Accelerating Fuzzy Clustering

Christian Borgelt

European Center for Soft Computing
Campus Mieres, Edificio Científico-Tecnológico
c/ Gonzalo Gutiérrez Quirós s/n, 33600 Mieres, Spain

email: christian.borgelt@softcomputing.es

Abstract

This paper extends earlier work [6] on an approach to accelerate fuzzy clustering by transferring methods that were originally developed to speed up the training process of (artificial) neural networks. The core idea is to consider the difference between two consecutive steps of the alternating optimization scheme of fuzzy clustering as providing a gradient. This "gradient" may then be modified in the same way as a gradient is modified in error backpropagation in order to enhance the training. Even though these modifications are, in principle, directly applicable, carefully checking and bounding the update steps can improve the performance and can make the procedure more robust. In addition, this paper provides a new and much more detailed experimental evaluation that is based on fuzzy cluster comparison measures [9], which can be used nicely to study the convergence speed.

1 Introduction

Clustering large data sets with an iterative parameter update scheme can be computationally expensive, since a considerable number of update steps may be necessary in order to reach convergence and each update step usually requires a full traversal of the data set. Therefore it is worthwhile to look for methods that have the potential to reduce the number of update steps that are needed in order to (almost) achieve convergence. This paper reviews and extends an approach to transfer techniques that were originally developed for improving the training process of (artificial) neural networks—in particular multilayer perceptrons—to prototype-based clustering—in particular fuzzy clustering. The result is a family of update rules, which increase the convergence speed considerably.

The rationale underlying the presented approach is that both neural network training and fuzzy clustering aim at the minimization of an objective function, namely the error function of an (artificial) neural network and the sum of squared distances for fuzzy clustering. Although they rely on different principles (gradient descent versus alternating optimization), they share the

characteristic property that both execute iterative steps in the parameter space in order to approach the minimum of the respective objective function.

However, since in neural network training only the direction of the update step is actually known (from the direction of the gradient of the error function), while the step size is a matter of choice (using the size of the gradient directly is usually a fairly bad idea), several schemes for deriving the step size have been studied (see Section 2 for a review). In fuzzy clustering, on the other hand, each step of the alternating optimization scheme actually computes the optimum for one set of the parameters (cluster parameters or membership degrees), given that the values of the parameters of the other set are fixed to their current values (conditional optimum). Hence modifying an update step may not appear to be beneficial, since any modification necessarily deviates from the conditional optimum. As a consequence, update modifications are rarely studied.

But this view is a bit myopic, since the next update step usually also changes the parameters beyond the current conditional optimum (except at a convergence point), since the other parameter set will have changed its values. This leads to the idea that the differences between cluster parameters, as they are computed in two consecutive update steps, can be seen as yielding a gradient, also providing mainly a direction, but not necessarily the optimal step width. This "gradient" may thus be modified with basically the same methods that are used to modify the gradient in (artificial) neural network error backpropagation in order to accelerate the convergence to the (possibly only local) joint optimum of the two parameter sets (cluster parameters and membership degrees).

This paper is organized as follows: Section 2 briefly reviews the general technique of training (artificial) neural networks with error backpropagation as well as four well-known and popular update modifications that have been suggested as improvements. Section 3 recalls the necessary basics of fuzzy clustering, while Section 4 discusses how the update modifications can be transferred. In doing so, special attention is paid to how the update steps should be checked and bounded in order to obtain robust update procedures. Section 5 reviews how fuzzy clustering results can be compared and explains how the used measures can be applied to study the convergence speed. In Section 6 several experiments are reported that provide a consistent assessment of the performance of the different methods, so that clear recommendations can be given. Finally, Section 7 summarizes the discussion, draws conclusions, and, in particular, compares the results to those of [6], where the core idea of the studied approach was first presented, though in a less sophisticated form.

2 Neural Network Training

After a brief recall of the basic training scheme of (artificial) neural networks (Sections 2.1 and 2.2), this section reviews some of the best-known methods for improving the gradient descent procedure, including a momentum term (Section 2.3), parameter-specific self-adapting learning rates (Section 2.4), resilient backpropagation (Section 2.5) and quick backpropagation (Section 2.6).

2.1 Gradient Descent Training

The basic scheme to train an (artificial) neural network, especially a multilayer perceptron, is a gradient descent on the error surface [16, 32, 1]. That is, the error of the output of the (artificial) neural network (w.r.t. a given data set of training samples) is seen as a function of the network parameters, that is, of the connection weights and bias values of the neurons. Provided that a differentiable activation function is employed, the partial derivatives of this error function w.r.t. these parameters can be computed, yielding the direction of steepest ascent of the error function (error gradient). Starting from a random initialization of the parameters, they are then changed in the direction opposite to this gradient, since the objective is, of course, to minimize the error. At the new point in the parameter space the gradient is recomputed and another step in the direction of steepest descent is carried out. The process stops when a (local) minimum of the error function is reached.

2.2 Standard Backpropagation

In order to have a reference point, recall the weight update rule for standard error backpropagation. This rule reads for a (weight) parameter w:

$$w(t+1) = w(t) + \Delta w(t) \quad \text{where} \quad \Delta w(t) = -\eta \nabla_w e(t).$$

That is, the new value of the parameter (in step $t+1$) is computed from the old value (in step t) by adding a weight change, which is computed from the gradient $\nabla_w e(t)$ of the error function $e(t)$ w.r.t. this parameter w. η is a learning rate that influences the size of the steps that are carried out (relative to the size of the current gradient). The minus sign results from the fact that the gradient points into the direction of the steepest *ascent*, but we have to carry out a gradient *descent*. Depending on the definition of the error function, the gradient may also be preceded by a factor of $\frac{1}{2}$ in this formula, in order to cancel a factor of 2 that results from differentiating a squared error.

Note that it can be difficult to choose an appropriate learning rate, because a small learning rate can lead to unnecessarily slow learning, whereas a large learning rate can lead to oscillations and uncontrolled jumps.

2.3 Momentum Term

The momentum term method [29] consists in adding a fraction of the weight change of the previous step to a normal gradient descent step. The rule for changing the parameters (weights) thus becomes

$$\Delta w(t) = -\eta \nabla_w e(t) + \beta \Delta w(t-1),$$

where β is a parameter, which must be smaller than 1 in order to make the method stable (for $\beta \geq 1$ the parameter changes can grow without bounds). In neural network training β is usually chosen between 0.5 and 0.95.

The additional term $\beta\Delta w(t-1)$ is called *momentum term*, because its effect corresponds to the momentum that is gained by a ball rolling down a slope. The longer the ball rolls in the same direction, the faster it gets. So it has a tendency to keep on moving in the same direction (momentum term), but it also follows, though slightly retarded, the shape of the surface (gradient term).

By adding a momentum term training can be accelerated, especially in areas of the parameter space, in which the error function is (almost) flat, but descends in a uniform direction. It also slightly mitigates (but does not fully eliminate) the problem of how to choose the value of the learning rate η. However, this aspect is not relevant for the transfer of the method to fuzzy clustering.

2.4 Self-Adaptive Backpropagation

The idea of (super) self-adaptive backpropagation (SuperSAB) [20, 30] is to introduce an individual learning rate η_w for each parameter of the neural network, i.e., for each connection weight and each bias value. These learning rates are then adapted (before they are used in the current update step) according to the values of the current and the previous gradient. The exact adaptation rule for the learning rates is

$$\eta_w(t) = \begin{cases} \gamma^- \cdot \eta_w(t-1), & \text{if } \nabla_w e(t) \cdot \nabla_w e(t-1) < 0, \\ \gamma^+ \cdot \eta_w(t-1), & \text{if } \nabla_w e(t) \cdot \nabla_w e(t-1) > 0 \\ & \wedge\ \nabla_w e(t-1) \cdot \nabla_w e(t-2) \geq 0, \\ \eta_w(t-1), & \text{otherwise.} \end{cases}$$

γ^- is a shrink factor ($\gamma^- < 1$), which is used to reduce the learning rate if the current and the previous gradient have opposite signs. In this case we have leapt over the minimum of the error function, so smaller steps are necessary to approach it. Typical values for γ^- are between 0.5 and 0.7.

γ^+ is a growth factor ($\gamma^+ > 1$), which is used to increase the learning rate if the current and the previous gradient have the same sign. In this case two steps are carried out in the same direction, so it is plausible to assume that we have to run down a longer slope of the error function. Consequently, the learning rate should be increased in order to proceed faster. Typically, γ^+ is chosen between 1.05 and 1.2, so that the learning rate grows relatively slowly.

The second condition for the application of the growth factor γ^+ prevents that the learning rate is increased immediately after it has been decreased in the previous step. A common way of implementing this is to simply set the previous gradient to zero in order to indicate that the learning rate was decreased. Although this also suppresses two consecutive reductions of the learning rate, it has the advantage that it eliminates the need to store $\nabla_w e(t-2)$.

In order to prevent the weight changes from becoming too small or too large, it is common to limit the learning rate to a reasonable range. It is also recommended to use batch training (that is, the parameters are updated only after all training patterns have been processed), as online training (that is, the parameters are updated after each training pattern) tends to be unstable.

2.5 Resilient Backpropagation

The resilient backpropagation approach (Rprop) [27] can be seen as a combination of the ideas of Manhattan training (which is like standard backpropagation, though only the sign of the gradient is used, so that the learning rate determines the step size directly) and (super) self-adaptive backpropagation. For each parameter of the (artificial) neural network, that is, for each connection weight and each bias value, an individual *step width* Δw is introduced, which is adapted according to the values of the current and the previous gradient. The adaptation rule reads

$$\Delta w(t) = \begin{cases} \gamma^- \cdot \Delta w(t-1), & \text{if } \nabla_w e(t) \cdot \nabla_w e(t-1) < 0, \\ \gamma^+ \cdot \Delta w(t-1), & \text{if } \nabla_w e(t) \cdot \nabla_w e(t-1) > 0 \\ & \quad \wedge\ \nabla_w e(t-1) \cdot \nabla_w e(t-2) \geq 0, \\ \Delta w(t-1), & \text{otherwise.} \end{cases}$$

In analogy to self-adaptive backpropagation, γ^- is a shrink factor ($\gamma^- < 1$) and γ^+ a growth factor ($\gamma^+ > 1$), which are used to decrease or to increase the step width. The application of these factors is justified in exactly the same way as for self-adaptive backpropagation. The typical ranges of values also coincide (that is, $\gamma^- \in [0.5, 0.7]$ and $\gamma^+ \in [1.05, 1.2]$).

Like in self-adaptive backpropagation the step width is restricted to a reasonable range in order to avoid far jumps as well as slow learning. It is also advisable to use batch training as online training can be very unstable.

In several applications of (artificial) neural networks resilient backpropagation has proven to be superior to a lot of other approaches (including momentum term, self-adaptive backpropagation, and quick backpropagation), especially w.r.t. training time [32]. It is commonly regarded as one of the most highly recommendable methods for training multilayer perceptrons.

2.6 Quick Backpropagation

The idea underlying quick backpropagation (Quickprop) [10] is to locally approximate the error function by a parabola (see Figure 1) and to change the weight in such a way that we end up at the apex of this parabola. That is, the weight is simply set to the value at which the apex lies. If the error function is "good-natured", that is, can be approximated well by a parabola, this enables us to get fairly close to the true minimum in one or very few steps.

The rule for changing the weights can easily be derived from the derivative of the approximation parabola (see Figure 2). Obviously, it is (consider the shaded triangles, both of which describe the slope of the derivative)

$$\frac{\nabla_w e(t-1) - \nabla_w e(t)}{w(t-1) - w(t)} = \frac{\nabla_w e(t)}{w(t) - w(t+1)}.$$

Solving for $\Delta w(t) = w(t+1) - w(t)$ and exploiting $\Delta w(t-1) = w(t) - w(t-1)$ we get

$$\Delta w(t) = \frac{\nabla_w e(t)}{\nabla_w e(t-1) - \nabla_w e(t)} \cdot \Delta w(t-1).$$

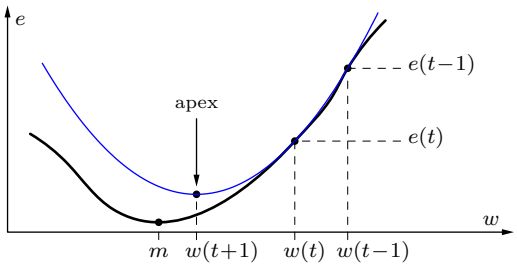


Figure 1: Quick backpropagation uses a parabola to locally approximate the error function. m is the value of the true minimum.

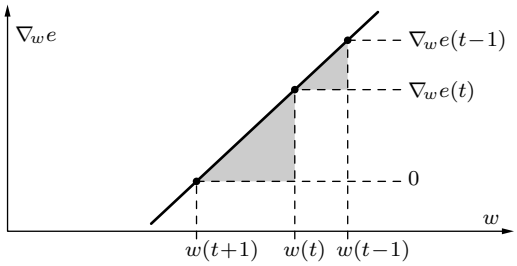


Figure 2: The formula for the weight change can easily be derived from the derivative of the approximation parabola.

However, it has to be taken into account that the above formula does not distinguish between a parabola that opens upwards (ascending derivative) and one that opens downwards (descending derivative), so that a maximum of the error function may be approached. Although this can be avoided by checking whether

$$\frac{\nabla_w e(t) - \nabla_w e(t-1)}{w(t) - w(t-1)} = \frac{\nabla_w e(t) - \nabla_w e(t-1)}{\Delta w(t-1)} > 0$$

holds (in this case the parabola opens upwards), this check is often missing in implementations (maybe because this case occurs only fairly rarely in practice). Furthermore a growth factor is usually introduced, which limits the weight change relative to the previous step. That is, it is made sure that

$$|\Delta w(t)| \leq \alpha \cdot |\Delta w(t-1)|$$

holds, where α is a parameter that is commonly chosen between 1.75 and 2.25.

In addition, (artificial) neural network implementations of this method often add a normal gradient descent step if the two gradients $\nabla_w e(t)$ and $\nabla_w e(t-1)$ have the same sign, that is, if the minimum does not lie between the current and the previous weight value. Finally, it is advisable to limit the weight change to some maximum in order to avoid far (and thus unstable) jumps.

If the assumptions underlying the quick backpropagation method—that is, that the error function can be approximated locally by a parabola, which opens upwards, and that the parameters can be changed fairly independently of each other—hold and if batch training is used, it is one of the fastest training methods for multilayer perceptrons. Otherwise it tends to be somewhat unstable and is susceptible to oscillations (that is, it may fail to converge reliably).

3 Fuzzy Clustering

Fuzzy clustering is an objective function based method (see Section 3.1 for a formal definition) to divide a data set into a set of groups or clusters [2, 3, 4, 17, 8]. In contrast to standard (crisp) clustering, fuzzy clustering offers the possibility to assign a data point to more than one cluster, so that overlapping clusters can be handled conveniently, and to assign it with degrees of membership. Each cluster is represented by a prototype, which consists of a cluster center and maybe some additional information about the size and the shape of the cluster (see Section 3.2 for examples). The degree of membership, to which a data point belongs to a cluster, is computed from the distances of the data point to the cluster centers w.r.t. the size and shape information.

3.1 Objective Function

Formally, in fuzzy clustering we are given a data set $\mathbf{X}$ of n data points $\vec{x}_j \in \mathbb{R}^m$, which is to be divided into c clusters, where c has to be specified by a user (however, see, for example, [9] for automatic ways to determine c). The fuzzy clustering result is obtained by minimizing the objective function

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{\kappa} \, d^2(\vec{x}_j, \mathbf{c}_i)$$

subject to

$$\forall i \in \{1, \ldots, c\}: \quad \sum_{j=1}^{n} u_{ij} > 0 \qquad \text{and} \qquad \forall j \in \{1, \ldots, n\}: \quad \sum_{i=1}^{c} u_{ij} = 1.$$

Here $u_{ij} \in [0, 1]$ is the degree of membership with which the data point $\vec{x}_j$ belongs to the i-th cluster, $\mathbf{c}_i$ is the prototype of the i-th cluster, and $d(\vec{x}_j, \mathbf{c}_i)$ is the distance between the data point $\vec{x}_j$ and the prototype $\mathbf{c}_i$. $\mathbf{C}$ denotes the set of all c cluster prototypes $\mathbf{c}_1, \ldots, \mathbf{c}_c$. The $c \times n$ matrix $\mathbf{U} = [u_{ij}]$ is called the fuzzy partition matrix and the parameter κ is called the *fuzzifier*. This parameter determines the "fuzziness" of the clustering result. With higher values for κ the boundaries between the clusters become softer, with lower values they get harder. Usually $\kappa = 2$ is chosen. The two constraints ensure that no cluster is empty and that the sum of the membership degrees for each datum equals 1, so that each data point has the same weight.

Unfortunately, the objective function J cannot be minimized directly w.r.t. both the cluster prototypes and the membership degrees. Therefore an iterative algorithm is used, which alternately optimizes the cluster prototypes and the membership degrees, each time keeping the other set of parameters fixed at their current values. The update formulae are derived by (partially) differentiating the objective function (extended by Lagrange multipliers to incorporate the constraints) w.r.t. the parameters to optimize and setting this derivative equal to zero. For the membership degrees one thus obtains the update rule

$$u_{ij} = \frac{\left(d^2(\vec{x}_j, \mathbf{c}_i)\right)^{\frac{1}{1-\kappa}}}{\sum_{k=1}^{c} \left(d^2(\vec{x}_j, \mathbf{c}_k)\right)^{\frac{1}{1-\kappa}}}, \qquad \text{or} \quad u_{ij} = \frac{d^{-2}(\vec{x}_j, \mathbf{c}_i)}{\sum_{k=1}^{c} d^{-2}(\vec{x}_j, \mathbf{c}_k)} \quad \text{for } \kappa = 2.$$

That is, the membership degrees basically reflect the relative inverse squared distance of the data point to the different cluster centers. This formula is independent of what kind of cluster prototype and what distance function is used.

3.2 Cluster Prototypes and Distance Functions

In order to obtain the update formulae for the cluster parameters, one has to distinguish between algorithms that use different kinds of cluster prototypes and, consequently, different distance functions. The most common fuzzy clustering algorithm is the fuzzy c-means algorithm [2]. It uses only cluster centers as prototypes, that is, $\mathbf{c}_i = (\vec{c}_i)$. Consequently, the distance function is simply the Euclidean distance between data point and cluster center, that is,

$$d^2_{\text{fcm}}(\vec{x}_j, \mathbf{c}_i) = (\vec{x}_j - \vec{c}_i)^\top (\vec{x}_j - \vec{c}_i),$$

which leads to the update rule

$$\vec{c}_i = \frac{\sum_{j=1}^{n} u_{ij}^{\kappa}\, \vec{x}_j}{\sum_{j=1}^{n} u_{ij}^{\kappa}}$$

for the cluster centers. Due to the Euclidean distance, the fuzzy c-means algorithm yields (hyper-)spherical clusters. Since there is no (cluster-specific) size parameter, all clusters have the same size.

A more flexible fuzzy clustering variant is the Gustafson–Kessel algorithm [13]. It yields (hyper-)ellipsoidal clusters by extending the cluster prototypes with (cluster-specific) fuzzy covariance matrices. That is, $\mathbf{c}_i = (\vec{c}_i, \mathbf{S}_i)$, where $\mathbf{S}_i$ is the fuzzy covariance matrix, which is normalized to determinant 1 in order to unify the cluster size. The distance function is

$$d^2_{\text{gk}}(\vec{x}_j, \mathbf{c}_i) = (\vec{x}_j - \vec{c}_i)^\top \mathbf{S}_i^{-1} (\vec{x}_j - \vec{c}_i),$$

that is, a (cluster-specific) Mahalanobis distance. This leads to exactly the same update rule for the cluster centers as for the fuzzy c-means algorithm (see above). The fuzzy covariance matrices are updated according to

$$\mathbf{S}_i = |\mathbf{\Sigma}_i|^{-\frac{1}{m}} \mathbf{\Sigma}_i \qquad \text{where} \qquad \mathbf{\Sigma}_i = \sum_{j=1}^{n} u_{ij}^{\kappa} (\vec{x}_j - \vec{c}_i)(\vec{x}_j - \vec{c}_i)^\top.$$

There is also a restricted version of the Gustafson–Kessel algorithm, in which only the variances of the individual dimensions are taken into account [23]. That is, this variant uses only the diagonal elements of the matrix $\mathbf{S}_i$ and all other matrix elements are set to zero. Since this means that the clusters are axes-parallel (hyper-)ellipsoids, it is common to refer to this variant as the axes-parallel Gustafson–Kessel algorithm.

Other approaches to incorporate shape and size information into the cluster prototypes have been studied, for example, in [12, 21, 22, 7, 8] and several other publications. In this paper, however, only the three approaches reviewed above are considered, that is, standard fuzzy clustering as well as axes-parallel and general Gustafson–Kessel clustering. It should be noted, though, that the general scheme of the update modifications described below is, in principle, applicable to such alternative approaches as well.

4 Transfer of the Update Modifications

In order to be able to apply the update modifications reviewed in Section 2, the (negated) gradient $-\nabla_w e(t)$ w.r.t. a (weight) parameter w is identified with the difference $\delta\theta(t)$ between two consecutive values of a cluster parameter θ (that is, a center coordinate or matrix element), as it can be computed with the standard center or covariance matrix update rules. In this way all modifications of a pure gradient descent scheme can be applied (almost) directly.

The following description assumes that only the update of the cluster parameters (centers and covariance matrices) is modified, while the membership degrees are updated in the standard way (that is, with the update rule stated in Section 3.1). However, it should be noted that, in principle, the same update modifications could be applied to the membership degrees as well.[1] The only problem with this option is that it requires (at least for most update modifications) to store the membership degrees as well as maybe several additional parameters. This is not necessary for an implementation of standard fuzzy clustering, where it suffices to have the membership degrees on a point by point basis, and can immediately "forget" the membership degrees for a data point once the corresponding parameter update terms have been computed.

4.1 Update Step Expansion

Even though the standard form of error backpropagation (see Section 2.2) corresponds, in a way, to the standard form of updating the cluster parameters, it still yields an update modification, because one may introduce a kind of "learning rate", which influences the step size, into the clustering process as well.

Of course, in fuzzy clustering this learning rate should be no less than 1, because a factor of 1 simply gives us the standard update step, which is optimal for fixed membership degrees. Factors larger than 1 can obviously be beneficial

[1] I am grateful to Thomas Runkler for pointing out this possibility to me, which I had overlooked initially due to the way in which I had implemented standard fuzzy clustering.

if one assumes that the next update step will change the parameters in roughly the same direction as the current one. However, for factors smaller than 1 to be useful, the standard update step would have to overshoot the (final) optimum in every step, so that reducing the step size gets us closer to the (final) optimum. Even though this is imaginable, it is not what can be observed in practice. In preliminary experiments, smaller factors always slowed down convergence.

As a consequence, this factor is only used to expand the step size, never to shrink it (that is, $\eta \geq 1$). The extended update rule thus reads

$$\Delta\theta(t) = \eta\delta\theta(t),$$

where $\delta\theta(t)$ is the change of the cluster parameter θ as it is computed with the standard update rule in step t. Experiments were conducted for $\eta \in [1.05, 2]$.

4.2 Momentum Term

The momentum term approach (see Section 2.3) can be transferred fairly directly, namely with the straightforward update rule

$$\Delta\theta(t) = \delta\theta(t) + \beta\Delta\theta(t-1) \qquad \text{with } \beta \in [0, 1).$$

In the experiments momentum coefficients $\beta \in [0.05, 0.9]$ were tried. However, relying on the same plausible reasoning as in the preceding section, the update step was bounded from below by $\delta\theta(t)$, that is, by a standard update step. In addition, preliminary experiments showed that it is beneficial to bound the update step size from above by a maximum that can be specified as a multiple of the standard update step $\delta\theta(t)$. This bounding improved robustness and even speed, since improper update steps have to be "corrected" by later update steps, which can lose time. In other words, $\Delta\theta(t)$ is clamped to $[\delta\theta(t), \eta_{\max}\delta\theta(t)]$ with $\eta_{\max} = 1.8$ (see the next section for a justification of this value).

4.3 Self-adaptive Step Expansion

Self-adaptive backpropagation (see Section 2.4) can also be transferred with ease, yielding a self-adaptive expansion factor. However, the update rule for the expansion factor update can be simplified to

$$\eta_\theta(t) = \begin{cases} \gamma^- \cdot \eta_\theta(t-1), & \text{if } \delta\theta(t) \cdot \delta\theta(t-1) < 0, \\ \gamma^+ \cdot \eta_\theta(t-1), & \text{if } \delta\theta(t) \cdot \delta\theta(t-1) > 0, \\ \eta_\theta(t-1), & \text{otherwise.} \end{cases}$$

That is, the expansion factor is allowed to grow immediately after a shrink (which was ruled out in Section 2.4), as this seemed to have no harmful effects. In the experiments $\gamma^- = 0.7$ and $\gamma^+ = 1.2$ were used, thus introducing a fairly strong tendency to increase the expansion factor and to keep it large. However, as this led to instabilities in some runs, the expansion factor was also bounded from above. In preliminary experiments an upper bound of 2 worked well in

most cases, but not in all. An upper bound of 1.8 was perfectly robust and was therefore used in all experiments. As an alternative one may consider smaller values of γ^-, and especially of γ^+, which also help to increase the robustness, but unfortunately also reduce the convergence speed. Finally, the expansion factor was bounded from below by 1, so that at least a standard update step is carried out. In other words, $\eta_\theta(t)$ is clamped to $[1, \eta_{\max}]$ with $\eta_{\max} = 1.8$.

4.4 Resilient Update

Resilient backpropagation (see Section 2.5) is transferred in the same way as self-adaptive backpropagation, with the same simplification of the update rule to

$$\Delta\theta(t) = \begin{cases} \gamma^- \cdot \Delta\theta(t-1), & \text{if } \delta\theta(t) \cdot \delta\theta(t-1) < 0, \\ \gamma^+ \cdot \Delta\theta(t-1), & \text{if } \delta\theta(t) \cdot \delta\theta(t-1) > 0, \\ \Delta\theta(t-1), & \text{otherwise.} \end{cases}$$

In the experiments the same values $\gamma^- = 0.7$ and $\gamma^+ = 1.2$ were used as for a self-adaptive expansion factor. In addition, $\Delta\theta(t)$ is clamped to $[\delta\theta(t), \eta_{\max}\delta\theta(t)]$ with $\eta_{\max} = 1.8$, based on the same arguments as in the preceding sections.

4.5 Quick Propagation Analog

Quick backpropagation (see Section 2.5) is also transferred directly, using the same parabola approximation scheme, which leads to the update rule

$$\Delta\theta(t) = \frac{\delta\theta(t)}{\delta\theta(t-1) - \delta\theta(t)} \cdot \Delta\theta(t-1).$$

The application of this rule is preceded by a check for a parabola that opens upwards. If the values indicate that the parabola opens downwards (and thus a local maximum would be approached with the above formula), a standard update step is carried out instead. As for quick backpropagation a growth factor α (set in the experiments to $\alpha = 2$) is used to ensure

$$|\Delta\theta(t)| \leq \alpha|\Delta\theta(t-1)|$$

and the update step is clamped to $[\delta\theta(t), \eta_{\max}\delta\theta(t)]$ with $\eta_{\max} = 1.8$ as explained before, although this update method seemed to be less susceptible to stability issues, possibly due to the bounding effect of the growth factor α.

4.6 Updating Covariance Matrices

Updating center coordinates with a modified update rule does not pose particular problems, since basically any values are feasible and there are no formal dependences between different coordinates. However, when updating the elements of covariance matrices, one has to take care of the fact that they have a bounded range of values and formally depend on each other. For example, the diagonal elements (the variances) must be positive and if s_x^2 and s_y^2 are the

variances for two attributes x and y and s_{xy} is the corresponding covariance, it must be $s_{xy}^2 \leq s_x^2 s_y^2$. (Actually even equality is not acceptable, since then the covariance matrix is singular.) Furthermore, there are certain dependences between different covariances, which hinder changing them freely.

As a consequence, applying the modified update rules directly to the elements of covariance matrices can yield matrices that are singular or even not positive definite, thus rendering it impossible to interpret them as covariance matrices. In order to cope with this problem, the result matrix of a modified update has to be checked whether it is positive definite, which can easily be done, for example, with Cholesky decomposition. (Note that this incurs basically no additional costs, because for the distance computations we need the inverse covariance matrix, which is most conveniently obtained with Cholesky decomposition anyway.) If a matrix is found to be not positive definite, it is replaced by the covariance matrix that results from an unmodified update. This procedure replaces the cruder approach of [6] where it was tried to "repair" an infeasible matrix resulting from a modified update by pushing it back into the feasible range by shifting its eigenvalues.

For diagonal covariance matrices (axes-parallel Gustafson–Kessel algorithm, see Section 3.2), the above check can be simplified and made more flexible as follows: since the variances are not formally dependent on each other, each of them can be treated independently. Hence for each variance it is checked whether the modified update assigns a positive value to it. If not, the variance is set to the value that results from a standard update step, thus ensuring consistent variances. Note that this procedure may lead to a matrix in which some variances have been obtained by a modified update, while others are obtained by a standard update. In contrast to this, the procedure described in the preceding paragraph can only lead to matrices in which either all entries are obtained by a modified update, or all entries are obtained by a standard update.

It should also be noted that all update modifications are carried out on the unnormalized covariance matrices $\mathbf{\Sigma}_i$, not on the normalized matrices $\mathbf{S}_i$ (which have determinant 1), since the normalization can have strange deteriorating effects. In contrast to this, in [6] the modifications were applied after normalization, which rendered the method less robust.

As an alternative to modify the update of the covariance matrices, one may consider an approach which works on the lower triangular matrix resulting from a Cholesky decomposition (the *Cholesky factor*) rather than the covariance matrices themselves.[2] This has the advantage, that the product of a lower triangular matrix with its own transpose yields at least a positive semi-definite matrix. However, there are (at least) two problems with such an approach.

In the first place, it is not enough to ensure that the resulting matrix is positive semi-definite. In practice, even positive-definite is not enough, but one must ensure that all eigenvalues are sufficiently large (that is, that the matrix is not *ill-conditioned*), because otherwise it is not useful for distance computations. Hence the fundamental problem may still persist with this approach.

[2] I am grateful to one of the anonymous reviewers for pointing out this possibility.

Secondly, which is actually a more serious issue, updates of the Cholesky factor can have fairly unintuitive effects on the resulting (covariance) matrix. Consider as an extreme case that the upper left element, which is simply the square root of the corresponding variance, becomes negative due to an overshoot in the update. Then the resulting variance (as its square) is, of course, still positive, but appears to have strangely "bounced back" from zero by an uncontrolled amount. It may even end up larger than the variance before the update, although the standard update set out to reduce it.

5 Convergence Evaluation

The convergence evaluation is based on the idea to compare the state of the cluster parameters in the current update step (intermediate clustering result) with the state that is reached in the convergence limit by the update process (final clustering result). For this purpose a measure for comparing clustering results is needed, for which so-called relative cluster evaluation measures are a well-known and straightforward choice (see Section 5.1). In order to obtain a general assessment, a proper way of averaging these measures over several trials is needed, so that average convergence curves can be drawn and a kind of convergence speed coefficient can be computed from them (see Section 5.2).

5.1 Comparing Clustering Results

Relative cluster evaluation measures compare two partitions of given data, which are described by (fuzzy) partition matrices. One of these partitions is a clustering result, while the other can be either also a clustering result or may be given by an independent classification or a user-defined grouping. In the latter case one also speaks of *external cluster evaluation measures* [14, 15], although the methods and measures used in both cases are usually the same.

The main problem of comparing two partitions of given data is how to relate the clusters of the one partition to the clusters of the other. A common solution to this problem is to find the best permutation of the rows of one partition matrix (mapping rows with the same index onto each other). That is, we find the best one-to-one mapping of the clusters from the two partitions. The actual comparison of the partition matrices can then be computed, for example, by a simple mean squared difference of their elements. That is, we compute

$$Q_{\text{diff}}\big(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}\big) = \min_{\pi \in \Pi(c)} \frac{1}{cn} \sum_{i=1}^{c} \sum_{j=1}^{n} \Big(u_{ij}^{(1)} - u_{\pi(i)j}^{(2)}\Big)^2 .$$

$\mathbf{U}^{(k)} = (u_{ij}^{(k)})_{1 \leq i \leq c, 1 \leq n}$ for $k = 1, 2$ are the two partition matrices to compare, n is the number of data points, c the number of clusters, and $\Pi(c)$ is the set of all permutations of the numbers 1 to c. The minimum captures formally what is meant by the "best" permutation (smallest mean squared difference).

Obviously, this measure becomes 0 if the two partition matrices coincide, which is the main reason why it is the only measure that was used in the experiments.

Alternatives to this simple and straightforward measure include the *(cross-classification) accuracy* and the F_1*-measure* [28], which can be extended to fuzzy partition matrices by introducing a t-norm as a parameter (see [9] for details). However, in the fuzzy case these alternatives do not have a fixed value for two identical partition matrices (the value depends on the cluster structure), which make them a lot less convenient to use, especially for the purpose at hand.

As an alternative to comparing partition matrices directly, one may first compute from each of them an $n \times n$ *coincidence matrix*, also called a *cluster connectivity matrix* [25], which states for each pair of data points whether they are assigned to the same cluster or not. Formally, such a coincidence matrix $\mathbf{\Psi} = (\psi_{jl})_{1 \leq j,l \leq n}$ is computed from a partition matrix $\mathbf{U}$ by

$$\psi_{jl} = \sum_{i=1}^{c} u_{ij} u_{il}.$$

Note that in the fuzzy case the product (which represents a conjunction of membership statements) may also be replaced by some other t-norm.

After coincidence matrices $\mathbf{\Psi}^{(1)}$ and $\mathbf{\Psi}^{(2)}$ are computed from the two partition matrices $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$, the comparison is carried out by computing statistics of the number of data point pairs that are in the same group in both partitions, in the same group in one, but in different groups in the other, or in different groups in both. Measures based on this approach are the *Rand statistic* or *Rand index* [26], the *Jaccard coefficient* or *Jaccard index* [19], the *Fowlkes–Mallows index* [11], and the *Hubert index* [18] (also known as the *Hubert–Arabie index* or the *adapted Rand statistic/index*), which in the fuzzy case may be parameterized by a t-norm (see [9] for details). In this paper, however, I refrain from using these measures as they are more time consuming to compute.

5.2 Assessing the Convergence Speed

The basic principle of the convergence evaluation applied here is to observe the development of the difference between the current state of the cluster parameters (intermediate clustering result) and the state that is reached in the convergence limit by the update process (final clustering result). Because a single trial is, of course, not significant, it is clear that several clustering trials have to be executed and averaged in order to obtain an assessment of the average or expected convergence speed. Since the difference to the final clustering result in terms of the mean squared difference of the membership degrees of the data points decays exponentially (linear convergence, see the next section), the decimal logarithms of the measure Q_{diff} are averaged over several trials. These averages are then plotted over a certain number of parameter update epochs.

In order to cope with the problem that the mean squared difference may actually vanish (at least in some trials), thus making it impossible to compute a logarithm in this case, the following argument was used: even with double

precision floating point numbers, the mean squared difference of the membership degrees can only be computed to very limited precision. The reason is that the computation necessarily also compares membership degrees close to one, but the smallest difference between two double precision floating point numbers in the vicinity of 1 is $\epsilon \approx 2.22 \cdot 10^{-16}$ (that is, there is no representable number between 1 and $1 - \epsilon$). Therefore -16 is used as a substitute for the logarithm in the case of a vanishing mean squared difference. However, this has a negligible influence on the results, since such cases occurred very rarely.

Another problem that has to be taken care of is the fact that due to local optima of the objective function the clustering result may not be unique. This makes it necessary to find for each trial the final result it approaches, rather than using a fixed final result. It should be noted in this respect, that if there are alternative final results, the standard update process and the one that uses a modified update rule may reach different results even if they are started from the same initialization: the update modifications may change the regions in the parameter space from which the clustering process converges to a given result (this could actually be observed in some experiments). Hence one cannot, even for a fixed initialization of the cluster parameters, simply compare to the final result of the standard clustering process. Fortunately, in most of the experiments the final result was unique, only in few cases there were two alternatives. In order to avoid having to run each trial until convergence, the possible final results were precomputed and stored in these cases, so that each trial had to be executed only for a limited number of update epochs.

As already mentioned, the difference to the final result decays basically exponentially, so that the average logarithms of the mean squared membership difference lie on (almost) straight lines (linear convergence) most of the time (at least after the update process has settled in after a few initial epochs—see the next section). As a consequence a kind of "convergence coefficient" can be computed by estimating the slope of these convergence lines. This provides a simple and expressive assessment of the benefits that can be achieved.

6 Experimental Results

The described update modifications were tested on four well-known data sets from the UCI machine learning repository [5]: iris (petal and sepal length and width of iris flowers, 150 cases, 4 attributes plus a class label), wine (chemical analysis of Italian wines from different regions, 179 cases, 13 attributes plus a class label), breast (Wisconsin breast cancer, tissue and cell measurements, 683 cases, 10 attributes plus a class label), and abalone (physical measurements of abalone clams, 4177 cases, 7 attributes plus a class label). In order to avoid scaling effects, the data was fully normalized, so that in each dimension the mean value was 0 and the standard deviation 1.

Since the listed data sets are actually classified (although the class labels were not used in the experiments), the number of clusters to find is known (iris: 3, wine: 3, breast: 2, abalone: 3). Therefore the clustering experiments

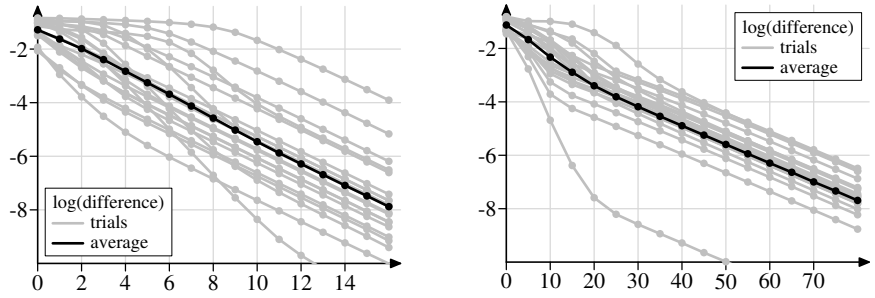


Figure 3: Clustering trials and their average for the iris data with 3 clusters; left: fuzzy c-means algorithm, right: general Gustafson–Kessel algorithm.

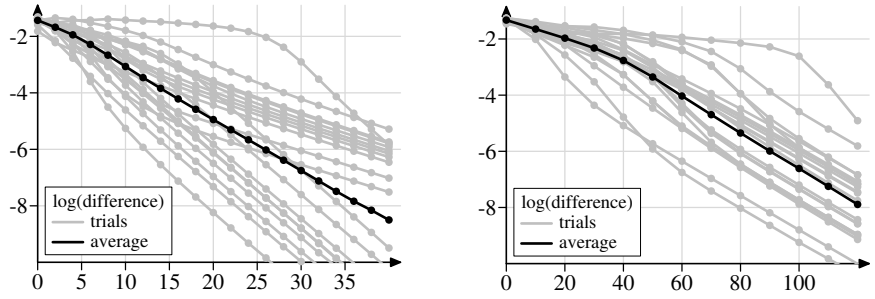


Figure 4: Clustering trials and their average for the wsel data with 6 clusters; left: fuzzy c-means algorithm, right: general Gustafson–Kessel algorithm.

were run using these numbers. In addition, experiments with 6 clusters were run for the abalone and the wine data set. Furthermore, for the wine data set additional experiments were carried out, in which only attributes 7, 10 and 13 were used, which are commonly considered to be the most expressive w.r.t. the class structure. In the following these experiments are referred to by the data set name "wsel" (wine data with a selection of attributes).

For each data set and each selection of clusters 20 trials were run for every parameterization of the update methods and then averaged. In order to give an impression of how the trials relate to the average, Figures 3 and 4 show them for both the standard fuzzy c-means algorithm and the standard Gustafson–Kessel algorithm for the iris data and the wsel data, respectively. Here an interesting observation can be made for the left diagram in Figure 4: clearly the trials form two groups, which show different convergence speed. The reason for this is that there are two possible final clustering results for the fuzzy c-means algorithm in this case, which are approached with different convergence speed.[3]

[3]The right diagram in Figure 4, for the general Gustafson–Kessel algorithm, does not show two groups, because in this case there is only one final clustering result.

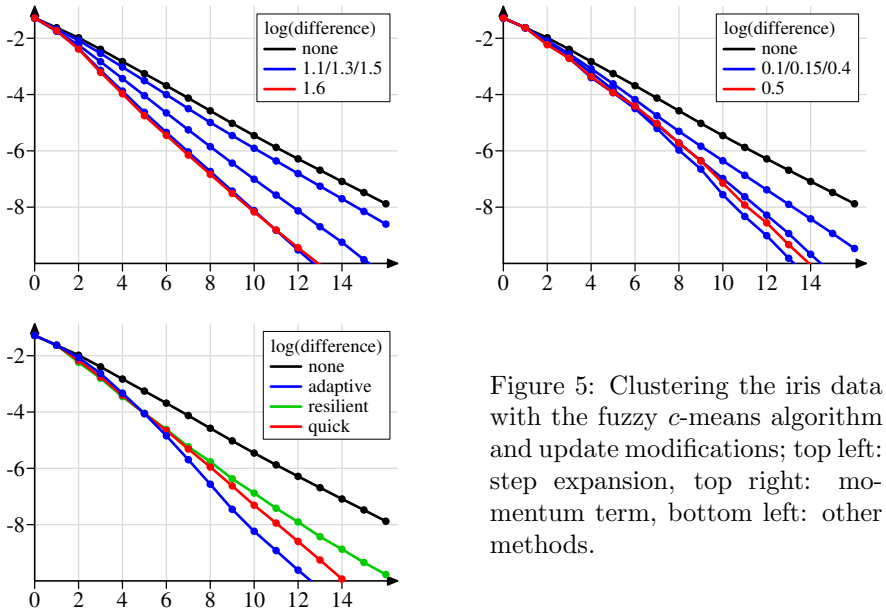


Figure 5: Clustering the iris data with the fuzzy c-means algorithm and update modifications; top left: step expansion, top right: momentum term, bottom left: other methods.

The diagrams also show that the Gustafson–Kessel algorithm takes a lot more epochs to converge to the same precision (note the different scales on the x-axis). This is, of course, not surprising, because the number of adaptable parameters is much higher for this algorithm. This observation can be made on all data sets, with the exception of the wine data (with all attributes) and 6 clusters, where all three algorithms (fuzzy c-means and axes-parallel and general Gustafson–Kessel) behave basically in the same way.[4]

Figure 5 shows the results for clustering the iris data with the fuzzy c-means algorithm. In each diagram the solid grey line corresponds to the standard update rule and thus provides the reference (it is identical to the black line in the left diagram of Figure 3). As can be seen, all of the update modifications improve over the standard approach, some of them considerably. For the step expansion and the momentum term, of course, the improvement depends on the value of the parameter (expansion factor η and momentum coefficient β), for which different values were tried ($\eta \in [1.05, 2.0]$ and $\beta \in [0.05, 0.9]$). The black lines refer to parameter values that yield a (mostly) stable behavior, while the grey dotted line indicates a limiting parameter, for which no further improvement or even a deterioration of the convergence speed can be observed. Since the parameters for the other three update methods were fixed ($\gamma^- = 0.7$ and $\gamma^+ = 1.2$ as well as $\alpha = 2$), as their dependence on these parameters is much less pronounced, there is only one curve for each of these methods.

[4]Due to space constraints it is, of course, not possible to show diagrams for all experiments in this paper. The full set of result diagrams is available at
http://www.borgelt.net/papers/nndvl.pdf (color version) and
http://www.borgelt.net/papers/nndvl_g.pdf (greyscale version).

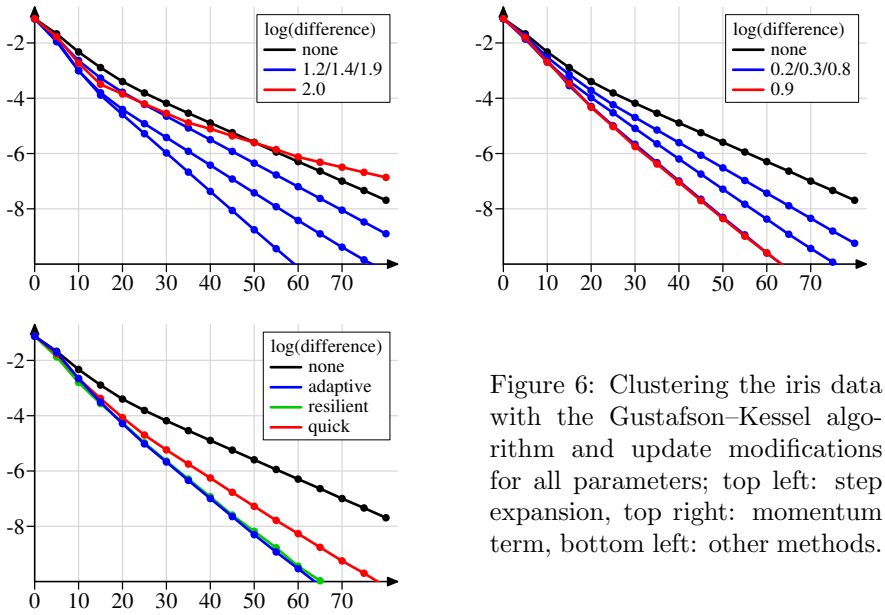


Figure 6: Clustering the iris data with the Gustafson–Kessel algorithm and update modifications for all parameters; top left: step expansion, top right: momentum term, bottom left: other methods.

Because the curves are (almost) straight lines, it is easy to compute a convergence coefficient as the (negative) slope of these lines. For the standard update rule this yields 0.42 (reduction of the decimal logarithm of the mean squared difference per update epoch). An expansion factor of 1.5, however, yields a convergence coefficient of 0.7, a momentum term of 0.3 a coefficient of 0.73. The best method, namely using an adaptive expansion factor, yields a coefficient of about 0.76, even though the convergence curve is not quite straight in this case. The quick propagation analog (0.65) and the resilient update method (0.53) are somewhat worse, but still yield good improvements. In summary, the best methods increase the convergence coefficient by a factor of roughly 1.8.

Figure 6 shows analogous results for the general Gustafson–Kessel algorithm (that is, with full covariance matrices), with the update modifications applied for all parameters. The speed increase achieved by the update modifications is almost as large: the convergence coefficient is raised from about 0.07 to about 0.12 for the best methods, thus yielding an increase by a factor of about 1.7.

It should be noted that in order to obtain these gains it is actually very important that the update of all cluster parameters (including the elements of the covariance matrices) is modified. If the modifications are applied only to the center coordinates, much lower gains result. This can be seen in Figure 7, which shows the results if the covariance matrices are updated only with the standard method. Although the update modifications still yield some improvement, the slope of the convergence curves changes only slightly.

Results on the wine data set with all attributes, clustered with the axes-parallel and the general Gustafson–Kessel algorithm in order to find 6 clusters

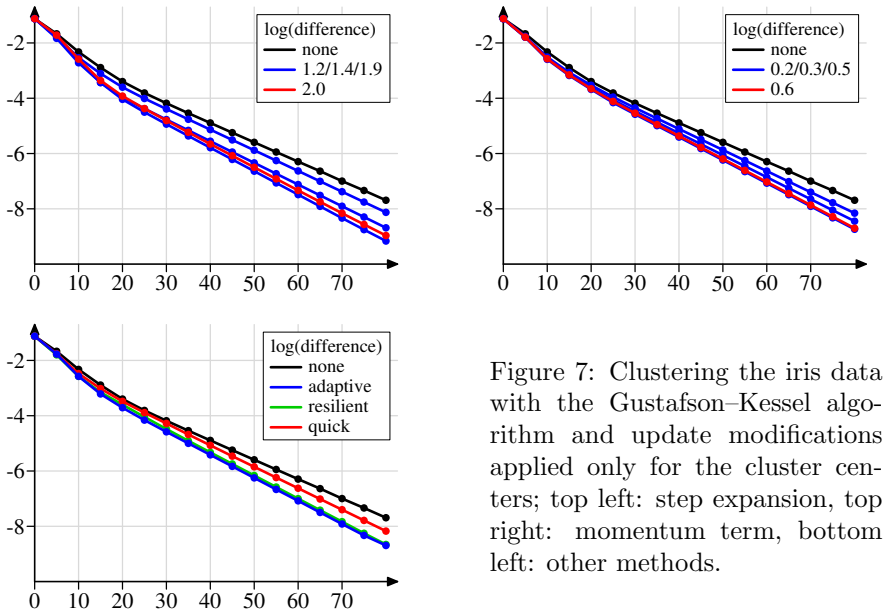
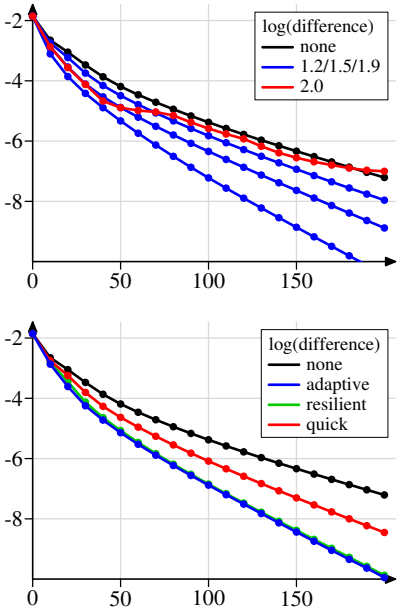


Figure 7: Clustering the iris data with the Gustafson–Kessel algorithm and update modifications applied only for the cluster centers; top left: step expansion, top right: momentum term, bottom left: other methods.

are shown in Figures 8 and 9. Furthermore, Figures 10 and 11 show results for the abalone data set, clustered with the fuzzy c-means algorithm and the general Gustafson–Kessel algorithm in order to find 3 clusters. These results confirm the observations made above: all update modifications lead to significant improvements by considerably increasing the slope of the convergence curves.

It is pleasing to observe that the different update modifications behave very similarly on the different data sets: the adaptive expansion factor is usually better than the resilient update (although the difference is often very small), which in turn is better than the quick propagation analog. Step expansion and momentum term can achieve equally large improvements if the parameters are properly chosen. Of course, choosing the parameters is tricky in practice, but judging from the experiments I conducted it is fairly safe to choose an expansion factor of about 1.5 or a momentum term of 0.3, since with these values the method is usually still fairly stable and not too far away from the optimum.

A much better option, however, is simply to rely on an adaptive expansion factor, which yields results that are either the best overall or only slightly worse than those that can be obtained with the best expansion factor or the best momentum term. This confirms the conclusion of [6]. However, the negative results about the resilient update method and the quick propagation analog cannot be confirmed: even though they do not outperform an adaptive expansion factor, it seems that the careful bounding of the update steps has eliminated the stability issues these methods suffered from. However, they are still worse than the adaptive expansion factor approach and even though in some cases they can reach its performance, they are not consistent in their behavior.



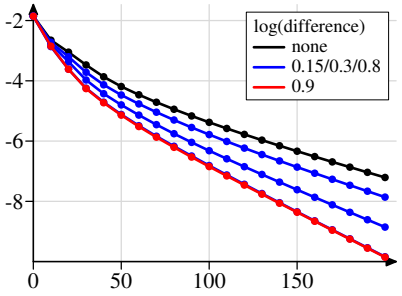
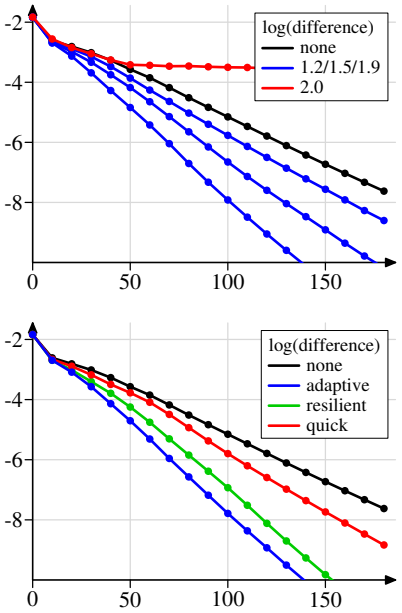


Figure 8: Clustering the wine data with the axes-parallel Gustafson–Kessel algorithm and 6 clusters; top left: step expansion, top right: momentum term, bottom left: other methods.



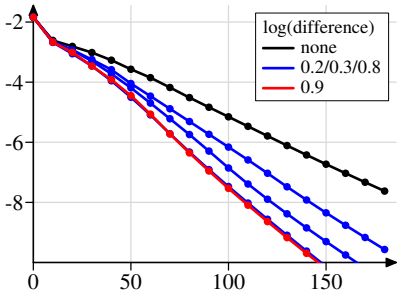
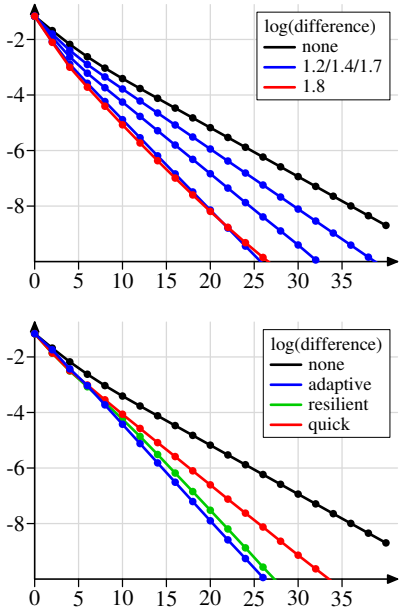


Figure 9: Clustering the wine data with the general Gustafson–Kessel algorithm and 6 clusters; top left: step expansion, top right: momentum term, bottom left: other methods.



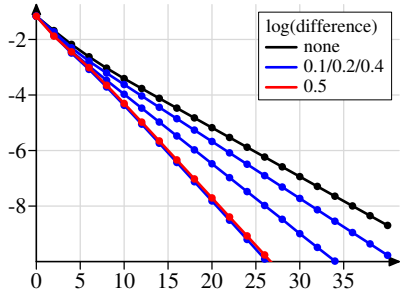
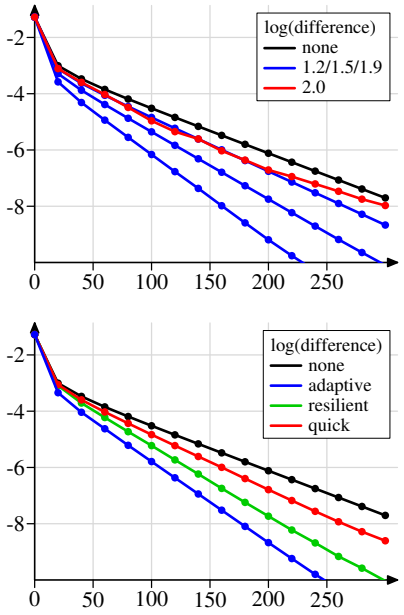


Figure 10: Clustering the abalone data with the fuzzy c-means algorithm and 3 clusters; top left: step expansion, top right: momentum term, bottom left: other methods.



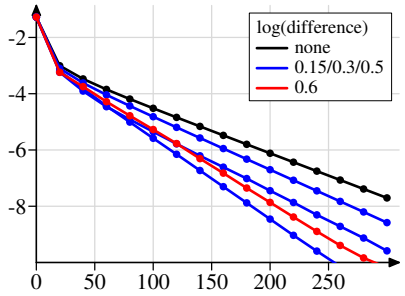


Figure 11: Clustering the abalone data with the general Gustafson–Kessel algorithm and 3 clusters; top left: step expansion, top right: momentum term, bottom left: other methods.

7 Summary and Conclusions

In this paper I explored in detail how ideas from (artificial) neural network training can be used to speed up fuzzy clustering. Particular attention was paid to how the computed update steps have to be checked and bounded—especially when updating covariance matrix elements—in order to achieve a robust procedure. In addition, I discussed a method to study the convergence behavior of the clustering process, which is based on cluster comparison measures. This method allows to highlight the benefits of the update modifications by enabling us to compute a convergence coefficient for the different update rules.

The experimental results show that basically all update modifications can significantly accelerate the clustering process: with these approaches the convergence speed, as measured by a coefficient that describes the slope of the convergence curve, can be increased by a factor that often exceeds a value of 1.5 for the best methods and parameterizations. The most highly recommendable update modification is clearly the self-adaptive expansion factor, which in several cases yields the best results, and in the rest comes fairly close to the optimum. It has the advantage that it is not necessary to chose a proper parameter (like for the step expansion and momentum term methods), since the default parameters yield excellent performance. Nevertheless step expansion and momentum term should not be dismissed, since stable parameter ranges could be identified and these methods need fewer auxiliary values to be stored.

Software and Diagrams

The programs (written in C), the data sets, and a shell script (for bash) that have been used to carry out the experiments are available at:

http://www.borgelt.net/cluster.html

The full set of diagrams for the experiments is available at:

http://www.borgelt.net/papers/nndvl.pdf (color version)
http://www.borgelt.net/papers/nndvl_g.pdf (greyscale version)

References

[1] J.A. Anderson. *An Introduction to Neural Networks.* MIT Press, Cambridge, MA, USA 1995

[2] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, New York, NY, USA 1981

[3] J.C. Bezdek and S.K. Pal, eds. *Fuzzy Models for Pattern Recognition.* IEEE Press, Piscataway, NJ, USA 1992

[4] J.C. Bezdek, J. Keller, R. Krishnapuram, and N.R. Pal. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing.* Kluwer, Dordrecht, Netherlands 1999

[5] C.L. Blake and C.J. Merz. UCI Repository of Machine Learning Databases. Department of Information and Computer Science, University of California, Irvine, CA, USA 1998.
`http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[6] C. Borgelt and R. Kruse. Speeding Up Fuzzy Clustering with Neural Network Techniques. *Proc. 12th IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'03, St. Louis, MO, USA)*. IEEE Press, Piscataway, NJ, USA 2003

[7] C. Borgelt and R. Kruse. Shape and Size Regularization in Expectation Maximization and Fuzzy Clustering. *Proc. 8th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD 2004, Pisa, Italy)*, 52–62. Springer-Verlag, Berlin, Germany 2004

[8] C. Borgelt. *Prototype-based Classification and Clustering.* Habilitation thesis, Otto-von-Guericke-University of Magdeburg, Germany 2005

[9] C. Borgelt. Resampling for Fuzzy Clustering. *Int. Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 15(5):595–614. World Scientific, Hackensack, NJ, USA 2007

[10] S.E. Fahlman. An Empirical Study of Learning Speed in Backpropagation Networks. In: [31].

[11] E.B. Fowlkes and C.L. Mallows. A Method for Comparing Two Hierarchical Clusterings. *Journal of the American Statistical Association* 78(383):553–569. American Statistical Association, Alexandria, VA, USA 1983

[12] I. Gath and A.B. Geva. Unsupervised Optimal Fuzzy Clustering. *IEEE on Trans. Pattern Analysis and Machine Intelligence (PAMI)* 11:773–781. IEEE Press, Piscataway, NJ, USA 1989. Reprinted in [3], 211–218

[13] E.E. Gustafson and W.C. Kessel. Fuzzy Clustering with a Fuzzy Covariance Matrix. *(IEEE CDC, San Diego, CA)*, pp. 761-766, IEEE Press, Piscataway, NJ, USA 1979

[14] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering Validity Checking Methods: Part I. *ACM SIGMOD Record* 31(2):40–45. ACM Press, New York, NY, USA 2002

[15] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering Validity Checking Methods: Part II. *ACM SIGMOD Record* 31(3):19–27. ACM Press, New York, NY, USA 2002

[16] S. Haykin. *Neural Networks — A Comprehensive Foundation.* Prentice-Hall, Upper Saddle River, NJ, USA 1994

[17] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy Cluster Analysis.* J. Wiley & Sons, Chichester, United Kingdom 1999

[18] L. Hubert and P. Arabie. Comparing Partitions. *Journal of Classification* 2:193–218. Classification Society of North America, USA 1985

[19] P. Jaccard. Nouvelles recherches sur la distribution florale. *Bulletin Société Vaudoisedes Sciences Naturelles* 44:223–270. Société Vaudoisedes Sciences Naturelles, Paris, France 1908

[20] R.A. Jakobs. Increased Rates of Convergence Through Learning Rate Adaption. *Neural Networks* 1:295–307. Pergamon Press, Oxford, United Kingdom 1988

[21] A. Keller and F. Klawonn. Fuzzy Clustering with Weighting of Data Variables. *Int. J. of Uncertainty, Fuzziness and Knowledge-based Systems* 8:735-746. World Scientific, Hackensack, NJ, USA 2000

[22] A. Keller and F. Klawonn. Adaptation of Cluster Sizes in Objective Function Based Fuzzy Clustering. In [24], 181–199.

[23] F. Klawonn and R. Kruse. Automatic Generation of Fuzzy Controllers by Fuzzy Clustering. *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (Vancouver, Canada)*, 2040–2045. IEEE Press, Piscataway, NJ, USA 1995

[24] C.T. Leondes, ed. *Database and Learning Systems IV*. CRC Press, Boca Raton, FL, USA 2003

[25] E. Levine and E. Domany. Resampling Method for Unsupervised Estimation of Cluster Validity. *Neural Computation* 13:2573–2593. MIT Press, Cambridge, MA, USA 2001

[26] W.M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association* 66:846–850. American Statistical Association, Alexandria, VA, USA 1971

[27] M. Riedmiller and H. Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *Int. Conf. on Neural Networks (ICNN-93, San Francisco, CA)*, 586–591. IEEE Press, Piscataway, NJ, USA 1993

[28] C.J. van Rijsbergen. *Information Retrieval*. Butterworth, London, United Kingdom 1979

[29] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning Representations by Back-Propagating Errors. *Nature* 323:533–536. Nature Publishing Group, Basingstoke, United Kingdom 1986

[30] T. Tollenaere. SuperSAB: Fast Adaptive Backpropagation with Good Scaling Properties. *Neural Networks* 3:561–573. Pergamon Press, Oxford, United Kingdom 1990

[31] D. Touretzky, G. Hinton, and T. Sejnowski, eds. *Proc. Connectionist Models Summer School (Carnegie Mellon University, Pittsburgh, PA)*. Morgan Kaufman, San Mateo, CA, USA 1988

[32] A. Zell. *Simulation Neuronaler Netze*. Addison-Wesley, Bonn, Germany 1994