Artificial neural networks

Assignment 1: Supervised learning and generalization

Lood, Cedric

April 15, 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. In this assignment, we were asked to explore the different training algorithms that can be used with backpropagation in order to learn non-linear functions. These function have different yields in terms of performance, training time, and more importantly generalization. Some comparisons of the approaches are explored in this report.
For illustration purposes, I worked with 2 functions illustrated here (graphics created using ggplot2 and R [1]):

- A simple polynomial function with equation $f(x) = x^3 - 11x + 2$ illustrated on the left of figure 1.
- A more *"wiggle-y"* function, that exhibits some dampening, with equation $g(x) = e(-x^2)sin(10x)$ illustrated on the the right of figure 1.

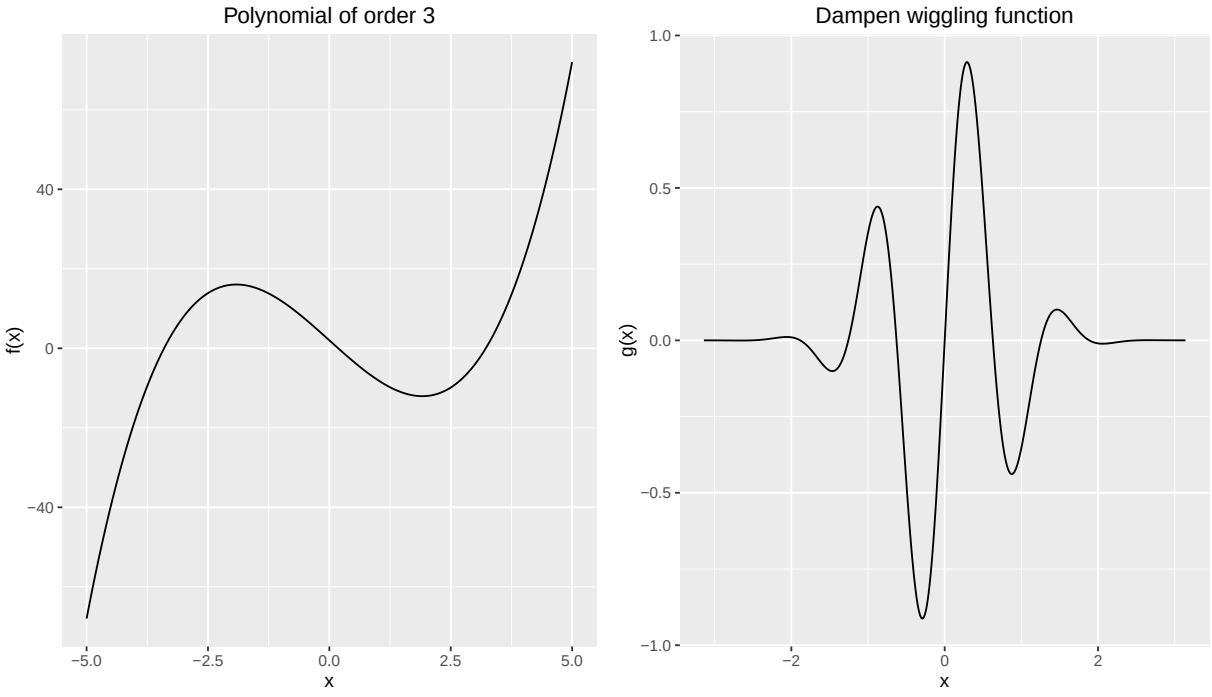

Figure 1: Graphs of the underlying true functions used in the analysis

[1] All matlab and R code can be found online at: https://github.com/milt0n/ANN-Experiments

2 Noise-free learning

In this section, we explore the learning in a supervised manner of a dataset produced by a known underlying function. The entry for the learning algorithm is thus a set of tuples (x, y) with x chosen over a given interval, and $y = f(x)$.

The architecture chosen for the feedforward multilayer neural net consisted of 1 hidden layer, with 10 neurons, and 1 neuron in the output layer. The transfer function for the hidden layer neurons is the sigmoid, and the output neuron has a linear transfer function $f(x) = x$.

On figure 2, one can see that the training of the neural net using gradient descent (*traingd* in matlab) had very poor result.

top graph that represents the polynomial function $f(x) = x^3 - 11x + 2$ performed the worst, with a problem of computation of the gradient during the training. The gradient at each *epoch* of the training becomes larger and larger, and eventually too large to represent (*NaN* in matlab), and the training fails so I kept it below 50 *epochs*.

bottom graph the gradient descent performed a little bit better on $g(x) = e(-x^2)sin(10x)$ but the performance were still not satisfying in terms of approximation. I tried raising the number of *epochs* to 10000 and 100000 but the behaviour there was that the approximation was getting closer and closer to a flat function like $y = 0$, hence traversing

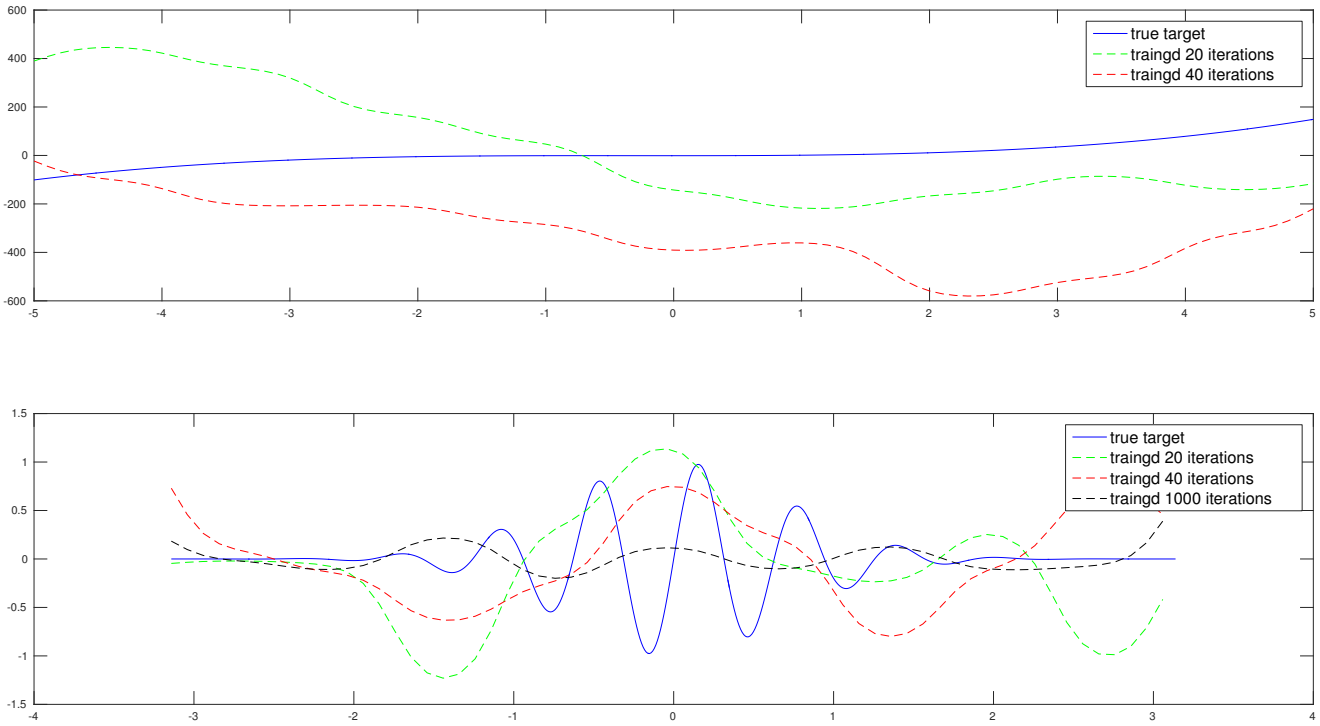

Figure 2: Training with gradient descent

On the other hand, other learning methods worked really well with the 2 functions. The results in terms of fitting and performance are illustrated in 3

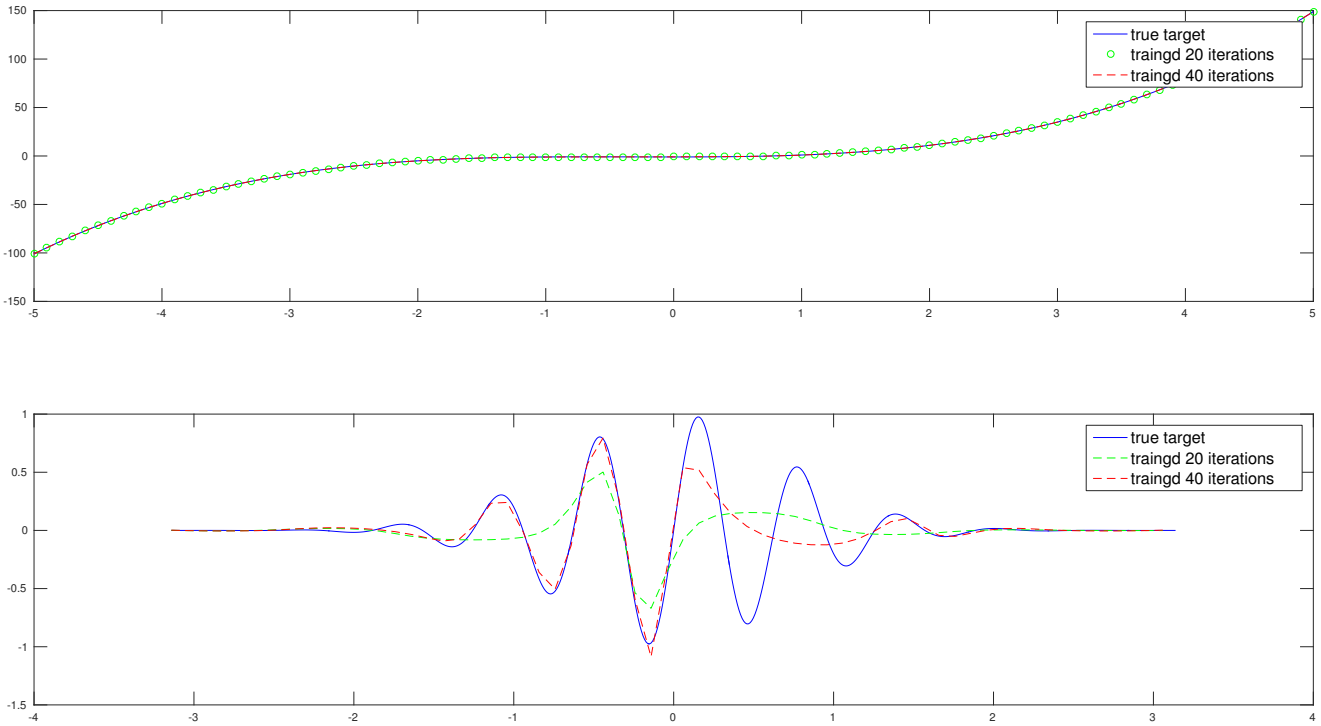

Figure 3: Training with gradient descent

3 Noisy learning

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.