

# Implementation of Evolutionary Algorithms for Deep Architectures

Sreenivas Sremath Tirumala

No Institute Given

**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 their flexibility and proven effectiveness, evolutionary learning techniques 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 meet 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. Big corporates like Google, Microsoft, Apple, Facebook, Yahoo etc. established their deep learning research groups for implementing this concept in their products. Deep learning has 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 vs Cats Competition etc. It has been rated as the top most topic of research interests by MIT.

Deep learning as its name implies is based on deep architecture, a hierarchical learning paradigm different from traditional learning methods i.e, narrow learning. The theoretical concepts of deep architecture were proposed in 1998 by Lecun [3]. Since then, there was a continuous research on implementing a standard learning mechanism for deep architecture which were unsuccessful until 2006 when LeCun, G.E.Hinton and Yoshua Bengio were the first to implement

deep learning algorithms with Convolutional Neural Networks, Deep Belief Networks and Stacked Auto-encoders respectively [4] [5] [6]. Deep learning has been applied on various other machine learning paradims like SVM, RL, etc.

The longest path between input and output points in a flow graph constitute depth. In case of a feed forward ANN, the depth is the summation of number of hidden layers plus 1 (output layer). A Deep Neural Network (DNN) is an ANN with multiple hidden layers. The importance of studying deep architectures is motivated from the deep architecture found in human brain. Studies on visual cortex enables us to understand that the information in the brain is stored in a hierarchical representation with each level representing the features as abstractions with most high level being at the top. Similarly, cognitive science proves that the human learning process is hierarchical with easy task being learnt first before moving up the ladder to reach task with highest difficulty. It is a common practice to reduce a high level problem into a set of low level problems in a hierarchy manner with the problem that is easy to solve at the bottom. Further, this hierarchical abstractions givesn intermediate observations that can be re-used for another process similar to the statistical sharing principle which has been proven efficient. Though insufficient depth may result in low efficiency, deep architectures may not be always needed. However, Deep Architectures based systems can achieve the learning that a shallow architecture can, but the vice versa is not feasible [7]. Re-using various intermediate tasks and components of deep architectures will reduce uncertainty which is similar to distributed representations. This principle of sharing statistical strengths is the core idea of machine learning and similar approach is also followed in self-taught learning.

There are several major problems for DNNs. Over-fitting is one of them that frequently appears in many learning systems. To tackle over-fitting, the dropout mechanism was proven to be both general and effective [8]. In addition to over-fitting, due to the extensive use of gradient descent based learning techniques, the learning system may be easily trapped by some 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.

In this paper, we argue that Evolutionary Computation (EC) techniques can, to a large extent, present satisfactory and effective solutions to all these problems. In fact, several Neuroevolutionary 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 comprehensive review of the 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 history of deep architectures. Section 2 provides a detailed study of various deep architectures. Recent implementations of evolutionary algorithms on deep architectures is explored in section 3. A discussion about applying evolutionary algorithms constitute section 4 and finally, section 5 summarizes the paper with 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 concretized. By this, the learning procedure follows a hierarchy by transforming a low level representation at the first layer to a very high level feature at the last layer with multiple intermediate stages. The learning of these intermediate stages can also be utilized. Deep architectures empower deep learning strategy using greedy-layer-wise training mechanism which enables to extract only those features that are useful for learning. Apart from layer-wise training, a pre-unsupervised training with unlabeled data makes deep learning successful.

Shallow architectures have only two levels of computation and learning elements which makes them inefficient to handle training data [9]. Deep architectures require fewer computational units that allows 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 efficient than traditional narrow architectures in terms of functional representation for problem solving [9]. Traditionally Artificial Neural Networks (ANNs) are considered to be most suitable for implementing deep architectures.

The Development of ANNs learning paradigm started in 1940s [10] the introduction of perceptron in 1962 is responsible for increased research attention for ANNs [11] followed by a rapid development with introduction of new techniques to train ANNs more efficiently [12] [13]. A noticeable work on ANNs is done by J. J. Hopfield in 1982 by introducing Hopfield nets, a form of recurrent ANNs which addressed the issue of converging to local minima [14]. The implementation of Multilayer Perceptron (MLP) type feedforward networks is called Group Method of Data Handling (GMDH). GMDH nets of 1965 can be considered as the first learning method for deep architecture based system [15] which is considered as a starting point for first practically developed system in 1971 [16]. Each layer of GMDH nets are trained by regression analysis while increasing the number of hidden layers and the output is validated against a separate validation set.

In 1980 Fukushima proposed Neocognition using Convolutional Neural Networks (ConvNets) [17] which served as a successful model for later works on deep architectures. The Fukushima ConvNets used unsupervised learning rules to set

the initial weights [18] [19] whereas later works improved the concepts by using supervised Back Propagation (BP) for the same [20]. Though LeCun ConvNets based deep model was successful, it inherited the vanishing (exploding gradients) problem of ANNs with BP. This problem is known as Long Time Lag problem which is considered as the fundamental problem of deep learning [21] [22]. In 1998, Lecun used gradient descent training for implementing deep ConvNets that produced good results till that time in pattern recognition [3] But, the execution of ConvNets was very time consuming. Another problem with BP is that it cannot perform well without providing labeled data at the beginning which is not feasible in case of real world problems.

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 [4] in which he was not successful earlier, Hinton implemented Deep Belief Networks (DBNs) [23] and Yoshua Bengio proposed Stacked Auto-encoders [6].

A basic classification of deep architectures is presented next. Though there are some new emerging architectures, these architectures serve as a basis for most of the implementations.

## 2.1 Deep Neural Networks

A simple form of deep architecture implementation is DNNs, feed-forward ANNs with more than one hidden layer units that make it more efficient than a normal ANNs [24]. DNNs are trained with BP by discriminative probabilistic models that calculate the difference between target outputs and actual outputs [25]. The weights in the DNNs are updated using stochastic gradient descent as defined below

$$\Delta w_{ij}(t+1) = \Delta w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}} \quad (1)$$

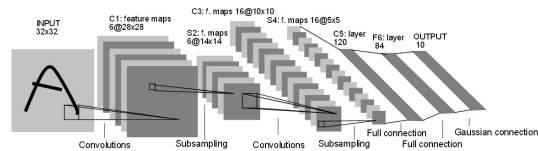
where  $\eta$  represents the learning rate,  $C$  is the cost function associated and  $w_{ij}$  represents weight. For larger training sets, DNNs may be trained in multiple batches of small sizes without losing the efficiency [26]. However it is very complex to train DNNs with many layers and many hidden units since the number of parameters to be optimized are very high.

## 2.2 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 [27] and the earliest implementation is by Fukushima in 1980 [17] for pattern recognition followed by Lecun in 1998 [3].

ConvNets diverge by applying local connections, sub sampling and sharing the weights which was the principle approach for ANNs in early 60s. In ConvNets

each unit in the layer receives input from set of units in small groups from its neighboring 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.



**Fig. 1.** ConvNets Structure proposed by LeCun [4]

The training procedure of the ConvNets is shown in fig.1. The first layer takes a raw 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 will be taken in the 3rd layer (sub sampling) layer. For the 4th layer, another Convolutional with 16 kernels was exploited with the same 5 x 5 windows. Then 5th layer is done using again sub sampling. This procedure continues till the last layer where the entire structure is developed as Gaussian connections.

### 2.3 Deep Belief Networks

Deep Belief Network (DBN) is a type of DNNs proposed by Hinton in 2006 [5]. 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. 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 layers will be the input for the third layers and the process continues as shown in Figure 2 The transformation of data can be 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.

In 1982, John Hopfield proposed a recurrent artificial neural networks called Hopfield nets that has content addressable memory. A Boltzmann Machine is a stochastic generative adaptation of Hopfield nets. Apart from having stochastic binary units, Boltzmann Machines are similar to Hopfield nets constructed by a network of binary units with a defined global energy  $E$  defined as

$$E = \sum_{i < j} w_{ij} s_i s_j + \sum_i \theta_i s_i \quad (2)$$

where  $w_{ij}$  is the connection strength between  $i$  and  $j$ ,  $s_i$  is state defined as binary  $s_i \in \{0, 1\}$

A stochastic ANN with input and hidden units with each and every connection connecting a hidden and visible units constitute RBM. RBMs are capable of learning probabilistic distributions on their inputs where as a recurrent neural network is an unrestricted BM that has multiple connections between input and hidden units. RBMs are first proposed by Paul Smolensky in 1986 [28]. RBM acts as a layer builder for DBNs.

In 2006, Hinton proposed a greedy layer-wise unsupervised pre-learning algorithm for training that addresses the problem of training multilayer ANNs. In DBNs, the lower level features of the input are extracted as lower layers and an abstract representation of the input is done at the higher layers. The training procedure of DBNs is done 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 i.e, learning is done by efficient layer-by-layer procedure and the dependency between one layer on its above layer 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 an observed data vector in the bottom layer using generative weights in the reverse direction.

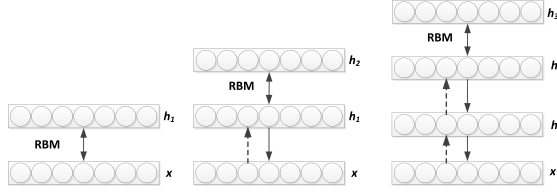
An RBM with two layers, a visible layer as layer 1 and a hidden layer as layer 2 is the simplest form of DBN. The units (visible) of the visible layer is used to represent data and the units (hidden with no connection between them) will learn to represent features. If a hidden layer 3 is added to this, 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 Figure2.

If  $v$  is a activities vector of visible units, the probability of generating  $v$  can be determined by

$$p(v) = \sum p\left(\frac{h}{v}, W\right) p\left(\frac{v}{h}, W\right) \quad (3)$$

where  $W$  is a matrix of symmetrically weighted connections between visible and hidden units,  $h$  is a sample vector taken alternatively from  $p\left(\frac{h}{v}, W\right)$  and  $p\left(\frac{v}{h}, W\right)$  represents the posterior distribution.

Provided a vector of activities for units of visible layer  $v$ , a sample  $h$  for hidden units can be achieved using factorial posterior distribution. The learning signal can be determined by the difference between the pairwise correlations of the visible and hidden units. To simplify further the probability of generating visible vector  $v$ . So, the difference pairwise correlations of the visible and hidden units at the beginning and end of the sampling constitute the learning signal similar to a RBM. Once  $W$  is achieved,  $p\left(\frac{v}{h}, W\right)$  will be replaced by a better model of non-posterior distribution achieved by averaging all available posterior



**Fig. 2.** Structure of Deep Belief Networks [5]

distributions over hidden vectors which showed better performance and learning rate [23]. For a DBN with  $l$  number of hidden layers, the joint distribution of probability of generating visible vectors for an input  $x$  can be represented as

$$P(x, h^1, h^2, \dots, h^{l-1}, h^l) = P(x|h^1)P(h^1|h^2)\dots P(h^{l-1}|h^l) \quad (4)$$

Hinton used binary random vector to represent each hidden layer which made the training procedure easy and efficient. The generative model for the  $i^{th}$  hidden layer  $h^i$  with  $j$  units and  $n^i$  elements with bias  $b$  is

$$P(h^i, h^{i+1}) = \prod_{j=1}^{n^i} P(h_j^i, h^{i+1})P(h_j^i = 1|g^{i+1}) = \text{sigm}(b_j^i + \sum_{k=1}^{n^{i+1}} W_{kj}^i h_k^{i+1}) \quad (5)$$

where  $\text{sigm}(t) = 1/(1 + e^{-t})$

This model is also applicable for the first layer  $P(h^0, h^1)$  where  $x$  is denoted as  $h^0$ . DBNs proved efficient in image recognition [23] [29] [6] and various other applications.

## 2.4 Stacked Auto-encoders

The concept of auto-encoders came from the process of reducing dimensionality of data by identifying efficient method to transform high dimensional data which is complex to optimize into a lower dimensional code using an encoding multi-layer 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 [30] and this entire system is named as auto-encoders [5]. 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 pattern. 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 [9]. With input vector  $x, x_i \in (0, 1)$  (same as binary for RBMs), the reconstruction probability is  $p(x_i)$  for bit  $i$  with probability vector of  $p(x) = \text{sigm}(c + W \text{sigm}(b + W^1 x))$  where  $W$  is the weight matrix and  $b$  is hidden biases column vector and  $c$  is the input biases column vector. The reconstruction cross-entropy is minimized using  $R = -\sum x_i \log p_i(x) + (1 - x_i) \log(1 - p_i(x))_i$ . The considerable change that has been applied in this model by Yoshua Bengio is changing the unsupervised training to supervised to identify the significance of training paradigm. By greedy layer wise supervised training means considering one hidden layer at a time as a hidden layer of a 3 layer ANN (the output of last trained layer will be the input layer for the current hidden layer i.e., a simple ANN) and the parameters of hidden layer serve as pre-training parameters for the next layer. However, the results were not efficient since the network becomes too greedy [9]. 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 was not efficient.

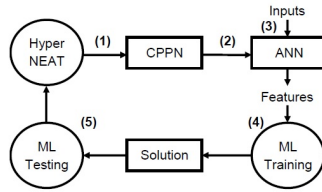
Stacked auto-encoder was not successful in ignoring random noise in its training data due to which the performance of stacked auto-encoders based deep architecture is slightly less than (almost equal performance but not same) the performance of RBMs based architecture [31]. To overcome this Stacked Denoising auto-encoder algorithm was proposed in 2010 with which the performance gap between RBM based and auto-encoder based deep architectures was narrowed [32].

### 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 [33]. Compositional pattern producing network (CPPN) is an indirect encoding procedure of HyperNEAT that encodes weight patterns of ANN using composite functions. The topology and weight 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 evaluating the ANNs performance for the task for which it is evolved. Diverging from traditional approach, this approach trains ANN to learn features by transforming input into features. Then these features are evaluated by another ML approach by applying to the tasks, thus defining the fitness of CPNN. Thus, this process will maximize the performance of the learned solution since HyperNEAT determines the best features out of other ML approach. HyperNEAT without any modifications, generates weight patterns for fully connected feed forward ANN substrate with only sigmoid activation functions. From this point, the only visible part for HyperNEAT



**Fig. 3.** Generative Neuroevolution for Deep Learning [33]

is a geometric coordinate structure of the neurons similar to a graph. ConvNets can be represented in a graph like structure with coordinates of the nodes associated with each other which is similar to HyperNEAT structure. This similarity enables to apply HyperNEAT on ConvNets based architectures as shown in 3.

For the experiment, an eight dimensional Hypercube representation of CPPN is used with f-axis as feature axis, x-axis constitute neuron constellation of each feature with y-axis being respective 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 correspond 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 tripe constituting number of features(F) with X and Y dimensions. HyperNEAT is applied to the LeNet-5 which is the first successful CNN architecture proposed by Lecun [3]. The experiment is conducted on MNIST database with a population size of 250 with 30 runs for 2500 generations. With the 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, a Master student Joshua proposed a learning method for deep architectures using genetic algorithm [34]. A DNN for image classification is implemented using a genetic algorithm and train each layer (similar to DNN training procedure) 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 where as the idle 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. Initial population is constructed by choosing individuals with only one randomly chosen gene. The length of the genotype can be increased by crossover whereas mutation has no affect on it. For each trivial addition to the genotype a penalty

has to be added to discourage the addition processes which might increase the size of the network with inefficient genotypes. However this process will not stop encouraging to have a large network with efficient genotypes. This makes computation faster but each addition of gene will complicate the mutation and cross over process. In the proposed design the mutation occurs by removing a gene from the parent and adding a new gene with random position and magnitude. By proposing a 50% chance of passing a gene from a parent to the child, the biasing of the solution towards large or smaller solutions is avoided and the children may contain same number of genes as that of parents or sometimes more. This 50% addition will further allow significant variation in the length of the gene.

The proposed algorithms based on sparse network design are tested on image data with normalizing in the range of 0.0 and 1.0. To speed up the experiment process, the convergence rate is increased by increasing the rate of mutation which had success for little time. The next step is Removing elitism, the ability of very old individuals to compete with a new generation which reduced the delay of execution. Further, maintaining the penalty is crucial as a small penalty may improve the speed whereas a large penalty may reduce the quality of the solution. The speed of the process may be further increased by executing each fitness calculation as an independent process. The selection procedure to remove an individual from the population or deciding which individual to keep, is based on time and reconstruction error.

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 noise claim to prove that the algorithm is not just copying set of block of data rather generating the connection in the data and reconstructing the image. The theoretical limitation 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 Discussion

With the understanding from previous sections, it is evident that evolutionary algorithms can be used to train deep neural networks considering the success of its application for various ML paradigms. The advantages of using an evolutionary algorithm instead of another learning method are that several defining features of the neural network can be genetically encoded and co-evolved at the same time and that the definition of a performance. It is noteworthy that evolutionary algorithms may not be a complete replacement for other 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.

The NeuroEvolution for deep learning approach discussed in previous section uses a fixed LeNet-5 topology and trains it using HyperNEAT. In other words, it's a comparative study between pure HyperNEAT and HyperNEAT with deep architecture. Though with respect to the application of HyperNEAT on deep architectures, the success of the proposed method cannot be determined since CNN holds the best classification for MNIST database. However, this drives a way of implementing NE algorithms on deep architectures.

Similarly with the second work of applying genetic algorithms for deep architectures, it is highly unlikely to compare the efficiency of genetic algorithm based learning approach with deep learning approach or conclude that the proposed approach is more efficient. This work rather justifies the possibility of using genetic algorithms for training deep architectures but does not show any signs of comparative studies of its efficiency with time or quality.

These evolutionary algorithms implementations on deep architectures is for improving their learning procedure. A future direction could be evolving a optimized deep architecture based neural networks using Neuroevolutionary principles. This could provide a warm start to the deep learning process and could improve the performance of the deep learning algorithms.

## 5 Conclusion

This paper provides a theoretical review of various types of deep architecture and studies the possibilities of implementing evolutionary computation principles for 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 analyzes the implications of applying evolutionary algorithms on deep architectures with details of two such implementations and a critical review on their achievement.

## 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. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.
4. 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.
5. G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, 2006.
6. 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.
7. 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.

8. 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.
9. 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.
10. W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 7, pp. 115–133, 1943.
11. F. Rosenblatt, *Principles of Neurodynamics*. Spartan, New York, 1962.
12. S. Grossberg, "Some networks that can learn, remember, and reproduce any number of complicated space-time patterns, I," *Journal of Mathematics and Mechanics*, vol. 19, pp. 53–91, 1969.
13. K. S. Narendra and M. A. L. Thathatchar, "Learning automata – a survey," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 4, pp. 323–334, 1974.
14. J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. of the National Academy of Sciences*, vol. 79, pp. 2554–2558, 1982.
15. A. G. Ivakhnenko and V. G. Lapa, *Cybernetic Predicting Devices*. CCM Information Corporation, 1965.
16. A. G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE Transactions on Systems, Man and Cybernetics*, no. 4, pp. 364–378, 1971.
17. 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.
18. K. Fukushima, "Artificial vision by multi-layered neural networks: Neocognitron and its advances," *Neural Networks*, vol. 37, pp. 103–119, 2013.
19. K. Fukushima, "Training multi-layered neural network Neocognitron," *Neural Networks*, vol. 40, pp. 18–31, 2013.
20. 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.
21. S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München," 1991. Advisor: J. Schmidhuber.
22. Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.
23. 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.
24. G. Tesauro, "Practical issues in temporal difference learning," in *Machine Learning*, pp. 257–277, 1992.
25. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," in *Neurocomputing: Foundations of Research* (J. A. Anderson and E. Rosenfeld, eds.), ch. Learning Representations by Back-propagating Errors, pp. 696–699, Cambridge, MA, USA: MIT Press, 1988.
26. 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.

27. 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.
28. P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, eds.), pp. 194–281, Cambridge, MA, USA: MIT Press, 1986.
29. 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.
30. D. C. Plaut and G. E. Hinton, "Learning sets of filters using back propagation," *Computer Speech and Language*, pp. 35–61, 1987.
31. H. Larochelle, D. Erhan, and P. Vincent, "Deep learning using robust interdependent codes," in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, pp. 312–319, Apr. 2009.
32. 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.
33. P. Verbanics and J. Harguess, "Generative neuroevolution for deep learning," *CoRR*, vol. abs/1312.5355, 2013.
34. J. Lamos-Sweeney, "Deep learning using genetic algorithms. Master thesis, Institute Thomas Golisano College of Computing and Information Sciences," 2012. Advisor: Gaborski, Roger.