Artificial Neural Networks and Deep Architectures, DD2437

Lab assignment 4

Deep neural network architectures with 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 stacked autoencoders. After completing this assignment, you should be able to

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

2 Scope and resources

In this lab you can either develop your own implementation of an autoencoder layer for a stacked auroencoder network (DNN), based on the experience gained in lab 1, or you could use a deep learning library of your choice (available in Matlab, Python, Java among others). The functionality needed for the lab simulations encompasses backprop learning with regularization. Yet, I would recommend that you explore nuances of setting up your networks and their implementation. 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). 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.

3 Tasks and questions

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 Autoencoder for binary-type MNIST images

Your task here is to develop a simple autoencoder as a building block for the DNN. More specifically, please first initialize the weight matrix with small (normally distributed) random values with hidden and visible biases initialized to 0. Use the sigmoid function for the activation function, $f(z) = \frac{1}{1+\exp(-z)}$. Then, iterate the training process relying on gradient descent based error minimization for a number of epochs (i.e. full swipes through the training data) until convergence (you can experiment a bit, also adjusting the learning rate). Importantly, please focus on undercomplete representations, i.e. with the number of hidden units lower than the input dimensionality, and incorporate a regularisation term during training (so that you minimise the sum of mean square error plus the penalty term, e.g. $\lambda \|w\|^2$, by means of backprop).

- 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 sizes of the hidden layer representation and compare errors. Also, measure the average sparseness of hidden layer representations.

- Next, plot the 784 bottom-up final weights for each hidden unit, using a separate 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, 80 and 120 nodes.

Report your observations and illustrate your findings with plots as well as both quantitative and qualitative arguments. Choose most interesting comparisons and effects to demonstrate. Interpret your findings and discuss how you could force a higher degree of sparseness on hidden layer representations.

3.2 Stacked autoencoders for MNIST digit classification

Taking advantage of the developments in the previous task, 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 gradient descent or conjugate gradient optimization. Next, perform test and quantify the classification performance on the test set. 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. 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 stacked autoencoder configurations of your choice (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 the 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.

As before, please share your key observations and illustrate 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 hidden layer representations and features.

Good luck!