http://wwww.ira.uka.de/~riedml/publications.html

Rprop - Description and Implementation Details

Technical Report, January 1994

Martin Riedmiller
Institut für Logik, Komplexität und Deduktionssyteme
University of Karlsruhe
W-76128 Karlsruhe
FRG
riedml@ira.uka.de

I. Rprop

A. General Description

Rprop stands for '**R**esilient back**prop**agation' and is a local adaptive learning scheme, performing supervised batch learning in multi-layer perceptrons. For a detailed discussion see also [1], [2], [3]. The basic principle of Rprop is to eliminate the harmful influence of the size of the partial derivative on the weight step. As a consequence, only the sign of the derivative is considered to indicate the *direction* of the weight update. The *size* of the weight change is exclusively determined by a weight-specific, so-called 'update-value' $\Delta_{ij}^{(t)}$:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\Delta_{ij}^{(t)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0 & , \text{ else} \end{cases} \quad (1)$$

where $\frac{\partial E}{\partial w_{ij}}^{(t)}$ denotes the summed gradient information over all patterns of the pattern set ('batch learning').

It should be noted, that by replacing the $\Delta_{ij}(t)$ by a constant update-value Δ, equation (1) yields the so-called 'Manhattan'-update rule.

The second step of Rprop learning is to determine the new update-values $\Delta_{ij}(t)$. This is based on a sign-dependent adaptation process, similar to the learning-rate adaptation in [4], [5].

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ * \Delta_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- * \Delta_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \Delta_{ij}^{(t-1)} & , \text{ else} \end{cases} \quad (2)$$

$$\text{where } 0 < \eta^- < 1 < \eta^+$$

In words, the adaptation-rule works as follows: Every time the partial derivative of the corresponding weight w_{ij} changes its sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update-value $\Delta_{ij}^{(t)}$ is decreased by the factor η^-. If the derivative retains its sign, the update-value is slightly increased in order to accelerate convergence in shallow regions. Additionally, in case of a change in sign, there should be no adaptation in the succeeding learning step. In practice, this can be achieved by setting $\frac{\partial E}{\partial w_{ij}}^{(t-1)} := 0$ in the above adaptation rule (see also the description of the algorithm in the following section).

In order to reduce the number of freely adjustable parameters, often leading to a tedious search in parameter space, the increase and decrease factor are set to fixed values. The choice of the decrease factor η^- was lead by the following considerations. If a jump over a minimum occurred, the previous update-value was too large. For it cannot be derived from gradient information how much the minimum was missed, we have to estimate the correct value. In average it will be a good guess to halve the update-value (maximum-likelihood estimator), so we choose $\eta^- := 0.5$. The increase factor η^+ on the one hand, has to be large enough to allow fast growth of the update-value in shallow regions of the errorfunction, but on the other hand the learning process can be considerably disturbed, if a too large increase factor leads to persistent changes of the direction of the weight-step. In several experiments, the choice of $\eta^+ = 1.2$ gave very good results, independent of the examined problem. Slight variations of this value did neither improve nor deteriorate convergence time. So in order to get parameter choice more simple, we decided to constantly fix the increase parameter to $\eta^+ = 1.2$.

For Rprop tries to adapt its learning process to the topology of the errorfunction, it follows the principle of 'batch learning' or 'learning by epoch'. That means, that weight-update and adaptation are performed after the gradient information of the whole pattern set is computed.

B. Algorithm

The following pseudo-code fragment shows the kernel of the RPROP adaptation and learning process. The **minimum (maximum)** operator is supposed to deliver the minimum (maximum) of two numbers; the **sign** operator

returns +1, if the argument is positive, −1, if the argument is negative, and 0 otherwise.

$\forall i,j : \triangle_{ij}(t) = \triangle_0$
$\forall i,j : \frac{\partial E}{\partial w_{ij}}(t-1) = 0$
Repeat
  Compute Gradient $\frac{\partial E}{\partial w}(t)$
  For all weights and biases{
    if $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) > 0)$ **then** {
      $\triangle_{ij}(t) =$ **minimum** $(\triangle_{ij}(t-1) * \eta^+, \triangle_{max})$
      $\triangle w_{ij}(t) = -$ **sign** $(\frac{\partial E}{\partial w_{ij}}(t)) * \triangle_{ij}(t)$
      $w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t)$
      $\frac{\partial E}{\partial w_{ij}}(t-1) = \frac{\partial E}{\partial w_{ij}}(t)$
    }
    else if $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0)$ **then** {
      $\triangle_{ij}(t) =$ **maximum** $(\triangle_{ij}(t-1) * \eta^-, \triangle_{min})$
      $\frac{\partial E}{\partial w_{ij}}(t-1) = 0$
    }
    else if $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) = 0)$ **then** {
      $\triangle w_{ij}(t) = -$ **sign** $(\frac{\partial E}{\partial w_{ij}}(t)) * \triangle_{ij}(t)$
      $w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t)$
      $\frac{\partial E}{\partial w_{ij}}(t-1) = \frac{\partial E}{\partial w_{ij}}(t)$
    }
  }
Until (converged)

C. Parameters

The Rprop algorithm takes two parameters: the initial update-value $\triangle_0$ and a limit for the maximum step size, $\triangle_{max}$.

When learning starts, all update-values are set to an initial value $\triangle_0$. Since $\triangle_0$ directly determines the size of the first weight step, it should be chosen according to the initial values of the weights themselves, for example $\triangle_0 = 0.1$ (default setting). The choice of this value is rather uncritical, for it is adapted as learning proceeds.

In order to prevent the weights from becoming too large, the maximum weight-step determined by the size of the update-value, is limited. The upper bound is set by the second parameter of Rprop, $\triangle_{max}$. The default upper bound is set somewhat arbitrarily to $\triangle_{max} = 50.0$. Usually, convergence is rather insensitive to this parameter as well. Nevertheless, for some problems it can be advantageous to allow only very cautious (namely small) steps, in order to prevent the algorithm getting stuck too quickly in suboptimal local minima. The minimum step size is constantly fixed to $\triangle_{min} = 1e^{-6}$.

D. Discussion

To summarize, the basic principle of Rprop is the direct adaptation of the weight update-values $\triangle_{ij}$. In contrast to learning-rate based algorithms (as for example gradient descent), Rprop modifies the size of the weight-step *directly* by introducing the concept of resilient update-values. As a result, the adaptation effort is not blurred by un-foreseeable gradient behaviour. Due to the clarity and simplicity of the learning laws, there is only a slight expense in computation compared with ordinary backpropagation.

Besides fast convergence, one of the main advantages of RPROP lies in the fact, that for many problems no choice of parameters is needed at all to obtain optimal or at least nearly optimal convergence times.

Another often discussed aspect of common gradient descent is, that the size of the derivative decreases exponentially with the distance between the weight and the output-layer, due to the limiting influence of the slope of the sigmoid activation function. Consequently, weights far away from the output-layer are less modified and do learn much slower. Using RPROP, the size of the weight-step is only dependent on the sequence of signs, not on the magnitude of the derivative. For that reason, learning is spread equally all over the entire network; weights near the input layer have the equal chance to grow and learn as weights near the output layer.

References

[1] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In H. Ruspini, editor, *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pages 586 – 591, San Francisco, 1993.

[2] M. Riedmiller. Untersuchungen zu Konvergenz und Generalisierungsverhalten überwachter Lernverfahren mit dem SNNS. In A. Zell, editor, *SNNS 1993 Workshop-Proceedings*, Stuttgart, September 1993.

[3] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons - from backpropagation to adaptive learning algorithms. *Int. Journal of Computer Standards and Interfaces*, 1994. to appear.

[4] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4), 1988.

[5] T. Tollenaere. Supersab: Fast adaptive backpropagation with good scaling properties. *Neural Networks*, 3(5), 1990.