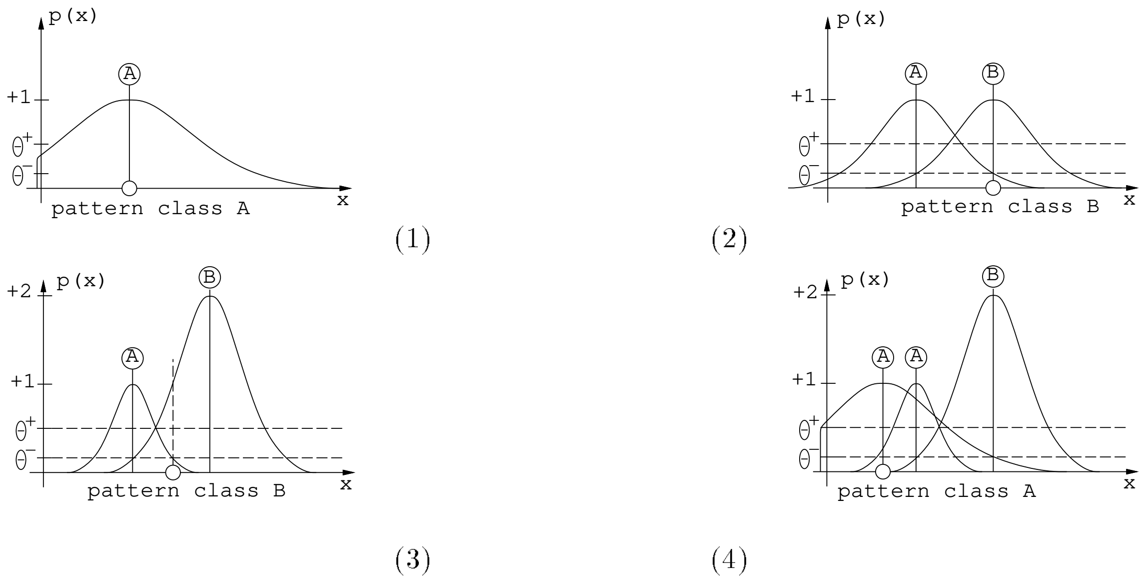

Figure 9.11: An example of the DDA-Algorithm: (1) a pattern of class A is encountered and a new RBF is created; (2) a training pattern of class B leads to a new prototype for class B and shrinks the radius of the existing RBF of class A; (3) another pattern of class B is classified correctly and shrinks again the prototype of class A; (4) a new pattern of class A introduces another prototype of that class.

9.11.2 Using RBF–DDA in SNNS

The implementation of the DDA-Algorithm always uses the Gaussian activation function `Act_RBF_Gaussian` in the hidden layer. All other activation and output functions are set to `Act_Identity` and `Out_Identity`, respectively.

The *Learning function* has to be set to `RBF-DDA`. No *Initialization* or *Update* functions are needed.

The algorithm takes three arguments that are set in the first three fields of the `LEARN` row in the *control panel*. These are θ^+, θ^- $(0 < \theta^- \leq \theta^+ < 1)$ and the maximum number of RBF units to be diplayed in one row. This last item allows the user to control the appearance of the network on the screen and has no influence on the performance. Specifying 0.0 leads to the default values $\theta^+ = 0.4, \theta^- = 0.2$ and to a maximum number of 20 RBF units displayed in a row.

Training of an RBF starts with either:

- an **empty** network, i.e. a network consisting only of input and output units. No connections between input and output units are required, hidden units will be added during training. This can easily be generated with the tool `BIGNET` (choice `FEED FORWARD`)
- a **pretrained** network already containing RBF units generated in an earlier run of

RBF-DDA (all networks not complying with the specification of an RBF-DDA will be rejected).

After having loaded a training pattern set, a *learning-epoch* can be started by pressing the ALL button in the *control panel*. At the beginning of each epoch, the weights between the hidden and the output layer are automatically set to zero. Note that the resulting RBF and the number of required learning epochs can vary slightly depending on the order of the training patterns. If you train using a single pattern (by pressing the SINGLE button) keep in mind that every training step increments the weight between the RBF unit of the correct class covering that pattern and its corresponding output unit. The end of the training is reached when the network structure does not change any more and the Mean Square Error (MSE) stays constant from one epoch to another.

The first desired value in an output pattern that is greater than 0.0 will be assumed to represent the class this pattern belongs to; only one output may be greater than 0.0. If there is no such output, training is still executed, but no new prototype for this pattern is commited. All existing prototypes are shrunk to avoid coverage of this pattern, however. This can be an easy way to define an "error"–class without trying to model the class itself.

9.12 ART Models in SNNS

This section will describe the use of the three ART models ART1, ART2 and ARTMAP, as they are implemented in SNNS. It will not give detailed information on the Adaptive Resonance Theory. You should already know the theory to be able to understand this chapter. For the theory the following literature is recommended:

[CG87a] Original paper, describing ART1 theory.

[CG87b] Original paper, describing ART2 theory.

[CG91] Original paper, describing ARTMAP theory.

[Her92] Description of theory, implementation and application of the ART models in SNNS (in German).

There will be one subsection for each of the three models and one subsection describing the required topologies of the networks when using the ART learning-, update- or initialization-functions. These topologies are rather complex. For this reason the network creation tool **BigNet** has been extended. It now offers an easy way to create ART1, ART2 and ARTMAP networks according to your requirements. For a detailed explanation of the respective features of **BigNet** see chapter 7.

9.12.1 ART1

9.12.1.1 Structure of an ART1 Network

The topology of ART1 networks in SNNS has been chosen to to perform most of the ART1 algorithm within the network itself. This means that the mathematics is realized in the

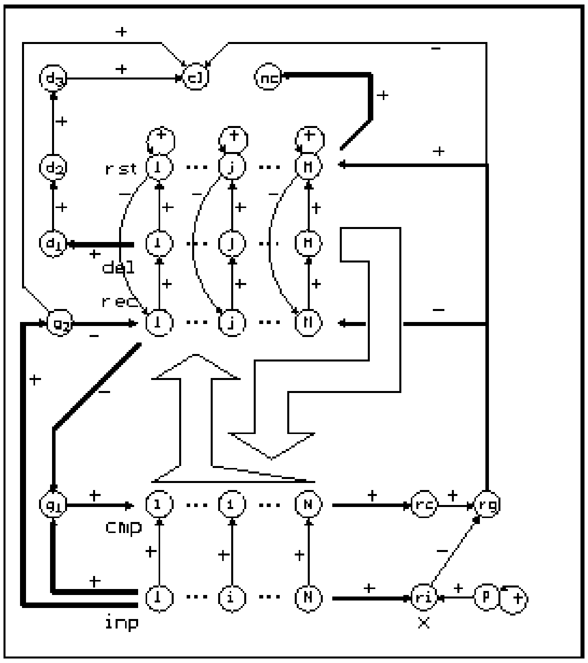

Figure 9.12:
Structure of an ART1 network in SNNS. Thin arrows represent a connection from one unit to another. Fat arrows which go from a layer to a unit indicate that each unit of the layer is connected to the target unit. Similarly a fat arrow from a unit to a layer means that the source unit is connected to each of the units in the target layer. The two big arrows in the middle represent the full connection between comparison and recognition layer and the one between delay and comparison layer, respectively.

activation and output functions of the units. The idea was to keep the propagation and training algorithm as simple as possible and to avoid procedural control components.

In figure 9.12.1.1 the units and links of ART1 networks in SNNS are displayed.

The F_0 or input layer (labeled **inp** in figure 9.12.1.1) is a set of N input units. Each of them has a corresponding unit in the F_1 or comparison layer (labeled **cmp**). The M elements in the F_2 layer are split into three levels. So each F_2 element consists of three units. One recognition (**rec**) unit, one delay (**del**) unit and one local reset (**rst**) unit. These three parts are necessary for different reasons. The recognition units are known from the theory. The delay units are needed to synchronize the network correctly[8]. Besides, the activated unit in the delay layer shows the winner of F_2. The job of the local reset units is to block the actual winner of the recognition layer in case of a reset.

Finally, there are several special units. The **cl** unit gets positive activation when the input pattern has been successfully classified. The **nc** unit indicates an unclassifiable pattern, when active. The gain units $\mathbf{g}_1$ and $\mathbf{g}_2$ with their known functions and at last the units **ri** (reset input), **rc** (reset comparison), **rg** (reset general) and ρ(vigilance), which realize the reset function.

For an exact definition of the required topology for ART1 networks in SNNS see section 9.12.4

[8]This is only important for the chosen realization of the ART1 learning algorithm in SNNS

9.12.1.2 Using ART1 Networks in SNNS

To use an ART1 network in SNNS several functions have been implemented: one to initialize the network, one to train it and two different update functions to propagate an input pattern through the net.

ART1 Initialization Function First the ART1 initialization function `ART1_Weights` has to be selected from the list of initialization functions.

`ART1_Weights` is responsible to set the initial values of the trainable links in an ART1 network. These links are the ones from F_1 to F_2 and the ones from F_2 to F_1 respectively.

The $F_2 \rightarrow F_1$ links are all set to 1.0 as described in [CG87a]. The weights of the links from F_1 to F_2 are a little more difficult to explain. To assure that in an initialized network the F_2 units will be used in their index order, the weights from F_1 to F_2 must decrease with increasing index. Another restriction is, that each link-weight has to be greater than 0 and smaller than $1/N$. Defining α_j as a link-weight from a F_1 unit to the jth F_2 unitthis yields

$$0 < \alpha_M < \alpha_{M-1} < \ldots < \alpha_1 \leq \frac{1}{\beta + N}. \tag{9.3}$$

To get concrete values, we have to decrease the fraction on the right side with increasing index j and assign this value to α_j. For this reason we introduce the value η and we obtain

$$\alpha_j \equiv \frac{1}{\beta + (1 + j\eta)N}. \tag{9.4}$$

η is calculated out of a new parameter γ and the number of F_2 units M:

$$\eta \equiv \frac{\gamma}{M}. \tag{9.5}$$

So we have two parameters for `ART1_Weights`: β and γ. For both of them a value of 1.0 is useful for the initialization. The first parameter of the initialization function is β, the second one is γ. Having chosen β and γ one must press the INIT-button to perform initialization.

The parameter β is stored in the bias field of the unit structure to be accessible to the learning function when adjusting the weights.

One should always use `ART1_Weights` to initialize ART1 networks. When using another SNNS initialization function the behavior of the simulator during learning is not predictable, because not only the trainable links will be initialized, but also the fixed weights of the network.

ART1 Learning Function To train an ART1 network select the learning function `ART1`. To start the training of an ART1 network, choose the vigilance parameter ρ (e.g.: 0.1) as first value in both `LEARN` and `UPDATE` row of the control panel. Parameter β, which is also needed to adjust the trainable weights between F_1 and F_2, has already been specified as initialization parameter. It is stored in the bias field of the unit structure and read out by `ART1` when needed.

ART1 Update Functions To propagate a new pattern through an ART1 network without adjusting weights, i.e. to classify a pattern, two different update functions have been implemented:

- `ART1_Stable` and
- `ART1_Synchronous`.

Like the learning function, both of the update functions only take the vigilance value ρ as parameter. It has to be entered in the control panel, the line below the parameters for the learning function. The difference between the two update functions is the following:

`ART1_Stable` propagates a pattern until the network is stable, i.e. either the **cl** unit or the **nc** unit is active. To use this update function, you can use the TEST-button of the control panel. The next pattern is copied to the input units and propagated completely through the net, until a stable state is reached.

`ART1_Synchronous`, performs just one propagation step with each call. To use this function you have to press the RESET-button to reset the net to a defined initial state, where each unit has its initial activation value. Then copy a new pattern into the input layer, using the buttons < and >. Now you can choose the desired number of propagation steps that should be performed, when pressing the STEP-button (default is 1). With this update function it is very easy to observe how the ART1 learning algorithm does its job.

So use `ART1_Synchronous`, to *trace* a pattern through a network, `ART1_Stable` to propagate the pattern until a stable state is reached.

9.12.2 ART2

9.12.2.1 Structure of an ART2 Network

The realization of ART2 differs from the one of ART1 in its basic idea. In this case the network structure would have been too complex, if mathematics had been implemented within the network to the same degree as it has been done for ART1. So here more of the functionality is in the control program. In figure 9.12.2.1 you can see the topology of an ART2 network as it is implemented in SNNS.

All the units are known from the ART2 theory, except the **rst** units. They have to do the same job for ART2 as for ART1 networks. They block the actual winner in the recognition layer in case of reset. Another difference between the ART2 model described in [CG87b]

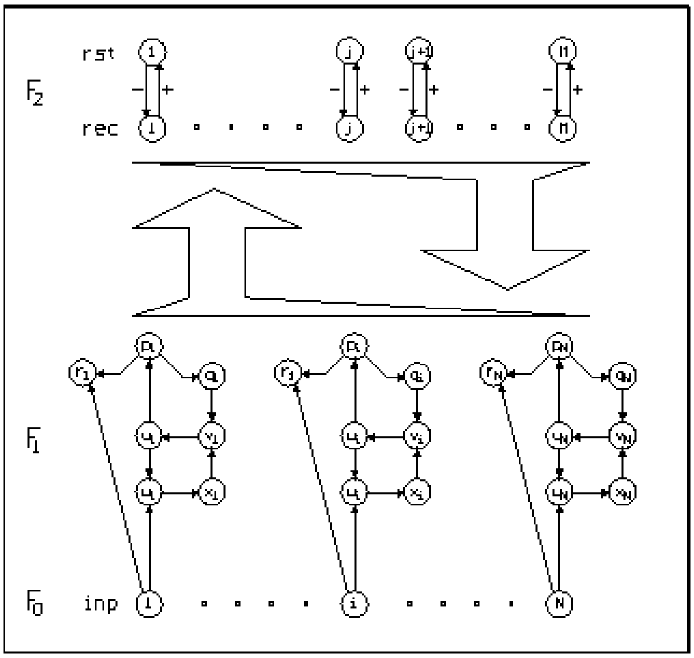

Figure 9.13:
Structure of an ART2 network in SNNS. Thin arrows represent a connection from one unit to another. The two big arrows in the middle represent the full connectivity between comparison and recognition layer and the one between recognition and comparison layer, respectively.

and the realization in SNNS is, that originally the units $\mathbf{u}_i$ have been used to compute the error vector $\mathbf{r}$, while this implementation takes the input units instead.

For an exact definition of the required topology for ART2 networks in SNNS see section 9.12.4

9.12.2.2 Using ART2 Networks in SNNS

As for ART1 there are an initialization function, a learning function and two update functions for ART2. To initialize, train or test an ART2 network, these functions have to be used. The description of the handling, is not repeated in detail in this section since it is the same as with ART1. Only the parameters for the functions will be mentioned here.

ART2 Initialization Function For an ART2 network the weights of the top-down-links ($F_2 \rightarrow F_1$ links) are set to 0.0 according to the theory ([CG87b]).

The choice of the initial bottom-up-weights is determined as follows: if a pattern has been trained, then the next presentation of the same pattern must not generate a new winning class. On the contrary, the same F_2 unit should win, with a higher activation than all the other recognition units.

This implies that the norm of the initial weight-vector has to be smaller than the one it has after several training cycles. If J $(1 \leq J \leq M)$ is the actual winning unit in F_2, then equation 9.6 is given by the theory:

$$||\mathbf{z}^J|| \rightarrow \left|\left|\frac{\mathbf{u}}{1-d}\right|\right| = \frac{1}{1-d}, \tag{9.6}$$

where $\mathbf{z}^J$ is the the weight vector of the links from the F_1 units to the Jth F_2 unit and where d is a parameter, described below.

If all initial values $z_{ij}(0)$ are presumed to be equal, this means:

$$z_{ij}(0) \leq \frac{1}{(1-d)\sqrt{N}} \qquad \forall\, 1 \leq i \leq N, 1 \leq j \leq M. \tag{9.7}$$

If equality is chosen in equation 9.7, then ART2 will be as sensitive as possible.

To transform the inequality 9.7 to an equation, in order to compute values, we introduce another parameter γ and get:

$$z_{ij}(0) = \frac{1}{\gamma(1-d)\sqrt{N}} \qquad \forall\, 1 \leq i \leq N, 1 \leq j \leq M, \tag{9.8}$$

where $\gamma \geq 1$.

To initialize an ART2 network, the function `ART2_Weights` has to be selected. Specify the parameters d and γ as the first and second initialization parameter. (A description of parameter d is given in the subsection on the ART2 learning function.) Finally press the INIT -button to initialize the net.

WARNING! You should always use `ART2_Weights` to initialize ART2 networks. When using another SNNS initialization function the behavior of the simulator during learning is not predictable, because not only the trainable links will be initialized, but also the fixed weights of the network.

ART2 Learning Function For the ART2 learning function `ART2` there are various parameters to specify. Here is a list of all parameters known from the theory:

- ρ Vigilance parameter. (first parameter of the learning and update function). ρ is defined on the interval $0 \leq \rho \leq 1$. For some reason, described in [Her92] only the following interval makes sense: $\frac{1}{2}\sqrt{2} \leq \rho \leq 1$.
- a Strength of the influence of the lower level in F_1 by the middle level. (second parameter of the learning and update function). Parameter a defines the importance of the expection of F_2, propagated to F_1: $a > 0$. Normally a value of $a \gg 1$ is chosen to assure quick stabilization in F_1.
- b Strength of the influence of the middle level in F_1 by the upper level. (third parameter of the learning and update function). For parameter b things are similar to parameter a. A high value for b is even more important, because otherwise the network could become instable ([CG87b]). $b > 0$, normally:$b \gg 1$.
- c Part of the length of vector $\mathbf{p}$ (units $\mathbf{p}_1 \ldots \mathbf{p}_N$) used to compute the error. (fourth parameter of the learning and update function). Choose c within $0 < c < 1$.

d Output value of the F_2 winner unit. You won't have to pass d to `ART2`, because this parameter is already needed for initialization. So you have to enter the value, when initializing the network (see subsection on the initialization function). Choose d within $0 < d < 1$. The parameters c and d are dependent on each other. For reasons of quick stabilization c should be chosen as follows: $0 < c \ll 1$. On the other hand c and d have to fit the following condition: $0 \ll \frac{cd}{1-d} \leq 1$.

e Prevents from division by zero. Since this parameter does not help to solve essential problems, it is implemented as a fix value within the SNNS source code.

Θ Kind of threshold. For $0 \leq x, q \leq \Theta$ the activation values of the units $\mathbf{x}_i$ and $\mathbf{q}_i$ only have small influence (if any) on the middle level of F_1. The output function f of the units $\mathbf{x}_i$ and $\mathbf{q}_i$ takes Θ as its parameter. Since this noise function is continuously differentiable, it is called `Out_ART2_Noise_ ContDiff` in SNNS. Alternatively a piecewise linear output function may be used. In SNNS the name of this function is `Out_ART2_Noise_PLin`. Choose Θ within $0 \leq \Theta < 1$.

To train an ART2 network, make sure, you have chosen the learning function `ART2`. As a first step initialize the network with the initialization function `ART2_Weights` described above. Then set the five parameters ρ, a, b, c and Θ, in the parameter windows 1 to 5 in both the `LEARN` and `UPDATE` lines of the control panel. Example values are 0.9, 10.0, 10.0, 0.1, and 0.0. Then select the number of learning cycles, and finally use the buttons SINGLE and ALL to train a single pattern or all patterns at a time, respectively.

ART2 Update Functions Again two update functions for ART2 networks have been implemented:

- `ART2_Stable`
- `ART2_Synchronous`.

Meaning and usage of these functions are equal to their equivalents of the ART1 model. For both of them the parameters ρ, a, b, c and Θ have to be defined in the row of update parameters in the control panel.

9.12.3 ARTMAP

9.12.3.1 Structure of an ARTMAP Network

Since an ARTMAP network is based on two networks of the ART1 model, it is useful to know how ART1 is realized in SNNS. Having taken two of the ART1 (ART^a and ART^b) networks as they were defined in section 9.12.1, we add several units that represent the MAP field. The connections between ART^a and the MAP field, ART^b and the MAP field, as well as those within the MAP field are shown in figure 9.14. The figure lacks the full connection from the F_2^a layer to the F^{ab} layer and those from each F_2^b unit to its respective F^{ab} unit and vice versa.