2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Least Confidence [1]
Margin Sampling [1]
Entropy Sampling [1]

A New Active Labeling Method for Deep Learning

Dan Wang and Yi Shang

***Abstract*—Deep learning has been shown to achieve outstanding performance in a number of challenging real-world applications. However, most of the existing works assume a fixed set of labeled data, which is not necessarily true in real-world applications. Getting labeled data is usually expensive and time consuming. Active labelling in deep learning aims at achieving the best learning result with a limited labeled data set, i.e., choosing the most appropriate unlabeled data to get labeled. This paper presents a new active labeling method, *AL-DL,* for cost-effective selection of data to be labeled. *AL-DL* uses one of three metrics for data selection: least confidence, margin sampling, and entropy. The method is applied to deep learning networks based on stacked restricted Boltzmann machines, as well as stacked autoencoders. In experiments on the MNIST benchmark dataset, the method outperforms random labeling consistently by a significant margin.**

I. Introduction

SHALLOW neural networks (NNs) with an input layer, a single hidden layer, and an output layer require more computational elements or are hard to model complex concepts and multi-level abstractions. In contrast, multi-layer neural networks provide better representational power and could derive more descriptive multi-level models due to their hierarchical structures, with each higher layer representing higher-level abstraction of the input data. Unfortunately, it is difficult to train all layers of a deep neutral network at once [1]. With random initial weights, the learning is likely to get stuck in local minima.

In 2006, Hinton et. al. introduced Deep Belief Networks (DBNs) to efficiently train multi-layer neural networks to learn features from unlabeled data [2]. A DBN trains a multi-layer neural network in a greedy fashion, each layer being a restricted Boltzmann machine [3]. The trained weights and biases in each layer can be thought of as the features or filters learned from the input data. Then the weights and biases act as the initial values for the supervised fine-tuning using backpropogation. In short, a DBN discovers features on its own and does semi-supervised learning by modeling unlabeled data first in an unsupervised way and then incorporates labels in a supervised fashion.

Similar to stacked RBMs, stacked autoencoders also perform deep learning in a layer-by-layer greedy fashion [1]. Each autoencoder is a one-hidden-layer neural network with the same number of nodes in the input and output layers, which tries to learn an approximation of the identity function by backpropagation learning. The hidden layer with fewer units (or with a sparsity constraint) is a compressed representation of the input data, thus discovering high-level features about the data. An autoencoder serves as a building block for deep neural networks similar to an RBM. The supervised fine-tuning stage could also be applied to stacked autoencoders by adding a softmax classifier layer.

A practical problem in deep learning is how to choose samples to be labeled. As an example, there may be many candidate drugs for a particular disease and testing each drug on a large set of subjects is an expensive and time-consuming process. Since getting labeled data can be expensive and difficult, it is desirable to choose the most informative samples to be labeled and then deep learning fine-tunes the classification models using these labeled data. Existing research in deep learning assumes labeled data are passive, either available or randomly selected to be labeled by human experts. Active labeling in deep learning aims at achieving the best learning result with a limited labeled data set, i.e., choosing the most appropriate unlabeled data to get labeled. Given a budget on the number of samples to be labeled, the objective is to learn the best classifier by selecting a subset of samples to be labeled. This falls into the well-defined active learning framework [4]. However, how to assess which samples are more informative remains unexplored in deep learning.

To the best of our knowledge, this work is the first to apply active learning in deep learning. A new active labeling method, called *AL-DL,* is proposed for cost-effective selection of data to be labeled for deep learning. *AL-DL* uses one of three metrics for data selection: least confidence, margin sampling, and entropy. The method is applied to deep learning networks based on stacked restricted Boltzmann machines, as well as stacked autoencoders. In experiments on the MNIST dataset and a sleep stage dataset, the performances of the active labeling method of using the three metrics are compared with a random labeling strategy.

This paper is organized as follows. Section II reviews the basics of deep learning networks, including stacked RBMs and stacked autoencoders. In Section III, a new active labeling method, *AL-DL,* and three selection criteria are presented. Section IV presents experimental results on the widely used MNIST dataset. Section V presents experimental results on a sleep stage-recognition dataset. Finally, Section VI concludes the paper.

II. Basics of deep Learning networks

Deep belief networks (DBNs) consist of multiple layers of restricted Boltzmann machines (RBMs) [3]. Fig.1 shows an example with three hidden layers. A DBN is trained

Dan Wang and Yi Shang are with the Department of Computer Science, University of Missouri, Columbia, MO 65211 USA (e-mail: dwdy8@mail.missouri.edu, shangy@missouri.edu).

This work was supported in part by NIH under Grant R01-GM100701.

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

layer-by-layer in a greedy fashion [2, 5] in two stages: pre-training and fine-tuning. In pre-training, no labels are involved and the training is done in an unsupervised way. As shown in Fig. 1 (a), the training starts off with the bottom two layers to obtain features in hidden layer 1 from the input layer. Then the training moves up to hidden layer 1 and layer 2, treating layer 1 as the new input to get its features in layer 2. The greedy layer-wise training is performed until reaching the highest hidden layer. The pre-training stage trains a generative model as weights between layers capturing input's features, resulting in better starting point for the fine-tuning stage than randomly assigned initial weights. In fine-tuning, a new classifier, e.g. a softmax, layer is added on top of the stacked RBMs to construct a discriminative model.

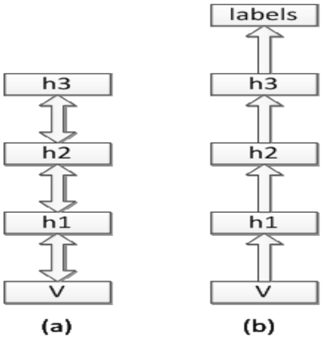

Fig. 1. The two stages of training a DBN with three hidden layers: (a) The pre-training stage with unlabeled data and (b) the fine-tuning stage with labeled data.

Similar to DBNs (i.e., stacked RBMs), stacked autoencoders is another type of deep learning networks [1]. They use deterministic feedforward neural networks, instead of stochastic NNs, as building blocks.

A. Restricted Boltzmann Machines as Building Blocks

As the building block of DBNs, a restricted Boltzmann machine has a visible layer consisting of stochastic, binary nodes as the input and a hidden output layer consisting of stochastic, binary feature detectors as the output, connected by symmetrical weights between nodes in different layers. There is no connection between the nodes in the same layer. A graphical depiction of an RBM is shown in Fig. 2.

An RBM is a generative stochastic neural network that can learn a probability distribution over its set of inputs. A joint configuration (v, h) of the visible nodes v and hidden nodes h can be represented by the following energy function

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij} - \sum_i b_i v_i - \sum_j b_j h_j \qquad (1)$$

where v_i is the binary state of visible node i, h_j is the binary state of hidden node j, w_{ij} is the weight between node i and j, and b_i is the bias term of visible node i and b_j the bias term of hidden node j.

The probability of a given configuration is the normalized energy function:

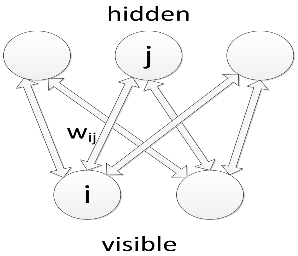

Fig. 2. A graphical depiction of an RBM.

$$p(v,h) = \frac{e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}} \qquad (2)$$

The binary nodes of the hidden layer are Bernoulli random variables. The probability that node h_j is activated, given visible layer v, is

$$P(h_j = 1 \mid v) = \sigma(b_j + \sum_i w_{ij} v_i) \qquad (3)$$

$$where\ \sigma(x) = \frac{1}{1+e^{-x}} \qquad (4)$$

The probability that node v_i is activated, given hidden layer h, can be calculated in a similar way as follows

$$P(v_i = 1 \mid h) = \sigma(b_i + \sum_j w_{ij} h_j) \qquad (5)$$

Restricted Boltzmann machines are trained to maximize the product of probabilities of a set of training examples *X*:

$$\underset{W}{\operatorname{argmax}} \prod_{x \in X} P(x) \qquad (6)$$

or equivalently to maximize the log likelihood

$$\underset{W}{\operatorname{argmax}} \sum_{x \in X} \log P(x) \qquad (7)$$

It is intractable to compute the gradient of the log likelihood. Therefore, [6] proposed contrastive divergence by doing k iterations of Gibbs sampling to approximate it:

$$\Delta w_{ij} = \epsilon(< v_i h_j >^0 - < v_i h_j >^k) \qquad (8)$$

where $<\cdot>^m$ is the average in a contrastive divergence iteration *m* and ϵ is the learning rate.

The above-mentioned rule works, but a few tricks are used to accelerate the learning process and/or prevent overfitting. Three commonly used techniques are minibatch, momentum, and weight decay [7].

Minibatch is a variation of Eq. (8), in which w_{ij} is updated by taking the average over a small batch instead of a single training vector. This produces two benefits: a) a less noisy estimate of the gradient since outliers does not impact as much and b) allowing a matrix by matrix product instead of a vector by matrix product, which can be taken advantage by modern GPUs or Matlab to speed up the computation. However, it is a bad idea to make the minibatch size too large because the number of updates will decrease accordingly, eventually resulting in inefficiency.

Momentum is used to speed up learning by simulating a ball moving on a surface. It is an analogy to the acceleration as if w_{ij} were the distance and Δw_{ij} were the velocity. Instead of using the estimated gradient to change weights directly as shown in Eq. (8) , the momentum method changes the velocity of weight-change as follows:

$$\Delta\theta_i(k) = v_i(k) = \alpha v_i(k-1) - \epsilon \frac{dE}{d\theta_i}(k) \qquad (9)$$

where α is a hyper-parameter to control the weight given to the previous velocity.

Weight decay is a standard L2 regularization to prevent the weights from getting too large. The updated rule is changed to

$$\begin{aligned}\Delta\theta_i(k) = v_i(k) = \alpha v_i(k-1) \\ - \epsilon(\frac{dE}{d\theta_i}(k) + \lambda\theta_i(k))\end{aligned} \qquad (10)$$

where λ is the weight cost which controls how much penalty should be applied to weight $\theta_i(k)$.

B. Autoencoders as Building Blocks

An autoencoder [8-10] is a feedforward NN that attempts to learn an identity function by setting the outputs equal to the inputs (or at least minimizing the reconstruction error), Fig. 3 shows an example. By placing some restrictions on the network to make it as a regularized autoencoder, we can learn compressed representation of the input data. The easiest way to do so is limiting the number of nodes in the hidden layer to force fewer nodes to represent features. The discovered low-dimensional representations are similar to PCA's. In this sense, the mapping from the input layer to the hidden layer is called encoding and the mapping from the hidden layer to the output layer is called decoding. In summary, a basic autoencoder tries to find the weights:

$$\underset{W,b}{\text{argmin}} J(W,b) = ||\, h_{W,b}(x) - x \,|| \qquad (11)$$

where x is the input, W is the weights, b is the biases, and h is the function mapping input to output.

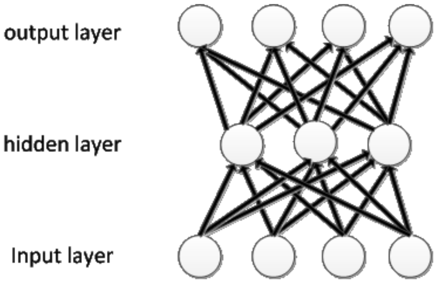

Fig. 3. An autoencoder example

When the number of hidden nodes is large, sparsity regularization can be applied on the hidden nodes to force them to learn compressed representations. Specifically, let

$$\hat{p}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(2)}(x^{(i)})\right] \qquad (12)$$

be the average activation of hidden unit j over the training set of size m. The objective is to approximate the sparsity parameter p to $\hat{p}$. An extra penalty term can be added to Eq. (11) to measure the difference between p and $\hat{p}$:

$$R = \sum_{j=1}^{s_2} p log \frac{p}{\hat{p}_j} + (1-p) log \frac{1-p}{1-\hat{p}_j} \qquad (13)$$

where j is a hidden node, s_2 is the number of nodes in the hidden layer. The value reaches its minimum of 0 at $\hat{p}_j = p$ and goes to infinity as $\hat{p}_j$approaches 0 or 1. The overall cost function becomes

$$J_{sparse}(W,b) = J(W,b) + \beta R \qquad (14)$$

where β is a weight.

Backpropagation can be used to update W and b and the derivation is similar to that of an RBM.

C. Unsupervised Learning Stage

A single RBM or autoencoder may not be enough to model sufficient features in input data. Iteratively we could build another layer on top of a trained RBM or autoencoder by treating the learned feature detectors in the hidden layer as visible input layer for the new layer, as shown in Fig. 1(a). The unsupervised learning stage has no labels involved and solely relies on unlabeled data. The learned weights and biases will be used as the starting point for the fine-tuning supervised learning stage.

D. Supervised Learning Stage

The supervised learning stage adds a label layer as the highest layer and removes the links in the top to down direction [2], as shown in Fig. 1(b). The standard backpropogation is conducted as suggested by [11] instead of the original contrastive wake-sleep algorithm for supervised learning in [2]. The goal is to minimize classification errors given labeled samples. The weights and biases are initialized as the values learned in the unsupervised learning stage, except for those between the highest two layers, which are randomly initialized.

When the newly added top layer has multiple output units the probabilities of turning the units on (p_j) must add up to 1, which is the softmax function:

$$p_j = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}} \qquad (15)$$

where $x_j = P\left(h_j = 1 \,\middle|\, v\right)$.

III. *AL-DL,* A NEW ALGORITHM FOR ACTIVE LABELING IN DEEP LEARNING

In real world applications, unlabeled data are relatively easy to get at low cost, but labeled data are usually expensive and time consuming to get. Due to limited resources, only very few labeled data can be obtained given a certain budget. For example, in a classification problem on physiological data from biosensors, the unlabeled data can be obtained by simply asking subjects to wear sensors day and night, but labeled data may not be available until human experts manually make annotations on selected unlabeled samples. Therefore, to make the best use of the budget for a discriminative learning task, it would be useful to propose an algorithm to carefully choose unlabeled samples to be labeled.

Active learning (AL) [4] select appropriate unlabeled samples to be labeled by an oracle (e.g., a human annotator). The goal is to achieve high classification accuracy using as few labeled samples as possible. An active learning method selects the samples in the unlabeled pool that are most ambiguous or uncertain for the current model. Then, the newly labeled samples are added to the labeled pool and used to retrain the model. The two processes form a cycle as shown in Fig. 4.

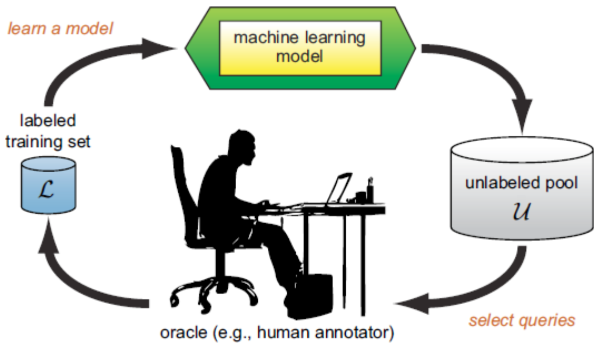

Fig. 4. A typical active learning cycle [4].

To the best of our knowledge, there are no active learning algorithms applied in DBNs except for Active Deep Networks (ADNs) proposed by [12]. ADNs follow previous work on active learning for SVMs by defining the uncertainty of an unlabeled sample as its distance from the separating hyperplane and it only works on binary classification problems.

In this work, we propose a method to effectively use the built-in classification uncertainty in deep learning networks to select unlabeled samples to be labeled. Since the top softmax layer of either stacked RBMs or stacked autoencoders outputs the probabilities of class labels, we use the probabilities as indicators of classification uncertainty, which forms the foundation of sample selection.

A. AL-DL Algorithm

The problem of active labeling in deep learning is formulated as follows. Given an unlabeled sample set X^U and a labeled sample set X^L, the algorithm needs to take k samples from X^U, have them labeled, and add them to X^L, in order to minimize the classification error of a deep learning model fine-tuned by X^L, where k is a constant.

The new *AL-DL* algorithm is depicted in Fig. 5. Its basic idea is to greedily select the samples that are most difficult to classify by the current deep learning network. Three heuristic metrics are used to measure difficulty, uncertainty, or ambiguity: least confidence (LC), margin sampling (MS), and entropy.

Given an unlabeled sample set X^U and a labeled sample set X^L, *AL-DL* first use all samples (not using labels) to perform unsupervised learning (pre-training) to construct a deep learning network of stacked RBMs or autoencoders. Then, labeled samples are used to training the top softmax classifier, followed by the fine-tuning of the overall deep network. Finally, unlabeled samples are fed into the lowest input layer and the top layer, layer N, uses Eq. (15) to softmax the activations in the top layer. Class j is predicted as the class of an unlabeled sample x_i if the neuron for Class j has the largest probability among all the units $p\left(h_j^N|x_i\right)$ as follows:

$$y_j = \begin{cases} 1 & if\ j = \underset{j}{\operatorname{argmax}}\left(p\left(h_j^N|x_i\right)\right) \\ 0 & otherwise \end{cases} \tag{16}$$

Algorithm: AL-DL

Input: an unlabeled sample set X^U, a labeled sample set X^L, and parameter *k*.

1. Pre-training (unsupervised): Use all samples in X^U and X^L to train a deep learning network (stacked RBMs or stacked autoencoders) layer-by-layer. The weights and biases will be used in Step 3 as the initial values for fine-tuning the deep learning network.
2. Use X^L to train the last layer, the softmax classifier layer, of the deep learning network.
3. Use X^L and run backpropagation to fine-tune all layers of the deep learning network.
4. Use the complete deep learning network to classify all unlabeled samples in X^U. Select k samples from X^U based on one of the following metrics:
    a. least confidence (LC) defined in Eq. (17)
    b. margin sampling (MS) defined in Eq. (18)
    c. entropy defined in Eq. (19)

Output: *k* unlabeled samples to be labeled.

Fig. 5. *AL-DL*, a new active labeling method for deep learning.

Since the true class of sample x_i is unknown, the predicted label y_j is not of much interest for our purpose. Instead, we use the probabilities, $p\left(h_j^N|x_i\right)$, to measure the confidence of the prediction and use three heuristics: Least confidence (LC), margin sampling (MS) [13], and entropy [14] to pick the most uncertain samples x_i.

(a) Active labeling with least confidence (*AL-LC*): pick the sample with the smallest of the maximum activations as follows:

$$x_i^{LC} = \underset{x_i}{\operatorname{argmin}} \max_j(p\left(h_j^N|x_i\right)) \tag{17}$$

where x_i is an input vector, h_j^N is the activation of the unit j in the top layer, layer *N*. To select more than one sample, the process is performed in a greedy fashion, sample-by-sample. Its basic idea is that if the probability of the most probable label for a sample is low then the classification of the sample is uncertain. It doesn't consider the label probability distributions of other class labels.

(b) Active labeling with margin sampling (*AL-MS*): pick the sample with the smallest separation of the top two class predictions:

$$x_i^{MS} = \underset{x_i}{\operatorname{argmin}}(p(y_1|x_i) - p(y_2|x_i)) \tag{18}$$

where y_1 and y_2 are the first and second most probable class labels predicted by the deep learning network. Intuitively, if the probabilities of predicting a sample to its most likely class and to its second most likely class are too close, the classifier is uncertain about the sample. Therefore some information is needed from the oracle to help the classifier discriminate these two classes.

(c) Active labeling with entropy (*AL-Entropy*): pick the sample with the largest class prediction information entropy. This metric takes all class label probabilities into consideration to measure uncertainty:

$$x_i^{Entropy} = \underset{x_i}{\operatorname{argmax}} - \sum_j p\left(h_j^N|x_i\right) logp\left(h_j^N|x_i\right) \tag{19}$$

IV. Experimental Results on MNIST Dataset

A. MNIST Dataset

The MNIST handwritten digits dataset [15] has 70,000 samples in total, conventionally divided into a training set of 50,000 samples, a validation set of 10,000 samples, and a test set of 10,000 samples. The digits have been size-normalized and centered in a 28 x 28 pixel image. The classes are 0 through 9. The MNIST dataset has been broadly used to evaluate the performance of machine learning algorithms.

B. Dimensionality Reduction by PCA and Whitening

Principal component analysis (PCA) is a popular data pre-processing and linear dimensionality reduction technique. It can be used to find a lower-dimensional subspace representation onto which the original data is projected. A dataset is first normalized to have zero mean. Then its covariance matrix is computed and decomposed by singular value decomposition to get eigenvectors and eigenvalues. A subset of eigenvectors with the largest eigenvalues forms a lower-dimensional subspace representation of the original data. The percentage of variance retained in a lower-dimensional representation represents how much information in the original data is preserved.

Whitening is a technique to make features less correlated and have the same variance. The detailed PCA and whitening implementation can be found in [16].

C. Stacked RBMs on Raw Data

In this set of experiments, the performances of *AL-DL* using three different heuristic metrics are compared with random labeling on the MNIST dataset. The experiment uses 50,000 training examples and 10,000 testing samples. The DBN tested has three hidden layers, 500-500-500, i.e., 500 hidden units in each of the three layers. Other parameters for unsupervised pre-training and supervised fine-tuning are listed in Table 1.

Table 1. DBN learning parameters used in the experiments.

Unsupervised pre-training stage	
learning rate ϵ	0.05
number of epochs	100
minibatch size	100
momentum α	0.5 for the first 5 epochs, 0.9 thereafter
weight cost λ	0.0002
Supervised fine-tuning stage	
learning rate ϵ	0.05
number of epochs	50
minibatch size	100

The experiments start with 50,000 unlabeled samples (X^U), from which 1,000 are randomly selected to make up the labeled sample set (X^L). 80% of the labeled samples (i.e., 800) are used for training and 20% (i.e., 200) for validation in training the softmax classifier and fine-tuning. These samples are balanced among all the classes.

A sequence of 10 experiments (10 iterations) is conducted for each method. In each iteration, the random labeling method randomly picks 200 unlabeled samples, labels them, and adds them to the training set and validation set (80% for training and 20% for validation). The training and validation set are used for training the softmax classifier and fine-tuning. The classification accuracy reported for performance comparison is obtained based on the 10,000 test samples.

AL-DL is experimented in a similar way. In each iteration, *AL-DL* uses the deep learning network trained in the previous iteration to predict unlabeled samples and select 200 unlabeled samples to be labeled and added into the training and validation set. Note the samples picked are not necessarily balanced.

Ten random trails are done for each setting and the mean and standard deviation of classification accuracy of these four methods are reported.

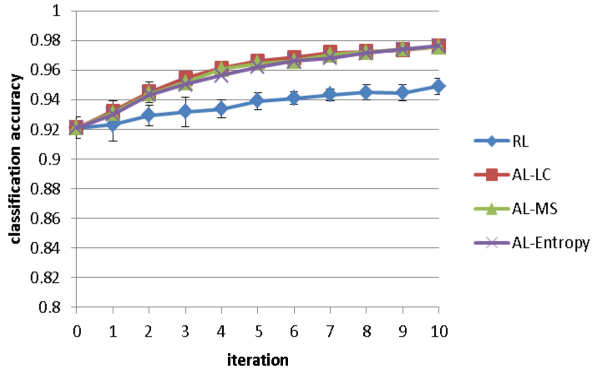

Fig. 6. Performance comparison of *AL-DL* with 3 different selection metrics (*AL-LC, AL-MS*, and *AL-Entropy*) and random labeling (*RL*) on raw MNIST data, using stacked RBMs deep learning networks.

Fig. 6 shows the performance comparison of *AL-DL* with 3 different selection metrics (*AL-LC, AL-MS*, and *AL-Entropy*) and random labeling (*RL*) on raw MNIST data. Iteration 0 has 1,000 labeled samples and then 200 more labeled samples are added in subsequent iteration, i.e.
1,200 in Iteration 1, 1,400 in Iteration 2, and so on, so forth. The results show that the performances of the three *AL-DL* algorithms are comparable and are significantly better than that of RL. The standard deviations of the three *AL-DL* algorithms are smaller than that of RL, suggesting more consistent performance.

Fig. 7 compares the 200 unlabeled samples selected in the 1st iteration by *RL* and *AL-LC*. The principle of *AL-DL* algorithms is to select the most uncertain or hardest unlabeled samples to be labeled. The result in Fig. 7 reveals that the digits in the bottom graph look harder to recognize, indicating *AL-LC* does pick more challenging samples than *RL*.

Fig. 8 compares the classification errors of *RL* and *AL-LC* on their respective training and validation sets in the first iteration. Note that although *RL* and *AL-LC* starts with the same training and validation set (the labeled set), they have different training and validation set after the first iteration because they add different labeled samples. It is interesting to observe that the *AL-DL* algorithms have larger classification errors on training and validation set than *RL* because they always add the most challenging samples to the training set and validation set. Nevertheless, because *AL-DL* algorithms learn from more challenging samples, they outperform the random labeling method on test samples.

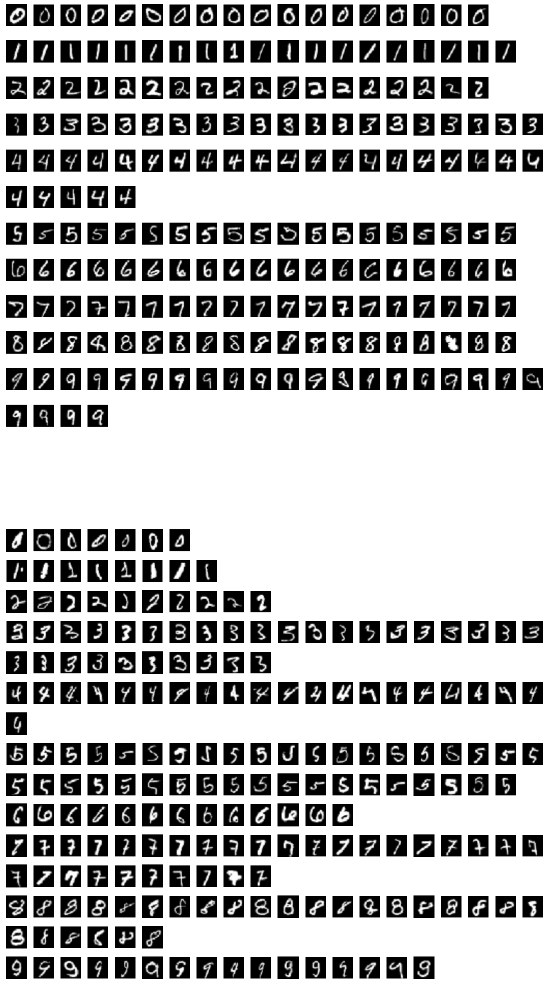

Fig. 7. Comparison of the 200 unlabeled samples selected in the 1st iteration by *RL* (top) and *AL-LC* (bottom). The digits in the bottom graph look harder to recognize, indicating *AL-LC* does pick more challenging samples than *RL*.

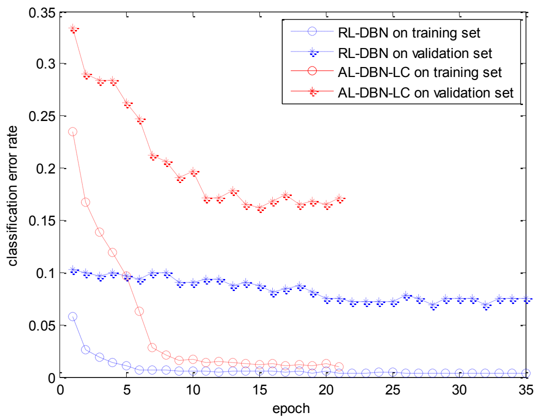

Fig. 8. Classification error comparison of *RL* and *AL-LC* on their respective training and validation sets in the first iteration. *AL-LC* has much larger error on training and validation sets because it picks more challenging samples to add into its training and validation sets.

D. Stacked Autoencoders on Raw Data

In comparison to stacked RBMs, similar experiments for performance comparison between *AL-DL* algorithms and *RL* are carried out using stacked autoencoders with 2 hidden layers, each having 200 nodes. Fig. 9 shows *AL-DL* algorithms outperform *RL* significantly. *AL-MS* is slightly better than the other two *AL-DL* algorithms.

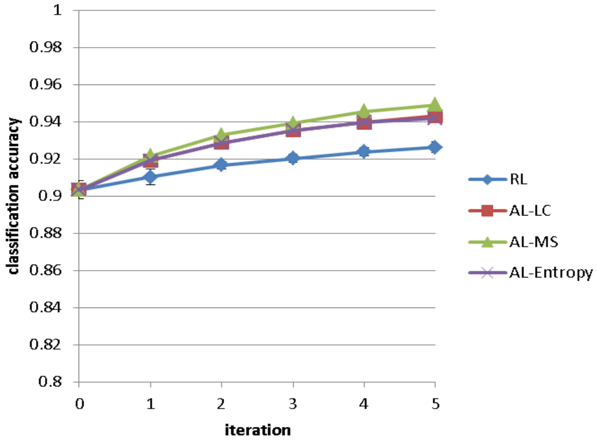

Fig. 9. Performance comparison of *AL-DL* with 3 different selection metrics (*AL-LC, AL-MS*, and *AL-Entropy*) and random labeling (*RL*) on raw MNIST data, using stacked autoencoders deep learning networks.

E. Stacked Autoencoders on Pre-processed Data Using PCA at Retention Rate 95% and Whitening

These experiments compare the performances of *AL-DL* algorithms and *RL* on stacked autoencoder deep learning networks working on pre-processed data, instead of raw data. MNIST input images are first pre-processed by PCA at retention rate 95% and whitening. The input dimension is reduced from 784 (28 x 28 pixel) to 121.

Fig. 10 shows the results using stacked autoencoders with 1 hidden layer of 300 nodes. Again, *AL-DL* algorithms outperform *RL* significantly, up to 4% in iteration 5. *AL-MS* is slightly better than the other two *AL-DL* algorithms.

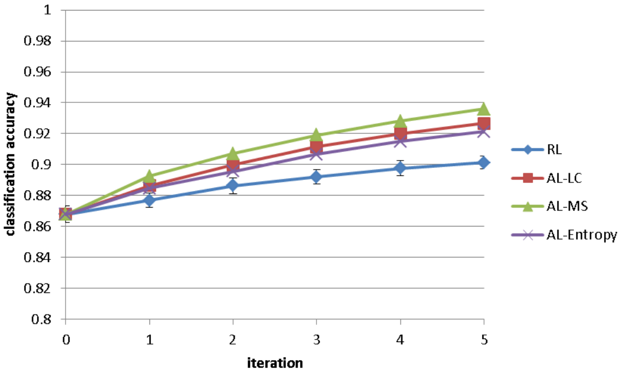

Fig. 10. Performance comparison of *AL-DL* with 3 different selection metrics (*AL-LC, AL-MS*, and *AL-Entropy*) and random labeling (*RL*) on PCA-reduced MNIST data, using stacked autoencoders deep learning networks.

V. Experimental Results on Sleep Stage Dataset

In this set of experiments, a real-world sensor data set on sleep stage research [17] is used to investigate the performance of *AL-DL*. Different from the MNIST data set, this data is more complex and noisy.

The experiments use acquisitions from 5 subjects in the dataset, each consisting of 1 EEG channel (C3-A2), 2 EOG channels, and 1 EMG channel downsampled at 64Hz. Each acquisition lasts about 7 hours on average. Each sample is taken from a 1-second window with 256 dimensions, normalized to the range [0, 1]. The labels are 5 sleep stages: awake, stage 1 (S1), stage 2 (S2), slow wave sleep (SWS) and rapid eye-movement sleep (REM).

To pre-process the data, a band-pass filter and a notch filter at 50Hz are applied to all channels. Then, the 30-second data before and after a sleep stage switch are removed. Finally, all classes are balanced, i.e. having the same number of samples, based on the class with the fewest samples. The data is randomly divided into 7 groups, with 6 groups for training and 1 for testing, which results in 51042 samples for training and 8508 for testing.

In addition, following previous work, 28 manually-designed features are extracted from 1-second samples. The details of how to calculate these features can be found in [18-22].

AL-DL and *RL* algorithms start with a base set of 1,000 labeled samples. These samples are balanced among all 5 classes. The experiments run for 10 iterations for stacked RBMs and 5 iterations for stacked autoencoders. In each iteration, 200 unlabeled samples are selected, labeled, and added to the training set. 10 random trials are done for each algorithm on each setting.

A. Stacked RBMs on Raw Data

Here, the performances of *AL-DL* algorithms and *RL* are compared using a DBN with a single hidden layer of 200 hidden nodes on the raw sleep stage data. Fig. 11 shows the results and all 4 algorithms perform similarly.

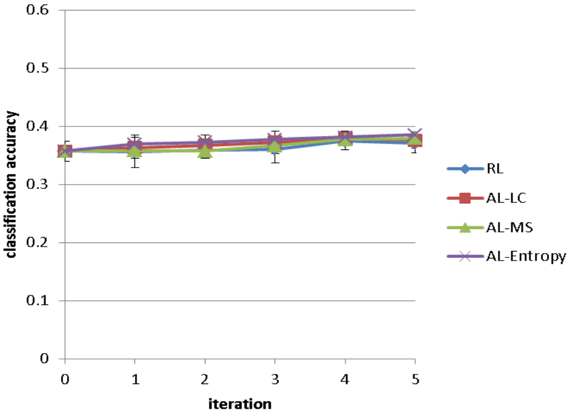

Fig. 11. Performance comparison of *AL-DL* with 3 different selection metrics (*AL-LC, AL-MS*, and *AL-Entropy*) and random labeling (*RL*) on raw sleep stage data, using stacked RBMs.

B. *Stacked RBMs on Manually-Designed Features*

Instead of using raw data, this set of experiments use the 28 manually-designed features as proposed in previous work as inputs to DBNs. The DBN has a single hidden layer of 200 hidden nodes. Fig. 12 shows the results and all 4 algorithms perform similarly. Compared to Fig. 11, higher classification accuracies are obtained by all algorithms using manually-designed features.

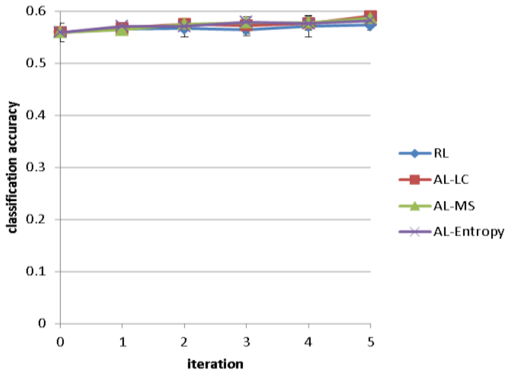

Fig. 12. Performance comparison of *AL-LC, AL-MS*, *AL-Entropy*, and random labeling (*RL*) on manually-designed features of the sleep stage data, using stacked RBMs.

C. *Stacked Autoencoders*

Instead of stacked RBMs, stacked autoencoders were also used in extensive tests to compare *AL-DL* and *RL* algorithms on the sleep stage data and manually-designed features. The results are similar to those on stacked RBMs: all *AL-DL* and *RL* algorithms perform similarly.

VI. Conclusion

Inspired by the general active learning framework, we propose a new active labeling method, *AL-DL,* for cost-effective selection of data to be labeled. On the MNIST dataset, *AL-DL* performs much better than random selection, no matter what uncertainty measurements (least confidence, marginal sampling, or entropy), what learning units (RBMs or autoencoders), or what data pre-processing techniques (raw or PCA processed data) are used.

However, *AL-DL* shows no improvement over random labeling on the more challenging sleep stage dataset. This could be explained by the noisy nature of the data. Since active labeling proactively picks the most uncertain samples to be labeled, in the challenging data set, these samples may be mislabeled or not representative, introducing false information to the model.

References

[1] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems,* vol. 19, pp. 153-160, 2007.

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science,* vol. 313, pp. 504-507, 2006.

[3] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel distributed processing: explorations in the microstructure of cognition*, ed Cambridge, MA, USA: MIT Press, 1986, pp. 194-281.

[4] B. Settles, "Active learning literature survey," University of Wisconsin, Madison2010.

[5] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation,* vol. 18, pp. 1527-1554, 2006.

[6] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation,* vol. 14, pp. 1771-1800, 2002.

[7] G. Hinton, "A practical guide to training restricted Boltzmann machines," *Momentum,* vol. 9, p. 1, 2010.

[8] Y. Le Cun, "Modèles connexionnistes de l'apprentissage," PhD dissertation, Paris 6, 1987.

[9] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics,* vol. 59, pp. 291-294, 1988.

[10] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and Helmholtz free energy," *Advances in neural information processing systems,* pp. 3-10, 1994.

[11] D. F. Wulsin, J. R. Gupta, R. Mani, J. A. Blanco, and B. Litt, "Modeling electroencephalography waveforms with semi-supervised deep belief nets: fast classification and anomaly measurement," *Journal of Neural Engineering,* vol. 8, pp. 1-13, 2011.

[12] S. Zhou, Q. Chen, and X. Wang, "Active deep networks for semi-supervised sentiment classification," in *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, 2010, pp. 1515-1523.

[13] N. Roy and A. McCallum, "Toward optimal active learning through monte carlo estimation of error reduction," *ICML, Williamstown,* pp. 441-448, 2001.

[14] C. E. Shannon and W. Weaver, "A mathematical theory of communication," *The Bell System Technical Journal,* vol. 27, pp. 379-423, 623-656, July, October 1948.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, pp. 2278-2324, 1998.

[16] A. Ng. (2013). *UFLDL Tutorial*. Available: http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

[17] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, *et al.*, "Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals," *Circulation,* vol. 101, pp. e215-e220, 2000.

[18] M. Längkvist, L. Karlsson, and A. Loutfi, "Sleep Stage Classification Using Unsupervised Feature Learning," *Advances in Artificial Neural Systems,* vol. 2012, 2012.

[19] A. R. Osborne and A. Provenzale, "Finite correlation dimension for stochastic systems with power-law spectra," *Physica D: Nonlinear Phenomena,* vol. 35, pp. 357-381, 1989.

[20] L. Zoubek, S. Charbonnier, S. Lesecq, A. Buguet, and F. Chapotot, "Feature selection for sleep/wake stages classification using data driven methods," *Biomedical Signal Processing and Control,* vol. 2, pp. 171-179, 2007.

[21] E. Pereda, A. Gamundi, R. Rial, and J. Gonzalez, "Non-linear behaviour of human EEG: fractal exponent versus correlation dimension in awake and sleep stages," *Neuroscience letters,* vol. 250, pp. 91-94, 1998.

[22] T. Gasser, P. Bächer, and J. Möcks, "Transformations towards the normal distribution of broad band spectral parameters of the EEG," *Electroencephalography and clinical neurophysiology,* vol. 53, pp. 119-124, 1982.