Neural Networks
Machine learning

Alexandre Sarazin & Kevin Serrano
Robotics Engineering
University degli Studi di Genova
Ecole Centrale de Nantes
Email: alexandre.sarazin@eleves.ec-nantes.fr
kevin.serrano-vilchis@eleves.ec-nantes.fr

***Abstract*—The purpose of this work is to become familiar with the Matlab Neural Network Toolbox for data fitting and pattern classification problems using Shallow Neural Networks and the autoencoder feature.**

1. Introduction

An autoencoder is an artificial neural network used for unsupervised learning of efficient codings. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Recently, the autoencoder concept has become more widely used for learning generative models of data.

Architecturally, the simplest form of an autoencoder is a feedforward, non-recurrent neural network very similar to the multilayer perceptron (MLP) having an input layer, an output layer and one or more hidden layers connecting them. It is trained using the same pattern as both the input and the target. So for instance input pattern $x(l, :)$ has target $t(l, :) = x(l, :)$.

The purpose of an autoencoder is to reconstruct its own inputs. It learns an internal, compressed representation for the data. Therefore, autoencoders are unsupervised/self-supervised learning models.

The dataset is composed of the 'Semeion Handwritten Digit data' wich is composed by 1593 handwritten digits from around 80 persons. The digits have been scanned and stretched in a rectangular box of 16x16 in a binary scale. All the data used in this work is available online at this page: https://dibris.aulaweb.unige.it/course/view.php?id=936

The goal of this work is to, given a binary handwritten digit data, retrieve if the observation belongs to a chosen digit class or not.

The report is structured as follow: the first part presents the concepts tackled in the Matlab tutorial, the second part deals with the autoencoder, and the third - last, part concludes. Because parts are quite different, the results are presented throughout the report and do not have a separate part.

2. Matlab tutorial

2.1. Task 0: How to fit data with a shallow neural network

In the first task, we are going to learn how to fit data with a shallow neural network.

The tutorial is available here: https://it.mathworks.com/help/nnet/gs/fit-data-with-a-neural-network.html.

In fitting problems, we want a neural network to map between a data set of numeric inputs and a set of numeric targets.

Examples of this type of problem include estimating house prices from such input variables as tax rate, pupil/teacher ratio in local schools and crime rate, estimating engine emission levels based on measurements of fuel consumption and speed or predicting a patient's body fat level based on body measurements.

We first load an predefined data set, for example the 'House pricing' data set, used to train a neural network to estimate the median house price in a neighborhood based on neighborhood statistics.

The neighborhood statistics, which are the inputs in our case, are as follow:

1) Per capita crime rate per town
2) Proportion of residential land zoned for lots over 25 000 sq. ft.
3) Proportion of non-retail business acres per town
4) 1 if tract bounds Charles river, 0 otherwise
5) Nitric oxides concentration (parts per 10 millions)
6) Average number of rooms per dwelling
7) Proportion of owner-occupied units built prior to 1940
8) Weighted distances to five Boston employment centers
9) Index of accessibility to radial highways
10) Full-value property-tax rate per 10 000$
11) Pupil-teacher ratio by town
12) $1000 * (Bk - .63)^2$, where Bk is the proportion of black coloured people by town
13) Percent lower status of the population

The output is simply a median values of owner-occupied homes in each neighborhood in 1000's of dollars.

Then we split the data set by defining the percentage of the data set to use for the training, for the validation and finally for the training.

The training set is presented to the network during training, and the network is adjusted according to its error.

The validation set is used to measure network generalization, and to halt training when generalization stops improving.

The test set has no effect on training and so provide an independent measure of network performance during and after training.

In the example we use the following repartition: 70% training set, 15% validation set and 15% test set.

The standard network used for function fitting is a two-layer feedforward network, with a sigmoid transfer function in the hidden layer and a linear transfer function in the output layer (fig.1). The default number of hidden neurons is set to 10. We may increase the number of hidden neurons if the network training performance is poor. The number of output neurons is set to 1, which is equal to the number of elements in the target vector, that is to say the median house price in our example.

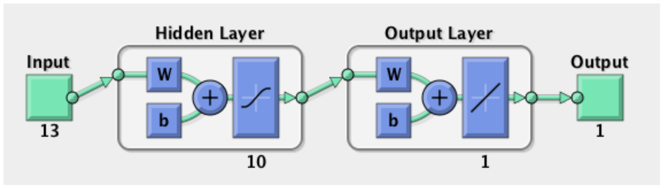


Figure 1: Two-layer feedforward neural network

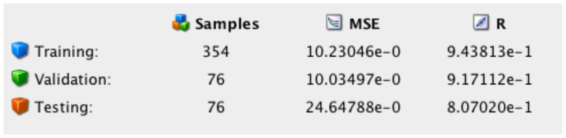
The network is train using Levenberg-Marquardt algorithm. This algorithm typically requires more memory but less time. Generally speaking, Levenberg-Marquardt (trainlm) is recommended for most problems, but for some noisy and small problems Bayesian Regularization (trainbr) can take longer but obtain a better solution. For large problems, however, Scaled Conjugate Gradient (trainscg) is recommended as it uses gradient calculations which are more memory efficient than the Jacobian calculations the other two algorithms use.

Training automatically stops when generalization stops improving, as indicated by an increase in the mean square error of the validation samples.

If the results are not satisfying, the network can be retrain, this will change the initial weights and biases of the network.

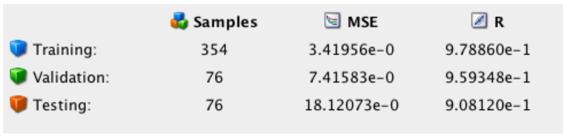
Typically, the training is satisfying for a R values in each case of 0.93 or above.

In our case, the first case is not satisfying. See fig.2. After retraining several times we get better results but still they are not satisfying (fig.3).



	Samples	MSE	R
Training:	354	10.23046e-0	9.43813e-1
Validation:	76	10.03497e-0	9.17112e-1
Testing:	76	24.64788e-0	8.07020e-1

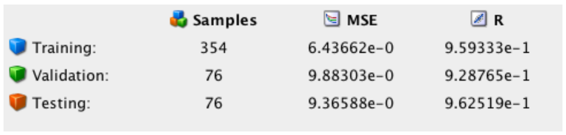
Figure 2: Result after first training



	Samples	MSE	R
Training:	354	3.41956e-0	9.78860e-1
Validation:	76	7.41583e-0	9.59348e-1
Testing:	76	18.12073e-0	9.08120e-1

Figure 3: Result after several training

We decided to increase the number of hidden neurons to 17 to improve the performances. The 17 hn networks gets indeed better results with higher overall values for R (see fig.4).



	Samples	MSE	R
Training:	354	6.43662e-0	9.59333e-1
Validation:	76	9.88303e-0	9.28765e-1
Testing:	76	9.36588e-0	9.62519e-1

Figure 4: Result with 17 hidden neurons network

The training is, in this case, satisfying.

The regression plot is used to validate the network performance. The following regression plots (see fig.5) display the network outputs with respect to targets for training, validation, and test sets. For a perfect fit, the data should fall along a 45 degree line, where the network outputs are equal to the targets. For this problem, the fit is reasonably good for all data sets, with R values in each case of 0.928 or above.

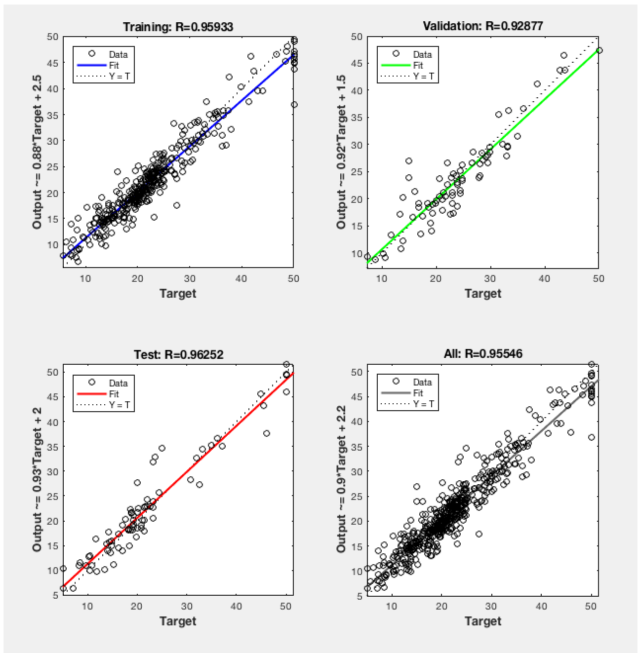


Figure 5: Regression plot for 17 hn network

The error histogram gives additional verification of network performance, see fig.6.

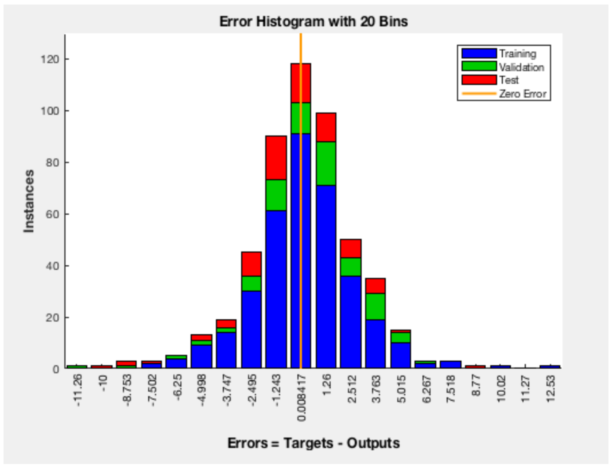


Figure 6: Error histogram for 17 hn network

The blue bars represent training data, the green bars represent validation data, and the red bars represent testing data. The histogram can give us an indication of outliers, which are data points where the fit is significantly worse than the majority of data. In this case, we can see that while most errors fall between -10 and 10, there is a training point with an error of 12.53 and validation points with errors of -11.26. These outliers are also visible on the testing regression plot. It is a good idea to check the outliers to determine if the data is bad, or if those data points are different than the rest of the data set. If the outliers are valid data points, but are unlike the rest of the data, then the network is extrapolating for these points. We should collect more data that looks like the outlier points, and retrain the network.

In our case, the outlier points are reasonably not too far from the majority of data so we consider them as acceptable.

Through the task, we learned how to use a shallow neural network to fit data with the Matlab Neural Network Toolbox. The GUI makes the practical application very simple, it is also possible to use the command line tool which looks similarly simple to use. We obtained very good results with the median house price estimation based on neighborhood statistics by training a 17 hidden neurons network.

2.2. Task 1: Classify patterns with a shallow neural network

In the second task (Task 1), we are going to learn how to classify patterns with a shallow neural network.

The tutorial is available here: https://it.mathworks.com/help/nnet/gs/classify-patterns-with-a-neural-network.html.

More informations related to the Neural Network Toolbox are given here https://it.mathworks.com/help/nnet/gs/shallow-networks-for-pattern-recognition-clustering-and-time-series html and here https://it.mathworks.com/help/nnet/ug/multilayer-neural-networks-and-backpropagation-training.html.

In pattern recognition problems, we want a neural network to classify inputs into a set of target categories.

For example, recognize the vineyard that a particular bottle of wine came from, based on chemical analysis, or classify a tumor as benign or malignant, based on uniformity of cell size, clump thickness, mitosis.

In this task, we are using the 'Wine Vintage' data set. This dataset is used to create a neural network that classifies wines from three winerys in Italy based on constituents found through chemical analysis.

The 'Wine Vintage' inputs are:

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of Ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoids phenols
9) Proanthocyanins
10) Color intensity
11) Hue
12) OD280/0D315 of diluted wines
13) Proline

The 'Wine Vintage' outputs/targets is a vector of dimension 3 indicating which vineyard the wine come from ('Vineyard #1', 'Vineyard #2', 'Vineyard #3'). 1 if the wine come from the considered vineyard, 0 otherwise.

In the example we use the following repartition: 70% training set, 15% validation set and 15% test set.

The standard network that is used for pattern recognition is a two-layer feedforward network, with a sigmoid transfer function in the hidden layer, and a softmax transfer function in the output layer (fig.7). The default number of hidden neurons is set to 10. We may increase the number of hidden neurons if the network training performance is poor. The number of output neurons is set to 3, which is equal to the number of elements in the target vector (the number of categories).

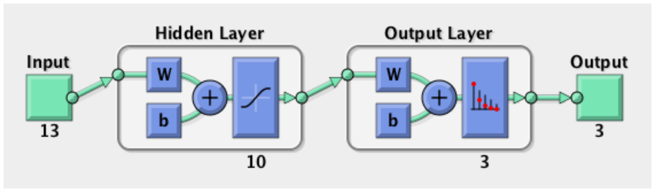
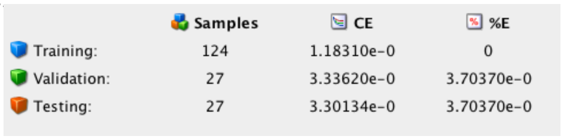


Figure 7: Two-layer feedforward neural network

The network is train using scaled conjugate gradient backpropagation.

Training automatically stops when generalization stops improving, as indicated by an increase in the cross-entropy error of the validation samples.

The results are shown in fig.8



	Samples	CE	%E
Training:	124	1.18310e-0	0
Validation:	27	3.33620e-0	3.70370e-0
Testing:	27	3.30134e-0	3.70370e-0

Figure 8: Result

Some definitions to understand the results.

- CE: Minimizing Cross-Entropy results in good classification. Lower values are better. Zero means no error.
- %E: Percent Error indicates the fraction of samples which are miss-classified. A value of 0 means no miss-classifications, 100 indicates maximum miss-classifications.

Here the CE is quite low, under 1.2 for training and under 3.4 for validation and testing. %E is equal to zero for training and under 3.71% for validation and testing. The results for these indexes are satisfying.

The confusion matrix is used to validate the network performance. The figure 9 shows the confusion matrices for training, testing, and validation, and the three kinds of data combined. The network outputs are very accurate, as we can see by the high numbers of correct responses in the green squares and the low numbers of incorrect responses in the red squares. The lower right blue squares illustrate the overall accuracies. The obtained confusion matrices' coefficients are satisfying: training accuracy of 100%, validation accuracy of 96.3%, test accuracy of 96.3% and overall accuracy of 98.9%.

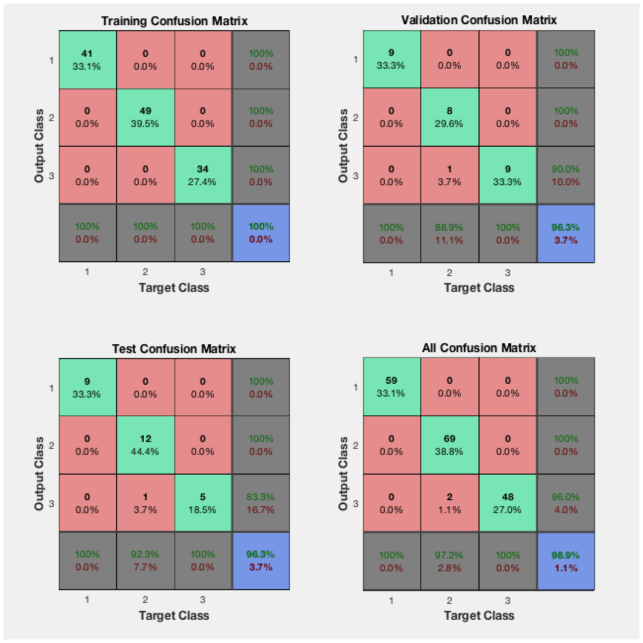


Figure 9: Confusion matrices

The Receiver Operating Characteristic (ROC) curves (fig.10) are also used to validate the network performance.

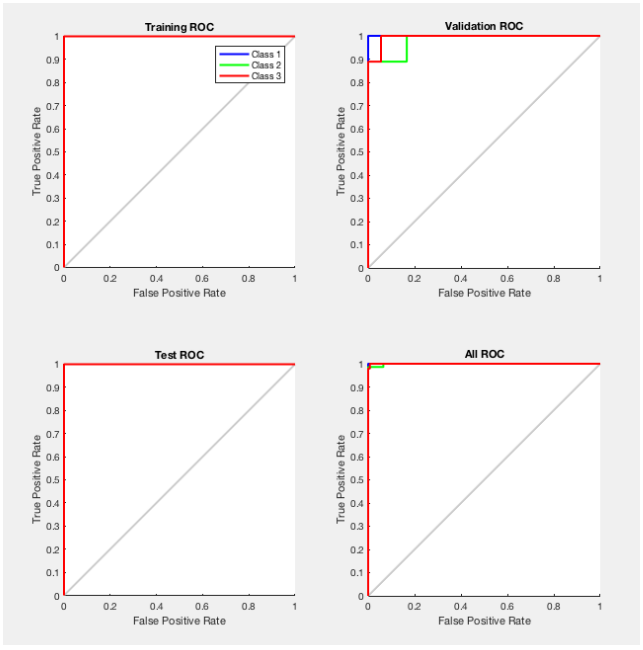


Figure 10: Receiver Operating Characteristic

The colored lines in each axis represent the ROC curves. The ROC curve is a plot of the true positive rate (sensitivity) versus the false positive rate (1 - specificity) as the threshold is varied. A perfect test would show points in the upper-left corner, with 100% sensitivity and 100% specificity. For our problem, the network performs very well.

Through this task, we learned how to use a shallow neural network to classify patterns with the Matlab Neural Network Toolbox. We obtained very good results by classifying wines to there corresponding vineyard.

3. Autoencoder

In this task, an interesting neural network architecture is presented: the autoencoder or *autoassociator*. Basically, it is an MLP where there are as many outputs as inputs, and the required outputs are defined to be equal to the inputs. Mathematically speaking, the autoencoder tries to learn a function $h_{W,b}(x) \approx x$. In other words, it is trying to learn an approximation to the identity function. This would seem a trivial task, however, by applying constraints on the network (such as limiting the number of hidden units), we can discover interesting structure about the data. Once the MLP is trained, the first layer (from the input to the hidden layer) acts as an encoder, and the values of the hidden units make up the encoded representation. The second layer, which goes from the hidden layer to the output, acts as a decoder.

MATLAB already counts with an implemented `Autoencoder` class, therefore out goal is to use it, tweak available parameters, and analyze the results. A few things to be mentioned about this implementation:

- Activation functions:
  - sigmoid $f(z) = (1 + e^{-z})^{-1}$.

- saturation $f(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{if } 0 < z < 1 \\ 1 & \text{if } 1 \leq z \end{cases}$

- Loss function:

$$E = \underbrace{\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}(x_{kn} - \hat{x}_{kn})^2}_{\text{mean squared error}} + \lambda \cdot \underbrace{\Omega_{weights}}_{\text{L2 regularization}} + \beta \cdot \underbrace{\Omega_{sparsity}}_{\text{sparsity regularization}}$$

- Parameters to tune:
  - λ: coefficient for L2 regularization
  - β: coefficient for sparsity regularization
  - `SparsityProportion`: Desired proportion of training examples a neuron reacts to.

Although the loss function may be slightly out of the scope of this report, we'll briefly describe its main components. As usual, we want to minimize the loss function and the first element of this function is the mean squared error (MSE) whose function is to drive the output towards a desired one. Then we find the L2 regularization term which is mainly used to avoid the over-fitting problem, a high λ will result in under-fitting and vice versa. Lastly, we have the sparsity regularization which, in MLP, is used as a constrain to make neurons inactive most of the time, leading to *specialized* neurons in the hidden layer by only giving a high output for a small number of training examples.

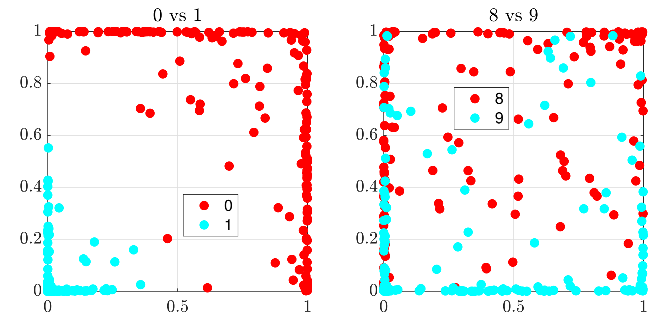
Several tests were conducted using the following structure:

1) Split the semeion-data into subsets of different classes.
2) Create a training set with only 2 classes.
3) Train an autoencoder on the reduced training set.
4) Plot the data

We'll be using only 2 neurons in the hidden layer since 2D data is easier to visualize, however, this implies dimensionality reduction from 256 to 2. The 2D encoded data (using default MATLAB values) can be visualized in figure 11 comparing 0 vs 1 (highly distinguishable digits) and 8 vs 9 (prone to confusion). Moreover, the sigmoid activation was selected as default for all our experiments since its output can be interpreted as a *probability*.

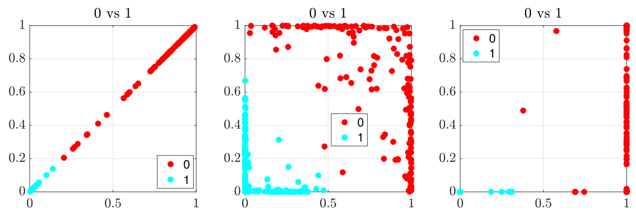
3.1. L2 regularization

How does the L2 regularization coefficient affect the encoder's performance? As previosuly mentioned, regularization allows us to prevent over and under-fitting. Since we're only using 2 hidden neurons, we can *easily* analyze this problem; under-fitting would imply that only a few parameters (weights) are taken into account, thus yielding an approximate linear solution, whereas over-fitting imply *specialized* parameters and extremely non-linear solution.



(a) Clear division between classes (b) Different classes overlap

Figure 11: Encoded data using 2 hidden nuerons

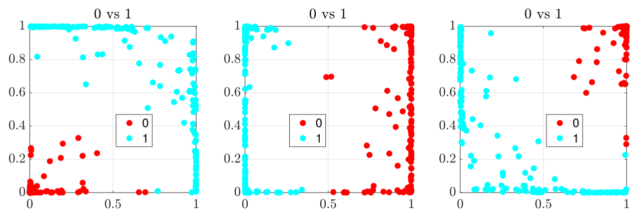


(a) under-fitting (b) mid-ground (c) over-fitting

Figure 12: L2 regularization impact on encoded data.

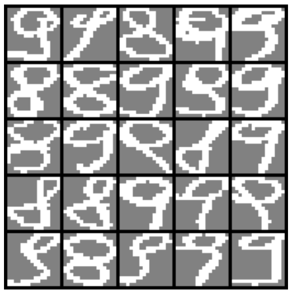
3.2. Sparsity regularization

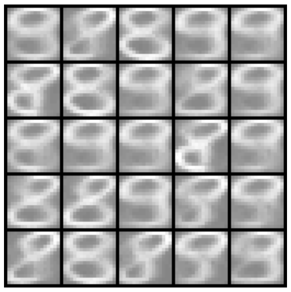
Since this method constraints the average activation of each neuron, we must give a suitable value for it, otherwise we risk obtaining a high cost configuration since the training algorithm is unable to satisfy or optimize the given constraints. In the ideal scenario, we would like one neuron to activate each time it sees one class and deactivate when seeing the other; if classes are balanced, our activating average for each neuron would be around 0.5. In figure 13 we notice that, despite the sparsity proportion, the difference between classes is still evident. Nontheless, when SP = 0.5 the encoded data is evenly separated.

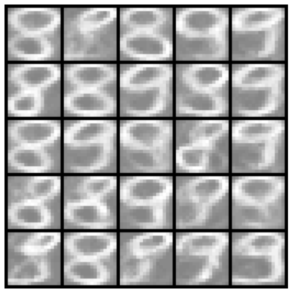


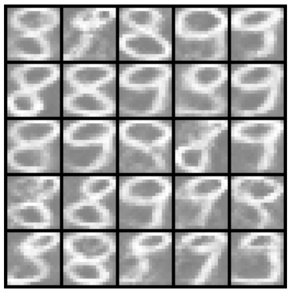
(a) SP = 0.1 (b) SP = 0.5 (c) SP = 0.9

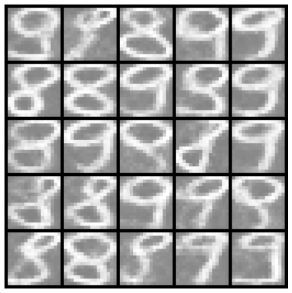
Figure 13: Sparsity Proportion (SP) impact on encoded data.


(a) Random samples


(b) nh = 3, MSE = 0.167


(c) nh = 10, MSE = 0.129


(d) nh = 25, MSE = 0.097


(e) nh = 50, MSE = 0.077

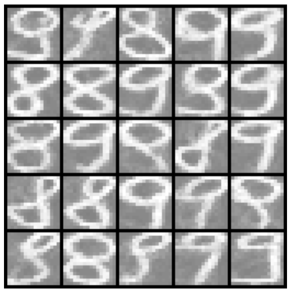

(f) nh = 100, MSE = 0.060

Figure 14: Reconstructed data varying number of hidden neurons (nh).

3.3. Number of hidden neurons

The previous results show the response of the autoencoder to different parameter configurations and this can be extrapolated to higher dimensions. Now we would like to see how the different number of hidden neurons affect the performance and since high dimensional is not easy to visualize, we'll plot the output image instead and see how close the autoencoder gets to the identity $h_{W,b}(x) \approx x$. In other words, we'll measure the MSE between the original and the reconstructed data.

As it is expected, the error decreases as we increment the number of neurons in the hidden layer. On the other hand, already with 10 neurons we obtain a good error response and increasing the number of hidden neurons doesn't drastically change the final output. This suggests that there exist some structure in the data where 10 neurons are enough to represent most of its variance.

4. Conclusion

The presented work tackled data fitting and pattern classification problems using Shallow Neural Networks as well as the design of an autoencoder in order to reconstruct its own input. We obtained very good results for the data fitting problem with the median house price estimation based on neighborhood statistics using the Matlab Neural Network Toolbox. Very good results were also obtained for the pattern classification problem with the vineyards' wine classification.

Although the autoencoder can be implemented as an MLP, MATLAB makes it easier with its built-in class. The available parameters, such as l2 and sparse regularization can improve performance but won't drastically change the final output. Apart from being an encoder, it could also be used as a tool for dimensionality reduction given that it is able to learn a representation of the data using n hidden neurons.

Neural networks nowadays can appear to be a *black box*, but it is essential to have a basic understanding of how they work in order to achieve reliable results.