Cognition and Computation Essay - Option 2

Silvia Poletti
Student nr. 1239133

1. Introduction

The goal of this simulation is to explore how a Deep Belief Network (DBN) learns to represent the data. The evaluation of the model is made considering the accuracy in multilevel classification task.
It resulted that a DBN is able to learn some high-level features of the data, so that it approaches human brain abilities. Moreover, the DBN has shown robust to noise, but criticisms arised in case of Adversarial attacks.

2. Materials and methods

This simulation considered the EMNIST Letters dataset, containing 145600 characters from 26 balanced classes (latin alphabet) and the EMNIST Digits dataset, containing 280000 characters from 10 balanced classes (arabic numerals).
Data preproccessing involved the partition in train ($\approx 85\%$) and test ($\approx 15\%$) sets, normalization, reshaping into mini-batches of size 125 and one-hot encoding.

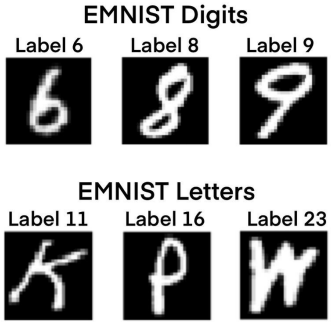

Figure 1. Some samples converted into 28x28 pixel images.

Generative unsupervised learning is performed by implementing a DBN, which is a hierarchical generative model consisting in some Restricted Boltzman Machines (RBM) staked one over the other. These are stochastic graphical neural networks in which the hidden neurons model the latent statistical structure of data observations, that are clamped to the visible neurons.
According to [3] the probability distribution over pairs of visible-hidden vectors can be defined as

$$P(v,h) = \exp(-\|v\|^2/2 + v^T b_V + h^T b_H + v^T W h)/Z. \quad (1)$$

RBMs are trained using the Contrastive Divergence algorithm that uses an approximation of the gradient

$$\frac{\delta L}{\delta W} = < V \cdot H^T >_{P(H|V)\tilde{P}(V)} - < V \cdot H^T >_{P(H,V)} \quad (2)$$

where

$$L = \frac{1}{|Tr|} \sum_{v \in Tr} \log(P(v)). \quad (3)$$

Training is carried out in two phases: correlation $v \cdot h^T$ is computed during the positive phase on training data, while correlation $v' \cdot h'^T$ is computed during the negative phase according to the model's expectation. The difference between these correlations gets smaller over learning time. The weight updating rule

$$\Delta W = \epsilon(v \cdot h^T - v' \cdot h'^T) \quad (4)$$

is made more efficient by applying Momentum, that accellerates learning in case of noisy or small gradients [4]. The idea is to accumulate an exponential moving average of past gradient

$$\nu \leftarrow \beta\nu + \epsilon\frac{\delta L}{\delta W} \quad (5)$$

where β is the momentum parameter, and keep moving in that direction:

$$W \leftarrow W + \nu. \quad (6)$$

The training of the whole DBN proceeds bottom-up, starting from the first RBM and continuing one layer at a time, for 30 epochs; while training a layer the weights of lower layers are freezed.
To allow a more intuitive interpretation of each neuron representation, sparseness is forced on the last hidden layer [5].

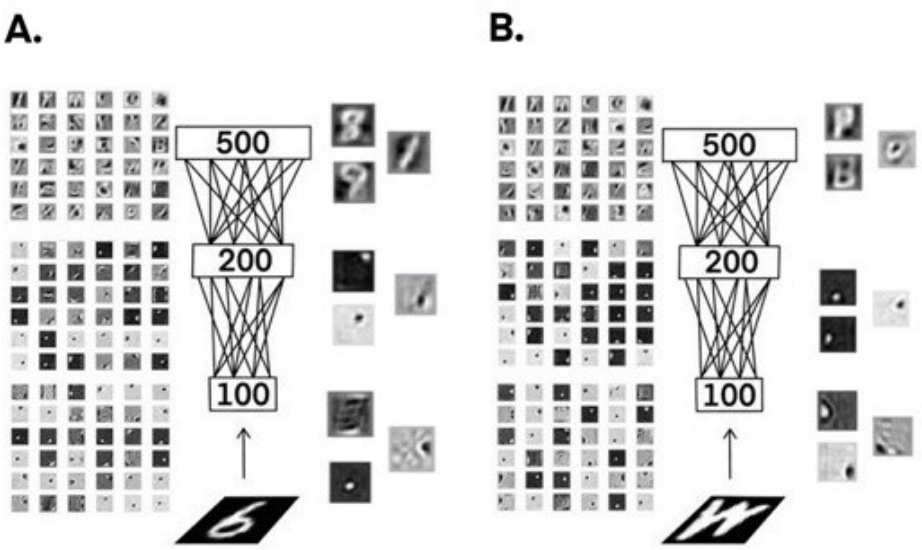

Figure 2. Receptive fields of neurons in the layers of the hierarchy: each square represents the region of sensory space that activates the neuron. [A] EMNIST Digits [B] EMNIST Letters.

An output layer is added to the deepest layer of the DBN to carry out a classification task. This only needs a linear read-outs because the reconstructed images are high-level representations of the data; in fact the layers of the hierarchy learn increasingly complex and abstract features of the data.
Since many hidden units should encode more specific data characteristics [5], a gradually increasing architecture was selected, with a maximum of 500 neurons in the last layer due to feasibility constraints (codes were runned with CPU on Matlab, Python was used just for graphical representations).

3. Internal representation

As shown in Figure 2 for both datasets the network developed center-sorround detectors at the first layer, that become less blurred in the second and encode even more complex features in the third: the shape of digits/letters clearly emerges in some receptive fields of the highest layer neurons.
This is consistent with what Figure 3 shows: internal feature representation of the data is visualized with a scatterplot that becomes less blurred and more clustered as going up in the hierarchy.
Scatterplots were generated with the use of sklearn.manifold.TSNE[1] on 5000 samples.

A clear improvement emerges from the comparison between the representation at raw level and third level, expecially for digits dataset where the network achieved a better accuracy (see next section).
To understand the meaning of these plots, Figure 4 focuses on locally nearby points from EMNIST Digits. These points were identified by means of a pointer on the scatterplots, implemented using bokeh.models.plots[2].
A further feature representation at the highest level of the hierarchy is given in Figure 5 that was obtained using scipy.cluster.hierarchy[3]. The dendogram shows that digits 8-3-5 and 7-4-9 shares some visual features, and this is made more clear in the clusters of Figure 6 that was obtained using sklearn.cluster.AgglomerativeClustering[4]. According to [2] the same numbers are also commonly confused by humans.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b

[2] https://docs.bokeh.org/en/latest/docs/reference/models/plots.html

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html
https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/

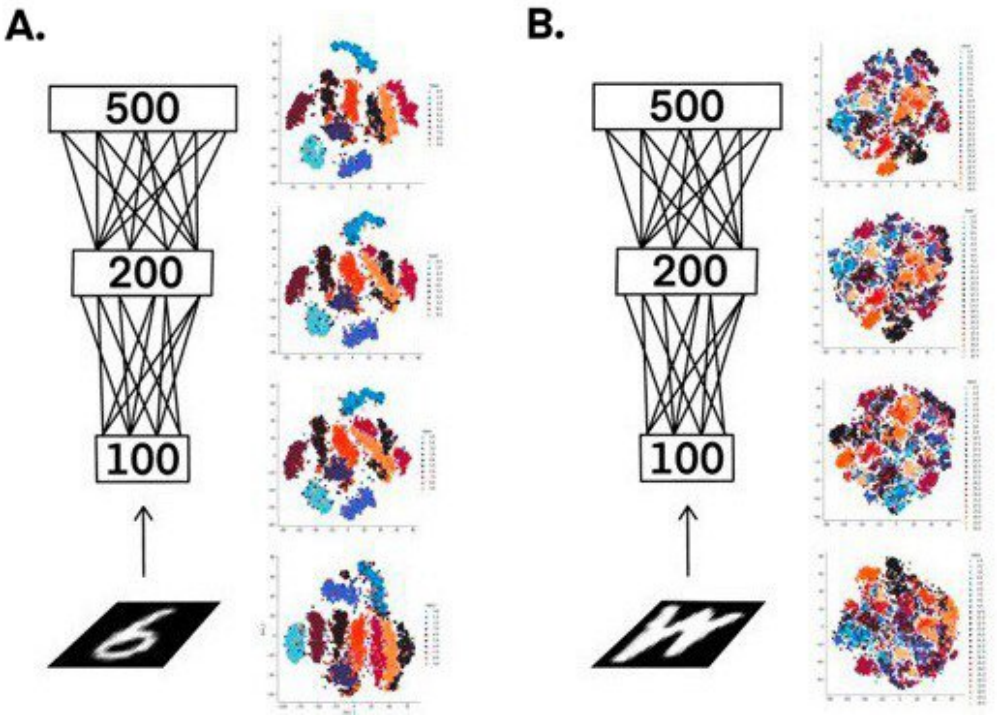

Figure 3. t-Distributed Stochastic Neighbor Embedding: dimensionality reduction (from 784 to 2) by minimizing the divergence between the pairwise similarity distributions of the input and of the corresponding low-dimensional points in the embedding [6]. [A] EMNIST Digits [B] EMNIST Letters.

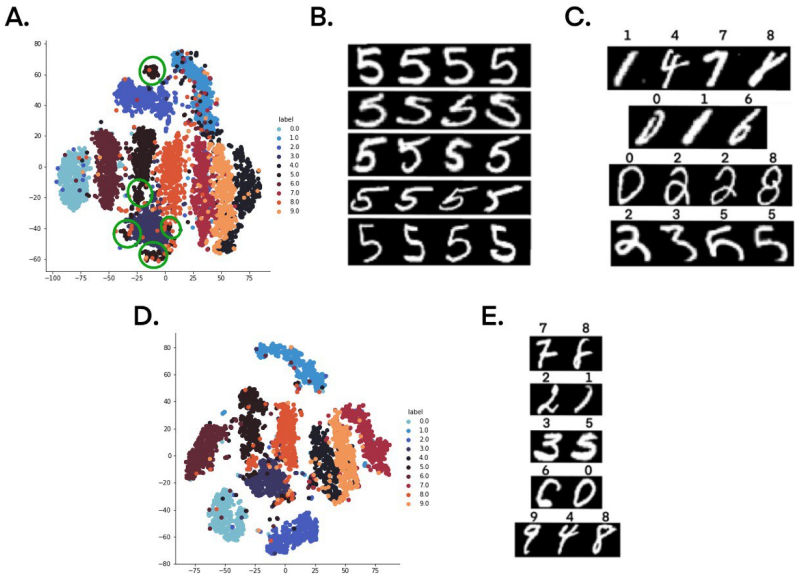

Figure 4. Similar images tend to be grouped together in the scatterplot. To see that, some digits that are near in the scatterplots are reported in the panels in the same row. A,B and C refer to raw images, D and E refer to DBN reconstructions at third layer. [A] At raw level some clusters of 5s, indicated with green circles and reported in [B], are clearly separated and [C] there's a naive association of shapes. [D] At the third level of the hierarchy there's a more complex feature representation, in fact the previous clusters of 5s are grouped togheter and [E] digits with different labels are associated only if there exists a strong similarity of shapes.

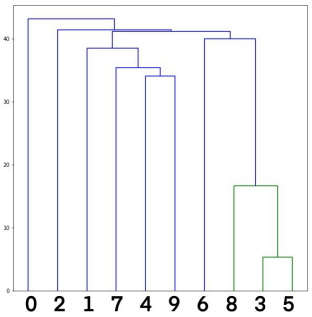

Figure 5. The dendrogram illustrates how each cluster is composed by drawing a U-shaped link between a cluster and its children, where the length of the links represents the Euclidean distance between the child clusters.

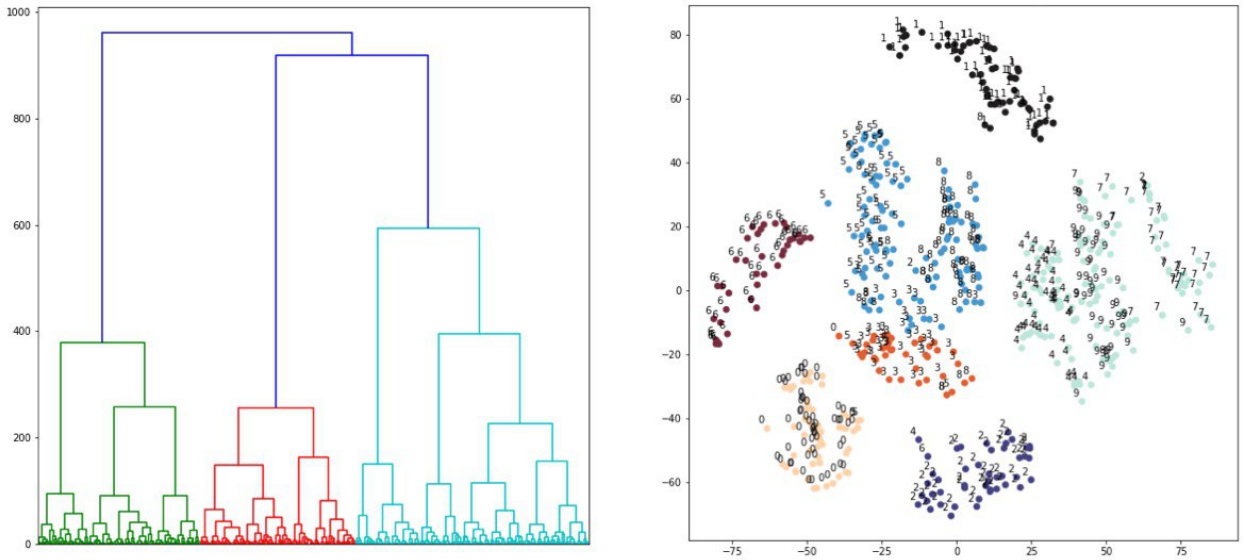

Figure 6. Dendogram on the left considered 500 samples rather than 10, and the scatterplot on the right hilights 7 clusters obtained by recursively merging the pair of clusters that minimally increases a given linkage distance in the dendogram.

4. Model evaluation

A linear read-outs is performed at different levels of the hierarchy to investigate whether representations become increasingly more disentangled at the deepest layers. See results in Figure 7.
For both datasets, the use of pixels (raw data) leads to a relatively low accuracy, because the perceptron tries to directly classify the data, which may not be linearly separable. When higher-level representations are used to feed the perceptron, accuracy increases because the DBN was able to extract key features from the data.
Digits are almost linearly separable and the number of train samples is almost twice the one of letters, while the number of classes is about three times less the one of letters. Therefore the performance on letters is quite inferior and further analisys will be conducted considering only the more robust model trained and tested on digits.

From Figure 8 emerges that the model makes errors that are consistent with its internal representation: 5s are often classified as 3s, 9s as 4s (and vice versa) and 7s ad 9s. These errors are often similar to what one would expect from a human observer (see Figure 9).

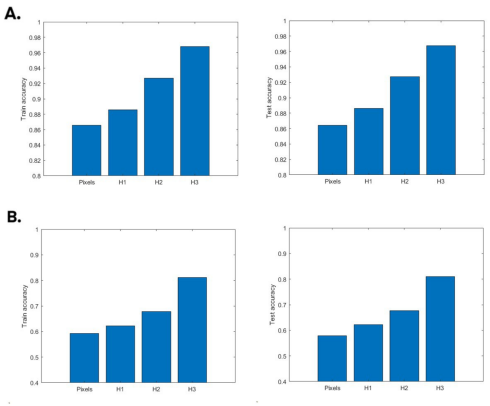

Figure 7. Train and test accuracy for [A] EMNIST Digits and [B] EMNIST Letters.

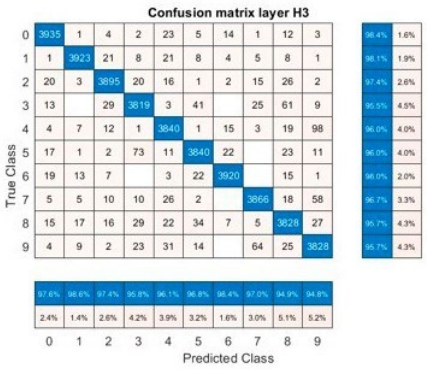

Figure 8. Third layer confusion matrix for digits.

Figure 9. Some wrong predictions.

When dealing with raw data, accuracy decreases to the chance level even with a small amount of noise (Gaussian, mean 0, sd 0.05) in the test set. Notably, the DBN is robust to noise: feeding the perceptron with the DBN reconstruction of the test set with a non indifferent amount of noise (Gaussian, mean 0.1, sd 0.1) leads to a relatively good accuracy (see Figure 10) considering that such level of perturbation would make the task difficult even for a human observer.

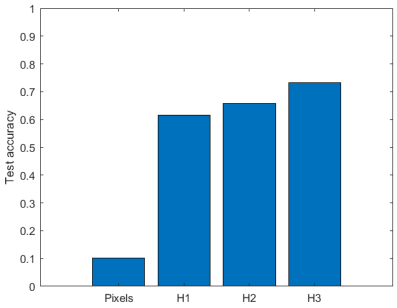

Figure 10. Test accuracy with noise.

5. Data augmentation

Data augmentation of a neural network's train set has a regularization effect that can improve generalization performance [1].
The transformations chosen to augment the train set, from 240000 to 398000 samples, are:

- Gaussian noise (mean 0, sd 0.05) on 50000 samples
- Salt and Pepper noise (density 0.05) on 50000 samples
- Rotation of ±10 degrees, each on 25000 samples
- Rotation of 180 degrees on 2000 samples labelled as 6 (to obtain 9) and on 2000 samples labelled as 9 (to obtain 6)
- Rotation of 180 degrees on 1000 samples of 8 and 1000 samples of 0 (for symmetry labels don't change)
- Reflection on y axis of 1000 samples of 8 and 1000 samples of 0 (for symmetry labels don't change).

Then, the network was tested on the original version of the test set and some manipulated versions (less than a half of the original set was manipulated). To challenge the model, noise injected during testing (Gaussian, mean 0, sd 0.08 and Salt&Pepper, density 0.08) was higher than the one used for train data augmentation. Results are reported in Tables 1 and 2.
Althoug a slighty decreasing of the accuracy can be observed on the original test set, the generalization error on manipulated versions is lower.

Manipulations in Test set	Accuracy
No	96.4
Gaussian+Salt&Pepper noise	94.1
Rotation 10 degrees	94.8
Rotation 20 degrees	92.6
All of the previous	86.3

Table 1. Results without data augmentation.

Noise injection in Test set	Accuracy
No	96.0
Gaussian+SaltAndPepper noise	95.4
Rotation 10 degrees	95.6
Rotation 20 degrees	94.4
All of the previous	89.4

Table 2. Results with data augmentation.

6. Adversarial attack

It's possible to fool a deep network by injecting a small percentage of noise in the data such as the image looks almost the same to the human eyes. Noise isn't random: the image is modified in the direction of the gradient that maximizes the loss function with respect to the input image.
The following analisys focuses on the Fast Gradient Sign Method Attack [5]:

$$adv\_x = x + \epsilon \cdot sign(\nabla_x L(W, x, y)) \tag{7}$$

where x and y are the original input image and label. The gradient was approximated as in the Appendix and two test sets were considered: train set with Adversarial noise and train set with Gaussian noise. Taking into account that the accuracy reached on the original train set is 95.6, the results in Table 3 show a weak resistance to Adversarial attacks: even if to human eyes Gaussian noisy images seem more perturbated than Adversarial images (see Figure11), the model achieved much less accuracy in classifying the latter.

Test set	Accuracy
Adversarial ($\epsilon = 0.1$)	88.9
Gaussian (mean 0, sd 0.01)	94.9
Adversarial ($\epsilon = 0.2$)	54.5
Gaussian (mean 0, sd 0.1)	70.4

Table 3. Effect of adversarial attacks.

[5] https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

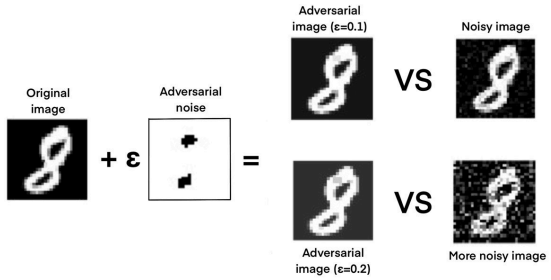

Figure 11. Example of adversarial and noisy images.

References

[1] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization, 2008.

[2] Matthew Grissinger. Misidentification of alphanumeric symbols plays a role in errors, 2009.

[3] Ilya Sutskever, Geoffrey Hinton, and Graham Taylor. The recurrent temporal restricted boltzmann machine, 2008.

[4] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning, 2013.

[5] Alberto Testolin, Ivilin Stoianov, and Marco Zorzi. Modeling language and cognition with deep unsupervised learning: a tutorial overview, 2017.

[6] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne, 2008.

7. Appendix

```
tr_patt = inputdata;
H1_tr = 1./(1 + exp(-tr_patt*DN.L{1}.vishid -
        repmat(DN.L{1}.hidbiases, size(tr_patt,1),1)));
H2_tr = 1./(1 + exp(-H1_tr*DN.L{2}.vishid -
        repmat(DN.L{2}.hidbiases, size(H1_tr,1),1)));

% Gradient of H2_tr wrt tr_patt:
g = exp(-H1_tr*DN.L{2}.vishid - repmat(DN.L{2}.hidbiases, size(H1_tr,1),1));
g = g./(1+g).^2;          %240000x500
g = g*DN.L{2}.vishid.'; %240000x500 * 500x100
g = g*DN.L{1}.vishid.'; %240000x100 * 100x784
s = sign(g);

adv = zeros(size(inputdata));
for i = 1:size(inputdata,1)
        adv(i,:)= inputdata(i,:)+0.1*s(i,:);
end
```

8. Google Drive link to source code

https://drive.google.com/drive/folders/1nE3IDuBk7wR7SaLTip0MrtaHy7TjHkbW?usp=sharing