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 net: a novel architecture for fast learning

Ying Liu, Chao Xiang

University of Chinese Academy of Sciences, Beijing 100049, China

Abstract

Currently, neural networks have succeeded in object recognition tasks based on images, natural language translation, and voice recognition, to name a few. However, neural networks are customly built for different applications and vary a lot in achitectures and model hyperparameters like learning rate and parameter initialization, what's worse, these hyperparameter settings generally play a big role in performances of training and testing, and the best settings for specific applications are so far only available by manually repeatly trying different configurations which is really huge work. We, thereby, present a novel neural network achitecture, called Hybrid Learning Net(**HLN**), with Self Organizing Maps(SOMs) embedded in each layer to learn from samples in **both** unsupervised and supervised way, targeting to achieve a much **faster** net learning for general applications with good robustness to a few key hyperparameters such as the parameter initialization and the net strcuture variation. We've also experimented our architecture over the MNIST dataset, it has proved the impressive improvement on both training and testing phases of general applications, say compared to the traditional architecture, our method speed up the training process by up to **40** times. In addition, on big scale of input dimension and/or with deeper architecture, where the traditional architecture fails to learn at all, our method still have a fast learning, and can retrieve the same testing accuracy on MNIST.

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

1. Introduction

Since the first mathematical model of artificial neural network was proposed in 1943[1], the neural network has been designed into many architectures, such as Convolutional Neural Networks(CNNs) for image recognition [2], Recurrent Convolutional Neural Networks(R-CNNs) for object detection in videos[3], and Long Short Term Memorys(LSTMs) for speech recognition [4] and many, many more to make it a list. These specially designed neural network models are trained by lots of efficient methods with tons of carefully chosen little skills which we may call tricks. Such neural network architectures suit well for their specific applications but may have plain or worse performances on others, and their best performances rely heavily on hyperparameter configuration in general. Thus, it is an in-demand job to propose a relatively universal architecutre that enables equal or similar performances among varied applications.

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

We notice that, although there're plenty of choices to train a neural network, literally all these methods can be reduced into 3 categories, supervised learning [5], unsupervised learning [6], and the semi-supervised [7]. In supervised learning, one can only train a model from labeled samples, however in real applications, labeling a large dataset is a tough and costly task, the unlabeled ones are therefore the primary data available. To make use of the majority unlabeled data, a few unsupervised training algorithms arose. These unsupervised learning methods, denoted as pretraining, attempt to produce an optimized parameter initialization [8]. Thus such unsupervised techniques can only be applied before the supervised training phase, once the supervised begins, the unsupversied learning will become unavailable for the model. While the semi-supervised learning allows the model to learn from both labeled and unlabeled samples at the same time, however they use a regularizer to embed the semi-supervised learner to original optimizing object, and generally a balance constraint is required to avoid the trival solution [9]. Such a enhanced learner brings more hyperparameters(say the balance constraint), making it even more difficult to search for best settings for current architectures. Additionally, this integration way makes it impossible to separate the two learners as the semi-supervised regularizer is built on the supervised learner, and therefore requires an early supervised training alone with profound labeled data before the semi-supervised regularizer can be applied.

We therefore introduce a new architecture that learn in both supervised and unsupervised ways at the same time, and requires no extra techniques on training. Our architecture, Hybrid Learning Networks(HLNs), made of stacked layers, with each hidden layer embedded with a Self Organizing Map(SOM), training in the simplest way of backprop, demonstrate much faster learning capability and remain robust to parameter initialization and network configuration such as the net depth of layer.

The main contributions of our work are:

- We propose HLN, a hybrid learning architecture to learn both unlabeled and labeled data at the same time. Our architecture HLN overcomes the problem brought by traditional semi-supervised learning methods that the semi-supervised learning requires an early standalone training for the supervised learner which contradicts with the fact: the labeled data is far expensive than the unlabeled.
- A SOM-embedding layer strucutre is designed to learn a cluster mapping function from unlabeled data to enable a classified thus much faster learning on labeled data.
- We propose a nonlinear function $h(x)$, which measures the whole state of a SOM in each iteration of the training, to determine a sparsity mask for every hidden layer.

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

The rest of the article is as follows. In section 2 we describe existing semi-supervised algorithms for neural network models and recall the SOM that will be applied in our neural network embeddings. In section 3, we introduce our novel architecture HLN, show how to embed SOMs into deep architectures of neural networks. In section 4 we explain the exact training theory for HLNs. Section 5 gives experimental comparisons between networks with HLN architecture and without, and the last section concludes.

2. Related work and backgrounds

2.1. Semi-supervised learning for neural network

A key assumption in semi-supervised algorithms developed so far, is the structure assumption: two samples with similar distribution on the same mapping structure tend to have high probability of 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 do this, such as cluster kernels [10], Low Density Separation(LDS) [11], label propagation [12], to name a few. In such algorithms, designing a regularizer to enable the model to learn the representation or structure of raw data, in order to improve the supervised learning performance, 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 $\mathbf{x}_j$, $\mathbf{W} = \{W_{ij}|i, j = 1, \cdots, N\}$, we're to find the best embedding function, $f(\mathbf{x})$, for each sample $\mathbf{x}_i$, to minimize:

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

in which,

- $L(\cdot)$ is the loss function of 3 variables: $\left\langle f(\mathbf{x}_i), f(\mathbf{x}_j), W_{ij}\right\rangle$, such as

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

  if Euclidean metric is used for the loss.
- $f(\mathbf{x}) \in \mathbb{R}^n$ is the embedding function, it tries to produce a vector from $\mathbf{x}_i$, similar to that of $\mathbf{x}_j$ 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 $\mathbf{X}$.

The overall optimization object thus will be

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

where λ is proposed as the balance coefficient for embedding the semi-supervised regularizer Δ_f.

2.2. *Self Organizing Map*

The Self Organizing Map(SOM) is an effective software tool 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[13]. 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

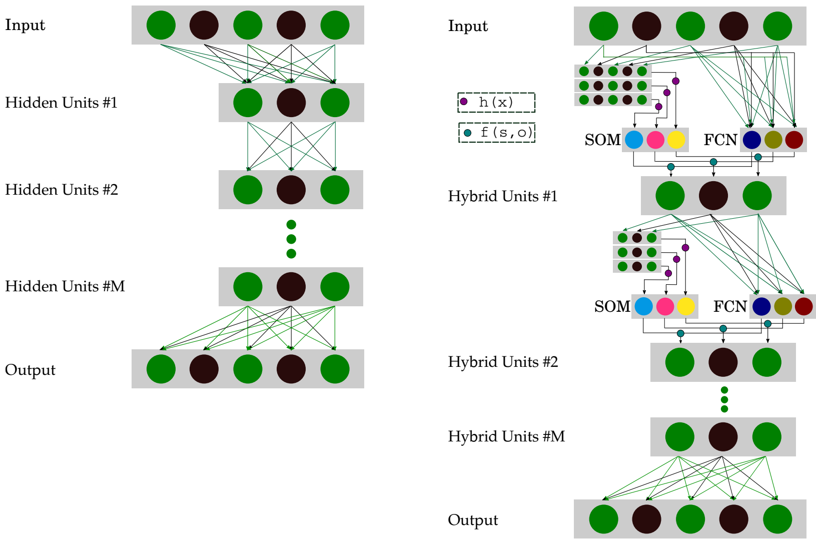
$$h_{c(x),i} = \alpha(t)\exp\left(-\frac{||\mathbf{r}_i - \mathbf{r}_c||^2}{2\sigma^2(t)}\right) \tag{4}$$

where $\mathbf{m}_i \in \mathbb{R}^n$ is 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 with training preceeding. $\mathbf{r}$ is 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. Our architecture: the Hybrid Learning Net

We propose the Hybrid Learning Net(HLN) as an architecture to enhance arbitrary networks on their training efficiency and robustness to some hyperparameters. Each layer in the HLN architecture embeds a SOM into its original layer, for the simplest situation where neurons connected in fully connected way, which we call the Fully Connected Neurons(FCN) architecture, we have such an embedding solution as described in Figure 1.

In the HLN architecture from Figure 1, $h(x)$ is an unifying function we proposed for SOMs to convert from pattern disimilarity $||\mathbf{x}-\mathbf{m}_i||$, into a semi-supervised learning factor for different hidden units. The semi-supervised learning factors as a whole act as a dynamic neuron activation sparsity mask for each hybrid learning layer. It works in a way like the Dropout technique[14], enabling the neural networks to improve the model robustness and prevent overfitting by learning their submodels for each batch. However, the HLN differs from techniques like the Dropout in which, the HLN does not generate sparsity with randomness, it uses the SOM to unsupervisedly learn



(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.

the *static* policy of neuron-activation distribution for each layer, the net sparsity generator thereby will stablize with training steps, and the randomness in sparsity will disappear automatically. We propose $h(x)$ with the form as followings,

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

δ_{max} is the maximum disimilarity between an arbitrary vector in $\mathbf{m}$ and the input pattern vector $\mathbf{x}$, δ_{min} is similar,

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

and the scale of all disimilarities is

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

Finally we unify the disimilarities and convert them into sparsity mask as

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

in which $\mathbf{m}_i$ and $\mathbf{x}$ is the same way defined in equation (3). ϵ is a constant of pretty small value like $\epsilon = 10^{-5}$ to avoid *division-by-zero* errors that may, though not very likely occur. Notice that if any other metric be preferred, we can always replace the pattern disimilarity $\|\mathbf{m}_i - \mathbf{x}\|$ with its corresponding form. $f(s, o)$ 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 choose the multiplication operator,

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

For the computing flow described in Figure 1(b), $f(\cdot)$ is followed by an activation function, it can be any arbitrary nonlinear function that takes only one dimension inputs and outputs a single real, such as the sigmoid function, $tanh(x)$ and the 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 **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.

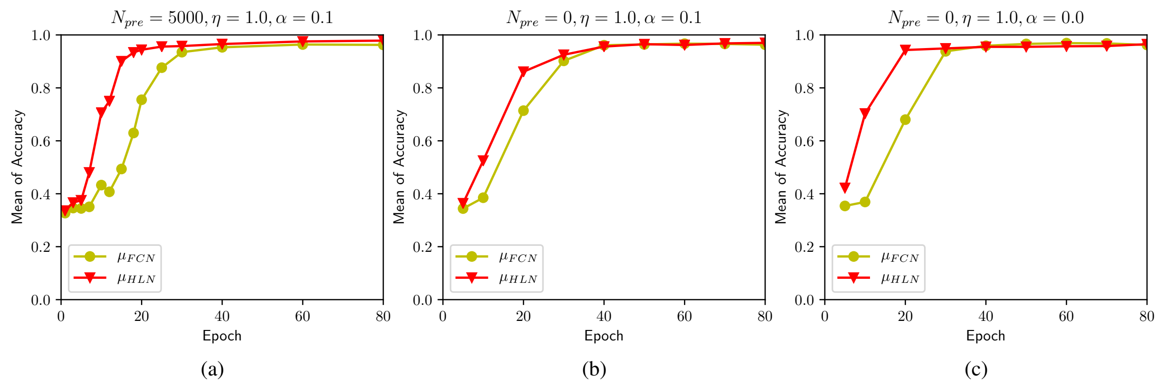


Fig. 2. Comparison on mean of accuracy with synthetic data: μ is the mean accuracy for multiple replays with the same configuration. N_{pre} is the unlabeled sample volume for pretraining, η is the initial learning rate, α is the decay factor of learning rate $\eta(t)$.

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.

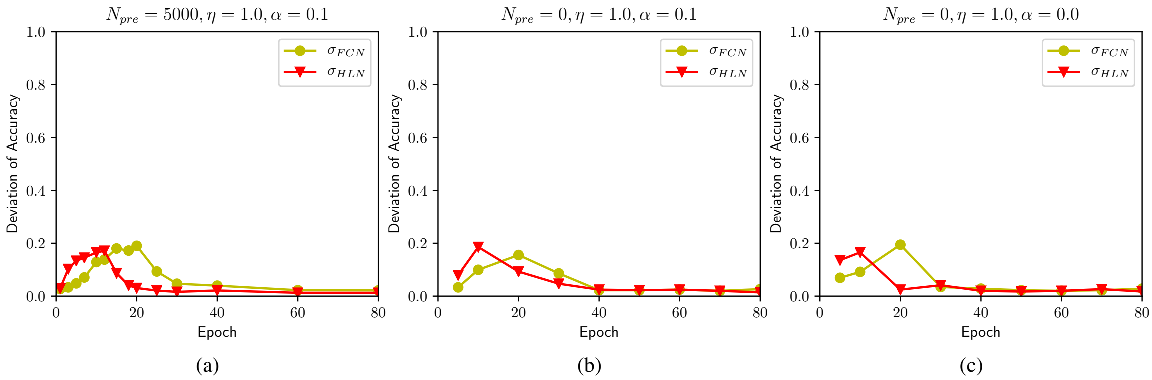


Fig. 3. Comparison on deviation of test accuracy with synthetic data: the deviation takes the form as δ is the deviation of multiple replays with the same configuration as $\delta = \sum_1^N (\gamma_i - \mu)/N$, where γ_i is i^{th} replay accuracy.

- 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.

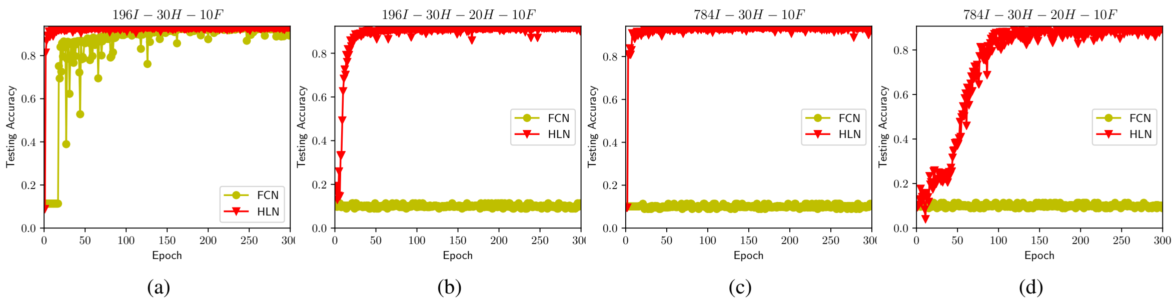
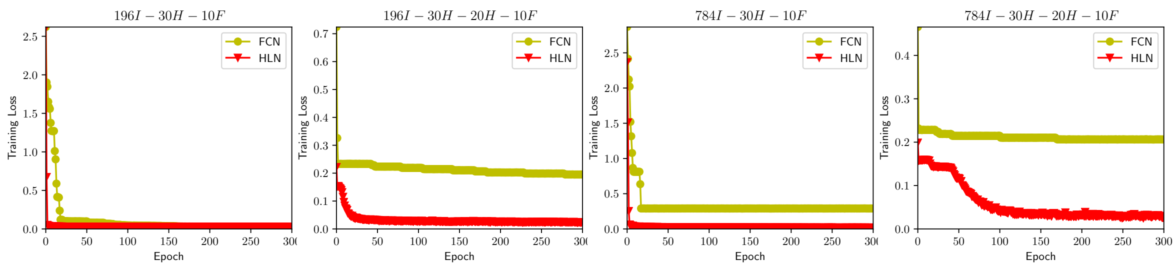


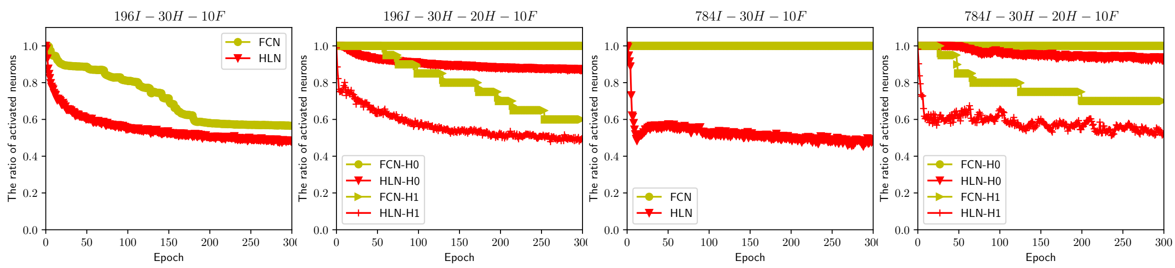
Fig. 4. Comparison of testing accuracy on MNIST with 4 different network models: $196I$ refers to the input layer of dimension 196, $30H$ refers to the hidden layer of dimension of 30, and $10F$ is the fully connected output layer of dimension 10.

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



(a) (b) (c) (d)

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



(a) (b) (c) (d)

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

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

These and the Reference headings are in bold but have no numbers. Text below continues as normal.

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] 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.
[6] 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.
[7] 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.
[8] 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.
[9] 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.
[10] O. Chapelle, J. Weston, B. Schölkopf, Cluster kernels for semi-supervised learning, in: Advances in neural information processing systems, 2003, pp. 601–608.
[11] O. Chapelle, A. Zien, Semi-supervised classification by low density separation., in: AISTATS, 2005, pp. 57–64.
[12] X. Zhu, Z. Ghahramani, Learning from labeled and unlabeled data with label propagation.
[13] T. Kohonen, The self-organizing map, Neurocomputing 21 (1) (1998) 1–6.
[14] 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.