Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2017) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

Information Technology and Quantitative Management (ITQM 2017)

Hybrid Learning Network: A Novel Architecture for Fast Learning

Ying Liu[a,b], Chao Xiang[a]

[a]*School of Computer and Engineering, University of Chinese Academy of Sciences, Beijing, 100190 China*
[b]*Key Lab of Big Data Mining and Knowledge Management, Chinese Academy of Sciences, Beijing, 100190 China*

Abstract

There're many efficient architectures of the artificial neural network(ANN). For which the training is a hard work. The cost for training an ANN increases exponentially when the ANN gets deeper or wider. We therefore propose a novel achitecture, the Hybrid Learning Network(HLN), to achieve a fast learning with good stablity. The HLN can learn from both labeled data and unlabeled data at the same time in a hybrid learning manner. It uses a Self Organizing Map unified by the specially designed nonlinear function as the sparsity mask for a hidden layer to improve the training speed. We experiment our architecture on a synthetic dataset to test its regression capability against the traditional architecture, the result is promising. We also experiment on the well-known MNIST dataset, it demonstrates an even more impressive learning efficiency by up to 40× speed-up on training.

Selection and/or peer-review under responsibility of ITQM2017.

Keywords: HLN; hybrid learning; neural network; sparsity mask; SOM; MNIST; fast learning

1. Introduction

Since the first mathematical model of the artificial neural network(ANN) was proposed in 1943[1], lots of different architectures have been proposed to develop the model, such as the Convolutional Neural Network(CNN) for image recognition [2], the Recurrent Convolutional Neural Network(R-CNN) for object detection in videos [3], and the Long Short Term Memory(LSTM) for speech recognition [4], etc. These specially designed neural networks are trained by a few efficient methods such as the stochastic gradient descent(SGD)[5]. Such architectures suit well for their specific applications but may have plain or worse performances on others, and their best performances rely heavily on the hyperparameter configuration. Therefore a relatively universal architecutre that enables equal or similar performances among varied applications is in real demand.

Although a number of different neural network architectures have been proposed in the past years, literally all of them can be classifed into 3 categories, supervised [6], unsupervised [7], and semi-supervised learning [8]. In supervised learning, one can only train a model on labeled samples. However in real applications, labeling a large dataset is infeasible as a result unlabeled samples are the major. To make use of unlabeled data, a few unsupervised

*Corresponding author. Tel.: +86-183-0114-2368;
E-mail address: xiangchao215@mails.ucas.ac.cn.

training algorithms emerged. These unsupervised learning methods attempt to produce an optimized parameter initialization [9]. Thus, such unsupervised techniques can only be applied before the supervised training phase. Once the supervised begins, unsupversied learning will quit. Semi-supervised learning allows the model to learn from both labeled and unlabeled samples at the same time, where a regularizer is embedded in the supervised optimizing object, and generally a balance constraint is required to avoid the trival solution [10]. Such a enhanced learner brings more hyperparameters(such as the balance constraint), making it even more difficult to search for the best settings for current architectures. Additionally, this integration manner makes it impossible to separate the two learners as the semi-supervised regularizer is built on the supervised learner, and therefore an early supervised training alone with profound labeled data is necessary before the semi-supervised learning starts.

Therefore in the paper, we introduce a new architecture that combines the supervised and unsupervised learning equiring no extra techniques in training. Our architecture, the Hybrid Learning Network(HLN), consists of stacked hybrid learning layers, each of which embeds a Self Organizing Map(SOM). In the similar manner of backpropagation, HLNs demonstrate higher learning capability and stablity.

The main contributions of our work are:

- HLN is proposed as a hybrid learning architecture to learn both unlabeled and labeled data simultaneously. HLN overcomes the problem of traditional semi-supervised learning methods which requires an early standalone training for the supervised learner.
- A SOM-embedding layer structure is designed to learn a cluster mapping function from unlabeled data to speed up the supervised learning from labeled data.
- A nonlinear function $h(x)$, is proposed to measure the state of a SOM in each iteration of the training, and then determine a sparsity mask for every hidden layer.

Our proposed HLN is implemented in Python and all our code and results of experiments are available at `https://github.com/hiroki-kyoto/hybrid-learning-net`.

The rest of the paper is organized as follows. In section 2 we introduce existing semi-supervised algorithms for neural network models and SOM. In section 3, we present our novel architecture HLN and how to embed SOMs into deep architectures of neural networks. In section 4 we explain the training theory for HLNs. Section 5 presents the experimental results, and the last section concludes our work.

2. Related work and background Knowledge

2.1. Semi-supervised learning for neural network

A key assumption in semi-supervised algorithms, is the structure assumption: two samples with similar distribution on the same mapping structure tend to have high probability belonging to the same class. Based on this assumption, one can use large unlabeled data to uncover such structures. There're already a few algorithms dedicated to it, such as cluster kernels [11], Low Density Separation(LDS) [12], label propagation [13], etc. In such algorithms, designing a regularizer to enable the model to learn the representation or structure of raw data becomes the key point.

Let's firstly focus on the general algorithm description of semi-supervised learning. Given a set of unlabeled samples, $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}(\mathbf{x} \in \mathbb{R}^d)$, and the similarity labels between any $\mathbf{x}_i$ and $\mathbf{x}_j$, $\mathbf{W} = \{W_{ij} | i, j = 1, \cdots, N\}$, our goal is to find the best embedding function, $f(\mathbf{x})$, for each sample $\mathbf{x}_i$, in order to minimize:

$$\Delta_f = \sum_{i=1}^{N} \sum_{j=1}^{N} L\Big(f(\mathbf{x}_i), f(\mathbf{x}_j), W_{ij}\Big) \tag{1}$$

where,

- $L(\cdot)$ is the loss function of 3 variables: $\Big\langle f(\mathbf{x}_i), f(\mathbf{x}_j), W_{ij} \Big\rangle$. For example, if Euclidean distance is used,

$$L\Big(f(\mathbf{x}_i), f(\mathbf{x}_j), W_{ij}\Big) = \Big(\|f(\mathbf{x}_i) - f(\mathbf{x}_j)\| - W_{ij}\Big)^2$$

- $f(\mathbf{x}) \in \mathbb{R}^n$ is the embedding function, trying to produce an output vector for $\mathbf{x}_i$, similar to that for $\mathbf{x}_j$ with $W_{ij} = 0$, and disimilar with $W_{ij} = 1$.
- $W_{ij} \in \mathbb{R}$ is the similarity label of the sample pair $\left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle$ from $\mathbf{X}$.

As a result, the overall optimization object is

$$\arg\min \sum_i \ell\left(f(\mathbf{x}_i), \hat{\mathbf{y}}_i\right) + \lambda \Delta_f \tag{2}$$

where $\ell(\cdot)$ is the loss function for supervised learning. λ is proposed as the balance coefficient to embed the semi-supervised regularizer Δ_f.

2.2. Self Organizing Map

Self Organizing Map is an effective method for the visualization of high-dimensional data. It can also be used as an automatic clustering method. The SOM consists of a two-dimensional regular grid of nodes. The models are automatically organized into a meaningful two-dimensional order in which similar models are closer to each other in the grid than the more dissimilar ones[14]. Rules used to update models are:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{c(x),i}\left(\mathbf{x}(t) - \mathbf{m}_i(t)\right) \tag{3}$$

and the learning rate is dynamically determined as

$$h_{c(x),i} = \alpha(t) \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_c\|^2}{2\sigma^2(t)}\right) \tag{4}$$

where $\mathbf{m}_i \in \mathbb{R}^n$ denotes the i^{th} model vector, $\mathbf{x}$ is an input pattern, $c(x)$ relates to the best match vector index in $\mathbf{m}$ for input pattern $\mathbf{x}$, and $\alpha(t)$ is a learning rate that decreases as the training preceeds. $\mathbf{r}$ denotes the model vector location in the map, and $\sigma(t)$ corresponds to the width of the neighborhood function, which also decreases monotonically with the regression steps.

3. Hybrid Learning Network

Hybrid Learning Network is a novel architecture aiming at enhancing arbitrary networks on their training efficiency stablity. Each hidden layer in HLN architecture is embedded with a SOM. For the simplest case where neurons are connected with fully connection, the traditional architecture, called Fully Connected Network(FCN) architecture, is presented in Figure 1(a). The proposed HLN architecture is presented in Figure 1(b).

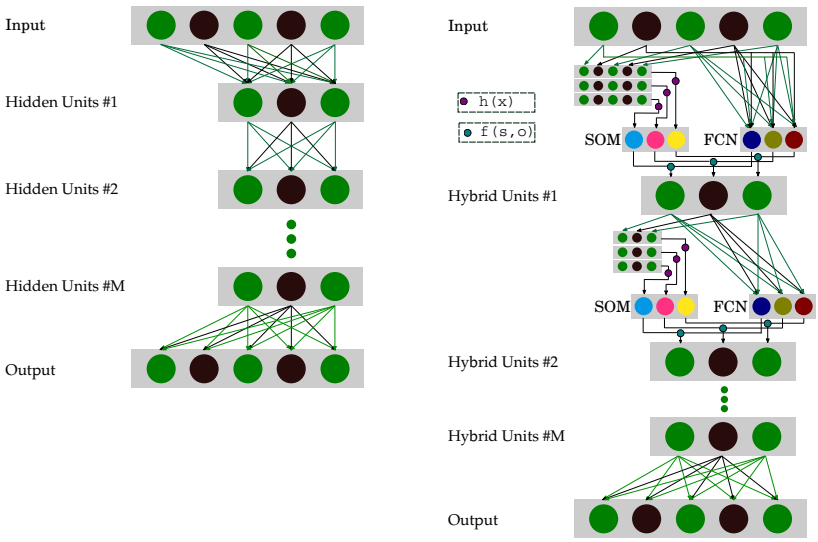
In HLN architecture, we propose $h(x)$, an unifying function for SOM, to convert pattern disimilarity $\|\mathbf{x} - \mathbf{m}_i\|$ to a semi-supervised learning factor for different hidden units. The semi-supervised learning factors act as a dynamic neuron activation sparsity mask for each hybrid learning layer. It works in the similar manner as the Dropout technique[15], enabling the neural networks to improve the model robustness and prevent overfitting by learning their submodels for each batch. However, HLN differs from techniques like Dropout because HLN does not generate sparsity with randomness. HLN uses the SOM to unsupervisedly learn the *static* policy of the neuron-activation distribution for each hidden layer, the network sparsity generator thereby will stablize with training steps, and the randomness in sparsity will disappear automatically. We propose $h(x)$ as follows,

$$\delta_{max} = \max_i \left(\|\mathbf{m}_i - \mathbf{x}\| + \epsilon\right) \tag{5}$$

$$\delta_{min} = \min_i \left(\|\mathbf{m}_i - \mathbf{x}\| + \epsilon\right) \tag{6}$$

where δ_{max} is the maximum disimilarity between an arbitrary vector $\mathbf{m}_i$ and the input pattern vector $\mathbf{x}$, δ_{min} is minimum one. The scale of all disimilarities is

$$\delta_{scale} = \frac{\delta_{min}}{\delta_{max} - \delta_{min} + \epsilon} \tag{7}$$



(a) The FCN architecture (b) The HLN architecture

Fig. 1. The left is FCN architecture, the right is HLN architecture. The big solid circles are the neurons, the medium ones are the middle states of neurons, and the small ones are the vector maps $\mathbf{m}^k$ of SOMs, finally the tiny ones are the function modules. The parts omitted refers to repeating hidden layers connected in same way.

We then unify the disimilarities and convert them into sparsity mask as

$$h(x)|_{x=\|\mathbf{m}_i-\mathbf{x}\|} = \frac{\delta_{max}\cdot\delta_{scale}}{\|\mathbf{m}_i-\mathbf{x}\|+\epsilon} - \delta_{scale} \tag{8}$$

where $\mathbf{m}_i$ and $\mathbf{x}$ is the same way defined as in equation (3). ϵ is set at a small constant value, such as $\epsilon = 10^{-5}$, to avoid *division-by-zero* errors, which rarely happens. Notice that if any other metric is preferred, we can always replace the pattern disimilarity $\|\mathbf{m}_i - \mathbf{x}\|$ with other forms. $f(s,o)$ is the function to combine the sparsity mask s generated by $h(\cdot)$ and the fully connected linear summation output o. In most cases, we use the simple multiplication as the operator,

$$f(s,o) = s\cdot o = h\left(\left\|\mathbf{m}_i - \mathbf{y}^{\mathbf{k-1}}\right\|\right)\cdot\sum_j\left(w_j^{k,i}y_j^{k-1}\right) + b^{k,i}, k>1 \tag{9}$$

As shown in Figure 1(b), $f(\cdot)$ is followed by an activation function, which could be any arbitrary nonlinear function that takes only one dimension inputs and outputs a single real value, such as the sigmoid function, $tanh(x)$ and ReLU. With the new architecture, we update the forward computing rules as

$$y_i^k(\mathbf{x}) = \sigma\left(h\left(\left\|\mathbf{m}_i^k - y^{k-1}(\mathbf{x})\right\|\right)\cdot\sum_j\left(w_j^{k,i}y_j^{k-1}(\mathbf{x})\right) + b^{k,i}\right), k>1 \tag{10}$$

and for the first hybrid learning layer,

$$y_i^1(\mathbf{x}) = \sigma\left(h\left(\left\|\mathbf{m}_i^1 - \mathbf{x}\right\|\right)\cdot\left(\sum_j w_j^{1,i}x_j\right) + b^{1,i}\right) \tag{11}$$

where m_i^k denotes the i^{th} vector in the k^{th} SOM of the net(all indexes start from 1), σ is a nonlinear activation function.

4. Training the HLNs

Let's compare the embedding theories between HLNs and the ones mentioned in the section of *Related work and backgrounds*. The existing semi-supervised algorithms embed a regularizer into the supervised learner, making it impossible to train both of the supervised learner and the unsupervised separately. As analyzed before, performances of such regularizer solutions depends on the standalone supervised pretraining for which a profound labeled data is required. However, in our architecture HLN, we assign each of the learning methods a completely separate optimizing object, with no priority orders restricted.
For the supervised learning, we may have such form of optimizing object as

$$\arg\min_{g} \sum_{i=1}^{L} \ell\left(g(\mathbf{x}_i), \hat{\mathbf{y}}_i\right) \tag{12}$$

where $g(\mathbf{x})$ is the function describing the mapping from the input $\mathbf{x} \in \mathbb{R}^d$ to the output $\mathbf{y} \in \mathbb{R}^n$, parameterized with the neural connection weights $\mathbf{W}$ and the biases $\mathbf{b}$ for all non-input layer neurons in the HLN architecture. This equation (12) may become the following one when Enclidean metric be applied for the loss,

$$\arg\min_{\mathbf{W},\mathbf{b}} \frac{1}{2} \sum_{i=1}^{L} \left\| y^k(\mathbf{x}_i)\big|_{\mathbf{W},\mathbf{b}} - \hat{\mathbf{y}}_i \right\|^2 \tag{13}$$

For the unsupervised learning, the optimizing objects are

$$\arg\min_{\mathbf{m}^k} \sum_{i}^{L+U} \left(\min_{j} \left(\left\| \mathbf{m}_j^k - \mathbf{x}_i^k \right\| \right) \right), \quad k = 1, 2, \cdots \tag{14}$$

with the predefined notation:

$$\mathbf{x}_i^k = \begin{cases} \mathbf{x}_i, & k = 1 \\ y^{k-1}(\mathbf{x}_i), & k > 1 \end{cases}, \quad \mathbf{x}_i \in L + U \tag{15}$$

For a net with M hybrid learning layers, there should be 1 supervised and M unsupervised optimizing objects. As we know a multi-objective problem may not have a global solution, however in HLNs, each object is optimized on its own isolated parameter space, thus each optimizing object have its corresponding global optimization solution, they together makes the whole one.

Using the well-known Back Propagation(BP) algorithm, we can obtain parameter updating rules for training the HLNs. We're only focusing on differences compared to original BP algorithm for the FCN model. Let's denote the linear summation for i^{th} unit in k^{th} layer as o_i^k, then linear summations of hybrid learning layers are (with $\mathbf{y}^0 = \mathbf{x}$):

$$o_i^k = h\left(\left\| \mathbf{m}_i^k - \mathbf{y}^{k-1} \right\| \right) \sum_j W_j^{k,i} y_j^{k-1}, \quad k = 1, 2, \cdots, M \tag{16}$$

Apply the non-linear activation function (taking the sigmoid as an instance),

$$\frac{\partial E}{\partial o_i^k} = \frac{\partial E}{\partial y_i^k} \sigma'(o_i^k) = \frac{\partial E}{\partial y_i^k} y_i^k \left(1 - y_i^k\right), \quad k = 1, 2, \cdots, M+1 \tag{17}$$

Thus the recursive layer gradient computing is

$$\frac{\partial E}{\partial y_i^{k-1}} = \sum_j \frac{\partial E}{\partial o_j^k} h\left(\left\| \mathbf{m}_j^k - \mathbf{y}^{k-1} \right\| \right) W_j^{k,i} \quad k = 1, 2, \cdots, M \tag{18}$$

The partial error gradients for the connection parameters $\mathbf{W}$:

$$\frac{\partial E}{\partial W_j^{k,i}} = \sum_i \frac{\partial E}{\partial o_i^k} h\left(\left\| \mathbf{m}_i^k - \mathbf{y}^{k-1} \right\| \right) y_i^{k-1}, \quad k = 1, 2, \cdots, M \tag{19}$$

For the partial error gradients on biases:

$$\frac{\partial E}{\partial b^{k,i}} = \frac{\partial E}{\partial o_i^k}\frac{\partial o_i^k}{\partial b^{k,i}} = \frac{\partial E}{\partial o_i^k} \times 1, \quad k = 1, 2, \cdots, M+1 \tag{20}$$

Thus we can iterate recursively along layers with equations (16-20) to update all parameters of the supervised training, and the equations corresponding to the unsupervised training refer directly to equations (3) and (4).

5. Empirical Study

5.1. Regression capability experiment using small synthetic data

A synthetic dataset is used. It maps 2-dimension vectos into 3 classes, where 3.5% of the noise is mixed to simulate a real sampling dataset. We then apply the data to the HLN(the net with HLN architecture) and the FCN(the net without), and compare the regression capabilities of the two. The FCN and HLN applied in this experiment follow the exact design as in Figure 1 with only a single hidden layer. Figure 2-3 present the comparison results.

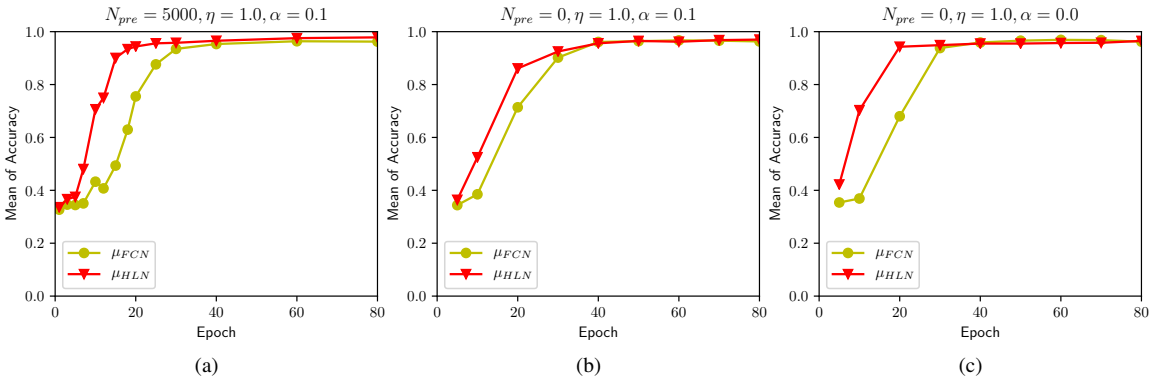


Fig. 2. Comparison on mean of accuracy with synthetic data: μ is the mean accuracy for multiple replays with the same configuration. N_{pre} is the unlabeled sample volume for pretraining, η is the initial learning rate, α is the decay factor of learning rate $\eta(t)$.

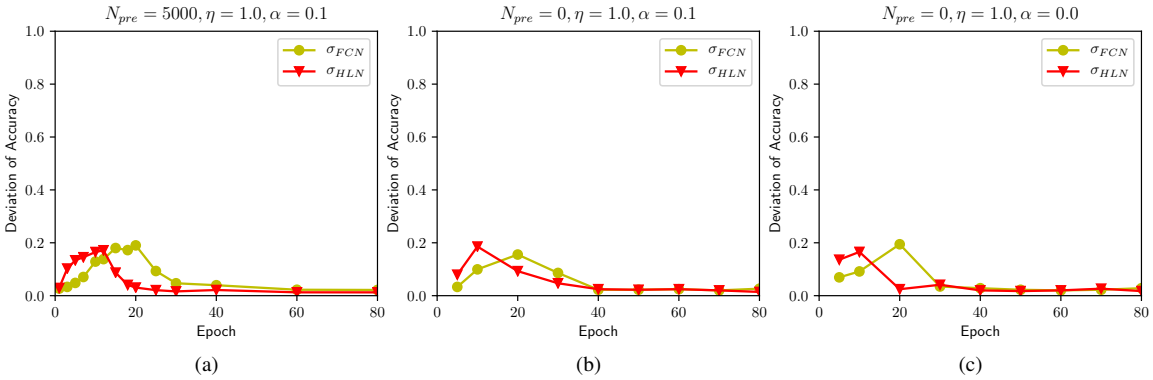


Fig. 3. Comparison on deviation of test accuracy with synthetic data: the deviation takes the form as δ is the deviation of multiple replays with the same configuration as $\delta = \sum_1^N(\gamma_i - \mu)/N$, where γ_i is i^{th} replay accuracy.

An explanation may be helpful: we split the data into two partitions, one is for the training of two architectures and the another is for testing, the train/test ratio is 0.8. Considering the possible impact from the parameter

initialization, we replay the complete workflow of training and testing with varied random parameter initialization for each net with different configurations. From Figure 2, easy to see that networks with our HLN architecture win over networks without in all conditions concerned. The overall results we may obtain from this experiment would be:

- The HLN architecture learns much faster than the FCN.
- The pretraining with unlabeled data does improve the performance of HLN, however the improvement is limited.
- A constant learning rate is more advisal than a decreasing one with steps.

Each of the net training is repeated for 30 times in our experiments, and within each training, the net parameters are reinitialized and relearned from the very beginning of training. Thus we can analyze the robustness of architectures to the parameter initialization. Figure 3 shows the HLN stablize quickly under all conditions, however the FCN requires more training to get itself stablized. The HLN proves to be robuster to the neural network hyperparameter variation of parameter initialization.

5.2. Experiments on MNIST

Experiments on the well-known MNIST benchmark demonstrate greater enhencement over the traditional architecture. Moreover the advantage of HLN increases with higher input dimension and deeper network. To enable the input dimension variation, a max-pooling is applied before the input of a neural net. The MNIST dataset provides 50000 images for training and 10000 images for testing, with each image resized as 28×28 of pixels. A max-pooling with the kernel of 2×2 leads to a input dimension of 196, and particularly a max-pooling with the kernel 1×1 will keep the original input dimension as 784. All networks used are based on architectures in Figure 1.

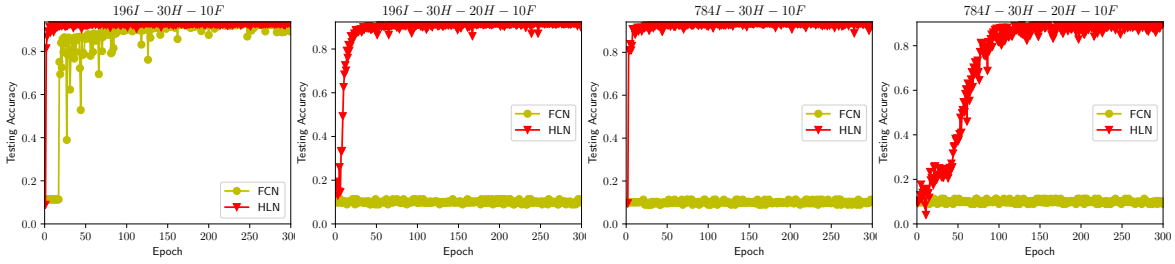


Fig. 4. Comparison of testing accuracy on MNIST with 4 different network models: $196I$ refers to the input layer of dimension 196, $30H$ refers to the hidden layer of dimension of 30, and $10F$ is the fully connected output layer of dimension 10.

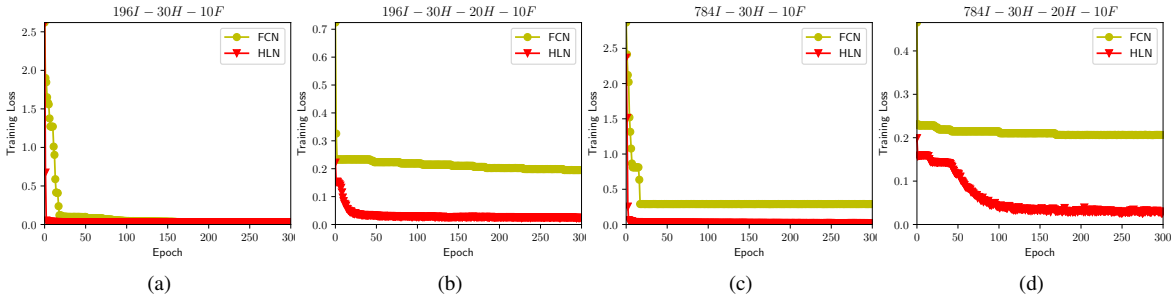


Fig. 5. Comparison of training loss on MNIST with 4 different network models

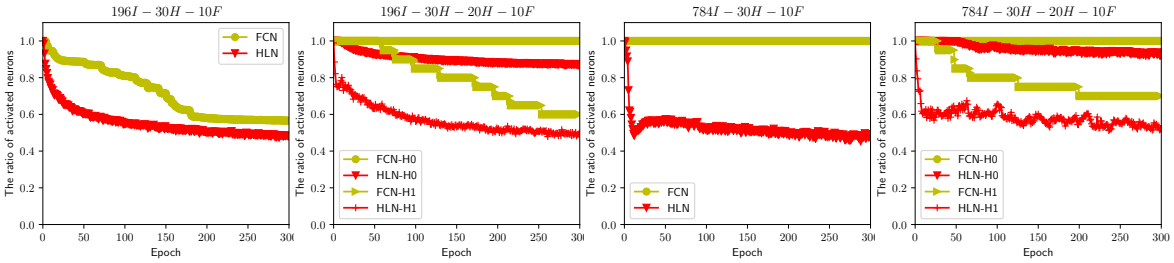


Fig. 6. Comparison of neuron activation sparsity on MNIST with 4 different network models

Seen from Figure 4(a), where a single hidden layer architecture is used, the FCN requires about 150-200 epoches to stablize at its best test accuracy of 93%, the HLN however only takes 4-5 epoches to achieve the same test accuracy or a bit higher. It's a 40× speed up on the training without a sacrifice of the model capability. The HLN additionally performs much higher stablity at all time during the whole training process. This may be due to the hybrid learning method used by the HLN, which enables the network to learn the neuron activation clustering maps(the SOMs) of every non-output layer especially the input layer at the same time for the supervised learning. Each sample for the input is more profoundly taken in use, making the HLN a much faster learning.

For Figure 4(b) with a deeper architecture, the FCN seems fail to learn at all, however our HLN remains a fast learning. Though the HLN architecture seems *slow down* a bit for a double hidden layer net compared to the single one's, it satisifies the emprical knowledge: a deeper architecture brings a tougher parameter tuning.

For Figure 4(c), we use a max-pooling with a kernel of 1×1, which equals to original input as in MNIST. The input dimension of the net is therefore 4 times larger than that in the first test, however the HLN learns as fast as the one of low input dimension with stablity held the same strong. While the FCN in this situation fails to learn at all either.

Let's focus on Figure 4(d), where the HLN takes about 120 epoches to stalize at the best performance and the FCN seems never start to learn. Compared to the third test, the networks have a deeper architecture, making the model more difficult to learn as expected. Compared to the second, networks in the fourth experiment have a larger input dimension, which leads to a tougher learning too. The two factors putting together turns out to be even more harmful to the training efficiency as we may expect in a linear manner.

Figure 5 further confirms the *facts* drawn from results in Figure 4. The test accuracy is not improved at all with training steps for deeper architectures, whereas the training loss is dropping, though, at a very slow pace. This may conclude to a guess: with the epoch number increasing to a rather larger one, the training loss may drop to a level where a visible learning expressed by test accuracy improvement at length begins.

Figure 6 shows the neuron activation sparsity for every hidden layer in networks. Notice that a SOM-embedding output in HLNs is used as a sparsity mask to control, or more precisely, to increase the sparsity of every hidden layer. It's obvious that networks with the HLN architecture tend to have a higher sparsity than ones without seen from Figure 6.

6. Conclusions

We proposed a hybrid learning architecture, the HLN, for neural networks to learn from labeled and unlabeled data at the same time and to achieve a much faster learning with fair stablity. We showed the solution to training networks of the HLN architecture, and also demonstrated the advantages in training speed and robustness to some hyperparameter variation by experimenting on synthetic data and the MNIST. In the future, we will investigate if our architecture works well on more public benchmark datasets.

Acknowledgements

This project was partially supported by Grants from Natural Science Foundation of China #71671178/ #91546201/ #61202321, and the open project of the Key Lab of Big Data Mining and Knowledge Management. It was also supported by Hainan Provincial Department of Science and Technology under Grant No. ZDKJ2016021, and by Guangdong Provincial Science and Technology Project 20162016B010127004.

References

[1] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of mathematical biology 5 (4) (1943) 115–133.
[2] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.
[3] R. Girshick, Fast r-cnn, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.
[4] A. Graves, N. Jaitly, A.-r. Mohamed, Hybrid speech recognition with deep bidirectional lstm, in: Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on, IEEE, 2013, pp. 273–278.
[5] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.
[6] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, L. D. Jackel, Handwritten digit recognition with a back-propagation network, in: Advances in neural information processing systems, 1990, pp. 396–404.
[7] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, Journal of Machine Learning Research 11 (Dec) (2010) 3371–3408.
[8] O. Chapelle, B. Scholkopf, A. Zien, Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews], IEEE Transactions on Neural Networks 20 (3) (2009) 542–542.
[9] Q. V. Le, Building high-level features using large scale unsupervised learning, in: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, 2013, pp. 8595–8598.
[10] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, C. D. Manning, Semi-supervised recursive autoencoders for predicting sentiment distributions, in: Proceedings of the conference on empirical methods in natural language processing, Association for Computational Linguistics, 2011, pp. 151–161.
[11] O. Chapelle, J. Weston, B. Schölkopf, Cluster kernels for semi-supervised learning, in: Advances in neural information processing systems, 2003, pp. 601–608.
[12] O. Chapelle, A. Zien, Semi-supervised classification by low density separation., in: AISTATS, 2005, pp. 57–64.
[13] X. Zhu, Z. Ghahramani, Learning from labeled and unlabeled data with label propagation.
[14] T. Kohonen, The self-organizing map, Neurocomputing 21 (1) (1998) 1–6.
[15] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting., Journal of Machine Learning Research 15 (1) (2014) 1929–1958.