A neural model for multi-expert architectures

Marc Toussaint   January 29, 2002

Institut für Neuroinformatik, Ruhr-Universität Bochum, ND 04, 44780 Bochum—Germany
mt@neuroinformatik.ruhr-uni-bochum.de

Abstract

We present a generalization of conventional artificial neural networks that allows for a functional equivalence to multi-expert systems. The new model provides an architectural freedom going beyond existing multi-expert models and an integrative formalism to compare and combine various techniques of learning. (We consider gradient, EM, reinforcement, and unsupervised learning.) Its uniform representation aims at a simple genetic encoding and evolutionary structure optimization of multi-expert systems. This paper contains a detailed description of the model and learning rules, empirically validates its functionality, and discusses future perspectives.

I   Introduction

When using multi-expert architectures for modeling behavior or data, the motivation is the separation of the stimulus or data space into disjoint regimes one which separate models (experts) are applied (Jacobs 1999; Jacobs, Jordan, & Barto 1990). The idea is that experts responsible for only a limited regime can be smaller and more efficient, and that knowledge from one regime should not be extrapolated onto another regime, i.e., optimization on one regime should not interfere with optimization on another. Several arguments indicate that this kind of adaptability cannot be realized by a single conventional neural network (Toussaint 2002). Roughly speaking, for conventional neural networks the optimization of a response in one regime always interferes with responses in other regimes because they depend on the same parameters (weights), which are not separated into disjoint experts.

To realize a seperation of the stimulus space one could rely on the conventional way of implementing multi-experts, i.e., allow neural networks for the implementation of expert modules and use external, often more abstract types of gating networks to organize the interaction between these modules. Much research is done in this direction (Bengio & Frasconi 1994; Cacciatore & Nowlan 1994; Jordan & Jacobs 1994; Rahman & Fairhurst 1999; Ronco, Gollee, & Gawthrop 1997). The alternative we want to propose here is to introduce a neural model that is capable to represent systems that are functionally equivalent to multi-expert systems within a single integrative network. This network does not explicitly distinguish between expert and gating modules and generalizes conventional neural networks by introducing a counterpart for gating interactions. What is our motivation for such a new representation of multi-expert systems?

- First, our representation allows much more and qualitatively new architectural freedom. E.g., gating neurons may interact with expert neurons; gating neurons can be a part of experts. There is no restriction with respect to serial, parallel, or hierarchical architectures—in a much more general sense as proposed in (Jordan & Jacobs 1994).

- Second, our representation allows in an intuitive way to combine techniques from various learning theories. This includes gradient descent, unsupervised learning methods like Hebb learning or the Oja rule, and an EM-algorithm that can be transferred from classical gating-learning theories (Jordan & Jacobs 1994). Further, the interpretation of a specific gating as an action exploits the realm of reinforcement learning, in particular Q-learning and (though not discussed here) its TD and TD(λ) variants (Sutton & Barto 1998).

- Third, our representation makes a simple genetic encoding of such architectures possible. There already exist various techniques for evolutionary structure optimization of networks (see (Yao 1999) for a review). Applied on our representation, they become techniques for the evolution of multi-expert architectures.

After the rather straight-forward generalization of neural interactions necessary to realize gatings (section II), we will discuss in detail different learning methods in section III. The empirical study in section IV compares the different interactions and learning mechanisms on a test problem similar to the one discussed by Jacobs et al. (1990).

II Model definition

Conventional multi-expert systems. Assume the system has to realize a mapping from an input space X to an output space Y. Typically, an m-expert architecture consists of a gating function $\hat{g} : X \rightarrow [0,1]^m$ and m expert functions $f_i : X \rightarrow Y$ which are combined by the softmax linear combination:

$$y = \sum_{i=1}^{m} g_i f_i(x) , \quad g_i = \frac{e^{\beta \hat{g}_i(x)}}{\sum_{j=1}^{m} e^{\beta \hat{g}_j(x)}} , \qquad (1)$$

where x and y are input and output, and β describes the "softness" of this winner-takes-all type competition between the experts, see Figure 1. The crucial question becomes how to train the gating. We will discuss different methods in the next section.

Neural implementation of multi-experts. We present a single neural system that has at least the capabilities of a multi-expert architecture of several neural networks. Basically we provide additional competitive and gating interactions, for an illustration compare Figure 1 and Figure 2-B. More formally, we introduce the model as follows:

The architecture is given by a directed, labeled graph of neurons (i) and links (ij) from (j) to (i), where $i, j = 1..n$. Labels of links declare if they are ordinary, competitive or gating connections. Labels of neurons declare their type of activation function. With every neuron (i), an activation state (output value) $z_i \in [0,1]$ is associated. A neuron (i) collects two terms of excitation x_i and g_i given by

$$x_i = \sum_{(ij)} w_{ij} z_j + w_i \qquad (2)$$

$$g_i = \begin{cases} 1 & \text{if } N_i = 0 \\ \frac{1}{N_i} \sum_{(ij)^g} z_j & \text{else} \end{cases} , \quad N_i = \sum_{(ij)^g} 1 , \qquad (3)$$

where $w_{ij}, w_i \in \mathbb{R}$ are weights and bias associated with the links (ij) and the neuron (i), respectively. The second excitatory term g_i has the meaning of a gating term and is induced by N_i g-labeled links $(ij)^g$.

In case there are no c-labeled links $(ij)^c$ connected to a neuron (i), its state is given by

$$z_i = \phi(x_i) g_i . \qquad (4)$$

Here, $\phi : \mathbb{R} \rightarrow [0,1]$ is a sigmoid function. This means, if a neuron (i) has no gating links $(ij)^g$ connected to it, then $g_i = 1$ and the sigmoid $\phi(x_i)$ describes its activation. Otherwise, the gating term g_i multiplies to it.
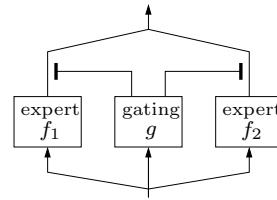

Figure 1: Ordinary multi-expert architecture. Gating and experts modules are explicitly separated and the gating may not depend on internal states or the output of experts.

Neurons (i) that are connected by (bi-directed) c-labeled links $(ij)^c$ form a *competitive group* in which only one of the neurons (the *winner*) acquires state $z_{\text{winner}} = 1$ while the other's states are zero. Let $\{i\}^c$ denote the competitive group of neurons to which (i) belongs. On such a group, we introduce a normalized distribution y_i, $\sum_{j \in \{i\}^c} y_j = 1$, given by

$$y_i = \frac{\psi(x_i)}{X_i} , \quad X_i = \sum_{k \in \{i\}^c} \psi(x_k) . \qquad (5)$$

Here, ψ is some function $\mathbb{R} \rightarrow \mathbb{R}$ (e.g., the exponential $\psi(x) = e^{\beta x}$). The neurons states $z_j \in \{0,1\}$, $j \in \{i\}^c$ depend on this distribution y_i by one of the following competitive rules of winner selection: We will consider a selection with probability proportional to y_i (softmax), deterministic selection of the maximum y_i, and ϵ-greedy selection (where with probability ϵ a random winner is selection instead of the maximum).

Please see Figure 2 to get an impression of the architectural possibilities this representations provides. Example A realizes an ordinary feed-forward neural network, where the three output neurons form a competitive group. Thus, only one of the output neurons will return a value of 1, the others will return 0. Example B realizes exactly the same multi-expert system as depicted in Figure 1. The two outputs of the central module form a competitive group and gate the output neurons of the left and right module respectively—the central module calculates the gating whereas the left and right modules are the experts. Example C is an alternative way of designing multi-expert systems. Each expert module contains an additional output node which gates the rest of its outputs and competes with the gating nodes of the other experts. Thus, each expert estimates itself how good it can handle the current stimulus (see the Q-learning method described below). Finally, example D is a true hierarchical architecture. The two experts on the left compete to give an output, which is further processed and, again, has to compete with
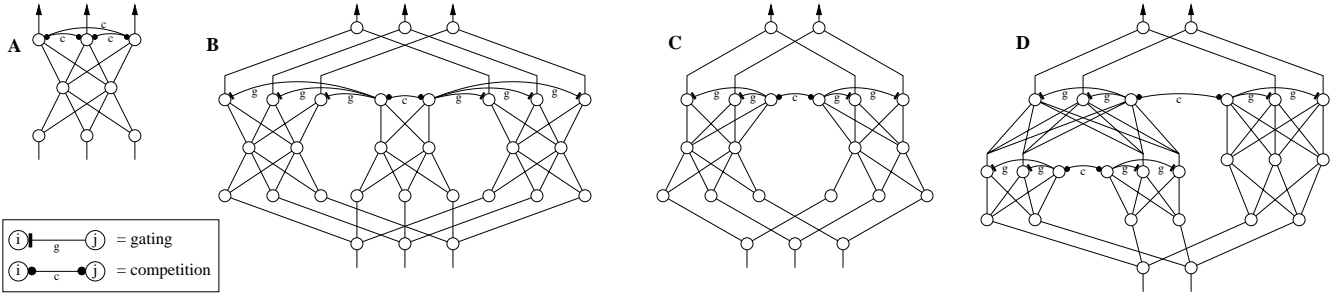
2

Figure 2: Sample architectures. Please see section II for a description of these architectures.

the larger expert to the right. In contrast, Jordan & Jacobs (1994) describe an architecture where the calculation of one single gating (corresponding to only one competitive level) is organized in a hierarchical manner. Here, several gatings on different levels can be combined in any successive, hierarchical way.

III Learning

In this section we introduce four different learning methods, each of which is applicable independent of the specific architecture. We generally assume that the goal is to approximate training data given as pairs (x, t) of stimulus and target output value.

The gradient method. To calculate the gradient, we assume that selection in competitive groups performed with probability proportional to the distribution y_i. We calculate an approximate gradient of the conditional probability $\mathcal{P}(y|x)$ that this system represents by replacing the actual state z_i in Eq. (2) by its expectation value y_i for neurons in competitive groups (see also Neal 1990). For the simplicity of notation, we identify $z_i \equiv y_i$. Then, for a neuron (i) in a competitive group obeying Eq. (5), we get the partial derivatives of the neuron's output with respect to its excitations:

$$\frac{\partial z_i}{\partial x_j} = \frac{\psi'(x_i)\,\delta_{ij}}{X_i} - \frac{\psi(x_i)}{(X_i)^2}\left[\psi'(x_j)\,\delta_{j\in\{i\}^c}\right]$$

$$= \frac{\psi'(x_j)}{X_i}\left[\delta_{ij} - z_i\,\delta_{j\in\{i\}^c}\right] , \qquad (6)$$

$$\frac{\partial z_i}{\partial g_j} = 0 , \qquad (7)$$

where $\delta_{j\in\{i\}^c} = 1$ iff j is a member of $\{i\}^c$. Let $E = E(z_1, .., z_n)$ be an error functional. We write the delta-rule for back-propagation by using the notations $\check{\delta}_i = \frac{dE}{dz_i}$ and $\left(\delta_i = \frac{dE}{dx_i},\ \delta_i^g = \frac{dE}{dg_i}\right)$ for the gradients at a neuron's output and excitations, respectively, and $e_i = \frac{\partial E}{\partial z_i}$ for the local error of a single

(output) neuron. From Eqs. (2,3,6,7) we get

$$\check{\delta}_i = \frac{dE}{dz_i} = e_i + \sum_j \frac{dE}{dx_j}\frac{\partial x_j}{\partial z_i} + \sum_j \frac{dE}{dg_j}\frac{\partial g_j}{\partial z_i}$$

$$= e_i + \sum_{(ji)} \delta_j\, w_{ji} + \sum_{(ji)^g} \delta_j^g\, \frac{1}{N_i} , \qquad (8)$$

$$\delta_i = \frac{dE}{dx_i} = \sum_j \check{\delta}_j \frac{\partial z_j}{\partial x_i} = \frac{\psi'(x_i)}{X_i}\Big[\check{\delta}_i - \sum_{j\in\{i\}^c} \check{\delta}_j\, z_j\Big] , \tag{9}$$

$$\delta_i^g = \frac{dE}{dg_i} = \sum_j \check{\delta}_j \frac{\partial z_j}{\partial g_i} = 0 . \qquad (10)$$

(In Eq. (9) we used $X_i = X_j$ for $i \in \{j\}^c$ and $i \in \{j\}^c \Leftrightarrow j \in \{i\}^c$.) For neurons that do not join a competitive group we get from Eq. (4)

$$\frac{\partial z_i}{\partial x_j} = \phi'(x_i)\, g_i\, \delta_{ij} , \quad \frac{\partial z_i}{\partial g_j} = \phi(x_i)\, \delta_{ij} , \qquad (11)$$

$$\delta_i = \frac{dE}{dx_i} = \sum_j \check{\delta}_j \frac{\partial z_i}{\partial x_i} = \phi'(x_i)\, g_i\, \check{\delta}_i , \qquad (12)$$

$$\delta_i^g = \frac{dE}{dg_i} = \sum_j \check{\delta}_j \frac{\partial z_i}{\partial g_i} = \phi(x_i)\, \check{\delta}_i , \qquad (13)$$

where $\check{\delta}_i$ is given in Eq. (8). The final gradients are

$$\frac{dE}{dw_i} = \delta_i , \quad \frac{dE}{dw_{ij}} = \delta_i\, z_j . \qquad (14)$$

The choice of the error functional is free. E.g., it can be chosen as the square error $E = \sum_i (z_i - t_i)^2$, $e_i = 2(z_i - t_i)$ or as the log-likelihood $E = \ln \prod_i z_i^{t_i} (1 - z_i)^{t_i}$, $e_i = \frac{t_i}{z_i} - \frac{1-t_i}{1-z_i}$, where in the latter case the target are states $t_i \in \{0, 1\}$.

The basis for further learning rules. For the following learning methods we concentrate on the question: *What target values should we assume for the states of neurons in a competitive group?* In the case

3

of gradient descent, Eq. (8) gives the answer. It actually describes a linear projection of the desired output variance down to all system states z_i—including those in competitions. In fact, all the following learning methods will adopt the above gradient descent rules except for a redefinition of $\check{\delta}_i$ (or alternatively δ_i) in the case of neurons (i) in competitive groups. This means that neurons "below" competitive groups are adapted by ordinary gradient descent while the local error *at* competitive neurons is given by other rules than gradient descent. Actually this is the usual way for adapting systems where neural networks are used as internal modules and trained by back-propagation (e.g., see Anderson & Hong 1994).

An EM-algorithm. We briefly review the basic ideas of applying an EM-algorithm on the problem of learning gatings in multi-experts (Jordan & Jacobs 1994). The algorithm is based on an additional, very interesting assumption: Let the outcome of a competition in a competitive group $\{c\}$ be described by the states $z_i \in \{0, 1\}$, $\sum_{i \in \{c\}} z_i = 1$ of the neurons that join this group. Now, we assume that there exists a *correct* outcome $h_i \in \{0, 1\}$, $\sum_{i \in \{c\}} h_i = 1$. Formally, this means to assume that the complete training data are triplets (x, h_i, t) of stimuli, competition states, and output values.[1] However, the competition training data is unobservable or *hidden* and must be inferred by statistical means. Bayes' rule gives an answer on how to infer an expectation of the hidden training data h_i and lays the ground for an EM-algorithm. The consequence of this assumption is that now the y_i of competitive neurons are supposed to approximate this expectation of the training data h_i instead of being free. For simplification, let us concentrate on a network containing a single competitive group; the generalization is straightforward.

- Our system represents the conditional probability of output states z^o and competition states z^c, depending on the stimulus x and parameters $\theta = (w_{ij}, w_i)$:

$$\mathcal{P}(z^o, z^c | x, \theta) = \mathcal{P}(z^c | x, \theta)\,\mathcal{P}(z^o | z^c, x, \theta)\,. \tag{15}$$

- (E-step) We use Bayes rule to infer the expected competition training data h_i hidden in a train-

---

[1] More precisely, the assumption is that there exists a teacher system of same architecture as our system. Our system adapts free parameters w_{ij}, w_i in order to approximate this teacher system. The teacher system produces training data and, since it has the same architecture as ours, also uses competitive groups to generate this data. The training data would be complete if it included the outcomes of these competitions.

ing tuple $(x, \cdot, t)$, i.e., the probability of h_i when x and t are given.

$$\mathcal{P}(h_i | x, t) = \frac{\mathcal{P}(t | h_i, x)\mathcal{P}(h_i | x)}{\mathcal{P}(t | x)} \tag{16}$$

Since these probabilities refer to the training (or teacher) system, we can only approximate them. We do this by our current approximation, i.e., our current system:

$$\begin{aligned}
\mathcal{P}(h_i | x, t, \theta) &= \frac{\mathcal{P}(t | h_i, x, \theta)\,\mathcal{P}(h_i | x, \theta)}{\mathcal{P}(t | x, \theta)} \\
&= \frac{\mathcal{P}(t | h_i, x, \theta)\,\mathcal{P}(h_i | x, \theta)}{\sum_{z^c} \mathcal{P}(z^c | x, \theta)\,\mathcal{P}(t | z^c, x, \theta)}\,.
\end{aligned} \tag{17}$$

- (M-step) We can now adapt our system. In the classical EM-algorithm, this amounts to maximizing the expectation of the log-likelihood (cp. Eq. (15))

$$E[l(\theta')] = E[\ln \mathcal{P}(h | x, \theta') + \ln \mathcal{P}(t | z^c, x, \theta')]\,, \tag{18}$$

where the expectation is with respect to the distribution $\mathcal{P}(h | x, t, \theta)$ of h-values (i.e., depending on our inference of the hidden states h); and the maximization is with respect to parameters θ. This equation can be simplified further—but, very similar to the "least-square" algorithm developed by Jordan & Jacobs (1994), we are satisfied to have inferred an explicit *desired* probability $\hat{y}_i = \mathcal{P}(h_i = 1 | x, t, \theta)$ for the competition states z_i that we use to define a mean-square error and perform an ordinary gradient descent.

Based on this background we define the learning rule as follows and with some subtle differences to the one presented in (Jordan & Jacobs 1994). Equation (17) defines the desired probability $\hat{y}_i$ of the states z_i. Since we assume a selection rule proportional to the distribution y_i, the values $\hat{y}_i$ are actually target values for the distribution y_i. The first modification we propose is to replace all likelihood measures involved in Eq. (17) by general error measures E: Let us define

$$Q_i(x) := 1 - E(x) \quad \text{if } (i) \text{ wins.} \tag{19}$$

Then, in the case of the likelihood error $E(x) = 1 - \mathcal{P}(t | x, \theta)$, we retrieve $Q_i(x) = \mathcal{P}(t | h_i = 1, x, \theta)$. Further, let

$$V(x) := \sum_i Q_i(x)\,y_i(x). \tag{20}$$

By these definitions we may rewrite Eq. (17) as

$$\hat{y}_i(x) = \frac{Q_i(x)\,y_i(x)}{V(x)} = \frac{Q_i(x)\,y_i(x)}{\sum_j Q_j(x)\,y_j(x)} \ . \quad (21)$$

However, this equation needs some discussion with respect to its explicit calculation in our context—leading to the second modification. Calculating $Q_j(x)$ for every j amounts to evaluating the system for every possible competition outcome. One major difference to the algorithm presented in (Jordan & Jacobs 1994) is that we do not allow for such a separated evaluation of all experts in a single time step. In fact, this would be very expensive in case of hierarchically interacting competitions and experts because the network had to be evaluated for each possible combinatorial state of competition outcomes. Thus we propose to use an approximation: We replace $Q_j(x)$ by its average over the recent history of cases where (j) won the competition,

$$\bar{Q}_j \leftarrow \gamma\,\bar{Q}_j + (1-\gamma)\,Q_j(x) \quad \text{whenever } (j) \text{ wins} \,, \quad (22)$$

where $\gamma \in [0,1]$ is a trace constant (as a simplification of the time dependent notation, we use the algorithmic notation $\leftarrow$ for a replacement if and only if (j) wins). Hence, our adaptation rule finally reads

$$\check{\delta}_i = -\alpha_c \left[y_i - \frac{Q_i\,y_i}{\sum_{j \in \{i\}^c} \bar{Q}_j\,y_j} \right] \quad \text{if } (i) \text{ wins,} \quad (23)$$

and $\check{\delta}_i = 0$ if (i) does not win; which means a gradient descent on the squared error between the approximated desired probabilities $\hat{y}_i$ and the distribution y_i.

Q-learning. Probably, the reader has noticed that we chose the notations in the previous section in the style of reinforcement learning: If one interprets the winning of neuron (i) as a decision on an action, then $Q_i(x)$ (called *action-value function*) describes the (estimated) quality of taking this decision for stimulus x; whereas $V(x)$ (called *state-value function*) describes the estimated quality for stimulus x without having decided yet, see (Sutton & Barto 1998). In this context, Eq. (21) is very interesting: it proposes to adapt the probability $y_i(x)$ according to the ratio $Q_i(x)/V(x)$—the EM-algorithm acquires a very intuitive interpretation. To realize this equation without the approximation described above one has to provide an estimation of $V(x)$, e.g., a neuron trained on this target value (a *critic*). We leave this for future

<div style="border:1px solid">

– The adaptation rate is $\alpha = 0.01$ for all algorithms (as indicated in Eqs. (23,24,26), the delta-values for neurons in competitive groups are multiplied by α_c).
– Parameters are initialized normally distributed around zero with standard deviation $\sigma = 0.01$.
– The sigmoidal and linear activation functions are $\phi_s(x) = \frac{1}{1+\exp(-10x)}$ and $\phi_l(x) = x$, respectively.
– The competition function ψ for softmax competition is $\psi_s(x) = e^{5x}$.
– The Q-learning algorithm uses ϵ-greedy selection with $\epsilon = 0.1$; the others select either the maximal activation or with probability proportional to the activation.
– The values of the average traces $\bar{Q}_i$ and $\bar{V}$ are initialized to 1.
– The following parameters were used for the different learning schemes:

	gradient	EM	Q	Oja-Q
α_c	–	1	10	100
γ	–	0.9	–	0.9
ψ	ψ_s	ϕ_s	ϕ_l	ϕ_l
selection	proportional	max	greedy	max

Here, α_c is the learning rate factor, γ is the average trace parameter, and ψ is the competition function.

</div>

Table 1: Implementation details

research and instead directly address the Q-learning paradigm.

For Q-learning, an explicit estimation of the action-values $Q_i(x)$ is modeled. In our case, we realize this by considering $Q_i(x)$ as the target value of the excitations x_i, $i \in \{c\}$, i.e., we train the excitations of competing neurons toward the action values,

$$\delta_i = \alpha_c \begin{cases} x_i - Q_i & \text{if } (i) \text{ wins} \\ 0 & \text{else} \end{cases} . \quad (24)$$

This approach seems very promising—in particular, it opens the door to temporal difference and TD(λ) methods and other fundamental concepts of reinforcement learning theory.

Oja-Q learning. Besides statistical and reinforcement learning theories, also the branch of unsupervised learning theories gives some inspiration for our problem. The idea of hierarchically, serially coupled competitive groups raises a conceptual problem: Can competitions in areas close to the input be trained without functioning higher level areas (closer to the output) and vice versa? Usually, back-propagation is the standard technique to address this problem. But this does not apply on either the EM-learning or the reinforcement learning approaches because they generate a direct feedback to competing neurons in any layer. Unsupervised learning in lower areas seems

5

to point a way out of this dilemma. As a first approach we propose a mixture of unsupervised learning in the fashion of the normalized Hebb rule and Q-learning. The normalized Hebb rule (of which the Oja rule is a linearized version) can be realized by setting $\delta_i = -\alpha_c\, z_i$ for a neuron (i) in a competitive group (recall $z_i \in \{0,1\}$). The gradient descent with respect to adjacent input links gives the ordinary $\Delta w_{ij} \propto z_i z_j$ rule. Thereafter, the input weights (including the bias) of each neuron (i), $i \in \{c\}$ are normalized. We modify this rule in two respects. First, we introduce a factor $(Q_i - \bar{V})$ that accounts for the success of neuron (i) being the winner. Here, $\bar{V}$ is an average trace of the feedback:

$$\bar{V} \leftarrow \gamma\,\bar{V} + (1-\gamma)\,Q_i(x) \quad \text{every time step ,} \tag{25}$$

where (i) is the winner. Second, in the case of failure, $Q_i < \bar{V}$, we also adapt the non-winners in order to increase their response on the stimulus next time. Thus, our rule reads

$$\delta_i = -\alpha_c\,(Q_i - \bar{V})\begin{cases} z_i & \text{if } Q_i \geq \bar{V} \\ z_i - 0.5 & \text{else} \end{cases}. \tag{26}$$

Similar modifications are often proposed in reinforcement learning models (Barto & Anandan 1985; Barto & Jordan 1987). The rule investigated here is only a first proposal; all rules presented in the excellent survey of Diamantaras & Kung (1996) can equally be applied and are of equal interest but have not yet been implemented by the author.

IV   Empirical study

We test the functionality of our model and the learning rules by addressing a variant of the test presented in (Jacobs, Jordan, & Barto 1990). A single bit of an 8-bit input decides on the subtask that the system has to solve on the current input. The subtasks itself are rather simple and in our case (unlike in (Jacobs, Jordan, & Barto 1990)) are to map the 8-bit input either identically or inverted on the 8-bit output. The task has to be learned online. We investigate the learning dynamics of a conventional feed-forward neural network (FFNN) and of our model with the 4 different learning methods. We use a fixed architecture similar to an 8-10-8-layered network with 10 hidden neurons but additionally install 2 competitive neurons that receive the input and each gates half of the hidden neurons, see Figure 3. In the case of the conventional FFNN we used the same architecture but replaced all
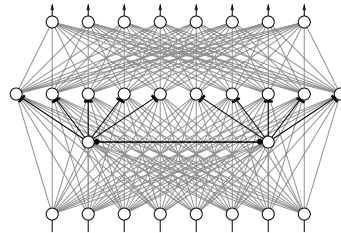

Figure 3: The architecture we use for our experiments. All output neurons have linear activation functions $\phi(x) = x$. All except the input neurons have bias terms.

gating and competitive connections by conventional links.

Figure 4 displays the learning curves averaged over 20 runs with different weight initializations. For implementation details see Table 1. First of all, we find that all of the 4 learning methods perform well on this task compared to the conventional FFNN. The curves can best be interpreted by investigating if a task separation has been learned. Figure 5 displays the frequencies of winning of the two competitive neurons in case of the different subtasks. The task separation would be perfect if these two neurons would reliably distinguish the two subtasks. First noticeable is that all 4 learning methods learn the task separation. In the case of Q-learning the task separation is found rather late and remains noisy because of the ϵ-greedy selection used. This explains its slower learning curve in Figure 4. EM and Oja-Q realize strict task separations (maximum selection), for the gradient method it is still a little noisy (softmax selection). It is clear that, if the task separation has been found and fixed, all four learning methods proceed equivalently. So it is no surprise that the learning curves in Figure 4 are very similar except for a temporal offset corresponding to the time until the task separation has been found, and the non-zero asymptotic error corresponding to the noise of task separation. (Note that Figure 5 represents only a *single*, typical trial.)

Generally, our experience was that the learning curves may look very different depending on the weight initialization. It also happened that the task separation was not found when weights and biases (especially of the competing neurons) are initialized very large (by $\mathcal{N}(0, 0.5)$). One of the competitive neurons then dominates from the very beginning and prohibits the "other expert" to adapt in any way. Definitely, a special, perhaps equal initialization of competitive neurons could be profitable.

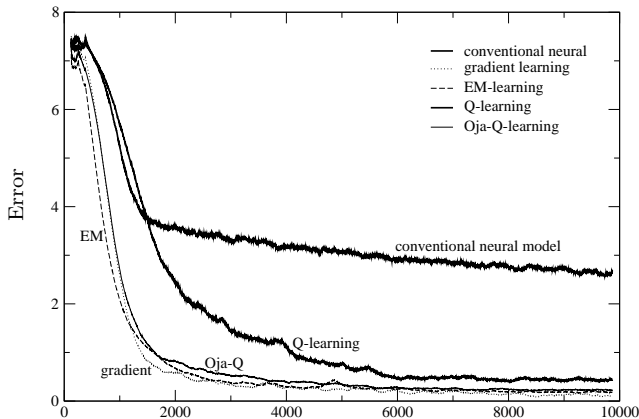Finally, also the conventional FFNN only *some-*

Figure 4: Learning curves for the conventional neural network and the four different learning schemes.

times solves the task completely—more often when weights are initialized relatively high. This explains the rather high error offset for its learning curve.

V   Conclusion

We generalized conventional neural networks to allow for multi-expert like interactions. We introduced 4 different learning methods for this model and gave empirical support for their functionality. What makes the model particularly interesting is:

1. The generality of our representation of system architecture allows new approaches for the structure optimization of multi-expert systems, including arbitrary serial, parallel, and hierarchical architectures. In particular evolutionary techniques of structure optimization become applicable.

2. The model allows the combination of various learning methods within a single framework. Especially the idea of integrating unsupervised learning methods in a system that adapts supervised opens new perspectives. Many more techniques from elaborated learning theories can be transfered on our model. In principle, the uniformity of architecture representation would allow to specify freely where it is learned by which principles.

3. The model overcomes the limitedness of conventional neural networks to perform task decomposition, i.e., to adapt in a decorrelated way to decorrelated data (Toussaint 2002).
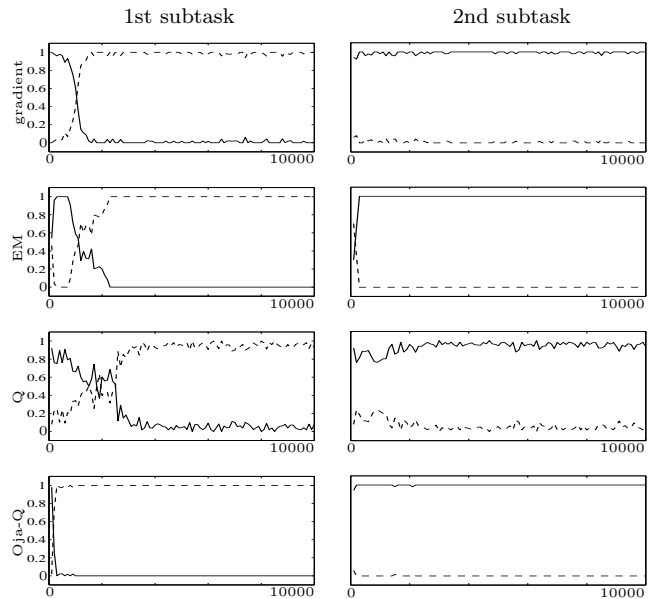

Figure 5: The gating ratios for single trials for the four different learning schemes: The four rows refer to gradient, EM-, Q-, and Oja-Q-learning; and the two columns refer to the two classes of stimuli—one for the "identical" task, and one for the "not" task. Each graph displays two curves that sum to 1 and indicate how often the first or second gating neuron wins in case of the respective subtask.

References

Anderson, C. W. & Z. Hong (1994). Reinforcement learning with modular neural networks for control. In *IEEE International Workshop on Neural Networks Application to Control and Image Processing*.

Barto, A. G. & P. Anandan (1985). Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics* **15**, 360–375.

Barto, A. G. & M. I. Jordan (1987). Gradient following without back-propagation in layered networks. In *Proceedings of the IEEE First Annual Conference on Neural Networks*, pp. II629–II636.

Bengio, Y. & P. Frasconi (1994). An EM approach to learning sequential behavior. Technical Report DSI-11/94, Dipartimento di Sistemi e Informatica, Universita di Firenze.

Cacciatore, T. W. & S. J. Nowlan (1994). Mixtures of controllers for jump linear and non-linear plants. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in Neural Information Processing Systems*, Volume 6, pp. 719–726. Morgan Kaufmann Publishers, Inc.

Diamantaras, K. I. & S. Y. Kung (1996). *Principle compo-*

nent neural networks: theory and applications. Wiley & Sons, New York.

Jacobs, R. A. (1999). Computational studies of the development of functionally specialized neural modules. *Trends in Cognitive Sciences* **3**, 31–38.

Jacobs, R. A., M. I. Jordan, & A. G. Barto (1990). Task decomposistion through competition in a modular connectionist architecture: The what and where vision tasks. Technical Report COINS-90-27, Department of Computer and Information Science, University of Massachusetts Amherst.

Jordan, M. I. & R. A. Jacobs (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation* **6**, 181–214.

Neal, R. M. (1990). Learning stochastic feedforward networks. Technical Report CRG-TR-90-7, Department of Computer Science, University of Toronto.

Rahman, A. F. R. & M. C. Fairhurst (1999). Serial combination of multiple experts: A unified evaluation. *Pattern Analysis & Applications* **2**, 292–311.

Ronco, E., H. Gollee, & P. Gawthrop (1997). Modular neural networks and selfdecomposition. Technical Report CSC-96012, Center for System and Control, University of Glasgow.

Sutton, R. S. & A. G. Barto (1998). *Reinforcement Learning.* MIT Press, Cambridge.

Toussaint, M. (2002). On model selection and the disability of neural networks to decompose tasks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2002)*.

Yao, X. (1999). Evolving artificial neural networks. In *Proceedings of IEEE*, Volume 87, pp. 1423–1447.