Neural Comput & Applic (2017) 28:429–436
DOI 10.1007/s00521-015-2082-x

ORIGINAL ARTICLE

Explicit guiding auto-encoders for learning meaningful representation

Yanan Sun[1] · Hua Mao[1] · Yongsheng Sang[1] · Zhang Yi[1]

Received: 16 July 2015 / Accepted: 11 October 2015 / Published online: 20 October 2015

Abstract The auto-encoder model plays a crucial role in the success of deep learning. During the pre-training phase, auto-encoders learn a representation that helps improve the performance of the entire neural network during the fine-tuning phase of deep learning. However, the learned representation is not always meaningful and the network does not necessarily achieve higher performance with such representation because auto-encoders are trained in an unsupervised manner without knowing the specific task targeted in the fine-tuning phase. In this paper, we propose a novel approach to train auto-encoders by adding an explicit guiding term to the traditional reconstruction cost function that encourages the auto-encoder to learn meaningful features. Particularly, the guiding term is the classification error with respect to the representation learned by the auto-encoder, and a meaningful representation means that a network using the representation as input has a low classification error in a classification task. In our experiments, we show that the additional explicit guiding term helps the auto-encoder understand the prospective target in advance. During learning, it can drive the learning toward a minimum with better generalization with respect to the particular supervised task on the dataset. Over a range of image classification benchmarks, we achieve equal or superior results to baseline auto-encoders with the same configuration.

Keywords Auto-encoders · Deep learning · Representation learning · Neural network

✉ Hua Mao
huamao@scu.edu.cn

[1] Machine Intelligence Laboratory, College of Computer Science, Sichuan University, Chengdu 610065, People's Republic of China

1 Introduction

It is well understood that the performance of machine learning algorithms is highly dependent on the choice of data representation (or features). The representation learning field is developing rapidly to better address questions on how to learn meaningful representations of given data [1]. Deep learning (DL) algorithms, which facilitate learning of hierarchical representations, have contributed to numerous research areas such as computer vision, speech recognition, and natural language processing [2–8]. However, for several well-known reasons, training using these algorithms is difficult. For one, the objective function is an extremely non-convex function, which leads to many distinct local minima in the parameter space [9]. An effective training method for deep architectures was introduced in 2006 with the algorithms for training deep belief networks using stacked auto-encoders [2, 10]. The basic idea used in this method is greedy layer-wise unsupervised pre-training followed by supervised fine-tuning.

Auto-encoders, which are the building blocks of stacked auto-encoders, learn the representation of the input pattern during the pre-training phase. This phase helps the entire network obtain a better solution in the fine-tuning phase. However, auto-encoders cannot learn meaningful representations without some implicit or explicit guidance in the pre-training phase. Several auto-encoder variants have been proposed to tackle this problem implicitly, including the sparse auto-encoder [11, 12], the denoising auto-encoder (DAE) [13], and the contractive auto-encoder (CAE) [14]. In sparse auto-encoders, a sparsity-constrained term, which penalizes extreme activation of the neurons, is added to the cost function of regular auto-encoders to encourage sparsity

in the representation [12]. DAEs randomly corrupt input data to reconstruct the original data by assuming that the high-dimensional input data exist in a low-dimensional underlying manifold. The representation is learned from the corrupted data to the manifold [13]. CAEs use an additional penalty term, which is the Frobenius norm of its Jacobian matrix, to decrease the sensitivity of the representation to small permutations of the input data [14].

More precisely, the sparsity-constrained term is perceived as prior knowledge of the input data. Practically, we must test many values of the hyper-parameters to select the proper degree of sparsity. This is inefficient as we cannot test every possible value to ascertain the intrinsic sparsity of the input data. When training with DAEs, the same question arises about how to determine the level and type of corruption. Although the representation learned by CAEs is robust, this representation is not always meaningful for specified tasks. In attempting to learn a good representation, the auto-encoders mentioned above follow the principle of minimizing the reconstruction error and maximizing robustness of the representation. However, by using these implicit methods to learn the representation in the unsupervised pre-training phase means that the auto-encoders do not have any knowledge of the particular supervised task in the fine-tuning phase. In other words, auto-encoders do not know what a meaningful representation would be in the fine-tuning phase. At one extreme, the common representation among all categories of training data could be learned, while at the other, every sample in each category must be considered. For example, in the Mixed National Institute of Standards and Technology (MNIST) handwritten digit recognition problem [15], the representation learned from the dataset with ten categories would be the common representation, which is not discriminative with respect to each category for a multiclass classification task.

Considering this, we propose a new auto-encoder variant, called the explicit guiding auto-encoder (EGAE). EGAEs learn meaningful representations for specified tasks. In EGAEs, a penalty term is added to the traditional reconstruction cost function to measure the benefit of the representation to the supervised task. More precisely, the penalty term is the classification error with respect to the representation learned by the auto-encoder during the pre-training phase of a classification task. Using benchmark datasets, we show that EGAEs can learn meaningful representations by driving the learning toward the basin of the optimization space with better generalization, thereby improving the performance of the final task. Additionally, qualitative analyses of the results show that the representations learned by EGAEs are interpretable.

2 Variants of auto-encoders

The simplest form of an auto-encoder [16] is a neural network with three layers, namely the input layer, hidden layer, and output layer. Moreover, the desired output from the output layer is a reconstruction of the input data. The number of units in the output layer is equal to that in the input layer. The data transformation from the input layer to the hidden layer is an encoder denoted by $F(\cdot)$, and the transformation from the hidden layer to the output layer is a decoder denoted by $G(\cdot)$. Given m instances of n-dimensional input data and an auto-encoder with k units in the hidden layer, these two transformations are formulated as:

$$\begin{cases} Y = F(W_e X + b_e) \\ \tilde{X} = G(W_d Y + b_d), \end{cases} \tag{1}$$

where $X \in \mathbb{R}^{n\times m}$ denotes the input data, and $W_e \in \mathbb{R}^{k\times n}$, $b_e \in \mathbb{R}^k$, $W_d \in \mathbb{R}^{n\times k}$, and $b_d \in \mathbb{R}^n$ denote the weight matrix for the encoder, bias column vector in the hidden layer, weight matrix for the decoder, and bias column vector in the output layer, respectively.

Generally, the cost function of auto-encoders is defined as:

$$\mathcal{J}_{\mathrm{AE}} = L\big(X, \tilde{X}\big), \tag{2}$$

where $L(\cdot)$ denotes the reconstruction error of auto-encoders. Two particular forms are often considered, namely the mean square error:

$$L(X,\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\big(X_i^j - \tilde{X}_i^j\big)^2, \tag{3}$$

and the cross-entropy error:

$$L(X,\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\big(X_i^j log(\tilde{X}_i^j) + (1 - X_i^j) log(1 - \tilde{X}_i^j)\big), \tag{4}$$

where $\theta = \{W_e, b_e, W_d, b_d\}$ and X_i^j denotes the jth element of the ith component in X.

Auto-encoders were first introduced in [17, 18] to reduce data dimensionality; however, they are now frequently employed to extract numerous features [14]. For this purpose, several variants of auto-encoders have been proposed for learning good representations. In the following subsections, we briefly introduce the auto-encoder variants compared in this study.

2.1 Sparse auto-encoder

By simply imposing the reconstruction error, it is possible to learn an identity function by minimizing the Function 2 with $k \geq n$ [3, 4, 12]. In this case, the representation of the

input data is a trivial solution, i.e., a direct copy of the input. An infinite number of such possible trivial solutions exist. Among all these representations, many studies on neuroscience have suggested a sparse solution with the greatest number of zero components. This property can be induced by adding a sparsity-constrained term to the cost function. This constraint yields a group of more invariant features against small permutations of the input data [3, 4, 19]. The cost function for the sparse auto-encoder is defined as:

$$\mathscr{J}_{\text{AE+sparse}} = L\left(X, \tilde{X}\right) + \lambda S(Y). \tag{5}$$

Function $S(Y)$ measures the degree of sparsity with respect to Y and λ is used to balance how much $S(Y)$ accounts for in Function 5.

2.2 Auto-encoder with weight decay

We need to decrease the complexity of the network as much as possible in the case where the model overfits the training data or underfits the test data. Thus, the generalization ability of the model should be taken into account [20–22]. The most commonly used method to achieve this is to target either the number or magnitude of parameters of the network. Considering the latter, weight decay has been proposed to penalize those parameters with large values. With the small weight constraint, the impact of a few distortions in the input data is limited. The cost function of an auto-encoder with weight decay is given as:

$$\mathscr{J}_{\text{AE+WD}} = L\left(X, \tilde{X}\right) + \lambda \sum_{i}^{m} \sum_{j}^{n} W_{ij}^{2}, \tag{6}$$

where hyper-parameter λ controls the penalty level.

2.3 Denoising auto-encoder

A DAE [13] maps the representation of its artificially corrupted input data back to the original uncorrupted samples. It assumes that the underlying low-dimensional manifold is embedded in the high-dimensional input space. The mapping learns the manifold from the input data with additional corrupted data. Moreover, the learning territory is enlarged by training the corrupted input data, thereby increasing the learning ability of the DAE. The cost function has the form:

$$\mathscr{J}_{\text{DAE}} = L\left(X', \tilde{X}\right). \tag{7}$$

In particular, $X' = F_{\text{noise}}(X)$ and function $F_{\text{noise}(\cdot)}$ denotes the operation of adding some kind of noise. Three types of noise have been suggested for the DAE [13].

2.4 Contractive Auto-encoder

In CAEs, the cost function includes an interesting penalty term that encourages invariants or insensitivity of the representation against small permutations in the input data, thereby leading to robust features. This penalty term involves partial differentiation of representation Y with respect to input data X. It is obvious that by imposing this constraint, Y can become robust within a local region of X. The cost function is given as:

$$\mathscr{J}_{\text{DAE}} = L\left(X, \tilde{X}\right) + \lambda \sum_{j}^{m} \sum_{i}^{k} \sum_{p}^{n} \left(\frac{\partial Y_i^j}{\partial X_p^j}\right)^2, \tag{8}$$

where λ is the balancing hyper-parameter. During the pre-training phase of a CAE, tied weight (i.e., W_d equal to the transpose of W_e) is often employed to overcome the trivial optimization problem caused by the contractive term. The reason behind this is that W_e must be almost zero to enable the contractive term to satisfy minimization of the cost function in the Eq. 8.

3 Explicit guiding auto-encoder

An EGAE is a regular auto-encoder with an extra explicit guiding term that guides the learning toward obtaining a meaningful representation of the input data. A meaningful representation means that a network using the representation as input has a low classification error in the classification task. This explicit guiding term can inform the representation in the training phase, while ensuring improved performance in the fine-tuning phase. By minimizing the cost function of an EGAE, a meaningful representation can be learned.

In a multiclass classification task with a modern deep learning framework, we use p auto-encoders for unsupervised feature learning in the pre-training phase and one softmax layer for supervised learning in the fine-tuning phase. Let $a(k)$ denote the kth auto-encoder, and $X(k)$, $Y(k)$, and $\tilde{X}(k)$ denote the input data, representation of the input data, and reconstruction in $a(k)$, respectively. We define the cost function of an EGAE as:

$$\mathscr{J}_{\text{EDAE}} = \sum_{k=1}^{p} L\left(X(k), \tilde{X}(k)\right) + \beta \sum_{i=1}^{m} C(W_c, Y(i), T_i) + \lambda R(\cdot). \tag{9}$$

Specifically,

$$C(W_c, Y(i), T(i)) = -[T(i) log(\text{pred(i)}) + (1 - T(i)) log(1 - \text{pred(i)})], \tag{10}$$

and

$$\mathrm{pred(i)} = \mathrm{softmax}(\mathrm{W_c}, \mathrm{Y(i)}).$$

In Eq. 9, $R(\cdot)$ denotes the constrained term commonly used in other auto-encoders, such as the sparsity constraint or contractive term, m denotes the number of samples in the training set, $T(i)$ denotes the label of the ith sample, W_c is the classification matrix (discussed in detail in Sect. 3.2), and λ and β are balancing terms. In most cases, β is set to one based on the underlying principle of comparably emphasizing reconstruction ability with a low classification error. This is also affirmed by the experimental results in Sect. 4.

The architecture of auto-encoders using the EGAE algorithm is shown in Fig. 1. This specific example involves three auto-encoders enclosed by the green rectangle.

3.1 Relationship with other auto-encoders

Two principles have been suggested for training auto-encoders: first, the representation should, as much as possible, retain the original information; and second, the representation should be robust against small variations in the input data [14]. The representations used by the auto-encoder variants mentioned in Sect. 2 improve the performance of the supervised tasks with their extra terms or by artificially corrupting the input. More specially, sparse auto-encoders work with sparsity-constrained term, weight decay auto-encoders take effect by penalizing connection weights with large magnitude, DAEs adopt different train criteria from other auto-encoders variants, and CAEs function by encouraging small values of the partial differentiation. However, their meaningfulness is *implicit* with the help of the specially designed term. Conversely, meaningfulness of the EGAE takes effect by its *explicit* guiding term that reflects to what degree the representation improves the performance of training the auto-encoder. Evaluation of the performance of the supervised task performance takes place in the feature learning phase. Thus, it is not surprising that we can achieve better performance using a combination of implicit and explicit terms, as shown empirically in Sect. 4.

3.2 Training with EGAE

Traditional deep learning algorithms have a pre-training and a fine-tuning phase. Training with EGAE follows this process with an additional procedure to obtain the classification matrix W_c by training a multiple layer perception (MLP) included in the pre-training phase. The algorithm for W_c is described in Algorithm 1.

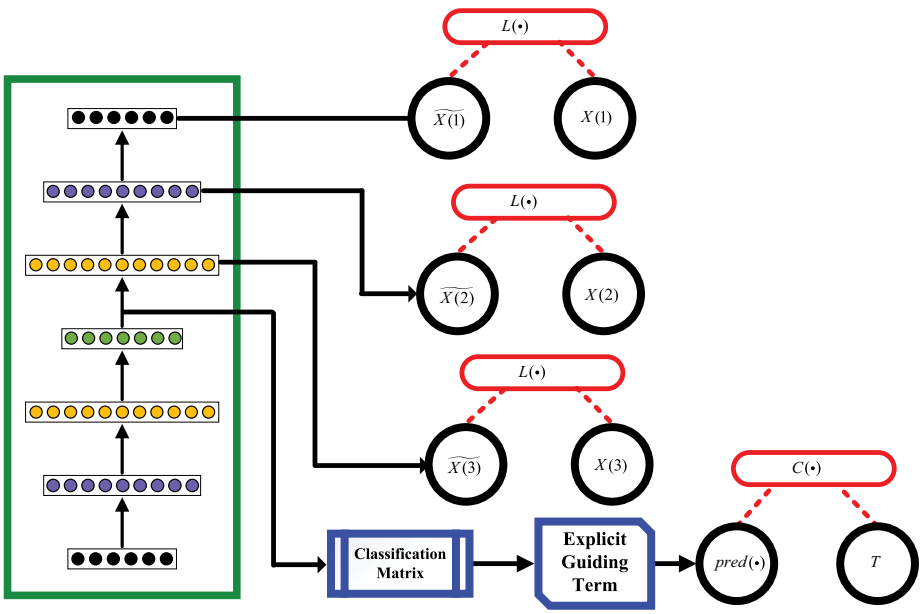

Fig. 1 Illustration of the architecture for explicit guiding auto-encoder

Algorithm 1 Obtaining classification matrix in explicit guiding auto-encoders

1: Initialize an MLP with input layer l_{in}, output layer l_{out}, and hidden layers $l_h(1), l_h(2), l_h(3), \cdots, l_h(n)$ in the auto-encoders
2: Initializing the weight matrix $W_c \leftarrow random()$
3: **while** not converging **do**
4: Update weights in the MLP with Back-propagation algorithm
5: **end while**
6: **return** W_c

In the pre-training phase of the EGAE, we stack all the auto-encoders to carry out joint training as illustrated in Fig. 1 using the cost function given in Eq. 9 and classification matrix W_c. After the pre-training phase, we discard the decoder and extract an MLP using Algorithm 1 with corresponding weights. The MLP is then fine-tuned in the supervised task.

4 Experiments

In this section, we present our experiments using the EGAE algorithm and auto-encoder variants discussed in Sect. 2. The following benchmark datasets are considered:

- MNIST: The well handwritten digital recognition problem [15] comprises 50,000 training samples and 10,000 test samples.
- Canadian Institute for Advanced Research (CIFAR)-bw: the grayscale version of the CIFAR-10 dataset [23] consists of ten categories for image classification. There are 50,000 training images and 10,000 test images.
- Variation on MNIST: A variation of the MNIST dataset comprises five subsets: (1) MNIST basic, (2) MNIST with background images, (3) MNIST with random background, (4) rotated MNIST digits, and (5) rotated MNIST digits with background images [5]. Each subset has a training dataset, validation dataset, and testing dataset with 10,000, 2,000, and 50,000 samples, respectively.

In the MNIST and CIFAR-bw datasets, 10,000 samples were randomly selected from the training datasets as the corresponding validation datasets. First, we evaluated networks of varying depths with and without the explicit guiding term. Second, qualitative analysis was carried out by visualizing the features learned by the EGAE with three hidden layers. Third, the learning trajectories of the model with and without the explicit guiding term were also compared to obtain a more intuitive understanding of their differences while learning the representation. Finally, quantitative experiments were carried out using the EGAEs as well as other auto-encoder variants.

4.1 Experiments setup

Tied weights were used in our experiments; for all the auto-encoders, $W_e = W_d^T$. We used the sigmoid activation function for both the encoder and decoder. The mean squared error cost function was used as the reconstruction error, while a cross-entropy error was used to evaluate the performance of the classification task. The neural networks were trained by stochastic gradient descent with the AdaDelta technique to select the learning rate adaptively [24]. We adopted the widely used decay rate $\rho = 0.95$ and constant $\epsilon = 10^{-6}$. Mini-batches were used with batch size 200. Moreover, training terminated when the performance stopped decreasing with a validation frequency of 50 or reached the maximum epoch of 1000. Regarding the other hyper-parameters in each model, we chose values that yielded the best performance in our experiments on the corresponding validation dataset. The hyper-parameters in the auto-encoders were selected according to the performance of the supervised task after the feature learning phase. Grid search of hyper-parameters was performed.

To initialize the networks, weights of the auto-encoders were initialized by uniform sampling between $[-t, t]$ [25].

$$t = 4.0/\sqrt{6.0/(n_{\text{in}} + n_{\text{out}})}, \tag{11}$$

where $n_{\text{in}}, n_{\text{out}}$ denote the number of neurons in the input and output layers, respectively. All the bias and weights for the softmax layer were initialized to zero. Values of the coefficients of regularizations, such as sparsity, weight decay, and contractive, are selected from $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. Masking noise was considered in training the model of the DAE, while the corruption level of the DAE was selected from {10, 20, 30, 40, 50 %}.

All the experiments were implemented with Theano [26, 27] using graphics processing units (GPUs). Two GPU devices were used in our experiments: NVIDIA GTX750Ti and GTX780.

4.2 Effect of depth in EDAE

We compared the models for a range of depths {1, 2, 3, 4, 5}. For each setting, the models were trained with and without the explicit guiding term on the MNIST dataset. Comparison of the results is shown in Fig. 2.

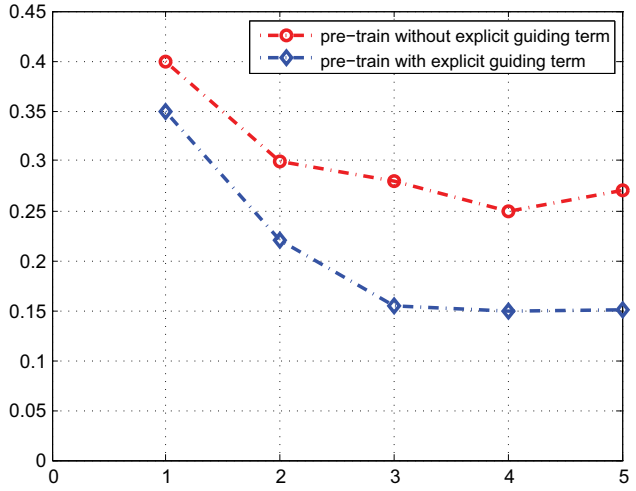

Fig. 2 Effect of depth in the models on performance. Models with and without explicit guiding term are compared

This experiment verifies that the EGAEs achieve superior performance for a varying range of depths compared with the model without the explicit guiding term. Moreover, the performance of the EGAEs improved as the depth increased, which means that the representation becomes increasingly meaningful as the multilayer auto-encoders learn the hierarchical representation with explicit guiding terms.

4.3 Visualization of features learned by EGAE

Visualization of the features learned by the first layer of the model with the explicit guiding term is shown in Fig. 3. We also visualized the features on the second and third layers by maximizing the activation method introduced in [28] to show that meaningful representations were learned. We computed the features by applying the gradient descent optimization method to the activation function at the corresponding layer to maximize its output.

On the MNIST dataset, EGAEs learned features resembling strokes in the first layer. Similar results with other auto-encoder variants have been reported in many works. The features in the third layer have been relatively unexplained until now; they have been described as a kind of high-level feature as reported in [28]. Normally, the features learned in the second layer fit between the features in the first and third layers.

4.4 Learning trajectory

Although visualization of features can help us understand the qualitative effect of EGAEs, it is not clear how the explicit guiding term affects the learning procedure. Thus, we performed experiments to show the learning procedure. The parameters (i.e., weights and bias in the network) indicate the position of learning at each iteration during the training phase. However, we cannot compare the parameters directly because different parameters may lead to the same model. We employed the functional approximation method [9] to show the learning trajectory. To be more precise, the output of each sample in the test dataset was concatenated to form a long vector using the parameters trained at each iteration of the training phase. We applied dimensionality reduction in the vector space using ISOMAP [29], the MATLAB (matrix laboratory) implementation of which can be downloaded from http://isomap.stanford.edu/. Dimensionality reduction was applied to both results of the EGAE and the auto-encoder without the explicit guiding term for visualization in the same space.

We compared the learning trajectories of a model with and without the explicit guiding term based on the optimal effect of the hyper-parameters on performance over the validation dataset. The trajectories, shown in Figs. 4 and 5, clearly show that the respective learning of the two models starts at a slightly different position and moves apart after the third epoch. Finally, these converge at completely different minima. In particular, learning with the explicit guiding term moves directly toward the minimum along the line after the ninth epoch, whereas the

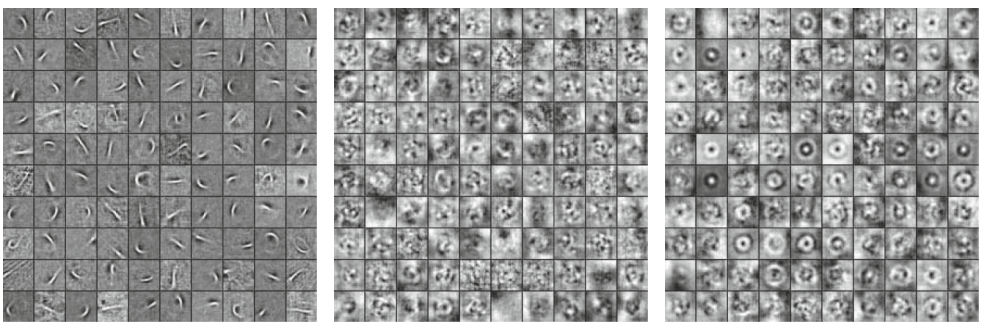

Fig. 3 Visualization of the features learned by EGAE. **a** Features in the first layer. **b** Features in the second layer. **c** Features in the third layer

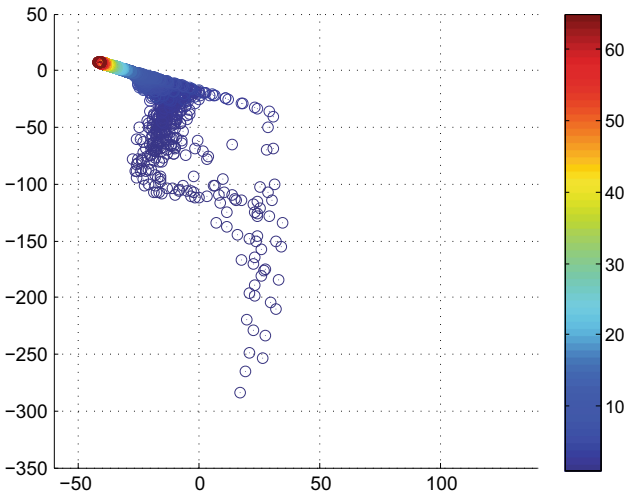

Fig. 4 The learning trajectory of a model with three layers of explicit guiding auto-encoders

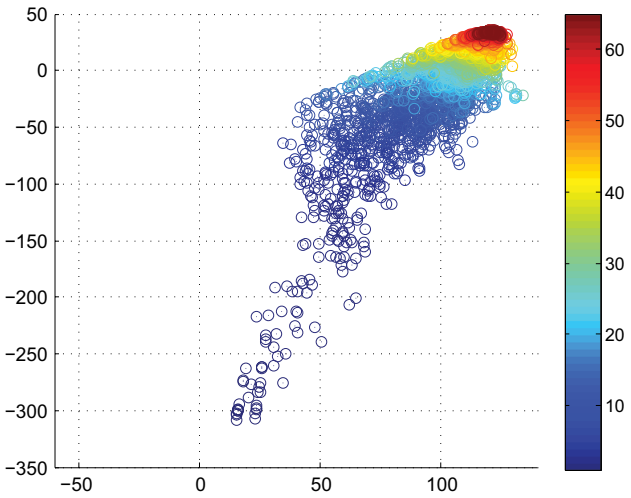

Fig. 5 The learning trajectory of a model with three layers of auto-encoders

model without the explicit guiding term shows oscillation before convergence. Together with the results reported in Sect. 4.2, we concluded that the explicit guiding term drives the learning toward the basin of minima at which the model has a better ability to generalize. Generalization ability gives rise to better classification performance because of the meaningful representation learned by the algorithm.

4.5 Quantitative comparisons

We compared the performance of the proposed EGAEs with that of other auto-encoder variants on variations of the MNIST dataset and the CIFAR-bw dataset. The best results of each model tuned by grid search on the validation datasets are reported in Table 1.

It is clear that the efficiency of the model pre-trained with the explicit guiding term is superior to that of an MLP. The performance of the EGAEs substantially surpassed that of the other auto-encoder variations under the same configuration.

5 Conclusion

EGAEs are auto-encoders with an extra explicit guiding term that guides the auto-encoder to learn a meaningful representation, thereby giving the network better performance. This explicit guiding term in the EGAE can also be combined with other implicit terms for auto-encoders. Quantitative and qualitative experiments demonstrate that the EGAEs learned an interpretable representation similar to other well-known auto-encoders. However, the EGAEs also learned more meaningful representation. Additionally, from the comparison of the learning trajectories of the models with

Table 1 The classification error rates on the variations on MNIST and CIFAR-bw datasets

Algorithm	Data set					
	basic	rot	bg-rand	bg-img	bg-img-rot	cifar-bw
AE+sparse	4.40	13.11	17.05	19.29	50.46	51.76
DAE-b	3.54	12.71	**11.11**	17.65	47.06	49.75
CAE	3.47	12.79	12.96	17.70	48.95	51.97
MLP	5.11	13.26	17.03	19.14	49.04	53.92
EGAE	**2.98**	**10.92**	11.48	**16.37**	**46.55**	**49.27**

The best result on each dataset was highlighted with bold font

and without the explicit guiding term, it is obvious that the EGAEs guide the learning toward a better solution with better generalization ability.

Acknowledgments This work was supported by the National Science Foundation of China (Grant Nos. 61432012 and 61402306).

References

1. Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828
2. Bengio Y, Lamblin P, Popovici D, Larochelle H et al (2007) Greedy layer-wise training of deep networks. Adv Neural Inf Process Syst 19:153
3. Boureau Y-L, Cun YL, et al (2008) Sparse feature learning for deep belief networks. In: Platt JC, Koller D, Singer Y, Roweis ST (eds) Advances in neural information processing systems, Curran Associates, Inc., NY, USA, pp 1185–1192
4. Poultney C, Chopra S, Cun YL, et al (2006) Efficient learning of sparse representations with an energy-based model. In: Schölkopf B, Platt JC, Hoffman T (eds) Advances in neural information processing systems, MIT Press, Cambridge MA, USA, pp 1137–1144
5. Larochelle H, Erhan D, Courville A, Bergstra J, Bengio Y (2007) An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of the 24th international conference on machine learning, ACM, pp 473–480
6. Osindero S, Hinton GE (2008) Modeling image patches with a directed hierarchy of Markov random fields. In: Platt JC, Koller D, Singer Y, Roweis ST (eds) Advances in neural information processing systems, Curran Associates, Inc., NY, USA, pp 1121–1128
7. Collobert R, Weston J (2008) A unified architecture for natural language processing: deep neural networks with multitask learning. In: Proceedings of the 25th international conference on machine learning, pp 160–167
8. Weston J, Ratle F, Mobahi H, Collobert R (2012) Deep learning via semi-supervised embedding. In: Montavon G, Orr G, Müller KR (eds) Neural networks: tricks of the trade, Springer, NY, USA, pp 639–655
9. Erhan D, Bengio Y, Courville A, Manzagol P-A, Vincent P, Bengio S (2010) Why does unsupervised pre-training help deep learning? J Mach Learn Res 11:625–660
10. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
11. Jarrett K, Kavukcuoglu K, Ranzato M, LeCun Y (2009) What is the best multi-stage architecture for object recognition? In: Computer vision, 2009 IEEE 12th international conference on, IEEE, pp 2146–2153
12. Olshausen BA, Field DJ (1997) Sparse coding with an overcomplete basis set: a strategy employed by v1? Vis Res 37(23):3311C3325
13. 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, ACM, pp 1096–1103
14. Rifai S, Vincent P, Muller X, Glorot X, Bengio Y (2011) Contractive auto-encoders: explicit invariance during feature extraction. In: Proceedings of the 28th international conference on machine learning (ICML-11), pp 833–840
15. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
16. Bengio Y (2009) Learning deep architectures for AI. Found Trends Mach Learn 2(1):1–127
17. Rumelhart DE, Hinton GE, Williams RJ (1988) Learning representations by back-propagating errors. Cogn Model 5:3
18. Baldi P, Hornik K (1989) Neural networks and principal component analysis: learning from examples without local minima. Neural Netw 2(1):53–58
19. Goodfellow I, Lee H, Le QV, Saxe A, Ng AY (2009) Measuring invariances in deep networks. In: Bengio Y, Schuurmans D, Lafferty JD, Williams CKI, Culotta A (eds) Advances in neural information processing systems, Curran Associates, Inc., NY, USA, pp 646–654
20. Baum EB, Haussler D (1989) What size net gives valid generalization? Neural Comput 1(1):151–160
21. Schwartz D, Samalam V, Solla SA, Denker J (1990) Exhaustive learning. Neural Comput 2(3):374–385
22. Tishby N, Levin E, Solla SA (1989) Consistent inference of probabilities in layered networks: Predictions and generalizations. In: Neural networks. IJCNN., international joint conference on, IEEE, pp 403–409
23. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. Computer Science Department, University of Toronto. Technical report 1(4):7
24. Zeiler MD (2012) Adadelta: an adaptive learning rate method, arXiv preprint arXiv:1212.5701
25. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: International conference on artificial intelligence and statistics, pp 249–256
26. 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 preprint arXiv:1211.5590
27. 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), vol 4, Austin, TX, p 3
28. Erhan D, Bengio Y, Courville A, Vincent P (2009) Visualizing higher-layer features of a deep network. Department of IRO, Universit de Montral, Technical report
29. Tenenbaum JB, De Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. Science 290(5500):2319–2323