Unsupervised Feature Learning by Pre-Route Simulation of Auto-Encoder Behavior Model

Youngjae Jin, Daeshik Kim

***Abstract*—**This paper describes a cycle accurate simulation results of weight values learned by an auto-encoder behavior model in terms of pre-route simulation. Given the results we visualized the first layer representations with natural images. Many common deep learning threads have focused on learning high-level abstraction of unlabeled raw data by unsupervised feature learning. However, in the process of handling such a huge amount of data, the learning method's computation complexity and time limited advanced research. These limitations came from the fact these algorithms were computed by using only single core CPUs. For this reason, parallel-based hardware, FPGAs, was seen as a possible solution to overcome these limitations. We adopted and simulated the ready-made auto-encoder to design a behavior model in VerilogHDL[1] before designing hardware. With the auto-encoder behavior model pre-route simulation, we obtained the cycle accurate results of the parameter of each hidden layer by using MODELSIM[2]. The cycle accurate results are very important factor in designing a parallel-based digital hardware. Finally this paper shows an appropriate operation of behavior model based pre-route simulation. Moreover, we visualized learning latent representations of the first hidden layer with Kyoto natural image dataset.

***Keywords*—**Auto-encoder, Behavior model simulation, Digital hardware design, Pre-route simulation, Unsupervised feature learning.

I. Introduction

THE performance of the deep learning algorithm is very dependent on input data. This explains that much of the practical effort in exploiting the algorithms start into a design strategy of representation learning of input [16]. That is to say, the learning strategy, called unsupervised feature learning, extracts and organizes the underlying explanatory factors latent in the observed milieu of low-level sensory data [1]. This pre-training internal distributed representations of the raw data mostly helps initializing parameters of the deep architectures in an optimal conditions to be trained well. Then after the pre-training, the deep networks can be fine-tuned with not randomized but the appropriate learned weight values [2], [3].

On the other hand, there has been notable approach in increasing performance of the deep learning algorithms, especially for training procedure by using FPGAs[4], [5],[17]. A reconfigurability and inherent parallelism are important features of the FPGAs. The deep learning algorithms can be designed through the original and useful hardware architecture.

With this low-level structure, the algorithms have been shown to reduce the whole training time, increase performance of real time classification tasks, and discover high-level abstract features in more complex deep algorithms [4], [5], [17], [18]. Despite of these several advantageous parts, designing such high-level algorithms on FPGAs is still very intractable and it can be very stubborn work since it makes many researchers complicate to directly modify the designed algorithm [6]. For this reason, we proposed a pre-route simulation regarded as a bridge between an inflexible hardware and a software relatively changeable on modification of designed algorithm. The simulation can be quite essential to design the high-level systems, such as deep learning or unsupervised pre-training algorithms, based on complex digital hardware.

This paper highlighted making use of our own designed a pre-route simulation model of the sparse auto-encoder algorithm. Throughout result of utilizing the simulation, we presented a feasibility of greedy layer-wise unsupervised pre-training digital hardware circuit, FPGAs [8]. In other words, we applied a novel digital hardware simulation framework to the auto-encoder to discover very generic features of the input. We used a Kyoto natural image dataset.

The algorithm's simulation composed of the low-level behavior model in VerilogHDL. Using the designed behavior model, we pre-trained the auto-encoder architecture with L-BFGS optimization method [7] which is our own designed in VerilogHDL, on MODELSIM simulator. Given the extracted unknown structure from the input distribution, we showed significant results by using MATLAB. We visualized the high-level abstraction of the input natural images, from the behavior model simulation [20].

The main objective of this paper, first, is to describe the feasibility of high-level algorithm, unsupervised auto-encoder learning algorithm on digital hardware circuits. To show the possibility, we introduced the pre-routing simulation methodology with showing the cycle accurate result of the algorithm. It is possible to fast and accurate modify the revised algorithm through the procedure of an intractable hardware design. Also, case study of designing the auto-encoder learning structure with the pre-route simulation platform can be of help to who want to implement pre-route simulation before designing the synthesizable RTL[3] model.

Youngjae Jin is with the Brain Reverse Engineering and Imaging Laboratory, Department of Electrical Engineering Korea Advanced Institute of Science, 291, Daehak-ro, Yuseong-gu, Daejeon, Korea (phone: 82-42-350-8174; e-mail:yj.jin@kaist.ac.kr).

Daeshik Kim is with the Brain Reverse Engineering and Imaging Laboratory, Department of Electrical Engineering Korea Advanced Institute of Science, 291, Daehak-ro, Yuseong-gu, Daejeon, Korea (phone: 82-42-350-3490; e-mail: daeshik@kaist.ac.kr).

[1] VerilogHDL is a hardware description language used to model digital circuit systems.

[2] MODELSIM is a verification and simulation tool for Verilog Hardware Description Language.

[3] Register-Transfer Level

II. Unsupervised Feature Learning

A. Unsupervised Feature Learning

An unsupervised feature learning method is to capture the unknown factors of latent variation that underlie the input distribution. Such unsupervised learning of representations often learns the posterior distribution P(y|x) of the underlying features for the raw input distribution P(x) [8].

According to Tesauro [9], deep learning architectures was too difficult to train, since a gradient descent optimization method might get stuck in poor solutions, such as local minimum, with randomly initialized starting parameters. In other words, it was a quite difficult task to search the parameter space of deep architectures. However, Hinton et al. [1] published a greedy layer-wise unsupervised learning method to address the training problem. By using this unsupervised feature training strategy, we can initialize the parameters in a region near an optimal local minimum for fast and accurate optimization process. With the optimal loaded parameters, it shows better performance of generalization or classification with learned high-level abstractions of the input. Considering the learning method from a different point of view, unsupervised pre-training acts as a regularizer, by adding an infinite penalty term on the cost function of auto-encoder [10]. More intuitively, we can consider the optimal initialization parameters as inherently imposing penalty on the parameters of the networks. Each layer trained in unsupervised learning method corresponds with sparse auto-encoder, we will describe in next section.

B. Auto-Encoder with Sparsity

An auto-encoder is a sort of unsupervised learning algorithm which applies back-propagation algorithm for training architecture [11], [12]. There are three layers, input, hidden and output layers, in the single auto-encoder. First two layers, input and hidden, represent an encoder part. Next two layers, hidden and output, act as a decoder. Setting identical number of units in the input and the output layers can make output values same as an input values.

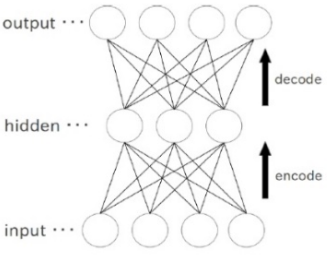


Fig. 1 Auto-encoder structure

It seems like useless arranging the units to reconstruct the input data similar as identity function. However, by limiting the number of hidden units in a constraint, we can extract significant features about input data. As mentioned above, the latent features of the input data are critical for recognition and classification of the input one.

Besides, we used the logistic sigmoid function for activation function of each unit, applied element wise of the vector z.

$$\mathrm{g(z)} = \frac{1}{1+e^{-z}}$$

Cost function has a sparse constraint which imposes on the hidden layer. Overall cost function of auto-encoder is as follows.

$$J(W,b) = \left[\frac{1}{2m}\sum_{i=1}^{m}\left(||h_{W,b}(x^i) - y^i||^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{O_l}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^l\right)^2$$

The notation of $h_{W,b}(x)$ is a nonlinear form of hypothesis, i.e. the output of each computational hidden unit. The first term is an average of sum of squared error term. The second one is a regularization term which tends to prevent over-fitting by decreasing the magnitude of the weight. The λ is a weight decay hyper-parameter which controls the balance between two terms.

To reduce the complexity of computation, we enforce a sparse constraint on the hidden units. An average activation of each hidden unit j would be close to specific sparsity parameter (ρ) as below.

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left(a_j^{(2)}(x^{(i)})\right)$$

To satisfy this restraint, we choose a KL divergence[4] for hidden unit activation penalty term as follows [20]:

$$\sum_{j=1}^{s_2} KL(\rho||\widehat{\rho_j}) = \sum_{j=1}^{s_2}\left(\rho log\frac{\rho}{\widehat{\rho_j}} + (1-\rho)log\frac{1-\rho}{1-\widehat{\rho_j}}\right)$$

Notations of above two formulas are descripted in footnote5. Thus, to minimize the above penalty term, we add it to our cost function.

$$J_{sparse}(W,b) = J(W,b) + \beta\sum_{j=1}^{s_2} KL(\rho||\widehat{\rho_j})$$

The hyper-parameter β is control parameter for sparsity penalty term. Now, to complete the pre-training of auto-encoder, we have only to optimize the objective function by using optimization method. We chose the L-BFGS optimization method with exponentially decreasing step length [13] instead of any other adaptive learning methods, such as Armijo or Wolfe conditions. The simple procedure of the back-propagation algorithm for auto-encoder is as follows.

First step is to perform a “feed forward pass” to compute all the activations from input layer to output layer, the output value

[4] Kullback-Leibler divergence: non-symmetric measure of the difference between two probability distributions.

[5] Notation i and j are the index of input and hidden unit. $s_l$is the index of layer. $m$is the number of input data. $O_l$is the output layer.

is the hypothesis, $h_{W,b}(x)$. Second, computing an "error term for each node in each layer" which is regarded as criteria of how much the node has responsibility for error in the last layer. Finally, the simple auto-encoder ends up learning a reductive representation closely akin to PCAs.

III. Optimization Method

We chose the L-BFGS optimization method, which is a limited-memory quasi-Newton method for unconstrained non-linear optimization [14]. It minimizes the auto-encoder's cost function without line search [15]. Because of the implementation issue of the auto-encoder behavior model in VerilogHDL, we selected such exponentially decreasing learning step [19].

IV. Pre-Route Simulation

To design the specific algorithm through the digital hardware circuits, one should clarify the cycle accurate results synchronized on clock. However, it is hard to change the algorithm after routing on the hardware platform. To be more concrete, modifying the algorithm on hardware design level has a lot of risks and it is an intractable process. Of course, someone can use the reconfigurable FPGAs to adjust the algorithm and these are an attractive platform to design high-level system. However, design effort for FPGAs implementation remains quite high [16]. For these reasons, digital clock synchronized pre-route simulation is very important. The simulation is coded by any hardware description languages, such as VerilogHDL, VHDL and SystemC. It shows that a cycle accurate results of the algorithm to be designed on a digital circuit.

V. Dataset and Experiment

We used the Kyoto natural image dataset and randomly downloaded natural images from Google image search. We used the VerilogHDL to design behavior model of an unsupervised feature learning, auto-encoder. In the first experiment, we tried to verify results from the behavior model operation on MODELSIM and MATLAB as same as possible. Since MODELSIM has no mathematical functions, such as exponential, square root and natural logarithm, we should have designed and verified each function. Then, to verify the cycle accurate results of the behavior model simulation synchronized by clock, we set the same input data and same initial weight values. Secondly, we inputted the same sensory data, but different initial parameters between both simulators. It is expected that same representation values of the first hidden layer comes from the both simulators. We reported Fig.7, which is similar distribution of learning completed parameters started from very different initial values.After the both pre-trainings on the MODELSIM simulator, we plotted the extracted representation values. Also, we traced the difference of weight and bias of the two results in every 100 iterations, as depicted in Fig. 6. The visible units are 196 and a hidden layer size is 100 for auto-encoder structure. The synchronized clock speed on behavior model is 10ns. The size of the input and output data is 64bit, in binary form. We designed the model by using the FSM[6] to describe process of the auto-encoder.

VI. Result and Analysis

The result of the first experiment which is to verify the cycle accurate simulation results of the auto-encoder is in Figs. 2-4 below.

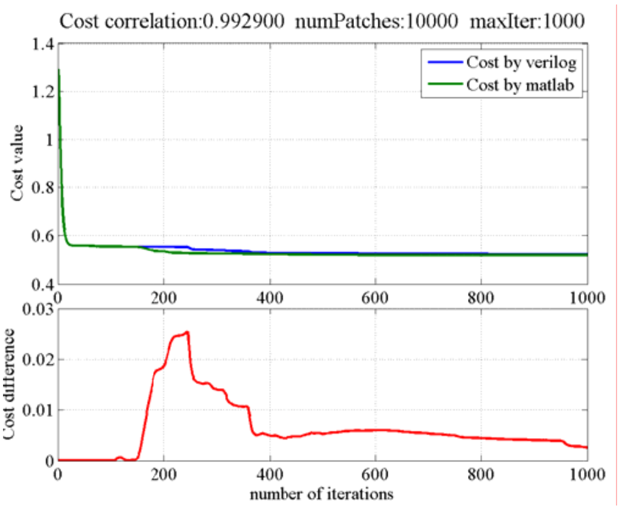


Fig. 2 Cost value and cost difference

Since we designed our own mathematical function, there was difference of parameter between both simulators. The converged cost values are shown in Fig. 2 with the difference. The difference converges to the value smaller than 0.01 with a bounce point at around 200 iteration. Fig. 3 also shows the aspect values of the each parameter in every 100 iterations.

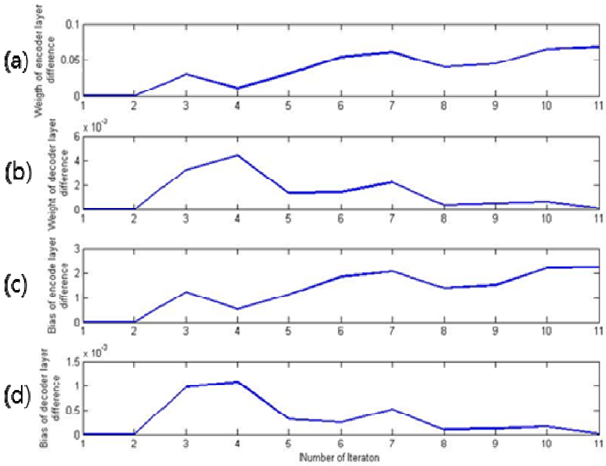


Fig. 3 Difference of parameters between the MATLAB and MODELSIM (a) Weight of encoder layer (b) Weight of decoder layer (c) Bias of encoder layer (d) Bias of decoder layer.

It was observed that correlation between results was varied across a scale of 0 to 0.9, with the greatest iteration of 1000, in

[6] Finite State Machine

terms of every 200 iterations, as seen in Fig. 4. The higher correlation they get, the more two weight sets become similar. It is important to focus on the diagonal direction of the correlation plot because each weight set should be compared in the same iteration.

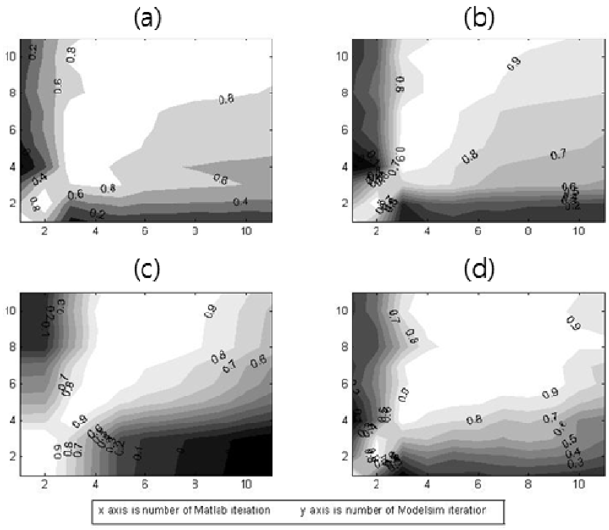


Fig. 4 Correlation of parameter from verification experiment (a) Weight of encoder layer (b) Weight of decoder layer (c) bias of encoder layer (d) Bias of decoder layer

Fig. 5 shows the plot of the learned feature of the auto-encoder with natural image. In addition, the contrast of white and black edge shaped is much clear in results of MODELSIM rather than MATLAB's. It means that MODELSIM optimized the parameter sets better than MATLAB. Finally, this edge filter showed that the results from both simulations are almost the same.

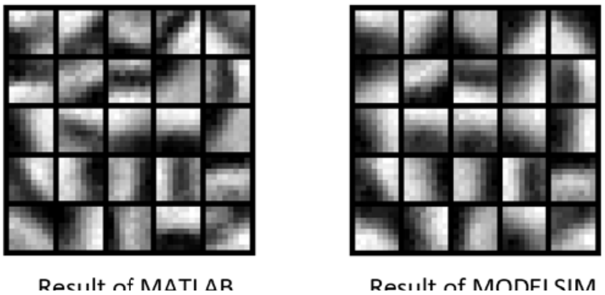


Fig. 5 Weight of encoder plot

Secondly, we tested the behavior model with same input patches and randomly selected initial parameters. Although the initial weight values were different, results from the weight values became very similar (See Figs. 6-8.) Fig. 6shows the cost values and cost difference. We observed that the cost values became almost equal after 200 iterations. The weight distributionwe obtained after the pre-training is plotted in the graph of Fig. 7. The distribution from both simulators verifies that the first layer feature becomes very similar after whole training process. It means that the designed behavior model and MATLAB code of auto-encoder, finally, reached to almost similar point of the weight space. Fig. 8 shows the process of change of the weight representation in every 100 iterations. The visualized features were changed from randomly selected weight space to edge shaped representation in the both plots.

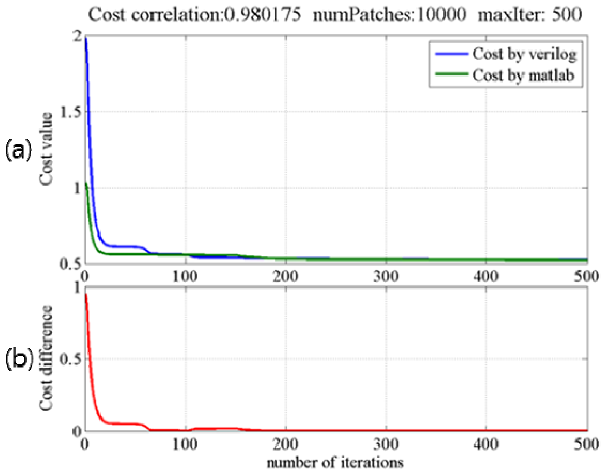
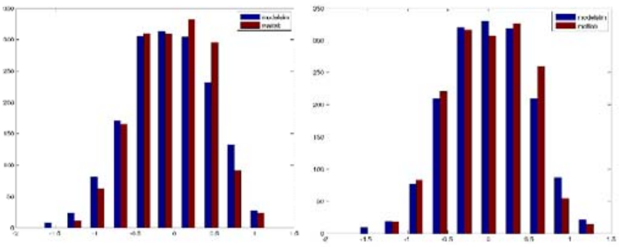


Fig. 6 The cost values and difference (a) blue line is cost from MODELSIM and red one from MATLAB (b) The difference value of two cost values



(a) (b)

Fig. 7 Weight distributions of same patches input and different initial weight value Blue stick is from MODELSIM and Red one is from MATLAB (a) Weight of encoder part (b)Weight of decoder part

VII. Conclusion

In this paper, we emphasized how pre-route simulation improves the way of designing a hardware with high-level deep learning algorithm, such as unsupervised learning auto-encoder algorithm. This may be a significant finding, because, with pre-route simulation, we could obtain the cycle accurate result of deep networks that is an essential factor in designing a digital hardware. The cycle accurate result of algorithms can serve as indicator for designing and modifying deep learning hardware in the digital method.

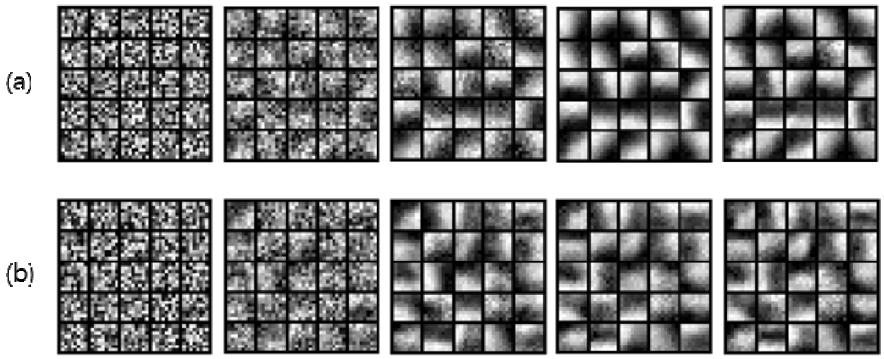


Fig. 8 Visualization of the process of change of auto-encoder architecture's first layer representations (a) Evolution of the first layer's feature representation from MATLAB in every 200 iteration (b) Evolution of the first layer's feature representation from MODELSIM (behavior model) in every 200 iteration.

First of all, we attempted to show the whole process by verifying the appropriate operation of the auto-encoder's behavior model in terms of pre-route simulation. We compared the weight space with the same input patches and initialized weight values between two simulators. We observed the similarity of extracted weights by plotting the difference and correlation between MATLAB's result and MODELSIM's one. Also, we found the high similarity of the latent representations by observing the visualized filter set. These results proved that the behavior model of unsupervised feature learning auto-encoder algorithm works well. Then, we tested the behavior model simulation with same input patches and different initial weight parameters. The result of the experiments showed that almost equal optimized cost values and the same weight distribution.

In addition, the verified behavior model of single auto-encoder can be extended to stacked auto-encoder's behavior model. With stacking our new-made single auto-encoder behavior model, it is possible to the greedy layer-wise unsupervised pre-training followed by fine-tuned in supervised learning through pre-route simulation. This was, moreover, a case study of designing the auto-encoder on the pre-route simulation.

REFERENCES

[1] Hinton, G. E., Osindero, S., Teh, Y., (2006). "A fast learning algorithm for deep belief nets", Neural Computation, 18, 1527~1554.

[2] Ranzato, Marc'Aurelio, et al. "Unsupervised learning of invariant feature hierarchies with applications to object recognition." Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on. IEEE, 2007.

[3] Ranzato, Marc'Aurelio, and Yann Adviser-Lecun. "Unsupervised learning of feature hierarchies." (2009).

[4] Sang Kyun Kim, Lawrence C. McAfee, Peter L. McMahon, Kunle Olukotun, "A highly scalable Restricted Boltzmann Machine FPGA implementation", FPL2009.

[5] Farabet, Clément, et al. "Hardware accelerated convolutional neural networks for synthetic vision systems." Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on. IEEE, 2010.

[6] Liang, Yun, et al. "High-level synthesis: Productivity, performance, and software constraints." Journal of Electrical and Computer Engineering 2012.

[7] Liu, Dong C., and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization." Mathematical programming 45.1-3 (1989): 503-528.

[8] Bengio, Yoshua, et al. "Greedy layer-wise training of deep networks." Advances in neural information processing systems 19 (2007): 153.

[9] G. Tesauro(1992), "Practical issues in temporal difference learning. Machine Learning, 8, 257~277",

[10] Erhan, Dumitru, et al. "Why does unsupervised pre-training help deep learning?." The Journal of Machine Learning Research 11 (2010): 625-660.

[11] Andrew Ng, "Sparse autoencoder", Stanford CS294A Lecture notes.

[12] Coates, Adam, Andrew Y. Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning." International Conference on Artificial Intelligence and Statistics. 2011.

[13] Ritter, Helge J., Thomas M. Martinetz, and Klaus J. Schulten. "Topology-conserving maps for learning visuo-motor-coordination." Neural networks 2.3 (1989): 159-168.

[14] Bottou, Léon. "Large-scale machine learning with stochastic gradient descent." Proceedings of COMPSTAT'2010. Physica-Verlag HD, 2010. 177-186.

[15] Ngiam, Jiquan, et al. "On optimization methods for deep learning." Proceedings of the 28th International Conference on Machine Learning.

[16] Bengio, Yoshua. "Deep Learning of Representations for Unsupervised and Transfer Learning." Journal of Machine Learning Research-Proceedings Track 27 (2012): 17-36.

[17] Farabet, Clément, et al. Large-scale FPGA-based convolutional networks. Cambridge, UK: Cambridge University Press, 2011.

[18] Omondi, Amos R., and JagathChandanaRajapakse, eds. FPGA implementations of neural networks. Vol. 365. New York, NY, USA:: Springer, 2006.

[19] Erhan, Dumitru, et al. Visualizing higher-layer features of a deep network. Technical report, University of Montreal, 2009.

[20] Kullback, "Information theory and statistics", John Wiley and sons, NY, 1959.