Neural Computing and Applications (2020) 32:8127–8142
https://doi.org/10.1007/s00521-019-04309-3

ORIGINAL ARTICLE

SRS-DNN: a deep neural network with strengthening response sparsity

Chen Qiao[1] · Bin Gao[1] · Yan Shi[1]

Received: 19 December 2018 / Accepted: 17 June 2019 / Published online: 26 June 2019

Abstract
Inspired by the sparse mechanism of biological neural systems, an approach of strengthening response sparsity for deep learning is presented in this paper. Firstly, an unsupervised sparse pre-training process is implemented and a sparse deep network is begun to take shape. In order to avoid that all the connections of the network will be readjusted backward during the following fine-tuning process, for the loss function of the fine-tuning process, some regularization items which strength the sparse responsiveness are added. More importantly, the unified and concise residual formulae for network updating are deduced, which ensure the backpropagation algorithm to perform successfully. The residual formulae significantly improve the existing sparse fine-tuning methods such as which in sparse autoencoders by Andrew Ng. In this way, the sparse structure obtained in the pre-training can be maintained, and the sparse abstract features of data can be extracted effectively. Numerical experiments show that by this sparsity-strengthened learning method, the sparse deep neural network has the best classification performance among several classical classifiers; meanwhile, the sparse learning abilities and time complexity all are better than traditional deep learning methods.

Keywords Deep neural network · Strengthening response sparsity · Sparse backpropagation algorithm · Unified residual formulae

1 Introduction

As a state-of-the-art learning paradigm in machine learning, deep learning can learn the layer-by-layer structure by greedy layer-wise training, and the neurons can represent the multilayer nonlinear expression of the data. Thus, deep learning can profoundly reveal the complex abstract information of the data, which are conducive to feature extraction and classification, regression and other learning tasks [1, 2].

✉ Chen Qiao
qiaochen@xjtu.edu.cn

Bin Gao
guitarpz@stu.xjtu.edu.cn

Yan Shi
shiyan93617@stu.xjtu.edu.cn

1 School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an 710049, People's Republic of China

For deep learning, there are usually two steps of learning processes, one is pre-training and the other is fine-tuning. In the pre-training stage, the abstraction features and distributions of the data can be learned in an unsupervised way. Then, a fine-tuning process will be executed according to special learning tasks. The well-known backpropagation (BP) algorithm is the most commonly used fine-tuning method [3]. By BP, it repeatedly adjusts the weights of the connections top-down as to minimize a measure of the difference between the actual output vector of the network and the desired output vector. As a result of weight adjustments, internal hidden neurons come to represent important features of the task domain.

The classical BP results in neurons between neighboring layers having a widely interconnected structure. That is because, by implementing gradient descent, every one of the weights will be updated in an unbiased way, and thus, the data representation will be distributed on all layers of the entire network. However, studies show that in the human brain's organizational structure and information processing process, there do exist sparse properties, namely response sparseness and connection sparseness. For example, Olshausen and Field [4] suggest that by a combination of experimental, computational and theoretical studies, it has been verified that the existence of an underlying principle involved in sensory information processing, namely the information is represented by a relatively small number of simultaneously active neurons out of a large population, commonly referred to as sparse coding. In [5], it points out that there exists sparse connectivity within the basal ganglia. These two kinds of sparsity are also known as the sparse representation of neurons and sparse structure or sparse topology of the network [6]. Compared with the distributed information representation, sparse representations and sparse connections are more able to provide high-quality storage capabilities and have better generalization capabilities [5]. One group of DeepMind points out that sparseness is quite critical to the emergence of grid-like representations when they study automatic tracking like humans by deep learning methods [7]. Moreover, for sparse learning, there are also some other important works, such as transforming tree-structured data into vectorial representations [8, 9]. How to better understand the intrinsic sparsity of human neural network system, so as to effectively obtain a sparse BP learning algorithm and reduce the high complexity of deep learning models, thus achieve a rapid and accurate feature learning, all are of great concern.

At present, there exist many approaches to get the sparsity of networks by regularization algorithms, and most of them are carried out in the pre-training process. The methods of weighting constraints in the pre-training process includes Gaussian regularizer, Laplace regularizer, weight elimination and soft weight sharing, and most of them use L_2-norm or L_1-norm of the connective weights as a penalty function. In addition to adding regular items on the weights, simulating the sparse response of the hidden neurons is another way to improve the generalization ability of the pre-training network [10–16], while only by sparse pre-training, the final sparseness of the entire network cannot be held. This is due to that the process of BP will re-adjust all weights of the entire network top-down, and finally complete the whole learning tasks. This is clearly contrary to the ultimate goal, i.e., establishing a network architecture which conforms to the sparse representations and sparse connections in human neural systems.

Obviously, if the sparsity for responses of hidden neurons and network connections can be emphasized in the fine-tuning BP stage, then by combining with sparse pre-training, even if the network is randomly initiated with dense connections, eventually only a small amount of neurons will be responded and connected in the network. This can not only bring a sparse deep network architecture but also can ease over-fitting, and thereby will enhance the generalization capacity as well as the identification performance of the network. For achieving sparsity in BP algorithm, some approaches have been proposed by deleting the least important connections or enforcing a large proportion of connections to have small weights through pruning techniques, dropout or adding regularization terms [17–20], while, only obtaining the connections sparsity cannot get the responses sparsity; small connections cannot ensure small activation value of the corresponding neuron since it also receives information from other connections [21].

It is noted, however, when regularization terms are coupled to the loss function, especially those containing the response of each hidden neurons in all hidden layers, deriving the gradient descent formula with a succinct form becomes fairly complicated, and not to mention, finally embed the formula to the codes of BP algorithm. Since one crucial reason that BP is widely applied lies on that the residual formula of it is concise which can be applied for networks with arbitrary layers; thus, how to obtain general residual formulae for the loss function combining regularization terms, so as to get a unified sparse updating formula of BP is very meaningful. In [22], Andrew Ng. adds Kullback–Leibler (KL) divergence in BP process to get the responses sparsity for autoencoders. While, due to the difficulty of deducing the updating formula (the complexity of formulas for two or more hidden layers can be found in Sect. 3.2.1), it is achieved only for one hidden layer, let alone to obtain a unified formula for networks with arbitrary hidden layers. Zhang et al. [21] uses L_1-norm constrains directly on the response of hidden neurons in the BP process, while the result is complex and not in a concise form. All these prompt us to find some more general and unified results for the responses sparsity in BP learning.

In this paper, we devote to present the unified residual formulae, which promise the sparse-strengthened capability for both responses and connections of the network, and can be applied for deep learning with arbitrary layers. A deep neural network (DNN) with a deep belief network (DBN) as the pre-training model will be introduced first. For the unsupervised DBN learning procedure, both KL divergence of hidden neurons and L_1-norm penalty on connection weights of each hidden layers are considered, which are implemented to decrease the complexity of the model and thus improve the generalization capabilities of the network. Thus, the DBN can learn the effective data representation in a sparse way. Based on the learned sparse DBN, in the BP stage, since KL divergence is a standard function for measuring how unequally two different distributions are, and L_1-norm is generally used as a substitution of L_1-norm to get the sparse solution, we further introduce regularization terms with KL divergence and Laplace penalty on hidden neurons in the loss function. By inducing the residual formulae of loss function with the regularization terms on KL divergence and Laplace penalty, we overcome the difficulty in gradient descent derivation when coupling penalty terms on the hidden neurons. Finally, a unified and concise updating formula for DNN with arbitrary layers is achieved, which is a significant improvement of the existing results, such as those in [21, 22]. The proposed sparse-strengthened BP method ensures the sparse representation and the sparse architecture of the final learned deep network, which can promise the generalization capacity, accuracy rate of classification, compression rate and speed up the convergence rate of the deep network.

Experiments on the Fashion-MNIST database show that the proposed deep neural networks with strengthening sparsity has fairly good sparse response ability and sparse topology structure, thus it can effectively learn the sparse representation of the data. Compared with the classical classifiers, the classification accuracy is greatly improved, and the speed of discrimination is only a half of the traditional DNN.

2 Unsupervised sparse learning: DBN with sparse architecture and sparse representation

2.1 Basic learning rules of RBM and DBN

DBN is a probability generation model, consisting of one layer of visible units and multiple layers of hidden units, with connections between different layers but not between units within the same layer [1]. A DBN can be viewed as a composition of simple, unsupervised networks such as restricted Boltzmann machines (RBMs). An RBM is a stochastic artificial neural network, which can learn the probability distribution over the raw features. A DBN uses a greedy layer-wise training method [23]. For each upper layer, it takes the output of the previously trained layer as its input, i.e., using the trained hidden units of the previous RBM as pre-training initialization. DBN and its learning procedure with stacked RBMs are shown in Fig. 1. When trained on a set of examples in unsupervised manner, a DBN can learn to probabilistically reconstruct its input data, and at each layer, it performs feature abstraction on the current inputs [24]. By the unsupervised greedy layer-by-layer pre-training of DBN, the improved weights of multilayer RBMs can be obtained, which provides the network with a better structure based on the understanding of the data.

In an RBM, there are two layers: one is a visible input layer, and the other is a hidden layer. There are connections between the layers but no connection between units within each layer. Let $v=(v_1,v_2,\ldots,v_{N_v})^{\mathrm{T}}$ be the visible units, $h=(h_1,h_2,\ldots,h_{N_h})^{\mathrm{T}}$ be the hidden units, $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_{N_v})^{\mathrm{T}}$ be the bias of the visible units, $\beta=(\beta_1,\beta_2,\ldots,\beta_{N_h})^{\mathrm{T}}$ be the bias of the hidden units, and $W=\{W_{ij}\}_{N_v\times N_h}$ with each w_{ij} being the connection weight between v_i and h_j. A joint configuration, (v,h), of the visible and hidden units has an energy given by $E_\theta(v,h)=-\alpha^T v-\beta^T h-v^T Wh$, in which $\theta=\{W,\alpha,\beta\}$ is the set of all parameters. $P_\theta(v)=\frac{1}{Z_\theta}\sum_h e^{-E_\theta(v,h)}$ is defined as the distribution of the observed data v_θ, here $Z_\theta=\sum_{v,h}e^{-E_\theta(v,h)}$ is known as the partition function or normalizing constant. $P_\theta(v)$ also has another name, i.e., likelihood function. Maximizing it is just the task of training an RBM. That is equal to determining the parameters to fit the given training samples, i.e., to find a θ^*, such that $\theta^*=\arg\max_{\Theta}\mathcal{L}(\theta)$ with $\mathcal{L}(\theta)=\log P_\theta(v)$. By performing stochastic steepest ascent in the log probability on the training data, the weights are updated by $\Delta W_{ij}=\epsilon\cdot\frac{\partial\mathcal{L}}{\partial W_{ij}}=\epsilon\cdot(\langle v_i h_j\rangle_{\text{data}}-\langle v_i h_j\rangle_{\text{model}})$, where ϵ is the learning rate and $\langle\cdot\rangle$ is the operator of expectation with the corresponding distribution denoted by the subscript. The absence of direct connections between hidden units in an RBM makes it's easy to get an unbiased sample of $\langle v_i h_j\rangle_{\text{data}}$. While, due to the difficulty of calculating Z_θ,

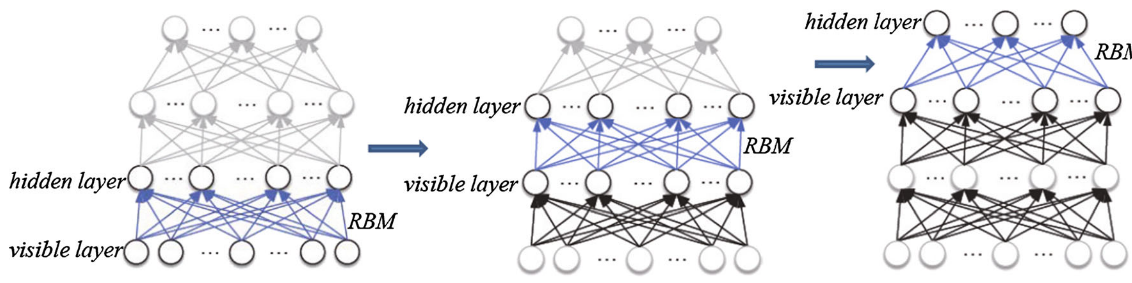


Fig. 1 Learning process of DBN. DBN uses a layer-wise training method by stacked RBMs

obtaining an unbiased sample of $\langle v_i h_j \rangle_{\text{model}}$ is quite difficult. Although it can be solved by starting from any random state of the visible units and performing alternating Gibbs sampling, it is an extremely time-consuming process. A much faster learning method is by the contrastive divergence (CD) [25].

2.2 Sparse learning for RBM and DBN

For each RBM module, if there is no restriction on the response of hidden neurons and on the connections between layers, an unstructured network will be generated, which leads to poor interpretability, cumbersome computational complexity and huge space resource consumption of the network. Since sparse regularization is more likely to learn the structural characteristics of the data [26, 27], in the following, we will introduce the sparse regularization method on an RBM to obtain sparse connections and sparse responses of the hidden layer neurons. Thus, we could get more distinct architecture of the network and more effective representation of the raw data. In order to achieve these aims, we introduce KL divergence on the hidden neurons to get their response sparseness and introduce L_1-norm on the connection weights. As we know, L_0-norm corresponds to the number of nonzero elements, which is the best sparse measurement, but solving it is an NP-hard problem. L_1-norm allows a large number parameters equal to zero, and it is one optimal convex approximation of L_0-norm [28]. At the same time, we also introduce L_2-norm on the connection weights. Since the degree of weights' attenuation is proportional to the values of the connection weights, L_2-norm penalty term can quickly reduce the weights and overcome over-fitting of the model.

The objective function of the sparse RBM can be improved as follows

$$\max_{\Theta} \mathcal{L}_{\text{new}}(\Theta) = \mathcal{L}(\Theta) - \frac{1}{2}\lambda_1 \|W\|_2^2 - \lambda_2 \sum_{j=1}^{N_h} KL(\rho \parallel p_j) - \lambda_3 \|W\|_1 \tag{1}$$

where the KL divergence, $KL(\rho \parallel p_j) = \rho \log \frac{\rho}{p_j} + (1-\rho)\log\frac{1-\rho}{1-p_j}$ is the relative entropy between two random variables with mean ρ and mean p_j. The KL divergence is used to measure the difference between two different distributions. KL is a monotonically increasing function of the distance between p_j and ρ, and $KL(\rho \parallel p_j) = 0$ when $p_j = \rho$. The sparse parameter ρ is usually taken as a small value close to 0. The purpose of introducing KL divergence is to force the average response value of the hidden neurons to be approximately equal to the default initial value ρ. Here, we take $p_j = \frac{1}{N_s}\sum_{q=1}^{N_s} \frac{1}{1+e^{-\sum_{i=1}^{N_v} v_i^{(q)} w_{ij} - \beta_j}}$ as the average activation probability of the j-neuron in the hidden layer with N_s samples, N_v is the number of nodes in the current visual layer, λ_2 and λ_3 denote the penalty coefficients for KL divergence and L_1-norm, respectively. The derivative calculation of KL divergence on the parameters is given in "Appendix 1." By performing stochastic steepest ascent and CD sampling process, the parameters of a sparse unsupervised RBM are updated as

$$\begin{aligned}\Delta W_{ij} =& \epsilon \cdot \left(\langle v_i h_j\rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{recon}}\right) - \lambda_1 W_{ij} \\ &- \lambda_2 \cdot \frac{1}{N_s}\left(-\frac{\rho}{p_j} + \frac{1-\rho}{1-p_j}\right)\sum_{q=1}^{N_s} \sigma_j^{(q)}(1-\sigma_j^{(q)})v_i^{(q)} \\ &- \lambda_3 \cdot sign(W_{ij})\end{aligned} \tag{2}$$

$$\Delta \alpha_i = \epsilon \cdot \left(\langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{recon}}\right) \tag{3}$$

$$\begin{aligned}\Delta \beta_j =& \epsilon \cdot \left(\langle h_j\rangle_{\text{data}} - \langle h_j \rangle_{\text{recon}}\right) \\ &- \lambda_2 \cdot \frac{1}{N_s}\left(-\frac{\rho}{p_j} + \frac{1-\rho}{1-p_j}\right)\sum_{q=1}^{N_s} \sigma_j^{(q)}(1-\sigma_j^{(q)})\end{aligned} \tag{4}$$

According to formula (2)–(4), we can obtain an RBM with sparse expression and sparse connection properties. Since a DBN is constructed by multilayer RBMs in a stack way [1], then based on the sparse RBMs, the entire unsupervised sparse DBN can be obtained by stacking those sparse RBMs. That is, we use the hidden layer of each previous RBM which has been trained sparsely as the visible layer of the next RBM, and then train it also in a sparse way by (2)–(4).

For the above unsupervised DBN pre-training process, the learned information is only derived from the data itself. To achieve a more satisfactory performance for a specific learning task, a fine-tuning procedure should be executed after the pre-training.

3 Sparse fine-tuning process: strengthening sparsity of BP algorithm

The commonly used fine-tuning procedure is the BP algorithm. BP is standard, useful, while it usually takes a long training time, and has the risk of over-fitting. When applied to deep networks, a so-called vanishing gradient phenomenon always happens: from up to down, the signal strength of error correction is getting weaker and weaker, and ultimately cause the training to fall into a local minimal. However, if we introduce sparse regularity terms of weights and responses of hidden neurons in BP, then based on the sparse DBN obtained in Sect. 2, all those bottlenecks will be effectively avoided. This is because the initial parameters learned from the pre-training sparse DBN have been already located in the optimal sparse subspace of the whole parameters' space, and the pre-training has also provided a good layered sparse feature expression for the data. Furthermore, with the use of the sparse BP algorithm to be derived, the fine-tuning procedure will effectively keep this sparse representation of the data learned from the pre-training process, rather than redistributing it to the entire network. All will promise the deep network to converge faster to the global optimal solution.

In what follows, based on the original derivation formulas of the traditional BP algorithm, we will present sparse-strengthened BP algorithm with sparse regularization terms on the response values of hidden neurons. With the introduction of some expression which can be considered as the residual forms for Kullback–Leibler divergence and Laplace penalty, the complicated gradient derivation can be simplified deeply. This sparse-strengthened BP algorithm will be conducive to a totally sparse learning of deep networks, and they will finalize a sparse interpretation of the data; thus, the learned sparse networks are promised to have a better performance including the generalization ability, identification accuracy and running times.

3.1 Traditional backpropagation algorithm

After the forward propagation process, a loss function is constructed to calculate the error between the actual output of the network and the desired output. The error will propagate down from the output layer to the input layer. The BP algorithm will evaluate the effect of the error on each layer, and especially calculate the gradient of this error on the weights as well as the biases of each layer backward.

Let the number of layers in the entire deep network be L and the number of neurons in the l-th layer be n_l $(l = 1, \ldots, L)$. The first layer corresponds to the input layer, the L-th layer is the output layer and other layers between them are hidden layers. $W^{(l)}$ is the connection weight matrix between the l-th and the $(l+1)$-th layer, $b^{(l)}$ is the bias of the l-th layer, and $a^{(l)}$ is the activation vector of neurons in the l-th layer. When $l = 1$, $a^{(1)} = \{a_{qj}^{(1)}\}$ representing the value of the j-th neuron for the q-th sample. Assume that N is the size of the sample set. Let $e = [1, \ldots, 1]^{\mathrm{T}}$ be an N-dim vector. For each $l = 1, 2, \ldots, L-1$, let $a^{(l)} = [a^{(l)}, e]$, $W^{(l)} = [W^{(l)}; b^{(l)}]$, $z^{(l)} = a^{(l-1)} \cdot W^{(l-1)}$, in which F is the activation mapping of this layer, and $F(z^{(l)}) = (f_1(z^{(l)}), f_2(z^{(l)}), \ldots, f_N(z^{(l)}))^{\mathrm{T}}$. Suppose $f_i = f_j \triangleq f$ $(\forall i, j = 1, 2, \ldots, N)$. Generally, f is selected as the sigmoid function.

The loss function of the traditional BP is

$$J(W) = \frac{1}{2N} \sum_{q=1}^{N} \sum_{j=1}^{n_L} (a_{qj}^{(L)} - y_{qj})^2 \tag{5}$$

where N is the training sample size, y_{qj} is the target output of the j-th neuron in the output layer corresponding to the q-th sample, and $a_{qj}^{(L)}$ is the actual output of it. Let η_1 be the learning rate, then by introducing the residual formula, the updating formula for the network parameters is

$$W^{(l)} = W^{(l)} - \eta_1 \cdot \frac{1}{N} \sum_{q=1}^{N} \Delta W_q^{(l)}\_J \tag{6}$$

The definitions of residual formula and $\Delta W_q^{(l)}\_J$, and the whole derivation of (6) can be found in "Appendix 2."

It should be noticed that the most important reason of BP being widely applied lies in that the residual formula of it is quite concise and unified for arbitrary layers. Thus, in order to obtain an easily applied updating formula of BP which adds sparse regularization in the loss function, one should also get a general residual formula at first.

3.2 Sparsity-strengthened backpropagation algorithm

As the most commonly used method for updating parameters of a neural network, the traditional BP algorithm has been widely used in deep learning. When applied it to fine-tune the network pre-trained by the sparse DBN, however, the response value of each hidden layer neuron will lie on the whole [0, 1] interval again, and the connection of the entire network will no longer be sparse. That means, the sparse representation of the original data learned through sparse DBN will not hold, and they will be redistributed on the entire network architecture once more. The sparse structure will be destroyed and the role or response of each hidden neuron in the data representation will no longer be clear.

In order to intensify the sparse expression ability of deep learning, especially clarify those neurons which play key positions in the data expression, and also improve the generalization ability of the BP algorithm, we will introduce some restrictions on the responsiveness of hidden neurons to obtain a lower complexity of the networks in the BP training process. In detail, for the loss function of the network, the penalty items that describe the sparse responsiveness are added, which include a KL divergence regular term and a Laplace penalty term (i.e., the L_1-norm constraint) on the responses of the hidden neurons. These restrictions will be conducive to sparse learning of BP, and they will finalize sparse interpretation of the data.

Let $J_{KL}(W)$ denote the sum of KL divergence on all hidden neurons' responses (where W is the hidden variable), $J_{RL_1}(W)$ denote the sum of all hidden neurons' L_1 norm (W still is the hidden variable), then the improved loss function of the sparse BP algorithm is:

$$\begin{aligned} J_{sps}(W) &= J(W) + \tau_1 J_{KL}(W) + \tau_2 J_{RL_1}(W) \\ &= J(W) + \tau_1 \sum_{l=2}^{L-1} \sum_{k=1}^{n_l} KL(\rho \parallel p_k^{(l)}) \\ &\quad + \tau_2 \sum_{l=2}^{L-1} \sum_{k=1}^{n_l} |p_k^{(l)}| \end{aligned} \tag{7}$$

where τ_1 and τ_2 are parameters which control the two sparsity penalty items, and they are used to balance the total error of the model itself and the structural complexity of the network. $p_k^{(l)} = \frac{1}{N}\sum_{q=1}^{N} a_{qk}^{(l)}$ denotes the average activation degree of the k-th neurons in the l-th layer, $a_{qk}^{(l)} = f(z_{qk}^{(l)})$ is the response value of the k-th neuron in the l-th layer with the q-th input sample.

In what follows, we will surmount the complexity of gradient calculation, and give the gradient formulas of both $J_{KL}(W)$ and $J_{RL_1}(W)$ on W, respectively, and then achieve a unified updating formulas for parameters. Based on such a strengthening sparsity BP algorithm, the response sparsity of hidden neurons for the whole network can be obtained finally.

3.2.1 The gradient calculation formula of $J_{KL}(W)$ for each layer

For $J_{KL}(W)$, we have

$$\begin{aligned} \frac{\partial J_{KL}(W)}{\partial W_{ij}^{(L-1)}} &= \frac{\partial \sum_{k=1}^{n_L} KL(\rho \parallel p_k^{(L)})}{\partial p_j^{(L)}} \frac{\partial p_j^{(L)}}{\partial W_{ij}^{(L-1)}} \\ &= \frac{1}{N}\sum_{q=1}^{N}\left(-\frac{\rho}{p_j^{(L)}} + \frac{1-\rho}{1-p_j^{(L)}}\right) f'(z_{qj}^{(L)}) a_{qi}^{(L-1)} \end{aligned} \tag{8}$$

Let

$$S_{qj}^{(l)}(L-l+1) = \left(-\frac{\rho}{p_j^{(l)}} + \frac{1-\rho}{1-p_j^{(l)}}\right) f'(z_{qj}^{(l)})$$

then, (8) can be simplified as

$$\frac{\partial J_{KL}(W)}{\partial W_{ij}^{(L-1)}} = \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L)}(1) \cdot a_{qi}^{(L-1)}$$

In general, let

$$S_{qj}^{(l)}(M) = \begin{cases} \left(-\frac{\rho}{p_j^{(l)}} + \frac{1-\rho}{1-p_j^{(l)}}\right) f'(z_{qj}^{(l)}), & M = L+1-l \\ \sum_{k=1}^{n_{l+1}} W_{jk}^{(l)} \cdot S_{qk}^{(l+1)}(M) \cdot f'(z_{qj}^{(l)}) & l = L-1, \ldots, 1 \\ & M = 1, 2, \ldots, L-l \end{cases}$$

$$S_{qj}^{(l)} = \sum_{M=1}^{L+1-l} S_{qj}^{(l)}(M)$$

Then, we have

1) The gradient calculation formula of $J_{KL}(W)$ on $W^{(L-1)}$ is

$$\frac{\partial J_{KL}(W)}{\partial W_{ij}^{(L-1)}} = \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L)} \cdot a_{qi}^{(L-1)} \tag{9}$$

2) The gradient calculation formula of $J_{KL}(W)$ on $W^{(L-2)}$ is

$$\begin{aligned} &\frac{\partial J_{KL}(W)}{\partial W_{ij}^{(L-2)}} \\ &= \sum_{k=1}^{n_L} \frac{\partial KL(\rho \parallel p_k^{(L)})}{\partial p_k^{(L)}} \sum_{q=1}^{N} \frac{\partial p_k^{(L)}}{\partial a_{qk}^{(L)}} \frac{\partial a_{qk}^{(L)}}{\partial a_{qj}^{(L-1)}} \frac{\partial a_{qj}^{(L-1)}}{\partial W_{ij}^{(L-2)}} \\ &\quad + \frac{\partial \sum_{k=1}^{n_{L-1}} KL(\rho \parallel p_k^{(L-1)})}{\partial p_j^{(L-1)}} \frac{\partial p_j^{(L-1)}}{\partial W_{ij}^{(L-2)}} \\ &= \frac{1}{N}\sum_{q=1}^{N}\sum_{k=1}^{n_L}\left(-\frac{\rho}{p_k^{(L)}} + \frac{1-\rho}{1-p_k^{(L)}}\right) f'(z_{qk}^{(L)}) \\ &\quad \cdot W_{jk}^{(L-1)} f'(z_{qj}^{(L-1)}) a_{qi}^{(L-2)} \\ &\quad + \frac{1}{N}\sum_{q=1}^{N}\left(-\frac{\rho}{p_j^{(L-1)}} + \frac{1-\rho}{1-p_j^{(L-1)}}\right) f'(z_{qj}^{(L-1)}) a_{qi}^{(L-2)} \\ &= \frac{1}{N}\sum_{q=1}^{N}\sum_{k=1}^{n_L} S_{qk}^{(L)}(1) \cdot W_{jk}^{(L-1)} f'(z_{qj}^{(L-1)}) a_{qi}^{(L-2)} \\ &\quad + \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-1)}(2) \cdot a_{qi}^{(L-2)} \\ &= \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-1)}(1) \cdot a_{qi}^{(L-2)} + \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-1)}(2) \cdot a_{qi}^{(L-2)} \\ &= \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-1)} \cdot a_{qi}^{(L-2)} \end{aligned} \tag{10}$$

3) The gradient calculation formula of $J_{KL}(W)$ on $W^{(L-3)}$ is

$$
\begin{aligned}
\frac{\partial J_{KL}(W)}{\partial W_{ij}^{(L-3)}} =& \sum_{k=1}^{n_L} \frac{\partial KL(\rho \parallel p_k^{(L)})}{\partial p_k^{(L)}} \sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}} \frac{\partial p_k^{(L)}}{\partial a_{qk}^{(L-1)}} \frac{\partial a_{qk}^{(L-1)}}{\partial a_{qt}^{(L-1)}} \\
&\cdot \frac{\partial a_{qt}^{(L-1)}}{\partial a_{qj}^{(L-2)}} \frac{\partial a_{qj}^{(L-2)}}{\partial W_{ij}^{(L-3)}} + \sum_{k=1}^{n_{L-1}} \frac{\partial KL(\rho \parallel p_k^{(L-1)})}{\partial p_k^{(L-1)}} \sum_{q=1}^{N} \frac{\partial p_k^{(L-1)}}{\partial a_{qj}^{(L-2)}} \frac{\partial a_{qj}^{(L-2)}}{\partial W_{ij}^{(L-3)}} \\
&+ \frac{\partial \sum_{j=1}^{n_{L-2}} KL(\rho \parallel p_j^{(L-2)})}{\partial p_j^{(L-2)}} \frac{\partial p_j^{(L-2)}}{\partial W_{ij}^{(L-3)}} \\
=& \frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}\sum_{k=1}^{n_L}\left(-\frac{\rho}{p_k^{(L)}}+\frac{1-\rho}{1-p_k^{(L)}}\right) \\
&\cdot f'(z_{qk}^{(L)})W_{tk}^{(L-1)}f'(z_{qt}^{(L-1)})W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)} \\
&+\frac{1}{N}\sum_{q=1}^{N}\sum_{k=1}^{n_{L-1}}\left(-\frac{\rho}{p_k^{(L-1)}}+\frac{1-\rho}{1-p_k^{(L-1)}}\right) \\
&\cdot f'(z_{qk}^{(L-1)})W_{jk}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)} \\
&+\frac{1}{N}\sum_{q=1}^{N}\left(-\frac{\rho}{p_j^{(L-2)}}+\frac{1-\rho}{1-p_j^{(L-2)}})f'(z_{qj}^{(L-2)}\right)a_{qi}^{(L-3)} \\
=& \frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}\sum_{k=1}^{n_L} S_{qk}^{(L)}(1)\cdot W_{tk}^{(L-1)}\cdot f'(z_{qt}^{(L-1)})W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)} \\
&+\frac{1}{N}\sum_{q=1}^{N}\sum_{k=1}^{n_{L-1}} S_{qk}^{(L-1)}(2)\cdot W_{jk}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)} \\
&+\frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-2)}(3)\cdot a_{qi}^{(L-3)} \\
=& \frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}\sum_{k=1}^{n_L}(W_{tk}^{(L-1)}S_{qk}^{(L)}(1)\cdot f'(z_{qt}^{(L-1)}))\cdot W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)} \\
&+\frac{1}{N}\sum_{q=1}^{N}\sum_{k=1}^{n_{L-1}} W_{jk}^{(L-2)}S_{qk}^{(L-1)}(2)\cdot f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)} \\
&+\frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-2)}(3)\cdot a_{qi}^{(L-3)} \\
=& \frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}} S_{qt}^{(L-1)}(1)\cdot W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)} \\
&+\frac{1}{N}\sum_{q=1}^{N}\sum_{k=1}^{n_{L-1}} S_{qj}^{(L-2)}(2)\cdot a_{qi}^{(L-3)} + \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-2)}(3)\cdot a_{qi}^{(L-3)} \\
&= \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-2)}(1)\cdot a_{qi}^{(L-3)} + \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-2)}(2)\cdot a_{qi}^{(L-3)} \\
&+\frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-2)}(3)\cdot a_{qi}^{(L-3)} = \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(L-2)}\cdot a_{qi}^{(L-3)}
\end{aligned} \tag{11}
$$

Generally, one can get that

$$\frac{\partial J_{KL}(W)}{\partial W_{ij}^{(l)}} = \frac{1}{N}\sum_{q=1}^{N} S_{qj}^{(l+1)}\cdot a_{qi}^{(l)} \quad (l = L-1,\ldots,1) \tag{12}$$

For each sample $q = 1,2,\ldots,N$, let $S_q^{(l)}$ be the row vector which is composed by $\{S_{qj}^{(l)}\}_{j=1,\ldots,n_l}$, then by (12), we calculate $\Delta W_q^{(l)}\_KL = (a^{(l)})_q^{\mathrm{T}}\cdot S_q^{(l+1)}$ for each sample, and thus we have update formula for the network parameters in a concise matrix form

$$W^{(l)} = W^{(l)} - \eta_2\cdot\frac{1}{N}\sum_{q=1}^{N}\Delta W_q^{(l)}\_KL \tag{13}$$

where η_2 is the learning rate.

3.2.2 The gradient calculation formula of $J_{RL_1}(W)$ on W for each layer

Let

$$R_{qj}^{(l)}(M) = \begin{cases} f'(z_{qj}^{(l)}), & M = L+1-l \\ \sum_{k=1}^{n_{l+1}} W_{jk}^{(l)}\cdot R_{qk}^{(l+1)}(M)\cdot f'(z_{qj}^{(l)}) & l = L-1,\ldots,1 \\ & M = 1,2,\ldots,L-l \end{cases}$$

$$R_{qj}^{(l)} = \sum_{M=1}^{L+1-l} R_{qj}^{(l)}(M)$$

On noting that $p_k^{(l)}$ is the average activation degree of the k-th neurons in the l-th layer, thus each $p_k^{(l)}$ is nonnegative, and we have the following recursive formula.

1) The gradient calculation formula of $J_{RL_1}(W)$ on $W^{(L-1)}$ is

$$
\begin{aligned}
\frac{\partial J_{RL_1}(W)}{\partial W_{ij}^{(L-1)}} &= \frac{\partial \sum_{k=1}^{n_L} p_k^{(L)}}{\partial W_{ij}^{(L-1)}} = \frac{1}{N}\sum_{q=1}^{N} f'(z_{qj}^{(L)})a_{qi}^{(L-1)} \\
&= \frac{1}{N}\sum_{q=1}^{N} R_{qj}^{(L)} a_{qi}^{(L-1)}
\end{aligned} \tag{14}
$$

2) The gradient calculation formula of $J_{RL_1}(W)$ on $W^{(L-2)}$ is

$$
\begin{aligned}
\frac{\partial J_{RL_1}(W)}{\partial W_{ij}^{(L-2)}} &= \sum_{k=1}^{n_L}\sum_{q=1}^{N}\frac{\partial p_k^{(L)}}{\partial a_{qj}^{(L-1)}}\frac{\partial a_{qj}^{(L-1)}}{\partial W_{ij}^{(L-2)}} + \frac{\partial \sum_{k=1}^{n_{L-1}} p_k^{(L-1)}}{\partial W_{ij}^{(L-2)}} \\
&= \frac{1}{N}\sum_{q=1}^{N}\sum_{k=1}^{n_L} f'(z_{qk}^{(L)})W_{jk}^{(L-1)}f'(z_{qj}^{(L-1)})a_{qi}^{(L-2)} \\
&\quad + \frac{1}{N}\sum_{q=1}^{N} f'(z_{qj}^{(L-1)})a_{qi}^{(L-2)} \\
&= \frac{1}{N}\sum_{q=1}^{N}\sum_{k=1}^{n_L} R_{qk}^{(L)}(1)\cdot W_{jk}^{(L-1)}f'(z_{qj}^{(L-1)})a_{qi}^{(L-2)} \\
&\quad + \frac{1}{N}\sum_{q=1}^{N} R_{qj}^{(L-1)}(2)\cdot a_{qi}^{(L-2)} \\
&= \frac{1}{N}\sum_{q=1}^{N} R_{qj}^{(L-1)} a_{qi}^{(L-2)}
\end{aligned} \tag{15}
$$

3) The gradient calculation formula of $J_{RL_1}(W)$ on $W^{(L-3)}$ is

$$\begin{aligned}
&\frac{\partial J_{RL_1}(W)}{\partial W_{ij}^{(L-3)}}\\
&=\sum_{k=1}^{n_L}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}\frac{\partial p_k^{(L)}}{\partial a_{qt}^{(L-1)}}\frac{\partial a_{qt}^{(L-1)}}{\partial a_{qj}^{(L-2)}}\frac{\partial a_{qj}^{(L-2)}}{\partial W_{ij}^{(L-3)}}\\
&\quad+\sum_{t=1}^{n_{L-1}}\sum_{q=1}^{N}\frac{\partial p_t^{(L-1)}}{\partial a_{qj}^{(L-2)}}\frac{\partial a_{qj}^{(L-2)}}{\partial W_{ij}^{(L-3)}}+\frac{\partial\sum_{s=1}^{n_{L-2}}p_s^{(L-2)}}{\partial W_{ij}^{(L-3)}}\\
&=\frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}\sum_{k=1}^{n_L}f'(z_{qk}^{(L)})\\
&\quad\cdot W_{tk}^{(L-1)}f'(z_{qt}^{(L-1)})W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)}\\
&\quad+\frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}f'(z_{qt}^{(L-1)})W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)}\\
&\quad+\frac{1}{N}\sum_{q=1}^{N}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)}\\
&=\frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}\sum_{k=1}^{n_L}R_{qk}^{(L)}(1)\cdot W_{tk}^{(L-1)}f'(z_{qt}^{(L-1)})\\
&\quad\cdot W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)}\\
&\quad+\frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}R_{qt}^{(L-1)}(2)\cdot W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)}\\
&\quad+\frac{1}{N}\sum_{q=1}^{N}R_{qj}^{(L-2)}(3)\cdot a_{qi}^{(L-3)}\\
&=\frac{1}{N}\sum_{q=1}^{N}\sum_{t=1}^{n_{L-1}}R_{qt}^{(L-1)}(1)\cdot W_{jt}^{(L-2)}f'(z_{qj}^{(L-2)})a_{qi}^{(L-3)}\\
&\quad+\frac{1}{N}\sum_{q=1}^{N}R_{qj}^{(L-2)}(2)\cdot a_{qi}^{(L-3)}+\frac{1}{N}\sum_{q=1}^{N}R_{qj}^{(L-2)}(3)\cdot a_{qi}^{(L-3)}\\
&=\frac{1}{N}\sum_{q=1}^{N}R_{qj}^{(L-2)}a_{qi}^{(L-3)}
\end{aligned}\tag{16}$$

In general, we have

$$\frac{\partial J_{RL_1}(W)}{\partial W_{ij}^{(l)}}=\frac{1}{N}\sum_{q=1}^{N}R_{qj}^{(l+1)}\cdot a_{qi}^{(l)}\quad(l=L-1,\ldots,1)\tag{17}$$

For each sample $q=1,2,\ldots,N$, let $R_q^{(l)}$ be the row vector which is composed by $\{R_{qj}^{(l)}\}_{j=1,\ldots,n_l}$, then by (17), we calculate $\Delta W_q^{(l)}\_RL_1=(a^{(l)})_q^{\mathrm{T}}\cdot R_q^{(l+1)}$ for each sample, and thus get the update formula for the network parameters in a matrix form:

$$W^{(l)}=W^{(l)}-\eta_3\cdot\frac{1}{N}\sum_{q=1}^{N}\Delta W_q^{(l)}\_RL_1\tag{18}$$

where η_3 is the learning rate.

In all, by (23), (7), (13) and (18), we derived a unified BP update formula for parameters of the deep network with strengthening sparsity

$$\begin{aligned}W^{(l)}=&W^{(l)}-\frac{1}{N}\sum_{q=1}^{N}(\eta_1\Delta W_q^{(l)}\_J\\&+\tau_1\eta_2\Delta W_q^{(l)}\_KL+\tau_2\eta_3\Delta W_q^{(l)}\_RL_1)\end{aligned}\tag{19}$$

where τ_1 and τ_2 are the parameters controlling the sparsity penalty items, and η_1-η_3 are the corresponding learning rates for the loss function and the regularizer.

The goal of the put forward sparsity restrictions in BP procedure is to divide the neurons of hidden layers into two categories, one class with response values close to 0, and those neurons are called as non-response neurons; another with relatively large response values, known as the response neurons, and the neurons in this class are of only a small part. These restrictions are conducive to the sparse learning of deep networks, and they will finalize a sparse interpretation of the data.

Due to the difficulty of deducing a unified and coding easily updating formula for the parameters in fine-tuning process, especially when the loss function contains complex regularization terms of responses on hidden neurons, very few researches have been done on this field. Ng [22] incorporates KL divergence into BP process to get the responses sparsity for autoencoders, but the result is deduced for only one step of chain rules of derivative and it is a special case of formula (12) and (13). In [21], the authors use L_1-norm constrains on responses of hidden neurons in BP, while the updating formula is not in a concise way. In addition, lacking a form like the residual in backpropagation, it is not quite suitable for coding. The derived unified updating formula (19) finds a laconic solution for DNN with arbitrary layers when the loss function contains regularization terms of hidden neurons' responses.

3.3 SRS-DNN: DNN with strengthening response sparsity

With the unsupervised sparse learning of DBN presented in Sect. 2, and further with the sparse-strengthened fine-tuning process in Sect. 3.2, we have proposed a novel deep neural network with strengthening response sparsity (SRS-DNN). In the pre-training stage, SRS-DNN applies formula (2)–(4) to achieve an unsupervised sparse DBN with sparse expression and sparse connection properties. In the fine-tuning stage, SRS-DNN uses the sparsity-strengthened BP algorithm and the parameters updating formula (19). Thus, SRS-DNN maintains the sparse structure of DBN, and assures the data to be described layer-wise in a sparse way, with a more powerful capacity of discernment.

By such a SRS-DNN learning, the essential sparsity characteristics of human brains can be simulated, and finally the network will own a sparse topology structure with sparse responsiveness. The method proposed here guarantees the deep neural networks to have a high processing, feature extraction, storage, promotion as well as generalization capability.

4 Experiments

In this section, we use the Fashion-MNIST database to show the validation of the sparse deep learning architecture learned in Sect. 3. Zalando (the e-commerce company) develop a novel image classification dataset called Fashion-MNIST in hopes of replacing MNIST, and it is intended to serve as a new benchmark of machine learning algorithms. Fashion-MNIST contains of 70,000 fashion products from ten categories, with 7000 images per category. The training set has 60,000 images and the test set has 10,000 images. All of the digits have been size-normalized and centered in a fixed-size image. Each image has a total of 28×28 grayscale pixels, and the 784-dimensional pixels are listed into a vector. For each pixel, it has a single pixel value associated with it, and the value is between 0 (represents white), and 255 (represents black). Intermediate pixel values represent shades of gray. The dataset is freely available at https://github.com/zalandoresearch/fashion-mnist.

For the training data with 60,000 images of ten categories, the DBN is pre-trained in a sparse way by formulas (2)–(4) in Sect. 2.2, and it contains five layers, in which one is a visible layer, the other three are hidden layers with 2100, 1000, 500 nodes, respectively, and the top one is a decision layer with ten nodes. Further, the sparse fine-tuning process is applied by the sparsity-strengthened backpropagation algorithm, with the improved sparse loss function described in (7). Based on the unified gradient calculation formula (19), finally, a sparsity-strengthened deep neural network is trained, with quite good performances in many aspects.

The parameters used in the SRS-DNN are as follows: τ is 0.1 and the max epoch in BP is 220, the sparse parameter ρ in DBN is 0.01, and that in BP is 0.05. Some other sparse parameters, i.e., λ_2 and λ_3, which are the parameters of sparsity penalty items for KL and L_1 in RBM, are chosen as 0.005 and 0.0001, respectively. τ_1 and τ_2, which control the sparsity penalty items of KL and L_1 in BP, are 0.0001 and 0.0002. The above sparse parameters are selected from several parameter sets listed in Table 6 (see, "Appendix 3").

4.1 Classification performances on Fashion-MNIST database

4.1.1 Classification accuracy rates of fifteen methods

In Table 1, several typical classifiers and their classification accuracy rates (AR) on Fashion-MNIST testing data from [29] are listed. The classifiers are Decision Tree Classifier (DTC), Passive Aggressive Classifier (PAC), Extra Tree Classifier (ETC), Perceptron (Pct), Gaussian Naive Bayesian (GNB), Random Forest Classifier (RFC), Gradient Boosting Classifier (GBC), Stochastic Gradient Descent Classifier (SGDC), K-Neighbors Classifier (KNC), Support Vector Classification (SVC), Linear SVC (LSVC), Support Vector Machine (SVM), Logistic Regression (LR), Multilayer Perception Classifier (MLPC). All of the classifiers are applied for 30 times, and the average AR of them is adopted. The results are shown in Table 1. It can be seen that SRS-DNN has the best performance, especially in comparing with the classical methods such as SVM, GBC, KNC, LR, MLPC, RFC and SVC; the AR has been increased as 7.37%, 7.90%, 7.01%, 7.98%, 5.84%, 6.16% and 10.67%, respectively. Besides comparing with other 14 classifiers listed in Table 1, we also use DNN on this database. Some comparisons of detailed performance for both DNN and SRS-DNN will be presented in Sect. 4.2.

4.1.2 Classification results on all ten categories data with SVM and SRS-DNN

To specify the performance of SRS-DNN on all ten categories data, we further evaluate the AR for different categories in the testing set. By experiment, one can find that SVM owns quite well AR on each ten categories data, so we use both SVM and SRS-DNN to appraise the validation of SRS-DNN. The comparison results are shown in Table 2 and Fig. 2. From both of them, it can be observed that

Table 1 Testing accuracy (TA) of fifteen classifiers

Classifier	TA	Classifier	TA
DTC	0.7864	PAC	0.7747
ETC	0.7587	Pct	0.754
GNB	0.511	RFC	0.8538
GBC	0.84	SGDC	0.8152
KNC	0.847	SVC	0.819
LSVC	0.7244	SVM	0.8442
LR	0.8394	SRS-DNN	**0.9064**
MLPC	0.8564		

Bold value indicates the best testing accuracy among all the classifiers

Table 2 Classification accuracy rates for ten categories data by SVM and SRS-DNN

Labels	Accuracy rate (%)		Labels	Accuracy rate (%)	
	SVM	SRS-DNN		SVM	SRS-DNN
T-shirt	0.814	0.865	Sandal	0.934	0.977
Trouser	0.959	0.987	Shirt	0.555	0.726
Pullover	0.770	0.852	Sneaker	0.934	0.962
Dress	0.849	0.905	Bag	0.914	0.983
Coat	0.769	0.832	Ankle boot	0.944	0.968

compared with SVM, the accuracy rates of SRS-DNN improve significantly for each categories of the dataset.

It is interesting to note that in the ten categories of data, some categories (say, Pullover, Coat and Shirt) are very different to be classified even by our human being (see, e.g., Fig. 3). For all those recognition tasks, SRS-DNN, however, performs very well. For the Shirt data, SRS-DNN obtains 0.726 AR, i.e., the improvement is about 20%; from 0.555 to 0.726. At the same time, the accuracy rates by SRS-DNN on the other two categories also are better than those by SVM. It thus shows the reliable classification ability of SRS-DNN.

4.1.3 Visualization of Fashion-MNIST by t-SNE

In order to further demonstrate the capability of SRS-DNN on discovering the essential difference for ten categories data, we use the t-distributed stochastic neighbor embedding (t-SNE) projection method [30] to show the distributions of them in the original high-dimensional space, and in the higher three representation spaces by SRS-DNN (i.e., the activation performances of the data in the three hidden layers).

Figure 4 shows that with the increase in layers, i.e., from down to up in SRS-DNN, the abstract features in higher layers are more distinctive than those in lower layers, which are very conducive to intensify the data of a same category to be tied together and highlight the differences between categories. It thus shows clearly that the original data for each category are changed from the initial scattered way to aggregate tightly, especially for Pullover, Coat and Shirt. These observations are coincident with the results in Sect. 4.1.2.

4.2 The sparse performances and time complexity of both SRS-DNN and DNN

In Sect. 4.1.1, we have already known that the AR of SRS-DNN on the testing data is 90.64%, which is the best one among the 15 classifiers. In addition, on the same testing data, we also get the AR of original DNN. The parameters of the network are the same, i.e., the DNN has three hidden layers with 2100, 1000, 500 nodes and a decision layer, with τ being 0.1 and the max epoch in BP being 220.

The classification accuracy of DNN is 89.11%, which is still lower than the result of SRS-DNN. Some other crucial

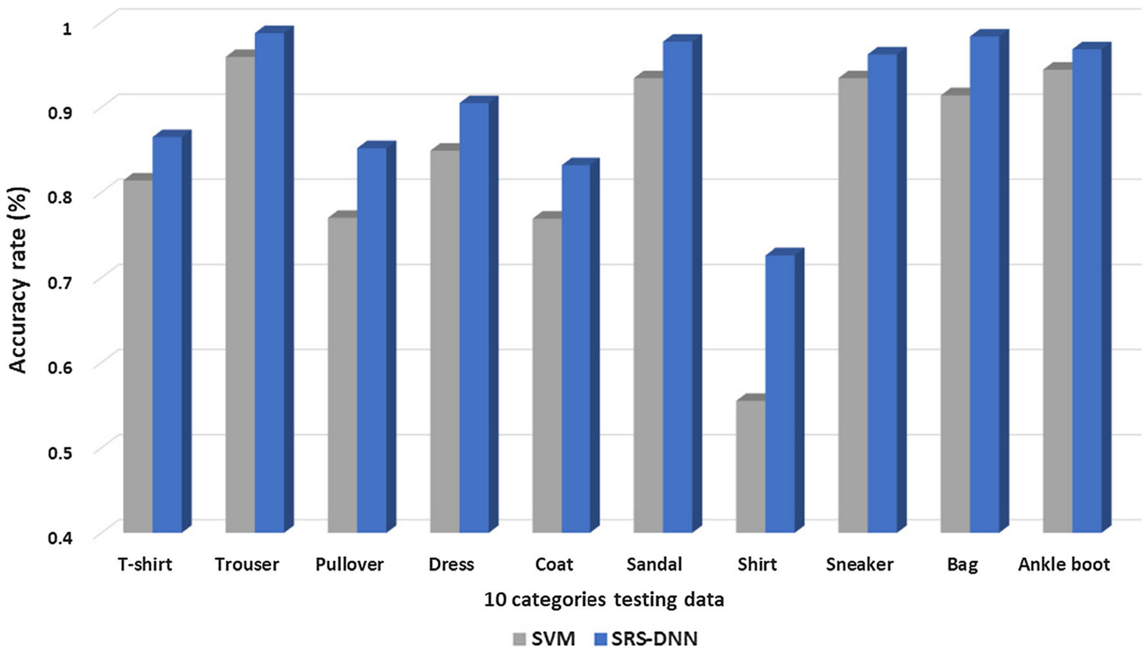


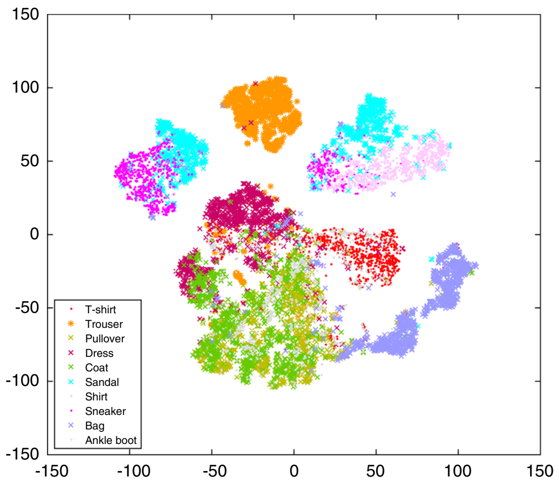
Fig. 2 Classification accuracy for ten categories of data in SVM and SRS-DNN–SVM

(a) Pullover

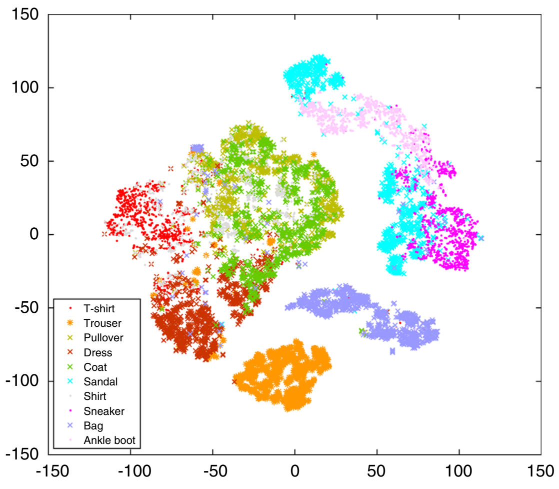
(b) Coat

(c) Shirt

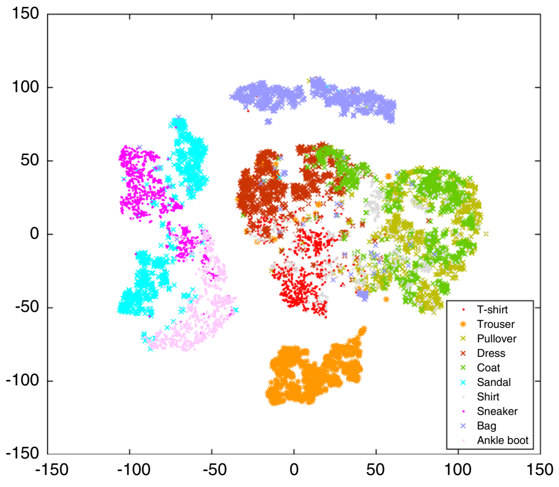
Fig. 3 Grayscale images of three quite similar categories (partial samples)



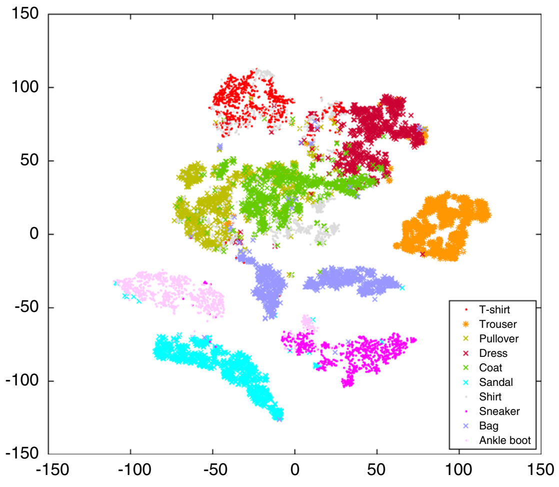
(a)



(b)



(c)



(d)

Fig. 4 Different space representations for Fashion-MNIST data in different layers of SRS-DNN by using t-SNE projection. **a** Visualization results of the original data. **b** Visualization results on the abstract features learned by the first hidden layer. **c** Visualization results on the abstract features learned by the second hidden layer. **d** Visualization results on the abstract features learned by the third hidden layer

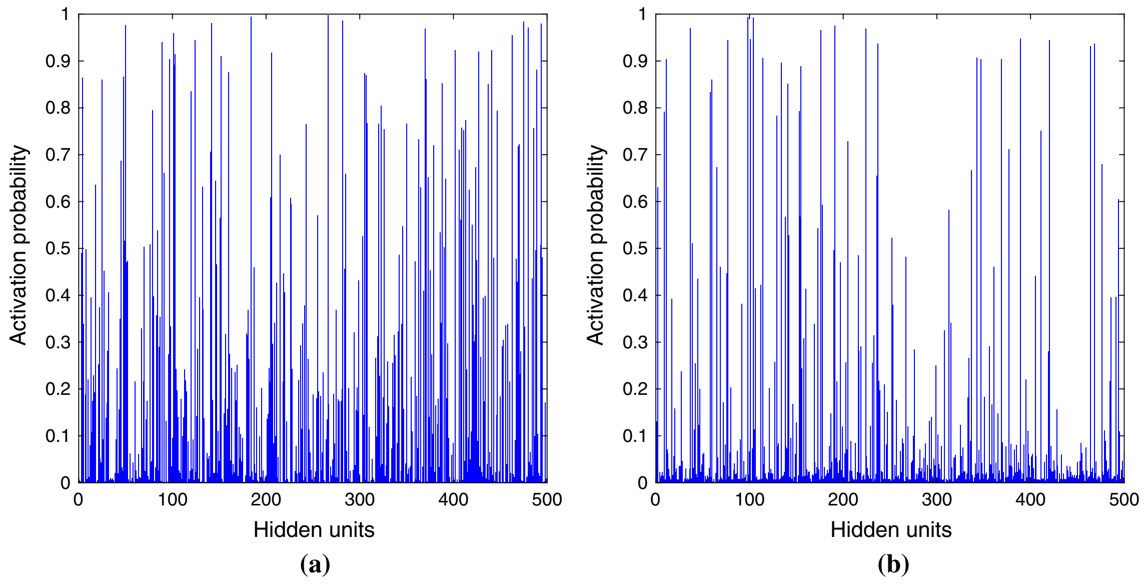


Fig. 5 Activation probability of hidden neurons for DNN and SRS-DNN with random selected samples. **a** The activation probability for each hidden neurons in the third layer of DNN, and **b** the results of SRS-DNN

Table 3 Nonzero response ratio of different categories data by DNN and SRS-DNN

	Layer 1		Layer 2		Layer 3	
	DNN	SRS-DNN	DNN	SRS-DNN	DNN	SRS-DNN
T-shirt	0.2757	0.1586	0.2510	0.1740	0.2510	0.1740
Trouser	0.2714	0.1519	0.1060	0.1170	0.1060	0.1170
Pullover	0.2705	0.1614	0.2810	0.2220	0.2810	0.2220
Dress	0.3086	0.1895	0.1900	0.1620	0.1900	0.1620
Coat	0.2252	0.1081	0.1550	0.1390	0.1550	0.1390
Sandal	0.2562	0.1757	0.1220	0.1080	0.1220	0.1080
Shirt	0.2229	0.1071	0.1560	0.1410	0.1560	0.1410
Sneaker	0.2110	0.1043	0.0900	0.0710	0.0900	0.0710
Bag	0.3071	0.1800	0.2890	0.2170	0.2890	0.2170
Ankle boot	0.2648	0.1510	0.2230	0.1610	0.2230	0.1610

performances of both DNN and SRS-DNN are also compared in this subsection.

4.2.1 Response sparsity of hidden neurons

SRS-DNN has more sparse activation performances compared with DNN. In particular, the average nonzero response ratio of three layers reduces as 43.08%, 18.84% and 32.47%, respectively. Figure 5 shows the response capability of hidden neurons for original DNN and SRS-DNN. In Fig. 5, the horizontal axis is the hidden units and the vertical axis represents the corresponding activation probabilities. It shows that SRS-DNN has much sparser responsiveness than DNN.

In addition, results for nonzero response ratios on different layers for Fashion-MNIST data are shown in Table 3.

Further, Fig. 6 shows that for the hidden neurons of SRS-DNN, 77.2% of them have almost no response. For DNN, near a half, i.e., 49.4%, hidden neurons are active. Thus, SRS-DNN has a very strong response sparsity. At the same time, most of the responses with big values are fallen in the interval [0.9, 1], which indicates that the SRS-DNN achieves the response sparsity simultaneously, namely they are either quite small or near 1. It can be seen that the method effectively achieves the number sparseness of response neurons as well as the response values sparseness of them.

4.2.2 Sparsity of weights

Further, we show that by the presented strengthening response sparsity method, the weights sparsity learned in

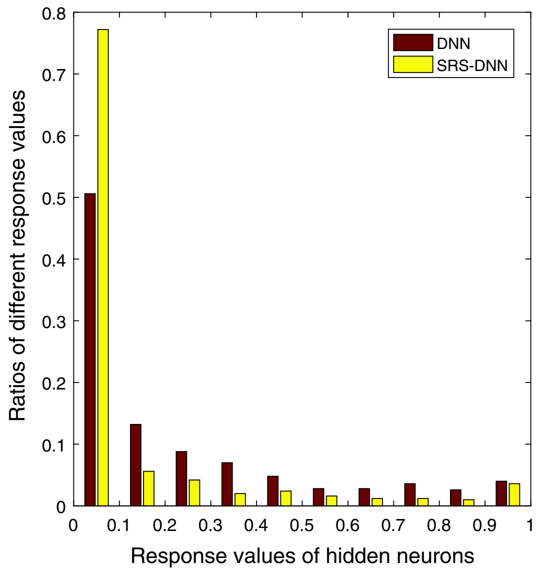


Fig. 6 Response ratio in different range of response values

Table 4 Connection sparsity of DNN and SRS-DNN

	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$
DNN	0.9747	0.991	0.9954
SRS-DNN	0.7862	0.4402	0.4062

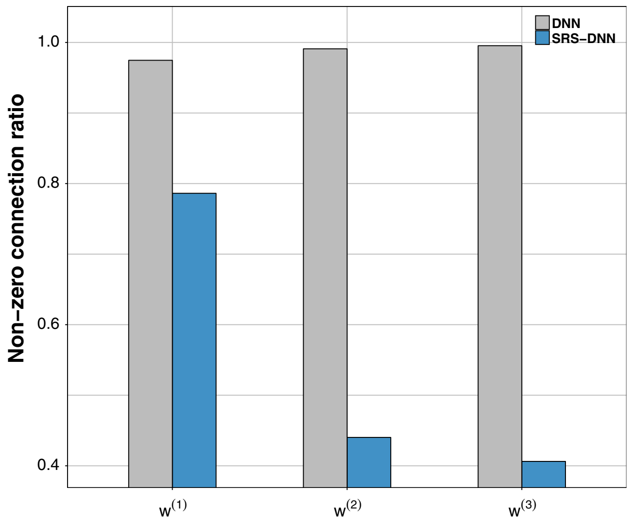


Fig. 7 Nonzero ratio of connections in DNN and SRS-DNN

the unsupervised sparse DBN process can also be held. We compare the weights sparsity of DNN and SRS-DNN in Table 4 and Fig. 7. $w^{(1)}$, $w^{(2)}$ and $w^{(3)}$ refer the ratios of connection weights with nonzero values between different layers.

From Table 4 and Fig. 7, we can see that the SRS-DNN method proposed in this paper not only obtains the response sparsity of hidden neurons, but also retains the connection sparsity of the network. They are due to the advantages of the sparse strengthening method.

Thus, a sparse deep learning model which has the sparse topology structure and the sparse learning capability of the data is established successfully.

Table 5 Comparison of parameters and running time for DNN and SRS-DNN

Methods	Pre-training		Fine-tuning		Time (s)
	λ_2	λ_3	τ_1	τ_2	
DNN	–	–	–	–	97255.972
SRS-DNN	0.005	0.0001	0.0001	0.0002	42686.165

4.2.3 Time complexity

As it has been shown above, for SRS-DNN, in the unsupervised learning phase, i.e., DBN, the KL divergence of hidden layer neurons is introduced to achieve the responses sparsity, and the L_1-norm of connection weights is used to achieve the connections sparsity. Based on the learned sparse structure of DBN, in the fine-tuning phase, penalty terms on responses sparsity are added to the loss function. In this way, the sparse framework learned in the unsupervised phase can be preserved and further fine-tuned sparsely, which greatly reduces the model complexity, and the network has a better explanatory and generalization ability.

In addition, the time complexity is also pretty reduced due to the sparse learning architecture. The sparse parameters of DNN and SRS-DNN, and the total running times of them are given in Table 5. All experiments were conducted with Inter(R) Core(TM) i7-7700 CPU@3.60 GHs and 32G RAM. Based on the results, we can see that the running times with SRS-DNN are less than half of that with DNN.

4.3 Remark

The above experiments show that by the sparsity-strengthened strategy, the proposed SRS-DNN has the best classification performance among general used classifiers, such as DNN, SVM, Gradient Boosting Classifier, K-Neighbors Classifier, Random Forest Classifier and other typical classifiers. By the classification results for each one of the ten categories, it shows more in detail that by SRS-DNN, the distinguishable features can be efficiently extracted in a sparsely layer-wise way, which confirms why SRS-DNN has the best performances.

In addition, comparing with DNN, less than a quarter hidden neurons of SRS-DNN are active while that for DNN is a half, and much more responses of SRS-DNN are fallen in the interval [0.9, 1], which dictate that the response sparsity ability of SRS-DNN is much better than that of DNN. Such kinds of response sparsity are more consistent with the response mechanism of neurons in human neural system, namely the neurons either respond or not for a stimulus, and most of the neurons are not response. Furthermore, by the sparsity-strengthened learning, SRS-DNN holds only 40% connections between higher layers (while nearly all connections are kept for DNN), which is more convenient for abstract features extraction and time saving.

5 Conclusion

The proposed deep neural network with strengthening response sparsity in this paper is inspired by the sparse response mechanism of neurons in biology neural systems, and we devote to obtain a sparse network which owns pretty generalization ability. Based on the sparse structure learned by an unsupervised sparse DBN, the network is fine-tuned by enhancing the sparseness for both the numbers of hidden neurons as well as the values of them. The unified and concise residual formulae for sparse BP learning can be considered as the substantial improvement of the existing sparse BP learning methods, such as the work of Andrew Ng. for sparse autoencoders. Experiments show that the put forward sparsity strengthening restrictions for responses are quite consistent with the response mechanism of neurons in human neuron systems, i.e., for a certain stimulus, the non-response neurons make up the majority of the whole neurons, and most of the response neurons are with very large response values. In addition, the fine-tuned network also has sparsity of connections. The sparse learning ability of the network ensures it to process the good performances including the generalization ability, identification accuracy and running times. It is promised that deep learning with strengthening response sparsity can be applied further to high-dimensional data with small sample size, such as the neuroimaging data and genomic data, which are under our current research.

Acknowledgements This research was funded by NSFC Nos. 11471006 and 11101327, the Fundamental Research Funds for the Central Universities (No. xjj2017126), the Science and Technology Project of Xi’an (No. 201809164CX5JC6) and the HPC Platform of Xi’an Jiaotong University.

Compliance with ethical standards

Conflict of interest The authors declare that there are no financial or other relationships that might lead to conflict of interest of the present article.

Appendix 1: The deviation of KL divergence for the parameters in Sect. 2.2

$$\begin{aligned}
\frac{\partial}{\partial W_{ij}}\sum_{j=1}^{N_h} KL(\rho \parallel p_j) &= \frac{\partial}{\partial W_{ij}}\sum_{j=1}^{N_h}(\rho \log\frac{\rho}{p_j} + (1-\rho)\log\frac{1-\rho}{1-p_j})\\
&= \left(-\frac{\rho}{p_j} + \frac{1-\rho}{1-p_j}\right)\frac{\partial p_j}{\partial W_{ij}}\\
&= \left(-\frac{\rho}{p_j} + \frac{1-\rho}{1-p_j}\right)\frac{1}{N_s}\sum_{q=1}^{N_s}\sigma_j^{(q)}(1-\sigma_j^{(q)})v_i^{(q)}\\
&= \frac{1}{N_s}\left(-\frac{\rho}{p_j} + \frac{1-\rho}{1-p_j}\right)\sum_{q=1}^{N_s}\sigma_j^{(q)}(1-\sigma_j^{(q)})v_i^{(q)}\\
\frac{\partial}{\partial \beta_j}\sum_{j=1}^{N_h} KL(\rho \parallel p_j) &= \frac{\partial}{\partial \beta_j}\sum_{j=1}^{N_h}(\rho \log\frac{\rho}{p_j} + (1-\rho)\log\frac{1-\rho}{1-p_j})\\
&= \left(-\frac{\rho}{p_j} + \frac{1-\rho}{1-p_j}\right)\frac{\partial p_j}{\partial \beta_j}\\
&= \left(-\frac{\rho}{p_j} + \frac{1-\rho}{1-p_j}\right)\frac{1}{N_s}\sum_{q=1}^{N_s}\sigma_j^{(q)}(1-\sigma_j^{(q)})\\
&= \frac{1}{N_s}\left(-\frac{\rho}{p_j} + \frac{1-\rho}{1-p_j}\right)\sum_{q=1}^{N_s}\sigma_j^{(q)}(1-\sigma_j^{(q)})
\end{aligned}$$

here

$$\sigma_j^{(q)} = \sigma(\sum_{i=1}^{N_v} v_i^{(q)}W_{ij} + \beta_j) = \frac{1}{1+e^{-\sum_{i=1}^{N_v} v_i^{(q)}W_{ij}-\beta_j}}$$

.

Appendix 2: The derivation of updating formula for the traditional BP in Sect. 3.1

For the traditional BP, the total error of the network in the backpropagation process, i.e., the loss function is

$$J(W) = \frac{1}{2N}\sum_{q=1}^{N}\sum_{j=1}^{n_L}(a_{qj}^{(L)} - y_{qj})^2$$

where N is the training sample size, y_{qj} is the target output of the j-th neuron in the output layer corresponding to the q-th sample, and $a_{qj}^{(L)}$ is the actual output of it. For simplicity, we first give the parameter updating formula for one sample. Consider $J(W) = \frac{1}{2}\sum_{j=1}^{n_L}(a_j^{(L)} - y_j)^2$ as the error of the network for one sample. Let η_1 be the learning rate, $W_{ij}^{(l)}$ be the connection weight of the i-th node in the l-th layer and the j-th node in the $(l+1)$-th layer $(1 \le i \le n_l + 1,\ 1 \le j \le n_{l+1})$, then we have the following update formula for the network parameters

$$\begin{aligned} W_{ij}^{(l)} &= W_{ij}^{(l)} - \eta_1 \frac{\partial J(W)}{\partial W_{ij}^{(l)}} = W_{ij}^{(l)} - \eta_1 \frac{\partial J(W)}{\partial z_j^{(l+1)}} \cdot \frac{\partial z_j^{(l+1)}}{\partial W_{ij}^{(l)}} \\ &= W_{ij}^{(l)} - \eta_1 \delta_j^{(l+1)} a_i^{(l)} \end{aligned}$$

where $\delta_j^{(l+1)} = \frac{\partial J(W)}{\partial z_j^{(l+1)}}$ is the residual of the j-th node in the $(l+1)$-th layer. For the L-th layer, i.e., the output layer, the residual of the j-th node is

$$\begin{aligned} \delta_j^{(L)} &= \frac{\partial J(W)}{\partial z_j^{(L)}} = \frac{\partial}{\partial z_j^{(L)}} \frac{1}{2} \sum_{k=1}^{n_L} (a_k^{(L)} - y_k)^2 \\ &= \frac{\partial}{\partial z_j^{(L)}} \frac{1}{2} \sum_{k=1}^{n_L} (f(z_k^{(L)}) - y_k)^2 \\ &= (f(z_j^{(L)}) - y_j) f'(z_j^{(L)}) \end{aligned}$$

Suppose

$$\begin{aligned} \delta^{(l)} &= (\delta_1^{(l)}, \delta_2^{(l)}, \ldots, \delta_{n_l}^{(l)}) \\ f'(z^{(l)}) &= (f'(z_1^{(l)}), f'(z_2^{(l)}), \ldots, f'(z_{n_l}^{(l)})) \end{aligned}$$

thus the residual vector of the L-th layer is

$$\delta^{(L)} = (a^{(L)} - y). * (f'(z^{(L)})) \quad (20)$$

where $.*$ is the vector product operator (Hadamard product), which is defined as the product of the corresponding elements for one vector or matrix.

The residual of the j-th node for the $(L-1)$-th layer is and the residual of the j-th node for the $(L-1)$-th layer is

$$\begin{aligned} \delta_j^{(L-1)} &= \frac{\partial J(W)}{\partial z_j^{(L-1)}} = \frac{\partial}{\partial z_j^{(L-1)}} \frac{1}{2} \sum_{k=1}^{n_L} (a_k^{(L)} - y_k)^2 \\ &= \frac{\partial}{\partial z_j^{(L-1)}} \frac{1}{2} \sum_{k=1}^{n_L} (f(z_k^{(L)}) - y_k)^2 \\ &= \frac{1}{2} \sum_{k=1}^{n_L} \frac{\partial}{\partial z_j^{(L-1)}} (f(z_k^{(L)}) - y_k)^2 \\ &= \sum_{k=1}^{n_L} (f(z_k^{(L)}) - y_k) \cdot f'(z_k^{(L)}) \cdot \frac{\partial z_k^{(L)}}{\partial z_j^{(L-1)}} \\ &= \sum_{k=1}^{n_L} \delta_k^{(L)} \cdot \frac{\partial z_k^{(L)}}{\partial z_j^{(L-1)}} \\ &= \sum_{k=1}^{n_L} \delta_k^{(L)} \cdot \frac{\partial}{\partial z_j^{(L-1)}} \sum_{s=1}^{n_{L-1}} a_s^{(L-1)} \cdot W_{sk}^{(L-1)} \\ &= \sum_{k=1}^{n_L} W_{jk}^{(L-1)} \delta_k^{(L)} f'(z_j^{(L-1)}) \end{aligned}$$

The residual of the j-th node for the l-th layer ($l = L-1, \ldots, 2, 1$) is $\delta_j^{(l)} = (\sum_{k=1}^{n_{l+1}} W_{jk}^{(l)} \delta_k^{(l+1)}) f'(z_j^{(l)})$, thus the vector form of the residual for the l-th layer is

$$\delta^{(l)} = ((\delta^{(l+1)}) \cdot (\bar{W}^{(l)})^{\mathrm{T}}). * f'(z^{(l)}) \quad (21)$$

where $\cdot$ is the matrix product, $\bar{W}^{(l)}$ is the first n_l rows of $W^{(l)}$. Let

$$\Delta W^{(l)} = (a^{(l)})^{\mathrm{T}} \cdot \delta^{(l+1)} \quad (22)$$

in which $\delta^{(l+1)}$ is defined by (20)–(21) ($l = L-1, \ldots, 2, 1$).

For the N samples case, by (22), we have $\Delta W_q^{(l)}\_J = (a^{(l)})_q^{\mathrm{T}} \cdot \delta_q^{(l+1)}$ for each sample $q = 1, 2, \ldots, N$. Thus, the update formula for the network parameters in a matrix form is

$$W^{(l)} = W^{(l)} - \eta_1 \cdot \frac{1}{N} \sum_{q=1}^{N} \Delta W_q^{(l)}\_J$$

Appendix 3: AR results of SRS-DNN with different parameter sets

The following Table 6 contains different sparse parameter sets of sparsity penalty items for KL and L_1 in RBM as well as in BP, i.e., λ_2, λ_3, τ_1 and τ_2, and also their corresponding accurate rates of classification. Based on which, in this paper, we select λ_2, λ_3, τ_1 and τ_2 to be 0.005, 0.0001, 0.0001 and 0.0002, respectively.

Table 6 AR results of SRS-DNN with different parameter sets

λ_2	λ_3	τ_1	τ_2	AR
0.005	0.0001	0.0001	0.0002	0.9064
0.005	0.0001	0.005	0.0001	0.8997
0.0005	0.0002	0.005	0.001	0.9013
0.0005	0.0001	0.001	0.0001	0.9035
0.0001	0.0005	0.0001	0.0002	0.8915

References

1. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
2. LeCun Y, Bengio Y, Hinton GE (2015) Deep learning. Nature 521(7553):436–444
3. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
4. Olshausen BA, Field DJ (2004) Sparse coding of sensory inputs. Curr Opin Neurobiol 14(4):481–487
5. Morris G, Nevet A, Bergman H (2003) Anatomical funneling, sparse connectivity and redundancy reduction in the neural networks of the basal ganglia. J Physiol Paris 97(4–6):581–589
6. Ji N, Zhang J, Zhang C et al (2014) Enhancing performance of restricted Boltzmann machines via log-sum regularization. Knowl Based Syst 63:82–96

7. Banino A, Barry C et al (2018) Vector-based navigation using grid-like representations in artificial agents. Nature. https://doi.org/10.1038/s41586-018-0102-6
8. Zhang H, Wang S, Xu X et al (2018) Tree2Vector: learning a vectorial representation for tree-structured data. IEEE Trans Neural Netw Learn Syst 29:1–15
9. Zhang H, Wang S, Zhao M et al (2018) Locality reconstruction models for book representation. IEEE Trans Knowl Data Eng 30:873–1886
10. Barlow HB (1972) Single units and sensation: a neuron doctrine for perceptual psychology. Perception 38(4):795–798
11. Nair V, Hinton G E (2009) 3D object recognition with Deep Belief Nets. In: International conference on neural information processing systems, pp 1339–1347
12. Lee H, Ekanadham C, Ng AY (2008) Sparse deep belief net model for visual area V2. Adv Neural Inf Process Syst 20:873–880
13. Lee H, Grosse R, Ranganath R et al (2011) Unsupervised learning of hierarchical representations with convolutional deep belief networks. Commun ACM 54(10):95–103
14. Ranzato MA, Poultney C, Chopra S, LeCun Yann (2006) Efficient learning of sparse representations with an energy-based model. Adv Neural Inf Process Syst 19:1137–1144
15. Thom M, Palm G (2013) Sparse activity and sparse connectivity in supervised learning. J Mach Learn Res 14(1):1091–1143
16. Wan W, Mabu S, Shimada K et al (2009) Enhancing the generalization ability of neural networks through controlling the hidden layers. Appl Soft Comput 9(1):404–414
17. Jones M, Poggio T (1995) Regularization theory and neural networks architectures. Neural Comput 7(2):219–269
18. Williams PM (1995) Bayesian regularization and pruning using a laplace prior. Neural Comput 7(1):117–143
19. Weigend A S, Rumelhart D E, Huberman B A (1990) Generalization by weight elimination with application to forecasting. In: Advances in neural information processing systems, DBLP, pp 875–882
20. Nowlan SJ, Hinton GE (1992) Simplifying neural networks by soft weight-sharing. Neural Comput 4(4):473–493
21. Zhang J, Ji N, Liu J et al (2015) Enhancing performance of the backpropagation algorithm via sparse response regularization. Neurocomputing 153:20–40
22. Ng A (2011) Sparse autoencoder. CS294A Lecture Notes for Stanford University
23. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
24. Bengio Y, Lamblin P, Popovici D, Larochelle H (2006) Greedy layer-wise training of deep networks. In: Proceedings of the advances in neural information processing systems, pp 19:153–160
25. Hinton GE (2002) Training products of experts by minimizing contrastive divergence. Neural Comput 14:1771–1800
26. Hinton GE (2010) A practical guide to training restricted Boltzmann machines. Momentum 9(1):599–619
27. Fischer A, Igel C (2014) Training restricted Boltzmann machines: an introduction. Pattern Recognit 47(1):25–39
28. Donoho DL (2006) Compressed sensing. IEEE Trans Inf Theory 52(4):1289–1306
29. Xiao H, Rasul K, Vollgraf R (2017) Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms arXiv:1708.07747v1
30. Maaten LV, Hinton GE (2008) Visualizing data using t-SNE. J Mach Learn Res 9(11):2579–2605