Implementation of Evolutionary Algorithms for Deep Architectures

Sreenivas Sremath Tirumala

Abstract. Deep learning is becoming an increasingly interesting and powerful machine learning method with successful applications in many domains, such as natural language processing, image recognition, and hand-written character recognition. Despite of its eminent success, limitations of traditional learning approach may still prevent deep learning from achieving a wide range of realistic learning tasks. Due to the flexibility and proven effectiveness of evolutionary learning techniques, they may therefore play a crucial role towards unleashing the full potential of deep learning in practice. Unfortunately, many researchers with a strong background on evolutionary computation are not fully aware of the state-of-the-art research on deep learning. To close this knowledge gap and to promote the research on evolutionary inspired deep learning techniques, this paper presents a comprehensive review of the latest deep architectures and surveys important evolutionary algorithms that can potentially be explored for training these deep architectures.

Index terms — Deep Architectures, Deep Learning, Evolutionary Algorithms

1 Introduction

Deep Learning is a topic of high interest with its extensive application in natural language processing, image recognition [1] [2] and computer vision. Corporate giants like Google, Microsoft, Apple, Facebook, Yahoo etc. established their deep learning research groups for implementing this concept in their products. Applications based on deep learning have won numerous machine learning competitions in ICML and NIPS with considerable margins which were earlier dominated by other machine learning approaches like Support Vector Machines. In 2013 it has topped in Chinese Handwriting Recognition Competition, Galaxy Zoo Competition, MICCAI 2013 Challenge, Merck Drug Discovery Competition, Dogs versus Cats Competition etc. Deep Learning is rated as the most interesting topic of research interests by Massachusetts Institute of Technology (MIT).

The importance of studying deep architectures is motivated from the deep architecture found in human brain. It is a common practice to reduce a high level problem into a set of low level problems in a hierarchical manner with easiest problem at the bottom. Interestingly, deep architectures based systems can achieve the learning that a shallow architecture can, but the vice versa is not feasible [3]. A Deep Neural Network (DNN) is an Artificial Neural Network

(ANN) with multiple hidden layers. One of major problems of DNNs is overfitting which was unaddressed till 2014 [4]. Further, due to the extensive use of gradient descent based learning techniques, DNNs may easily be trapped into local optima, resulting in undesirable learning performance. Moreover, the initial topology of DNN is often determined through a seemingly arbitrary trial and error process. However, the fixed topology thus created may seriously affect the learning flexibility and practical applicability of DNNs. Deep learning has been applied on other machine learning paradigms like Support Vector Machines and Reinforcement Learning.

In this paper, we argue that Evolutionary Computation (EC) techniques can, to a large extent, present satisfactory and effective solutions to above mentioned problems. In fact, several Neuroevolutoinary systems have been successfully developed to solve various challenging learning tasks with remarkably better performance than traditional learning techniques. Unfortunately, many researchers with a strong background in evolutionary computation are still not fully aware of the state-of-the-art research on deep learning. To meet this knowledge gap and to promote the research on evolutionary inspired deep learning techniques, this paper presents a review of latest deep architectures and surveys important evolutionary algorithms that can potentially be explored for training these deep architectures. This paper is divided into 5 sections. Section 1 details the history of deep architectures. Section 2 provides a detailed study on various deep architectures. Recent implementations of evolutionary algorithms on deep architectures are explored in section 3. Section 4 summarizes the paper with outcomes and conclusion.

2 Deep Architectures

Deep architecture is a hierarchical structure of multiple layers with each layer being self-trained to learn from the output of its preceding layer. This learning process i.e., 'deep learning' is based on distributed representation learning with multiple levels of representation for various layers. In simple terms, each layer learns a new feature from its preceding layer which makes the learning process concrete. Thus, the learning process is hierarchical with low level feature at the bottom and very high level feature at the top with intermediate features in the middle that can also be utilized. From these features, greedy-layer-wise training mechanism enables to extract only those features that are useful for learning. Along with this, a pre-unsupervised training with unlabelled data makes deep learning more effective.

Shallow architectures have only two levels of computation and learning elements which makes them inefficient to handle training data [5]. Deep architectures require fewer computational units that allow non-local generalization which result in increased comprehensibility and efficiency that has been proved with its success in Natural Language Processing (NLP) and image processing. According to complexity theory of circuits, deep architectures can be exponentially more efficient than traditional narrow architectures in terms of functional

representation for problem solving [5]. Traditional Artificial Neural Networks (ANNs) are considered to be most suitable for implementing deep architectures.

In 1980 Fukushima proposed Neocognition using Convolutional Neural Networks (ConvNets) [6] which served as a successful model for later works on deep architectures which later been improved by Lecun [7]. The theoretical concepts of deep architecture were proposed in 1998 by Lecun [8]. The Breakthrough in the research of training deep architectures was achieved in 2006 when Lecun, G.E. Hinton and Yoshua Bengio proposed 3 different types of deep architectures with efficient training mechanism. Lecun implemented efficient training mechanism for ConvNets [9] in which he was not successful earlier. Hinton implemented Deep Belief Networks (DBNs) [10] and Yoshua Bengio proposed Stacked Autoencoders [11].

A simple form of deep architecture implementation is DNNs, feed-forward ANNs with more than one hidden layer units that make them more efficient than a normal ANNs [12]. DNNs are trained with BP by discriminative probabilistic models that calculate the difference between target outputs and actual outputs. The weights in the DNNs are updated using stochastic gradient descent defined as $\Delta w_{ij}(t+1) = \Delta w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}}$, where η represents the learning rate, C is the associated cost function and w_{ij} represents weight. For larger training sets, DNNs may be trained in multiple batches of small sizes without losing the efficiency [13]. However it is very complex to train DNNs with many layers and many hidden units since the number of parameters to be optimized are very high.

2.1 Convolutional Neural Networks (ConvNets)

ConvNets are a special type of feed-forward ANNs that performs feature extraction by applying convolution and sub sampling. The principle application of ConvNets is feature identification. ConvNets are biologically inspired MLPs based on virtual cortex principle [14] and the earliest implementation is by Fukushima in 1980 [6] for pattern recognition followed by Lecun in 1998 [8]. ConvNets diversify by applying local connections, sub sampling and sharing the weights which is similar to the principle approach of ANNs in early 60s. In ConvNets each unit in the layer receives input from set of units in small groups from its neighbouring layer which is similar to earlier MLP model. Using local connections for feature extraction has been proven successful, especially for extracting edges, end points and corners. These features extracted at the initial layer will be combined subsequently at the later layers to achieve higher or better features. The features that are detected at the initial stages may also be used at the subsequent stages. The training procedure of the ConvNets is shown in Fig. 1. The first layer takes a raw

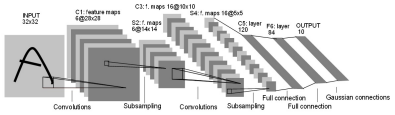

Fig. 1. ConvNets Structure proposed by Lecun [9]

pixel with 32 x 32 from the input image. The second layer consists of 6 kernels with 5 x 5 local window. From this, a sub sampling is done in the 3rd layer (sub sampling) layer. For the 4th layer, another ConvNets with 16 kernels was exploited with the same 5 x 5 windows. Then the 5th layer is also constructed using sub sampling. This procedure continues till the last layer and the entire structure is developed as Gaussian connections.

2.2 Deep Belief Networks

Deep Belief Network (DBN) is a type of DNN proposed by Hinton in 2006 [15]. DBN is based on MLP model with greedy layer-wise training. DBN consists of multiple interconnected hidden layers with each layer acting as an input to the next layer and is visible only to the next layer. Each layer in a DBN has no lateral connection between its nodes present in that layer. The nodes of DBN are probabilistic logic nodes thus allowing the possibility of using activation function. Restricted Boltzmann machine (RBM) is stochastic ANN with input and hidden units with each and every connection connecting a hidden and visible unit. RBMs act as the building blocks of DBNs because of their capability of learning probabilistic distributions on their inputs. Initially the first layer of the DBNs is trained as RBM that transforms input into output. The output thus received is used as data for the second layer which is treated as a RBM for the next level of training and the process continues. Similarly the output of the second layer will be the input for the third layer and the process continues as shown in Fig. 2 .The transformation of data is done using activation function or sampling. In this way the subsequent hidden layer becomes a visible layer for current hidden layer so as to train it as a RBM. An RBM with two layers, a visible layer as layer 1 and a hidden layer as layer 2 is the simplest form of DBN. The units of the visible layer are used to represent data and the units (hidden with no connection between them) will learn to represent features. If a hidden layer 3 is added to this, then layer 2 will be visible to only layer 3 (still hidden to layer 1) and now the RBM will transform the data from layer 2 to layer 3. This process is illustrated in Fig. 2.

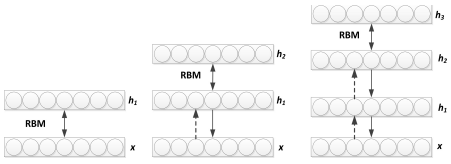

Fig. 2. Structure of Deep Belief Networks [15]

In 2006, Hinton proposed a greedy layer-wise unsupervised pre-learning algorithm for training that addresses the problem of training multilayer ANNs [10]. In DBNs, the lower level features of the input are extracted as lower layers and an abstract representation (high level features) of the input is performed at the higher layers. The training procedure of DBNs is carried out in three phrases. Each layer of the DBN is pre-trained with greedy layer wise training followed by unsupervised learning for each layer and finally training the

entire network with supervised training. The significance of this training procedure is determined by the generative weights. After learning, the values of the latent variables in every layer can be inferred by a single, bottom-up pass that starts with observed data vector in the bottom layer using generative weights in the reverse direction. DBNs proved to be the most efficient in image recognition [10], Face Recognition [16], Character Recognition [11] and various other applications.

2.3 Stacked Auto-encoders

The idea of auto-encoders is evolved from the process of reducing dimensionality of data by identifying efficient method to transform complex high dimensional data into lower dimensional code using an encoding multilayer ANN. A decoder network will be used to recover the data from the code. Initially both encoder and decoder networks are assigned with random weights and trained by observing the discrepancy between original data and output obtained from encoding and decoding. After this the error is back propagated first through the decoder network followed by encoder network and this entire system is named as auto-encoders [15].

An auto-encoder with input $x \in R^d$ is "encoded" as $h \in R^{d^1}$ using deterministic function defined as $f_\theta = \sigma(Wx + b), \theta = W, b$. To "decode", a reverse mapping of $f : y = f_{\theta^1}(h) = \sigma W^1 h + b^1$ with $\theta = (W^1,\ b^1)$ and $W^1 = W^T$ with encoding and decoding with the same inputs. This process continues for every training patten. For i training x_i is mapped to h_i with a reconstruction y_i. Parameter optimization is achieved by minimizing the cost function over the training set. However, optimizing an auto-encoder network with multiple hidden layers is difficult. Being similar to DBN greedy layer wise training procedure, this approach replaces RBMs by auto-encoders that perform learning by reproducing every data vector from its own feature activation [5]. The considerable change that has been applied in this model by Yoshua Bengio is changing the un supervised training procedure to supervised in order to identify the significance of training paradigm.

The process of greedy layer wise training is as follows. In the entire ANN, three layers are considered at one instance with the middle layer being the hidden layer. In the next instance, the middle layer becomes input layer and the output layer of the previous instance become hidden layer (the parameters from the output becomes the training parameters) and the layer next to it will be the new output layer. This process continues for the entire network. However, the results were not efficient since the network becomes too greedy [5]. It can be concluded that, the performance of stacked auto-encoders with unsupervised training was almost similar to that of RBNs with similar type of training whereas stacked auto-encoders with supervised pre-training is less efficient. Stacked auto-encoders were not successful in ignoring random noise in its training data due to which its performance is slightly less (almost equal performance but not same) than RBM based deep architectures. However, this gap is narrowed by stacked de-noising auto-encoder algorithm proposed in 2010 [17].

3 Applying Evolutionary Algorithms on Deep Architectures

3.1 Generative Neuroevolution for Deep Learning

In 2013 Phillip Verbancsics and Josh Harguess proposed Generative Neuroevolution for Deep Learning by implementing HyperNEAT as a feature learner on a ANN similar to ConvNets [18]. Compositional pattern producing network (CPPN) is an indirect encoding procedure of HyperNEAT that encodes weight patterns of ANN using composite functions. The topology and weights required for CPNN is evolved by HyperNEAT. In HyperNEAT process, CPPN defines an ANN as a solution for required problem. CPNNs fitness score is determined by evaluating the ANNs performance for the task for which it is evolved. Diverging from traditional methods, this approach trains ANN to learn features by transforming input into features. Then these features are evaluated by Machine Learning (ML) approach thus defining the fitness of CPNN. Therefore, this process will maximize the performance of the learned solution since HyperNEAT determines the best features out of other ML approach. ConvNets can be represented in a graph like structure with coordinates of the nodes associated with each other which are similar to HyperNEAT structure. This similarity enables to apply HyperNEAT on ConvNets based architectures.

For the experiment, an eight dimensional Hypercube representation of CPNN is used with f-axis as feature axis, x-axis as neuron constellation of each feature and y-axis being pixel locations. HyperNEAT topology is a multilayer neural network with layers traveling along z-axis with CPPN representing the points in an eight-dimensional Hyper-cube that corresponds to connections in the four dimensional substrate. The location of each neuron can be identified using (x,y,f,z) coordinate and each layer can be represented with a trait constituting number of features(F) with X and Y dimensions. HyperNEAT is applied to the LeNet-5 [8]. The experiment is conducted on MNIST database with a population size of 250 with 30 runs for 2500 generations. With this comparative results its been concluded that HyperNEAT with ANN architectures is overthrown by HyperNEAT with CNN architecture.

3.2 Deep Learning using Genetic Algorithm

In 2012, Joshua proposed a learning method for deep architectures using genetic algorithm [19]. A DNN for image classification is implemented using a genetic algorithm and training each layer using generic algorithm. Further this study tries to justify the possibility of using genetic algorithms to train non trivial DNNs for feature extraction. Initially a matrix representing the DNN is generated with Sparse Network Design with most of the values being close to zero, whereas the ideal solution in this case is an identity matrix. The genetic sequence of individuals with non-zero elements (which is considered as a gene) is kept and computed instead of re-generating the complete matrix which will reduce the amount of data required to store in the matrix and the process complexity. The

position of the gene in the matrix can be determined by row and column and every gene has a magnitude.

The proposed algorithms are tested on image data normalized in the range of 0.0 and 1.0. Apart from applying to image data, the algorithm has been applied to handwriting, face image (small and large) and cat image identification. The experimental results section shows the reconstruction (of input) error rate for each experiment. Another experiment for reconstruction of faces with noisy data claim to prove that this algorithm is not just copying blocks of data, but generating the connections in the data and reconstructing the initial image. The theoretical limitations of the algorithm is not addressed. The cost of reconstruction becomes 0 for a single training image as it will be efficient only with a large set of data.

4 Conclusion

This paper provides a theoretical review of standard deep architectures and study the possibilities of implementing evolutionary computation principles on deep architectures. Apart from introducing various types of deep architecture, this paper provides a detailed explanation of their training procedure and implementations. Further, this paper analyses the implications of applying evolutionary algorithms on deep architectures with details of two such implementations and a critical review on their achievement. The Neuroevolution approach for deep architectures that is discussed in previous section is with respect to the application of HyperNEAT on deep architectures. The success of this proposed method cannot be determined since CNN holds the best classification for MNIST database. But, this drives a way of implementing Neuroevolution algorithms on deep architectures. Similarly, the second work of using genetic algorithms for training DNNs, justifies the possibility of using genetic algorithms for training deep architectures but does not show any signs of comparative studies of its efficiency with respect to speed or quality.

It is noteworthy that evolutionary algorithms may not be a complete replacement for deep learning algorithms at least not at this stage. However, the successful application of evolutionary techniques on deep architectures will lead to an improved learning mechanism for deep architectures. This might result in reducing the training time which is the main drawback for deep architectures. Future direction in this research could be evolving an optimized deep architecture based neural networks using Neuroevolutonary principles. This could provide a warm start to the deep learning process and could improve the performance of the deep learning algorithms.

References

1. Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 253–256, May 2010.

2. J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *In NIPS*, 2012.
3. Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009. Also published as a book. Now Publishers, 2009.
4. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.
5. Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," in *Large Scale Kernel Machines* (L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, eds.), MIT Press, 2007.
6. K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.
7. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems (NIPS 1989)* (D. Touretzky, ed.), vol. 2, (Denver, CO), Morgan Kaufman, 1990.
8. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.
9. M. A. Ranzato, C. S. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *NIPS*, pp. 1137–1144, 2006.
10. G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, pp. 1527–1554, July 2006.
11. Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. Montral, and M. Qubec, "Greedy layer-wise training of deep networks," in *In NIPS*, MIT Press, 2007.
12. G. Tesauro, "Practical issues in temporal difference learning," in *Machine Learning*, pp. 257–277, 1992.
13. G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.
14. D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *Journal of Physiology (London)*, vol. 195, pp. 215–243, 1968.
15. G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, 2006.
16. Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.
17. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010.
18. P. Verbancsics and J. Harguess, "Generative neuroevolution for deep learning," *CoRR*, vol. abs/1312.5355, 2013.
19. J. Lamos-Sweeney, "Deep learning using genetic algorithms. Master thesis, Institute Thomas Golisano College of Computing and Information Sciences," 2012. Advisor: Gaborski, Roger.