

# Artificial neural networks

## Assignment 1: Supervised learning and generalization

Lood, Cedric  
Master of Bioinformatics

May 9, 2016

### 1 Context

Artificial neural networks that have at least 1 hidden layer have the property of being universal approximator[1, 2]. Hence, any non-linear function can be realized using classical feedforward multilayers networks. Here, I explore different training algorithm and compare their yields in terms of performance, training time, and generalization. For the sake of simplification, I used the underlying, known function  $f(x) = e(-x^2)\sin(10x)$  to generate the dataset for batch supervised learning. The architecture of the neural network consists of 10 hidden units with sigmoid transfer function, 1 input, and 1 output (linear).

### 2 Noise-free learning

The function  $f(x)$  was learned using noiseless datapoints over the range  $[-\pi; \pi]$  by interval of  $\delta x = 0.1$ . The training algorithm selected are the ones reviewed during the lecture. Their relative performances can be appreciated by looking at the figure 1. On the left-hand side, the approximation of the different learning schemes after 100 epochs. On the right-hand side, the training error as it evolve during training.:

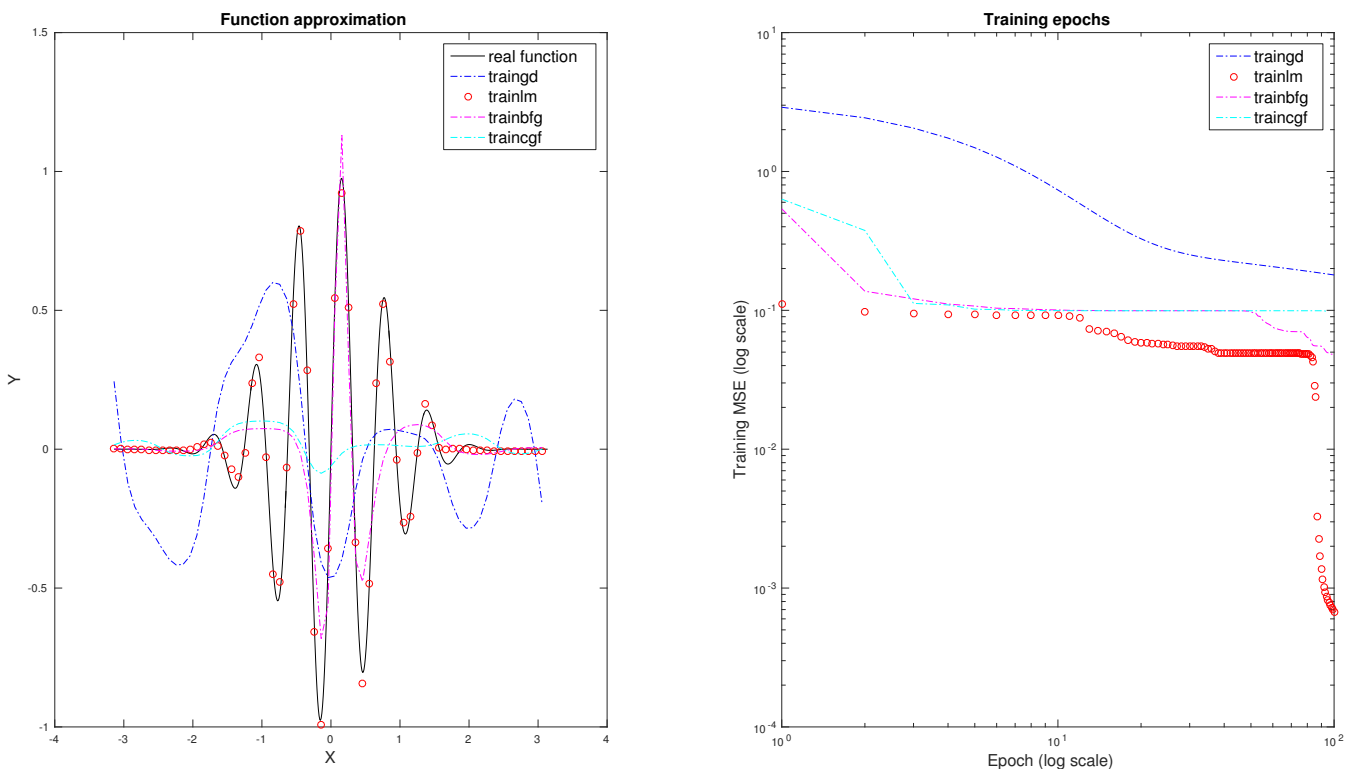


Figure 1: Training performance

**trainingd** is the gradient descent algorithm. Its performance were rather terrible in terms of function approximation. The implementation details are simple, but it suffers from slow convergence.

**trainlm** was the best performer overall with the different functions I tried to approximate. The algorithm resembles the Newton method but has an extra-term that enables solving otherwise ill-conditioned problems.

**trainbfg** is a quasi-Newton method. This algorithm avoids the computation of the Hessian matrix, and works instead with an approximation, sparing CPU cycles.

**traincgf** is a conjugate gradient approach, which can be really useful when dealing with networks that have a lot of hidden neurons and/or interconnections. This algorithm avoids storing the Hessian matrix, sparing the RAM needed otherwise to store potentially huge matrices.

### 3 Learning in noisy setting

For this section, I worked with a slightly easier function  $f(x) = x^3 - 11x + 2$ . To simulate the presence of noise in the dataset, I chose to add a normally distributed  $\epsilon$  to  $f(x)$ , for example:  $f_{noisy} = f(x) + N(0, 1)$ .

See graphic 2 where the noisy dataset was used for training. Typical overfitting can be seen when raising the value of the *epochs*, represented in the graph by the red curve, which upon closer inspection seems to be following the noise rather than fitting the underlying function. Strategies to decide when to stop training can be implemented, such as validation set, cross validation, and bootstrapping.

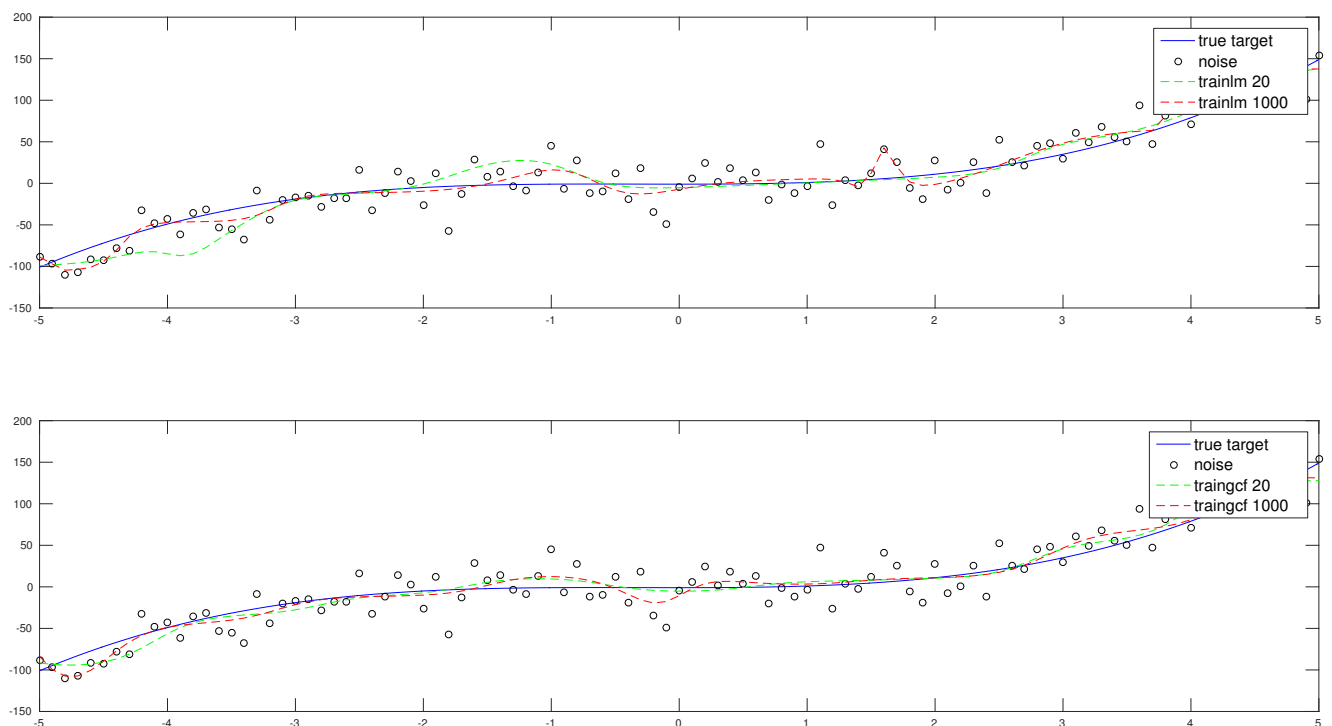


Figure 2: Example of overfitting in noisy dataset

## References

- [1] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [2] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,” *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.