Artificial Neural Networks and Deep Architectures,
DD2437

Lab assignment 4

Deep neural network architectures with restricted Boltzmann machines and autoencoders

1 Purpose

The main purpose for this lab is for you to get familiar with some of the key ingredients of deep neural network (DNN) architectures. The focus in this assignment is on restricted Boltzmann machines (RBMs) and autoencoders. After completing this assignment, you should be able to

- explain key ideas underlying the learning process of RBMs and autoencoders,
- apply basic algorithms for greedy training of RBM and autoencoder layers with the use of commonly available deep learning libraries,
- design multi-layer neural network architectures based on RBM and autoencoder layers for classification problems,
- study the functionality (including generative aspects) and test the performance of low-scale deep architectures developed using RBMs (deep belief networks, DBNs) and autoencoders.

2 Scope and resources

In this lab assignment you are encouraged to rely on the existing deep learning libraries (available in Matlab, Python, Java among others; examples are scikit-learn, deeplearning4j and Deep Neural Network Matlab toolbox by M. Tanaka). Yet, I would recommend that you explore nuances of setting up RBMs and their implementation. Hintons guide [1] could be helpful in this regard. Consequently, while reusing library implementations, it is strongly advised that you should not just copy ready examples from the libraries that address very similar problems to those in the assignment (it is possible to find such example scripts on the web since the lab makes use of one of the classical benchmark tests). Unlike typical benchmark applications of DBNs, here the data have already been normalized and converted to “pseudobinary” distributions, so that you could directly employ a Bernoulli type of RBMs, not Gaussian. You can read more on the comparative analysis of different versions of RBMs in [2]. Finally, please be aware that you are given a fair deal of freedom in many choices that have to be made in this lab. Please motivate your decisions even if you find them somewhat arbitrary. Also, since this lab is only reported in writing please invest extra effort into your reports.

3 Tasks and questions

The lab consists of two main tasks, each one covering two approaches - based on RBMs and autoencoders. The data for the assignment can be downloaded from the course website. In particular, the MNIST dataset consists of four csv files with data for training, `bindigit_trn` and `targetdigit_trn`, and the other two for testing, `bindigit_tst` and `targetdigit_tst`. Data in bindigit files represent 28-by-28 matrices organized into 784-dim binary vectors (strings of 0s and 1s). There are 8000 such vectors in bindigit_trn and 2000 in`bindigit_tst`. Analogously, `targetdigit_trn` file contains a vector of 8000 integer values and `targetdigit_tst` has 2000 integers between 0 and 9, which describe corresponding labels for the 28-by-28 images of handwritten digits (data adapted from the MNIST database, normalized with grey levels converted to simpler binary representations). Data in both training and test sets are relatively balanced with respect to 10 classes. You can verify this by examining histogram of the available labels. Furthermore, in Matlab you can plot a digit image represented by the k-th vector in bindata matrix using `imshow(reshape(bindata(k,:),28,28))`.

3.1 RBM and autoencoder features for binary-type MNIST images

Your task here is to train a) an RBM and b) autoencoder (weights connecting visible and hidden unit layers). More specifically, please first initialize the weight matrix with small (normally distributed) random values with hidden and visible biases initialized to 0. Then, iterate the training process, contrastive divergence for RBM and gradient descent based error minimization for the autoencoder, for a number of epochs (i.e. full swipes through the training data) equal to 10, 20 until convergence (you can experiment a bit, also adjusting the learning rate).

- For each image compute the mean error between the original input and the reconstructed input. Then use it to compute the total error on the dataset for the current epoch. Once training is completed, plot the total error as a function of the epochs. Finally, sample one image from each digit class and obtain its reconstruction, then plot both the original and reconstructed images. Try different number of hidden nodes, say 50, 75, 100, 150, and compare errors.
- Next, plot the 784 bottom-up final weights for each hidden unit, using one different figure for each hidden unit (reshape the weight vector as a matrix and plot it as an image). Do this part for the configurations with 50 and 100 nodes.

Discuss your observations and illustrate your findings with plots as well as both quantitative and qualitative arguments. Choose most interesting comparisons and effects to demonstrate.

3.2 DBN and stacked autoencoders for MNIST digit classificatione

Taking advantage of the developments in the previous task, here you are requested to extend a single-hidden layer network to a “deeper” architecture by following the idea of greedy layer-wise pretraining (without labels as in the previous task for a single layer). This time however you will add at the top of the network,s hidden layers the output layer, i.e. the layer with output nodes corresponding to the classification output. Please train the weights of the connections from the top-most hidden layer to the output layer with a generalized delta rule or conjugate gradient optimization. Next, perform test and quantify the classification performance on the test set. In particular, please address the following tasks and questions.

3.2.1 Classification with deeper architectures

Compare the classification performance obtained with different number of hidden layers (1,2 and 3). Add to this analysis, please, a network configuration with a simple classification layer operating directly on the raw inputs as the no-hidden-layer option. As the size of hidden layers, first choose the optimal number of nodes in the first hidden layer based on your experiences in the previous task (3.1) and then decide on the size of the other layers within a similar range with tendency to have less and less units. Run these comparisons/analyses independently for stacked autoencoders and DBNs (“stacked” RBMs). Examine the hidden layer representations (beyond the first hidden layer already studied in the previous task). Observe the effect of images representing different digits on hidden units in the hidden layers. Finally, compare the deep network configurations of your choice, DBN and stacked autoencoders (two- or three-layer networks with selected number of hidden units), pre-trained in a greedy layer-wise manner and containing a classification layer trained in supervised mode with an analogous MLP architecture trained from scratch using backprop. An intermediate option would be a comparison with deep networks pretrained as before but with backprop-type fine tuning of the weights in hidden layers (not only in a supervised output layer).

Additional remarks

- Instead of using an extra supervised network layer to connect hidden-layer-representations via gradient descent or conjugate gradient optimization (some sort of gradient neural supervised learning), you can perform tests using a commonly employed logistic regression.
- In addition, if the simulations take heavy computational toll on your PC/laptops etc., please feel free to subsample your training set maintaining the class balance.

3.2.2 Generative mode of DBNs (optional part)

After training, the DBN can be used to generate sample digit images for a selected class (digit). To this end, the desirable output class configuration should first be clamped to particular values (0s and 1s for the output one-out-of-n coding). Next, the remaining visible units of the top-level RBM should be sampled randomly according to their bias terms, which initializes the visible data vector for the top-level RBM to a reasonable unbiased starting point. Next, alternating Gibbs sampling, which was used in the learning process as part of contrastive divergence, should run for many steps. It is expected that after many steps the network settles close to its equilibrium distribution given the clamped labels. Then, a single top-down pass converts the binary feature activations into an image consistent with the sample from the top-level RBM. In this task, you could qualitatively examine the DBN,s "perception" of digit images. This capability to sample data from a trained network constitutes a valuable and interesting property of generative models like DBNs.

As before, please share your key observations and illustrate your findings. Choose most interesting comparisons and effects to demonstrate, be selective (with different hyperparameter configurations of your choice, comment on the sensitivity if you decide to examine a selected hyperparameter more systematically). Mention compute time aspects, convergence, reconstruction and classification errors across layers. Briefly discuss/interpret hidden layer representations and features.

Good luck!

References

Hinton, G. E. (2012). A practical guide to training restricted Boltzmann machines. In Neural networks, Tricks of the trade (pp. 599-619). Springer, Berlin, Heidelberg.

Yamashita, T., Tanaka, M., Yoshida, E., Yamauchi, Y., and Fujiyoshii, H. (2014). To be Bernoulli or to be Gaussian, for a restricted Boltzmann machine. In the 22nd International Conference on Pattern Recognition (ICPR) (pp. 1520-1525).