Artificial Neural Networks and Deep Architectures, DD2437

Lab assignment 4

Restricted Boltzmann Machines and Deep Belief Nets

Naresh Balaji Ravichandran and Pawel Herman

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 deep belief nets (DBNs). After completing this assignment, you should be able to

- explain key ideas underlying the learning process of RBMs,
- apply basic algorithms for unsupervised greedy pretraining of RBM layers and supervised fine-tuning of the resultant DBN,
- design multi-layer neural network architectures based on RBM layers for classification problems,
- study the functionality of DBNs including generative aspects

2 Scope and resources

You can use in this lab some existing libraries of your choice for developing RBM layers of a DBN. However, you are strongly encouraged to rely on the attached code framework made available in Python (read carefully `readme` file). The functionality needed for the lab simulations includes contrastive divergence and wake-sleep learning algorithms. In case youi still decide to use library implementations, especially in Tensorflow or Keras, it is advised that you should not just copy ready examples from the libraries that address very similar problems to those in the assignment).

Finally, although the Background section provides a brief and useful description of RBMs, DBNs and the associated learning algorithms, it is necessary that you thoroughly read the literature listed in the References, in particular a **practical guide by Hinton (2012) [1] and Hinton et al. (2006) [2]**. These sources provide not only theoretical insights but also cast light on practicalities directly relevant to this lab.

3 Background

3.1 Restricted Boltzmann machine

A Restricted Boltzmann Machine (RBM) is a two layer neural network with undirected (symmetric) connections that can be used to learn features from data. One set of units represent the data, called the "visible" layer and denoted by $\mathbf{v}$, and another set represents latent features of the data, called the "hidden" layer and denoted by $\mathbf{h}$. In its simplest form, all the units are binary units - they can be either turned ON or OFF corresponding to the activity level of 1 or 0, respectively. Importantly, since the units are stochastic, they get activated (ON state, i.e. $x = 1$) with the probability $p(x = 1)$[1]. The joint state of the RBM is therefore a statistical sample and constitutes a set of assignments of either ON or OFF to every visible and hidden unit.

The parameters of the RBM are weights w_{ij} between every visible unit i and hidden unit j, biases b_i for each visible unit i, and biases b_j for each hidden unit j. There are no connections within the layers (hence the term "restricted"). The parameters assign a probability to each state of the RBM, that is, they describe a joint probability $p(\mathbf{v}, \mathbf{h})$ for all assignments of $\mathbf{v}$ and $\mathbf{h}$. Learning in this model (by maximum likelihood) amounts then to updating all the parameters such that the observed visible states are the most likely states, that is, they are assigned maximum probability by the network.

We can also, for a particular value of parameters, infer the most likely state of the RBM. This is particularly simple since the RBM does not have any connections within a layer. We start at either of the two layers, say the visible layer, and assign a state for each unit i in that layer by sampling from the probabilities $p(v_i = 1) = \sigma(b_i)$, where $\sigma(\cdot)$ is the sigmoid activation function. The probability of a hidden unit j turning ON can then be driven from the visible units by sampling from $p(h_j = 1) = \sigma(b_j + \sum_i w_{ij} v_i)$. And subsequently, the probability of turning ON visible units using the hidden units is $p(v_i = 1) = \sigma(b_i + \sum_i w_{ij} h_j)$. This back and forth sampling of the two layers is called alternating Gibbs sampling (see Figure 1), and the probability of the state is guaranteed to converge.

Theoretical question Why is there a guarantee to converge after sufficient number of alternating Gibbs sampling?

对比分歧

For learning the parameters of the RBM, the contrastive divergence learning (CD_k) gives the update of weight between ith visible unit and jth hidden unit to be : $\Delta w_{ij} \propto \langle v_i h_j \rangle^{t=0} - \langle v_i h_j \rangle^{t=k}$, where k is the number of steps of alternating Gibbs sampling (see Figure 1). The v_i and h_j in the first term are the activities of the visible unit (which we choose to be the data) and the activities of hidden unit driven by the visible layer. The second term has v_i and h_j which are the activities of visible and hidden unit, respectively, after k steps of Gibbs sampling.

[1] In your priogramming assignment you can draw a Bernoulli sample (binomial distribution).

binary distribution
prob: p=q
prob: p=1-q

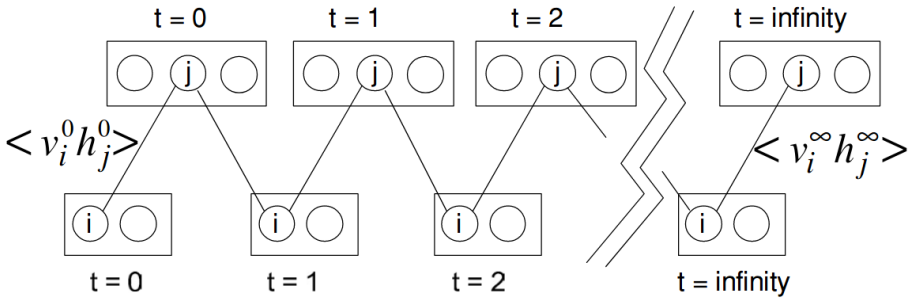

Figure 1: Maximum liklihood and contrastive divergence learning. Image from Hinton et al. [2]

3.2 Pretraining deep networks with greedy layer-by-layer training

After learning the RBM, the weights can be frozen (kept fixed) and the representations obtained in the hidden layer can be treated as input training data for another RBM stacked on top. For the new RBM, the visible layer coresponds to the hidden layer of the RBM below in the stack. Once the new RBM has been trained, more and more RBMs can be stacked on top and, as a result, weights of a deep network are pretrained layer-by-layer. This whole procedure is a greedy approach since we assume that separate learning of each RBM contributes to a globally useful representation of the true data. At the same time the learning algorithm does not rely on any labels and is thus considered as an unsupervised (pre)learning scheme.

The resulting network is called a deep belief net (DBN). The network can be used as a generative model by running alternating Gibbs sampling at the top RBM and then subsequently sampling from the top to the bottom layer. However, for practical considerations, we would like to use the network for recognition (classifying data, i.e. assigning labels) as well as generation (generating data from labels) tasks. This requires that the learnt data representations are associated with their corresponding true labels. This can be done by training the top RBM in the stack, not just with the representations from the previous RBM, but with the concatenation of representation from the last RBM and the true labels (see Figure 2).

联合成

Stacking more layers on top of an RBM changes the way the weights can be used to drive the network activity: the undirected connections between visible and hidden units are untied into two sets of directed connections, one from visible to hidden units and another from hidden to visible units. Consequently, when the data is presented as an input to the bottom layer the network can only be driven bottom-to-top, and we should not, for instance, perform alternative Gibbs sampling in the RBM at the bottom of the stack. The top RBM, however, has no more RBMs stacked up above so we can perform alternating Gibbs sampling like in any regular RBM.

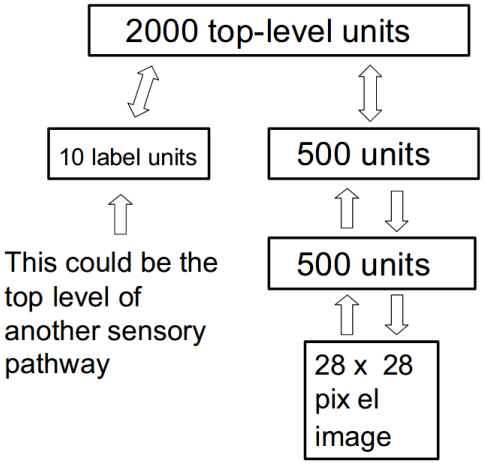

Figure 2: Deep belief network. Image from Hinton et al. [2]

Theoretical question Why does stacking RBMs make the originally undirected connections directed? And why does the top RBM retain its undirected connections? Recall that the undirected connections was due to the joint probability distribution over hidden and visible units [2].

保留

3.3 Fine-tuning deep networks

Layer-wise pretraining of DBNs is useful for weight initialisation but, due to its greedy nature, the lower layers could not adapt to representations learnt in the higher layers and, crucially, to the true labels. A fine-tuning step can be used to update all the weights of the network simultaneously based on the recognition and generation performance. We will use the wake-sleep (same as up-down) learning rule for this purpose. In the wake phase, the network is driven bottom-up in a single pass starting with the data provided in the bottom layer. In the top RBM (which has undirected connections) just a few iterations of alternating Gibbs sampling are run. Now the top RBM holds representation of the presented visible data, and the activities of all layers except the top RBM can be removed. In the sleep phase, the network is driven top-down in a single pass, and the activities obtained in the bottom layer can be interpreted as the network's reconstruction of visible data seen in the wake phase.

The key idea of learning in the wake-sleep algorithm is to learn the generative weights while driving the network bottom-up with recognition weights (the wake phase), and learn the recognition weights while driving the network top-down using generative weights (the sleep phase). The wake phase fills up the layers with activities, from bottom to top, and generative connections/weights are used to predict (these predictions do not change the network activities) and learn the generative weights from a unit i to a unit j in the layer below as follows: $\Delta w_{ij} \propto x_i(y_j - \tilde{y}_j)$. The activities y_j and x_i are the activities from the wake-phase and $\tilde{y}_j$ is the prediction made using generative weights. In a

very similar procedure, while driving the network top-down in the sleep phase, the bottom-up recognition weights are trained. The top RBM has undirected connections and is a regular RBM, so the ordinary CD_k learning can be applied. The wake-sleep algorithm trains simultaneously the recognition and generative connection weights. The resulting DBN is both a discriminative and generative model of the data. In other words, it can be used to predict the labels when given the data $p(label|visible)$, and generate data when clamping (fixing) the labels $p(visible|label)$.

4 Tasks and questions

The data for the assignment can be downloaded from the course website. This is a standard machine learning benchmark dataset called MNIST and consists of grayscale images of handwritten digits with the corresponding labels (0-9) - there are four binary files with 60k samples for training, `train-images-idx3-ubyte` and `train-labels-idx3-ubyte`, and the other two files with 10k test samples, `t10k-images-idx3-ubyte` and `t10k-labels-idx3-ubyte` Images are represented as 28-by-28 matrices organized into 784-dimensional vectors (the data can be loaded with a `load_idxfile` in `util.py`). 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. You can also plot some example digit images.

Importantly, as mentioned earlier, you have a Python code available that is documented in `readme` file. The code also contains parameters set to values that worked for us. Apart from most common libraries, `numpy` and `matplotlib.pyplot` you need a library `struct` and `matplotlib.animation`).

4.1 RBM for recognising MNIST images

Your task here is to develop an RBM with binary stochastic units and train it with a contrastive divergence algorithm CD_1 (weights connecting visible and hidden unit layers) to learn data representations in an unsupervised manner. The architecture with 500 units in the hidden layer is recommended.

- More specifically, please first initialize the weight matrix (including hidden and visible biases) with small random values (normally distributed, N(0,0.01)). Then, iterate the training process (CD) for the number of epochs varying between 10 and 20 for minibatches of size 20 (i.e. each epoch corresponds to a full swipe through a training set divided into mini-batches). The idea here is to obtain some level of convergence or stability in the behaviour of the units. How can you monitor and measure such stability?

- Please investigate how the average reconstruction loss (the mean error between the original input and the reconstructed input) is affected by the number of hidden units decreasing from 500 down to 200.

- After learning you should examine the outcomes. Since the training has been conducted without labels the evaluation boils down to examining

保真度

接受的

the fidelity of the reconstructed images as well as the nature of the receptive fields that each unit develops throughout the learning process. To study receptive fields it is recommended that weights to the visible layer are visualised for the hidden units of interest. Please demonstrate these receptive fields as images (reshape the weight vector as a matrix and plot it as an image) for selected units in the hidden layer. You can use the existing implementation for this purpose (function `viz_rf` in `util`). Can you interpret some of them?

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

4.2 Towards deep networks - greedy layer-wise pretraining

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. Following unsupervised learning of data repreentations, DBN can be further pretrained to account for labels and thus enable image recognition. The ultimate objective is to examine the generalisation performance of the resulting DBN. As discussed in section 3.2, please bear in mind that label units are concatenated with the hidden layer of the last RBM in the stack and another layer is added at the very top with bi-directional connections to perform Gibbs sampling (cf. Fig. 2). As before, you are encouraged to exploit the available Python code. Please, address the following tasks and questions.

- Use the RBM machinery in the previous section to build a network with two RBMs in the stack, trained greedily one layer after another, with CD algorithm. It is recommended that you design your network based on the 500-hidden-unit architecture, i.e. 784-500-500 (cf. Fig. 2). Please report the reconstruction losses for both RBMs in the stack.

- In the next stage, you are requested to pretrain a DBN to perform image recognition.To this end, extend the architecture in line with Fig.2 and section 3.2, and train the topmost RBM (500+10 - 2000) using CD algorithm. It is important that once you included label units, you clamp them correspondingly to the representations of digit images. Once this process is finalised, you can now test your recognition performance. First, initialise label units with the value of 0.1 and then perform feedforward propagation of the input images through hidden layers of DBN, followed by multiple iterations of Gibbs sampling (accounting for the topmost layer of 2000 units). It is suggested that you include soft-max operation for the 10 label units. Please examine the convergence of label units and the resulting recognition/classification accuracy.

- Pretrained DBN can also be used in a generative mode. In other words, you can study 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. To investigate the nature of generated images by your DBN, first clamp the label units (keep these label units fixed: 0s and single 1 that implement one-out-of-ten coding) and

perform Gibbs sampling for at least 200 iterations (cf. Fig. 2 and section 3.2). As a result, you obtain for each unit in the concatenated (500+10) layer of the top RBM a probability of activation, $p(h_j = 1)$. Then you can sample binary values from these probabilities across the hidden layer, propagate them down to the hidden layer of the lower RBM where you obtain another set of probabilities, which can in turn be used for sampling the binary activity states in this layer. Finally, these binary activities in the hidden layer of the bottom RBM can be further projected down to the input (visible) layer and represent the generated images. Please make sure that you use weights of the downward generative projections. Since you obtain probabilites in the top RBM, the sampling process and propagation of activity down in the network to the input layer can be done multiple times and thus for the same clamped label you can generate numerous samples. To facilitate visual inspection you can stitch them together and make an animation using the attached function `stitch_video` in `util.py` (Python library `matplotlib.animation` is needed).
What are the key factors that determine the quality of generated images? How do the generated images compare visually to the original input patterns?

??

缝

4.3 Supervised fine-tuning of the DBN

To improve the classification accuracy reported earlier, you are requested to fine-tune the DBN, pretrained in the previous section, using a supervised learning approach that rests on the wake-sleep learning rule (section 3.3). Then you will design and train a simpler network to test whether the generalisation performace can be retained. In particular, please address the following tasks.

- Building on the pretrained DBN in the previous section (with the architecture 784, 500, 500+10, 2000), train the network by implementing the wake-sleep approach that exploits the information about the target labels for input samples. This step is well documented, including typical parameter values, in the attached `readme` file and discussed in section 3.3. Please compare the classification performance of the fine-tuned vs pretrained DBN.

- Although the main objective has been to compare the DBNs in terms of their classification performance, you are also requested to report how the generative model improved after fine-tuning. To investigate this please follow the same generative procedure as described in section 4.2 and make a comparative (qualitative) evaluation. Please perform analysis for both architectures studied in this section.

- Design a simpler network than in the previous task, with one hidden layer removed, i.e. 784, 500+10, 2000. Pretrain it first using the CD algorithm, as described in section 3.2 (see also section 4.2), and then fine-tune it with the wake-sleep learning. Make a thorough comparative analysis of the generalization performance (classification accuracy reported on the test set) obtained with the pretrained vs fine-tuned DBNs.

Overall, 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.
Also, do not necessarily prioritise achieving the maximal accuracy since your compute and time resources are limited.

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.

Hinton, G. E., Osindero, S., and The, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18(7):1527-1554.*