Advanced Supervised learning in multi-layer perceptrons - From backpropagation to adaptive learning algorithm

Venkatramani Rajgopal

Department of Mathematics
University of Applied Sciences, Mittweida

14 December 2016

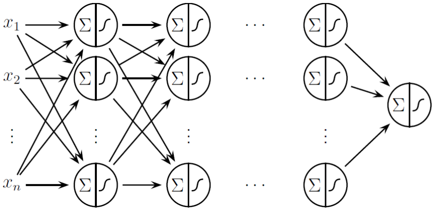
Outline

Introduction

- Discuss the concept of supervised learning in multi layer perceptrons based on gradient descent technique.
- We introduce *Backpropagation* which is one of the most popular training algorithms for multilayer perceptrons.
- Some problems and drawbacks of backpropagation learning procedure.
- Over the last years many improvement strategies have been developed to speed up backpropagation. We look at some of many different speedup techniques.

Preliminaries
Multi Layer Perceptrons

Multi layer perceptron (Werbos 1974, Rumelhart, McClelland, Hinton 1986), is a *feed-forward network*, consisting of neurons connected by weighted links.
It is a finite acyclic graph. The nodes are neurons with sigmoid activation.



Units are organised namely, an input layer, hidden layer/s and an output layer.

Preliminaries

Multi Layer Perceptrons

- Nodes that are no target of any connection are called input neurons. A MLP that should be applied to input patterns of dimension n must have n input neurons, one for each dimension.
- Nodes that are no source of any connection are called output neurons. A MLP can have more than one output neuron. The number of output neurons depends on the way the target values (desired values) of the training patterns are described.
- All nodes that are neither input neurons nor output neurons are called hidden neurons

Preliminaries

Multi Layer Perceptrons

Variables for calculation.

- *Succ(i)* and *Pred(i)* is the set of all neurons j for which connection $i \rightarrow j$ and $j \rightarrow i$ exists respectively.
- The weight of the connection $j \rightarrow i$ is w_{ij}.
- All hidden and output neurons have a bias weight named as θ_i for neuron i.
- Hidden and output neurons have some variable net_i (network input)and s_i as its (activation/output).

Preliminaries

Multi Layer Perceptrons

Applying $\vec{x}$ to the MLP,

- for each input neuron the respective element of the input pattern is presented as, $s_i \leftarrow x_i$.
- for all hidden and output neurons i, calculate net_i and s_i as : $net_i = \sum_{j \in pred(i)} s_j w_{ij} - \theta_i$
- The activation of unit i, s_i is computed by passing the net input through a non-linear activation function, usually sigmoid logistic function.

$$s_i = f_{log}(net_i) = \frac{1}{1 + e^{-net_i}}$$

- A nice property of this function is its easily computable derivative.

$$\frac{\partial s_i}{\partial net_i} = f'_{log}(net_i) = s_i * (1 - s_i)$$

Preliminaries

Multi Layer Perceptrons

Applying $\vec{x}$ to the MLP,

- for each input neuron the respective element of the input pattern is presented as, $s_i \leftarrow x_i$.
- for all hidden and output neurons i, calculate net_i and s_i as : $net_i = \sum_{j \in pred(i)} s_j w_{ij} - \theta_i$
- The activation of unit i, s_i is computed by passing the net input through a non-linear activation function, usually sigmoid logistic function.

$$s_i = f_{log}(net_i) = \frac{1}{1 + e^{-net_i}}$$

- A nice property of this function is its easily computable derivative.

$$\frac{\partial s_i}{\partial net_i} = f'_{log}(net_i) = s_i * (1 - s_i)$$

Preliminaries

Supervised Learning

Objective: To tune the weights in the network such that the network performs a desired mapping of input to output activations.

- The mapping is given by a set, the so called pattern set $\mathscr{P}$.
- Each pattern pair p, consist of an input activation vector x^p and target activation vector t^p.
- After training the weights, when an input activation x^p is presented, the resulting output vector s^p should equal the target t^p.
- The distance between the target and the actual output vector, is measured by the following cost function E :.

$$E := \frac{1}{2} \sum_{p \in \mathscr{P}} \sum_{n} (t_n^p - s_n^p)^2$$

Preliminaries

Supervised Learning

Objective: To tune the weights in the network such that the network performs a desired mapping of input to output activations.

- The mapping is given by a set, the so called pattern set $\mathscr{P}$.
- Each pattern pair p, consist of an input activation vector x^p and target activation vector t^p.
- After training the weights, when an input activation x^p is presented, the resulting output vector s^p should equal the target t^p.
- The distance between the target and the actual output vector, is measured by the following cost function E :.

$$E := \frac{1}{2} \sum_{p \in \mathscr{P}} \sum_{n} (t_n^p - s_n^p)^2$$

Preliminaries

Supervised Learning

Objective: To tune the weights in the network such that the network performs a desired mapping of input to output activations.

- The mapping is given by a set, the so called pattern set $\mathscr{P}$.
- Each pattern pair p, consist of an input activation vector x^p and target activation vector t^p.
- After training the weights, when an input activation x^p is presented, the resulting output vector s^p should equal the target t^p.
- The distance between the target and the actual output vector, is measured by the following cost function E :.

$$E := \frac{1}{2} \sum_{p \in \mathscr{P}} \sum_{n} (t_n^p - s_n^p)^2$$

Preliminaries

Supervised Learning

Learning means: calculating weights for which the error E becomes minimal.

The weights in the network are changed along a *search direction* $d(t)$,

$$\Delta w(t) = \epsilon * d(t)$$

where the learning rate ϵ, scales the size of the weight step.

To determine the search direction $d(t)$, we use the first order derivative, the gradient

$$\Delta E = \frac{\partial E}{\partial w}$$

The Back propagation algorithm, performs successive computations of ΔE, by propagating the error back from output towards the input layer.

Idea: Compute the partial derivative $\partial E/\partial w_{ij}$ for each weight in the network, by repeatedly applying the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial w_{ij}}$$

where,

$$\frac{\partial s_i}{\partial w_{ij}} = \frac{\partial s_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}} = f'_{log}(net_i)s_j$$

Preliminaries

Back Propagation algorithm

The Back propagation algorithm, performs successive computations of ΔE, by propagating the error back from output towards the input layer.
Idea: Compute the partial derivative $\partial E/\partial w_{ij}$ for each weight in the network, by repeatedly applying the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial w_{ij}}$$

where,

$$\frac{\partial s_i}{\partial w_{ij}} = \frac{\partial s_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}} = f'_{log}(net_i)s_j$$

Preliminaries

Back Propagation algorithm

To compute $\partial E/\partial s_i$, we look at the two cases:

- If i is an output unit then,

$$\frac{\partial E}{\partial s_i} = \frac{1}{2}\frac{\partial (t_i - s_i)^2}{\partial s_i} = -(t_i - s_i)$$

- If i is not an output unit, then we apply the chain rule again;

$$\begin{aligned}\frac{\partial E}{\partial s_i} &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial s_i} \\ &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial net_k}\frac{\partial net_k}{\partial s_i} \\ &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k} f'_{log}(net_k) w_{ki}\end{aligned}$$

Preliminaries

Back Propagation algorithm

To compute $\partial E/\partial s_i$, we look at the two cases:

- If i is an output unit then,

$$\frac{\partial E}{\partial s_i} = \frac{1}{2}\frac{\partial (t_i - s_i)^2}{\partial s_i} = -(t_i - s_i)$$

- If i is not an output unit, then we apply the chain rule again;

$$\begin{aligned}\frac{\partial E}{\partial s_i} &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial s_i} \\ &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial net_k}\frac{\partial net_k}{\partial s_i} \\ &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k} f'_{log}(net_k) w_{ki}\end{aligned}$$

Preliminaries

Back Propagation algorithm

To compute $\partial E/\partial s_i$, we look at the two cases:

- If i is an output unit then,

$$\frac{\partial E}{\partial s_i} = \frac{1}{2}\frac{\partial (t_i - s_i)^2}{\partial s_i} = -(t_i - s_i)$$

- If i is not an output unit, then we apply the chain rule again;

$$\begin{aligned}
\frac{\partial E}{\partial s_i} &= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial s_i} \\
&= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial net_k}\frac{\partial net_k}{\partial s_i} \\
&= \sum_{k \in succ(i)} \frac{\partial E}{\partial s_k} f'_{log}(net_k) w_{ki}
\end{aligned}$$

The next step in backpropagation is to compute the weight update.

- Weight update is a scaled step in the opposite direction of the gradient.
- The negative derivative is multiplied by a constant value, the learning-rate, ϵ.
- We call this minimization technique as gradient descent:

$$\Delta w_{ij}(t) = -\epsilon * \frac{\partial E}{\partial w_{ij}}(t)$$

The next step in backpropagation is to compute the weight update.

- Weight update is a scaled step in the opposite direction of the gradient.
- The negative derivative is multiplied by a constant value, the learning-rate, ϵ.
- We call this minimization technique as gradient descent:

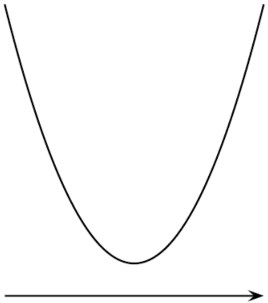
$$\Delta w_{ij}(t) = -\epsilon * \frac{\partial E}{\partial w_{ij}}(t)$$

Preliminaries

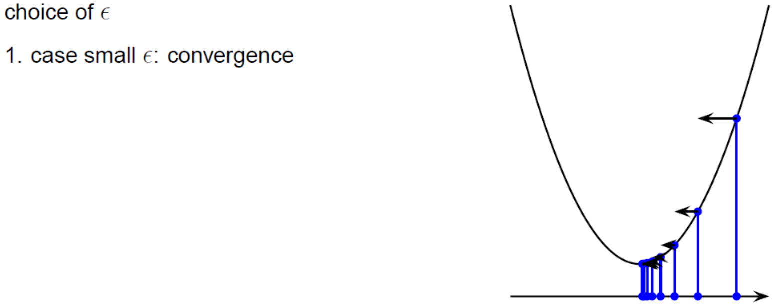
Gradient Descent

Choosing the learning rate.
A good choice depends on the error-function.

choice of ϵ



Choosing the learning rate.
A good choice depends on the error-function.



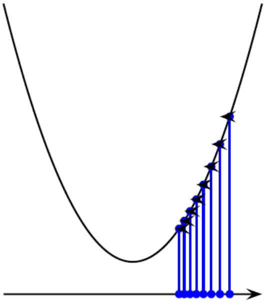
Preliminaries

Gradient Descent

Choosing the learning rate.
A good choice depends on the error-function.

choice of ϵ

2. case very small ϵ: convergence, but it may take very long



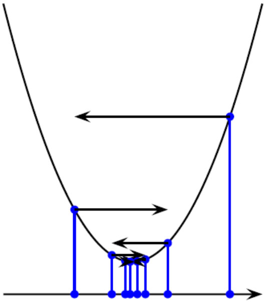
Preliminaries

Gradient Descent

Choosing the learning rate.
A good choice depends on the error-function.

choice of ϵ

3. case medium size ϵ: convergence



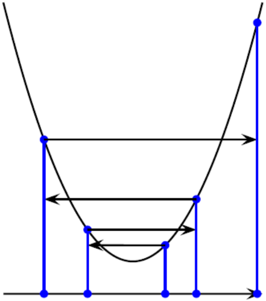
Preliminaries

Gradient Descent

Choosing the learning rate.
A good choice depends on the error-function.

choice of ϵ

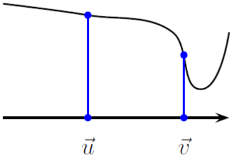
4. case large ϵ: divergence



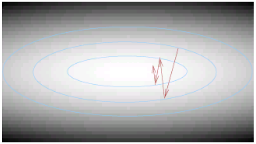
Preliminaries

Gradient Descent - Problems

- The weight step is dependent on both the learning parameter and the size of the partial derivative $\partial E/\partial w_{ij}$.
- Flat spots and steep valleys:
  We need larger ϵ in $\vec{u}$ to jump over the flat area but need smaller ϵ in $\vec{v}$ to meet the minimum.



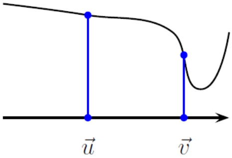
- Zig-zagging In higher dimensions: ϵ is not appropriate for all dimensions.



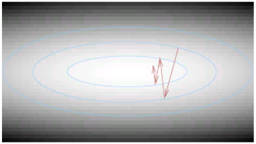
Preliminaries

Gradient Descent - Problems

- The weight step is dependent on both the learning parameter and the size of the partial derivative $\partial E / \partial w_{ij}$.
- Flat spots and steep valleys:
  We need larger ϵ in $\vec{u}$ to jump over the flat area but need smaller ϵ in $\vec{v}$ to meet the minimum.



- Zig-zagging In higher dimensions: ϵ is not appropriate for all dimensions.



Finding the right ϵ is annoying. Approaching the minimum is time consuming.

Heuristics to overcome problems of gradient descent:

- Gradient descent with momentum
- Individual learning rates for each dimension
- Adaptive learning rates

Preliminaries

Gradient Descent with momentum

To make learning more stable, one of the idea was to introduce a momentum term, μ :

$$\Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1)$$

It scales the influence of previous weight step on the current one.

Usually, when using gradient descent with momentum, the learning rate should be decreased to avoid unstable learning.

Preliminaries

Gradient Descent with momentum

To make learning more stable, one of the idea was to introduce a momentum term, μ :

$$\Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1)$$

It scales the influence of previous weight step on the current one.

Usually, when using gradient descent with momentum, the learning rate should be decreased to avoid unstable learning.

Preliminaries

Gradient Descent with momentum

Advantages of momentum

- Smoothes zig-zagging
- Accelerates learning at flat spots
- Slows down when signs of partial derivatives change

Preliminaries

Learning by pattern vs learning by epoch

Two methods for computing weight update.

- In learning by pattern method a weight update is performed after computation of respective gradient. This is known as online learning.
- Learning by epoch first sums the gradient information for the whole pattern set, then performs the weight update. Known as batch learning.

Global Adaptive Techniques

Steepest Descent

Adaptive learning rate. Idea:

- Make learning rate individual for each dimension and adaptive
- If signs of partial derivative change, reduce learning rate
- If signs of partial derivative don't change, increase learning rate

Algorithms, that use the global knowledge of the entire network, like *direction* of the overall weight update are referred as global techniques.

Global Adaptive Techniques

Steepest Descent

Adaptive learning rate. Idea:

- Make learning rate individual for each dimension and adaptive
- If signs of partial derivative change, reduce learning rate
- If signs of partial derivative don't change, increase learning rate

Algorithms, that use the global knowledge of the entire network, like *direction* of the overall weight update are referred as global techniques.

Global Adaptive Techniques

Steepest Descent

The steepest descent tries to take an optimal weight step by finding an individual scaling parameter $\epsilon(t)$, each iteration.

- To find such a parameter, is regarded as line search.
- A small initial learning rate is used, which is increased until the error function no longer decreases.

Drawback: For every iteration, the evaluation of the error function E is required, which is a costly propagation, to compute the new value of E.

Global Adaptive Techniques

Steepest Descent

The steepest descent tries to take an optimal weight step by finding an individual scaling parameter $\epsilon(t)$, each iteration.

- To find such a parameter, is regarded as line search.
- A small initial learning rate is used, which is increased until the error function no longer decreases.

Drawback: For every iteration, the evaluation of the error function E is required, which is a costly propagation, to compute the new value of E.

Global Adaptive Techniques

Steepest Descent

While applying Steepest Descent, it can be shown that two successive weight steps are perpendicular.

$$\frac{\partial(w(t+1))}{\partial \epsilon} = 0$$

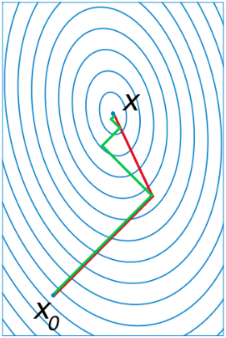
Then,

$$\begin{aligned}
\frac{\partial(w(t+1))}{\partial \epsilon} &= \frac{\partial(w(t+1))}{\partial w(t+1)} \frac{\partial(w(t) + \epsilon * \partial(t))}{\partial \epsilon} \\
&= \nabla E(t+1) d(t) \\
&= 0 \qquad (1)
\end{aligned}$$

Means that the new gradient $\nabla E(t+1)$ that determines the new direction $d(t+1)$ and old direction $d(t)$ are perpendicular.

Global Adaptive Techniques

Conjugate gradient method



Comparison of the convergence of gradient descent with optimal step (in green) and conjugate vector (in red).

Global Adaptive Techniques

Conjugate gradient method

The condition from Equation (1) also holds good for the weight step,

$$d(t)\nabla E(t+2) = 0$$

and it can be shown that above is fulfilled if,

$$d(t)\mathscr{H}d(t+1) = 0 \tag{2}$$

where $\mathscr{H}$ denotes that Hessian Matrix, containing second order derivatives of the weights. Two vectors fulfilling the above are called

conjugate.

Global Adaptive Techniques

Conjugate gradient method

To determine the new search direction $d(t+1)$ that fulfills equation (2) we set,

$$d(t+1) = -\nabla E(t+1) + \beta * d(t)$$

This means that the new search direction is a combination of the direction indicated by the gradient and the previous search direction.

The parameter β is computed using the Polak-Ribiere rule:

$$\beta = \frac{(\nabla E(t+1) - \nabla E(t))\nabla E(t+1)}{(\nabla E(t))^2}$$

Global Adaptive Techniques

Conjugate gradient method

To determine the new search direction $d(t+1)$ that fulfills equation (2) we set,

$$d(t+1) = -\nabla E(t+1) + \beta * d(t)$$

This means that the new search direction is a combination of the direction indicated by the gradient and the previous search direction.

The parameter β is computed using the Polak-Ribiere rule:

$$\beta = \frac{(\nabla E(t+1) - \nabla E(t))\nabla E(t+1)}{(\nabla E(t))^2}$$

Local adaptive techniques

Delta bar delta rule

R Jacobs, proposed the weight specific learning rates. He determined the evolution of learning rates according to the estimation of the shape of the error function.

- Based on the observed behaviour of the partial derivatives during two successive weight steps.
- If derivatives have same sign, the learning rate is linearly increased by a small constant.
- On the other hand, a change in sign of the two derivatives indicates that the procedure has over shot a local minimum. (i.e the the previous weight step was too large).
- As a consequence, the learning rate is decreased.

Local adaptive techniques

Delta bar delta rule

R Jacobs, proposed the weight specific learning rates. He determined the evolution of learning rates according to the estimation of the shape of the error function.

- Based on the observed behaviour of the partial derivatives during two successive weight steps.
- If derivatives have same sign, the learning rate is linearly increased by a small constant.
- On the other hand, a change in sign of the two derivatives indicates that the procedure has over shot a local minimum. (i.e the the previous weight step was too large).
- As a consequence, the learning rate is decreased.

Local adaptive techniques

Delta bar delta rule

Which is,

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

with $0 < \eta^{-} < 1$.

Weight update is the same as with backpropagation learning, except that, the fixed learning rate ϵ is replaced by the weight specific, dynamic learning rate $\epsilon_{ij}(t)$

$$\Delta w_{ij}(t) = -\epsilon_{ij}(t)\frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1)$$

Local adaptive techniques

Delta bar delta rule

Which is,

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

with $0 < \eta^{-} < 1$.

Weight update is the same as with backpropagation learning, except that, the fixed learning rate ϵ is replaced by the weight specific, dynamic learning rate $\epsilon_{ij}(t)$

$$\Delta w_{ij}(t) = -\epsilon_{ij}(t)\frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1)$$

Local adaptive techniques
SuperSAB

- Based on the idea of sign-dependent learning rate adoption.
- Change here, is to increase the learning rate exponentially instead of linearly like in Delta Bar Delta rule.

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

with $0 < \eta^{-} < 1 < \eta^{+}$.

Also, in case of a change in sign of two successive derivatives, the previous weight step is reverted.

Local adaptive techniques

SuperSAB

Advantage:
Fast convergence. Often faster than ordinary gradient descent.

Disadvantage:
Determination of large number of parameters to achieve good convergence.
Initial learning rate, the momentum factor and the increase (decrease) factor.

$$\Delta w_{ij}(t) = -\epsilon_{ij}(t)\frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1)$$

Local adaptive techniques
Quickprop

Local adaptive techniques are based on weight specific information, such as the behaviour of the partial derivative.

Idea: To find a solution in a short time, taking the largest step possible, without overshooting the solution.

- Here we make explicit use of the second derivative of the error with respect to each weight.
- It is a second-order method, based loosely on Newton's method.
- Everything proceeds as in standard back-propagation, but for each weight, keep a copy of $\partial E / \partial w(t-1)$, the error derivative computed during the previous training epoch, along with the difference between the current and previous values of this weight.

Local adaptive techniques

Quickprop

Local adaptive techniques are based on weight specific information, such as the behaviour of the partial derivative.

Idea: To find a solution in a short time, taking the largest step possible, without overshooting the solution.

- Here we make explicit use of the second derivative of the error with respect to each weight.
- It is a second-order method, based loosely on Newton's method.
- Everything proceeds as in standard back-propagation, but for each weight, keep a copy of $\partial E/\partial w(t-1)$, the error derivative computed during the previous training epoch, along with the difference between the current and previous values of this weight.

Quickprop

Assumption

Local error function for each weight is assumed to be a 'Parabalo whose arms are wide open'.

For each weight, independently, we use the previous and current error slopes and the weight-change between the points at which these slopes were measured to determine a parabola; we then jump directly to the *minimum point of this parabola.*

So the update rule we have;

$$\Delta w_{ij}(t) = \frac{\frac{\partial E}{\partial w_{ij}}(t)}{\frac{\partial E}{\partial w_{ij}}(t-1) - \frac{\partial E}{\partial w_{ij}(t)}} \Delta w(t-1) \tag{3}$$

Quickprop

Assumption

Local error function for each weight is assumed to be a 'Parabalo whose arms are wide open'.

For each weight, independently, we use the previous and current error slopes and the weight-change between the points at which these slopes were measured to determine a parabola; we then jump directly to the *minimum point of this parabola.*

So the update rule we have;

$$\Delta w_{ij}(t) = \frac{\frac{\partial E}{\partial w_{ij}}(t)}{\frac{\partial E}{\partial w_{ij}}(t-1) - \frac{\partial E}{\partial w_{ij}(t)}} \Delta w(t-1) \tag{3}$$

Quickprop

Assumption

Local error function for each weight is assumed to be a 'Parabalo whose arms are wide open'.

For each weight, independently, we use the previous and current error slopes and the weight-change between the points at which these slopes were measured to determine a parabola; we then jump directly to the *minimum point of this parabola.*

So the update rule we have;

$$\Delta w_{ij}(t) = \frac{\frac{\partial E}{\partial w_{ij}}(t)}{\frac{\partial E}{\partial w_{ij}}(t-1) - \frac{\partial E}{\partial w_{ij}(t)}} \Delta w(t-1) \tag{3}$$

Quickprop

- The update rule is equivalent to Newtons approximation method.
- The objective is to find a minimum of $f(x)$
- Newtons method computes updates of x according to ;

$$x(t+1) = x(t) + \Delta x(t)$$

where,
$\Delta x(t) = -\frac{f'(x(t))}{f''(x(t))}$
Approximation using the first order derivatives:

$$f''(x(t)) = \frac{f'(x(t)) - f'(x(t-1))}{x(t) - x(t-1)} \quad = \frac{f'(x(t)) - f'(x(t-1))}{\Delta x(t-1)}$$

Quickprop

- By substitution we have,

$$\Delta x(t) = \frac{f'(x(t))}{f'(x(t-1)) - f'x(t)} \Delta x(t-1) \tag{4}$$

which corresponds to the update rule $\Delta w_{ij}(t)$. (Equation (3))

- The update rule is composed of $\Delta w_{ij}(t)$ and a small gradient step.

- To avoid large weight steps, coming from small denominator, the present weight step is restricted to at most ν times as large as the previous step.

- The Quickprop thus has two parameters.
Learning rate ϵ for gradient descent and a second parameter ν which limits the step size.

Quickprop

- By substitution we have,

$$\Delta x(t) = \frac{f'(x(t))}{f'(x(t-1)) - f'x(t)} \Delta x(t-1) \tag{4}$$

  which corresponds to the update rule $\Delta w_{ij}(t)$. (Equation (3))
- The update rule is composed of $\Delta w_{ij}(t)$ and a small gradient step.
- To avoid large weight steps, coming from small denominator, the present weight step is restricted to at most ν times as large as the previous step.
- The Quickprop thus has two parameters.
  Learning rate ϵ for gradient descent and a second parameter ν which limits the step size.

Rprop - Resilient backpropagation

The basic idea here is to eliminate the harmful influence of the size of the partial derivative on the weight step.

- To achieve this, we introduce for each weight its individual update value Δ_{ij}, which solely determines the size of the weight update.

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

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

Rprop - Resilient backpropagation

The basic idea here is to eliminate the harmful influence of the size of the partial derivative on the weight step.

- To achieve this, we introduce for each weight its individual update value Δ_{ij}, which solely determines the size of the weight update.

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

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

Rprop - Resilient backpropagation

Adaptation rule:

- 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 Δ_{ij} is decreased by a factor of η^{-}.
- If the derivative retains its sign, the update-value is slightly increased in order to accelerate convergence in shallow regions.

Rprop - Resilient backpropagation

Once the update-value for each weight is adapted, the weight-update itself follows a very simple rule:

- If the derivative is positive (ie if we have an increasing error), the weight is decreased by its update-value.
- If the derivative is negative, the update-value is added.

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

Rprop - Resilient backpropagation

Exception: If the partial derivative changes sign, i.e. the previous step was too large and the minimum was missed, the previous weight-update is reverted:

$$\Delta w_{ij}^{(t)} = -\Delta w_{ij}^{(t-1)}, \text{ if } \qquad \frac{\partial E^{(t-1)}}{\partial w_{ij}} * \frac{\partial E^{(t)}}{\partial w_{ij}} < 0$$

Rprop - Resilient backpropagation

Parameters.

- Beginning: all update values Δ_{ij} are set to an initial value of Δ_0.
- The second paramater is the upper bound Δ_{max}. This is set inorder to prevent the weights from becoming too large, max weight step determined by the size of the update value is limited.
- The increase and decrease factors are fixed to $\eta^+ = 1.2$ and $\eta^- = 0.5$.

Rprop - Resilient backpropagation

Main advantages of RPROP - For many problems no choice of parameters is needed at all to obtain optimal convergence.

To summarize,

- Rprop is the direct adaptation of the weight update values Δ_{ij}.
- It modifies the size of the weight step directly by introducing a concept of *resilient update-values.*
- As a result adaptation effort is not blurred by unforeseeable gradient behaviour.

Test Results

6 Bit Parity				
Algorithm	ϵ/Δ_0	$\mu/\nu/\Delta_{max}$	# epochs	success
BP by ep.	0.3	0.0	279.4	16/20
SSAB	0.01	0.9	82.6	20/20
QP	0.005	*	50.5	20/20
RPROP	0.05	*	52.8	20/20

Figure: Results for different learning procedures

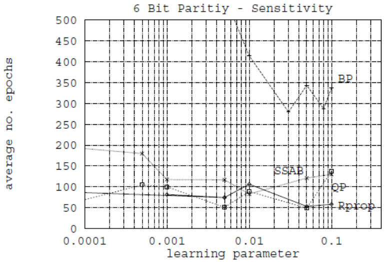


Figure: Sensitivity of different learning procedures to choice of learning parameter

References I

Martin Riedmiller
Advanced supervised learning in Multi layer perceptrons. From Back propagation to adaptive learning algorithms
Institut für Logik, Komplexitdt und Deduktionssysteme, University of Karlsruhe,Germany

Scott E. Fahlman
An Empirical Study of Learning Speed in Back-Propagation Networks
September 1988, CMU-CS-88-162

Martin Riedmiller, Heinrich Braun
A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm
Institut für Logik, Komplexitdt und Deduktionssysteme, University of Karlsruhe,Germany