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 and test the performance of low-scale deep architectures developed using autoencoders.

2 Scope and resources

Although you can develop in this lab your own implementation of an autoencoder layer for a stacked autoencoder network (DNN), based on the experience gained in lab 1, you are encouraged to use an existing 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 to achieve the effect of sparsification, and the with mean squared error (MSE) and cross-entropy loss. Consequently, while reusing library implementations, especially in Tensorflow or Keras, 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 ($f(z) = \frac{1}{1+\exp(-z)}$) as the activation function. You could also experiment with the rectifier (ReLU with the logistic function for the approximation of the derivative). Please bear in mind however that the last layer should still rely on the sigmoid to produce the outout in the requested range ([0,1]). Then, iterate the training process relying on gradient descent based error/loss 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). Please focus first on undercomplete representations, i.e. with the number of hidden units lower than the input dimensionality. In the next step experiment with overcomplete representations for larger number of hidden units. For this you should incorporate a regularisation term during training (so that you minimise the error/loss plus the penalty term, e.g. $\lambda \|w\|^2$ or $\lambda \|w\|$, by means of backprop). If you examined the use of ReLU in the hidden layer you could comment on how they compare with the sigmoidal units, especially in terms of the convergence rate (particularly interesting for the overcomplete autoencoders). Finally, please employ also in your study denoising autoencoders (only for the undercomplete case), where the target outputs should be reconstructed from the original input samples corrupted with additive Gaussian noise (you coud test with varying level of noise but iverall it should remain rather low, below 0.4).

- 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 assuming some low threshold below which the absolute value of the unit activation is neglected (considered as 0 and thus contributing to the sparse representation).

- 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 100, 200 and 400 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 the higher degree of sparseness can be promoted in 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 (a bit more or less; the classification performance on the training or validation dat subset should guide this process) 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.

Additional remarks

- Instead of using an extra supervised network layer to connect hidden-layer-representations and train them with gradient descent, you can perform tests using a commonly employed logistic regression.

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!