International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 5 (2013) pp. 216-226

Adaptation of a deep learning machine to real world data

Ahmad A. Al Sallab[1] and Mohsen A. Rashwan[2]

[1] Department of Electronics and Communications, Faculty of Engineering,
Cairo University
ahmad.elsallab@gmail.com

[2] Professor, Department of Electronics and Communications, Faculty of Engineering,
Cairo University
mrashwan@rdi-eg.com

***Abstract*: Adaptation is a property of intelligent machines to update its knowledge according to actual situation. Self-learning machines (SLM) as defined in this paper are those learning by observation under limited supervision, and continuously adapt by observing the surrounding environment. The aim is to mimic the behavior of human brain learning from surroundings with limited supervision, and adapting its learning according to input sensory observations. Recently, Deep Belief Networks have made good use of unsupervised learning as pre-training stage, which is equivalent to the observation stage in humans. However, they still need supervised training set to adjust the network parameters, as well as being non-adaptive to real world examples. In this paper, SLM is proposed based on deep belief networks and deep auto encoders to adapt to real world unsupervised data flowing in to the learning machine during operation. As a proof of concept, the new system is tested on two AI tasks; number recognition on MNIST dataset, and E-mail classification on Enron dataset.**

***Keywords*:** *Deep Belief Networks (DBN), Restricted Boltzmann Machine (RBM), Adaptive learning*

I. Introduction

Traditional pattern recognition systems are composed of two phases; training and testing. These two phases are sequential. The training of the system with a well-known dataset is executed first and then the system is ready to be tested with the test set. Adaptive systems are able to update their learning according to certain events.

Recognition in human brains develops first by unsupervised observation of surroundings to learn differences between separate entities. Once the basic structure of the environment is captured inside the brain, supervision role starts so as to give labels to different categories. This role could be achieved by transfer of experience, or by asking pertinent questions to clarify ambiguity and learn the names of different entities. As the task proceeds, learning is adapted as more examples flow in and more experience is gained. New examples, never seen before, of the different categories are learned automatically to belong to the correct class, and hence the system is adapted. As we can see, adaptation is a principal component of the learning process in human beings.

Self learning machines, as defined in this paper, are those following the process described above to learn to classify patterns. A clear example of the above process is human child learning the names of figures. First, he observes different figures with no supervision to learn the differences between them. Then he is told their names, or he asks explicitly for the names of certain examples of each category and generalizes the name to the whole class. As he gets older, he faces more and more different examples, with new shapes never seen before, and yet he can adapt and develop himself to learn those new examples to belong to the right category according to his own belief.

Three components are identified to compose this self learning system. The first one is the unsupervised pre-training, which is responsible of feeding unsupervised examples to the machine to adapt itself to different clusters. The second is the supervised training to learn to classify examples based on true labels, so as to fine tune the network parameters based on the supervised data. And finally adaptive learning continuously adapting the machine to the new examples coming from real data, which could have never been seen before in unsupervised pre-training or supervised training phases. The unsupervised pre-training has proved to be effective in Deep Belief Networks [1] [3]. In this paper, the supervised training and adaptive learning are focused on.

In many practical learning domains, there is a large supply of high-dimensional unlabeled data and very limited labeled data. Applications such as information retrieval and machine vision are examples where large amounts of unlabeled data is readily available [3]. The need to supervised data with semi or minimal supervision in such applications is high, to compensate the lack of labeled data.

Minimum supervision means using few labeled examples, and using them to generalize to broader dataset. If efficient clustering of the unsupervised dataset is possible, then it is sufficient to know the labels of the means of clusters, or at least one supervised example of

each cluster to apply the same labeling to the whole cluster automatically.

In [3] non-linear dimensionality reduction is performed using deep auto-encoders, learned using greedy layer by layer learning algorithm in [1]. Applying linear mapping, like Neighborhood Component Analysis (NCA), on the non-linear generated codes of the deep auto-encoder creates non-linear mapping and gets the data to a new space where the examples can be efficiently clustered.

The continuous flow of data during operation is essential for adaptation of the system to real world examples. A system needs to be developed to handle learning adaptation based on incoming data. To be able to perform this task, the learning algorithm itself should exploit the unsupervised data in the first place. A typical candidate algorithm for that is the one presented in [1] and [3] of learning deep networks, or Deep Belief Networks (DBN) based on greedy Deep Boltzmann Machine (DBM) learning algorithm. This greedy layer by layer learning algorithm can make efficient use of very large sets of unlabeled data, and so the model can be pre-trained in completely unsupervised fashion. The limited labeled data can then be used to only slightly fine-tune the model for a specific task at hand using standard gradient based optimization. Adaptation as presented in our paper proposes that, the same learning process can be repeated whenever more supply of unsupervised data is available from normal operation.

A novel approach is developed in this paper based on deep networks concepts to achieve two goals; the first one is utilizing few examples during supervised learning phase by automatic labeling the unsupervised training data set using non-linear unsupervised clustering on top of the deep auto-encoder. The second one is network adaptation with the real world unsupervised examples, based on the greedy layer by layer unsupervised learning of DBN, then fine tune the network again with the already available supervised examples. With every batch of new input data, the learning is repeated and the network is updated periodically or on need basis.

Merging the two components of automatic labeling and adaptive learning constitutes the Self Learning Machine (SLM) as defined earlier. Those machines are able to learn with minimal supervision and adapt to real world data. SLMs are generic learning systems in terms that they can be adapted to any pattern recognition task. All what is needed are few representative examples of each class to be recognized, and then the system will build the network from the unsupervised input data flowing in. Recursively the network adapts its parameters as more and more new data is fed to the system, and the performance is enhanced automatically.

The next sections are organized as follows; first related work is demonstrated. The next section presents the automatic labeling component. Then adaptive learning component is described. The next section presents the experiments results and details. Finally the paper is concluded with the conclusion and future work.

II. Related work

An approach to imitate the behavior of V1 cortex in humans is found in recent work of Hinton et al. [1] on DBN, where the structure of the world is first captured by unsupervised pre-training, so the system is capable of generating similar examples of those learned without supervision [2]. The role of unsupervised pre-training for classification task is to get the network to a point in space near to the global minimum so that back propagation can start without getting stuck in local minima [1].

In [1],[3] and [4] it is proved that high-level representations can be built from a large supply of unlabeled inputs and the very limited labeled data can then be used to only slightly adjust the model for a problem-specific task.

In [3] work has been done on learning nonlinear mappings that preserve class neighborhood structure. This is the main idea on which automatic dataset labeling is built on. In this work, it was demonstrated how K-nearest neighbor classification can perform well on non-linear transformation of the input.

Future work in [3] suggests semi-supervised learning of deep Boltzmann's machines to target the applications of large supply of high-dimensional unlabeled data and very limited labeled data, like information retrieval and machine vision.

III. Deep Restricted Boltzmann Machine Training Review

Artificial Neural Networks (ANN) have been used for decades for classification tasks. It has grabbed the attention of researchers due to the similarity between its architecture and the human brains. The ANN is formed of several stacked D layers, each of width N_i neurons. The weights relating each consecutive layer i and $i+1$ is $W_{(i)(i+1)}$ matrix of dimension $N_i \times N_{i+1}$. The width of input layer is denoted by N_0 and the last layer by M.

The main focus of research in ANN is on training the network, i.e. to find the right weights that can be used to correctly classify the input examples. The most successful algorithm in that area is the famous back propagation algorithm.

The problem with back propagation is the following; ANN represents a non-linear mapping $f(X,W)$, where X is the input vector and W is the weight matrix of the whole network, as the number of layers increases, the function f gets more complicated, such that it contains multiple local minima. The back propagation algorithm converges to a certain minimum based on the initialization of weights W. Sometimes it gets stuck to a poor performance local minimum and not the global one. For some AI tasks, those local minima are fine, but not accepted for other cases.

Also, back propagation training time is not scalable with the depth of the network. As the number of layers increases, training time gets much higher. This could not be a big problem with the increasing power of computer nowadays.

Another disadvantage of back propagation is that it requires high supply of labeled data, which could not be available for many AI tasks requiring classification.

For the aforementioned problems, Hinton et al. have introduced a fast learning algorithm based on Deep Belief Networks (DBN) and Restricted Boltzmann Machine (RBM) to train a deep ANN [1].

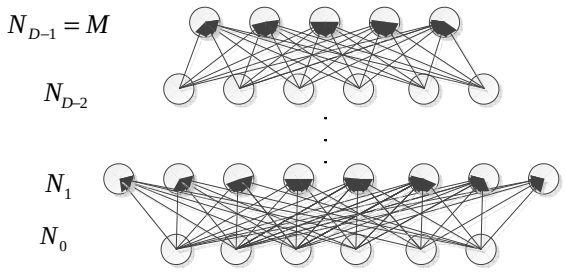

Figure 1 Deep Artificial Neural Network

A. Deep Belief Network model

The algorithm is based on modifying the ANN model from discriminative to generative model. The discriminative model is the one which models the classification performance of the network. The objective function to be minimized is the error between the required classification targets and the obtained ones. On the other hand, the generative model is the one that models the generation of original data capability of the network. The objective is to minimize the error between the model-generated data and the original one.

The generative model must be able to re-generate the original data given the hidden units states, this represents its belief of the real world data. This kind of model is called Deep Belief Network (DBN). The DBN model enables the network to generate visible activations based on its hidden units' states, this represents the network belief.

The problem now is how to get the hidden units states corresponding to the visible data. A Restricted Boltzmann Machine is proposed between each two consecutive layers of the network. The difference between ANN, DBN and RBM is shown in Figure 2

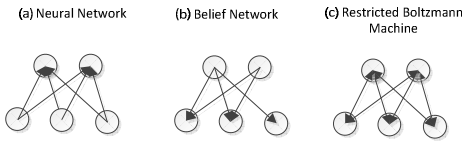

Figure 2 (a) ANN (b) DBN (c) RBM

B. Pre-training phase

To optimize a given configuration of visible and hidden units, the model energy is to be minimized.

$$E(v,h) = -\sum_{i,j} v_j h_j w_{ij}$$

$$w = w + \Delta w$$

$$\Delta w = -\frac{\partial E(v,h)}{\partial w_{ij}} = v_j h_j = LearningSignal$$

Our objective for a generative model is to maximize the probability of visible units' activations *p(v)*. To get *p(v)* we have to marginalize the *p(v,h)* probability for the whole configuration.

$$p(v,h) = \frac{e^{-E(v,h)}}{AllPossibleConfigurations} = \frac{e^{-E(v,h)}}{\sum_{<v,h>} e^{-E(v,h)}}$$

$$\sum_{<v,h>} e^{-E(v,h)} = PartitionFunction = PF$$

$$\therefore p(v) = \sum_h p(v,h) = \frac{\sum_h e^{-E(v,h)}}{\sum_{<v,h>} e^{-E(v,h)}}$$

For mathematical convenience, let's maximize the log *p(v)* instead of *p(v)*. Moreover, to maximize the data generation with respect to model belief, we adjust the weights such that model belief is decremented while the real data is incremented as follows:

$$\Delta w = \alpha(< \frac{\partial \log p(v)}{\partial w_{ij}} >_{\text{Realdata}} - < \frac{\partial \log p(v)}{\partial w_{ij}} >_{\text{Model}})$$

$$= \alpha(< v_i h_j >_{\text{Realdata}} - < v_i h_j >_{\text{Model}})$$

Finally the pre-training algorithm is simply to apply the data on the input layer of the 2-layer RBM to get the hidden activations, and then re-generate the model visible activations, and finally generate the model hidden activations. In this way the RBM weights can be updated for the given input data.

Having the current layer trained, its weights are frozen, and the hidden layer activations are used as the next layer visible inputs, and the same training algorithm is applied. The obtained Deep network weights are used to initialize a fine tuning phase.

C. Fine-tuning phase

The fine tuning phase is simply the ordinary back propagation algorithm. For classification tasks, a layer of width equals the number of targets or classes, is added on top of the network. Each neuron of this layer is activated for each class label while the others are deactivated. The back propagation starts from the weights obtained in pre-training phase. The top layer activations are obtained for each training set example, or batch of examples, is obtained in the forward path, and then the error signal between the obtained activations and required targets is back propagated in the network for weights adjustment.

IV. Automatic labeling system

Artificial intelligence tasks requiring classification and regression are mostly based on statistical learning of supervised training set, on the hope that this training set is sufficient to generalize well to the test set and real world examples. This classical model requires large supply of labeled training set. One of the major obstacles facing development of such systems is the availability of such huge labeled training set. In many practical learning domains, there is a large supply of high-dimensional unlabeled data and very limited labeled data. Applications such as information retrieval and machine vision are examples; where large amounts of unlabeled data are readily available.

A self learning system shall follow the same model, but it should not require such large amount of supervised examples, instead it should be able to learn with few supervised examples and generalize the learned concept to other examples in the training set.

The root cause of the problem is that if the input raw data or features are well structured, then the system could discover similarities in the examples and generalize the learned labels to all similar examples. However, the input data is not necessarily structured by nature.

Suppose for the moment that somehow structured input data is obtained, such that an efficient clustering algorithm could be run successfully to discover related examples, such that the number of clusters is the number of classes at hand. In this case only few labeled representative examples are needed of every cluster, and the system can then apply the same labeling on all the members of the cluster, and hence the whole dataset can be fully labeled. However, the assumption of structured input data or features is not always valid. It depends on the type of features extracted for the data.

A. *Deep versus shallow architectures*

To discover similarity between cluster members, it is traditional to make some transformation on the original data to improve the clustering algorithm performance. Among the most common algorithms used for such purpose are Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and Neighborhood Component Analysis (NCA), which performs linear transformation on the data to map it to another space where within class similarities are improved. A linear transformation has a limited number of parameters and it cannot model higher-order correlations between the original data dimensions.

Using a nonlinear transformation function low-dimensional representations that work much better than existing linear methods can be discovered, provided that the dataset is large enough to allow the parameters to be estimated [3].

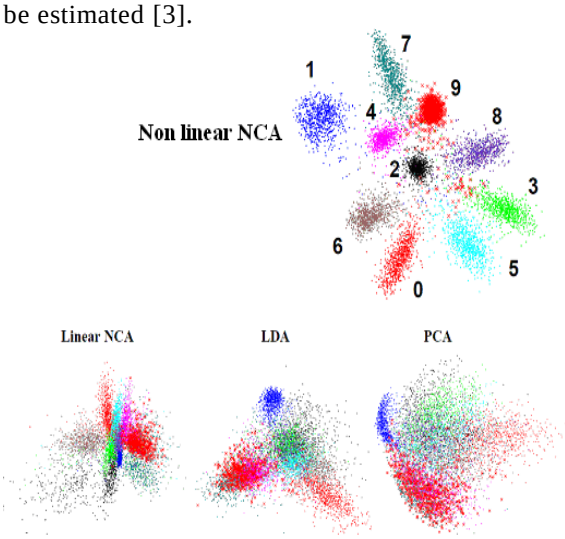

Figure 3 Non-linear NCA performance vs. other linear methods[3]

Figure 3 [3] describes the different mapping results of the MNIST dataset using different methods, linear methods: LDA, PCA and Linear NCA give poor structure for different classes of digits. The upper right graph shows the mapping with NCA on top of deep auto-encoder; non-linear NCA. It is clear how the different classes are well structured such that the operation of a clustering algorithm like K-means is enhanced.

B. *Deep auto encoder clustering*

To obtain the non-linearity discussed in IV.A using greedy unsupervised learning algorithm, deep auto-encoders can be used. Deep auto-encoder is a multilayer, nonlinear encoder network that transforms the input data vector x into a low-dimensional feature representation. The non-linear mapping function $f(x;W)$ can be trained using the algorithm described in IV.E. After the initial pre-training, the parameters can be fine-tuned by performing gradient descent in the Neighborhood Component Analysis (NCA) objective function [3].

Figure 4 shows the deep auto-encoder network architecture. The layers get narrower as we go deeper into the network. Number of layers is D. Original data is presented at Layer-0, and then next layers activations continue till Layer-D-1. The activations of Layer-D-1 are the code words representing the original data.

The upward generation path (right arrow in Figure 4) represents the code word generation. The downward generation path (left arrow in Figure 4) represents the model belief, where the original data is generated from the model according to the corresponding code word.

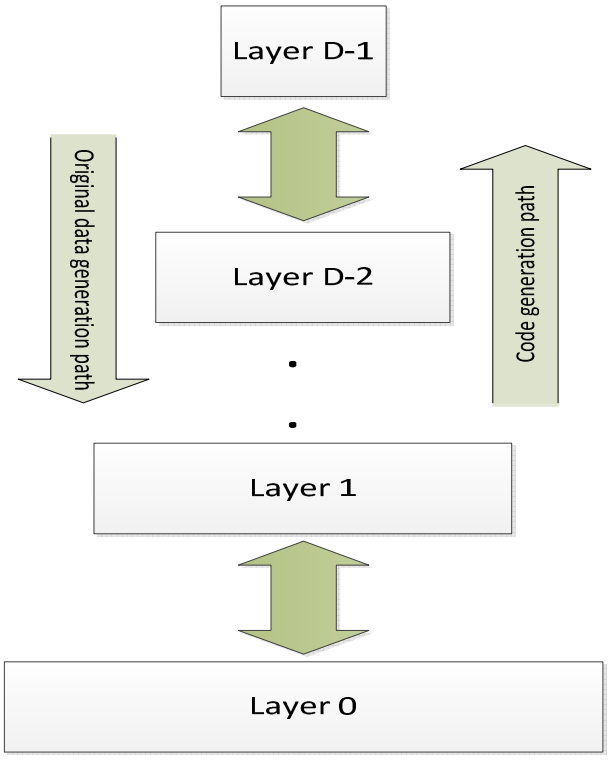

Figure 4 Deep auto-encoder network architecture

C. *Minimal supervision labeling*

After having obtained the desired structured data, an efficient clustering algorithm can be run to discover the structure of the data. Number of clusters will be the number

of classes, and the initial means of clusters will be the most representative examples of each class. For example, for the digit recognition task, the number of classes shall be 10, and the initial clusters means shall be representative image of each digit class. If the right labels for such representative means are available, then the whole cluster shall hold the same label, and hence a labeled dataset is obtained.

If generative model like deep auto-encoder is used, the system could be designed to generate the original raw input corresponding to the mean of each cluster, and ask the user explicitly for the label of this data, and then generalize the label to the whole cluster. This is very close to the behavior of human learning achieving minimal supervision.

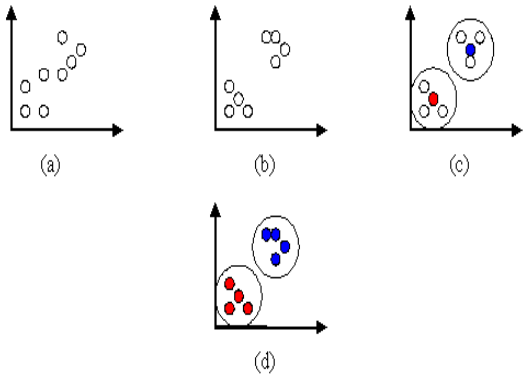

Figure 5 (a) Raw data (b) Structured data after non-linear transformation in the deep auto-encoder (c) Clustering algorithm discovers clusters means with their labels obtained (d) Having the labels of the means the whole cluster is labeled

Figure 5 describes the different mappings and processing on the input unlabeled data until a labeled dataset is obtained. First non-linear mapping is done using deep auto-encoder, such that structured dataset is obtained. Then a clustering technique (like K-means) is applied to the obtained codes. The labels of the clusters means or some representative class examples are provided by the user as supervised examples, form which the whole cluster can be labeled, and hence a supervised dataset is obtained.

D. Deep-auto encoder training algorithm

As described in IV.A and IV.B, deep auto encoder performs non-linear dimensionality reduction on the unlabeled data. Being a deep architecture, the structure in the mapped data is most likely to be well-clustered. Figure 6 describes the training algorithm of the deep auto-encoder.

The algorithm is based on the fast greedy learning algorithm in [1]. Let's denote the number of layer as *N_layers*. The activation of each layer as *data(i)*, with *data(0)* is the original data example. $W_{N_i \times N_{i+1}}$ is the weights matrix linking layer i to layer $i+1$, of size $N_i \times N_{i+1}$. ΔW is the weights update signal resulting from training. When a variable is meant for the whole network, sub-scripts are dropped for simplifying the notation. The sub-routing *generate_original_data* is used to obtain the model believed data to be the original data generate the code-word at the highest level (see Figure 4).

For each layer, weights are updated as if no more upper layers are going to be stacked over it. The neurons are binary, with S-shaped sigmoid functions. First the unlabeled data example is presented to the first layer, at which activations are obtained. RBM unlabeled training is performed as in [1] to obtain the $\Delta W_{N_i \times N_{i+1}}$ for each layer. After all layers are trained, the whole network weights are adjusted.

Deep auto-encoder architecture is a generative model. Hence, the error to be minimized or objective function to be optimized is the absolute difference between the originally presented data and the model believed data. This error signal is then used to be propagated back in the network to adjust the weights using the traditional back propagation algorithm. The final adjusted weights after *back_propagation_fine_tuning* are the result of the algorithm.

```
Sub_routine_train_deep_auto(unlabeled_dataset)
data(0) := unlabeled_dataset
for i := 1 : N_layers
    [ΔW_{N_i×N_{i+1}}, data(i)] := pre_train_rbm(Layer(i))
end_for
W = W + ΔW
code = data(N_layers)
Model_data = generate_original_data(code)
Original_data = data(0)
err_signal = | Model_data − Original_data |
[W] = back_propagatio n_fine_tuning(err_signal, W)
return (W)
```

Figure 6 Deep auto-encoder training algorithm

E. Auto-labeling algorithm

The proposed algorithm here is to label a dataset of features vectors with minimal human effort. Labeling shall be based on clustering the given data into coherent classes with similar values. To guarantee that the obtained clusters represent the real class labels, the data representation of the vectors must be designed to maintain the class neighborhood relation of the given vectors, and in the same time provide well-structured data so that efficient clustering becomes possible.

The algorithm pseudo code is presented in Figure 7. A deep auto-encoder is first trained like in the algorithm described in IV.D. The unlabeled dataset is the input of the algorithm. For each member of the unlabeled set, a code word is generated (see the upward path in Figure 4), and stored in a large array of all code words representing the unlabeled data set in the new mapped domain.

The obtained code words are fed to the K-means algorithm, with the number of clusters *K_classes* is just the number of classes' labels of the dataset. The K-means algorithm gives the clustered data and their means.

To obtain the labels of the new clusters with minimal supervision, only the clusters means' labels need to be knows, which represent *K_classes* examples of the whole dataset. Here user intervention is needed to label those means. However, the means are now represented in the form of code word, so the user does not know what they represent in the original dataset. For example, for number recognition, the user can only give a label of the number

image, but he cannot label its code word. So, the original data (number image) is re-generated (the downward path in Figure 4) and presented to the user to label it, which is done in the sub-routine *ask_user_for_label*. Once the means labels are obtained, the same label is applied to the whole dataset, and hence a labeled dataset is obtained.

```
Sub_routine_auto_label(unlabeled_dataset)
[W] = call Sub_routine_train_deep_auto(unlabeled_dataset)
 foreach data_example in unlabeled_dataset
     code = generate_code_word(data_example)
     codes = [codes; code]
end_foreach
[clusters, means] = call kmeans_clustering(codes, K_classes)
means_data = generate_code_word(means)
labels = call ask_user_for_label(means_data)
labeled_dataset = [unlabeled_dataset, labels]
return labeled_dataset
```

Figure 7 Auto-labeling algorithm

V. Adaptive learning system

Traditional pattern recognition model is composed of training and test phases. After training the model, the system goes into test phase or normal operation. While the system is in normal operation, new patterns keep flowing in to be classified. According to the generality of the training set, the empirical classification error is determined. So, if the new patterns were encountered in the training phase or similar ones, then they should be correctly classified. On the other hand, if the training set was not covering some part of the feature space, and a new pattern is encountered in that part during operation, it will be misclassified.

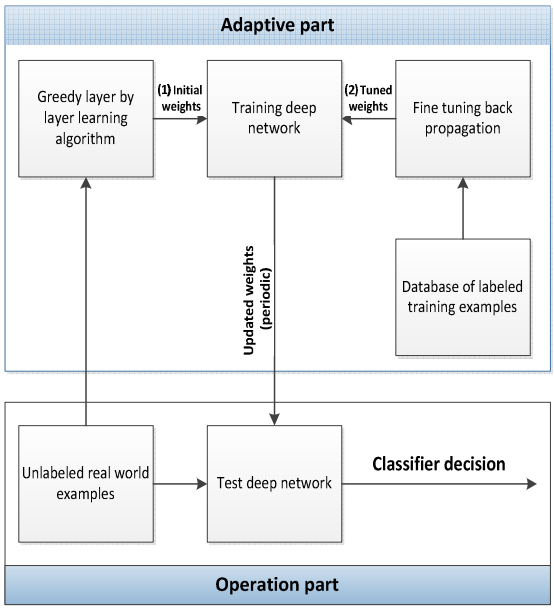

Figure 8 Adaptive learning system

A. Unsupervised examples improve learning

In conventional pattern recognition system the training and test phases are sequential. In adaptive system, they are done in parallel.

Figure 8 describes the adaptive learning system. To use real world patterns to update the model a learning algorithm that makes use of unsupervised examples is needed. Deep Belief Nets (DBN) trained using greedy layer wise learning algorithm [1] are typical candidates for such purpose.

The learning algorithm can make efficient use of very large sets of unlabeled data, and so the model can be pre-trained in completely unsupervised fashion. The very limited labeled data can then be used to only slightly fine-tune the model for a specific task at hand using standard gradient based optimization [3]. The unsupervised examples help in getting the initialization of weights to a point near to the minimum of the objective function to be optimized, protecting against getting stuck in poor local minima.

During operation of the system, the more unsupervised examples flowing in can be used to adjust the network to a better initial location. This process will be performed offline, in parallel with normal classification going on in the network, and the whole learning process is repeated to calculate the new weights. New updated weights calculated offline are then applied to the network periodically and only if test error is improved with the new weights, otherwise the old weights are kept.

The DBN architecture in [1] performs back propagation for weight adjustment after the unsupervised training phase. In adaptive learning, there are two options; the first is to apply the new unsupervised examples as separate unsupervised pre-training phase, then run back propagation, and the other one is to append the already existing database of old unsupervised examples with the new ones, then re-run the whole learning process from scratch. The choice depends on timing and storage issues.

B. Over-fitting or user adaptation

The above proposed system may be considered to suffer from over fitting to certain examples that flow during operation. For example if it is applied in handwriting recognition, then the system shall over fit to the user style that is using the system, because the network will be continuously adjusted to the examples of his own style. On the other hand, this could be viewed as user style adaptation. Both points of view could be desirable or not according to the application and the method of implementation.

C. Adaptation algorithm

The proposed algorithm for adapting the DBN network using unlabeled examples is presented in Figure 9. The presented subroutine is to adapt an existing DBN for a new batch of unlabeled data *new_unlabeled_batch_data*.

A stack of RBM's are first trained using the greedy layer-wise training algorithm in [1] using *pre_train_rbm sub-routine*. This is repeated for the *N_layers* of the network. The activation of each layer is denoted as *data(i)*, with *data(0)* is the original data example. $W_{N_i \times N_{i+1}}$ is the weights matrix linking layer i to layer $i+1$, of size $N_i \times N_{i+1}$. ΔW is the weights update signal resulting from training. Having all layers trained, the pre-training stage is over.

After pre-training, the back propagation fine tuning is performed on the stored labeled dataset to adjust the weights. Before applying the new weights, the error rate is tested. If improvement in the error rate is achieved, then the new weights are applied, otherwise the old weights are kept.

```
Sub_routine_adaptation_algorithm(new_unlabeled_batch_data)
data(0) := new_unlabeled_batch_data
for i := 1:N_layers
    [ΔW_{N_i×N_{i+1}}(i), data(i)] := pre_train_rbm(Layer(i))
end_for
Old_W = W
W = W + ΔW
[W] = call back_propagation_fine_tune(original_labeled_dataset, W)
test_err = call calculate_new_err_rate(original_labeled_dataset)
if (test_err < old_test_err)
    return(W)
else
    return(Old_W)
end_if
return(W)
```

Figure 9 Adaptation algorithm

D. Implementation

When it comes to practical implementation, there are two options to implement adaptive learning model; the first one is the on-board implementation, the other one is the distributed implementation.

1) On board implementation

In this scheme, both the adaptive learning module and the classifier modules are on the same system, operating on the same user inputs. In this case user style adaptation is desired, and hence over fitting is not an issue. This type is probable in hand-held devices and embedded systems. The choice could be constrained by the cost of the system and resources available. A separate parallel coprocessor shall be used to handle the adaptation separate from the ongoing classification task. Updates of the network weights should be scheduled periodically.

2) Distributed client-server model

In cases of multi-users system, over fitting to certain user style is not desirable. In such cases; it is more convenient to use distributed client-server model; where the client is the light weight device performing classification with the already calculated weights, under real time constraints, while the server part is performing adaptation using the classification examples. Multi-clients communicate their examples to the server, which performs adaptation, and then communicate back the updated weights to the clients. Updates could be scheduled periodically or on need basis.

To avoid over fitting, the server should perform adaptation by balancing the examples coming from all clients to be used in the learning process, and avoid being biased to certain user examples.

VI. Experimental results

The proposed adaptive algorithm was tested on number recognition task on MNIST dataset. On the other hand, Auto-labeling algorithm was tested on character recognition task on MNIST dataset, and also on E-mail classification task on Enron dataset. In the following sections results on both tasks are presented.

A. MNIST Dataset

MNIST dataset of handwritten digits was used [19]. This dataset has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. The input images have sizes of 28x28 pixels. The images with their labels are stored in a certain file format. Data are stored in big endian form, with MSB first. Pixels are organized low wise. Pixel values range is 0..255.

B. Enron Dataset

Enron dataset was collected and prepared by the CALO Project (A Cognitive Assistant that Learns and Organizes). It contains data from about 150 users, mostly senior management of Enron, organized into folders. This data was originally made public, and posted to the web, by the Federal Energy Regulatory Commission (FERC) during its investigation. The email dataset was later purchased MIT, and turned out to have a number of integrity problems. The dataset was further processed by SRI (see [15] and [16]).

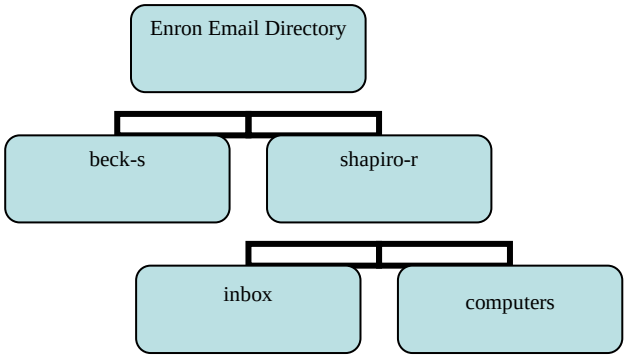

Figure 10 Enron email directory structure

Figure 10 shows the directory structure of the dataset. Each user has a folder, containing a folder representing each mail category. Categories could further split into sub-folder representing sub-categories. Some of these folders are irrelevant for classification task, like “sent”, “inbox”, “deleted”…etc… Pre-processing is needed to extract useful categories. Also, preprocessing is needed to extract useful vocabulary, build features vectors…etc. Training and test sets are built by randomly splitting the processed dataset into training and testing examples. This random splitting guarantees independence between test and training sets.

1) Users selection

E-mails of the Enron employees are diverse, so that not limited number of common categories can be easily identified among all users. For example, for a certain user, the corpus is divided into: “eol”, “ces”, “entex”, “industial”…etc. While for another user his mails are categorized into: “duke”, “ecogas”, “bastos”…etc. This creates a difficulty assigning labels to each e-mail based on

its category, since categories are different between users. To overcome this problem, users are ordered discerningly in terms of the number of e-mails in their directories, and then the top users with largest number of e-mails were selected as the input datasets, so that, each user directory represents a separate dataset. This ensures coherency between each category e-mail examples.

2) Vocabulary building

E-mail classification is based on categorizing a features vector $x^{(i)}$ of the i^{th} example into one of $N_{targets}$ categories. The features themselves are just indicators of the existence of certain, most-encountered words in the dataset, called the vocabulary of the corpus, or the bag-of-words. The size L of $x^{(i)}$ is the number of vocabulary words. Since, each user mails are considered a separate dataset; hence, each user is assigned a separate vocabulary vector V.

To build this vocabulary vector, the whole corpus of each user is parsed, and the words are ordered in a descending order in terms of the frequency they are encountered in the dataset, and then the most L frequently encountered words are selected as the members of *V*. The parameter L is chosen based on experimental results, where L is chosen to give the best accuracy. During vocabulary building, irrelevant words are ignored (“he”, “she”, “when”…etc). Also, e-mail header is excluded, which contains the actual class label.

3) Categories building

Categories of email messages are simply the different class targets of the classification problem at hand. Each user has own set of categories. Number of categories is denoted by $N_{targets}$, which is the dimension of class labels $y^{(i)}$ of the i^{th} example.

Selection of categories per each user directory is done first by counting the number of emails in each category. This counting is done recursively, i.e. if the category contains sub-categories, then messages in the sub-folders are also counted. Then the categories are ordered discerningly, and the highest score ones are selected as the targets labels $N_{targets}$. Each target of $N_{targets}$ is assigned a binary code of $N_{targets}$ bits, with only one bit set to 1 and the others set to 0.

Each category is assigned an integer number from 0 to $N_{targets}-1$ which is denoted by label. The target label $y^{(i)}$ of the i^{th} email is a binary vector of length $N_{targets}$, with only 1 set at 2^{label} position. $N_{targets}$ is chosen based on experimental results so as to give the best accuracy.

During the aforementioned process, the irrelevant categories are dropped, since they have no relevance to the classification problem, like “inbox”, “sent”, “deleted”…etc.

4) Features extraction

The features vector representing the i^{th} email $x^{(i)}$ could be one of two cases; either binary or word-count. For both cases, the vocabulary of the bag-of-words V is considered, and $x^{(i)}$ is an L size vector.

For the binary case, the features vector $N_{targets}$ is just a vector of 1’s or 0’s. The “1” indicates the existence of the corresponding vocabulary word in the e-mail, while “0” marks its absence. For the word-count case, values in $x^{(i)}$ are integers, marking the frequency of word repetition within the given email. In the proposed classifier, binary features were tested to give the good results, while word-counts give poor results, and hence binary features shall be considered.

5) Training and test sets selection

For each user, the features and labels are extracted as described in the above sections. Now, to split the processed dataset into training and testing sets, a complete random approach was followed, such that training and testing emails were selected randomly from the final dataset. This ensures independence between training and testing datasets.

6) Processed Enron Dataset

Enron dataset pre-processing generates different dataset for each user. The different parameters are:

- Training set size
- Testing set size
- Number of categories/Class labels
- Number of features

User	*Training set size (e-mails)*	*Test set size (e-mails)*	*Number of categories*	*Number of features*
arnold-j	90	10	10	100
baugmann	952	106	5	1000
beck-s	891	100	10	2000
blair-l	1123	15	16	1000
cash-m	216	23	6	1000
griffith-j	352	64	8	1000
haedicke-m	60	31	2	1000
hayslett-r	256	658	4	2000
kaminski-v	1791	55	10	1000
kean-s	1146	231	4	10000
ruscitti-k	92	79	3	1000
shackleton-s	490	168	4	1000
shapiro-r	490	118	5	10000
steffes-j	503	95	7	1000
ward-k	283	95	8	1000
farmer-d	2589	665	11	1000
kitchen-l	1992	864	10	1000
lokay-m	2073	61	6	1000
sanders-r	711	281	6	1000
williams-w3	1974	26	5	1000
campbell-l	14	14	6	1000

Table 1 Users datasets details

Table 1 shows the details of each user’s dataset after pre-processing.

C. Auto-labeling results

The auto labeling algorithm described in IV.E is applied to MNIST dataset. The training dataset is composed of 60,000 labeled examples. The label is removed to test automatic labeling performance, and so 60,000 unlabeled example results. The deep auto-encoder

architecture used has 1000 neurons in the first layer, 500 neurons in the second layer, 250 neurons in the third layer and 30 neurons in the last layer. Hence the code word dimension is 30.

Figure 11 and Figure 12 show the resulting 2-D and 3-D codes of the unlabeled MNIST training dataset, which represent 60,000 examples. For Figure 11 and Figure 12 the last layer width is 2 and 3 respectively. The classes (digits) are represented by different colors in the figures. Some classes are well clustered in 2-D and 3-D codes; however, others are not, which indicates the need for higher dimensional codes. 2-D and 3-D codes were used for visualization purpose as it is not possible to visualize higher dimensional codes.

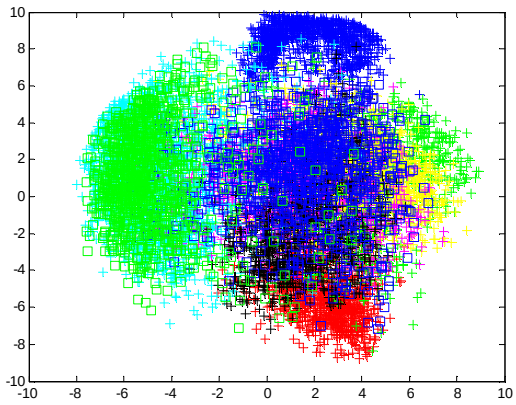

Figure 11 2-D codes of MNIST dataset

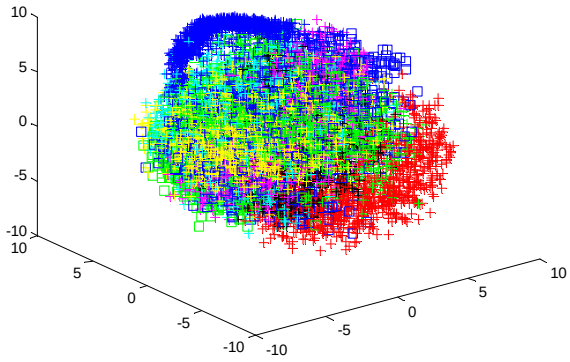

Figure 12 3-D codes of MNIST dataset

The experiment starts by running the automatic labeling algorithm described in IV.E. The input to the algorithm is the unlabeled 60,000 examples of MNIST dataset, and the output is the automatically labeled set. Then to test the algorithm performance, the automatically labeled dataset is used to pre-train and fine tune a DBN classification network as in [1] and VI.D. The architecture used for classification is 3-layered 1000-500-2000 with 10 targets neurons. The unlabeled set is used first to pre-train an RBM of the given architecture. Then the obtained weights are used to initialize the network for back propagation fine tuning using the automatically labeled dataset.

$$
\begin{aligned}
&[labeled\_dataset] = call \quad Sub\_routine\_auto\_label(unlabeled\_dataset)\\
&[W\_\Pr etrain] = call \quad r\text{bm\_pre\_train}(labeled\_dataset)\\
&[W] = call \quad back\_propagation\_fine\_tune(labeled\_dataset)\\
&err = test\_err\_rate(labeled\_dataset, W)
\end{aligned}
$$

Figure 13 Automatic labels experiment

The obtained results of error rate of misclassification are obtained on the test dataset of MNIST of 10,000 examples, completely independent of the training ones. The results are compared to results on the original labeled dataset, trained using the same fast algorithm in [1]. The error rate is nearly the same the original labeled dataset. For original MNIST training set, the error is 0.98%, while for automatically labeled one it is 1.04% as shown in Figure 14.

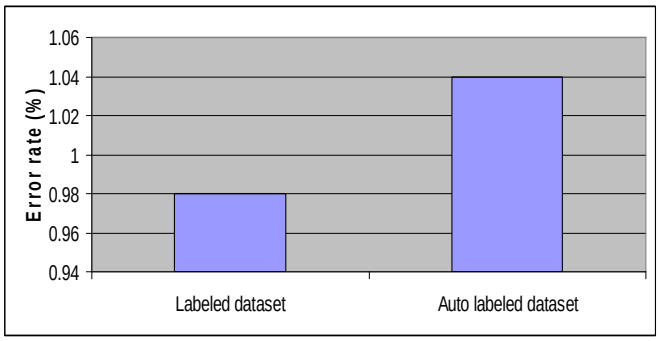

Figure 14 Automatic labeled dataset results

Also the auto-label algorithm was tested on Enron dataset for different users. The results are shown in Figure 15. The average accuracy of using DBN classifier alone against using auto-label algorithm on top of DBN classifier is shown in Table 2. The accuracy of auto-label algorithm is nearly the same as that of DBN classifier alone.

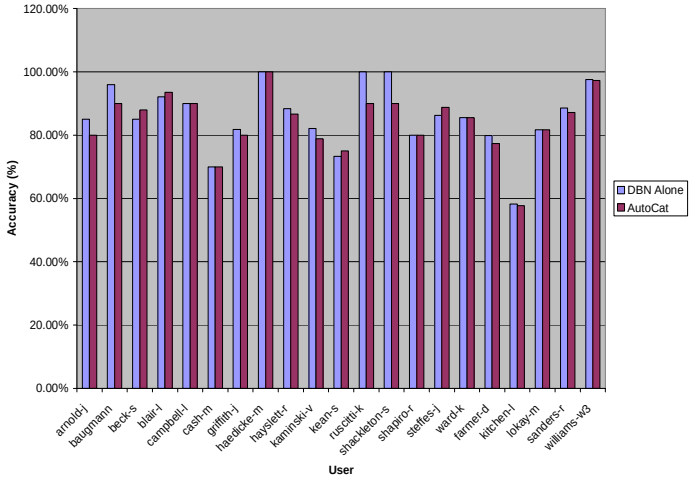

Figure 15 Auto-label algorithm vs. DBN Classifier with manually categorized dataset

Method	**Average Accuracy (%)**
DBN	85.78%
Auto-label	84.17%

Table 2 Average Auto-label accuracy vs. DBN accuracy with manually categorized dataset

D. Adaptation results

Adaptive learning is tested on MNIST dataset [19] using modified version of the MATLAB code in [18]. The DBN architecture used was 3 layer network; with 500 neurons in the first layer, 500 neurons in the second layer and 1000 neurons in the third layer. A layer of 10 units is added on the top layer and tuned to give the labels of characters. The number of iterations for greedy RBM training or back propagation fine tuning was 50 epochs.

To simulate adaptation, the training set was subdivided into balanced mini-batches each containing 100 examples with total of 600 batches. The experiment goes on by feeding more batches and performing back propagation each time. With every update of the network the classification error performance is tested.

Note that; the target was not to achieve best error rate in comparison to existing systems, but to prove that feeding more unsupervised examples do improve network performance in terms of classification error rate.

The experiment starts by sub-dividing the dataset of MNIST into number of batches. For each batch the adaptation algorithm is run and the error rate is tested.

```
sub_batches = sub_divide_dataset(unlabeled_dataset, N_batches)
foreach batch in sub_batch
    [W] = call Sub_routine_adaptation_algorithm(batch)
    [W] = call back_propagation_fine_tune(labeled_dataset)
    err(batch) = test_err_rate(labeled_dataset, W)
end_foreach
```

Figure 16 Adaptation experiment pseudo code

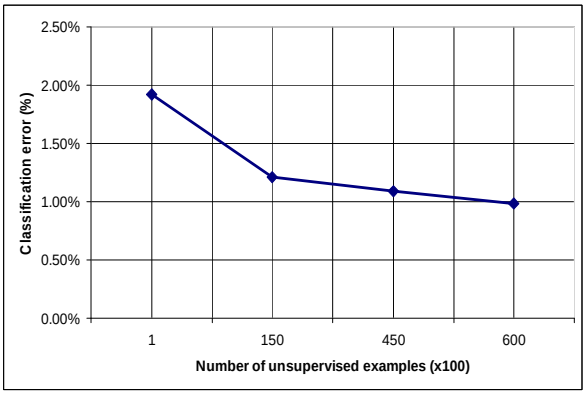

Figure 17 Adaptive learning performance curve

Figure 17 shows that the error performance is improved as more unsupervised examples are fed to the network. As more examples are used to pre-train the DBM network, a better initialization point of weights is obtained in weight space, so that supervised training phase using back propagation can start from a point nearer to the global minimum, instead of falling in a poor local minimum. In addition, as the greedy algorithm is trained on more unsupervised real world examples, its probabilistic model is improved to make those examples more probable, and hence improving the regeneration performance of those examples. This is similar to teaching a human person some figures, so that he can generate similar figures from his imagination, which indicates a better learning of those figures.

VII. Conclusion

In this paper a self-learning machine is proposed in terms of its ability to learn with minimal supervision and adapt to real world examples during operation. The first contribution of this paper is the automatic labeling component based on non-linear transformation using deep auto-encoders, followed by clustering step. The second contribution is the adaptive learning component based on unsupervised pre-training of Deep Belief Nets. Merging the two components constitutes a Self-Learning Machine (SLM). Those machines are able to learn with minimal supervision and adapt to real world data. SLMs are generic learning systems in terms that they can be adapted to any pattern recognition task. All what is needed are few representative examples of each class to be recognized, and then the system will build the network from the unsupervised input data flowing in. Recursively the network adapts its parameters as more and more new data is fed to the system, and the performance is enhanced automatically. The proposed system target is to mimic human learning behavior that it needs few supervision and the ability to build own beliefs based on experience.

Automatic labeling was tested on MNIST and Enron datasets. Results show that the error rate obtained using originally labeled dataset is nearly the same as the automatically labeled one.

Practical results on MNIST dataset prove the adaptive learning concept, showing improved classification error performance as more unsupervised examples are used for pre-training. This is due to better initialization weights for back fitting supervised stage, thus, improving the generation performance of the network of real world examples. This is similar to the ability of human being of drawing familiar figures that are well learned.

Future work includes testing the proposed auto labeling algorithm and adaptive learning on different datasets other than MNIST. Also, different implementation issues described in adaptive learning are to be addressed. The on board implementation choice could be studied to evaluate practically the parallel co-processor hardware needed to perform adaptation. Also, the communication method needed in case of distributed implementation needs to be studied. Over fitting avoidance strategy in case of adaptive distributed implementation need to be addressed too.

References

[1] G. E. Hinton, S. Osindero, and Y. Teh, “A fast learning algorithm for deep belief nets” *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[2] G. E. Hinton, “To Recognize Shapes, First Learn to Generate Images” *Computational Neuroscience: Theoretical Insights into Brain Function, Elsevier*, UTML TR 2006 – 004, October 26, 2006

[3] Ruslan Salakhutdinov, “Learning Deep Generative Models” PhD thesis, *Graduate Department of Computer Science, University of Toronto*, 2009

[4] G. Montavon, M. Braun, K. R. Müller, “Layer-wise Analysis of Deep Networks with Gaussian Kernels” *Advances in Neural Information Processing Systems (NIPS)*, 2010

[5] Yoshua Bengio, and Yann Lecun, “Scaling Learning Algorithms towards AI”, *Large-Scale Kernel Machines, MIT Press* (2007)

[6] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. *In Proceedings of NIPS'2006*. pp.153~160
[7] Arel, I., Rose, D.C., Karnowski, T.P., "Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]", *Computational Intelligence Magazine, IEEE*, volume: 5, issue:4, pp: 13 – 18, Nov 2010
[8] Olshausen BA, Field DJ. "How close are we to understanding v1?" Neural Comput. 2005 Aug;17(8):1665-99.
[9] Matthew Blaschko, Andrea Vedaldi, Andrew Zisserman, "Simultaneous Object Detection and Ranking with Weak Supervision", *Advances in Neural Information Processing Systems (NIPS)*, 2010
[10] George E. Dahl, Marc'Aurelio Ranzato, Abdel-rahman Mohamed, and Geoffrey E. Hinton, "Phone Recognition with the Mean-Covariance Restricted Boltzmann Machine", Advances in Neural Information Processing Systems (NIPS), 2010
[11] Li Deng, Mike Seltzer, Dong Yu, Alex Acero, Abdel-rahman Mohamed, and Geoff Hinton , "Binary Coding of Speech Spectrograms Using a Deep Auto-encoder", Interspeech 2010
[12] Yuanqing Lin, Tong Zhang, Shenghuo Zhu, Kai Yu, "Deep Coding Network", Advances in Neural Information Processing Systems (NIPS), 2010
[13] Ruslan Salakhutdinov, Geoffrey E. Hinton "An Efficient Learning Procedure for Deep Boltzmann Machine", Computational Cognitive Science, MIT Press (2010)
[14] Marc' A. Ranzato, Christopher Poultney, Sumit Chopra, Yann Lecun , "Efficient Learning of Sparse Representations with an Energy-Based Model", Advances in Neural Information Processing Systems (NIPS), 2006
[15] Bryan Klimt and Yiming Yang, "The Enron Corpus: A New Dataset for Email Classication Research", Language Technologies Institute, Carnegie Mellon University, CEAS conference, 2004.
[16] Ron Bekkerman, Andrew Mccallum, G. Huang, "Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora", Citeseer, vol 418, p 1-23 , 2004
[17] http://www.cs.cmu.edu/~enron/, 1/11/2012
[18] http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html, 5/24/2011 2:31 PM
[19] http://yann.lecun.com/exdb/mnist/, 5/24/2011 2:31 PM

Author Biographies

Ahmad A. Sallab has acquired his B. Sc. and M.Sc. in Communications and Electronics from the Faculty on Engineering, Cairo University in 2005 and 2009. Currently he is a Ph.D. candidate at Cairo University. He works as a Software Leader at Valeo, Egypt.

Mohsen A. Rashwan has acquired his Ph.D. from Queen's University, Canada. He is a Professor of Electronics and Communications, Faculty of Engineering, Cairo University. He is currently the CEO of RDI Corporation.