along a previously visited trajectory. However, the matrix is not good enough for directly checking loops. There are several further steps need to be done.

2.4 Rank Reduction

Since $\bar{D}$ is a real symmetric matrix, it can be decomposed by the eigen-value decomposition:

$$\bar{D} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T = \sum_{i=1}^{N} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T \tag{15}$$

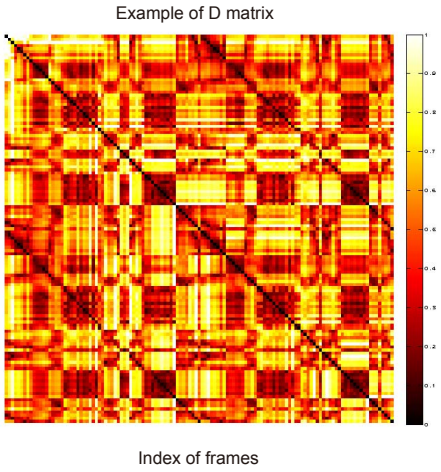

Fig. 3: An example of difference matrix

where $\boldsymbol{V}$ is eigen vectors and $\boldsymbol{\Lambda}$ is a diagonal matrix: $\boldsymbol{\Lambda} = diag(\lambda_1, \ldots, \lambda_N)$ with defending order $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$. According to [19], a rank reduction can be performed by discarding the large eigen values, which will reduce the affect of scene ambiguity:

$$D_R = \sum_{i=k}^{N} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T \tag{16}$$

Fig. 4 shows the difference matrix after removing five largest eigen values. It can be seen that the loop closure is more obvious than that in Fig. 3. This matrix is used for detecting loops in the experiments decribed in later sections.

3 Experiments

A set of experiments is demonstrated in this section to evaluate the performance of the proposed method. Our implementation is a python program based on Theano library [20]. The trained data is from the freiburg2_slam of the open dataset provided by Sturm et al. [17], which contains both RGB-D images and groundtruth trajectories. The dataset has total 2198 image files within a trajectory about 43.07 m. However, it does not have enough loops. So we extend it by adding an additional loop of all frames (see Fig. 5). The hyper-parameters used in the experiment are listed in Table 1.

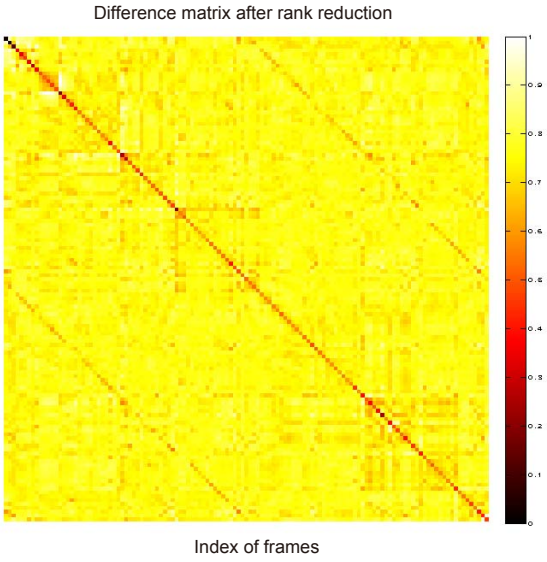

Fig. 4: Difference matrix after rank reduction.

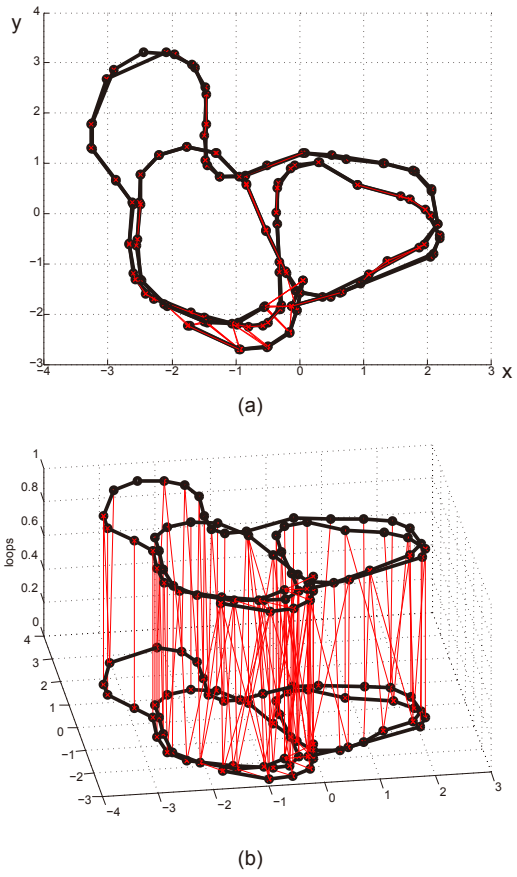

Fig. 5: Keyframes of the dataset. (a) Top view. (b) 3D view with z axis denoting the additional loop. The black dots are keyframes and red lines are groundtruth loops.

Table 1: Hyper-parameters used in the experiments

Name	Value	Description
n_{hidden}	[200,50,50]	The hidden units in each layer.
η	0.1	The learning rate.
λ	0.005	The penalty factor of sparsity.
γ	0.001	The penalty factor of continuity.
l_c	0.1	The corruption level.
s_h	0.05	The sparsity threshold.
r_{image}	240×180	The resolution of images.
n_{batch}	10	The batch size of SGD.

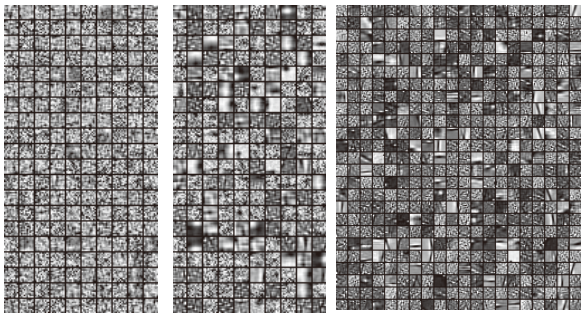

Fig. 6: Structure trained from the first hidden layer with different corruption level. (a) 0% corruption. (b) 1% corruption. (c) 10% corruption with 500 units.

3.1 Learned features

We first describe the features trained from the neural network. Fig.6 illustrates the weights $\boldsymbol{W}$ of the first layer hidden unit. Since the input of the network is image patches $I_{10\times10}$, the $\boldsymbol{W}$ is a matrix with the size of 200×100. Each row of $\boldsymbol{W}$, i.e., $w \in R^{1\times100}$, represents the weight of an unit. We visualize this matrix as a set of 10×10 gray-scale images. Fig. 6 (a) shows the weight matrix with zero corruption. This matrix is not trained well because we cannot see any meaningful images in it. The Fig. 6 (b) and (c) are trained with corruption and the weights of units seem as sparse edge detectors. Compared with BoW model, the weight matrix is like the dictionary and the response of hidden units are words, but in an nonlinear form.

Fig.7 shows an example of comparing images. We arbitrary select two pairs of image with one pair is different and another is similar. The figure shows that similar image generally have similar feature response, so $\|\boldsymbol{h}_i - \boldsymbol{h}_j\|_1$ can be used for measuring the difference of two frames.

An example of the recovered images $\boldsymbol{y}$ are shown in Fig. 8. If no corruption is added into the original data, the recovered image is just the same as the original one. However, in this case the weights do not show any meaningful results so it is not preferred. The 10 % corruption may slightly affect the recovered image, but lead to a very useful dictionary.

Fig. 9 shows the mean response of hidden units among all frames. With a sparse penalty in the object function, the trained results contain more useful information in units that have non-zero entries.

3.2 Detecting the Loops

The difference matrix is used for detecting loops in our experiments. The entry that is smaller than a threshold d

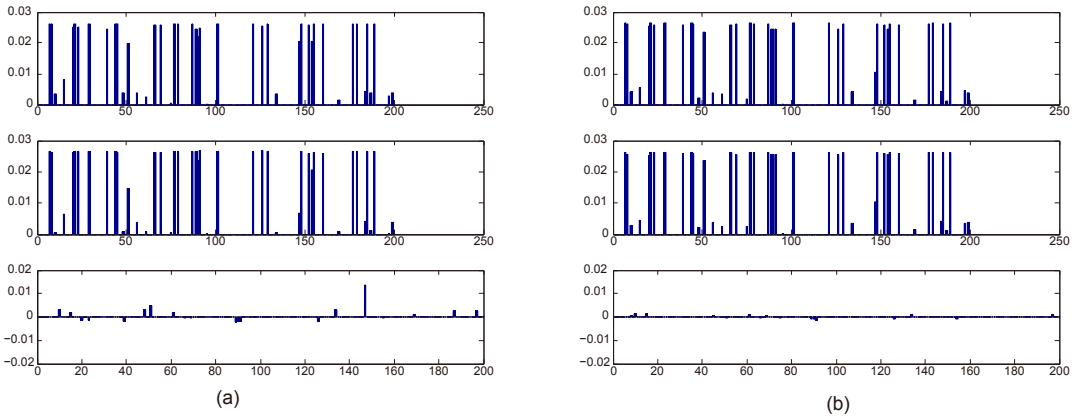

Fig. 7: Feature map for comparison. (a) Two different images. (b) Two similar images. The first row is feature response (output of hidden units)of image 1 $\boldsymbol{h}_1$. The second row is feature of image 2 $\boldsymbol{h}_2$. The third row is $\boldsymbol{h}_1 - \boldsymbol{h}_2$.

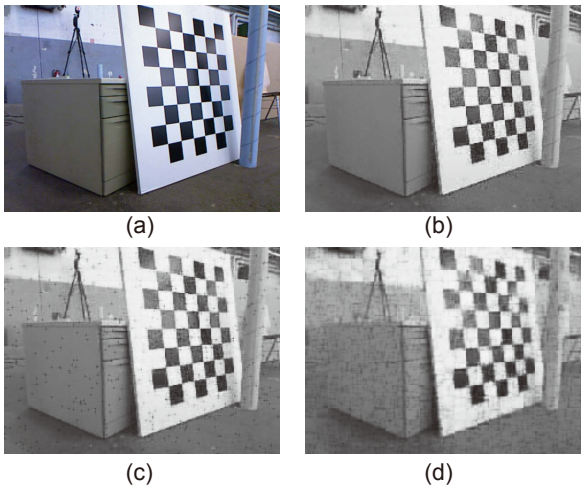

Fig. 8: Recovery of input image with different corruption levels. (a) original image. (b) recovered image with 0% corruption. (c) 1% corruption. (d) 10% corruption.

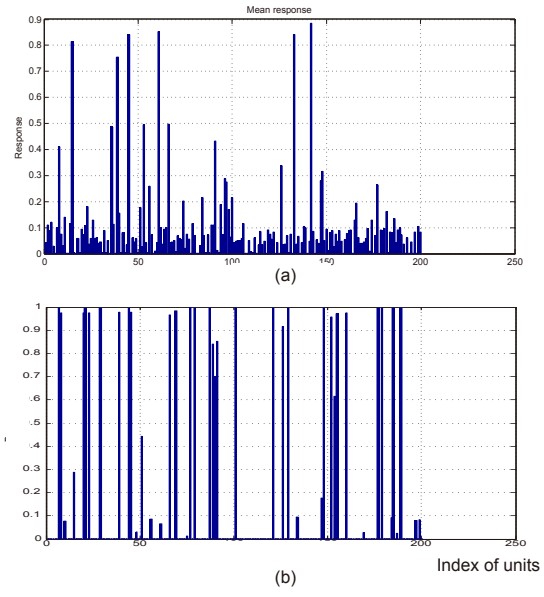

Fig. 9: Mean response of hidden units in all keyframes. (a) Without sparse penalty. (b) With sparse penalty.

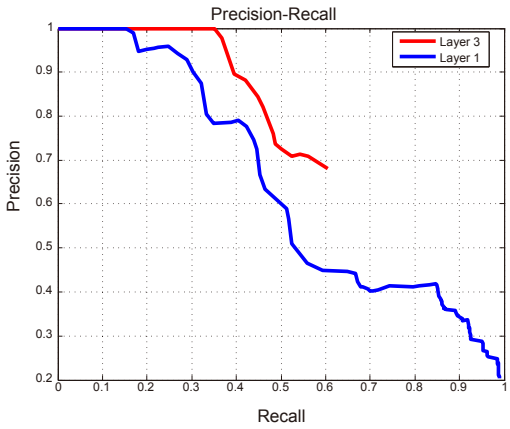

Fig. 10: The precision-recall curve of the algorithm. Red line: detecting with layer 3 features. Blue line: detecting with layer 1 features.

will be marked as a loop candidate. In order to evaluate the performance of algorithm we calculate the precision-recall curve by scanning the difference threshold, which is shown in Fig. 10. We compare the features from the first and the third layer, which are illustrated as blue and red lines in the figure. The third layer outperforms the first one because it can represent more complicated structure in the data, according to the deep learning theory. Without any later verifing process, Its precision remains about 70 % while the recall rate is 50%.

The detected loop closure is shown in Fig. 11. The blue lines are frames considered as loops, among which the lines that are not parallel to z-axis are false positive. Most of the loops are correct. In practical environments, once a loop is checked, its neighbor frames will also be check with the current frame, therefore a recall rate about 50% will be enough. In summary, the experiment shows that our method is feasible to check loop closure in visual SLAM systems.

4 Conclusion

In this paper, a loop closure detection method based on deep neural network is proposed. We provide the formu-

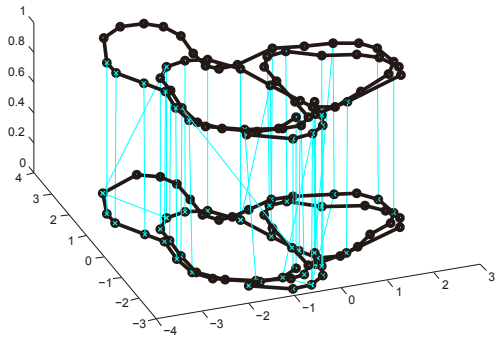

Fig. 11: The loops detected by the proposed method in this paper.

lation of modeling the problem as a stacked auto-encoder. The details of the training process and the method of defining difference matrix are investigated. We modify the object function of traditional auto-encoder by adding the denoising, sparsity and continuity cost item into it, and then evaluate the effect of corruption by experiments. The approach is tested on an widely used open dataset. The feasibility of it is verified by the precision-recall curve. Our future work is to study how to make use of the color and depth information of the sensor to robustify the detecting algorithm.

References

[1] M. Labbe and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.

[2] M. Cummins and P. Newman, "Appearance-only slam at large scale with fab-map 2.0," *International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.

[3] H. Wang, Z.-G. Hou, L. Cheng, and M. Tan, "Online mapping with a mobile robot in dynamic and unknown environments," *International Journal of Modelling, Identification and Control*, vol. 4, no. 4, pp. 415–423, 2008.

[4] K. Granstrom, T. B. Schon, J. I. Nieto, and F. T. Ramos, "Learning to close loops from range data," *International Journal Of Robotics Research*, vol. 30, no. 14, pp. 1728–1754, 2011.

[5] L. Teslić, I. Škrjanc, and G. Klančar, "Ekf-based localization of a wheeled mobile robot in structured environments," *Journal of Intelligent & Robotic Systems*, vol. 62, no. 2, pp. 187–203, 2011.

[6] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[7] D. Filliat, "A visual bag of words method for interactive qualitative localization and mapping," in *2007 IEEE International Conference onRobotics and Automation (ICRA)*, pp. 3921–3926, IEEE, 2007.

[8] B. Williams, G. Klein, and I. Reid, "Automatic relocalization and loop closing for real-time monocular SLAM," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 33, no. 9, pp. 1699–1712, 2011.

[9] H. Kwon, K. M. A. Yousef, and A. C. Kak, "Building 3d visual maps of interior space with a new hierarchical sensor fusion architecture," *Robotics and Autonomous Systems*, vol. 61, no. 8, pp. 749–767, 2013.

[10] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[11] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision–ECCV 2006*, pp. 404–417, Springer, 2006.

[12] K. Pathak, A. Birk, N. Vaskevicius, and J. Poppinga, "Fast registration based on noisy planes with unknown correspondences for 3-d mapping," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 424–441, 2010.

[13] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[14] L. Bo, X. Ren, and D. Fox, "Learning hierarchical sparse features for rgb-d object recognition," *International Journal of Robotics Research*, vol. 33, no. 4, pp. 581–599, 2014.

[15] I. Kostavelis and A. Gasteratos, "Learning spatially semantic representations for cognitive robot navigation," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1460–1475, 2013.

[16] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, "Autoencoder for words," *Neurocomputing*, vol. 139, pp. 84–96, 2014.

[17] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "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, 2012.

[18] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[19] K. Ho and P. Newman, "Detecting loop closure with scene sequences," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 261–286, 2007.

[20] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, #jun# 2010. Oral Presentation.