1
Efficient BackProp*

Yann A. LeCun[1], Léon Bottou[1], Genevieve B. Orr[2], and Klaus-Robert Müller[3]

[1] Image Processing Research Department AT& T Labs - Research, 100 Schulz Drive, Red Bank, NJ 07701-7033, USA
[2] Willamette University, 900 State Street, Salem, OR 97301, USA
[3] GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany
{yann,leonb}@research.att.com, gorr@willamette.edu, klaus@first.gmd.de

Abstract. The convergence of back-propagation learning is analyzed so as to explain common phenomenon observed by practitioners. Many undesirable behaviors of backprop can be avoided with tricks that are rarely exposed in serious technical publications. This paper gives some of those tricks, and offers explanations of why they work.

Many authors have suggested that second-order optimization methods are advantageous for neural net training. It is shown that most "classical" second-order methods are impractical for large neural networks. A few methods are proposed that do not have these limitations.

1.1 Introduction

Backpropagation is a very popular neural network learning algorithm because it is conceptually simple, computationally efficient, and because it often works. However, getting it to work well, and sometimes to work at all, can seem more of an art than a science. Designing and training a network using backprop requires making many seemingly arbitrary choices such as the number and types of nodes, layers, learning rates, training and test sets, and so forth. These choices can be critical, yet there is no foolproof recipe for deciding them because they are largely problem and data dependent. However, there are heuristics and some underlying theory that can help guide a practitioner to make better choices.

In the first section below we introduce standard backpropagation and discuss a number of simple heuristics or tricks for improving its performance. We next discuss issues of convergence. We then describe a few "classical" second-order non-linear optimization techniques and show that their application to neural network training is very limited, despite many claims to the contrary in the literature. Finally, we present a few second-order methods that do accelerate learning in certain cases.

* Previously published in: Orr, G.B. and Müller, K.-R. (Eds.): LNCS 1524, ISBN 978-3-540-65311-0 (1998).

G. Montavon et al. (Eds.): NN: Tricks of the Trade, 2nd edn., LNCS 7700, pp. 9–48, 2012.

1.2 Learning and Generalization

There are several approaches to automatic machine learning, but much of the successful approaches can be categorized as *gradient-based learning methods*. The learning machine, as represented in Figure 1.1, computes a function $M(Z^p, W)$ where Z^p is the p-th input pattern, and W represents the collection of adjustable parameters in the system. A cost function $E^p = C(D^p, M(Z^p, W))$, measures the discrepancy between D^p, the "correct" or desired output for pattern Z^p, and the output produced by the system. The average cost function $E_{train}(W)$ is the average of the errors E^p over a set of input/output pairs called the training set $\{(Z^1, D^1), ...(Z^P, D^P)\}$. In the simplest setting, the learning problem consists in finding the value of W that minimizes $E_{train}(W)$. In practice, the performance of the system on a training set is of little interest. The more relevant measure is the error rate of the system in the field, where it would be used in practice. This performance is estimated by measuring the accuracy on a set of samples disjoint from the training set, called the test set. The most commonly used cost function is the Mean Squared Error:

$$E^p = \frac{1}{2}(D^p - M(Z^p, W))^2, \qquad E_{train} = \frac{1}{P}\sum_{p=1} E^p$$

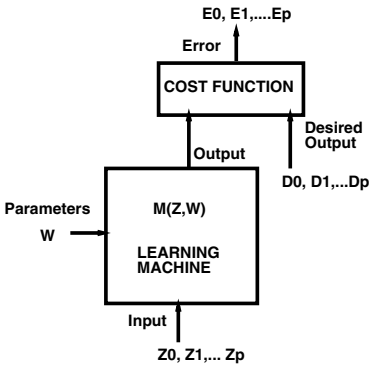

Fig. 1.1. Gradient-based learning machine

This chapter is focused on strategies for improving the process of minimizing the cost function. However, these strategies must be used in conjunction with methods for maximizing the network's ability to *generalize*, that is, to predict the correct targets for patterns the learning system has not previously seen (e.g. see chapters 2, 3, 4, 5 for more detail).

To understand generalization, let us consider how backpropagation works. We start with a set of samples each of which is an input/output pair of the function to be learned. Since the measurement process is often noisy, there may be errors in the samples. We can imagine that if we collected multiple *sets* of samples then each set would look a little different because of the noise and because of the different points sampled. Each of these data sets would also result in networks with minima that are slightly different from each other and from the